A Design Flow for Click-Based Asynchronous Circuits Design With Conventional EDA Tools

Hui Wu, Zhe Su, *Student Member, IEEE*, Jilin Zhang, Shaojun Wei, *Fellow, IEEE*, Zhihua Wang, *Fellow, IEEE*, and Hong Chen, *Senior Member, IEEE*

***Abstract*—The "event-driven" feature of asynchronous circuits enables the circuits to work when and where needed, making it a good alternative to design low-power circuits. However, asynchronous circuits are not widely adopted as a consequence of the lack of support by conventional EDA tools. In this article, we propose a novel design flow to implement the Click-based asynchronous bundled-data circuits efficiently down to mask layout with conventional EDA tools. To ensure timing correctness, we put forward an adaptive delay matching (ADM) method and perform accurate static timing analysis for the circuits. Compared with other asynchronous toolsets, the proposed design flow is more efficient and convenient to implement asynchronous circuits. An asynchronous convolution neural network accelerator is implemented in TSMC 180- and 65-nm CMOS process, respectively, to verify the proposed design flow. The silicon test results show that the asynchronous accelerator has 30% less power in the computing array than the synchronous one in the TSMC 65-nm CMOS process, and the energy efficiency of the asynchronous and synchronous accelerators are 1.539 TOPS/W and 1.37 TOPS/W, respectively. The energy efficiency of the asynchronous accelerator in the TSMC 180-nm CMOS process is 133 GOPS/W.**

***Index Terms*—Adaptive delay matching (ADM), asynchronous circuits, click-based bundled-data (BD) circuits, conventional EDA tools.**

I. Introduction

Asynchronous circuits have significant potential advantages in low power consumption and high performance [1], [2]. The "event-driven" feature of asynchronous circuits is much like the way our brain works so asynchronous circuits are widely adopted in neuromorphic chip design, such as TrueNorth [3] and Loihi [4] to achieve low power consumption. The applications in the Internet of Things and mobile platform also expect hardware to run in an energy-efficient manner which needs to adopt a low-power design methodology. Therefore, asynchronous circuits are a promising alternative to design low-power circuits. However, asynchronous circuits are not widely adopted as a consequence of the lack of support by conventional EDA tools, making it challenging to design asynchronous circuits efficiently for designers.

Manuscript received March 21, 2020; revised June 25, 2020 and September 15, 2020; accepted November 4, 2020. Date of publication November 16, 2020; date of current version October 20, 2021. This work was supported in part by the National Science and Technology Major Project from the Ministry of Science and Technology, China, under Grant 2018AAA0103100; in part by the National Natural Science Foundation of China under Grant 61674090; in part by the Beijing National Research Center for Information Science and Technology under Grant 042003266; and in part by the Beijing Engineering Research Center under Grant BG0149. This article was recommended by Associate Editor L. P. Carloni. *(Corresponding author: Hong Chen.)*

The authors are with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China, also with the Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China, and also with the Beijing Engineering Center of Technology and Research on Wireless Medical and Health System, Tsinghua University, Beijing 10084, China (e-mail: hongchen@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TCAD.2020.3038337

Compared with quasi delay insensitive (QDI) circuits, asynchronous bundled-data (BD) circuits cost less in terms of area and power. Many efforts have been made on the design flow for asynchronous BD circuits, including synthesis, physical implementation, and static timing analysis (STA). Petrify [5] synthesizes the asynchronous BD controller circuits based on the signal transition graph (STG). Balsa is put forward in [6] and is used to describe and synthesize asynchronous circuits based on syntax-directed compilation into communicating handshake circuits. Conventional EDA tools, such as Synopsys design compiler (DC) and IC compiler (ICC), are used in the design flows proposed in [7] and [8] to design asynchronous BD circuits, in which, timing paths are obtained by using command *get_timing_path* by specifying them one by one and delay matching is performed by using command *set_min_delay* in DC tools. The physical design of the asynchronous circuits is finished with ICC. In [9] and [10], a combination of clocks is adopted to describe every possible event propagation path in the design based on the STG, allowing the EDA tools to fully capture the relative timing constraints. This method can be used to perform STA on asynchronous BD circuits in the whole design flow. EDA tools such as Petrify which synthesizes only controllers and Balsa with the limitation due syntax-directed approach are less efficient. Design flows proposed in [7] and [8] need many manual steps to specify the timing paths one by one which are too complex to handle large-scale asynchronous BD circuits design. For large-scale asynchronous circuits, it is difficult to draw STG proposed in [9] and [10] and an efficient delay matching method to guarantee the asynchronous circuits fulfill the timing constraints should be provided.

