Constraining Operation Delay for Dynamic Power Optimization of Asynchronous Circuits

Shunya Hosaka
University of Aizu
Aizuwakamatsu 965-8580, Japan
m5191127@u-aizu.ac.jp

Hiroshi Saito
University of Aizu
Aizuwakamatsu 965-8580, Japan
hiroshis@u-aizu.ac.jp

ABSTRACT

In this paper, we propose a dynamic power optimization method for asynchronous circuits with bundled-data implementation constraining operation delay. In asynchronous circuits, we can change the execution time of each operation freely under a given latency constraint. Therefore, the proposed method relaxes the execution time of operations which consume more power while it tightens the execution time of operations which consume less power. In the experiments, we evaluate the effects of the proposed method by comparing area, execution time, power consumption, and energy consumption among synchronous circuits, asynchronous circuits without the proposed method, and asynchronous circuits with the proposed method. As a result, we can reduce dynamic power and energy consumption about 11% and 14% on average.

Categories and Subject Descriptors

VLSI [RTL Design]: Asynchronous Circuits

General Terms

Theory

Keywords

Dynamic Power Optimization, Mobility

1. INTRODUCTION

Current VLSIs are mostly based on synchronous circuits where circuit components are controlled by global clock signals. However, synchronization failures caused by clock skews and power consumption on the clock network are getting to be the critical problems when the integration technology of VLSIs is more and more advanced. There are no problems related to clock signals in asynchronous circuits since they do not use global clock signals. Instead, circuit components in asynchronous circuits are controlled by local handshake signals. As circuit components are executed when required, asynchronous circuits are potentially low power consumption and low electro-magnetic interference.

However, it is well known that designing asynchronous circuits is more difficult than designing synchronous circuits. We have to consider delay model and data encoding when we design asynchronous circuits. If the delay model and data encoding are not appropriate, power reduction by asynchronous circuits may be small. To reduce power consumption, it is necessary for asynchronous circuits to optimize power.

In this paper, we propose a dynamic power optimization method for asynchronous circuits with bundled-data implementation constraining operation delay. In asynchronous circuits, we can change the execution time of operations freely under a given latency constraint. Therefore, the proposed method relaxes the execution time of operations which consume more power while it tightens the execution time of operations which consume less power.

Several low power design methods for asynchronous circuits have been proposed. Huang et al. [2] proposed a power optimization method which isolates the operands of functional units if operations are not required. Hansen et al. [1] proposed an optimization method to optimize latency, area, and energy considering many-to-many functional units mappings using the branch-and-bound algorithm. Jeong et al. [3] proposed an optimization method to reduce the overhead of dual-rail asynchronous circuits based on eager evaluation using multiple outputs block level relaxation. Plana et al. [4] proposed a throughput optimization method for non-pipelined asynchronous circuits avoiding data hazards and glitches using a concurrent sequencer. The improvement of the throughput results in energy optimization. Compared to these methods, out proposed method reduces dynamic power constraining operation delays.

The rest of this paper is as follows. In section II, we describe asynchronous circuits with bundled-data implementation, data flow graph, and the calculation of mobility. In section III, we describe the proposed dynamic power optimization method. In section IV, we evaluate the proposed method in terms of area, performance, dynamic power, and energy, comparing with synchronous circuits and asynchronous circuits without the proposed method. Finally, in section V, we describe conclusions and future work.

IWAIT '15, Oct. 8 – 10, 2015, Aizu-Wakamatsu, Japan.

