Dual-Rail/Single-Rail Hybrid Logic Design for High-Performance Asynchronous Circuit

Zhengfan Xia, Shota Ishihara, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi,980-8579, Japan
Email: {xiazhengfan@kameyama., ishihara@kameyama., hariyama@, kameyama@ }ecei.tohoku.ac.jp

Abstract—This paper presents a fine-grain pipelined asynchronous circuit that uses a mixture of dual-rail and single-rail logic. Dual-rail logic is limited to construct a stable critical path. Based on this critical path, the handshake control circuit is greatly simplified, which improves the performance of speed and power consumption. On the other hand, non-critical paths are composed of single-rail logic which has small logic overhead and the entire pipelined circuit has no intermediate registers or latches. To evaluate the proposed design method, an array style multiplier is designed and simulated in a 65nm design rule. The multiplier works as high as 4.35G data-set/s. Compared to the classical synchronous circuit, the proposed circuit has no active power consumption when there are no data operation. Even the circuits work at peak speed, the proposed circuit still reduces the power consumption by 35%.

Keywords— asynchronous circuit, dual-rail logic, single-rail logic

I. INTRODUCTION

Nowadays, the semiconductor industry is giving serious consideration to the adoption of asynchronous circuit technology which uses local handshake instead of externally supplied global clock. As technology scaling progresses, the clock design becomes an obstacle to high-performance VLSI systems. Clock distribution is a serious challenge with the requirement of high-speed and low-power in VLSI systems design [1], and the clock source has large electromagnetic missions which affects circuits design, for example, the mixed-signal circuit design [2]. On the other hand, asynchronous circuits avoid issues related to global clock. It has potential for high-speed, low-power, low electromagnetic interference, and a natural match with heterogeneous system timing [3]. Although asynchronous circuit has these attractive properties for VLSI systems, it also has some drawbacks. One of the problems is that the handshake control logic that implements the handshaking normally has large overhead. It not only consumes silicon area but also affects the circuit speed and power consumption.

Depending on the encoding scheme, asynchronous circuit implementations are separated into two types:

- Bundled-data asynchronous circuit
- Dual-rail asynchronous circuit

Fig.1 shows asynchronous circuits. Fig.1(a) is a bundled-data asynchronous circuit and Fig.1(b) is a dual-rail asynchronous circuit. Bundled-data asynchronous circuit is quite similar to synchronous circuit, which just replaces the global clock tree by local handshake control circuit. It maintains the usage of single-rail logic, or static logic, and storage elements, registers or latches. On the other hand, dual-rail asynchronous circuit just uses a completion detector as handshake control circuit at each stage, and the overhead of explicit storage elements are all removed. The problem is that, such efficient design is realized at the expense of dual-rail overhead and the overhead of the completion detector is quite large, which cause it unapplicable at large function block design.

This paper proposes a new asynchronous circuit design method that uses a mixture of dual-rail and single-rail logic. The data detectability of dual-rail logic is used to realize the efficient pipeline architecture same as the dual-rail asynchronous circuit, and single-rail logic is applied to achieve small overhead in logic block. In the conventional dual-rail asynchronous circuit, the completion detector needs to generate a total done signal by detecting the entire datapaths of a function block [4]. This is because the critical datapath varies from one to other datapath depending on input data patterns (the data-dependency problem). However, if a stable critical datapath is available, a simple completion detector is enough to generate a total done signal by detecting the critical
data is separated by a spacer. In fact, bundled-data asynchronous circuit is also based on such idea. Instead of detecting datapaths, a Req signal is sent simultaneously with a bundled-data. In order to guarantee the correct action, Req signal needs to be delayed that it arrives at the next stage later than the bundled-data. Therefore, Fig.1(a) shows that a matched delay is added on the control line. In practice, an accurate technique to implement the matched delay is to duplicate the worst-case critical path of the function block and uses it as delay line [5]. In our proposed design, the worst-case critical path is redesigned to a data-dependency free dual-rail critical path instead of duplicating it and the non-critical paths remain single-rail logic. Fig.2 shows a diagram of dual-rail/single-rail hybrid asynchronous circuit. As a result, the completion detector is simplified to a static NOR gate to detect only the critical datapath. Moreover, the non-critical datapaths have no dual-rail overhead and there are no explicit storage elements between pipeline stages.

