IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 3, 2021, accepted November 23, 2021, date of publication November 25, 2021,
date of current version December 9, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3130954

An FPGA Compliant Single-Rail Encoded Asynchronous Pipeline

ADNAN GHAFOOR[1], **MUHAMMAD WAQAR MUGHAL**[1], **AND ARBAB A. KHAN**[2]
[1]Faculty of Information Technology, University of Central Punjab, Lahore 54000, Pakistan
[2]Faculty of Engineering and Technology, International Islamic University, Islamabad 44000, Pakistan

Corresponding author: Adnan Ghafoor (adnan.ghafoor@ucp.edu.pk)

ABSTRACT Asynchronous systems are native to a full custom domain. Their implementation using auto place-and-route tools requires dynamic calibration of interconnects delays in addition to the placement of predefined static delay elements. This paper presents a completion detector for a single-rail bit encoded datapath that, as an adaptive-delay element, eliminates the need to insert any predefined delay element and caters to routing delays dynamically. A programmable pulse-generator is also proposed that empowers the designers to generate clock signals based on the timing report obtained from the CAD tool to drive various synchronous subsystems and embedded resources like BRAMs in FPGAs. Employing these components, we present an asynchronous pipeline model with implicit control to expedite migration from the traditional synchronous pipelines to their asynchronous counterparts. A single-rail bit encoded datapath has been used to utilize chip area effectively instead of a delay-insensitive dual-rail datapath, and a two-phase handshake protocol has been adopted as opposed to a four-phase handshake protocol to lower handshaking overhead. A RISC processor validates the proposed asynchronous pipeline model, exhibiting a smooth functionality and power-delay parameter comparable to that of a synchronous pipeline, in addition to ease of routing and avoiding clock skews in a complex system-on-chip.

INDEX TERMS Adaptive delay, asynchronous pipeline, auto place-and-route, microprocessor.

I. INTRODUCTION

Down-scaling of semiconductor technology increases the number of transistors exponentially, allowing integration of higher density Systems on Chip (SOC) to build larger circuits [1]. The interconnect delays have become so significant that it is challenging to distribute the clock network over the chip area without ignoring clock skews [2]. The high power and clock skew associated with distributed networks compels researchers to consider asynchronous design methodologies [3].

FPGAs provide a platform to implement any digital system. They have gained popularity due to their support for quick system prototyping and low costs. However, the logic implemented on FPGAs consumes more area and power than ASIC implementation of the same logic. Commercial FPGAs are synchronous except "Speedster" by Achronix [4], [5], which supports asynchronous logic. Implementation of an asynchronous system on conventional FPGAs requires knowledge of synthesis, placement, and routing tools.

A pipeline is a well-proven technique for enhancing the throughput of a system. In an asynchronous pipeline, the absence of a common clock eliminates the imposed restrictions on the system and lets the pipeline stages work independently of others. This independence brings some positive implications in terms of elasticity, modularity, increased performance, on-demand dynamic power dissipation [6], and electromagnetic compatibility.

The path connecting two stages in an asynchronous pipeline can either be a single-rail or dual-rail bit encoded. In a single-rail bit encoding, one wire represents a single bit [2], whereas a dual-rail encoding requires two wires to represent a single bit. Both datapaths require handshake protocols [7], either two-phase or four-phase [8], [9], comprising Request (Req) and Acknowledge (Ack) signals, to synchronize various stages. The single-rail encoding scheme necessitates the placement of a delay element, typically an inverter chain or a counter, in the Req line to slow down request signals to match the speed of the datapath. This may

The associate editor coordinating the review of this manuscript and approving it for publication was Pinjia Zhang.

also be necessary in the Ack line to meet the hold time requirements of inter-stage latches [10]. The delay-element placement might not work as the auto Place-And-Route (PAR) tools are unaware of asynchronous design methodology [11]; therefore, they do not preserve the delay ratios between control and various nets of datapaths [12]. This delay mismatch may cause setup and hold time violations and result in design failure. The problem leads to a need for a circuit that calibrates the delays after the design is placed and routed.

To deal with the delays, we propose a Completion Detector (CD), which senses its input for complete receipt of incoming data and generates the Ack signal when the hold time is satisfied. The proposed CD enables the PAR tools to implement the design without notable design constraints except routing optimization. Synchronous components such as RAM, counters and registers may also be placed within pipeline stages requiring a clock signal that meets their timing requirements. Hence, a programmable pulse generator is also proposed.

Our key contributions are as follow:

- Designing a logic-based completion detector that acts as an adaptive-delay element in an asynchronous pipeline to store only valid results in inter-stage latches/registers.
- Designing an on-chip ring oscillator to drive shift registers whose adjustable size as a programmable delay-element allows quick modeling of delays.
- Designing an asynchronous pipeline model that combines adaptive and programmable delays, enabling designers to insert synchronous and FPGA's embedded components inside the pipeline stages.
- Validation of the proposed asynchronous pipeline model by using a complex application of a pipelined RISC machine.

The remainder of the paper is organized as follows. In section II, conventional pipelines are discussed, whereas section III presents proposed building blocks. Section IV presents the proposed asynchronous pipeline model using building blocks, while its application is presented in section V. Section VI presents results, comparisons, and discussion. Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

Asynchronous systems have widely been implemented in full custom ASICs. Their implementation using auto PAR tools requires a delay-insensitive bit encoding scheme (like dual-rail bit encoding). In dual-rail bit encoding, where two wires are required for propagation of a single bit, the circuit requires more chip area, impacting performance and power parameters [13]. However, efficient utilization of the resources and better power-delay products require single-rail encoding schemes. Single-rail encoded datapaths require placement of precise delay elements in control path parallel to the datapath covering both logic and interconnects delays which are intricate to handle without the support of PAR tools. Manual PAR requires the same effort and time that is required for full custom designs.