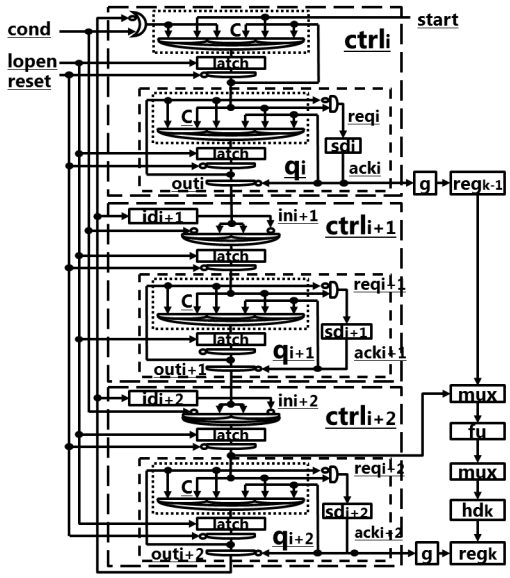

Figure 1: Bundled-data implementation

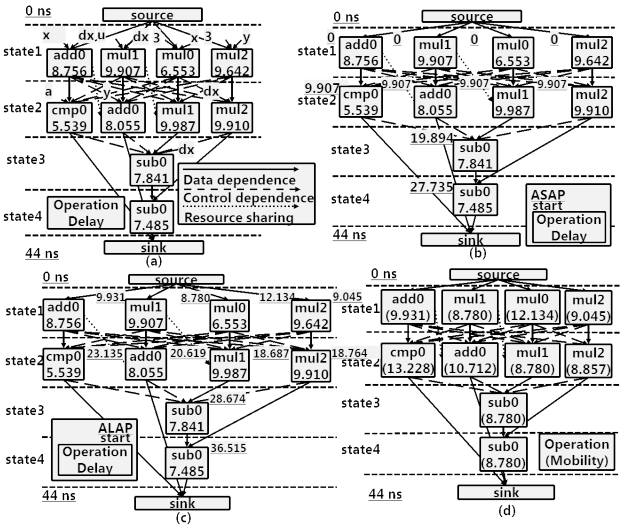

Figure 2: (a) DFG, (b) ASAP scheduling, (c) ALAP scheduling, (d) Mobility

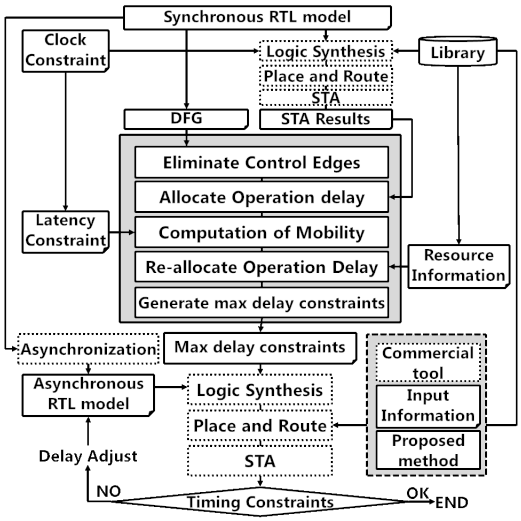

Figure 3: Design flow

2. BACKGROUND

2.1 Bundled-data Implementation

Bundled-data implementation represents N-bit data using N+2 signals called "bundle"'. Additional two signals represent local handshake signals, request and acknowledge. In bundled-data implementation, delay elements whose delays are longer than data-path delays are put on request signals to guarantee the completion of operations and the timing of register writing. Therefore, the performance of bundled-data implementation depends on the delay of the control circuit including delay elements. The detail of bundled-data implementation is described in [5].

2.2 Data Flow Graph

Data Flow Graph (DFG) represents data flow of an application. DFG is defined by the following expression.

$$DFG =< N, E > \quad (1)$$

N is the set of nodes. A node n_m shows an operation in the application. Node n_m has a label of allocated resource with the operation delay. E is the set of edges. An edge represents data dependence, resource sharing, or control dependence. A data dependence represents dependence of data between operations. A resource dependence represents resource sharing of operations. A control dependence represents which state an operation is executed. Fig.2(a) shows an example of DFG. In Fig.2(a), source and sink shows start and end nodes. There are no operations.

