Proceedings of International Conference on Technology and Social Science 2024(ICTSS 2024)

Peripheral Specific Hardware Interface Architecture for Super General Purpose SoC

Hibiki Shinozaki[1, a], Akira Yamawaki[1,b]

[1]Department of Electrical Engineering and Electronics, Kyushu Institute of Technology, 1-1 Sensui cho, Tobata-ku, Kitakyushu City, Fukuoka, 804-8550, Japan

[a]<shinozaki.hibiki457@mail.kyutech.jp>, [b]<yama@ecs.kyutech.ac.jp

Keywords: general purpose system on chip, dynamic reconfiguration, register transfer level description, high-level synthesis

Abstract. The progress of the Internet of Things, IoT, has led to the creation of a wide range of products. However, the development costs of new System on Chips, SoCs, are increasing due to technological innovation and miniaturization. Therefore, we aim to develop a super general purpose SoC with hardware interfaces compatible with all kinds of peripherals. The realization of this SoC will eliminate the need to develop a new SoC, resulting in reduced development costs. A super general purpose SoC supporting all peripherals dynamically reconfigures dedicated hardware interfaces for each peripheral. This paper proposes an architecture in which the procedure-based part is designed by high-level synthesis and the time-based part by register transfer level description, and each part cooperates using a unified interface and data structure. This paper evaluates the generality and performance of the proposed architecture.

1. Introduction

We are working towards the realization of a super general System on Chip, EPoC [1], which virtually has dedicated hardware interfaces for all peripherals and physically reconfigures [2-6] the hardware interfaces for peripherals dynamically according to the application behavior. We are also aiming to formulate a concise and general hardware interface architecture to enrich the hardware interface library. Hardware interface libraries store device drivers as hardware circuit configuration data rather than software execution files. In the development of hardware-oriented device drivers, high-level synthesis, HLS [7-10], is used to automatically generate hardware descriptions from software to achieve rapid and flexible development. As a design policy, register transfer level description, RTL description, is used for the parts that require process on a clock cycle basis, while HLS is used for the parts that can be described on a procedure basis, although the operation is complicated.

In this paper, we propose a general dedicated hardware interface architecture that is independent of real peripherals. As an application example, the proposed architecture is used to convert device drivers for the BME280 and BME680, real SPI devices, to hardware and the generality of the proposed architecture is evaluated. The performance of the proposed architecture is evaluated through demonstration experiments on actual devices.

We organize this paper as follows. In section 2, we present an overview of EPoC. Section 3 proposes the architecture of the hardware interface dedicated to peripherals. In section 4, to ascertain its flexibility, we apply it to the realization of the device drivers for the BME280 and BME680. Section 5 presents the validation results and discussion, and finally section 6 concludes the thesis.

2. Super General Purpose System on Chip, EPoC

2.1 Virtual EPoC

Embedded devices require a dedicated System on Chip, SoC, to be designed from scratch for each product, leading to increased development costs. EPoC is considered useful for reducing these costs.

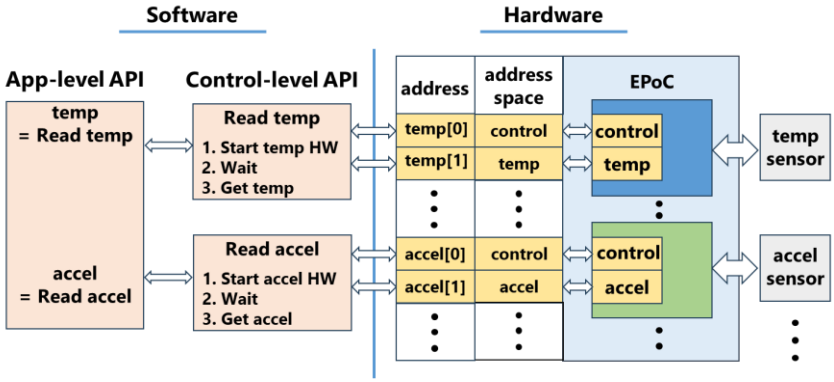

Fig. 1. Software and hardware organization of a virtual EPoC

EPoC is a SoC with memory mapped registers, MMR, and hardware interfaces for all peripherals. Fig. 1 shows the software and hardware organization of a virtual EPoC. This consists of application- and control-level Application Programming Interface, API, on the software side and interfaces optimized for peripherals on the hardware side. The two communicate synchronously via MMR with a unified framework. Examples are control and status registers whose roles are identical, and data registers that are expressed only in terms of essential data, such as temperature or humidity.

