



ICT-56-2020 - Next Generation Internet of Things

# D 4.7

## SIEMENS Report on cognitive IoT hardware optimizations

Document information	
<b>Contract number</b>	957197
<b>Project website</b>	<a href="http://www.vedliot.eu">www.vedliot.eu</a>
<b>Dissemination Level</b>	Public
<b>Nature</b>	Report
<b>Contractual Deadline</b>	31.01.2024
<b>Author</b>	Yufei Mao (Siemens)
<b>Contributors</b>	Franz Meierhoefer (Siemens), Roland Weiss (Siemens), Joseph Daiaa (Siemens)
<b>Reviewers</b>	Oliver Brunnegard (MAGNA), Kevin Mika (UBI)
The VEDLIoT project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957197.	

Changelog		
<b>v0.1</b>	2023-08-02	Initial of Deliverable 4.7 - Outline
<b>v0.2</b>	2023-01-11	Initial inputs from Siemens
<b>v0.3</b>	2024-01-29	Review integration
<b>v1.0</b>	2024-01-30	Finalization

## Table of contents

Executive Summary.....	4
1 Introduction .....	5
2 Design of cognitive IoT devices .....	7
3 VEDLIoT Smart Field Device with AI .....	9
3.1 Hardware Design Process .....	9
3.2 Hardware Overview .....	10
3.3 Design Schematic .....	10
4 Software Workflow.....	16
4.1 Training .....	16
4.2 Synthesis.....	18
4.3 Deployment.....	19
5 Evaluation.....	20
6 Conclusion .....	22
7 References .....	23
8 List of Figures .....	25
9 List of Listings.....	26
10 List of Tables.....	27

## Executive Summary

This deliverable is the final report on Task 4.8 in WP4, which focuses on optimizing the cognitive IoT hardware platform for the integration of artificial intelligence capability, with an emphasis on its ultra-low power feature. The following chapters provide detailed information on the development and optimization procedure and results. The resulting hardware undergoes system design, manufacturing, testing, and evolves into its second version. Ultimately, this report also showcases the synergy between cognitive IoT principles, the AI capabilities of the MAX78000 chip, and the industrial use case motor condition monitoring, which is presented in Deliverable 7.5 [1]. The work serves as a tangible manifestation of our vision, demonstrating the potential for sophisticated intelligence within the industrial IoT landscape.

## 1 Introduction

The fusion of Artificial Intelligence (AI) with Internet of Things (IoT) systems heralds a transformative era in modern technology, offering unprecedented potential for intelligent decision-making, predictive maintenance, and enhanced operational efficiency. The integration of AI within IoT systems encompasses varied architectural approaches, each with its own unique set of advantages and limitations.

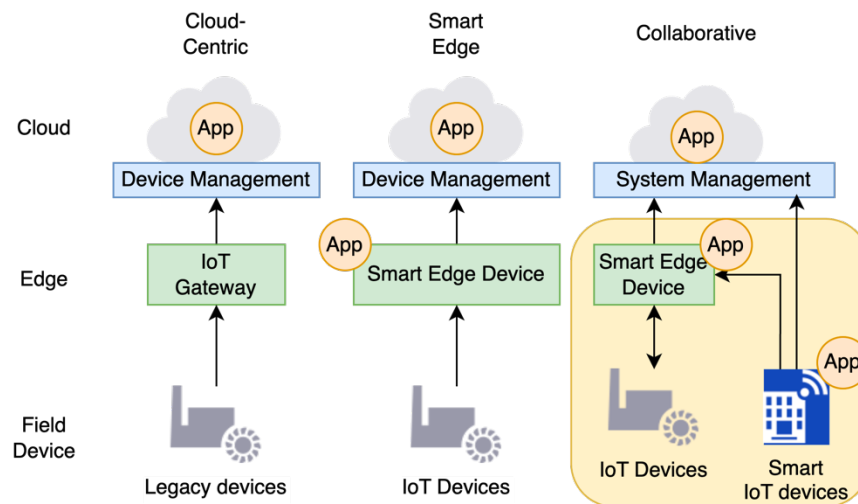


Figure 1 Architectures for industrial IoT [2]

Figure 1 shows different possible architectures for integrating AI into IoT systems, which also represents the evolution of IIoT from left to right. The cloud centric industrial IoT (IIoT) involves centralized AI processing, where data collected by IoT devices is transmitted to a centralized cloud or edge server for AI analysis. This architecture enables comprehensive data analysis and complex AI algorithms but is susceptible to latency issues and high bandwidth requirements for data transmission. Conversely, a smart edge IIoT architecture distribute computational tasks to smart edge devices near the IoT devices, fostering real-time decision-making without relying extensively on remote servers. This approach mitigates latency concerns and minimizes data transmission, ensuring faster response times. However, it might pose constraints on computational power and memory within the edge devices with the scaling up of the system. Collaborative IIoT represents the current trend of leveraging a blend of cloud-based, edge, and on-device AI processing. This amalgamation optimizes resource utilization, balancing computational capabilities with data transmission efficiency. Yet, it demands careful orchestration to strike a harmonious equilibrium between decentralized processing and cloud-based resources.

As shown in Figure 2, the VEDLIoT project aims to provide hardware solutions for the integration of AI in different levels of IoT systems within the Work Package (WP) 4. For the central cloud and edge devices, the RECS carriers provide heterogeneous computation platform. For embedded AI applications in collaborative IIoT systems, the cognitive IoT end device described in this deliverable is designed. The incorporation of AI processing on embedded system empowers them to discern intricate patterns, detect anomalies in real-time, and derive predictive insights from the data they collect. This transformative capability not only enhances operational efficiency but also enables predictive maintenance, proactive troubleshooting, and adaptive behavior—a cornerstone in revolutionizing industries across sectors.

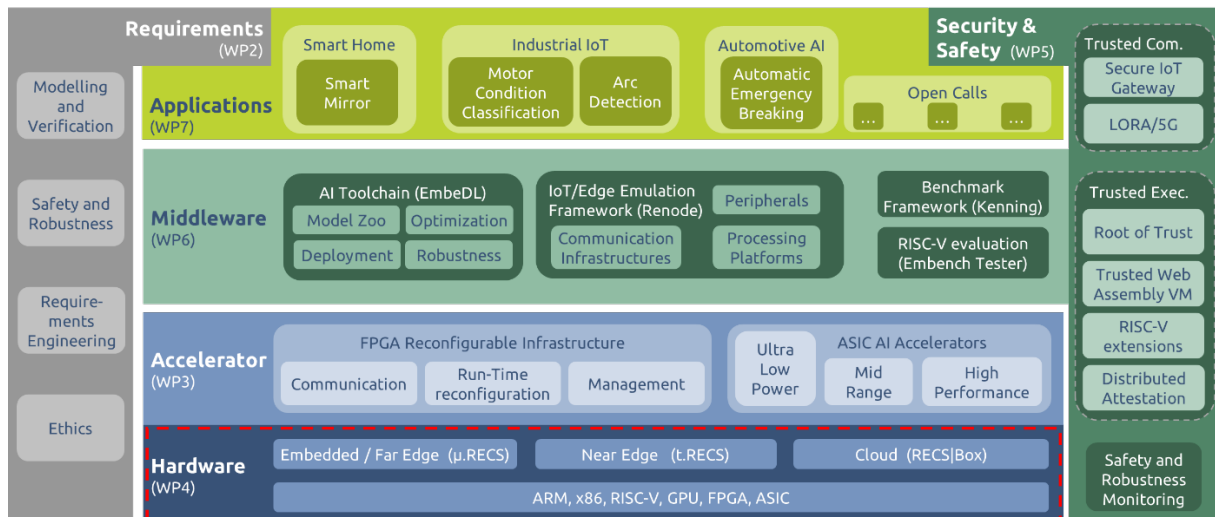


Figure 2 Overview of the VEDLIoT project with WP4 highlighted with red box [3]

The rest of this deliverable is structured as follows: chapter 2 describes the baseline Smart Field Device (SFD) where the AI power should be integrated, offers insights into the basic requirements and design motivation for this work; chapter 3 presents an overview of the hardware design and details for each module; chapter 4 further shows the workflow of working with the hardware, listing the necessary tools and steps; chapter 5 demonstrates the energy consumption estimation for final system, provides evaluation result for the system; finally, chapter 6 summarize the outcomes of the work.

