

ESSENTIALS OF EDGE COMPUTING

Robert Oshana, Editor-in-Chief





All proceeds go to IGNITE Worldwide

NXP is committed to inspiring the next generation of technologists. All proceeds from the sale of this book will go to IGNITE Worldwide, a nonprofit organization dedicated to advancing gender and racial equity in Science, Technology, Engineering, and Math (STEM).

IGNITE Worldwide provides a sustainable, scalable, and effective award-winning program that works directly with educators during the school day to reach girls, nonbinary, transgender, and agender students. The IGNITE Program, which is provided cost-free to schools, removes barriers faced by low-income and marginalized youth, providing them the possibility to envision, engage, and pursue STEM education and career opportunities.

To learn more about IGNITE Worldwide visit www.igniteworldwide.org.

FOREWORD

We anticipate a huge leap in the smart things that make up the edge of everything in our homes, offices, factories, and vehicles. The 50 billion smart connected devices expected by 2025 can influence how individuals, communities and entire industries communicate, learn and operate. These devices will anticipate our needs and automate our environments. As an industry, it is up to all of us to ensure they meet the ultimate goals for society – a greener, safer, and more productive society. No single company can do this alone. It's a collective effort that requires expertise across an ecosystem of semiconductor suppliers, software partners and device makers.

Widespread adoption of these smart connected devices means enormous amounts of data will be created, which is why the edge is fast becoming a requirement for the next era of the IoT. The edge puts processing power where data is generated. At the heart is silicon, but the end-to-end architecture of an edge device is much more complex than the silicon itself. Deeply intertwined in the silicon is software that brings advancements in security, low power, machine learning, and connectivity.

This book was created to share knowledge and insights to help the industry unravel this complexity and drive forward the enormous potential of the edge. Whether you're creating SoCs or edge products, you are an enabler of a society that is greener, safer and more productive. I hope you find information in this book useful for your tasks towards realizing the future edge of everything.

Ron Martino

Executive Vice President and General Manager
Edge Processing
NXP Semiconductors

PREFACE

Computing at the edge of a network is a fundamentally simple concept, but it requires a broad range of capabilities to achieve optimal security, energy efficiency, connectivity and machine learning intelligence. In *Essentials of Edge Computing*, you'll find insights and best practices on a wide range of these emerging edge computing design concepts that you can apply in your next application.

I would like to thank many key contributors who have made this book possible, including Mohit Arora, Jean-Christophe Bodet, Antoine Boiteau, Cristi Caciuloiu, Brian Carlson, Nicolas Collonvillé, Julien Delplancke, Silvano di Ninno, Alexandra Dopplinger, Mihai-Andrei Dragnea, Natraj Ekambaram, Sebastian Grigore, Doru Gucea, Michal Hanak, Mathieu Imbault, Saleem Kala-Janssen, Prabhu Loganathan, Nihaar Mahatme, Pascal Mareau, Jason Martin, Guillermo Michel, Sujata Neidig, Ali Osman Örs, Nicu Penișoară, Laurent Pilati, Wim Rouwet, Erich Styger, Marc Vauclair, and Francois Villeneuve. Special thanks goes to the reviewers who helped convey the concepts, including Gowri Chindalore and Monica Davis.

A well-architected edge computing system requires multiple domains of expertise, and these contributors are the experts who are sharing their knowledge and expertise with you. We would love to hear from you about your experiences in this new era of edge computing. Let the journey begin!

Robert Oshana

Vice President, Edge Processing Software R&D
NXP Semiconductors
Technical Editor-in-Chief

CONTENTS

Foreword	3
Preface	4
Chapter 1: Introduction	6
Chapter 2: Hardware and Software Architectures	14
Chapter 3: Security	28
Chapter 4: Machine Learning Intelligence	50
Chapter 5: Connectivity	64
Chapter 6: Device Life-Cycle Management	104
Chapter 7: Energy Efficiency and Optimization	120
Chapter 8: Human Machine Interface	140
Chapter 9: Use Cases	156
Local Voice at the Edge Optimized for Power Efficiency	158
5G Technology as an Enabler for Industry 4.0	162
Wearables	169
Time-Sensitive Networking and Distributed Real-time Computing at the Edge	175
Intelligent Connected Vehicles	183
Chapter 10: Development Tools	192
Glossary	204
Contributors	209

Chapter 1

EDGE COMPUTING INTRODUCTION

THE ROAD TO THE EDGE

In the “early days” of the internet of things (IoT), processing and most storage was performed in cloud data centers because only the cloud had the computational resources needed to perform complex analysis. But as more connected applications were deployed, the limitations of cloud processing became apparent.

One of the limitations was latency — the time required for the data generated by sensors at the source, to traverse the path to the cloud for processing and then back to deliver actionable results. Milliseconds of latency might not be important in a smart thermostat, but an industrial robot and other real-time systems require even less time for ensuring safety and productivity. For the sensor-based safety features on modern vehicles, latency can be a matter of life and death.

Even a modest application with sensors can create enormous amounts of data that consumes costly bandwidth available across the network. The cloud-based approach could also expose sensitive information, including intellectual property (IP), that must be protected. Today, security has become one of the most critical aspects of the IoT.

A better solution is to divide the processing tasks between cloud-based servers and processors operating at the location where the data is generated, which is generally called the edge. More precisely, it's the edge of the network, or, from a data center's perspective, the far edge. Note that some processing has always been performed at the edge, principally in gateways that aggregated the data produced by sensors into a standard format, and then sent outward (Figure 1.1).

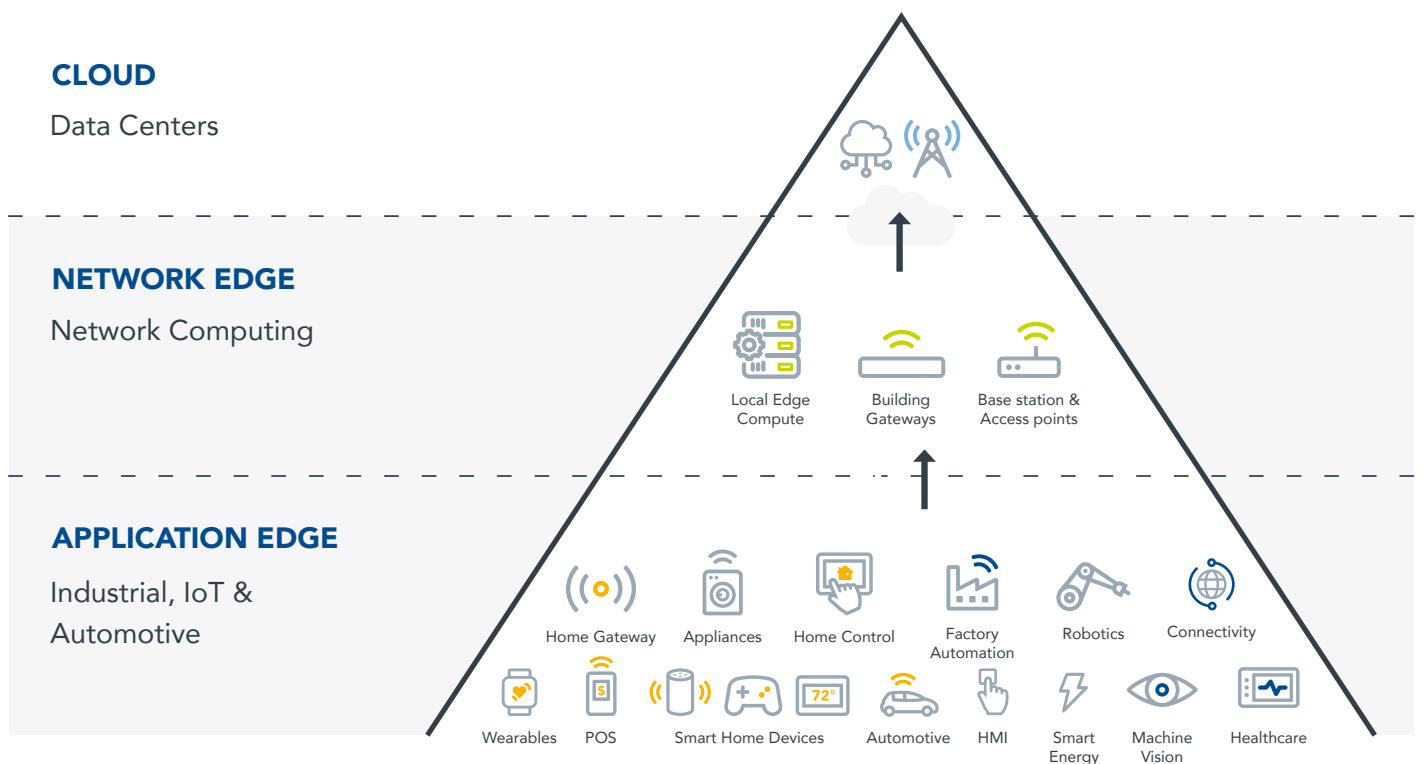


Figure 1.1. Conceptual representation of edge computing

NXP EdgeVerse™ edge processing solutions span across IoT, industrial and automotive markets.

However, in this new scenario, significant processing is performed at the edge on processors to satisfy the needs of real-time applications that require responses almost instantaneously. Several solutions emerged to achieve this new scenario.

In the first, mini data centers placed near the end-user location perform reasonably high levels of processing. Because they are nearby, latency is dramatically reduced. In the next approach, the gateway became a more formidable processing element with less computational horsepower but the ability to reduce latency even further. None of these approaches are small or consume little power.

Recent advancements in computing, including applications processors and microcontrollers, have brought significant compute capability to the edge. These devices can now perform machine learning, creating a plethora of edge applications that span beyond gateways and to the sensors and things in between.

TINY POWERHOUSES

Microcontrollers and microprocessors have made great leaps forward in performance, capabilities and cost reduction. They can include multiple cores dedicated to specific tasks, support several wireless protocols, feature power management and offer other impressive features. Today, they are powerful enough to make decisions based on data aggregated from multiple sensors at the edge.

They can perform analysis that was formerly the sole domain of the cloud, send commands to machines with virtually no latency and transfer only a summary of the information (a much smaller amount) to the cloud. An intelligent door lock, for example, can facilitate unlocking doors when it recognizes a face because it knows the person, and it can store and process the image data locally for a speedier response and enhanced privacy.

This new approach has become so important for IoT that there's a name for it: TinyML. It shrinks deep learning networks to fit on microcontrollers. The concept is not new. Smartphones have neural networks that enable music identification, multiple camera modes and various other functions to be performed, even in a smartwatch. But only recently has TinyML been applied to edge computing.

ENERGY-EFFICIENT COMPUTATION

Many modern MCUs are designed to operate with low power consumption, which enables the devices to run unplugged on batteries for weeks, months and, in some cases, even years while running ML applications on the edge device.

IoT edge devices can run on battery or solar power or be plugged into the wall. Energy is a costly commodity, whether battery or line powered. Computing is not the only consumer of energy; any process involving data aggregation, wired or wireless data transmission and data analysis consumes energy. To understand where to save on energy costs, look at what is consuming it. Decreasing power consumption requires reducing the time that a processor is active. This can be achieved by separating a processor into functional blocks capable of fine-grained power partitioning and management. Edge compute processors are an example of a recent innovation that makes this a reality.

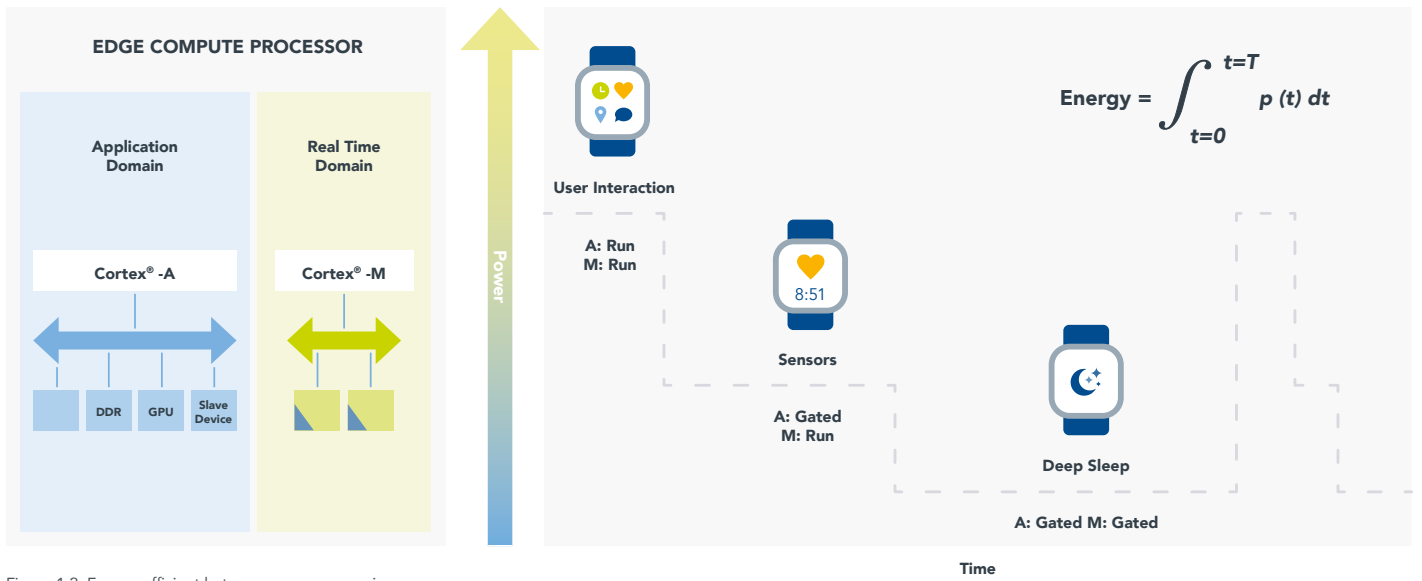


Figure 1.2. Energy-efficient heterogeneous processing

Figure 1.2 shows an edge compute processor application for a smartwatch. In this example, the NXP processor has two separate Arm® Cortex® core domains: a Cortex-A domain for running the watch and a Cortex-M core for real-time processing. Both domains have fine-grained power management and partitioning, allowing the domain to be shut down into a deep sleep mode. Each core's operation is highlighted with the different use cases that show how energy consumption is minimized.



INTELLIGENT PRODUCTIVITY

Intelligence and the capability of an edge device to make decisions locally imply the use of ML inference, the basis of an intelligent edge device. This significant rise in ML inferencing at the edge is partly due to the improvements in inference processing techniques and the development of energy-efficient inference “engine” accelerators. The number of potential inference applications that can be conducted at the edge increases with the applications’ energy efficiency (Figure 1.3).

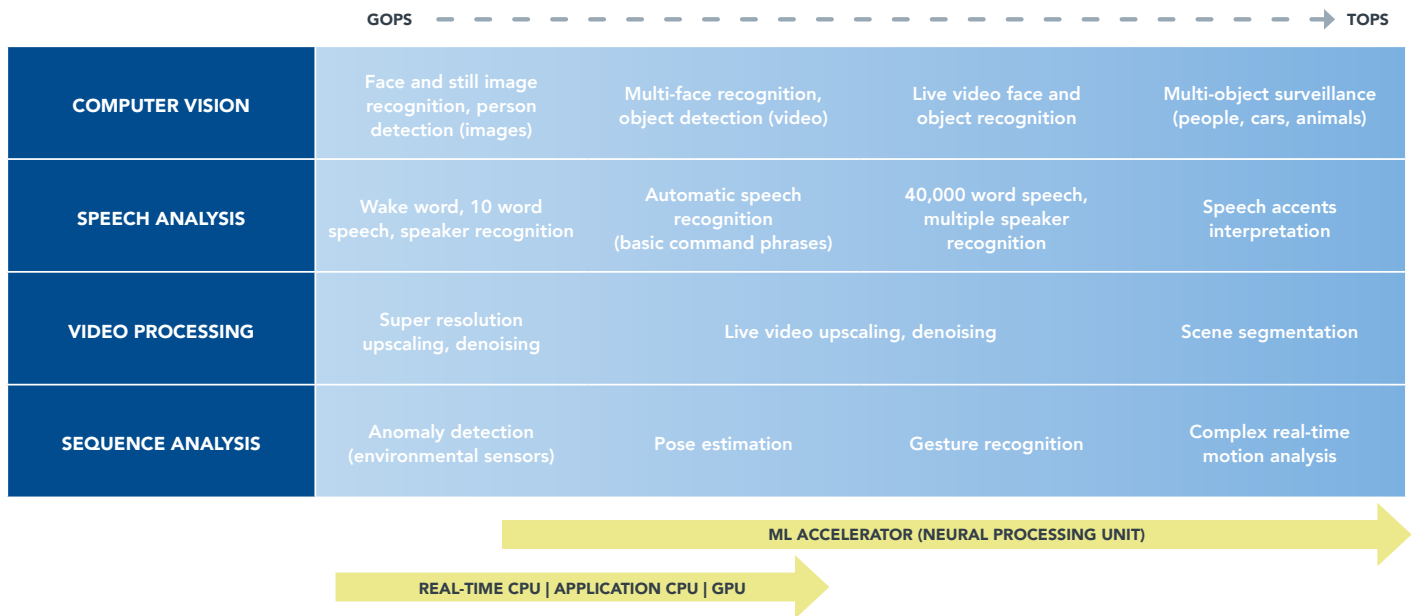


Figure 1.3. More possibilities with energy-efficient inference

Potential applications for inference at the edge are limited only by the imagination. Consider a smart home application that monitors for unusual sounds when the occupants are not at home. It detects running water sounds, perhaps indicating a pipe leak or a leaking water faucet. The sound of breaking glass indicates a window being broken. These and many more scenarios trigger the smart home application to alert the owners of problems. In addition to energy efficiency, intelligent productivity adds a societal and safety dimension to the technology benefits.

DATA SECURITY AND PRIVACY

Keeping data secure and private is the number one priority for any application. With more data collected and processed locally, any intelligent edge application needs to remain vigilant. Security applies to securing the edge application code, the data being processed and any data communication to the cloud. The use of encryption keys can help validate firmware, authenticate techniques for cloud communication and prevent adversaries from gaining control of the device. Embedded security, isolated secure subsystems and secure software enablement are the foundation of any intelligent edge processor (see Figure 1.4). Only by taking a holistic approach to device security and data privacy that embraces collective security knowledge and best practices can the IoT transform into the “internet of trust”.

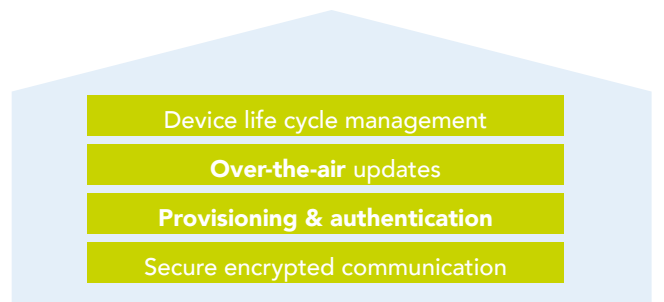
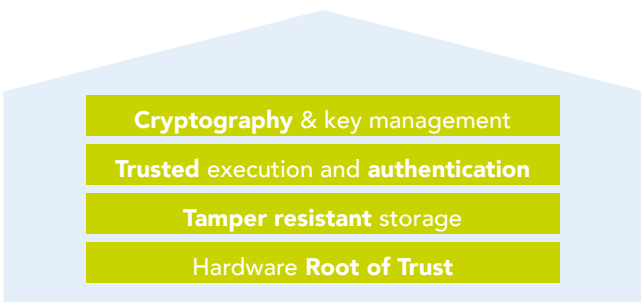
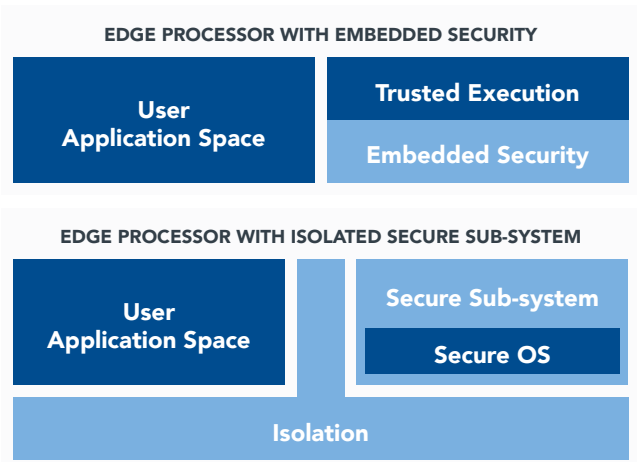


Figure 1.4. Typical data security and privacy factors for an edge processor



EDGE AWARENESS

As edge computing adoption expands, more intelligence will be added to devices to achieve a level of “awareness” across a network of devices. These systems can perform ML, make decisions without any external assistance and use minimal power in a footprint the size of a postage stamp. Connect multiple intelligent devices across a network, and it becomes a data-generating, decision-making aware network.

An intelligent device may process only voice, but intelligent devices across an aware network could combine nuances such as the tone of the voice, facial expressions and body gestures. For example, data from multiple smart home sensors connected across an aware network could recognize danger signals such as someone falling and send an alert to a remote caregiver. And if some of the data must be sent to the cloud, an intelligent edge processor could anonymize it along with other information such as audio profiles of a person’s voice and others who are authenticated.

Another example is a home security vision system that scans faces and recognizes family members. A typical security camera captures video of car theft in a driveway. If awareness and context are added, the system can spot a person it doesn’t recognize and send an alert to authorities.

Our homes and buildings can also become occupancy-aware, recognizing when the house is empty and then arm the security system, dim the lights and lower the temperature.

SUMMARY

Computing at the edge is not a replacement for processing in the cloud. Instead, it complements the cloud while providing a way to maintain data safety by keeping it local, eliminating the cost of transmitting gargantuan data to and from a remote data center and allowing real-time applications to get the answers they need in a few milliseconds rather than minutes (or longer). In short, edge computing is now a component of IoT architectures for a vast number of applications.

Credit for the ability to achieve this at low cost with low power consumption must be given to the microcontrollers and microprocessors that have evolved into powerful compute platforms capable of making decisions previously unachievable a few years ago. As their performance increases in coming years, they will take on even more functions to further complement the cloud.

In the chapters that follow, details of edge computing technologies will be discussed in greater detail along with many other technologies that make this approach possible.



Chapter 2

EDGE COMPUTING HARDWARE AND SOFTWARE ARCHITECTURES

CONTRIBUTORS

Mohit Arora, Director, Low-Power & Security Architecture, NXP Semiconductors

Rob Oshana, Vice President, Edge Processing Software R&D, NXP Semiconductors



Edge computing architectures are evolving. This chapter explores the hardware and software that make up these architectures and how they connect the real world with the cloud.

EDGE-TO-CLOUD COMMUNICATION

Computing advancements have enabled IoT devices to process data locally at the edge without sending data to the cloud. Edge computation can also act as a proxy between the real world and computation in the cloud. End-device sensors collect information from the real world. That data is processed locally (filter and compute), and the resulting meaningful information can be sent to the cloud. Edge computing involves “data in motion,” and cloud-based computation involves “data at rest.” In that sense, edge computing is enabling communication from the edge to the cloud, so edge computing technology must work with multiple communication protocols and convert these to cloud-based protocols.

Edge computing also is responsible for the business rules of processing data in preparation for transmission to the cloud. For example, an engine sensor emits rotation status multiple times each second, but the analytics in the cloud operate on this data less frequently — once every 30 seconds. The edge computation must prepare the data locally and send it to the cloud every 30 seconds. This is an example of a “smart” gateway, where this intelligence is embedded locally in the edge device.

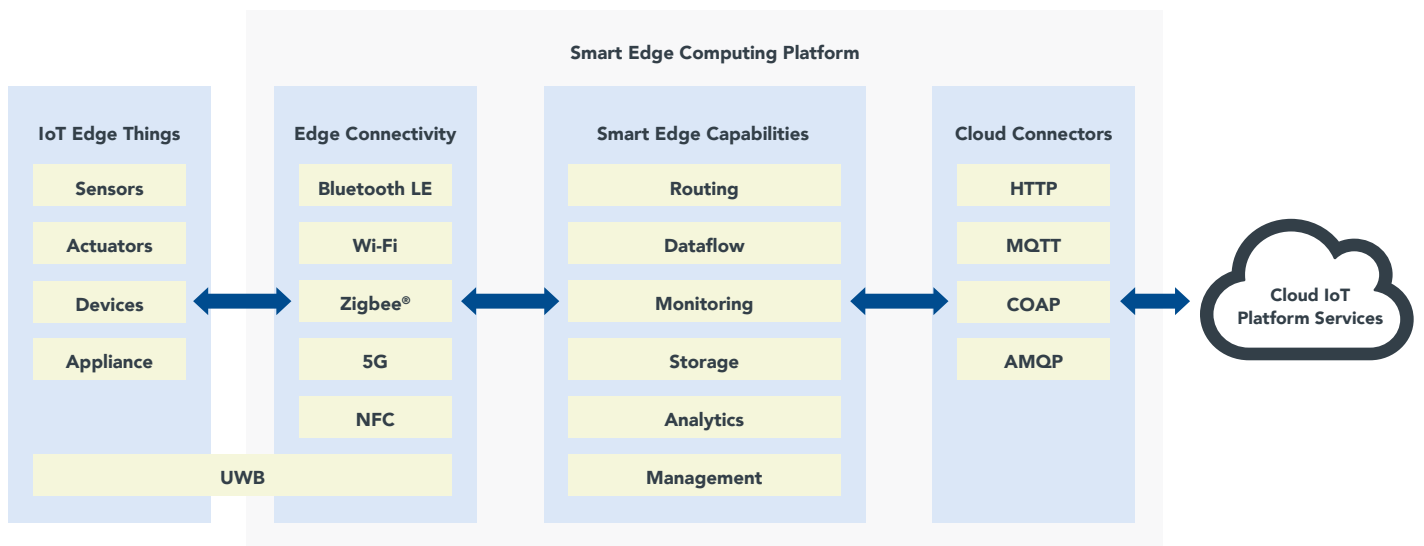


Figure 2.1. Edge computing architecture model

SENSORS AND ACTUATORS

Sensors and actuators (see Figure 2.2) operate in the trenches. They must have robust mechanisms to communicate with each other as well as the gateway and sometimes directly with the cloud. These communication technologies include Bluetooth®, Bluetooth® Low Energy (Bluetooth LE), Zigbee®, Wi-Fi and near-field communication (NFC).

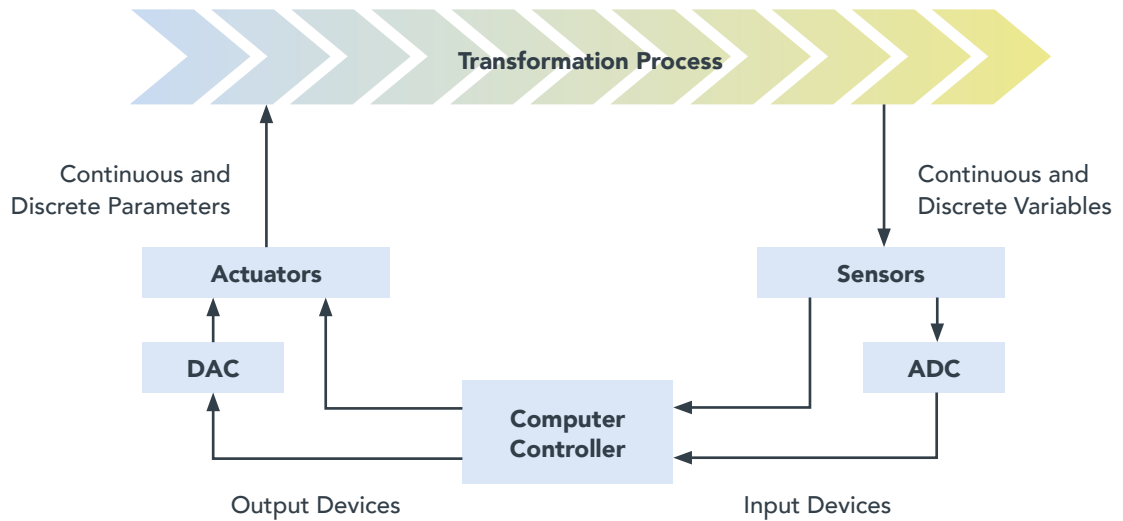


Figure 2.2. Sensors and actuators operating in the real-world environment

Sensors and actuators in edge computing systems usually communicate with microcontrollers where data is converted (analog to digital and/or digital to analog). Figure 2.3 shows an example of a microcontroller for an edge processing end node. It has a rich set of connectivity, processing and multimedia capabilities. These serve applications that require graphics such as wearables or smart meter systems. The microcontroller also has robust security capabilities to protect data flowing in and out as well as real-time capabilities such as interrupt processing and real-time scheduling.



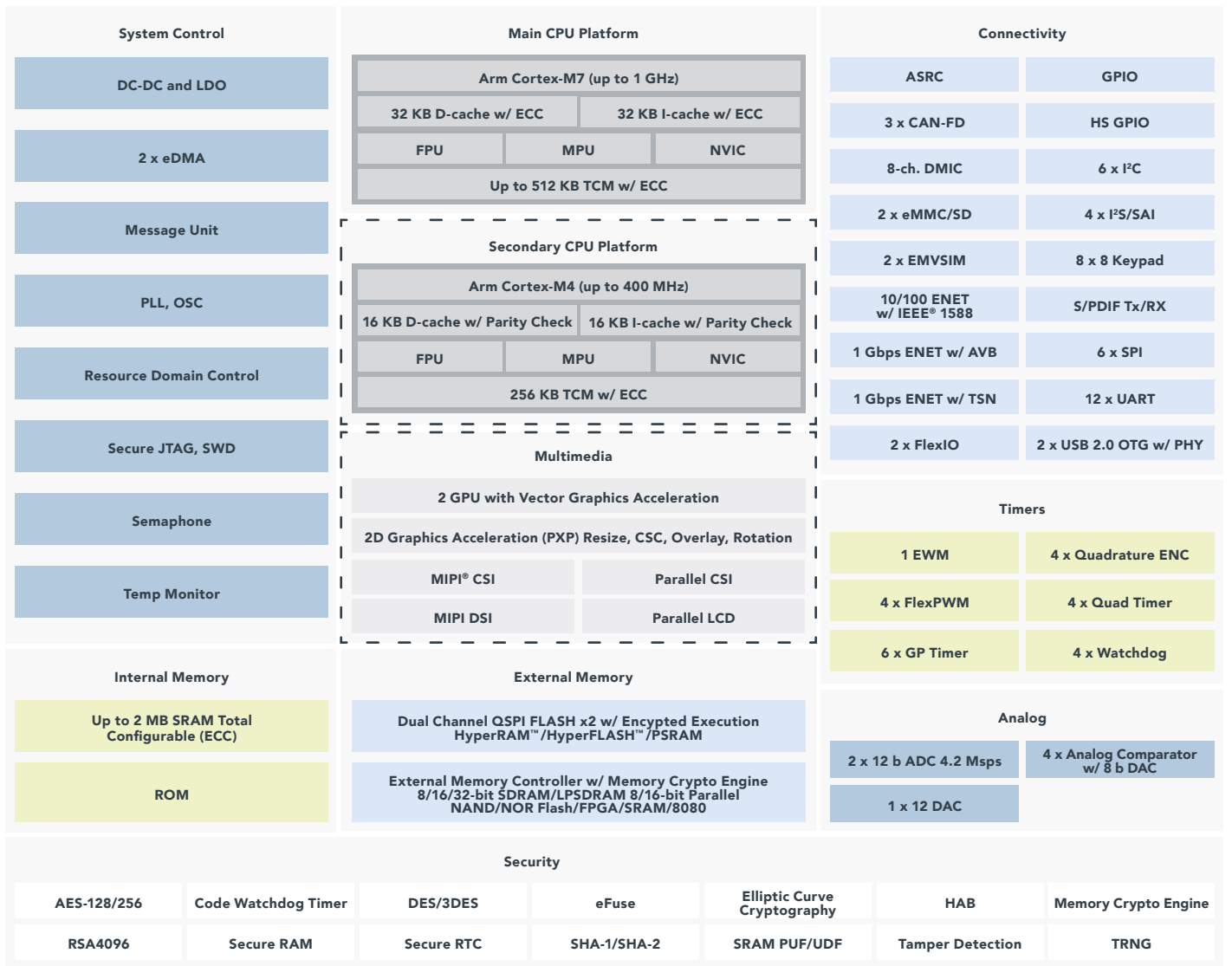


Figure 2.3. Hardware architecture for edge processing using a microcontroller for example



MICROPROCESSOR AND MICROCONTROLLER BASICS

A microprocessor is a general-purpose digital computer central processing unit. To make a complete microcomputer, a number of additional components, including memory (ROM and RAM), interfaces and I/O ports are required, as shown in Figure 2.4.

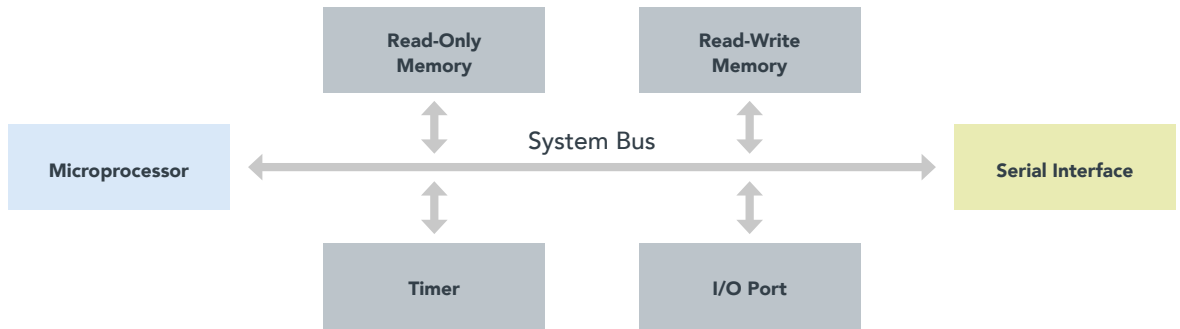


Figure 2.4 Typical microcontroller software architecture for edge processing

In Figure 2.4, support devices, including read-only memory, read-write memory, serial interface, timers and I/O port are all external and interfaced to the microprocessor via the system bus. The system bus, which connects components of a system, is composed of the data bus to carry information, an address bus to determine its destination or where it will read from, and a control bus to determine its operation. The primary use of a microprocessor is to read data, perform extensive calculations on that data, and store the results in a mass storage device or display the results.

The design of the microcontroller is driven by the desire to make it as expandable and flexible as possible. A microcontroller is a functional computer system-on-a-chip. Microcontrollers include an integrated processor, memory (a small amount of RAM, program memory, or both) and peripherals capable of input and output (see Figure 2.5).

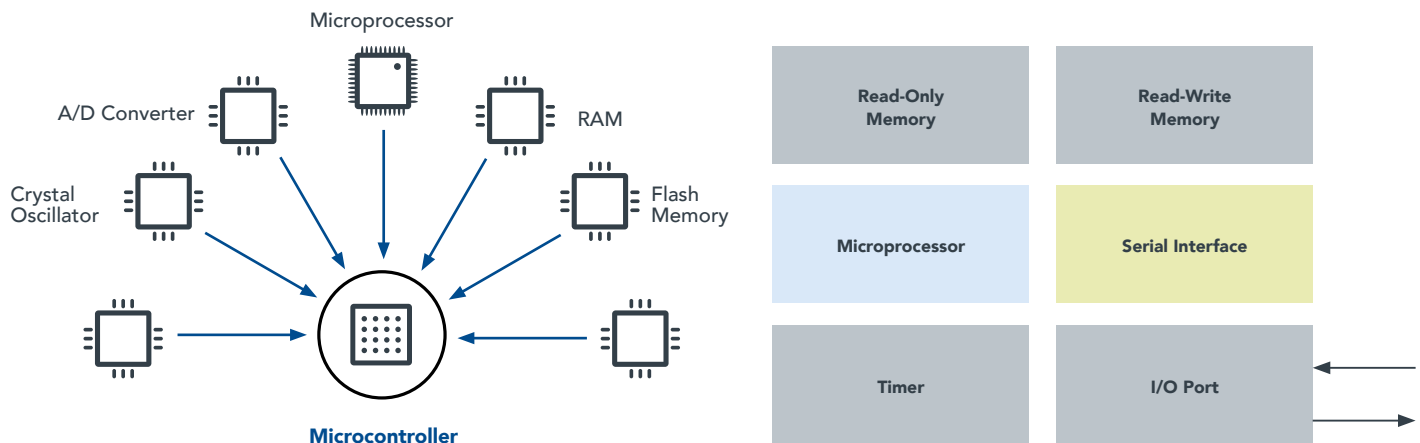


Figure 2.5 Microcontroller based system

Although the microprocessor is considered to be a powerful compute machine, it is limited with how it communicates to the peripheral environment. For communication with the peripheral environment, the microprocessor must use specialized circuits, which are external chips. The microprocessor, by definition, is the heart of the computer. On the other hand, the microcontroller is designed to be all of that in one: No other specialized external components are needed for its applications because all of the necessary circuits are already built into it. This saves both time and space when designing an end device.

Generally in the embedded world, the term "MPU" is used for "microprocessing unit" or "microprocessor" and does not include flash memory in the system-on-chip. Likewise the term "MCU" is used for "microcontroller" and includes on-chip flash memory in the system-on-chip.

EDGE DEVICE DESIGN CHALLENGES

An edge device design must balance numerous design attributes, as shown in Figure 2.6.

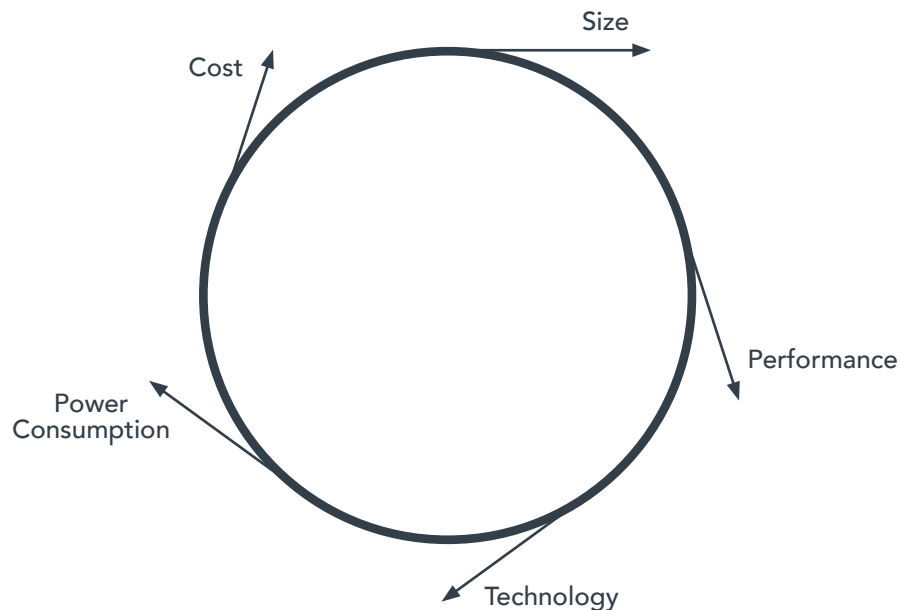


Figure 2.6 Parameters that control embedded system success

These design attributes typically compete with one another: improving one often leads to degradation of another. For example, if the die size is reduced, the features and performance of the edge device may suffer. Moving to a lower technology node to reduce the die size and cost, however, may increase the leakage and have an adverse impact on power consumption.



Performance

Edge device performance depends on more than processor speed. Real-time performance, or how quickly a system reacts to a specific event, is key. An edge device running a real-time operating system (RTOS) would often guarantee a response within specific time frame, which offers determinism. Conversely, the typical response for desktop computing is non-deterministic and guaranteed response time is not critical.

Systems used for many mission critical applications must process data in real-time. The real-time processing fails if it is not completed within a specified deadline relative to an event; deadlines must always be met, regardless of system load.



Power Consumption

Low power consumption is a critical parameter for an edge device. Compared to typical systems or desktop computers that are always powered, many edge devices are powered by battery. Though this is use-case dependent, an edge device may have a conflicting need for low power consumption and more performance.

Some applications may be continuously powered by battery including water or gas meters that measure the flow of water in a residential or commercial complex. Meters are required to work for several years without replacing the battery. Since they are often idle, ultra-low power modes are often used. Consumption can be measured in sleep mode, and enabled for counter overflows or data transmission to a remote network or cloud.



Technology, size and design cost

Unlike the desktop world where performance requirements drive the technology choices, there are number of factors that affect that decisions in edge device design. Some edge devices must be highly reliable to operate in extreme conditions for long operational hours without failure. A stable technology node that is well tested under extreme conditions is recommended. Furthermore, it is reasonable to assume that a system-on-chip for an embedded device would include analog-to-digital convertors (ADC/DAC) and integrated power management controllers (PMICs) that are tuned to specific technology. These need to be re-designed every time a new technology node is adopted which adds significant risk and design cost. Further, switching between different technology nodes can impact power consumption of the device and low power modes, affecting the chip architecture.

Since switching between technology nodes adds non-recurring engineering costs, volumes have to be high to justify the decision. To reduce the per unit cost of embedded systems on a chip (SoCs), it is necessary to reduce the die size either by restricting feature set or by switching to a lower technology node, which may be a natural transition once the technology is stable and the transition cost is justified. There is always a fine balance between technology, die size and design cost when determining an SoC for an edge device.

SOFTWARE

An edge processing microcontroller requires a comprehensive set of enablement software. Figure 2.7 shows a software architecture that features:

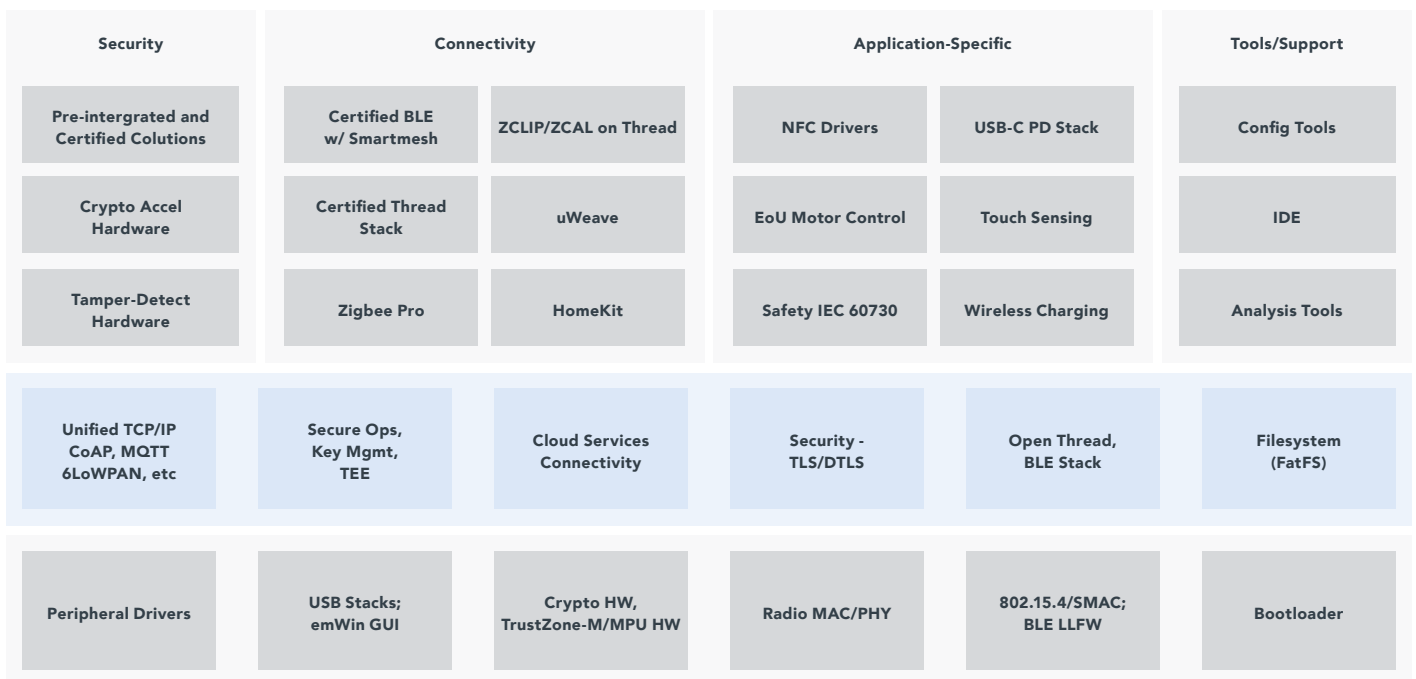


Figure 2.7. Microcontroller software architecture for edge processing

ADVANCED PROCESSING

Edge processing often requires more advanced processing capability for the devices operating on the edge and aggregating information from one or more end-node devices. Figure 2.8 shows an example of an edge processing device. This multimedia device has a range of capabilities to support various edge processing application types:



Processing power using multicore technology



Camera, video and audio interfaces



2D and 3D graphics capabilities



Connectivity protocols



Security protocols and performance



ML capabilities

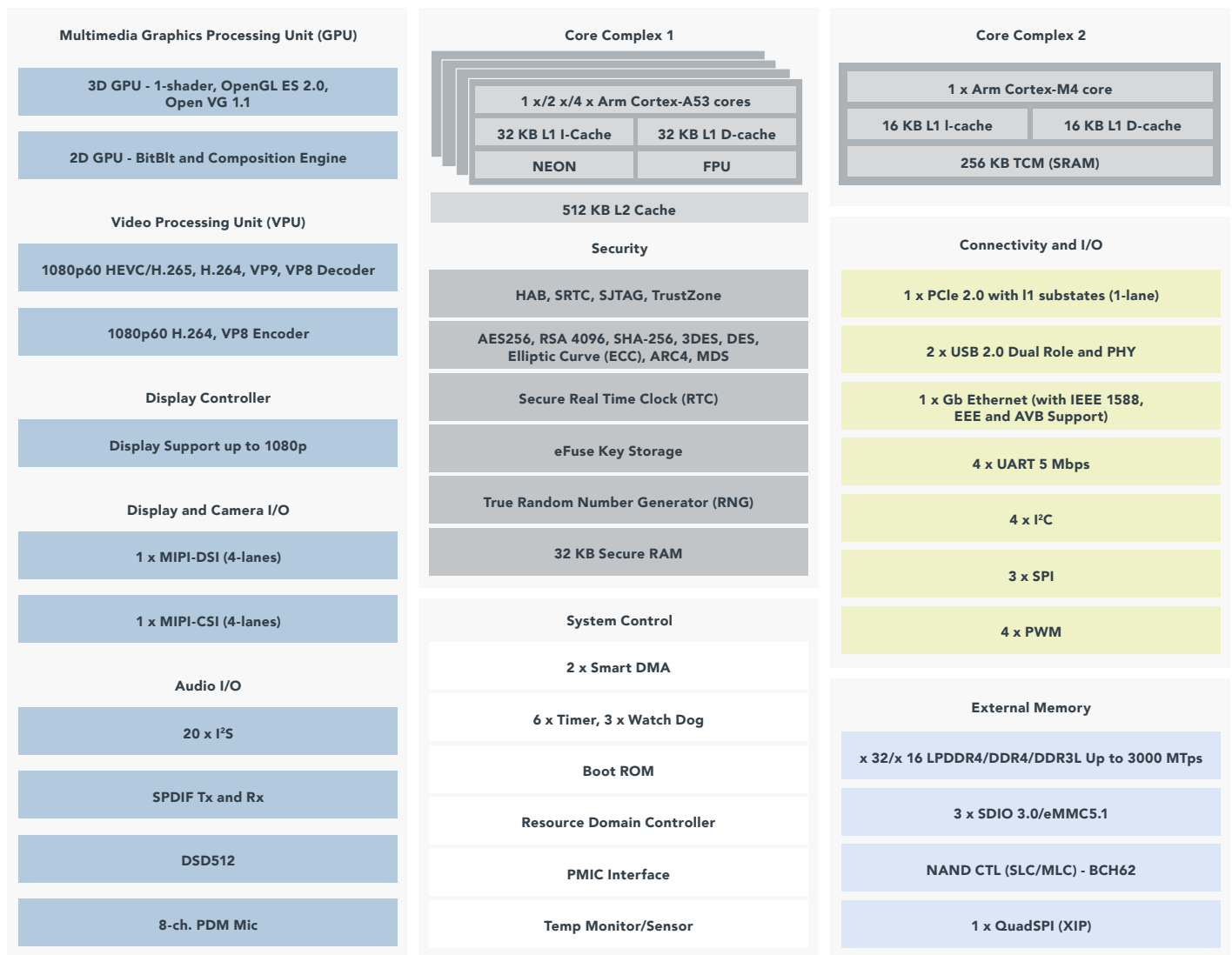


Figure 2.8. Edge computing hardware architecture for advanced processing



Edge computing devices such as the one shown in Figure 2.8 require more advanced software enablement. Figure 2.9 shows a software architecture common in many edge processing devices. This architecture contains rich OSs such as Linux® and Android, and software enablement for the advanced hardware capabilities shown in Figure 2.8.

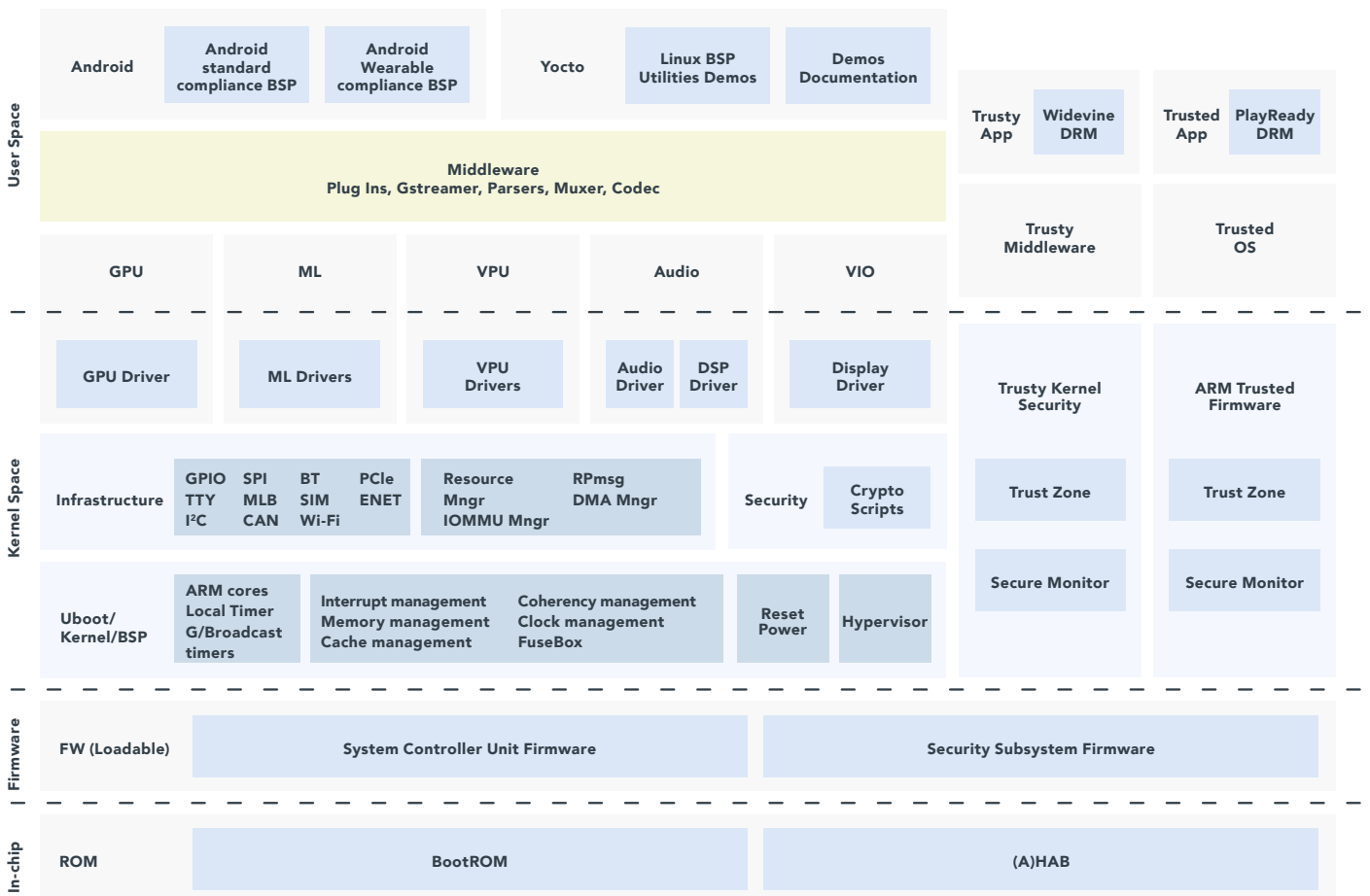


Figure 2.9. Edge processing software architecture for advanced hardware capabilities

For certain market segments, this software enablement can be customized. For example, Figure 2.10 shows software customized for industrial applications, with support for more advanced industrial protocols such as time-sensitive networking, EtherCAT and PROFINET.

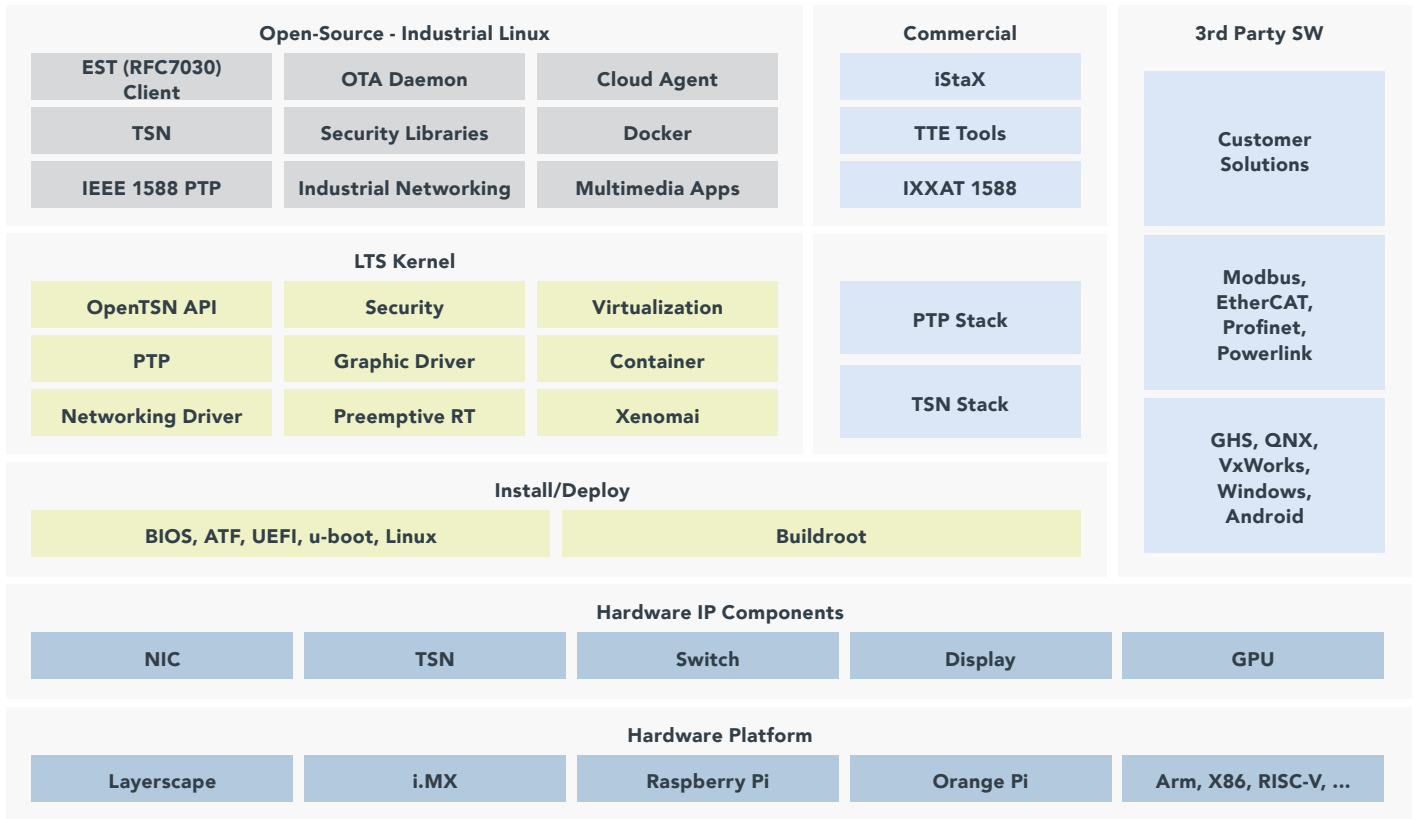


Figure 2.10. Edge computing industrial software architecture

EDGE COMPUTING DEVICE LIFE CYCLE

Another important component of an edge processing system is the ability to manage the life cycle of the edge computing devices deployed in the field. This includes:



Figure 2.11 shows a high-level flow of this device life-cycle management process. See Chapter 6 for more details on life-cycle management.

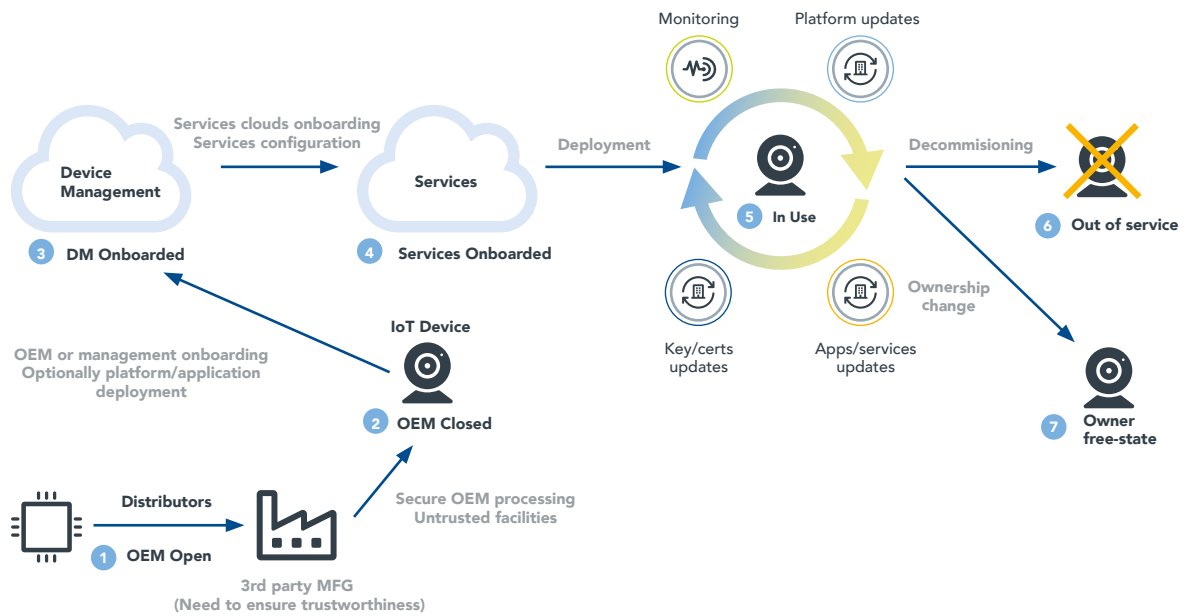


Figure 2.11. Device management in an edge computing environment

OPEN-SOURCE EDGE COMPUTING

In addition, some open-source edge computing platforms support general edge computing applications. An example is EdgeX (see Figure 2.12), which is vendor-neutral software that interacts with the capabilities in the real world (sensors, actuators and other objects). It's essentially a middleware layer that manages physical sensing and actuating and the cloud IT systems.

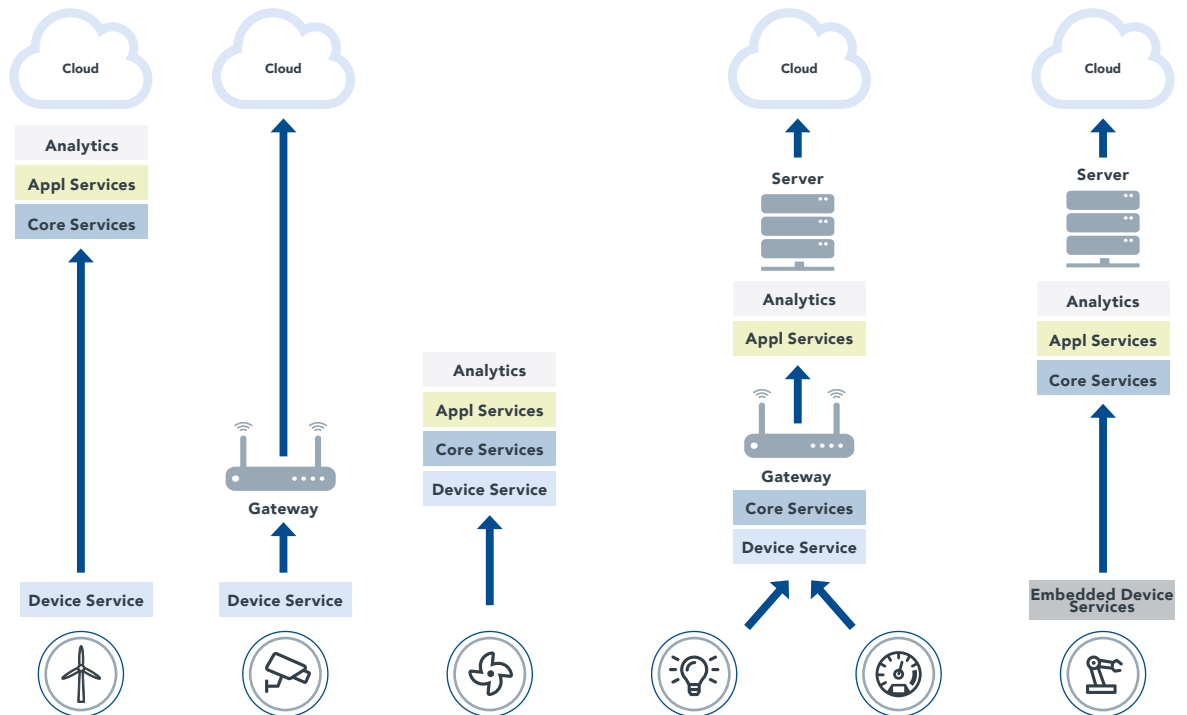


Figure 2.12. The EdgeX open-source edge computing software platform

COMMUNICATION PROTOCOLS FOR EDGE COMPUTING

Once data is at the edge, messages must be organized via protocols before communicating with the cloud. These “cloud” protocols include Message Queue Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol (CoAP) and, of course, Hypertext Transfer Protocol (HTTP).

Edge computing also requires a rich set of run-time capabilities. One category referred to as “dataflow” includes receiving and processing sensor data using the communication protocols in the previous paragraph. Once data is received from the sensors and end node devices, it is filtered, cleansed and transformed using various control protocols and then aggregated, among other things.

For example, in a smart building application, temperature sensors distributed throughout the building send temperature information to the edge device using the Bluetooth LE communication protocol. Software functions on the edge device implement business rules that determine, based on the received data, that the air conditioning must be turned on.

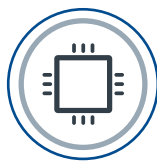
The software function sends air temperature commands to the air conditioners using the Zigbee communication protocol. Temperature information is also sent to the cloud for general analytics purposes. The edge computing software also does this using a cloud-based application programming interface (API) call.

Security constraints, such as authentication, must be applied as well.

DECISION-MAKING AT THE EDGE

Many decisions must be made close to the device for latency and security reasons. Sending information to the cloud for every decision is not practical. A more obvious example of this is real-time anomaly detection on a factory floor. This capability helps manufacturers adjust robots to optimize production capacity and yield as well as to identify potential defects as soon as possible so that any affected equipment can be removed and serviced immediately. This is usually implemented using reactive monitoring, which is different from the broader function of data analytics, a mostly passive and non-real-time process.

Many other applications featuring “independence from the cloud” offer these same benefits. EdgeX has four middleware services:



Core



Supporting



Application



Device

This framework can be used with the device architectures mentioned earlier in use cases including:



Smart building optimizing efficiency across facility



Factory automation interoperability across sensors, machines and robots



Water treatment loss detection in real-time

SUMMARY

Edge computing essentially offloads computation and storage from a centralized cloud to the network's logical extremes. Edge architecture is a distributed computing architecture that includes both hardware and software. Edge devices can connect to other edge devices and may ultimately connect to a centralized system or cloud. In addition to the hardware, communication protocols, enablement software and device life cycle management are important factors to edge computing architecture.



Chapter 3

EDGE COMPUTING SECURITY

CONTRIBUTORS

Silvano Di Ninno, Security Software Engineer, NXP Semiconductors

Nicusor Penisoara, Senior Director, Security, Graphics & Machine Learning Software R&D, NXP Semiconductors

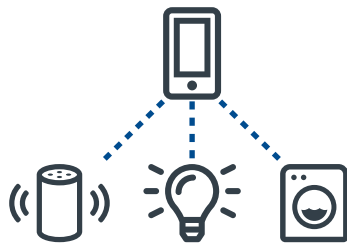
Marc Vaclair, Senior Security System Architect, Technical Fellow, NXP Semiconductors

The security of edge devices is increasingly important as more of them are incorporated into systems. This chapter explores functional and platform security methods to address weaknesses pinpointed by risk assessments conducted during the design stage.

SECURITY: A HOLISTIC SYSTEM PROPERTY

Future systems will encompass billions of interconnected devices. They will form an attractive target for attackers. Edge devices are part of these systems. They process data closer to where it is generated while connecting with remote and cloud-based services. They collect raw data from sensors, analyze and extract relevant information to be delivered to the cloud and present it to local users. In most cases, this information contains sensitive data that needs to be protected. Also, some edge devices have actuators to control utilities and machinery that can be misused; therefore, the commands received from remote services need to be verified as authentic and originating from the proper controlling entities. This makes them interesting targets for the attackers.

Security is a holistic system property¹; security is not an add-on. A system is as secure as its weakest component that an attacker can reach. Edge devices will play several roles in system security:



Edge devices are privileged targets (the return on investment for an attacker is higher for an edge device controlling many devices), and these devices have to be protected accordingly.



They can help improve system security by providing the necessary isolation of less secure subsystems or by delivering secure services to them.

Most of the fundamental concepts used to secure networking devices work for securing edge devices as well:

- **Data in transit**
- **Data at rest**
- **Access control mechanisms**

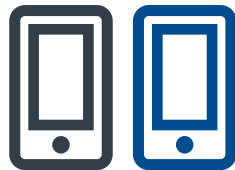
However, although a data-center infrastructure (where cloud services are typically deployed) offers inherent physical protection for the data being stored and processed, edge devices are placed in locations where little or no physical protection is guaranteed. This implies that additional protection and some level of intrusion-detection capability needs to be deployed for edge devices. At the system-on-chip (SoC) level, these needs must be supported by the following hardware capabilities combined with specific software mitigation techniques that protect and react against such attacks²:

- **Root of trust (RoT)**
- **Tamper detection**
- **Secure boot**
- **Secure enclaves**

¹Recommended reading: "From the Internet of Things to the Internet of Things" (<https://www.nxp.com/docs/en/white-paper/NXP-FROM-IOT-TO-IOTRUST-WP.pdf>)

²Readers interested in more information on the security features in SoCs for improving system security can consult the NXP white paper on security primitives here: https://www.nxp.com/docs/en/white-paper/SEC_PRIMITIVES_WP.pdf.

The edge device and its data protection capabilities make up one critical part of the edge computing security story, but protecting the network and cloud services from edge devices transformed into remote controlled weapons is also important. Edge devices need to process increasingly larger amounts of data in a variety of new ways. To address that, both compute power and the available network bandwidth have grown over the years. This trend will only continue with the emergence of edge artificial intelligence (AI) and the deployment of 5G and faster Wi-Fi technology. Given the ever-tighter interaction between edge devices and the cloud infrastructure, the risk of disrupting the cloud's services is increasing through the following methods, among others:



Device cloning



Denial of service (DoS) attacks

Mirai, a malware able to command and control hundreds of thousands of devices to mount denial of service (DoS) attacks, and its variants have inflicted massive damage multiple times recently.



The security for edge devices is two-fold³:

- **Functional security** — These security primitives (most of the time the cryptographic features) ensure that sensitive data remains secure (confidentiality, integrity, authenticity) and private when required. For example, using encrypted messages between two edge devices is a functional security feature.
- **Platform security** — These security primitives ensure that the implementation of the functional security remains secure in the presence of remote or sometimes local attackers. For example, protecting the secret key used to encrypt and decrypt messages in each of the two communicating edge devices is a platform security feature (one potential implementation of this security feature is secure storage with access control and isolation so that remote attacks on other parts of the software of the edge device under attack cannot recover this secret key). But if local attacks (i.e., those with physical access to the edge device) are considered part of the threat model, more advanced hardware security features such as side-channel and fault-attack resistance should be implemented.

For both kinds of security, some fundamental cryptographic features can be used to secure edge nodes. Through cryptographic protocols and key management techniques, in-transit and at-rest data can be protected and the edge device booting and running authentic, authorized firmware and software can be ensured. Additionally, a unique identity for a device can be constructed to verify and protect it on the edge device side while using it on the cloud infrastructure side for access control. This unique identity can help detect if a device was corrupted and needs to be isolated to avoid DoS attacks.

³This chapter only skims the security topic. A detailed introduction to security can be found in "Security Engineering: A Guide to Building Dependable Distributed Systems" by Ross Anderson, Wiley, 3rd Edition, 2020.

The ability to create and protect multiple software execution environments on a single SoC is another key functionality required on more advanced edge computing devices. It's needed to isolate sensitive software processing. To integrate applications with third-party software stacks as part of an edge computing offering, secure communication between those different software stacks must be guaranteed.

Finally, security features and countermeasures need to be seamlessly integrated into software stacks so they can be deployed easily by applications running on the edge device. Using a standard and/or well-established open-source application programming interface (API) is essential to maximize software reuse.

CRYPTOGRAPHIC FEATURES

Cryptographic features⁴, relying on cryptographic algorithms, ensure functionality by providing one or more of the following capabilities in a system:



Integrity — The data received is identical to the data sent.



Confidentiality — A third party listening to the communication cannot understand the message.



Authenticity — The receiver of a message can verify the authenticity of the message.



Nonrepudiation — The sender of a message cannot deny having sent it.

Algorithm	Integrity	Confidentiality	Authenticity	Nonrepudiation
Hash	✓			
Symmetric cryptography		✓		
MAC	✓	✓		
Public key cryptography	✓	✓	✓	✓

Table 3.1. Security properties of algorithms

⁴To learn more about cryptography, read "Cryptography — Theory and Practice" by Douglas Robert Stinson and Maura Paterson, Chapman and Hall/CRC, 4th Edition, 2018. This is a good introduction to the theory of cryptography, while "Cryptography Engineering — Design Principles and Practical Applications" by Niels Ferguson, Bruce Schneier and Tadayoshi Kohno, Wiley, 2010, is a more practical introduction to the use of cryptography.

HASH FUNCTION

A hash function is a cryptographic function that can take any amount of data as input and provide a fixed number of bits as output. It is a function (see Figure 3.1) for which generating the output from the known input can be achieved fairly easily, while retrieving the input data based only on the output result is infeasible. Another property is that easily creating two different input data points that give the same output result is infeasible as well.

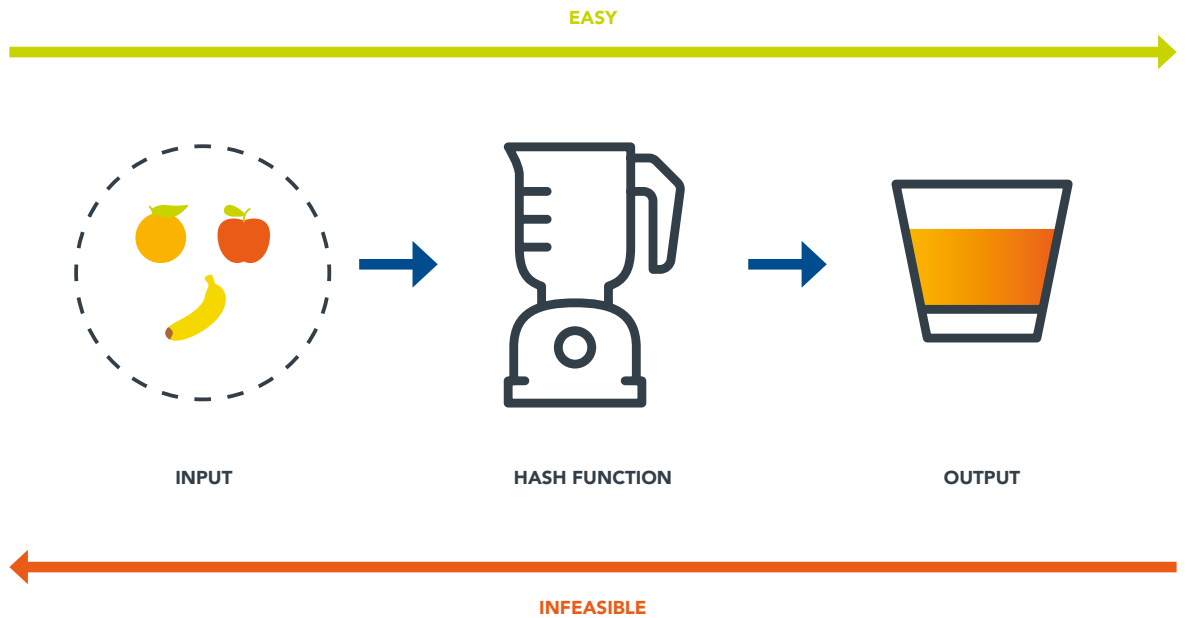


Figure 3.1. Hash function one-way property

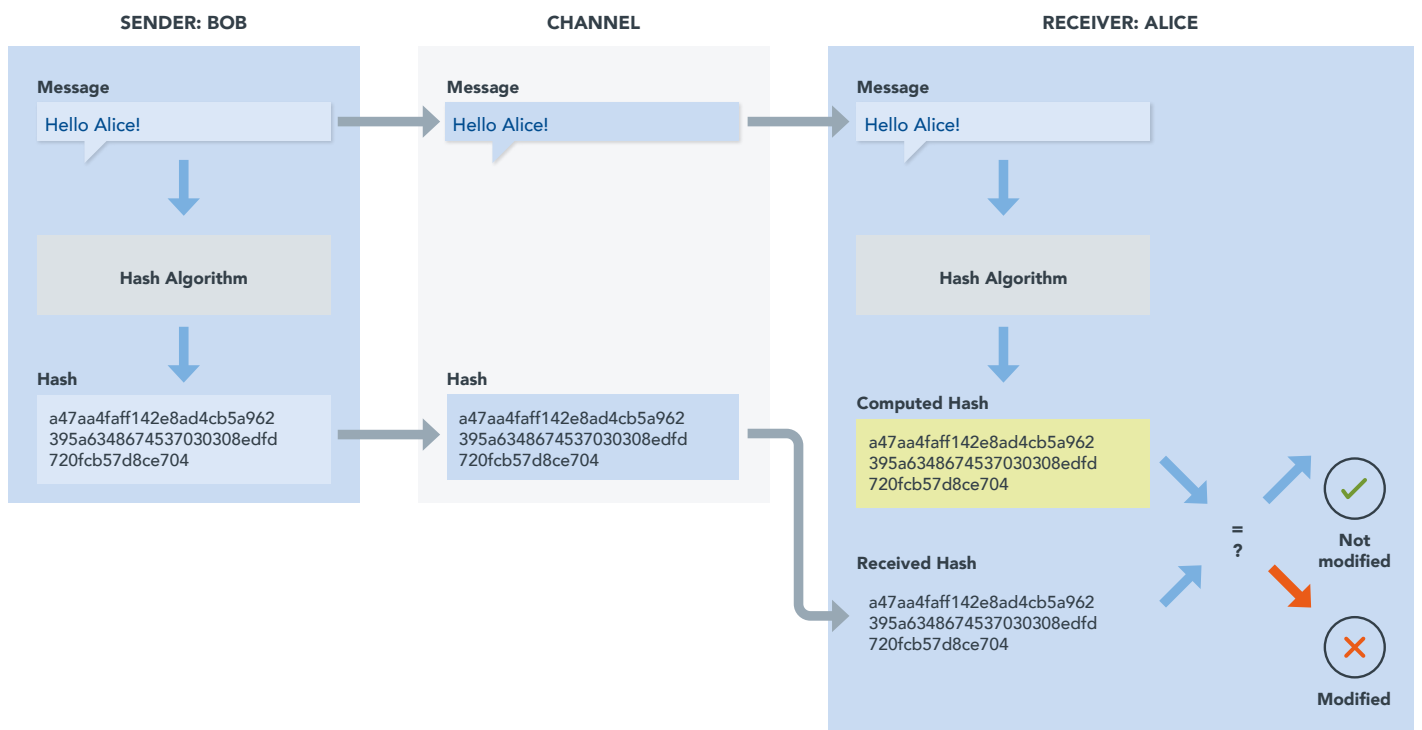


Figure 3.2. Use of hash function to guarantee the integrity of messages

This function is often used for the following (see Figure 3.2):



Integrity checking

This use validates that the data received was not corrupted. The common example is a file transfer or a download from the internet. Downloaded files are sometimes associated with the results of a hash function run over the content of the file, for example, the outcome from the `sha256sum` hash command. The user downloading the file can run the same type of hash function with a downloaded file as an input. The user then compares the output with the `sha256sum` value contained in the file that was associated. If there is a match, it is highly probable the data wasn't corrupted.



User password storage

Storing passwords on a server is potentially dangerous; passwords can be lost if the server is hacked. Instead of storing the password in plain text, the hash of that password with additional fixed data (known as "salt;" the salt is added to prevent rainbow dictionary attacks) is stored. When the user is prompted to enter the password, the same process is used (the verifier software collects the password from the user, collects the associated salt from the system and computes the salted hash of the password), and the resulting salted hash is compared with the data stored on the server. For additional user protection, the user password never appears in plain text on the server.



Signature generation and verification

Unlike integrity checking, here the hash function result, in association with other cryptographic operations, is used to validate not only the data's integrity but also the origin of the data. These cryptographic functions used alongside the hash function are part of the public key cryptography. See the "Public Key Cryptography" section later in this chapter.



Proof of work

Bitcoin uses a hash-based mechanism as proof of work.

Several hash functions are available. Some like MD5 and SHA-1 are considered obsolete and weak. They should be avoided, although some are still used for weak file integrity checking or file identification. In these cases, don't rely on MD5 or SHA-1 for authenticity. As of today, the standardized and recommended functions are the ones from the SHA-2⁵ families; these are mandatory if used for signature and signature verification.

⁵Reference: https://en.wikipedia.org/wiki/Secure_Hash_Algorithms



SYMMETRIC KEY CRYPTOGRAPHY

Symmetric key cryptography (see Figure 3.3) uses a secret key shared by the sender and receiver to encrypt and decrypt data. In other words, Alice and Bob use the same key both for encryption and decryption. This function involves relatively fast operations, so users prefer it when exchanging significant amounts of data. The main disadvantage of this function is that the secret key used to encrypt/decrypt data (i.e., the symmetric key) must be shared between the communicating parties appropriately to ensure it does not fall into the wrong hands. Securely generating and sharing a symmetric key is problematic when exchanging data between two remote parties. This problem is exacerbated when confidential information sharing among a larger number of participants is needed. Also, the base encryption/decryption function provides only data confidentiality. This function is often used alongside public key cryptography for key distribution and authentication functions for integrity verification. The AES⁶ family (AES-128, AES-192, AES-256) is the most common among the many symmetric key algorithms.

