An Efficient Asynchronous Circuits Design Flow with Backward Delay Propagation Constraint

Lingfeng Zhou[1], Shanlin Xiao[1,2,*], Huiyao Wang[1], Jinghai Wang[1], Zeyang Xu[1], Bohan Wang[1] and Zhiyi Yu[1,2,*]
[1] School of Microelectronics Science and Technology, Sun Yat-sen University, China
[2] Guangdong Provincial Key Laboratory of Optoelectronic Information Processing Chips and Systems, China
E-mail: {zhoulf25, wanghy373, wangjh335, xuzy55,wangbh9}@mail2.sysu.edu.cn
{xiaoshlin, yuzhiyi}@mail.sysu.edu.cn

***Abstract*—Asynchronous circuits have recently become more popular in Internet of Things (IoT) and neural network chips because of their potential low power consumption. However, due to the lack of Electronic Design Automation (EDA) tools, the asynchronous circuits design efficiency remains low and faces challenges in large-scale applications. This paper proposes a new asynchronous circuits design flow using traditional EDA tools, and applies a new backward delay propagation constraint (BDPC) method. In this method, control paths and data paths are tightly coupled and analyzed together to improve the accuracy of static timing analysis. Compared to previous works, the proposed design flow and constraint method offer significant advantages in terms of accuracy and efficiency. To verify this flow, an asynchronous RISC-V processor was implemented on TSMC 65nm process. Compared to synchronous version, asynchronous processor achieves a power optimization of 17.4% while maintaining the same speed and area.**

***Index Terms*—asynchronous circuits, low-power, methodology, RISC-V, static timing analysis**

I. Introduction

As the scale of integrated circuits expands, power consumption, particularly clock network's power has become the key barrier to advance performance. For this, asynchronous circuits offer a solution that meets strict power requirements and have the potential to enhance performance by replacing the global clock with handshakes [1]. Therefore, asynchronous circuits have been applied in neural networks like Loihi [2], TrueNorth [3] and AsNNP [4], as well as in Internet of Things (IoT) devices to save power [5]. However, due to lack of Electronic Design Automation (EDA) tools, designing high-performance and low-power asynchronous circuits is still a difficult work.

Asynchronous circuits can be divided into Quasi-Delay-Insensitive (QDI) and Bundled-Data (BD) circuits. QDI circuits typically use dual-rail circuits to enhance robustness, but the large area cost and speed degradation are unacceptable [4]. In contrast, BD circuits are more popular due to similar area to synchronous circuits, but require relative timing constraints (RTC) and delay matching units. Unless otherwise specified, asynchronous circuits refer to BD circuits in this paper.

* Corresponding authors: Shanlin Xiao; Zhiyi Yu.

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62334014; in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101410004.

Many works have been made to design asynchronous circuits. Petrify [6] synthesizes asynchronous controllers based on signal transition graph (STG), but only controllers. Balsa [7] is used to describe and synthesize asynchronous circuits based on syntax-directed compilation into handshake circuits, but the efficiency is low. [8]–[11] use set_min/max_delay commands to manually specify and constrain the data paths step by step, which are too complex. In [12], [13], a combination of clocks is used to describe paths based on STG, allowing EDA tools to capture RTC, but draw STGs is difficult.

Moreover, adaptive delay matching (ADM) method [14] was proposed to implement click-based asynchronous circuits, in which sets of generated clocks are used to constrain the data path. ADM method is a good design flow with some successful cases, such as an asynchronous spiking neural networks accelerator designed using the ADM method achieved good results in [15]. However, in ADM method, the clocks propagated path are broken, and use a fixed value to describe the phase relationship between clocks, which will lead to inaccurate static timing analysis (STA). In addition, conditional data-flow components like MUX and DEMUX are vital parts of asynchronous circuits, but the STA of these is ignored by ADM flow.

In order to implement asynchronous circuits more efficiently, this paper proposes a new asynchronous circuits design flow using traditional EDA tools, and apply a new constraint method called backward delay propagation constraints (BDPC). The new design flow and constraint methods significantly simplify the design flow and improve the accuracy of STA. The main contributions of this study are as follows:

- A new asynchronous circuits design flow using traditional EDA tools is proposed. Compared with previous works, the proposed design flow is simpler and more efficient.
- A new constraint method, the backward delay propagation constraint (BDPC), is proposed. In this method, control paths and data paths are tightly coupled and analyzed together to improve the accuracy of STA. Compared with the previous works, the BDPC method is more accurate.
- An asynchronous RISC-V processor was implemented on TSMC 65nm to verify the proposed flow. Compared to synchronous version, the asynchronous processor significantly reduces power consumption while maintaining the same speed and area.

