Design of an Asynchronous Processor with Bundled-data Implementation on a Commercial Field Programmable Gate Array

Jukiya Furushima, Masamitsu Nakajima and Hiroshi Saito
University of Aizu, Aizu-Wakamatsu 965-8580, Japan
E-mail: {m5201118, m5191117, hiroshis}@u-aizu.ac.jp

Keywords: asynchronous circuits, FPGAs, processors

Received: November 18, 2016

In this paper, we propose a modeling method and a design flow to design asynchronous processors with bundled-data implementation on commercial Field Programmable Gate Arrays (FPGAs). The modeling method mainly concerns modeling of an asynchronous control circuit on commercial FPGAs. In addition to the use of a design environment provided by FPGA vendor, the design flow includes constraint generation, timing analysis, and delay adjustment to design asynchronous processor from a prepared model to FPGA programming. In the experiments, we design three asynchronous MIPS processors. Comparing with the synchronous counterpart, one of them reduces global cycle time which results in 13.8% performance improvement and another one reduces energy consumption 9.3% for a multiplication and 8.8% for a matrix multiplication.

Povzetek: Opisan je razvoj novega sinhronega procesorja na osnovi tržnega FPGA.

1 Introduction

Field Programmable Gate Arrays (FPGAs) are reconfigurable circuits where circuit structure can be changed by designers freely. Therefore, compared to Application Specific Integrated Circuits (ASICs), the lifetime of FPGAs is long. In addition, the design cost is low because FPGA vendors provide the design environment free of charge. Recently, due to the advance of the FPGA technology, FPGAs are well adopted in embedded systems and servers for data centers [1]. As there are a rich number of resources on FPGAs, we can accelerate performance by implementing Multi-Processor System-on-Chip (MPSoC). Current advanced FPGAs such as Altera Cyclone V include an ARM Cortex processor as a hard-macro to support MPSoC.

Most of commercial FPGAs are synchronous circuits. Circuit components in synchronous circuits are controlled by global clock signals. In synchronous circuits, clock skew, power consumption, and electromagnetic radiation will be significant problems when the semiconductor submicron technology is advanced more and more. In addition, generally, the power efficiency of FPGAs is worse than ASICs. Therefore, low power designs on FPGAs are very important.

Compared to synchronous circuits, circuit components in asynchronous circuits are controlled by local handshake signals. Due to the absence of global clock signals, asynchronous circuits are potentially low power consumption and low electromagnetic radiation. Therefore, asynchronous circuits may be useful for FPGAs where low power design is important. However, the design of asynchronous circuits is more difficult than the design of synchronous circuits. To represent circuit behaviors, circuit model including delay model, data encoding scheme, and handshake protocol should be considered. Based on the considered model, asynchronous circuits are designed. In addition, asynchronous circuit designs are also difficult for commercial FPGAs because the design environment provided by FPGA vendors is assuming synchronous circuit designs.

There are many approaches to design asynchronous circuits on commercial FPGAs [2, 3, 4, 5, 6]. Tranchero proposed a design method to design asynchronous circuits on commercial FPGAs in [2]. Ho et al. described to implement C-element [7] into a logic block on commercial FPGAs, showed that there is no hazards, and designed a 4-bit adder with the C-element in [3]. We proposed a floorplan method to place asynchronous logics to commercial FPGAs. All of these literatures address neither design constraint generation (e.g., the maximum delay constraints for paths) nor timing verification (i.e., whether correct timing to control resources is guaranteed or not). We also proposed a design method for asynchronous circuits with bundled-data implementation like this paper in [5]. However, it does not target asynchronous processors. As modeling, constraint generation, and timing verification of processor designs are different, we need a design method to implement asynchronous processors on commercial FPGAs. Minas, et. al., proposed an asynchronous processor with the concurrent error detection scheme to detect transient errors in [6]. It was implemented on an commercial FPGA. On the other hand, modeling, constraint generation, and timing verification described in this paper are not addressed in [6].

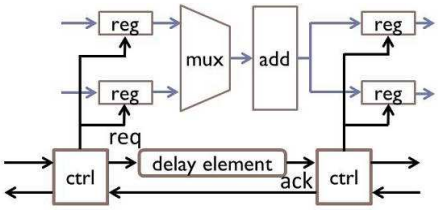

Figure 1: Circuit structure of asynchronous circuits with bundled-data implementation.

In this paper, we propose a modeling method and a design flow to design asynchronous processors with bundled-data implementation on FPGAs. We address how to implement an asynchronous control circuit on the commercial FPGAs, how to synthesize the asynchronous processor with the generation of design constraints, and how to carry out timing verification correctly. We design three pipelined MIPS processors using the proposed method and design flow to evaluate area, execution time, dynamic power, and energy consumption.

The rest of this paper is organized as follows. In section 2, we describe asynchronous circuits with bundled-data implementation. In section 3, we describe about FPGAs. In section 4, we describe the proposed modeling method and design flow. In section 5, we describe the experimental results by designing three MIPS processors. Finally, in section 6, we conclude this work.

2 Asynchronous circuits with bundled-data implementation

Asynchronous circuits with bundled-data implementation shown in Fig.1 are one of data encoding schemes in asynchronous circuits. Timing of data operations is guaranteed by delay elements on request signals. Therefore, the performance depends on the delay of the control circuit. Compared to other implementations such as dual-rail implementations [7] where one bit signal is represented by two wires and the completion detector is required, bundled-data implementation can be realized easily because we can use the same data-path resources as synchronous circuits. In addition, the circuit area and the power consumption become smaller and lower than other implementations.