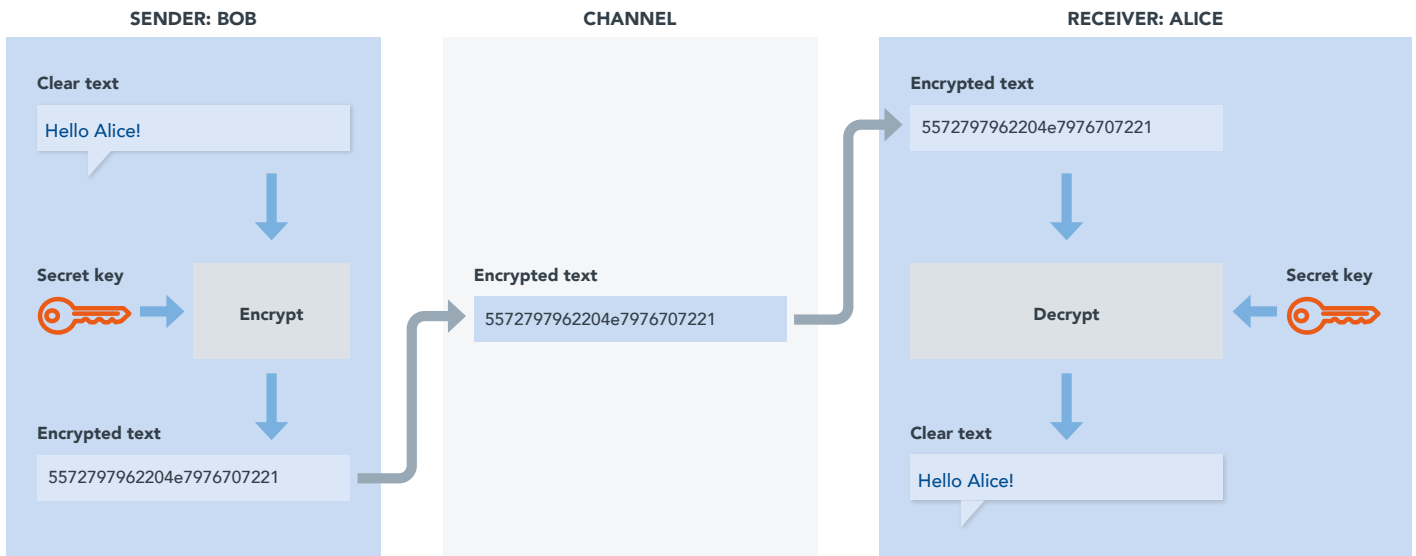


Figure 3.3. Use of symmetric key cryptography to ensure the confidentiality of messages

⁶Reference: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Symmetric key cryptography is often used in the following cases:



Protecting data at rest

This is used when the data is not moving from one system to another; instead, it's in physical storage such as a hard disk or a flash drive. Data is encrypted so it's protected when the system is off and the data is not being used. On a Linux® OS, dm-crypt can be used for this purpose; on a Windows® OS, BitLocker can be used. Recent SoCs provide on-the-fly automatic encryption and decryption as it loaded or saved to the various memory areas (RAM, flash and so on). For this purpose, the Prince algorithm is used because it has low-latency properties that do not introduce delays in memory accesses.



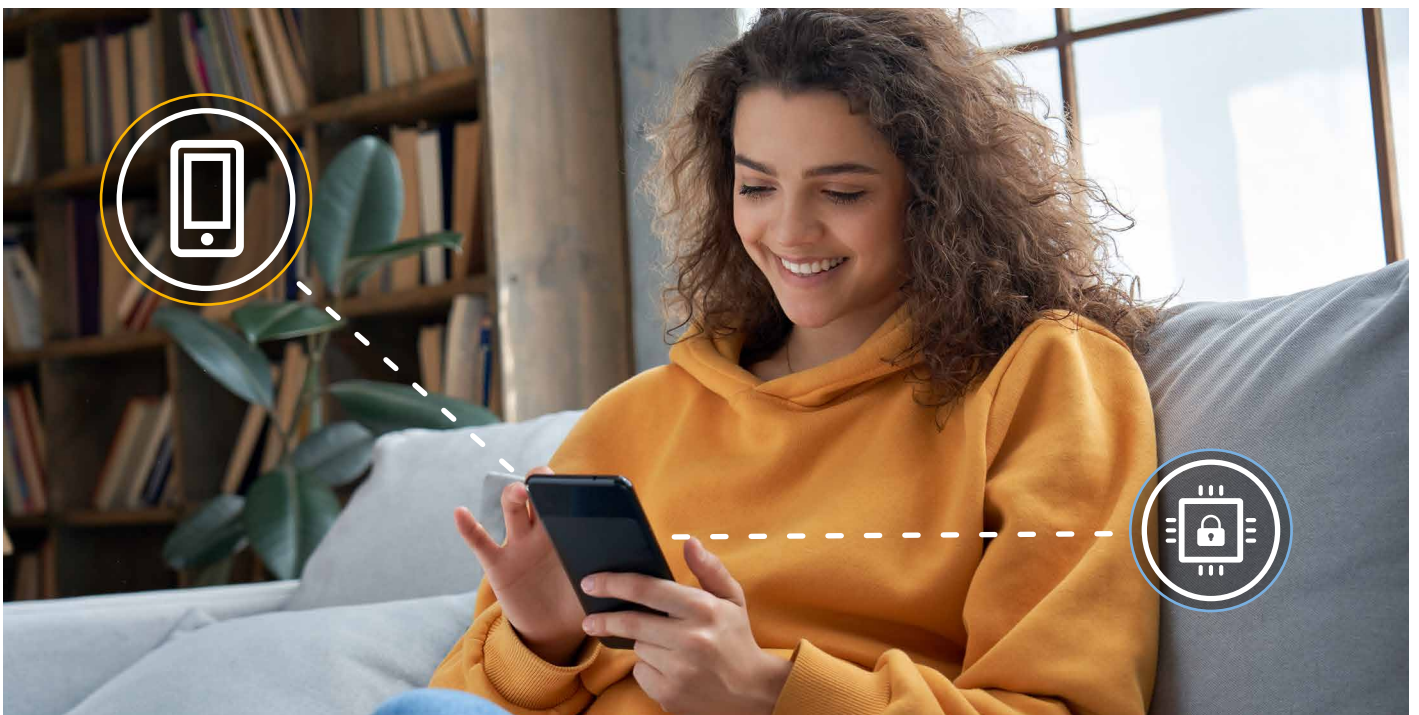
Wrapping keys

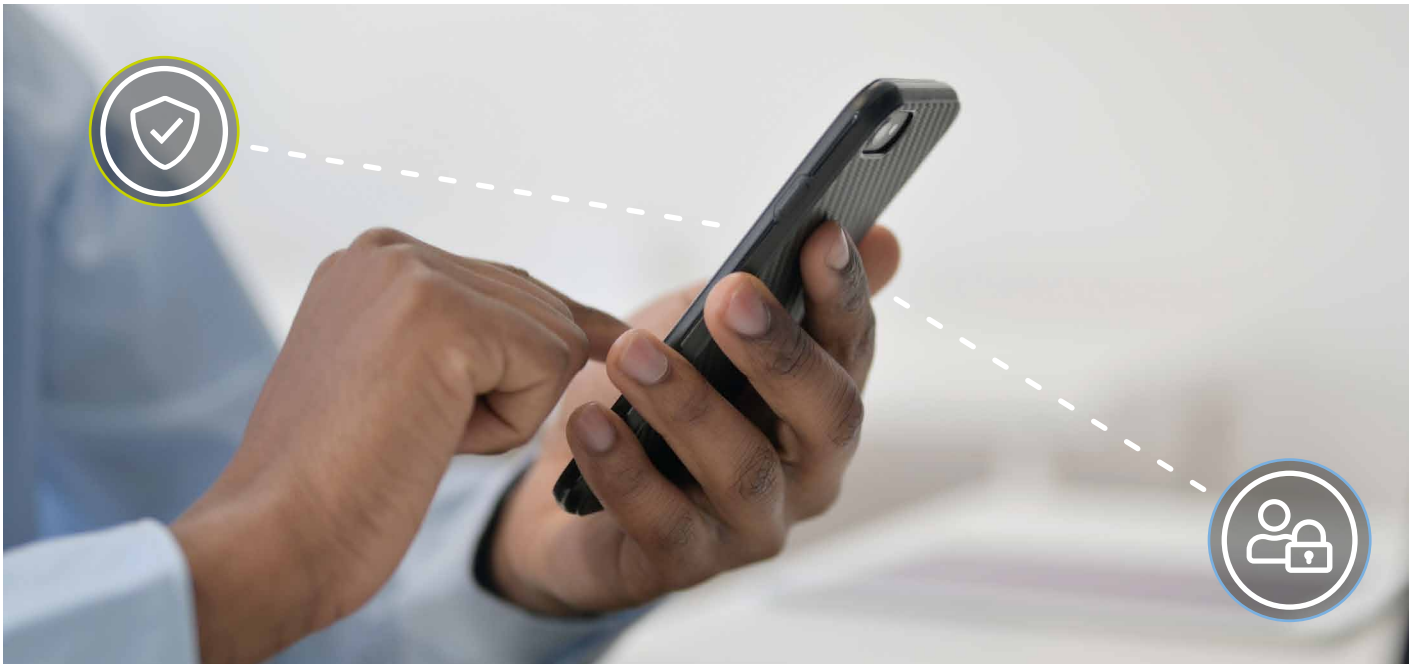
This use case is similar to protecting data at rest, but the data protected is another key (i.e., the data stored or transmitted is a secret key encrypted with another secret key). Selecting an algorithm stronger or a key longer than the algorithm associated with the wrapped key or the length of the wrapped key, respectively, is important when wrapping keys.



Protecting data in transit

Data exchanges between two or more systems are usually implemented through a connection using a communication protocol on wired (e.g., Ethernet) or wireless (e.g., Wi-Fi) network links. In this case, protocols commonly used are Internet protocol security (IPsec), transport layer security (TLS) or secure shell (SSH). But the connection also can be local within a system, such as an SoC connected to a secure element or trusted platform module (TPM) through an I²C interface. In this case, a specific protocol (sometimes a proprietary one) is used.





MESSAGE AUTHENTICATION CODE ALGORITHM

The message authentication code (MAC)⁷ algorithm is a cryptographic function for integrity and authenticity checking. This function is like the hash function with the addition of a shared secret key (see Figure 3.4). Anyone can verify a hash, but only the ones who know the shared secret can verify a MAC. MAC algorithms are categorized in two families: hash MAC uses one of the hash algorithms as an underlying function for the operation, and cipher-based MAC uses symmetric key cryptography (e.g., Advanced Encryption Standard or AES) as an underlying function.

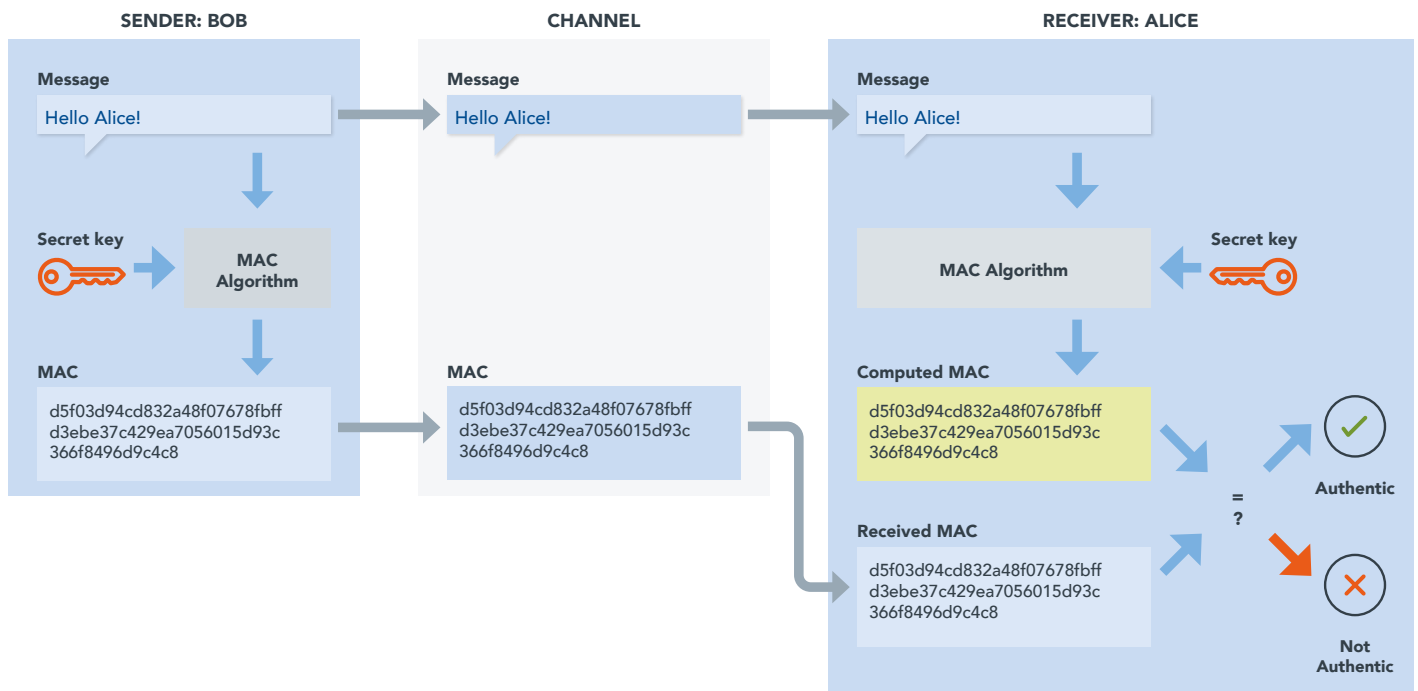


Figure 3.4. Use of MAC to guarantee the authenticity of a message

⁷Reference: https://en.wikipedia.org/wiki/Message_authentication_code

This function is often used for the following:



Data in transit

MAC algorithms are commonly used alongside a symmetric cryptography algorithm to authenticate the data being exchanged. This is the case for the network protocol IPsec or TLS. It also can be used locally within a system, for example, the connection between an SoC and an enabled multimedia card (eMMC) chip. To read and write to eMMC storage, a specific protocol is used, and frames are exchanged between the SoC and the physical storage for reading and writing data. In a recent version of the eMMC specification, a special partition was introduced for security purposes. A replay protected memory block (RPMB) and the frame sent and received are protected by an authenticated field. The key used to generate the MAC is stored in the eMMC chip, and only the software in the SoC knows that the key can read and write data in that RPMB partition. Modern protocols use Authenticated Encryption with Augmented Data (AEAD) algorithms that combine the MAC computation with the encryption computations to simultaneously provide confidentiality and authenticity in one operation (e.g., AES in CCM).



Key derivation function (KDF)

A MAC algorithm also can be used as the underlying function to derive a new key from an existing one. This is a desirable feature in secure systems that requires a key being used only for one purpose (e.g., a key used for encryption cannot be used for authenticity). A simple representation of this function is "KDF (IKM, salt)." KDF is the function for the derivation. It can be, for example, an HMAC function using an SHA256 algorithm. The input key material (IKM) is the secret part of the input function (typically an existing secret key). The salt is a nonsecret input data point. The key derivation technique is frequently used in resource-constrained devices where a single secret key (the IKM) can be stored in fuses — embedded in some ROM in SoC die or a combination of both — and used to generate other keys for a variety of purposes in the system.



PUBLIC KEY CRYPTOGRAPHY (ASYMMETRIC CRYPTOGRAPHY)

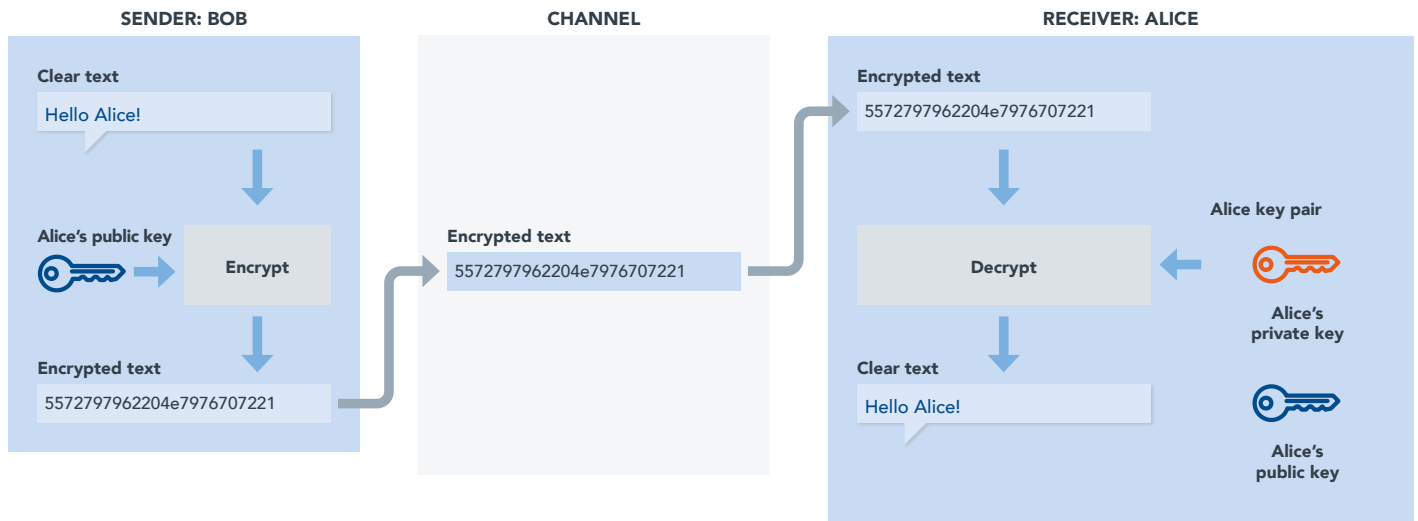


Figure 3.5. Use of asymmetric cryptography to protect the confidentiality of small messages

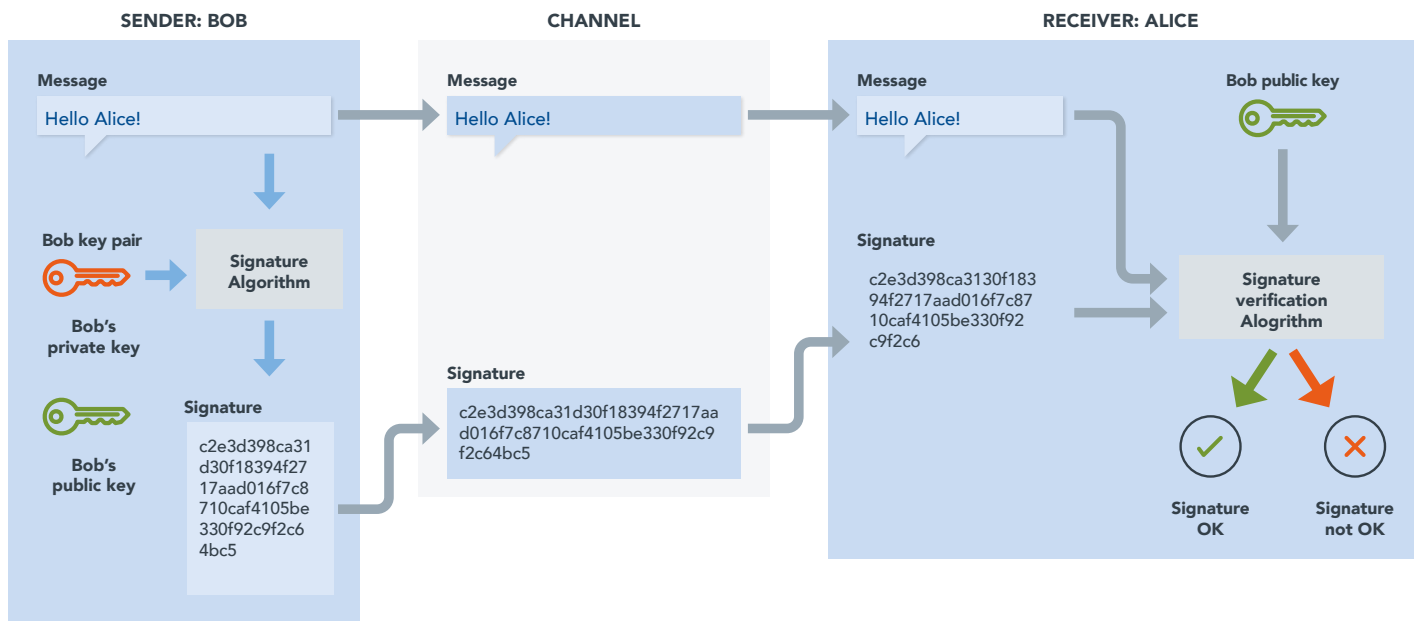


Figure 3.6. Use of asymmetric cryptography for digitally signing messages

As opposed to symmetric key cryptography that uses a single key for both encryption and decryption operation, asymmetric cryptography, which is also known as public key cryptography (PKC), is using a key pair. Each participant has a key pair made of a private key (only known by the owner of the key pair) and a public key (known by all participants). Those keys can be used in different ways. The following two major uses of asymmetric cryptography are illustrated in figures 3.5 and 3.6.



Encryption to provide integrity and confidentiality

A participant encrypts a message using the public key of the intended receiver of the message, and only the intended receiver is capable of decrypting the message using their private key. For example, the Rivest-Shamir-Adleman (RSA) encryption algorithm can be used for this purpose.



Digital signature to provide authenticity and nonrepudiation

A participant signs a message using their private key and sends the message together with the signature. All other participants can verify the signature using the public key of the sender. For example, the RSA algorithm and an Elliptic Curve Digital Signature Algorithm (ECDSA) can be used for this purpose. As mentioned earlier, the digital signature algorithm relies on a hash function; it is the hash of the document that is signed.

PKC has solved the fundamental problem of allowing two entities that have never been in contact before to establish an authentic and confidential message exchange between them.

The PKC function is more computationally intensive than the symmetric functions, so it is commonly used to process small amounts of data and support the following use cases:

- **Establishment of communication channels** — The digital signature capability allows participants to identify the parties in the communication, while the encryption capability is used as part of a key-exchanged mechanism to agree on a new key to be used later for the communication. The TLS protocol is an example of this usage.
- **Secure boot mechanism** — The digital signature is used to authenticate that a binary is genuine. The manufacturer signs the software using their private key and distributes their public key so that any other entity can verify the genuineness of the software.
- **Public key infrastructure (PKI)** — When PKC is used, the identities of the participants need to be bound to their public keys and the freshness of those keys needs to be ensured. This is the role of PKI: to certify these links by emitting certificates and revocation status information. See the “Public key infrastructure” section later in this chapter.
- **Attestation services** — Device attestation means that a device has been provisioned by the manufacturer with a key pair unique to the device. Whenever an entity of the network in which the device is connected wants to challenge the device for genuineness, the entity can send a challenge to the device and the device can send back a signature of the challenge proving the knowledge of the right private key and, therefore, their genuineness.

HARDWARE ACCELERATION

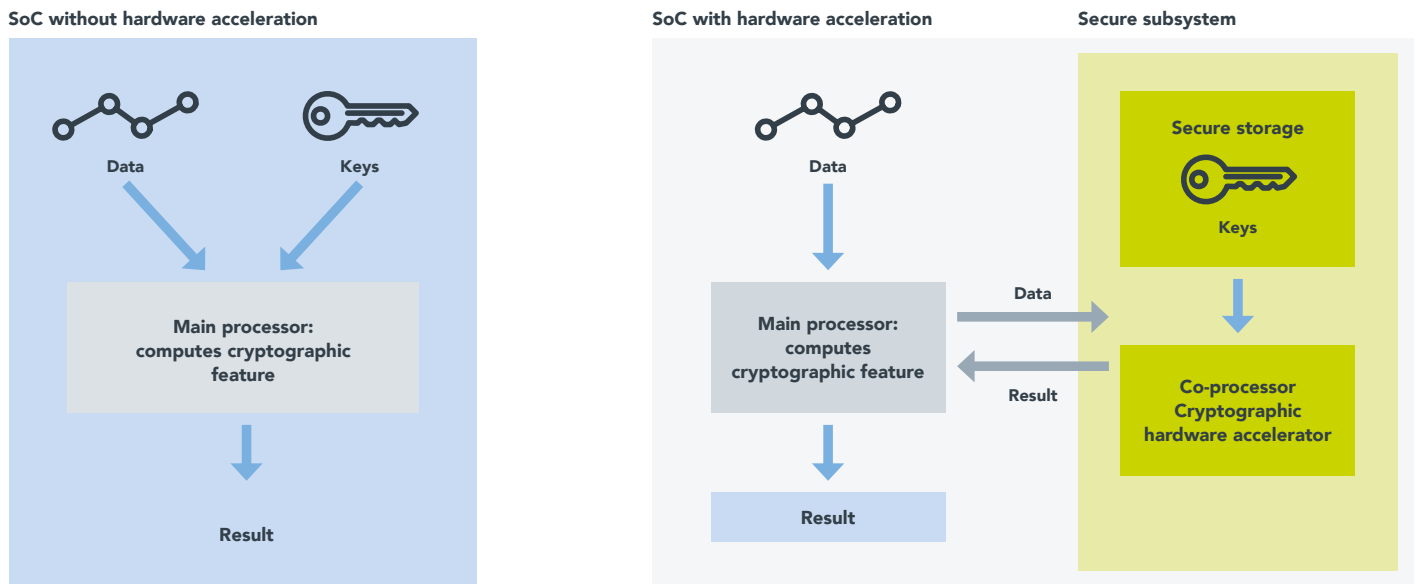


Figure 3.7. Hardware acceleration

With hardware acceleration, an application can offload certain computing tasks to specialized hardware components within the system. To optimize performance and keep the power budget low, the cryptographic functions discussed earlier are often accelerated in hardware on many edge computing devices.

Depending on the edge computing device, one or more hardware accelerators may be available. In some devices, the main core processor is extended with special accelerated cryptographic operation codes; on many other edge devices, one or more additional coprocessors are integrated in the SoC.

For example, the NXP i.MX product family has a hardware IP called the Cryptographic Accelerator and Assurance Module (CAAM) for cryptographic computation hardware acceleration. Beyond the acceleration capabilities, the CAAM offers special key-wrapping functions that protect sensitive keys from being exposed. This key-wrapping mechanism relies on a collection of fuses that are not accessible by the main core of the SoC. This implies that even if an attack succeeds in corrupting the behavior of the main core, the attackers don't have access to the wrapped keys because they don't have access to the internal mechanisms of the CAAM. The key store constructed around these functions is not accessible by the main core of the SoC even if the software running on the SoC is compromised.

Cryptographic hardware accelerators are also hardware isolation devices. The accelerator has its own processing unit, including registers, and its own local memories. These resources are not directly accessible by the other cores in the SoC.

The use of hardware accelerators, which come with device drivers, is integrated in cryptographic software libraries to abstract the cryptographic computations from the point of view of the application software.

Hardware accelerators also provide a guaranteed and sometimes certified implementation of the various cryptographic algorithms. In addition, they can incorporate additional security countermeasures to prevent side-channel attacks or fault-injection attacks that are trying to access the cryptographic credentials used during the cryptographic computations.

HARDWARE RANDOM NUMBER GENERATORS

The security of many security primitives and, more specifically, many cryptographic algorithms and protocols relies on the true randomness of some of the data processed during the computations. For example, a network protocol for a secure channel is secure only if the challenges sent between the communicating parties are truly random (if that's not the case, an attacker can guess the session keys and listen to the communication). The digital signature of a message using the ECDSA is valid only if the ECDSA can rely on a truly random number during the signature computation (if that's not the case, an attacker can guess and compute the private key of the signer and forge new signatures in their name). The only way to produce truly random numbers is by the addition of a hardware true random number generator (TRNG). This generator is used by the software. Modern SoCs incorporate a hardware TRNG that's accessible through protected interfaces. The TRNG is mostly used to seed a pseudorandom number generator because the amount of entropy generated by a TRNG per time unit is limited while some protocols require more random material per time unit.

PUBLIC KEY INFRASTRUCTURE

A Public Key Infrastructure (PKI) defines and certifies the binding between public keys and entities, individuals and/or organizations. It facilitates the exchange of data between two or more entities that are using PKC by introducing one or more third parties that can attest the identities of those entities and the relationship to the public keys they advertise. This topic is becoming more relevant in edge computing because in some architectures, edge devices can act as these third parties. They have the necessary implementation security features that ensure they cannot be abused.

A digital certificate can be used to represent this binding between an identity and a public key. The most common is the X.509 v3 certificate⁸. As shown in Figure 3.8, it contains information about not only the identity of the organization linked to the key pair but also who has issued and signed the certificate as well as a validity period.

Version
Certificate Serial Number
Period of Validity
Subject Name
Public Key Information
Issuer Name
Extensions e.g., Key Usage
Signature Algorithm Identifier
Signature

Figure 3.8. Example of x509.3 certificate content

⁸Reference: <https://en.wikipedia.org/wiki/X.509>

The issuers of those certificates are called certification authorities (CAs). A CA is responsible for checking the link between an identity and a public key along with additional information before it emits a digital certificate. Either the certified entities receive these certificates or the certificates are published in public repositories accessible by anyone. When another entity receives a certificate, it can verify the validity of the certificate thanks to the knowledge of the public key of the certificate issuer (the CA that has signed the certificate) and the trust that the verifying entity has in this CA. The receiving entity can also check whether the certificate has been revoked between its emission and its current use. Having one CA for all purposes is neither feasible nor desirable for many reasons: scalability, availability, confidentiality and so on. That's why instead of having one CA, a PKI is made of several CAs that are commonly organized in a hierarchy. A top CA certifies subauthorities that, in turn, certify entities. CAs also commonly cross-certify each other's certificates.

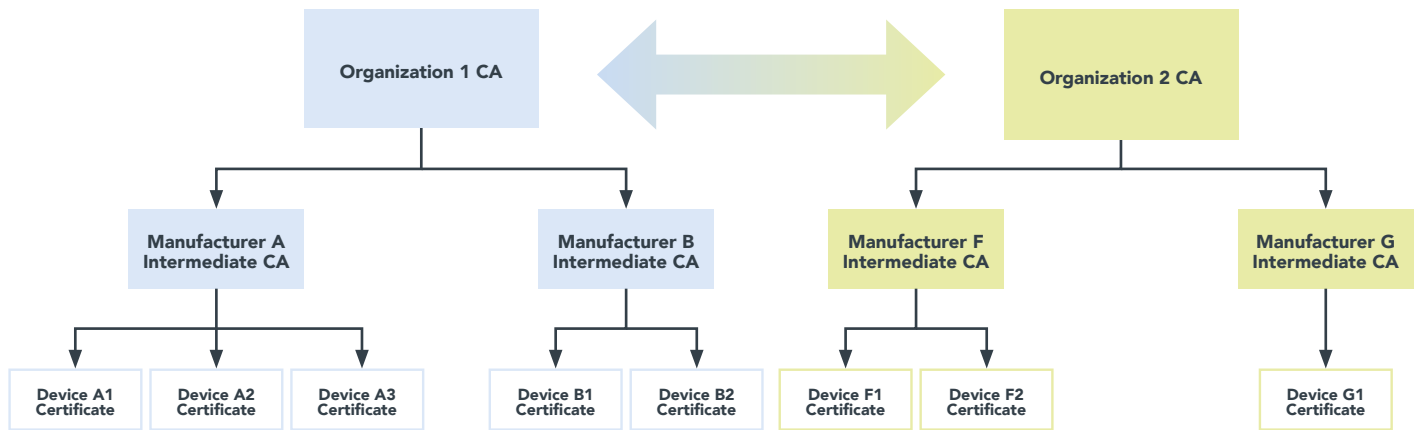




Figure 3.9. Example of a PKI infrastructure

Certificates do not have an infinite life; they have a start-of-use date and an end-of-use date. In addition, if for any reason (disappearance of the owner of the certificate, security breach of the private key associated with a public key and so on) the certificate is revoked, the CA publishes this end of validity. If the revoked certificate is presented in a digital transaction, it can be rejected by an entity that first checks whether it has been revoked.

A PKI is a set of hardware and software components as well as defined policies. It provides the following services:

- 
Creation of public/private key pair
- 
Digital certificate revocation
- 
Digital certificate creation and signature
- 
Digital certificate validation/verification

PKIs should not be perceived as a single entity that manages all available certificates. A PKI can be not only a public implementation at a large scale in data centers but also a private, single corporation or even an implementation in a single SoC.

The most common use of PKIs is with secure socket layer (SSL) certificates when browsing the internet with "https" URLs. Edge devices may also use "https" URLs to reach out to the internet. However, device life-cycle management and secure boot are other good examples of PKI use by an edge-connected device.

SECURE BOOT: CHAIN OF TRUST

A key aspect of building a secure system is guaranteeing that genuine binaries are running and that these binaries are trusted by the OEM that owns or supplies the system. The secure boot functionality provides this guarantee. Secure boot typically starts from an immutable memory area within the SoC, which is always inherently trusted and always runs correctly. This immutable part of the SoC and its corresponding features are sometimes referred to as a root of trust (RoT). PKC algorithms are often used as the underlying technology that supports the secure boot of a system.

The OEM or the entity responsible for the initial firmware running on the chip creates a public/private key pair. This operation takes place in a secure facility, and the private key is safely stored with controlled access. Often specialized hardware security modules (HSMs) are used to generate, store and apply this key (i.e., the key never leaves the HSM and only authorized users use it).

The public part of that key becomes part of the device's RoT. This is the first trusted component that is used to verify any further software. Care must be taken to guarantee the correct public key is placed in the devices; this step is usually implemented on the manufacturing line during a process called "trust provisioning" (see "OEM closed state" section in Chapter 6).

The private key is used to sign the firmware that will be executed on the secure system. This is a sensitive process, and the ability to sign the binary needs to be correctly controlled by the OEM. If, for example, mistakes are made and binaries that are not ready for production are signed, these binaries can later be used to compromise deployed edge devices.

Consider many of the heterogeneous i.MX applications processors. They use a mechanism similar to the secure boot process called High Assurance Boot (HAB), which is shown in Figure 3.10.

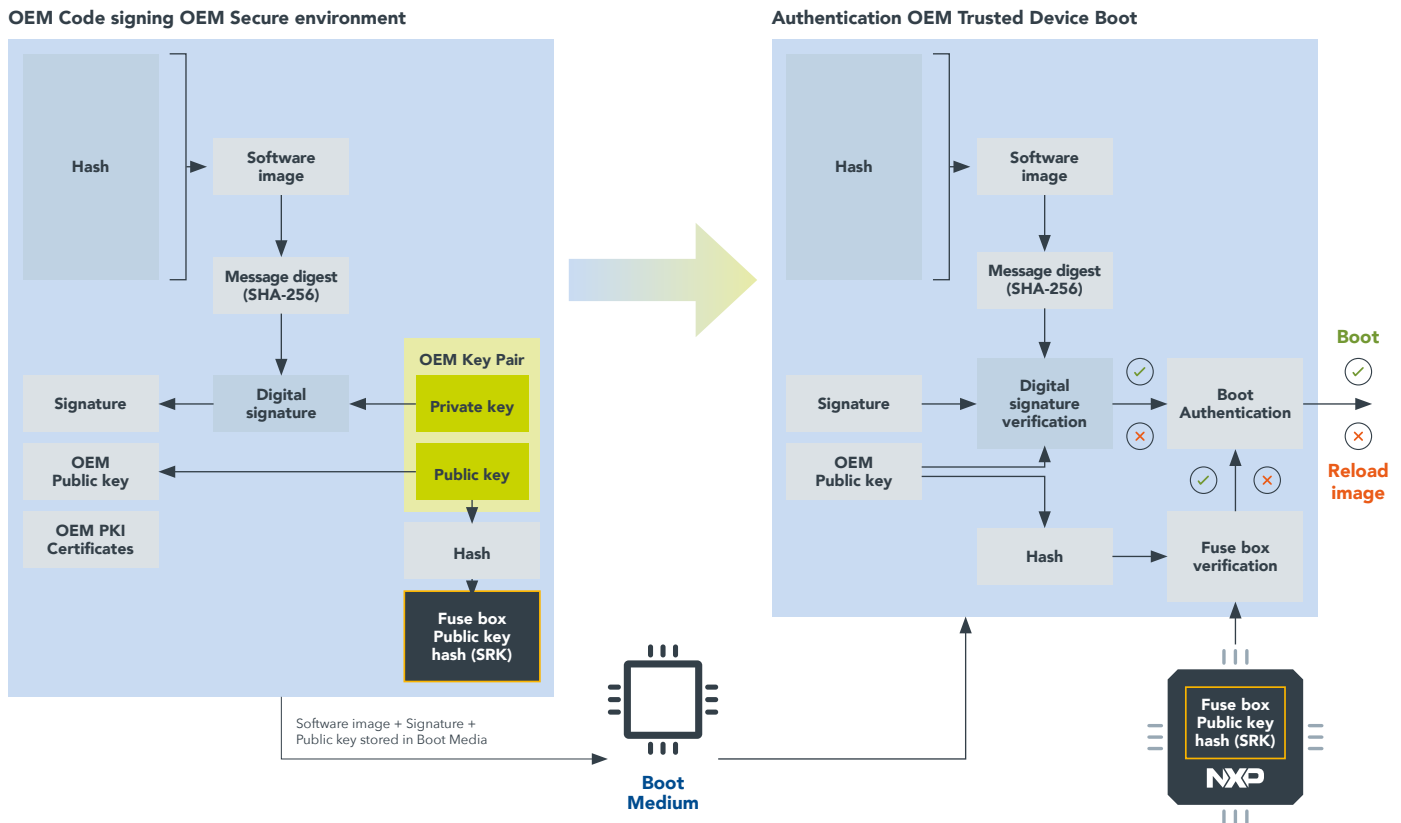


Figure 3.10. High Assurance Boot

SECURITY AND ISOLATION

Cryptographic algorithms and protocols have been around for decades. They are trusted to be robust and reliable if used correctly. However, as shown by several cyberattacks, logical errors (also known as bugs) have proven to be easy-to-exploit weak links in securing edge computing systems. A bug that can be used to compromise the security of a system is known as a vulnerability. The larger the code base, the higher the number of vulnerabilities the software running on the system has. Android and Linux are good examples of this reality (see also Table 3.2.)

A mitigation technique used in systems that require a certain level of security is isolating the storage and processing of sensitive information to dedicated hardware. For example, secure elements have been used to protect keys and sensitive data in the smart card industry. This is also true for the PC industry, which uses Trusted Platform Module (TPM) technology to guarantee the integrity of the software running. This has proven to be an effective method that still provides some of the best protection available today. However, for several use cases, this solution has not scaled well because the need for processing power for sensitive operations has increased over the years. Hardware extensions were integrated in the SoC's CPU to add isolation capabilities to the system while taking full advantage of the performance advances of the CPU. Those isolation techniques were first introduced in PC, server and mobile devices, but they are now equally applicable to the SoCs used in edge devices.

Modern SoCs implement one or more of the following isolation techniques:



Temporal isolation

The core of the processor can be in different execution modes; the execution mode determines the amount of access the core has to the SoC resources (see "Trusted Execution Environment" section later in the chapter).



Virtualization

The resources of the SoC are split, and a hypervisor controls access to them (see "Virtualization" section later in the chapter). The hypervisor provides access to several OSs and applications that are all being executed in isolation.



Multicore isolation

In SoCs with more than one core, the tasks can be split over the various cores and different security policies can be applied per core.



Secure enclaves

These are additional cores in the SoC that are secure/hardened and that have their own private resources not accessible to the rest of the SoC.



Discrete components attached to the SoC

These are additional hardware components that can be attached to the SoC. They include secure elements, memory protection units, secure memory management units and discrete hardware firewalls around the SoC.

The next section explores one of these isolation techniques: the trusted execution environment.

TRUSTED EXECUTION ENVIRONMENT

The first category of hardware extension is designed to isolate trusted functions. The concept of a Trusted Execution Environment (TEE) was introduced in the early 2000s to address these new requirements. The TEE implements a safe zone within the application processor. Large OSs such as Linux and Android potentially expose numerous vulnerabilities, so they are considered untrusted and execute outside the TEE in a zone known as the Rich Execution Environment (REE). Software that is carefully written with security in mind and is easily auditable runs inside the TEE. To satisfy those needs, CPU architecture extensions, including hardware support for this partitioning of trusted and untrusted software, have emerged from several hardware vendors. Some of these extensions include Intel SGX⁹, AMD Secure Encryption Virtualization (SEV)¹⁰ and Arm® TrustZone^{®11}. Several software implementations are available, including proprietary and open-source options. Most of these implementations follow a similar software architecture for which the software running in the TEE offers a collection of services to the software running in the REE. The TEE stack itself is usually made of the following two components:

- The **TEE core component**, also known as the **TEE kernel**, offers the core isolation mechanism as well as basic services like cryptographic and storage services.
- TEE applications, running on top of the **TEE core components**, offer customization and user programmability for dedicated services to the **REE**.

On edge computing devices, several trusted service applications run in the TEE including the following:



Local HSM (for example, to serve as a local certification authority as mentioned in PKI)



Content protection and digital rights management (DRM)



Secure boot



Machine learning protection



Firmware update

TEEs are used to partition not only the processor and the memories but also the peripherals. Some TEE technologies enable a separation of the devices so that some devices are accessible only by the TEE and not by the REE.

⁹Reference: <https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html>

¹⁰Reference: <https://www.amd.com/en/processors/amd-secure-encrypted-virtualization>

¹¹Reference: <https://developer.arm.com/ip-products/security-ip/trustzone>

SOFTWARE STACK EXAMPLE

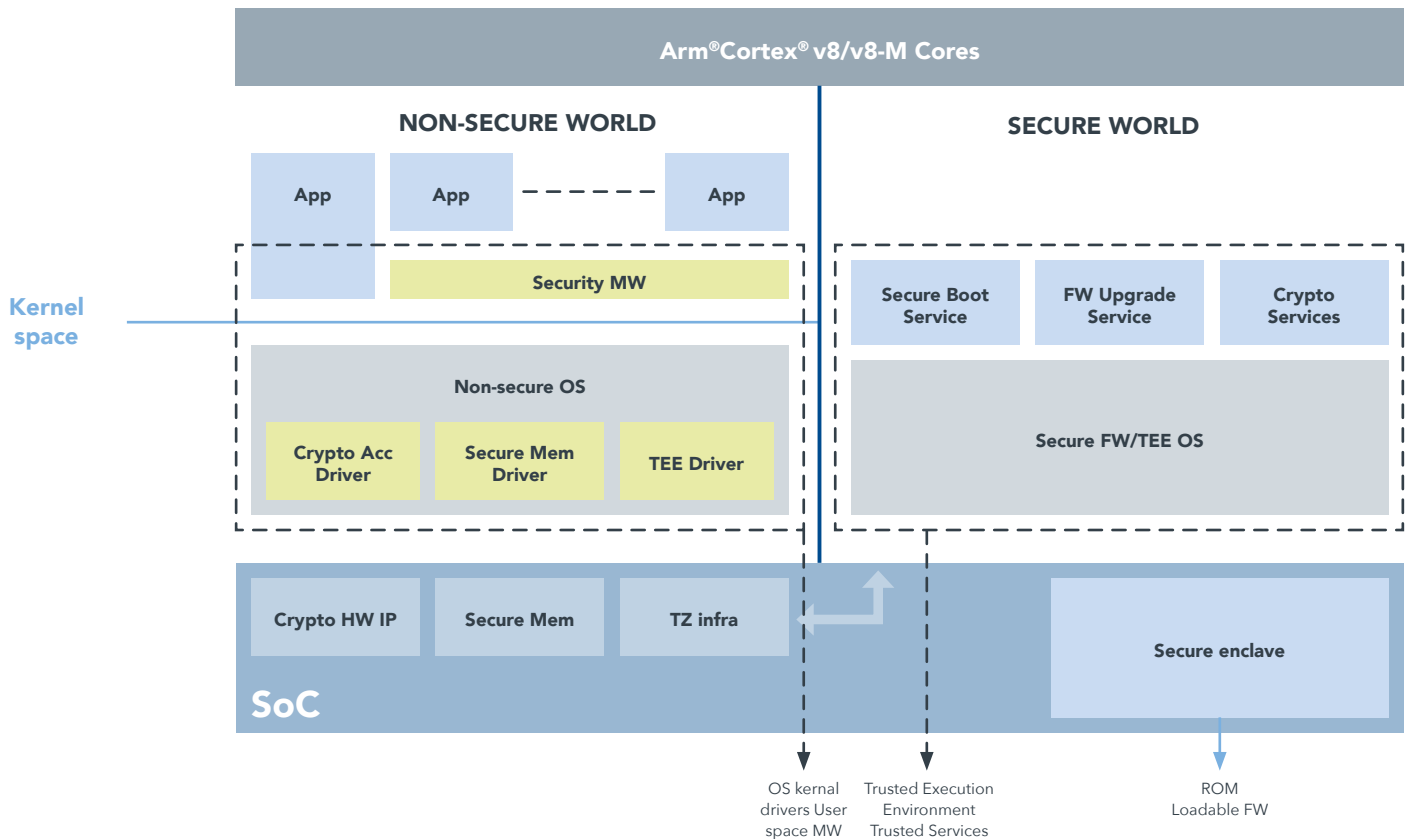


Figure 3.11. Edge device software stack with nonsecure world in Linux and secure world in OP-TEE

OpenSSL is an open-source software library that provides TLS protocols implementation and cryptographic operation functions. It is widely used by applications running Linux OS on edge devices for securing communication channels. One OpenSSL example shows how an edge device software stack can be deployed to secure key assets material while keeping, from an application point of view, the same interface.

Figure 3.11 shows an edge device software stack. It relies on Arm TrustZone isolation to provide two software execution environments. The nonsecure world execution environment is running the rich Linux OS. The secure world execution is running open, portable OP-TEE¹². It is an open-source implementation of TEE that complies with the Global Platform TEE specification. It works through API core functionalities such as cryptographic, key and storage operations. Those functions also can be accelerated in hardware. This is the case for cryptographic operations that are accelerated using the CAAM hardware on i.MX devices. Trusted applications use those APIs to offer services to the Linux OS. In the OpenSSL example, this trusted application offers cryptographic operation with keys that are protected by the TEE and saved in secure storage.

¹²Reference: <https://www.op-tee.org/>

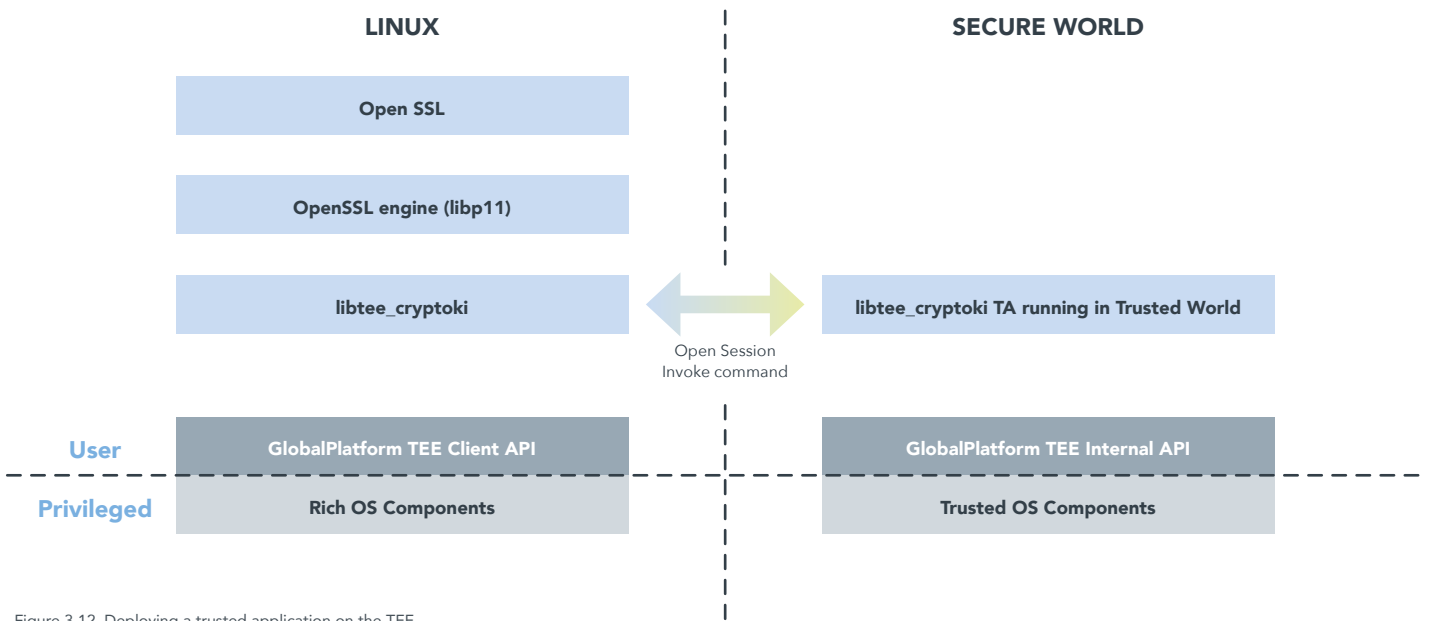
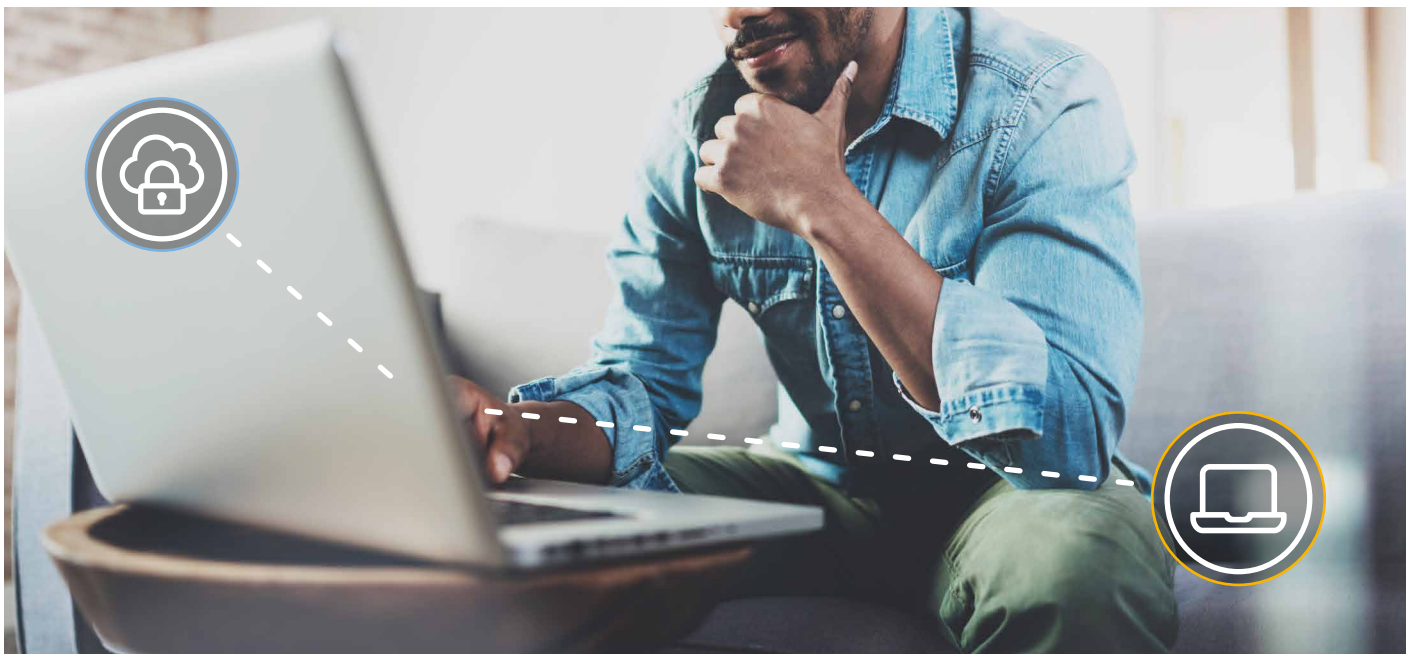


Figure 3.12. Deploying a trusted application on the TEE

A standard interface is used to prevent the OpenSSL from being modified. PKCS11 is one of the PKC standards. It is a platform-independent API for accessing tokens like HSMs or smart cards. Most of the Linux applications that use PKC have an option to use the PKCS11 interface. This is the case for most web browsers, including Firefox and Chrome, and for OpenSSL.

Figure 3.12 shows the final architecture for the OpenSSL example. A trusted application is deployed on the TEE and exposes PKC functions to a library running on Linux OS. This library exposes a PKCS11 API that is then used by OpenSSL to offer cryptographic functions to the entire application running on the rich OS.

The keys of the edge-connected device are safely stored in the TEE, while a Linux application, such as a trust provisioning or firmware update, can use them to establish a safe TLS connection with a remote server.



VIRTUALIZATION

Another isolation technique is virtualization (with or without support of the hardware). The applications performed by the edge device are clustered in virtual machines that are executed by a hypervisor that manages the sharing of the SoC hardware resources among the different virtual machines (see Figure 3.13). Malware infesting one of the virtual machines (e.g., a virtual machine installing an application from an unreliable source on the internet) cannot alter the behavior of the other virtual machines. It also cannot access the sensitive information managed by the other virtual machines as long as the attacker does not succeed in attacking the hypervisor as well. There are several ways to virtualize, for example, using a XEN hypervisor or a “docker” implementation. This approach is more secure because a hypervisor is a specific software implementation of a minimal OS. It has a smaller attack surface, and it can be audited for security when compared with Linux or Android complex systems. Table 3.2 illustrates this difference by listing the number of Common Vulnerabilities and Exposures (CVE) referring to XEN, Linux and Android. Note: Just because the OS is mentioned in a Table 3.2 entry doesn't mean it is per se an issue, but inclusion in this table shows the relative numbers for a hypervisor compared with those of large, complex OSs. In addition, modern cores used in SoCs provide some hardware support for this virtualization. The most common one is a memory manager that ensures the secure partitioning of the memory.

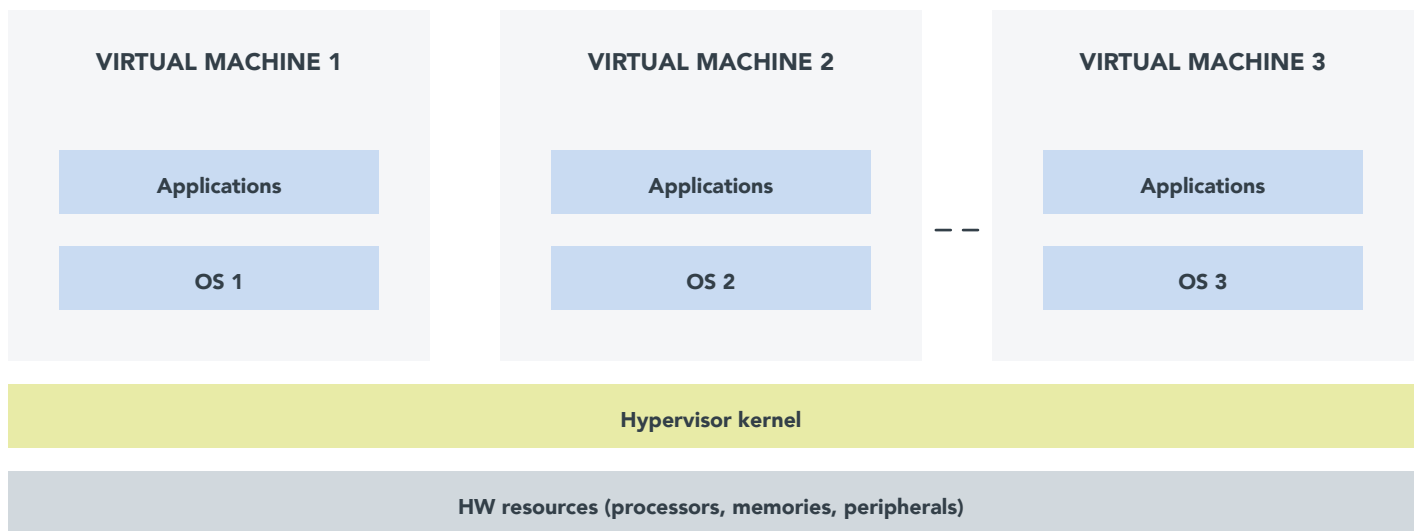


Figure 3.13. Virtualization

Year	Total number of entries in CVE database	Total number of entries mentioning XEN	Total number of entries mentioning Linux	Total number of entries mentioning Android
2018	21837	95	650	725
2019	21431	90	710	1445
2020	30890	100	405	1241

Table 3.2. Number of CVE entries per OS¹³

¹³Reference: <https://cve.mitre.org/>



SECURITY CERTIFICATIONS

Security is about trust. This implies that when edge devices are incorporated in a system to adhere to a given level of security, those edge devices must be trusted to operate at that security level. When an edge device is manufactured and further sold, manufacturers claim some features offer some level of security. To substantiate these claims, one approach is to have a trusted third party that conducts security evaluations according to agreed upon certification schemes examine the edge device. These third-party evaluators deliver certificates that are recognized and trusted by all the stakeholders exploring the security of the system containing those edge devices. Some certification schemes are proprietary like the Arm Platform Security Architecture (PSA) scheme; others are open like the SESIP scheme from GlobalPlatform. Some schemes are specific to special use cases like the Common Criteria certification scheme used in high-end security chips for payment and identity. Other schemes are made mandatory by regulations like the FIPS 140-x certification scheme required for devices sold to U.S. governmental entities.



PRIVACY BY DESIGN

This chapter has addressed the broad topic of system security for edge devices, but there is more. Edge devices will be incorporated in systems that must comply with privacy-preserving regulations (e.g., the General Data Protection Regulation, or GDPR, in Europe). Moving the collection and processing of data (some considered personally identifiable information) to edge devices is an approach that lessens the burden on cloud services, but it will make edge devices subject to privacy impact assessments. Privacy, like security, is a holistic system property. When privacy is a requirement, a “privacy by design” approach must be adopted the same way a “security by design” approach is used to achieve system security.



SECURITY BY DESIGN

Implementing security correctly is a difficult and complex task. Incorporating security features in edge devices is not sufficient; they have to be used judiciously to achieve the expected result. System security architects must envision the use cases of the system and conduct a risk and threat analysis. From this analysis, they identify the required security features and how to combine them. This approach is called “security by design”. Note that in this approach, security must be considered from Day 0 of an edge device’s conception. The sooner security is considered, the higher the probability of achieving the expected security at the lowest budget. Security is never an afterthought.

Chapter 4

EDGE COMPUTING INTELLIGENCE

CONTRIBUTORS

Natraj Ekambaram, Director, AI and ML Enablement, NXP Semiconductors

Ali Osman Örs, Director, AI and ML Strategy and Technologies, NXP Semiconductors

Laurent Pilati, Director, ML and Voice Engineering, NXP Semiconductors

This chapter presents the benefits of machine learning inference at the edge, such as uninterrupted processing, lower latency and user privacy. It examines workflows, frameworks and tools, hardware, software, application examples and other edge processing machine learning topics.

MACHINE LEARNING AT THE EDGE

Machine learning (ML) is a subset of artificial intelligence (AI) that enables computer algorithms to improve automatically through experience. ML can be classified into supervised ML and unsupervised ML categories. In supervised ML, algorithms are “trained” using large sets of previously collected and labeled data from one or more sensors. In unsupervised ML, the algorithm learns over time to identify outliers and differentiation in the sensor data it is exposed to during operation.

ML is predominantly conducted in the cloud with servers and large compute and storage capacities. However, as ML models and algorithms matured, ML inferencing moved from cloud to edge devices. Billions of internet of things (IoT) devices perform control and data gathering operations. Compute power continues to increase as more complex control and operational decisions are moved to edge devices. These secure and self-reliant, albeit memory- and power-constrained, edge devices can perform real-time ML inferencing tasks locally with occasional cloud connection.

For example, ML in the cloud is the key technology applied when anyone uses a voice assistant with a smartphone or smart speaker. It also is the technology behind how social media and even smartphones group together photos featuring a specific person. But those use cases all rely on ML running on a server somewhere in the cloud.

Running ML inference at the edge has advantages. All the ML inference runs locally on an edge processor, which means that the application continues to run even if access to the network is disrupted. This is critical for applications such as surveillance or a smart home alarm hub, or when operating in remote areas without network access. It also provides much lower latency during decision-making than if the data had to be sent to a server and processed, and the server had to send the results back. Low latency is important, for example, when performing industrial factory floor visual inspection and deciding whether to accept or reject products whizzing by.

Another key benefit of ML on the edge is user privacy. The personal data collected, such as voice communication and commands, face recognition, video and images captured by the edge device, are processed and stay local on the edge. Information is not sent for processing to the cloud, where it can be recorded and tracked. The user’s privacy remains intact, so individuals can choose whether to share their personal information in the cloud.

Given the need for ML on the edge, the question becomes how much ML performance is needed. One way to measure ML performance requirements is the number of operations per second. These are usually referred to as TOPS or tera (trillion) operations per second. This is a rudimentary benchmark because overall system performance depends on many other factors. Nonetheless, it is one of the most widely quoted ML measurements.

For example, performing full speech recognition (not just keyword spotting) on the edge takes around 1 to 2 TOPS, depending on the algorithm used and whether one wants to understand what the user is saying rather than just converting from speech to text. Performing object detection (using an algorithm such as Yolov3) at 60 frames per second (FPS) also takes around 2 to 3 TOPS.

ML DEVELOPMENT WORKFLOW

Figure 4.1 shows a high-level ML development workflow. Developing ML technology to deploy to an edge node requires both operation and dataflow. These steps include:

- **Collecting raw data** — Identify and collect data that will be used to train an ML model.
- **Augmenting data** — Artificially expand labeled training datasets to improve the performance of an ML model.
- **Extracting features** — Reduce the number of features in the dataset by creating new features that summarize the original features with less information.
- **Creating training and validation sets** — Separate the raw, augmented data into two datasets: one for training the ML model and one for validating the model. To test for bias and over-fitting bias, these should be separate datasets.
- **Selecting a model** — Choose a model that meets application and performance requirements (image classification, object detection, speech recognition, anomaly detection, etc.).
- **Training the model** — Use an ML algorithm and the raw data to create a model for performing predictions on new data.
- **Validating the model** — Run a separate set of data through the trained ML model to test for accuracy and correctness.
- **Converting and quantizing the model** — Approximate a floating-point-based ML network with a low-fixed-point model that reduces memory bandwidth and computational cost. In neural networks, quantization is converting a data floating-point number to a fixed-point number. Edge devices with ML accelerators are largely capable of computing at 8-bit fixed-point precision. By converting a 32-bit floating-point value to an 8-bit fixed-point integer value, model size is instantly reduced by four times. Quantization also enables faster weight transfers due to reduced precision from the main memory to local compute engines. ML accelerators typically have large local memory that can store weights, thus benefiting from reduced data transfers between the main memory and local memory.
- **Inferencing** — Run new data (from a sensor or other data collection mechanism) through the ML algorithm (or model) to determine an output (for example, the classification of an object).

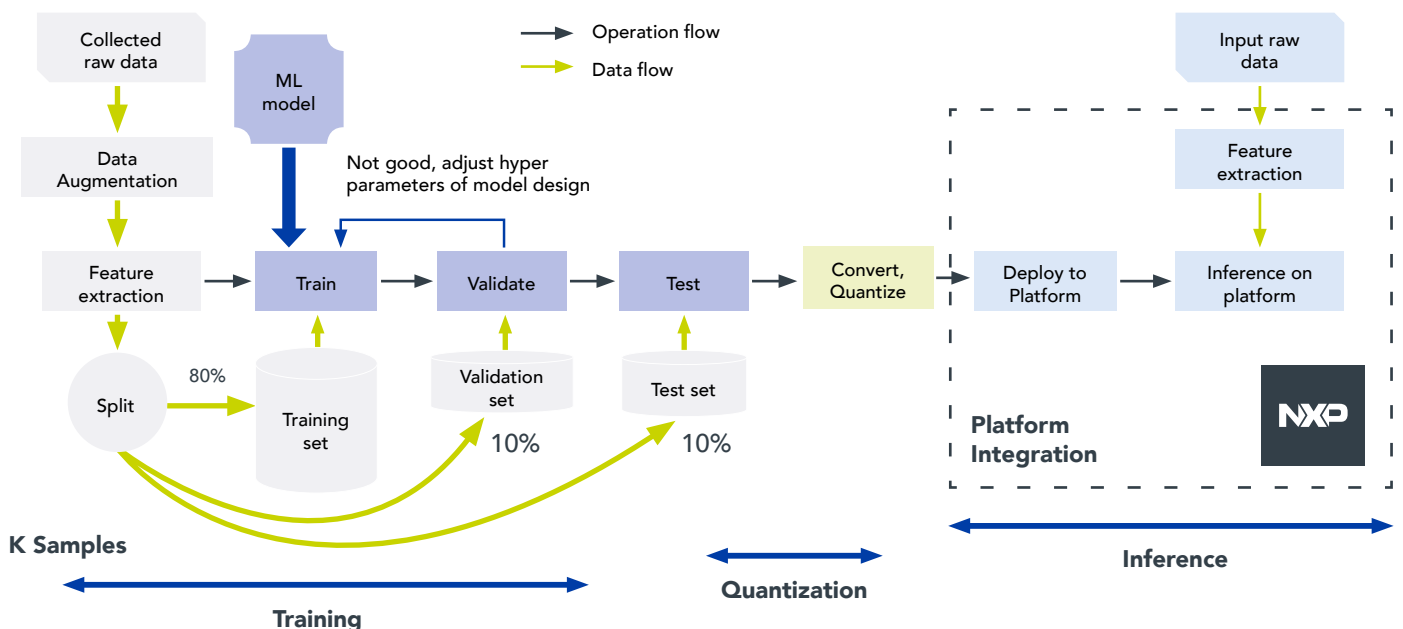


Figure 4.1. A high-level ML development workflow

EDGE ML TOOLS

Edge-based ML requires tools to create and deploy ML models starting in the cloud and ending with inferring performed by edge device ML software stacks with optimized run times on the key edge device hardware such as graphics processing units (GPUs), central processing units (CPUs), digital signal processors (DSPs) and ML or neural processing unit (NPU) accelerators. These components are shown in Figure 4.2 along with the ML workflow example. Users with different roles such as embedded developers, data scientists and ML algorithm experts use ML toolkits. Because many cloud vendors provide tools for model training, edge-based tools should support the deployment of ML technology from the cloud to an edge device.

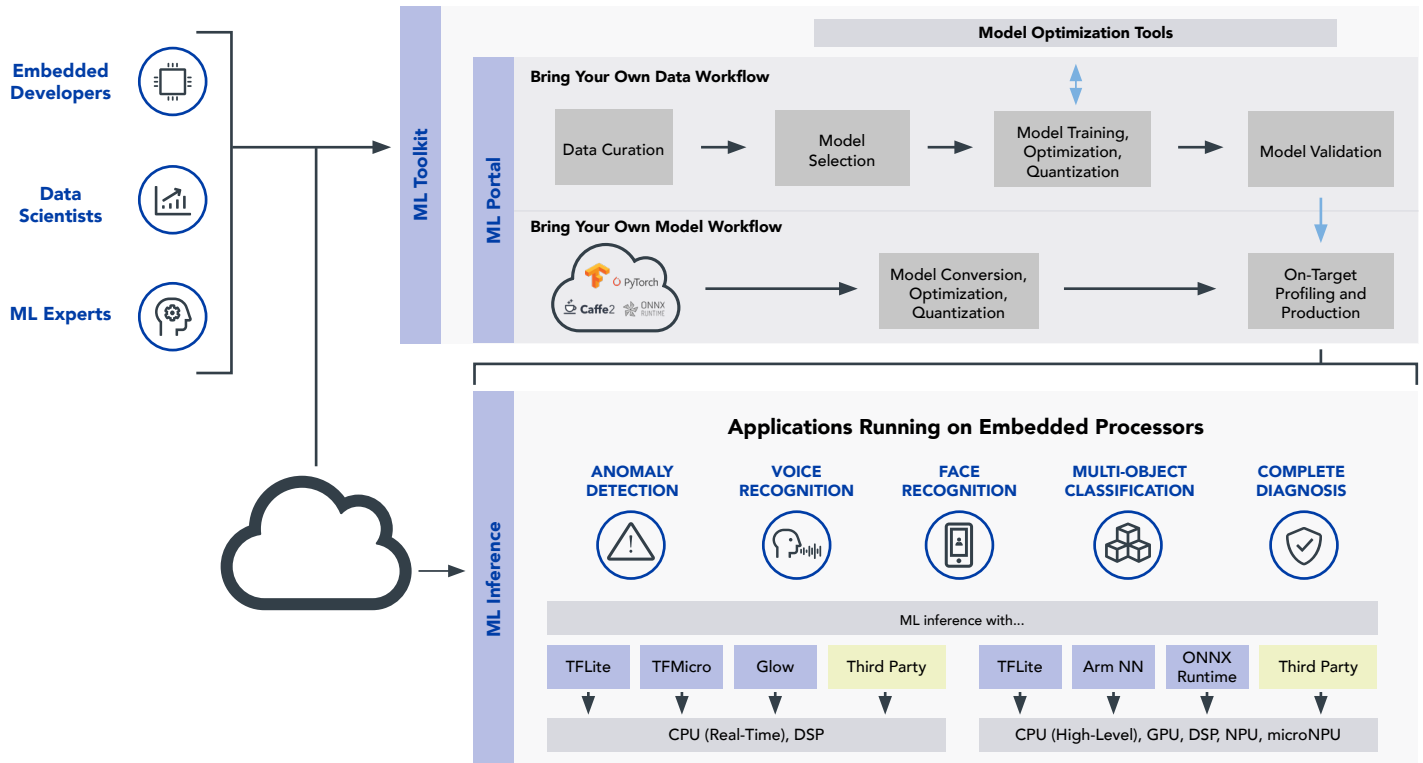


Figure 4.2. ML workflow and Edge ML tools



EDGE ML FRAMEWORKS

Edge ML tools use several open ML frameworks. An ML framework combines libraries and tools that enable embedded developers to build, optimize and deploy ML models easier and faster. These frameworks democratize the development of ML models and abstract some but not all the underlying algorithmic details. Some of these frameworks provide pretrained models for speech recognition, object detection, natural language processing (NLP) and image recognition and classification, among others. Table 4.1 describes some of the popular edge-based ML frameworks.

ML Framework	Edge device applicability	Key features
TensorFlow	Numerical computation using dataflow graphs, regression, classification, neural networks, CPUs and GPUs	Google, open-source, open-source deep learning framework, training and inferencing
TensorFlow Lite	Designed for mobile platforms and embedded devices; specifically designed for inference on devices with limited compute (phones, IoT and other embedded devices) and low latency	Faster, smaller in size and less computationally expensive; cannot be used for training like TensorFlow; only can be used for device inferencing
PyTorch	Regression, classification, neural networks, CPUs and GPUs	Facebook, object-oriented programming and many options for optimizing algorithms
TensorFlow Lite Micro	Inference library based on Arm® Mbed™ technology: targeted to on-device inferencing	Targets memory resource constrained Microcontrollers such as Arm Cortex®-M cores

Table 4.1. Common edge processing ML frameworks



EDGE ML HARDWARE

Artificial neural networks (ANN), commonly called neural networks (NN), are computing systems inspired by biological neural networks. Loosely modeled on the neurons in a biological brain, an ANN is based on a collection of connected units or nodes called artificial neurons. NN-based ML algorithms such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been shown to be very effective in ML inference tasks. These algorithms consist of multiple compute and transform layers that analyze data to detect patterns. CNNs are feed-forward neural networks. In feed-forward networks, all the computation is performed as a sequence of operations on the outputs of a previous layer. The final set of operations generates the output of the network such as the probability that an image contains a particular object, that an audio sequence contains a particular word, that a bounding box in an image surrounds an object or that the proposed action is taken. In CNNs, the network has no memory and the output for a given input is always the same irrespective of the sequence of inputs previously given to the network. RNNs use internal memory to allow long-term dependencies to affect the output. RNNs use time-series information for giving outputs and predicting future actions and results based on current and past data for language processing tasks, sensor analytics, anomaly detection and so on. The major computation in CNNs and RNNs is the “weighted sum” operation that typically uses multiply-accumulate (MAC) operations. Because a MAC operation involves a multiplication followed by an addition, each MAC comprises two operations. Both CNN and RNN computations benefit from fast memory on the embedded hardware device. RNN performance and support are impacted more by limited memory because the feedback path requires it.

Typically, neural networks are trained with data in a 32-bit floating-point (FP32) representation. But because memory and compute profiles are limited in embedded devices, inference on the edge often generates a large incentive to quantize from FP32 to a fixed-point integer representation that’s 16-bit, 8-bit and even lower. Lower-bit mathematical operations with quantized parameters combined with quantized intermediate calculations of a neural network results in large computational gains and higher performance. Quantization decreases accuracy, so additional methods are needed to recover the accuracy loss to an acceptable level. The energy and area of a fixed-point multiply scale approximately quadratically with the number of bits. Reducing the precision also reduces the energy and area cost for storage, which is important because memory access and data movement dominate energy consumption and memory is limited in embedded systems.

Based on estimates done in 45nm technology¹

- An **8-bit integer ADD** operation consumes **30X less energy** than a **32-bit floating point ADD**.
- An **8-bit integer MUL** operation consumes **18.5X less energy** than a **32-bit floating point MUL**.

Edge processing ML can be performed on SoC processing elements such as CPUs, GPUs, DSPs and dedicated accelerators, or a combination of these processing elements. Each has advantages and disadvantages.

CPUs work well for embedded applications that require parsing or interpreting complex logic in code. They are not optimized for ML computation, but they can be used if necessary. CPUs dedicate more area to caches and control flow to handle complex logic and more sequential processing.

DSPs have been used in embedded systems for many years to efficiently and economically handle various forms of complex signal processing. Using DSPs to analyze sensor data for feature extraction is common. DSPs, which continue to evolve, use special instructions such as MACs to accelerate common signal processing structures. Vector processing units built around MAC units are being used to accelerate neural network computations. Wider single instruction, multiple data (SIMD) units are also being used with very long instruction word (VLIW) DSP architectures.

¹M. Horowitz, “1.1 Computing’s energy problem (and what we can do about it),” 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 2014, pp. 10-14, doi: 10.1109/ISSCC.2014.6757323.

Figure 4.3 shows a DSP used for ML processing on a low-cost edge device. In this example, an Arm[®]-based Cortex Microcontroller Software Interface Standard (CMSIS) DSP standardizes the DSP code running on Cortex-M cores. PowerQuad, a coprocessor designed by NXP to improve energy efficiency and performance when implementing DSP algorithms using its MCUs based on Cortex-M33 cores, can leverage this application programming interface (API). Preprocessing of mathematical functions, like FFT, square root; activation functions like sigmoid and softmax; as well as matrix operations are supported.

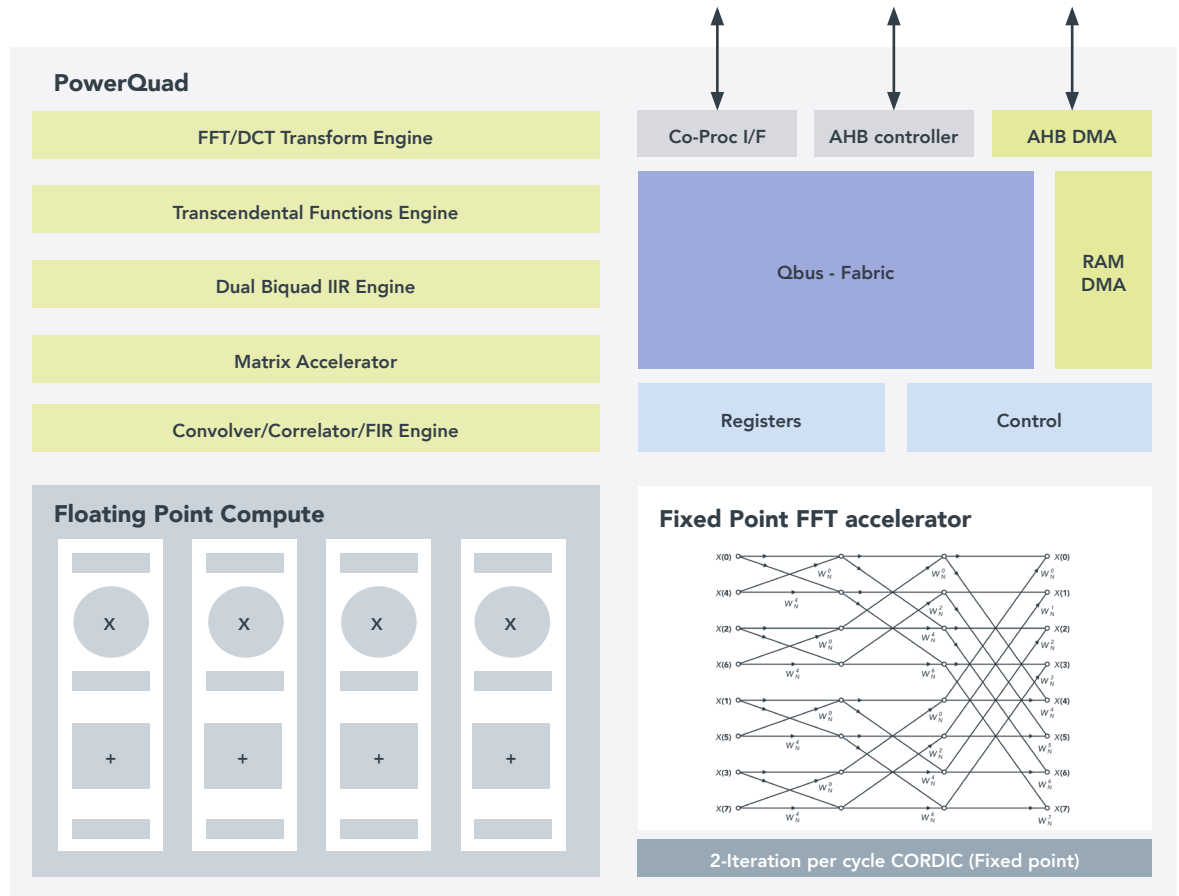


Figure 4.3. Using a PowerQuad coprocessor on a low-cost edge device for ML

GPUs originated from dedicated graphical rendering engines for computer games. They have evolved to accelerate additional geometric calculations such as transforming polygons or rotating images into different coordinate systems. GPUs have more logical cores such as arithmetic logic units (ALUs), which allow them to process multiple computations simultaneously. ML also requires large amounts of data, which works well with GPUs architected for high memory bandwidth. GPUs are primarily designed for pixel processing; however, highly parallel matrix mathematics can be achieved using the shader cores for ML computations.

NPUs are optimized for common edge-based use cases such as object detection and segmentation at much higher levels of performance and much lower power than CPUs. The accelerators in Figure 4.4 process complex workloads under a rich OS in Cortex-A systems with wide bus interfaces (128-bit) and dynamic random access memory (DRAM) support. Other optimizations such as integrated direct memory access (DMA) allow for neural network weights and activations to be fetched ahead of time using a DMA connected to system memory. The heavy compute operations, such as convolution, pooling, activation functions and primitive element wise functions, run directly on the NPU. Other kernels run automatically on a tightly coupled CPU (such as Cortex-M). Another approach to increase performance and reduce memory requirements is to conduct offline compilation and optimization of neural networks, including operator and layer fusion as well as layer reordering.