The high-performance asynchronous pipelines based on single-rail encoding schemes such as Micropipelines [14], MOUSETRAP [15], GasP [16] use predefined delays in their Req lines and exist in full custom ASICs. However, the William PSO pipeline [17] and lookahead pipelines [18] use dual-rail encoded datapath and generate Ack signal using a dual-rail completion detector. The completion detector for single-rail encoded datapath was first introduced by [19] and the work was further carried out by [20], [21] [22]. They detect logic completion by activity monitoring using current sensing. The circuit declares logic completion if the current gets stable or no variation in current is detected for a finite amount of time. The reliable data can be stored in an inter-stage latch only after logic completion. In case of using FPGA platforms to implement asynchronous systems, it is pertinent to mention here that FPGAs neither contain current-sensing completion detection circuitry nor can the same be built using existing configurable logic blocks.

There has been little work on implementing asynchronous pipelines on FPGAs with a single-rail encoding scheme. The authors in [12], [23]–[26] implemented asynchronous pipelines in FPGAs for various applications and inserted predefined delays in Req lines parallel to logic and wire delays. They do not offer dynamic calibration of delays, and there is no detail for inserting sequential components like RAMs and counters that require clock signals between pipeline stages. There has been no concrete implementation of an asynchronous pipeline on FPGAs using single-rail encoding schemes with an adaptive-delay element or the implementation that allows the insertion of sequential components in the pipeline targeting various applications.

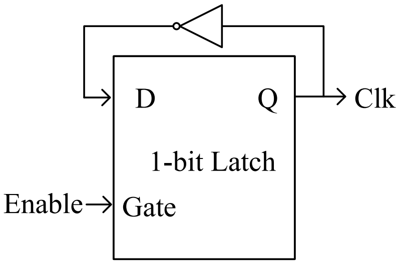

FIGURE 1. On-chip ring oscillator.

III. PROPOSED METHOD

A. RING OSCILLATOR

An on-chip Ring Oscillator (RO) is proposed, as shown in Figure 1, which automatically adapts the underlying technology and produces clock frequency in synchronization with a latch to meet the setup and hold time requirements of on-chip D Flip-flops. The 'Enable' signal at the gate input of the 1-bit latch makes the latch transparent such that the output of the latch going through the NOT gate causes the output of the latch to flip continuously as long as the gate input remains HIGH. The propagation delay of the latch and the NOT gate defines the time period of the clock with a 50% duty cycle.

The generated frequency can be slowed down further according to the requirement to drive on-chip sequential components in a design. The circuit that produces a single pulse after a programmable delay upon the 'Enable' signal's assertion is shown in figure 2. A high 'Enable' signal enables RO to produce clock pulses to drive the N-bit Shift-Left Register (SLR). The logical one is connected to the Least Significant Bit (LSB) of the SLR that serially shifts it left on each clock cycle until it reaches MSB. The 'Pulse' signal is taken from MSB as 'R(N)' of SLR. When the 'Enable' signal goes low, it clears the SLR asynchronously. The size of SLR is chosen as per the needed delay. Figure 3 shows post-layout simulation of pulse generator circuit for Xilinx XC7a100t-1csg324 FPGA.The generated pulse can be fed as a clock to any sequential component in a pipeline or generate glitch-free Req and Ack signals.

B. COMPLETION DETECTOR

Completion detection is a critical need of delay-adaptive asynchronous systems because it must indicate the completion of data execution by the logic blocks, and thus the results can be stored safely. The duration after which the output of a circuit becomes stable is determined by the propagation delay of gates, the wire lengths of interconnects, and the circuit complexity. As an example, in a Ripple Carry Adder (RCA), the carry from the least significant bit may continue to change the resultant bits for a delay that corresponds to its word length. The resultant sum, thus, cannot be latched safely till the most significant bit gets stable. The minimum delay occurs in many cases when no carry is involved [27]. The random delay is expressed by Log2 N, where N is the number of bits of RCA [28].

The operation of the RCA circuit reveals that the logic completion time of combinatorial circuits may also depend upon the data at its input, and it correlates with the transitions at its output. Based on this correlation, we propose a completion detector for a single-rail encoded datapath. It senses the transitions at its input and generates the completion signal if no transitions are detected for a finite amount of time.

C. THE WORKING PRINCIPLE OF THE CD

The working principle behind the presented CD is to evaluate the derivative of data at its input, which is nonzero when there are transitions and becomes zero when transitions are wiped out, i.e., when results become valid, as shown in equation 1. Here, $\frac{dD}{dt}$ denotes the rate of change of the input data. The equivalent circuit compares the incoming current data with its neighbouring preceding one, i.e., the delayed version, as shown in figure 4, where Δ denotes the delay.

The delayed version can be achieved by propagating data through a chain of even number inverters or a level-triggered latch whose enable is kept high. The comparator can be either logical AND of bitwise 2-input XNOR gates or a standard ALU in subtraction mode whose zero flag is the completion signal. During the transition period, the comparator's mismatch keeps its output low, and when the inputs become stable, the comparator's output goes high.

$$\frac{dD}{dt} = \begin{cases} \neq 0 & \text{if there are transitions in results} \\ 0 & \text{when results become stable} \end{cases} \tag{1}$$

D. THE BASIC CIRCUIT OF CD

As shown in figure 5, an N-bit CD employs DFFs from the configurable logic blocks of an FPGA as D-type level-triggered latches based on the concept discussed above. Such latches on a bus remain transparent to data as long as their gate signal remains active. The propagation delay of a latch separates input and output data. The inputs and outputs of an N-bit latch as two sets of inputs to a comparator that compares its inputs for equality. The output of the comparator will go low only for the duration of the mismatch between both of its inputs. The comparator's high output guarantees stability of data at the latch's input and, hence, detects completion of logic execution by the CD's preceding circuitry. This way, the comparator's output becomes a logic completion signal 'Done' and is asserted only when no transitions are detected within that delay.

