ISSN: 2976-7466

Journal of Biosensors and Bioelectronics Research

SCIENTIFIC Research and Community

Review Article

Open Access

Self-Locking Domino Logic Pipelines: Application in RISC-V Architectures in FPGA

Florian Deeg*, Xingzhou Chen and Sebastian M Sattler

Chair of Reliable Circuits and Systems Friedrich-Alexander-University Erlangen-Nuremberg Paul-Gordan-Str. 5, 91052 Erlangen, Germany

ABSTRACT

This paper presents the design and implementation of a self-locking domino logic pipeline controller for a RISC-V processor implemented on an FPGA. The emphasis is on asynchronous circuit design, which offers advantages such as enhanced resilience to supply voltage fluctuations, optimized power efficiency, and the elimination of clock-related issues such as skew and single-point failures. By leveraging the asynchronous Globally Asynchronous Locally Synchronous (GALS) systems and domino logic, the controller ensures hazard-free operation while maintaining race-free processing. The asynchronous approach, integrated into a 32- bit RISC-V processor, allows for flexible and energy-efficient operation, thereby demonstrating its potential for performance-critical applications. This paper high- lights the contrasts between the asynchronous design and the traditional synchronous multicycle processor, demonstrating the benefits of asynchronous systems in terms of power consumption and performance. A significant contribution of this design is the pipeline's completion detection mechanism, which ensures that each processing stage locks until valid results are obtained, thereby markedly enhancing system stability. Furthermore, the paper investigates the parallelization of domino gates and introduces an asynchronous Arithmetic Logic Unit (ALU), which further optimizes performance through self-locking mechanisms. The power, performance, and area (PPA) analysis of the design demonstrates considerable improvements in throughput (up to 10%) and reduced latency per instruction in comparison to its synchronous counterpart, while maintaining moderate resource utilization on an FPGA. The results indicate that asynchronous domino logic pipelines may offer a promising approach for achieving energy-efficient and high-performance processors in future computing architectures.

***Corresponding author**
Florian Deeg, Chair of Reliable Circuits and Systems Friedrich-Alexander-University Erlangen-Nuremberg Paul-Gordan-Str. 5, 91052 Erlangen, Germany.

Received: October 21, 2024; **Accepted:** October 25, 2024; **Published:** November 04, 2024

Keywords: Domino Logic, Asynchronous Design, Self-Locking, Risc-V, GALS, FPGA

Introduction

Synchronous circuits represent the pinnacle of circuit design innovation. Nevertheless, asynchronous circuits are becoming increasingly significant in the field of circuit design, offering a multitude of advantages over their synchronous counterparts. These include enhanced performance, reduced power consumption, increased modularity, the absence of a single point of failure, and the elimination of clock skew, among others. For further information, please refer to [1]. Furthermore, asynchronous circuits demonstrate enhanced resilience to fluctuations in supply voltage and temperature. In asynchronous designs, local faults are often constrained to the affected area, thereby enhancing fault tolerance. Moreover, they generate less electromagnetic interference and are therefore more suitable for applications in which electromagnetic compatibility is a significant consideration [2]. However, there are also some disadvantages to this approach, including the necessity for more complex design methods and an associated lack of design tools. FPGAs are a special type of hardware component distinguished by their high performance, flexibility, and energy efficiency. In contrast to conventional integrated circuits, which are pre-programmed for a specific function, FPGAs can be reconfigured after manufacture to undertake new tasks or optimize performance. This feature renders them an optimal platform for the development of demanding applications that require high computing power, low latency, and customizability. For a considerable period, the market for processors was divided between two architectures: x86 and ARM, which are mainly used in mobile devices. In recent years, however, a new contender has emerged, offering a novel approach in the form of RISC-V. RISC-V is a license-free instruction set architecture (ISA) that originated at the University of California, Berkeley [3]. In contrast to the x86 architecture, which has evolved over time and is characterized by a high degree of complexity, the RISC-V architecture was developed from scratch. The principle of simplicity was given precedence. The objective of this simplicity is twofold: firstly, to reduce the cost of hardware, and secondly, to enhance flexibility. RISC-V is becoming an increasingly significant player in the field of processors [4]. A significant advantage is that it is not subject to licensing restrictions. This allows a multitude of companies and research groups to develop and utilize processors based on RISC-V. This has resulted in the emergence of a diverse range of RISC-V processors, which are employed in a multitude of applications. The spectrum encompasses a range of devices, from those designed for energy efficiency in the Internet of Things (IoT) to high-performance computers. The simplicity, flexibility, and license-free nature of RISC-V make it an attractive option for many developers. Other positive aspects of RISC-V include its energy efficiency, scalability, and security, as the basic architecture of RISC-V is so simple and offers little scope for attack.

Structure of the Paper

A brief review of the literature is presented to distinguish this paper from others in the field. The following section presents the circuit structure, which comprises the self-locking pulse circuit, the dual-rail domino logic circuit, and the entire pipeline with completion detection and its realization in the FPGA. Subsequently, an existing synchronous multicycle RISC-V processor is introduced, after which a control automaton for this Turing-complete processor is realized as a domino logic pipeline. The pipeline is demonstrated as a means of controlling a GALS system, which can be divided into subcircuits at will in order to achieve the highest possible speed and safety. The subsequent chapter presents the results and offers a comparison with synchronous automata. Subsequently, we will present a blueprint for a completely asynchronous central processing unit (CPU). In conclusion, the potential for future work in this area is discussed.

Related Work

In their study, presented a method for optimizing the clocking process in self-resetting domino pipelines [5, 6]. This method employs the use of soft synchronizers and roadblocks to facilitate time borrowing, thereby maximizing throughput and eliminating latch overhead. The authors introduced a high-performance clocking methodology for self-resetting domino pipelines that optimizes the clock rate through time borrowing and robust handling of clock skew while eliminating latch overhead. However, their approach does not sufficiently simplify the complex clocking and synchronization management, nor does it provide a robust precharge management system. Furthermore, it does not provide a streamlined implementation and testing methodology.

In the studies a method for optimizing the clocking process in self-resetting domino pipelines was presented [5, 6]. This approach employs soft synchronizers and roadblocks to facilitate time borrowing, thereby maximizing through- put and eliminating latch overhead. The authors introduced a high-performance clocking methodology that optimizes the clock rate through time borrowing and robust handling of clock skew while removing latch overhead. However, their approach does not adequately simplify the complex management of clocking and synchronization, nor does it provide a robust precharge management system. Additionally, it lacks a streamlined implementation and testing methodology.

In a high-speed add-compare-select unit for Viterbi decoders using locally self-resetting CMOS was proposed [7]. This design achieves significantly higher data rates in comparison to static and domino CMOS designs. However, it is associated with increased power consumption and design complexity due to the necessity of careful device sizing and additional components.

In contrast, the authors of introduced a dual keeper structure and delay logic gates to enhance the performance and noise margin of domino logic gates, ensuring high-speed switching and robustness to noise and timing variations [8]. However, their approach introduces additional design complexity and does not sufficiently address scalability issues.