2.3 Mobility

Mobility of operations used for dynamic power optimization in the proposed method is defined using As Soon As Possible (ASAP) scheduling and As Late As Possible (ALAP) scheduling. ASAP scheduling schedules each operation as soon as possible when they can be scheduled. Fig.2(b) shows ASAP scheduling for the DFG in Fig.2(a). ALAP scheduling schedules each operation as late as possible under a given latency constraint. Fig.2(c) shows ALAP scheduling for the DFG in Fig.2(a). The mobility of node n_m is defined by the following expression.

$$mobility_{n_m} = t_{alap_{n_m}} - t_{asap_{n_m}} \quad (2)$$

$t_{alap_{n_m}}$ shows ALAP start time of node n_m. $t_{asap_{n_m}}$ shows ASAP start time of node n_m. Fig.2(d) shows calculated mobility for nodes in the DFG of Fig.2(a).

3. PROPOSED METHOD

3.1 Design Flow

The proposed method generates max delay constraints to relax the execution time of operations which consume more power while tightening the execution time of opera-

tions which consume less power. Fig.3 shows a design flow with the proposed method. The proposed method is used in the process of generating an asynchronous RTL model from a synchronous RTL model. As inputs of the proposed method, we need to prepare a DFG corresponding to the synchronous RTL model's data-path circuit, data-path delays from Static Timing Analysis (STA), resource information file, and latency constraint. First, we allocate data-path delays from STA to each node in DFG. Next, we compute the mobility of operations from ASAP scheduling and ALAP scheduling. Then, we re-allocate operation delays to utilize mobility well. Finally, we generate max delay constraints for each data-path.

We consider two cases for re-allocation of operation delays.

- Case 1. Keep the state that operations are executed.
- Case 2. To maximize the use of mobility, we change the state of operations in non-critical paths to be executed.

Case1 reduces dynamic power by changing each state's execution time. In addition to Case1, Case2 reduces dynamic power by changing the state of operations in non-critical paths to be executed.

3.2 Inputs

We need to prepare a DFG corresponding to the synchronous RTL model's data-path circuit, data-path delays from STA, a resource information file, and latency constraint. To correspond to the synchronous RTL model's data-path circuit, in DFG, node n_m is categorized into each state, and node n_m is re-allocated a label that used operation. Between nodes are connected by edges that represent data dependence, resource sharing, and control dependence. Fig.2(a) shows an example of DFGs. We use STA to analyze data-path delays. The resource information file represents the range of each operation's delay. The proposed method gives high priority to operations that have long execution delay. Operation that takes long execution time can use mobility as much as possible. Latency constraint that represents the constraint from arrival of input signals to generation of output signals is computed by the product of clock cycle time and clock cycles.

3.3 Operation Delay Re-allocation

In the Case2, we eliminate control edges from a given DFG since it allows to change the execution timing of operations in non-critical paths. However, in the case that one operation's source register is shared as other operation's destination register in the same state, it may change the source register if the other operation completes early. In such a case, we need to keep control edges. Fig.4(a) shows the DFG that is eliminated control edges from the DFG in Fig.2(a). In this figure, source register of add0 is shared as destination register of mul1 at state2. Because of this, the control edge that exists between add0 to sub0 that comes executed after mul1 is kept. Next, we apply ASAP scheduling and ALAP scheduling and compute mobility in both Case1 and Case2. Fig.2(d) and Fig.4(b) represent Case1's mobility and Case2's mobility.

We execute operation delay re-allocation according to the flows in Fig.5. Fig.5(a) represents the Case1. First, we identify the critical path. We explore the number of resource