979-8-3503-4859-0/DATE24/© 2024 EDAA

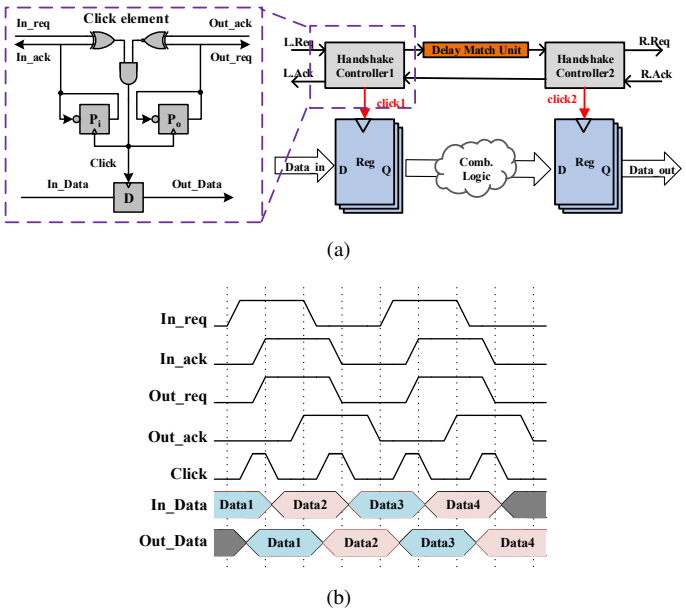


Fig. 1. Asynchronous pipeline based on Click elements. (a) The architecture of asynchronous pipeline. (b) The operational timing diagram of Click elements.

II. Background

A. Asynchronous Pipeline and Click Elements

Different from synchronous circuits, asynchronous circuits use handshake protocols to communicate and generate local clocks. A popular asynchronous controller, Click elements and its phase-decoupled template were proposed in [16], [17]. Fig. 1(a) shows an asynchronous pipeline based on Click elements, consisting of control path and data path. Fig. 1(b) shows the data transmission in asynchronous pipeline: 1) When $In\_req \neq Out\_ack$, means that the upper-level sender has prepared the data. 2) When $Out\_req = In\_ack$, means the downstream receiver has received the data. 3) Then the Click element generate pulse and trigger the flip-flop. 4) In order to ensure the validity of data, delay units are inserted with the delay always longer than the longest transmission time.

B. Timing Requirements of Asynchronous Circuits

The requirements of period (T_{period}), setup time (T_{setup}), and hold time (T_{hold}) are shown in equations (1), (2), and (3):

$$T_{period} = T_{delay} + T_{click2} \tag{1}$$

$$T_{setup} + T_{c-q,max} + T_{comb,max} + T_{skew2} < T_{period} + T_{skew1} \tag{2}$$

$$T_{hold} + T_{skew2} < T_{c-q1,min} + T_{click1,min} + T_{comb,min} + T_{skew1} \tag{3}$$

where T_{delay} denotes the delay of the delay march unit; T_{click1} and T_{click2} denote the delay of the asynchronous controller to generate pulse; T_{c-q} denotes the delay from the register's clock pin to output; T_{comb} denotes the delay of combinational logic; while T_{skew1} and T_{skew2} denote the clocks skews.

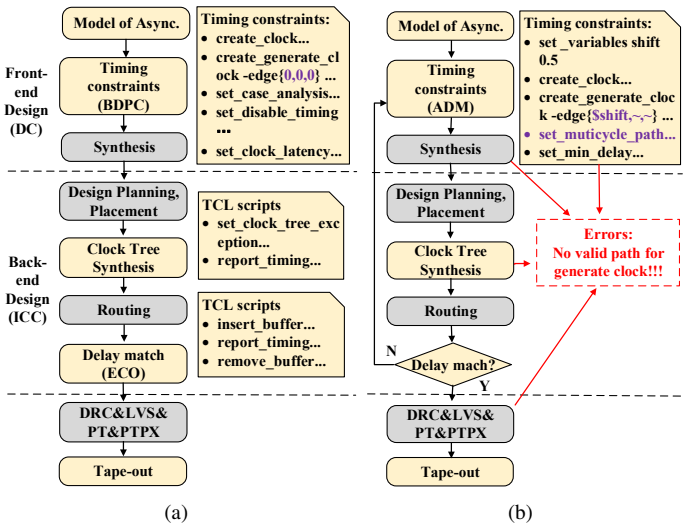


Fig. 2. Comparison of proposed BDPC design flow and ADM [14] design flow. (a) BDPC design flow. (b) ADM design flow.