## 2 Design of cognitive IoT devices

The design of the target hardware is based on the embedded system that has been implemented in industrial applications. The end device is designed to be battery-powered for situations where mounted end devices cannot be easily cable charged all the time. This section provides an overview of this baseline system and some of its application scenarios.



Figure 3 Battery powered SFD



Figure 4 An instance of the SFD

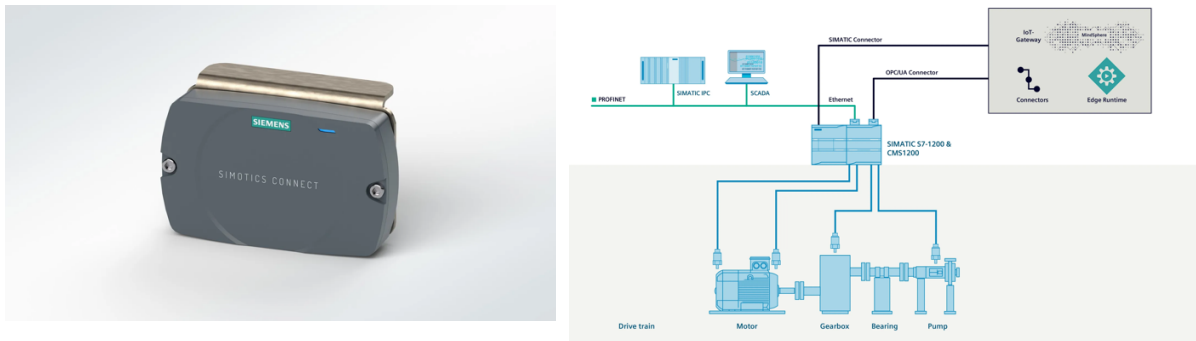
Figure 3 and Figure 4 show the design of the battery-powered small sensor box and an example of the SFD. This module has been implemented in multiple applications such as condition monitoring of train and locomotive transformers [4], Smart Multi-sensor for process industry [5] etc.

The design of the SFD has the four main building blocks sensing, communication, power management and data processing

- Sensing:
  - Vibration: 3 axes with an analog bandwidth of more than 1 kHz and 16 g full scale and a power consumption less than 300  $\mu$ W in run mode.
  - Magnetic field: 3 axes with an analog bandwidth of more than 1 kHz, 2 low power axes and 1 high bandwidth, high resolution for special monitoring tasks.
  - Temperature: High accuracy temperature sensor for local and remote sensing. For low accuracy tasks, the integrated sensors of the sensors or the microcontroller can be used.
  - Environment: Relative humidity and barometric pressure with a power consumption of less than 2  $\mu$ W.
- Communication: The design has the capability to operate in two different communication modes. Low power, permanent online mesh network and high bandwidth, burst transfer peer to peer (p2p) network. Both modes have, depending on the used parameters, a comparable energy consumption.
- Power management: All used peripherals like sensors, communication modules or memory devices can be switched off, to save energy. In addition to that, the power supply, the bus pullups, and battery monitoring can also be controlled to optimize energy consumption.
- Data Processing: Each bit that doesn't need to be transmitted saves energy. Thus, the data processing is one of the key components for a low power device. This can

be done either by data compression or information extraction. The cognitive IoT device is equipped with additional RAM and flash to perform information extraction with classical algorithms in an efficient manner.

A prominent example use case is the Simotics Connect 400 for condition monitoring [6] of low-voltage motors. It serves as a plug-and-play connectivity module for measuring and pre-processing capability. For this application, data transmission is conducted through the Wi-Fi module. And the battery can last up to 2 years with a data transmission interval of 5 minutes. However, this SFD is only capable of data processing with classical algorithms, which still leads to communication overheads.



- *Figure 5 Simotics Connect SFD (left) and IoT system based on it(right) [6]*

Therefore, to enable the potential power of SFD in the collaborative IIoT architecture and further reduce the overhead in communication, the SFD is to be integrated with AI power and optimized in aspect of power efficiency.



### 3 VEDLIoT Smart Field Device with AI

The integration of deep learning on SFDs has two challenges. Firstly, it requires maintaining efficiency and low power consumption while accommodating the computational demands of AI algorithms. Secondly, achieving this integration without compromising existing functionalities and scalability needs dedicated algorithm design.

Our initiative began with a vision to transcend the limitations of conventional IoT devices by infusing AI capabilities without sacrificing power efficiency. Based on the evaluation of AI accelerators in Deliverable 3.3 [7], we selected the ultra-low power AI accelerator Maxim MAX78000 as the core processor [8]. The design also combines the array of sensors from the original SFD, creating a harmonized system for intelligent inference and real-time data collection.

#### 3.1 Hardware Design Process

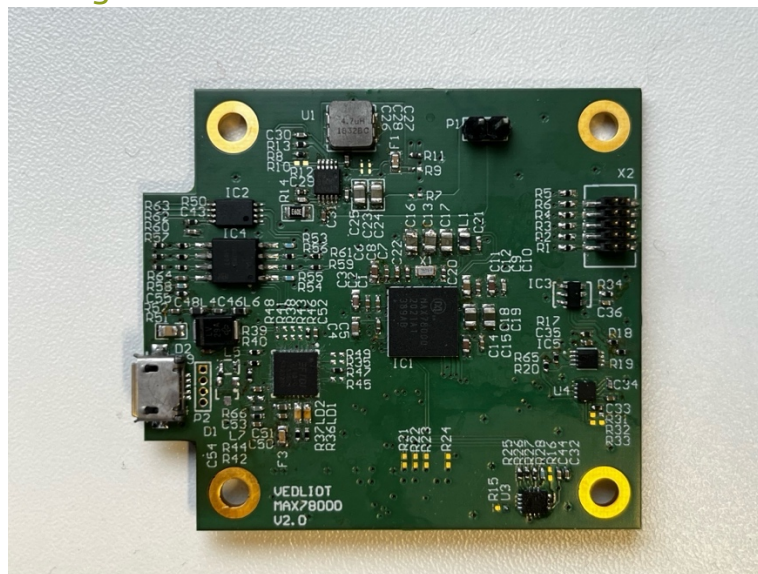


Figure 6 SFD with MAX78000, version two

The main result from task 4.8 is the hardware prototype on printed circuit boards (PCB) shown in **Error! Reference source not found.** Figure 6. This showcase is created following hardware design process:

1. Specification and requirements: determine the functionality of the hardware and requirements for its target system.
2. Microcontroller and chips selection: the microcontroller is settled to MAX78000 due to its ultra-low power feature and integrated CNN accelerator.
3. Architecture design: details in section 3.2
4. Register transfer level (RTL) design and verification: the connection of all components is established in schematic design as described in section 3.3.
5. Synthesis and place-and-route: physical layer design of routing for the integrated circuit (IC).
6. Manufacturing: the physical design is printed on printed circuit boards (PCB).
7. Test and validation: components are soldered, and every module of the hardware are to be tested for hardware validation.
8. Iteration: issues can be identified from the validation step and hardware design can be evolved to newer version. Besides, changes of requirements can also lead to iteration of the hardware design.