In their work they developed a new type of logic gate with input disable functionality for use in rapid and power-efficient arithmetic operations [9]. They demonstrated the application of these gates in a 16-bit parallel adder. However, their work primarily focuses on the development of new arithmetic circuits and does not extensively validate the logic in broader applications or address the issue of implementation complexity.

In two novel synchronization approaches for clockless pipelining of coarse grain datapaths using self-resetting stage logic were proposed as a means of achieving high throughput. However, these approaches are not scalable and are associated with increased implementation complexity [10]. In their seminal work they introduced the Self-Resetting Logic with Gate Diffusion Input technique, which enables the creation of low-power, high-speed logic circuits [11]. They demonstrated the effectiveness of this technique through the design and simulation of various adders. However, their approach results in an increased number of transistors and a more complex design.

In their study a method for designing high-throughput and ultra-low-power asynchronous domino logic pipelines based on a constructed critical data path was presented [12]. However, their approach does not fully address the challenges of design automation, placement, routing optimization, and timing verification. The implementation of low-power and high-performance asynchronous dual-rail interconnect using domino logic gates in 16-nm technology was proposed in [13]. The integration of self-locking mechanisms or the detailed implementation of a complete RISC-V pipeline controller remains an open issue. In a novel framework for automating the design of asynchronous logic control in AMS electronics was discussed, integrating formal verification and specialized analog-to-asynchronous interface components for handling non-persistent signals [14]. However, this framework does not comprehensively address the challenges of design automation and efficient handling of non-persistent signals within FPGA implementations.

In their study the authors presented a methodology for implementing asynchronous phase-decoupled circuits using traditional electronic design automation (EDA) tools. The authors demonstrated the implementation of an asynchronous RISC-V processor on the Xilinx ZCU102 FPGA, achieving a threefold improvement in dynamic power efficiency compared to its synchronous counter-part while maintaining similar resource utilization [15]. This approach illustrates the potential of asynchronous design in reducing power consumption for IoT and neuromorphic applications, despite the challenges in commercial tool support. This work builds upon the findings presented in [16, 17].

Self-Locking Domino Logic

This section presents the structure and realization of the self-locking domino logic in the FPGA. The delay-insensitive domino logic was selected to minimize constraints in the design process while maintaining hazard-free and race-free operations. This approach contrasts with one-step designs where complex algorithms are employed to construct the automaton without clocking [18]. The programming in the FPGA occurs at the lowest level of abstraction to ensure that the structure is built in the same way, without the software attempting to optimize the structure. This is because the synchronous optimization process is used to build the structure. The asynchronous design cannot be simulated, so it must be built in accordance with the structure and verified with tests. This is to ensure that any known error models are excluded. The structural comparison of domino logic on the FPGA was conducted in [17]. This section will subsequently discuss the individual realizations in the FPGA at the low level.

Globally Asynchronous Locally Synchronous (GALS)
GALS is a design methodology for electronic circuits. It addresses the challenge of ensuring safe and reliable data transfer between independent clock domains within a system. A GALS system breaks down the circuit into independent blocks, each with its own local clock. These blocks communicate with each other asynchronously using handshaking protocols [19]. This allows for flexibility because blocks can operate at different speeds based on their needs, and scalability because the system can be easily expanded without worrying about the global clock. Furthermore, the GALS methodology results in reduced power consumption, as only active blocks are clocked, thereby increasing the system's power efficiency.

Asynchronous Handshake Protocol: An asynchronous handshake protocol represents the communication agreement between two or more entities, allowing them to exchange data without the necessity of a common clock [20]. This is in contrast to synchronous protocols, which rely on the timing of a common clock to regulate communication. In contrast to synchronous protocols, which rely on the timing of a common clock to regulate communication, asynchronous handshake protocols employ a pair of signals to regulate data transmission. The initial signal is used to initiate the transmission of data (REQ), while the subsequent signal is utilized to confirm the successful completion of the data transmission (ACK). Firstly, the four-phase protocol, as outlined in, is elucidated through the lens of a hypothetical goods purchase, see Figure 1 [1]. The process commences with the opening of the channel by B, who then makes an offer by setting acknowledgement (ack) to a low value. Subsequently, the customer (A) wishes to place an order and does so by setting the request (req) parameter to a high value. Following this, the supplier (B) generates the invoice. The delivery process then commences with the shipment of the goods and the setting of the ack parameter to a high value. The customer (A) then receives the goods and resets the request parameter to a low value. The supplier (B) then creates the receipt. Once the supplier (B) has received the payment, it acknowledges the process, thereby allowing a new order process to begin.

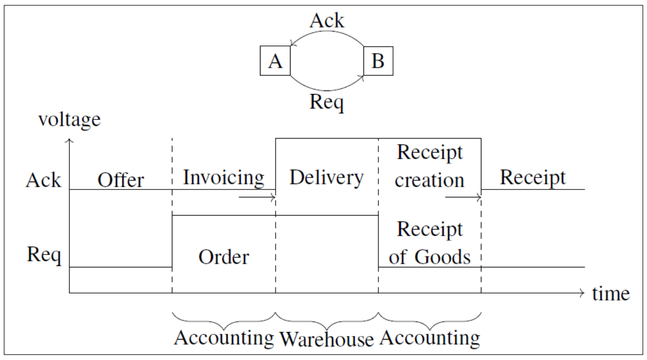

Figure 1: 4-Phase Handshaking Protocol time

Pulse Circuit
The objective of self-locking is to facilitate the system's re-accessibility subsequent to a single traversal of the circuit branches and their subsequent establishment in a valid state. The Lookup table (LUT) structure input pulse circuit, which serves to lock the input, can be observed in Figure 2, its table in Table I, and the signal flow graph in Figure 3. The equation for Q is listed in

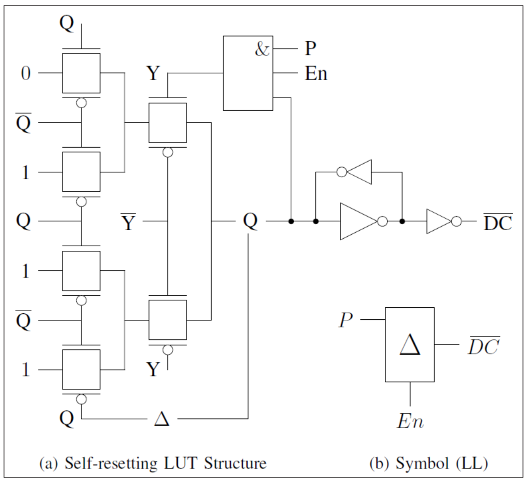

Figure 2: Pulse Circuit for Self-locking and Duty Cycle

Table I: Truth Table of Pulse Circuit

Δ(Q)	Y	Q	Comment
0	0	1	Switch
0	1	1	Switch
1	0	1	Hold
1	1	0	Switch

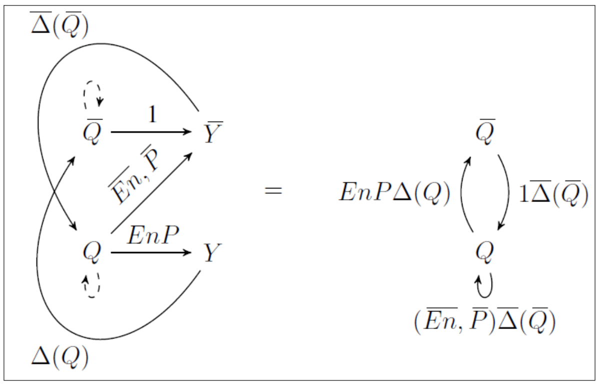

Figure 3: Signal Flow Graph of Pulse Circuit

Equation 1:

$$Q \Leftarrow \bar{Q}1\bar{\Delta}(\bar{Q}), Q(\bar{E}, \bar{P})\bar{\Delta}(\bar{Q}) \qquad (1)$$
$$\bar{Q} \Leftarrow QEnP\Delta(Q) \qquad (2)$$

