Multirate Model Parallelization for MPSoC with FPGA in Model-Based Development: A Case Study

Ryota Yamamoto[1,a)] Masataka Ogawa[1] Masahiro Oinuma[1] Masaki Kondo[2] Shinya Honda[3]
Masato Edahiro[1]

Abstract: This paper presents an integrated development environment, HS-MBP, to generate parallelized code and executables for MPSoC with FPGA, named Field-Programmable Heterogeneous SoC (FP-HSoC), in model-based development. In our environment, we propose an automatic code generation for communication mechanism between sub-models with different control periods. Using this mechanism, our environment is capable of executable generation from multirate models, which are hard to design when multirate systems like robot IoT are constructed on FP-HSoC. Our environment has been applied to an multirate model as a case study. The results show that HS-MBP successfully generated parallelized codes of multirate systems for several patterns of block assignment to FP-HSoC. With the results of the case study, we discuss workflow for model-based parallelization of multirate models on FP-HSoC and future work.

Keywords: Heterogeneous Multicore, MBD (Model Based Development), Multirate, FPGA, Co-Design

1. Intoroduction

In recent years, embedded control systems have become larger and more complex with a demand for shortening the Time To Market. In addition, higher computation applications are increasing because these systems are desired to equip intelligent functions such as machine learning in the future. Consequently, accelerative hardware platforms are indispensable with a development environment to achieve short design time.

Traditionally, embedded control systems have been implemented on single cores. Multicores are often utilized in current systems to perform high computation applications. However, the multicores cannot meet low energy consumption requirements in some systems especially for the higher computation applications, for example, automotive driving and computer visions in robot IoT systems. Therefore, heterogeneous multicore SoC with FPGA, named *Field-Programmable Heterogeneous SoC (FP-HSoC)*, is demanded to archive these requirements instead of multicores [8]. **Figure 1** shows a typical example of FP-HSoC targeted in this paper. It is costly and difficult to implement applications with FP-HSoC because developers have to optimize codes for each PE (Processing Elements) and implement inter-PE synchronization architectures.

As an approach to shorten the development period of embedded control systems, model-based development (MBD) is made use of to make the level of design abstraction higher [2]. In MBD, since developers design algorithms in a model with blocks, they can verify the model in algorithm levels [2]. Other advantages of MBD are the consistency between documents and source code,

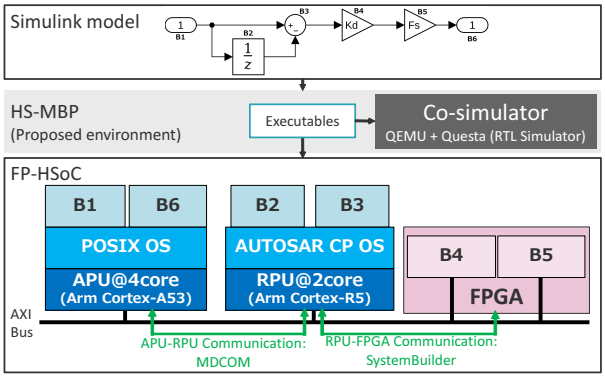


Fig. 1: Architecture of the FP-HSoC[16]

easy specification change, and high code reusability and so on [2].

MATLAB/Simulink [4], [5] is a typical MBD tool. Simulink models, which represent the process with graphic symbols, are inputs to generate sequential C language source code (i.e., for single cores) with Embedded Coder: C source code generator. With Embedded Coder, developers not only can reduce the coding time, but also keep the consistency between the model and the source code, and the source code can be updated only by changing the model.

If an automatic implementation for FP-HSoC is available in MBD, both development periods and performance can be improved. However, as a problem to utilize both of them at the same time, it is necessary to rewrite the sequential code (s-code) generated by Embedded Coder to parallelized code (p-code) for FP-HSoC, which requires a communication interface, optimization for each PE, and synchronization architectures.

For parallelization in MBD, we have been researching on model-based parallelization (MBP) for multicores with GPU

[1] Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] NEC Solution Innovators, Ltd., Kawasaki, Kanagawa 213–8511, Japan
[3] Nanzan University, Nagoya, Aichi 464–8673, Japan
[a)] muku@ertl.jp

environments [17], [18]. However, software-hardware co-generation for FP-HSoC was outside of the target system.

In this study, we develop the *HS-MBP (Heterogeneous System - MBP)* environment, which is an MBP environment targeting FP-HSoC. The HS-MBP, with the help of the HW/SW codesign environment: SystemBuilder [1], [3], provides communication functions between processors and code parallelization for FPGA with high-level synthesis (HLS). In order to verify the correctness of generated code and effectiveness of our environment, HS-MBP was tested with two multirate models as a case study.

Our contributions are as follows:

- We develop the HS-MBP environment, and generate p-code for FP-HSoC on the environment.
- HS-MBP generates p-code with APIs for inter-PE communication mechanism. As a result, developers unconsciously implement inter-PE communication automatically.
- To tune HW design, HS-MBP provides HW-SW co-simulation environmemt. Developers can measure the latency of HW design by the co-simulation environment easily.