II. ARCHITECTURE

The proposed asynchronous circuit is constructed based on a stable critical path. How to get the stable critical path is the main problem to be discussed in this paper. The proposed circuit is fine-grain pipelined, the depth of each pipeline stage is only a single gate. Therefore, the critical path at each pipeline stage would be the slowest gate. Conventional logic gate has data-dependency problem, the slowest gate in function block would vary from one to others depending on input data patterns. Synchronizing logic gates (SLGs) are used to solve this problem. Another problem is that the slowest gate in a pipeline stage might be early triggered by outputs from previous pipeline stage. Then, synchronizing logic gates with a latch function (SLGLs) are extended to solve it. As a result, a stable critical path is available with the help of SLGs and SLGLs. Beside the critical path construction problems, the compatibility between single-rail encoding and dual-rail encoding also need to be considered. Fortunately, a simple converter solves it.

A. 4-phase dual-rail encoding

4-phase dual-rail encoding is used in our design, which encodes a bit onto two wires, \((w_f, w_{	ext{not}})\). Table I shows the code table of the 4-phase dual-rail encoding. The data value 0 is encoded as \((0, 1)\) and 1 is encoded as \((1, 0)\); the spacer is encoded as \((0, 0)\); \((1, 1)\) is not used. Fig.3 shows an example of the 4-phase dual-rail encoding. In the data transfer, every data is separated by a spacer.

| Codeword | \((i, l, w_f, w_{	ext{not}})\) |
|-----------------|--|--|--|--|
| Data 0 | \((0, 1)\) |
| Data 1 | \((1, 0)\) |
| Spacer | \((0, 0)\) |
| Not used | \((1, 1)\) |

Fig. 3. Example of the 4-phase dual-rail encoding.

B. Logic gates

1) Synchronizing logic gates: In dual-rail dynamic logic, the reason of data-dependency problem is that the possible pull-down transistor paths have different stacks and different connections according to different inputs patterns. In order to solve the problem, SLGs are proposed in [6]. In an SLG, there is exactly one path activated according to one data pattern and the stack of all possible paths is kept constant at the sequential position in the pull-down network.

Fig.4 shows dual-rail AND gates. Fig.4(a) shows the conventional dual-rail AND gate. The true side is implemented by \(\text{out}_t = a_t \times b_t\) and the false side by \(\text{out}_f = a_t + b_t\). Depending on whether one or both of the parallel-connected NMOS transistors turn on, the delay of the false side can vary by a factor of two. On the other hand, the NMOS transistors in the true side are in serial connection. For the gate delay, this factor is even larger. Fig.4(b) shows the modified logic gate, synchronizing AND gate, which solves the variation problem. To reduce the gate delay variation, the false side logic expression is changed to \(\text{out}_f = a_t \times b_f + a_f \times (b_t + b_f)\). There is exactly one path activated according to one data pattern and the stack of all possible paths is three at the sequential position. All logic gates can be designed in this way to solve the data-dependency problem.

The characteristics of SLGs are summarized as follows:
- An SLG has a constant stack in all possible pull-down transistor paths.
- The absence of any inputs will postpone the evaluation of an SLG. It means that the SLGs can synchronize the inputs.

Since the evaluation speed of an SLG is strongly related to the magnitude of its stack, the evaluation speed of SLGs is comparable by comparing the magnitude of their stacks. This can be used to roughly find the slowest logic gate in a pipeline stage.
Fig. 6. Structure of dual-rail/single-rail hybrid logic design.