On the hardware side, a dedicated interface converts information from the peripheral into the desired data, such as temperature and humidity, and writes it into the unified MMR. Therefore, the software developer uses only the application-level API and only acquires data such as temperature and humidity.

2.2 Physical EPoC with Dynamic Reconfiguration

Virtual EPoC with countless dedicated hardware interfaces are realized by dynamic reconfiguration. Fig. 2 shows the build concept of an application for EPoC. Compilation is done based on the executable program, circuit data library and connection status with sensors to generate a circuit configuration table, circuit data and a CPU execution file. The fixed part is the hardware that is commonly used for all processes in the execution file. The circuit configuration table has values held in MMR. The control level API in Fig. 1 dynamically reconfigures the hardware based on this table. Fig. 3 shows an overview of the dynamic reconfiguration in temperature acquisition. These are examples of the use of EPoC for reading out the values of the air temperature sensor and the acceleration sensor. In Fig. 3(a), the fixed part is configured from the reconfiguration port. Fig. 3(b) shows the hardware for the temperature is configured from the reconfiguration port. In Fig. 3(c), the temperature value acquisition hardware and the temperature sensor are connected, and in Fig. 3(d), the value is acquired and written to MMR. The same process applies to acceleration acquisition.

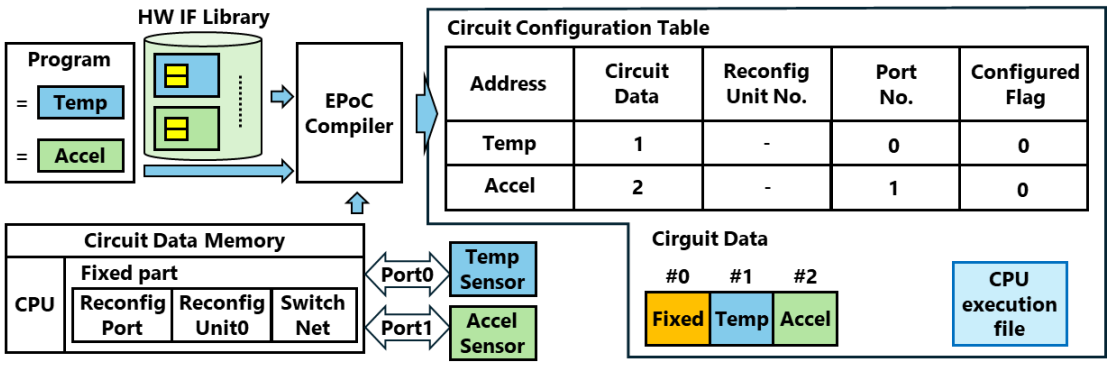

Fig. 2. Build concept of an application for EPoC

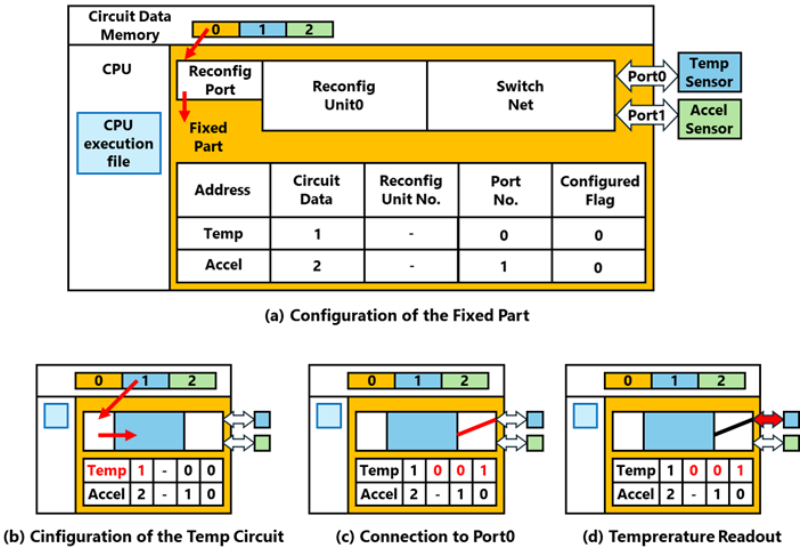

Fig. 3. Overview of dynamic reconfiguration

3. Peripheral-Specific Hardware Interface Architecture

3.1 Architecture Overview

Fig. 4 shows an overview of the hardware interface dedicated to peripherals. The structure consists of Comm Unit that actually communicates with the peripherals, Control Unit that enables various functions and HLS Unit that enables data processing.

The Comm Unit that drives communication with peripherals and the Control Unit that controls it need to be designed on a clock-cycle time basis. Therefore, it is designed and developed using RTL description and accesses peripherals using combinational circuits and sequential circuits. The Control Unit holds the data (read-write data, amount of data, etc.) necessary to realize the functions, and the Comm Unit communicates using these data. The output from the Control Unit are logical sum and the input is shared. The Comm Unit depends on the communication standard with the peripheral devices, and the Control Unit differs only in details, such as the required data values, which depend on the peripheral devices.