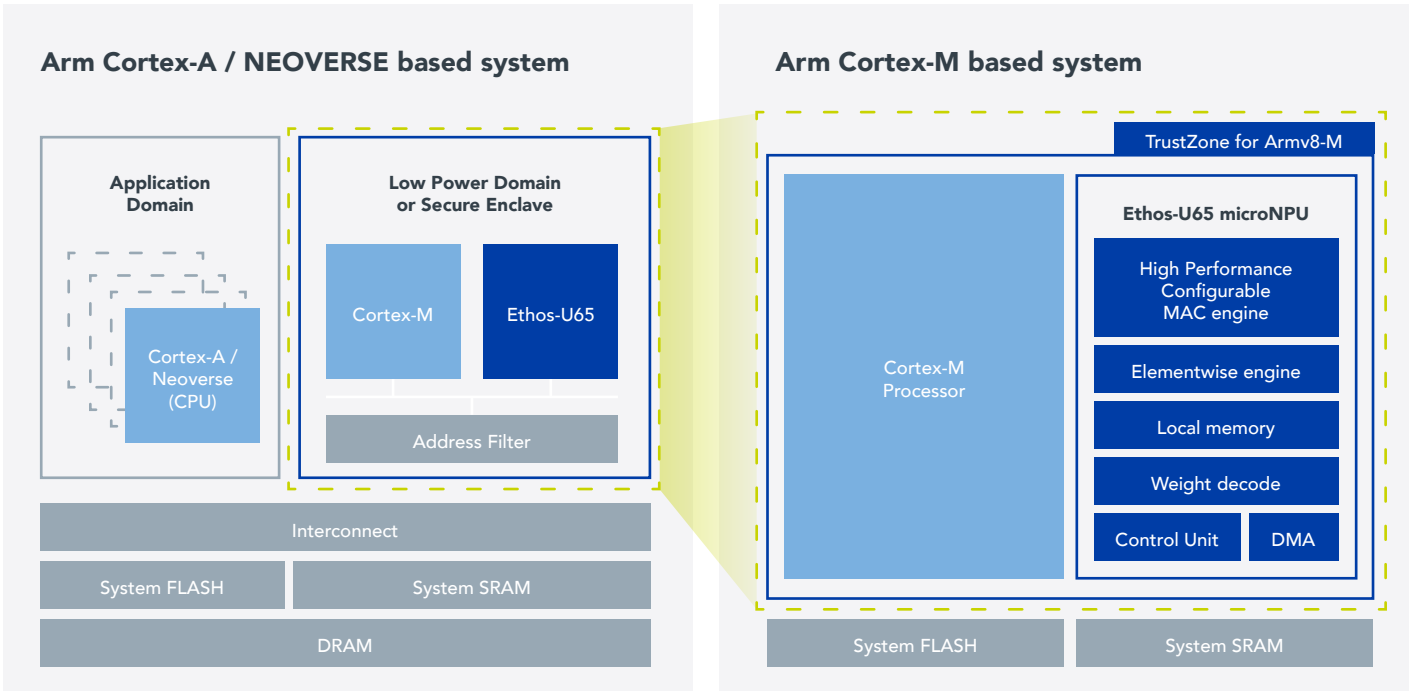
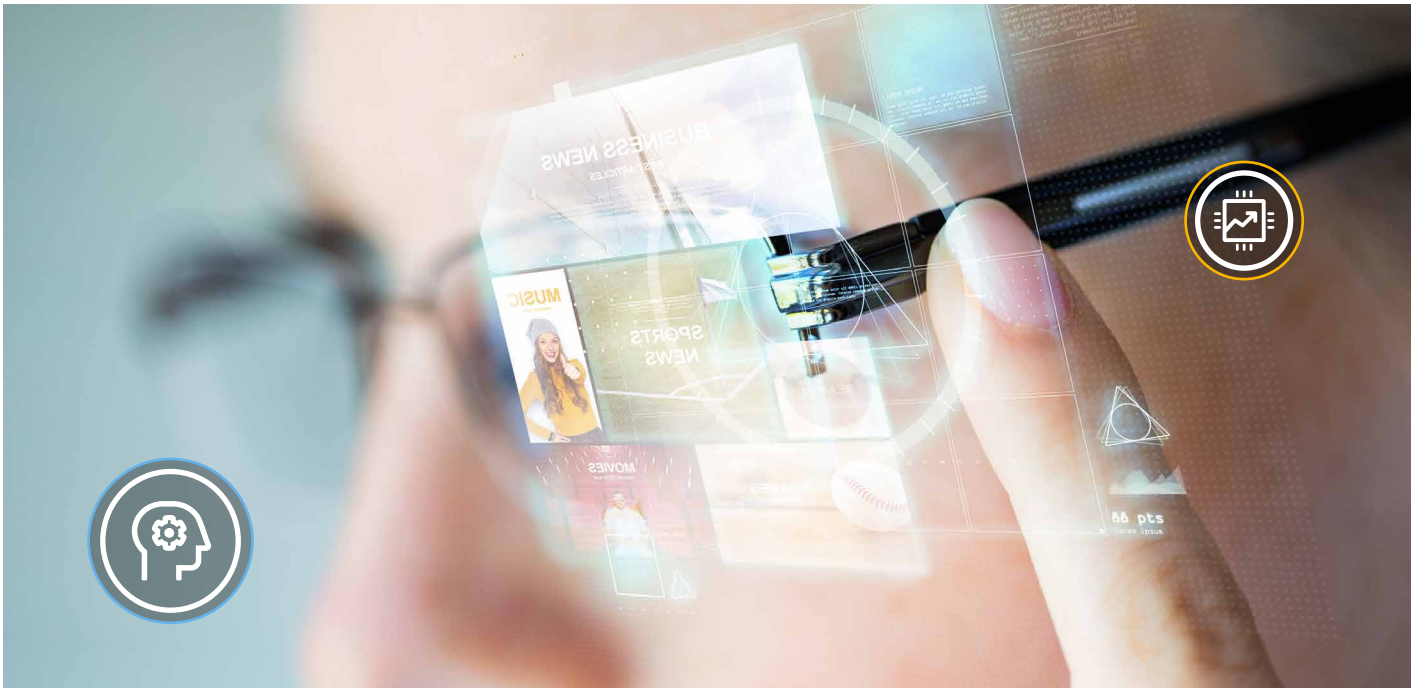


Figure 4.4. An NPU for edge ML processing

Edge processing SoCs contain multiple processing elements including one or more of the types mentioned previously. These processing elements can be used independently or together to perform ML at the edge. Various optimized ML pipelines can be designed to efficiently leverage the available processing power of the SoC. Edge ML computing is a system-level optimization exercise for which multiple processing elements on a SoC (see Figure 4.5) can be used and enabled properly to support advanced real-time edge ML processing.

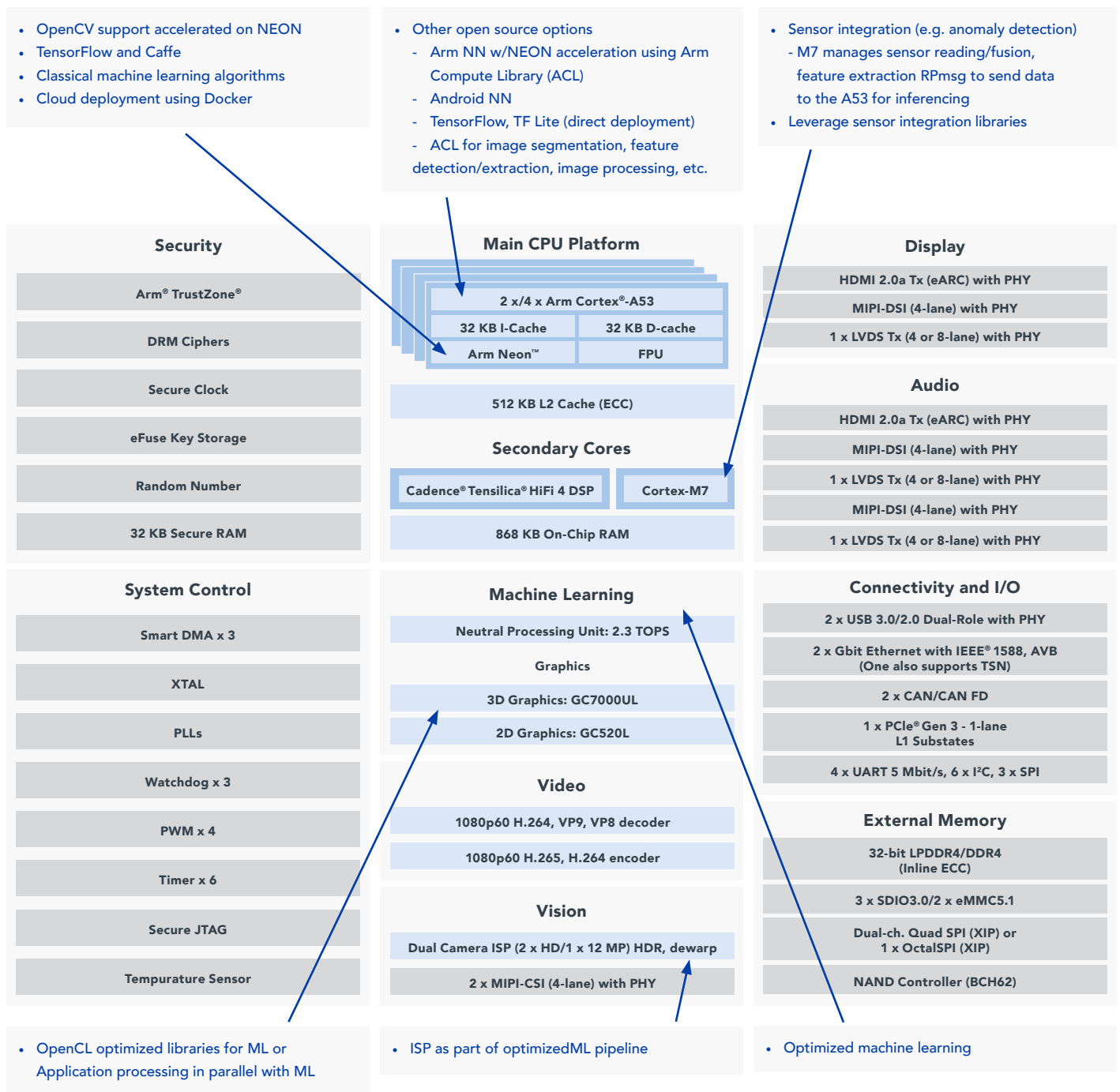


Figure 4.5. Edge processing SoC with NPU for ML acceleration

Consider the ISP shown in Figure 4.5. Camera-based systems always include image signal processor (ISP) functionality, though sometimes it can be either integrated into a camera module or embedded in an applications processor and potentially hidden to the user. ISPs typically conduct many types of image enhancement along with their key purpose: converting the one color component per pixel output of a raw image sensor into the RGB or YUV images that are more commonly used elsewhere in the system.

Applications processors without ISPs work well in vision-based systems when the camera inputs are coming from network or web cameras that are typically connected to the applications processor by Ethernet or USB. For these applications, the camera can be up to 100m away from the processor. The camera itself has a built-in ISP and processor to convert the image sensor information and encode the video stream before sending it over the network.

For relatively low-resolution cameras, applications processors without ISPs also work well. At resolutions of 1 megapixel or below, image sensors often feature an embedded ISP and can output RGB or YUV images to an applications processor, so an ISP is not needed in the processor.

But at a resolution of around 2 megapixels (1080p) or higher, most image sensors do not have an embedded ISP; instead, they rely on an ISP somewhere else in the system. This may be a stand-alone ISP chip (which works but adds power and cost to the system) or an ISP integrated in the applications processor as shown in Figure 4.5.

With the combination of an ML accelerator and an ISP, the edge SoC processor can perform embedded vision system applications at the edge, whether they be for smart homes, smart buildings, smart cities or industrial IoT applications. With its embedded ISP, the edge SoC processor can be used to create high image quality optimized systems connected directly to local image sensors. It even can be used to feed this image data to the latest ML algorithms, all offloaded in the local ML accelerator.



A more generic ML development approach for edge processing includes these steps:

1. Define the use case and the corresponding type of machine learning and model.
2. Use a ML framework that is self-contained and does not rely on the underlying hardware.
3. Prototype the chosen ML paradigms with the framework on a PC, cloud or higher end embedded device.
4. Characterize the network model in terms of memory and computational overhead.
5. Choose a hardware platform while considering the memory and computational constraints. Then cross-compile the network for the specific embedded device.
6. Train the model on a higher end machine and transfer the weights over to the embedded device (the weights do not change, so they can be stored as a constant array in memory).
7. Perform relevant network optimizations (pruning, quantization, precision reduction)
8. Perform relevant hardware optimizations (alignment, SIMD instructions).
9. Test the performance of the deployed network model and determine whether this implementation can be iterated over after deployment.



Edge processing also can be implemented on low-end microcontrollers. A possible development flow for ML on low-end MCU includes these steps:

1. Upload the labeled data to a PC. You can use a universal asynchronous receiver-transmitter (UART), Secure Digital or an SD card.
2. Experiment with the data and an ML toolkit using tools such as scikit-learn. Make sure an off-the-shelf method can produce good results before moving forward.
3. Experiment with feature engineering and selection. Try to achieve the smallest feature set possible to save resources.
4. Write an ML method to use on the embedded system (perceptrons or decision trees are good because they don't need a lot of memory). If no floating-point unit is available, integers and a fixed-point unit can be used.
5. Implement the normal training procedure. Use cross-validation to find the best tuning parameters, integer bit widths, radix positions, etc.
6. Run the final trained predictor on your holdout testing set.
7. If the trained predictor performance is satisfactory on the testing set, move the code that calculates the predictions and the model trained (for example, weights) to the MCU. The model weights will not change, so they can be stored in nonvolatile flash memory such as a constant array.

OPTIMIZING ML PIPELINES FOR EDGE DEVICES

Embedded edge devices are growing more complex and powerful as they incorporate more hardware components such as CPUs, GPUs, DSPs and ML accelerators to perform various forms of ML. However, these complex hardware components must be used efficiently. Edge devices with dedicated accelerators such as GPUs and NPUs can perform matrix multiplication significantly faster than CPUs. ML frameworks can efficiently leverage these hardware components. For example, TensorFlow Lite interpreters use the concept of “delegates” that can hand over the compute intense operations to the dedicated hardware for acceleration. Software architectures to support ML can optimize the execution flow of ML in the SoC to provide high-performance, low-power solutions.

The application-specific processing pipeline shown in Figure 4.6 is designed in multiple stages with multiple steps in the pipeline that can be leveraged for ML processing. Key application segments include:



Vision pipelined for object/face detection/recognition



Voice and audio pipeline for speech analysis



Series data processing pipeline for anomaly detection

Processing pipelines and flexible software architectures provide out-of-the-box SoC and application-type optimized run-time support. This facilitates complete exploitation of heterogeneous SoC capabilities for ML and maximizes component reuse. Key benefits of this approach include improved out-of-box experience (OOBE) and ease of use; comprehensive SoC and hardware resource usage, with configurability over I/O interfaces; acceleration option configuration for different use cases; processing domains for easier customization; scalability across SoCs; and the use of open-source and other community components.

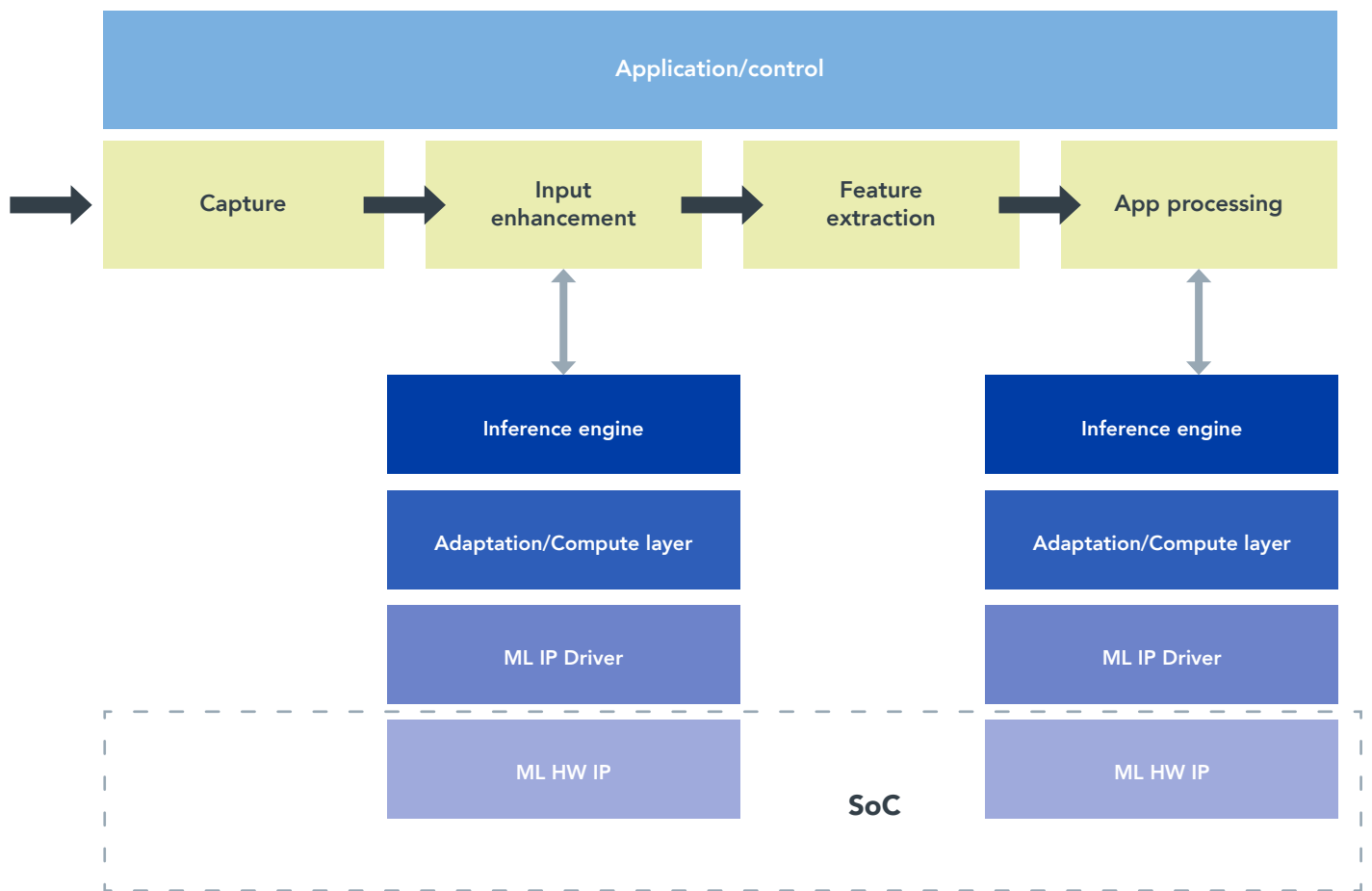


Figure 4.6. Edge device optimized ML pipeline

As an example of an ML-optimized pipeline in Figure 4.6, consider the growing demand for video intelligence (industrial inspection, face/person/object detection and classification, action recognition). This intelligence has pushed the vision paradigm to quickly incorporate ML-based techniques. The traditional vision techniques based on handcrafted feature extraction and usage are still greatly used, but the emergence of powerful hardware to run inference engines combined with the widely available ML frameworks and vision-based models lowered the barriers to fully (or almost fully) using ML to address machine vision use cases.

A capable edge SoC for ML processing in this application must first be chosen. The device in Figure 4.5 embeds an NPU, 2D and 3D GPUs, a dual-image signal processor and two camera inputs for an effective advanced vision system. This SoC has all the hardware elements required to address complex ML-based vision use cases.

Software must enable these hardware components. Figure 4.7 shows an example of an edge device software architecture to support optimized ML at the edge. This software includes:

- Video streams and image processing from the Linux® kernel drivers to the de facto standard media stream framework GStreamer. These software components enable local and remote camera capture, local and remote video stream and picture presentation, and hardware-accelerated single picture processing (scaling, rotation, color space conversion).
- Adaptation and optimization of the major natural language frameworks (TensorFlow Lite, ONNX, ArmNN, Glow) to run efficiently on the SoC NPU, GPU and (coming soon) DSP.
- GStreamer plug-ins to provide a vendor-agnostic neural network integration framework that eases the integration and connection of the different hardware and software components involved in a machine vision use case. This framework, NNStreamer, an open-source technology, supports the major ML frameworks (TensorFlow Lite, ArmNN, Caffe2) and features the following:
 1. Neural network framework connectivity (TensorFlow, Caffe, etc.)
 - Stream frameworks like GStreamer.
 2. AI project streaming — Apply efficient and flexible stream pipelines to neural networks.
 3. Intelligent media filters — Use a neural network model as a media filter/converter.
 4. Composite models — Apply multiple neural network models in a single stream pipeline instance.
 5. Multimodal intelligence — Use multiple sources and stream paths for neural network models.
- Methods to construct media streams with neural network models using the de facto media stream framework, GStreamer. GStreamer users can apply neural network models as if they were just another media filter. Neural network developers can manage media streams easily and efficiently.
- Real-time performance profiling of the full pipeline (CPUs, GPUs, NPUs, DSPs and memory profiling).

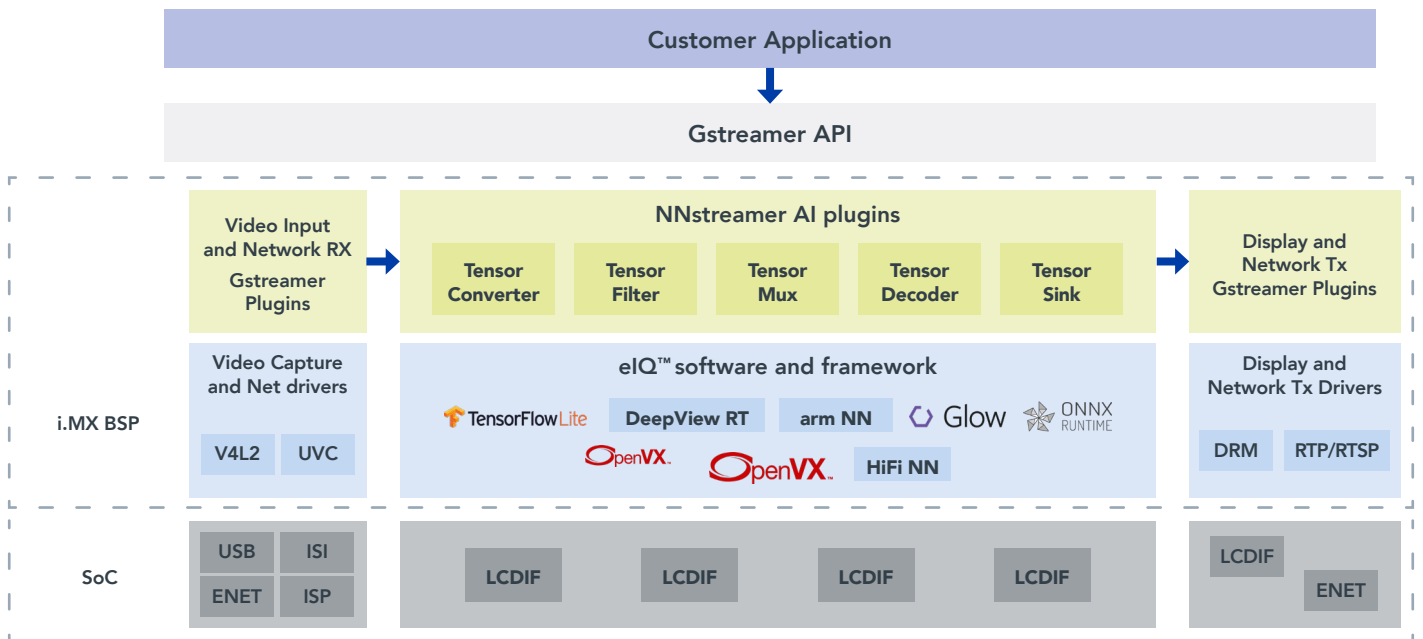


Figure 4.7. NXP eIQ ML Development Environment supports optimized machine learning at the edge

This concept can be expanded further by conducting the parallel processing of ML algorithms on a single SoC. Figure 4.8 shows a voice and audio ML pipeline configured to run on an Arm Cortex-M core on top of a real-time OS while a vision ML pipeline executes on the Arm Cortex-A core on top of a rich OS such as Linux.

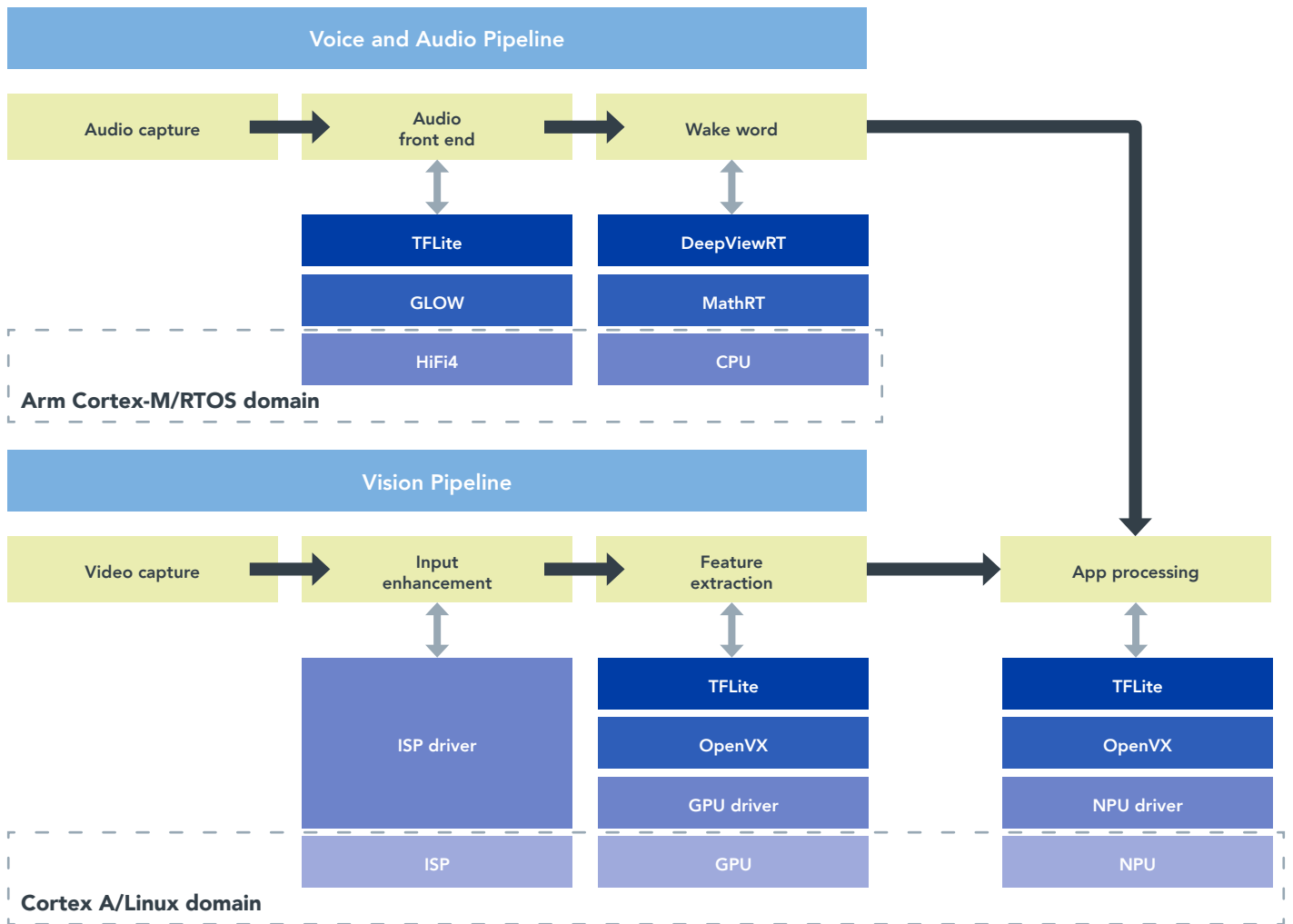


Figure 4.8. Application run time simultaneously leveraging voice and vision pipelines on an NXP i.MX 8M Plus edge computing SoC

In summary, running ML at the edge requires an awareness of the compute and memory resources available. It also requires modifications to the ML models and the process flow to fit the resource profile. In return, running ML at the edge has many advantages such as improved privacy, reduced or no dependency on a network connection, reduced power dissipation and the capability to make real-time low-latency decisions.

Chapter 5

EDGE COMPUTING CONNECTIVITY

CONTRIBUTORS

Cristi Caciuloiu, Connectivity Senior Software Engineer, NXP Semiconductors

Mihai-Andrei Dragnea, Senior Embedded Software Engineer, NXP Semiconductors

Sebastian Grigore, Manager, Connectivity Software R&D, NXP Semiconductors

Doru Gucea, Connectivity Senior Software Engineer, NXP Semiconductors

Prabhu Loganathan, Senior Director of Marketing, Wireless Connectivity Solutions, NXP Semiconductors

Jason Martin, Senior Director of MCU Software Enablement, NXP Semiconductors

Sujata Neidig, Director, Wireless Connectivity MCUs, NXP Semiconductors

Robert Oshana, Vice President, Edge Processing Software R&D, NXP Semiconductors

Connectivity is a fundamental part of the edge processing system, and one of the basic tenets of smart devices and edge processing is “connect.” This chapter provides an overview of several of the wireless connectivity standards.

WIRELESS CONNECTIVITY OVERVIEW

Wireless connectivity communication protocols (see Figure 5.1) can be proprietary or proposed by global standardization organizations to achieve greater interoperability.

The selection of a wireless connectivity protocol for a given application depends on several factors including data rate, range and power consumption. Security requirements and support of an internet of things (IoT) ecosystem like Matter should also be considered. Multimode IoT devices can implement several of these protocols simultaneously.

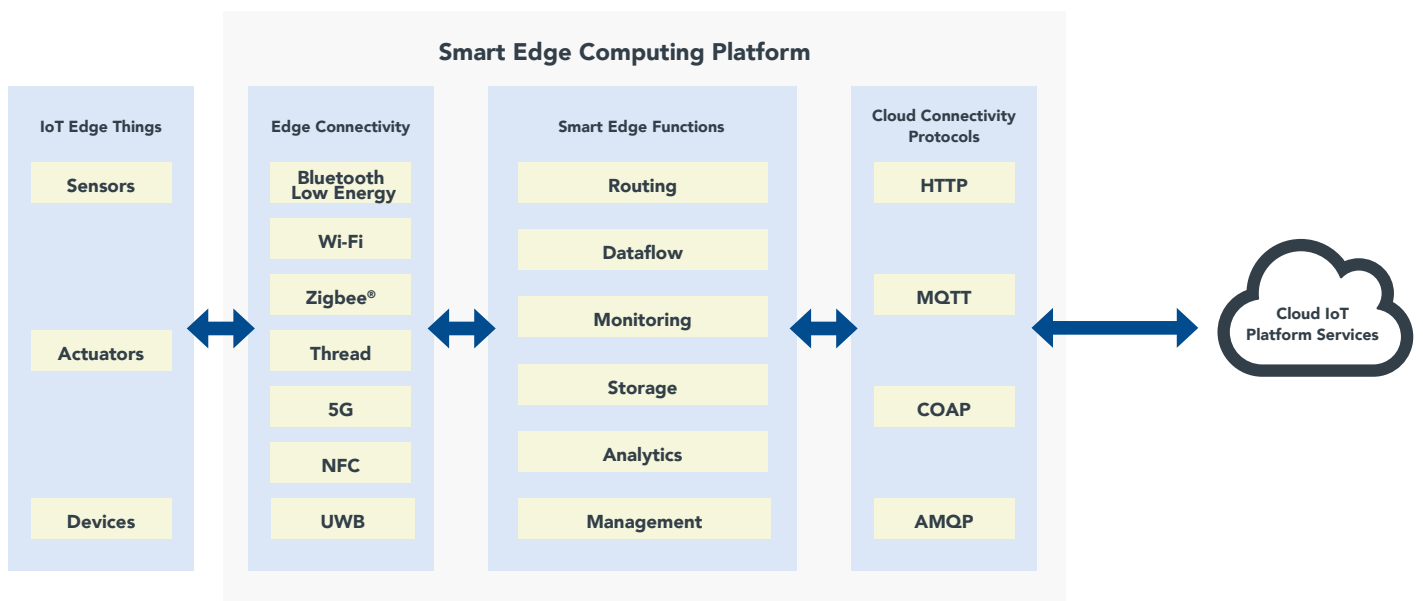


Figure 5.1. Connectivity technologies

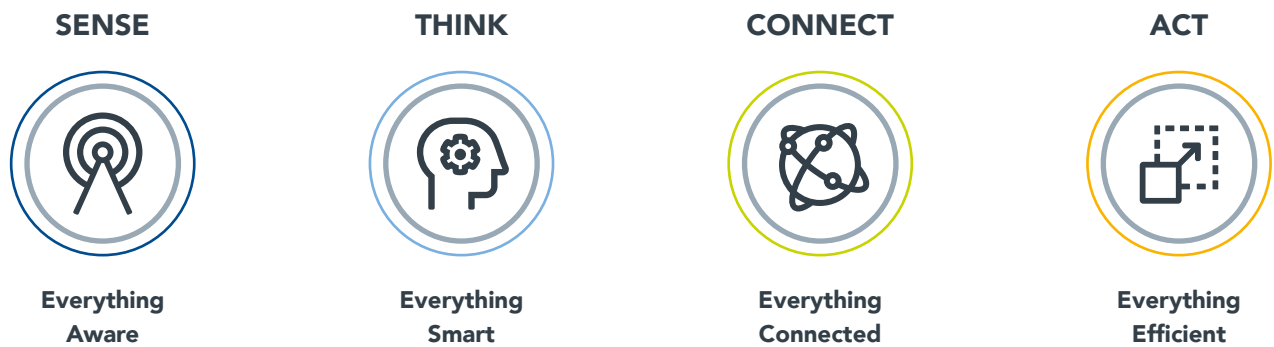


Figure 5.2. Key tenets of sense, think, connect and act

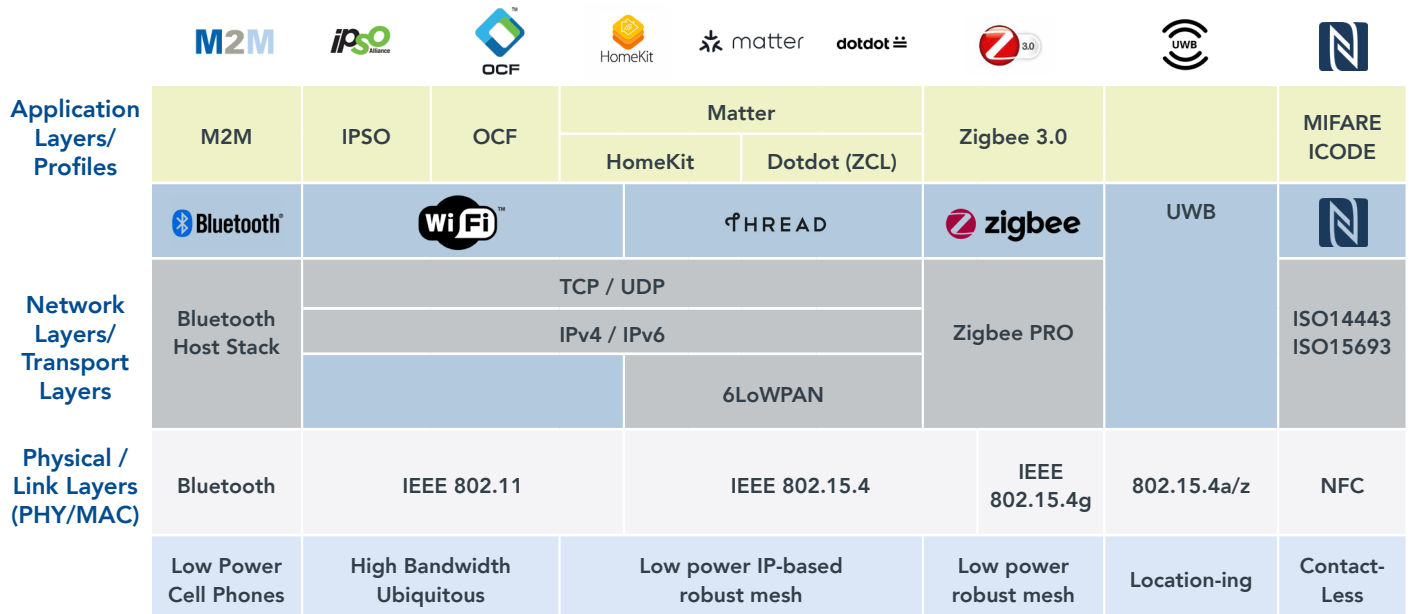


Figure 5.3. Connectivity software stack mapped approximately as OSI layers

The IoT ecosystem comprises a variety of devices with diverse purposes. End devices or end nodes, such as sensors can collect environmental, physical or mechanical data and send it to the edge device. The edge device, for example, a smart home gateway, aggregates the data and adds the necessary intelligence either locally or through the cloud. Specialized edge devices such as thermostats can perform data collection and analysis and, at the same time, support end nodes while also connecting to a gateway or the cloud.

Often the ecosystem is supported through a wireless network that allows data to flow securely and reliably between the devices. Those wireless networks use standard communication protocols to allow interoperability and an optimal user experience. Because of this, wirelessly connected devices enable an increasing number of services and applications in the diverse field of IoT such as home automation, smart buildings, industrial automation, medical devices, wearables and consumer appliances.

The reliability of these systems depends on the ability of the nodes to communicate with each other. Many end nodes and specialized edge nodes are battery powered, so the power consumption of an application needs to be managed to guarantee long battery life while maintaining functionality. This also helps reduce battery cost and maintenance.

Energy efficiency is relevant for applications that are connected to the power grid. In this case, the environmental impact of applications should be controlled or physical limits such as thermal constraints need to be addressed.

Wireless connectivity communication protocols can be proprietary or they can implement specifications proposed by global standardization organizations to achieve greater interoperability.

Multimode IoT devices often support several of these protocols. Figure 5.4 shows the connectivity software stack of a single microcontroller device that works with Wi-Fi, Bluetooth® Low Energy, Zigbee® and Thread™.

IoT Application						Bootloaders			
Application layer (Matter)			Classic BT Profiles	Bluetooth Low Energy Profiles	Mesh Profiles	Application layer (eg Matter)	Application profile (eg Zigbee Cluster Library)		MCUXPresso Middleware
wlcmgr	LwIP	DHCP	BT/BLE/Mesh core		OpenThread	Zigbee core			
Module driver			HCI		SPINEL	MAC interface		RTOS	
IEEE 802.11 MAC			Bluetooth Link Layer		IEEE 802.15.4 MAC	IEEE 802.15.4 MAC	Power Mgr	Coex	NVM
IEEE 802.11 PHY			Bluetooth PHY		IEEE 802.15.4 PHY	IEEE 802.15.4 PHY	Peripheral drivers		

Figure 5.4. Connectivity software stack of a multimode MCU

Wi-Fi

Wi-Fi® has had a tremendous impact on the modern world and will continue to do so. From our home wireless networks, to offices and public spaces, the ubiquity of high speed connectivity without reliance on cables has radically changed the way computing happens.

Over the past decades Wi-Fi technology has evolved significantly to support higher bandwidth, speed, lower latency, large number of reliable connections, Quality of Service and robust security that make it ubiquitous and quite often the preferred mode of connectivity at home and in enterprise settings.

Wi-Fi NETWORK

Wi-Fi operates in the 2.4, 5 and 6GHz bands. A typical connection involves an Access Point to which multiple client devices like smartphone, televisions, security cameras, thermostats, etc. connect to access the internet and to communicate with each other.

The Access Point usually acts as the gateway to the internet besides managing communication between devices that are connected on the local network. Access Points usually support concurrent operation in multiple bands (2.4, 5 and 6GHz). Client devices incorporate the capability to operate in one or more of the bands.

In a typical network configuration high bandwidth, latency sensitive applications like video streaming, gaming, enterprise workloads etc. are hosted on the 5/6GHz band which support higher speeds and legacy devices or IoT devices with low data and longer range requirements in the 2.4GHz band.

In addition to this, Wi-Fi allows for modes in which two client devices can establish a direct connection between two client devices without the need for an Access Point for example like connecting from a smartphone to a printer to print a photo. This is called a Point-to-Point connection or Wi-Fi Direct.

Wi-Fi GENERATIONAL NOMENCLATURE

To help users identify capability of Wi-Fi enabled devices, Wi-Fi Alliance introduced simplified generational names, based upon major Wi-Fi technology (PHY-Physical Layer) releases that may appear in device names and product descriptions. The current generation of Wi-Fi, based on the IEEE 802.11ax standard, is known as Wi-Fi 6, which includes devices that can operate in the 6 GHz band, referred to as Wi-Fi 6E.

Generational name	Technology supported
Wi-Fi 7	802.11be (in development)
Wi-Fi 6	802.11ax
Wi-Fi 5	802.11ac
Wi-Fi 4	802.11n

Table 5.1. Generational nomenclature for Wi-Fi technologies

Wi-Fi devices may also use a user interface (UI) visual on the display to identify the generation of a network connection. The visuals will display a Wi-Fi signal indicator and a numerical representation of the connection. UI visuals will adjust as users move between Wi-Fi networks that provide a different user experience. When a device displays a signal indicator visual accompanied by the number 6, indicating a Wi-Fi 6 connection, that device is utilizing the most advanced version of Wi-Fi available.




Generational of network connection	Sample user interface visual
Wi-Fi 6	
Wi-Fi 5	
Wi-Fi 4	

Table 5.2. Sample user interface visuals for Wi-Fi technologies

Source: <https://www.wi-fi.org/who-we-are/our-brands>

Wi-Fi 6/6E

Previous generations of Wi-Fi primarily focused on servicing one device at a time irrespective of the capability of the Access Point or the connected client devices. Factors such as device density and radio frequency (RF) contention can hinder the ability to achieve the peak individual device and network performance in this configuration.

Wi-Fi 6 was introduced to address these issues. Originally named “High Efficiency Wi-Fi”, Wi-Fi 6 aims to improve the overall network performance through the use of better spectrum usage, new modulation, and multi-user capabilities. The focus shift from individual device performance to holistic network performance creates a better Wi-Fi experience for all users on the network.

Wi-Fi 6 delivers faster speeds with low latency (enabled by multi-user capability to facilitate multiple simultaneous communication streams), high network utilization, longer range and power saving technologies that provide substantial benefits spanning all the way from high density enterprises, bandwidth and latency sensitive video streaming/gaming applications to enabling battery operated low power IoT devices.

To address the issue of congestion in existing 2.4 and 5GHz bands driven by rapidly growing adoption in IoT, automotive and other applications, Wi-Fi was granted up-to 1.2GHz of new spectrum in the 6GHz band. Devices that support the 6GHz band sport the 6E moniker.

Peak user data rates of multi-gigabits per second are supported by Wi-Fi 6/6E and a typical Access Point supports several dozen connected client devices. To further improve range Wi-Fi supports Wi-Fi EasyMesh, a standards-based approach protocol that ensures interoperability.

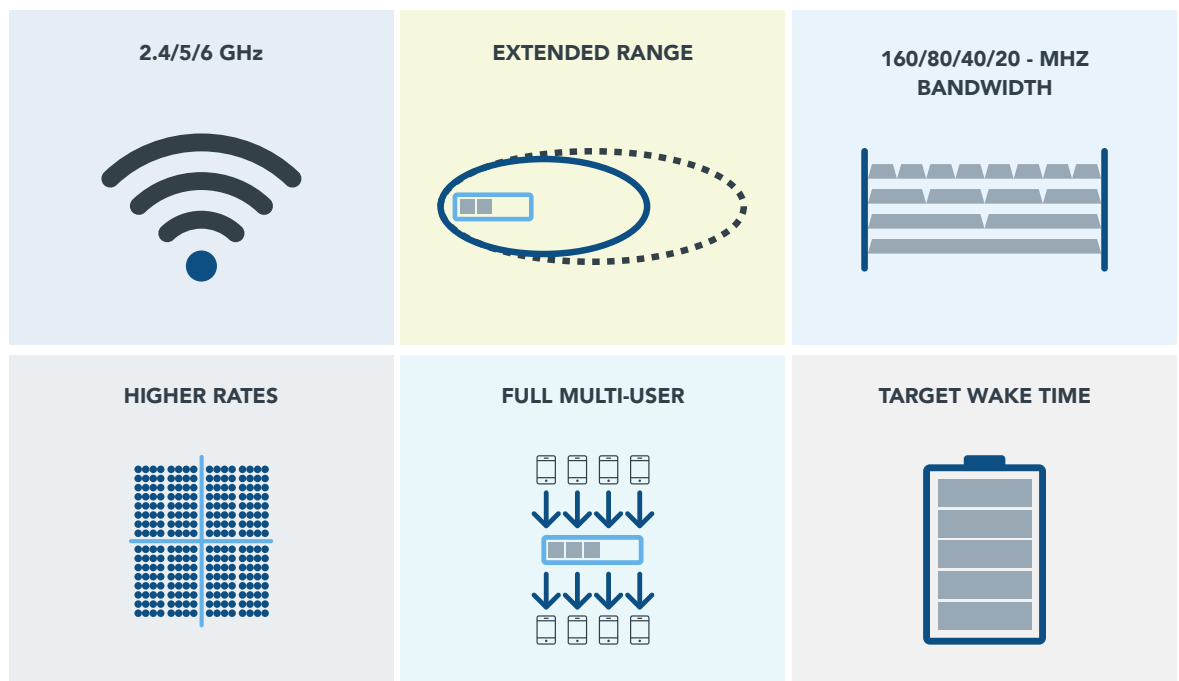


Figure 5.5. Wi-Fi 6E technology advantages

NXP offers a complete portfolio of Wi-Fi4/5/6/6E solutions, including standalone and combo devices incorporating Wi-Fi, Bluetooth and 802.15.4 connectivity with robust security.

Wi-Fi SECURITY

Since 2003, Wi-Fi Alliance has enabled individuals and businesses to increase the protection of information moving across Wi-Fi networks through the Wi-Fi Protected Access® family of technologies. Security features of Wi-Fi Protected Access constantly evolve to include stronger protections and new security practices as the security landscape changes.

The Wi-Fi Protected Access security family includes solutions for personal and enterprise networks.

Wi-Fi CERTIFIED WPA3™

WPA3™ (Wi-Fi Protected Access 3) provides cutting-edge security protocols to the market. Building on the widespread success and adoption of Wi-Fi security, WPA3 adds new features to simplify Wi-Fi security, enable more robust authentication, deliver increased cryptographic strength for highly sensitive data markets, and maintain resiliency of mission critical networks. WPA3 networks:

- **Use the latest security methods**
- **Disallow outdated legacy protocols**
- **Require use of Protected Management Frames (PMF)**

Since Wi-Fi networks differ in usage purpose and security needs, WPA3 includes additional capabilities specifically for personal and enterprise networks. Users of WPA3-Personal receive increased protections from password guessing attempts, while WPA3-Enterprise users can now take advantage of higher-grade security protocols for sensitive data networks.

WPA3 is a mandatory certification for Wi-Fi CERTIFIED™ devices.

WPA3-PERSONAL

WPA3-Personal brings better protections to individual users by providing more robust password-based authentication, even when users choose passwords that fall short of typical complexity recommendations. This capability is enabled through Simultaneous Authentication of Equals (SAE). The technology is resistant to offline dictionary attacks where an adversary attempts to determine a network password by trying possible passwords without further network interaction.

- **Natural password selection:** Allows users to choose passwords that are easier to remember
- **Ease of use:** Delivers enhanced protections with no change to the way users connect to a network
- **Forward secrecy:** Protects data traffic even if a password is compromised after the data was transmitted

WPA3-Enterprise with 192-bit mode

WPA3-Enterprise also offers an optional mode using 192-bit minimum-strength security protocols and cryptographic tools to better protect sensitive data.



Authentication: Extensible Authentication Protocol – Transport Layer Security (EAP-TLS) using Elliptic Curve Diffie-Hellman (ECDH) exchange and Elliptic Curve Digital Signature Algorithm (ECDSA) using a 384-bit elliptic curve



Authenticated encryption: 256-bit Galois/Counter Mode Protocol (GCMP-256)



Key derivation and confirmation: 384-bit Hashed Message Authentication Mode (HMAC) with Secure Hash Algorithm (HMAC-SHA384)



Robust management frame protection: 256-bit Broadcast/Multicast Integrity Protocol Galois Message Authentication Code (BIP-GMAC-256)

The 192-bit security mode offered by WPA3-Enterprise ensures the right combination of cryptographic tools are used and sets a consistent baseline of security within a WPA3 network.

Source: <https://www.wi-fi.org>

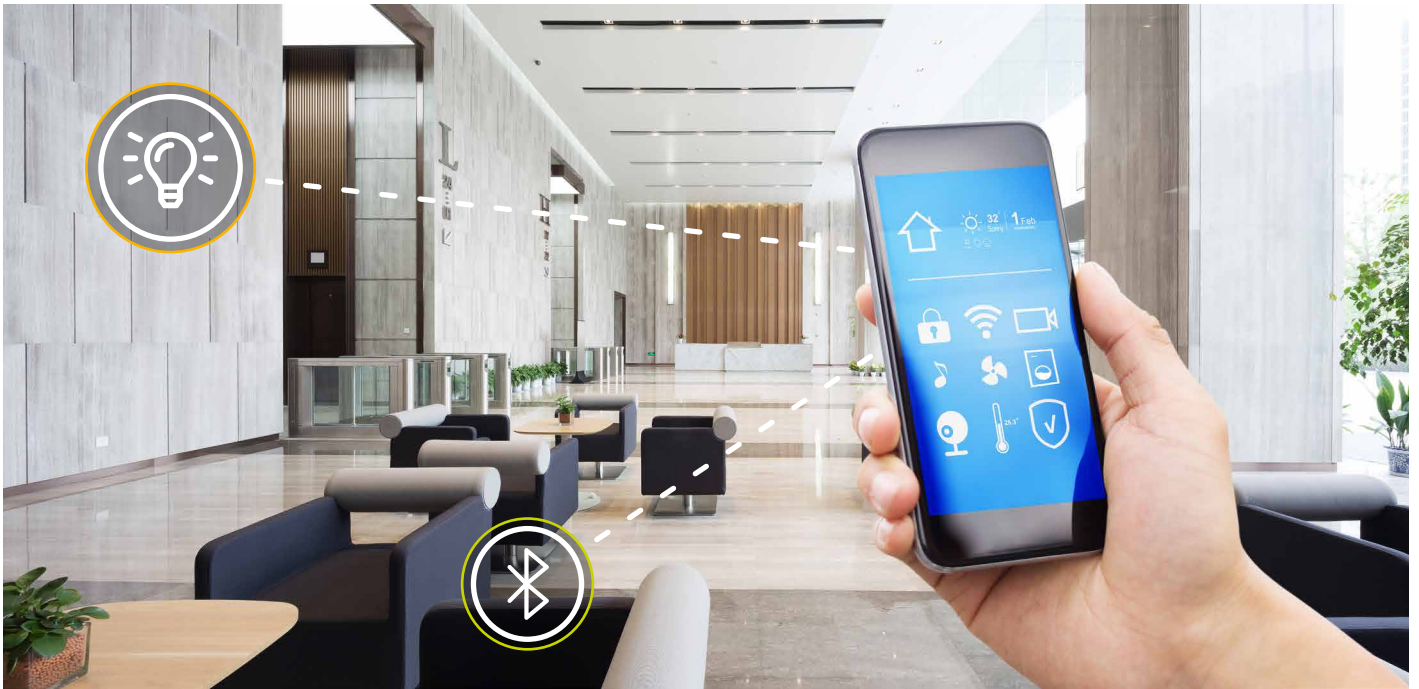
Wi-Fi IN EDGE COMPUTING

In-addition to being the commonly preferred mode of connectivity pipe in interfacing edge devices to the cloud, Wi-Fi devices at the edge can also serve other functions like less intrusive safety/security using CSI (monitor number of people in a facility, presence detection etc.). These compute heavy functions benefit from real time edge processing where the round trip delay to the cloud is not ideal.

Analytics processing and Wi-Fi rate adaptation also benefit from a hybrid approach of using edge compute capabilities for real time edge processing and the cloud for heavier workloads that are not time sensitive.

Wi-Fi COMBO CHIPSETS

NXP offers Wi-Fi chipsets based on the latest standards in Wi-Fi alone format or more commonly integrated with Bluetooth (including Bluetooth Low Energy) and 802.15.4 on the same chipset. This has the benefit of enabling compact board design, lower power consumption, reduced system cost and better co-existence between the various radios.



BLUETOOTH LOW ENERGY

Bluetooth Low Energy is a short-range radio communication technology that is independent of and incompatible with Bluetooth Classic. Still, Bluetooth Low Energy and Bluetooth Classic can coexist in the same radio spectrum.

The Bluetooth Low Energy protocol stack has excellent power consumption due to several factors such as low latency, low duty cycle (data is exchanged only on connection intervals) and smaller data packet size compared with Bluetooth Classic.

Bluetooth Low Energy hardware is usually designed to support advanced low-power modes, thus enabling long lifetimes for battery-operated devices (the common use case is a coin cell).

Typical Bluetooth Low Energy topologies are point-to-point, star and tree, as shown in Figure 5.6.

Legend

c - Central device

p - Peripheral device

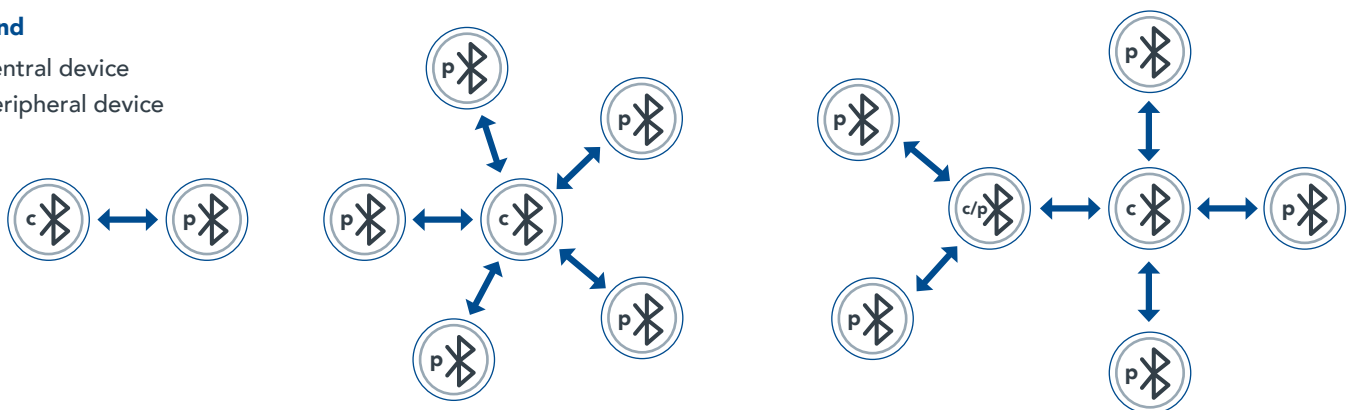
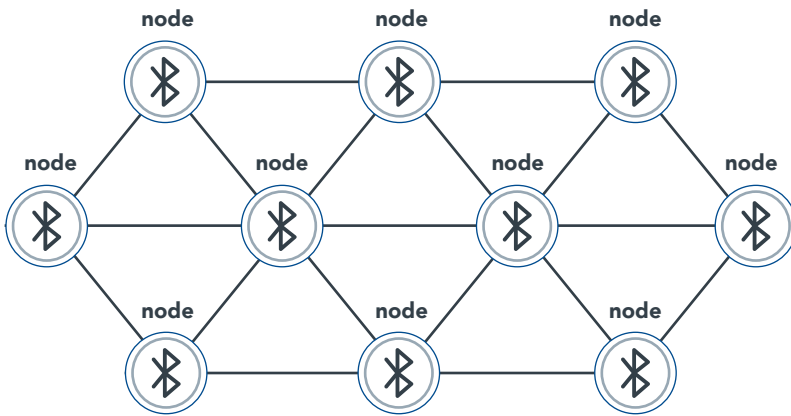


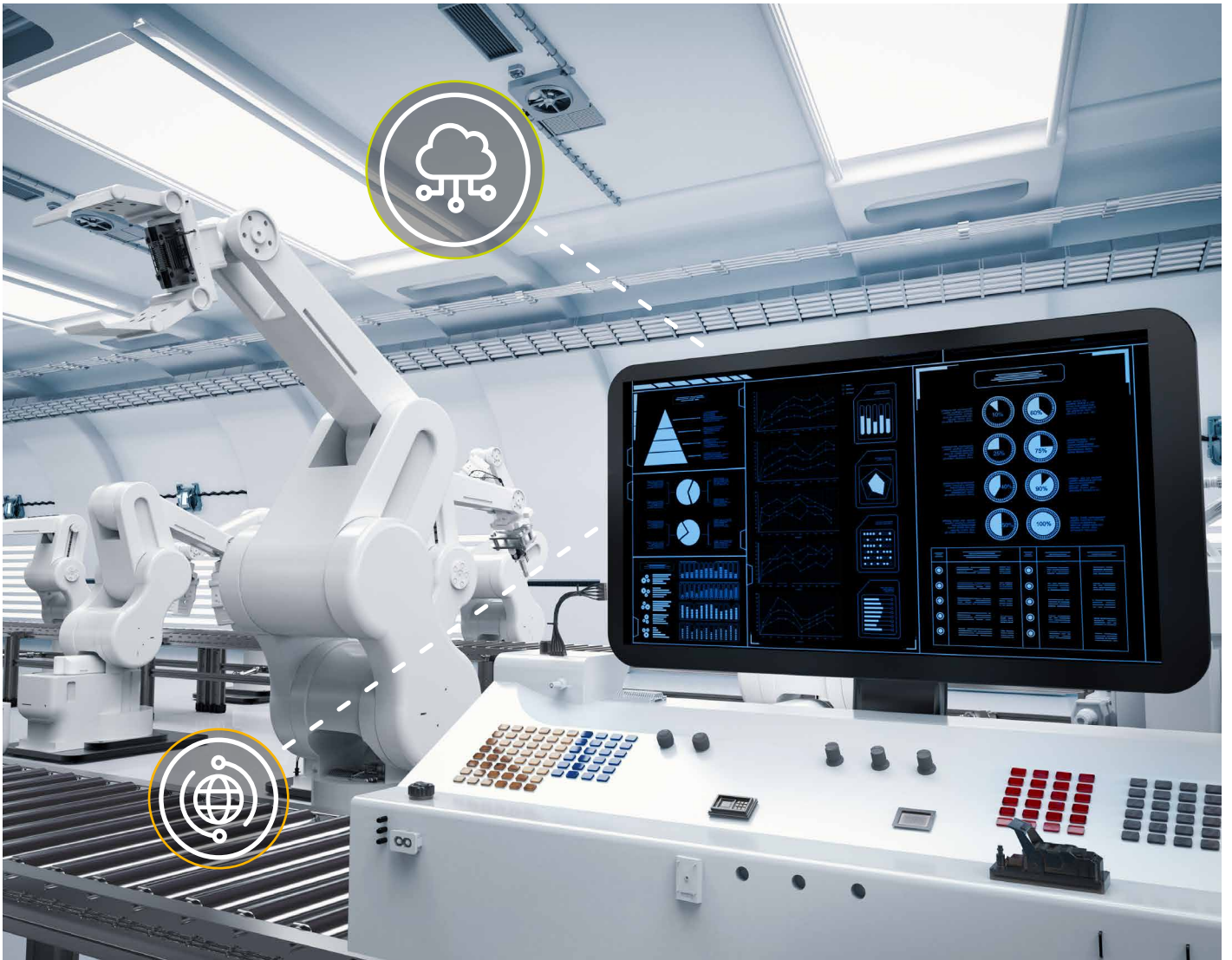
Figure 5.6. Typical Bluetooth Low Energy topologies

Bluetooth Low Energy also features the mesh network topology, which allows many-to-many communication over network radio links (see Figure 5.7).



A mesh topology that incorporates **Bluetooth Low Energy** is ideal for creating large-scale, industrial-grade device networks such as sensor, asset tracking, lighting and building automation networks.

Figure 5.7. Mesh topology with Bluetooth Low Energy



The Bluetooth Low Energy connectivity architecture is shown in Figure 5.8.

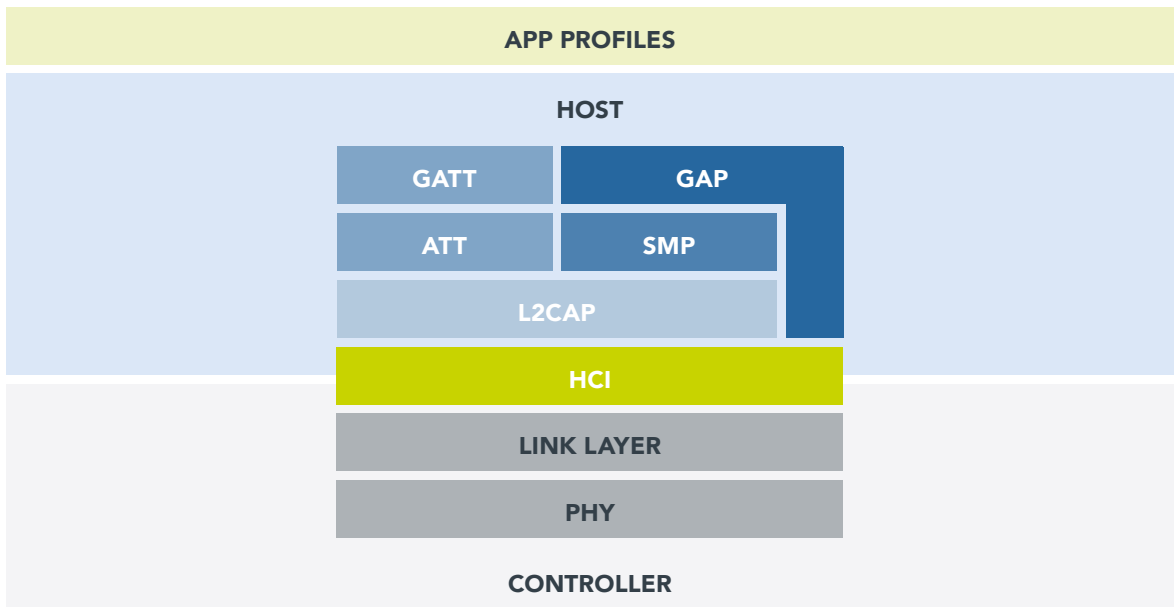


Figure 5.8. Bluetooth Low Energy architecture

The key components in a Bluetooth Low Energy stack are:

- **Application Profiles** — Include one or more General Attribute Profile services and how the services can be used to enable an application.
- **Generic Access Profile (GAP)** — Encompasses the base functionality common to all Bluetooth devices such as modes and access procedures used by the transports, protocols and application profiles. GAP services include device discovery, connection modes, security, authentication, association models and service discovery.
- **Generic Attribute Profile (GATT)** — Describes the hierarchy of services, characteristics and attributes.
- **Attribute Protocol (ATT)** — Implements the peer-to-peer protocol between server and client.
- **Security Manager Protocol (SMP)** — Manages pairing, authentication and encryption.
- **Logical Link Control and Adaptation Protocol (L2CAP)** — Allows higher level protocols to transmit and receive upper-layer data packets.
- **Host Controller Interface (HCI)** — Provides a uniform and standardized command interface between the host and the controller.
- **Link Layer** — Manages connection advertising, scanning, creation and maintenance.
- **Physical Layer (PHY)** — Configures and controls the physical parameters of the transmitted and received data.

In a typical scenario, the Bluetooth Low Energy peripheral device sends advertising packets announcing its presence. A Bluetooth Low Energy central device in scanning mode sends a connect request to the Bluetooth Low Energy peripheral, or it may alternatively request additional information from the peripheral device.

For example, the mouse in Figure 5.9 is a Bluetooth Low Energy peripheral device, and the PC is a Bluetooth Low Energy central device.



Figure 5.9. Bluetooth Low Energy peripheral and central devices

Another example is a Bluetooth Low Energy beacon device, which is a transmitter-only device that is not connectable. It uses advertising packets to send information to a central or observer device. Usually, the beacon information consists of an identifier that can be picked up by a compatible device to perform specific action(s). Common use cases are indoor positioning, tracking and message distribution at various points of interest (e.g., train station, bus stop, museum, etc.).

Bluetooth Low Energy has a variety of security features that ensure protection against passive eavesdropping, man-in-the-middle attacks and device tracking.

The following methods and techniques are used to ensure reliable security:



Pairing

Generates and exchanges one or more shared secret keys for two Bluetooth Low Energy devices so they can establish a secure link.



Bonding

Securely stores the keys exchanged during the pairing process for use in further connections.



Device Authentication

Checks that two Bluetooth Low Energy devices have the same keys.



Encryption

Transforms information from plaintext to ciphertext to ensure data confidentiality.



Message Integrity

Ensures that the data was not tampered with during transit.

There is a broad market for Bluetooth Low Energy technology. Uses cases that implement this technology range from simple beacon applications to complex mesh networks for industrial or building automation.

These markets feature associated applications:



Portable Medical

Heart rate sensor, blood pressure monitor, thermometer, glucose sensor, pulse oximeter sensor



Sports and Fitness

Cycling speed and cadence sensor, cycling power sensor, weight scale



PC Peripherals

Headphones, HID devices, gaming controllers



Automotive

Tire pressure monitoring, car sharing, remote keyless entry with UWB (see Figure 5.10), infotainment



Building and Industrial

Mesh networks



Proximity

Beacon, proximity reporter



Home Control

Appliances, lighting, HVAC, security

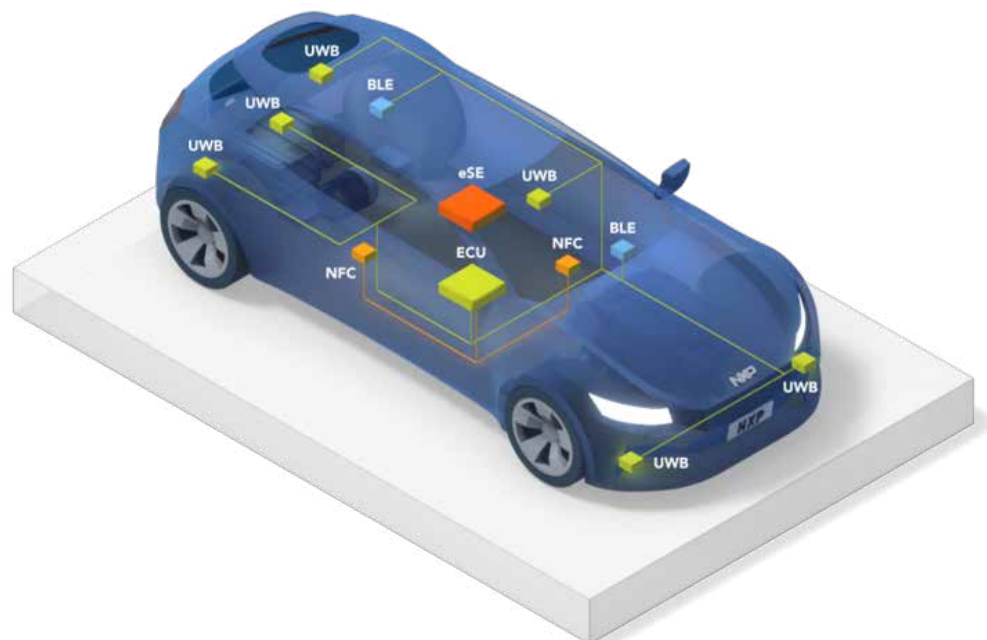
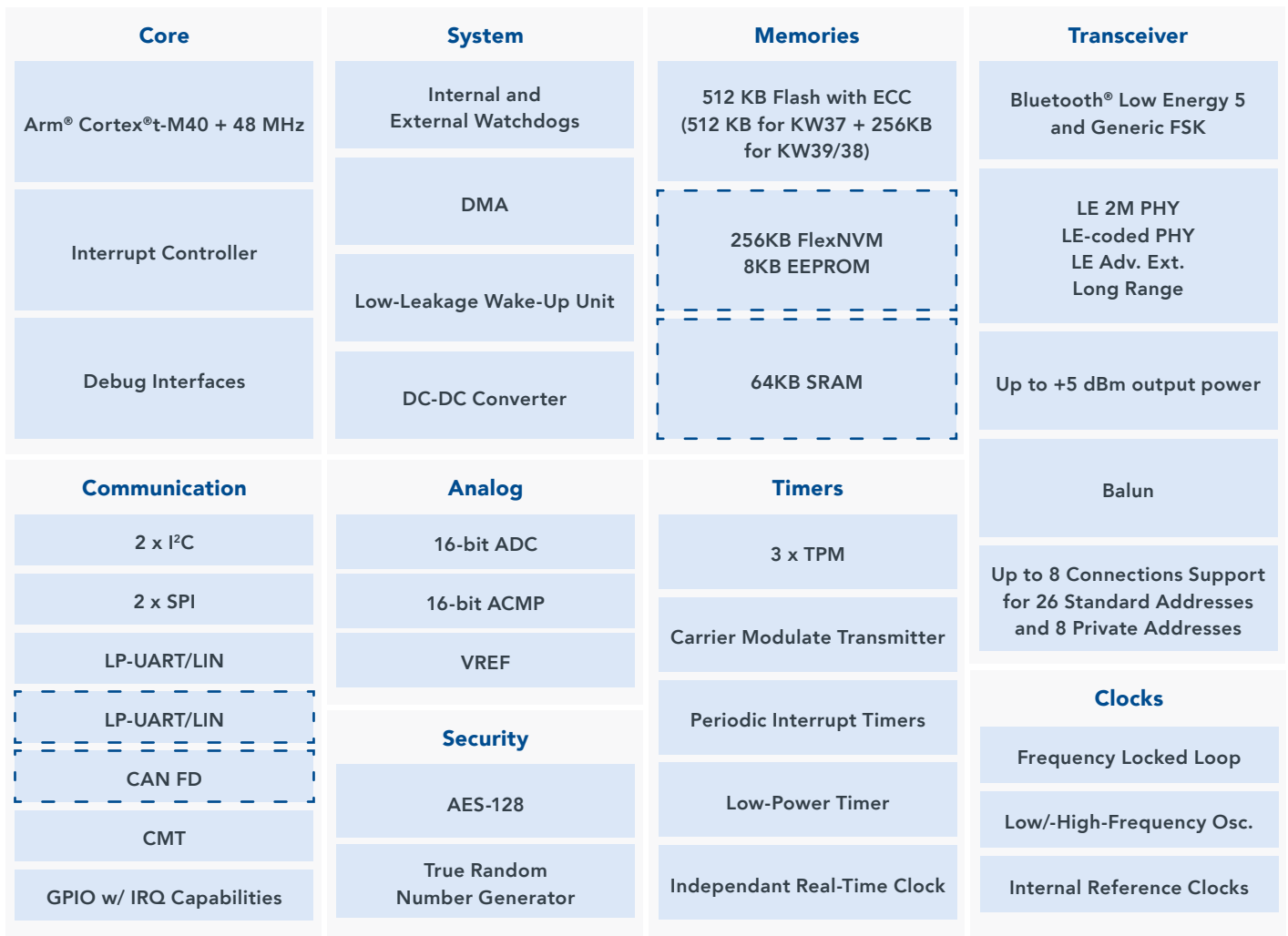


Figure 5.10. Bluetooth Low Energy helps activate UWB ranging for secure car access

Embedded microcontroller devices connect to edge processing platforms using technologies including Bluetooth Low Energy. An example is shown in Figure 5.11. This microcontroller integrates long-range capability with Bluetooth Low Energy and generic frequency-shift keying (FSK) radio. Many Bluetooth Low Energy-enabled microcontrollers can achieve -105 dBm sensitivity with 125 kbit/s data rates to enable connections in harsh environments and across extended distances. Data stream buffers allow the capture of radio parameters without stalling the processor or direct memory access (DMA) operations. This enables the high-accuracy measurements needed for distance and angle approximations. These types of radios can support up to eight simultaneous secure connections in any leader/follower combination, which allows multiple authorized users to communicate with the device. Peripherals such as FlexCAN enable integration into an automobile's in-vehicle or industrial controller area network (CAN).



[] - NXP KW39/38 wireless MCUs only

Figure 5.11. An example of a Bluetooth Low Energy-enabled microcontroller

ULTRA-WIDEBAND TECHNOLOGY

Another interesting connectivity technology is ultra-wideband (UWB), which can be used not so much for the basic transmission of information but for sensing and ranging through the concept of “localization”.

Localization is the determination of an object’s physical location, which can be pinpointed in terms of range (distance) estimation, angle estimation and joint estimation of range and angle.

UWB was originally designed for high-rate data communication competing against other technologies such as Wi-Fi. The technology has undergone several transformations. UWB’s evolution from a data communication technology based on orthogonal frequency-division multiplexing (OFDM) to an impulse radio technology specified in IEEE 802.15.4a makes it a uniquely secure fine-ranging technology. Its security extension specified in IEEE 802.15.4z (at the PHY/MAC level) adds to that security.

Range estimation

Radio waves can be used to determine the distance between devices because of these characteristics:

- **Waves are attenuated in a predictable way over distance.**
- **Waves travel at the speed of light.**
- **The number of cycles a wave completes over a fixed distance varies with the frequency of the wave.**

Unlike Bluetooth and Wi-Fi, which transmit narrowband signals and use Received Signal Strength Indicator (RSSI) to determine location, UWB transmits wideband signals (500 MHz) and uses time of flight (ToF) to determine location.

Because waves travel at the speed of light, distance can be estimated by measuring the time it takes a message to travel from one radio to another. This technique is shown in Figure 5.16.

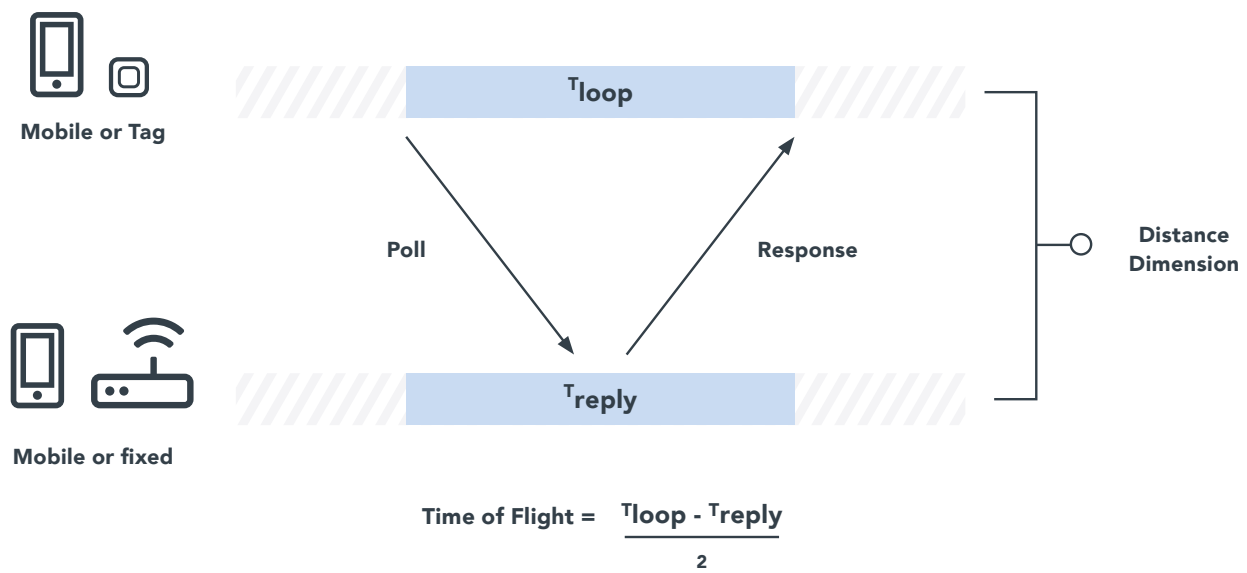


Figure 5.16. Determining a device’s relative position with UWB localization capability

Some UWB devices combine angle detection and distance estimation to fully locate the position of a device. Knowing the distance alone indicates a target device is within a ring of possible positions relative to the anchor (see Figure 5.17A). If the angle of the incoming wavefront is known, it can be used to precisely determine the position of the target device (see Figure 5.17B).

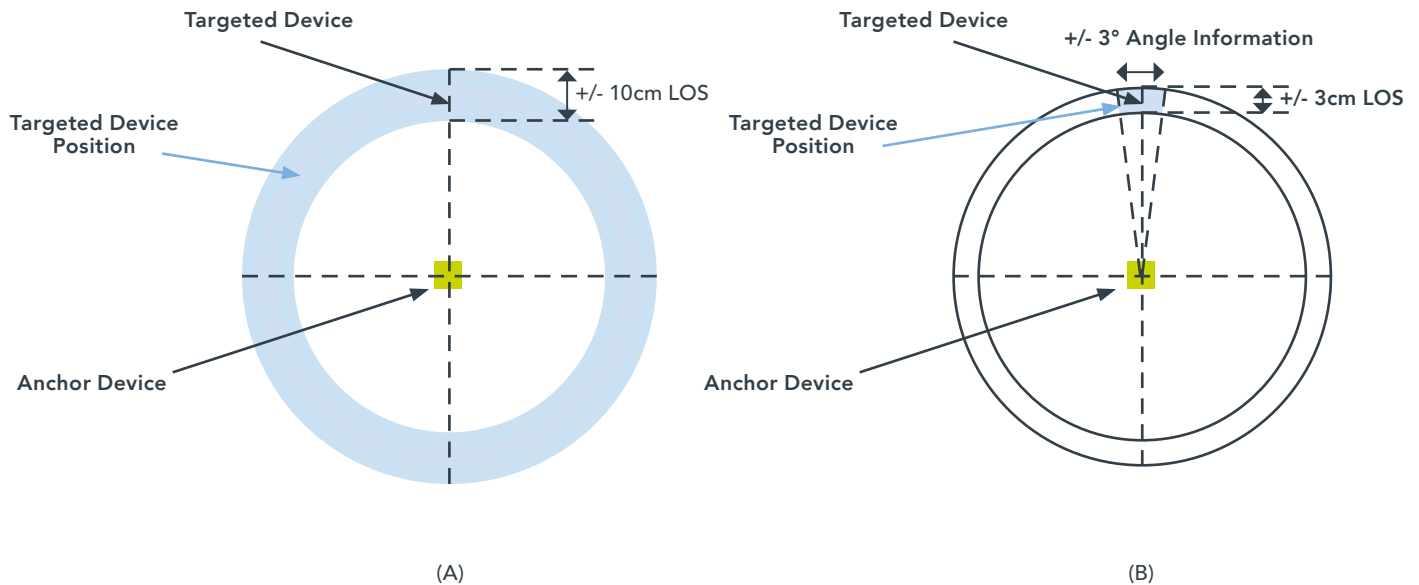


Figure 5.17. Locating a device using a combination of angle detection and distance estimation

UWB technology uses ranging and ToF for precision ranging. A UWB radio is embedded into a device such as a smartphone, wearable or smart key. When this device comes within range of another UWB-enabled device, a process called "ranging" happens. This is done by performing a ToF measurement between the devices. Depending on the application, one of the UWB-enabled devices computes a precise location of the other device. Many use cases assume one UWB device is mobile and one is fixed. For example, a smart key would be the mobile UWB device and the corresponding automobile would be the fixed device (assuming the driver is approaching a parked car).

As a short-range wireless technology like Bluetooth, Wi-Fi and NFC, UWB can operate at frequencies between 6.5 GHz and 10 GHz compared with Bluetooth's fixed 2.4 GHz. The general rule is that the higher the frequency, the shorter the range.

