IPSJ Transactions on System LSI Design Methodology Vol. 2 103–113 (Feb. 2009)

Regular Paper

An Asynchronous IEEE-754-standard Single-precision Floating-point Divider for FPGA

Masayuki Hiromoto,[†1] Hiroyuki Ochi[†1]
and Yukihiro Nakamura[†2]

Synchronous design methodology is widely used for today's digital circuits. However, it is difficult to reuse a highly-optimized synchronous module for a specific clock frequency to other systems with different global clocks, because logic depth between FFs should be tailored for the clock frequency. In this paper, we focus on *asynchronous* design, in which each module works at its best performance, and apply it to an IEEE-754-standard single-precision floating-point divider. Our divider is ready to be built into a system with arbitrary clock frequency and achieves its peak performance and area- and power-efficiency. This paper also reports an implementation result and performance evaluation of the proposed divider on a Xilinx Virtex-4 FPGA. The evaluation results show that our divider achieves smaller area and lower power consumption than the synchronous dividers with comparable throughput.

1. Introduction

Most digital circuits today are designed as synchronous sequential circuits, because synchronous systems are easy to be specified, and design methodology for synchronous circuits is widely studied. In fact, various convenient CAD tools for developing synchronous systems are available these days.

However, the synchronous system has several problems [1]. One is large power consumption caused by many Flip Flops (FFs) connected to a global clock tree which must be distributed all over the chip with high frequency. This can represent even 45% of total power consumption in some cases [2]. Increasing clock skew, which can be seen in today's miniaturized process technologies, is another problem, leading to degradation in circuit speed and yield. For these problems, some solutions have been proposed, such as a circuit technology that absorbs process variation after fabrication [3] and its design methodology [4], or an approach of globally-asynchronous locally-synchronous design [5].

Yet another problem is design reusability. To develop the best possible synchronous circuit in terms of performance, area, and power consumption for a given process technology, it is indispensable to optimize it depending on the target clock frequency. It includes not only logic-level optimization, but also architecture-level optimization. To achieve the best result, logic levels between FFs should be adjusted to fit the clock period. Assume that there is a module which has been developed for a particular system. To reuse it for another system where available clock frequency is different from the original system, we must re-optimize (or even re-design) the module to fit the given clock period. For example, it is clear that a module optimized for a 100 MHz system is not reused for a 150 MHz system since it does not work properly (**Fig. 1**). It must be also noted that a module optimized for a 150 MHz system is not suitable to be reused in 100 MHz system since its throughput and area would be improved by reducing excessive stage registers. To modify the position of stage registers, the design should be modified in RTL description phase, but considerable iteration between RTL description phase and post-place timing analysis phase is needed to optimize throughput. Ochi, et al. [6] developed a library of IEEE-754-standard single-precision floating-point dividers which consists of designs of several target frequencies for a specific process. It is beneficial to library users if there is a design tailored for the desired clock frequency, but very large man-hours is required to develop such libraries.

To resolve the above problems of synchronous systems, we introduce an asynchronous scheme. Using an asynchronous scheme, we can expect following enhancements:

(1) Low power consumption.
(2) Clock-skew-free design.
(3) Applicability for systems of any global clock frequency.

(1) is because a global clock tree which wastes power is removed. For this advantage, some LSIs for portable devices partly adopt asynchronous circuits to meet the strong demand for low power consumption. (2) is also a benefit of

†1 Graduate School of Informatics, Kyoto University
†2 Research Organization of Science and Engineering, Ritsumeikan University

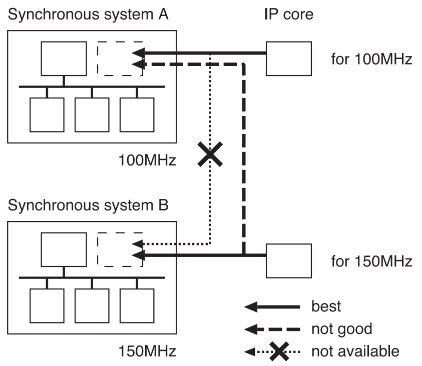


Fig. 1 Reusability of synchronous IP cores.

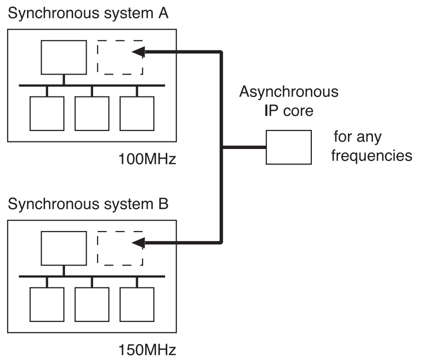


Fig. 2 Reusability of asynchronous IP cores.

clockless system. This suppresses long wires for a widely spread global clock tree which causes clock skew. (3) is because each module may work independently of a global clock, and inter-module interface is also realized by handshaking without synchronization using a global clock. Unlike synchronous IP cores, asynchronous IP cores can be built into systems of different clock frequencies as shown in **Fig. 2**. We call this portability as 'reusability' in this paper. Moreover, each module can take its optimal circuit to achieve the best performance.