On the other hand, the HLS Unit, whose operation is complicated but can be described on a procedure basis, makes use of HLS. For example, the correction of sensor values is relevant. The burden of developing dedicated hardware interfaces can be significantly reduced because most of the dependence on peripherals can be developed by HLS. The presence or absence of the corresponding HLS module is determined according to the function to be realized, and if HLS Unit is not required, the control part is used as the first interface. This architecture consists of the three layers described above, and dedicated hardware interfaces can be developed by changing the modules in these layers according to the peripherals.

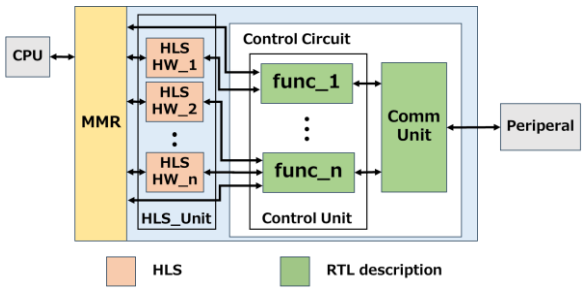

Fig. 4. Overview of hardware interface dedicated to peripherals

3.2 Hardware Operation Details

Fig. 5 shows an overview of the operation when correcting the acquired data using the proposed hardware interface shown in Fig. 4. The Control Unit starts the Comm Unit and transmits data for communication. The Comm Unit communicates with peripherals and returns acquisition data as appropriate. When all data has been acquired, the Control Unit notifies the HLS Unit of its completion, and the HLS Unit starts the data correction process. When the correction is completed, it writes the correction value to the MMR, notifies the Control Unit of its completion and returns to the initial state.

4. Application to Real Peripherals

To demonstrate the compatibility and scalability of the proposed architecture, it is necessary to ascertain the flexibility of HLS Unit, which is the dependent part on the peripherals. Therefore, we have developed a hardware interface for the case where HLS hardware, HLS HW, are divided for each data and signals that need to be shared are realized by connecting each one (Fig. 6(a)) and for the case where multiple data are processed by one HLS HW (Fig. 6(b)).

Fig. 6(a) shows the hardware interface when the proposed architecture is applied to the BME280. Fig. 7(a) shows the software descriptions for HLS and their roles of HLS Temp and HLS Press in Fig. 6(a). In HLS, input and output ports are defined by arguments. HLS HW uses the handshake, hs, to wait for the Control Unit to read raw sensor data, sd, and calibration data, cd. The output of HLS Temp can be used as input to HLS Press by including the argument hw in both functions. The data that has been processed is written to mmr, allowing the CPU to retrieve temperature and pressure data.