Under line-of-sight (LOS) conditions, UWB's operating range can stretch to 100 m. Of course, the real-world range depends on many variables in the end-product design and the environment for which it's planned. For instance, antenna design, power levels, channel frequency, propagation environment complexity and the kinds of materials that the signal may have to pass through all influence operating range. UWB performs poorly in metal environments, but it does pass through other materials such as wood, plaster and even brick, so the density of materials impacts the range. Though many applications use UWB for its short-range benefits, it can stretch a lot further.

By supporting the IEEE's work with an interoperable HRP standard, the FiRa Consortium develops service-specific protocols for multiple verticals and defines the necessary parameters for a wide-range of applications.

Because UWB is mostly used as a secure fine-ranging impulse radio technology, it fits better in the category of sensing technologies that can pinpoint objects more accurately. Other technologies such as Bluetooth Low Energy and Wi-Fi feature improved positioning accuracy, but for sensing, the physics don't compare. They still rely on a modulated sine wave carried over a narrow frequency, whereas UWB has a unique pulse signal (2 ns) operating over 500 MHz of frequency.

Bluetooth Low Energy and Wi-Fi, which use the RSSI technique, are known to be more susceptible to environmental factors, including obstructions and interference from other radios. This results in reduced accuracy. An obstruction can cause severe attenuation of signal power, leading to erroneous measurements that can be off by multiple meters.

Because of this, UWB is a satisfactory option for localization with good accuracy and reliability under a variety of conditions.



Figure 5.18 shows some UWB applications:



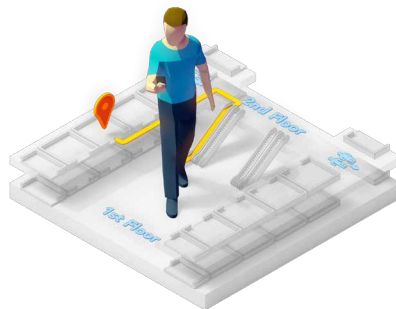
Hands-free payments

Leave a UWB-enabled phone in a pocket, in a bag or mounted on a dashboard and still make a payment. Also trigger personalized ads and offers that reflect personal interests and preferences.



Secure hands-free access

Automatically unlock a UWB-enabled door lock on a car, front door or warehouse just by being near it. ToF calculations prevent UWB from susceptibility to relay attacks based on signal amplifications.



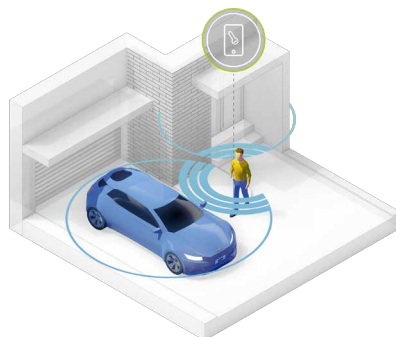
Indoor navigation

Navigate large buildings such as shopping malls and easily find a car, a store, a friend and even items with the precision of a few centimeters using GPS-style location services indoors.



Item tracking

Set up lights, speakers and any other connected device with UWB sensing capability to follow users from one room to another. UWB-enabled objects can be pinpointed instantly using a mobile device.



Credential sharing

Share access credentials to a rental, coworking space or apartment securely and grant temporary access. Smart cars will allow specifications for a car's use, including feature unlocking.

Figure 5.18. UWB applications

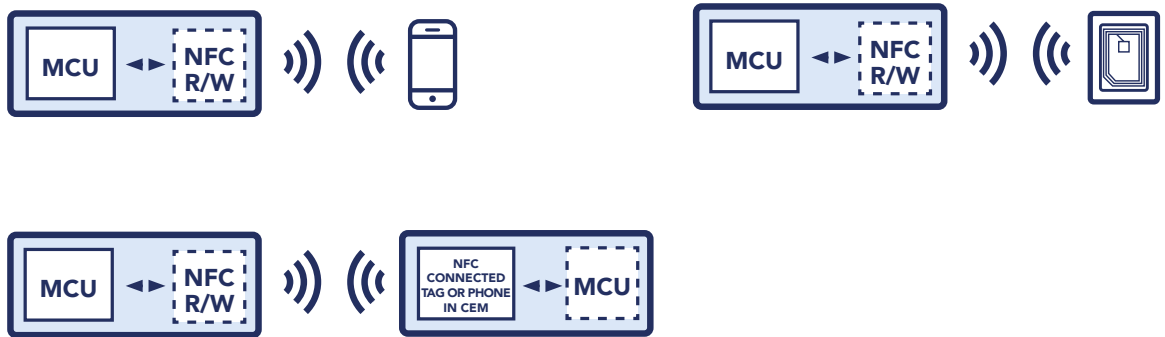
NFC

Near Field Communication (NFC) delivers short-range communication with the ability to store and transmit data in much the same way that Radio-frequency identification (RFID) tags and contactless smartcards do. NFC is a proximity technology, which means it only works when two devices are brought close together or actually touch. When devices aren't near each other, NFC is dormant, so it's not drawing power or sharing information when it shouldn't. Only one of the devices needs to be powered for a two-way interaction to take place. The second device can save its battery for other things, or not have a battery at all.

NFC Communication Modes

Read/Write Mode

This is where NFC spends most of its time, with one NFC-enabled device interacting with another to get information or initiate an action. The initiating device can read data in from the second device or write data out to it.



Peer-to-Peer Mode

Sometimes referred to as "P2P" mode, this is the one you can use to exchange files between smartphones, or receive loyalty points when making a purchase.



Card Emulation Mode

This mode, used almost exclusively by NFC smartphones, lets the system behave as an ISO/IEC 14443-compliant contactless smartcard. That means your phone can be used in the existing contactless infrastructure, for things like ticketing, access control, transit, tollgates, and payments. The mode takes very little power, and can work even when the phone is off.

NFC for Edge Computing

NFC is a well-established technology for access control and contactless mobile payments. Besides the proximity technology offers new opportunities since any NFC-enabled phone or tablet can serve as a temporary touchscreen for another product, enabling sophisticated interactions and increased configurability.



PAIRING & COMMISSIONING

- Enable two-way interactions with Peer-to-Peer mode
- Pair Bluetooth or Wi-Fi devices faster with NFC
- Identify a device instantly, without entering codes or creating device conflicts
- Exchange credentials securely, just by tapping
- Use protocol-agnostic operations to trigger actions



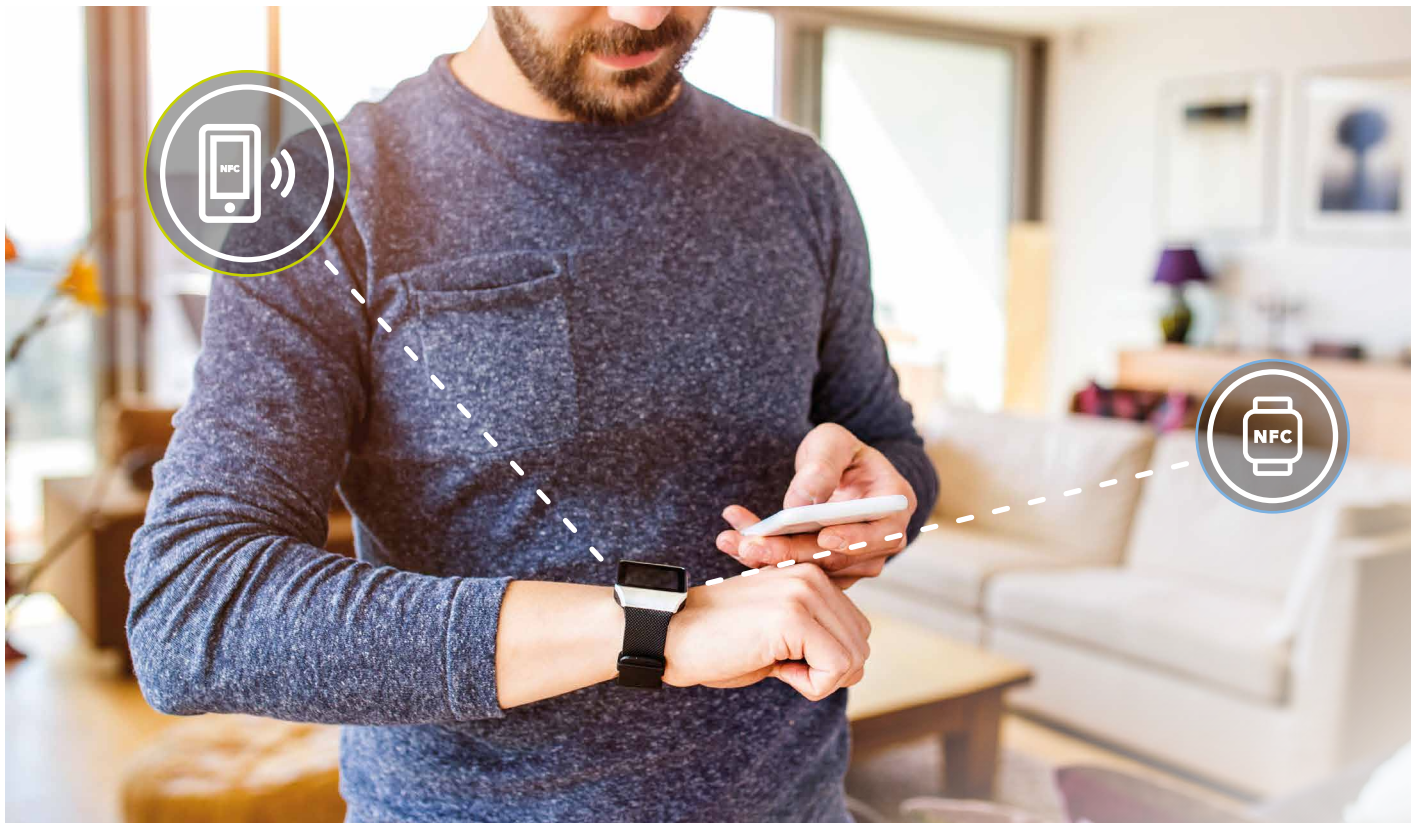
AUTHENTICATION & IDENTIFICATION

- Authenticate replacement parts and automatically adjust settings of the main unit based on the accessory attached
- Identify users and immediately provide personalized settings
- Send notifications when accessories are nearing replacement, and make offers based on usage patterns



PARAMETRIZATION & DIAGNOSIS

- Device can be unpowered
- No ambiguities - the device you tap is the device you connect to



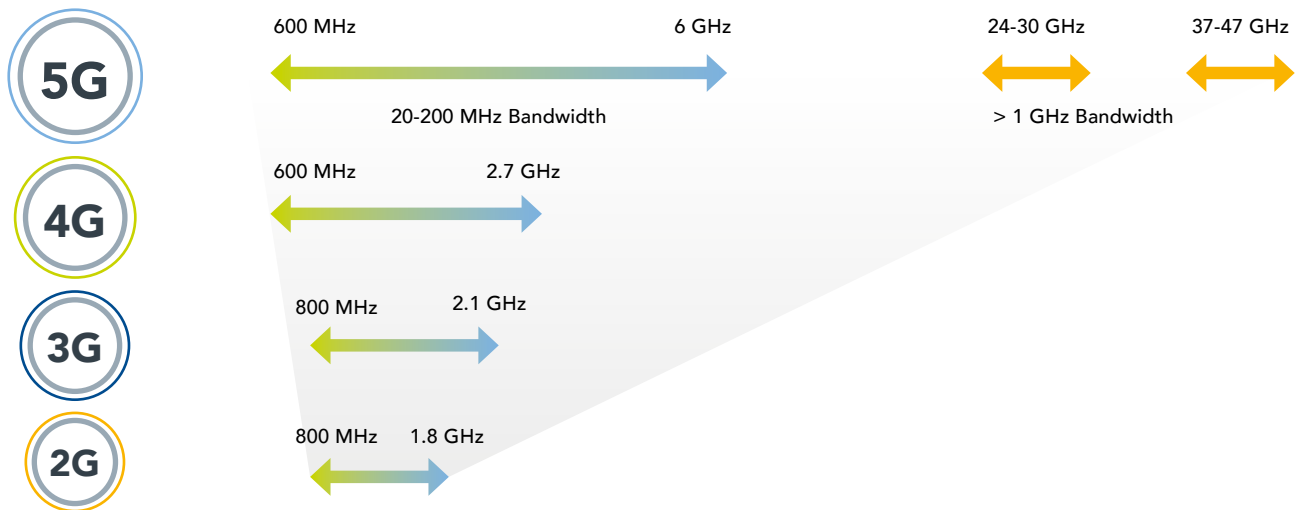
5G

5G is the fifth generation of wireless technology for broadband cellular networking. This generation of technology will provide increased data speeds into the multi-gigabit per second range, larger network capacity and significantly improved latency. From an edge processing perspective, 5G will enable real-time edge computing and the low latency required for machine learning and artificial intelligence on the edge. It also will support other applications such as vehicle-to-everything (V2X) connectivity, virtual reality applications and industrial smart manufacturing.

Three things that set 5G apart from previous generations are 1) its operating location in the wireless spectrum, 2) the antenna structure used to transmit and receive signals and 3) the transition to a more software-based approach to managing and optimizing operation.

- **New spectrum**

As shown in Figure 5.19, 5G extends the existing cellular spectrum to include the area between 2.7 GHz and 6 GHz and adds a completely new part of the spectrum above 25 GHz. This new portion of the spectrum, known as millimeter wave (mmWave), was previously reserved for other services such as medical imaging, microwave remote sensing, amateur radio, terahertz computing and radio astronomy. It will enable ultra-high bandwidth and ultra-low latency use cases, but it presents a steep learning curve for engineers used to working below 6 GHz.



5G adds new frequency spectrum

- **New antenna configurations**

5G uses active antennas, which are more highly integrated and complex than the passive antennas traditionally associated with cellular. Active antennas require a sophisticated mix of hardware and software and use massive multiple input, multiple output (mMIMO), a technique that involves dozens (if not hundreds) of antennas working together to expand capacity within the same bandwidth. Working with so many antennas is a complex, compute-intensive task that requires careful optimization to ensure reliable, interference-free operation.

- **More code**

5G takes advantage of virtualization, with more being implemented in the cloud, and often uses machine learning algorithms for optimized network management, orchestration in the core, traffic monitoring and load balancing. A typical 5G base station has millions of lines of code, software to add new features like support for more devices, increased capacity and expanded coverage to accommodate more traffic. Heavy reliance on software changes how the network is deployed and operated. It also changes the security models.

Preparing the infrastructure for 5G operation, through what's known as 5G densification, involves adding different layers of coverage. Each layer provides the throughput improvements needed for a given area or use case. As shown in Figure 5.19, these layers consist of traditional 5G macro cells, 5G mMIMO cells, 5G mmWave cells and small cells.

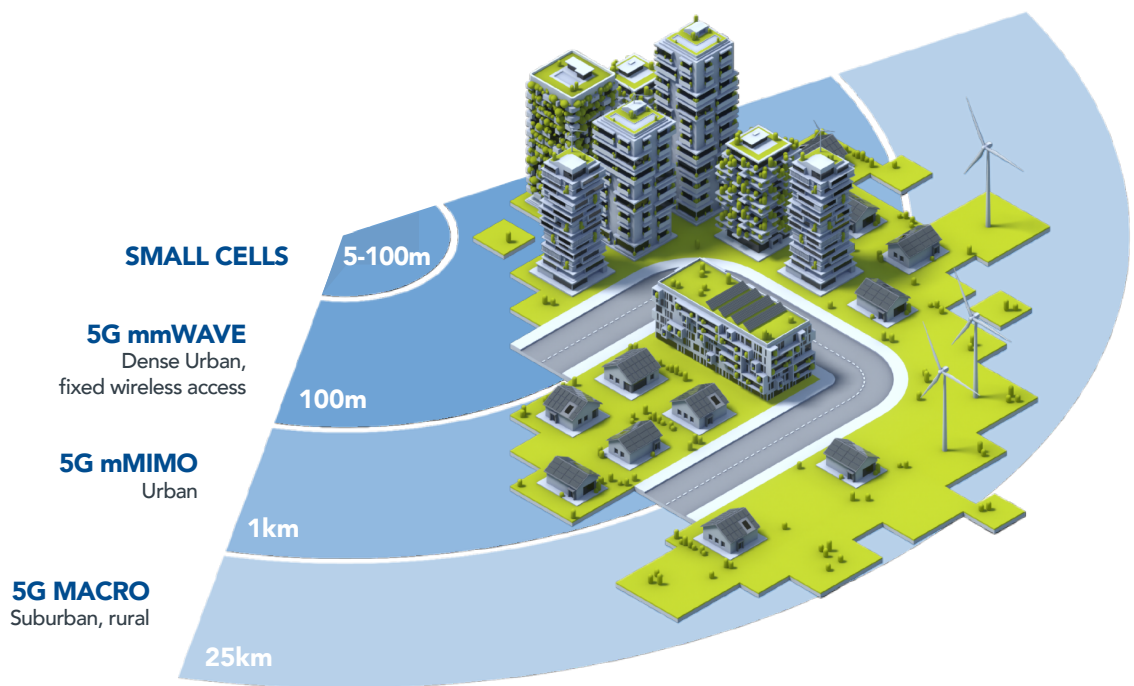


Figure 5.19. 5G densification for capacity in layers

- **Traditional 5G macro cells**

Servicing a wide area of about 25 km, traditional 5G macro cells mainly provide capacity in suburban and rural environments. Similar to their 4G counterparts, traditional 5G macro cells are big, high-powered base stations that live on towers, monopoles and rooftops. They are sometimes made to look like giant trees. They typically incorporate a passive antenna system and a simple MIMO configuration for transmission and reception that either uses four antennas (4T4R) or eight antennas (8T8R) driven at 40 W.

- **5G mMIMO cells**

Servicing a smaller area of about 1 km, 5G mMIMO cells are used to provide capacity in urban environments that have a higher device density than suburban and rural areas. The traditional macro cell's passive antenna and radio unit are replaced with an active antenna system. The active antenna system combines mMIMO configurations with other 5G features, such as beam forming, to increase throughput while reducing interference. Typical configurations use 32 antennas driven at 10 W or 64 antennas (64T64R) driven at 5 W.

- **5G mmWave cells**

Servicing an area of about 100 m, 5G mmWave cells use the new spectrum above 25 GHz to provide capacity in urban environments with very high device density and to support fixed wireless access (FWA) in buildings. Working in the mmWave spectrum means exceptionally high bandwidth, made possible using hundreds of antennas in a mMIMO configuration driven at 200 mW, but because mmWave signals have a lower wavelength, they don't travel as far. To balance the tradeoff between bandwidth and power consumption, 5G mmWave cells have limited range.

- **5G Small Cells**

Servicing an area of between 5 m and 100 m, 5G small cells are backpack-sized, low-power base stations that provide targeted capacity in network "hotspots". Small cells are compact and lightweight, so they can be mounted just about anywhere, and they prevent 5G signals from being dropped in crowded areas, such as city centers or sports venues. A 5G small cell is, in many ways, a miniature, low-power version of a traditional 5G macro cell. It typically uses a passive antenna with a smaller MIMO setup of just four antennas (4T4R) driven at a much lower power of just 1 W.



5G and Wi-Fi

Cellular and Wi-Fi are complementary technologies. Today's smartphones already combine the two, making it possible to switch from cellular to Wi-Fi when the cell signal is weak or to save on data usage. This trend is expected to continue, with Wi-Fi 6/6E and 5G small cells working in tandem to support more devices accessing more data, all at once.

5G can even boost the deployment of Wi-Fi 6/6E by making it easier to connect Wi-Fi signals to the core network. In FWA applications, for example, 5G mmWave cells can be used as the backhaul service, replacing the expensive fiber-optic cabling currently used to link many Wi-Fi gateways to the core network.

Figure 5.20 shows an example of 5G and Wi-Fi technologies coexisting in an infrastructure to support edge processing.

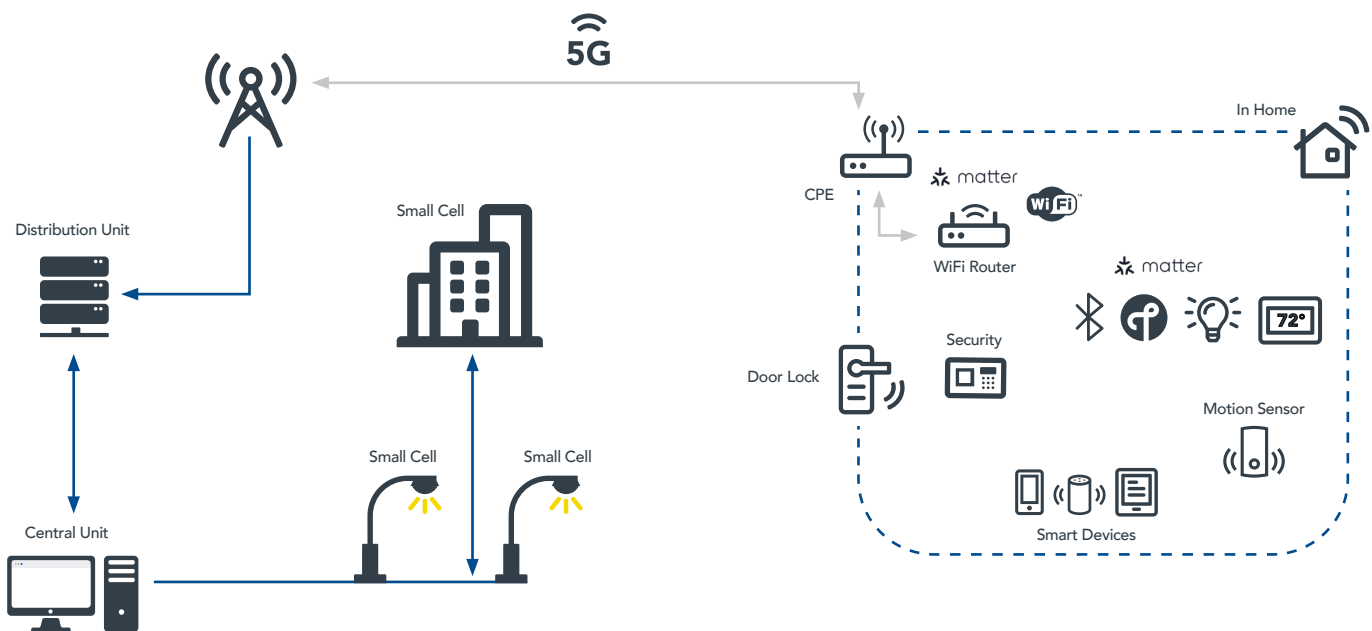


Figure 5.20. Coexistence of 5G and Wi-Fi to support edge processing

Edge processing includes 5G cellular and wireless connectivity for wireless router applications to manage network traffic and network connectivity through high-performance processors in distribution and radio units. This includes not only infrastructure but also small cells on streetlights and in buildings as well as equipment in homes. Central units leverage network interface controllers (NICs) to route data across the network.

In the home, wireless technologies including Thread, Bluetooth Low Energy and Wi-Fi provide connectivity in devices such as smart assistants, security systems, smart thermostats, light bulbs and many other applications. Distribution units route cell tower data to and from the central unit using network processors. Small cells, like distribution units, contain a radio to handle RF signals and communicate with the central unit without a distribution unit.

Customer premises equipment (CPE) technology is similar to a wireless modem and router except it connects to a cellular network rather than a wired internet connection. Designed to operate in the home, CPE provides internet service through the cellular network. This enables an FWA solution for last-mile gigabit broadband connectivity to homes on 5G networks. The infrastructure in Figure 5.20 provides the performance, security and scalability required for edge processing.

Reaching the cloud

One of the first factors that product makers must consider when defining a new product is whether it will be a “connected” device. IoT “air gapped” products are increasingly rare. Designing a connected device benefits the device maker and the end customer. Device makers can easily manage the millions of devices as they are released, and consumers can benefit from the automatic firmware updates and diagnostics. To deliver the additional capabilities, product developers are turning to the cloud. From scalability to expanded compute capability, cloud services can give more life to the lowest end electronic devices.

Cloud providers such as Amazon Web Services (AWS), Microsoft® Azure® and Google Cloud provide services to the IoT. Traditionally, cloud providers have focused on the back-end aspects of business operations, including hosting web presences and storage, so their customers can concentrate on their content and data. To help drive the acceptance of cloud services in the IoT space, cloud providers offer services that benefit IoT product developers. Services like over-the-air (OTA) updates, data aggregation and analytics, and device management provide IoT device makers with the opportunity to focus on their products (see Figure 5.21).

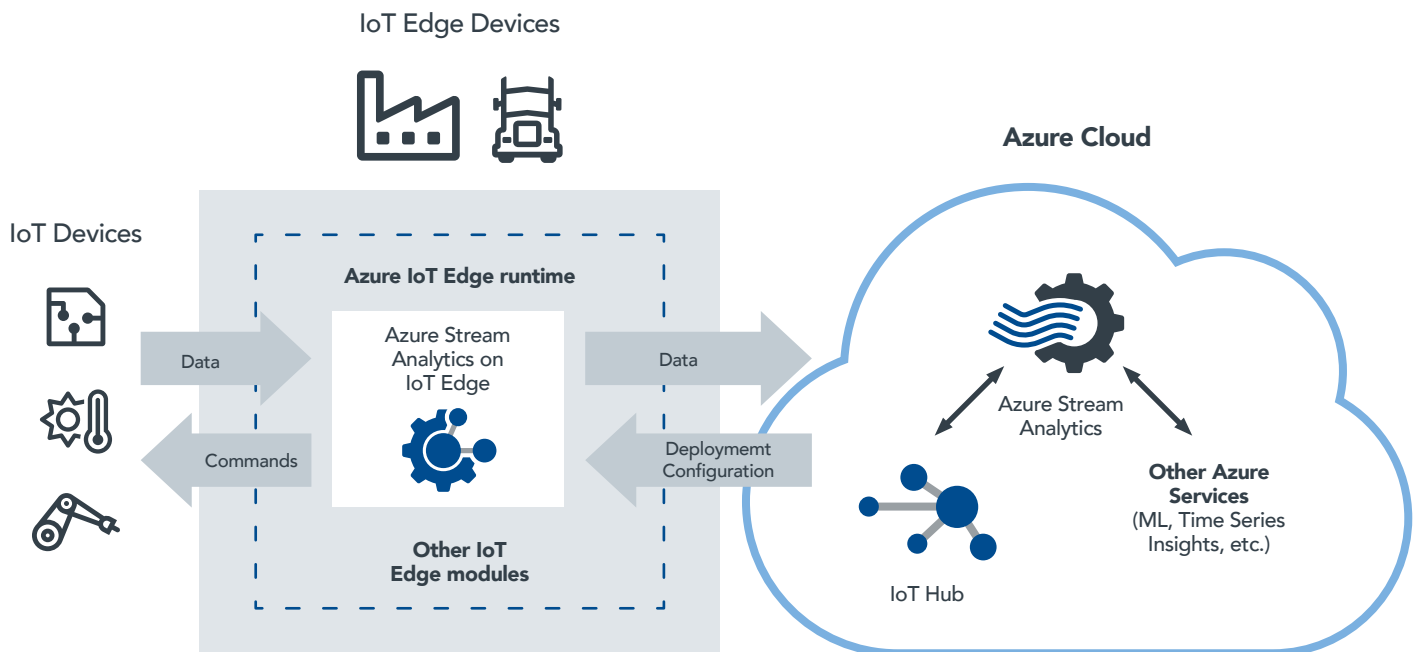


Figure 5.21. Edge processing connectivity to the cloud using Microsoft as an example.

There are several cloud provider service options. The more traditional way to access the cloud is through a direct IP connection over Wi-Fi. This solution is the most convenient because it allows the device to directly attach to cloud services, but power requirements for Wi-Fi are substantially higher than other connectivity options. Device makers also have locally hosted cloud service options. Connections to a locally hosted cloud solution via an IP connection over Wi-Fi also allow for the introduction of a border router, which provides other wireless communication options such as Bluetooth Low Energy, Zigbee and Thread. A cloud service provider consolidates all the cloud access in a local edge server, which then communicates with cloud services (see Figure 5.22).

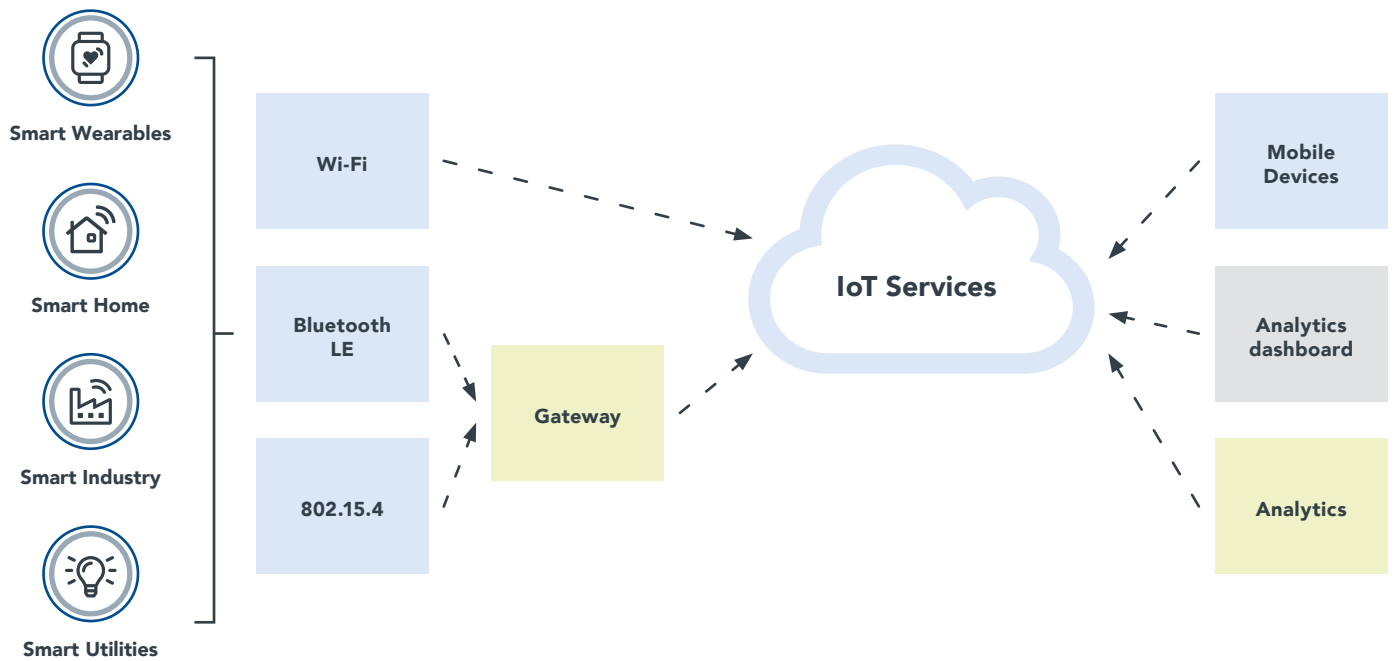


Figure 5.22. Edge to cloud connectivity

Attaching to cloud services is getting easier. Cloud providers such as AWS and Microsoft now have embedded RTOS technology such as FreeRTOS and Azure RTOS, respectively, to more easily enable cloud connectivity from an IoT device. IoT software development kits (SDKs) provide integrated support for the major cloud providers.

Different connectivity options are available from Wi-Fi, NB-IoT and other narrowband technologies such as Bluetooth Low Energy and Zigbee. Each of these options provides different benefits depending on the end goal. One of the major considerations in any IoT device design is power consumption.

Cloud provider SDKs

SDKs provide the software necessary to connect to the cloud provider. Each of these pre-integrated SDKs offers a starting point for device makers to help build their cloud-connected products. Though each cloud SDK varies slightly, they all focus on two main components: connectivity and security.

For example, the AWS IoT Device SDK is built around the FreeRTOS kernel. In addition to the kernel, Amazon provides critical connectivity and security services to create a cloud-connected device. To ease the connectivity aspects of accessing AWS IoT Core, Amazon developed a Wi-Fi management library. This library abstracts the underlying Wi-Fi stack from the upper-level application. The Wi-Fi library provides features such as authentication (WEP, WPA, WPA2, WPA3), access point scanning, power management and network profiling. This library makes it possible for device makers to maintain portability with their application while swapping out the underlying Wi-Fi hardware or software.

Amazon also provides a Bluetooth Low Energy library for Bluetooth Low Energy-connected devices. These devices can subscribe and publish to Message Queue Telemetry Transport (MQTT) topics through a Bluetooth Low Energy proxy device, such as a mobile phone. The library provides support for configuring Wi-Fi networks, data transfer and network abstraction over Bluetooth Low Energy. In addition to these AWS connectivity features, the library offers application programming interfaces (APIs) for directly accessing lower-level Bluetooth Low Energy stack functionality. The benefits of using an abstraction layer like this are similar to the benefits of the Wi-Fi management library: the greatest portability for the end-user application.

Once a physical connection is made to the AWS cloud, it must use one of two protocols to communicate with AWS: MQTT or Hypertext Transfer Protocol (HTTP). Through the MQTT library, the user application can easily subscribe, unsubscribe and publish messages to a given topic. The popularity of the MQTT core library in embedded microcontrollers is due to its small footprint. The HTTP library provides a similar set of abstracted HTTP client interfaces used to access HTTP/1.1 powered services. Though the HTTP client offers a more flexible approach to cloud services, it comes at a price. The HTTP target library is nearly 3X bigger than the MQTT core library.

Another important aspect of cloud-connected applications is the security. Access to the AWS IoT Core is controlled through public key cryptography (PKC), namely the PKCS11 standard. This standard documents the way in which secrets are exchanged between the end-user device and the cloud. Amazon has simplified the way user applications establish a PKCS11 session through their PKCS11 Library. The library helps with both device provisioning (should this device be granted access to a given cloud service) and transport layer security (TLS), the ongoing communication of the end-device with the cloud services. In both cases, the application developer should use the PKCS11 Library to perform these tasks.

Users interested in creating their own AWS-connected device can follow these steps to get connected:

- Create an AWS account.
- Configure a “thing” in the AWS IoT Console.
- Make sure you have internet access for your device.
- Open and build the example project.
- Connect a USB cable between the PC host and the OpenSDA port on the target board.
- Download the program to the target board with the CMSIS-DAP or J-Link debugger and execute your cloud-connected application on the target.

Application Layer

The connectivity protocols discussed here explore a broad range of coverage and data rates required for various edge processing applications (Figure 5.23). This also requires edge devices to support multiple protocols, sometimes operating simultaneously in many use cases in the smart connected world.

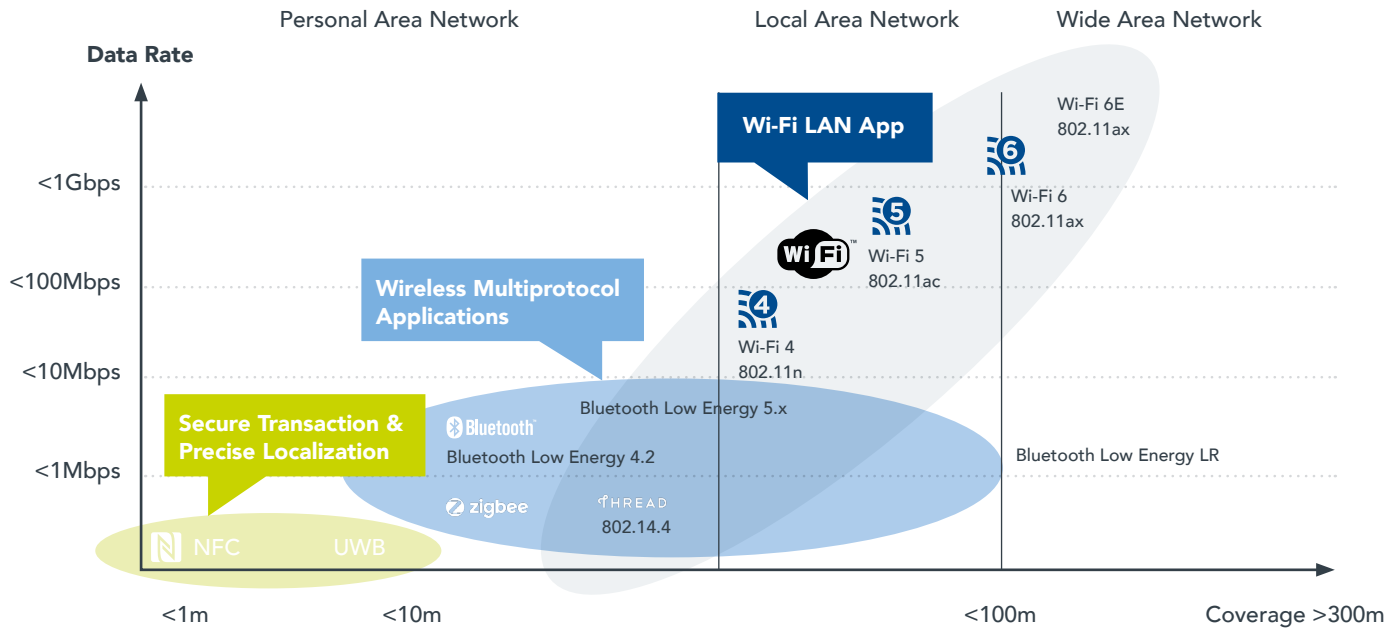


Figure 5.23. Multiple data rate and coverage requirements for edge connectivity with accompanying connectivity protocols

Figure 5.24 shows an embedded device that supports Bluetooth Low Energy and IEEE 802.15.4 protocols concurrently in a single chip. The SDK connectivity software has a mobile wireless system (MWS) coexistence software component that arbitrates the use of the radio hardware resource. It is essentially a set of APIs that allow higher layers of the software to request access to the radio resource. MWS natively gives priority to Bluetooth Low Energy, allowing it to abort ongoing IEEE 802.15.4 transactions even when they have already started. If this happens, the IEEE 802.15.4 transaction restarts when the Bluetooth Low Energy transaction completes.

Application			
Bluetooth Low Energy profiles		Zigbee or Thread	
Generic Attribute Profile	Generic Access Profile	MAC API	Security Library
Attribute protocol	Security Manager		
L2CAP		PHY API	Transceiver Manager
HCI			
Link Layer		Packer Processor	Sequence Manager
Mobile Wireless Systems Coexistence MWS			
2.4GHz Transceiver			

■ Hybrid application ■ Bluetooth Low Energy ■ Zigbee ■ IEEE 802.15.4 ■ Hardware

Figure 5.24. Mobile wireless system coexistence

Matter

The wide variety of connectivity protocols present a lot of complexity when determining how to structure a system requiring different protocols that aren't necessarily interchangeable (Figure 5.25). This can be expensive, wasteful and resource-intensive. Developers have to consider multiple formats for physical/media, network, transport and application layers.

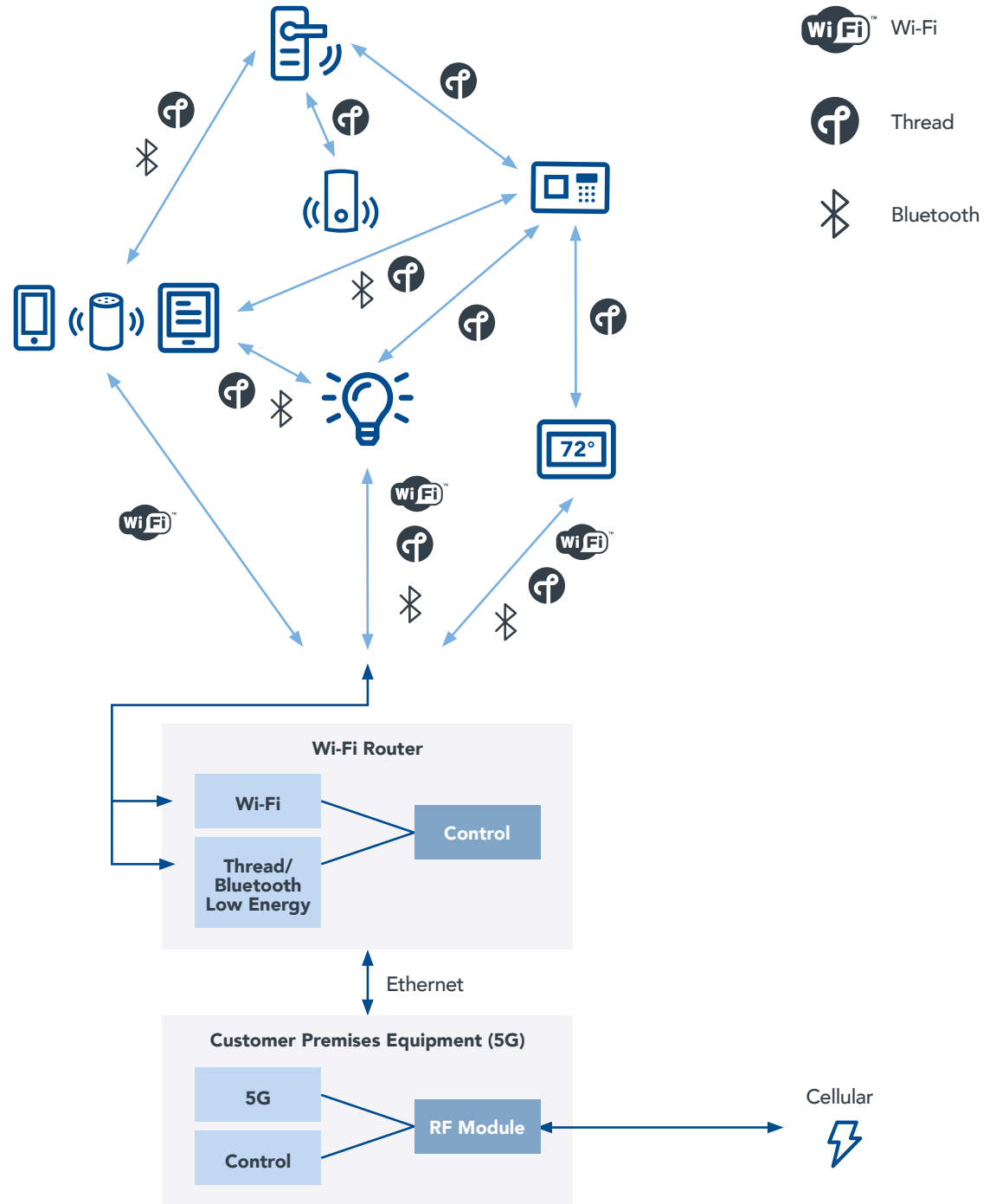


Figure 5.25. Complexity created by multiple connectivity protocols

This is especially true in the smart home market where consumers are challenged by the interoperability of devices made by different companies that use different standards. For example, sensors, door locks and other devices that require a low data rate for control and configuration need to be very power-efficient. The standard requirement is that a given device needs to run for two to five years on a single small-sized battery. The short-range network protocols that support this kind of low-power operation include Zigbee, Thread, Bluetooth LE and Bluetooth Mesh. Thread and Zigbee PRO are mesh technologies based on the IEEE 802.15.4 radio providing for robust networks with Thread based on IP. Bluetooth Low Energy is designed for point-to-point communications. The other end of the spectrum contains devices like video cameras and appliances with large, interactive screens, which use more data and consume more power. These devices require a Wi-Fi connection. Depending on the end application, one or multiple protocols will be used. For example, Bluetooth Low Energy is often used to join devices to an existing home automation network and then Thread or Zigbee is used to interact with other devices on the same network (see Figure 5.26).

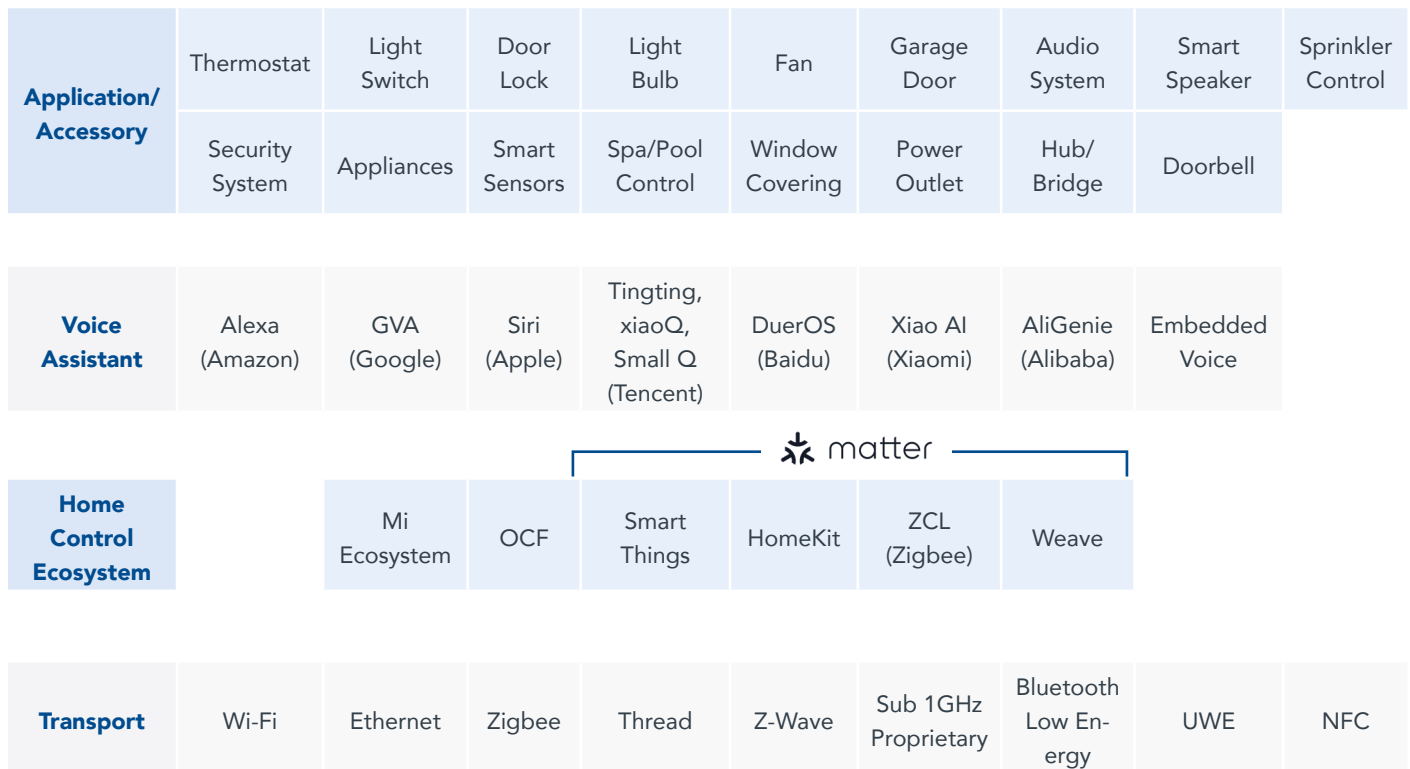


Figure 5.26. Multiple transport protocols for home control

Matter is a unifying, IP-based connectivity protocol built on proven technologies designed to enable developers to connect and build reliable, secure IoT ecosystems and increase compatibility among smart home devices. Part of the Connectivity Standards Alliance, Matter is an open-source royalty-free connectivity standard. The long-term goal is to simplify IoT development, increase compatibility for consumers, ensure security and privacy, and create truly smarter homes. Connectivity Standards Alliance board member companies Amazon, Apple, Google, IKEA, Legrand, NXP Semiconductors, Resideo, Samsung SmartThings, Schneider Electric, Signify (formerly Philips Lighting), Silicon Labs, Somfy, Wulian and more than 200 other companies are leading the efforts and adopting Matter.

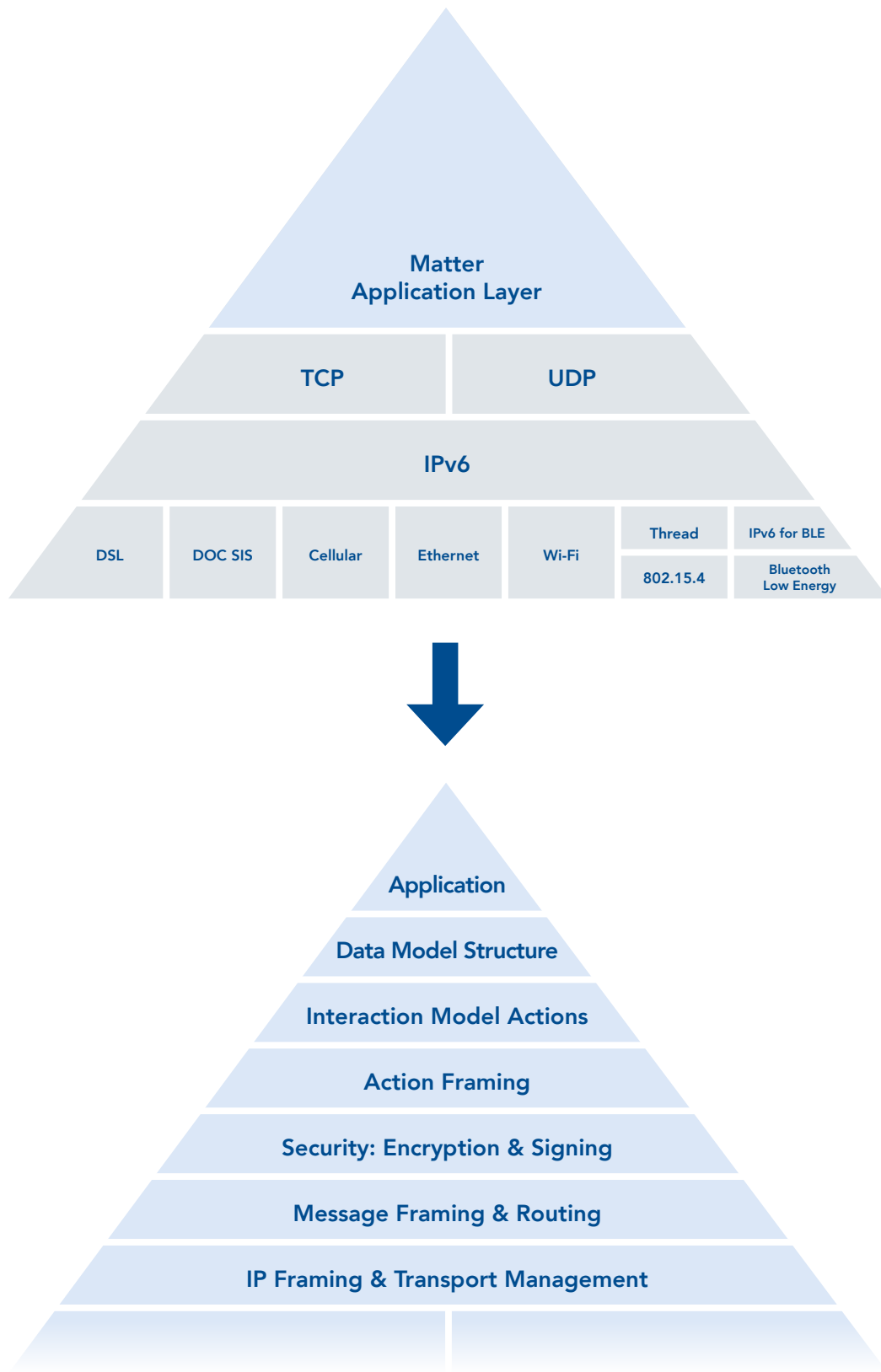


Figure 5.27. Matter for unifying the network layer and simplifying the application layer

Matter provides both compatibility and interoperability to devices for the smart home, so it's easier to design, deploy and manage these devices (see Figure 5.29). A key capability of Matter is multi-admin, which enables users to connect Matter devices to multiple apps and ecosystems simultaneously, locally and securely. Matter is creating a platform, supported by ecosystem-agnostic technology, that lets smart home devices communicate locally with each other on any IP network using a common language. The platform draws on the best of today's market-proven protocols and ecosystems, including Apple HomeKit, Google Weave and the Amazon ecosystem, which uses the Zigbee Cluster Library (ZCL).

Matter uses IP to create a dedicated application layer for smart home technology that provides methods for device addressing, routing, host-to host communication and other mechanisms for transporting data. Standardizing on IP significantly reduces the need for translation because it enables devices to communicate with each other without having to repackage data or packets. Standardizing on IP in the application layer also makes it easier to address important issues like data security and safe provisioning of devices onto the network. Choosing IP-based connectivity lets smart home technology standardize on the subcomponents of the network layer, including TCP/UDP for the transport layer, which are familiar protocols of internet-based communication. Matter will begin with support for Thread, Ethernet and Wi-Fi for communications and Bluetooth Low Energy for provisioning.



THREAD

Thread is an open standard for reliable, cost-effective, low-power, wireless device-to-device (D2D), device-to-mobile and device-to-cloud mesh communication. It is designed specifically for connected home and building applications where IP-based networking is desired and a variety of application layers, such as HomeKit and Matter, can be used on the stack. Thread is an IP-addressable protocol that also supports cloud access and basic Advanced Encryption Standard (AES) security. Thread offers proprietary implementations and an open-source, BSD-licensed version (OpenThread).

The key characteristics of the Thread connectivity protocol include:

- **Simple network installation, startup and operation** — The simple protocols for forming, joining and maintaining Thread networks allow systems to self-configure and fix routing problems as they occur. In addition, as an IP-based protocol, Thread eliminates the need for dedicated proprietary hubs or translators which reduces infrastructure investment, complexity and maintenance costs. Instead, any powered device that implements the Thread Border Router role provides the connection from the Thread network to other networks, i.e. Wi-Fi.
- **Security** — Devices do not join the Thread network unless authorized and all communications are encrypted and secure. From a system-level, Thread’s IP foundation enables end-to-end security.
- **Scalable mesh networks** — Home networks range from several devices to hundreds of devices communicating seamlessly. The network layer is designed to optimize the network operation based on the expected use.
- **Range** — Typical devices in conjunction with mesh networking provide sufficient range to cover a normal home. Spread spectrum technology is used at the physical layer to provide good immunity to interference. Also, Thread networks implement a mesh topology so a Thread network extends its range as more powered devices are added to the network.
- **No single point of failure** — Thread is designed to provide secure and reliable operations even with the failure or loss of individual devices.
- **Low power** — Host devices can typically operate for several years on AA batteries using suitable duty cycles.
- **Responsive** - Low latency means faster response times for instant control and automation.

THREAD	STANDARD
Application Layer	RFC 768, RFC 6347, RFC 4279, RFC 4492, RFC 3315, RFC 5007
UDP + DTLS	
Distance Vector Routing	RFC 1058, RFC 2080
IPv6	RFC 4944, RFC 4862, RFC 6282, RFC 6775
6LowPAN	
IEEE 802.15.4 MAC (including MAC security)	IEEE 802.15.4 (2006)
Physical Radio (PHY)	

Figure 5.12. Thread protocol stack



The Thread standard is based on the IEEE 802.15.4 physical (PHY) and medium access control (MAC) layers operating at 250 kbps in the 2.4 GHz band (see Figure 5.12). The MAC layer, used for basic message handling and congestion control, includes a carrier sense multiple access (CSMA) mechanism so devices can listen for a clear channel. The MAC layer also contains a link layer to handle retries and acknowledgement of messages for reliable communications between adjacent devices.

In a system comprised of devices running the Thread stack, none of these devices represents a single point of failure. Though several devices in the system perform special functions, Thread is designed to self-heal by re-routing signals when needed so that there is no impact to the ongoing communication within the Thread network. Devices in the Thread stack support the IPv6 addressing architecture. Each device that joins the Thread network is assigned a 16-bit short address.

All Thread devices use a communication protocol called IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN), which targets low-power devices with limited processing capabilities. Complementary to Thread, Matter is another protocol standard that runs over 6LoWPAN to enable home automation. Techniques such as header compression are used in the Thread network, and devices transmitting messages compress the IPv6 header as much as possible to minimize the size of the transmitted packet.

The Thread stack supports mesh connectivity between all devices in the Thread network. The actual topology is based on the number of devices in the network. Mesh networks increase radio system reliability by allowing radios to forward messages for other radios. For example, if a node cannot send a message directly to another node, the mesh network forwards the message through one or more intermediary nodes. In the Thread network, all router nodes preserve routes and connectivity with each other so the mesh is constantly maintained and connected.

There are three key roles that are part of the Thread network topology.



Border Router provides functionality such as routing traffic and connecting to the cloud or other networks. There must be at least one Border Router in a Thread network.



Router is a powered device that forwards messages from it to other devices acting as a range extender. A network may have many routers.



End Device is a battery or line-powered device that consumes low energy and only sends and receives data; it has no routing capability. A network may have many end devices.

The Thread wireless network features a mesh topology composed of Border Router(s), Routers and End Devices. End devices can be battery-powered (sleepy end device) or line-powered (router eligible end device). To optimize for low power, sleeping devices poll parents for messages. They are not required to check-in which allows lower power operation. Parents hold messages for sleeping devices and will switch to another parent device if current connection to is lost.

In a mesh topology, the communication is enhanced by router nodes that implement routing protocols and maintain routes in the network. A Thread network automatically assigns one of the routers as the Leader which then manages network parameters, coordinates commissioners and makes network decisions. If a Leader fails, another Router will automatically be assigned the Leader role. A Leader can also promote Router Eligible End Devices to Routers to improve connectivity if required. Implementing the mesh in the way means that Thread networks have no single point of failure, will self-heal and reconfigure when a device is added or removed.

A unique aspect of Thread is the Border Router. The Border Router must be a powered device and enables Thread devices to be discovered and communicate with devices outside of the Thread network. Because Thread is based on IP, translator function/look-up tables are not needed. A Border Router can be built into existing devices from any product manufacturer thereby minimizing the need for additional dedicated hardware.

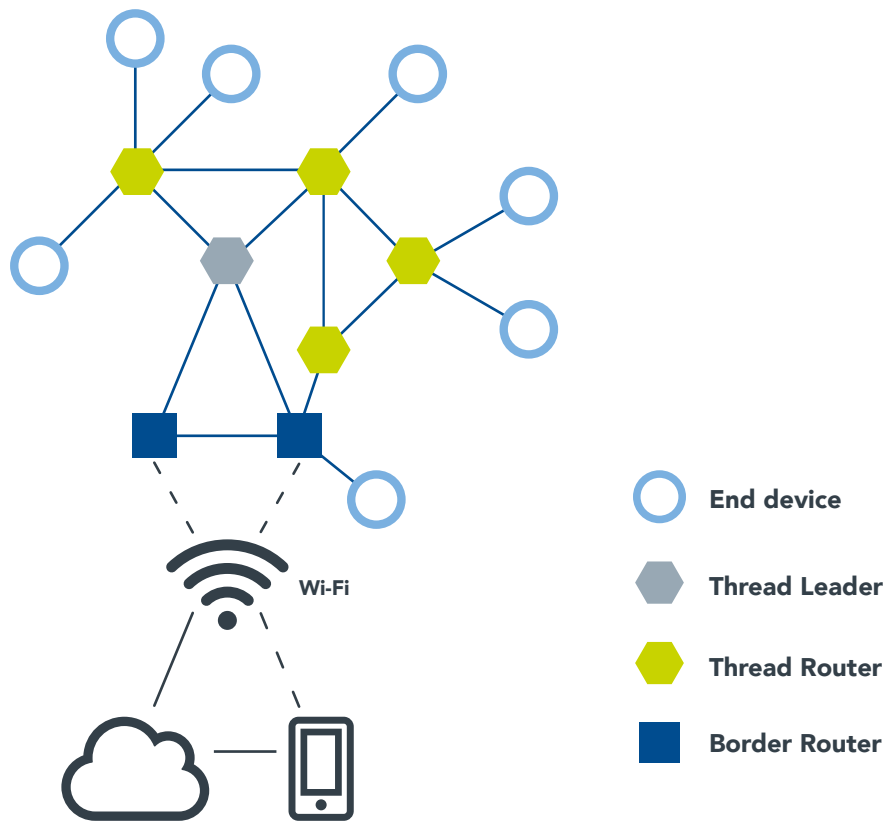


Figure 5.12B. A Thread wireless network.



ZIGBEE®

The Connectivity Standards Alliance (previously known as the Zigbee Alliance) developed a low-cost, low-power wireless communication standard that is somewhat simpler and less expensive than Bluetooth or Wi-Fi. This makes it suitable for applications such as wireless light switches, building automation, smoke detection, office and home energy monitoring, and other consumer and industrial applications for which short-range, low-data-rate transfer is acceptable.

Similar to Thread, Zigbee is built on the foundation of the IEEE 802.15.4 protocol: PHY and MAC layers. The PHY layer provides connectivity for various market requirements by defining frequency ranges in both the 868/915 MHz band (for the European region, the United States or Australia) and the universal 2.4 GHz band. The MAC layer controls access to the radio channel.

A wireless network comprises a set of nodes that can communicate with each other through radio transmissions according to a set of routing rules for passing messages between nodes. A Zigbee wireless network includes the following three types of nodes (see Figure 5.13):



Coordinator

This first established node is responsible for forming the network by allowing other nodes to join the network through it. Once the network is established, the coordinator's routing role is to relay messages from one node to another and to send/receive data. Every network must have one — and only one — coordinator.



Router

This node features routing capability and sends/receives data. It also allows other nodes to join the network through it, so it helps extend the network. A network may have many routers.



End Device

This node only sends and receives data; it has no routing capability. A network may have many end devices.

The Zigbee network layer features star and mesh topologies. In spoke-and-hub star topology, the Zigbee coordinator is responsible for initiating the network, maintaining the other nodes and providing authorization and authentication mechanisms when it runs the protocol function called Trust Center. In an interconnected mesh topology, the communication is enhanced by router nodes that implement routing protocols and maintain routes in the network. Zigbee's advanced routing feature adopts a mesh topology to allow self-healing routes between communicating devices, thus avoiding single points of failure and ensuring reliable packet delivery.

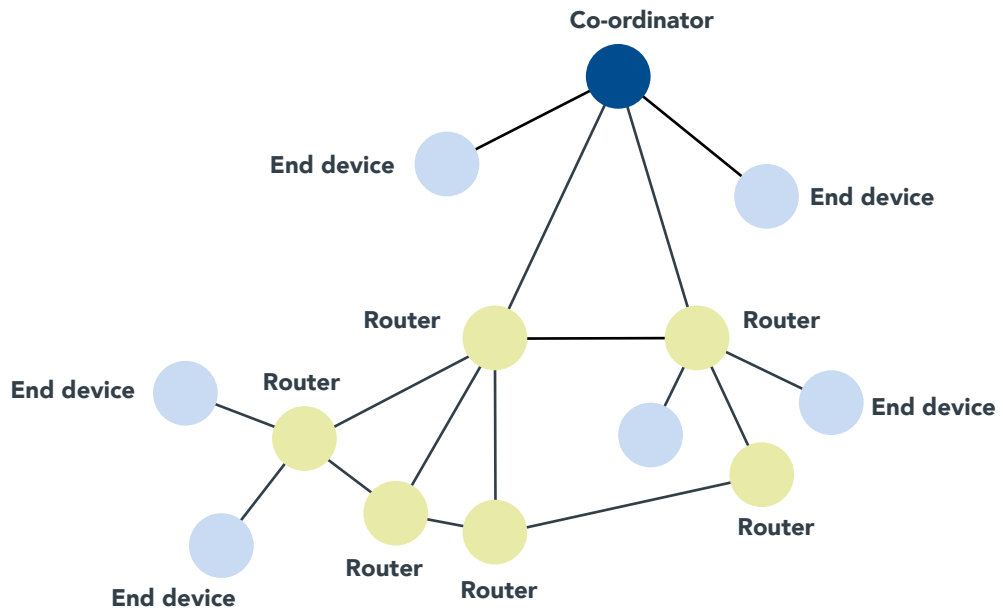


Figure 5.13. A Zigbee wireless network

To manage and determine the best routes, the Zigbee routers maintain routing tables and compute a path cost metric based on the asymmetric link costs associated with each route. The network layer also performs security functions. This includes 128-bit-wide AES encryption and application layer encryption services. These measures have been incorporated to prevent intrusion from potentially hostile parties and from neighboring Zigbee networks. Zigbee also provides privacy for communication between nodes of the same network.

A Zigbee node may have several applications running on it. For example, a node in a smart home network may incorporate an occupancy sensor and a light switch, each of which is an application. Access to application instances is provided through endpoints, which act as communication ports for the applications.

A data entity (e.g., temperature measurement) handled by a Zigbee endpoint is referred to as an attribute (see Figure 5.14). The application may communicate via a set of attributes. For example, a thermostat application may have attributes for temperature, minimum temperature, maximum temperature and tolerance. Zigbee applications use the concept of a "cluster" for communicating attribute values. A cluster comprises a set of related attributes combined with a set of commands to interact with the attributes, for example, commands for reading the attribute values. A cluster corresponds to a specific piece of functionality for a device application. The total functionality for the application is determined by the Zigbee device type that it implements and the clusters that the device type uses. Thus, clusters are the functional building blocks of devices. The output/client and input/server sides of a cluster are illustrated in Figure 5.14.



Figure 5.14. A Zigbee communication model

The basic Zigbee network stack is shown in Figure 5.15. In this figure, the Zigbee device object (ZDO) is the application running as “endpoint 0” within the application layer that controls the network layer and performs other important system functions including the following:

- Defines the type of network device — coordinator, router or end device
- Initializes the node to allow applications to be run
- Performs the device discovery and service discovery processes
- Implements the processes needed to allow a coordinator to create a network and to allow routers and end devices to join and leave a network
- Initiates and responds to binding requests (Binding is the relationship where two nodes that were found through service discovery to be compatible may be paired so their output is automatically routed only to the paired node. For example, a light switch may be paired with a particular light, and this light switch must switch on/off only the light that it is intended to control.)
- Provides security services that allow secure relationships to be established between applications
- Allows remote nodes to retrieve information from the node, such as routing and binding tables, and to perform remote management of the node, such as instructing it to leave the network

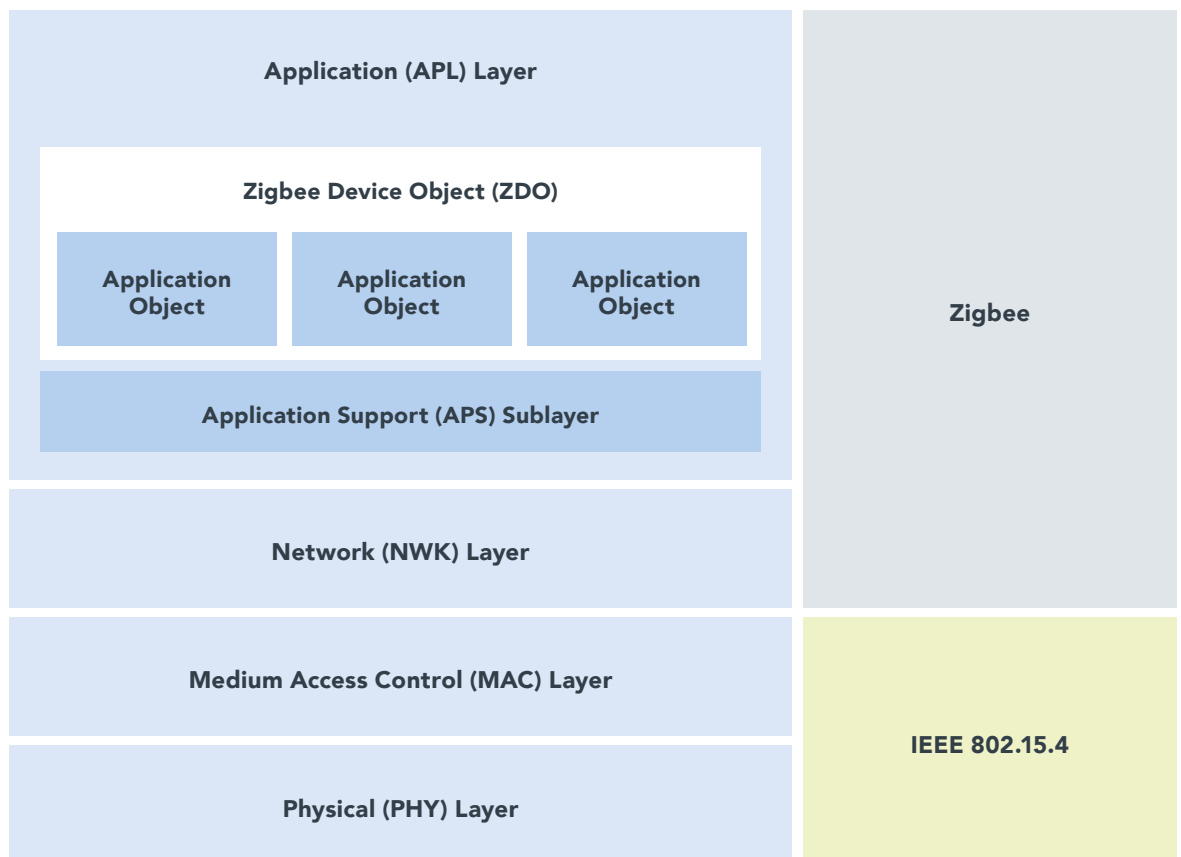


Figure 5.15 Zigbee network stack

SUMMARY

This chapter showed fundamental connectivity protocols for edge processing systems. As shown in Figure 5.28, these connectivity technologies provide network flexibility, ambient awareness, simplified provisioning and security. These attributes combine to contribute to overall edge processing intelligence.

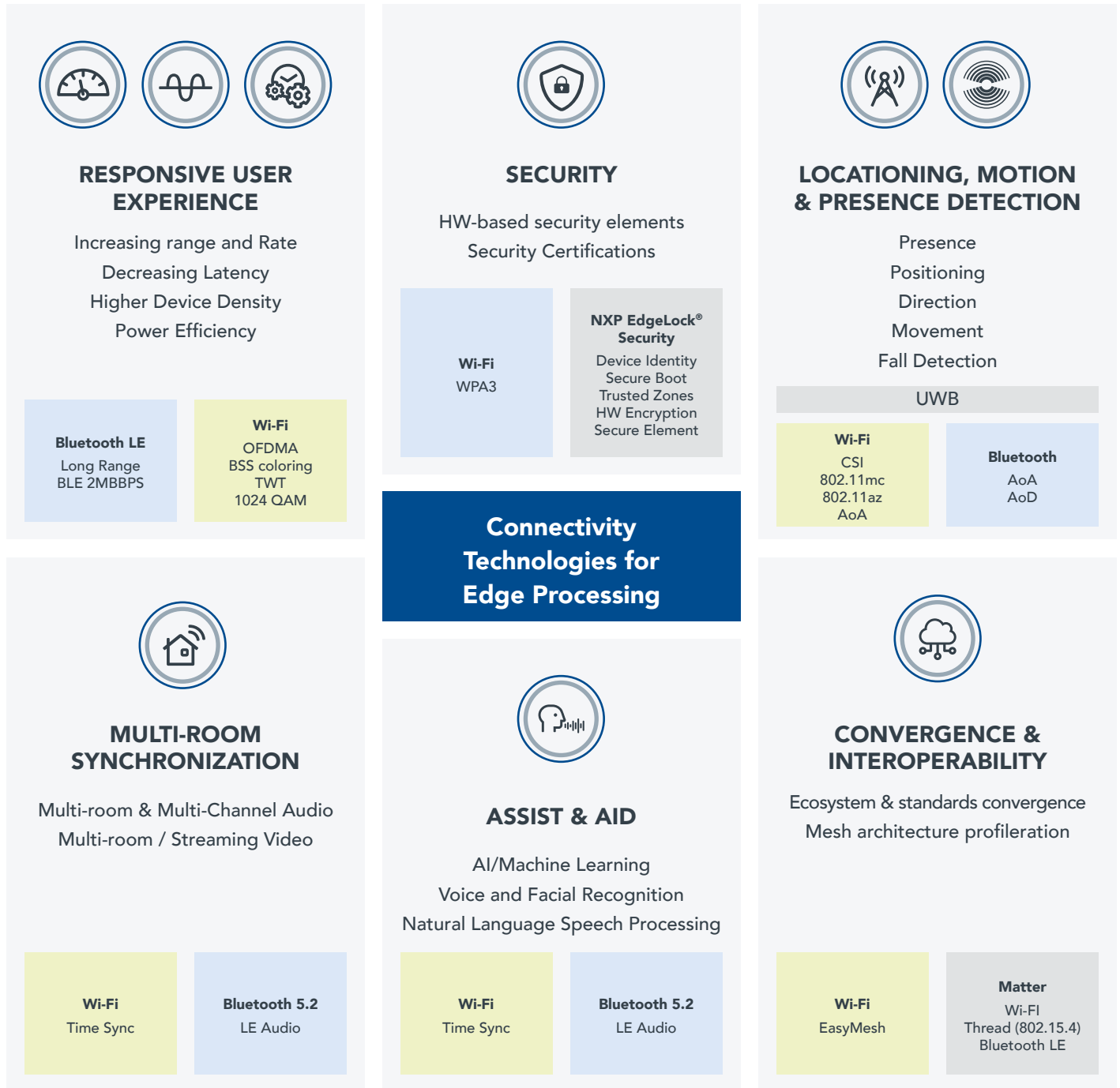


Figure 5.28. Connectivity technologies for edge processing

Chapter 6

EDGE COMPUTING DEVICE LIFE CYCLE MANAGEMENT

CONTRIBUTORS

Julien Delplancke, Product Manager, EdgeLock 2GO IoT Service Platform, NXP Semiconductors

Nicisor Penisoara, Senior Director, Security, Graphics & Machine Learning Software R&D, NXP Semiconductors

This chapter addresses why the life cycle concept is relevant for today's edge devices and how it's part of their manufacturing, deployment and management. Examining this topic from multiple perspectives (device user, OEM/device provider, cloud/services provider) provides a high-level definition for the life cycle of an edge device and a secure and efficient approach to implementing it.

LIFE CYCLE OVERVIEW

A straightforward definition for the life cycle of any device is the succession of phases the device goes through, from manufacturing and configuration to its use in the field and decommissioning. In each of these phases, specific procedures are followed and tools are used to provide the necessary product components and assets, appropriately configure specific access rights and device functions, and follow up on information generated by the device.

Life cycle management for edge devices

Not long ago, especially from an end-user perspective, the life cycle was basically nonexistent. A device (even with some firmware running) was purchased, used and then sold/disposed of without much concern from the user or the device OEM. The reality is that some users expect things to work the same way today, but that is no longer the case. If done correctly, device life cycle management can be mostly transparent and painless for the user and efficient to deploy for the OEM.

The roles of firmware and software are increasing in each device. They fundamentally define a product, the functionality it offers and the way a user interacts with it. Slick graphical user interfaces and software drivers for NAND memory have replaced mechanical buttons and tape heads. The firmware and software need updates, thus connectivity has been added (best practices advise implementing those updates over the air). Security is now a concern, services have moved to the cloud, legislators are now protecting consumers' rights to privacy and so on. The pinnacle of this evolution, shown in Figure 6.1, is something that we all carry in our pockets = the ubiquitous mobile phone.



Figure 6.1. The ubiquitous mobile phone

Edge devices are connected, and, in many cases, a significant part of their functionality is tied to services and capabilities hosted in the cloud. Especially for devices offering paid-for and value-added services, the **secure deployment**, configuration, management of services and association of the user identity are important. This is typically a critical and difficult-to-manage phase during product deployment. In certain cases, while the device is in use, service consumption should be tracked. The edge device must be maintained over its **in-use** lifetime through software upgrades, bug fixes and new features. Its deprecated features and services that the user should no longer access must be disabled.

Not having a solid and secure grip on deployed services and node identity can lead to not only lost revenue (pretty obvious) but also excess charges when the device-offered services are hosted by the OEM in a third-party cloud paid for by the OEM.



Quite a few edge devices store (even if for a limited amount of time and for only local processing) user information and/or environmental data that can be tied to the user. This leads to the need to protect this information and securely handle the "end of life" phase of the device. From a user perspective, that can be **decommissioning** or the transfer of ownership.

These life-cycle phases primarily represent the user perspective; however, additional steps need to be considered when looking at the complete process from an OEM perspective: manufacturing the device potentially in third-party facilities, **provisioning** OEM IP and assets and **onboarding** the device in service provider clouds. Extending this view to include the system on chip (SoC) supplier manufacturing the MCU or MPU that powers the device (and can be leveraged as the silicon root of trust for the device) completes the picture.

OEM and **service providers** need to comply with legal requirements that imply having and exercising control over the user data collected, its manipulation, its accessibility and, very importantly, the proper management over device decommissioning and change of ownership (i.e., erasing user data).

The European Union's General Data Protection Regulation (GDPR, 2018) imposes obligations on any entity collecting data related to EU citizens.

California's Privacy Rights Act (CPRA, 2020), building on the California Consumer Privacy Act (CCPA, 2018), imposes obligations on entities that meet certain criteria and process the personal data of California residents.

Identifying and describing phases with their associated attributes (access rights, available capabilities, procedures, etc.), processes and tools to transition between phases is a needed exercise for edge devices. This exercise is defining the edge device life cycle.

Life-cycle states and transitions

The life cycle of a device features a number of states, with the device provisioned and configured as expected in each state, and several transitions that involve trigger(s) and condition(s) to progress through these states.

The elements of the life cycle are defined as follows:

- **Provisioning** — The process of injecting assets (public and secret data, firmware, etc.) into the device and functionality configuration.
- **Attributes applying to data and functionality:**
 1. [N/A] Unprovisioned — When referring to data, this means that the respective asset is not yet provisioned into the device. When referring to functionality, this means that the respective functionality is not yet deployed/configured or is not usable in its full capability.
 2. [X] Access restricted — Access to the respective item is restricted permanently to specific entities or requires an authentication process. Access restrictions are applied in a specific manner to various items. Restrictions to the same item might vary with the life-cycle state.
 3. [Y] Item accessible — Data or capability is present and usable/accessible without specific restrictions.
- **Life-cycle state** — The combination of assets provisioned to the device, configuration of functionality enabled and associated access rights. The life-cycle state is typically retained in nonvolatile memory and enforced by hardware and firmware/software.
- **Transition** — The change of the device life-cycle state, resulting in the availability of a different combination of assets and functionality as well as access rights for various users of the device.

The edge node life cycle

The edge device life-cycle management can be described from different perspectives that focus on specific parts of the life cycle. This section first considers the OEM and user perspective and then adds the SoC view. Combining these perspectives generates suggestions for secure and efficient implementations. Though the purpose of this exercise is to provide a complete, realistic and applicable view, certain assumptions and simplifications are made and noted in the relevant places.

This exercise involves the following generic set of assets, functions and access rights found in life-cycle states:



Assets

1. OEM root of trust (RoT)
2. OEM keys and certificates
3. Device management ID
4. Device services ID
5. User ID
6. User data



Functions

1. Secure boot
2. Debug platform
3. Updates
4. Configure services
5. Decommission
6. Erase user

In a typical life cycle, these assets and functions are adapted to the needs of the specific edge device that is being built and its deployment targets (services, environment, usage and user type). This list generally works for most applications.

THE OEM AND USER PERSPECTIVE

Figure 6.2 captures the edge device life cycle from the OEM and user perspective. It begins with the device being manufactured, depicting the SoC that plays the role of the silicon RoT, and progresses through relevant states and transitions leading to the final stage when the user stops using the device.

The device management (DM) cloud and the services cloud are two key life-cycle entities. Most edge devices need some form of management by the OEM, if only for basic capabilities (firmware and software updates, remote configuration, etc.). This is implemented in a specific, OEM-controlled cloud called the DM cloud. The OEM-controlled cloud can offer additional services on top of those related to edge device control, but, from the device life-cycle perspective, delineating how the device exposes cloud-based services offered by third-party suppliers is important, and this is represented by the services cloud. Figure 6.2 depicts only one services cloud for simplicity, but deploying support for and managing multiple services clouds is common.