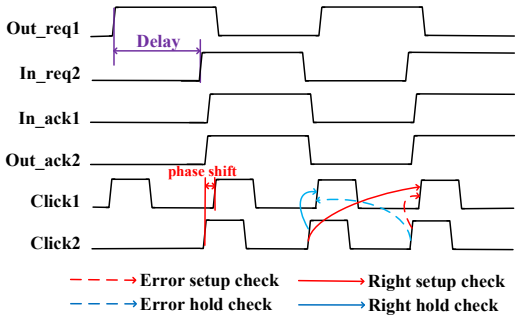


Fig. 3. Handshake causes phase shift and ADM needs special constraints.

III. Design Flow Based On Backward Delay Propagation Constraint

In this section, we will introduce an new asynchronous circuits design flow as shown in Fig. 2(a). And the ADM design flow is shown in Fig. 2(b). The gray part represents the same steps as synchronous circuits, while the yellow part represents the special steps of asynchronous circuits.

A. Model of Asynchronous Circuits

The most commonly method for designing asynchronous circuits is through the conversion of synchronous circuits. As shown in Fig. 1, we replace the global clock with a series of local clocks generated by asynchronous controllers and use handshake to communicate between asynchronous controllers.

Asynchronous circuits can also utilize multiple handshake channels to perform selective handshaking or change the number of pipeline stages. This enables improvements in speed and power consumption reduction. In the subsequent section on asynchronous processors, this paper will provide a more detailed introduction to the application of selective handshake.

B. Backward Delay Propagation Constraint Method

As shown in Fig. 3, after triggering $click1$, it is necessary to wait for $click2$ triggered (ack signal feedback) before

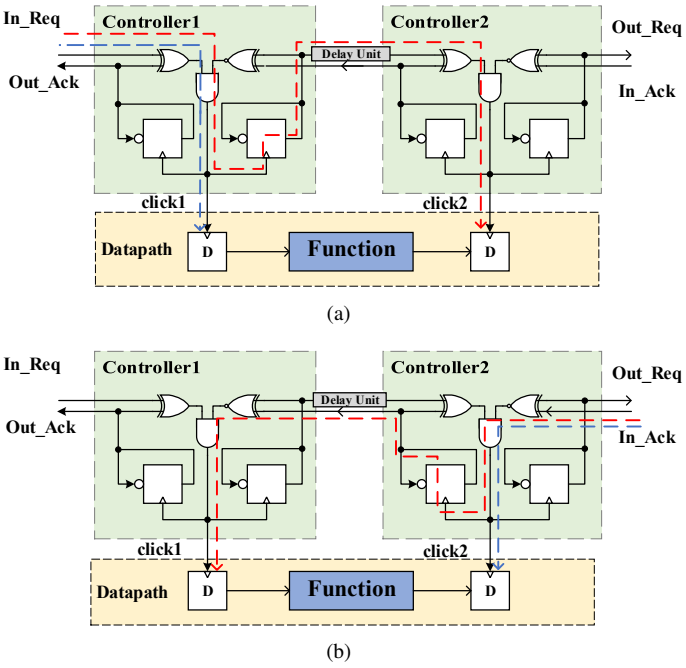


Fig. 4. Both perspectives on observing timing propagation are supported by BDPC. (a) Forward delay propagation. (b) Backward delay propagation.

triggering $click1$ again. There will be a phase shift between $click1$ and $click2$, which impacts setup and hold. The phase shift tightens the setup and relaxes the hold from $click1$ to $click2$, while relaxes the setup and tightens the hold from $click2$ to $click1$. Therefore, the data paths and control paths need to be analyzed together in tightly coupled form.

However, ADM do not achieve this, but separates them and ignores the analysis of control paths. It uses the $create\_generate\_clock$ command to define clocks at non leaf cells to cut off clock connections, with $-edge\ \{\$shift\ ...\}$ option to define a fixed estimated phase shift between clocks. Then use the $set\_multicycle\_path$ command to modify the sampling points of the EDA tool.