In this article, we propose an efficient and convenient design flow to implement Click-based (i.e., Click element is adopted as control circuits) asynchronous BD circuits with conventional EDA tools. There are some well-known BD asynchronous pipeline styles: Micropipeline [11], Mousetrap [12], and Click [13]. Click element is chosen as the handshaking circuits in our design because its output pulse could be treated as a local clock, which makes it possible to adapt the Synopsys DC tool to synthesize the Click-based asynchronous BD circuits. In particular, we propose a method called adaptive delay matching (ADM) to ensure correct timing for Click-based asynchronous BD circuits during the synthesis process. The encounter digital implementation (EDI) tool is adopted to implement the physical design, and PrimeTime (PT) is used to perform STA of the circuits. Each step of the design flow is realized with the conventional EDA tools.

The main contributions of this article and the difference from other design flows are as follows.

1) The proposed design flow implements Click-based asynchronous BD circuits more efficiently than other tools with conventional EDA tools.
2) The proposed design flow synthesizes both synchronous and asynchronous circuits simultaneously in one synthesis process.

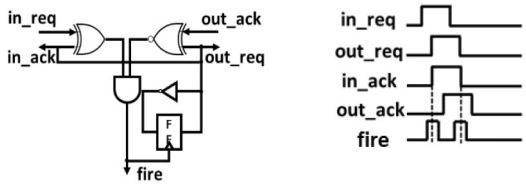

Fig. 1. Click elements [13]. (a) Schematic of Click. (b) Waveforms of Click.

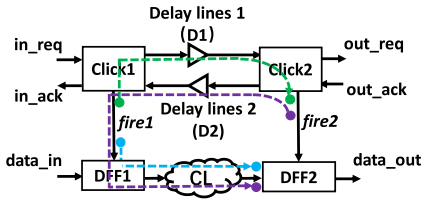

Fig. 2. Click-based BD asynchronous circuits.

3) Compared with the design flow proposed in [6] and [7], no extra manual steps are needed to specify the timing paths one by one and all timing paths can be captured with the clocks created in DC and PT.
4) Compared with the method proposed in [9] and [10], the method in this article is simpler to create clocks to obtain all the timing paths without using STG based on Click-based asynchronous circuits, and an ADM method is first proposed to ensure correct timing of the Click-based BD asynchronous circuits automatically.

The remainder of this article is organized as follows. Section II introduces the Click-based asynchronous BD circuits and timing constraints. In Section III, the design flow for Click-based asynchronous BD circuits is described in detail. In Section IV, CNN accelerators are implemented in silicon to verify the efficiency of the proposed methodology as a case study. Section V concludes this article.

II. Click-Based Bundled-Data Asynchronous Circuits

A. Click Element

The Click element is proposed by Peeters *et al.* [13], which adopts two-phase non-return-to-zero handshake communication protocol. The schematic of the Click element and its waveforms are shown in Fig. 1(a) and (b), respectively. A local pulse *fire* is generated to accept a new data item when the XOR gate checks that a new data item has arrived from the left, and the XNOR checks that the previous data item has been acknowledged from the right. Either the rising edge or the falling edge of *in_req* generates a pulse *fire* that is used as a local clock for data path. The transition of the *in_req* signal can be delivered to *out_req*, which becomes the *in_req* signal of the next stages. The width of pulse *fire* is decided by the delay lines (not shown in Fig. 1) in the circuits.

B. Asynchronous Bundled-Data Circuits and Timing Constraints

Fig. 2 illustrates a simple asynchronous BD circuit with Click elements. The circuit adopts the two-phase handshake protocol. When the data is ready, an transition of the request signal *in_req* is used to generate a local clock *fire1* to enable *DFF1* to capture the *data_in* and the *fire1* will trigger *Click2* to generate *fire2*. Then, a transition of the acknowledge signal *in_ack* represents the completion of this action. Both the rising and falling transition of the request signal indicate the validity of the data, and both the rising and falling transition of the acknowledge signal indicate the data has been captured [2]. The latency between *fire*1 and *fire*2 equals to the delay between the Click elements.