2. FP-HSoC

Field-Programmable Heterogeneous SoC (FP-HSoC) consists of multiple processors with different architectures including FPGA. By selecting suitable FP-HSoC architecture for the application, energy consumption and execution time can be improved, compared to homogeneous multi-core architecture.

In this paper, we use Zynq UltraScale+ MPSoC (ZynqMP) as FP-HSoC, which consists of Application Processing Unit (APU), Realtime Processing Unit (RPU), and FPGA.

The PEs (Processing Elements) for the target of this paper (refer Figure 1) described as follows:

APU consists of ARM Cortex-A53 (4 Cores) @1.5GHz. This PE has L2 cache, and we assume that Linux runs on this PE.

RPU consists of ARM Cortex-R5 (2 Cores) @600MHz. This PE has a low-latency interrupt controller and local RAM access, and lockstep for functional safety. We assume that RTOS runs on this PE.

FPGA is used for high computation applications such as image processing and deep learning. Since it is hard to design in hardware description language (HDL), High-Level Synthesis (HLS) tools contribute to design hardware on FPGA.

3. HS-MBP

HS-MBP is an environment to generate executables for FP-HSoC. **Figure 2** shows HS-MBP flow. In this section, we describe the implementation of FP-HSoC at 3.1, and the communication mechanism between PEs at 3.2.

3.1 Design Flow

HS-MBP takes a Simulink model that can generate s-code as input. We describe the procedure of HS-MBP environment based on Figure 2, assuming the case of targeting APU, RPU and FPGA.

MBP supports generating parallelized code (p-code) for homogeneous multicores from Simulink model. The following steps from **P1-1** to **P1-6** show the procedure in MBP environment.

P1-1 At (mbp1) in Figure 2, s-code(d1) is generated automatically from a Simulink model by Embedded Coder available from MATLAB/Simulink.

P1-2 At (mbp2), BLXML (Block-Lebel XML, d2 shown in **Figure 3**) is generated from a Simulink model.

P1-3 At (mbp3), BLXML with code snippets (C-BLXML, d3 shown in **Figure 5**) is generated from BLXML (d2) and s-code.

P1-4 At (mbp4), a CSV file (d4) that has block names and block IDs is generated from C-BLXML (d3).

P1-5 At (mbp5), a new CSV file (d5) is created manually by adding PE assignment to the CSV file (d4), and BLXML with code snipets and PE assignments (CA-BLXML, d6 shown as **Figure 6**) is generated from (d3) and (d5) automatically.

P1-6 At (mbp6), p-code and interface definition files (d7) to (d11) are generated from (d6) automatically.

For above procedure, we developed the tools except for MATLAB/Simulink and Embedded Coder. HS-MBP is semi-automatic because automatic PE assignment tools for FP-HSoC is currently under development. Therefore, we assign PE manually in this study. In the future, when PE assignment tool is available, HS-MBP will be full-automatic.

Next, we descrive automatic executable generators **P2-1** to **P2-2**, and **P3-1** to **P3-4**.

First, an executable for APU is generated as follows:

P2-1 At (mdcom1), Communication I/F code for APU (d12) and for RPU (d13) are generated from (d8) shown in **Figure 7**.

P2-2 At (apu1), an executable for APU is built from (d7) and (d12).

Second, an executable for RPU is generated as follows. Note that, regarding the communication generation (mdcom1) for RPU is similar to that for APU described above. Therefore, we describe SystemBuilder part for RPU and FPGA as follows:

P3-1 At (sb1), (d14) to (d17) are generated from SDF file (d10) shown in **Figure 8**.

P3-2 For RPU, an executable for RPU is built from (d9), (d13), (d14) and (d16) at (rpu1).

P3-3 For FPGA, HDL (Hardware Description Language) files (d18) are synthesized from C code (d11) and (d15) by an HLS tool. Here, CyberWorkBench (CWB) [11] and Vivado HLS [15] are available for HLS tool. In this study, we use CWB.

P3-4 The HW design project file (d19) for SystemBuilder is prepared in advance, and a bitstream file is synthesized from HDL files ((d16) to (d19)). In this paper, we use only Vivado [14] to synthesize for ZynqMP.

As a result, developers can generate executables for FP-HSoC without coding. Once the executables are written to an SD card, it is easy to execute them on ZynqMP.

Furthermore, HS-MBP provides a simulation environment. For APU and RPU, QEMU is available for HS-MBP. For RPU-FPGA co-simulation, it is realized by HDL simulator and QEMU.