The ADM method is simple, but has two significant drawbacks: 1) It forcefully interrupts the propagation paths between clicks, causing the tool to be unable to analyze the phase shift caused by handshake. 2) It incorrectly assumes that the phase shift is a fixed value. In fact, the rising and falling delays of gates during handshake are different, and the delay also varies under different process conditions. That means relying solely on specifying fixed estimated values is inaccurate and incomplete.

Therefore, in order to achieve accurate STA, it is necessary to analyse the click control path, which cause the phase shift by handshake. There are two different perspectives to analyse. As shown in Fig. 4(a), the forward delay propagation approach, which means that it propagates from click1 to click2, then to click3, and so on, along the req signal sequence. In contrast to the forward delay propagation, the backward delay propagation shown in Fig. 4(b) starts from the last click and propagates forward along the ack signal to the first click.

Our flow can analyze timing through both perspectives, but the backward delay propagation can achieve better results. The

```
Timing constraints:

create_clock -name click3 -period ... [get_pins ...click3]...
create_generated_clock -name click2 -source [get_pins ...click3] ... [get_pins ...click2]...
create_generated_clock -name click1 -source [get_pins ...click2] ... [get_pins ...click1]...
...

set_case_analysis 0 [get_cells .../xnr/A1]...
...

set_disable_timing -from A1 -to Z ... [get_cells .../xor]...
...

set_clock_latency 0.1 [get_clocks click2]...
set_clock_latency 0.2 [get_clocks click1]...
...
```

Fig. 5. Example of BDPC Timing Constraints.

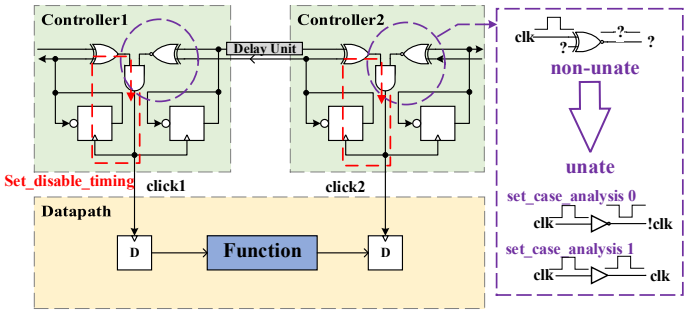


Fig. 6. Constraints to help restore and identify timing propagation path.

steps of proposed flow with BDPC method are as follow and Fig. 5 shows an example:

1) Use the $create\_clock$ command to define the clock generated by the last click as the source clock, and then uses the $create\_generate\_clock$ command sequentially defines the clock forward (all define points are leaf cells, and without using $-edge\{\}$ to specify phase shift).
2) Use the $set\_clock\_latency$ command to simulate phase shift. It is not necessary, but recommend using. The value serves as a guide for synthesis and will be automatically replaced with the actual circuit's phase shift after clock tree synthesis.
3) Use the $set\_disable\_path$ command to cut off timing loops in click elements. As shown in the Fig. 6, the signal follows the red path, passes through the AND gate, the trigger, the XOR gate, and then returns to the origin. Therefore, the $set\_disable\_path$ command needs to be used to cut off the path returning from the XOR gate.
4) Use the $set\_case\_analyze$ command to eliminate the non-unate gate. As shown in Fig. 6, non-unate gates are those for which the tool cannot determine whether the signal will remain unchanged, invert, or disappear when the clock path passes through a logic gate. Therefore, the $set\_case\_analyze$ commands are used to equate the function of the XOR gate to a buffer for analysis.

Through the first and second steps, BDPC retains the handshake analysis and converts phase shift into the form of clock delay. The third and fourth steps help tools accurately identify and analyze handshake paths. Therefore, BDPC analyzes data paths and control paths together in tightly coupled form.

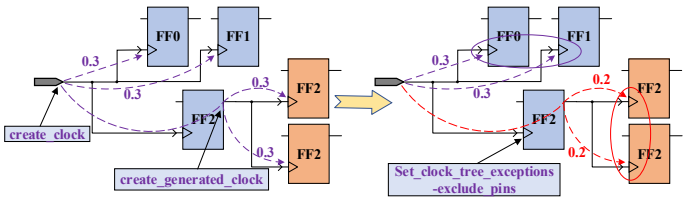


Fig. 7. Set clock exception to cut off the connection between the clock trees to follow the requirements of asynchronous circuits.