Obviously, the circuits in Fig. 2 should meet the requirement of timing constraint to ensure correct function. For example, in order to ensure *DFF2* obtain correct data from *CL*, the delay on the green path should be larger than that on the blue path. To specify conveniently, we use the following terminology: D flip-flop (DFF) setup time: t_{su}, DFF hold time: t_{hold}, the contamination delay of the DFF: $t_{c-q,cd}$, the maximum propagation delay of the DFF: t_{c-q}, the time interval from the rising edge *fire1* to *fire2:* t_{d1}, the time interval from the rising edge *fire2* to *fire1:* t_{d2}, the contamination delay of combinational logic: $t_{\text{logic},cd}$, and the maximum delay of combinational logic: t_{logic}.

As we know, the setup and hold timing constraint of *DFF1* and *DFF2* should be satisfied to ensure correct function of circuits (in Fig. 2). The setup timing constraint can be described as follows [14]:

$$t_{d1} > t_{c-q} + t_{\text{logic}} + t_{su}. \tag{1}$$

In order to satisfy the hold timing constraint, the delay of the timing path, which is marked in purple dotted line in Fig. 2, should be larger than t_{hold}. The hold timing constraint is as follows.

$$t_{\text{hold}} < t_{d2} + t_{c-q,cd} + t_{\text{logic},cd}. \tag{2}$$

To meet the requirement of setup timing constraint [shown in (1)], asynchronous BD circuits need to insert delay lines (D1 shown in Fig. 2) between the controllers which is called delay matching [2]. In most standard cell library, the minimum contamination delay of all DFFs, $t_{c-q,cd}$, is always lager than the maximum t_{hold} of the DFFs. Additionally, according to the structure of Click element, t_{d2} is obviously larger than t_{hold} even without D2 in Fig. 2. That is, no buffer is needed to avoid the hold violations in the circuits during the synthesis process.

III. Design Flow

We propose a design flow (illustrated in Fig. 3) to implement Click-based asynchronous BD circuits. The first step is *hardware description of Asyn.* BD *circuits*, in which the Click-based asynchronous BD circuits are described with Verilog codes. And the second step is *Synthesis with* ADM, in which DC is used to synthesize the Verilog codes to generate a netlist of the circuits. During the synthesis process, we put forward an ADM method. After synthesis, the physical design of the circuits including place and route (PnR) will be implemented with EDI. Then, design rule check (DRC) and layout versus schematic (LVS) check are carried on. The final step is STA with ADM which is an important step to check the timing of the circuits based on our ADM method. We describe all the steps in detail in the following parts.

A. Hardware Description of Click-Based Asynchronous BD Circuits

Fig. 4 shows Click-based asynchronous BD pipeline circuits [shown in Fig. 4(b)] and synchronous pipeline circuits [shown in Fig. 4(a)]. Click elements are the handshake circuits in each stage in the asynchronous pipeline circuits, while the synchronous pipeline circuits have a global clock signal *clk*. The fire signals from the Click elements can be treated as local clocks for the corresponding DFFs. Therefore, when we use Verilog to describe the asynchronous BD circuits in the behavioral level, we instantiate the Click elements to build the handshaking channel and take the fire signals as local clocks. The combinational logic (*CL* in Fig. 4) circuits in the asynchronous circuits are the same as that in synchronous circuits.

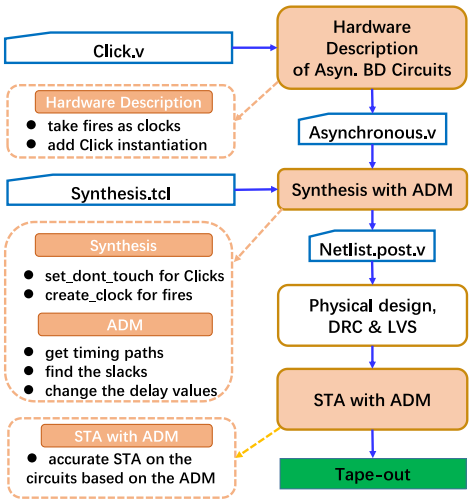

Fig. 3. Design flow for Click-based asynchronous BD circuits.