2.1 Circuit model

Figure 2 represents a bundled-data implementation model used in this paper. It is a pipelined processor model with several pipeline stages i. The left side is the control circuit and the right side is the data-path circuit. The data-path circuit consists of Program Counter (PC), Memories (IMEM and DMEM), pipeline registers (pipereg), Decoder, Register File (RF), ALU, and delay elements $wd_{i,k}$ and hd_k. PC stores the address of the instruction memory. IMEM

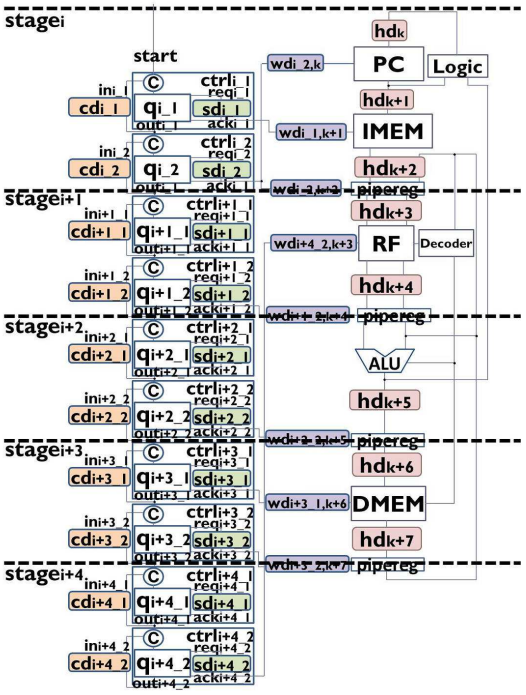

Figure 2: An asynchronous processor model with bundled-data implementation.

is a memory to store instructions. DMEM is a memory to store data. $piperegs$ are registers to separate pipeline stages. RF is a collection of registers. Data from DMEM and ALU are written into RF. $wd_{i,k}$ and hd_k are delay elements for registers or memories to guarantee simultaneous writing constraints and hold constraints.

The control circuit consists of control modules $ctrl_{i\_j}$ ($j = 1, 2$). A control module $ctrl_{i\_j}$ consists of a Q-module q_{i_j} [8], delay elements sd_{i_j} and cd_{i_j}, and a C-element c_{i_j} [7]. sd_{i_j} is used to guarantee setup constraints. cd_{i_j} is used to guarantee control initialization constraints. The C-element c_{i_j} is a synchronization component. The output of the C-element is 0 when all inputs are 0. The output is 1 when all inputs are 1. Otherwise, the output does not change. Logical 1 for the output of the C-element means that the execution at the previous control module and the initialization of the current control module finish.

There are two notes in the control circuit. First, compared to ordinal asynchronous pipelined circuits such as Micropipelines [9] where a feedback signal for the C-element is generated from the output of the C-element in the next control module, the feedback signal in this control circuit is generated from out_{i_j}. This is because to keep the same execution time in all pipeline stages. Second, we use two control modules $ctrl_{i\_1}$ and $ctrl_{i\_2}$ to control a pipeline stage i to hide the overhead caused by handshake signals.

Control modules $ctrl_{i\_j}$ operate as follows. When the execution at the previous control module and the initialization of the current control module finish, c_{i_j} asserts in_{i_j} to trigger the Q-module q_{i_j}. The Q-module q_{i_j} asserts

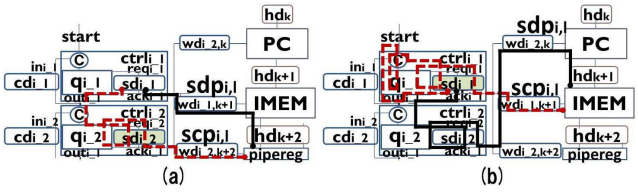

Figure 3: Data-path $sdp_{i,l}$ and control path $scp_{i,l}$ for setup constraints: (a) forward path and (b) backward path.

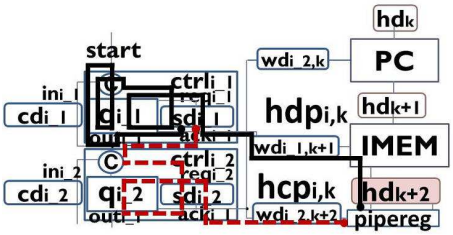

Figure 4: Data-path $hdp_{i,k}$ and control path $hcp_{i,k}$ for a hold constraint.

req_{i_j}. After the signal passes to sd_{i_j}, it is returned to q_{i_j} with the assertion of ack_{i_j}. Then, the Q-module q_{i_j} deasserts req_{i_j}. After the signal passes to sd_{i_j} again, it is returned to q_{i_j} with the deassertion of ack_{i_j}. The deassertion of ack_{i_j} asserts out_{i_j} to move the control to the next control module. Memories and registers in the data-path circuit are controlled by the output of sd_{i_j}. The initialization of control modules $ctrl_{i\_j}$ starts immediately after out_{i_j} is asserted. It is tuned to the deassertion of in_{i_j} and out_{i_j}. The next operation starts immediately after the execution at the previous control module finishes.

2.2 Timing constraints and cycle time

The bundled-data implementation model used in this paper must satisfy five types of timing constraints, setup constraints, hold constraints, control initialization constraints, and simultaneous writing constraints.

Setup constraints mean that input data of registers must be stable before writing to registers. Figure 3 represents paths related to setup constraints. $sdp_{i,l}$ (solid line) represents a data-path from sd_{i_1} to the destination register $pipereg$ where data is written through the source memory IMEM. $scp_{i,l}$ (dotted line) represents a control path from sd_{i_1} to the destination register $pipereg$ through the control module $ctrl_{i\_2}$. $t_{minscp_{i,l}}$, $t_{maxsdp_{i,l}}$, t_{setup_k}, and $sm_{i,l}$ represent the minimum delay of $scp_{i,l}$, the maximum delay of $sdp_{i,l}$, the setup time for the destination register $pipereg$, and the margin for $t_{maxsdp_{i,l}}$. The setup constraint can be represented by the following equation:

$$t_{minscp_{i,l}} > t_{maxsdp_{i,l}} + t_{setup_k} + sm_{i,l} \qquad (1)$$

If this constraint is violated, we need to adjust the delay element sd_{i_1} or sd_{i_2}.

