Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática
MAESTRÍA EN DISEÑO ELECTRÓNICO

Optimization of a TSPC D flip-flop using MatLab

Trabajo recepcional que para obtener el grado de
Maestro en Diseño Electrónico

Presenta: **IVÁN MARTINEZ FLORES**

Director **JOSÉ LUIS CHÁVEZ HURTADO**

Tlaquepaque, Jalisco. 8 de enero de 2025.

Acknowledgments

In loving memory of my father, José de Jesús Martínez†.

I want to thank my mother, Dolores Flores, who has always supported my choice.

I want to thank my wife, Laura, my older child, Pablo, and my younger child, Ivanna, for helping me finish this work.

I would also like to thank Dr. José Luis Chavez-Hurtado; this work would never have been possible without his support.

Also, I would like to thank Dr. Esteban Martinez Guerrero and Dr. Cuauhtémoc Aguilera for supporting my original work.

Summary

This case study builds upon previous work by the same author, as referenced in [1], and represents a continuation of that research in pursuit of a Master’s degree in Electronic Design. This work focuses on transitioning from a conventional Successive Approximation Register Analog-to-Digital Converter (SAR-ADC) to a time-based architecture. My specific contribution to this project involved the design and optimization of a True Single-Phase Clock (TSPC) D flip-flop, which serves as the arbiter in the time-based SAR-ADC design. The arbiter is a critical component that requires an exceptionally low setup time, which was a primary objective of this research.

The TSPC D flip-flop was simulated using the WinSpice simulation tool, with the circuit based on the TSMC 0.18 µm CMOS technology node. The main objective of this thesis was to identify the optimal circuit configuration by adjusting the transistor dimensions—precisely the width (W) and length (L)—in the flip-flop circuit. The goal was to minimize the setup time, which is a key performance metric for high-speed digital circuits.

To achieve this, several optimization algorithms embedded in MATLAB were employed, including *fminsearch, fmincon,* and *genetic algorithms*. These algorithms were used to find the optimal transistor dimensions without the need for manual implementation of the optimization functions. Instead, existing MATLAB libraries were leveraged to streamline the process and focus on improving the circuit’s performance.

The results of the optimization process revealed that the genetic algorithm outperformed the other methods in terms of setup time minimization. Specifically, the genetic algorithm produced a setup time that was closer to zero compared to the results obtained using *fminsearch* and *fmincon*. This indicates that the genetic algorithm is more effective at refining the circuit's parameters to achieve the desired low setup time, making it the most suitable choice for this type of optimization.

In summary, this work demonstrates the effectiveness of numerical optimization methods, particularly genetic algorithms, in minimizing the setup time of the TSPC D flip-flop. The research contributes to the broader field of time-based SAR-ADC design and highlights the potential for further applications of these optimization techniques in digital circuit design.

Table of Contents

Introduction

The central focus of this case study is the simulation and optimization of a True Single-Phase Clock (TSPC) D flip-flop using WinSpice for circuit simulation and MATLAB for optimization. The primary objective of this research is to minimize the setup time of the TSPC D flip-flop, thereby enabling faster input processing and improving overall circuit performance [1], [3].

A TSPC D flip-flop is a specialized version of a D flip-flop that operates with a single-phase clock throughout its design, avoiding the use of a negated clock signal that can introduce unwanted delays and timing issues [1]. This characteristic makes it particularly suited for high-speed applications [4]. In Section 1, an overview of typical flip-flop designs is presented, including SR, T, JK, and D flip-flops [6], [7], [8]. This section serves as a foundation for understanding the specific advantages of the TSPC D flip-flop, which is employed in a modified version of a Successive Approximation Register Analog-to-Digital Converter (SAR-ADC) [1], [11]. Unlike standard D flip-flops, the TSPC D variant utilizes dynamic logic, which enhances speed while maintaining power efficiency. The use of a single-phase clock further contributes to reduced power consumption and increased circuit efficiency [1]. In the context of the SAR-ADC design, this flip-flop functions as an "Arbiter," responsible for determining which signal arrives first in the time domain, a crucial function in time-based architectures [1].

Section 2 delves into the optimization algorithms used to find the optimal solution for minimizing the setup time of the TSPC D flip-flop. Various algorithms, including *fmincon*, *fminsearch*, and genetic algorithms (GA), are explored in detail [3]. Additionally, this section discusses the selected circuit topology and explains the rationale behind shifting from a voltage-based design to a time-based design [3], [11]. The chosen topology focuses on the arbiter, modeled using MOSFET transistors, and emphasizes the TSPC D flip-flop's critical role in the overall architecture [1].

Section 3 presents the results of the optimization process using MATLAB's native functions. For each algorithm, the code is provided along with a thorough analysis of the generated plots and performance outcomes [3]. These results illustrate how the different optimization approaches affect the setup time and highlight the genetic algorithm as the most effective method for achieving minimal setup time [3].

The work concludes with Appendix A, which contains the complete codebase, including the scripts used for each function, along with the corresponding file names. This appendix ensures reproducibility of the results and provides a detailed reference for future researchers seeking to replicate or extend this study [3].