In this paper, we mainly focus on (3) above, aiming to provide excellent IP cores of operation units independent of a global clock. High performance and good reusability of asynchronous IP cores have been proven by some previous works[7)–9)], which present large asynchronous processor cores. There are also contributions on asynchronous arithmetic functions targeted to ASICs, including dividers[10),11)] and multipliers[12),13)]. However, to our best knowledge, there is no contribution on asynchronous arithmetic IP cores designed for commercial FPGAs. FPGAs are very useful for rapid prototyping of digital systems, and there is a strong demand on high-performance and portable IP cores for arithmetic functions and other basic elements useful for developing custom hardware engines on FPGAs.

Therefore, as a first step of developing an asynchronous IP core library for FPGAs, here we pick up IEEE-754-standard single-precision floating-point dividers by Ochi, et al.[6)] and develop an asynchronous version based on them. Our proposed divider has the following features:

- It is tuned to achieve the best performance on FPGAs.
- It does not need dedicated external clock oscillator or PLL.
- It adopts asynchronous interfaces to be built into systems using an arbitrary clock frequency.

This paper is organized as follows. In Section 2, asynchronous systems and floating-point division are reviewed. In Section 3, the proposed asynchronous floating-point divider is introduced. Section 4 shows experimental results, and Section 5 provides conclusions.

2. Preliminaries

2.1 Asynchronous Systems

In this paper, a synchronous system is defined as a digital system whose circuits are all synchronized with unique global clock, and an asynchronous system is defined as one which has no global clock.

In synchronous systems, all registers transfer data simultaneously with a single global clock as shown in **Fig. 3**. On the other hand, in asynchronous systems, each module needs to determine timing to transfer data from/to each other because there is no global timing signal. There are two major methods for asynchronous data communication: a bundled-data method, which uses another wire of timing signal in addition to data bus, and a dual-rail encoded

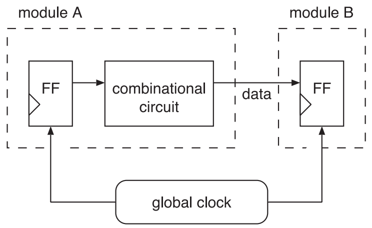


Fig. 3 Data transfer in synchronous system.

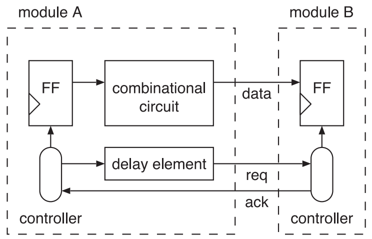


Fig. 4 Data transfer in asynchronous system.

data method, which uses a pair of 1-bit wires to distinguish “0”, “1”, and “invalid” states [1),14)]. For circuit efficiency, we adopt the bundled-data method for single-precision floating-point dividers in which long-word data are mainly communicated, since the bundled-data method requires less additional wires than the dual-rail encoded data method.

The handshake protocol with a bundled-data method is illustrated in **Fig. 4**. The module A sends a request signal (req) accompanied with data, and when the module B receives the req signal, the data from module A is stored to a register and an acknowledge signal (ack) is sent back to the module A. A delay element is introduced so that req signal arrives at the module B after the data.

2.2 Single-precision Floating-point Division

2.2.1 Arithmetic of Single-precision Floating-point

In this paper we use IEEE-754-standard single-precision floating-point format [15)]. It consists of a 1-bit field s for sign, an 8-bit field e for exponent, and a 23-bit field f for mantissa. In a most typical case ($0 < e < 255$), it represents a real number as follows:

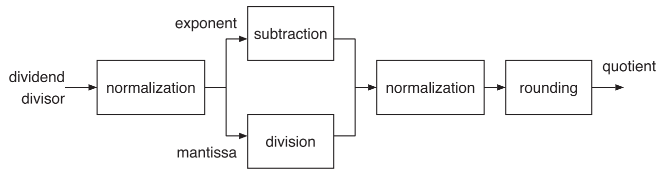


Fig. 5 A block diagram for floating-point division.

$$R = (-1)^s \times (1.f) \times 2^{(e-127)}. \tag{1}$$

In addition to the above case whose absolute value ranges widely from 2^{-126} to $(1-2^{-24}) \times 2^{128}$, the format is also able to represent exceptional values, including zero, infinity, NaN (Not a Number), and denormalized numbers.

2.2.2 Procedure for Single-precision Floating-point Division

A block diagram of floating-point division is shown in **Fig. 5**. First, given divisor and/or dividend are normalized if they are denormalized numbers. Next, division for mantissa parts and subtraction for exponent parts are performed to generate a quotient. Finally, the results are again normalized and rounded. In addition to the operations above, sign-bit generator for the quotient and a handler for various exceptions are also required.

2.3 Algorithm for Division of Mantissa

This part describes an algorithm for integer division that is required for calculation of mantissa parts.

There are two major types of division algorithms, digit-recurrence algorithms, such as restoring method and iterative algorithms, such as Newton's iteration method. In this paper we take the restoring division method in digit-recurrence algorithms since it requires less circuit area than the other methods. Let X, Y, and Q be a dividend, a divisor, and the quotient. Let

$$Q = \sum_{j=0}^{n-1} q_j 2^{-j}, \tag{2}$$

where n is the number of digits. Let R_{j+1} be the intermediate remainder after q_j is determined.