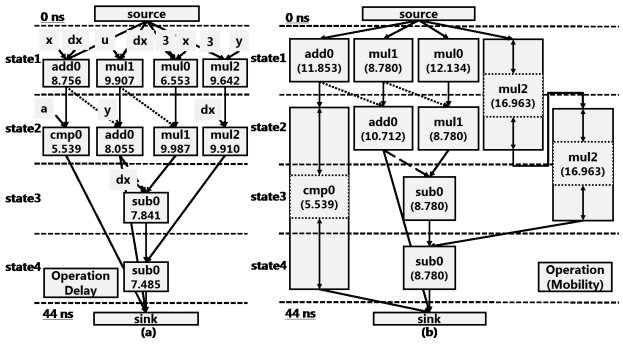

Figure 4: (a) DFG reduced control edges from Fig.2 (a), (b) Case2's mobility

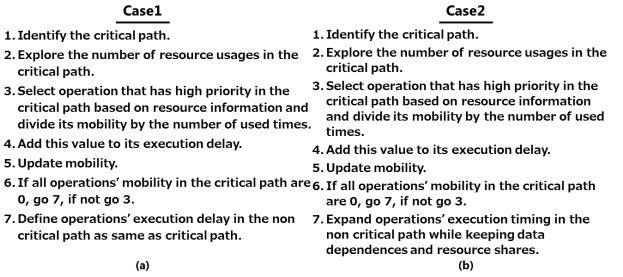

Figure 5: Re-allocation of execution delay (a) Case1, (b) Case2

usages in the critical path. Next, we select the operation in the critical path which has the highest priority based on the resource information file. Then, we divide the mobility by the number of resource usages of the corresponding resource. Next, we add this value to the execution delays of operation in the critical path that uses the same resource. Re-allocated execution delay corresponds to the delay of the state that belonging selected operation. Next, we update the mobility of all operations and select the operation that has the next highest priority and compute the execution delay. We repeat this processes until all mobility in the critical path become 0. When operation delay re-allocation for the critical path is finished, we set the execution delay of operating in non-critical paths based on the operation delay in the critical path. Fig.6 shows the operation delay re-allocation result for Case1.

Fig.5(b) represents the Case2. As same as Case1, first we re-allocate execution delay to operations in the critical path. However, there is a difference for the re-allocation of execution delay to operating in non-critical paths. In the Case2, there are less control edges than Case1. So we can use mobility more effectively. We may allocate more delay to operation in non-critical paths while keeping data dependences and resource sharing. Fig.7 shows the operation delay re-allocation result for Case2.

3.4 Outputs

Finally, we generate max delay constraints for all data-path based on re-allocated operation delays. They are classified paths, between registers, paths from primary inputs

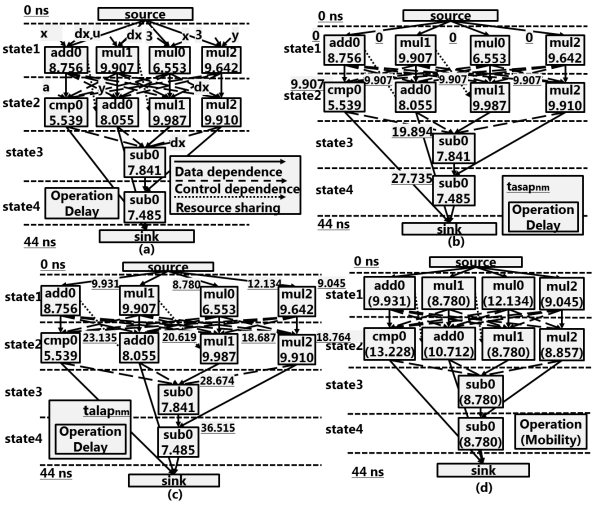

Figure 6: Operation delay re-allocation result for Case1

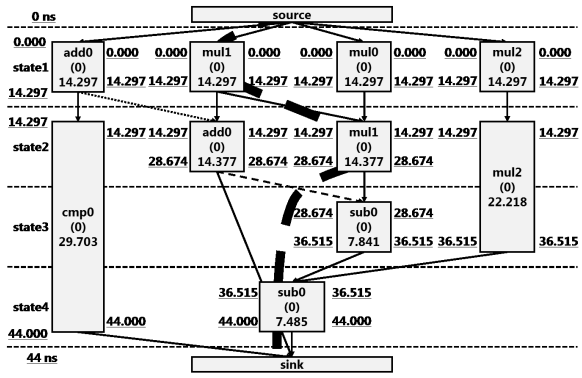

Figure 7: Operation delay re-allocation result for Case2

to registers, paths from registers to primary outputs, and paths from control module's latch to registers. Max delay constraints were generated using *set_max_delay* command as follows.