The digital timing diagram can be found in Figure 4. The self-resetting input pulse circuit is utilized for self-locking, whereby the input is directly locked following an initial pulse req. Subsequently, a precharge phase for the domino logic is initiated by the circuit's self-resetting feedback, which subsequently disables the input. The propagation delay, denoted as τΔ, of the self-reset circuit determines the length of the precharge phase. It is thus imperative to guarantee that the precharge phase is sufficiently prolonged to ensure that all internal nodes are pulled to VDD. Once this has been achieved, the dual-rail domino logic (DRDL) gates have no disjunctive outputs and subsequently trigger the evaluation phase after the system has self-reset, which occurs after a delay of τΔ. Subsequently, the rising edge of $\overline{dc}$ initiates the transfer of states and input signals to a D-FF at the input, where they are stabilized until the next evaluation phase. The circuit thus blocks the input, generates a duty cycle, and ensures stable signals during the evaluation step. Once the subsequent block is complete, it will set an enable signal to 1 (ACK) and unlock the input once more.

Domino Logic

The domino logic family is an asynchronous logic family based on the principle of the domino effect, as described in [21]. The domino effect describes the sequence of events that occurs when one domino falls and causes the subsequent domino in the sequence to fall as well. In domino logic, these effects are employed for the transmission of data through a switching network. The general mode of operation of a domino logic gate can be divided into two phases: precharge and evaluate. A domino gate represents the fundamental unit of construction in domino logic. It is composed of two transistor circuits, one for the pull-up phase and one for the pull-down phase, which are integrated into a single unit. For an illustrative example, see Figure 5, which depicts an AND2 single-rail domino gate with a keeper on transistor level (TL).

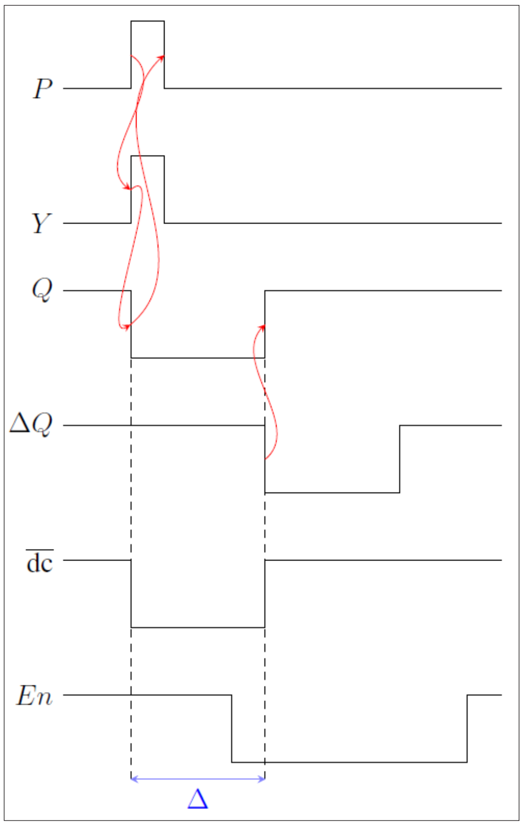

Figure 4: Digital Timing Diagram

In the precharge phase, the inner node is charged to VDD, and the logic state after the inverter is 0. Upon transitioning to the evaluate phase, namely when the duty cycle switches from 0 to 1, the node is pulled to ground (GND) when the pull-down is active (i.e., when the equation is fulfilled) and logic 1 is present at the output. It is now possible to connect Domino logic gates in series and have them propagate through the pipeline. In light of the recognition of the domino logic inherent to the FPGA, a cursory examination of the structural viability of the aforementioned logic is warranted. To this end, the logic has been mapped to a multiplexer (MUX) structure of pass transistors, which realizes a LUT, as shown in Figure 6 [22].

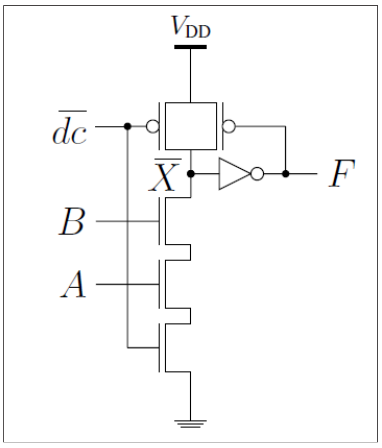

Figure 5: Single Rail Domino Logic on Transistor Level

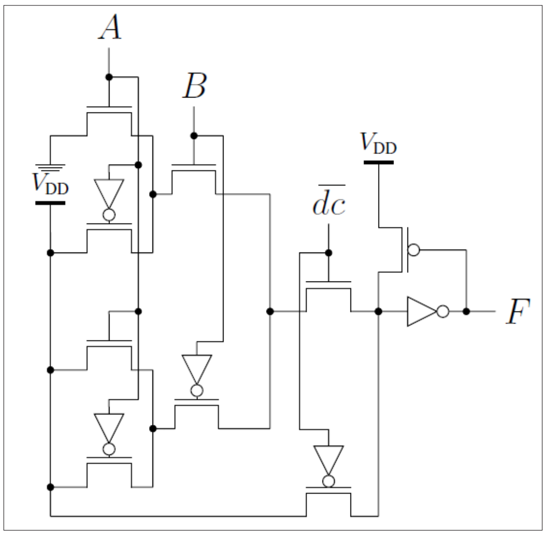

Figure 6: Single-Rail Domino Logic mapped to LUT3

As the lower path for $\overline{dc} = 0$ is to charge the inverter, all assignments are mapped to 1. The structure was not drawn to include this path for simplicity; only the connection to VDD is included. The node situated prior to the NMOS, which is regulated by $\overline{dc}$, can only be charged to VDD or remain in a high-impedance state and retain the charge. Accordingly, this simplification accurately reflects the structure. Furthermore, the node $\overline{X}$ can be pulled solely over AB to GND, indicating that the top path is the sole one capable of triggering the transition from 1 to 0. Consequently, the top path loads from 0 to 1 (precharge), and the bottom path discharges from 1 to 0 (evaluate). By implementing this simplification and demonstrating solely the paths for the transitions, we achieve the structure depicted in Figure 6 with the exception that the transistor for Evaluate is closer to the output, as shown in Figure 7.