C. Clock Tree Synthesis

The ADM method cuts off the connection between Clicks, so no additional operations are needed. But it ignores the error messages and its accuracy is low.

BDPC only uses $set\_clock\_tree\_exception\ -excludepins$ command to break the interconnections between clock trees before clock tree synthesis, as shown in the Fig. 7. This is because the default clock tree synthesis will balance all registers driven by the source clock and sub clock, such as the FF3 and FF2, which violates the principles of asynchronous circuits. Therefore, it is necessary to set clock tree exception to remove balance between source clock and sub clock. After the setting, every click can share own local clock tree.

D. Delay marching with ECO

The ADM method utilizes the set_min_delay command in timing constraints before synthesis to allow the EDA tool insert the minimum delay. However, if the final result can not meet the requirements, it is necessary to go back to the initial steps and re-synthesis, leading to lower fault tolerance rate and efficiency.

In comparison, the BDPC method exhibits higher efficiency and accuracy in delay matching. The processes of the BDPC flow completes the delay matching are as follow:

1) First, observe the gap between the real period and the requirements through timing reports or post-gate simulations after completing routing.
2) Then, calculate and insert suitable buffers based on the process manual by Engineering Change Order (ECO).
3) Observe whether the requirements are met. If not, recalculate and iterate.

Although this flow requires manual completion, the workload is not significant, and if errors occur, there is no need to roll back a lot, resulting in extremely low iteration costs.

IV. Case Study: Asynchronous RISC-V Processor

In order to validate proposed design flow, we designed a representative example: an asynchronous RISC-V processor.

A. Asynchronous RISC-V Processor Architecture

We designed a super-scalar processor with the architecture shown in Fig. 8, based on an open-source project, biriscv [18]. The processor features a 7-stage pipeline, including PC predict and update, Instruction Fetch, Pre-decode, Issue and Operand Fetch, Execution, and Write back. It supports the concurrent dispatch of two instructions. In execution stage, the processor is equipped with 2 ALU compute units, 1 multiplier unit, 1 memory access unit, and 1 CSR unit.

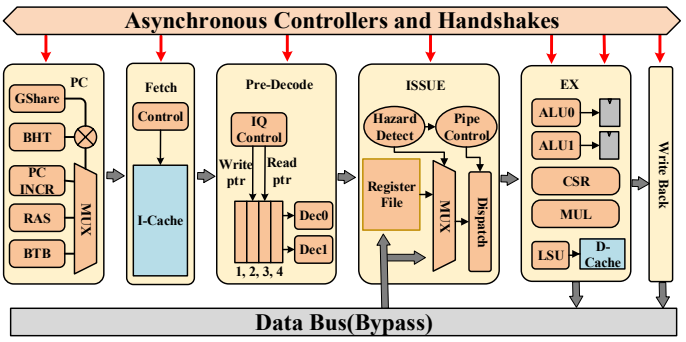


Fig. 8. Asynchronous super-scalar RISC-V processor architecture.

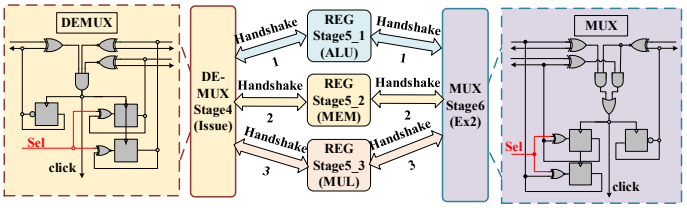


Fig. 9. CLICK_MUX/DEMUX structure for selective handshakes.

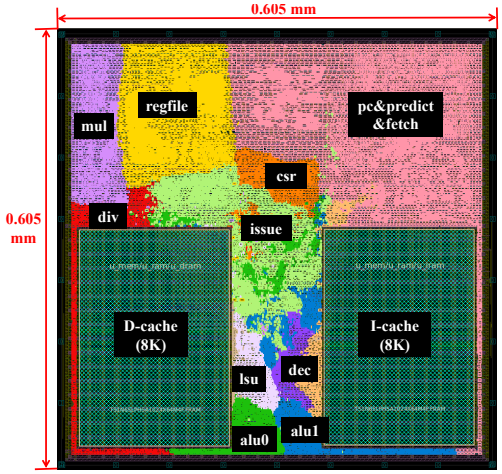


Fig. 10. Asynchronous RISC-V processor layout with BDPC flow.