E. VERIFICATION OF VALIDITY OF THE CD CONCEPT

To verify the validity of the presented concept, we connected the output of a 32-bit multiplier to the proposed CD. The completion signal 'Done' goes low for the duration when changes at the inputs of the multiplier produce transitions at its output. When the transitions at the output of the multiplier die out, and the result becomes valid, the 'Done' signal goes back to high, indicating that the multiplier has completed its execution. This can be seen from the post-route simulation viewgraph figure 6. This way the CD detects the execution completion by sensing data stability at its input. Thus, the high state of the 'Done' signal guarantees that the latch has stored only the valid data.

The time for which the 'Done' signal remains low depends upon the transition time of the output of the logic circuit connected to the CD, plus the propagation delays of the latch and the comparator of the CD, as shown in equation 2. Here, d and t denote the delay of the logic propagation and time, respectively. Since the latch and comparator work in parallel and get stable with transition stabilization in the incoming data; therefore, the time taken from the last transition till the assertion of the 'Done' signal is the overhead. If the output of the logic circuit is to be stored, then this overhead will be eliminated depending upon the application.

$$t_{Done} = t_{transitions} + d_{latch} + d_{comparator} \tag{2}$$

F. CD WITH HANDSHAKE SIGNALS

The complete CD with additional logic circuity that performs as a 2-phase sequence controller for pipeline processing is shown in figure 7. To understand the signaling sequence of the CD-based sequence controller in a pipeline, initially start with all the inputs and outputs signals of the CD at logic low. The Execution and Data Transfer (EDT) cycle begins with the flip

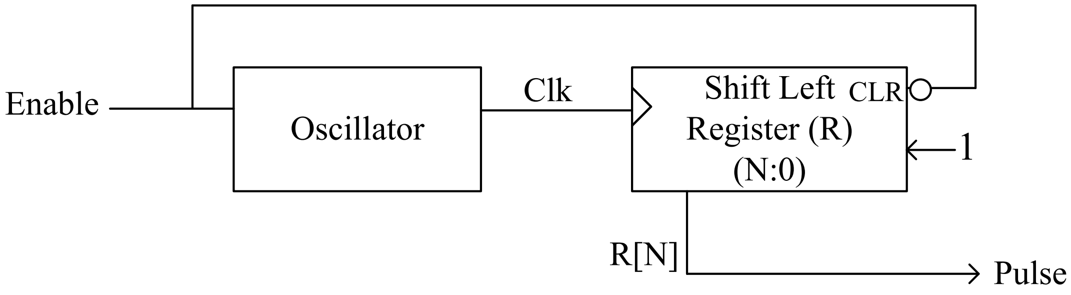

FIGURE 2. Programmable delay-element.

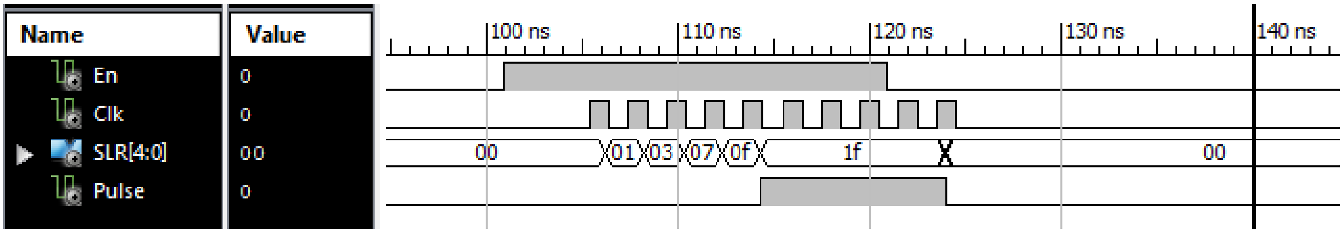

FIGURE 3. Post layout simulation of Pulse-generator on XC7A100t-1CSG324 FPGA.

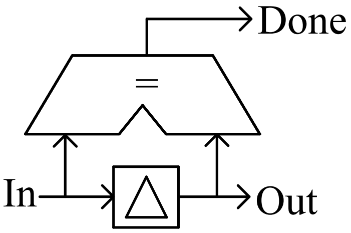

FIGURE 4. Basic concept of logic based completion detector.

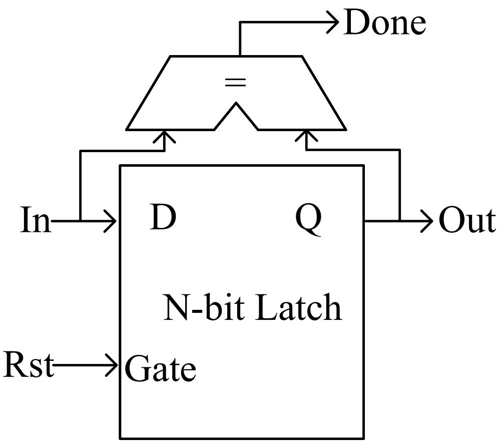

FIGURE 5. Basic completion detector.

of ‘Req_in’ signal. The XNOR gate compares the ‘Ack_in’ and ‘Ack_out’ signals for equality. Their same state turns the XNOR gate’s output high, making the N-bit latch transparent for the comparator to perform its function. The XOR1 gate compares the states of its inputs, ‘Req_in’ and ‘Ack_out’. The mismatch at the input of XOR1 gate produces high output, that enables the AND gate to propagate the output of the comparator to the DFF1.

When results at the N-bit latch input become transitions-free and valid, the comparator’s output goes high that through the AND gate triggers DFF1 to store a copy of ‘Req_in’ in it. This copy matches the XOR1 and XNOR gate’s input and mismatches the XOR2 gate’s inputs; thus, low outputs of XNOR and XOR1 gates disable the comparator’s output and make the N-bit latch opaque, respectively. At the same time, the mismatch caused at the inputs of XOR2 will make its output high, enabling the pulse generator to generate a clock pulse after a programmable delay. The pulse will trigger the DFF2 to save the copy of DFF1 and trigger any other sequential component.