The analysis of the target system motor condition monitoring is described in the Deliverable 7.2 [9]. Besides, to accommodate general requirements for SFDs, the cognitive device “embedded accelerator” or short “EmbedX” is designed to be used in three different operational modes:

- Evaluation mode: The device can be connected via USB to a PC. The input data for the model can either be recorded by the sensors or fed via USB / UART.
- Extension mode: The EmbedX is stacked on an SSI-Web system (refer to Figure 4, an example of SFD device). The input data for the model are provided by the SSI-Web System. The Device is only used for model calculation and switched off again when the result is transmitted.
- Standalone mode: The input data for the model are recorded by the internal sensors. The calculated results can be stored in the external memory.

## 3.2 Hardware Overview

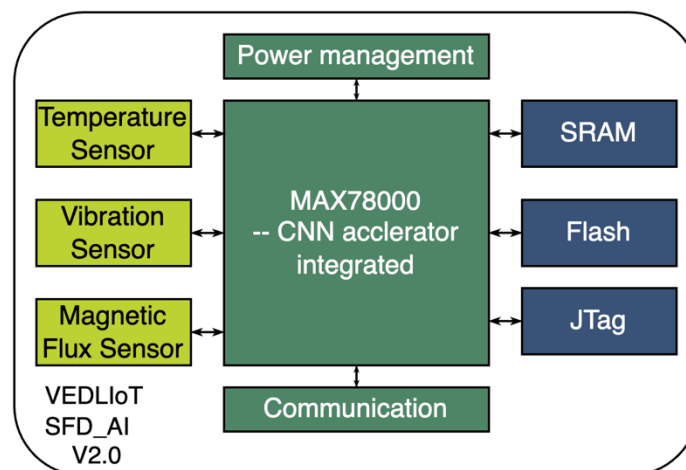


Figure 7 Block diagram of the SFD with MAX78000 as core accelerator

The hardware is designed for the use case motor condition monitoring as described in Deliverable 7.3 [10]. The design concept is shown in the block diagram in Figure 7.

At the center of the diagram shows the core processor MAX78000 of the cognitive IoT device. The processor is integrated with a CNN accelerator. For the sensing and data collection, three sensors shown on the left side are connected directly to the processor: temperature sensor, vibration sensor and magnetic flux sensor. Besides, a communication interface is reserved for further extensions. Same as the SFD mentioned before, a power management module is also integrated to provide various voltage options to supply different components. Despite these essential components, external SRAM and flash are added for extending the capacity of the core processor. JTAG module is attached for flashing and software debugging.

## 3.3 Design Schematic

### 3.3.1 MAX78000 Schematic

The MAX78000 block diagram is depicted in Figure 8. It has in addition to its conventional cortex M4 core, an embedded CNN accelerator with 64 parallel processors. This CNN engine has a SRAM based weight storage memory with 442 KB and a data storage memory of 512 KB to enable fast vector calculations [11].

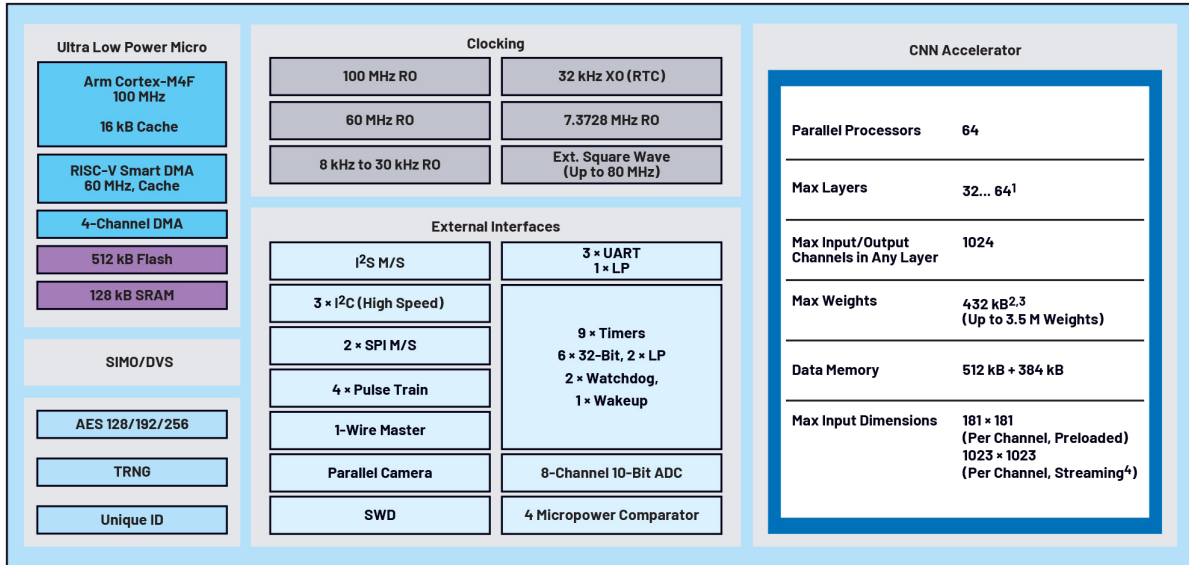


Figure 8 MAX78000 Block Diagram [12]

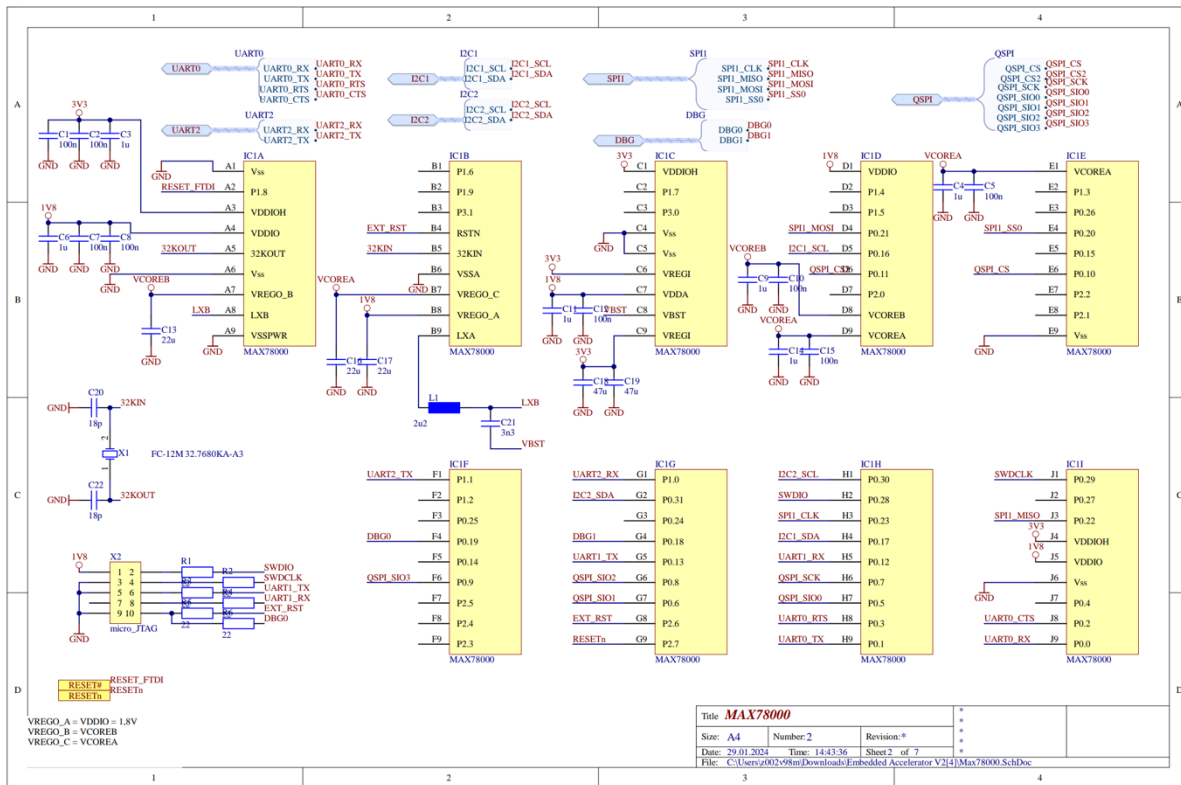


Figure 9 RTL design of MAX78000 chip with peripherals

Based on the official data sheet of MAX78000 [11], we connected all components with their communication interfaces and verified these connections carefully. The schematic design is shown in Figure 9.