$$set\_max\_delay \ -from \ START \ -to \ END \ -through \ THROUGH \ DELAY$$

START and END represents path's start and end point. THROUGH represents multiplexer or functional unit's input or output port. DELAY represents the re-allocated execution delay. It may be required to change register write signals and multiplexer control signals in Case2. So, the proposed method generates a txt file which represents the change of control timing. Fig.8 shows the txt file. Designers are required to chamge asynchronous RTL model.

4. EXPERIMENTS

In the experiments, we synthesize differential equation solver (Diffeq) and ellipse wave filter (EWF) using the proposed design flow in Fig.3. We evaluate area, execution time, dynamic power consumption, and energy consumption of synchronous circuits (Sync), asynchronous circuits that the

```
[9]                          [20]
reg8 : ack7 -> ack8          reg1 : ack10 -> ack11
mux  : in7 -> in8            mux  : in10 -> in11
```

Figure 8: txt output following Case2

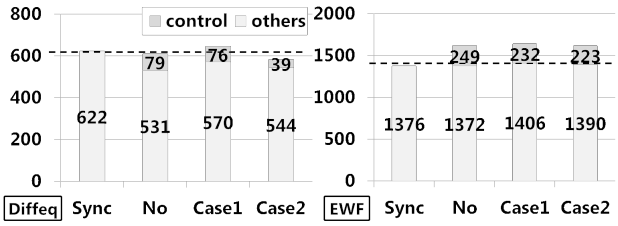

Figure 9: # of Logic Elements

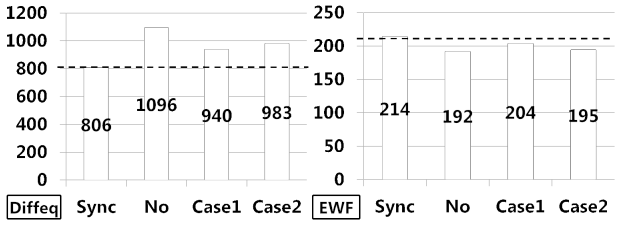

Figure 10: Execution time [ns]

proposed method is not applied (No), and asynchronous circuits using the proposed method (Case1 and Case2) to confirm the effect of the proposed method for reducing dynamic power consumption. For the experiments, we implemented the proposed method using Java and Eclipse.

We used Altera FPGA Cyclone IV (EPCE115F29C7). We used Quartus II 13.1 for synthesis. First we explore the fastest synchronous circuits satisfying timing constraints. Clock cycle time of Diffeq was 11ns and clock cycle time of EWF was 14ns. We generated asynchronous RTL models from synchronous RTL models. These models are modeled by Verilog HDL. The execution delay of operating used in the proposed method is defined by STA results for all data-paths of synchronous circuits. After that, we generated maximum delay constraints based on Case1 and Case2 under a given latency constraint that was computed by the product of clock cycle time and clock cycles. Finally, we synthesized asynchronous circuits using maximum delay constraints and adjusted delay elements repeatedly until all timing constraints for bundled-data implementation [5] were satisfied.

Fig.9 shows area. Area was evaluated from the report of Quartus II. In Diffeq, Case1 was increased about 6% and Case2 was reduced about 4%, compared with No. In EWF, Case1 was increased about 1% and Case2 was reduced about 1%, compared with No. In comparison with Sync, area was increased about 4% to 19%. This is because the overhead of the control circuits.