<table>
<thead>
<tr>
<th>Single-rail Encoding</th>
<th>Dual-rail Encoding (out_t, out_f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0</td>
<td>(0, 1)</td>
</tr>
<tr>
<td>Data 1</td>
<td>(1, 0)</td>
</tr>
</tbody>
</table>

Fig. 7. Single-rail to dual-rail encoding converter.

2) Synchronizing logic gates with a latch function: SLGLs are expressed by an AND gate with a latch function whose logic functions are implemented in the pull-down network. Without the presence of the enable signal, SLGLs can not start evaluation. In this way, the enable signal controls the opaque and transparent state of SLGLs. Fig.5 shows synchronizing AND gate with a latch function whose logic functions are expressed by $out_out_t = (a_t \times b_t) \times (en_t + en_f)$ and $out_out_f = (a_t \times b_f + a_f \times (b_t + b_f)) \times (en_t + en_f)$. The latch state is shown in the table.

For the critical path design, if the output of the slowest gate from the previous pipeline stage connects to the enable port of an SLGL, the SLGL would never be early triggered.

3) Single-rail to dual-rail encoding converter: Single-rail logic in our design represents single-rail dynamic logic which is necessary to realize the efficient latch-free architecture. Compared to dual-rail logic, single-rail logic has smaller logic overhead. The truth part of a dual-rail logic can be used as a single-rail logic. For example, the truth part of the dual-rail AND gate in Fig.4 is a single-rail AND gate. The advantage of single-rail logic is that there is no active power consumption when it transfers data 0.

When the logic block is designed using dual-rail/single-rail hybrid logic, the compatibility between the dual-rail encoding and single-rail encoding becomes a problem. It is better to design an encoding converter which has small overhead. Fig.7 shows single-rail to dual-rail encoding converter which just uses a single NOT gate. Dual-rail to single-rail encoding converter is unnecessary because the truth part of dual-rail encoding is same to single-rail encoding.

C. Structure of the proposed asynchronous circuit

Fig.6 shows the structure of dual-rail/single-rail hybrid logic design. The solid arrow represents the constructed critical path which is a dual-rail datapath, the dotted arrow represents non-critical paths which are single-rail datapaths, and the dashed arrow is a dual-rail datapath which is the output of single-rail to dual-rail converter. A static NOR gate is connected to the critical path to observe the bit-done signal for the entire datapaths at each pipeline stage. The output of NOR gate is connected to the precharge/evaluation control input, $pc$, of the previous stage with a drive buffer.

The protocol of the proposed asynchronous circuit is quite simple. Stage$(N)$ is precharged when Stage$(N + 1)$ finishes evaluation. Stage$(N)$ evaluates when Stage$(N + 1)$ finishes its reset, or precharge. In Fig.6, if we observe a single data flow through an initially empty pipeline which every pipeline stage is in evaluation phase, the complete cycle of events is as follows,

- Stage1 evaluates and data flows to Stage2.
- Stage2 evaluates and data flows to Stage3. Stage2’s completion detector detects completion of evaluation and sends a precharge signal to Stage1.
- Stage1 precharges and Stage3 evaluates. Stage3’s completion detector detects completion of evaluation and sends a precharge signal to Stage2.
- Stage2 precharges. Stage2’s completion detector detects the completion of precharge and sends an evaluation signal (enable signal) to Stage1. The evaluation signal enables Stage1 to evaluate new data once again.

In order to get a stable critical signal transition, we change the gate that has the largest number of inputs to the SLG in each pipeline stage, which is shown as the gray block in Fig.6. The signal transition on the output of the SLG will always be the slowest one on the outputs of the pipeline stage. The reasons are as follows:

- The SLG has the largest inputs compared with other logic gates in each pipeline stage.
- There is only one path activated and the path always has constant stack at the sequential position in the SLG.
- Other logic gates have smaller, or same number of inputs and might have parallel paths activated together in the pull-down network.