### 3.3.2 Power Supply Schematic

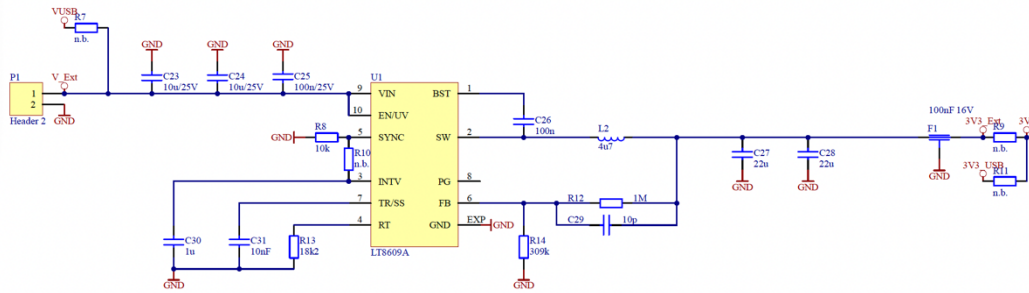


Figure 10 RTL design of power supply module

The power supply schematic contains an Analog Devices LT8609A [13], which is configured to provide 3.3V to the board from an external 5V voltage input. This is essential for the normal operation of all connected components.

### 3.3.3 SRAM and Flash Memory Schematic

The microcontroller MAX78000 comes with limited memory option. The integrated SRAM is 128kB, and the flash memory is only 512kB. Therefore, external SRAM and flash memory are provided on board to enable potential for more application scenarios. The external RAM selected to be on the embedded accelerator board is the Onsemi N01S830HAT221 [14] with 1MB storage space and the flash is MT25QU128ABA1ESE-0SIT [15].

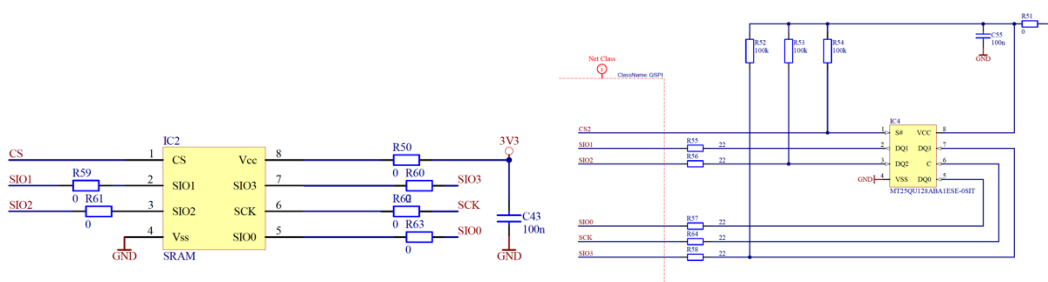


Figure 11 Schematic design for external SRAM and flash memory

External RAM and flash are connected to the MAX78000 with Quad Serial Peripheral Interface (QSPI). QSPI is a communication protocol that facilitates high-speed, full-duplex, and serial communication between a microcontroller and external peripherals. QSPI offers several advantages for connecting external RAM and Flash such as high speed, efficiency, flexibility with reduced number of pins.

### 3.3.4 Sensors Schematic

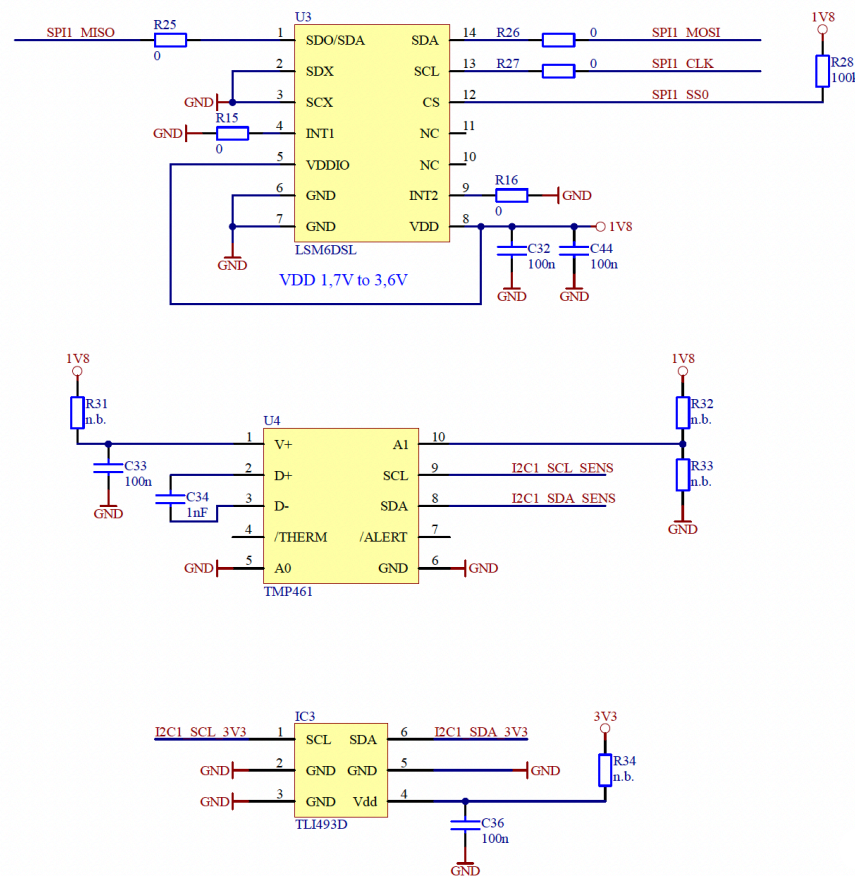


Figure 12 Schematic design of the vibration, temperature, and hall-effect sensor

As described in the hardware overview, three sensors are integrated in the IC and their schematics are presented in Figure 12. From top to bottom, the schematic shows the usage of the vibration sensor, the temperature sensor, and the magnetic field sensor.

The vibration sensor LSM6DSL from STMicroelectronics can capture an accelerometer and gyroscope of three dimensions. The sensor offers unparalleled precision in detecting and interpreting movement and orientation. Data collection from the sensor serves as a cornerstone in applications such as motion analysis, gesture recognition, and vibration characteristic of the attached surface. In order to obtain vibration information in terms of amplitude and frequency, the data need to be collected through serial peripheral interface (SPI), converted to frequency domain with techniques such as Fast Fourier Transform (FFT) and filtered.

For temperature sensing, TMP461 from Texas Instruments is integrated. Renowned for its precise temperature measurement capabilities, this sensor operates across an extensive temperature range. On the EmbedX, Inter-Integrated Circuit protocol is used for communication between the sensor and the controller.

As for the hall-effect sensor, TLI493D from Infineon Technologies is selected. It is capable of measuring magnetic fields with high precision and sensitivity. For motor condition monitoring applications, data concerning magnetic fields can provide information about rotation speed, rotor position and furthermore fault detection.