B. Selective Handshakes

Additionally, as shown in Fig. 9, in the EX stage, we also use the CLICK_MUX/DEMUX structure to create special control paths. The CLICK_MUX enables a multi-input, single-output handshake, while the CLICK_DEMUX enables a single-input, multi-output handshake. We employed CLICK_MUX and CLICK_DEMUX to create 3 separate handshake channels for different units. Through this structure, the controllers achieve selective handshake operations, triggering the registers of each channel only when needed to save power.

V. Implementation Result and Evaluation

We use two design flows to implement the asynchronous processor (BDPC's result called ASYNC), along with its synchronous version (SYNC), in TSMC 65nm CMOS technology.

TABLE I
DESIGN FLOWS COMPARISON FOR IMPLEMENTING ASYNCHRONOUS PROCESSORS UNDER DIFFERENT DEVIATION

Deviation magnitude [a]		0.1	0.5	1	1.5	2
Area(mm^2) [b]	ADM [14]	0.188	0.193	0.199	0.199	0.203
	BDPC	0.18	0.18	0.181	0.181	0.182
Freq.(MHz)	ADM [14]	358	358	358	344	333
	BDPC	358	358	358	358	351
Power(μW/MHz)	ADM [14]	100.8	103.4	104.9	105.7	111
	BDPC	99.9	101.9	102.2	103.6	104.6

[a] The unit of deviation magnitude is the delay of the standard 2 input NAND gate under the TSMC 65nm process.
[b] The area is the area of the standard cells.

TABLE II
DESIGN FLOWS COMPARISON FROM ACCURACY & EFFICIENCY ASPECTS

	BDPC	ADM [14]
Constraint comprehensiveness	Yes	No
Analysis accuracy	High	Low
Similarity with synchronous flow	High	High
Design efficiency	High	Medium

The layout of ASYNC is shown in Fig. 10. All processors employed clock-gating technology. All processors' functionality is firstly verified by the RISC-V test-tool: riscv-arch-test [19]. We use 5 representative test benches to evaluate power, including bubble sort, matrix transpose, fibonacci, factorial and std_bge.

A. Design Flow Comparison

Table I presents the results of two design flows. when the deviation magnitude is small, ADM can maintain a high frequency. However, as the deviation magnitude increases, the frequency of ADM gradually decreases. Although BDPC is also affected, it exhibits a much smaller reduction.

In terms of power and area, ADM's power and area continuously increase. On the other hand, BDPC demonstrates stability. When the deviation magnitude is 1, BDPC achieves the same speed and 3% reduction in power consumption compared to ADM, along with a 9% decrease in area. When the deviation magnitude is 2, BDPC achieves a 6% improve in speed and 6% reduction in power consumption compared to ADM, along with a 10.4% decrease in area.

This is because ADM's deviation persist throughout the design process, while BDPC's deviation are automatically replaced with accurate actual data during the clock tree stage.

Table II give a summary about the advantages of BDPC:

1) On the aspect of STA accuracy: The ADM flow has the lower constraint integrity as it actively disconnects clock relationships, making it unable to analyze control paths. While BDPC flow can perform control path analysis, so the BDPC flow achieves the higher accuracy.
2) On the aspect of design efficiency: When delay marching result not meet requirements, the ADM flow need to return to the beginning, while BDPC flow just need to re-ECO. That means BDPC flow is more efficient.

B. Comparison with Synchronous Baseline

Due to the significant influence of deviation magnitude on the results of the ADM flow and the difficulty in determining a representative deviation magnitude to accurately measure the level of ADM, we only compare the ASYNC and SYNC here.