HS-MBP generates executables for each PE from s-code gen-

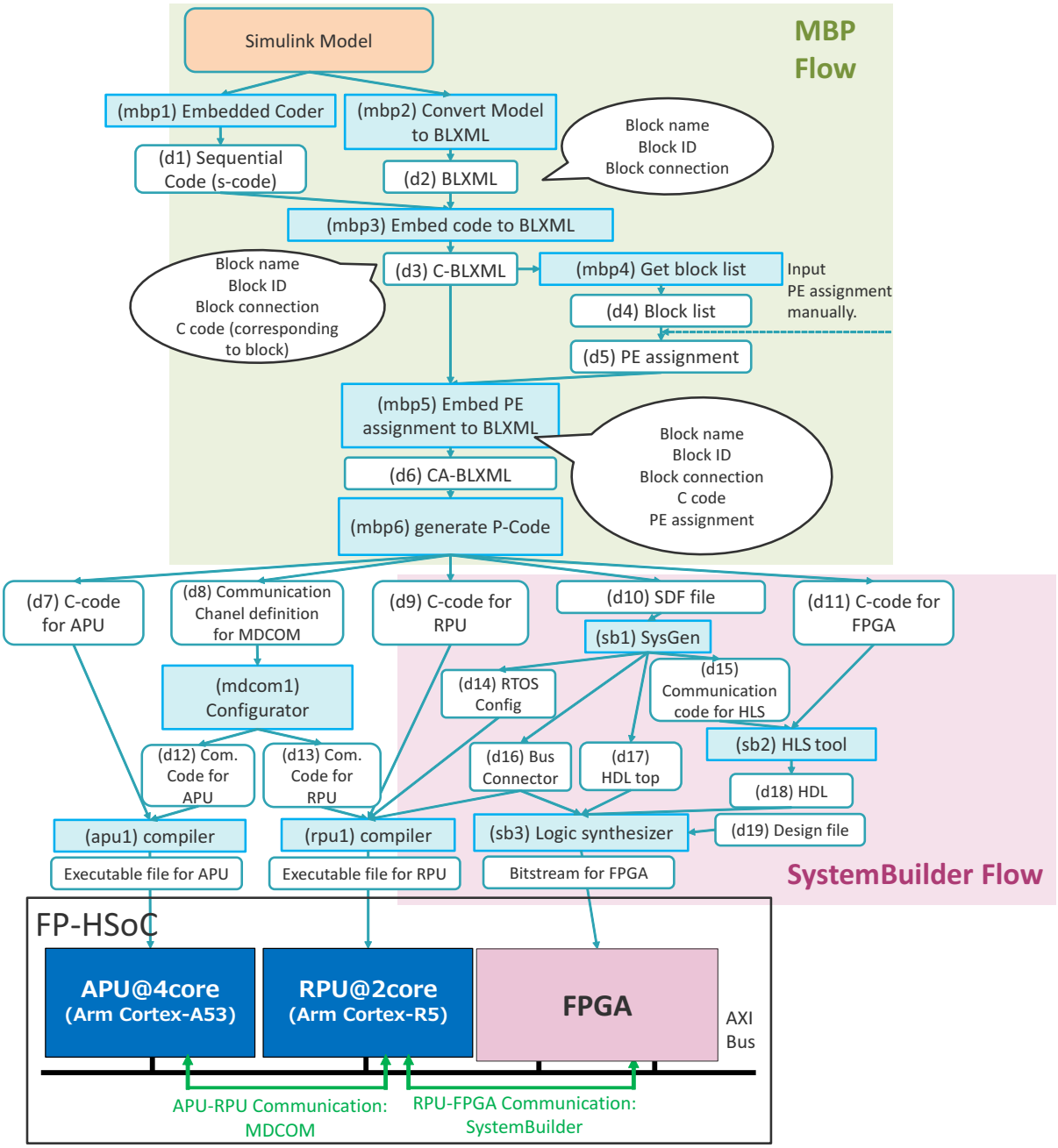


Fig. 2: Developent flow with HS-MBP

```
<block blocktype="Sum" name="model_Sum" rate="0.01">
  <input line="model_FromWorkspace_1" port="model_Sum_1">
    <connect block="model_FromWorkspace" port="model_FromWorkspace_1"/>
  </input>
  <input line="model_Plant_1" port="model_Sum_2">
    <connect block="model_Plant" port="model_Plant_Out1"/>
  </input>
  <output dimensions="[1 1]" line="model_Sum_1" port="model_Sum_1" usename="true">
    <connect block="model_pid" port="model_pid_error"/>
  </output>
</block>
```

Fig. 3: An Example of BLXML

erated by Embedded Coder. Our tool assigns functions in s-code to each PE because of code granularity for acceleration. The implementation on each PE described as follows:

- APU applications are implemented as threads on PetaLinux [12].
- RPU applications are implemented as tasks on TOPPERS/ATK2 kernel [9].
- HW modules on FPGA are designed as processes generated by SystemBuilder [1], [3].