### 3.3.5 Communication Schematic

Three type of communication interface can be found on board

- A JTAG (Joint Test Action Group) connector (in Figure 9 left bottom corner) for debugging, testing, and programming the integrated circuit.
- A USB connector with UART (Universal Asynchronous Receiver / Transmitter) interface shown in Figure 13 for data exchange between PC and the board. This is utilized especially under evaluation mode.
- A pin array shown in Figure 14 that exposes mainly the unused communication interface of MAX78000. This is designed either as the extension interface of the external module in standalone mode or the communication interface with the main controller in extension mode.

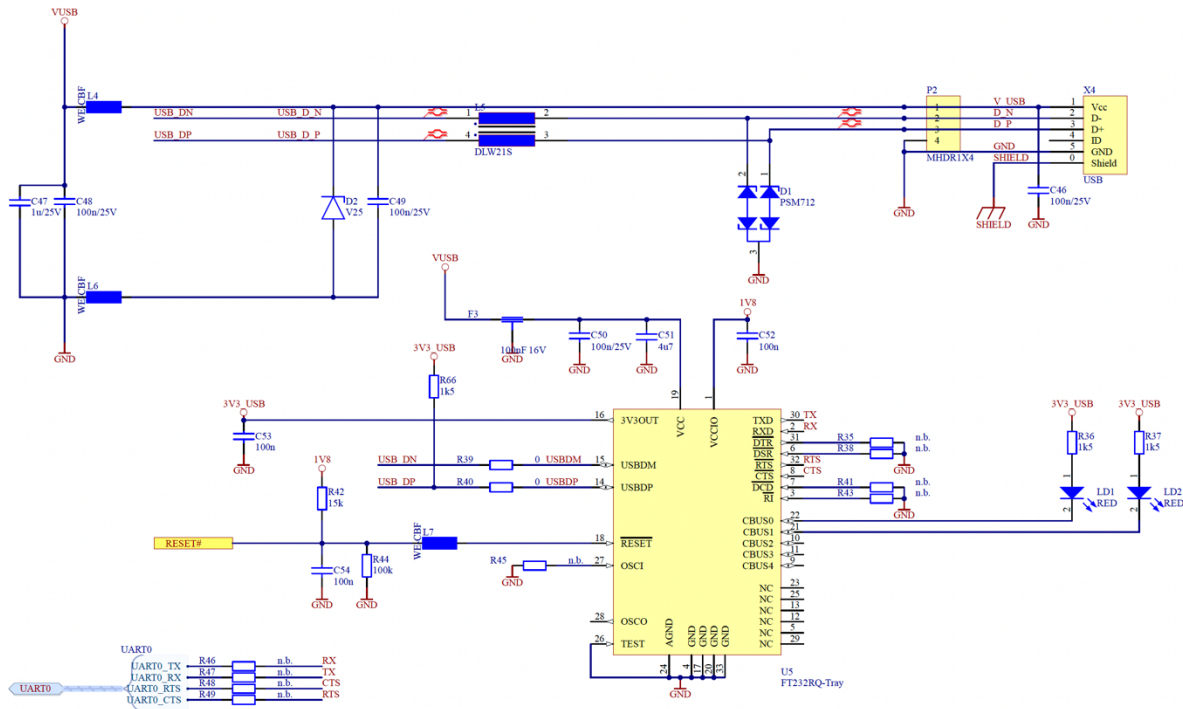


Figure 13 Schematic of the USB connector

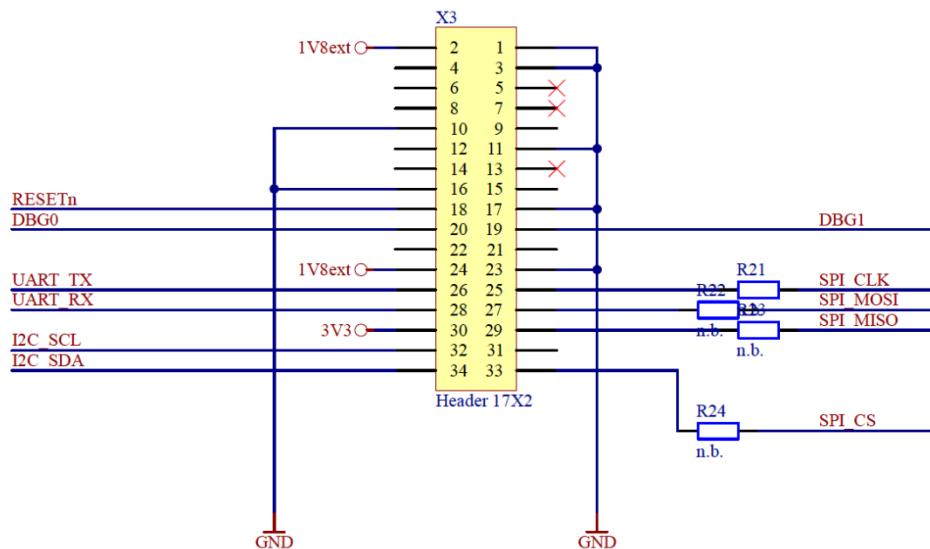


Figure 14 Schematic design of communication interface

External communication module such as Wi-Fi or LoRaWAN (long range wide area network) is not integrated in this integrated circuit because of two reasons. Firstly, the Wi-Fi module consumes high energy in operation and also produces heat that can affect temperature measuring on board. Secondly, in real applications, the hardware operates as an extension for the main board, which is already equipped with a wireless communication module.

## 4 Software Workflow

This section presents details on tools and methods for working with the embedded accelerator. The code snippets in this chapter are based on the motor condition monitoring use case. Figure 15 provides an overview of the workflow for the given hardware with AI power. The workflow is built based on the software package from MaximIntegratedAI [16]. There are three main steps: training, synthesis, and deployment. The training and synthesis are conducted on PC with the given software package, mainly in the programming language python. The deployment concerns development of embedded systems with program language embedded C. In the workflow, green blocks represent the open-source software package as development tool, and the orange blocks are information or files that need to be provided by developers, and they need to be customized based on different use cases.