When $j = 0$, if $X \geq Y$, then let $q_0 = 1$ and $R_1 = X - Y$, otherwise, let $q_0 = 0$

```
X=01111       01111    X
Y=01011     + 10101   -Y
-Y=10101     100100 ··· q0=1 → R1=X-Y
              01000    2R1
            + 10101   -Y
              11101 ··· q1=0 → R2=2R1
              10000    2R2
            + 10101   -Y
             100101 ··· q2=1 → R3=2R2-Y
```

Fig. 6 An example of restoring division algorithm.

and $R_1 = X$. When $j \geq 1$, repeat the following operation: if $2R_j - Y \geq 0$, then let $q_j = 1$ and $R_{j+1} = 2R_j - Y$, otherwise, let $q_j = 0$ and $R_{j+1} = 2R_j$. A calculation example is shown in **Fig. 6**, where the intermediate remainders are surrounded with dotted circles and the sign bits are with dotted boxes. Let $Y = 01011$ and $X = 01111$. A negative for Y is represented as $-Y = 10101$ with two's complement. First, since the sign bit of the result for $X - Y$ is 0, set 1 to q_0 and let $R_1 = X - Y$. Next since the sign bit of the result for $2R_1 - Y$ is 1, set 0 to q_1 and let $R_2 = 2R_1$. The subsequent process is similar to above.

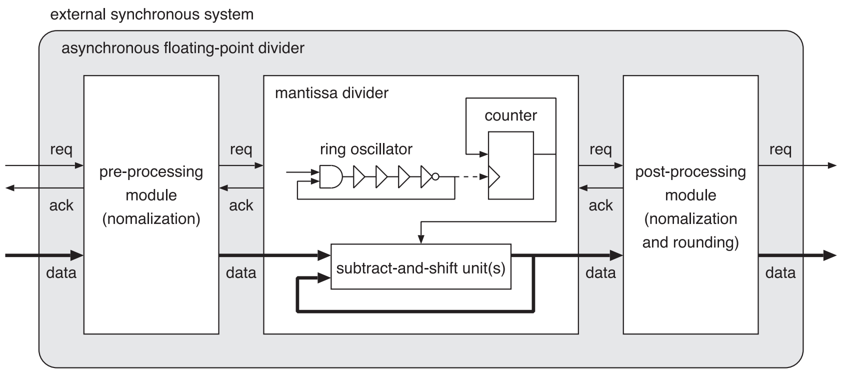
Hardware resources needed to realize restoring division circuit are mainly a subtracter for calculation of $2R_j - Y$, 2-input multiplexers to select values for R_j, and registers to hold intermediate data.

3. Asynchronous Design of Single-precision Floating-point Divider

3.1 Architecture Overview

This section introduces the proposed asynchronous floating-point divider based on the asynchronous systems and restoring divider described in the previous section.

When the floating-point divider as shown in Fig. 5 is implemented in hardware, the mantissa divider requires the largest portion of circuit area and calculation time. We focus on this mantissa divider and propose a smaller and more energy efficient module by using asynchronous scheme; in the concrete, we let the mantissa divider work independently from an external global clock but let be driven by an internal local clock which is the most suitable for the module. The other pre-/post-processing modules for normalization and rounding are connected to



external synchronous system
asynchronous floating-point divider
req
ack
data
pre-processing module (nomalization)
mantissa divider
counter
ring oscillator
subtract-and-shift unit(s)
post-processing module (nomalization and rounding)

Fig. 7 An overview of the proposed asynchronous floating-point divider.

each other with asynchronous bundled data method. This architecture realizes fast processing of a mantissa divider independently of the global clock and high reusability to be built into synchronous systems. An overview of the proposed asynchronous floating-point divider is shown in **Fig. 7**. The module can be built into external asynchronous or synchronous systems since the interfaces are also implemented using asynchronous handshakes. Note that the output interface does not implement a full handshake protocol in order to keep compatibility with synchronous dividers developed by Ochi, et al.[6]. The detail is described in the last part of Section 3.3.

The module for mantissa division consists of a local clock generator, a counter which counts 24 times for digit-recurrence division, and subtract-and-shift units. The mantissa divider in Fig. 7 performs calculation as follows: First, the mantissa divider receives a request signal from pre-processing module and starts a ring oscillator to generate local clock. Then the clock is delivered to the counter and subtract-and-shift units, and digit-recurrence division is performed for predefined repeat count. At the same time, the mantissa divider sends an acknowledge signal to the pre-processing module to let it perform pre-processing for the next data. When finishing mantissa division, the module sends result data with a request signal to the next post-processing module and stops the ring oscillator. The post-processing module sends an acknowledge signal to the mantissa divider after receiving the data, and then perform post-processing to calculate final results.

Table 1 Experimental results on mantissa dividers of N-stage subtract-and-shift units.