3.2 Communication mechanism

In this section, we describe the communication mechanism

```
<block blocktype="Sum" id="14" name="model_Sum" rate="0.01">
  <input line="model_FromWorkspace_1" port="model_Sum_1">
    <connect block="model_FromWorkspace" port="model_FromWorkspace_1"/>
  </input>
  <input line="model_Plant_1" port="model_Sum_2">
    <connect block="model_Plant" port="model_Plant_Out1"/>
  </input>
  <output dimensions="[1 1]" line="model_Sum_1" port="model_Sum_1" usename="true">
    <connect block="model_pid" port="model_pid_error"/>
  </output>
  <var line="model_Sum_1" mode="output" name="Sum" port="model_Sum_1" storage="model_B" type="real_T"/>
  <var line="model_FromWorkspace_1" mode="input" name="FromWorkspace" port="model_Sum_1" storage="model_B" type="real_T"/>
  <var line="model_Plant_1" mode="input" name="DiscreteTransferFcn" port="model_Sum_2" storage="model_B" type="real_T"/>
  <signal name="Sum" storage="model_B" type="real_T"/>
  <code file="model.c" line="161" type="task">  /* Sum: '&lt;Root&gt;/Sum' */
 model_B.Sum = model_B.FromWorkspace - model_B.DiscreteTransferFcn;
</code>
  <code file="model.c" line="296" type="init">  {
  model_B.Sum = 0.0;
 }
</code>
...
 </block>
```

Code Informartion

Fig. 4: An Example of C-BLXML

```
#<block-name>,<block-id>,<peinfo>[,<bind block>,<bind block-id>]
model_DataTypeConversion,1,RPU0
model_ForIter,2,HW0
model_FromWorkspace,8,RPU0
model_Plant_DiscreteTransferFcn,11,RPU0
model_Plant_Integrator,12,RPU0
model_Sum,14,RPU0
...
```

PE Assignment

Fig. 5: An Example of Block list with PE assignment

among different PEs. In this study, we consider communication patterns APU-RPU and RPU-FPGA. Although there is another communication pattern, between APU and FPGA, we do not directly support this pattern because Linux OS on APU is not suitable for real-time control of hardware on FPGA. In order to implement APU-FPGA communication, therefore, it is necessary to combine two communications: APU-RPU and RPU-FPGA.

In our method, we propose a multirate implementation method for these two communication patterns to support multirate models. In a multirate model, there are a plural of control periods, and each block is executed with one of them. Therefore, there are communication channels between two blocks whose control periods are different. We call the block with slower/faster period S-BLOCK/F-BLOCK. **Figure 9** depicts examples of multirate communication. Here, assume T_s as sender thread/task/process period, and T_r as receiver thread/task/process period. In the case that $T_s \leq T_r$, blocking channel (BC) is used for the communication. Otherwise, a non-blocking channel (NBC) is used. This method was proposed by Nakano et al.[6].

If the communication is implemented by BC when $T_s > T_r$, and the buffer depth is insufficient, transmission waits may occur. As a result of this implementation, S-BLOCKs may wait for the completion of the reception of F-BLOCKs. On the other hand, if the communication is implemented by NBC, S-BLOCKs do not have to wait for F-BLOCKs. Furthermore, in our method, the communication utilizes double buffers to guarantee robustness due to jitter, i.e., F-BLOCKs absolutely receives data one period before.

In the case that $T_s \leq T_r$, the communication is implemented by BC. For example, assuming $T_s = 3T_r$, F-BLOCKs send data per three executions to S-BLOCKs. It is possible to receive the data one period before, which reduces the overhead required for transmission.

Next, the communication methods for each PE pair are described below.

3.2.1 Communication between APU and RPU

APU (Linux) - RPU (AUTOSAR CP) communication is realized by MDCOM: communication library for heterogeneous multicore developed by Otake et al.[7] and TOPPERS Project Inc[10]. MDCOM provides the following two communication mechanisms:

- SMEM Channel: store/load messages to/from shared memory accessible from all domains. Only a single data area is provided.
- FIFO Channel: store/load messages to/from FIFO on shared memory.

For multirate implementation, SMEM Channel is used in the case that $T_s > T_r$, and FIFO Channel is used in the case that $T_s \leq T_r$.

3.2.2 Communication between RPU and FPGA

SystemBuilder provides RPU (AUTOSAR CP) - FPGA communication. Note that, SW-SW communication means only inter-core communication within RPU. Therefore, this communication can also be realized by MDCOM (refer to 3.2.1). Communication I/Fs are defined in SDF (System DeFinition) file and its APIs are generated by SysGen (SystemBuilder Generator) from the SDF file. The API can be called in the C code for HW and SW. SystemBuilder provides the following two communication mechanisms:

- Blocking Channel: Corresponds to FIFO. If the FIFO is empty, the read-side process waits for data, and if the FIFO is full, the write-side process waits until the data full is cleared.
- Non-Blocking Channel: Corresponds to register. Data are read/written without waiting.

```
<block blocktype="Sum" id="14" name="model_Sum" peinfo="RPU0" rate="0.01">
  <input line="model_FromWorkspace_1" port="model_Sum_1">
    <connect block="model_FromWorkspace" port="model_FromW
  </input>
  <input line="model_Plant_1" port="model_Sum_2">
    <connect block="model_Plant" port="model_Plant_Out1"/>
  </input>
  <output dimensions="[1 1]" line="model_Sum_1" port="model_Sum_1" usename="true">
    <connect block="model_pid" port="model_pid_error"/>
  </output>
  <var line="model_Sum_1" mode="output" name="Sum" port="model_Sum_1" storage="model_B" type="real_T"/>
  <var line="model_FromWorkspace_1" mode="input" name="FromWorkspace" port="model_Sum_1" storage="model_B" type="real_T"/>
  <var line="model_Plant_1" mode="input" name="DiscreteTransferFcn" port="model_Sum_2" storage="model_B" type="real_T"/>
  <signal name="Sum" storage="model_B" type="real_T"/>
  <code file="model.c" line="161" type="task">  /* Sum: '&lt;Root&gt;/Sum' */
 model_B.Sum = model_B.FromWorkspace - model_B.DiscreteTransferFcn;
</code>
  <code file="model.c" line="296" type="init"> {
  model_B.Sum = 0.0;
 }
</code>
...
 </block>
```

PE Assignment

Fig. 6: An Example of CA-BLXML

```
CRE_FIFOCH(multirate2_MDCOM_FIFOCH_0, 1, 8,
    NULL_FILTER, NULL_FILTER)
```

Fig. 7: An Example of MDCOM Configuration

```
SysName: multirate2

HW:
 - {core: FPGA, process: [multirate2_HWCore0_10_0_Task,
multirate2_HWCore0_30_0_Task]}
SW:
 - {core: Nios, id: 1, process: [multirate2_RPUCore0_10_0_Task]}
...

CyclicSync:
 init_start: true
 sw_base_nsec: 10000000
 hw_base_nsec: 1000

BlockingChannel:
 - {name: bc_0007_0029, size: 32, float}
...

NonBlockingChannel:
 - {name: nbc_0028_0033_0, size: 32, float}
...

StandardProcess:
 - name: multirate2_HWCore0_10_0_Task
   file: [multirate2_HWCore0_10_0_Task.c]
   ch(in): [bc_0007_0030]
   ch(out): [bc_0030_0024]
   cycle_nsec: 10000000
   start_offset: 0
...
```

Fig. 8: An Example of SDF file

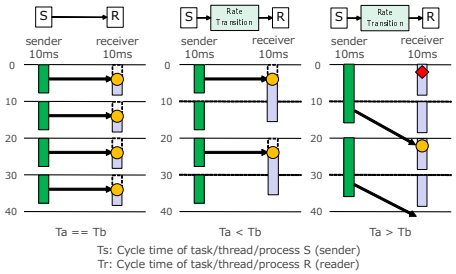


Fig. 9: Examples of Multirate Model

For multirate implementation, NBC is used in the case that $T_s > T_r$, and BC is used in the case that $T_s \leq T_r$.

4. Case Study

In this section, we present a case study for two multi-rate models. The first model is a multi-rate PID control model, and in the second model a For Iterator Subsystem block is added to the first model.

By using each multi-rate model, we investigate the following:

- The blocks operate at correct timing in each PE and communicate correctly between PEs.
- Development flow for tuning of code to FPGA

4.1 Target Models

The first Simulink model is shown in **Figure 10**. This model is a multirate model consisting of PID control and a plant, and the P controller shown in **Figure 10c** is executed with a 30-ms period, the I controller shown in **Figure 10d** is executed with a 20-ms period, and the D controller and the other blocks shown in **Figure 10e** and **Figure 10f** is executed with a 10-ms period.

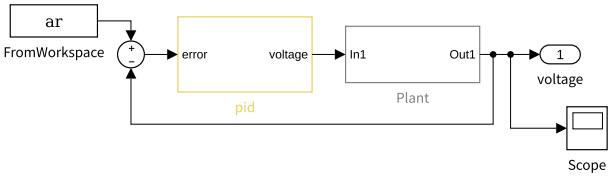
Figure 10b, Figure 10c, Figure 10d, Figure 10e and Figure 10f are SubSystems and they are specified as Atomic except for Figure 10f. An Atomic SubSystem is not flattened and is treated by its parent model as a single block. A non-Atomic SubSystem is flattened into its parent model. The input values given to the model in **Figure 10a** are periodically given as 0.0 or 2.0 of double type, and the output values are also obtained of double type.

The second model is shown in **Figure 11**. In this model, at the top of the model, we added SubSystem x2 to cast the input of For Iterate Subsystem to int type. The contents of each subsystem are shown in **Figure 11a**, **Figure 11b**, and **Figure 11c**.