There are two types of $sdp_{i,l}$. One is a forward path where the source register is controlled by a previous control module as shown in Fig.3(a) and the other is a backward path where the source register is controlled by a next control module as shown in Fig.3(b). We define local cycle time lct_i and global cycle time gct. The local cycle time lct_i is defined for each pipeline stage i in which is equal to the maximum delay of $scp_{i,l}$, $t_{maxscp_{i,l}}$, in pipeline stage i. The global cycle time gct is the maximum lct_i for all lct_i. The global cycle time with input data interval decides the throughput of asynchronous pipelined processors.

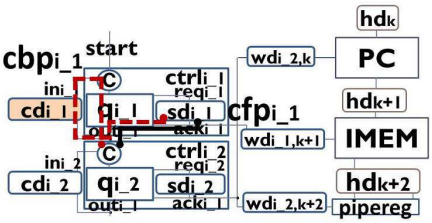

Figure 5: Forward path cfp_{i_1} and backward path cbp_{i_1} for a control initialization constraint.

Hold constraints mean that input data of registers must be stable during writing to registers. Figure 4 represents paths related to hold constraints. $hcp_{i,k}$ (dotted line) represents a control path from sd_{i_1} to the destination register $pipereg$ where data is written. $hdp_{i,k}$ (solid line) represents a data-path from sd_{i_1} to the destination register $pipereg$ through data-path resources. $t_{minhdp_{i,k}}$, $t_{maxhcp_{i,k}}$, t_{hold_k}, and $hm_{i,k}$ represent the minimum delay of $hdp_{i,k}$, the maximum delay of $hcp_{i,k}$, the hold time for the destination register $pipereg$, and the margin for $t_{maxhcp_{i,k}}$. The hold constraint can be represented by the following equation:

$$t_{minhdp_{i,k}} > t_{maxhcp_{i,k}} + t_{hold_k} + hm_{i,k} \qquad (2)$$

If this constraint is violated, we need to adjust the delay element hd_k.

Control initialization constraints mean that the initialization of control modules must be completed after the control signal by the assertion of out_{i_j} reaches to the next control module. Otherwise, the assertion is disabled. Figure 5 represents paths related to a control initialization constraint. cfp_{i_1} (solid line) represents a control path from sd_{i_1} to c_{i_2}. cbp_{i_1} (dotted line) represents a control path from sd_{i_1} to c_{i_2} through q_{i_1}. $t_{maxcfp_{i\_1}}$ represents the maximum delay of cfp_{i_1} and $t_{mincbp_{i\_1}}$ represents the minimum delay of cbp_{i_1}. cm_{i_1} represents the margin for $t_{maxcfp_{i\_1}}$. The control initialization constraint can be represented by the following equation:

$$t_{mincbp_{i\_1}} > t_{maxcfp_{i\_1}} + cm_{i\_1} \qquad (3)$$

If this constraint is violated, we need to adjust the delay element cd_{i_1}.

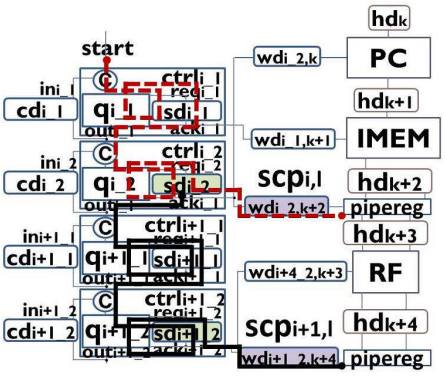

Figure 6: The maximum delays of two control paths $scp_{i,l}$ and $scp_{i+1,l}$ must be nearly equal to each other in simultaneous writing constraints.

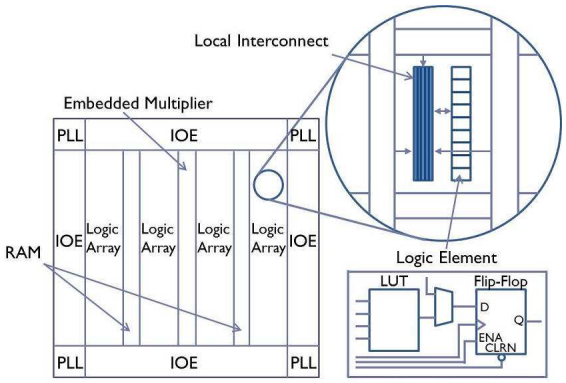

Figure 7: Structure of Altera Cyclone IV FPGA [10].

Simultaneous writing constraints mean that all of registers must be written to the same timing. In pipelined circuits, the throughput depends on the global cycle time gct. Therefore, to delay all of register writing timing until the global cycle time does not affect the throughput. In addition, as a difference of register writing timing may lead to setup/hold violations, to preserve these constraints reduces the occurrence of setup/hold violations. On the other hand, to satisfy simultaneous writing constraints results in behaviors like synchronous circuits. Different from synchronous circuits where global clock signals are used, registers are controlled by different control modules in this circuit model. Therefore, we expect low power consumption for designed asynchronous processors even though these constraints are preserved. Figure 6 represent two control paths $scp_{i,l}$ (dotted line) and $scp_{i+1,l}$ (solid line) for setup constraints. These constraints can be represented by the following equation:

$$t_{maxscp_{i,l}} \simeq t_{maxscp_{i+1,l}} \simeq gct \tag{4}$$

If the above relationship is violated, we adjust delay elements sd_{i_2} and $wd_{i\_2,k}$ for $t_{maxscp_{i,l}}$ and delay elements sd_{i+1_2}, and $wd_{i+1\_2,k}$ for $t_{maxscp_{i+1,l}}$.

3 Field programmable gate array

Field Programmable Gate Array (FPGA) is one of reconfigurable devices. FPGA has been used in many embedded systems because of the advantage such as lower design cost and flexibility to change circuit structure. Figure 7 shows the structure of Altera Cyclone IV FPGA.