The purpose of DFF1 is to generate ‘Ack_out’ signal for the preceding stage after storing valid data, and the purpose of DFF2 is to forward ‘Ack_out’ received through DFF1 as ‘Req_out’ to the proceeding stage after a programmable delay. The transition from the arrival of ‘Req_in’ to the generation of ‘Req_out’ signal completes EDT cycle. The new EDT cycle can now be initiated by flipping back the state of its ‘Req_in’. Following the same EDT cycle procedure, the states of its control signals will get restored after the second EDT cycle. This way, there are two EDT cycles in one cycle of control signals, i.e., 2-phase communication.

The CD can detect logic completion of any combinatorial or sequential circuit provided that the contamination delay of a circuit is less than the ‘Req_in’ to ‘Ack_out’ time of CD, as given in equation 3. The period between a combinatorial circuit’s input and the first transition on its output is referred to as contamination delay. This delay can also be calculated through post-route simulations if required. If the

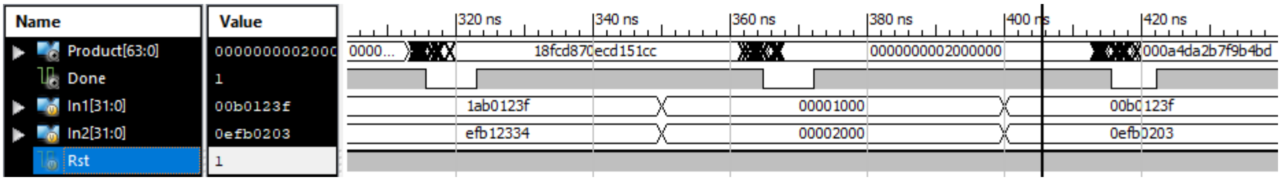

FIGURE 6. Post-layout simulation of completion detector on XC7A100t-1CSG324 FPGA.

first transition at the input of CD occurs after the generation of 'Ack_out', then the output of the comparator will go low again. However, the old data would be forwarded to the proceeding stage. It is observed that the subsystems, either generated through behavioral or gate-level Verilog HDL modeling techniques, satisfies equation 3, if they do not infer any embedded resources like BRAM and DSP blocks. However, in case of significant contamination delay caused due to these blocks, the programmable-delay element will cater to it.

$$d_{Contamination} < t_{Reqin-to-Ackout} \tag{3}$$

The operation of CD can be divided into two parts. One is the generation of Ack after receiving data, commencing the second part to generate Req for transmitting data onward. If incoming data is the same as stored in N-bit latch upon the assertion of Req_in, the minimum time taken by CD to generate 'Ack_out' is shown in equation 4. The d_{xor1} and d_{and} are the delays associated with xor1 & and gates, respectively whereas t_{su} and d_{cq} are the setup time and clock-to-q delay of DFF1, respectively.

$$t_{Reqin-to-Ackout} = d_{xor1} + d_{and} + t_{su} + d_{cq} \tag{4}$$

Since a gate logic is implemented in the Look-up Tables (LUT) inside FPGAs, the output of LUT goes through D Flip-flop for sequential logic and bypasses it for combinatorial logic. Therefore, the logic equivalent to xor1 & and gates will be implemented in LUT, and its output will follow the sequential path. Therefore, after technology mapping, the delay equation 4 will transform into equation 5. The d_{LUT} is the delay of single look-up table.

$$t_{Reqin-to-Ackout} = d_{LUT} + t_{su} + d_{cq} \tag{5}$$

Similarly, after technology mapping, the time required to generate 'Req_out' from the generation of 'Ack_out' with minimal size of SLR, i.e., 1-bit, is illustrated in equation 6. $T_{Osc-TimePeriod}$ is the time period of the frequency generated by RO. Here, the t_{su} and the d_{cq} are the setup and clock-to-q delay of DFF2, respectively.

$$t_{Ackout-to-Reqout} = d_{LUT} + t_{Osc-TimePeriod} + t_{su} + d_{cq} \tag{6}$$

The N-bit latch becomes transparent when the Ack_out and Ack_in become the same, permitting it to receive data without waiting for the assertion of Req_in. This way, a latch in a CD holds data until the receiver consumes it. The maximum delay will occur if the N-bit latch gets transparent along with the assertion of Req_in. In that case, the propagation delays of latch and comparator will also be included in equation 5 as illustrated in equation 7. The t_{su} and d_{cq} are the setup and clock-to-q delay of DFF1, respectively.

$$t_{Reqin-to-Ackout} = d_{LUT} + d_{latch} + d_{comparator} + t_{su} + t_{cq} \tag{7}$$

IV. ASYNCHRONOUS PIPELINE

The three-stage asynchronous pipeline model with 2-phase bundle data protocol, implementable in FPGAs, is shown in figure 8. An EDT cycle of the pipeline begins when data is placed at its input, and Req input is flipped. This flip, in turn, flips the output of its Muller-C element whose other input, i.e., Ack coming from the proceeding stage, was already matching with its Req input.

The output of Muller-C follows the state of its inputs when both become in the same logic state. The Muller-C 's output as 'Req_in' enables CD to receive new data and generate 'Ack_out' upon receipt of data and subsequently 'Req_out' after a programmable delay. Both 'Ack_out' and 'Req_out' will be in the same logic state as of 'Req_in'. Thus, 'Ack_out' and 'Req_out' will go to the preceding and proceeding CDs, respectively, allowing them to process new data. However, the latch in a CD will receive new data only if its proceeding CD has received its current data.

This way, in a pipeline, the 'Req_in' signal coming to a CD is generated by the Muller-C logic of the signals coming from both of its neighbouring stages, i.e., the 'Req_out' of the preceding stage (when it wants to send data) and 'Ack_out' of the proceeding stage (when it has saved data at its input), so that a CD can coordinate its operation with its neighbouring stages. Consequently, a proceeding stage does not allow data changes at its input, i.e., in the N-bit latch of its preceding stage, till it completes its execution with stable input and saves the valid result in its N-bit latch. Thus, a new EDT cycle can commence only when both the neighbouring stages will allow.This way, even and odd number CDs work alternately. Thus, all the stages of the pipeline safely process and save data.