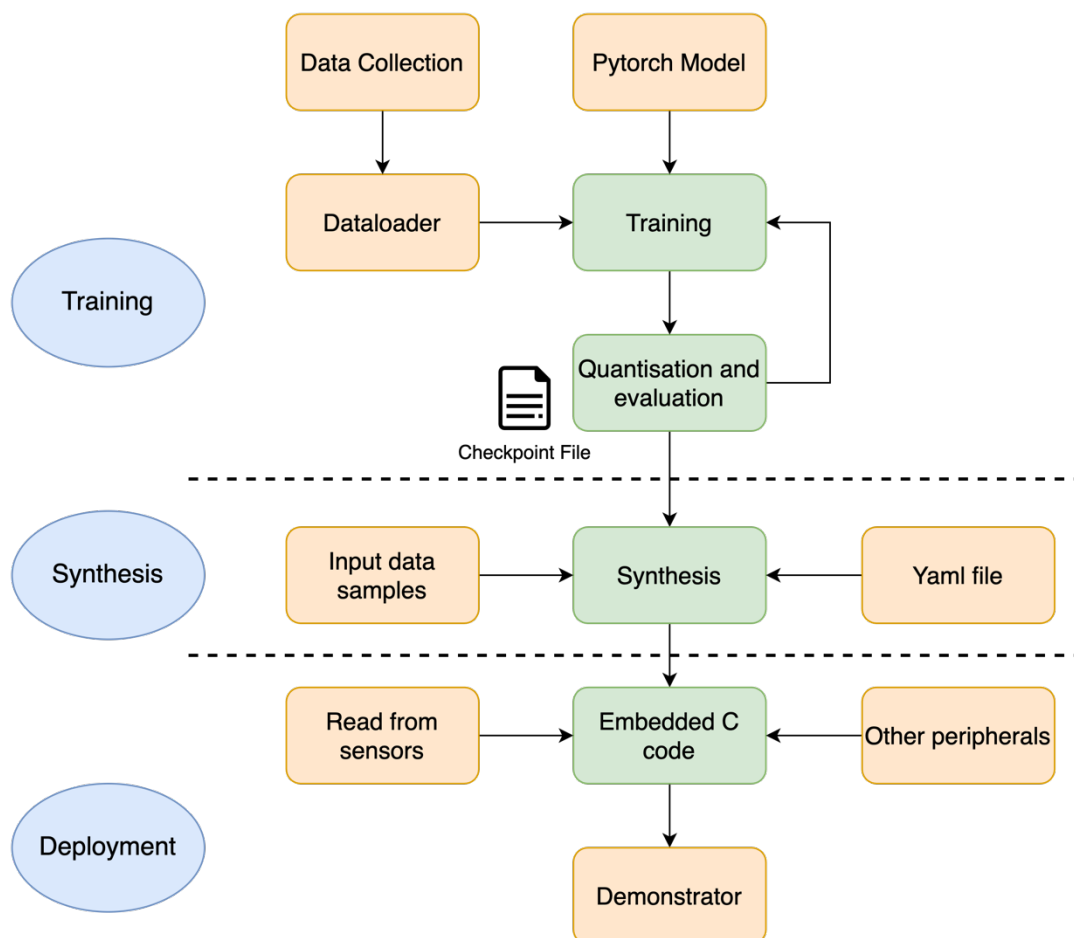


Figure 15 Overview of software workflow, based on software package from [16]

### 4.1 Training

The training procedure is the same as machine learning model training in common scenario. The only difference is that the training algorithm is provided by Maxim Integrated AI, which is based on Pytorch library. The training requires three main inputs, the dataset for the use case, a data loader and the pytorch model definition in certain form as interface for the training procedure.

Listing 1 shows part of the data loader definition. This function reads from given csv files and encapsulates data to dataflow that can be used for the training. In applications, data processing can also be conducted in this function.



---

```

def motor_get_datasets(data , load_train=True, load_test=True):
    (root_dir, args) = data
    transform = transforms.Compose([
        ai8x.normalize(args=args)
    ])
    if load_train:
        train_dataset = read_data(root_dir=root_dir,csv_file= '~/ai8x-training/data/motor1/motor_dataset.csv',
            transform = transform,d_type='train')
    else:
        train_dataset = None
    if load_test:
        test_dataset = read_data(root_dir= root_dir,csv_file= '~/ai8x-training/data/motor1/motor_dataset.csv',
            transform = transform,d_type='test')
    else:
        test_dataset = None
    return train_dataset , test_dataset

```

---

*Listing 1 Example of data loader*

Another key component is the Pytorch model file that structure like the following code snippet in Listing 2. The model structure is specified the same as pytorch model, only with functions from “ai8x” that have been adapted and optimized for CNN accelerator target training. This example shows a model architecture with three layers, two convolution layers with max pool layer each, and a dense layer as output layer.

---

```

class AI85motorNet2_on_off_off_on(nn.Module):
    def __init__(self,num_classes=3,num_channels=1,dimensions=(128, 1),bias=False,**kwargs):
        super().__init__()
        self.drop = nn.Dropout(p=0.2)
        # Time: 128 Feature : 3
        self.current_conv1 = ai8x.FusedConv1dReLU(num_channels, 64, 3, stride=1, padding=0,bias=bias,
**kwargs)
        # T: 126 F: 64
        self.motor_conv1 = ai8x.FusedMaxPoolConv1dReLU(64, 32, 2, stride=1, padding=1,bias=bias, **kwargs)
        # pool-stride is 2 and pool-padding is 0
        # T : 64 F: 16
        self.current_conv2 = ai8x.FusedConv1dReLU(32, 16, 3, stride=1, padding=0,bias=bias, **kwargs)
        # T: 62 F: 16
        self.motor_conv2 = ai8x.FusedMaxPoolConv1dReLU(16, 16, 2, stride=1, padding=1,bias=bias, **kwargs)
        # T : 32 F: 16
        self.fc = ai8x.Linear(512, num_classes, bias=bias, wide=True, **kwargs)

```

---

*Listing 2 Example of model definition for the motor use case*

With prerequisite ready, python script is executed for model training. An example of command is shown below. In this command, the train.py script reads data through the given data loader and trains the model defined in the definition file. Besides, hyperparameters for training such as epoch number, optimizer and learning rate are specified here.

---

```

python train.py --epochs 200 --optimizer Adam --lr 0.001 --deterministic --model
model_definition --dataset data_loader --data data/MCM --confusion --save-sample
10 --device MAX78000 "$@"

```

---

*Listing 3 Command for model training*

In this step, the model also needs to be quantized for further steps. The MAX78000 supports a symmetric quantization method that is based on the maximum and minimum weights. This method can be used with 1-, 2-, 4-, and 8-bit signed integers, while the weights can be configured on a per-layer basis.

The hardware implementation of the MAX78000 always utilizes signed integers for weights. However, during normal training process, floating-point values are commonly used for both data and weights, with the values being clipped (i.e., clamped) to a specific range. When using 8-bit quantization, the weight memory requirement is reduced by a factor of four. This decrease in memory size comes at the cost of a reduction in model accuracy. A way to assess the impact of weight quantization on model accuracy is to evaluate the quantized model using the test set that was used during training and compare its performance metrics, such as accuracy, with the performance of the original model. An evaluation of the impact of quantization on accuracy is presented in Deliverable 7.5 for the motor condition monitoring use case.

The quantization can be done with following command and model after quantization is save in the checkpoint file under given path "trained/ai85-motor-best-quantize.pth.tar". The checkpoint contains weight information and is further used in the next step synthesis.

---

```
python quantize.py trained/ai85_motor_best.pth.tar trained/ai85-motor-best-quantize.pth.tar --device MAX78000 -v "$@"
```

---

*Listing 4 Command for quantization*

## 4.2 Synthesis

During synthesis, the input data samples and a yaml file for model description are required. The input data samples are generated during the training process. And the model description in the yaml file as shown in Listing 5 must match the network that was used for training. This yaml file specifies the processors to be used, as well as the memory offsets for input and output layer for efficient neural network execution.

---

```
arch: ai85motornet2_on_off_off_on
dataset: motor2_on_off_off_on
```

```
# Define layer parameters in order of the layer sequence
```

```
layers:
```

```
  # Conv 1D - 4 layers
```

```
  - pad: 0
```

```
    activate: ReLU
```

```
    out_offset: 0x4000
```

```
    processors: 0x0000000000000001
```

```
    data_format: HWC
```

```
    operation: Conv1d
```

```
    kernel_size: 3
```

```
  - max_pool: 2
```

```
    pool_stride: 2
```

```
    pad: 1
```

```
    activate: ReLU
```

```
    out_offset: 0x00
```

```
    processors: 0x000000000000ffff
```

```
    operation: Conv1d
```

```
    kernel_size: 2
```

---

```

- flatten: true
  out_offset: 0x4000
  processors: 0x00000000000000ff
  operation: MLP
  output_width: 32
  activate: None
  