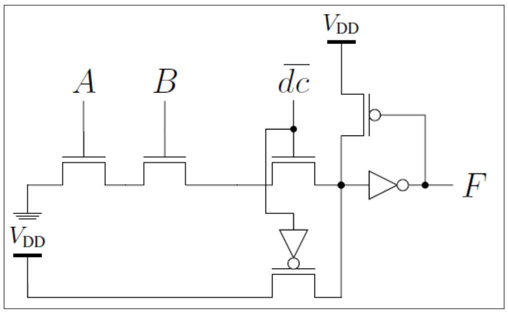

Figure 7: Transitions of SRDL mapped to LUT3

The structure can be replicated by exchanging the control inputs of the LUT. Domino logic can now be used for asynchronous handshaking and offers the aforementioned advantages of asynchronous circuits over traditional synchronous logic families. To achieve the objective of recognizing the transition through the gate, DRDL gates are employed, as illustrated in Figure 8.

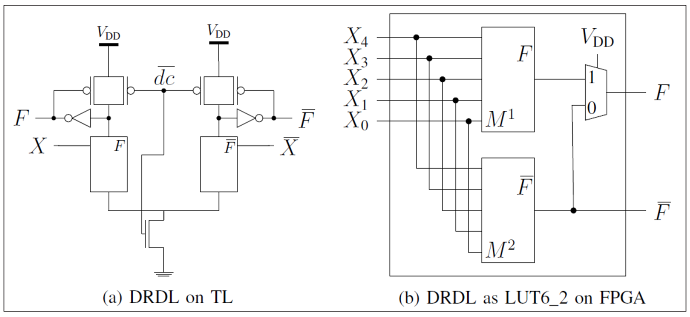

Figure 8: Dual Rail Domino Logic

These have an output designated as *F*, as well as a complementary output, which is represented by the symbol *F*. The same fundamental principle is applicable here: the precharge phase is first, followed by the evaluation phase. In the PC phase, both inner nodes are pulled to V_{DD}, the outputs are equivalent in their output value of logical 0, and then in the evaluation phase, one output becomes 1, while the other remains 0 due to the disjointness. This allows for the direct recognition of whether the domino gate has finished switching or not by linking both complementary outputs with an exclusive or (XOR). The dual-rail circuit thus provides a means of determining whether the circuit is in a valid state (i.e., the switched state) or an invalid state (i.e., the switching process is still underway). This information is always available, allowing the user to ascertain whether the circuit is currently occupied or ready for new data.

Pipeline with Completion Detection

It is now possible to construct a series of Domino gates in such a way that a pipeline is created, which is then operated in a sequential manner. This is in contrast to the usual operation of a pipeline, which is pipelined. This is accomplished by assigning a distinct state for each transfer of a “1,” that is, for each domino effect to the subsequent stage. Nevertheless, the use of real pipelining is also applicable in conjunction with the requisite holding elements between stages. Nevertheless, this approach would be infeasible for the processing of instructions and the multicycle processor under consideration. The serial composition is performed by establishing the dominoes from the initial stage, designated as f0, to the final stage, identified as $f_{n\text{-}1}$, to form the function f = fn-1. The composition of the serial pipeline is defined as follows:

$$f = f_{\text{n-1}}(f_{\text{n-2}}(\ldots(f_1(f_0))\ldots)) = f_0 \circ f_1 \circ \ldots f_{\text{n-2}} \circ f_{\text{n-1}}.$$

Firstly, all domino gates are subjected to a preliminary charge, the purpose of which is to energize the internal nodes to a voltage of V_{DD} and set the outputs to a value of 0. The evaluation phase then pulls each DRDL gate in a path to 0, thus producing a 1 at one output of *F* and $\overline{F}$. Upon achieving complementarity between all DRDL gates, the system is deemed complete and the input is thereby unlocked. The input pulse thus serves as a request signal and the *en* signal as an acknowledgment, thereby implementing an asynchronous handshaking protocol. In the pipeline circuit, it is sufficient to check only the last stage for disjunctivity, as the last stage can only switch as soon as the previous one has done so. However, a comprehensive analysis of all stages for disjointness has been conducted, whereby an XOR operation has been applied to each stage and the results rounded to ensure that each individual gate has switched and thus enhance safety.

Low-Level Primitives Design

The realization of our circuits is accomplished through the use of the Arty A7 Artix-7 FPGA Development Board, which is provided by Digilent and contains an FPGA manufactured by Xilinx Inc. The FPGA is programmed with the Vivado Design Suite (VDS) at the lowest possible level in order to precisely define how the structures are generated within the FPGA (what you see is what you get). The primitive libraries from ARTIX-7 are utilized for this objective [23]. Two commands have emerged as pivotal: firstly, the ability to incorporate combinatorial loops into the constraints, and secondly, the “don’t touch” commands to prevent the VDS from modifying any settings. The design is currently still completed manually, but it will be automated in the future. The logical design is based on LUTs. These LUTs are typically multiplexers that switch exactly one path to the output, depending on the input assignment. They are constructed in the shelf from NMOS pass transistors or transmission gates [22]. These low-level primitives can now be initialized as shown in the code snippet of listing 1.

In order to construct dual-rail domino logic circuits, it has been determined that the LUT6_2 structure will be utilized, as this configuration enables the

```
LUT6_2_inst : LUT6_2
generic map (
INIT => X"800000007FFF0000") --
port map (
O6 => f_int, -- 6/5-LUT output (1-bit)
O5 => fbar_int, -- 5-LUT output (1-bit)
I0 => '1', -- LUT input (1-bit)
I1 => '1', -- LUT input (1-bit)
I2 => x_int(0), -- LUT input (1-bit)
I3 => x_int(1), -- LUT input (1-bit)
I4 => _dc, -- LUT input (1-bit)
I5 => '1'-- LUT input (1-bit)
);
```

Listing 1: Low-Level LUT6_2 for AND2 DRDL Gate

Generation of two distinct outputs when input 5 is maintained at a fixed voltage of VDD. Nevertheless, this approach entails a trade-off in that one input is no longer available for use, and the number of table entries is reduced from 2^6 to 2^5. For designs with a maximum of five inputs, however, this has no adverse effects. The LUT is initialized with a hexadecimal number. In this case, the realized function is $F = I4 \wedge I3 \wedge I2 \wedge I1 \wedge I0$ for positive *P* in *F* and $\overline{F} = \overline{I4} \vee \overline{I3} \vee \overline{I2} \vee \overline{I1} \vee \overline{I0}$ for the complementary pin *F*. To generate the duty cycle for the self-locking input pulse circuit, a self-resetting LUT structure can be employed, wherein the LUT is set to one and transitions to a high state on the positive edge of P. Subsequently, the system resets itself asynchronously after a duration of τ_{Δ}. In lieu of utilizing the fundamental component of an FDCE, specifically a D-Flipflop with Clock Enable and Asynchronous Clear, as illustrated in, the corresponding LUT-based configuration from Figure 2 is employed, see 2 [16].