The standalone operation of the asynchronous pipeline requires the generation of the control signals along with the processing. The inversion of Ack from the second stage through AND gate goes as a request to the first stage. The second input of AND gate is the global reset signal (Rst) which is active Low. The last stage CD's 'Req_out' will be feed as 'Ack_in' to the same CD.

A. FPGA IMPLEMENTATION OF ASYNCHRONOUS PIPELINE

In an asynchronous pipeline, each stage runs at its own pace with its local clock. The clock is not free running but toggles

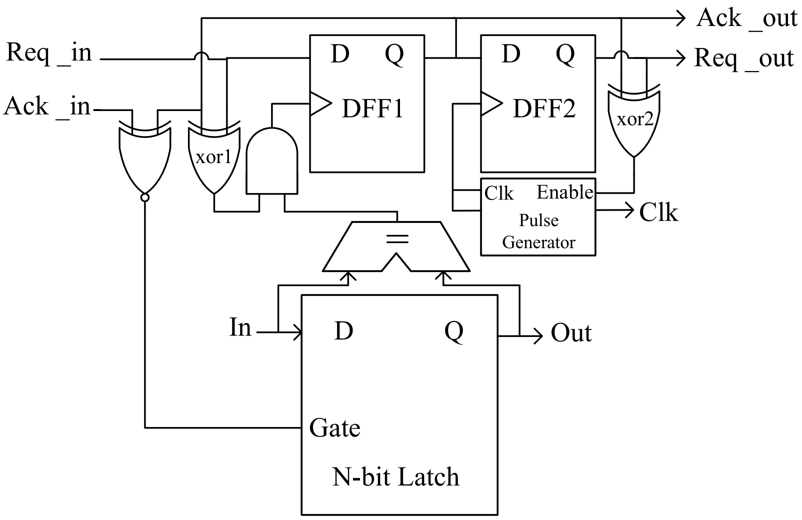

FIGURE 7. Completion detector circuit with handshake signals.

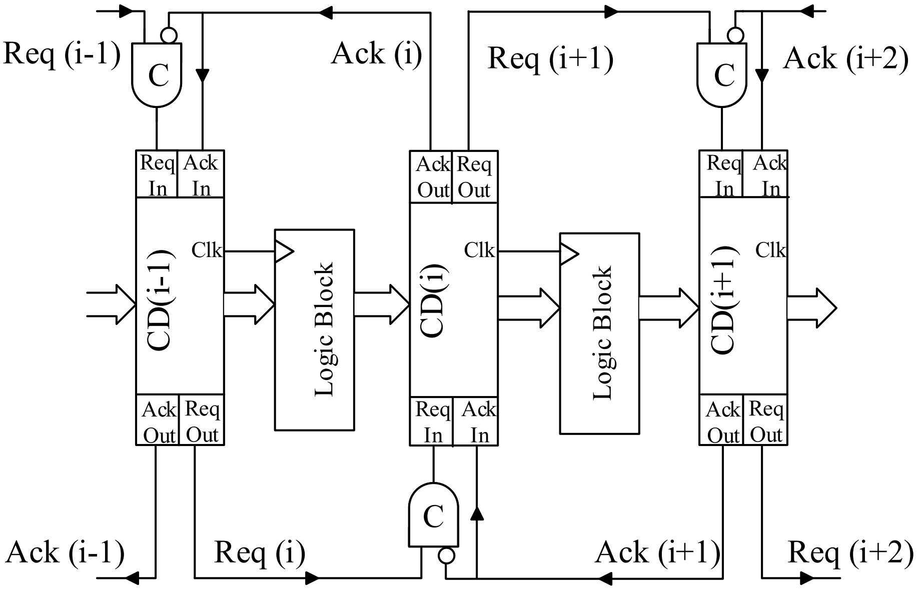

FIGURE 8. Asynchronous pipeline model.

only when needed. A pipeline stage may include FPGA's embedded resources, e.g., BRAM or DSP blocks which may be located at some distance within the FPGA, and thus the interconnect delay may slow down the pipeline. For proper functioning of the pipeline, in such cases, a pulse generator may be needed at the output of the CD to provide the required delay. The interconnect delays from CD to some sequential blocks are controlled by applying the timing constraint on the clock net. The FPGA's PAR tool reports clock timing based on interconnect delays from CD to sequential logic block. This information can be obtained in different ways. One way to calculate the clock delay is by synthesizing each stage separately. In a static timing report, source rise to destination rise time under clock-to-setup time requirement is divided by the oscillator's cycle time to get the size of SLR. This approach of calculating delay is hectic. Another way of obtaining delay information is by applying timing constraints on all the internal clock nets together in a design and read the timing report to check if the timing constraints have been met after the PAR process. On compliance, the delay information from the timing report is used to define the size of the SLR. The latter method is reliable; therefore, it should be adopted in defining the size of the SLR.

V. ASYNCHRONOUS PIPELINED RISC PROCESSOR

A tri-operand, five-stage pipelined RISC processor based on the above-proposed concepts is shown in figure 10. This processor supports a single type of instruction format, as shown in figure 9. There are 16 32-bit registers, so there is a need for a 4-bit address, with Rs, Rt, and Rd being the source,

TABLE 1. Contents of data and code memories.