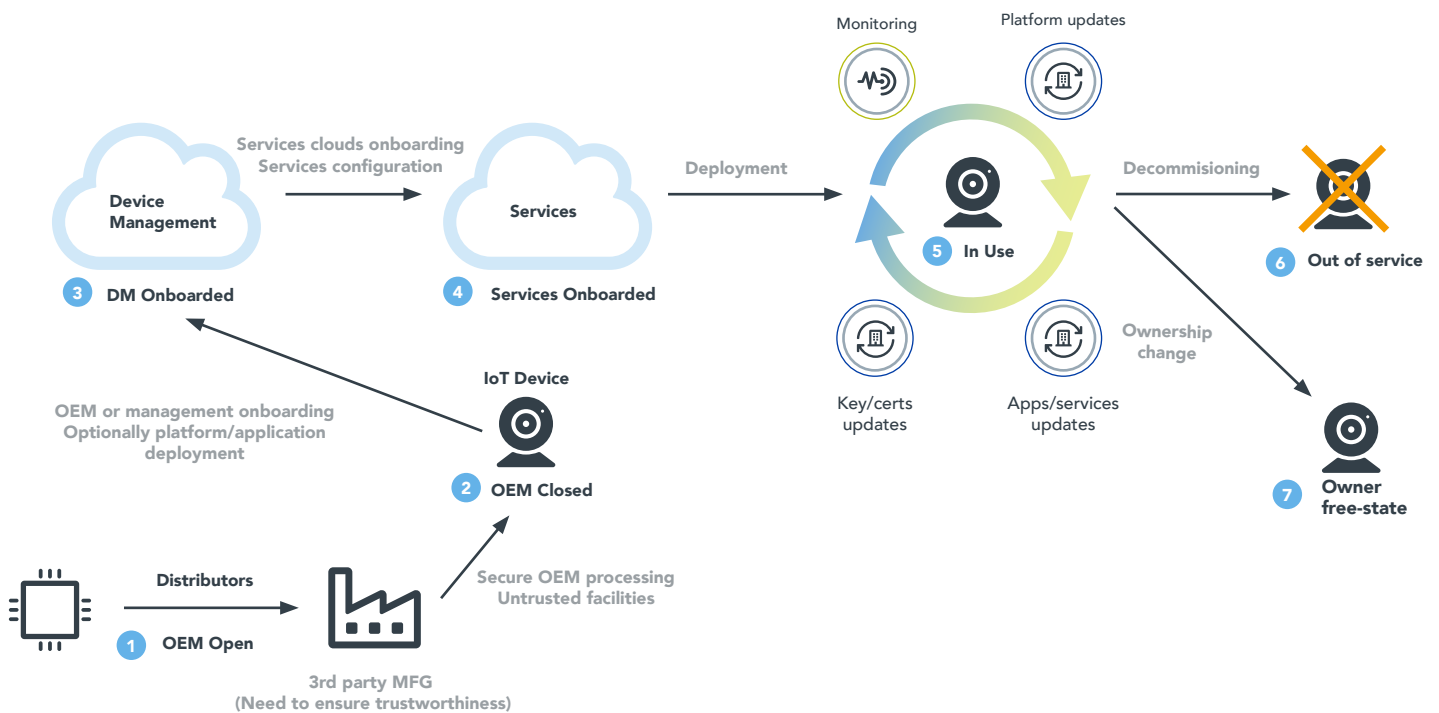


Figure 6.2. The edge device life cycle from manufacturing to decommissioning

OEM open state

This is the state in which the edge device is assembled; it is practically inherited from the SoC. In this state, any firmware/software can be loaded and run on the device. This is typically the state in which various tests are performed to ensure that the device, from a hardware perspective, functions correctly (memory integrity, interfaces, hardware performance tests, etc.). No OEM assets (public or secret) are deployed, but the SoC might contain SoC-specific assets, and these assets should be protected by the SoC itself without additional efforts from the OEM/manufacturing facility (more on this later). In this state, the device can be freely debugged/tested. OEM and potentially third parties and service providers develop firmware/software during this state.

From this state, the device transitions to the OEM closed state or is scrapped. Any sensitive assets deployed on the device are erased before scrapping the device.

Status	Asset	Status	Function
N/A	OEM RoT	N/A	Secure boot
N/A	OEM keys&certs	Y	Debug platform
N/A	Device Mgmt ID	N/A	Updates
N/A	Device Services ID	N/A	Configure services
N/A	User ID	N/A	Erase user
N/A	User data	N/A	Decommission

Table 6.1. Device summary in OEM open state

OEM closed state

This is the state in which the device is fully formed, the OEM RoT is installed and, typically, OEM baseline keys and certificates are finalized. The device also runs the OEM platform firmware/software, and the secure boot is enabled so the device boots only authorized (i.e., OEM signed) firmware/software. This firmware/software can be encrypted to protect any sensitive OEM (or third-party) IP. Debugging is restricted because it would likely expose OEM assets and IP. The debugging authentication allows only authorized users to debug the provisioned platform. Updates can be made at this point but typically not over the air because the device is not yet onboarded to the DM cloud. This means only local updates with signed images can be performed.

Status	Asset	Status	Function
X	OEM RoT	Y	Secure boot
X	OEM keys&certs	X	Debug platform
N/A	Device Mgmt ID	X	Updates
N/A	Device Services ID	N/A	Configure services
N/A	User ID	N/A	Erase user
N/A	User data	N/A	Decommission

Table 6.2. Device summary in an OEM closed state

The transition from the OEM open state to the OEM closed state involves the following steps:



Installing the OEM RoT

Copying the OEM public keys (or hash of the public keys) in the SoC one-time-programmable (OTP) memory that is used to authenticate the images started on the SoC.



Enabling secure boot

Configuring the SoC to boot only OEM-signed firmware/software and, optionally, enable encrypted boot (the deployment of the encryption key is necessary). This step also restricts debugging access, which can be permanently disabled.



Deploying OEM assets

Deploying OEM keys and certificates as well as OEM-signed (and potentially encrypted) firmware/software.

Because this transition is critical from the life-cycle management perspective, it needs to be performed with a certain level of security assurance, typically reflected in the following:

- SoC authentication is needed to verify the supplier, type and characteristics of the SoC. Ideally, this is implemented based on a die individual SoC certificate. Without a genuine SoC, the edge device that was built, provisioned and deployed doesn't instill much confidence.
- The installation of the OEM assets usually requires confidentiality. This applies to not only secret keys but also firmware/software that contains OEM or third-party IP. This can be ensured in many ways, but the typical approach is using a secure manufacturing location. However, this is an expensive approach and a business constraint because it limits the flexibility of working with (or having the option of changing easily) third-party manufacturing.
- The number of actual end devices typically needs to be controlled, especially when using third-party manufacturing.

When the device reaches the OEM closed state, it is ready to be onboarded to the DM cloud.

DM onboarded state

In this state, the fully formed device, running authentic OEM firmware/software, is onboarded to the DM cloud controlled by the OEM. The device is assigned a DM identity (typically in the form of a certificate), and is able to establish a secure connection to the DM cloud, over which it can receive messages, commands and updates. The edge device can verify the authenticity of data received from the DM cloud (typically this means that the data can be traced to the OEM or root certification authority) before performing any required actions. The device also can send updates and reports to the DM cloud to provide the OEM with the desired visibility and control over the device.

Status	Asset	Status	Function
X	OEM RoT	Y	Secure boot
X	OEM keys&certs	X	Debug platform
Y	Device Mgmt ID	X	Updates
N/A	Device Services ID	N/A	Configure services
N/A	User ID	N/A	Erase user
N/A	User data	N/A	Decommission

Table 6.3. Device summary in DM onboarded state

The transition from the OEM closed state to the DM onboarded state is straightforward and contingent on the specific mechanisms and tools used, but a possible sequence includes the following steps:

- Connect the edge device to the DM cloud over an internet connection.
- The DM cloud and edge device establish a secure connection, based on the OEM assets already provisioned on the device, and an OEM (ideally device unique) identity is provisioned in the DM cloud.
- The DM cloud generates a device-unique identity, which is sent to the edge device over the secure connection.



Services onboarded state

As mentioned previously, many edge devices expose cloud-based services that are offered by third-party suppliers and hosted on non-OEM clouds. Though they can be configured prior to the end user receiving the devices, third-party services also may be enabled and disabled on demand. Where a clear association between the OEM, end user and service provider is needed (maybe the user is paying separately for the service), the respective service(s) must be deployed after the end user is configured on the device. For simplicity, consider a case in which the edge device is onboarded before it is available to the end user.

In this state, the device is onboarded in the services cloud, and, consequently, it has a device service ID per the services cloud. The services are installed on the edge device, and one or more configuration interfaces are available.

Status	Asset	Status	Function
X	OEM RoT	Y	Secure boot
X	OEM keys&certs	X	Debug platform
Y	Device Mgmt ID	X	Updates
Y	Device Services ID	X	Configure services
N/A	User ID	N/A	Erase user
N/A	User data	N/A	Decommission

Table 6.4. Device summary of services onboarded state

The transition from the DM onboarded state to the services onboarded state for the case of configuring the services without an end-user association can be easily implemented by the DM management cloud, given that services clouds typically offer interfaces (representational state transfer, or REST, application programming interfaces (APIs)/resources) for device onboarding:

- The DM cloud establishes a secure connection to the services cloud.
- The DM cloud onboards the edge device to the services cloud and obtains the device services ID.
- The DM cloud establishes a secure connection to the edge device and injects the device services ID.
- The right access policies for the edge device are set in the services cloud.
- The right configuration for the service(s) offered by the services cloud is set on the edge device.

In-use state

This is the state in which the device spends most of its lifetime — in the end user’s hands, performing the function(s) the OEM intended the device to perform, through a combination of capabilities built in/offered by the OEM, and possibly services from third parties.

Edge devices typically have multiple physical users, but they generally have one registered user (think about a smart appliance). The fundamentals from the life-cycle management perspective are the same as the case in which the edge device has to manage multiple registered users with individual access policies, etc. User-relevant data is collected locally (per the device’s functional needs and configuration), stored/ processed and potentially sent to the various cloud-based services.

Status	Asset	Status	Function
X	OEM RoT	Y	Secure boot
X	OEM keys&certs	X	Debug platform
Y	Device Mgmt ID	X	Updates
Y	Device Services ID	X	Configure services
Y	User ID	X	Erase user
X	User data	X	Decommission

Table 6.5. Device summary of in-use state

From a management perspective, the typical operations that the edge device must support in this life cycle are:

- Monitoring its own health, which can range from very simple (checking that the device started correctly) to more complex (monitoring the available hardware resources, response time to the user, etc.) to highly sophisticated (using a trusted local entity to monitor the security status of the overall device). Reports are created and sent to the DM cloud on demand and/or regularly.
- Updating multiple firmware and software components typically through:
 1. **Platform firmware/software** — the OS, trusted execution environment (if one is used), boot firmware (not always updatable) and various other low-level firmware.
 2. **Applications/services** — ideally updatable individually; application containerization is recommended to support this.
- Updating keys and certificates, which again can range from very simple (OEM identity and secret keys, services identity and keys) to more advanced (different keys/certificates per type of operation, key revocation based on usage). The policies for keys and certificate updates are driven not only by OEM security policies but also, for certain applications, by market-specific practices and even lawful requirements.

The transition from the services onboarded state to the in-use state can be summarized as registering the user on the device. Though user registration can be implemented simply based on the physical possession of the device (and, for example, using a device-individual password provided with the device), the DM cloud can be used for edge devices that need additional security for this process. The below process is just one possible option:

- The user is registered to the OEM-controlled DM cloud, by its own means or by the reseller of the edge device, and receives user- and edge-device-specific credentials.
- The DM cloud establishes a secure connection to the edge device and injects the user credentials.
- The user can securely authenticate to the edge device to configure and start using the device.

From the in-use state, the device can be decommissioned to the out-of-service state, sold or returned to the OEM by moving into the owner-free state.

Out-of-service state

This is typically the terminal state for the edge device; the device is no longer usable and cannot be reinitialized to a state in which it performs its functions. It has no user, services cloud or OEM-relevant data assets. Ideally, nothing of value, or that can be used for nefarious purposes, is available on the device.

Status	Asset	Status	Function
N/A	OEM RoT	N/A	Secure boot
N/A	OEM keys&certs	N/A	Debug platform
N/A	Device Mgmt ID	N/A	Updates
N/A	Device Services ID	N/A	Configure services
N/A	User ID	N/A	Erase user

Table 6.6. Device summary of out of service state

The move from the in-use state to the out-of-service state can be implemented in several ways, depending on the degree of security needed for this operation. For simple devices that do not store valuable assets, this transition can be triggered directly through the local available user interfaces. For advanced edge devices that integrate cloud-based services and manage sensitive/valuable assets, the DM cloud typically coordinates the operation, so it needs to be appropriately secured to make sure that denial of service (DoS)/ransomware attacks are prevented. The DM cloud and appropriate support in the edge node system architecture can help here. A possible flow involves the following steps:

- The user, logged in to the DM cloud, can trigger the transition to the out-of-service state. Alternatively, the DM cloud can trigger the transition based on other driving factors/policies.
- The DM cloud establishes a secure connection to the edge device and sends the appropriate command(s).
- The edge node performs the authentication and appropriate verification of the request to transition, and then it proceeds to erase assets and render itself unusable.

The SoC itself can play a significant role in this transition and state. For example, the implementation of the OEM RoT support can be through secure OTP memory, so that once the RoT is erased, it cannot be replaced. The secure boot process can be hardware enforced so that once it's enabled, it cannot be disabled, and without a valid RoT, the SoC will not start.

This is a terminal state, the device cannot move to another state.

Owner-free state

This is the state in which the edge node, previously registered to a specific user, waits to be registered to a new user. Depending on how and when the cloud-based services are associated to the user, this state is equivalent to either:



DM onboarded

The services are user assigned/associated. In this case, the services configuration (at least) and association with the previous user are removed. For the new user, this association needs to be re-created, thus the device transitions again from DM onboarded to services onboarded. Then it is registered to the user.



Services onboarded

The services are not user specific. In this case, services and their configuration can be preserved; just the previous user and that user's data needs to be removed. The new user needs to be registered for the device in the DM cloud and, if needed, the device is transmitted to the services clouds before moving into the in-use state.

The above equivalence of states is represented in Figure 6.3.

Again, the DM cloud plays a key role. All these transitions need to be secured, triggered and operated through the DM cloud.

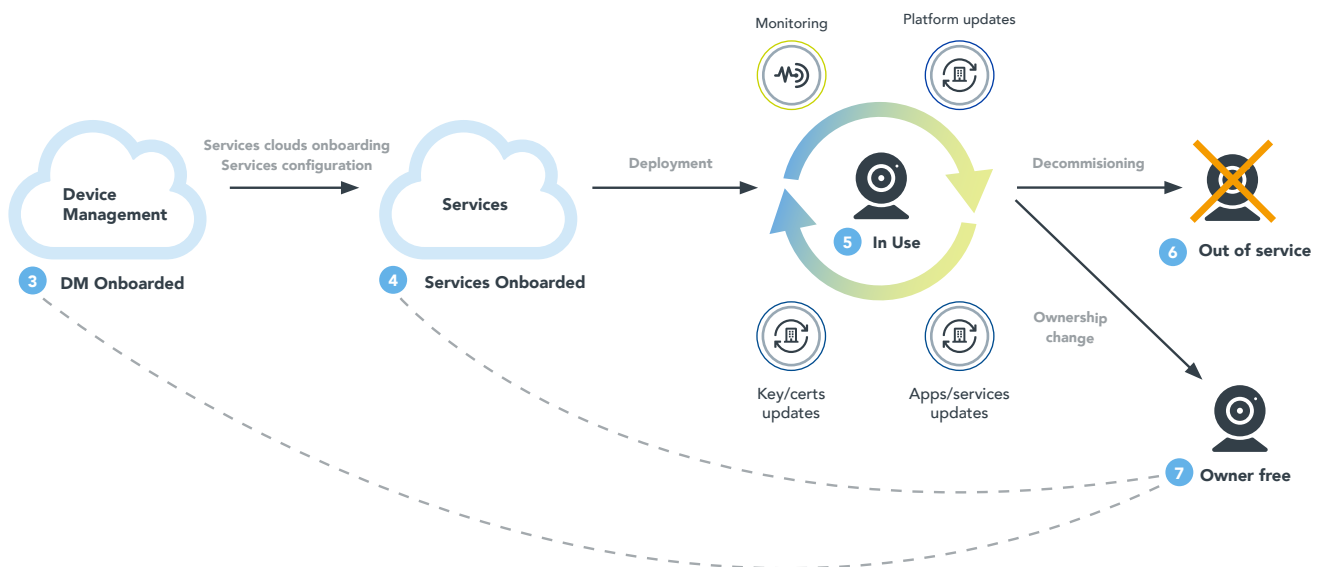


Figure 6.3. State equivalence for the owner-free state state

Vendor support for secure and efficient life-cycle management deployment

Critical security elements for the implementation and efficient deployment of the edge node life cycle include the following:

- The elements listed as representing the life-cycle state of the device need to be protected.
- The life-cycle state transitions need to be protected.
- Securely provisioning assets (and especially secret assets) can be expensive and/or can generate stickiness for specific manufacturing services providers. Ideally, this process should be cheap, and it should not generate significant dependencies on manufacturing.
- The onboarding of devices, if not automated, is tedious, error prone and expensive. Ideally, it should be a fully automated, “zero-touch” process.

The following sections describe vendor capabilities that can help with all of the above.

SoC life cycle

When the fundamental aspects of device management have been covered, an actual vendor and its products support implementing the edge node life-cycle concepts.

The edge node uses an SoC to implement its functionality (typically a single main SoC), and it is in this SoC and memory managed by the SoC that the assets and functionality that need to be protected reside. Thus, the life-cycle concept at the edge node level is easiest to implement and (very importantly) secure on an SoC that also has its own life cycle. In reality, it is pretty much impossible to secure an edge node if the main SoC does not offer the necessary security support, and the SoC life cycle is a key element of that.

All recent SoCs from the vendor implement a security life cycle that offers a solid base for securing the edge node and its life cycle. A short overview of the typical life cycle offered by vendor SoCs is shown in Figure 6.4.

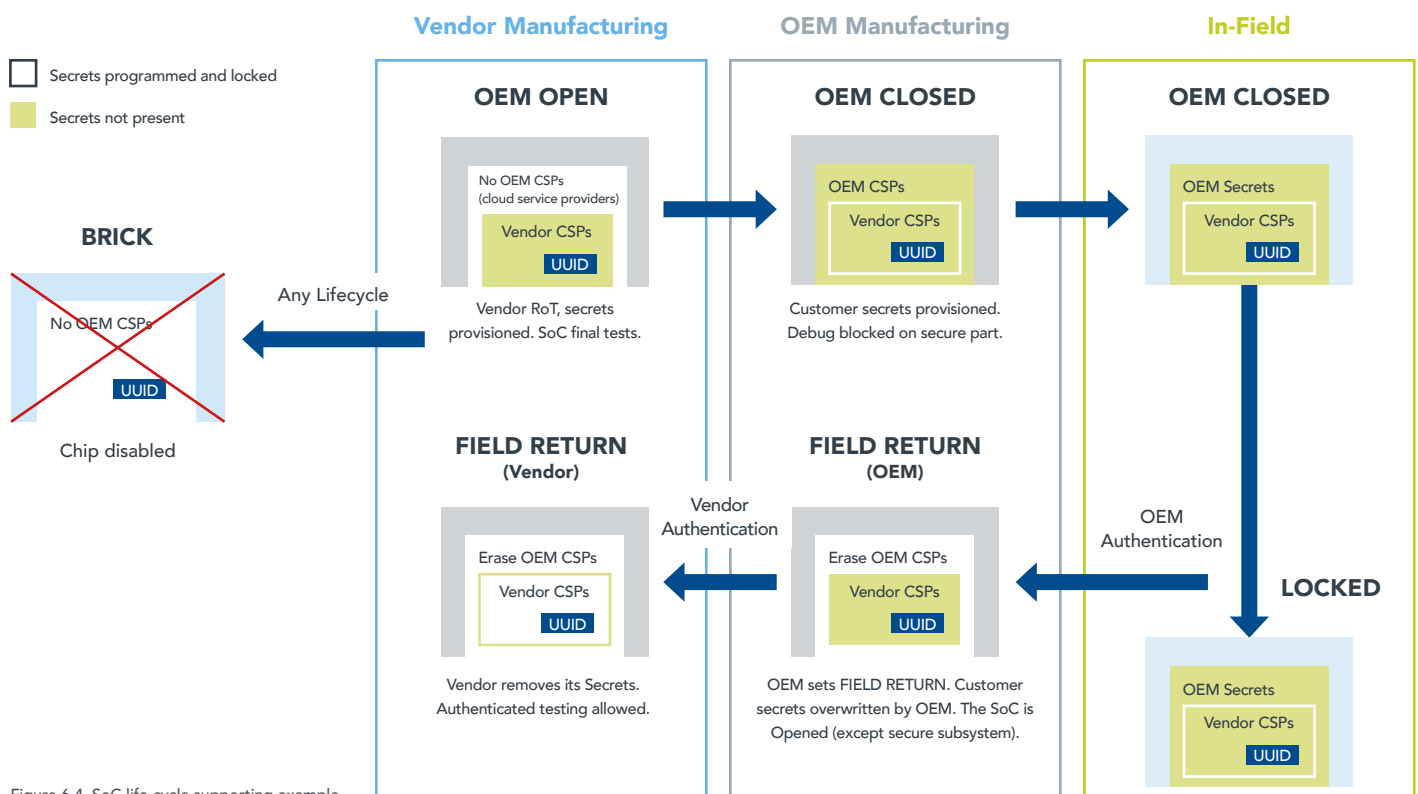


Figure 6.4. SoC life-cycle supporting example

From an OEM perspective, the SoC progresses from the OEM_OPEN state, where it contains only a vendor RoT and secrets, to the OEM_CLOSED state once the OEM RoT and secrets are provisioned. In OEM_CLOSED, the SoC boots only OEM-signed images and closes debugging interfaces by default; to open these, a successful authentication needs to be performed with the OEM as the signing entity.

The SoC needs to be in the OEM_CLOSED state while in the field. From there, it can be moved into the FIELD_RETURN_OEM state, which allows system debugging more freely (typically when quality issues need to be investigated). In this state, any final user data and OEM secrets have been erased, and the debugging is opened at the system level. Note that the security subsystem, which still contains vendor assets needed to ensure correct system behavior, cannot be debugged.

If the SoC itself needs to be investigated for potential issues, the device needs to be returned to the vendor, which can erase all assets from the security subsystem and open it for debugging. This state is called **FIELD_RETURN_VENDOR**.

Two other states can be used for the following purposes:

- **BRICKED** — As the name implies, all assets are erased, and the SoC is not usable anymore; it does not boot, and no investigations are possible. This state is used to make sure that a device is completely unusable for good.
- **OEM_LOCKED** — Once moved into this state from the OEM_CLOSED state, the SoC cannot be moved in any FIELD_RETURN_OEM state for investigations. The only state in which the SoC can be moved is BRICKED.

All SoC life-cycle transitions are secured. Commands to change the life cycle are authenticated, and based on the transition, the signing entity is the vendor or the OEM. Additionally, finer-grain capabilities in each life cycle fully support the needs of an edge node to secure itself and its final user data and implement its own life cycle.

Vendor trust provisioning and the SoC unique identity

One of the important assets that a vendor provides is a device's unique identity. Going beyond this, for some of a vendor's most recent products, each SoC has its own unique key pair, with the public key available in a vendor-signed certificate and the private key kept in the SoC security subsystem.

As described in the "Edge node life cycle" section, when the device using the SoC is manufactured, it is very important to establish that genuine silicon is used, and this is something that the vendor certificate allows the OEM to do through the following steps:



Verify the certificate to ensure that the unique SoC identity and public key belong to the expected vendor device.



Perform an authentication challenge with the SoC before it is installed on the device to ascertain that it owns the private key associated with the public key that is in the certification.

Ensuring that genuine silicon is used provides OEMs with the foundation they need to build the security of their products. Using counterfeit silicon runs the risk of adding security backdoors in the system, thus compromising the security of the final product.

Support for secure OEM provisioning in untrusted facilities

A critical step for the edge node is the OEM provisioning — injecting OEM assets, including certificates and potentially secret keys, in the OEM software with differentiating IP, etc. This is also the step that can be used to control manufacturing because a physically identical device without the OEM assets and software cannot perform its function. It doesn't have the same, or even any, value for the user. At minimum, the device should not be "recognized" by the OEM infrastructure and should not be enabled with the services and capabilities that a genuine device has.

Several methods can be used to implement the OEM provisioning. Traditionally, for security, the location of the provisioning needs to be "trusted". Whether that's on the manufacturing line or even after manufacturing is complete (i.e., at a trusted reseller), the trusted location typically requires the deployment of specific equipment (i.e., hardware security modules or HSMs) with dedicated tools, trained personnel and access control. This expensive method takes time to deploy and creates nonnegligible stickiness for an OEM to a manufacturing location, a provider of manufacturing services or a reseller.

The vendor can help make the secure OEM provisioning possible in untrusted facilities, which reduces cost, minimizes the time to deploy this capability and, when third-party manufacturing is used, eliminates the "stickiness" to manufacturing services suppliers.

The process involves the following capabilities:

- The vendor's advanced security subsystems found in its recent products, with trust provisioning support
- The die individual, vendor-signed certificates available for select products
- The EdgeLock® 2GO service platform

EdgeLock 2GO is an online cloud service specifically designed for zero-touch, secure deployment and management of IoT devices. It is an example of technology that helps implement critical stages in the edge device life cycle when the right hardware support is built into the platform. Though first used with edge devices to integrate a secure element, EdgeLock 2GO is being extended to work with devices that have the suitable hardware support (such as the SoC life cycle presented in Figure 6.4).

A **secure element** is a stand-alone integrated circuit designed to store and protect sensitive information including various keys and certificates as well as personal data such as fingerprints and facial biometrics. It also can be used to perform various cryptographic operations with the contained assets. It is primarily used in systems that need a certain (high) level of security/protection for these assets. This high level cannot be achieved with the other system capabilities (the main SoC of the device, for example).

The following outlines the process for OEM provisioning.

Preconditions

1. Through the vendor provisioning process, the EdgeLock 2GO service receives the die individual certificate for every supported part that the vendor ships. It can authenticate messages/data from the vendor's SoCs.
2. Through the vendor RoT in the security subsystem, every SoC can authenticate the EdgeLock 2GO service.
3. The OEM can use the EdgeLock 2GO service after providing and configuring the relevant OEM provisioning elements.

Example of a provisioning process with NXP's EdgeLock 2GO platform

1. After fully forming (i.e., memories and interfaces available), the device runs bootstrap firmware in any environment and connects to the EdgeLock 2GO service.
2. The vendor SoC and the EdgeLock 2GO service can authenticate each other and establish a secure connection. Data and commands exchanged are encrypted in a device-unique manner, so the surrounding environment cannot extract anything exchanged over this connection.
3. The device sends to EdgeLock 2GO an OEM identifier, typically called the OEM claim code that works with several other elements to indicate which OEM this device belongs to and which assets should be provisioned. Note: The additional elements that secure this step are too detailed for the purposes of this document.
4. The OEM RoT is inserted, and the device is "locked" to the OEM RoT (i.e., forward looking, it boots only OEM-signed software, but it can be adjusted to boot OEM-encrypted software as well).
5. The rest of the OEM provisioning elements are injected in the device.

Figure 6.5 shows this process. The OEM RoT and assets configured in the EdgeLock 2GO service are represented as state 0.

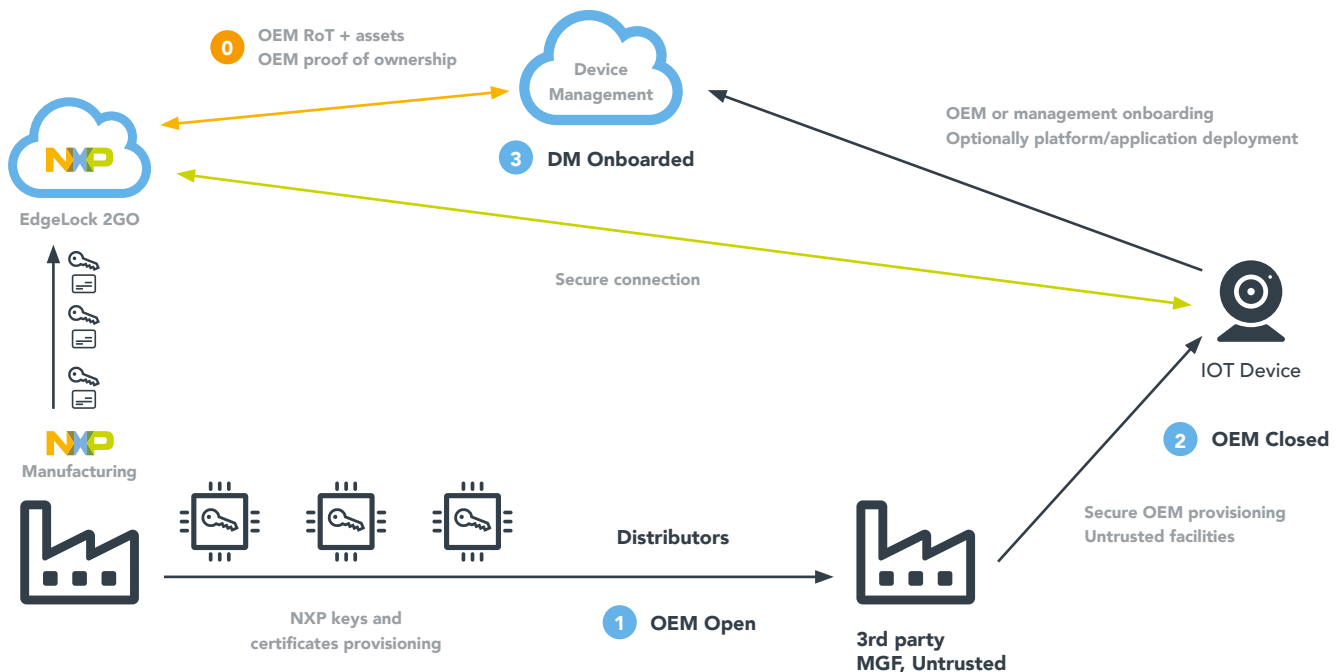


Figure 6.5. OEM secure provisioning in untrusted facilities leveraging EdgeLock 2GO services

Note that the process described here is one example and the actual process can be tailored to the OEM requirements. In particular, the keys and certificates can be injected into devices in a nonconnected environment (i.e., even if the device has no direct connection to the internet).

At the end of the process, the device is fully OEM provisioned, with no specific trust or security measures required for the location of provisioning. It can be implemented on the manufacturing line, at a reseller or by a technician installing the device. It also can be implemented when the final user first starts the device. The only requirement is an internet connection, though other solutions for manufacturing lines that don't offer internet connections can be applied with the needed quality/uptime, but these solutions are too detailed for the purposes of this book.

Chapter 7

EDGE COMPUTING ENERGY EFFICIENCY AND OPTIMIZATION

CONTRIBUTORS

Mohit Arora, Director, Low-Power & Security Architecture, NXP Semiconductors

Nicolas Collonville, Technical Lead, Connectivity Stack Integration, NXP Semiconductors

Mathieu Imbault, Manager, Edge Connectivity Software, NXP Semiconductors

Saleem Kala-Janssen, Director, Systems and Applications for Edge Processing, NXP Semiconductors

Nihaar Mahatme, Technology Strategist, Edge Processing, NXP Semiconductors

Pascal Mareau, Technical Leader, Power Technology Engineering, NXP Semiconductors

This chapter examines the processes and tools to achieve efficient power usage for edge processing applications. Learn how to benefit from low-power implementations, power estimators and power monitors.

BASICS OF LOW POWER OPTIMIZATION

When discussing low power optimization, there are two components that need to be considered;

- active power
- static power

Power consumption in CMOS circuits including MCUs and MPUs, is described as the sum of these two components;

$$P_{\text{total}} = P_{\text{active}} + P_{\text{static}}$$

Active (Dynamic) power refers to what is consumed when doing work. The goal is to minimize this part. Active power is defined as;

$$P_{\text{active}} = CV^2fN$$

Where;

- **C** = circuit capacitance
- **f** = frequency
- **V** = voltage
- **N** = number of nodes
- **I_q** = quiescent current from transistor leakage

We can reduce active power by lowering frequency and voltage and turning off hardware units when not in use.

We can reduce static power using low leakage process technology and transistors. For example, a smart phone uses active power when talking or playing a video. During standby mode, when the smart phone is not used, the power consumption is static.

$$C = Q / V$$

$$W = QV$$

$$W = CV^2$$

Capacitance (C) is the ability of a circuit to store energy. Work is the act of pushing something (a charge Q) across a "distance"—in electrostatic terms pushing Q from 0 to a voltage (V).

Standby static power is essentially the leakage when not switching. This is hard to avoid completely so the goal is to reduce as much as possible. Static power is defined as;

$$P_{\text{static}} \sim V \times I_q$$

Power is work over time, or how many times a second the circuit is oscillated.

$$P = W F \text{ where } F = 1/T \text{ (time)}$$

And with a little substitution;

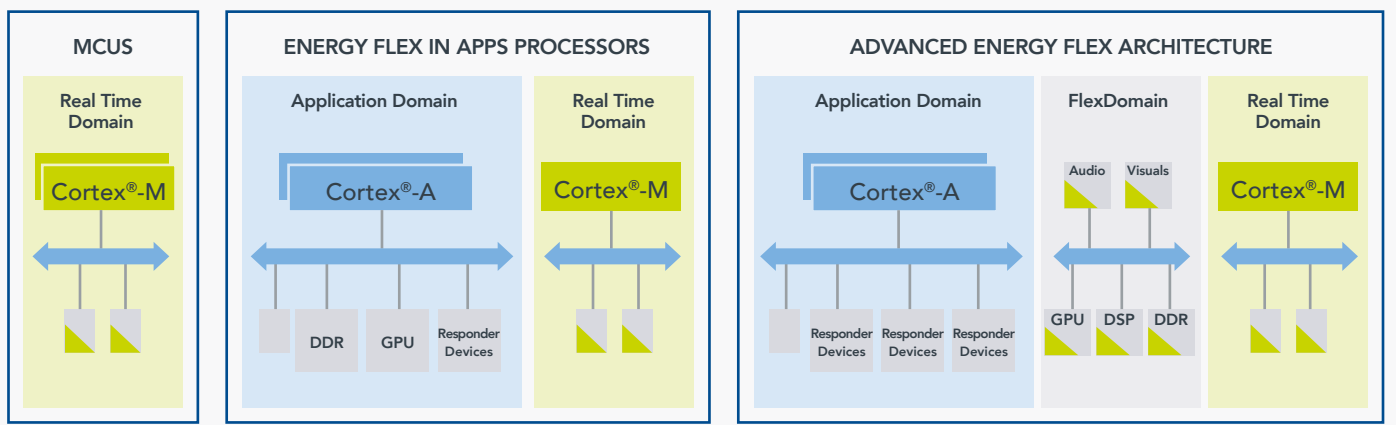
$$\text{Power} = CV^{2f}$$

So the goal when optimizing for low power is to only use the parts of the SoC that are needed and nothing else (reduce C), reduce the voltage if possible (V) and reduce the frequency if possible (F). This can be accomplished using hardware as well as software optimization.

An example of this is shown in Figure 7.1. NXP's EnergyFlex architecture shown here can be used to control multiple power domains in applications processors. For example, in NXP's i.MX 8ULP SoC, Energy Flex architecture features:

- Real time domain
- Application domain
- Flex domain

In edge applications requiring heterogeneous processing, where different kinds of applications need to be run, the chip can be effectively architected to achieve fine-grain power partitioning to reduce power at the system level.



INCREASING INTEGRATION, PERFORMANCE, FUNCTIONALITY AND ENERGY MANAGEMENT



Figure 7.1. Multiple Power Domains in iMX8ULP SoC

An application of this architecture is shown in Figure 7.2 for a smart watch device. In this example the three power domains are controlling three different parts of the application;

- **Application domain;** microprocessor architecture containing Arm® Cortex®-A cores running a rich operating system to provide advanced processing.
- **Flex domain;** allowing either application domain or real time domain access to display, low-power double data rate (LPDDR), or any other high-performance resources such as GPUs and DSPs.
- **Real time domain;** enables lower processing modes based on Arm Cortex-M core, giving significant battery savings. Can be used in conjunction with flex domain for displaying simple watch face (non-3D) or other smaller workload.

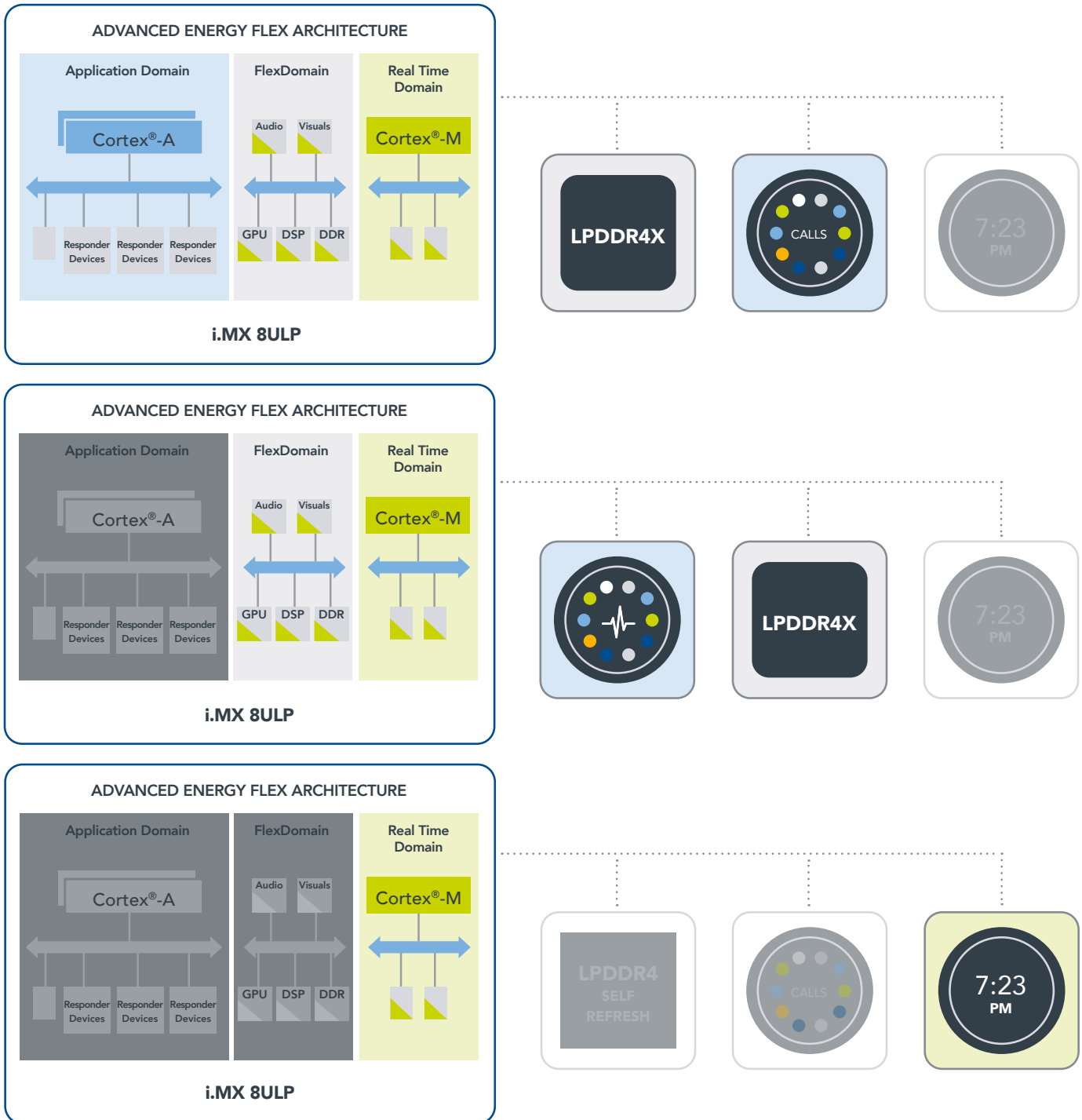


Figure 7.2. Power domains mapped to different forms of application processing

Figure 7.3 shows how this dynamic power domain control reduces power consumption over time by smartly turning on/off the various power domains based on the application processing profile. Lower energy consumption leads to longer battery life.

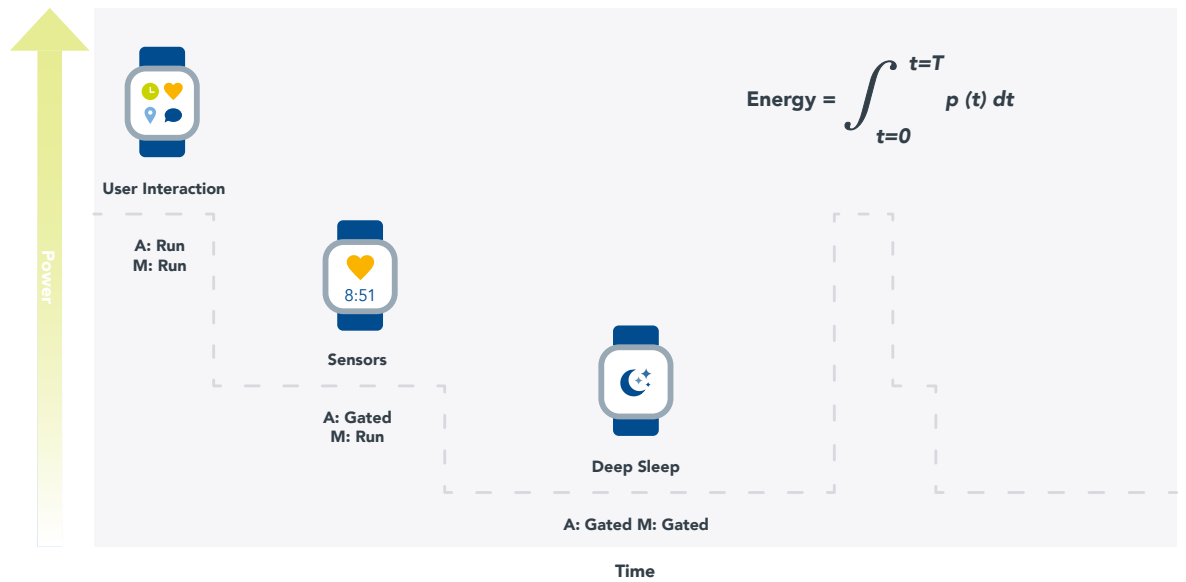


Figure 7.3. Power reduction due to dynamic power domain control in a smart watch application



Power efficiency for IoT applications

As discussed previously, energy efficiency relates to not only battery-powered devices but also line-powered devices. Indeed, energy is an expensive commodity that must be optimized to the lowest cost possible. During the planning of an internet of things (IoT) application, the power efficiency at multiple levels should be considered. This includes the printed circuit board (PCB) design, radio communication protocol, memory management, sleep and low power modes, wake-up times, compute power and speed of execution.

In a battery-powered device such as an insulin pump, the wireless microcontroller manages the communication with the mobile phone. The Bluetooth Low Energy (LE) protocol is commonly used in this type of application because of its interoperability with mobile phones and its low-power capabilities. Information such as the time of use, dose and pump mechanical data can be stored in the wireless microcontroller and transferred to the mobile phone when the latter is in the radio range of the device.

To connect with a mobile phone, the insulin pump needs to advertise to be discovered by the mobile phone (see Figure 7.4.)

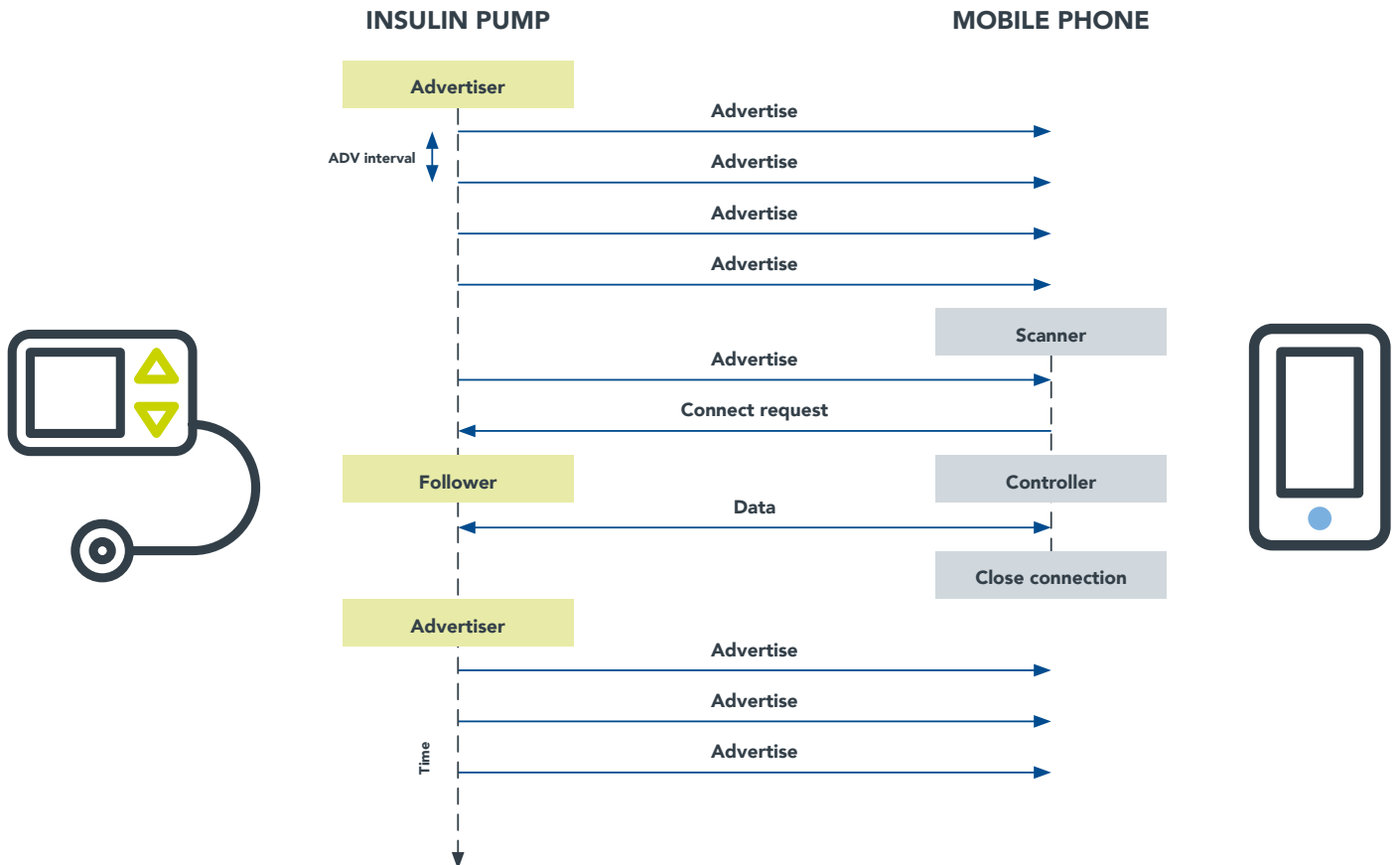


Figure 7.4. Bluetooth LE connection procedure

The wireless microcontroller sits in advertising mode for 95% of its lifetime. Consequently, the power consumption should be optimized during this mode.

The user should consider an advertising strategy thoroughly from both the system and application perspectives because the device does not need to be connected all the time. A flexible advertising interval can be established, and the device can be run in a low-power mode between advertising events.

Additionally, because the scanner (phone) and advertiser (insulin pump) are in close proximity, the output power of the transmitter can be reduced to save current consumption.

Finally, a device like this usually has shelf life before put to use, and it should consume the least possible current during this time. Therefore, ensuring the device is running in very low-power modes during which it is fully powered off and the consumption is lower than sub nA helps minimize power use.

In this next example, consider an industrial edge application for an Ethernet to Wi-Fi bridge. The final application is implemented in a harsh environment where the ambient temperature is above 60 °C. The power consumption of the device must be as low as possible to avoid running at the maximum junction temperature and impacting the reliability of the part. This compromise is therefore different from the previous example. Indeed, the focus is on not only thermal dissipation and the PCB design but also the absolute current consumption, which can be reduced by scaling down the CPU clock frequency, peripheral clocks or the core voltage.

LOW-POWER IMPLEMENTATION CONSIDERATIONS

As discussed previously, the overall energy consumed by an IoT device depends on its power consumption in active mode and low-power mode.

Figure 7.5 shows a typical energy profile of a device, including extended periods of low-power mode with periodic wake-ups and relatively shorter active mode periods.

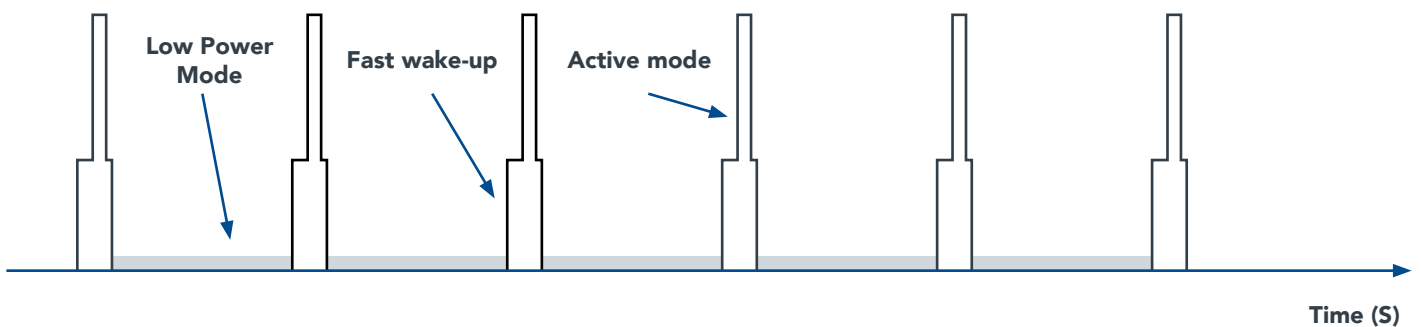


Figure 7.5. Energy profile for edge device with periodic wake-up

Active mode features high-power consumption; therefore, the amount of time the device is in this mode should be reduced. The time to wake up the system should be as short as possible, but it depends on the low-power mode from which the device exits. Obviously, the deepest low-power mode entails the longest wake-up time and, consequently, the longest active time. Therefore, depending on the use case, the right compromise among low-power mode, random access memory (RAM) retention and active time needs to be determined.

The device in Figure 7.5 remains in low-power mode for the longest time, so the power consumption is extremely limited. In this configuration, usually all the clocks are gated, but the supply domains are also power gated.

Further, to limit power consumption, only the necessary RAM content should be retained and RAM banks that are not useful should be power gated because embedded memories contribute significantly to leakage during standby modes when all other logic is shut down.



ACTIVE MODE

Active mode corresponds to a state during which the device has some activity to complete. In active mode, the active (dynamic) power is predominant compared to the static power. Therefore, optimizations should mainly focus on reducing the time spent in this period (profile) and the active power. The next sections will look at the active profile and engineering techniques to optimize the active power consumption.

Active profile

The Bluetooth LE advertising event in Figure 7.6 is an example of an active period.

The important parameter to consider when designing a low-power application is the area under the curve, which presents the total energy consumed by this event. Then it is imperative to reduce both the timing and the absolute current consumption.

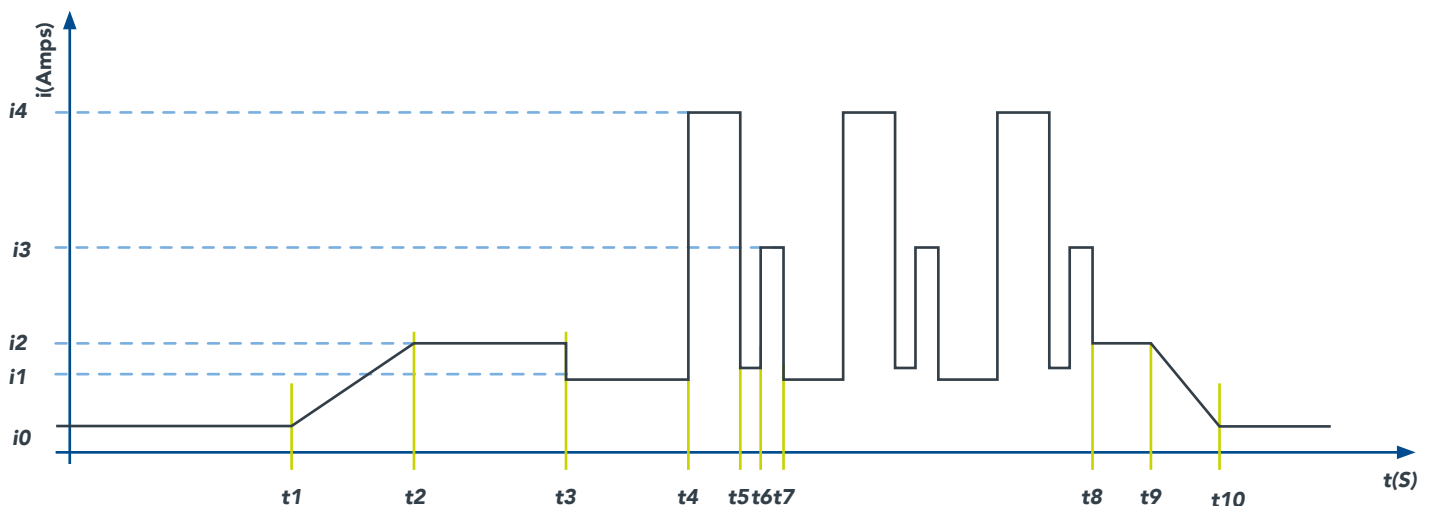


Figure 7.6. Energy profile of a Bluetooth LE advertising event

This active event includes the following phases:

- **$t_1 \geq t_2$** - Wake-up period where re-initialization begins.
- **$t_2 \geq t_3$** - The wake-up event is processed and a radio activity is scheduled. Note that radio activity is usually scheduled over a predefined period.
- **@ t_3** - The processor enters a low-power state usually called sleep mode. The processor and peripherals are clock-gated to reduce power consumption while the radio is active. In this state, the processor executes the wait for interrupt (WFI) instruction. Some RAM, system bus and flash can be clock gated to reduce current consumption. And the core voltage can be reduced if it is supported in the architecture of the wireless microcontroller.
- **$t_4 \geq t_5$** - Radio transmits.
- **$t_5 \geq t_6$** - The phase represents turnaround time.
- **$t_6 \geq t_7$** - Radio receives.
- **@ t_8** - The processor wakes up from sleep when radio activity completes on interrupts. It then optionally processes the received data (in connection mode, for instance).
- **@ t_9** - The processor selects the best low-power mode which is suited for the use case when no more activity is required. The processor then executes the low-power entry sequence for this mode.



Engineering techniques to optimize active power consumption

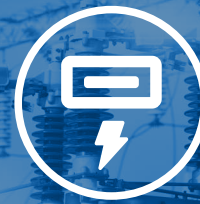
The dynamic current consumption is the main driving factor in this state. It depends on the operating clock frequency and the square of the voltage for a given load.

Frequency scaling



Using the lowest possible frequency during the active period is not necessarily the best option to save power because it can increase the time your device spends in active mode. If you do not need high-performance processing in active mode, you can lower the main clock frequency to a minimum without impacting the active time. However, during the wake-up sequence or the low-power entry sequence, increasing momentarily the clock frequency helps reduce the wake-up and low-power entry time, which reduces the total active time.

Voltage scaling



In addition to frequency scaling, some devices implement the voltage scaling in active mode. This can reduce the amount of energy consumed by the chip. However, reducing the voltage requires a lower clock frequency which could impact the time spent during active mode. Trade-offs between the voltage and the clock may be required to reach optimal power savings.

LOW-POWER MODE

Low-power mode corresponds to a state during which the device has no activity; therefore, the current consumption needs to be reduced to a minimum. In addition to frequency scaling, some devices implement the voltage scaling in active mode. This can reduce the amount of energy consumed by the chip. However, reducing the voltage running at lower clock frequency which could impact the time spent during active mode. Trade-offs between the voltage and the clock may be required to reach optimal power savings.

Low-power profile

The edge device implements a variety of clock domains, power domains and voltage domains, so the software can disable selectively the domains that are no longer required when activity ends.

Each power domain can be restored when the system needs to be activated; the restore time depends on the deepness of low-power mode. The deeper the low-power mode, the longer the wake-up time. For instance, light sleep low-power mode usually disables only a few clocks in the system, so the wake-up time is almost instantaneous. However, deep sleep low-power mode disables clocks, the main oscillator, the power domain and regulators, so the wake-up time is much longer.

Therefore, the software power manager of the application constantly needs to make trade-offs between the power savings in a deep sleep low-power mode versus wake-up requirements for the particular application and the extra power consumption.

Defining and understanding the use case

Consider the insulin pump example earlier in the chapter. Before patient use, the insulin pump can be stored at a drugstore. This is the shelf mode. The patient or end user then triggers the use mode during which the device periodically advertises to discover a Bluetooth phone for a connection. Once the phone connection is established, data is exchanged (for example, time, dose and error log). In this case, the end user takes a single insulin dose per day. Therefore, once the dose has been administered, the device can stay inactive for several hours until the next day. Timers can be used to keep track of time and the device can re-enter advertising mode. During this period the chip can be put into the lowest leakage power state.

For all these states, different power modes are used, as shown in Figure 7.7.

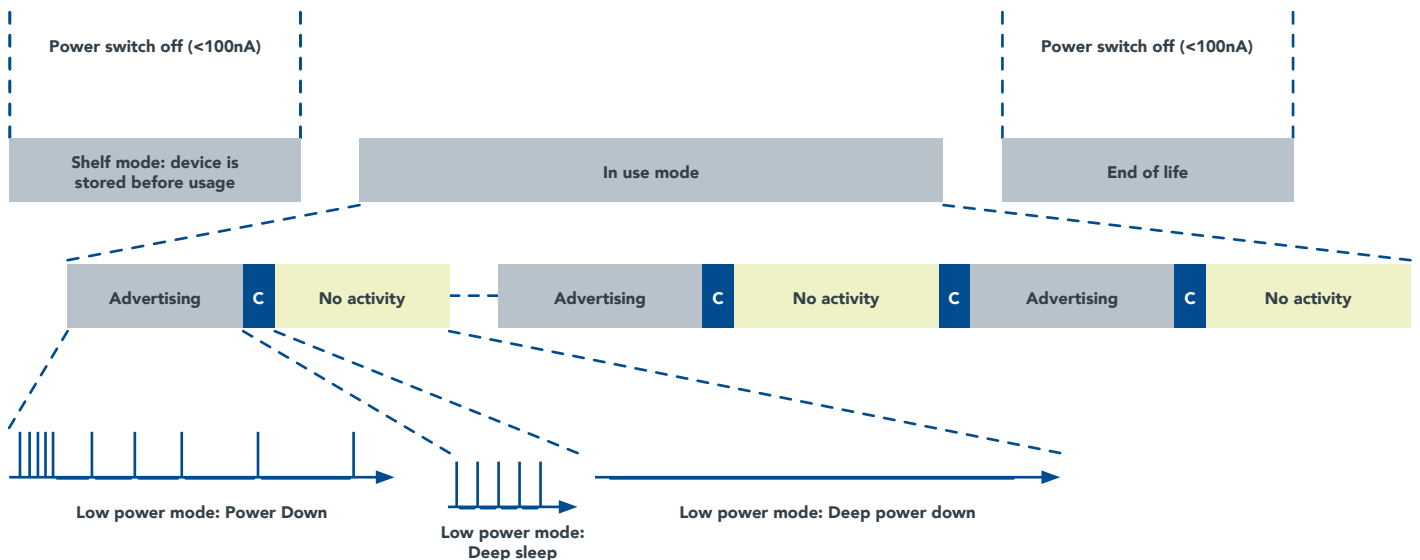


Figure 7.7. Power modes according to the usage of an IoT device

Engineering techniques to optimize low-power consumption

The static current consumption is the main driving factor in the low-power mode. It depends on several factors, including power domains and memory bank retention. Therefore, the low-power mode should be carefully selected depending on the use case and the application concerns. Some examples to consider for low-power mode selection are provided below.



Low-power time duration

The longer the low-power duration, the deeper the low-power mode can be. For short low-power periods, the extra activity required to wake up the system can cost more than the savings generated by a low-power mode. Remember, the critical parameter is the area under the curve. If the low-power duration is very short and the device does not have time to switch into low-power mode before the next active period, light sleep low-power mode should be selected instead.



Wake-up source capabilities

Another criterion in the selection of a low-power mode is its wake-up source capability. If a device needs to wake up from a particular event, then this wake-up event will be supported on the selected low-power mode. Most advanced devices provide several wake-up sources for all low-power modes. In this case, if a wake-up source capability is requested when going to low-power mode, the software source enable the required hardware resources to handle the wake-up detection. For instance, if the device needs to wake up when the battery voltage drops, the power domain containing the bandgap reference and the analog-to-digital converter (ADC) module should be kept on for the low-voltage detection.



Smart memory bank usage

On-chip volatile memories such as SRAM get smaller and faster with process technology scaling, but they also get leakier. So SRAM can be partitioned into banks so that the software can selectively retain some memory banks and power off some others. This depends on the amount of data that needs to be retained. An optimized software architecture helps reduce the amount of memory needed by providing a clear split between retained data and unused memory.



Warm boot: Acceleration of wake-up from core power-off low-power mode

When some power domains are turned off, the peripherals and clocks in that domain need to be reinitialized to be used by the software. The core power domain containing the processor and the interrupt controller also may be powered off in some low-power use cases. However, if the RAM is retained, the processor context can be restored and the controller can be interrupted quickly. This low-power exit sequence is called warm boot, and it features a very fast wake-up. Only some RAM banks need to be retained while the complete device, except for the always-on domain, is powered off.



Adjustment of low-power clock

If a time reference is required in low-power mode, for instance, to trigger timers for radio events or application events, then the 32 kHz crystal oscillator can still run albeit consuming a couple of hundreds of nanoamps. Optimization to reduce the power consumption could be foreseen by using an internal low-power free-running oscillator. However, the accuracy of the free-running oscillator usually very low, which can, for example, make a big impact by preventing leader-follower time synchronization in Bluetooth LE.

Other general low-power techniques

An application can be kept in a specific low-power mode, but the smarter move is to dynamically select the low-power mode depending on various constraints such as connection interval or a change of event. The software low-power policy manager usually handles this task.

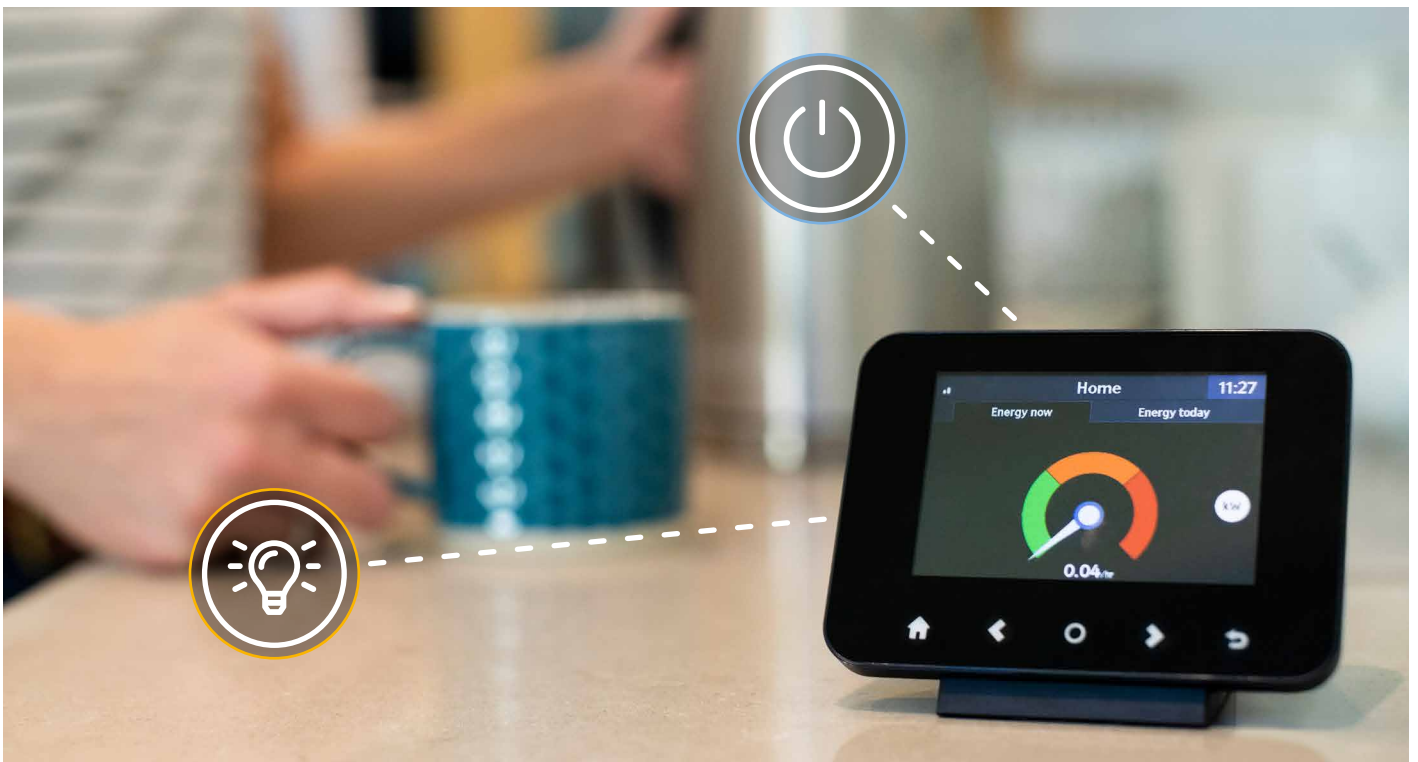
Another way to reduce the power consumption is to use a DC-DC converter. These converters reach maximum efficiency at a given operating point and cannot cover the complete current consumption of a wireless connectivity application that ranges from 100 nA to 100 mA. Therefore, the biggest power-consuming event over the lifetime of the device must be identified, and the appropriate operating point of the DC-DC converter must be selected. More than one type of regulator can also be used to more effectively cover the dynamic power consumption range.

Low-power software architecture overview

All these techniques rely on an optimized low-power software architecture (see Figure 7.8).

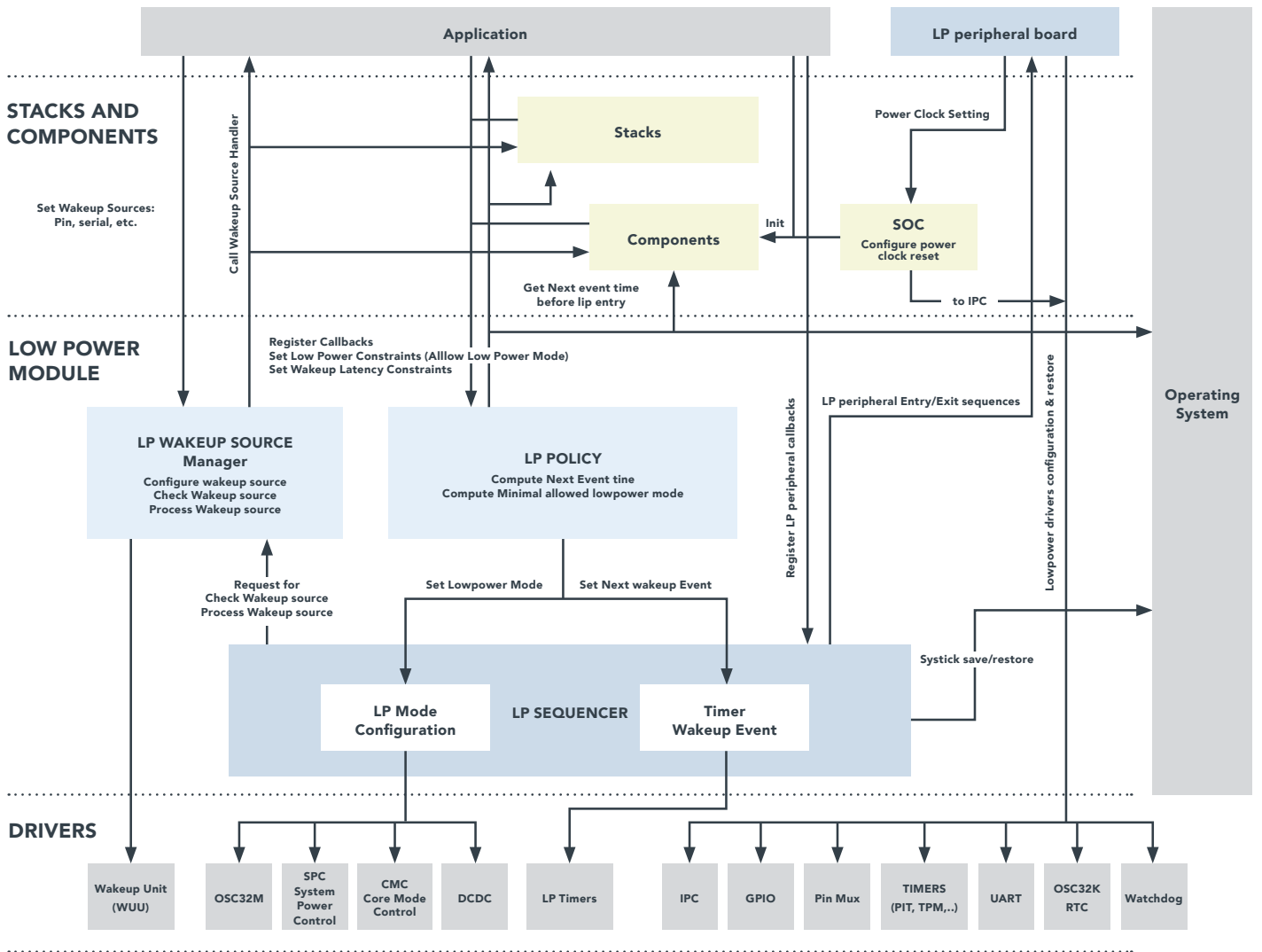
A low-power software architecture mainly consists of:

- **Low-power policy manager** — This module gathers all constraints from the upper layer, connectivity stacks and applications and then selects the best-suited mode. Low-power constraints include enabling the serial peripheral when transitioning to low-power mode because the application still needs to receive data from an external device. In this example, the power domain and clock connected to this peripheral need to be maintained. However, a lower frequency clock can be used to reduce low power if the serial bus frequency can afford reducing the frequency.
- **Low-power entry/exit sequence manager** — This module receives the selected low-power mode from the policy manager and handles low-power entry and exit sequences.
- **Low-power wake-up manager** — This module receives the wake-up source selection from the upper layer and programs the wake-up source before switching to low-power mode. On wake-up, the wake-up manager checks the wake-up source and calls the registered callback associated with this wake-up event.



LOW POWER SW ARCHITECTURE

APPLICATION



HARDWARE

Figure 7.8. Low-power software architecture example

Low-power entry/exit sequencer manager

The low-power entry sequence is triggered from the idle task scheduled by the operating system when no other activity is required on the chip. The following examples show various tasks the sequencer manager should do during low-power entry:

- **Mask the interrupts** — This atomic sequence should not be interrupted by the execution of interrupt handler functions.
- **Obtain the best-suited low-power modes from the policy manager based on next wake-up time and constraints** — As mentioned previously, if the sleep duration is too short, the policy manager selects a lighter sleep low-power mode with a shorter wake-up time.
- **Call peripheral callbacks** — These callbacks are used to uninitialized stacks, peripherals and other components by saving the hardware context values to RAM and switching off clocks, if required. These also reconfigure the pin multiplexing setting into a low-leakage mode; typically, the output pads are reconfigured in an input or a disabled state.
- **Obtain the next wake-up time event from the timer service and program the low-power timer** — Optionally, the system ticks are suppressed and replaced by a low-power timer.
- **Configure the wake-up sources**, usually located in the hardware wake-up unit module.
- **Program the hardware for the selected low-power mode** — This involves clock- and power-domain settings, RAM retention setting, flash configuration, low-power low-dropout (LDO) settings, DC-DC conversion, etc.
- **Execute WFI or wait for event (WFE) instructions to trigger the hardware to enter low-power mode.**



The low-power exit sequence is triggered on a wake-up event. Once the hardware has restored the LDO, DC-DC, memory, flash memory, etc., the software execution restarts either from the WFI or WFE instruction, if the core power domain was maintained, or from the reset handler if the core power domain was turned off. For RAM retention, the reset handler can execute a warm boot to speed up the reinitialization as described in the warm boot section. When the processor restarts, the software executes these actions:

1. Reconfigures the system clock and restores the processor registers and nested vector interrupt control (NVIC) registers for a warm boot. For a normal restart, it directly resumes code execution from WFI or WFE instruction by skipping this first step and proceeding to the second step.
2. Restores the main clock source and reinitializes the radio drivers and the connectivity link layer if required.
3. Obtains the wake-up source event from the hardware wake-up unit.
4. Obtains the low-power time duration and resynchronizes the internal time base or timer service. Optionally, it restores the system ticks if the OS uses it.
5. Calls the peripheral registered callbacks to 1) reinitialize the peripherals' clocks, 2) restore the hardware peripherals' registers if these were powered off, and 3) reconfigure the pin multiplexing setting used by these peripherals.
6. Calls the wake-up source callback depending on the wake-up source event.
7. Unmasks the interrupt to allow new interrupts to fire.

In summary, power efficiency is a key requirement when designing an application for edge devices. As discussed previously, it presents challenges that need to be addressed at the beginning of the design stage when considering use cases and engineering requirements. Power consumption is an important parameter, but the time required to perform the various tasks that make up a complete application is equally important.

The techniques used to minimize both current and execution time require hardware capabilities and a low-power-aware software architecture. These strategies to achieve a truly low-power design have been examined in a wireless product context. However, the same holds true for other types of applications because the principle of reducing the area under the curve is always valid.

The next section covers measuring and monitoring power efficiency.

TOOLS TO OPTIMIZE EDGE NODE POWER EFFICIENCY

As mentioned earlier, low-power operation is achieved by reducing voltage, current and timing. The most efficient power reduction can be attained by driving the voltage and current to zero as often as possible. This section discusses the tools and processes that can be used to measure and optimize power consumption by monitoring the current and voltage on individual power rails.

The number of rails monitored depends on the system on chip (SoC). A simple MCU may have one power rail, whereas an advanced SOC has multiple rails. The number of rails supplying modern advanced SoCs may be large due in part to being split into multiple voltage domains (for example, separate voltages) for each processor in a heterogeneous system and for each major functional block, such as the graphics processing unit (GPU), vision processing unit (VPU), dynamic random access memory (DRAM) controller and various inputs/outputs. This allows shutting off the voltage in unused domains or reducing voltages when dynamic voltage and frequency scaling (DVFS) is supported. The more rails that are available to control and measure, the more granularity of the overall power consumption can be achieved.

Developing the lowest energy edge device requires the use of a highly efficient computing processor or SoC that has distinct functions on separate power rails and drivers or a low-power manager so developers can easily take full advantage of these capabilities. Providing developer tools such as power consumption estimators, guidelines on how to achieve power-optimized products and power monitoring is the key to not only designing power-efficient products but also designing them faster, which leads to reduced development time.

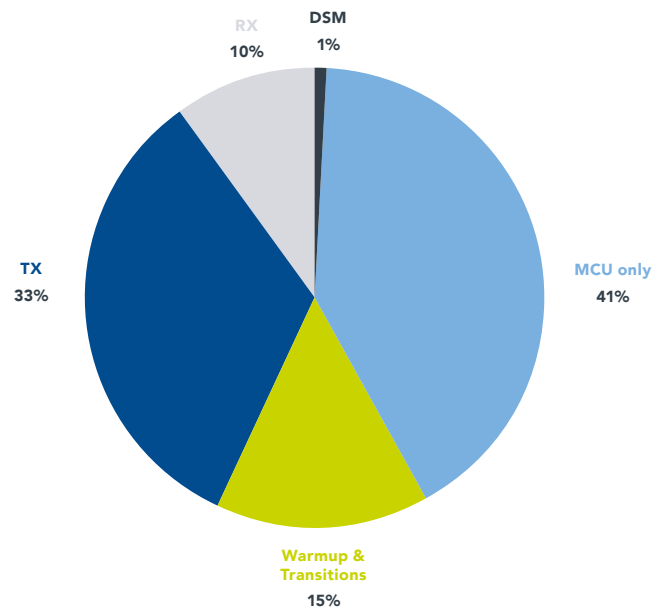


Power estimator tool and optimization guidelines

Tools to help estimate the power consumption of edge devices and guidance on how to improve them are key factors for implementing a power-efficient solution, especially for applications requiring connectivity. For example, as seen in Chapter 5, “Edge Computing Connectivity,” the Bluetooth LE specification provides improved power consumption over the Bluetooth standard. But having a good understanding of the protocol and the connectivity devices’ capabilities and performances is required to reach the best power efficiency.

Companies that provide connectivity devices usually offer simulation tools and guidelines via application notes on how to achieve the best performances with their SoCs. These can be used, for example, to assess the power consumption and timings of the different Bluetooth LE profiles such as advertising, scanning or connection. They provide a complete power consumption estimation breakdown (see Figure 7.9).

ADVERTISING POWER CONSUMPTION BREAKDOWN



ADVERTISING PROFILE

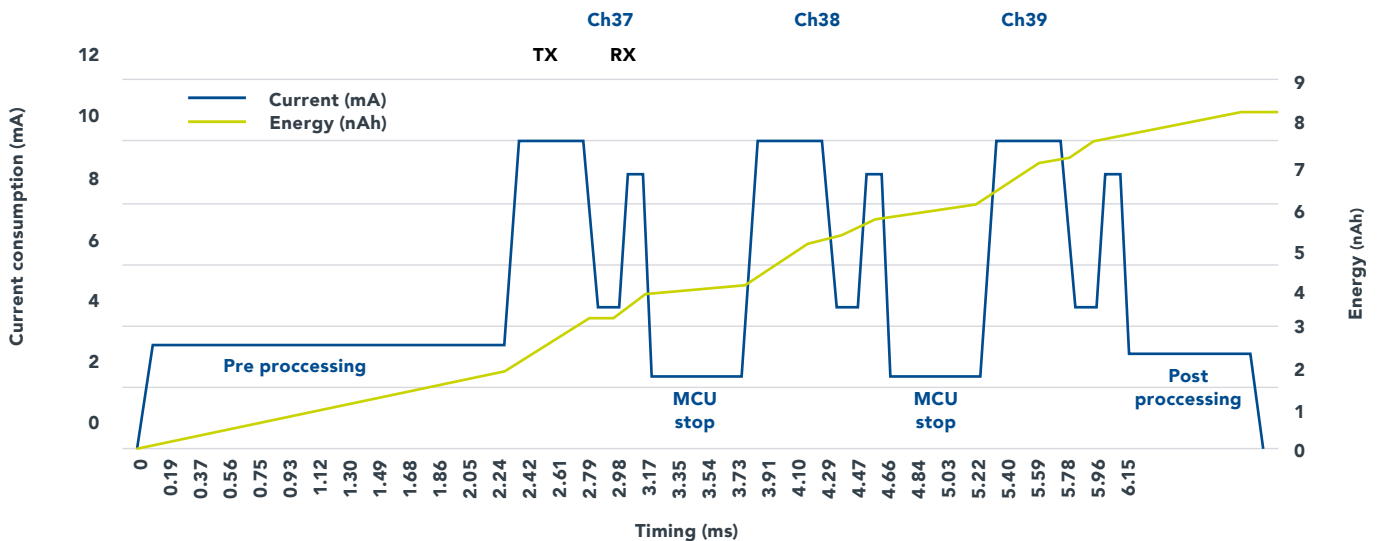


Figure 7.9. Example of Bluetooth LE power consumption breakdown and timings

Power monitoring tools

The development of software running on edge devices is started and performed mostly on the selected processor's evaluation kit, typically long before the first device prototypes are available. Embedding power monitoring functionalities directly into these evaluation kits enables the user to start writing power-efficient code from Day 1. The power impact of any change or new line of code can be monitored immediately.