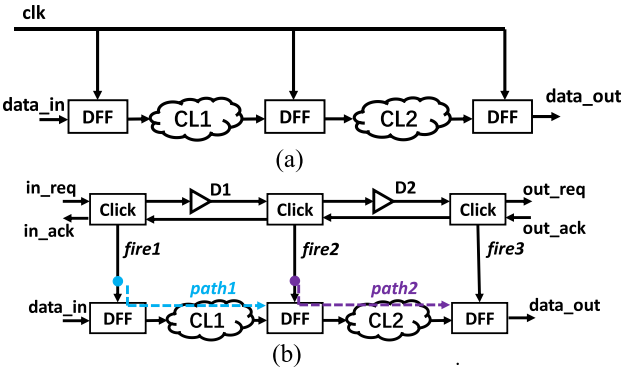

Fig. 4. (a) Synchronous and (b) asynchronous pipeline circuits.

B. Synthesis With the Method of Adaptive Delay Matching

We first design the Click elements in gate level by using the standard cells in the library. As mentioned in Section II, we take the fire signals generated by the Click elements as the local clocks. Therefore, we can adopt command *create_clock* to create clocks, with which we can synthesize the Click-based asynchronous BD circuits with DC. The synthesis process is different from that for synchronous circuits in the following aspects: 1) the Click element should not be modified to guarantee its correct operation once it is instantiated; 2) fire signals are treated as local clocks; 3) delay matching is needed; and 4) delay lines should be inserted carefully between the Click elements to make the circuits operate correctly without hazard. The way to create clocks and delay matching for the asynchronous circuits will be discussed in Section III-C.

C. Adaptive Delay Matching

Delay matching is essential to guarantee the timing constraint in expression (1) to be met for the BD circuits. Adaptive delay matching according to corresponding data paths is critical to ensure the function and better performance of the circuits. We could insert delay lines with the command *set_min_delay*. However, the most critical step is to determine the length of the inserted delay lines. Without a global clock, DC tools cannot obtain the timing path, making the delay matching impossible. In order to solve the problem, we put forward a method called ADM, as shown in Fig. 5, with which we can perform delay matching adaptively in asynchronous circuits.

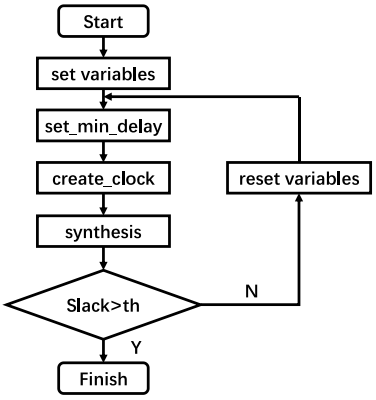

Fig. 5. Method of ADM.

```
1  # set variables
2  set var(1) 0.1
3  set var(2) 0.1
4  set_min_delay $var(1) ../..
5  set_min_delay $var(2) ../..
6  create_clock -name CK1 –period 30 –waveform {0 2} [get_pins fire_1]
7  create_generated_clock  -name CK2 –source [get_clocks fire1] –edges
   {1 3 5} -edge_shift {$var(1) $var(1) $var(1)} [get_pins fire_2]
8  create_generated_clock  -name CK3 -source [get_clocks fire1] -edges {1
   3 5} –egde_shift {$var(2) $var(2) $var(2)} [get_pins fire_3]
```

Fig. 6. Part of the synthesis scripts.

First, we generate local clocks for asynchronous circuits. In general, the DC tools will consider two clocks are synchronous if they share a common source and have a phase relationship. As discussed in Section II, the fire signals in the Click-based asynchronous BD circuits have a latency, which approximately equals to the buffer delay between the Click elements [such as *D1* or *D2* in Fig. 4(b)] plus the gates delay in Click elements. As a result, we can treat all the fire signals as local clocks which have a phase relationship. Therefore, we create clocks for the fire signals and describe their relationship using the command *create_generated_clock*. In this way, all the timing paths will be checked, and the combinational logic will also be optimized according to the latency of the fire signals by DC. Then, DC will find timing *path1* and timing *path2* as shown in Fig. 4(b) and the setup check will be performed by DC for timing *path1* and timing *path2*. During the synthesis process, the worst corner is used to perform the delay matching to guarantee the timing convergence. The phase relationship changes with different PVT corners and the STA on the circuits with different corners is described in detail in Section III-E.