The above analysis of the critical signal transition is based on an assumption that all logic gates in each pipeline stage evaluate at the same time. It is very difficult to satisfy this assumption in practical design. However, if logic gates in each pipeline stage evaluate at different time, the signal transition on the output of SLG might not be the critical one any more. In order to avoid this problem, SLGLs are used to make sure that every SLG or SLGL in each pipeline stage is the last
TABLE II
COMPARISON RESULTS BETWEEN DIFFERENT DESIGN METHODS

<table>
<thead>
<tr>
<th>Function</th>
<th>Proposed Design</th>
<th>Synchronous Pipeline</th>
<th>Bundled-data Async-Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Style</td>
<td>Dynamic Logic</td>
<td>Static Logic</td>
<td></td>
</tr>
<tr>
<td>Transistor Count</td>
<td>7375</td>
<td>9010</td>
<td>15236</td>
</tr>
<tr>
<td>Storage Element</td>
<td>0</td>
<td>278 (flip-flop)</td>
<td>791 (latch)</td>
</tr>
<tr>
<td>Latency</td>
<td>0.44ns</td>
<td>1.3ns</td>
<td>1.4ns</td>
</tr>
<tr>
<td>Throughput (data-set/s)</td>
<td>4.35G</td>
<td>4.17G</td>
<td>2.86G</td>
</tr>
</tbody>
</table>

Fig. 8. Diagrams of multiplier: (a) Dual-rail/single-rail hybrid logic design; (b) Synchronous design.

one to start evaluation. The solution is shown, as examples, in stage 3 and stage 4 in Fig. 6. If two SLGs in adjacent pipeline stages are not connected with each other, we cannot guarantee that the output of the SLG in the subsequent stage has the critical signal transition because it might evaluate earlier with quick arrived inputs than other gates. In this situation, SLGL would replace the SLG in the subsequent stage. For example, the SLGs in stage 2 and stage 3 are not connected with each other. Then, the SLG in stage 3 is changed to SLGL and the output of SLG in stage 2 connects to the enable port of SLGL. Because SLGL synchronizes its inputs, it cannot start evaluation without the presents of the critical signal transition from the previous pipeline stage. As a result, SLGL in stage 3 will be the last gate start evaluation, which guarantees the signal transition on the output of SLGL is the critical one. The same structure is also shown from stage 3 to stage 4.

III. EVA LAT I O N

In order to evaluate the proposed design, array style multiplier is chosen as a test case. 8 x 8 multipliers are respectively designed using the proposed asynchronous circuit, bundled-data asynchronous circuit and classical synchronous circuit. These circuits are simulated by HSPICE in a 65nm technology. Table II shows comparison results between different design methods. Fig. 8 shows diagrams of multiplier. Fig. 9 shows the performance of power consumption between the proposed design and other design methods. Because dynamic logic in proposed design provides an implicit latch function, intermediate storage elements are all removed which respectively saves 278 flip-flops and 791 latches compared to synchronous design and bundled-data asynchronous design. At the same time, it respectively improves the forward latency by 66.2% and by 68.6%, which is just 0.44ns. With the simplified handshake circuit, the proposed multiplier is work as fast as synchronous design (fine-grain pipelined). Moreover, it greatly improves the power consumption without the overhead of intermediate storage elements, clock tree and dual-rail implementation. Fig. 9 shows that the power consumption is reduced by 35% when the circuit works at peak speed. When the work loads are 50%, this number is up to 52.5%.

IV. CONCLUSION

This paper introduced an asynchronous circuit design method. The overhead of handshake logic circuit and function block circuit is reduced, which improves the performance of speed and power consumption. The evaluation of multiplier shows that the proposed design works at higher speed with better performance of power consumption than classical synchronous pipeline design.

ACKNOWLEDGMENT

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with STARC, e-Shuttle, Inc., Fujitsu Ltd., Cadence Design Systems Inc., Synopsys Inc. and Mentor Graphics, Inc.

REFERENCES