```

Listing 5 The yaml file for model description

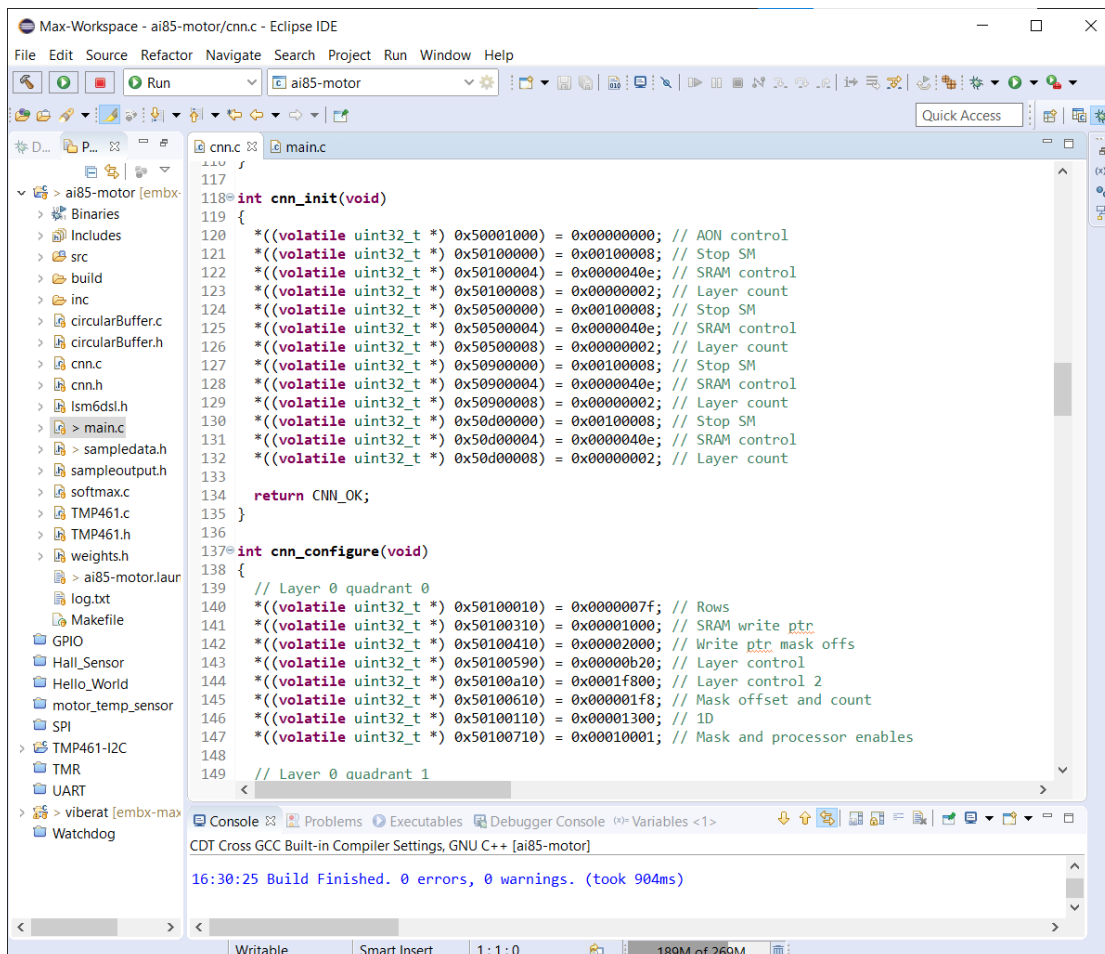
```

python ai8xize.py --verbose --test-dir sdk/Examples/MAX78000/CNN --prefix ai85-
motor --checkpoint-file trained/ai85-motor-best-quantize.pth.tar --config-file
networks/ai85-motor-hwc3_on_off_off_on.yaml --device MAX78000 --compact-data --
softmax --timer 0 --display-checkpoint
  
```

Listing 6 Command for synthesis -- conversion to C code

### 4.3 Deployment

The C-code, which is generated from the synthesis step, contains all information for the execution of the neural network. Including the structure, the quantized weights, and the processor plan for the computation of each layer. To complete the program for the applications, drivers for peripherals such as sensors are required same as functions for neural network inference. The development of the application is conducted with the MaximSDK [8] as integrated development environment (IDE) as shown in Figure 16.



```

117
118 int cnn_init(void)
119 {
120  *((volatile uint32_t *) 0x50001000) = 0x00000000; // AON control
121  *((volatile uint32_t *) 0x50100000) = 0x00100008; // Stop SM
122  *((volatile uint32_t *) 0x50100004) = 0x00000040e; // SRAM control
123  *((volatile uint32_t *) 0x50100008) = 0x00000002; // Layer count
124  *((volatile uint32_t *) 0x50500000) = 0x00100008; // Stop SM
125  *((volatile uint32_t *) 0x50500004) = 0x00000040e; // SRAM control
126  *((volatile uint32_t *) 0x50500008) = 0x00000002; // Layer count
127  *((volatile uint32_t *) 0x50900000) = 0x00100008; // Stop SM
128  *((volatile uint32_t *) 0x50900004) = 0x00000040e; // SRAM control
129  *((volatile uint32_t *) 0x50900008) = 0x00000002; // Layer count
130  *((volatile uint32_t *) 0x50d00000) = 0x00100008; // Stop SM
131  *((volatile uint32_t *) 0x50d00004) = 0x00000040e; // SRAM control
132  *((volatile uint32_t *) 0x50d00008) = 0x00000002; // Layer count
133
134  return CNN_OK;
135 }
136
137 int cnn_configure(void)
138 {
139  // Layer 0 quadrant 0
140  *((volatile uint32_t *) 0x50100010) = 0x0000007f; // Rows
141  *((volatile uint32_t *) 0x501000310) = 0x00001000; // SRAM write ptr
142  *((volatile uint32_t *) 0x501000410) = 0x00002000; // Write ptr mask offs
143  *((volatile uint32_t *) 0x501000590) = 0x00000b20; // Layer control
144  *((volatile uint32_t *) 0x501000a10) = 0x0001f800; // Layer control 2
145  *((volatile uint32_t *) 0x501000610) = 0x000001f8; // Mask offset and count
146  *((volatile uint32_t *) 0x50100110) = 0x00001300; // 1D
147  *((volatile uint32_t *) 0x50100710) = 0x00010001; // Mask and processor enables
148
149  // Layer 0 quadrant 1
  
```

Figure 16 MaximSDK as workspace, layer execution plan is show in in cnn\_configure function

## 5 Evaluation

The key metric for system evaluation is the runtime and energy consumption during model inference. These numbers depend highly on the model size and algorithm design. The evaluation in Table 1 Runtime scheduling for the SFD SSI-Web is the energy estimation for the product SIMOTIC [6] based on the scheduling in Table 1. Estimated energy consumption for the SFD is around 47 mWh per day when the measurement is conducted three times per hour and data is sent through Wi-Fi module once per day. The SFD is set to idle mode for the rest of time.

Input Parameters	
Measurement Interval (s)	1200
Communication Interval (s)	86400
Battery Capacity (Wh)	28.8

Table 1 Runtime scheduling for the SFD SSI-Web

SSI-Web	Energy Usage (mWh/d)
Sleep mode	4.07
Measurement (active mode)	4
Communication (with Wi-Fi)	16.20
Total (ideal)	24.27

Table 2 Energy estimation of the SFD SSI-Web

The energy estimation of the EmbedX is based on the motor condition monitoring use case. There are three parts that need to be included in the energy calculation: the CNN accelerator for the inference, the MAX78000 microcontroller and the sensors on board. Two modes are taken into consideration: active mode and standby mode.

**Error! Reference source not found.** summarizes the power and energy consumption for each part under different modes. To make the results comparable to the baseline system SSI-Web, the algorithm scheduling is set to be the same as in Table 1.

Assuming the system does the measurement three times an hour, and the microcontroller is up for 4 seconds with all peripherals on. Therefore, the system is in standby mode for 3588 seconds for every hour. In standby mode, the power consumption of the MAX78000 is

$$11.3\mu\text{A} \times 3.3\text{V} = 0.03729\text{mW}$$

where  $11.3\mu\text{A}$  is the current flow during standby mode [11]. This makes the energy usage per day 0.89mWh.

$$\frac{3588\text{s}}{3600\text{s}} * 24\text{h} * 0.03729\text{mW} = 0.89\text{mWh}$$

In comparison, the microcontroller STM32L4A6 has power consumption of  $11.8\mu\text{A} \times 3.3\text{V} = 0.03894\text{mW}$  under the Stop 1 mode [17] and makes the energy usage 0.93mWh per day.