Second, the proposed method of ADM not only finds out the timing paths in asynchronous circuits but also tries to find out the length of delay lines to be inserted between the Click elements to achieve the best performance of speed. We mark the total number of the pipeline stages as n. Obviously, we should set n delay variables using the command *set_min_delay*. As we do not know the exact delay values before synthesis, we first create n variables with the same value, and the latency between fire signals is set the same as the value in the command *set_min_delay*. For example, if n equals 2, part of the synthesis script is shown in Fig. 6 in which the clocks are created according to the initial variables. The initial values of the variables could be set to a small nonzero value, such as 0.1 ns in our case in the TSMC 65-nm process.

Finally, we synthesize the circuits and run the command *for_in_collection* to obtain the slack values of each timing path from the report of command *report_timing* cyclically. Taking the parasitic

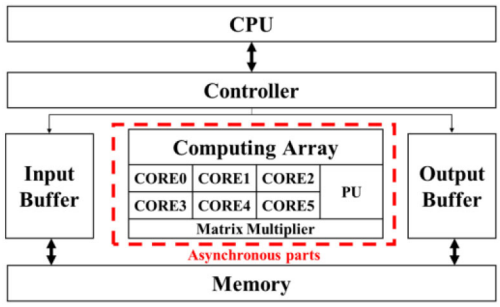

Fig. 7. Convolution core circuits.

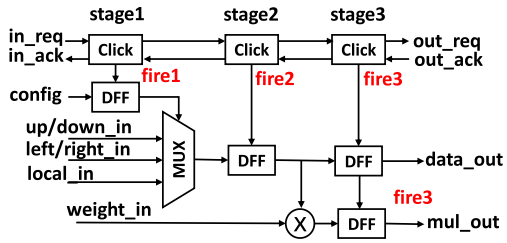

Fig. 8. PE circuits.

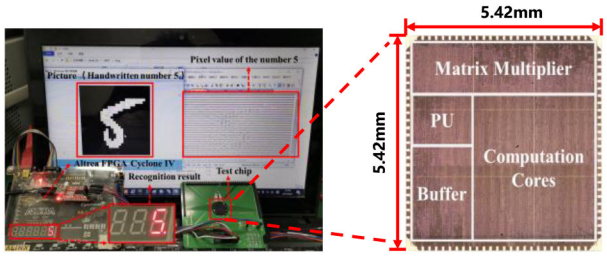

Fig. 9. Test platform and micro-photograph of asynchronous CNN accelerator in TSMC 180-nm CMOS process.

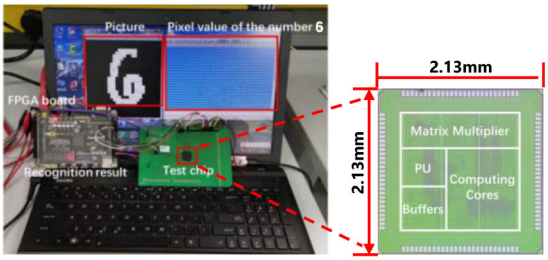

Fig. 10. Test platform and micro-photograph of asynchronous CNN accelerator in TSMC 65-nm process.

parameters after PnR into account, only if all the slacks reported have a margin (noted as *th*) of +10% with respect to the delay of the corresponding timing paths, will the synthesis be finished. If not, the delay variable of each stage will be reset to a new value, which is the delay value of the current timing path plus the threshold, and the circuit will be resynthesized. The synthesis process will be finished until the delay lines for each stage meet the timing constraint. Generally, only two iterations of the synthesis process are needed to finish the synthesis process according to our design if the initial value is set bigger than zero and smaller than 5 ns.