As placement and routing are fundamental to the safe implementation of asynchronous design, it is of the utmost importance to place the LUTs in a manner that precludes the emergence of structural hazards or functionally unstable feedback loops. In this case, it was crucial to employ the *LUT* 6 for the asynchronous self-resetting pulse circuit, as this represents the fundamental logical element of the Artix-7 FPGA. Consequently, the inputs are explicitly defined, thereby facilitating a comprehensive understanding of the design process. Some aspects of the design still require constraints, including locking the inputs to be utilized, establishing permissions for combinatorial loops, and implementing do-not-touch commands (see 3), which prevent the optimization of the implementation process, as this was optimized for synchronous design.

Parallelization of Domino Gates

As previously stated, switching can also occur in parallel, rather than in a cascaded manner. This is due to the fact that the switching processes are unidirectional and have a clear endpoint, which is enabled by the disjointed nature of the components and the self-clocking mechanism. This allows for hazard-free operation. This is due to the fact that the circuit is only unlocked once all switching components are disjoint from one another. It is similarly conceivable to design the ALU in parallel as a dual-rail domino gate, with the objective of maintaining the minimum processing delay. This is achieved by communicating with the controller using the handshake protocol and switching the individual gates in parallel until they are all disjoint. In this instance, the individual domino gates are composed in parallel to form the function $f = f_0(x) + f_1(x) + ... + f_{n-2}(x) + f_{n-1}(x)$, where $f_i(x)$ is the value of the i-th bit.

```
      attribute DONT_TOUCH of q_int : signal is "TRUE";
      attribute ALLOW_COMBINATORIAL_LOOPS : string;
      attribute ALLOW_COMBINATORIAL_LOOPS of q_int : signal is "TRUE";
  begin
        U2: y_LUT
          port map(
          Q => q_int,
          P=>P,
          En=>en,
          Y=>y_int
          );
          LUT_selfX : LUT6
          generic map (
           INIT => X"0000FFFFFFFFFFFF") -- Specify LUT Contents
          port map (
           O => q_int, -- 6/5-LUT output (1-bit)
          I0 => q_int, -- LUT input (1-bit)
          I1 => '1', -- LUT input (1-bit)
          I2 => '1', -- LUT input (1-bit)
          I3 => '1', -- LUT input (1-bit)
          I4 => y_int, -- LUT input (1-bit)
          I5 => '1' -- LUT input (1-bit)
          );
  dc<=q_int;
```

Listing 2: Low-Level Self-Resetting Pulse Circuit

```
component DRDL
port (
dcbar: in std_logic;
x: in std_logic_vector(3 downto 0);
f_out: out std_logic;
fbar_out: out std_logic
);
end component;
attribute dont_touch of DRDL : component is "yes";
```

Listing 3: Constraints for Asynchronous FPGA Design

Implementation For Risc-V Processor

The architectural design of RISC-V can be divided into two principal components: the datapath and the control unit. This paper presents the design of a control unit for a 32-bit Turing-complete RISC-V architecture. In the RV32 RISC-V architecture, memory addresses and data words are 32-bit. This represents a flexible and energy-efficient alternative to the dominant RISC and Complex Instruction Set Computer (CISC) architectures. The simplified instruction set at the core of RISC-V is small and orthogonal, allowing for a thriving ecosystem of innovation. This approach reduces the hardware requirements and improves overall performance by eliminating the complexity and overhead associated with complex instruction sets.

Reduced Instructions Set

The fundamental instructions, comprising a length of 32 bits, are classified into distinct categories based on the opcode. For illustrative purposes, Table II depicts the instruction formats that are Turing complete and are utilized in this work. The opcode is implemented in the initial seven bits of the instruction, while the remaining bits vary according to the specific operation. Arithmetic operations predominantly utilise the R-type format. The instruction has two source registers (rs1 and rs2) and one destination register (rd) as operands. The general function of the instruction is determined by the function field funct3, while the seven bits in funct7 are used to specify the exact function when the "funct3" of two instructions are the same. Instructions for loading operations are formatted according to the L-type format. This format is identical to that of the I-type. The I-type instruction format is analogous to the R-type format, with the exception that the funct7 and rs2 fields, which are present in the R-type format, are absent in the I-type format. Instead, the 12-bit field imm is utilized. Store instructions employ the S-type format, wherein the value in rs2 is stored at the address calculated by imm and rs1. The B-type format is utilized for branch instructions, wherein the values in rs1 and rs2 are compared, and the target address is contained within imm [24]. The jump and link instructions employ the J-type format, whereby the 20 bits within the imm field delineate the target address, while the return address is situated within rd [25].

Synchronous Multicycle CPU

We will now provide a brief introduction to the initial processor which can be observed in Figure 9 [24]. It is a synchronous processor that is Turing complete, which signifies its capacity to calculate all Turing-computable functions. The processor is realized as a multicycle processor in order to facilitate the design of different access times for different instructions, thereby enabling the division of an instruction into discrete individual steps. This differs from the approach taken with a single-cycle processor, where the worst-case path for the entire instruction is considered. Instead, in this case, the worst case for the individual processing steps is considered. However, the processor employs a Harvard architecture, which is evident from the fact that it has separate data and instruction registers in a block random access memory (BRAM) (i.e., with two different addresses).

Table II: RISC-V Instruction Formats

Type	Bits [31:25]	Bits [24:20]	Bits [19:15]	Bits [14:12]	Bits [11:7]	Opcode[6:0]
R-Type	funct7	rs2	rs1	funct3	rd	0110011
I-Type	imm[11:0]		rs1	funct3	rd	0010011
L-Type	imm[11:0]		rs1	funct3	rd	0000011
S-Type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	0100011
B-Type	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	1100011
J-Type	imm[20|10:1|11|19:12]				rd	1101111

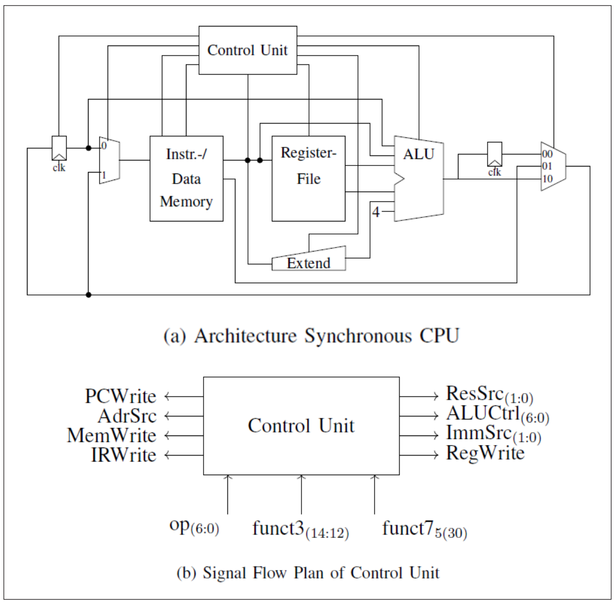

Figure 9: Synchronous CPU and the Synchronous State Transfer Function

Synchronous Control Unit

The control unit is described in brief below. In order to facilitate the processing of instructions, an automaton is generated that was derived from RISC-V and employs the input opcode 6:2 for the purpose of decoding the individual states. In contrast to [24], a few additional states were incorporated due to the necessity of two clock cycles for memory access. Consequently, the multicycle processor is divided into the following branches: load, store, r-type, i-type, btype, and jal. The clock cycles are distributed in order to achieve shorter access times, as illustrated in Figure 10.