Efficient board power monitoring requires a combination of hardware and software:

Power monitoring hardware can be used to precisely measure the SoC's voltages and current variations on each power rail. The power measurement application software collects power sample values from the ADC and send them to the user in an efficient manner, for example, using a graphical interface with power consumption. The application can be stand-alone or embedded into other tools, such as a debugger.

Onboard power monitoring pros

Power measurement devices are already populated and connected on the rails of interest, so no board rework is required. Boards also usually come with power measurement applications for collecting and analyzing measurement data already implemented. Collection and analysis can be conducted via a single application running, for example, on a remote host, or a separate data collection can be performed using a dedicated processor on the board and then transmitting that data to a host for remote analysis. With this option, power monitoring and analysis can be performed right out of the box.

Onboard power monitoring cons

The measurement precision and sampling rate of the onboard monitoring devices may be limited. For use cases exceeding onboard hardware capabilities, measurements give only a rough idea of the profiles, so connecting with and using external high-precision power monitoring equipment are important.

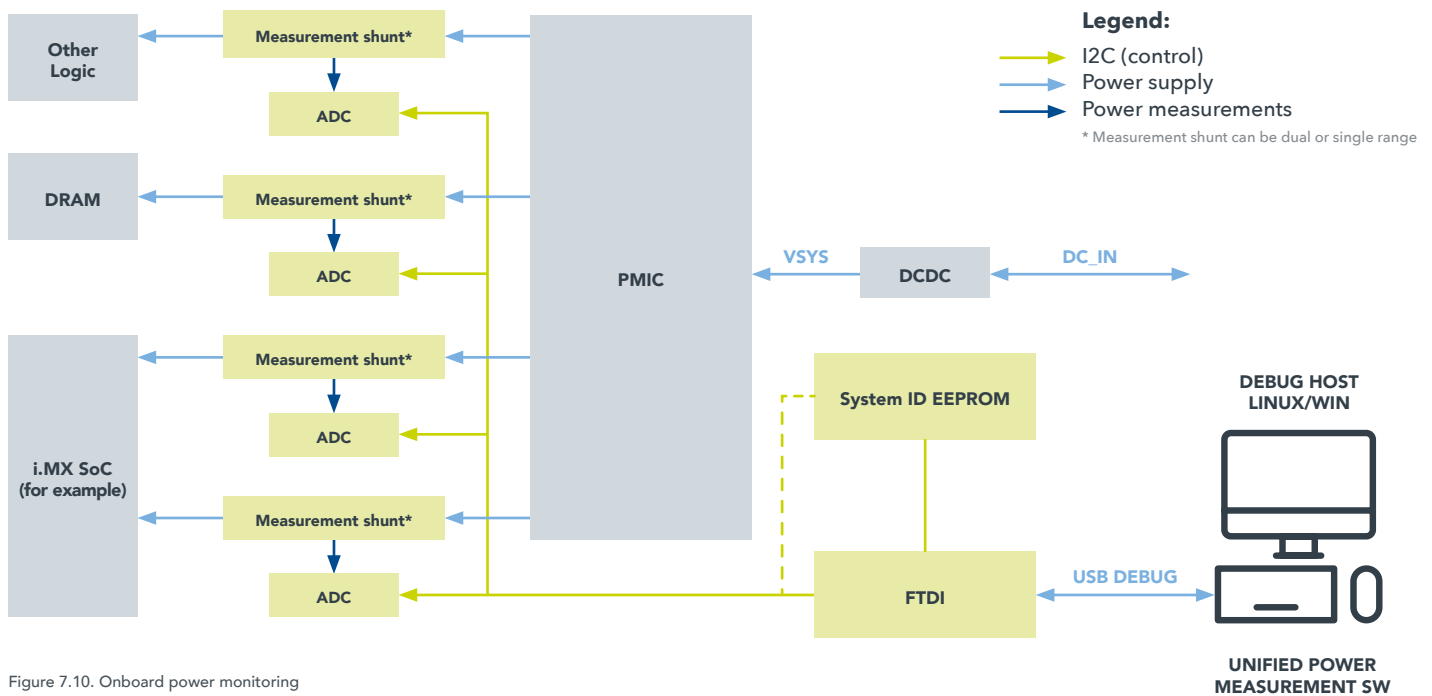


Figure 7.10. Onboard power monitoring

Off-board power monitoring pros

Off-board power monitoring means connecting external power acquisition equipment to the board to monitor any rails of interest using equipment that meets sampling and precision requirements. Some debuggers also include embedded power monitoring features. In that case, the power monitoring application is usually provided by the integrated development environment (IDE) through its power-aware debugging capabilities (see Figure 7.12).

Off-board power monitoring cons

Power measurements cannot be performed right out of the box; they often require planning and effort to connect the external power acquisition equipment. Also, 2-pin headers with jumpers are usually populated as probing access points on the power rail of interest to easily connect the monitoring equipment in place of the jumper. These headers, acting like antennas, can impact radio-sensitive measurements such as those with Bluetooth LE. In that case, implementing two different boards, depending on requirements, may be necessary:

- One board with no header for radio measurements but no power measurements
- One additional board with headers for power measurements but reduced radio performance

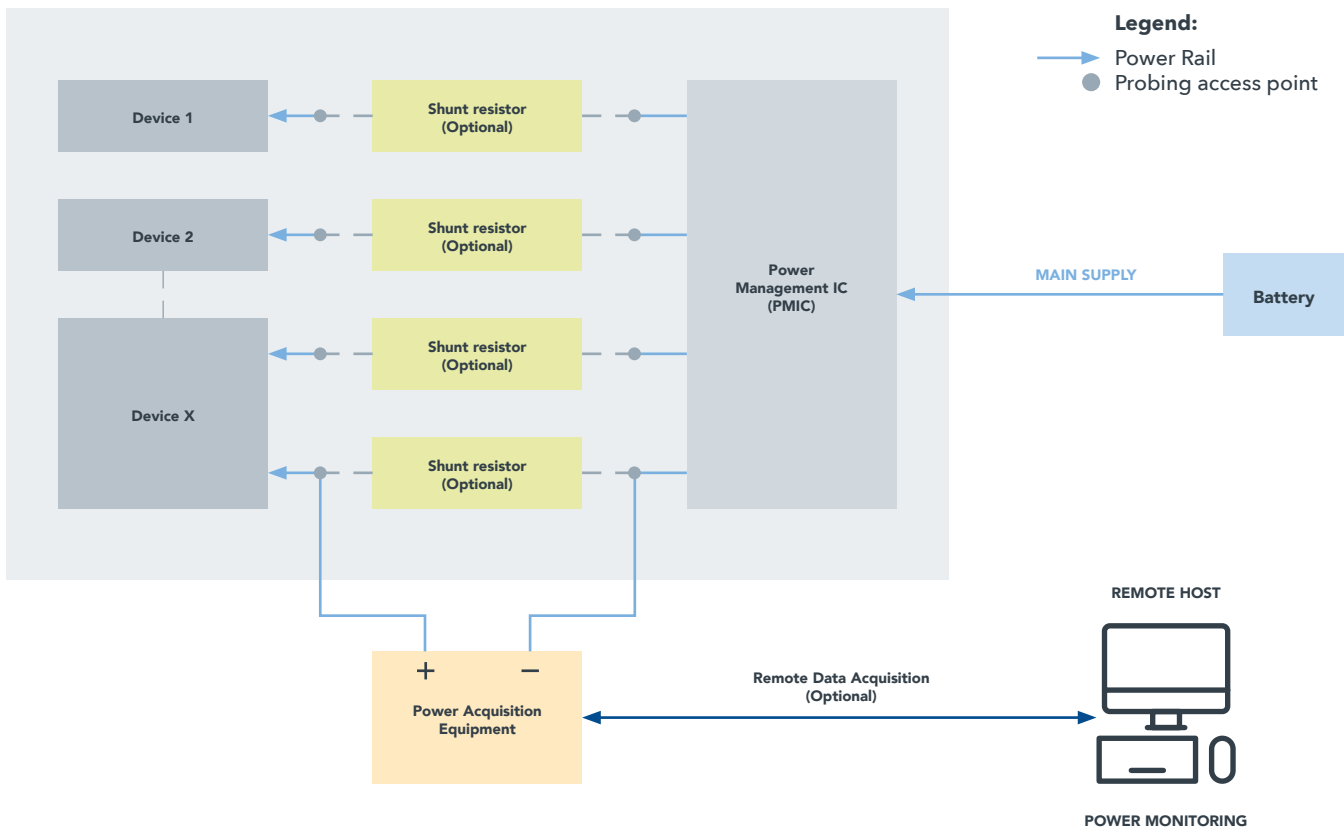


Figure 7.11. Off-board power monitoring

Developing the most efficient devices is a team effort between the user and the semiconductor manufacturer. Providing high-quality power measurement tools helps developers build the most power-optimized devices quickly and efficiently. Reducing the time developers spend measuring and optimizing the power consumption results in better products and shortens development cycles.

Here is a list of useful approaches to optimize power using software techniques;

1. Architect the software to have natural "idle" points (inc. low power boot)
2. Use interrupt-driven programming (do not use polling, use the operating system to block instead)
3. Code and data placement close to processor to minimize off-chip accesses (and overlays from non-volatile to fast memory)
4. Smart placement to allow frequently accessed code/data close to CPU (and use hierarchical memory models)
5. Size optimizations to reduce footprint, memory and corresponding leakage
6. Optimize for speed for more CPU idle modes or reduced CPU frequency (benchmark and experiment!)
7. Don't over calculate, use minimum data widths, reduce bus activity, smaller multipliers
8. Use DMA for efficient transfer (not the CPU)
9. Use co-processors to efficiently handle/accelerate frequent/specialized processing
10. Use more buffering and batch processing to allow more computation at once and more time in low power modes
11. Use the operating system to scale voltage and frequency where possible

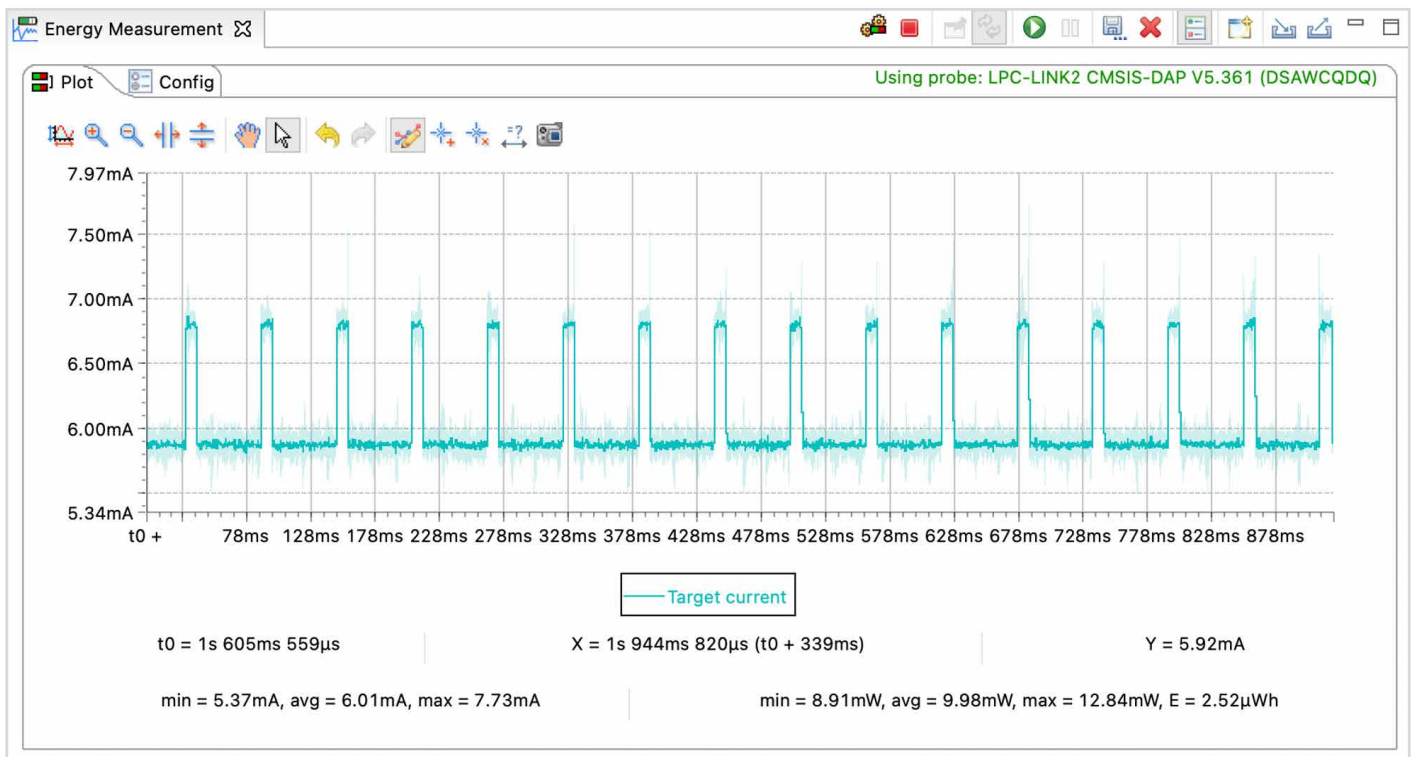


Figure 7.12. Example of a power profile provided by a power-aware IDE

Chapter 8

EDGE COMPUTING HUMAN MACHINE INTERFACE

CONTRIBUTORS

Alex Dopplinger, Director, Product Marketing, Building and Energy, NXP Semiconductors

Guillermo Michel Jimenez, Graphics Systems Engineer, NXP Semiconductors

Edge processing presents new opportunities for human machine interfaces (HMI). This chapter explores applications.

DATA VISUALIZATION THROUGH HMIs

Edge processing systems often include data presented to the user. Depending on the use case, different data may be exposed from raw data and processed data to intermediate data. Because this data is often sensitive, presenting it on the edge device makes more sense than presenting it on another device that is less secure because it is connected to the cloud. To support that functionality, edge device system on chip (SoC) solutions integrate graphics pipelines.

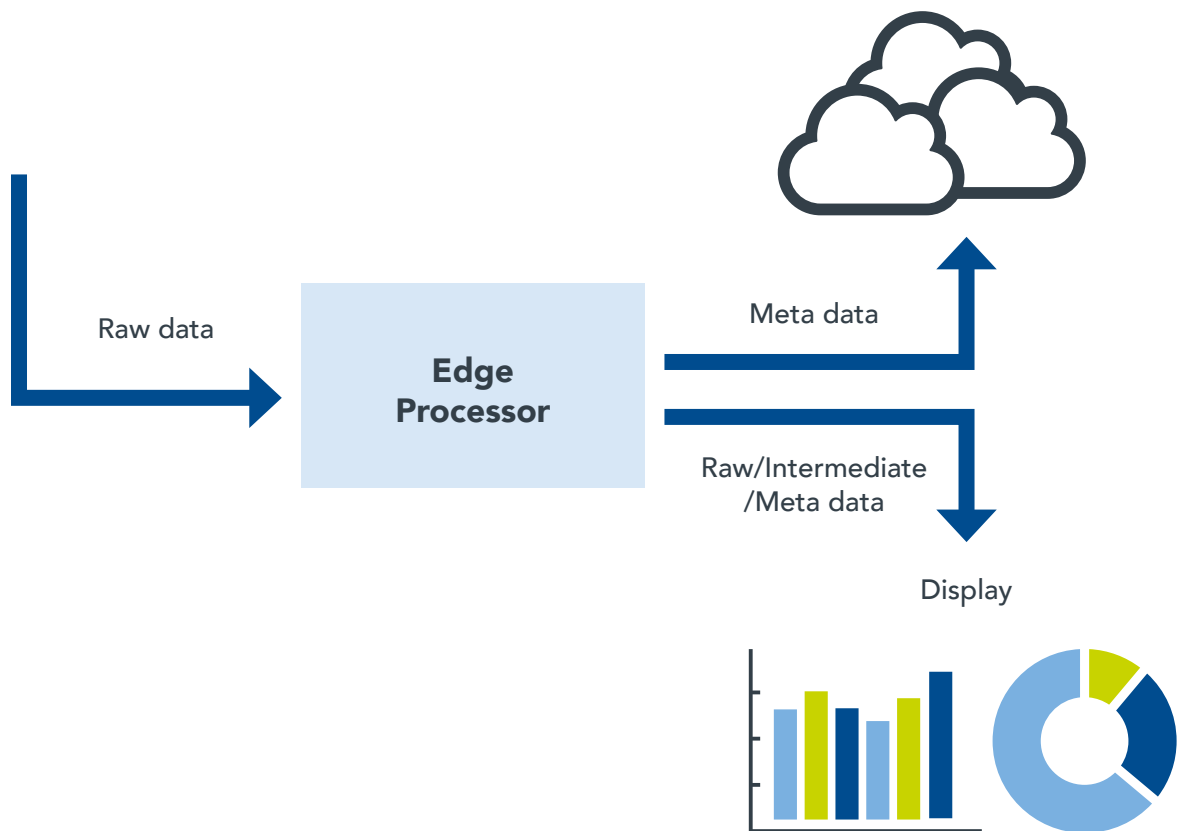


Figure 8.1. Edge processor simplified HMI diagram

However, communicating the data to a user can be challenging. Users often can't make decisions or understand processes just by looking at numbers, so data visualization can help them. Data visualization consists of presenting data in an arrangement of visual elements, such as charts or maps, so users can easily consume the data.

Gathering data, processing it on the edge (probably using artificial intelligence and machine learning), presenting it to the user and/or sending it to the cloud are common activities in almost every industry and even everyday life.

In the industrial sector, factory floors are overflowing with panels depicting information created from data gathered and artificial intelligence (AI) processed to increase efficiency, product quality and machine control. In the medical field, physicians and lab technicians constantly refer to the screens of digital assistants driven by AI, real-time data and historical data.

In the fitness world, biometric, preference and equipment usage data is gathered by exercise equipment while presenting informative dashboards and suggestions to athletes and coaches. Home equipment also tracks a variety of data. For example, high-end soundbars that are AI tuned show informative graphics of a room's acoustics.

In the automotive industry, car cockpits have transformed into an arrangement of digital dashboards showing car status, navigation and route information. Soon these dashboards will show personal and local information by connecting with other devices. Even wearables, appliances and internet of things (IoT) devices gather data, process it and present it on screens.

Consider some common data visualization implementations on edge devices.

REAL-TIME DASHBOARDS

Dashboards grew in importance during the COVID-19 pandemic. Most localities around the world updated dashboards daily with useful information such as active cases and hospital occupation percentages. These dashboards helped individuals make decisions regarding their personal actions during the pandemic.

Dashboards with edge processors are used to present real-time information updated constantly to streamline user decision-making or increase user trust for a variety of applications.

Consider an industrial use case such as a power plant or an assembly line. In one of these settings, edge processors gather data and control processes using AI and machine learning (ML) while presenting a dashboard with key sensor and processed data to operators in real time.





Designers develop dashboards with a UI creation tool that can deploy to edge processors. They use the tool to create buttons, sliders and graphs and to load 3D content with lights, particle systems and more. The tool exports the content to an application able to run on the edge processor Operating System (OS). The application makes use of standard graphics application program interfaces (APIs), such as EGL™, OpenGL® ES and Vulkan®, to generate frames through the graphics processing unit (GPU).

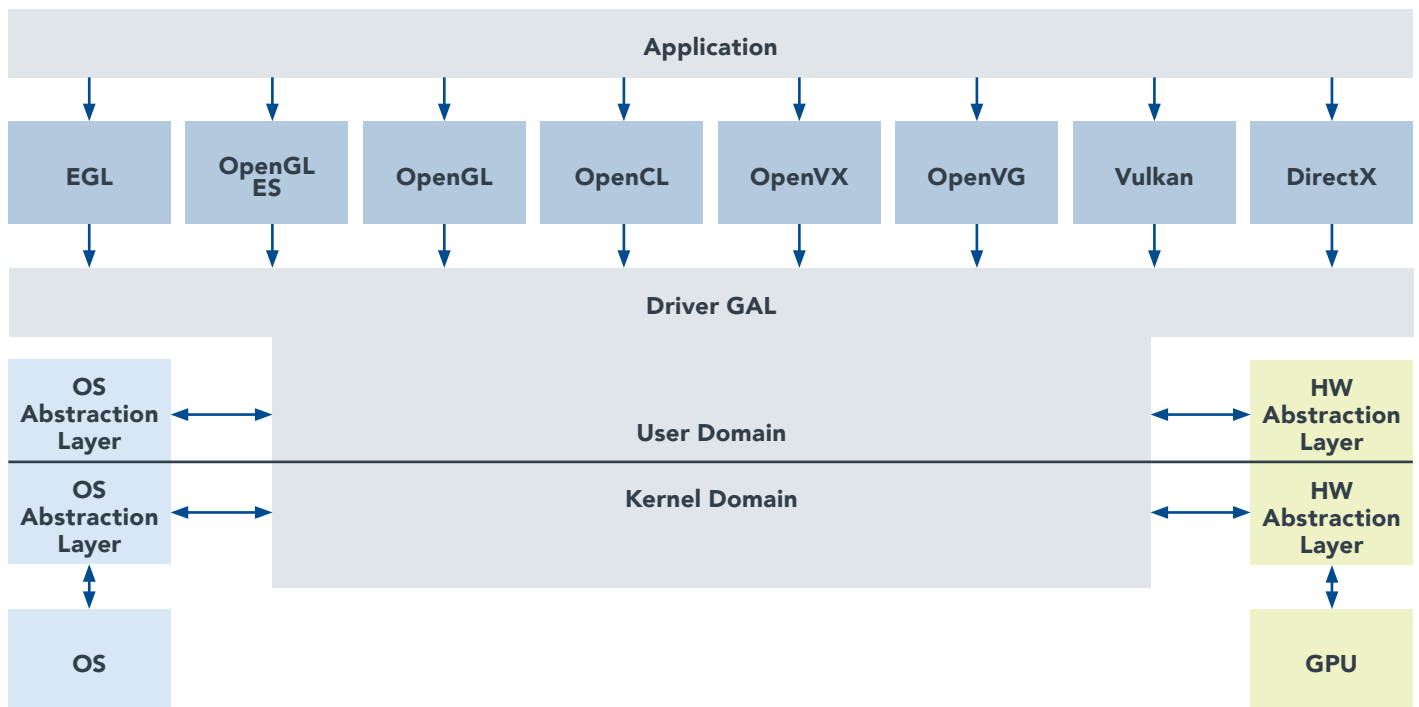


Figure 8.2. VeriSilicon graphics driver architecture

Modern industrial dashboards range from small Extended Graphics Array (XGA) resolutions to 4K displays. With these requirements in mind, 3D graphics and up to 4K resolution, consider the MPU block diagram in Figure 8.3.

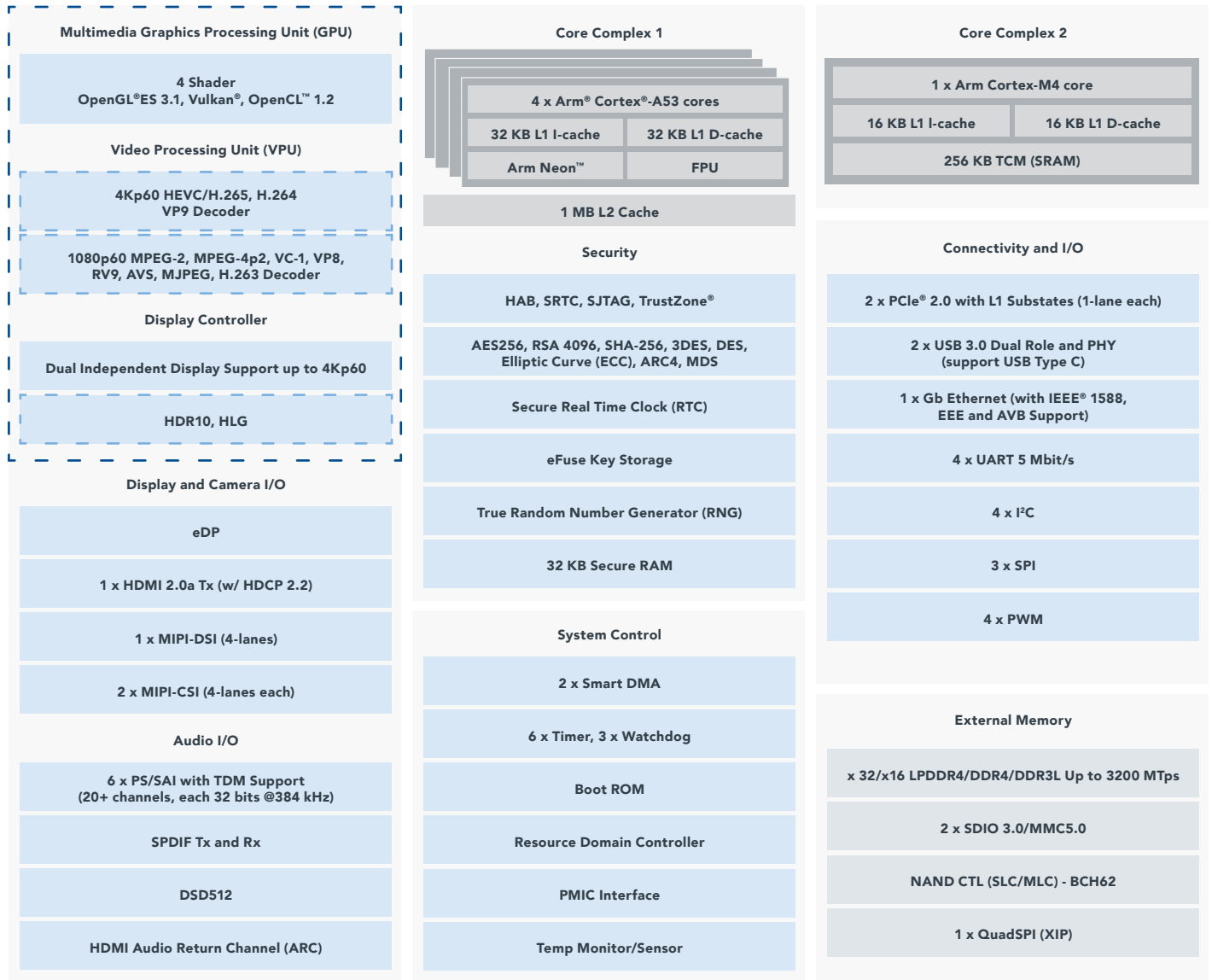


Figure 8.3. i.MX 8M block diagram

MPUs that feature a GPU to create 3D graphics and a display controller are able to support up to 4K resolution and are commonly used in industrial applications.

WEARABLE TRACKING INTERFACES

Fitness wearables generate data while users sleep, walk and move. They also use data visualization techniques to show information to the user.

However, these techniques have been adapted to the smaller displays of wearables. Figures 8.4 and 8.5 show two Crank Software tracking visualizations.

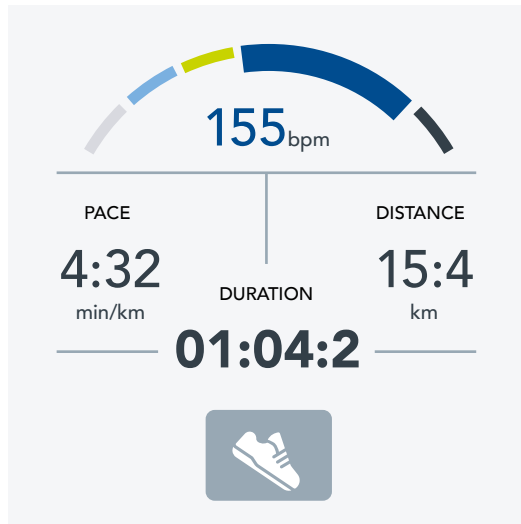


Figure 8.4. Running tracker (Source: Crank Software)

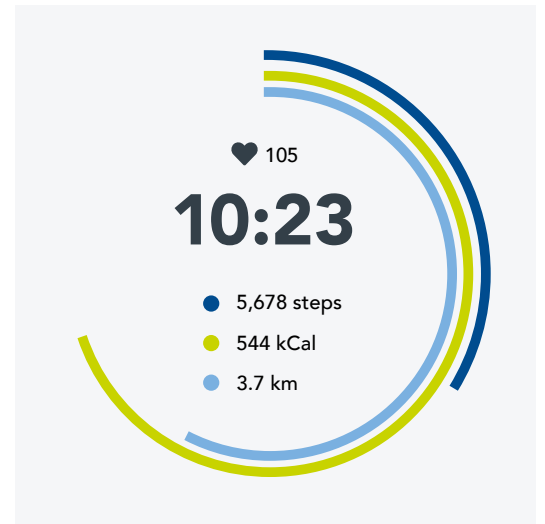


Figure 8.5. Step tracker (Source: Crank Software)

In addition to their graphics and data collection requirements, wearables and small devices must have a reliable, low-power solution with real-time response capability to drive trackers for hours or even days. MCUs are an efficient solution for low power, reliability and real-time OS (RTOS) support, while MPUs provide effective connectivity and graphics.

Wearables and small devices need all these features combined into in one solution: crossover MCUs. This technology provides the low power and reliability of MCUs and the connectivity and graphics of MPUs. Figure 8.6 shows the block diagram of a crossover MCU with graphics and connectivity.



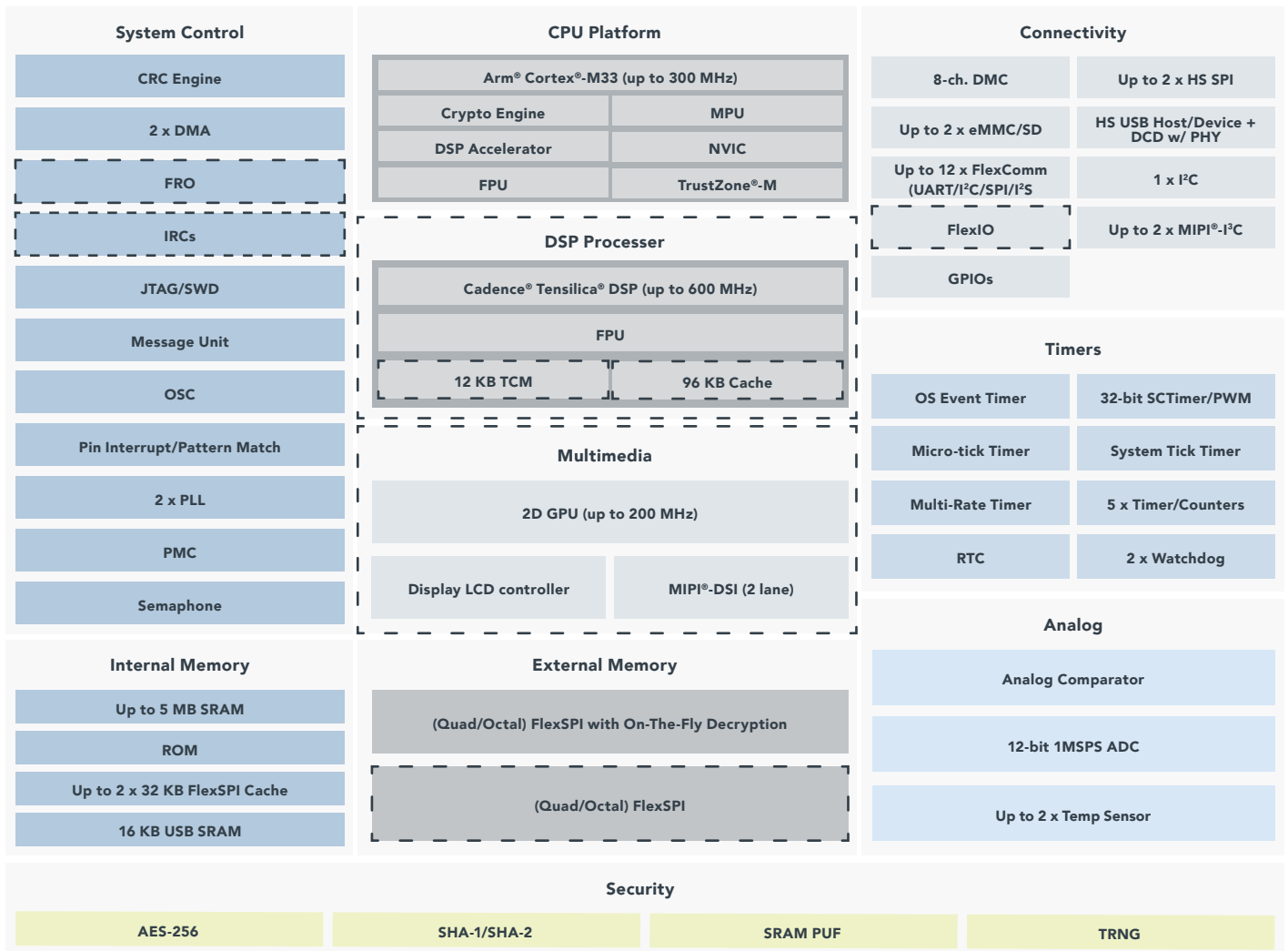


Figure 8.6. Example MCU block diagram

Instead of a 3D GPU, the crossover MCU has a 2D GPU. 3D GPUs are larger and consume more power, so 2D GPUs are a sensible choice for crossover MCUs. Like an MPU, the crossover MCU contains a DSP and a MIPI-DSI interface. Silicon vendors have worked hard to include all these features on the software develop kit (SDK) bare metal and RTOS solutions in crossover MCUs.

For a designer using UI tools that support both MCUs and MPUs, the experience is almost identical — practically the same design flow and features are present in both cases except for loading 3D models.

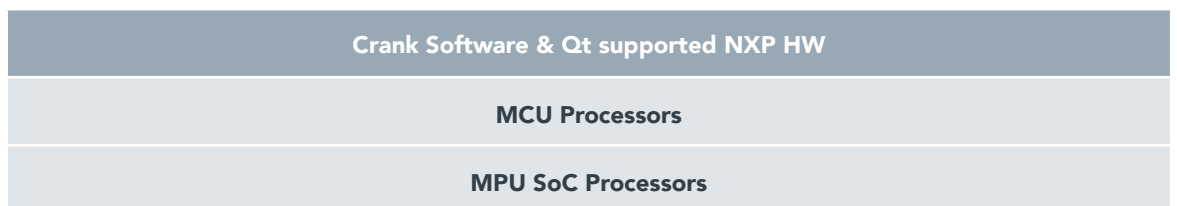


Figure 8.7. UI tools that support MPU and MCU processors

However, some MCUs offer new possibilities for embedded UIs by incorporating vector graphics units. Vector graphics are defined by points on a Cartesian plane that are connected by lines and/or curves.

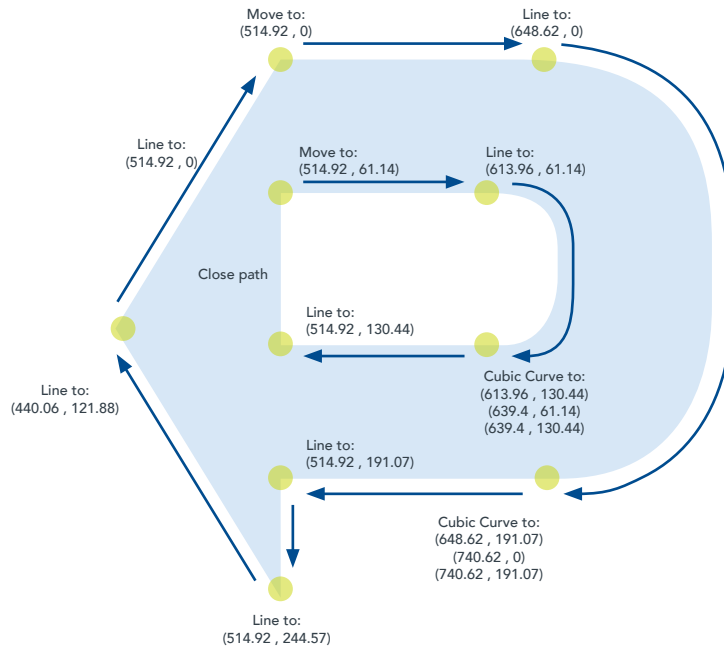


Figure 8.8. Vector graphics polygon example

Vector-defined 2D graphics can be modified with matrix transformations that easily can be mapped to GPU hardware. This allows scaling, rotations and other manipulations without losing quality or causing aliasing, so it is useful for both text and graphics on a wearable or handheld device.

You can apply the same transformations to bitmaps. For example, a perspective transform can be used on a bitmap to implement a cover flow interface.

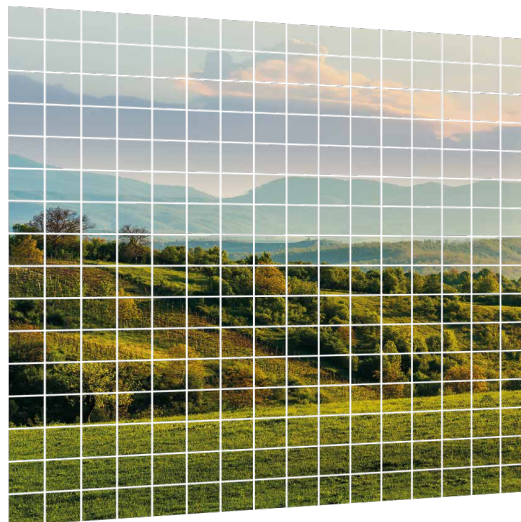


Figure 8.9. Transformed bitmap

Designers often combine vector graphics and data visualization tools to create effective HMIs. The most popular data visualization tools are vector graphics back ends and JavaScript vector graphics for the web. Most of these vector graphics implementations are created with Scalable Vector Graphics (SVG), a markup language to describe vector graphics supported by internet browsers.



Figure 8.10. Raster versus SVG graphics (Source: https://commons.wikimedia.org/wiki/File:Bitmap_VS_SVG.svg)

A vector graphics GPU in an MCU provides SVG support and ensures that the MPU is up to date with current data visualization trends. It also presents the possibility of asset reuse. Many UI tools vendors are responding to this demand by incorporating SVG loaders in their tools.

Using SVG results in crisp, zoomable and fluid graphics on wearable devices, which makes data consumption even easier for the user.

VEHICLE DATA VISUALIZATION

There are several examples of data visualization on the cockpits of vehicles, from obvious things like an inData visualization in the cockpits of vehicles ranges from obvious features like an instrument cluster showing gas, speed, and revolutions per minute to an infotainment system showing real-time battery usage and charge on a hybrid vehicle (Figure 8.11).

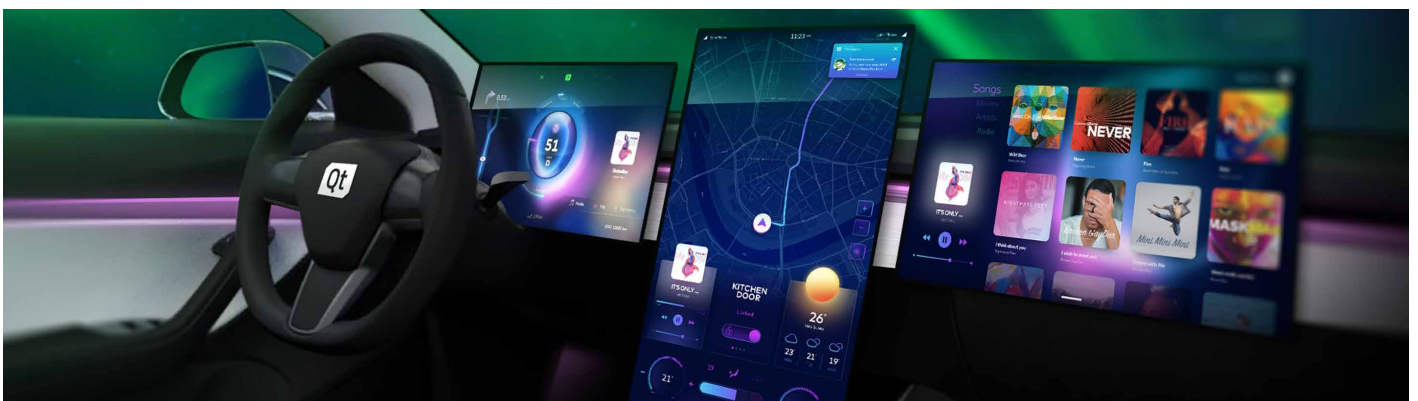


Figure 8.11. Example vehicle eCockpit display system (Source: Qt Company)

A vehicle eCockpit consists of a digital instrument cluster and a digital infotainment system. Automotive multimedia application processors that can run an eCockpit on a single device are fairly common nowadays. Some of these devices also incorporate driver monitoring systems (DMSs) and other AI/ML driver assistants.

Consider the multimedia block of the i.MX 8 block diagram in Figure 8.12. This SoC contains advanced 3D GPUs and display processors designed for eCockpits.

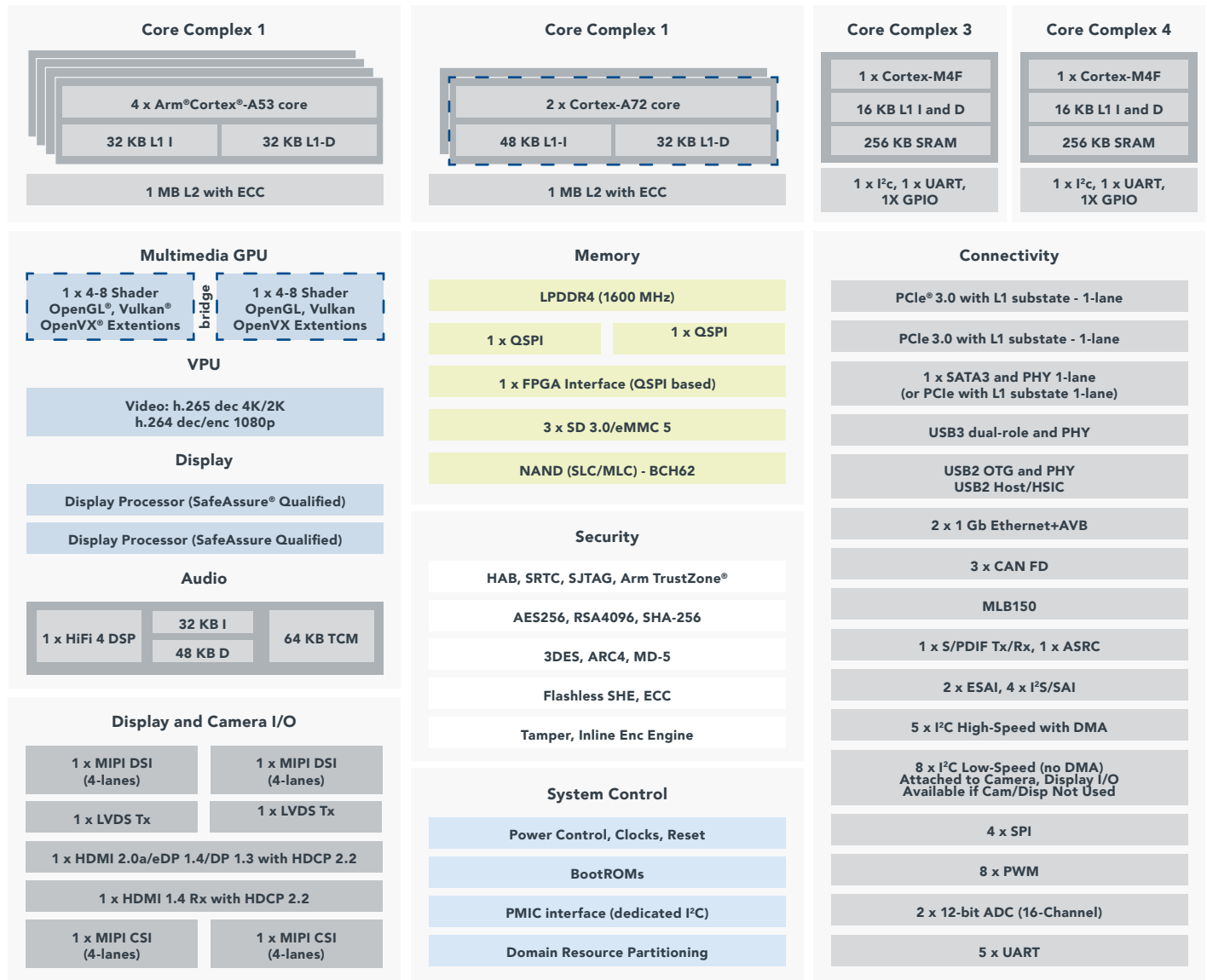


Figure 8.12. i.MX 8 block diagram

The security standards for instrument clusters require critical information to be periodically checked. Speed, revolutions per second and indicators are all critical information. SafeAssure® capabilities allow an MCU, which offers high reliability and fast response times, to control the display processor and perform periodic security checks on areas of the screen (see Figure 8.13).



Figure 8.13. Cluster and heads-up display with SafeAssure areas indicated in red

Emerging technologies such as vehicle to everything (V2X) will broaden those possibilities even more by enhancing the data presented by navigation systems with information from devices other than the car (see Figure 8.14).

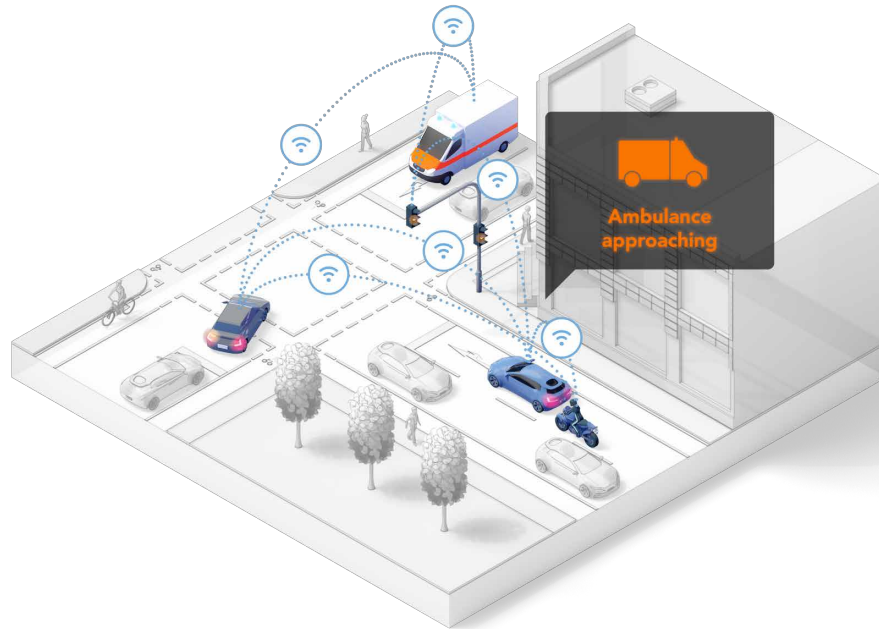


Figure 8.14. V2X communication (Source: NXP)

Level 3 autonomous vehicles also offer compelling data visualization examples. When vehicles enter a condition during which they can self-drive, such as navigating a highway, drivers see a screen showing data processed by the advanced driver assistant systems (ADAS) for objects detected, distance to the objects, choices made by the car and more (see figures 8.15).



Figure 8.15. ADAS visual representation (Source: NXP)

eCockpit multimedia application processors offer use cases such as these in a secure and reliable way.

eCockpit single MPU implementations require powerful GPUs and display processors to support several high-resolution displays and create complex 3D graphics running on safe virtualized environments.

HMIs are key to helping users make sense of this new data-driven world, and they are essential for it to reach its full potential. Edge processors have functionality to achieve this goal while taking care of sensitive data.

EXAMPLE: TOUCHLESS CONTROL FOR THE INDUSTRIAL IOT

Many people use HMIs with sleek glass touch screens and ubiquitous buttons daily in homes, vehicles, workplaces and public venues. But the recent pandemic raised concerns about multiple people spreading the virus by touching the same buttons or screens. The demand has increased for alternative interfaces to reduce the need for physical contact. Entering a PIN at an ATM terminal, purchasing a train ticket at a kiosk or simply selecting the desired floor in an elevator involve touchable surfaces that hundreds of people have used. Touchless alternatives for humans to interact with machines in the workplace, retail and hospital settings will be a growing trend.



Touchless controls

The industrial internet of things (IIoT) automates manufacturing and smart machine communications, but humans still must interact with machines occasionally. To reduce germ and virus transmission, touchless technology needs to replace traditional push-button and touchscreen controls.

Many users are familiar with voice assistant applications at home or in vehicles. However, this type of voice control is unreliable in noisy manufacturing facilities, active outdoor environments or groups of people who are speaking. For these cases, speech and gesture can be combined to provide a more adaptable and robust multimodal touchless interface.

With voice or vision-controlled systems, machines must quickly and reliably differentiate between deliberate user instructions and random or unintended inputs. For example, a machine should turn on only when the user intends this response and not simply because a person is standing near it and talking. Machine vision systems can recognize gestures such as hand and foot movements, head nods and finger pointing. Interpreting body language can become a more natural way for machines to respond to visible inputs from human operators.



Gesture-based solution development

The first step in developing a gesture-based solution is to identify which gesture types the system must recognize and interpret. For example, will the user communicate using hands only or a full-body movement? Will finger movements be easier for the vision system to capture than body movements, which could be partially obscured by clothing or other items a person is carrying?

Gesture complexities are also important design parameters. For instance, opening a door might need only a single hand wave, but adjusting environmental controls or changing a production line might require a range of intricate gestures.

Finally, the speed of the movement and environmental conditions (for example, lighting levels that are too low or too bright) can play a significant role. Understanding all these factors helps determine the number and type of camera sensors, field of view, focal length and resolution needed to detect and interpret the gesture.

A backup interface, such as voice control or a physical touch screen, should be offered in case the user cannot use the gesture method. For safety-critical functions in industrial environments, the application software may need a functional safety assessment and certification, such as IEC 61508 for industrial systems.



Use case 1: Door opening by gesture control	Machine inference or instruction
Notice a person near the door.	Recognize that a person, instead of a cat or car, is in the field of view.
Confirm the person wants to interact with the door.	Recognize eye contact.
Communicate that the control panel can accept gesture inputs.	Provide a first-time users guide if required.
Focus on the area of interest.	Identify the face, hand, foot or other area of interest.
Capture and recognize the gesture to safely open the door. There may be more than one type of acceptable gesture to open a door.	Correctly interpret the gesture(s) as a request to open the door.
Use case 2: Fast-food kiosk ordering using hand gestures	Machine inference or instruction
Notice a person near the kiosk.	Recognize that a person, not a robot floor cleaner, is in the field of view.
Confirm the person wants to interact with the kiosk.	Recognize eye contact for a defined time period or a hand approaching the display.
Communicate that gesture inputs can be used to place an order.	Provide a first-time users guide as a short video file if necessary.
Display a menu of food options.	Encourage the customer to select from the menu with visible or audible cues.
Accept user inputs by tracking the customer's pointing finger, swiping hand and clicking motions that don't need to touch the display.	Correctly interpret the finger or hand location with respect to each selectable menu option, and recognize the gesture to select the item, for example, "air" mouse with "air" click.
Capture the food order and request contactless payment.	Print, email or text receipt or confirmation.

Table 8.1 shows two use cases for gesture recognition.



ML for gesture recognition

Once the gesture, environment and camera types are understood, a gesture-recognition ML model needs to be acquired or built. The left side of Figure 8.18 shows the steps to convert gesture examples into an inference engine, which is the algorithm that recognizes the gesture. TensorFlow, ONNX and PyTorch are some commonly used tools for this purpose.

Identifying the hardware and software needed is next. Gesture recognition systems are typically built on industrial-grade embedded platforms ranging from a single, smart camera connected to a general-purpose computing core to multiple camera sensors feeding multicore processors with highly optimized vision and ML accelerators. Figure 8.18 shows two options for a gesture-recognition system: a lower cost microcontroller for simpler systems and a higher end system on chip (SoC) applications processor for more complex or faster responding gesture and vision systems.

Scalable Machine Learning and Vision Enablement

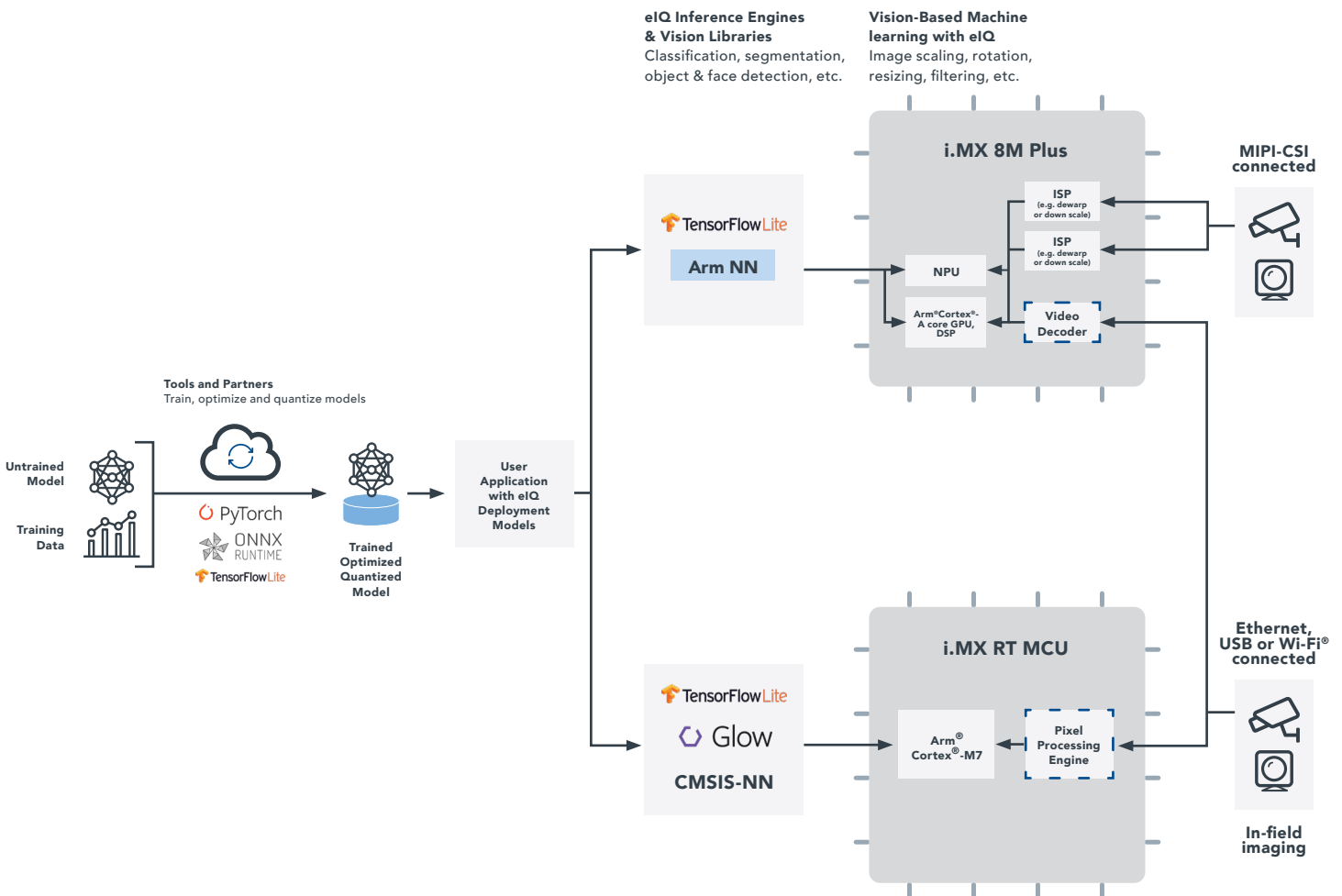


Figure 8.18. Vision solutions

Vision solutions range from MCUs with a single Arm® Cortex®-M7 core, to a more performance-centric applications processor which integrates a neural processing unit with flexible interface options.

Stereo vision cameras can use either MIPI-CSI, USB or Ethernet connections, together with audio inputs, to recognize speech and sound generators and provide audible user feedback. A display panel also can deliver visual instructions and feedback to the user. It may incorporate backup touchscreens in case the contactless control fails or the user cannot interact with it.

Vision systems require hardware and software components that are often bundled into development kits for the user. These include cloud-based tools for object-recognition tasks. This kit can be used to collect gesture examples and transfer them to the cloud tools to train the gesture-recognition models. Then the resulting inference needs to be loaded onto the same kit to recognize the gestures and inform the machine how to respond.

SUMMARY

Touchless controls not only keep users safe but also improve the way humans interact with machines in industrial and manufacturing environments. Existing hardware and software submodules can be used to build cost-effective gesture-based controls, which are responsive and reliable, and enable a new era of touchless UIs that will help industry thrive in the new normal.



Chapter 9

EDGE COMPUTING APPLICATIONS AND USE CASES

CONTRIBUTORS

Jean-Christophe Bodet, Senior Director, Edge Processing Software R&D, Industrial and IoT, NXP Semiconductors

Antoine Boiteau, Director, Software Engineering for Edge Processing, NXP Semiconductors

Brian Carlson, Director, Global Automotive Product and Solutions Marketing, NXP Semiconductors

Guillermo Michel Jimenez, Graphics Systems Engineer, NXP Semiconductors

Laurent Pilati, Director, ML and Voice Engineering, NXP Semiconductors

Wim Rouwet, Distinguished Member of Technical Staff, NXP Semiconductors

Francois Villeneuve, Director, Sensor Business Development, NXP Semiconductors

This chapter offers a deeper look at the following edge computing examples that use some of the key technologies covered in previous chapters.



Local voice at the edge optimized for power efficiency



5G technology as an enabler for Industry 4.0



Wearables



Time-sensitive networking (TSN) and distributed real-time computing at the edge



Intelligent connected vehicles

EXAMPLE: LOCAL VOICE AT THE EDGE OPTIMIZED FOR POWER EFFICIENCY

Voice user interface technology is usually composed of a wake word that wakes up devices and voice commands, or a small vocabulary, that is recognized within a few seconds after the wake word event (Figure 9.1). This example reviews the acoustic interfaces to deploy voice and undergo speech dataset preparation for building a voice wake word.

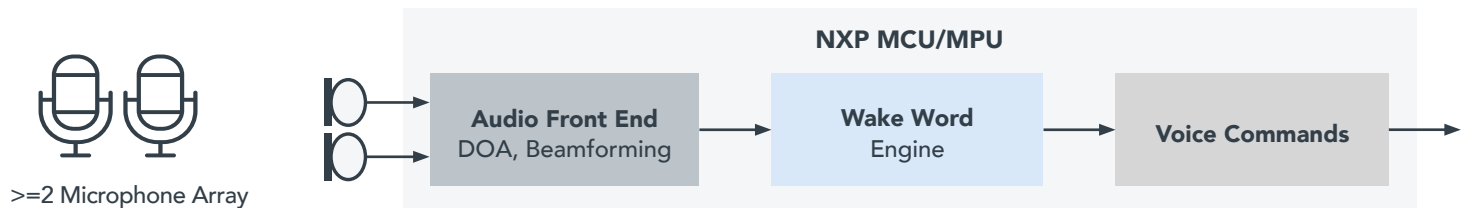


Figure 9.1. Local voice blocks

Edge versus cloud processing of voice commands

Deploying voice user interface technology that relies only on edge processing provides multiple benefits. The local voice system is always functional because users don't need data access. The latency, which is an important perception criterion in the user design, is completely under the user's control. Also, the system is private by design because voice data remains local without the need for sending or storing speech.

As far as accuracy is concerned, for a limited vocabulary (i.e., fewer than 20 voice commands), a local voice approach can be more accurate than the cloud. This is possible by limiting the research space to a few words or commands.

The voice user interface is designed to improve the user experience, the overall chain from acoustic to machine learning (ML) model deployment on a platform is important. The chain includes the acoustic interface and far-field technology.



Acoustic interface

The acoustic interface is responsible for translating acoustic signals in the digital domain.

Most devices use digital microphones. A device's digital microphone should cover the full acoustic dynamic range from lowest speech in far field to the highest sound pressure. The highest sound pressure that needs to be supported is mainly coming from audio playback that should be converted without saturation. This is known as the acoustic overload point, which could range from +120 dB sound pressure levels to 130 dB sound pressure levels. In the case of saturation, the non-linearities impact the acoustic echo canceller performance. In the lower part, the lowest sound pressure that needs to be supported is mainly coming from the talker distance. This is known as the signal-to-noise ratio, which can range from +60 dB to 70 dB.

The overall power budget of one digital microphone (~1 mW) compared with an MCU (~50 mW) running at full speed is negligible. However, if the user needs a low-power strategy (idle mode in silence), the MCU can sense the microphone every few milliseconds and then return to deep power sleep. In that condition, MCU average consumption can dip below 1 mW (some microcontrollers can drop down to 100 μ W on average in deep sleep mode with some active time for audio sensing), so using the digital microphone's low-power mode by reducing the digital clock is preferable. Microphone manufacturers now have dedicated analog front-end technology that can wake up the MCU in case of speech activity.



Far-field technology

Far-field technology in voice user interfaces has a clear definition. It is not related to a fixed distance between the talker and the device; instead, it is determined by the acoustic properties of the area where the device is located. Figure 9.2, shows the speaker source on the left. At any point, the user experiences two sound sources: the direct sound from the talker (in green), which is decreasing with the distance, and the reverberation in the room (in blue), which is constant no matter the user's position in the room. The distance at which these two sources have some energy is called the critical distance. After that distance is the far-field condition. The energy of the reverberation rises higher than the direct path. When this happens, the user has more difficulty focusing on the talker because the user cannot rely on the energy anymore. This is why supporting a far-field condition is complex.

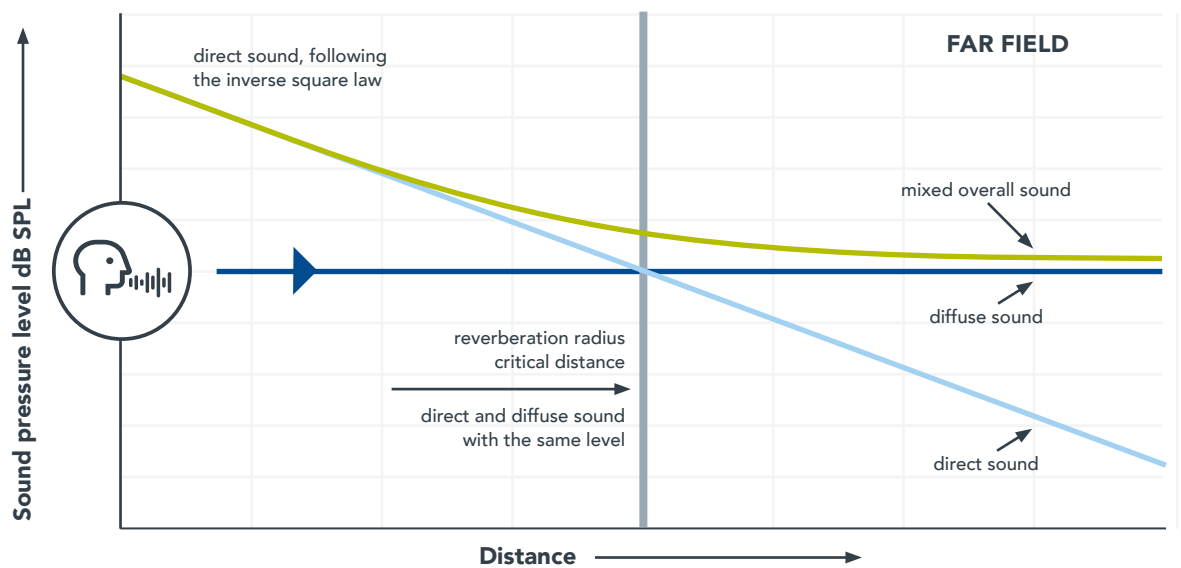


Figure 9.2. Far field

Performance criteria of the voice user interface

Voice user interface is one industry domain that does not rely on the usual standardization organizations such as the European Telecommunications Standards Institute or the International Telecommunication Union. It's mainly driven by Google and Amazon, which have already deployed millions of voice devices. Performance criteria focus on the true positive rate, the false positive rate and latency. One criterion that is difficult to achieve is the false positive rate, which corresponds to the number of times the wake word is incorrectly triggered. Assuming the user wants to generate fewer than three false positives per day, and given that the speech audio stream is classified every 10 ms, a false positive rate of 34.10-6% must be achieved, which means a true negative rate of 99.99996%. Obtaining this performance with ML is challenging because ML is often used for resolving an approximately unsolvable problem. This is the overall concept of the receiving operating curve (ROC) trade-off. Almost a perfect classifier is needed (see Figure 9.3) for rejecting all noises and nontarget speech.

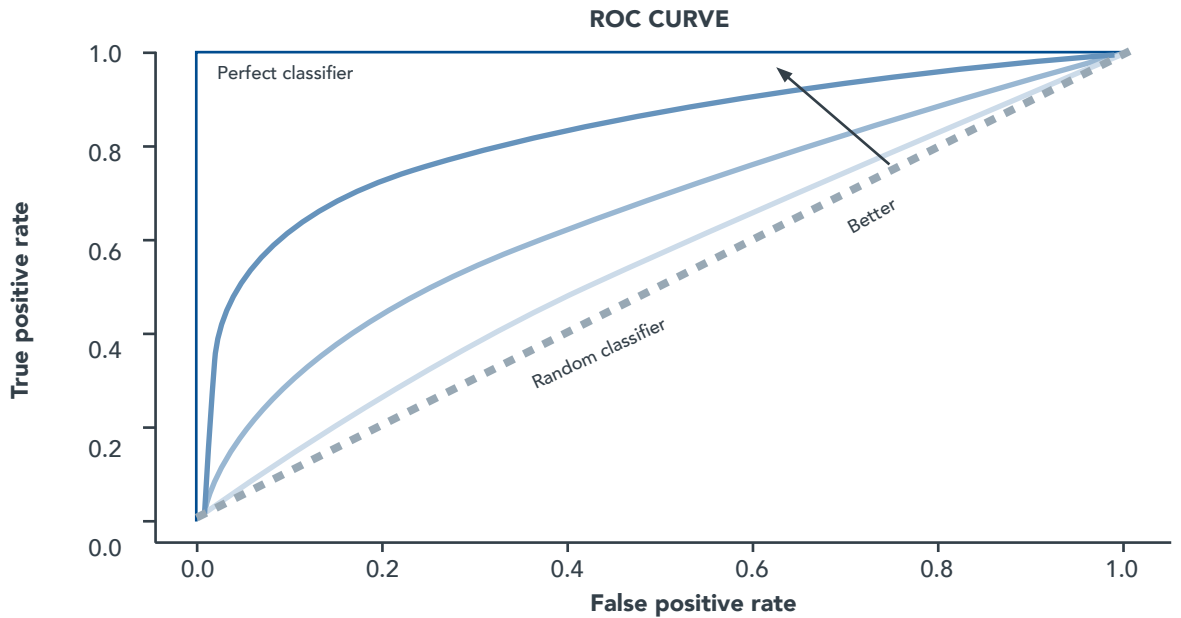


Figure 9.3. Receiving operating curve

Developing a wake word and voice command with ML

The next section examines data feature extraction, data augmentation and data visualization.



Feature extraction

The first processing step is feature extraction. Presenting time-domain data to the neural network is unusual despite successful approaches to it¹⁴. This is because the time domain is a complex representation (gain and phase completely change the data values while the human perception is almost identical). Moving from time to frequency domain and using the mel-frequency cepstral coefficients (MFCCs) to reduce the data vector size are more practical^{15,16}. Even if training a neural network with MFCC representation is possible, transforming MFCCs into another latent space for representing a higher level of speech such as phonemes is a better choice (see Figure 9.4). Indeed, having a neural network that can extract all the phonemes of one specific language guarantees that the user can detect any word and any sentence.

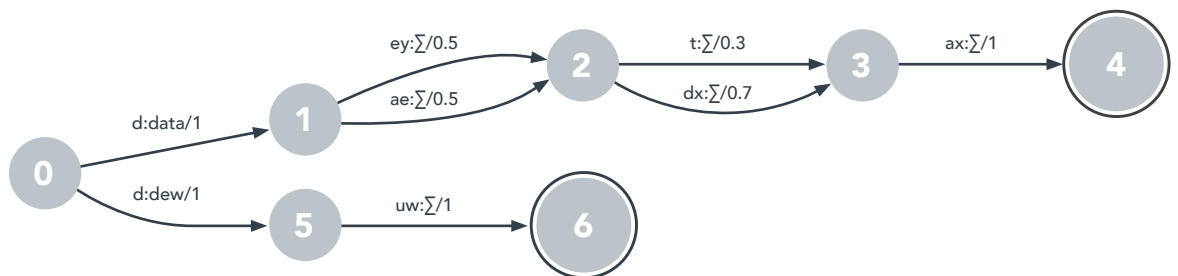


Figure 9.4. Weighted finite-state transducers for pause, play and stop

¹⁴Reference: Conv-TasNet: Surpassing Ideal Time-Frequency Magnitude Masking for Speech Separation, Yi Luo, Nima Mesgarani

¹⁵Reference: EFFICIENT KEYWORD SPOTTING USING DILATED CONVOLUTIONS AND GATING, Alice Coucke, Mohammed Chlieh, Thibault Gisselbrecht, David Leroy, Mathieu Poumeyrol, Thibaut Lavril - Snips, Paris, France

¹⁶Reference: <https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html>



Data augmentation

The false positive rate is a difficult metric to reach, and the true positive rate (how well a device can detect the wake word) is also challenging. As usual in ML, the dataset preparation before neural network training is the most important step. To generate the best performance, as much variability as possible must be shown to the neural network classifier so that it recognizes speech diversity (tones, pitch, speed). This process, called data augmentation, consists of presenting original training data with some modifications (noise, speed, reverberation).



Data visualization

Before starting the neural network training, the user needs to verify that the features presented to the neural network correctly address the problem to solve. The features need to not only accurately identify the target class but also separate the target class from the nontarget classes.

Because data features vectors are high dimension, some algorithms, such as principal component analysis (PCA), can be used to reduce the dimensionality. In Figure 9.5a, all the target “Alexa” wake word feature vectors are well clustered. Figure 9.5b incorporates nontarget speech and noise to verify that data features are correctly separating the two classes.

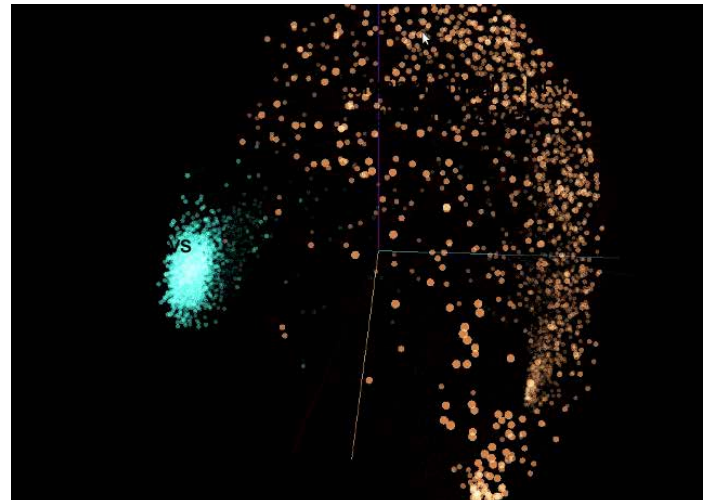
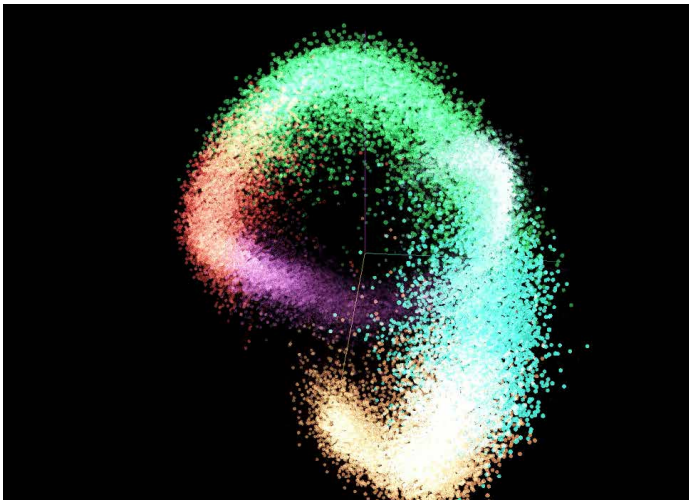


Figure 9.5a. PCA of 1,000 samples of the target wake word

Figure 9.5b. PCA of the target (light blue) and nontarget samples (brown)

The dataset is now ready to be used for ML training.

This section mainly focused on data. For the wake word, and in deep learning in general, the algorithm can never be blamed — only the data. How the data is augmented and presented to the neural network is key to achieving good performances.

Edge processing expertise in a variety of areas — embedded signal processing, ML on the edge and embedded firmware — is required for the successful deployment of this example. Because of this, achieving good performance for local voice products is difficult.

EXAMPLE: 5G TECHNOLOGY AS AN ENABLER FOR INDUSTRY 4.0

In many ways, industrial control applications are prime examples of edge computing because of their data locality, reliability and latency requirements. Data locality requires that the data stays on the physical premises of the industrial setting (for example, a factory) because data (or even meta data) is considered a key asset to the corporation. The importance of tight latency requirements is easily understood when considering that the settling time of a control loop (for example, to move a robotic arm) is directly related to the latency of the feedback loop. Reliability requirements are obvious; the cost of downtime is huge.

A modern industrial production domain is complex because of the large amount of coordination required between devices. This drives requirements for coordination and timing alignment between devices and, ideally, for some form of centralized management and control. These requirements, in turn, generated the need for wired or wireless communication in the edge computing environment.

Today's industrial networking environments incorporate local production domains that are predominantly connected through wired Ethernet networks using TSN capable of end-to-end latency in the tens of microseconds. This allows support for centralized operational control and coordination (see Figure 9.6). Wireless industrial Wi-Fi options are mostly used for links with limited time accuracy and reliability requirements.

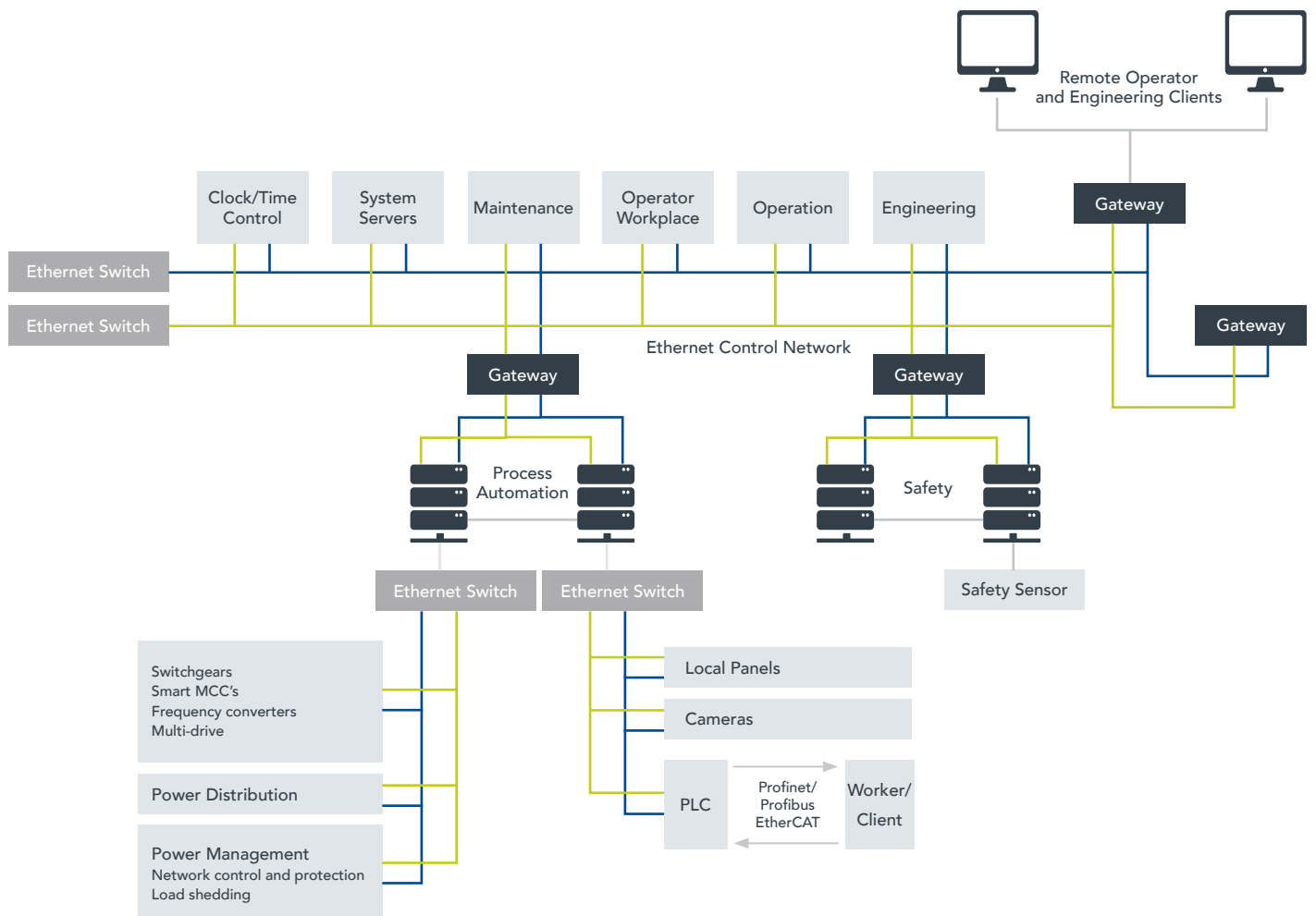


Figure 9.6. Traditional (wired) TSN industrial connectivity

Wired connection concepts don't align with Industry 4.0 concepts involving the use of smart technology and large-scale machine-to-machine (M2M) communication to enable flexible industrial environments with limited human intervention. Consider an example of a modern warehouse with unmanned guided vehicles (UGVs) or a modern assembly line that is no longer linear but routes work between cells as necessary. Both require low-latency, high-reliability connectivity as promised by 5G private networks.

5G replaces the connectivity between production domains and to/from centralized control with a wireless (5G) connection, thus enabling deployment flexibility, mobile connectivity and edge computing.

5G in industrial applications is enabled by several factors:

- **3GPP standards including ultra-reliable low-latency communication (URLLC)**
- **Deployment options such as edge computing and software-defined networking (SDN)**
- **Spectral availability supporting private network deployment**

3GPP standards

3GPP defines multiple deployment scenarios to guide requirements and protocol specifications in TS22.261, as shown in Table 9.1:

Scenario	End-to-End Latency (ms)	Reliability
Discrete automation (motion control)	1	99.9999%
Electricity distribution — high voltage	5	99.9999%
Remote control	5	99.999%
Discrete automation	10	99.99%
Intelligent transport systems — infrastructure backhaul	10	99.9999%
Process automation — remote control	50	99.9999%
Process automation — monitoring	50	99.9%
Electricity distribution — medium voltage	25	99.9%

Table 9.1. URLLC use cases and requirements

The following key performance indicators, or KPIs (3GPP TR 38.913), are enabled by targeted URLLC 5G standard options:



Latency

Latency is defined as the (one-way) latency for delivery of an application layer packet/message from the radio protocol layer 2/3 service data unit (SDU) ingress point to the radio protocol layer 2/3 SDU egress point via the radio interface in both uplink and downlink directions, where neither device nor base station reception is restricted by discontinuous reception (DRX). For URLLC, the target for user plane latency should be 0.5 ms for both uplink (UL) and downlink (DL) transmission.