The proposed delay matching method is totally automatic during synthesis. This ADM method for the Clicked-based asynchronous BD circuits can also be used to perform static analysis of the asynchronous circuits. For complex asynchronous pipeline structures, including fork, join, split, or merge, ADM can also be used to perform delay matching with the created clocks and control circuits for different handshaking protocols [2], which controls the data flow to guarantee the correctness of circuits.

D. Physical Design, DRC, and LVS

With the netlist generated by DC, we use EDI for physical design. Commands *set_interactive_constraint_modes* [*all_constraint_modes -active*] and *set_dont_touch* should be used to avoid the EDI removing the delay lines inserted during the synthesis. The DRC and LVS check are also needed for the layout of the circuits. We should insert *D*2 (shown in Fig. 2) after clock tree synthesis if hold violations exists.

E. Static Timing Analysis With ADM

After the physical design and the parasitic parameter is extracted, we perform STA for the circuits at all the corners with PT. As we know, in the synthesis process, timing closure is achieved and gate-level circuits are generated without considering the parasitic effect which makes the timing analysis in synthesis not accurate. In Section III, the width of the clock depends on the simulation results without considering the parasitic parameter extracted. When we perform STA of the circuits, we should know the accurate width of the fire. With command *report_timing*, the delay of the specified timing path can be reported which determines the width of the fire signal. According to the delay reported, we create the clocks for fire signals. By specifying the timing path (such as the green line in Fig. 2), we can calculate the accurate delay between the two fire signals. The clock and generated clocks will be created according to the delay values calculated by PT. With the clocks created, we can obtain all the timing paths and perform timing analysis at all corners with PT.

IV. Case Study

With our methodology, we implement a Lenet-5 CNN accelerator which contains computing array (CA), input buffer, output buffer, and memory (shown in Fig. 7). Each convolution cores contains 25 processing elements (PEs) to perform 5×5 convolution. Each PE includes a register and a multiplier and its design is shown in Fig. 8. Data reuse is considered when PE is designed. The PE has three pipeline stages: data are loaded in the first stage, in the second stage, the multiplication and truncation are accomplished, and the input data will be stored temporarily in the third stage [15].

In order to verify our design flow, we first implemented the asynchronous CNN accelerator in TSMC 180-nm CMOS process (shown in Fig. 9) which can work properly and achieve the energy efficiency of 133 GOPS/W. To further verify the design flow proposed and the performance of asynchronous circuits, we implement the asynchronous accelerator in TSMC 65-nm CMOS process. The micro-photograph of the chip with a size of 2.1×2.1 mm^2, and the chip test platform are shown in Fig. 10, in which the test chip recognizes the input picture data and sent the results to FPGA board to display. As we can see, the picture of number six is correctly recognized by the test chip. Fig. 11 illustrates the waveforms of the request and fire signals from the test chip in 65-nm process, in which we can find the time intervals between *fire1* and *fire2*, *fire2* and *fire3* are 4 and 5.5 ns, respectively. The two asynchronous chips verify the design flow proposed in different processes. In our design, the time taken to synthesize the asynchronous and synchronous CNN accelerators is 61 and 53 min, respectively, the time for placing is 1 h 45 min and 1 h 47 min, respectively, and the time for routing is 46 and 31 min, respectively.

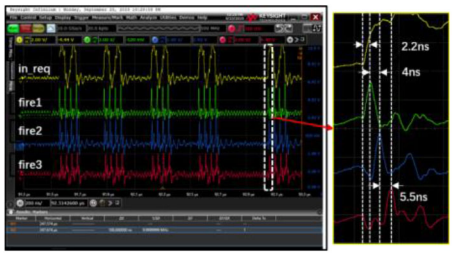

Fig. 11. Waveforms of the test chip.

TABLE I
TEST RESULTS OF ASYNCHRONOUS AND SYNCHRONOUS ACCELERATORS CHIPS IN TSMC 65-NM PROCESS

	Asynchronous	Synchronous
Area(mm^2)	1.96x1.96	1.93x1.93
Performance (GOPS)	60.9@150MHz	48.4@120MHz
Total Power(mW)	26.4(@100Mhz, 0.8V)	29.6@(100Mhz, 0.8V)
CA Power(mW)	4.15(@100Mhz, 0.8V)	5.91(@100Mhz, 0.8V)
Energy Efficiency (TOPS/W)	1.538	1.37

Compared with Petrify [5], Balsa [6], and desynchronization design flow [8], the proposed design flow can implement the asynchronous circuits down to mask layout without any special tools or new language with conventional EDA tools. Additionally, with ADM designers can perform ADM for asynchronous pipeline circuits automatically to achieve the best performance and ensure the timing for the asynchronous circuits.