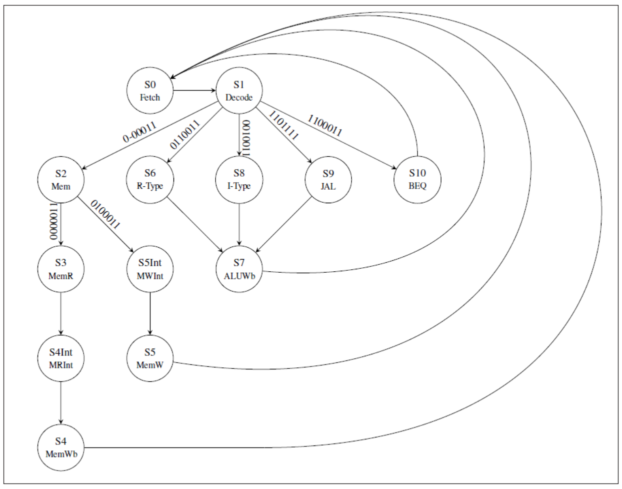

Figure 10: Automaton Graph of synchronous Moore Machine

This results in the longest instruction load and the shortest branch-if-equal (BEQ). A single-cycle processor would now accommodate the most unfavorable scenario for all instructions, whereas the multicycle processor would conserve three cycles for the branch-if-equal (BEQ) instruction. It is evident that the differing access times of the instructions necessitate the use of a multicycle processor. Subsequently, the automaton was constructed as a Moore machine, with each state having a single output. Subsequently, a synchronous automaton was devised at the high level, incorporating optimizations derived from VDS.

Design of the Asynchronous Controller

Subsequently, the asynchronous circuit is implemented in the synchronous RISC-V processor. The asynchronous handshaking protocol has the potential to markedly improve the processor's performance, as it enables the implementation of varying access times for distinct process steps (e.g., writing memory is considerably slower than addressing the register file). As the domino logic is realised as a pipeline, there is also a direct transfer to pipelines; however, this application is not addressed further in this paper. In order to facilitate the design of a pipeline cascade, which is a more complex process in domino logic, a Mealy automaton will be implemented that can generate different output values for its states depending on the input signal.

In contrast to the synchronous Moore automaton, which has predefined pro- cessing times for the various instructions, the Mealy automaton is clocked externally by REQ and ACK. This implies that the identical states for dis- parate instructions can have disparate access times. Consequently, the states are superimposed, and the edges are retained. Furthermore, self-locking can be applied to the output function, which negates the necessity for hazards and other potential issues. This results in a reduction in hardware requirements in comparison to the Moore Machine. The individual states were then encoded one-hot to enhance clarity in the domino output in the z-variables.

The automaton graph for the pipeline can be found in Figure 11 and its structure in Figure 12. The opcode of the BEQ branch is Edge A, as it requires only three state transitions. Edge B is given by the R-type, I-type, and JAL branches. Edge C represents the edge for the Store instruction, while the Load instruction reaches the final state, represented by the binary sequence [100000].

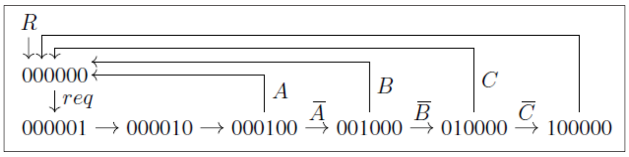

Figure 11: Automaton Graph Pipeline

Design of an Asynchronous ALU

To more effectively illustrate the advantages of the asynchronous handshake protocol, we have elected to configure the ALU as a parallelized, self-locking domino logic, which allows for a more comprehensive and accurate representation of the protocol's functionality. Given the considerable time required to access the ALU, the self-clocked variant represents a promising improvement. The following section will serve as a case study in domino logic, utilizing the AND instruction as a point of analysis. This approach will elucidate the design process. The initial stage of the process is the conversion of an asynchronous ALU into a domino logic structure. The bitwise AND can be implemented in a straightforward manner by utilizing the AND structure of a domino gate and combining each position of the 32-bit word in dual-rail. This can be accomplished entirely in parallel. The independence of all dual-rail gates indicates whether the ALU has undergone rounding, allowing the input to be unlocked by setting the en signal (ACK) to 1. The code snippet for the ALU's AND function as DRDL AND2 is provided below for reference. The resulting structure of the AND for the ALU in self-locking domino logic is illustrated in Figure 13. Moreover, the input incorporates a self-locking pulse circuit that generates a duty cycle, thereby initiating the precharge phase. This is followed by a scan of the source registers of the ALU multiplexers, after which the input is unlocked when each of the 32 DRDL gates has disjoint outputs.

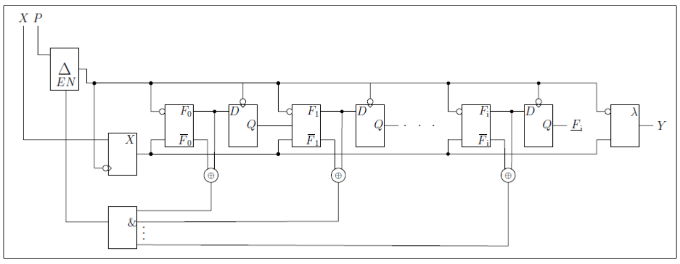

Figure 12: Structure of realized Pipeline in FPGA

Integration in the CPU

The self-locking machine can now be readily incorporated into the existing CPU and controlled via the clock, for instance, thus facilitating its integration into existing systems. While this does not immediately enhance performance if the other processor components do not exhibit GALS behavior, it demonstrates the simplicity of integrating self-timed circuits. Moreover, self-timing requires fewer flip-flops, which consequently results in reduced power consumption. To illustrate the advantages of asynchronous handshaking, the DRDL ALU was also integrated. The controller oversees the operation of other components in a synchronous manner, while simultaneously initiating the asynchronous handshaking process with the ALU, thereby accelerating the execution of instructions that utilize the ALU.

```
MY_GEN : for i in 0 to 31 generate
    DominoGate: dualRail
    port map(
    dcbar => dc,
    x(0)=>'1',
    x(1)=>'1',
    x(2)=>reg_b(i),
    x(3)=>reg_a(i),
    f_out=>f_int(i),
    fbar_out=>fbar_int(i)
    );
    CompletionDetection:  xor_LUT
    port map(
    A => f_int(i),
    B => fbar_int(i),
    Y => xor(i)
    );
    process(dc)
    begin
    if(falling_edge(dc)) then
        f_out(i)<= f_int(i);
        fbar_out(i)<= fbar_int(i);
    end if;
    end process;
end generate;
process(xor)
 begin
  if(xor=x"FFFFFFFF")
  then
   en_int<='1';
  else
   en_int<='0';
end if;
end process;
```

Listing 4: Low-Level 32-Bit DRDL AND

Power Performance and Area (PPA) Results

The PPA analysis of the implemented asynchronous processor on an FPGA definitively demonstrates its efficiency and viability for a range of applications. This section presents the findings from the synthesis, implementation, and simulation processes using the Vivado Design Suite. The results are presented in three subsections: power analysis, performance analysis, and area analysis.