Fig. 6(b) shows the hardware interface when the proposed architecture is applied to the BME680. Fig. 7(b) shows the software description of the HLS Data. The processing of four types of data in a single HLS HW was achieved by defining each output port as an argument and writing each process in sequence in software.

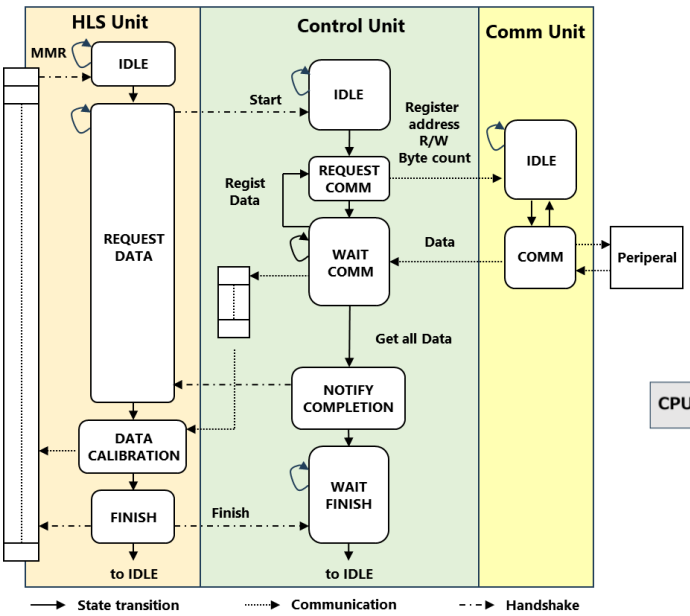

Fig. 5. Operation of the hardware interface dedicated to peripherals

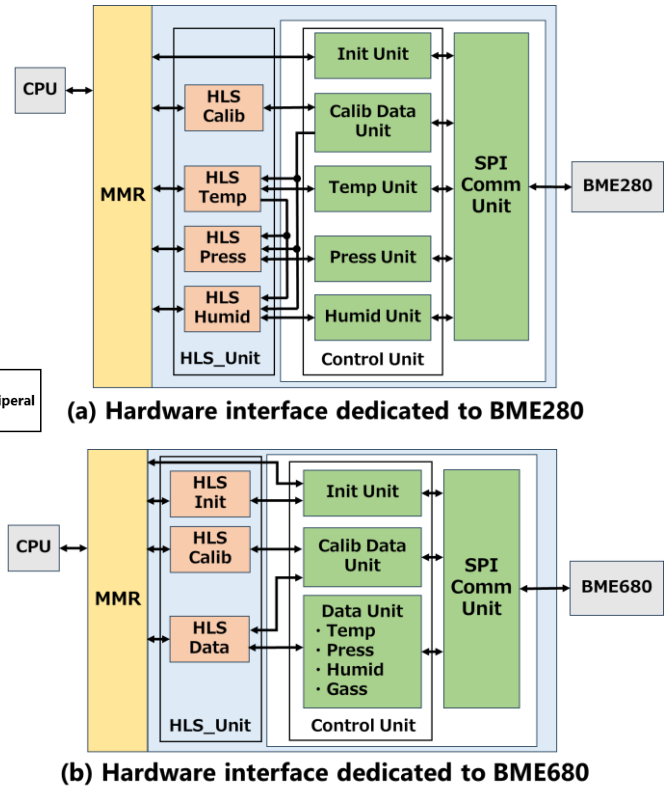

Fig. 6. Operation of the hardware interface dedicated to peripherals

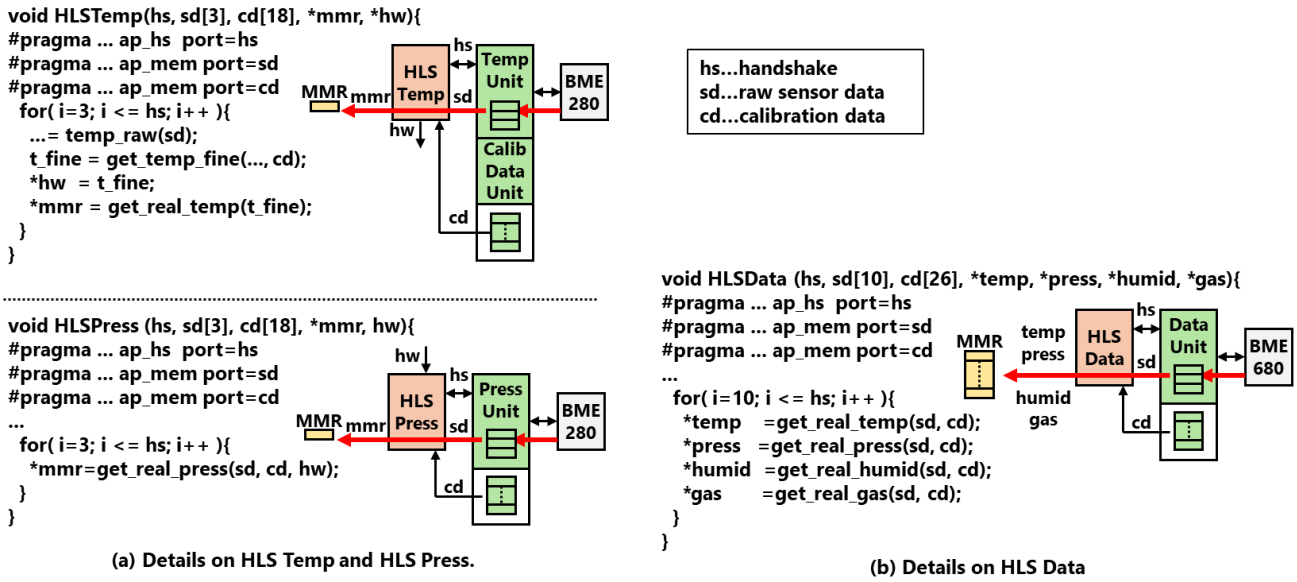

Fig. 7. software descriptions for HLS and their role

5. Experiments and Discussions

5.1 Experimental environment

The performance is evaluated by comparing a hardware interface dedicated to the BME280 and BME680 and a software-based readout using an interface for SPI communication. The Digilent ZYBO Z7-10 FPGA board from Digilent was used for the hardware implementation. The operating frequency of the FPGA is 100 MHz, while the operating frequency of the embedded CPU is 650 MHz. AE-BME280 from Akizuki Denshi Tsusho and Bosch Sensortec BME680 were used.

In development, HLS was done by using Xilinx's high-level synthesis application Vitis HLS 2022.2 and the software program was converted into a hardware description language. The generated HLS Unit, together and other units were converted into circuit data for writing into the FPGA using Xilinx's FPGA implementation tool Vivado 2022.2. The SPI communication frequency was set to 10 MHz.

5.2 Measurement Results

Fig. 8 shows the acquisition times of various data. Compared with software-based execution, the speed-up was 2.0, 1.7, 1.9 and 2.0 times respectively and execution time approached the ideal derived from the amount of communication. The reason for the speed-up is thought to be that the software overhead between byte transfers could be reduced by using hardware. Fig. 9 also shows that power efficiency has improved more than 11 times in all cases.