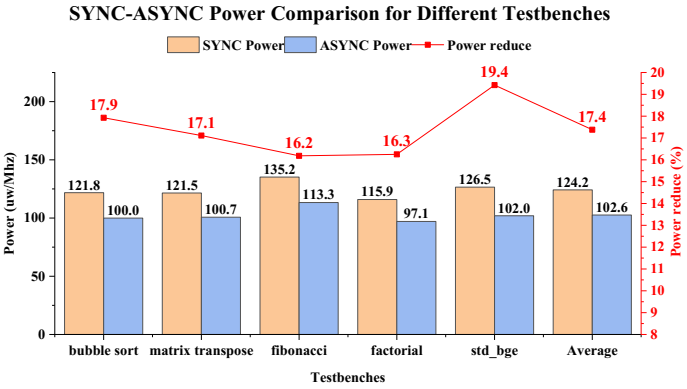
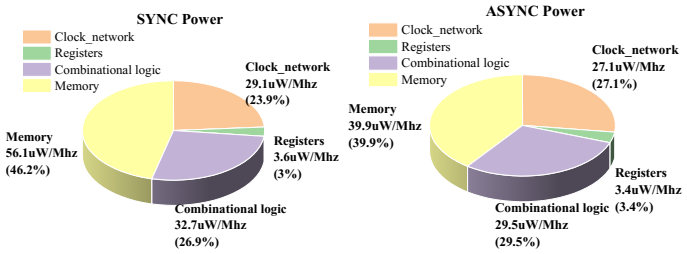


Fig. 11. Comparison of power consumption between ASYNC and SYNC at the same speed.



(a) (b)

Fig. 12. Power breakdown under bubble sort program. (a) SYNC power breakdown. (b) ASYNC power breakdown.

As shown in Fig. 11, compared to SYNC, ASYNC exhibit significant optimizations in power when at same speed. ASYNC achieve power savings of 17.9% in the bubble sort program, 17.1% in the matrix transpose program, 16.2% in the Fibonacci program, 16.3% in the factorial program, and 19.4% in the std_bge program, resulting in an average power reduction of 17.4%. It is worth noting that SYNC did not take into account for the power consumption of the phase-locked loop (PLL, required for synchronous circuits to work), while ASYNC do not require PLL, resulting in even greater power advantages.

C. Power Advantages Analysis

The power advantage of ASYNC is mainly due to the optimization of the clock network and the selective handshakes. Fig. 12 shows the power breakdown under the bubble program.

Compared to SYNC, ASYNC reduces the power of the clock network by 7% and the power of registers by 6%. This is achieved by employing local clock trees instead of a global clock tree, which mitigates the difficulties in achieving clock tree balance and reduces the network scale, lowering power.

Additionally, ASYNC achieves a 29% reduction in power for memory and a 10% reduction for combinational logic. This is due to the selective handshakes structure, which enables selective handshaking based on instructions and reduces register flip-flop rates, computational complexity of combinational logic, and memory read/write operations. Moreover, it also further contributes to the reduction of power in the clock network.

TABLE III
COMPARISON WITH OTHER RISC PROCESSORS

Core	This work		Others		
	ASYNC	SYNC	SiFive E31 [20]	XT E907 [21]	ARM Cortex A5 [22]
ISA	RV32IM	RV32IM	RV32IMAC	RV32IMAC	ARMv7
Tech.(nm)	65	65	28	28	40
Freq.(MHz) [a]	363.7	358	374	430	615
Area(mm^2) [b]	0.234	0.234	0.286 [c]	NA	0.713
Power(μW/MHz) [d]	102.6	124.2	141 [e]	156	211

[a] The data have been scaled to 65nm. Under the original process, the frequency of SiFive E31, XT E907, and ARM Cortex A5 are 870, 1000, and 1000MHz, respectively.
[b] The data have been scaled to 65nm. Under the original process, the area of SiFive E31 and ARM Cortex A5 are 0.053 and 0.27mm^2, respectively. All area do not include SRAMs.
[c] Area numbers are from 50MHz Post-Route.
[d] The data have been scaled to 65nm. Under the original process, the power of SiFive E31, XT E907, and ARM Cortex A5 are 26.6, 30, and 80μw/MHz, respectively. And the power of SYNC does not include PLL, while ASYNC does not need PLL.
[e] The power of SiFive E31 do not include RAMs, and were measured at 200MHz.

D. Comparison with Other RISC Processors

In this paper, three representative processors were chosen for comparison, SiFive E31 [20], XT E907 [21], and ARM Cortex A5 [22].

The comparison results are shown in Table III. It is evident that our SYNC already exhibit significant advantages in power compared to other processors. Moreover, ASYNC further enhances this advantage. Compared to SiFive E31, ASYNC achieved a power reduction of 27.3%. Compared to XT E907, the power was reduced by 34.2%. Additionally, compared to ARM Cortex A5, the power was reduced by 51.4%.