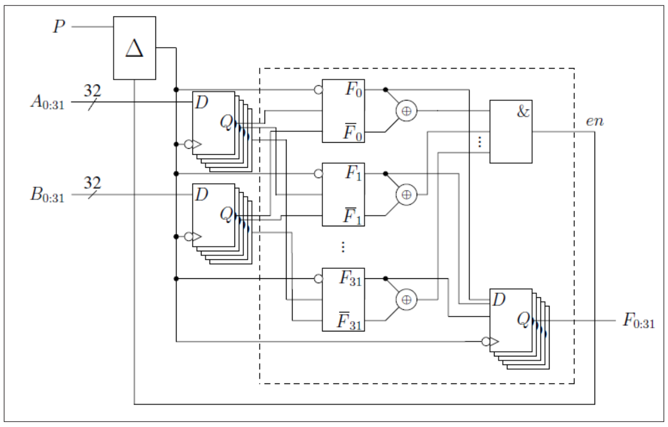

Figure 13: Domino Logic ALU

Power Consumption

The power consumption shares of the individual processors were obtained from the Vivado Power Analysis Tool, as detailed in Figure 14.

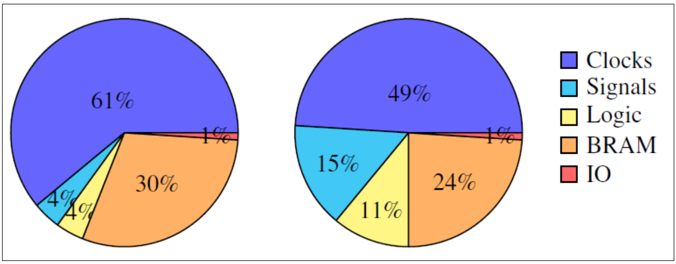

Figure 14: Power Consumption of the Synhronous vs. Asynchronous CPU

The total dynamic power consumption of both the asynchronous and synchronous processors is nearly similar, which can be attributed to the parallel operation of the asynchronous processor. Nevertheless, the asynchronous design results in significant power savings in the clock network, as demonstrated by the lower proportion of power consumption allocated to clocks in the asynchronous processor in comparison to the synchronous one. This underscores the efficacy of the asynchronous processor in curbing clock-related power consumption, which is a pivotal aspect of comprehensive power management and energy efficiency.

Performance Analysis

In this section, we evaluate the effectiveness of our asynchronous CPU design by comparing it with a synchronous CPU. To elucidate the advantages and potential trade-offs of the asynchronous CPU design in comparison to its synchronous counterpart, the SPECint2000 benchmark suite is employed. The SPECint2000 benchmark is a well-established and widely used tool for evaluating the performance of computer processors. It is maintained by the Standard Performance Evaluation Corporation (SPEC), which also develops and maintains the broader SPEC benchmark suite. SPECint2000 is particularly focused on integer processing. The benchmark is commonly applied to measure the performance of CPUs with integer-heavy workloads, providing a comprehensive assessment through a variety of real-world applications. The benchmark is composed of approximately 25 % loads, 10% stores, 11% branches, 2% jumps, and 52% R- or I-type ALU instructions [24]. The diverse range of operations included in the SPECint2000 benchmark renders it an optimal tool for evaluating and comparing the performance characteristics of different CPU architectures.

The synchronous CPU and the designated asynchronous CPU are applied to run the programs in the SPECint2000 benchmark set. Ultimately, the throughput and the average latency of every instruction of both CPUs are observed.

A loop was constructed around the test code with a branch-targeted indirect jump (BEQ) instruction using a clock frequency of 100 megahertz (MHz), and the throughput of the asynchronous and synchronous CPUs, as well as the average duration per instruction, were determined see Table III.

Table III: Performance Metrics

Parameter	Asynchronous CPU	Synchronous CPU
Throughput (MIPS)	25.64	22.73
Average Latency/ Instruction	39.5 ns	44 ns

The analysis demonstrates that the CPU performance for the SPECint2000 benchmark increased by approximately 10% as a result of enabling communication between the control unit and the ALU through the use of DRDL gates.

Area Analysis

The objective of the area analysis is to evaluate the utilization of FPGA resources, including LUTs, slice registers, and slices. The asynchronous design utilized 6.67% of the available LUTs, indicating a moderate complexity in logic implementation. Additionally, the design employed 4.85% of the available slice registers and leveraged 4.63% slices.

As anticipated, the area utilized exhibited an increase, yet remained within the anticipated range due to the implementation of DRDL gates within a single LUT.

Table IV: FPGA Resource Utilization

Resource Type	Utilization in (%) Async	Utilization in (%) Sync
LUTs	6.67%	6.32%
Slice Registers	4.85%	4.9%
Slices	9.27%	8.9%

Discussion

The PPA results indicate that the asynchronous processor exhibits substantial potential in terms of performance, which is of paramount importance for performance-critical applications. Although there was a slight alteration in power consumption, the area analysis demonstrates a balanced utilization of FPGA resources, thereby substantiating the feasibility of implementing such designs within reasonable silicon area constraints.

Blueprint of an Asynchronous CPU

The straightforward integration resulting from the modularity of asynchronous circuits allows for the presentation of a comprehensive blueprint for an asynchronous processor. The objective is to devise the entire processor in an asynchronous manner, whereby each circuit component operates with request and acknowledgment signals, eliminating the necessity for a global clock for the entire circuit, see Figure 15. It can be observed that the individual circuit components engage in communication with the controller through the use of a request and acknowledgement mechanism, which is represented by the double arrow symbol. The construction of combinational circuits is achieved through the utilisation of domino logic, whereas the implementation of D-FFs is accomplished by employing asynchronous D-latches with request and acknowledge signals. The memory operates on clock cycles, wherein the ack signal is set with precision after two cycles (the data access process necessitates this number of cycles). The feasibility of a single-cycle processor for this type of processor is also a potential avenue for future investigation, however, the current focus is on the development of a pipelined processor.