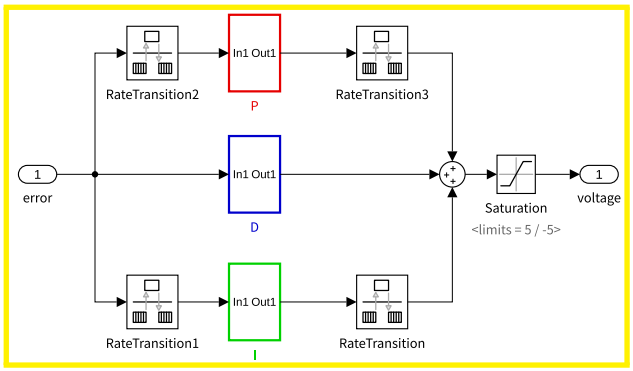
4.2 Procedure and results

We applied the procedure stated in 3.1, to the model in Figure 10 as input. Except for the creation of (d4) to (d5) in Figure 2, the steps explained in 3.1 can be executed on the GUI or simply by typing a command.

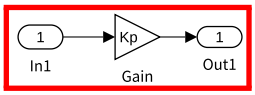
At (d4), a CSV file is created. The CSV file has information: block name, block ID, and PE assignment. The information except for the PE information is generated automatically by the tool,



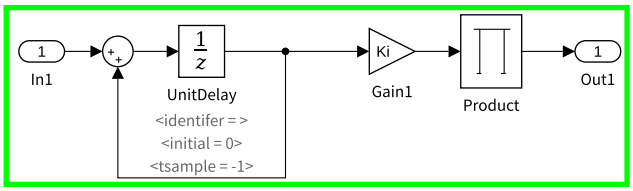
(a) Model top(1)



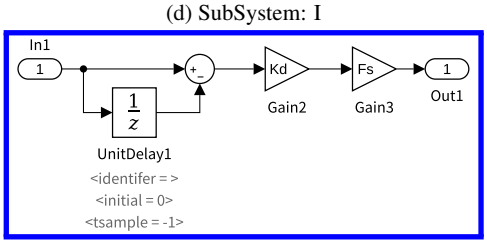
(b) SubSystem: pid



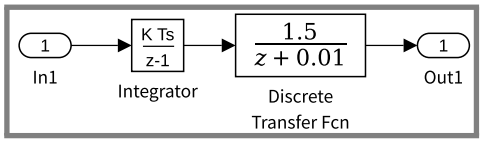
(c) SubSystem: P



(d) SubSystem: I



(e) SubSystem: D

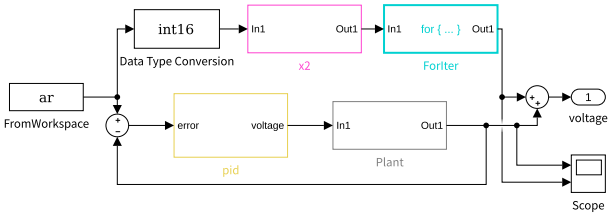


(f) SubSystem: Plant

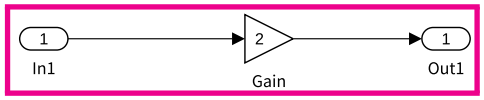
Fig. 10: Simulink models for case study (1)

and developers add the PE assignment information. A tool is available to add the information from the CSV file to BLXML so developers only add the PE assignment information to this CSV file. Note that, in this paper, we specify thread number for APU instead of core number.

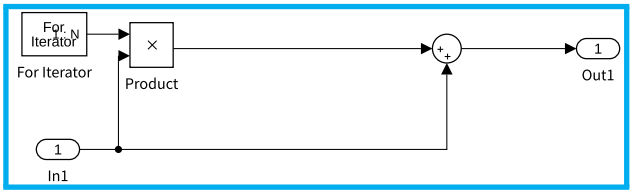
The purpose of this model in the case study is to investigate whether the input model can be implemented in various PE assignment patterns. For this investigation, we design 9 experimental patterns shown in **Table 1**, generate codes for each pattern, execute them in FP-HSoC, and check the output results. The experimental patterns are designed to validate communication and execution of the multirate implementation on FP-HSoC.



(a) Model top(2)



(b) SubSystem: x2



(c) SubSystem: ForIter

Fig. 11: Simulink models for case study (2)

Table 1: Execution Patterns

ID	Control Period		
	10ms(D)	20ms(I)	30ms (P)
C1	RPU0	RPU0	RPU0
C2	RPU0	RPU1	RPU1
C3	RPU0	APU0	APU0
C4	RPU0	APU0	APU1
C5	APU0	RPU0	RPU0
C6	RPU0	HW0	HW0
C7	RPU0	HW0	HW1
C8	HW0	RPU0	RPU0
C9	RPU0	APU0	HW0

Note that, since it is difficult for HW to output the execution results to the standard output, the RPU is used to check the output values. Therefore, even if we choose not to assign output blocks (of 10ms period) to the RPU, only the blocks that output the result are assigned to the RPU.