The FPGA consists of Logic Array Blocks (LABs), Embedded Multipliers, Random Access Memories (RAMs), Input/output Elements (IOEs), and Phase Locked Loops (PLLs). A logic array consists of 16 logic elements (LEs). A logic element consists of a D Flip-Flop (DFF) and a 4-to-1 Look Up Table (LUT). Any logic function with four inputs can be implemented on an LUT based on a Static RAM. Most of commercial FPGAs has the similar structure like this FPGA.

We use two primitives in Altera FPGAs. One is LCELL and the other is DLATCH. LCELLs are used to implement delay elements such as sd_{i_j} and DLATCHes are inserted after C-elements to carry out static timing analysis correctly with the initialization of C-elements. Both of them are mapped to LUTs.

4 Design of asynchronous processor on commercial FPGAs

In this section, we describe the proposed modeling method and design flow. Even though we target Altera FPGAs in this paper, as there is a similar design environment, we think that we can design asynchronous processors on other FPGAs such as Xilinx FPGAs with the modification of the proposed modeling method and design flow.

4.1 Modeling method

As shown in Fig.2, bundled-data implementation used in this paper consists of a control circuit and a data-path circuit. We use the same data-path resources as the ones used in synchronous circuits. Therefore, we mainly describe modeling of the control circuit.

The proposed modeling method extends the method described in [11] where FPGAs are not considered. Initially, pipeline stages are modeled by a Finite State Machine (FSM) where nodes represent a pipeline stage and edges represent a control flow between pipeline stages. Figure 8(a) represents an FSM for a 5 stage pipelined processor. IF, ID, EX, MEM, and WB represent instruction fetch from the instruction memory, instruction decode, execution, data memory access, and write back to the register file.

Figure 8(b) represents a modeling flow. For each node in the FSM, we split it into two nodes and map control modules ($ctrl_{i\_1}$ and $ctrl_{i\_2}$) . Splitting of nodes is required to hide handshake overhead by two control modules. Then, delay elements (sd_{i_j}), C-elements (c_{i_j}), feedback loops

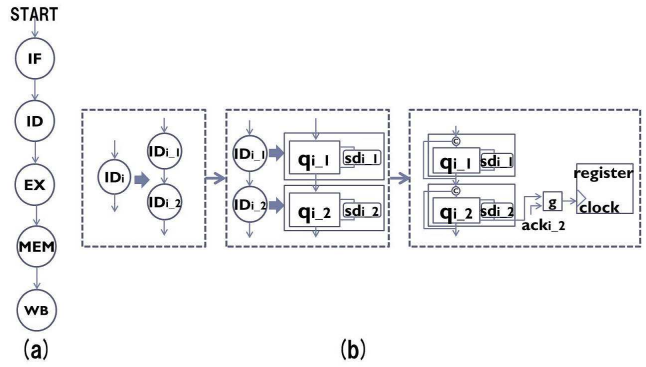

Figure 8: Modeling flow of control circuit: (a) FSM and (b) generation of a control circuit from FSM.

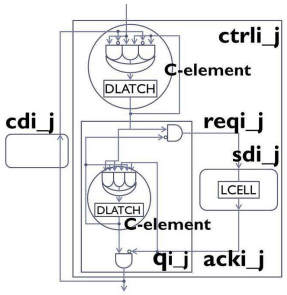

Figure 9: Internal structure of control modules.

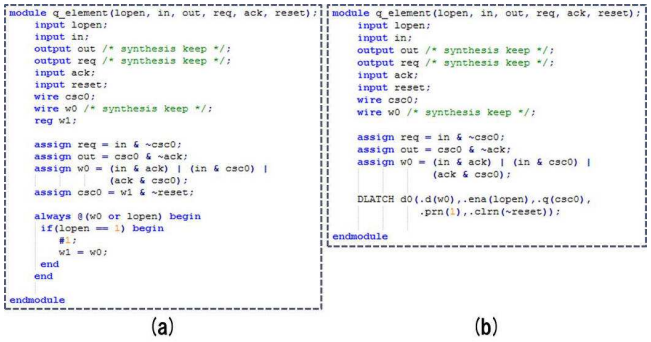

Figure 10: Verilog HDL models of an asynchronous processor: (a) simulation model and (b) synthesis model.

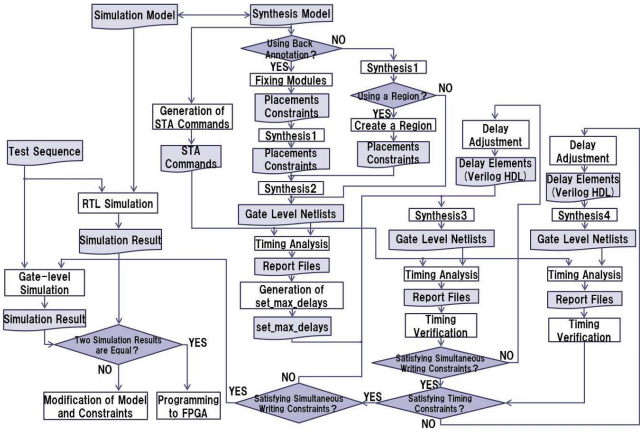

Figure 11: Proposed design flow.

from out_{i_j} to c_{i_j} are inserted. In the modeling, we just insert delay elements sd_{i_j} which consist of one LCELL. All other delay elements are inserted during delay adjustment after synthesis because other delay elements are required when timing constraints such as hold constraints are violated.

Resisters and memories are triggered by ack_{i_j} of corresponding control modules. We insert a glue logic to registers and memories to generate a local clock signal for them from ack_{i_j} and other conditional signals generated from the data-path circuit.