Address	Data Memory	Code Memory	Instructions	RTL
0	0000000A	00000000	Add R0,R0,R0	R0 ← R0 + R0
1	0000000B	70010001	Lw R1,(R0)1	R1 ← Data Memory[R0 + 1]
2	0000000C	70020002	Lw R2,(R0)2	R2 ← Data Memory[R0 + 2]
3	0000000D	80030003	Li R3,3	R3 ← 3
4	0000000E	80040004	Li R4,4	R4 ← 4
5	0000000F	80050005	Li R5,5	R5 ← 5
5	0000000F	80060006	Li R6,6	R6 ← 6
6	00000012	01270000	Add R7,R1,R2	R7 ← R1 + R2
8	00000000	00000000	Add R0,R0,R0	R0 ← R0 + R0
9	00000000	00000000	Add R0,R0,R0	R0 ← R0 + R0

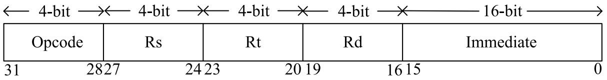

FIGURE 9. Instruction format.

target, and destination register operands, respectively. The 'Immediate' field is a 16-bit signed number. The 4-bit opcode allows 16 different instructions in the instruction set architecture. The positions of the operands in the instructions are fixed. A 32-bit instruction is fetched from the instruction memory. The Rs and Rt operands are read from the Register File (RF) in the decode phase. ALU operations are performed on Rs, Rt, and Immediate operands in the execute stage depending upon the opcode. The ALU output is fed as an address to the data memory in the case of load and store instructions, otherwise bypassed to the write-back stage where it is written in RF. Inspiration for this micro-architecture is taken from the MIPS architecture [29].

The CD with a pulse generator takes care of the setup and hold time requirements of register file, instruction, and data memories which require clock signals.The contents of memories are shown in the table 1. The addresses are listed in the first column, while the contents of the data and code memories are shown in the second and third columns, respectively. The assembly language representations of the instructions in the third column are in the fourth column. The Register Transfer Language (RTL) of these instructions is represented in the fifth column. The post-layout simulation of the RISC processor executing these instructions is shown in figure 11.

VI. RESULTS AND DISCUSSION

Apart from asynchronous implementation with adaptive delays, we also implemented as part of this research for comparison, synchronous and asynchronous versions with fixed delays of the RISC machine on the XC7A100t-1CSG324 FPGA device using ISE 14.7 with auto PAR tools. The same pipeline model was used in both asynchronous implementations except that the comparators in CDs were replaced with fixed delays. Fixed delays are implemented by inserting LUT chains in the Req and clock lines based on the delay information obtained from the PAR timing analysis report. Timing constraints were specified to optimize design routing. The optimization goal was set to speed rather than area, and the effort level was set to high. All of the three designs are implemented using the same parameters for comparisons.

The area, speed, and power dissipation parameters for asynchronous and synchronous systems are listed in Table 2. It was observed that asynchronous implementation with CD requires 85% more hardware than synchronous design due to the addition of comparators and sequencers for the handshaking protocol. Whereas the asynchronous implementation with predefined delays requires 82.5% more hardware than the synchronous design due to the sequencers and chains of LUTs on the clock and Req lines.

For comparison purposes, to execute a set of ten instructions, the synchronous processor took 15 cycles, each of 7.28ns, i.e., a time of 109.2ns, while both the asynchronous versions with adaptive delays and predefined delays required 106ns and 109ns respectively to execute the same set of instructions, with average clock cycle times of 106ns/15 = 7.06ns and 107ns/15 = 7.26ns. The asynchronous implementation with adaptive delays exhibits 3.0% better speed than its synchronous version due to the relaxed timing offered by CD. However, the predefined delays approach could not achieve the speed benefit due to fixed nature of delays. Unlike synchronous and asynchronous systems with predefined delays, the cycle time of an asynchronous system with adaptive delays is data-dependent.

It was also observed that both asynchronous implementations dissipate more dynamic power than their synchronous counterpart due to extra hardware and better speed. The absence of clock trees in asynchronous systems conserves dynamic power [30]. We used clock trees for clk1, clk2, and clk3 signals to meet timing constraints. Each sequential subsystem runs at its own pace using a dedicated clock, which can be gated to save power consumption [31]. However, all the clock signals are non-gated and toggle upon the arrival of request signal.

We used the same methods to calculate the speed and power dissipation for both asynchronous and synchronous designs. To conclude which design is better, a power-delay product [32] as a unified performance matric, where a lower value is better, is calculated and given in the Table2, which shows that asynchronous design with adaptive delays exhibits relatively better performance.

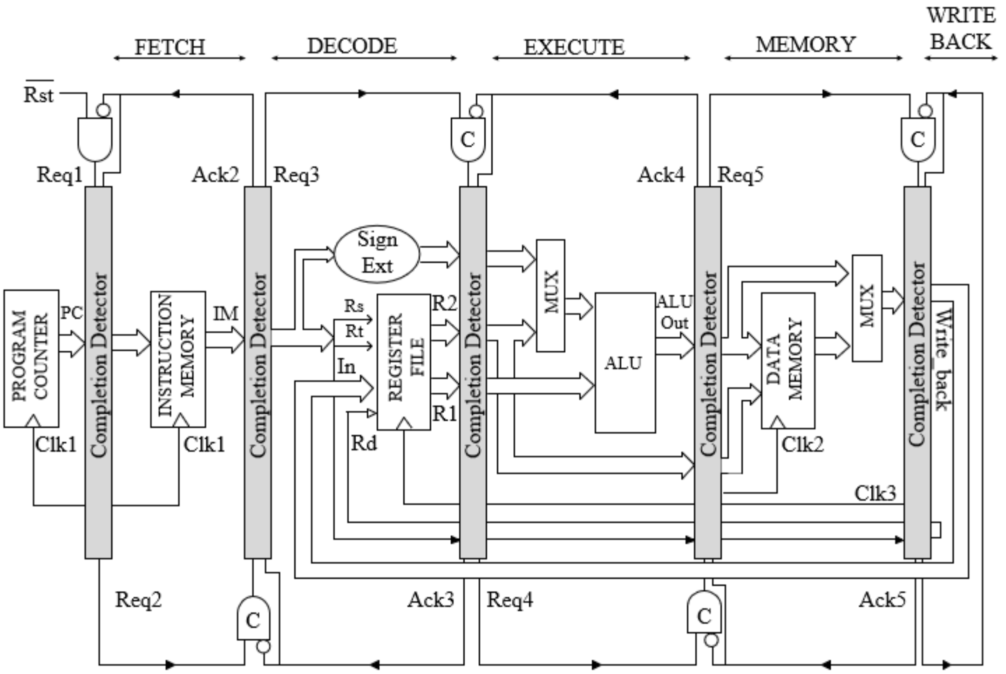

FIGURE 10. Asynchronous pipelined RISC processor.