N	1	2	3	4	6	8	12	24
Local clock frequency [MHz]	160.6	91.6	68.9	50.9	34.2	27.9	18.6	8.66
Clock cycles for a data	25	13	9	7	5	4	3	2
Circuit area [Slices]	146	129	165	179	229	303	416	839
Power [mW]	409	446	469	494	571	694	846	1511
Energy for a data [pJ]	64.7	64.3	62.3	69.0	84.8	199.9	138.1	351.0
Throughput [MFLOPS]	6.42	7.04	7.65	7.27	6.83	6.97	6.20	4.33
Throughput by area [MFLOPS/Slices]	0.0440	0.0546	0.0464	0.0428	0.0298	0.0230	0.0149	0.0052

3.2 Mantissa Divider

This part discusses the mantissa divider, which is the largest part of proposed asynchronous floating-point divider in Fig. 7 including implementation and performance evaluation.

3.2.1 Design Exploration of Mantissa Divider

In the restoring division algorithm, subtract-and-shift operations must be performed with repeat count equal to the desired number of digits of the quotient, e.g., 24 if single-precision floating-point format is intended. If N-stage subtract-and-shift units are used at the cost of circuit area so that N subtract-and-shift operations can be done per iteration, number of iteration will be reduced to $24/N$ times. Thus there is a trade-off relation between circuit area and throughput. In order to explore an optimal point of the design, we implemented and evaluated eight versions of mantissa dividers with N of 1, 2, 3, 4, 6, 8, 12, and 24.

The mantissa divider is driven by a local clock generated by a ring oscillator inside the module. For best performance, the ring oscillator is designed to generate the highest frequency of the local clock which can drive N-stage subtract-and-shift units properly.

3.2.2 Implementation Results

The eight versions of mantissa dividers were designed and described in HDL. Simulation, synthesis, layout and routing were performed to estimate their performance. We used Mentor Graphics ModelSim SE 6.0c for simulation and Xilinx ISE 8.1i for synthesis and implementation, and assumed the target device as Xilinx Virtex-II FPGA (XC2V1000). The power consumption was estimated by gate-level simulation using post place-and-route netlists and their timing information and analysis of the result with a power analyzer bundled with ISE.

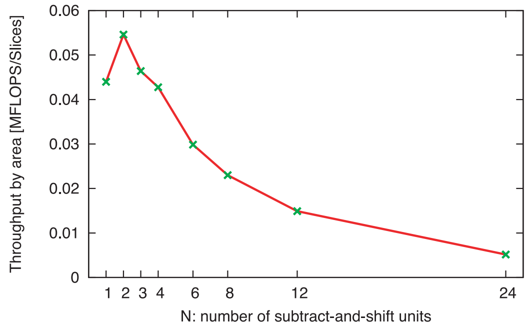


Fig. 8 Throughput by area of mantissa dividers of N-stage subtract-and-shift units.

Table 1 summarizes the experimental results on mantissa dividers of N-stage subtract-and-shift units. The divider with larger N requires a lower number of iterations but longer clock periods of local clock. Therefore, the throughput of each design is almost identical to each other.

The circuit area increases as N increases except $N = 1$. The design with $N = 1$ consumes a large area because it requires high frequency of the local clock and it leads to an increase of circuit area in order to enhance the driving power of the ring oscillator. Therefore, the design of $N = 2$ achieves the best performance and is a well-balanced design in terms of throughput by area, which is shown in **Fig. 8**.

On the other hand, energy consumption per division is relatively small for $N = 1,\ 2,\ 3$ designs and becomes larger as N increases. This seems because a design with large N has combinational circuits with large depth which increases unnecessary switching activities.

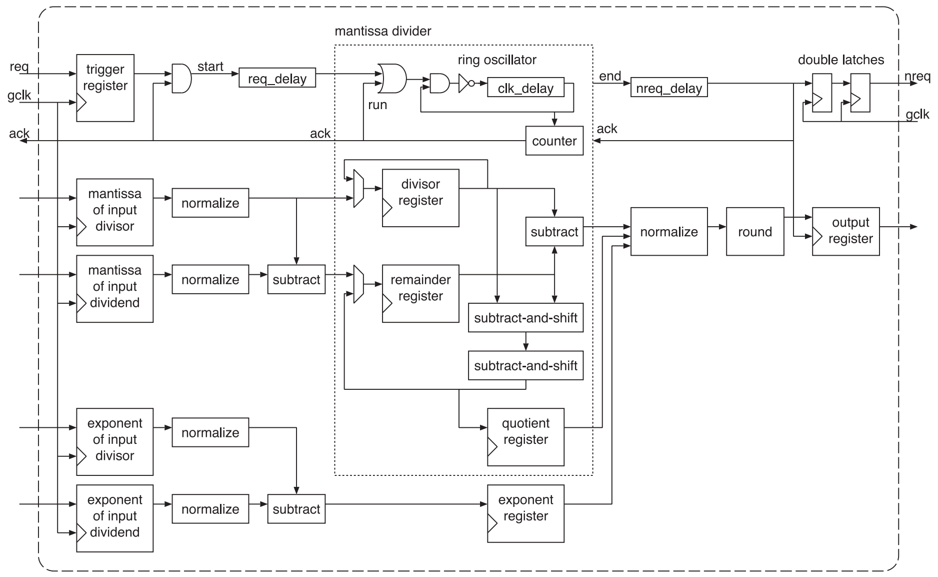


Fig. 9 Block diagram of proposed asynchronous floating-point divider.