For further comparison, we realize a synchronous CNN accelerator chip with the same architecture in TSMC 65-nm CMOS process, which differs in that the synchronous chip adopts synchronous pipeline while the asynchronous chip adopts Click-based pipeline when designing the CA. The performance comparison of the chips is presented in Table I, from which we can see that the area of the asynchronous one is almost the same as that of the synchronous one, but the asynchronous CA save 30% power compared with synchronous CA as a result of no clock power consumption and consuming power when and where needed. The highest working frequency of the asynchronous chip is 1.25× than that of the synchronous one. The energy efficiency of the asynchronous chip reaches 1.5 TOPS/W, consuming 26.4-mW power, both of which are 11% better than that of the synchronous one. Both the test results of the chips with 180- and 65-nm process verify our design flow for asynchronous BD circuits, which have a better energy efficiency of the CNN accelerator than that of the synchronous chip.

V. CONCLUSION

We put forward a design flow to implement Click-based asynchronous BD circuits using conventional EDA tools efficiently down to mask layout in this article. A novel delay matching method, ADM, is proposed for delay matching automatically to ensure the timing constraint and achieve the best performance. We perform STA based on ADM with PT. Asynchronous CNN accelerator chips in TSMC 180- and 65-nm process are implemented, respectively, to verify our design flow. Compared with other asynchronous toolsets, our design flow can realize asynchronous circuits more efficiently with conventional EDA tools. For further comparison, a synchronous CNN accelerator with the same structure is implemented in the same process, and the chips test results indicate that the asynchronous chip has 30% less power consumption in CA. Future work will focus on more automatic design flow to design larger scale and complex asynchronous circuits with different handshake circuits.

REFERENCES

[1] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design—A Systems Perspective*. Dordrecht, The Netherlands: Kluwer Acad. Publ., 2001, pp. 3–11.
[2] P. A. Beerel, R. O. Ozdag, and M. Ferretti, *A Designer's Guide to Asynchronous VLSI*. Cambridge, U.K.: Cambridge Univ. Press, 2010, pp. 7–9.
[3] F. Akopyan *et al.*, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," in *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.
[4] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," in *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.
[5] J. Cortadella *et al.*, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inf. Syst.*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.
[6] A. Bardsley and D. A. Edwards, "The Balsa asynchronous circuit synthesis system," in *Proc. Forum Design Lang.*, Sep. 2000.
[7] A. Ghiribaldi, D. Bertozzi, and S. M. Nowick, "A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems," in *Proc. Design Autom. Test Europe Conf. Exhibition (DATE)*, 2013, pp. 332–327.
[8] G. Miorandi, M. Balboni, S. M. Nowick, and D. Bertozzi, "Accurate assessment of bundled-data asynchronous NoCs enabled by a predictable and efficient hierarchical synthesis flow," in *Proc. IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, 2017, pp. 10–17.
[9] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static timing analysis of asynchronous bundled-data circuits," in *Proc. IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, 2018, pp. 110–118.
[10] G. Gimenez, J. Simatic, and L. Fesquet, "From signal transition graphs to timing closure: Application to bundled-data circuits," in *Proc. IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, 2019, pp. 86–95.
[11] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, 1989.
[12] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. VLSI*, 2001, pp. 9–17.
[13] A. Peeters, F. te Beest, M. de Wit, and W. Mallon, "Click elements: An implementation style for data-driven compilation," in *Proc. IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Grenoble, France, 2010, pp. 3–14.
[14] J. V. Manoranjan and K. S. Stevens, "Qualifying relative timing constraints for asynchronous circuits," in *Proc. IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Porto Alegre, Brazil, 2016, pp. 91–98.
[15] W. Chen, H. Wu, S. Wei, A. He, and H. Chen, "An asynchronous energy-efficient CNN accelerator with reconfigurable architecture," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2018, pp. 51–54.