C1 RPU @ 1-Core

C2 RPU @ 2-Cores

C3 RPU @ 1-Core: short-cycle blocks, APU @ 1-Thread: mid-cycle and long-cycle blocks

C4 RPU @ 1-Core: short-cycle blocks, APU @ 2-Threads: mid-cycle and long-cycle blocks

C5 RPU @ 1-Core: mid-cycle and long-cycle blocks, APU @ 1-Thread: short-cycle blocks

C6 RPU @ 1-Core: short-cycle blocks, HW (FPGA) @ 1-Process: mid-cycle and long-cycle blocks

C7 RPU @ 1-Core: mid-cycle and long-cycle blocks, HW (FPGA) @ 2-Processes: short-cycle blocks

C8 RPU @ 1-Core: mid-cycle and long-cycle blocks, HW (FPGA) @ 1-Process: short-cycle blocks

C9 RPU @ 1-Core: short-cycle blocks, APU @ 1-Thread: mid-cycle blocks, HW (FPGA) @ 1-Process: long-cycle blocks

We executed all the patterns from **C1** to **C9** and confirmed that the simulation results of the Simulink model and the execution results of those patterns were identical.

Next, we describe the case study with the second model. For

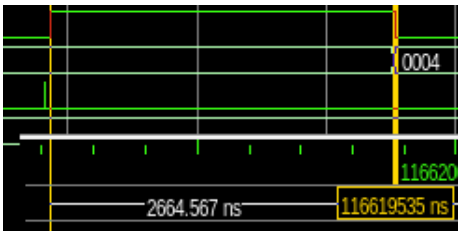


Fig. 12: HW latency without folding: 2664.567ns

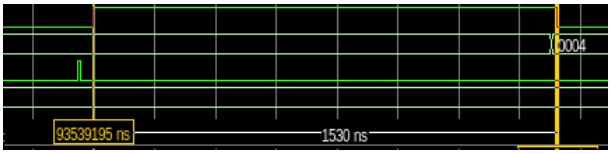


Fig. 13: HW latency with folding: 1530ns

this case study, only the For Iterator Subsystem was assigned to the FPGA and all other blocks were assigned to the RPU. As a result, we confirmed that the generated p-code works correctly without fixing. Although the For Iterator block has a simple data dependency that does not violate the multirate deadline, it can be accelerated.

SystemBuilder provides co-simulation environment for developers to test results and speedups on hardware. Using Questa+QEMU as a co-simulation environment, we confirmed the results before and after to apply loop folding.

Figure 12 and **Figure 13** are waves from HW execution. Yellow cursors in Figure 12 and Figure 13 show the latency of HW execution of a particular execution period. As a result, HW with folding (1530 ns) is faster than HW without folding (2665 ns).

5. Discussion

5.1 Implementation for each PE

First, the correspondence was validated between the Simulink model and implementation for each PE, as the execution results are equal to the simulation results. In our multi-rate model, the output results should be different if the timing of data acquisition is different. Based on these results, HS-MBP correctly generates executables for the multirate model.

The generation time of the executables was about 10 minutes at the most. Compilation time was dominant for APU and RPU, while HLS and logic synthesis were dominant for FPGA. The only manual tasks are PE allocation and command input; the time for PE allocation is less than one minute in this case study because the rules for core allocation were determined in advance. Based on the above, we conclude that the work time by HS-MBP could be reduced compared to design flow currently applied in actual design.

The present study uses a relatively small-scale model and manual PE allocation is easy. As the size of the model increases, the cost of manual PE allocation increases and the possibility of allocation errors increases. Therefore, it is desirable to automate PE allocation. MBP provides an automatic PE allocation tool, which enables automatic PE allocation even in a heterogeneous environment, based on the execution performance estimation. However, as explained in 5.2, we think that automation for FPGA is difficult because it requires tuning of the C code for FPGAs to estimate execution time, which is future work.

5.2 Co-simulation for FP-HSoC

First, Co-simulation can reduce work time. In the co-simulation environment, RTL simulator compilation is performed instead of logic synthesis, and RTL simulator compilation is shorter in time than logic synthesis.

Second, for work time, it is easy to search for acceleratable code snippets because the code is simple C code and the first author has been using the HLS tool for more than three years. Therefore, it took only about one minute to rewrite the code for the speed-up.

The HW latency is decreased from 2665ns to 1530ns as a result of the acceleration. The results of this acceleration were also obtained from the co-simulation. However, in order for developers to identify the acceleratable code, developers must be familiar with HLS. Therefore, it is necessary to support developers to accelerate HW.

The co-simulation environment in this paper has the problem of requiring expensive license fees to prepare the co-simulation environment. Currently, the emulator QEMU and the RTL simulator Questa are being used. Therefore, we are now preparing for Vivado simulator for co-simulation, which is available free of charge. We also consider that more efficient simulation environment for automatic PE allocation in the future.