Although the design with $N = 3$ achieve an energy consumption a little less than $N = 2$, we adopt $N = 2$ design that shows the highest throughput by area to mantissa divider of the proposed asynchronous divider.

3.3 Detailed Architecture

Figure 9 shows a block diagram of proposed asynchronous floating-point divider. This divider can be built into synchronous systems with any clock frequencies.

In Fig. 9, counter register, divisor register, remainder register, and quotient register are driven by a local clock generated by a ring oscillator inside the module, while input/output registers for a divisor and a dividend are synchronized with the external global clock. The area surrounded by dotted line is a mantissa divider described in the previous part, and it is connected to pre-/post-processing modules with bundled data method.

The proposed divider performs calculation as follows: First, given dividend and divisor are stored to registers synchronized by the global clock. Next they are normalized in the pre-processing module and the mantissa part is passed to the mantissa divider while the exponent part is sent to the subtracter. After that, the ring oscillator in the mantissa divider starts oscillation. Since we select $N = 2$ design, subtract-and-shift operations are iterated 12 times to calculate quotient of mantissas. When the division is completed, the mantissa module immediately stops its ring oscillator, passes the result to the post-processing module, and requests any subsequent data from the pre-processing module. Finally, the results are normalized and rounded in the post-processing module.

The final results are passed to an external synchronous circuit with a request signal (`nreq` in Fig. 9). To avoid metastability of the request signal, double latches are inserted in the request signal path before it is sent to the external circuit. Unlike usual interfaces with handshake methods, this divider does not receive an acknowledge signal from the external circuit. This is because the proposed divider is designed to be replaceable with the synchronous dividers developed by Ochi, et al. [6], which have the same output interfaces without acknowledge signals. Due to this specification, the external circuit is required to latch data at the output register after `nreq` is asserted until the output register is overwritten by the next result.

3.4 Design Flow

As shown in Fig. 9, this asynchronous design requires some delay elements for a ring oscillator (`clk_delay`), a pre-processing module (`req_delay`), and a post-processing module (`nreq_delay`). Since a target device is an FPGA in this paper, the delay elements are realized with cascades of Look Up Tables (LUTs), which are fundamental elements of an FPGA. Delay time of the delay element can be adjusted by changing a number of cascaded LUTs.

Figure 10 is a design flow to explore the number of cascaded LUTs in each delay element. First an initial number of LUTs in each delay element is set and RTL codes of an asynchronous divider including the delay elements are generated. Since it is difficult to obtain delay elements of desired delay time due to uncertainness of place-and-route results by synthesis tool, we use Relative LOCation (RLOC) constraints to arrange the LUTs in a line and make the delay time of the LUT cascade linear with the number of LUTs. Next the RTL codes with RLOC constraints are synthesized and mapped by a CAD tool to generate a post-place-

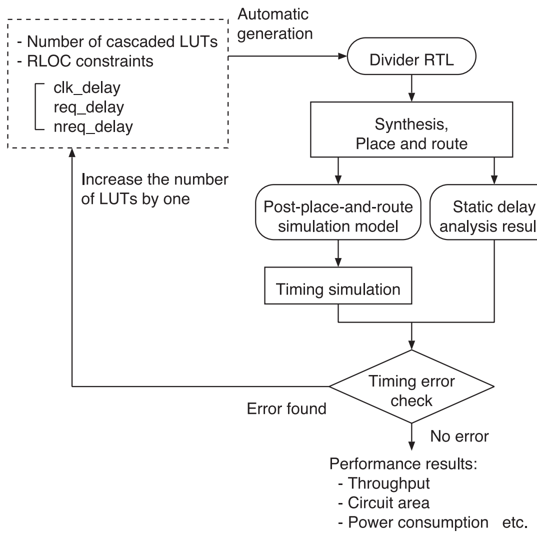


Fig. 10 A design flow of the proposed asynchronous divider.

and-route netlist for timing simulation. Then the timing simulation is performed to verify the calculation results against many test vectors. Through the simulation, the delay time of each delay element is also verified that it is longer than the data arrival time of the corresponding data path, which is obtained by static delay analysis. If any timing errors are found, the number of LUTs related to the timing errors is increased by one and the design flow (synthesis, place-and-route and simulation) is repeated until all errors are eliminated. Minimum number of necessary LUTs in each delay element is obtained by such repetition, in which the number of LUTs is increased by one from the initial number that is small enough to generate timing errors.

When we explore asynchronous circuits with minimum necessary delay elements according to the design flow above, plenty of man-hours are required to change the number of cascaded LUTs and repeat the flow manually. To design efficiently, we developed a design environment that automatically executes the design flow from RTL generation to timing verification just by assigning the number of cascaded LUTs in each delay elements. This design environment repeats the design flow to explore the minimum necessary delay elements and finally reports performance of the obtained design such as circuit area, power consumption and throughput [⋆1].

4. Implementation and Evaluation on FPGA

This section describes implementation and evaluation results of the asynchronous divider proposed in Section 3. In addition to the performance evaluation, implementation on an FPGA board is also described.

4.1 Evaluation of Proposed Floating-point Divider

The proposed asynchronous floating-point divider is implemented on an FPGA to evaluate its circuit area, power consumption and throughput, and to compare with synchronous floating-point dividers developed by Ochi, et al. [6]. The target device is Xilinx Virtex-4 FPGA (XC4VFX12), and we use Mentor Graphics ModelSim SE 6.2e for simulation and Xilinx ISE 9.2i for synthesis and implementation.