Fig.10 shows execution time that can be evaluated by simulating an arbitrary test pattern using ModelSim-Altera. In Diffeq, Case1 and Case2 were reduced about 14% and 11%, compared with No. In EWF, Case1 and Case2 were in-

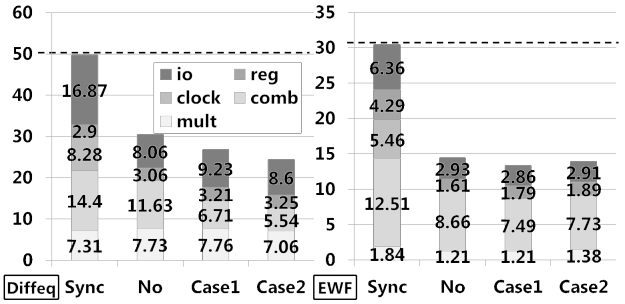

Figure 11: Dynamic power [mW]

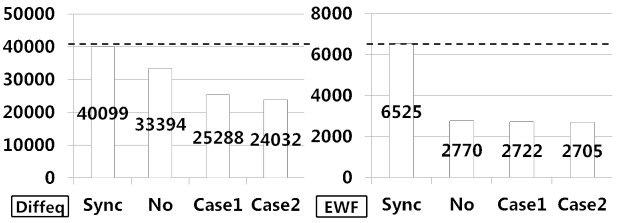

Figure 12: Energy [pJ]

creased about 6% and 1%, compared with No. Execution time was increased about 17% to 36% in Diffeq. In Diffeq, operations are repeatedly executed in a loop until a condition is satisfied. The idle phase of the control circuit between loop bodies could not be hidden. This is the reason why execution time is increased.

Fig.11 shows dynamic power that can be evaluated by PowerPlay Power Analyzer with a value change dump file obtained by simulation. In Diffeq, Case1 and Case2 were reduced about 12% and 20%, compared with No. In EWF, Case1 and Case2 were reduced about 7% and 4%, compared with No. Dynamic power is especially reduced in combinational logics due to the effect of the proposed method. In some cases where total area is increased, dynamic power could be reduced. This is considered as the reduction of dynamic power at rooting resources.

Fig.12 shows energy consumption that was calculated by the product of execution time and dynamic power. In Diffeq, Case1 and Case2 were reduced about 24% and 28%, compared with No. In EWF, Case1 and Case2 were reduced about 2% and 3%, compared with No.

In the experiments, we could reduce dynamic power consumption more than asynchronous Diffeq (No) and EWF (No) using the proposed method. Especially, Case2 reduced more compared to Case1 due to the use of mobility more.

5. CONCLUSIONS

In this paper, we proposed a dynamic power optimization method for asynchronous circuits constraining operation delay using the mobility of operations. The proposed method generates maximum delay constraints to relax the execution time of operations that consume more power, while it tightens the execution time of operations that consume less power. We proposed two types of operation delay reallocation methods. In the experiments, we have confirmed that the proposed method reduced dynamic power consumption and energy consumption about 11% and 14% on average.

6. REFERENCES

[1] J. Hansen and M. Singh. A fast branch-and-bound approach to high-level synthesis of asynchronous systems. Asynchronous Circuits and Systems (ASYNC), IEEE International Symposium on, 10:107–116, May 2010.

[2] Y. Huang and W. Shi. An optimized de-synchronization flow for power and performance optimization. Systems and Informatics (ICSAI), International Conference on, 5:71–75, May 2012.

[3] C. Jeong and S. Nowick. Block-level relaxation for timing-robust asynchronous circuits based on eager evaluation. Asynchronous Circuits and Systems (ASYNC), IEEE International Symposium on, 10:107–116, May 2008.

[4] L. Plana and S. Nowick. Architectural optimization for low-power nonpipelined asynchronous systems. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 10:56–65, August 2002.

[5] K. Takizawa, S. Hosaka, and H. Saito. A design support tool set for asynchronous circuits with bundled-data implementation on fpgas. Field Programmable Logic and Applications (FPL), International Conference on, 4:1–4, September 2014.