Reliability

Reliability can be evaluated by the success probability of transmitting X bytes within a certain delay, which is the time it takes to deliver a small data packet from the radio protocol layer 2/3 SDU ingress point to the radio protocol layer 2/3 SDU egress point of the radio interface at a certain channel quality (for example, coverage-edge). A general URLLC reliability requirement for one transmission of a packet is 10^{-5} to 10^{-6} for 32 bytes with a user plane latency of 1 ms.



Availability

This is the maximum coupling loss (MaxCL) in upload (UL) and download (DL) between the device and base station site (antenna connector(s)) for a data rate of 160 bps, where the data rate is observed at the egress/ingress point of the radio protocol stack in UL and DL. The target for coverage should be 164 dB. Link budget and/or link level analysis is used as the evaluation methodology.

Note that the performance-critical scenario involves discrete automation, which demands a 1 ms latency and 99.9999% reliability.

Building blocks to achieve these three KPIs include numerology and frame structure, fast turnaround, efficient control and data resource sharing, grant-free uplink transmission and advanced channel coding schemes.

Industrial use support is not limited to building blocks. More capabilities are expected to be standardized and developed to support specific market verticals. In this chapter, we try to outline techniques in a general way.

Deployment options

Industrial automation requires the use of private networks that are local to the premises. Driving requirements for this include:



Data locality

Production data is not allowed to leave the industrial premises for data protection reasons. This removes the option of (remote) cloud processing and reliance on public networks.



Latency

Public radio access network (RAN) latency is an order of magnitude higher than private network latency even if public networks are optimized toward enhanced Mobile Broadband (eMBB) deployment and centralized compute. In an optimal (lightly loaded) environment, public network RAN latency alone is a 10 ms order of magnitude.



RAN performance

When relying on a public network, guaranteeing operational quality of service (QoS) in every location may be difficult. Small cells can be used to improve random access network (RAN) performance, but they intrude on the IT infrastructure in the industrial environment when they are part of a carrier network.

Cost and (small) scale targets for industrial deployment can be achieved by implementing software-defined networking for control plane and core network functions. This removes the need to rely on high-end networking equipment designed for use by mobile network operators.

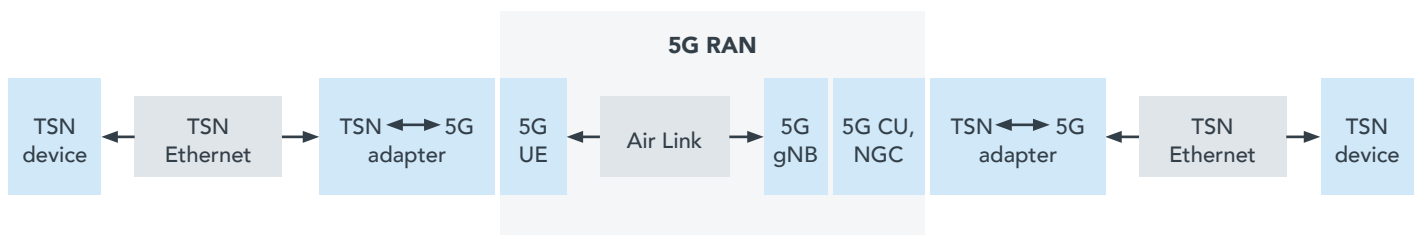


Figure 9.7. Combining 5G RAN and TSN

The 5G RAN itself can be physically implemented/partitioned in different ways. For example:

- Integrated small cells where RF, the physical layer and 3GPP radio link control/medium access control (RLC/MAC) layers are co-located into a single (“small cell”) unit, connecting over sub 1 Gbps links to the centralized unit that takes care of 3GPP protocol termination, cell site routing and connectivity to edge compute/core network.
- Distributed small cells where RF and the lower physical layer are separated from the upper physical layer and RLC/MAC functionality. In this case, connectivity between the two is provided by higher capacity Ethernet, typically dedicated 10/25 GbE.

These two implementation options present trade-offs in cost, power, programmability/flexibility and other factors. Specific URLLC protocol-level features can be implemented more easily in an integrated small cell environment (smaller slot times and low hybrid automatic repeat request, or HARQ, turnaround times) or in a distributed small cell (coordinated multipoint and interference management). The choice between implementation options depends on the use case.

A typical 5G deployment has one 5G user equipment/customer premises equipment per machine or station. It relies on wired networking to enable lower latency control loops for motors, drivers and actuators. This is where TSN comes in. TSN includes several features that map to 5G to provide deterministic communications (see Figure 9.7). It can reach control loops with a period of around 100 μ s. All relevant components can be enabled using hardware as shown in Figure 9.8.

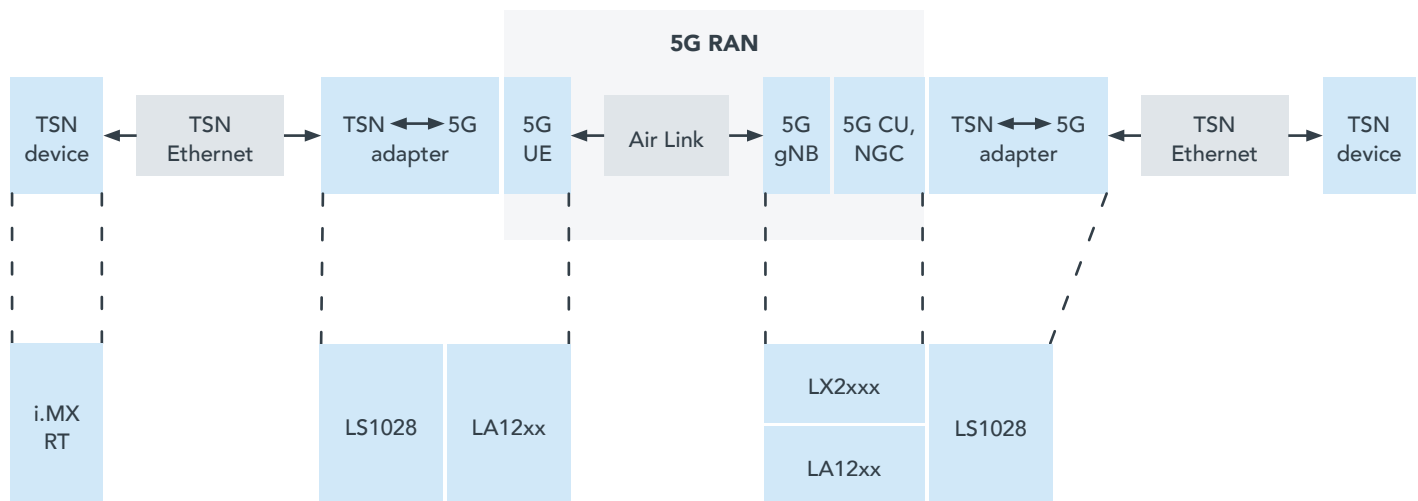


Figure 9.8. A typical 5G deployment enabled with NXP components

Spectral availability

The deployment of private 5G networks, including those inside an industrial facility, requires available spectrum. Spectrum availability typically defines the deployment format. Spectrum allocation is not distributed evenly around the globe, so solutions are focused in regions, for example, the United States, Germany and Japan.

United States — The Federal Communications Commission is opening 150 MHz of Citizens Broadband Radio Service (CBRS) spectrum in the 3.5 GHz (3550–3700 MHz) band. This spectrum is intended for three tiers of users (see Figure 9.9):

- **Incumbent (orange)** — These users can retain the rights to use the band for military and wireless internet service provider (WISP) purposes. WISPs will continue to have incumbent access to the 3650–3700 MHz band under the terms in place, with no modifications needed to deployed equipment.
- **Priority Access License or PAL (yellow)** — These users are allocated in the 3550–3650 MHz spectrum. Spectrum auction is implemented on a per-county basis (~3,200 in the U.S.). Each market has 7X 10 MHz TDD PALs, with a maximum of four channels assigned per licensee. PALs will mostly drive deployments of public networks.

- General Authorized Access or GAA (green)** — All other users can register with a Spectrum Access System (SAS) when the SAS determines the spectrum is not in use by incumbent or PAL users with higher access priority. Channels throughout the overall band (3550–3700 MHz) are available for GAA. Access to 70 MHz is shared with PAL users and subject to availability (PAL users have priority over GAA users), and access to the remaining 80 MHz is reserved for GAA users (but PAL users can also use this part of the band under the GAA provisions). GAA use requires spectrum sharing using coexistence techniques (see below). GAAs will mostly be used for private networks.

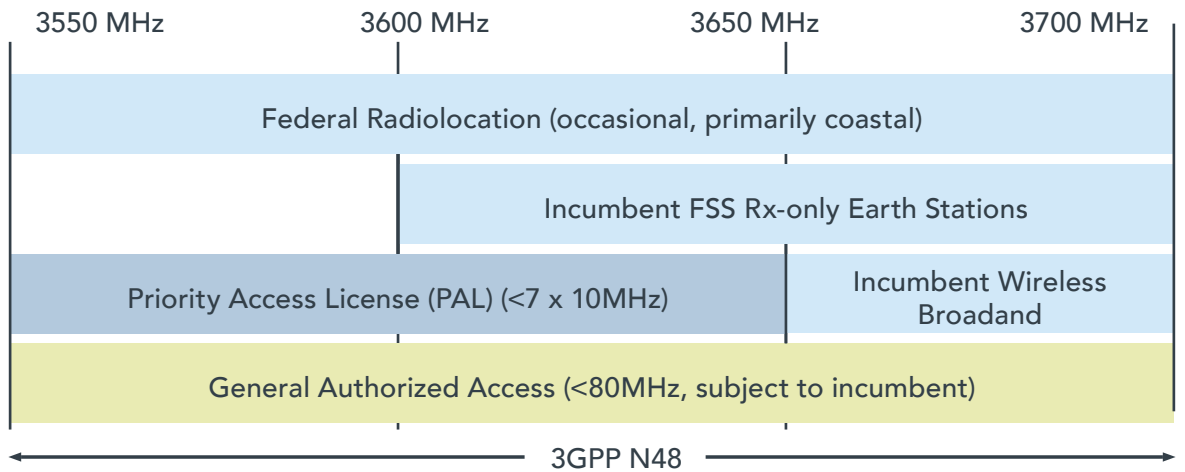


Figure 9.9. Spectrum allocation in the United States

This new regulation creates a deployment framework for 4G and 5G in this band, which was previously under-utilized.

Germany — European RF spectrum is defined on a per-country basis, with ongoing harmonization efforts. The Alliance for Connected Industries and Automation maintains a list. Germany allocated local licensed spectrum in the 3700–3800 MHz band range to industries for their applications in 2019. This spectrum is made available for private network use through a transparent and low-cost process by the Bundesnetzagentur (Federal Network Agency, or BNetzA). The BNetzA oversees the telecommunications industry in Germany (see Figure 9.10).

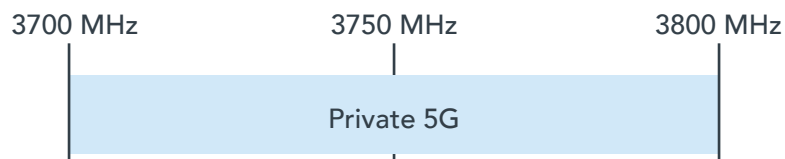


Figure 9.10. Spectrum allocation in Germany

Applications for spectrum are implemented in “chunks” of 10 MHz spectrum, with fees defined according to the following formula since December 2019. The fee = $1.000 + B * t * 5 * (6a1 + a2)$ where

- 1.000 is the base fee in €,
- B is the bandwidth in MHz (minimum 10 to maximum 100),
- t is the time of allocation in years (for example, 10 years),
- and a is the area in km²; differentiation between settlement and transport areas (a1) and other areas (a2) -> a1 applies to industrial use, and a2 applies to farming and forestry

Germany does not explicitly limit transmit power, but the regulations stipulate that the local network operators are responsible for a reasonable deployment that does not cause interference.

Note that in Germany, broadband fixed wireless access (BFWA) has been allocated to the 5.8 GHz spectrum (5755–5875 MHz) band and as such does not share the private network spectrum like it does in the United States.

Japan — Spectrum for local 5G was released in Japan at the end of 2019 for local government and enterprise use. The country’s goal is to realize “Society 5.0” through state-of-the-art technologies such as IoT, artificial intelligence (AI), robots and self-driving vehicles, as well as their incorporation into all industries and sectors related to lifestyle (see Figure 9.11).

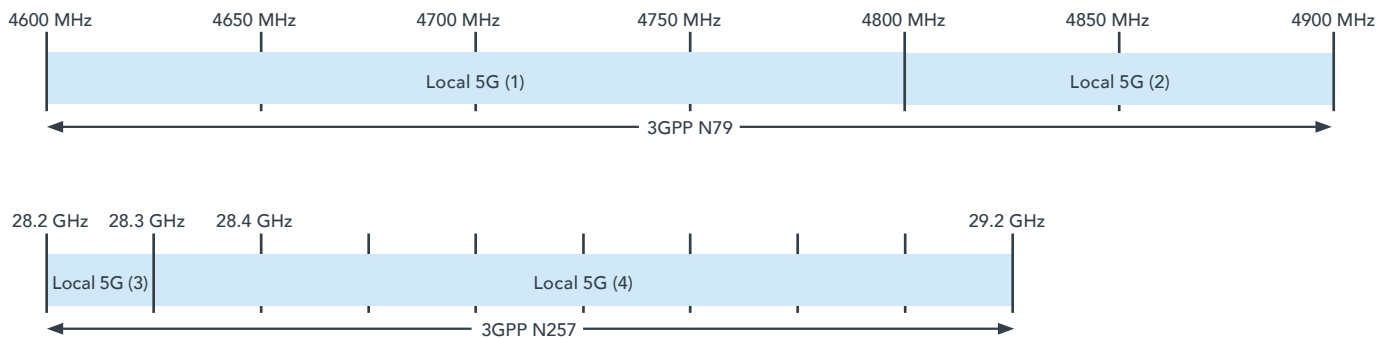


Figure 9.11. Spectrum allocation in Japan

A harmonized spectrum for private networks around the globe has not yet been defined. Implication, equipment choice, RF regulation and cost vary by region. Government harmonization efforts are underway but are not expected to solve this challenge in the short-term future.

This example summarizes the value proposition that 5G delivers to industrial applications and the specific options/capabilities that the 5G protocol supports to enable industrial wireless networking as a targeted use case. Elements beyond the protocol features make 5G applicable to industrial applications. First, spectrum is being made available worldwide specifically for innovative applications such as industrial networking. Secondly, hardware and software need to be available to make the required systems available to customers.

EXAMPLE: WEARABLES

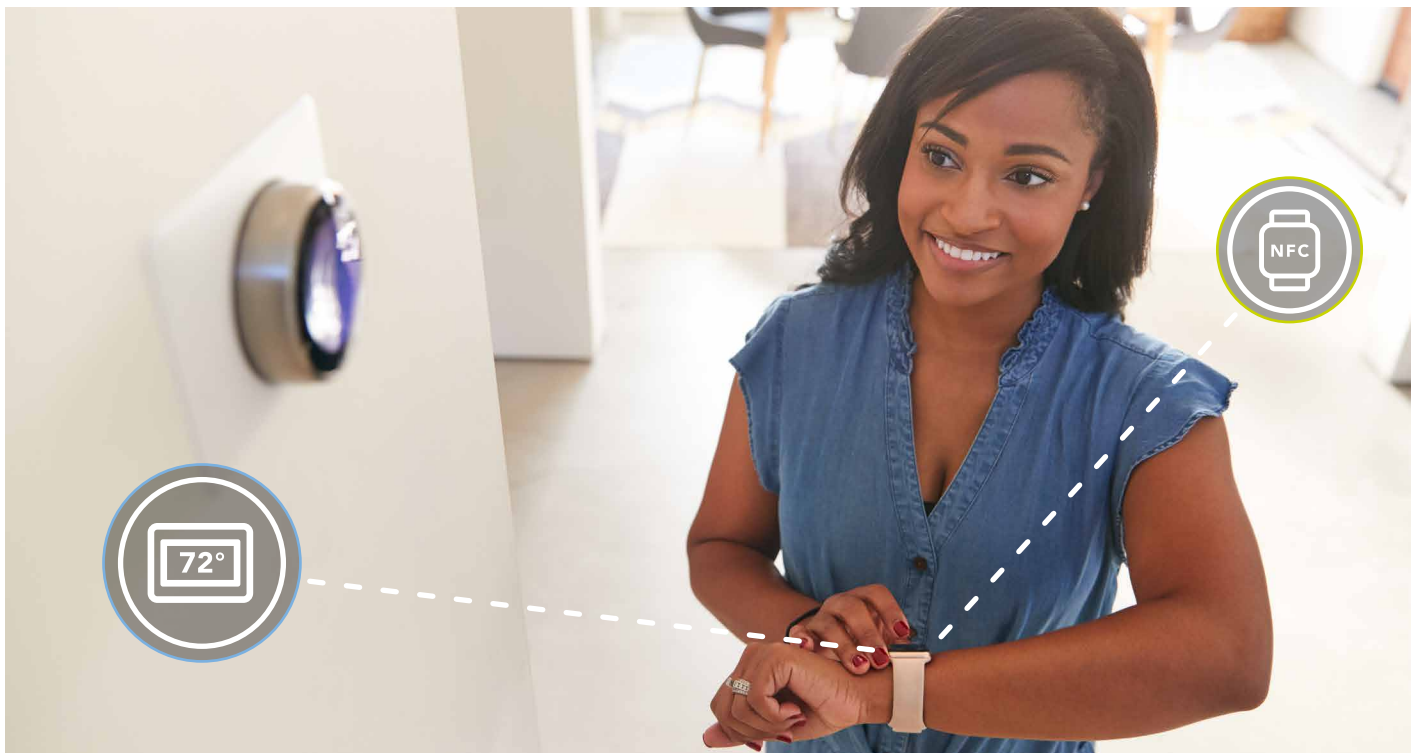
Wearables are expanding beyond the common activity tracking and fitness use cases found in nearly 1 billion connected devices today. The evolution of wearable technology can be attributed to the improvement of processor performance, sensors, wireless connectivity and tracking capabilities, contactless payment/access, extended battery life, and the demand for richer user experiences. These all create the ideal foundation for new health and well-being use cases.

Among edge processing applications for consumer IoT, the convergence of smart homes and smart watches (see Figure 9.12) technologies will gain momentum while delivering increased quality of life to consumers. The technology enabling the use of smart watches to control security systems, thermostats, TVs, coffee machines, lights, ovens, vacuum cleaners, dryers, washing machines, alarms, doorbells and so on can easily spread to the edge node.

The combination of smart watch technology and connected car tools are being introduced. New smart-watch-integrated apps can give users access to controls, such as the sunroof, seat ventilation and temperature, and alerts on speed and vehicle status, among other features. Advances in sensor technologies and software capabilities will make smart watches compatible with more devices in the coming years.

The following requirements are driving the trends in this market:

- Longer battery life
- Constant connection through Bluetooth Low Energy, Wi-Fi or Long Term Evolution (LTE)
- Intuitive user experience with smartphone-like display (organic light emitting diode) and exciting graphics and animations
- Expanding features such as voice control, greater audio, location/positioning and numerous sensors
- Reduced form factor that provides more functionality in less space
- Increased comfort and health
- Interoperability with other connected devices



Wearables at the edge

As virtual reality and augmented reality glasses and headsets take users to other worlds for applications including gaming, travel, education and training, health-related wearables are even more ubiquitous. These devices generate a lot of data that require fast and secure processing, which edge computing provides.

Sensing, processing and acting on the edge ensure data is collected and analyzed instantaneously. For simpler use cases, the need for broadband connection and cloud processing is eliminated to reduce power and ensure value to the end user.

Implantables, fitness trackers, smart jewelry, watches, shoes and clothing provide the following:

- Health monitoring, with movement and fall detection, emergency calling and vitals reporting (for example, body temperature, oxygen levels and cardiac signs)
- Sleep monitoring and therapy
- Meal/calorie tracking
- Hearing enhancement
- Social distancing and contact tracing and tracking

These technologies can be life-saving, especially for cognitively impaired and high-risk patients. Edge computing offers the fast processing and security needed to detect, interpret and protect crucial and private health data.

Energy consumption, connectivity, human machine interfaces (HMIs), graphics processing and security are optimized by enablement. Energy consumption is optimized by adjusting the system clocks and voltage levels on various power rails. It selects the appropriate power mode (run, wait, sleep, deep sleep) based on use cases. Software connectivity is enabled through Bluetooth LE, near-field communication (NFC) and Wi-Fi stacks. The HMI is enhanced through the graphics processing unit (GPU) software enablement. Security is enabled by secure boot, authentication and device management running in the secure software domain. Software processing information ranges from touch sensing to event triggering. Algorithms extract and analyze data from GPS and ultra-wideband (UWB) to establish positioning.

Consider these smart watch uses cases and associated technologies.



Localization

UWB technologies can help determine the location of a person with limited mobility who may have fallen. This technology also provides accurate localization to position a user in relation to a door lock. With a secure element, the system can identify the person standing there, and UWB can detect the distance for the door to operate more seamlessly.

Understanding the position of the user can trigger services to optimize travel and return to the point of origin. Localization technology also monitors user activity to encourage the user to do more or less to stay healthy. UWB provides indoor positioning. For example, homeowners leave, they can power up the GNSS to continue helping with positioning, and they can turn off the same GNSS to save power while UWB is active.

A small IoT device on a pet, powered by some form of energy harvesting and UWB, can help identify the position of the pet at any time and even track the pet's activity.



Voice control

The voice control capabilities in many wearable devices leverage a variety of AI technologies — speech recognition, natural language processing (NLP), context aware computing and ML — to enable human-like voice interaction. Users can access news, send messages, track exercise, play music and make flight reservations using these platforms without having to access apps manually. To save power and extend battery life, local voice technologies are now available. These implement processing at the edge without connecting to the cloud or processing data in the cloud. The size of these algorithms is much smaller than the cloud-based versions. Voice control technologies are based on three main pillars: 1) The microphone/ audio front panel that reduces environmental noise to provide a clearer voice signal to the 2) wake word engine, which indicates to the watch to listen to the owner saying a 3) command word to control a function of the watch. All of this must happen in record time with a high level of confidence to recognize the right command words and reject the wrong command words. That process is accelerated by a DSP on a chip in some edge processors to optimize performance and power use.



Security

Many smart devices connect to systems that exchange data with phones and the cloud, so they are susceptible to security attacks. Anyone could potentially hack a home or building system to trigger costly damages. The high number of security incidents demonstrates the need for robust security software at all layers as well as effective physical security.

Silicon vendors must offer customers the ability to securely boot and trust the embedded software and to store data on the device. The processor offers a secure boot with immutable hardware root of trust (RoT), unique key storage based on a static random access memory (SRAM) physically unclonable function (PUF), acceleration for symmetric (AES-256 and SHA2-256) and asymmetric (elliptic curve cryptography and Rivest-Shamir-Adleman, or RSA) cryptography and an optional fuse-based root key storage mechanism. Additionally, virtualization can be used to protect embedded systems from software attacks after updates, and ciphering technologies can be implemented to secure communication links between devices and servers.



Graphics

The smart home and smart watch markets have different target users, yet both require the ability to design tailored graphical user interfaces (GUIs). Today's kitchens or living rooms change more often than in the past while integrating appealing new electronic products with large displays, for example. Buildings are becoming more connected and taking advantage of complex electronics thanks to IoT. Maintenance is easier and configurations are error free through efficient UIs.

To satisfy the market's high expectations, silicon vendors provide GPUs that must be fully exploited by their customers through a set of ecosystem partners that provide GUIs. The GPU is a type of processing technology used to make GUIs as appealing as a smartphone UI by adapting to display size, color depth, layering and so on.



Battery life

The capacity of the lithium-ion batteries in smart watches, for example, is a few hundred milliamp hours (mAh). Battery size is restricted by the limited physical space on these watches. Today, edge processors are optimized to achieve the lowest possible power consumption at the required performance levels. Specifically, they offer a variety of reduced power modes and incorporate various low-power design techniques to enable long battery life in both active and sleep modes.

Consider the following power mode examples (implementations and definitions may vary from one vendor to another):

- In sleep mode, the CPU clock is stopped. All peripherals and memories enabled while the CPU is active are still functional. Any interrupt may resume execution.
- In deep sleep mode, the CPU clock is stopped. Nearly all peripherals and memories can be turned off via software. Memory partitions can be individually turned off or retained. Wake-up sources can be selected by software before entering deep sleep mode.
- In deep power down and full deep power down mode, power and clocks are shut off to the entire chip with the exception of the real-time clock (RTC), leaving no configuration options or memory retention.

One example is how a wearable uses FreeRTOS as its real-time OS. The different low-power modes are integrated in the "tickless" feature of FreeRTOS. Tickless is the mode an OS enters when it has nothing to schedule. It calls an implementation-specific function that stops the tick timer for a given amount of time (based on next task scheduler timing) and allows the chip to reduce its power consumption.

From a high-level point of view, this tickless function:

- Stops the tick timer (providing ticks to FreeRTOS).
- Selects the sleep mode and its configuration based on the expected sleep time provided by the OS and the peripherals and memories in use.
- Starts the RTC timer and configures it as a wake-up source.
- Enters the selected sleep mode.
- When waking up, provides the number of ticks skipped during the sleep back to the OS.

Deep power down mode can also be used. In that case, data that should be kept must be backed up in RAM. Data can then be retried going out of that power mode. Trade-offs need to be made between the power saved during deep power down phases and the extra power consumed for the backup/restore functions.

The wearable, for example, reduces power by remaining in sleep mode as long as possible and refreshing the display every second by waking up the chip. Figure 9.13 shows the power on the core and memories supply (voltage at drain or Vdd) in this use case.

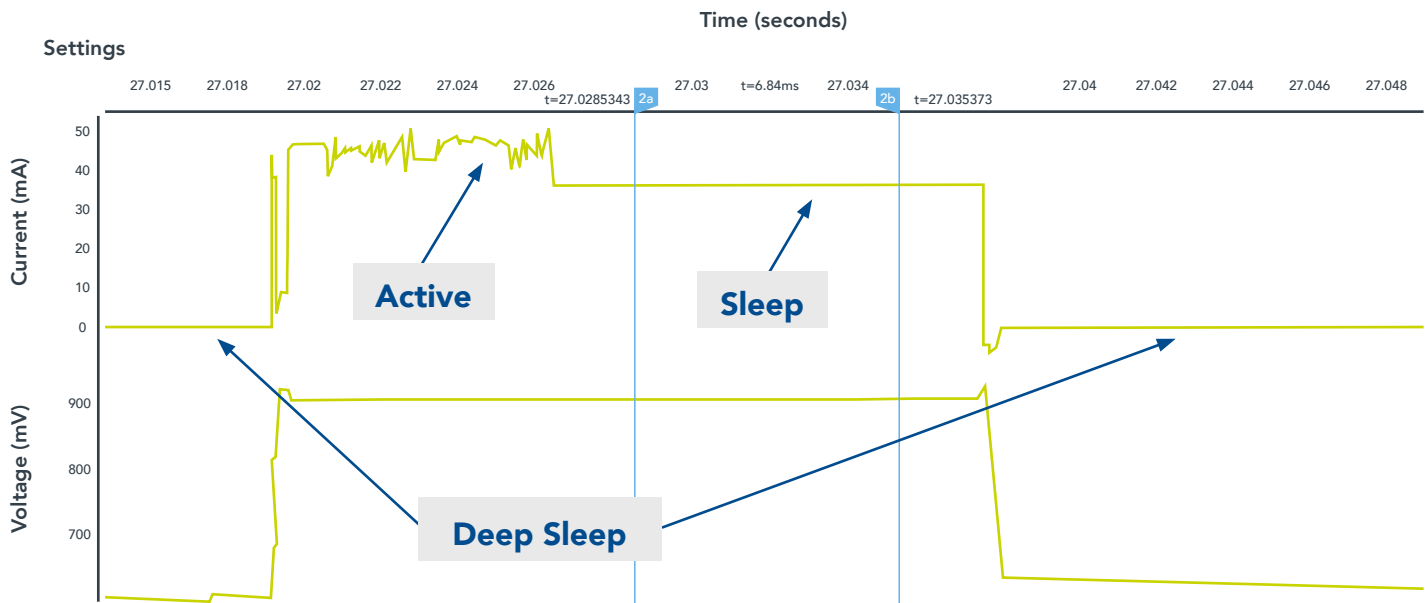
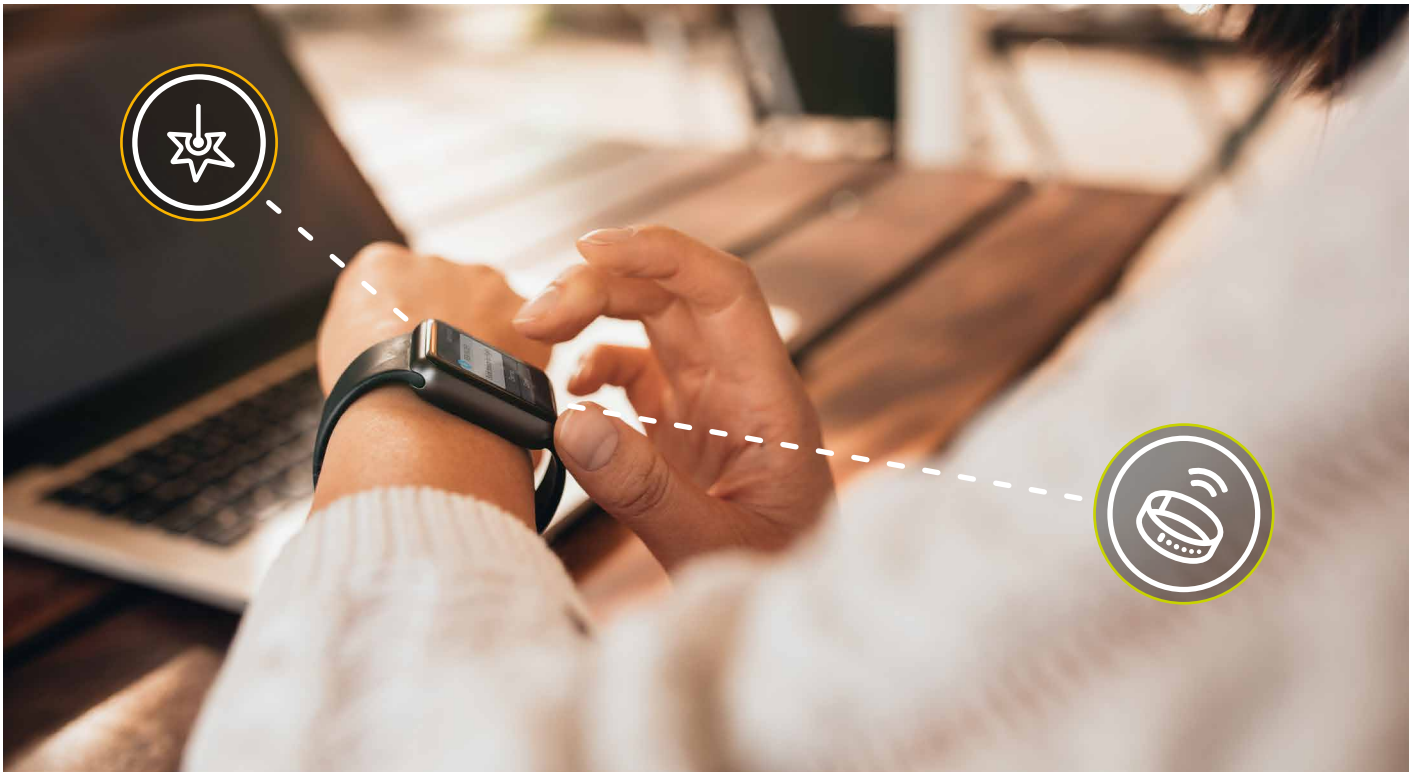


Figure 9.13. Smart watch power supply during wake-up and deep sleep mode



Motion

Wearable sensors can be worn by a person performing their daily activities. Sensors such as accelerometers, microphones, barometers and GPS sensors can record a person's physical condition, location change and speed. Because of these sensors, smart watches worn on the wrist can run activity-recognition applications. For those, the deterministic approach is inaccurate because of the large number of variables and environmental parameters (user typology, age, body language, behavior, context, etc.) and the range of devices and ML methods used in determining/measuring human activities. Different human activities can be classified through advanced ML methods by using accelerometer and step counter sensors in the smart watch. In addition, a reliable optical heart rate monitor embedded on the back of the watch compensates for motion artifacts by using the accelerometer data.

The accelerometer is a key sensor for applications that provide step count data, which is the main inputs for any advanced classification algorithm. In addition, low power and battery lifetime are critical parameters to observe. Therefore, an accelerometer with an embedded low-power step-counting function can improve the system power by avoiding computations in the host MCU. Indeed, some silicon vendors are proposing an interesting internal function called vector magnitude with related interrupt capabilities. This function, fully hardwired in the internal state machine, does not increase the intrinsic power consumption of the sensor (typically a few microamps at a 50 Hz sampling frequency, which is a good trade-off frequency for step count). The beauty of this function is that — with the right settings — it can provide an ultra-low power step count. Each step triggers an interrupt output that is incremented in the low-power register of the MCU. The host MCU can be in idle mode most of the time. The interrupt associated with the step count event simply needs to increment a counter; therefore, the host MCU contribution to the power budget is minimized.

A counter IP (hardware block) may be incremented autonomously with an external clock/line without even waking up the MCU. Step detection is solely based on detecting the impact of a step without considering the user's height, weight or gender. A classic way to proceed is to calculate the acceleration vector magnitude: $VM = \text{SQR}(X^2+Y^2+Z^2)$ where X, Y and Z represent a single accelerometer reading, normalized at 1 g. When the user is still, the value should be close to 1 g. Then the key two parameters for optimization are the vector magnitude threshold and the vector magnitude count, which is the number of consecutive samples exceeding the threshold for the "step event" to be raised.

In addition, the sensor for wake-up can be used for motion detection purposes. This configuration, offers the flexibility of the significant data change detection, which provides multiple options for absolute or relative wake-up. With the right settings, a simple motion or a more specific movement (such as wrist rotation) can wake up the device with a minimal power budget. Sample rates of 25 Hz are no longer needed, but using a lower sampling frequency around 1 Hz (0.78 Hz, specifically) triggers an acceptable response time for human use. In that configuration, the wake-up current consumption typically can drop to 600 nA, so the MCU sleep current and other application leakage should be added. The accelerometer can manage an automated wake-up switching from low sampling frequency to 50 Hz after a wake-up event is detected.



Wireless power transfer

Wireless charging works through contactless means. Instead of connecting a cable to a device to transfer power, wireless charging allows a device to receive a charge without plugging in anything. The most popular of the many wireless power transfer methods is called inductive charging. In this process, two coils are placed close to each other and are magnetically coupled. This allows energy to be transferred from one coil to the other. This technology is supported by industry standards, the most widely adopted being WPC Qi.

Wireless charging works well with NFC technology to identify a nearby object.

When an optimized hardware and software platform is used, the transmitter ICs implement and control the power transfer function. They also monitor and manage overall system states, including foreign object detection, temperature and system efficiency. With power transfer efficiency exceeding 75%, the transmitter offers digital demodulation for lower bill-of-material cost, algorithms for foreign object detection and low active and standby power modes.

The evolution of smart cities is, in many ways, directly tied to the evolution of contactless solutions. First deployed in 1994, NFC ICs were originally developed for automated fare collection in public transportation, but that was just the beginning. Since then, these ICs have enabled contactless transit, payment and access experiences for people no matter the location or time.

Smart city services can be deployed to NFC-enabled wearables through a cloud service. For example, MIFARE 2GO manages digitized MIFARE product-based credentials, and it can enable contactless access and micropayments using NFC-enabled wearables.

Part of a contactless solution's appeal is that it offers something for everyone. Government agencies, businesses and service operators use it to lower costs, add flexibility, increase control and deliver better consumer interactions. At the same time, these products, on the edge, help people feel safe and secure and contribute to sustainable prosperity.

By applying true multi-application functionality, mobile formats and certified Common Criteria security, these ICs combine convenience with efficiency. Using these products means choosing a solution that is already available in a widespread infrastructure, which reduces startup costs and increases scalability. Reader ICs on the edge can interact with NFC technology, and communicating through NFC means that reader-based products also can be managed and implemented via NFC-enabled mobile devices, including smart watches.



Data, analytics and ethics

Smart watches generate lots of data that requires storage and real-time analysis. They have limited storage and processing capabilities, so apps can synchronize user data on their respective vendors' cloud servers. By analyzing the server-stored data, vendors can improve their products, services, content and advertising. However, with growing privacy and cybersecurity concerns regarding the exchange of data between smart watches and cloud platforms, vendors are increasingly providing on-device computation (edge computing). GlobalData expects more edge computation on smart watches in the coming years, but vendors will still need to transfer some user data to cloud servers for broader analysis and research purposes. Finally, this leads to questions about the responsibility of the edge device and its hardware and software suppliers to comply with local authorities' ethics guidelines.

EXAMPLE: TSN AND DISTRIBUTED REAL-TIME COMPUTING AT THE EDGE

Now consider a practical example for industrial automation using distributed motor control over a time-sensitive network.

Computing at the edge is not limited to stand-alone processing on a single node. Portions of edge applications are suited for processing data locally to the edge node, and some applications require several edge nodes to act or sense in coordination and in real-time conditions. In these cases, a local network is created at the edge to aggregate edge nodes and form a distributed edge element that performs edge computing as a single macro component within timed constraints.

The industrial domain includes the systems used to make materials into products, real-time embedded systems for process control, workflow management and process monitoring. A factory may use an Industrial Ethernet technology that adapts standard Ethernet to deliver real-time response and work with legacy industrial communication protocols. Unfortunately, the many Industrial Ethernet protocols interoperate neither with each other nor with standard Ethernet, limiting the economies of scale for technology suppliers and thus slowing innovation. A single machine in a factory may connect to different Industrial Ethernet networks, each running its specific protocol, for different control functions, as Figure 9.14 shows. The manufacturer must deploy gateways to pass data among the different networks or systems.

With TSN technology, edge micro nodes can be aggregated in a single macro element that can be managed with upper layer protocols as a single entity in the edge infrastructure. This can take place in the diverse IoT fields including factory, smart grid and building, home and health — anywhere several nodes are required to operate in coordination in real time to achieve a common goal at the edge.

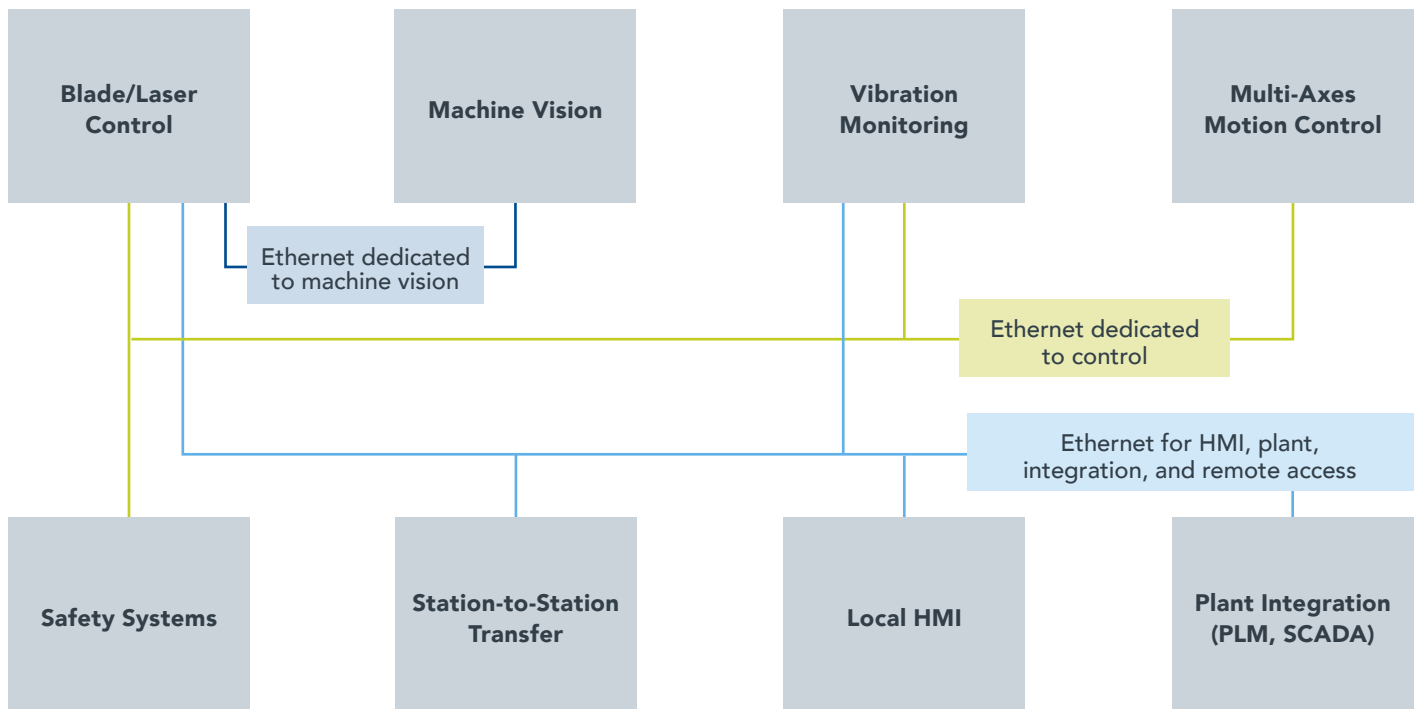


Figure 9.14 Modern machine networks and systems (Source AVNU)

TSN is based on the IEEE 802.3 Ethernet wired network infrastructure, and it can operate on wireless networks because of recent improvements made to the Wi-Fi and 5G standards for achieving precise time synchronization.

TSN standards

TSN, also called deterministic Ethernet, delivers real-time transport capabilities to Ethernet networks, which were originally designed as a best-effort communication method with no guarantee against transit delay. Several years ago, industry players who needed real-time networking capabilities in fieldbus systems overcame this limitation by defining a variety of proprietary protocols for Ethernet-based communications that are widely deployed today: EtherCAT (Beckhoff), PROFINET (Siemens), CC-Link (Mitsubishi) and others. As the IoT intensifies, TSN is emerging as an open and new standard defined by IEEE® to unify the technologies used at layers 1 and 2 of the Open Systems Interconnection (OSI) model for serving data transport in distributed real-time applications. Just as everything in the IoT world has become connected, TSN and its generic nature have been applied to many domains, including industrial, automotive and consumer systems, which ensures a common technology facilitates interoperability.

TSN is described in a set of IEEE standards, several of which are being incorporated into the more general IEEE Std 802.1Q™ that defines MAC services and operating principles for bridged networks. TSN improves the timing and synchronization standard by adding packet transmission schemes (time-aware shaper and frame preemption) to reduce inaccuracy in transmission time, increase the capacity to identify stream flows, and improve transport robustness through redundancy techniques.

Distributed real-time edge computing

A real-time application in a centralized system imposes time constraints on the whole system that hosts the application. Peripherals that interact with that application are locally interfaced and managed through I/O transactions. In a distributed architecture, peripherals and I/O are delocalized and spread over several remote nodes. A typical distributed application architecture splits the centralized application into two logical parts, one that manages local control of peripherals and defines a device entity while the other manages centralized intelligence and defines a controller entity. A controller may control one or several devices through the TSN network. Sophisticated and complex calculation is delegated to the controller, leaving the devices to concentrate on computations specific to the I/O transactions they manage. This also allows the use of processors with less processing capacity.



Real-time applications in the network — A real-time application relies on some time bases and time constraints. Moving to a distributed architecture spreads these time bases and constraints over the end nodes participating in the application and, as a result, on the network that connects those end nodes.

Real-time applications may operate in a cyclic or sporadic manner. A cyclic application is characterized by a process repeating over a defined time period. This is common in control and automation. A sporadic application, such as alarming, is event driven. Both types of real-time applications require different types of transport service quality from the network.

Cyclic applications use a TSN network to control critical traffic by synchronizing the application cycle with the network cycle. This is called an “isochronous” application. The TSN IEEE Std 802.1Qbv™ Enhancements for Scheduled Traffic standard serves isochronous applications, but sometimes the application and network cycles cannot be synchronized, typically when the application and network do not operate in the same time domain. One example is a factory where the whole plant is clocked by a universal time, but some actions in subdomains of the factory have their own working clock. In this case, the application is called “non-isochronous” because it requires transmission services from the network that are not aligned with network cycles. The TSN IEEE Std 802.1Qbu™ Frame Pre-emption standard serves non-isochronous applications.



Real-time applications in the end node — Real-time constraints for each local end node remain in the typical area of centralized configurations. The system must serve each software layer involved in real-time processing in a timely manner, from the low-layer drivers dealing with peripheral I/O transactions to the application layers that process those data to the crossing middleware layers in between.

The system on chip (SoC) must prevent any congestion or bottleneck at the hardware level that could jeopardize real-time reactions or even cause the loss or corruption of critical data. Priorities must be ensured for the network and I/O drivers and related interrupt handlers involved in the real-time chain.

The latency of the upper-layer tasks must be minimal to avoid exceeding the application’s time limits; thus, an OS’s task pre-emption capability is integral. Native real-time operating systems (RTOSs) with the simplest architectures and less sophisticated services can be used to meet strict real-time applications constraints (<1 ms), while more generic OSs such as Linux can serve less time-critical applications.

In all cases the data processing path across layers must be minimized so that it reduces OS interventions and the amount of instruction cycles to be executed before reaching the application code that processes the critical data. Direct processing path and minimized memory copies are usual optimization techniques required by real-time applications.

A practical example: Distributed motor control over TSN

This section presents an industrial automation use case involving TSN for motor control in a distributed environment. In this example, a controller remotely controls I/O devices to synchronize the motion of motors. The TSN network connects the controller and controlled devices (followers) as well as supports the movement of both time-critical traffic and concurrent background traffic. This illustrates the move from a centralized application to a distributed architecture.

The setup (see Figure 9.15) combines endpoint nodes with switch nodes to form a complete TSN solution. Both types of unit incorporate microprocessors that comply with the IEEE Std 802.1AS™ (gPTP) Timing and Synchronization for Time-Sensitive Applications standard as well as the IEEE Std 802.1Qbv standard.

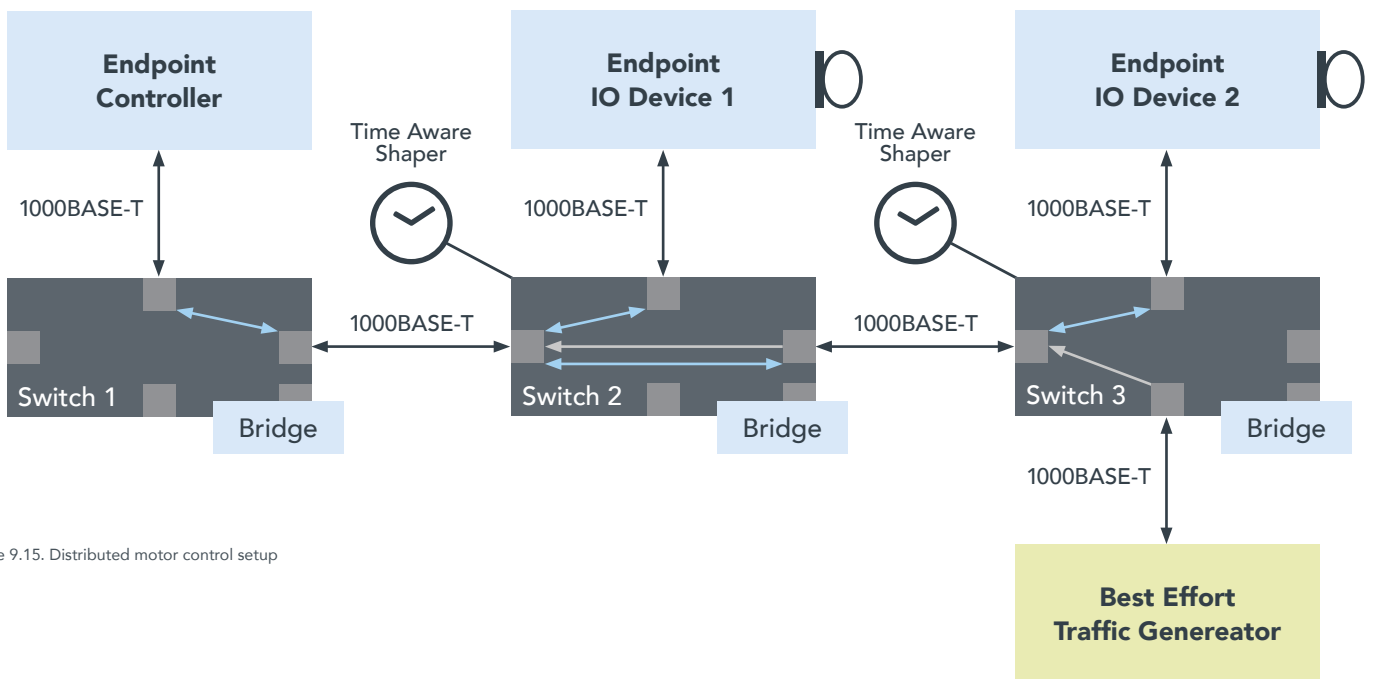


Figure 9.15. Distributed motor control setup

This setup characterizes TSN transport and real-time performance in terms of cycle time and associated jitter. Noncritical traffic in the background can affect performance stability, so the setup needs to prevent concurrent noncritical traffic from disturbing the required transport latency and accuracy of the critical traffic. This protects the stability of the motor control process.

Distributed real-time application — In a distributed environment, the motion control algorithm is divided into two parts: the follower device interfaced locally to the motor is running the current control loop at a 31.25 μs cycle time, and the controller is running the speed and position control loops at a 100 μs cycle time, exchanging information with the devices through TSN.

Real-time processing for motor control is performed on the endpoints in synchronization with the network cycle time, so this application is isochronous (see Figure 9.16).

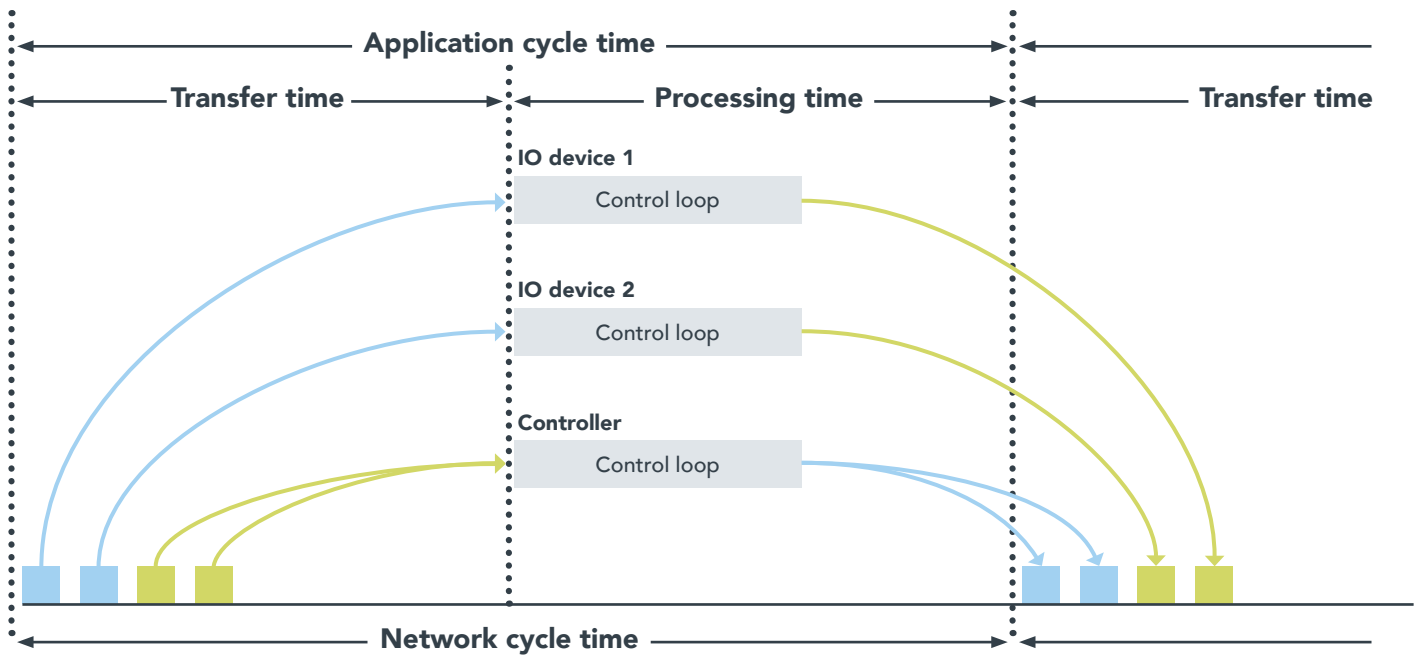


Figure 9.16. Industrial automation isochronous application

Motor control application network architecture — This architecture uses TSN to transport the data for controlling motors between the controller and several follower devices (see Figure 9.17). These exchanges, which represent the time-critical traffic, require short and bounded transport latency from the TSN network.

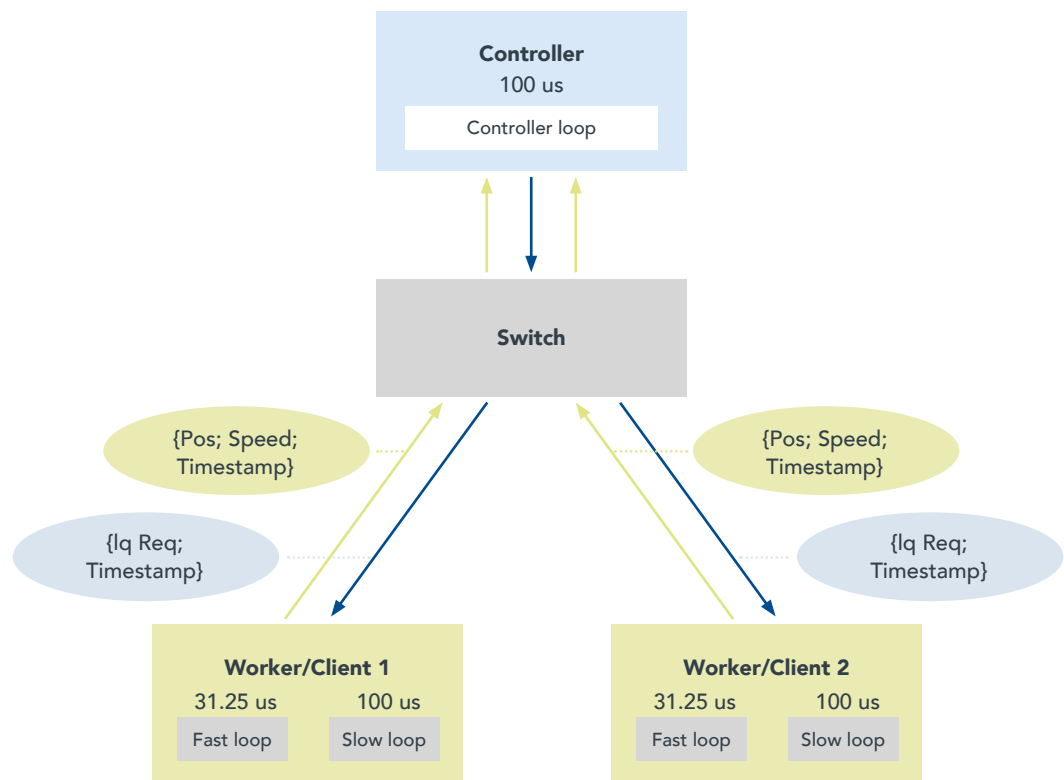


Figure 9.17. Distributed motor control application control loops



The endpoint application uses a proprietary protocol to transfer control data between the controller and follower devices. The controller sends a single multicast frame to propagate commands to the follower devices. In the opposite direction, followers use unicast transmission for the traffic directed to the controller.

Ethernet frames with virtual local area network (VLAN) headers prioritize the time-critical information over background traffic by using the priority code point (PCP) field of the VLAN header.

Each of the nodes participating in this distributed application uses the IEEE Std 802.1AS (gPTP) Timing and Synchronization for Time-Sensitive Applications standard to establish a common network time and synchronize its local clock to a common time reference. This forms the “working clock” used by both the network and application domains.

The TSN IEEE Std 802.1Qbv Enhancements for Scheduled Traffic standard ensures that the motor control traffic transmission is scheduled in a guaranteed and periodic time slot that is relative to a known timescale, i.e., the working clock established by the generic Precision Time Protocol (gPTP).

Motor control application distributed architecture — The controller hosts the “intelligent” part of the application. It incorporates algorithms that enable the motor to reach a given speed and position and ensures the follower devices remain synchronized. The controller processes the position and speed loops that output the Iq value, and the follower devices process the current loop locally (see Figure 9.18).

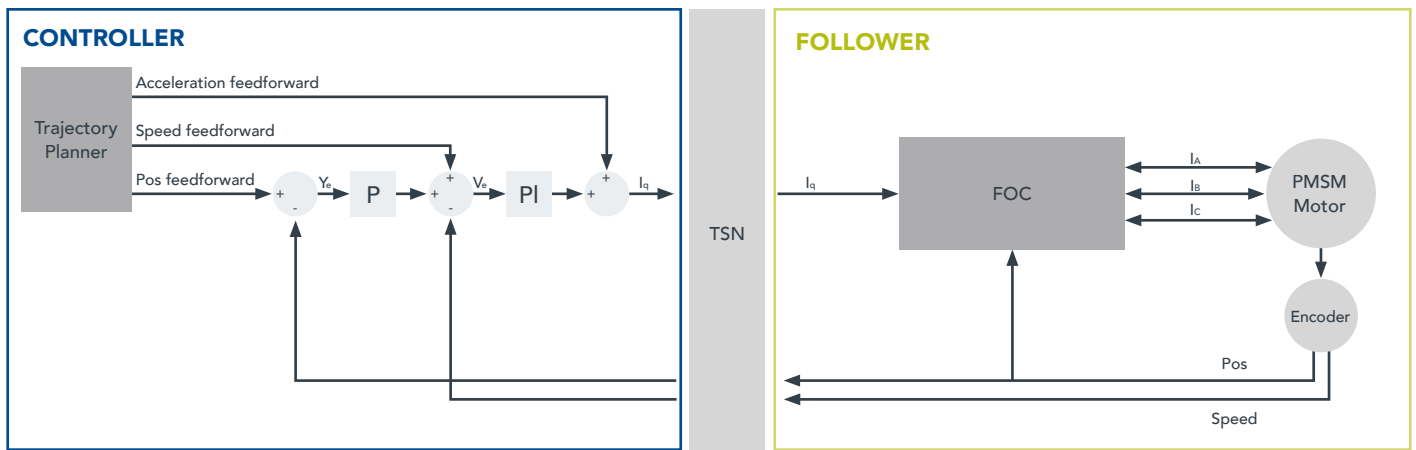


Figure 9.18. Distributed motor control application architecture

Followers are responsible for implementing field-oriented control to manage the motor. They use an I_q value as input, which is monitored with a P-PI controller and then transformed to concretely apply voltages on phases A, B and C of the motor. The followers also determine the actual position and speed of the rotor and sends this information back to the controller.

Motor control application timing — The application cycle running on endpoints is set to $100 \mu\text{s}$ using a hardware timer synchronized to the network time established by gPTP. This timer service is made available in the RTOS environment by the TSN software stack, which executes gPTP in addition to the IEEE 1588 hardware functions supported by the embedded processor. Those functions allow for the precise timestamping of the packet transmission and reception operations performed by the Ethernet MAC controller. This helps accurately calculate the propagation delay on each Ethernet segment interconnecting each node. After the endpoint motor control application registers with this timer service, it is scheduled synchronously with each of the other endpoints in the application. Hence, both controller and follower nodes are time synchronized to the network time established by gPTP.

Using that common time reference, the application cycle of the followers is shifted by half of the period to reduce the round-trip controller's/followers' processing latency (see Figure 9.19).

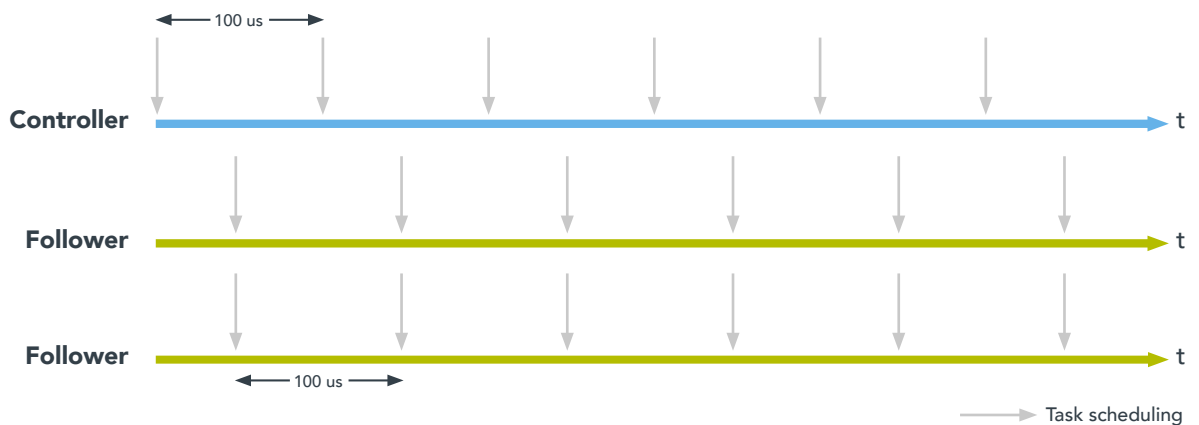


Figure 9.19. Distributed application timing

Motor control application real-time constraint — The controller must generate an I_q command in a $100\ \mu\text{s}$ period. To do this, the followers must transmit position and speed information for the rotors to the controller. For accurate and reliable control, this information must be delivered by the network before the start of the $100\ \mu\text{s}$ application period (see Figure 9.20).

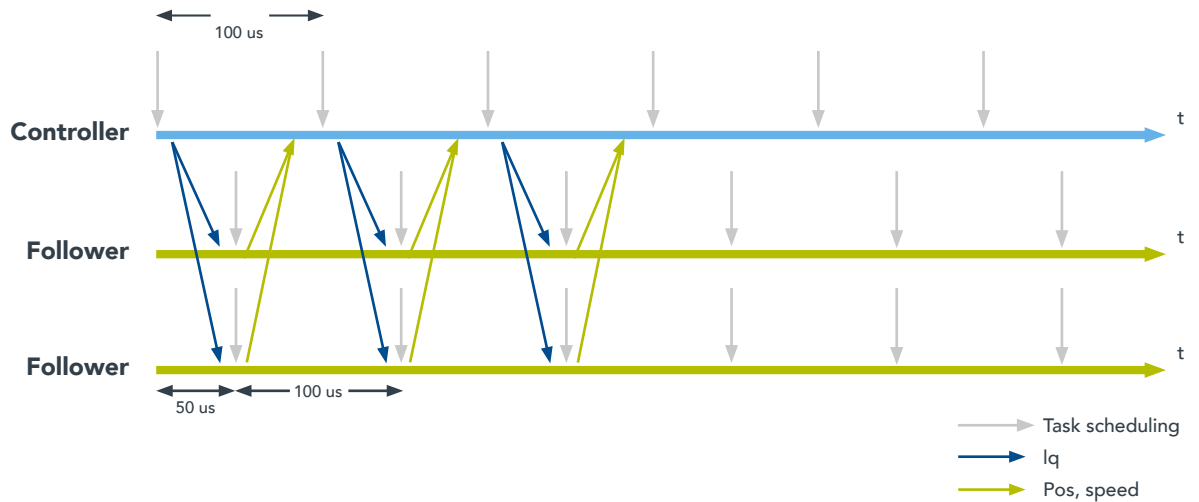


Figure 9.20. Distributed application real-time constraint

The shift of the application cycles leaves a $50\ \mu\text{s}$ budget for the application to process data received from the previous cycle and for the network to transport the output data to the remote application before the next remote application cycle starts.

The TSN Enhancements for Scheduled Traffic (EST) transmission is programmed to open the gate of the time-critical traffic queue $40\ \mu\text{s}$ after the start of the application cycle. This gives the critical traffic full priority and bandwidth to cross the network during a period of $3.696\ \mu\text{s}$ (see Figure 9.21) and pauses the transmission of any concurrent best-effort traffic. This EST period is set to cover transmissions of a motor control frame at $1\ \text{Gbps}$, plus a margin to account for the transit delay in each switch. All the switches on the network path between the controller and followers are aligned with the same EST window.

In this setup, a full load of concurrent best-effort traffic has no impact on the control process integrity.

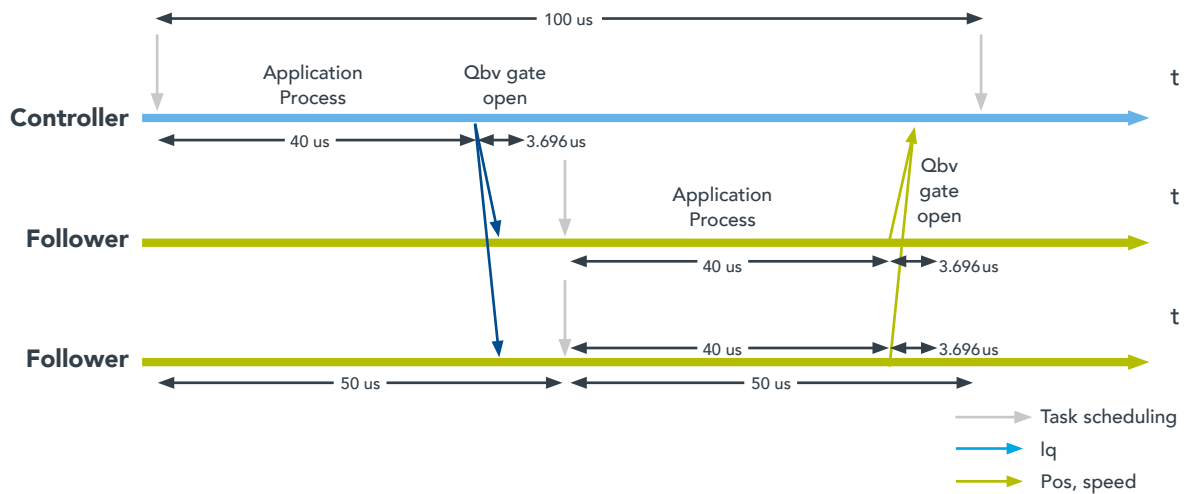


Figure 9.21. Application cycle and network cycle mapping

Distributed and real-time applications are building blocks of the edge computing pyramid. TSN is positioned as the common standard solution to offer deterministic network services to real-time IoT applications in the industrial, automotive and consumer domains. In these distributed environments, the real-time constraints propagate to all nodes participating in the end-to-end chain. That includes end nodes that host real-time applications and bridge nodes that transport critical data to serve the real-time goal, along with noncritical data.

EXAMPLE: INTELLIGENT CONNECTED VEHICLES

Intelligent connected vehicles may be considered the ultimate IoT edge devices. They combine multiple, high-performance edge compute nodes with high-bandwidth networking and connectivity to dozens of sensors. They also feature support for cloud-connected services with ML and over-the-air (OTA) updates (see Figure 9.22).

Intelligent connected vehicles must anticipate and counter security threats, while maintaining a high level of functional safety to reduce the risk of hazards caused by malfunctioning systems. Some people refer to these vehicles as being smartphones or data centers on wheels, but intelligent connected vehicles are significantly more diverse and complex. Their decades of evolutionary development involve many specialized computers called electronic control units (ECUs) featuring thousands of parts and hundreds of millions of lines of code from hundreds of suppliers.

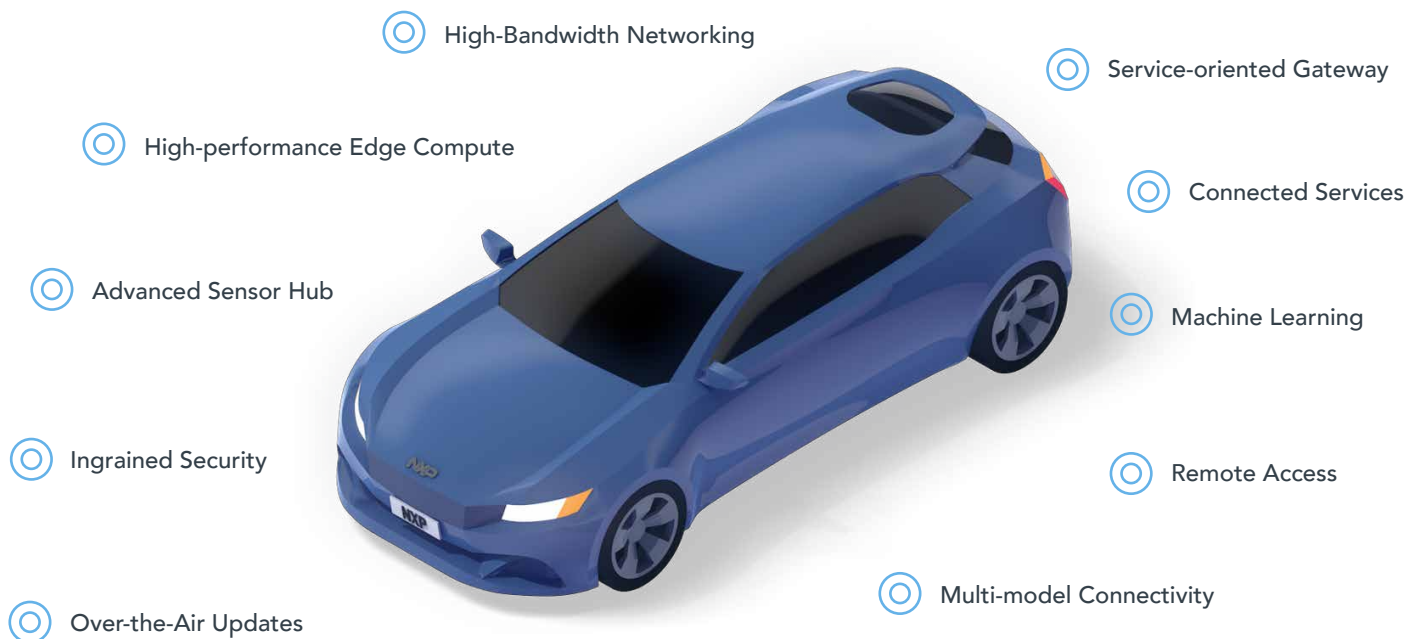


Figure 9.22: The automobile as the ultimate IoT edge device

THE END OF THE ROAD FOR TRADITIONAL VEHICLE ARCHITECTURES

However, vehicle electrical and electronic (E/E) architectures have reached a breaking point that can't scale using incremental ECUs for new functions. They can't provide the capabilities to address the automotive industry megatrends of connected, autonomous, service-oriented and electric (CASE). Connected vehicles require secure connectivity and edge processing of vehicle-wide data and support for vehicle-wide OTA services. Autonomous vehicles involve mega-sensing, high-performance compute, ML and the highest level of functional safety (Automotive Safety Integrity Level D or ASIL D).

Service-oriented vehicles require new software architectures based on high-level OSs, virtualization, containerization and middleware to support new vehicle services such as mobility/ride sharing and improved user experiences. Electric vehicles (EVs) demand high-performance real-time processing for advanced algorithms to manage electric motor control, vehicle dynamics, battery management, charging and energy optimization using virtualization to consolidate these functions on the same device. CASE is driving diverse, new edge processing requirements that go far beyond the traditional automotive microcontroller-based black box approaches of the past. It features SoC devices that integrate more advanced processing and application-specific acceleration to meet modern vehicle production challenges.

Today's high-end, complex vehicles incorporate over 150 ECUs. The wiring harnesses to support connections are the third highest contributor to vehicle weight; they are so thick and complex that they impact manufacturing. Weight reduces the fuel efficiency or range of vehicles and complexity increases manufacturing and overall cost. As a result, vehicle E/E architectures are evolving to consolidate ECUs into more powerful SoC processors that are connected with high-speed interfaces like gigabit Ethernet and PCI Express to drastically reduce the number of boxes and wires. New vehicle architectures are more centralized using domain control units that consolidate functions with each functional domain. Future zonal vehicle architectures may incorporate centralized, high-performance compute with separate compute at different vehicle zones to reduce the total number of boxes down to a handful. These new architectural approaches support the high-performance computing and high-speed vehicle data sharing needed in the software to realize CASE and will transform the automotive industry over the next decade and into the future.

THE GATEWAY TO FUTURE INTELLIGENT CONNECTED VEHICLES

New vehicle architectures are moving to a centralized compute resource called a service-oriented gateway (SoG) or, vehicle computer. The SoG provides central access to vehicle-wide data within the vehicle and the cloud, host vehicle services and edge processing while working securely and collaboratively with cloud services. It is key to enabling the intelligent connected vehicles that will collect vehicle data in the cloud for ML and algorithm optimizations and use OTA to deploy updates for continual vehicle improvement over their lifetime. These vehicles will get "smarter" over time thanks to a data life cycle that supports ML training in the cloud and optimizes inferencing at the vehicle edge.

Traditionally, a vehicle gateway's role has been to securely move data from one part of the vehicle to another over heterogeneous networks. However, this role has been expanded with the evolution to an SoG based on a service-oriented architecture (SoA). Instead of using legacy kilobits to megabits per second automotive interfaces like CAN, LIN and FlexRay, SOGs incorporate high-speed gigabit Ethernet to offer services that can access high-speed vehicle data while providing vehicle edge processing to support cloud services. An SoG requires over 10X higher performance than a traditional gateway microcontroller. This performance is achieved by incorporating a multicore SoC-based product with network acceleration such as the NXP S32G vehicle network processors optimized for this application.

As shown in Figure 9.23, the SoG is central to the vehicle and networks with the domain controllers in each of the vehicle's functional domains: ADAS/autonomous driving, in-vehicle experience (infotainment), powertrain and vehicle dynamics and body and comfort. Additionally, it interfaces to the wireless connectivity domain, or telematics control unit (TCU) to provide a secure communications path to the cloud over cellular or Wi-Fi technology. Being central to the vehicle, it has the advantage of access to the vehicle data and the ability to efficiently provide vehicle-wide services that are run alongside the other domains' functions and offer a unique location in the vehicle to create new opportunities.

With connectivity to all the functional domains, the SoG serves as the orchestrator for vehicle-wide OTA updates that continually improve a connected vehicle for the length of its service. OTA can be used to update software, firmware, ML models and more to keep software-defined vehicles at their best. It can also be used to provide new services that generate incremental revenue for vehicle OEMs.

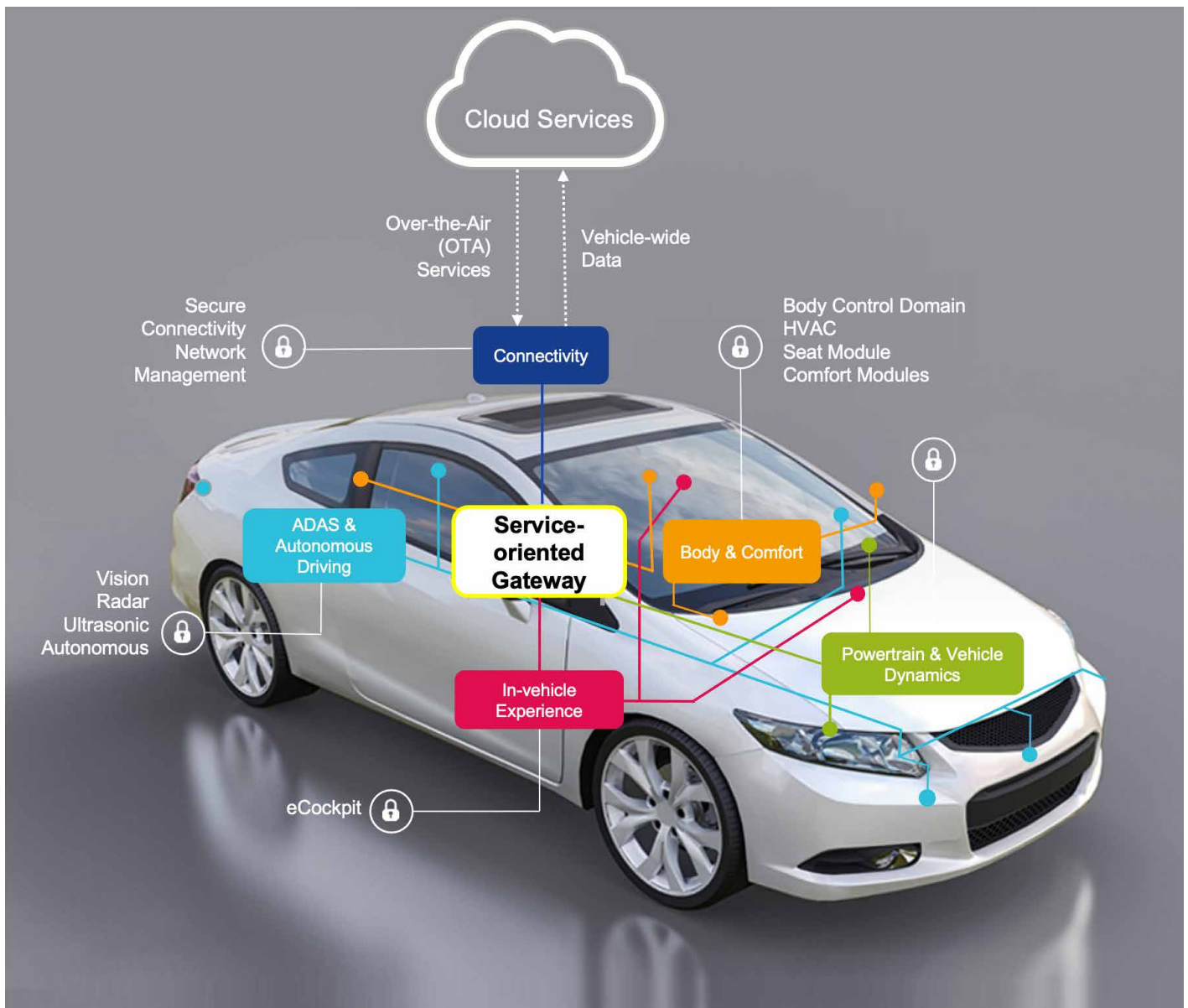


Figure 9.23. The service-oriented gateway

Laying the foundation for software-defined vehicles

SoA is also important to the evolution of software-defined vehicles because it enables a faster way to develop, integrate and deploy new software and services. As shown in Figure 9.24, software applications called services run on top of middleware such as the Adaptive AUTOSAR® platform that abstracts the underlying software and hardware. This is key to software updates; the vehicle provides this SoA that enables rapid deployment and portability across processors and even the cloud. An underlying high-level Portable Operating System Interface (POSIX) OS with PSE51 profile support such as a Linux OS running on an hypervisor provides virtualization and portability across multiple processor cores. Isolation of software is important in the SoA so that it can work with mixed-criticality applications as well as trusted and untrusted software on the same platform. An additional Trusted Execution Environment (TEE) using hardware isolation and access control to provide a secure world may also run in parallel to the Rich Execution Environment (REE). Typically, an Ethernet backbone offers cross-vehicle data and control sharing with the support of key network protocols including Scalable service-Oriented MiddlewarE over IP (SOME/IP) for control and event monitoring; Diagnostics-over-IP (DoIP) for diagnostics; Data Distribution Service (DDS) for distributed, real-time data sharing and Message Queuing Telemetry Transport (MQTT) for centralized data sharing while using underlying Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) transport protocols over IP with TSN for time-sensitive and deterministic network behavior.