Figure 9 shows the structure of $ctrl_{i\_j}$ for Altera Cyclone IV FPGA. There are two DLATCHes in a $ctrl_{i\_j}$. One is after the C-element c_{i_j} and the other is after the C-element in the Q-module q_{i_j}. They are used to initialize the output of C-elements and to execute static timing analysis correctly. Delay elements sd_{i_j}, hd_k, cd_{i_j}, and $wd_{i\_j,k}$ consist of LCELLs. They work as buffers. To avoid renaming of the output signals of delay elements by the synthesis tool Altera Quartus Prime, we assign $synthesis\_keep$ commands to the output signals of delay elements. Note that we need to avoid logic optimization of control modules and all of delay elements by Quartus Prime. If they are optimized after synthesis by Quartus Prime, we need re-synthesis by assigning $design\_partition$ commands to control modules and delay elements.

Finally, we prepare two models of the bundled-data implementation by Verilog Hardware Description Language (HDL) which is a standard modeling language for FPGAs. The first model is for Register Transfer Level (RTL) simulation before synthesis and the latter model is for synthesis. As RTL simulation does not allow to involve primitive cells DLATCHes and LCELLs, we represent them using logic expressions. Figure 10 (a) and (b) represent the simulation model and the synthesis model for q_{i_j} in control module $ctrl_{i\_j}$ using Verilog HDL.

4.2 Design flow

The proposed design flow uses the design environment Altera Quartus Prime. To design asynchronous processors with bundled-data implementation on commercial FPGAs, we need to consider timing analysis, constraint generation, and delay adjustment for asynchronous processors which are not supported by the design environment.

Figure 11 represents the proposed design flow to implement asynchronous processors on Altera FPGAs. The inputs of the design flow are the simulation model and the synthesis model of an asynchronous processor.

The proposed design flow starts from RTL simulation to check functional correctness for the simulation model

with a test sequence. We use the ModelSim-Altera for logic simulation.

After RTL simulation, we extract all of paths related to setup, hold, and control initialization constraints (i.e., $sdp_{i,l}$, $scp_{i,l}$, $hdp_{i,k}$, $hcp_{i,k}$, cfp_{i_j}, cbp_{i_j}) in the synthesis model. To analyze path delay such as $t_{maxsdp_{i,l}}$ correctly by TimeQuest Timing Analyzer in the Quartus Prime, we generate $report\_timing$ commands and $report\_path$ commands. $report\_timing$ commands are used to analyze path delays between registers to obtain setup and hold times t_{setup} and t_{hold} for registers. $report\_path$ commands are used for other paths. Note that Altera recommends us to set start and end points of paths with primary inputs, registers (flip-flops and latches), and primary outputs. On the other hand, most of paths related to timing constraints in the bundled-data implementation starts or ends by other pins or nets through registers. For example, $sdp_{i,l}$ starts from the output of sd_{i-1_2} to the destination register through the source register. In such cases, we divide paths into sub-paths and prepare $report\_timing$ and $report\_path$ commands for divided sub-paths. For example of $sdp_{i,l}$, a $report\_path$ command is prepared to analyze from the output of sd_{i-1_2} to the source register and a $report\_timing$ command is prepared to analyze from the source register to destination register.

In the design flow, we synthesize bundled-data implementation without any constraints at first (Synthesis1). Then, we decide whether we generate placement constraints or not. There are two possibilities to generate placement constraints. First is to fix the locations of placed resources in the first synthesis (Back Annotate). Second is to prepare a region to place logics of a given processor model (Create a Region). If we use the placement constraints, we carry out the second synthesis (Synthesis2). From the static timing analysis result for the first or second synthesis, we analyze the global cycle time gct of the synthesized circuit. Then, we generate the maximum path delay constraints for all paths with the global cycle time gct so that the global cycle time of iterative synthesis results closes to gct. Then, with the generated constraints, we repeat synthesis and static timing analysis (STA) until simultaneous constraints are satisfied (Synthesis3). Then, we repeat synthesis and and STA until all other timing constraints are satisfied (Synthesis4). If some of timing constraints are violated, we carry out delay adjustment for corresponding delay elements. Finally, through the gate-level simulation for the synthesized processor, we program the synthesized processor on the target FPGA. In the rest of this sub-section, we describe the generation of constraints and the approach for delay adjustment.

4.2.1 Generation of design constraints

Generation of the Maximum Delay Constraints. We assign the maximum delay constraints to all paths related to setup constraints using gct obtained from the STA results by TimeQuest Timing Analyzer in Quartus Prime with

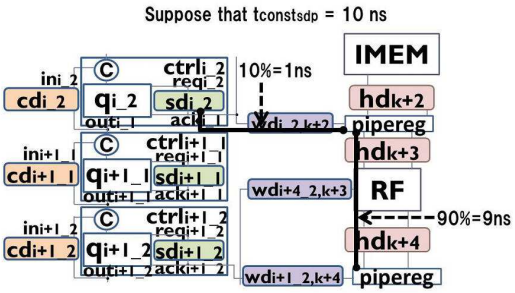

Figure 12: Generation of the maximum delay constraints.

$report\_timing$ and $report\_path$ commands.

From the STA results, first, we analyze local cycle time lct_i for each pipeline stage and global cycle time gct. Second, we decide the margin $sm_{i,l}$ for $t_{maxsdp_{i,l}}$. Third, we decide two parameters $scpmargin$ and $diff$. $scpmargin$ represents a margin between $t_{maxsdp_{i,l}}$ and $t_{minscp_{i,l}}$. Larger $scpmargin$ may result in that setup constraints seem to be satisfied easily. However, it degrades the performance of the synthesized processor because it may lengthen gct after the third synthesis. $diff$ represents the difference between $t_{maxscp_{i,l}}$ and $t_{minscp_{i,l}}$.