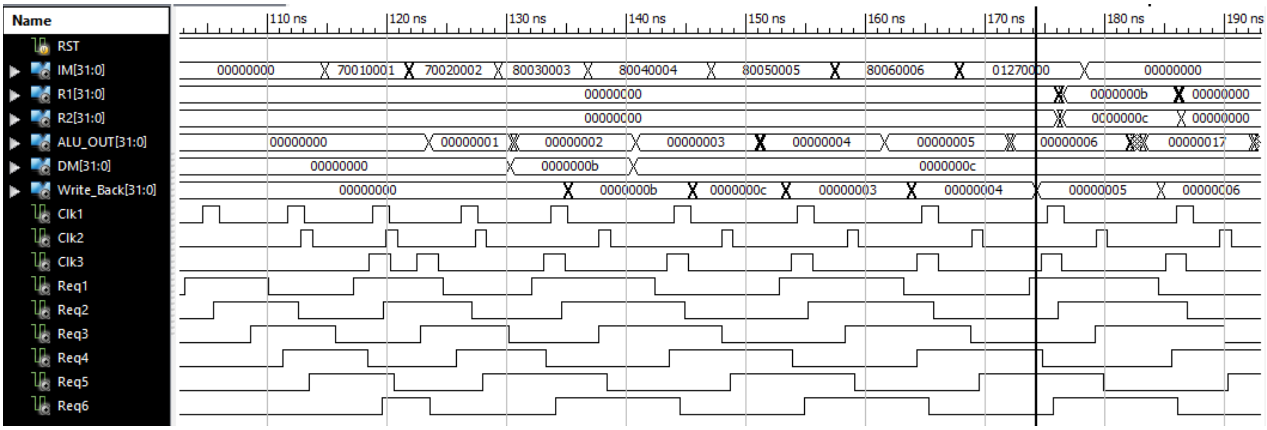

FIGURE 11. Post-layout simulation of asynchronous RISC processor on XC7A100t-1CSG324 FPGA.

TABLE 2. Power and performance comparison of conventional and adaptive pipeline.

Design	Cycle Time (nS)	Power (mW)	Area(Slices)	Power x Delay
Synchronous Pipeline	7.28	102	80	743
Asynchronous Pipeline with Fixed Delays	7.26	105	146	762
Asynchronous Pipeline with Adaptive Delays	7.06	105	148	741

Due to the adaptive nature of a ring oscillator, it adapts to the delays of the underlying technology. Designers must reconsider time constraints on clock nets for a design to work during the migration of FPGA technology platforms. Whereas, in the adaptive-delay-based (CD approach) asynchronous pipeline, only clock nets are to be adjusted for constraints. In the fixed delay-based pipeline, however, all requests and clock nets must be adjusted accordingly. Otherwise, timing violations may occur. It has been observed that the adaptive-delay-based pipeline exhibits smoother functionality upon technology migration due to self-adjustments of delays.

VII. CONCLUSION

It has been shown that the proposed completion detector delays the generation of the acknowledge signal until the transitions in the incoming data are entirely wiped out in response to the request signal, thus acting as an adaptive-delay element, which eliminates the need to insert any static delay-element at the pre-synthesis phase. The proposed pulse generator makes the completion detector robust and provides clock pulses to various sequential components and FPGA's embedded resources. The proposed asynchronous pipeline model lets designers migrate from synchronous to asynchronous domain quickly and reliably without worrying about modeling post-synthesis delays, a cumbersome job, and auto place-and-route tools to place designs without maintaining ratios between control and datapath.

REFERENCES

[1] N. Van Toan, D. M. Tung, and J.-G. Lee, "Measurements of metastability in MUTEX on an FPGA," *IEICE Electron. Exp.*, vol. 15, no. 1, 2018, Art. no. 20171165.

[2] Z. Tabassam, S. R. Naqvi, T. Akram, M. Alhussein, K. Aurangzeb, and S. A. Haider, "Towards designing asynchronous microprocessors: From specification to tape-out," *IEEE Access*, vol. 7, pp. 33978–34003, 2019.

[3] G. A. Abdelmalek, R. Ziani, and R. Mokdad, "Security and fault tolerance evaluation of TMR–QDI circuits," *IET Inf. Secur.*, vol. 13, no. 3, pp. 213–222, May 2019.

[4] P. BrieF, "Speedster FPGA family," Achronix Semicond., San Jose, CA, USA, Tech. Rep. PB001, 2008.

[5] T. Sueyoshi, "Basic knowledge to understand FPGAs," in *Principles and Structures of FPGAs*. Singapore: Springer, 2018, pp. 1–22.

[6] A. He, G. Feng, Z. Jilin, and J. Wu, "An asynchronous mesh NoC based booth multiplication," *IET Circuits, Devices Syst.*, vol. 13, no. 1, pp. 73–78, Jan. 2019.

[7] J. Spars and S. Furber, *Principles Asynchronous Circuit Design*. Boston, MA, USA: Springer, 2002.

[8] K. Bhardwaj and S. M. Nowick, "A continuous-time replication strategy for efficient multicast in asynchronous NoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 2, pp. 350–363, Feb. 2019.

[9] S. M. Nowick and M. Singh, "Asynchronous design—Part 1: Overview and recent advances," *IEEE Des. Test*, vol. 32, no. 3, pp. 5–18, Jun. 2015.

[10] Y. Zhang, J. Li, H. Cheng, H. Zha, J. Draper, and P. A. Beerel, "Yield modelling and analysis of bundled data and ring-oscillator based designs," *IET Comput. Digit. Techn.*, vol. 13, no. 3, pp. 262–272, May 2019.