6. Related work

Xilinx is developing Vitis [13] (SDSoC until version 2019.1), a SW/HW co-development environment for SoC, which allows developers to set up processing systems (PS), memory mapping, and signals before co-designing. Therefore, the design can be flexible in terms of HW configuration and use of peripheral functions. There are also features such as HW-level simulation, SW-level simulation, and run-time profiles of actual machines. For example, profiling results, such as memory transfer latency, and timing charts for execution time can be obtained for each device.

SystemBuilder, the system-level design environment used in the HS-MBP, also uses Vivado and Vivado HLS, so the C code is similar. However, there is a difference in terms of HW design. SystemBuilder basically uses a pre-designed HW design file to promote co-design, so the HW structure is hidden to developers. In other words, knowledge of HW design with Vivado is unnecessary to enable SW/HW co-design for SystemBuilder users. However, when changing the configuration of the HW, such as when using peripheral functions, knowledge of the HW configuration is required.

In terms of simulation functions, SystemBuilder has a co-simulation environment and a simulation using QEMU + Questa (RTL Simulator). Additionally, SystemBuilder also has a profiling function, which is similar to Vivado.

7. Conclusion

In this paper, we proposed HS-MBP: model-based parallelization environment for FP-HSoC. The environment enables easy implementation for each PE, RPUs, APUs and FPGAs of FP-

HSoC. The results of the case study confirmed this. We also conducted a case study to support tuning for FPGAs and confirmed the procedure for tuning FPGA with co-simulation.

One of the challenges for the future is automatic PE allocation; we will discuss how to organize code snippets for FPGAs and the possibility of automatic tuning to generate more efficient HWs automatically. It is also necessary to make block for C code directly with the S-Function block available so that the source code for HLS, which is pre-tuned for FPGAs, can be used.

Acknowledgments This paper is based on results obtained from a project, JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] Ando, Y., Honda, S., Takada, H. and Edahiro, M.: System-level design method for control systems with hardware-implemented interrupt handler, *Journal of Information Processing*, Vol. 23, No. 5, pp. 532–541 (2015).

[2] Bergmann, A.: Benefits and Drawbacks of Model-based Design, *Applied Science and Engineering Progress*, Vol. 7, No. 3, pp. 15–19 (2014).

[3] Honda, S., Tomiyama, H. and Takada, H.: SystemBuilder: A system level design environment, *IEICE Trans. Information & Systems*, Vol. 88, No. 2, pp. 163–174 (2005).

[4] MathWorks: MATLAB, `https://jp.mathworks.com/products/simulink.html`. Accessed: 2020-9-26.

[5] MathWorks: Simulink, `https://jp.mathworks.com/products/simulink.html`. Accessed: 2020-9-26.

[6] Nakano, Y., Honda, S., Edahiro, M. and Suzuki, H.: Runtime and code generation for Automotive RTOS of multirate modelin model base parallelization, *IPSJ SIG Technical Report*, Vol. 2017-SLDM-179, No. 4, pp. 1–6 (2017).

[7] Ohtake, F., Honda, S. and Takda, H.: MDCOM, communication library for heterogeneous processor, Vol. 2017-EMB-44, No. 34, pp. 1–6 (2017).

[8] Schmidt, A. G., Weisz, G., French, M., Flatley, T. and Villalpando, C. Y.: SpaceCubeX: A framework for evaluating hybrid multi-core CPU/FPGA/DSP architectures, *2017 IEEE Aerospace Conference*, IEEE, pp. 1–10 (2017).

[9] TOPPERS Project Inc.: MDCOM, `https://www.toppers.jp/atk2.html`. Accessed: 2020-9-26.

[10] TOPPERS Project Inc.: MDCOM, `https://www.toppers.jp/mdcom.html`. Accessed: 2020-9-26.

[11] Wakabayashi, K.: CyberWorkBench: integrated design environment based on C-based behavior synthesis and verification, *In VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT)*, IEEE, pp. 173–176 (2005).

[12] Xilinx: PetaLinux Tools, `https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html`. Accessed: 2020-9-26.

[13] Xilinx: Vitis Unified Software Platform, `https://www.xilinx.com/products/design-tools/vitis.html`. Accessed: 2020-9-26.

[14] Xilinx: Vivado Design Suite, `https://www.xilinx.com/products/design-tools/vivado.html`. Accessed: 2020-9-26.

[15] Xilinx: Vivado High-Level Synthesis, `https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`. Accessed: 2020-9-26.

[16] Xilinx: Zynq UltraScale+ MPSoC, `https://japan.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html`. Accessed: 2020-9-26.

[17] Zhong, Z. and Edahiro, M.: Model-Based Parallelizer for Embedded Control Systems on Single-ISA Heterogeneous Multicore Processors, *International SoC Design Conference*, IEEE, pp. 117–118 (2018).

[18] Zhong, Z. and Edahiro, M.: Model-based Parallelization for Simulink Models on Multicore CPUs and GPUs, *International SoC Design Conference*, IEEE, pp. 103–104 (2019).