The maximum delay constraints for $scp_{i,l}$, $t_{constcp}$, are calculated by the following equation.

$$t_{constcp} = gct \tag{5}$$

The maximum delay constraints for $sdp_{i,l}$, $t_{constdp}$, are calculated by the following equation.

$$t_{constdp} = t_{constcp} - scpmargin - diff - sm_{i,l} \tag{6}$$

As same as $report\_timing$ and $report\_path$, we assign the maximum delay constraints to sub-paths of $scp_{i,l}$ and $sdp_{i,l}$ if these paths include several registers. From the STA results, we decide the ratio of delay for each sub-path. For example, suppose that $t_{constdp}$ for $sdp_{i,l}$ in Fig.12 is 10 ns and the ratio of delay from sd_{i_2} to the source register is 10% of $t_{maxsdp_{i,l}}$ obtained from STA. Then, the maximum delay constraint from sd_{i_2} to the source register becomes 1 ns and the maximum delay constraint from the source register to the destination register becomes 9 ns.

We use set_max_delay commands to represent the maximum delay constraints. We prepare a Synopsys Design Constraint (SDC) file which includes all of set_max_delay commands.

Generation of Placement Constraints. There are two approaches to generate placement constraints. The first approach is to make a region for placement. In the first synthesis report (Synthesis1), we can get the information about the number of used logic elements. From the number of logic elements, we decide a region of FPGA. The region is created by using $LogicLock$ in Quartus Prime.

The second approach is to fix the locations of placed resources in the first synthesis. To realize the second ap-

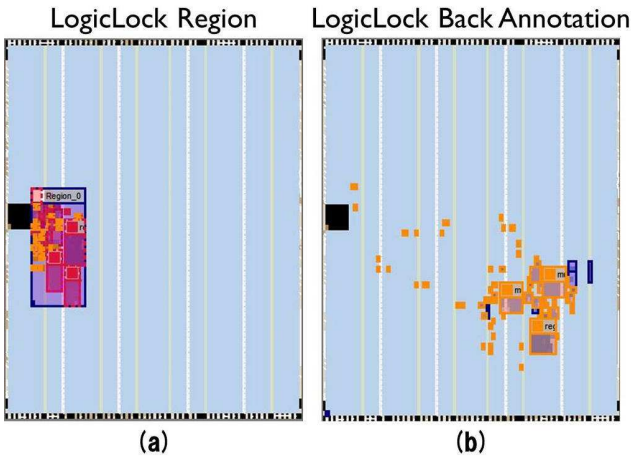

Figure 13: Effect of placement constraints: (a) based on a region and (b) based on the back annotation.

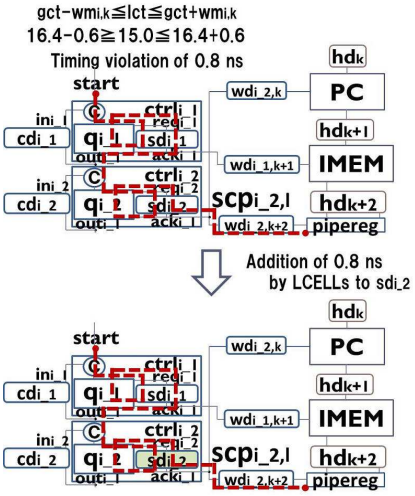

Figure 14: Delay adjustment for simultaneous writing constraints.

proach, we assign $LogicLock$ for each resource (i.e., datapath resources such as registers and control modules) in the synthesis model (Fixing Modules). Through the first synthesis with placement constraints by $LogicLock$, we back annotate the locations of used logic elements, pins, multipliers, and memories in the first synthesis result to a constraint file using Quartus Prime. The locations of resources are represented by $set\_location\_assignment$ commands in the constraint file.

As the second approach fixes the locations of the logic to the first synthesis result, it may reduce the number of delay adjustment. However, it may worse performance if more delay elements are required because the placed logic may affect the placement of newly introduced LCELLs for delay elements. On the other hand, the first approach may allow to place newly introduced LCELLs so that they can be placed freely inside the region. However, it may increase the number of delay adjustments. Figure 13(a) and (b) represent placed logics in the target FPGA when the first and the second approaches are used.

4.2.2 Delay adjustment

From the static timing analysis (STA) result with $report\_timing$ and $report\_path$ commands, simultaneous writing constraints are checked. For a given margin $gctmargin$ for the global cycle time gct, sd_{i_j} or $wd_{i\_j,k}$ are adjusted so that all of $t_{maxscp_{i,l}}$ are within $gct \pm gctmargin$. sd_{i_j} is adjusted if all of $t_{maxscp_{i,l}}$ are out of $gct \pm gctmargin$ while $wd_{i\_j,k}$ is adjusted if only corresponding $t_{maxscp_{i\_j,l}}$ is out of $gct \pm gctmargin$. We generate Verilog HDL models for adjusted delay elements. Figure 14 represents an example of delay adjustment for simultaneous writing constraints.

Next, we adjust setup, hold, and control initialization constraints. As the adjustment of hd_k affects to $sdp_{i,l}$, we adjust hold constraints at first and then we adjust setup constraints reflecting the added or removed delay for hd_k to $sdp_{i,l}$. The adjustment of cd_{i_j} does not affect to the paths related to setup and hold constraints. Therefore, there is no order for delay adjustment between setup (hold) constraints and control initialization constraints. From the STA result using $report\_timing$ and $report\_path$ commands, we assign the delays to both left side and right side of in the inequalities (1), (2), and (3) (see Section 2). By the subtraction of the right side value from the left side value, we add LCELLs to corresponding delay elements to satisfy the constraint if the subtraction result is a negative value (i.e., a timing violation). On the other hand, we remove LCELLs from corresponding delay elements if the left side value is larger than the right side value plus the margin $scpmargin$. Although that the left side value is larger than the right side value means no timing violation, the large left side value results in that gct becomes a large value. Therefore, we remove LCELLs if the left side value overs the right side value plus $scpmargin$. Figure 15 represents an example of delay adjustment for setup constraints.

After we generate Verilog HDL files for adjusted delay elements are generated, we repeat synthesis, STA, and delay adjustment until all of timing constraints are satisfied.