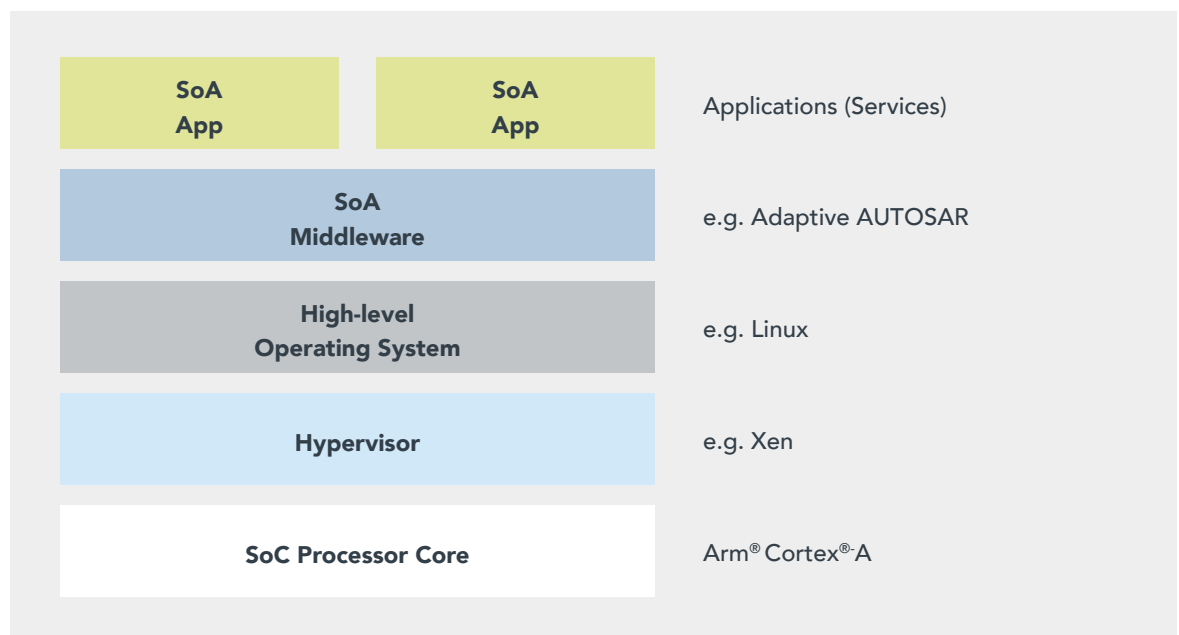


Figure 9.24 – Service-oriented Architecture (SoA) software stack

In addition to the traditional gateway and new services, the SoG provides a variety of functions including real-time applications and security processing that rely on diverse processing power (see Figure 9.25). Real-time processing and determinism is required for processing CAN and sensor data traffic as well as providing safety monitoring and real-time ECU consolidation functions. Applications processing prepares the vehicle for secure transmission to the cloud, manages vehicle-wide OTA updates, performs edge ML inferencing and processes higher level Ethernet packets. It also will run future deployed capabilities. Security is paramount and supported with a firewalled hardware security engine that is a root of trust with trusted boot. The RoT provides security services, cryptography and key management.

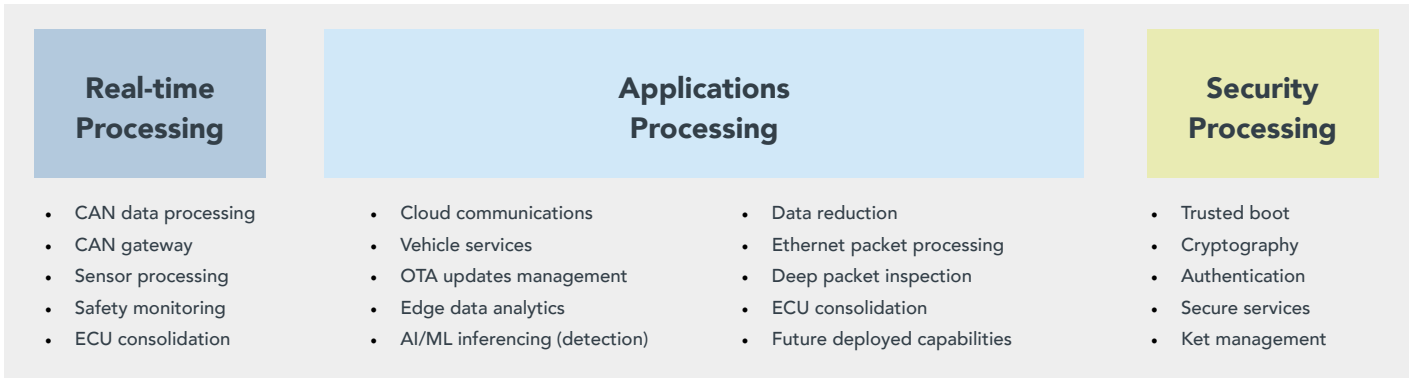


Figure 9.25 – Diverse Service-oriented Gateway processing

Key for a new generation of intelligent connected vehicles, the SoG is becoming the focal point for vehicle edge and cloud-enabled services. Vehicles have been connected since the late 1990s, evolving from information sharing and emergency services to vehicle monitoring and the data-driven intelligence used to continually improve the vehicle’s performance, efficiency, security and safety.

Connected vehicles offer many beneficial use cases (see Figure 9.26). Vehicle health management applications such as advanced vehicle diagnostics and predictive maintenance are becoming popular to determine vehicle component problems before the “Check Engine” light comes on or the vehicle breaks down. Vehicle health services are using advanced algorithms and ML to detect anomalies in sensor data, which requires machine inferencing support at the edge to reduce traffic to the cloud and send only notifications of events. In the end, this support keeps vehicles on the road, streamlines the supply chain by identifying needed parts and offers the owner a better user experience.



Figure 9.26 – Popular intelligent connected vehicle use cases

Symbiotic relationship drives significant benefits

The vehicle edge and the cloud can work together to provide significant benefits and new opportunities. The vehicle's SoG delivers key data processing at the vehicle edge to convert raw data to information, detect specific anomalies, make decisions locally on data to protect privacy, and compress data to make a connected vehicle economically viable since terabytes of data per hour cannot be sent to the cloud. Vehicle edge processing is critical to the success of intelligent connected vehicles. Figure 9.27 illustrates the processing implemented in the SoG at the vehicle edge and in the cloud. The vehicle edge-to-cloud communication is secured with encryption for privacy and authentication to prevent man-in-the-middle or unauthorized usage.

An exciting trend that will extend the use cases of intelligent connected vehicles is the combination of 5G ultra-low latency cellular networks with multi-access edge computing (MEC) that moves cloud servers closer to vehicles at the network's edge instead of a distant data center. This new approach reduces latency to single-digit milliseconds instead of hundreds of milliseconds which creates new real-time opportunities for symbiotic compute between vehicles and MEC servers.

Once data is in the cloud it can be stored in a data lake and combined with other enterprise data to provide business intelligence, analyzed across a vehicle fleet to indicate trends, used to feed ML training to improve vehicles, used to drive digital twin models to improve product design and much more. It can empower many new services from automotive manufacturers or their partners to create new business opportunities, reduce operational costs and offer new capabilities and improvements to their customers' vehicles. Automotive OEMs can evolve from being just manufacturers to more profitable, vehicle data-driven service providers.



VEHICLE SERVICE-ORIENTED GATEWAY

- Vehicle Services
- Vehicle Data Filtering
- Edge Analytics/ML inferencing
- Data Reduction
- Edge Storage
- Data Transmission

- Vehicle Data
- Edge Results

Secure Connection

- OTA Updates
- ML Models
- Vehicle Management



CLOUD SERVERS

- Data Ingestion
- Data Enrichment
- Data Analysis
- Machine Learning/Deployment
- Cloud Services
- Cloud Storage (Data Lake)

Figure 9.27 – Vehicle edge-to-cloud processing and storage

Enabling a smart data life cycle

By intelligently capturing and processing vehicle data at the edge and leveraging the cloud for heavy-duty processing of the data with access to thousands of CPUs and GPUs, ML training can be used to improve vehicles over their lifetimes. This smart data life cycle is key for the realization of intelligent connected vehicles. For example, ADAS/autonomous vehicles' vision processing can be improved for better decisions and safety and electric vehicle energy efficiency and battery management can be improved to extend EV range.

The edge is critical to enabling this life cycle by pre-processing only the data that is required to train specific ML models. This in turn can lead to much faster ML training and higher accuracy ML models that can run with lower latency at the edge. The benefits of the vehicle edge-to-cloud data life cycle with ML (see Figure 9.28) have been demonstrated by The Fusion Project - a collaboration of five automotive industry companies that brought together vehicle processing platforms (NXP), intelligent edge system software (Wind River), edge data AI (Teraki), cloud data platform (Cloudera) and OTA software management (Airbiquity).

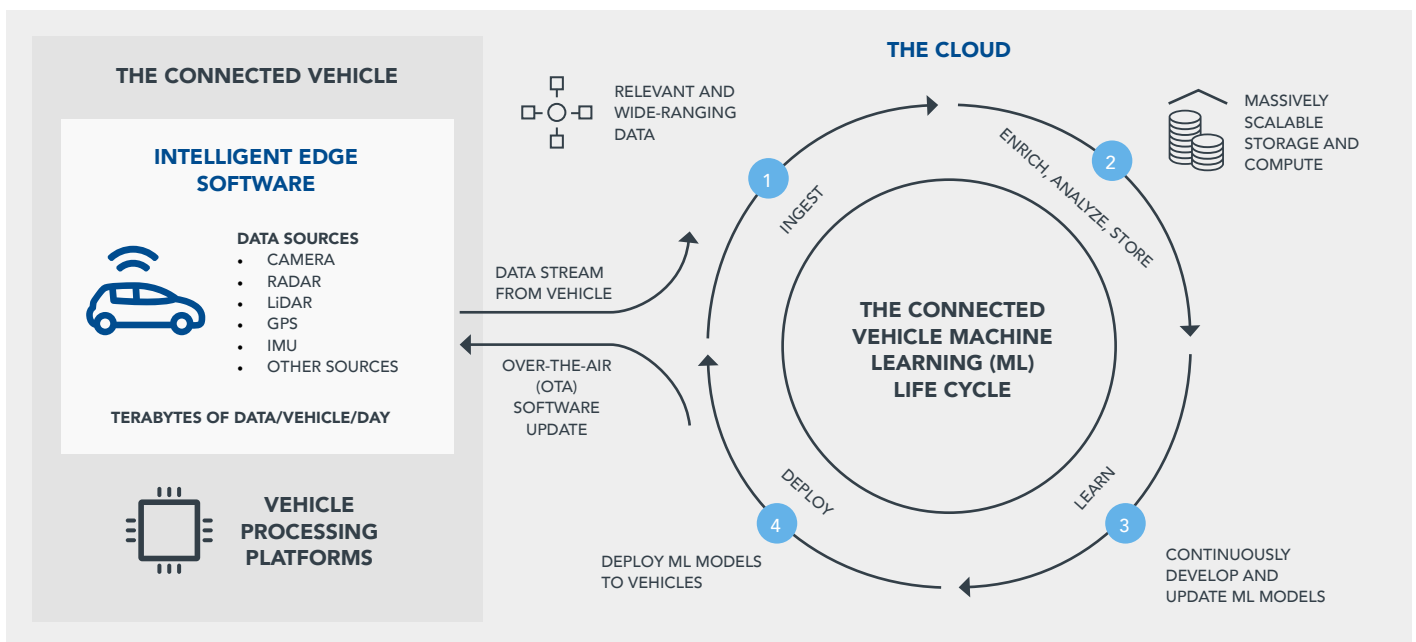


Figure 9.28– The vehicle edge-to-cloud data life cycle with machine learning

Vehicles generate terabytes of data per vehicle per day from multiple data sources such as sensors and internal ECUs that can provide valuable insights. Data is pre-processed on the vehicle edge and compressed to a format that maintains data fidelity for later cloud processing. The data is securely sent to the cloud for further enrichment, analysis and storage. The cloud provides massively scalable storage and compute for ML with model deployment back to vehicles through OTA software updates.

SECURING THE AUTOMOTIVE EDGE

Security is paramount to realize all the great benefits of intelligent connected vehicles. Being connected to the cloud increases the attack surface for a vehicle. Edge security is critical to maintain vehicle integrity and safety. Vehicles can cause death and destruction, so extra measures must be taken.

Security starts at the manufacturing of the chips incorporated in vehicle ECUs, and the security level is increased at each stage of a vehicle's development life cycle in a non-reversible and irrevocable manner. Security stages enable application development, debugging and failure analysis without compromising security in a production vehicle. At the chip manufacturer, unique chip identity and security keys are included in the chip. After the ECU assembly, Tier 1 security keys are incorporated into the device and the chip's life cycle is advanced to protect the keys. On the car manufacturer's product line, the chip inside an ECU is installed in a vehicle and provisioned with further configuration information and OEM security keys. Before the assembled vehicle leaves the factory, the chip's life cycle is once again advanced to enable all protection mechanisms of the device such as disabling debugging features. These steps are mandatory to secure secret keys in the chips that will be used for future encryption and authentication services in the vehicle. Figure 9.29 shows the chip security life cycle in automotive applications starting with chip provisioning at NXP and then progressing to the Tier 1 delivery, OEM production and in-field deployment stages of the life cycle.

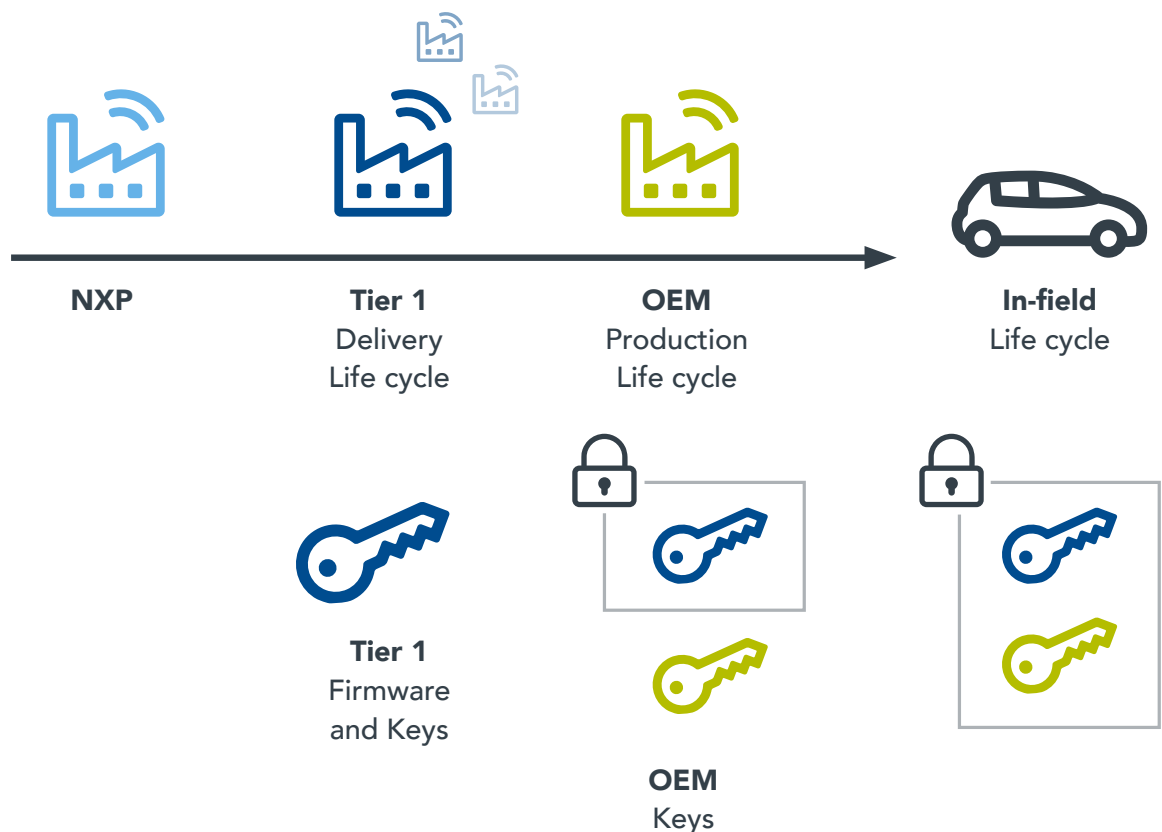


Figure 9.29 - Chip security life cycle in automotive applications

With high-end vehicles today surpassing hundreds of millions of lines of codes, inevitably software bugs can enable exploits. A variety of security measures and monitoring must be implemented to deal with these potential software exploits.

In the past, security researchers have demonstrated remote exploits using malicious code that allowed them to significantly control vehicles over wireless networks. End-to-end security measures are needed to secure data between the cloud and the vehicle to prevent data privacy leaks, data manipulation or exploitation of vehicle functions by an unauthorized party. New vehicle cybersecurity initiatives have to be considered. The ISO/SAE21434 and the UNECE WP.29 standards regulate new security requirements in process, design and operational efforts.

Multiple layers of end-to-end security from sensor authentication to secure communication to the cloud should be implemented. Secure interfaces can use encryption and authentication. Functional domains and a gateway can provide isolation with firewall capabilities. Hardware and software isolation within devices is important to localize and control potential security issues. Secure boot from an RoT with run-time integrity monitoring, tamper detection and side-channel attack preventive measures can help thwart hacking attempts of processors and secure elements. Networks can be secured with authentication and secure transceivers featuring active intrusion detection capabilities.

On top of all this, active monitoring or intrusion detection and prevention systems (IDPS) with data logging will become mandatory to provide real-time notifications of security threats. Also, a security operations center will monitor security threats across a fleet of vehicles and provide OTA security vulnerability patches to address them.

Security will always be a top priority for intelligent connected vehicles. Vehicles may someday have security star ratings like safety ratings to access their level of security measures against penetration tests. A vehicle's security rating can impact insurance rates and its resale value, so it can have major implications.



Chapter 10

EDGE COMPUTING DEVELOPMENT TOOLS

CONTRIBUTORS

Erich Styger, Professor and Researcher, HSLU University & Distinguished Member of Technical Staff, NXP Semiconductors

Michal Hanak, Senior Software Engineer, NXP Semiconductors

Pascal Mareau, Technical Leader, Power Technology Engineering, NXP Semiconductors

This chapter begins with a discussion of the software development challenges encountered during edge computing solution design and the tools to address them, from code development to remote debugging. Review techniques to debug real-time applications. Also see Chapter 2 for hardware and software architectures.

DEVELOPMENT TOOLS

Edge devices can be small yet complex pieces of equipment with advanced technologies; therefore, developers are always interested in any tool that can help them design and control their equipment efficiently.

Developing software for modern applications, including edge computing devices, is more challenging than designing a traditional embedded system or MCU development process not only because of the increased hardware complexity and the need for more features in the firmware but also because of the changes in the development flow to reach higher productivity. Choosing the right process paired with the right tools is crucial in developing high-quality applications within budget and time constraints.

Many development tool challenges for edge computing applications are unique because the firmware requires complex communication stacks running on multiple devices that are interconnected and dependent on each other. The development teams working on these applications tend to be larger compared with the teams creating traditional embedded systems. Due to the nature of networked hardware and distributed development teams, physical access to the device may be limited, so development requires remote access and debugging tools.

To understand the needs and challenges for this type of environment, consider the development flow and the typical tools used. Traditionally, the edit-build-debug flow is followed by deployment after the firmware is complete (see Figure 10.1). If issues arise during the build or debugging, the firmware goes back to the edit phase and starts again until everything is ready for the deployment phase.

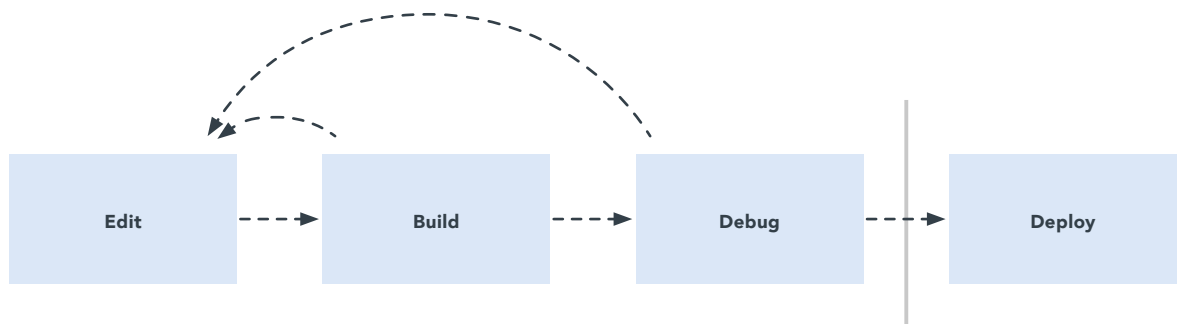


Figure 10.1. Basic edit-build-debug-deploy process

The exact tools needed depend largely on the complexity of the project and the preferences of the development team involved. Figure 10.2 indicates below the dashed line a typical set of tools that might be used with a Linux® host for cross development of a basic software project with few files and low complexity. The editor, VI (Visual), uses GNU Compiler Collection (GCC) and its associated build tools as the compiler, which uses GNU Debugger (GDB) for debugging and finalizing the binary. Then Secure Copy Protocol (SCP) is used to copy the file to a networked end device.

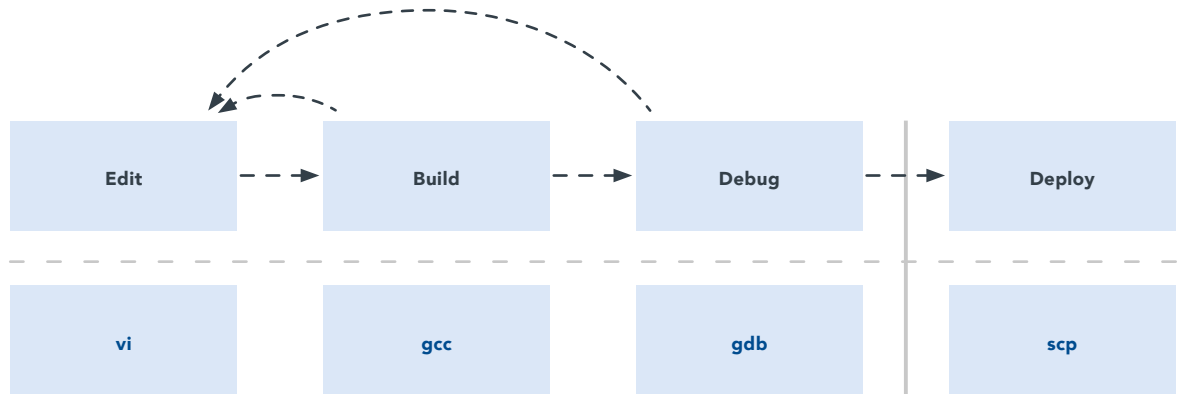


Figure 10.2. Simple flow with tools used

As the scope and size of a development team grows for more complex products, more collaboration and version control are needed. Using commits in a version control system (VCS) provides more formality and adds traceability, which is an important pillar for achieving and maintaining high quality in a development process. Another important aspect of a distributed VCS, such as Git, is that it encourages and allows cooperation and communication between the developers and teams.

The increasing number of files and project complexity create the need to use tools such as Make or cMake to perform the build. The cMake build system features cross-platform capability and is preferred for a distributed environment.

Lastly, using a distributed VCS offers the opportunity to deploy the binaries to a Git repository and publish them as a system release on GitLab or a similar service from where it can be distributed to the end devices (see Figure 10.3).

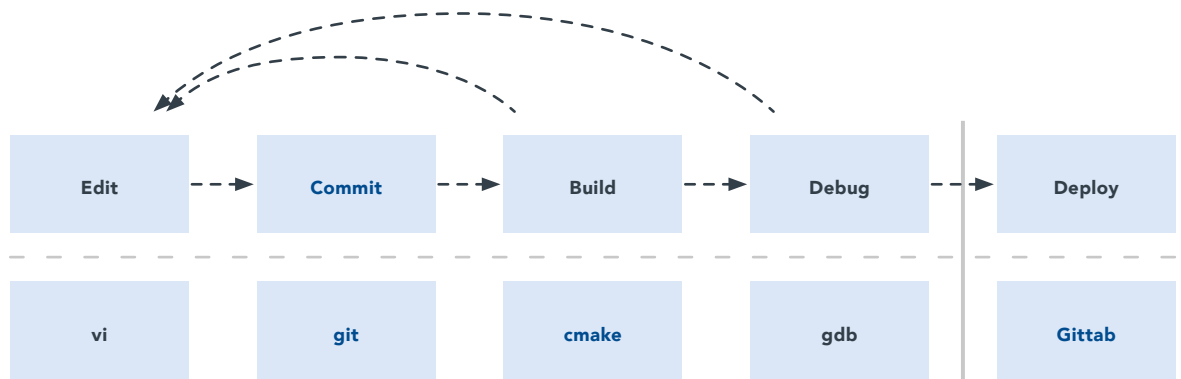


Figure 10.3. Development with version control

Next, consider debugging a real-time system. Though finding incorrect behavior in the system is important, it is often a manual process that is not easily scalable. GDB and other debuggers have an effective scripting engine for test automation.

To increase productivity, an automated test harness with unit and system tests is critical, along with continuous integration and continuous deployment. Whenever a developer commits a change into the version control repository, it triggers a build (for example, when using Jenkins) and a test through a test suite such as Unity that sends feedback to the developer. If the changed version fails the test, the system reverts back to the edit phase. If the changed version passes, the test is deployed automatically and can be loaded to the edge device using a bootloader and blhost (see Figure 10.4). The blhost application is used on a host computer to issue commands to a platform running an implementation of the MCU bootloader.

This continuation of the process is essential for any managed development methodology such as agile. Agile processes are effective for new and cutting-edge technologies with unclear requirements at the beginning of a project or requirements that change frequently during development. This is often the case for edge computing projects.

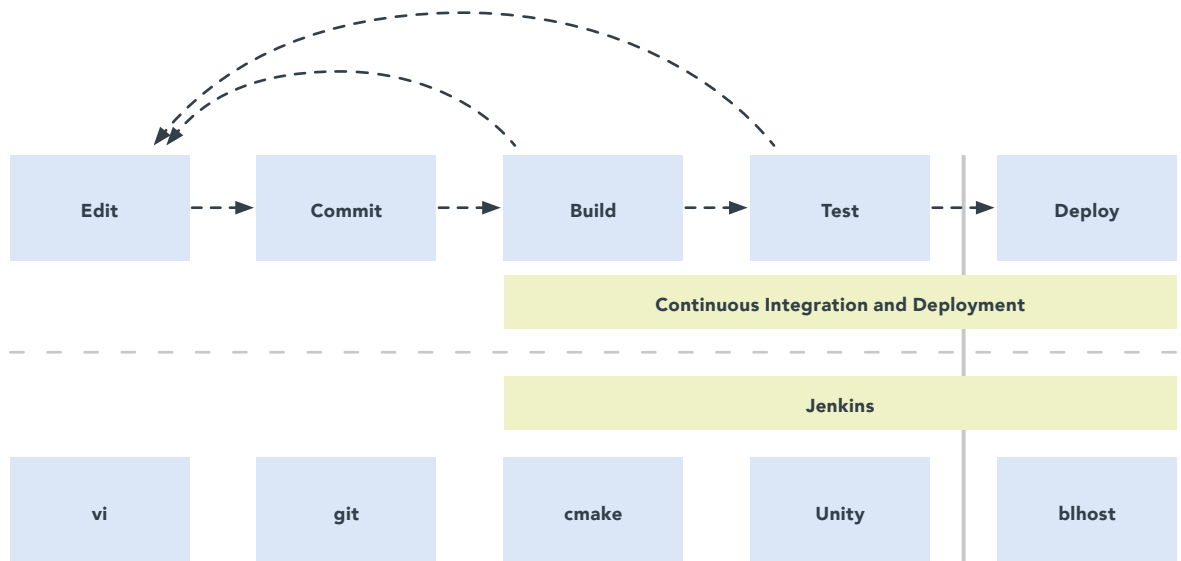


Figure 10.4. Development with continuous integration and deployment

Running system and unit tests is another way to ensure the quality of the result and avoid regression issues. The distributed nature of edge computing devices means they cannot all be updated simultaneously; therefore, the system needs to operate with heterogeneous firmware versions. So, ensuring compatibility and testing interoperability with different versions is key. To ensure that the tests have adequate coverage, tools such as cov can be used to collect additional data from instrumented binaries. To extend the dynamic test coverage (in addition to its static analysis), tools such as Cppcheck can be integrated into the build process to analyze the source code without running it and identify subtle issues that cannot be found otherwise.

Because projects tend to grow larger, traditional Make or cMake ninja or similar build engines can be combined to speed up the build process and increase productivity (see Figure 10.5).

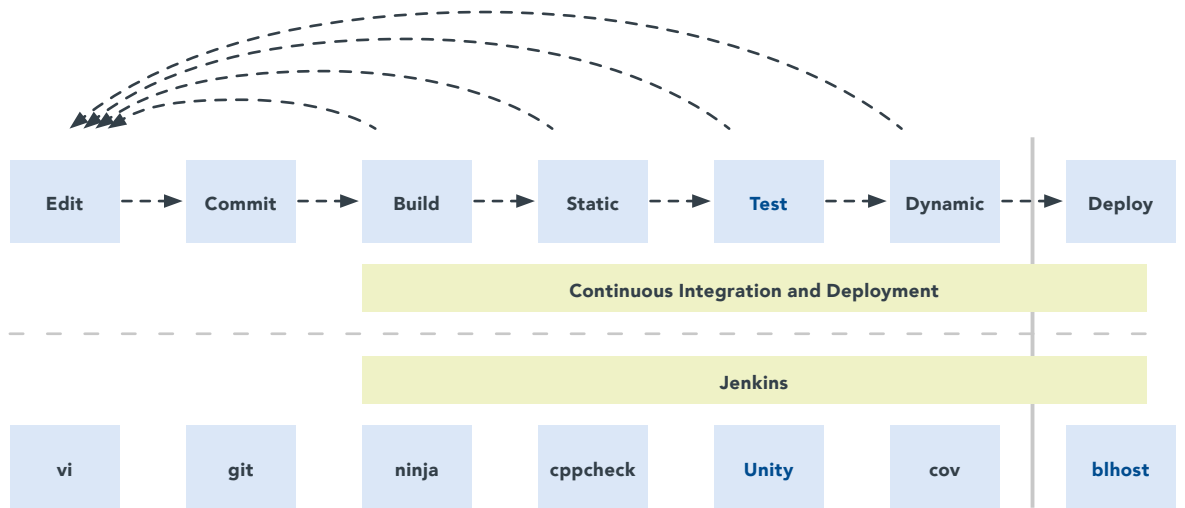


Figure 10.5. Development with analysis tools

To help further increase productivity and efficiency, an IDE integrates the tool and the development process.

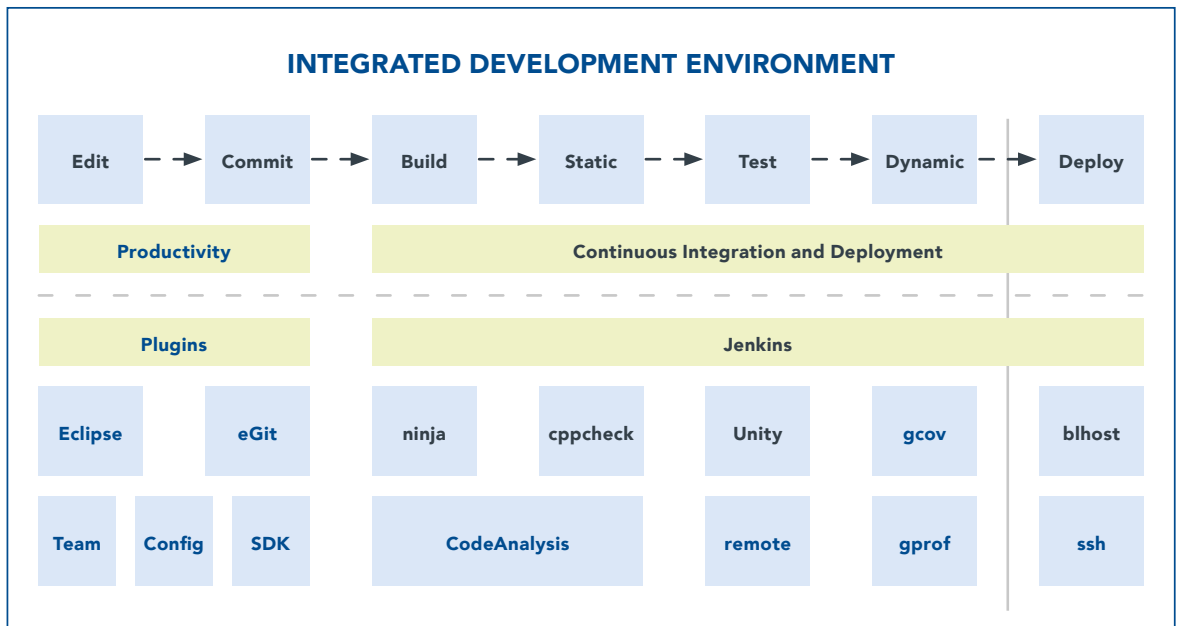


Figure 10.6. Development with an IDE

An environment such as Eclipse or Visual Studio provides not only an editor but also the interfaces needed to connect with tools and automation scripts. With a plug-in architecture, an environment that meets specific project needs and development flow can be adopted. Connectors such as eGit can be added to a VCS and team collaboration tools for discussions, issue tracking and source control. Configuration tools also can be used for device drivers, clocking or device peripherals, and software development kits (SDKs) with examples and driver code just to name a few (see Figure 10.6).

The IDE editor can intelligently parse (IntelliSense) the source code and assist the developer in writing or refactoring the firmware (see Figure 10.7).

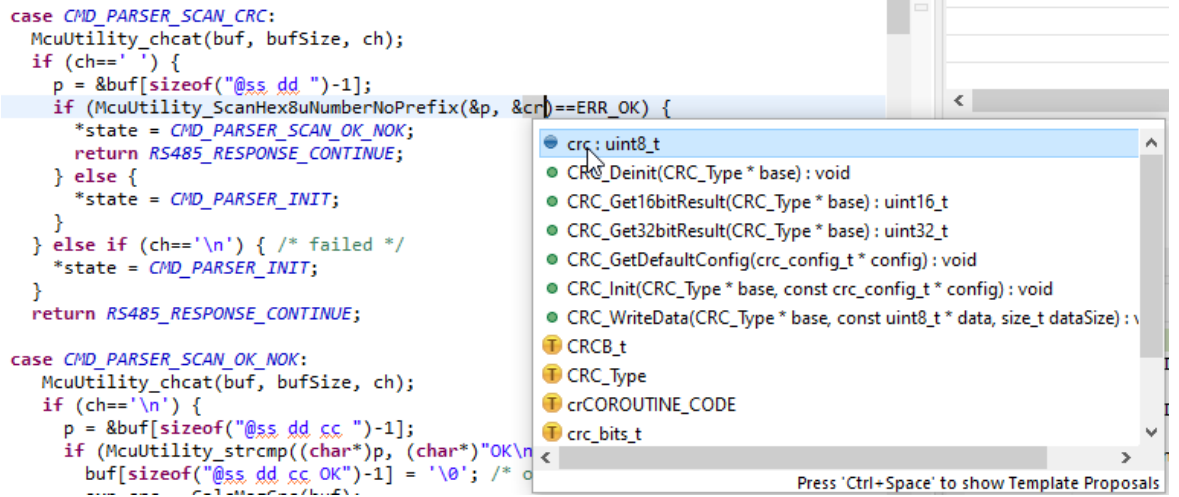


Figure 10.7. Code completion and IntelliSense in an IDE

Through CodeAnalysis, the IDE can be configured to detect potential problems while it writes the source code (see Figure 10.8).

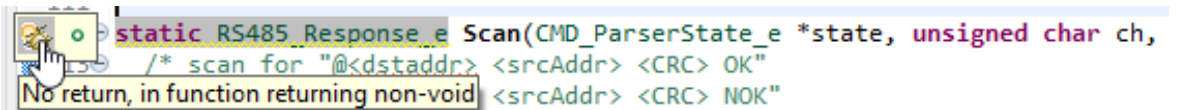


Figure 10.8. CodeAnalysis

Quality assurance and test tool views also can be integrated in an IDE. For example, running gcov displays a graphical view of test harness coverage (see Figure 10.9).

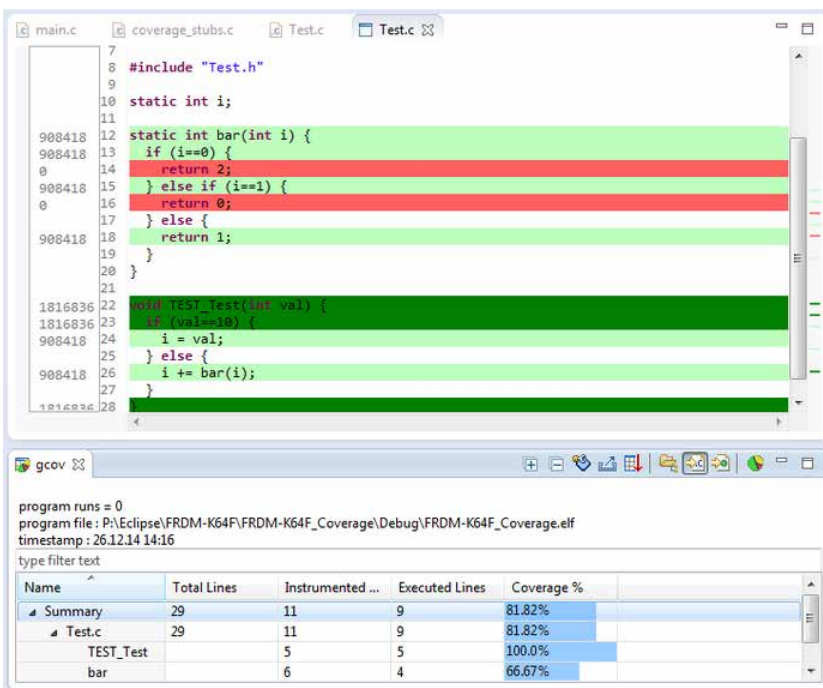


Figure 10.9. Coverage information in an IDE

With profiling, information is collected about where the application spends its time, so the developer can optimize the code and improve program performance (see Figure 10.10).

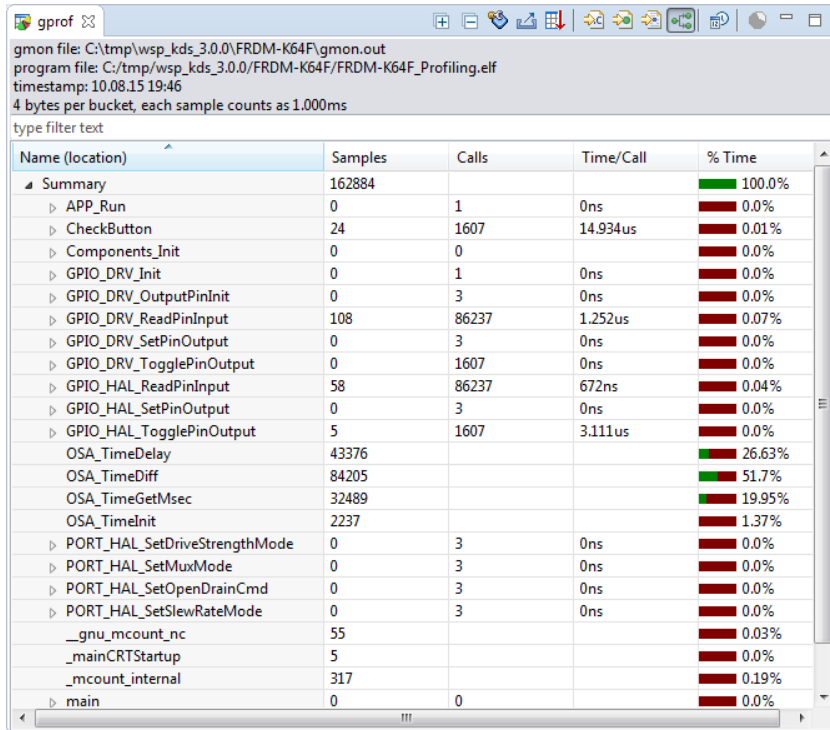


Figure 10.10. Profiling information in an IDE

Most edge computing devices run a Linux OS or real-time OS (RTOS). Traditional stop-mode debugging is limited in these environments, but IDEs offer both static and dynamic analysis of the running OS (see Figure 10.11). Combined with efficient code profiling, this analysis helps users assess and optimize the system activity, such as keeping the CPU idle and system in low-power mode as long as possible to optimize the power consumption and extend the battery life.

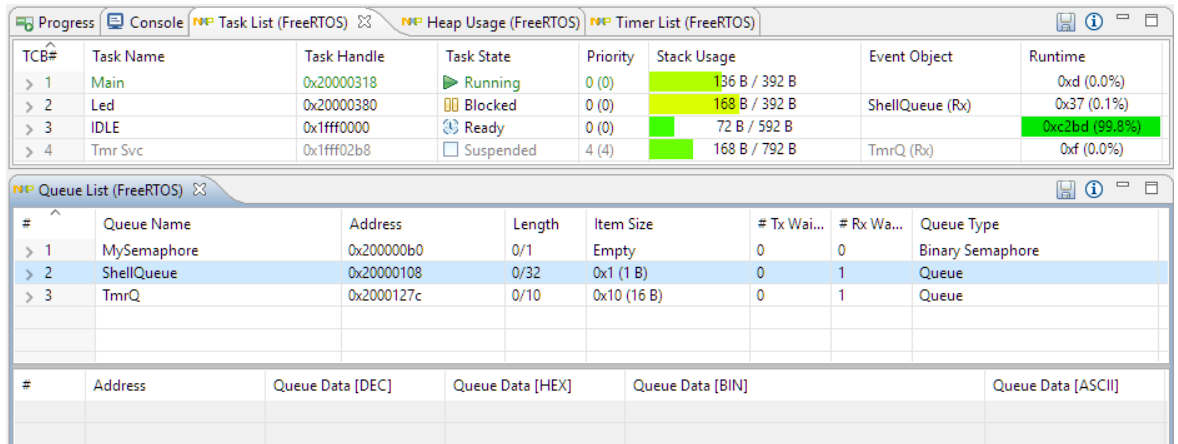


Figure 10.11. RTOS information in an IDE

Many edge computing devices do not have a direct physical connection to the outside world that can be used for inspection, control or debugging. The device shown in Figure 10.12 uses a printed circuit board (PCB) antenna for communication.



Figure 10.12. Wireless pressure sensor node

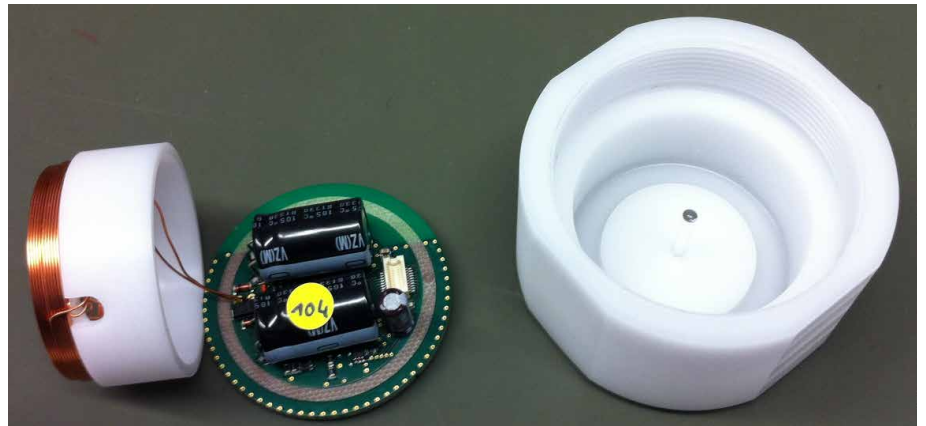


Figure 10.13. Wireless charging

The device gets attached on top of spray paint cans or other pressurized cans to check with a high-sensitive pressure sensor whether it is leaking. Because the check must be performed while the can is moving inside the fab, no wires can be attached. Batteries are not economical because they would need to be changed for hundreds of devices every few months. Instead, the energy to power the device is transmitted wirelessly, too (see 10.13). The computation is performed on the edge, with hundreds of devices running through the production.

Because the sealed device lacks external connectors, everything needs to be wireless, from programming the device to debugging it to updating it over the air (OTA) with a bootloader to remotely accessing it. The standard wired debugging connection from a GDB client to a GDB server and a USB debug probe connection to the target over a Joint Test Action Group (JTAG) or Serial Wire Debug (SWD) interface (see Figure 10.14) are not possible for edge computing devices.

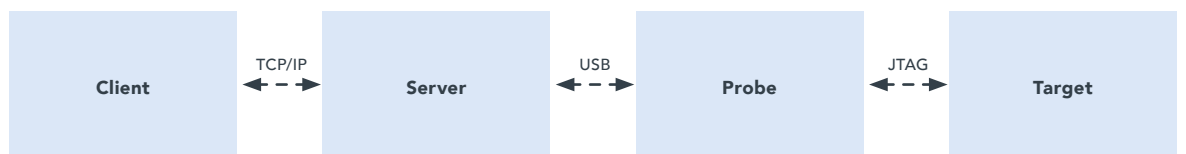


Figure 10.14. Standard wired debugging connection

By default, the client and server run on the same host machine. The architecture in Figure 10.14 allows the client and server to run on different machines, which means the server can be moved closer to the physical edge device. The socket transmission control protocol/internet protocol (TCP/IP) connection can be used over a wireless local area network (WLAN) connection, too. A networking debugging probe can be used to achieve a WLAN connection from the client to the probe (see Figure 10.15).

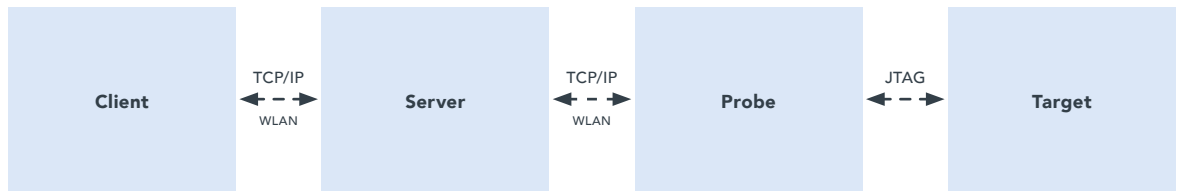


Figure 10.15. WLAN debugging connection

Using the connection from the debugging client to the networking debugging probe, an efficient and transparent debugging, software push and deployment can be achieved. The disadvantage of this is that it still requires either an attached debugging probe or an embedded debugging probe device on the target board, which is costly and/or requires too much space on an edge device.

The solution to this is to run a small “stub” or daemon on the target device that is used for debugging and communication (see Figure 10.16).

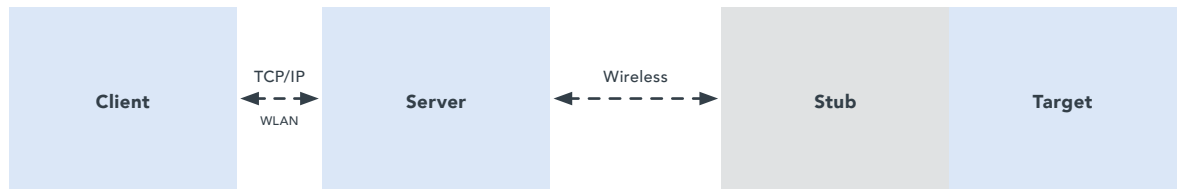


Figure 10.16. Remote target connection with a stub

A connection stub is the solution for the wireless pressure sensor edge device. It can use any protocol including a proprietary wireless connection. It can be applied not only as a debugging and deployment tool but also as a shell interface or a tool to transmit and analyze signal data (see Figure 10.17).

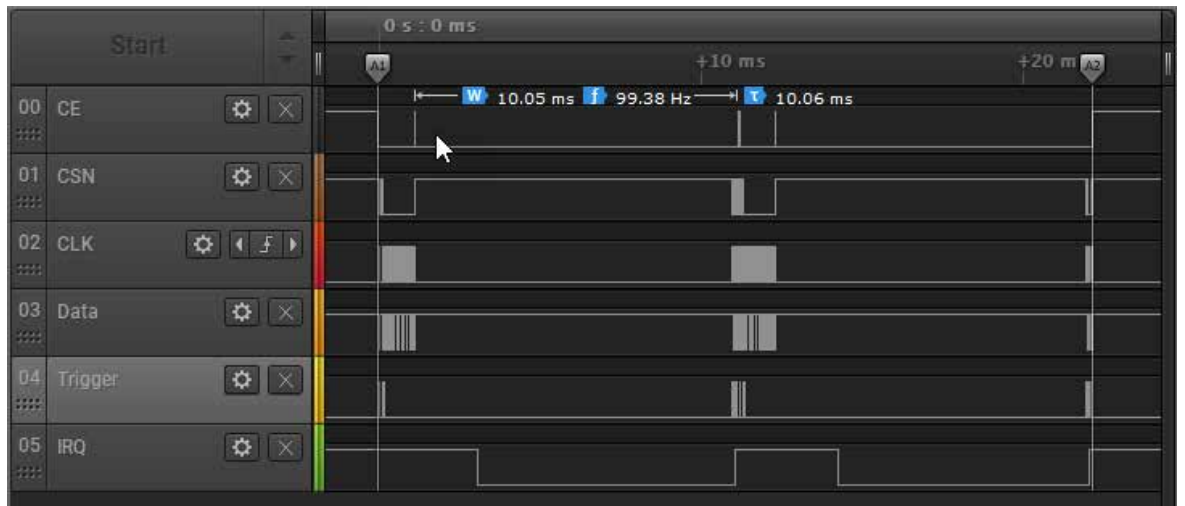


Figure 10.17. Signal data

By using the right tools and development flow, the barriers to achieving high quality and efficiency during edge device development can be minimized. A basic tool collection should include build tools with a fast build system and automated building as well as testing and delivery tools with both static and dynamic analysis capability. A high-quality IDE can deliver additional productivity by providing content assistance, automation and scripting. It also can offer visibility into the system, including the capability of remote deployment and debugging. This makes an IDE a key productivity tool for the entire development process.

DEBUGGING REAL-TIME APPLICATIONS

Debugging, a key part of software application development, is the process of identifying and resolving bugs, including common coding errors, algorithm errors and other issues that prevent expected operation. In edge computing applications and especially in embedded systems, the debugging may evolve into a complex activity requiring not only software and debugging expertise but also hardware skills and detailed knowledge of the complete system.

Real-time systems typically need to guarantee their ability to generate correct responses to external or internal events within a specific amount of time. Missing this timeframe may cause issues ranging from the degradation of system operation quality or safety to catastrophic damage and system failure. Intelligent critical sensors and motor control are examples of real-time edge applications.

Debugging real-time systems can be drastically different from debugging non-real-time applications. All software debugging tools and techniques can be used in both real-time and non-real-time applications, but the dynamic constraints and response-time requirements of the real-time world make standard methods impractical for these systems. Consider the most frequently used debugging techniques:

- Suspend execution after a breakpoint hit or manually examine the application task's state, variables and other data.
- Step over single lines of code or single assembler instructions while observing changes in variables and register values.
- Change the execution flow by modifying the program counter or variables while suspending the execution.
- Capture variable values automatically with the debugger tool when code execution hits defined watchpoints.

Only the last technique can be used for real-time debugging, and that is only when the overhead of data watchpoint processing does not affect the system responsiveness.

In short, the challenge of real-time debugging is to gather as much information about the system as possible without pausing or otherwise affecting its code execution timing. Developers and testers of real-time applications use different methods to debug and tune their systems for optimum performance.

A system can be observed passively or actively. Passive observation typically involves special equipment such as oscilloscopes, logic analyzers or communication analyzers to monitor:

- Hardware event and response I/O signals, analog-to-digital converter (ADC) inputs, pulse-width modulation (PWM) or timer outputs, etc.
- Output signals intentionally exercised by the application code to reflect internal algorithm processing, the timing of interrupts, idle-time signaling, etc.
- Communication lines between different processing units and other building blocks of the whole system

Active self-diagnosing systems use spare CPU cycles to log various diagnostic data, store it for offline analysis or even transmit it to external diagnostic tools in real time. The physical communication and format of diagnostic data are often proprietary to the manufacturer, but there are also documented protocols and standards such as the CAN Calibration Protocol (CCP) or its successor, the Universal Measurement and Calibration Protocol (XCP).

Tool and technique options in the mid-range between raw hardware analyzers and high-end diagnostic tools typically do not require deep hardware skills or expensive diagnostic and calibration equipment, and they may help users debug applications without pausing the code execution. The following techniques are popular with embedded application developers whether they are electronics hobbyists or high-tech company employees or both:

- Use JTAG or another background debugging interface to read or write the system memory directly. This technique may require “stealing” a few CPU cycles to access the memory, so it is not a completely passive method. Still, the overhead and probability of affecting the system response time are very low.
- Use a simple request-response communication protocol with minimum run-time overhead to help access internal resources, especially the memory. This technique typically involves a standard physical interface such as a universal asynchronous receiver-transmitter (UART), a serial peripheral interface (SPI) or an I2C. The CPU uses its idle time for protocol processing, so this is one of the active debugging methods. However, the trivial task of accessing the memory content can be simple and deterministic without affecting system responsiveness.

Accessing the target system memory and variables is the best solution to monitor, control, calibrate or debug real-time embedded applications. It is also the least expensive method, considering the overall complexity and cost of the equipment needed by other real-time debugging techniques. Reading selected variables may provide enough insight into algorithms, tasks and other running processes. Variables can be carefully modified to control the system, tune performance or diagnose behavior under specific conditions or in corner cases. On the other hand, testers and developers may find that the advantages of this direct memory “manipulation” approach are overshadowed by security weaknesses during the development phase, so they disable it in a production version of the software.

Using either technique requires some amount of software acting as the tool that provides access to the target system memory. This tool needs to know the memory layout and the details of the types and locations of the target application variables. All the required information can be read from symbolic information embedded in the application executable file (Executable and Linking Format or ELF) produced by the compiler and linker tools.

Another aspect that makes direct memory monitoring tools so popular is many semiconductor vendors provide these and other tools at no charge with their products. Also, the tools typically offer additional capability, advanced visualization and dynamic exchange of the acquired data with third-party applications or script languages (see Figure 10.18).

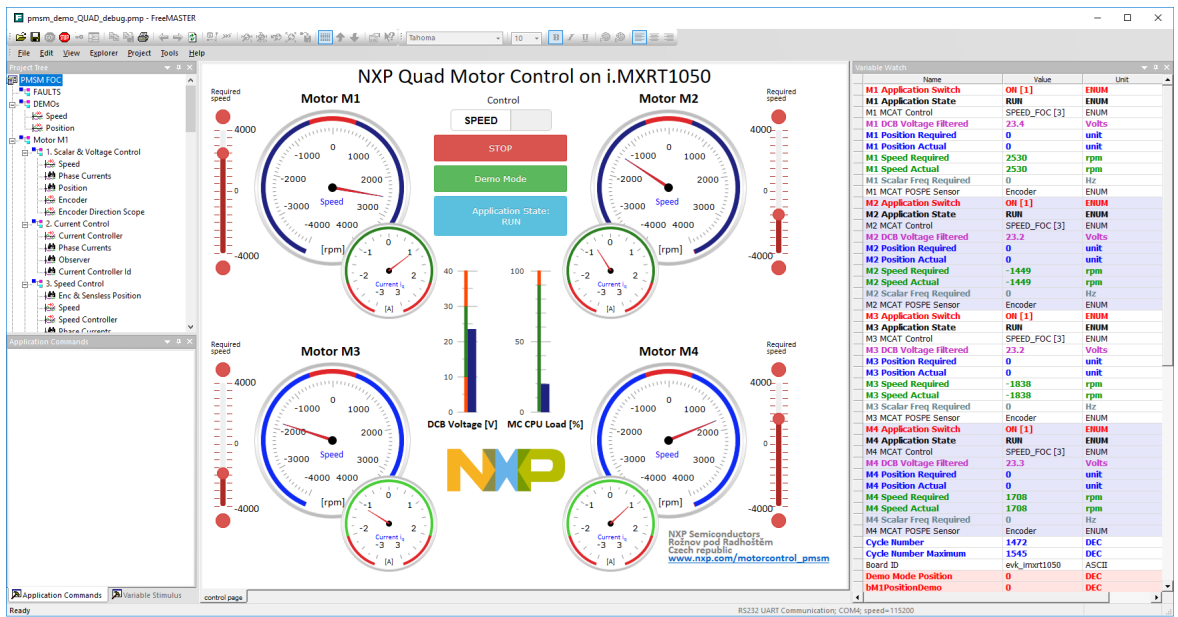


Figure 10.18. Example of a run-time debugging tool diagnosing for the quad-motor control application

SUMMARY

In summary, edge devices can be small yet complex pieces of equipment with advanced technologies; therefore, developers need a variety of code development, integration, debugging and optimization tools that can help them design and control their equipment efficiently.

Developing software for modern applications, including edge computing devices, is more challenging than designing a traditional embedded system or MCU development process not only because of the increased hardware complexity and the need for more features in the firmware but also because of the changes in the development flow to reach higher productivity. Choosing the right process with the right tools is crucial to develop quality applications within budget and time constraints.

GLOSSARY

Acronyms, definitions and concepts

6LoWPAN: IPv6 over Low-Power Wireless Personal Area Network

ADAS: advanced driver assistant systems

ADC: analog-to-digital converter

AEAD: Authenticated Encryption with Additional Data

AES: Advanced Encryption Standard

AI: artificial intelligence

ALU: arithmetic logic unit

AMQP: Advanced Message Queuing Protocol

Anti-rollback: A mechanism that prevents changing an asset (such as firmware and key stores) to an earlier version than the one currently running in a system (rollback).

Because, in many cases, updates are made to fix security vulnerabilities, a system rollback exposes these vulnerabilities again, and they can be exploited. Thus, rollbacks need to be prevented for a system to be secure.

AOP: acoustic overload point

API: application programming interface

AR: augmented reality

ASIL D: Automotive Safety Integrity Level D

ATT: Attribute Protocol

attestation: The act of verifying (typically cryptographic) evidence about a specific claim.

For edge node devices, elements such as the version of various firmware/software components (Am I running the latest, known to be good, version of the firmware? Or am I running an older version, with known and exploitable security issues?) and the authenticity of various assets (Is this key store truly provided by the OEM? Or is it a “fake” key store, injected in the device by an attacker?) are being verified/attested.

You may encounter several types of attestation:

- Static attestation refers to the secure boot process and the process of cryptographically verifying the signature of the image(s) loaded on the edge node.
- Remote attestation refers to the process of a remote entity asking for and verifying cryptographic proof for some relevant elements of an edge node provided by a trustworthy entity on the edge node. For example, a cloud-based service asks a secure enclave on an edge node to attest the version of the software that is running on the edge node (ensuring that it is a “known good version”) and the configuration of the edge node (ensuring that it has performed a secure boot and that it has not enabled debugging) before sending sensitive user data (for example, the results of a medical exam) to the edge node.

attacks: A cybersecurity attack targeting an edge node is an effort to defeat hardware and software mechanisms that protect assets and functionality, to run unauthorized firmware/software (i.e., break secure boot) and/or to configure the edge node in an unauthorized, exploitable way.

Edge-node-relevant cybersecurity attacks are classified into two groups:

- **Logical attacks** — No hardware means (probes, fault-injecting devices, etc.) are used to try and hack the device. Only existing software interfaces are used/exploited, and physical access to the device might not be needed. For example, a sequence of specifically crafted data packets provoke a crash of the edge device networking stack, leading to a buffer overflow exploit.
- **Physical attacks** — Hardware means are used to hack the device by attacking specific hardware/software mechanisms used by the edge node to securely perform its function.

For example, a voltage-glitching device is used to alter the execution flow of the edge node CPU to skip verification instructions for the image that is booted, thus potentially defeating the secure boot mechanism.

AWS: Amazon Web Services

BFWA: broadband fixed wireless access

Bluetooth LE: Bluetooth Low Energy

BSS: basic service set

CA: certification authority

CAAM: Cryptographic Accelerator and Assurance Module

CASE: connected, autonomous, shared and services, and electric

certificate: An electronic document containing a public key, the owner of the public key and a digital signature over both, provided by a signing entity.

The signing entity, a CA, plays the role of a trusted third party; if the user of the certificate already trusts the CA, and if the signature verification for the certificate is successful, the user also trusts that the listed certificate owner identity is truly the owner of the listed public key.

CCM: Cipher Counter Mode

CCP: CAN Calibration Protocol

CCPA: California Consumer Privacy Act

CMAC: cipher-based message authentication code

CMSIS: Cortex Microcontroller Software Interface Standard

CNN: convolutional neural network

CoAP: Constrained Application Protocol

CoMP: coordinated multipoint

CPE: customer premises equipment

CPRA: California Privacy Rights Act

CPU: central processing unit

CSMA: carrier sense multiple access

CST: code signing tool

CVE: Common Vulnerabilities and Exposures

D2D: device to device

DCT: discrete cosine transform

DDS: Data Distribution Service

DL: downlink

DM: device management

DMA: direct memory access

DMS: driver monitoring system

DoIP: Diagnostics over Internet Protocol

DoS: denial of service

DRAM: dynamic random access memory

DRM: digital rights management

DRX: discontinuous reception

DVFS: dynamic voltage and frequency scaling

ECC: elliptic curve cryptography

ECDSA: Elliptic Curve Digital Signature Algorithm

ECU: electronic control unit

E/E: electrical and electronic

ELF: Executable and Linking Format

emBB: enhanced Mobile Broadband

eMMC: embedded multimedia card

EST: Enhancements for Scheduled Traffic (IEEE standard)

EU: European Union

EV: electric vehicle

FOC: field-oriented control

FFT: fast Fourier transform

FSK: frequency-shift keying

FW: Firmware

GAA: General Authorized Access

GAP: Generic Access Profile

GATT: Generic Attribute Profile

GDPR: General Data Protection Regulation

GUI: graphical user interface

GPU: graphics processing unit

gPTP: generic Precision Time Protocol

HAB: High Assurance Boot

HARQ: hybrid automatic repeat request

HCI: Host Controller Interface

HID: human interface device

HMAC: hashed message authentication code

HMI: human machine interface

HSM: hardware security module

HTTP: Hypertext Transfer Protocol

HW: hardware

IC: integrated circuit

I2C: inter-integrated circuit

ID: identifier

IDE: integrated development environment

IDPS: intrusion detection and prevention systems

IEEE®: Institute of Electrical and Electronics Engineers

IIoT: industrial internet of things

IKM: input key material

IMG: image

IoT: internet of things

IP: intellectual property

IP: Internet protocol

IPSec: Internet protocol security

JADE: joint angle and delay estimation

KDF: key derivation function

KPI: key performance indicator

LDO: low dropout

LF: low frequency

LOS: line of sight

LPWAN: low-power wide-area network

LTE: long-term evolution

M2M: machine-to-machine

MAC: medium access control

MAC: multiply-accumulate

MAC: message authentication code

MaxCL: maximum coupling loss

MCU: microcontroller unit

MD5: Message Digest Algorithm Five

MEC: multi-access edge computing

MFC: mel-frequency cepstrum

MFCC: mel-frequency cepstral coefficient

MIPS: million instructions per second

ML: machine learning

mMIMO: massive multiple input, multiple output

mmWave: millimeter wave

MPU: microprocessor unit

MQTT: Message Queue Telemetry Transport

MW: middleware

MWS: mobile wireless system

NAND: not and

NB-IoT: Narrowband Internet of Things

NFC: near-field communication

NIC: network interface controller

NLP: natural language processing

NPU: neural processing unit

NVIC: nested vector interrupt control

OEM: original equipment manufacturer

OOBE: out of box experience

onboarding: The process of getting a device registered with a service (in this context a cloud-based service).

The registration process requires that the device has valid pre-provisioned credentials that can be used in the onboarding process.

As part of the process, typically over a secure connection, the device receives additional, service-specific credentials and assets.

OFDMA: orthogonal frequency-division multiplexing

OS: operating system

OSI: Open Systems Interconnection

OTA: over the air

OTP: one-time programmable

PAL: Priority Access License

PCA: principal component analysis

PCB: printed circuit board

PCP: priority code point

PHY: physical layer

PKC: public key cryptography — Cryptography that relies on pairs of keys for ensuring data confidentiality:

- Each pair contains a public key and a private key.
- The public key is used for encryption. As the name implies, it is typically shared with entities that need to send confidential messages to the owner of the key pair.
- The private key is used for decryption. As the name implies, it is securely kept by the owner of the key pair, and used to decrypt messages encrypted with the public key.
- Also called “asymmetric cryptography”, it also allows the creation of digital signatures and thus building an authentication infrastructure.

PKI: public key infrastructure — An infrastructure (hardware, software, tools, policies) that relies on PKC to create digital certificates to facilitate the unique identification and authentication of entities that are part of the infrastructure and/or using the infrastructure services.

PMIC: power management integrated circuit

POSIX: Portable Operating System Interface

PSA: Platform Security Architecture

PUF: physically unclonable function

PWF: pulse-width modulation

QoS: quality of service

RAM: random access memory

RAN: radio access network

REE: Rich Execution Environment

REST: representational state transfer

RLC/MAC: radio link control/medium access control

RNN: recurrent neural network

ROC: receiving operating curve

ROM: read only memory

RoT: root of trust (in the edge node context) — The entity/asset/element inherently trusted (i.e., not verified/authenticated) upon system boot. Its purpose is to verify/authenticate at least the first non-inherently trusted layer of the firewall (typically a bootloader) that runs in the system.

- In a PKI-based system, the root of trust is typically implemented as a set of public keys stored in non-volatile memory and secured in the best possible way by the systems using it. The corresponding private keys are used to sign the images that are authenticated prior to being executed on the system.
- In NXP systems, the OEM root of trust is typically implemented as a hash of the OEM public keys corresponding to the OEM private keys used to sign the images that the system will boot. The hash of the public keys is stored in one-time-programmable fuses that are protected by the SoC infrastructure from various attacks. Because the hash is used by the ROM-based secure SoC boot process, it is referred to as “silicon root of trust” or “hardware root of trust”.

RPMB: replay protected memory block

RSA: Rivest-Shamir-Adleman

RSSI: received signal strength indication

RTC: real-time clock

RTLS: real-time location system

RTOS: real-time operating system

RTT: round trip time

SAS: Spectrum Access System

SD: Secure Digital

SDK: software development kit

SCP: Secure Copy Protocol

SDU: service data unit

SHA: secure hash algorithm (variants: SHA-1, SHA-2, SHA256...)

SIMD: single instruction, multiple data

SMP: Security Manager Protocol

SNR: signal-to-noise ratio

SoA: service-oriented architecture

SoC: system on chip

SoG: service-oriented gateway

SOME/IP: Scalable service-Oriented MiddlewarE over Internet Protocol

SPI: serial peripheral interface

SPL: sound pressure level

SRAM: static random access memory

SSH: secure shell

SSL: secure socket layer

SVG: Scalable Vector Graphics

SW: software

TCP/IP: Transmission Control Protocol/Internet Protocol

TCU: telematics control unit

TDD: time-division multiplexing

TDNN: time delay neural network

TEE: Trusted Execution Environment

ToF: time of flight

TOPS: tera operations per second

TPM: trusted platform module

TRNG: true random number generator

TSN: time-sensitive networking

TZ: TrustZone

UART: universal asynchronous receiver-transmitter

UDP: User Datagram Protocol UE: user equipment

UGV: unmanned guided vehicle

UHD: ultra-high definition

UL: uplink

URLLC: ultra-reliable low-latency communication

UUID: Universally Unique Identifier

UWB: ultra-wideband

V2X: vehicle to everything

VCS: version control system

VDD: voltage at drain

VLAN: virtual local area network

VLIW: very long instruction word

VPU: vision processing unit

WFE: wait for event

WFI: wait for interrupt

WFST: weighted finite-state transducer

WISP: wireless internet service provider

WLAN: wireless local area network

WPC: Wireless Power Consortium

XCP: Universal Measurement and Calibration Protocol

XGA: Extended Graphics Array

ZCL: Zigbee Cluster Library

ZDO: Zigbee device object

CONTRIBUTORS

Rob Oshana is Vice President of Software Engineering R&D for the Edge Processing business line at NXP Semiconductors. He serves on multiple industry advisory boards and is a recognized international speaker. He has published numerous books and articles on software engineering and embedded systems. He is also an adjunct professor at Southern Methodist University and is a Senior Member of IEEE.

Mohit Arora is a Senior Architect for Edge Processing Product Innovation at NXP Semiconductors where he is responsible for product definition of the i.MX family of application processors. He has more than 20 years of industry experience that include chip architecture, systems engineering and chip design with expertise in embedded security and low power design. He has also authored two books, "The Art of Hardware Architecture" and "Embedded System Design", has 13 issued patents and has published in more than 50 international publications, magazines and conferences.

Jean-Christophe Bodet is Senior Director, Edge Processing Software R&D, Industrial and IoT, for Edge Processing business line at NXP Semiconductors. Jean-Christophe has nearly 30 years of international experience in France, Germany, and Americas with global responsibility of customer technical support, marketing, system and application engineering, software services and R&D for automotive, industrial, consumer and networking markets. He has been driving culture changes and transforming organizations related to mechatronics, automotive networking, power management, software services and R&D.

Antoine Boiteau is leading development of networked real-time software for time-sensitive applications for the Edge Processing business line at NXP Semiconductors. Antoine has more than 30 years of experience in embedded software, with more than 20 years in managing projects in semiconductor companies. He joined Freescale/NXP in 2014 to lead the development of deterministic Ethernet software based on IEEE AVB/TSN standards for NXP processors in the automotive, consumer and industrial markets. Previously at Conexant/Mindspeed, he led development for telecommunication software in the customer-premises equipment market and was part of the VoIP advent in the early 2000s.

Cristi Caciuloiu is a Senior Software Engineer at NXP Semiconductors. Having a MSc in Computer Science, Cristi worked for 20+ years in Motorola, Freescale and NXP in network processing, wireline and wireless systems for infrastructure and IoT markets, fulfilling SW R&D development and architecture roles. The projects addressed applications in technologies such as VoIP, virtualization, 4G L1 and L2, WiFi, Bluetooth, and Zigbee.

Brian Carlson leads global marketing of automotive processors at NXP Semiconductors. He has over 30 years of experience driving leading-edge, computing and communications products. He served on the MIPI Alliance board of directors where he led the mobile charge into adjacent markets including automotive and IoT. Brian holds a Master's of Science degree in Electrical Engineering from Southern Methodist University.

Nicolas Collonville is Technical Leader for connectivity stack integration at NXP Semiconductors. He has expertise in low power optimizations, over-the-air firmware update processes and other system-related topics,

including architecture definition and enablement software. Nicolas has more than 20 years of experience in cellular and narrowband embedded systems (2G, 3G, 4G, Wi-Fi and Bluetooth Low Energy). Prior to joining NXP, Nicolas focused on software development and system architecture at both Nvidia and Texas Instruments.

Julien Delplancke is Senior Product Manager at NXP Semiconductors. As part of the IoT security team, he is driving NXP's secure service offering for IoT products and collaborating with device manufacturers, service providers and cloud providers in order to help NXP customers to protect their devices and services.

Silvano di Ninno leads the Security Technology Engineering Center for the Business Line Edge Processing at NXP Semiconductors. He has more than 20 years of experience in embedded software development and managing projects with expertise in VoIP technology, fast packet processing, and security.

Alexandra Dopplinger, P.Eng., is Product Marketing Director for Building and Energy across the Edge Processing business line at NXP Semiconductors. With three decades of experience in the semiconductor and telecommunications industries, she holds a patent for a redundant network solution. Her career began with a B.Eng. Electrical Engineering from the Memorial University of Newfoundland (Canada), which led to diverse technical roles in hardware and system design, product management, business development, and global marketing. Today, she partners with leading industrial companies to develop joint growth strategies.

Mihai-Andrei Dragnea is a Senior Embedded Software Engineer at NXP. He has expertise in Bluetooth LE communication stacks and systems for the automotive and IoT industries. Mihai has more than 15 years of experience in embedded software development, with more than 11 years in various narrowband wireless technologies (Bluetooth LE, ZigBee, Thread).

Natraj Ekambaram leads the engineering team responsible for eIQ® Toolkit and eIQ inference engines such as TensorFlow Lite, TensorFlow Lite for microcontrollers, Glow, Arm NN, and ONNX runtime on NXP's EdgeVerse™ processors. Natraj has 25+ years of experience in the semiconductor industry with specialization in AI/ML at the Edge, IoT and Wireless Software development. Natraj has 15 pending/granted patents.

Sebastian Grigore leads the software team at NXP Semiconductors responsible for development of connectivity protocols (Zigbee, OpenThread, and Matter). He has more than 20 years of embedded software development and managerial experience, covering networking wireline and wireless technologies.

Doru Gucea is a Software Engineer at NXP Semiconductors. He has expertise with wireless technologies, covering OpenThread, Zigbee, Bluetooth Low Energy, Connected Home over IP, IEEE 802.15.4, IEEE 802.11. He has multiple patents in these areas.

Michal Hanak is technical lead for the software team responsible for development of FreeMASTER and other runtime debugging tools at NXP Semiconductors. Michal has over 25 years of experience in C/C++ development for embedded systems and Windows, FPGA design and communications with a focus of interconnecting embedded applications with host computers.

Mathieu Imbault develops connectivity software for the Business Line Edge Processing at NXP. Mathieu has 20 years of experience in developing and managing deployment of wireless protocol stacks on microcontrollers. Prior to joining NXP, Mathieu was director of 4G modem software development at Nvidia.

Saleem Kala-Janssen is Director of Systems and Applications for Edge Processing business line in EMEA at NXP Semiconductors where he focuses on deploying new products in the field and works closely with companies across IoT, industrial, mobile and automotive markets. Prior to this role, Saleem was an RF design lead and SoC lead for wireless connectivity products.

Prabhu Loganathan leads the connected IoT and access market segments as the Senior Director of Marketing in the Wireless Connectivity Solutions at NXP Semiconductors. Before NXP, Prabhu held various marketing leadership and engineering roles at Marvell, IDT, Redback Networks and LSI Logic.

Nihaar Mahatme received his Ph.D in Electrical Engineering from Vanderbilt University in 2014 and BS in EE from Mumbai University in 2009. At NXP, he has been involved in SRAM technology development for low-power and high reliability automotive applications. He is Technology Strategy Lead in the Edge Processing business line responsible for technology roadmaps and growth planning for IoT/edge devices for industrial, automotive and networking applications. His research interests include low-power, reliability and novel memories. He has authored and co-authored more than 45 publications and holds 11 US patents.

Pascal Mareau is the Technical Lead of the Power Technology Engineering Center team, focusing on power management optimization of edge processing systems. Pascal has more than 20 years of experience in embedded systems, spanning from hardware design up to Linux kernel software development and optimizations.

Jason Martin is Senior Director of MCU software at NXP Semiconductors where he is responsible for the development of the MCUXpresso software and tools. He has been in the embedded software industry for more than 20 years and has experience in different market verticals, including automotive and consumer products.

Guillermo Michel is a Graphics Systems Engineer for eCockpit solutions at NXP Semiconductors where he defines future products' graphics pipelines. He has more than 15 years of experience in graphics and more than a decade in embedded graphics for different markets including industrial, automotive, IoT and wearables.

Sujata Neidig has over 25 years of experience in the semiconductor industry and has served in a variety of roles ranging from product engineering to marketing and business development. She is currently Director of Marketing for Wireless Connectivity, Smart Home & Building. She leads NXP's standards efforts for IoT connectivity and represents NXP on the Thread Group and Connectivity Standards Alliance's board of directors. She also is vice president of marketing for Thread Group and a vice-chair for CSA. She earned a Bachelor of Science in Electrical Engineering from the University of Texas at Austin.

Ali Osman Örs is the Director of AI & ML Strategy and Technologies at NXP Semiconductors. Ali has served as a director on the board and on advisory boards for several industry consortia. He has published articles and spoken on technical and business topics covering processing products for vision processing, machine learning and

autonomous systems in consumer, industrial and automotive markets. Prior to joining NXP, Ali was VP of engineering for CogniVue Corp. and led the R&D teams developing vision SoC solutions and Cognition Processor IP cores. Ali holds an engineering degree from Carleton University in Ottawa, Canada.

Nicu Penișoară leads the advanced intellectual property enablement team, which covers security, machine learning, and graphics for the Edge Processing business line at NXP Semiconductors. He has more than 20 years of embedded software development and managerial experience, conceiving, building and delivering solutions for technical challenges spanning multiple domains.

Laurent Pilati is responsible for developing and deploying voice and audio algorithms used on NXP edge processing platforms. Laurent has more than 20 years of experience in developing and managing deployment of signal processing and machine learning algorithms on embedded platforms. Prior to joining NXP, Laurent was Senior Principal Engineer at Broadcom for its audio algorithms for Bluetooth and mobile platforms. Laurent has 10 granted patents.

Wim Rouwet is a Distinguished Member of Technical Staff at NXP Semiconductors. He has a MSc in Electrical Engineering from Eindhoven University of Technology in the Netherlands. He spent more than 15 years in Motorola, Freescale and NXP in networking and network processing, wireless algorithm development, system and modem architecture roles. His focus is on 3GPP LTE and 5G as well as 802.11 processing stacks and their implementation. In his job, he has been responsible for 4G and 5G stack development, small cells, and O-RAN implementations associated with many wireless infrastructure projects. He has led key next-generation R&D projects including multi-standard modem architecture, virtualization, 5G macro and small cell as well as client-side products.

Erich Styger is researching and teaching computer science and electrical engineering at the Lucerne University of Applied Sciences and Arts in Switzerland and is a Distinguished Member of Technical Staff at NXP Semiconductors. He started his professional career from the ETH Zurich in 1994. As director, principal engineer and university professor he designed microcontroller instruction sets, developed several C/C++ compilers with advanced optimizations, contributed and extended debuggers with real-time features, developed dedicated tools for embedded systems and has led and participated in standard committees. His current focus and research area is 'everything connected': from small sensor systems up to large distributed embedded systems.

Marc Vaclair is a Technical Fellow and Senior Security System Architect with more than 35 years of experience in research and development who focuses on innovative security architectures for embedded systems at NXP Semiconductors.

Francois Villeneuve leads global business development for the Sensor business line at NXP Semiconductors. He graduated from INSA de Lyon in Physics and Semiconductor Materials and has 20 years of experience in various engineering and marketing positions at ST Micro, Freescale and NXP, specializing in analog and sensor technologies.

www.nxp.com

EdgeLock, EdgeVerse, eIQ, ICODE, I-CODE, Layerscape, MIFARE, the MIFARE logo, MIFARE CLASSIC, MIFARE DESFire, MIFARE Flex, MIFARE Plus, MIFARE Ultralight, MIFARE4Mobile, the MIFARE4Mobile logo, NTAG, the NTAG logo, NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, and SafeAssure are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Mali, Mbed, Neon and TrustZone are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Amazon, AWS, Alexa, AWS GreenGrass and all related logos and motion marks are trademarks of Amazon.com, Inc. or its affiliates. Android is a trademark of Google LLC. The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license. Cadence, the Cadence logo, and the other Cadence marks found at www.cadence.com/go/trademarks are trademarks or registered trademarks of Cadence Design Systems, Inc. All rights reserved worldwide. EGL and the EGL logo are trademarks of the Khronos Group Inc. Microsoft, Azure and Windows are registered trademarks of the Microsoft Corporation. OpenGL is a registered trademark and the OpenGL ES logo is a trademark of Hewlett Packard Enterprise used by permission by Khronos. Oracle and Java are registered trademarks of Oracle and/or its affiliates. TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc. Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc. © 2021 NXP B.V.

NXP and the NXP logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2022 NXP B.V.

Document Number: EDGEEBOOK REV 2