VI. CONCLUSION

This paper proposes a new asynchronous circuits design flow using traditional EDA tools, and applies a new backward delay propagation constraint (BDPC) method. In this method, control paths and data paths are tightly coupled and analyzed together to improve the accuracy of STA. Compared to other design flows, the proposed design flow and constraint method offer significant advantages in accuracy, simplicity and efficiency. An asynchronous RISC-V processor was implemented on TSMC 65nm process to validate the flow. Compared to synchronous version, the asynchronous processor achieves a power optimization of 17.4% while maintaining the same speed and area.

REFERENCES

[1] J. Spars and S. Furber, *Principles of Asynchronous Circuit DesignA Systems Perspective*. Springer New York, NY, 2001.

[2] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[3] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[4] S. Xiao, W. Liu, J. Lin, and Z. Yu, "A Data-Driven Asynchronous Neural Network Accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1874–1886, Sep. 2021.

[5] Z. Wang, L. Ye, H. Zhanq, J. Ru, H. Fan, Y. Wang, and R. Huang, "A 57nW Software-Defined Always-On Wake-Up Chip for IoT Devices with Asynchronous Pipelined Event-Driven Architecture and Time-Shielding Level-Crossing ADC," *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 314–316, 2020.

[6] J. Cortadella, M. Kishnevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers," *IEICE Trans Information & Systems D*, vol. 80, no. E80-D, pp. 315–325, 1997.

[7] A. Bardsley and D. A. Edwards, "The Balsa Asynchronous Circuit Synthesis System," 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:8657240

[8] A. Ghiribaldi, D. Bertozzi, and S. M. Nowick, "A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 332–337.

[9] 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 *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2017, pp. 10–17.

[10] F. A. Kuentzer and M. Krstic, "Soft Error Detection and Correction Architecture for Asynchronous Bundled Data Designs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4883–4894, 2020.

[11] M. Gibiluka, M. T. Moreira, and N. L. Vilar Calazans, "A Bundled-Data Asynchronous Circuit Synthesis Flow Using a Commercial EDA Framework," in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 79–86.

[12] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static Timing Analysis of Asynchronous Bundled-Data Circuits," in *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2018, pp. 110–118.

[13] G. Gimenez, J. Simatic, and L. Fesquet, "From Signal Transition Graphs to Timing Closure: Application to Bundled-Data Circuits," in *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2019, pp. 86–95.

[14] H. Wu, Z. Su, J. Zhang, S. Wei, Z. Wang, and H. Chen, "A Design Flow for Click-Based Asynchronous Circuits Design With Conventional EDA Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2421–2425, 2021.

[15] J. Zhang, M. Liang, J. Wei, S. Wei, and H. Chen, "A 28nm Configurable Asynchronous SNN Accelerator with Energy-Efficient Learning," in *2021 27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2021, pp. 34–39.

[16] A. Peeters, F. te Beest, M. de Wit, and W. Mallon, "Click Elements: An Implementation Style for Data-Driven Compilation," in *2010 IEEE Symposium on Asynchronous Circuits and Systems*, 2010, pp. 3–14.

[17] A. Mardari, Z. Jelicov, and J. Spars, "Design and FPGA-implementation of Asynchronous Circuits Using Two-Phase Handshaking," in *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2019, pp. 9–18.

[18] ultraembedded., "biriscv." Accessed: Dec. 12, 2023. [Online]. Available: https://github.com/ultraembedded/biriscv.

[19] "riscv-arch-test," Accessed: Jul. 14, 2023. [Online]. Available: https://www.github.com/riscv/riscv-arch-test.

[20] I. SiFive Co., "SiFive E31 Standard Core - High-performance 32-bit embedded RISC-V Core IP." Accessed: Jul. 14, 2023. [Online]. Available: https://d2pn104n81t9m2.cloudfront.net/products/risc-v-core-ip/e3/e31/.

[21] L. T-Head Semiconductor Co., "T-Head XuanTie E907 Processor." Accessed: Jul. 14, 2023. [Online]. Available: https://www.t-head.cn/product/E907.

[22] I. ARM Co., "ARM Cortex-A5 Processor Performance and Specification." Accessed: Jul. 14, 2023. [Online]. Available: http://www.arm.com/products/processors/cortex-a/cortex-a5.php.