5 Experiments

5.1 Experimental results

In the experiments, we design asynchronous MIPS processor using the proposed modeling method and design flow. We refer to a synchronous MIPS processor in [12] for modeling of asynchronous MIPS processor. Figure 16 represents the block diagram of the MIPS processor. The execution of the synchronous MIPS processor is 5 stage pipeline (instruction fetch, instruction decode, execution,

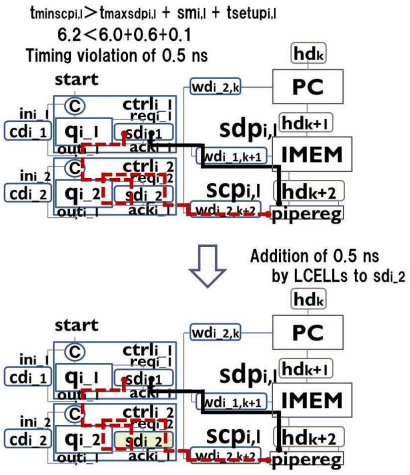

Figure 15: Delay adjustment for a setup constraint.

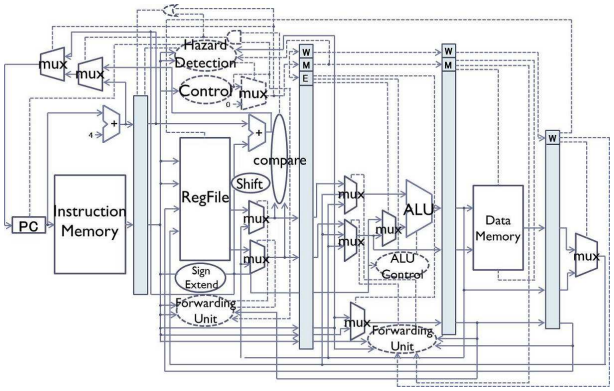

Figure 16: Block diagram of the MIPS processor in [12].

data memory access, and write back). The asynchronous MIPS processor supports 9 instructions (lw, sw, j, beq, add, sub, or, and, slt).

We compare the designed asynchronous MIPS processors with the synchronous MIPS processor in terms of area, execution time, dynamic power consumption, and energy consumption. The used synthesis tool and simulation tool are Altera Quartus Prime ver.15.1 and ModelSim-Altera ver.10.4b. TimeQuest timing analyzer in Quartus Prime is also used to analyze path delays. The target device is Altera Cyclone IV (EP4CE115F29C7).

Initially, we synthesize the synchronous MIPS processor "Sync" by changing clock cycle time so that the clock frequency is maximum. The clock cycle time of "Sync" is 16 ns. Then, we design three asynchronous MIPS processors. "Async1" is the one without placement constraints. "Async2" is the one that the asynchronous MIPS processor is placed inside a region represented by a placement constraint. "Async3" is the one that the locations of all used resources are fixed to the same locations as the first synthesis in the design flow. Table 1 represents parameters for three asynchronous MIPS processors. We decide

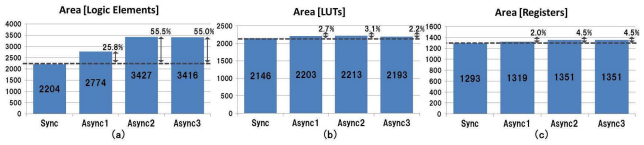

Figure 17: Area of MIPS processors: (a) the number of LEs, (b) the number of LUTs, and (c) the number of registers (DFFs).

Figure 18: Execution time of MIPS processors: (a) multiplication and (b) matrix multiplication.

$gctmargin$, sm_i, $scpmargin$, and $diff$ from the STA results for the first and second synthesis. gct is the value obtained by the STA result for synthesized circuit where all timing constraints are satisfied.

Table 2 represents the number of delay adjustments for three MIPS processors. "Simul" and "Others" represent the number of delay adjustments for simultaneous writing constraints and other timing constraints such as setup constraints.

Figure 17 represents the area of the MIPS processors. Figure 17(a), (b), and (c) represent the number of used LEs, LUTs, and registers (DFFs). These are reported by Quartus Prime. Comparing to "Sync", the increase of LUTs and registers in three asynchronous MIPS processors is less than 5%. However, the number of LEs is increased 25% for "Async1", 54.5% for "Async2", and 54.0% for "Async3". This is because LUTs and DFFs are separated to different LEs even though an LE has one LUT and one DFF.

Figure 18 represents the execution time obtained by gate-level simulation after the designs using ModelSim-Altera. We prepare two test benches, (a) a multiplication and (b) a matrix multiplication. In both cases, compared to "Sync", the execution time is increased 12.5% for "Async2" and 18.8% for "Async3" and decreased about 13.8% for "Async1". This is because the global cycle time of "Async2" and "Async3" is increased and the global cycle time of "Async1" is decreased compared to the cycle time of "Sync" (see Table 1).

Figure 19 represents the dynamic power consumption obtained by PowerPlay Power Analyzer in Quartus Prime assigning a value change dump (.vcd) file generated by gate-level simulation. Compared to "Sync", the dynamic power consumption of "Async1" is increased 22.0% for the multiplication and 22.7% for the matrix multiplication. On

Table 1: Used parameters for asynchronous MIPS processors ([ns]).

name	gct	$gctmargin$	sm_i	$scpmargin$	$diff$
Async1	13.5	1.2	0.6	0.6	1
Async2	17.0	1.2	0.6	0.6	0.9
Async3	18.9	1.2	0.6	0.6	1

Table 2: The number of delay adjustments.

name	Simul	Others
Async1	3	0
Async2	6	0
Async3	3	0

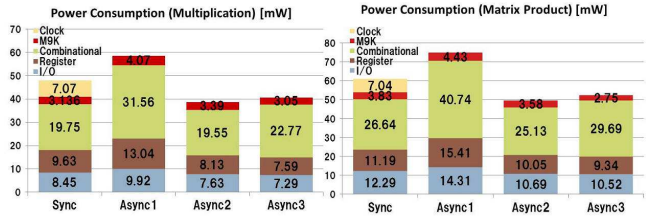

Figure 19: Dynamic power consumption of MIPS processors: (a) multiplication and (b) matrix multiplication.