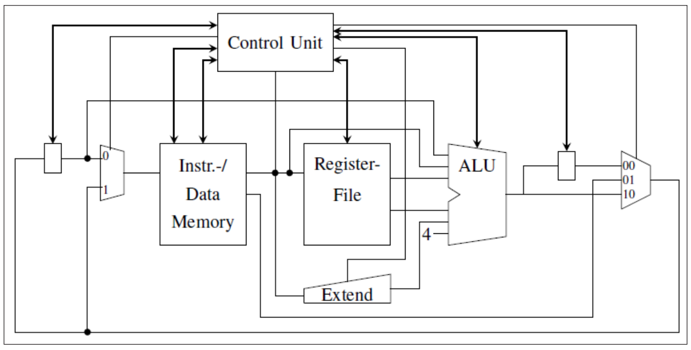

Figure 15: Asynchronous CPU

Conclusion and Future Work

This work proposes an asynchronous RISC-V CPU design based on self- locking domino control. The asynchronous approach offers advantages over traditional synchronous designs in terms of performance, power consumption, and modularity. The paper provides a comprehensive account of the design and implementation of the asynchronous control unit, which employs domino control on an FPGA development board. The control unit has been designed to accommodate a Turing-complete 32-bit RISC-V architecture. A noteworthy feature of the design is the incorporation of a self-locking mechanism, which guarantees that the circuit remains engaged until all processing stages have been completed. This eliminates the necessity for a global clock and simplifies error- free operation. Furthermore, the paper discusses the possibility of parallelizing the ALU using domino control to further improve performance. Subsequently, the paper illustrates the straightforward integration of the asynchronous control unit into an existing synchronous CPU, thereby demonstrating the potential benefits of self-timed circuits. Ultimately, the PPA analysis of the implemented asynchronous processor on an FPGA substantiates its considerable potential for diverse applications.

The power analysis indicates that while the total dynamic power consumption of the asynchronous processor is comparable to that of the synchronous processor, it achieves a significant reduction in power consumption within the clock network. This highlights the effectiveness of the asynchronous processor in reducing clock-related power consumption, which is a crucial aspect of comprehensive power management and energy efficiency. A performance analysis conducted using the SPECint2000 benchmark suite revealed that the asynchronous processor exhibited superior performance compared to the syn- chronous processor. This was demonstrated by a 10% increase in throughput and a reduction in average latency per instruction. This performance enhancement is achieved through the utilisation of handshaking with DRDL gates in the control unit and ALU. The area analysis indicates that the asynchronous design employs FPGA resources in a moderate manner, exhibiting a slight increase in LUT, slice register, and slice utilization in comparison to the synchronous design. Notwithstanding this increase, the resource utilization remains within acceptable limits, thereby substantiating the feasibility of implementing the asynchronous processor within reasonable silicon area constraints. In conclusion, the asynchronous processor exhibits notable advantages in terms of power efficiency, performance, and area utilization, thereby establishing its viability as a potential solution for performance-critical applications.

References

1. J Sparsø (2021) Asynchronous circuit design - a tutorial https://www.semanticscholar.org/paper/Asynchronous-circuit-design-A-tutorial-Spars%C3%B8/b2c73609b55a79af2c7a8577bb0937598b2c3eda.
2. F Bouesse, N Ninon, G Sicard, M Renaudin, A Boyer, et al. (2007) Asynchronous logic vs synchronous logic: Concrete results on electromagnetic emissions and conducted susceptibility https://www.researchgate.net/publication/266496771_Asynchronous_logic_Vs_Synchronous_logic_Concrete_Results_on_Electromagnetic_Emissions_and_Conducted_Susceptibility.
3. A Waterman (2016) Design of the risc-v instruction set architecture. https://escholarship.org/uc/item/7zj0b3m7.
4. Rich Collins (2024) How the RISC-V ISA offers greater design freedom and flexibility. Tech Talk 1-15.
5. A Dooply, K Yun (1999) Optimal clocking and enhanced testability for high-performance self- resetting domino pipelines. in Proceedings 20th Anniversary Conference on Advanced Research in VLSI 200-214.
6. K Yun, A Dooply (1999) Optimal evaluation clocking of self-resetting domino pipelines. in Proceedings of the ASP-DAC '99 Asia and South Pacific Design Automation Conference 121-124.
7. G Jung, JJ Kong, G Sobelman, K Parhi (2002) High-speed add-compare-select units using locally self-resetting cmos. in 2002 IEEE International Symposium on Circuits and Systems (ISCAS) 1: 1-1.
8. SO Jung, KW Kim, SM Kang (2003) Timing constraints for domino logic gates with timing- dependent keepers. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 96-103.
9. M Litvin, S Mourad (2005) Self-reset logic for fast arithmetic applications. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems 13: 462-475.
10. A Alsharqawi, A Einioui (2006) Clockless pipelining for coarse grain datapaths. in 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design 5.
11. U Ramadass, J ponnian, P Dhavachelvan (2014) New low power adders in self-resetting logic with gate diffusion input technique. Journal of King Saud University - Engineering Sciences 7: 3.
12. Z Xia, M Hariyama, M Kameyama (2015) Asynchronous domino logic pipeline design based on constructed critical data path. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems 23: 619-630.
13. H Rezaei, SA Moghaddam (2016) Implementation of low-power and high-performance asynchronous dual-rail join using domino logic gates in 16-nm technology. in 2016 24th Iranian Conference on Electrical Engineering (ICEE) 142-147.
14. D Sokolov, V Khomenko, A Mokhov, V Dubikhin, D Lloyd, et al. (2020) Automating the design of asynchronous logic control for ams electronics. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39: 952-965.
15. Z Li, Y Huang, L Tian, R Zhu, S Xiao, et al. (2021) A low-power asynchronous risc-v processor with propagated timing constraints method. IEEE Transactions on Circuits and Systems II: Express Briefs 68: 3153-3157.
16. F Deeg, X Wu, SM. Sattler (2024) Self-locking domino logic pipelined controller for risc-v in fpga. Athens Journal of Technology and Engineering 11: 201-218.
17. F Deeg, SM Sattler (2024) Self-locked asynchronous controller for risc-v architecture on fpga. In AmEC 2024 - Automotive meets Electronics & Control 1-5.
18. F Deeg, J Zhu, SM Sattler (2020) Asynchronous design. in AmE 2020 - Automotive meets Electronics; 11th GMM-Symposium 1-5.
19. M Krstic, E Grass, FK Gaijrkaynak, P Vivet (2007) Globally asynchronous, locally synchronous circuits: Overview and outlook. IEEE Design & Test of Computers 24: 430-441.
20. D Chapiro (1984) Globally-asynchronous locally-synchronous systems 9.
21. D Hodges, H Jackson, R Saleh (2004) Analysis and design of digital integrated circuits: In deep submicron technology / da hodges h g jackson, ra saleh 1.
22. C Chiasson, V Betz (2013) Should fpgas abandon the pass-gate. in 2013 23rd International Conference on Field programmable Logic and Applications 1-8.
23. (2012) Vivado Design Suite 7 Series FPGA Libraries Guide. XILINX UG953 https://ecen220wiki.groups.et.byu.net/media/lab_06/00_fdce.pdf.
24. S Harris, D Harris (2021) Digital Design and Computer Architecture, RISC-V Edition. Elsevier Science https://shop.elsevier.com/books/digital-design-and-computer-architecture-risc-v-edition/harris/978-0-12-820064-3.
25. A Waterman, K Asanovic (2017) The risc-v instruction set manual. User-level ISA, document version 1-133.