Since the synchronous dividers developed by Ochi, et al. [6] are designed for ASICs with 0.35 μm process technology and optimized for synchronous systems with specific clock frequencies, 50, 75, 100, 125, and 150 MHz, they may not be optimal designs for FPGA implementation. For fair comparison, we redesign them to be suitable for FPGAs and prepare five versions of synchronous dividers optimal for clock frequencies of 40, 60, 75, 110, 130 MHz by adjusting number of stages in the subtract-and-shift units. The proposed asynchronous divider is evaluated with these five external clock frequencies on the assumption that it is built into synchronous systems which run at the same frequencies.

Table 2 and **Table 3** show the results of the comparison between the proposed asynchronous divider and the synchronous ones with different global clocks.

4.1.1 Circuit Area

Figure 11 shows circuit area of synchronous and asynchronous dividers. The proposed asynchronous divider achieves smaller circuit area than the synchronous

⋆1 Although the designed divider can be used in a system with any global clock frequency, it can be used only for the specific FPGA architecture and speed grade, because dedicated delay elements are used in the bundled-data method. However, we can obtain proper delay elements for a different FPGA architecture and speed grade by applying design flow as shown in Fig. 10.

Table 2 Results for performance evaluation of synchronous dividers with different global clocks.

Global clock frequency [MHz]	40	60	75	110	130
Number of subtract-and-shift units, N	6	4	3	2	1
Circuit area [Slices]	629	550	554	516	546
Power [mW]	314.1	308.3	305.5	311.9	304.8
Dynamic [mW]	60.6	54.9	52.0	58.5	51.4
Quiescent [mW]	253.5	253.4	253.4	253.5	253.4
Dynamic energy for a data [pJ]	7.59	6.42	6.43	7.22	10.69
Throughput [MFLOPS]	8.00	8.57	8.33	7.86	4.82
Throughput by area [MFLOPS/Slices]	0.0127	0.0156	0.0150	0.0152	0.0088

Table 3 Results for performance evaluation of the proposed asynchronous divider.

Global clock frequency [MHz]	40	60	75	110	130
Number of subtract-and-shift units, N	2 (with 115 MHz local clock)				
Circuit area [Slices]	409				
Power [mW]	289.2	291.2	293.7	296.3	298.3
Dynamic [mW]	36.0	38.0	39.5	43.0	45.0
Quiescent [mW]	253.2	253.2	253.2	253.3	253.3
Dynamic energy for a data [pJ]	4.45	4.70	4.87	5.31	5.56
Throughput [MFLOPS]	8.10	8.10	8.10	8.10	8.10
Throughput by area [MFLOPS/Slices]	0.0198	0.0198	0.0198	0.0198	0.0198

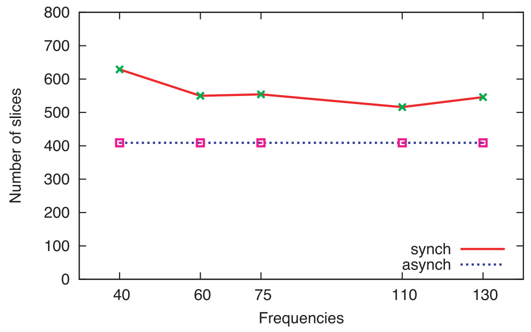


Fig. 11 Circuit area of different floating-point dividers.

ones for any clock frequencies. This is mainly because the asynchronous design does not require register insertion to meet timing constrains of a global clock. Since the synchronous dividers are designed to achieve high throughput by register balancing, the dividers for high-speed clocks (110 MHz and 130 MHz) contains more FFs than the asynchronous design. For example, number of FFs in the 110 MHz synchronous divider is 198 while in the asynchronous one is 135. Although the asynchronous design requires additional circuit area for delay elements, the proposed divider achieves smaller circuit area because the register insertion of the synchronous designs affects the whole area more strongly than area increase by the delay elements of the asynchronous design.

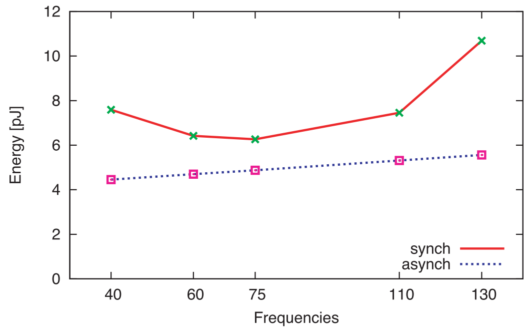


Fig. 12 Dynamic energy consumption of different floating-point dividers.

While circuit area of the synchronous dividers varies according to the target frequency because of different number of stages in subtract-and-shift units, area of the asynchronous one is constant since all global clock frequencies are covered by only one design. This shows that the proposed asynchronous divider achieves small and constant circuit area.