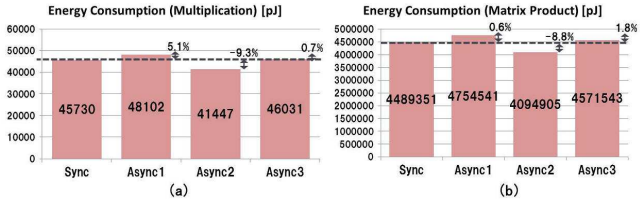

Figure 20: Energy consumption of MIPS processors: (a) multiplication and (b) matrix multiplication.

Table 3: Ratio of dynamic power consumption ([%]).

name	test bench	block	routing
Async1	multiplication	47.4	52.6
	matrix multiplication	46.9	53.1
Async2	multiplication	52.3	47.7
	matrix multiplication	50.9	49.1
Async3	multiplication	46.8	53.2
	matrix multiplication	46.3	53.7

the other hand, compared to "Sync", the dynamic power consumption of "Async2" and "Async3" is reduced 19.4% and 15.3% for the multiplication and 18.9% and 14.2% for the matrix multiplication. As dynamic power consumption depends on frequency which is a reciprocal of the clock cycle time, the longer global cycle time results in the lower dynamic power consumption. On the other hand, the dynamic power consumption caused by the global clock signals (Clock in Fig.19) is reduced in all of asynchronous MIPS processors.

Figure 20 represents the energy consumption which is obtained by the product of execution time and dynamic power consumption. Compared to "Sync", in both multiplication and matrix multiplication, the energy consumption is increased 5.1% and 0.6% for "Async1" and 0.7% and 1.8% for "Async3". On the other hand, compared to "Sync", in both multiplication and matrix multiplication, the energy consumption is reduced 9.3% and 8.8% for "Async2".

5.2 Discussion

The experimental results show that the proposed modeling method and design flow generate two possibilities of asynchronous processors on commercial FPGAs. First it to generate a high performance asynchronous processor like "Async1". As the global cycle time is smaller than the shortest clock cycle time, it increases throughput. To generate the high performance one, we should rely on commercial design environment without placement constraints. Second is to generate a low energy asynchronous processor like "Async2". To generate the low energy one, we should prepare a region to place the logics of processors.

In all of three asynchronous processor designs, there is no big difference for the number of delay adjustments. On the other hand, interestingly, to satisfy simultaneous writing constraints may reduce the possibilities of other timing violations.

To obtain more low power asynchronous processors on commercial FPGAs, we should reduce the number of used logic elements by packing LUTs and DFFs to the same LEs. As mentioned in Figure 17, LUTs and DFFs of three asynchronous processors are separated to different LEs compared to "Sync". This results in the increase of dynamic power consumption due to the use of routing resources such as switches among LABs in which consumes more power. In fact, in all of three asynchronous MIPS processors, the dynamic power consumption caused by routing resources is about half of the total dynamic power consumption as shown in Table 3. To pack LUTs and DFFs to the same LEs results in the reduction of the number of used LEs which in turn the reduction of dynamic power consumption by routing resources. We consider this issue in our future work.

6 Conclusions

In this paper, we proposed a modeling method and a design flow to implement asynchronous processors on commercial FPGAs. Using the proposed modeling method and design flow, we designed three asynchronous MIPS processors. Comparing with a synchronous MIPS processor, one of them reduced the global cycle time which results in 13.8% performance improvement and another one reduced the energy consumption 9.3% for a multiplication and 8.8% for a matrix multiplication.

In our future work, we are going to reduce the number of used logic elements to reduce the dynamic power consumption of routing resources. In addition, we are going to design different asynchronous processors to generalize the proposed method.

Acknowledgement

This work is partially supported by Grant-in-Aid for Scientific Research from Japan Society for the promotion of science (#15K00080).

References

[1] A. Putnam et al., "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services", Proc. ISCA'14, pp.13–24, 2014.

[2] M. Tranchero and L. M. Reyneri, "Exploiting synchronous placement for asynchronous circuits onto commercial FPGAs", Proc. FPL, pp.622–625, 2009.

[3] Q. T. Ho et al., "Implementing Asynchronous Circuits on LUT Based FPGAs", Proc. FPL, pp.36–46, 2002.

[4] H. Saito et al., "A Floorplan Method for Asynchronous Circuits with Bundled-data Implementation on FPGAs", Proc. ISCAS, pp.925–928, 2010.

[5] K. Takizawa et al., "A Design Support Tool Set for Asynchronous Circuits with Bundled-data Implementation on FPGAs", Proc. FPL, pp.1–4, September 2014.

[6] Nikolaos Minas et al., "FPGA Implementation of an Asynchronous Processor with Both Online and Offline Testing Capabilities", Proc. Async, pp.128–137, 2008.

[7] Jens Sparso and Steve Furber, "Principles of Asynchronous Circuit Design: A Systems Perspective", Springer, 2001.

[8] F. U. Rosenberger et al., "Q-Modules:Internally Clocked Delay Insensitive Modules", IEEE Transaction of Computer, vol. C-37, no.9, pp. 1005-1018, 1988.

[9] I. E. Sutherland, "Micropipelines", Communications of the ACM, vol.32, issue 6, pp.720–738, 1989.

[10] Altera Cyclone IV FPGA, "https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html"

[11] S. Iwasaki, "Design and Evaluation of a Low Power Asynchronous AVR Processor considering a Cycle Time Constraint", Master Thesis, the University of Aizu, 2014.

[12] D. A. Patterson and J. L. Hennessy, Computer Organization and Design", Morgan Kaufmann, 2013.