[11] R. K. Kavitha, A. Khajamastan, Y. R. Akhilesh, and B. Venkataramani, "High-performance asynchronous pipeline using embedded delay element," *Microprocessors Microsyst.*, vol. 73, Mar. 2020, Art. no. 102955.

[12] A. Motaqi, M. Helaoui, S. AghliMoghaddam, and M. R. Mosavi, "Detailed implementation of asynchronous circuits on commercial FPGAs," *Anal. Integr. Circuits Signal Process.*, vol. 103, no. 3, pp. 375–389, Jun. 2020.

[13] A. Wheeldon, J. Morris, D. Sokolov, and A. Yakovlev, "Self-timed, minimum latency circuits for the Internet of Things," *Integration*, vol. 69, pp. 138–146, Nov. 2019.

[14] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[15] M. Singh and S. Nowick, "MOUSETRAP: High-speed transition-signaling asynchronous pipelines," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 6, pp. 684–698, Jun. 2007.

[16] I. Sutherland and S. Fairbanks, "GasP: A minimal FIFO control," in *Proc. 7th Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Mar. 2001, pp. 46–53.

[17] T. E. Williams, "Self-timed rings and their application to division," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, USA, 1991.

[18] M. Singh and S. M. Nowick, "The design of high-performance dynamic asynchronous pipelines: Lookahead style," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 11, pp. 1256–1269, Nov. 2007.

[19] O. A. Izosimov, I. I. Shagurin, and V. V. Tsylyov, "Physical approach to CMOS module self-timing," *Electron. Lett.*, vol. 26, no. 22, pp. 835–836, Oct. 1990.

[20] M. E. Dean, D. L. Dill, and M. Horowitz, "Self-timed logic using current-sensing completion detection (CSCD)," *J. VLSI Signal Process. Syst. Signal, Image Video Technol.*, vol. 7, nos. 1–2, pp. 7–16, Feb. 1994.

[21] E. Nigussie, S. Tuuna, J. Plosila, P. Liljeberg, J. Isoaho, and H. Tenhunen, "Boosting performance of self-timed delay-insensitive bit parallel on-chip interconnects," *IET Circuits, Devices Syst.*, vol. 5, no. 6, pp. 505–517, 2011.

[22] L. Nagy, V. Stopjaková, and J. Brenkuš, "Current sensing completion detection in single-rail asynchronous systems," *Comput. Informat.*, vol. 33, no. 5, pp. 1116–1138, 2014.

[23] A. Changela, M. Zaveri, and A. Lakhlani, "FPGA implementation of asynchronous mousetrap pipelined Radix-2 CORDIC algorithm," in *Proc. Int. Conf. Current Trends towards Converging Technol. (ICCTCT)*, Mar. 2018, pp. 252–258.

[24] A. Ejnioui, "FPGA prototyping of a two-phase self-oscillating micropipeline," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Mar. 2007, pp. 437–438.

[25] D. L. Oliveira, K. Garcia, and R. d'Amore, "Using FPGAs to implement asynchronous pipelines," in *Proc. IEEE 5th Latin Amer. Symp. Circuits Syst.*, Feb. 2014, pp. 1–4.

[26] Z. Li, Y. Huang, L. Tian, R. Zhu, S. Xiao, and Z. Yu, "A low-power asynchronous RISC-V processor with propagated timing constraints method," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 9, pp. 3153–3157, Sep. 2021.

[27] C. H. van Berkel, M. B. Josephs, and S. M. Nowick, "Applications of asynchronous circuits," *Proc. IEEE*, vol. 87, no. 2, pp. 223–233, Feb. 1999.

[28] H. Jiang, L. Liu, F. Lombardi, and J. Han, "Approximate arithmetic circuits: Design and evaluation," in *Approximate Circuits*. Cham, Switzerland: Springer, 2019, pp. 67–98.

[29] D. A. Patterson and J. L. Hennessy, "Computer organization and design: The hardware/software," in *Computer Organization and Design, The Hardware/Software Interface*, 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, Aug. 2004.

[30] A. Samiee, Y. Sun, R. F. Demara, Y. Choi, and Y. Bai, "Energy efficient mobile service computing with differential Spintronic-C-elements: A logic-in-memory asynchronous computing paradigm," *IEEE Access*, vol. 7, pp. 55851–55860, 2019.

[31] J. Kathuria, M. Ayoubkhan, and A. Noor, "A review of clock gating techniques," *MIT Int. J. Electron. Commun. Eng.*, vol. 1, no. 2, pp. 106–114, 2011.

[32] Z. Han, "The power-delay product and its implication to CMOS inverter," *J. Phys., Conf. Ser.*, vol. 1754, no. 1, Feb. 2021, Art. no. 012131.

ADNAN GHAFOOR received the B.S. degree in computer engineering from the University of Arid Agriculture, Rawalpindi, Pakistan, in 2006, and the M.S. and Ph.D. degrees in electronic engineering from International Islamic University, Islamabad, Pakistan, in 2010 and 2019, respectively. He is currently working as an Assistant Professor with the University of Central Punjab, Lahore, Pakistan. His research interests include reconfigurable asynchronous digital systems and hardware realization of various communication and signal/image processing algorithms.

MUHAMMAD WAQAR MUGHAL received the bachelor's and master's degrees from the University of Central Punjab, Lahore, Pakistan. He has over four years of experience in academia. He is currently working as a Lecturer with the University of Central Punjab. His research interests include machine learning, computer vision, image processing, and mobile application.

ARBAB A. KHAN received the M.Sc., M.Phil., and Ph.D. degrees from the Department of Electronics, Quaid-i-Azam University, Islamabad, Pakistan, in 1989, 1993, and 1998, respectively. He is currently an Adjunct Faculty Member with the Faculty of Engineering and Technology, International Islamic University, Islamabad. He was a first recipient of the joint award ICO/ICTP by the International Commission for Optics and The Abdus Salam International Centre for Theoretical Physics, in 2000.