4.1.2 Power Consumption

As shown in Table 2 and Table 3, quiescent power makes up the greater part of chip-level power consumption in our experiments. In fact, an XC4VFX12 device has 5,472 slices and thus the resource utilization of each divider is only about 10%. Therefore, to clarify the characteristics of power consumption of dividers, we calculate *dynamic* energy consumption per operation as shown in Table 2, Table 3, and **Fig. 12**.

As in Fig. 12, energy consumption in synchronous designs differs according to the frequency of global clock since it is strongly affected by clock frequency and circuit area. The energy consumption of the asynchronous divider increases linearly corresponding to the clock frequency since the divider contains some

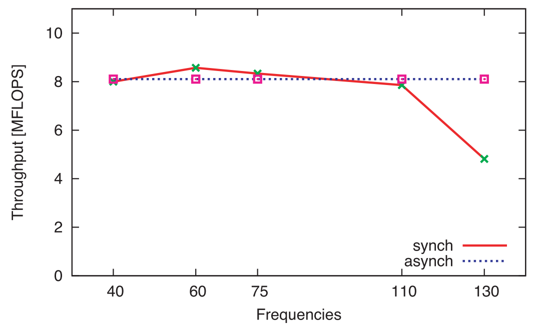


Fig. 13 Throughput of different floating-point dividers.

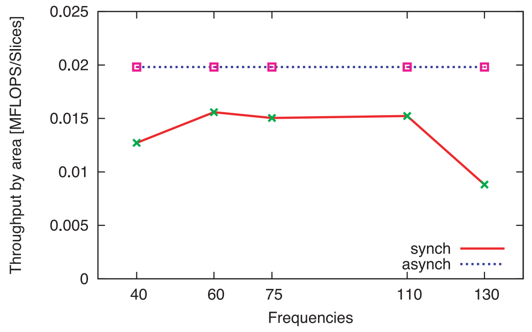


Fig. 14 Throughput by area of floating-point dividers.

interfaces driven by the global clock. However, the asynchronous divider achieves lower energy consumption than that of the synchronous ones. This shows that asynchronous circuits consume lower energy not only because the circuit area is smaller than synchronous ones, but also because FFs and wires are driven only when data is actually transferred there, while synchronous circuits always waste energy by FFs driven by a global clock.

4.1.3 Throughput

Figure 13 shows throughput of each design. Similar to the energy consumption, the asynchronous divider realizes almost constant performance independently of external clock frequency. The synchronous designs, however, show variation of performance according to clock frequency (especially at 130 MHz). The reason is that in synchronous systems it is difficult to design optimal circuits suitable for a given clock frequency since number of stages of subtract-and-shift unit between FFs are discrete and they do not always fit the clock frequency.

The throughput of the proposed divider is comparable or higher than the synchronous designs that are optimized for each clock frequency. This shows an asynchronous circuit achieves high performance under any frequencies without modification because each data-path is driven with its minimum delay time independent of the external clock, while synchronous circuits require optimization according to the clock frequencies.

Throughput by area is calculated to evaluate the area efficiency of dividers. As shown in **Fig. 14**, the asynchronous design is superior to synchronous ones. Therefore, the result shows the proposed asynchronous floating-point divider achieves not only good reusability but also higher area efficiency than the synchronous dividers.

4.2 Implementation on FPGA Board

The proposed asynchronous divider is implemented on an FPGA board with other synchronous peripheral circuits to test its function on an actual FPGA. We use Xilinx ML403 FPGA evaluation board, which mounts Virtex-4 (XC4VFX12), the same target device as in the experiments so far.

In order to test the proposed asynchronous IP embedded into synchronous systems, the proposed divider is implemented with synchronous circuits provided with the FPGA board. They include many controllers and interfaces to use I/Os on the board and PowerPC in the FPGA. We verify the proposed divider with millions of randomly-generated test-vectors by using driver software running on the PowerPC. As a result, the experiments show our asynchronous divider works properly and calculates correct answers while being part of a synchronous system.

5. Conclusion

In this paper, we discussed the benefit of asynchronous implementation in terms of IP reusability. As a case study, we developed an asynchronous single-precision floating-point divider IP which can be built into synchronous systems under arbitrary clock frequency with a single design. The proposed floating-point divider has interfaces to communicate data to the external system and its internal division module is driven by a local clock generated inside the module, which enables fast calculation independently of the external system. We designed and implemented the proposed divider on an FPGA to evaluate and compare performance with synchronous versions of dividers. The result shows that our divider achieves smaller circuit area and lower energy consumption than the synchronous ones with comparable throughput. In addition to the performance evaluation, the divider is implemented with other synchronous circuits on a real FPGA board. This shows our asynchronous IP works correctly in synchronous systems without any modifications for wide range of clock frequencies.

As future work, we are planning to develop a library of asynchronous IP cores in addition to the floating-point divider.

Acknowledgments This work is partly supported by Grants-in-Aid for Scientific Research (C) 18500036 from the Japan Society for the Promotion of Science (JSPS) and by a Global Center of Excellence (G-COE) program of the Ministry of Education, Culture, Sports, Science and Technology of Japan. This work is also partly supported by the VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Mentor Graphics, Inc.