Since the main peripheral components are identical on both boards for this use case, we assume that the energy consumption of those peripherals under comparable conditions are the same. Thus, we estimate the energy usage of EmbedX under standby mode to be

$$4.07\text{mWh/d} - 0.93\text{mWh/d} + 0.89\text{mWh/d} = 4.03\text{mWh/d}$$

For the active mode, similar estimation can be conducted. The system is active 72 times per day, 4 seconds each time. When operating under 80MHz, the microcontroller MAX78000 consumes 0.88mWh per day.

$$41.9\mu\text{A/MHz} \times 80\text{MHz} \times 3.3\text{V} \times 24\text{h} \times \frac{12}{3600} = 0.88\text{mWh}$$

Under the same condition, the microcontroller STM32L4A6 consumes 2.28mWh per day.

$$108\mu\text{A/MHz} \times 80\text{MHz} \times 3.3\text{V} \times 24\text{h} \times \frac{12}{3600} = 2.28\text{mWh}$$

In addition, CNN inference energy consumption is not included in MAX78000. Referring to the document of MAX78000 [11], the energy consumption of CNN accelerator is 4.02pJ/MAC, where MAC refers to multiple-accumulate operation in the neural network. For a small model shown in Listing 2, the energy is negligible in comparison.

<b>EmbedX</b>	<b>Energy Usage (mWh/d)</b>
Standby mode	4.03
Measurement (active mode)	2.60

*Table 3 Energy estimation of EmbedX for the motor use case*

The target hardware with its ultra-low-power feature can achieve lower energy consumption compared to the baseline system with STM32. One reason for this is that the MAX78000 has fewer peripheral units and components in its core compared to the STM32.

## 6 Conclusion

This deliverable aims at the optimization of cognitive IoT device in order to integrate AI power in end device, especially on embedded systems in far-edge. This enables more intelligent on-site data processing, and thus reduces the amount of data to be transmitted. This does not only reduce the congestion in IoT networks of large scale, but also release the burden on central computing.

The embedded hardware system presented in this work integrated the ultra-low power microcontroller MAX78000 and multiple peripherals for customized functions for target use case, motor condition monitoring. The hardware is designed, manufactured, and tested. Toolchain for the software development is also evaluated and demonstrated on the use case. This work shows the possibility of integrating deep learning neural networks in embedded systems under different operational models. On the other hand, its energy efficiency is also an attractive feature for future implementation.

## 7 References

- [1] VEDLIoT EU Project, "Deliverable D7.5 Final report on use case development, optimisation, benchmarking and evaluation," 2024.
- [2] Siemens AG, "Sensor System Integration," Siemens AG, 2023.
- [3] VEDLIoT EU Project, "Deliverable D4.1 First report on cognitive IoT hardware platform and microserver development," 2022.
- [4] A. Mbuy, "Tractronic Sensformer@," Siemens AG, [Online]. Available: <https://blog.siemens.com/2018/09/tractronic-sensformer-born-connected-the-worlds-first-intelligent-traction-transformer/>. [Accessed 05 01 2024].
- [5] Siemens AG, "SITRANS SCM IQ," Siemens AG, [Online]. Available: <https://www.siemens.com/global/en/products/automation/process-instrumentation/digitalization/smart-condition-monitoring.html#Ourportfolio>. [Accessed 05 01 2024].
- [6] Siemens AG, "Konnektivität für SIMOTICS Niederspannungsmotoren," [Online]. Available: <https://www.siemens.com/de/de/produkte/antriebstechnik/digitalisierung-antriebstechnik/konnektivitaet.html>. [Accessed 05 01 2024].
- [7] VEDLIoT EU Project, "Deliverable D3.3 Evaluation of the DL accelerator designs," 2022.
- [8] Analog Devices, "MAX78000 Artificial Intelligence Microcontroller with Ultra-Low-Power Convolutional Neural Network Accelerator," Analog Device, [Online]. Available: <https://www.analog.com/en/products/max78000.html#product-overview>. [Accessed 05 01 2024].
- [9] VEDLIoT Project, "D7.2 First report on use case development and optimisation," 2022.
- [10] V. Project, "D7.3 Second report on use case development and optimisation," 2022.
- [11] Analog Devices, "Artificial Intelligence Microcontroller with UltraLow-Power Convolutional Neural Network Accelerator," 5 2021. [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/MAX78000.pdf>. [Accessed 05 01 2024].
- [12] O. Dreessen, "Hardware conversion of convolutional neural networks," 18 07 2023. [Online]. Available: <https://www.embedded.com/hardware-conversion-of-convolutional-neural-networks/>. [Accessed 05 01 2024].
- [13] Analog Devices, "LT8609 LT8609A LT8609B Datasheet," 08 2021. [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/lt8609-8609a-8609b.pdf>. [Accessed 05 01 2024].
- [14] ON Semiconductor, "N01S830HA Mb Ultra-Low Power Serial SRAM," 2 2016. [Online]. Available: <https://www.onsemi.com/pdf/datasheet/n01s830ha-d.pdf>. [Accessed 05 01 2024].
- [15] Micro Technology, "Micron Serial NOR Flash Memory," 01 2013. [Online]. Available: [https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/nor-flash/serial-nor/n25q/n25q\\_128mb\\_1\\_8v\\_65nm.pdf](https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/nor-flash/serial-nor/n25q/n25q_128mb_1_8v_65nm.pdf). [Accessed 05 01 2024].

- [16] MaximIntegratedAI, "ADI MAX78000/MAX78002 Model Training and Synthesis," [Online]. Available: <https://github.com/MaximIntegratedAI/ai8x-training>. [Accessed 05 01 2024].
- [17] STMicroelectronics, "STM32L4A6xG Datasheet," 07 2022. [Online]. Available: <https://www.st.com/resource/en/datasheet/stm32l4a6ag.pdf>. [Accessed 29 01 2024].



## 8 List of Figures

Figure 1 Architectures for industrial IoT [2] .....	5
Figure 2 Overview of the VEDLIoT project with WP4 highlighted with red box [3].....	6
Figure 3 Battery powered SFD.....	7
Figure 4 An instance of the SFD .....	7
Figure 5 Simotics Connect SFD (left) and IoT system based on it(right) [6] .....	8
Figure 6 SFD with MAX78000, version two .....	9
Figure 7 Block diagram of the SFD with MAX78000 as core accelerator .....	10
Figure 8 MAX78000 Block Diagram [12] .....	11
Figure 9 RTL design of MAX78000 chip with peripherals .....	11
Figure 10 RTL design of power supply module.....	12
Figure 11 Schematic design for external SRAM and flash memory .....	12
Figure 12 Schematic design of the vibration, temperature and hall-effect sensor.....	13
Figure 13 Schematic of the USB connector .....	14
Figure 14 Schematic design of communication interface.....	14
Figure 15 Overview of software workflow, based on software package from [16].....	16
Figure 16 MaximSDK as workspace, layer execution plan is show in in cnn_configure function .....	19

## 9 List of Listings

Listing 1 Example of data loader .....	17
Listing 2 Example of model definition for the motor use case.....	17
Listing 3 Command for model training.....	18
Listing 4 Command for quantization .....	18
Listing 5 The yaml file for model description.....	19
Listing 6 Command for synthesis -- conversion to C code .....	19

### 10 List of Tables

Table 1 Runtime scheduling for the SFD SSI-Web ..... 20  
Table 2 Energy estimation of the SFD SSI-Web ..... 20  
Table 3 Energy estimation of EmbedX for the motor use case ..... 21