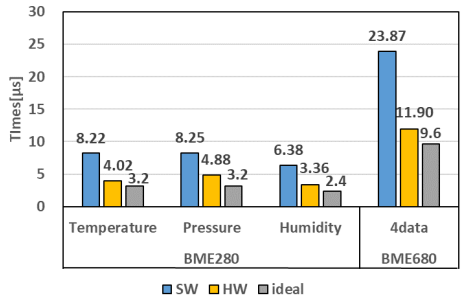

Fig. 8. Execution time

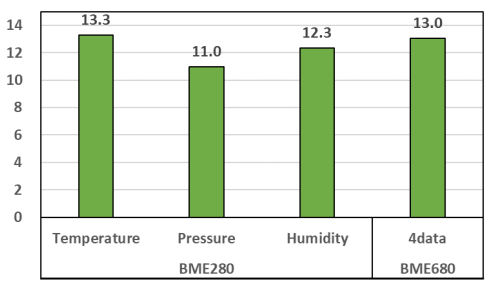

Fig. 9. Power efficiency

6. Conclusion

To enhance the dedicated hardware interface library optimized for each peripheral, this paper proposes an architecture that is easy to design and develop and highly scalable for modification, update and improvement. It uses RTL description for the parts where operation description was required on a clock cycle basis and HLS for complex, procedure-based tasks dependent on peripherals. This simplifies the development of dedicated hardware interfaces. We have confirmed that a dedicated hardware interface can be developed by developing internal modules corresponding to the BME280 and BME680 respectively, applying this architecture. As a result of the actual device verification, we achieved about 2-fold speed-up and 12-fold performance improvement for any case compared to software-based readout. In terms of compatibility and scalability, this paper also confirmed that HLS HW can be connected to each other and single HLS HW can process multiple data.

The communication standards and correction methods differ depending on the peripheral devices. However, we believe that a dedicated hardware interface can be developed by combining the Comm Unit, Control Unit and HLS Unit corresponding to them. To confirm this, we will apply this architecture to peripherals that use a communication standard different from SPI communication and develop a dedicated hardware interface.

References

[1] H. Shinozaki and A. Yamawaki, "Hardware Implementation of Calibration Data Loading in Device Driver for an SPI Peripheral", ***Proc. of the 2024 6th International Electronics Communication Conference,*** pp.19-23, 2024.

[2] L. Gong, C. Wang, X. Li and X. Zhou, " Improving HW/SW Adaptability for Accelerating CNNs on FPGAs Through A Dynamic/Static Co-Reconfiguration Approach", ***IEEE Transactions on Parallel and Distributed Systems***, Vol. 32 No. 7, pp1854-1865, 2020.

[3] D. Giri, K. Chiu, G. Eichler and P. Mantovani and L. Carloni "Accelerator Integration for Open-Source SoC Design", ***IEEE Micro,*** Vol.41, No.4, pp.8-14, 2021.

[4] C. Xu, S. Jiang, G. Luo, G. Sun, N. An, G. Huang and X. Liu "The Case for FPGA-Based Edge Computing", ***IEEE Transactions on Mobile Computing,*** Vol.21, No.7, pp.2610-2619, 2022.

[5] A. Dhar, E. Richter, M. Yu, W. Zuo, X. Wang, N. Kim and D. Chen " DML: Dynamic Partial Reconfiguration with Scalable Task Scheduling for Multi-Applications on FPGAs", ***IEEE Trans. on Comput.,*** Vol.71, No.10, pp.2577-2591, 2022.

[6] F. Restuccia, A. Meza, R. Kastner and J. Oberg " A Framework for Design, Verification, and Management of SoC Access Control Systems", ***IEEE Trans. on Comput.,*** Vol.72, No.2, pp.386-400, 2023.

[7] A. Moosa, N. Sarma and C. Karfa "ImageSpec: Efficient High-Level Synthesis of Image Processing Applications", ***Euromicro Conference on Digital System Design,*** 2022, https://ieeexplore.ieee.org/document/9996711

[8] A. Hosseiny and H. Jahanirad "Hardware acceleration of YOLOv7 tiny using high-level synthesis tools", ***Journal of Real-Time Image Processing,*** Vol.41, No.4, 2023.

[9] C. Sestito, S. Perri and R. Stewart "FPGA Design of Transposed Convolutions for Deep Learning Using High-Level Synthesis", ***J. Signal Process. Syst.,*** Vol.95, pp.1245-1263, 2023.

[10] H. Younes, A. Ibrahim, M. Rizk and M. Valle " Algorithmic-Level Approximate Tensorial SVM Using High-Level Synthesis on FPGA", ***Electronics,*** Vol.10, No.2, pp.205, 2021.