References

1) Sparsø, J. and Furber, S.B.: *Principles of Asynchronous Circuit Design: A Systems Perspective*, Kluwer Academic (2001).
2) Sakurai, T. and Kuroda, T.: Tutorial on Low-Power Design Methodology, *Proc. Synthesis and System Integration of Mixed Technologies* (*SASIMI*), pp.3–10 (1996).
3) Kawanami, T., Hioki, M., Nagase, H., Tsutsumi, T., Nakagawa, T., Sekigawa, T. and Koike, H.: Preliminary Evaluation of Flex Power FPGA: A Power Reconfigurable Architecture with Fine Granularity, *IEICE Trans. Inf. Syst.*, Vol.E87-D, No.8, pp.2004–2010 (2004).
4) Katsuki, K., Kotani, M., Kobayashi, K. and Onodera, H.: Extracting a Random Component of Variation from Measurement Results of a 90 nm LUT Array, *Proc. Synthesis and System Integration of Mixed Technologies* (*SASIMI*), pp.197–200 (2006).
5) Chapiro, D.M.: Globally-Asynchronous Locally-Synchronous Systems, PhD Thesis, Stanford University (1984).
6) Ochi, H., Suzuki, T., Matsunaga, S., Kawano, Y. and Tsuda, T.: Development of an IP Library of IEEE-754-Standard Single-Precision Floating-Point Dividers, *IEICE Trans. Fundamentals*, Vol.E86-A, No.12, pp.3020–3027 (2003).
7) Martin, A., Lines, A., Manohar, R., Nystrom, M., Penzes, P., Southworth, R. and Cummings, U.: The Design of an Asynchronous MIPS R3000 Microprocessor, *Proc. Seventeenth Conference on Advanced Research in VLSI*, pp.164–181 (1997).
8) Garside, J., Bainbridge, W., Bardsley, A., Clark, D., Edwards, D., Furber, S., Liu, J., Lloyd, D., Mohammadi, S., Pepper, J., et al.: AMULET3i-an Asynchronous System-on-Chip, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp.162–175 (2000).
9) Chong, K., Gwee, B. and Chang, J.: Energy-Efficient Synchronous-Logic and Asynchronous-Logic FFT/IFFT Processors, *IEEE Journal of Solid-State Circuits*, Vol.42, No.9, pp.2034–2045 (2007).
10) Matsubara, G. and Ide, N.: A Low Power Zero-Overhead Self-Timed Division and Square Root Unit Combining a Single-Rail Static Circuit with a Dual-Rail Dynamic Circuit, *Proc. Third IEEE International Symposium on Asynchronous Circuits and Systems*, pp.198–209 (1997).
11) Cornetta, G. and Cortadella, J.: A Multi-Radix Approach to Asynchronous Division, *Proc. Seventh IEEE International Symposium on Asynchronous Circuits and Systems*, pp.25–34 (2001).
12) Kearney, D. and Bergmann, N.: Bundled Data Asynchronous Multipliers with Data Dependent Computation Times, *Proc. Third IEEE International Symposium on Asynchronous Circuits and Systems*, pp.186–197 (1997).
13) Hensley, J., Lastra, A. and Singh, M.: A Scalable Counterflow-Pipelined Asynchronous Radix-4 Booth Multiplier, *Proc. 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pp.128–137 (2005).
14) Myers, C.: *Asynchronous circuit design*, John Wiley & Sons (2001).
15) IEEE: IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985 (1985).

(Received May 23, 2008)
(Revised August 22, 2008)
(Accepted October 9, 2008)
(Released February 17, 2009)

(Recommended by Associate Editor: *Hiroshi Saito*)

Masayuki Hiromoto received his B.E. degree in Electrical and Electronic Engineering and M.S. degree in Communications and Computer Engineering from Kyoto University in 2006 and 2007, respectively. Presently, he is a doctoral student at the Department of Communications and Computer Engineering, Kyoto University. He is a JSPS research fellow and a student member of IEEE, IEICE, and IPSJ.

Hiroyuki Ochi received the B.E., M.E., and Ph.D. degrees in Engineering from Kyoto University in 1989, 1991, and 1994, respectively. In 1994, he joined Department of Computer Engineering, Hiroshima City University as an associate professor. Since 2004, he has been an associate professor of Department of Communications and Computer Engineering, Kyoto University. He is a member of IPSJ, IEICE, IEEE, and ACM.

Yukihiro Nakamura received his B.S., M.S. and Ph.D. degrees in Applied Mathematics and Physics from Kyoto University, in 1967, 1969 and 1995, respectively. From 1969 to 1996, he was with Electrical Communications Laboratories, NTT. In NTT he engaged in research and development of a general purpose large-scale computer "DIPS", the behavioral description language "SFL" and the High-Level Synthesis System "PARTHENON". In 1996, he joined Graduate School of Informatics, Kyoto University as a professor. Since 2007, he has been a professor of Research Organization of Science and Engineering, Ritsumeikan University and also the President of Advanced Scientific Technology and Management Research Institute "ASTEM RI". He received Best Paper Award of IPSJ, Okochi Memorial Technology Prize, Minister's Prize of the Science and Technology Agency and Achievement Award of IEICE in 1990, 1992, 1994 and 2000, respectively. He has played roles as a member, Asian representative or chair for ICCAD, EDAC, ASP-DAC, CODES+ISSS and so on. He is a fellow of IEEE and a member of IPSJ, IEICE, and ACM.