This case study not only demonstrates the benefits of using TSPC D flip-flops in time-based SAR-ADC designs but also underscores the potential of employing advanced optimization techniques in electronic circuit design to improve performance metrics such as setup time, power consumption and circuit efficiency [1], [3], [4], [6].

1. Flip-flops

Flip-flops, also known as bistable multivibrators, have been fundamental components in digital electronics for many years, constructed from various electronic elements such as logic gates and transistors [5], [7]. Their primary characteristic is the ability to maintain two stable states, representing a logic "0" and a logic "1," making them essential for memory storage, clock synchronization, and sequential logic operations [6], [9].

A flip-flop is a key element in digital logic and sequential circuits, where its output changes only in response to shifts in the clock signal. These clock-driven components are referred to as synchronous latches [4], [8]. Flip-flops are typically built using logic gates such as AND, OR, NAND, and NOR, with the specific configuration varying depending on the type of flip-flop in question [6]. In sequential circuits, the synchronization provided by the clock ensures that output changes occur at precise intervals [5].

Transistors serve as the foundational building blocks for flip-flops, especially in modern CMOS (Complementary Metal-Oxide-Semiconductor) technology [11]. In CMOS circuits, flip-flops are constructed using complementary pairs of NMOS and PMOS transistors, arranged in specific configurations to realize the intended flip-flop functionality [1], [11]. The use of transistors in this context ensures the integration of flip-flops in highly efficient and scalable digital systems [7].

One of the defining features of a flip-flop is its clock input, which regulates when the flip-flop captures input data and updates its state [4], [9]. This clock signal ensures that the flip-flop operates synchronized with the broader digital circuit, preventing timing errors and enhancing coordination among different parts of the system [5]. Additionally, flip-flops often feature feedback loops, enabling the current state to influence future states [6], [10]. This characteristic makes flip-flops essential in memory elements such as RAM (Random Access Memory) and registers, where they store and maintain binary data for temporary or long-term use [8].

In digital systems, flip-flops play a crucial role in synchronizing operations with clock edges, ensuring that data transitions occur at specific points in time [4]. They are also commonly found in registers, where they handle various data storage and manipulation tasks essential for the operation of microprocessors and digital signal processors (DSPs) [9]. Their ability to store and retain data underpins the functionality of many digital devices [5], [7].

Flip-flops are indispensable in sequential logic circuits and state machines, where they help define the system's state and control its behavior based on both current and previous states [6], [8]. Their utility extends to digital clock and timer circuits, where they are used to generate precise timing signals required for tasks such as timekeeping, signal processing, and event coordination [10].

Moreover, flip-flops are integral to many digital communication systems, including modulators, demodulators, and data encoding/decoding circuits [4]. They are also vital in control logic circuits, where they store and manipulate control signals that direct the operation of digital systems [8], [10].

In summary, flip-flops are versatile components used across a broad spectrum of digital electronics, from memory systems and state machines to communication circuits and control logic [6], [7], [11]. Their ability to reliably store and transition between states underpins their widespread application in modern digital systems [1], [5].

1.1. Flip-flop types.

Flip-flops can be categorized into SR, T, JK, and D types, each with unique characteristics that make them suitable for specific applications [4], [6]. These fundamental building blocks are critical for constructing more complex circuits, such as registers, counters, and memory cells [5], [7].

SR (Set-Reset) flip-flops are among the simplest types, primarily used in basic memory applications, where their ability to hold a binary state is leveraged [6]. T (Toggle) flip-flops are widely utilized in counters, as they toggle their state on every clock pulse, simplifying the design of sequential counting circuits [9]. JK flip-flops offer more versatility by combining the functionalities of SR and T flip-flops while eliminating the indeterminate state, making them ideal for more intricate applications like frequency division and control circuits [8], [10].

Finally, D (Data or Delay) flip-flops are pivotal in registers and memory cells due to their straightforward design, where the output precisely follows the input at each clock pulse [1], [7]. This characteristic makes them indispensable for data storage and synchronization in digital systems [5].

Understanding the construction and functionality of these flip-flops is vital, as they serve as the foundation for more advanced digital circuit designs. By combining them, designers can create registers, which temporarily hold data; counters, which track occurrences; and memory cells, which provide longer-term data storage [6], [11].

1.1.1. SR flip-flop

The SR (Set-Reset) flip-flop is an essential memory element in digital electronics. It has some specific advantages [6]:

- Simplicity. It can be constructed using basic logic gates, typically NAND gates. This simplicity makes it easy to understand and implement.
- Asynchronous Operation. The SR flip-flop operates asynchronously. This means it can change its state in response to input changes at any time, regardless of the clock signal.
- Immediate Response. The output changes immediately when the S (Set) or R (Reset) inputs change. There is no need to wait for a clock signal, which can be advantageous in specific real-time applications.

- Bi-stable Operation. The SR flip-flop has two stable states, making it ideal for applications where you need to store some information without signal clocks. This makes it suitable for applications like latch circuits in control systems.
- Useful in Control Circuits. The SR flip-flop asynchronous nature makes it particularly useful in control circuits, where immediate action is often required. For instance, it can be used in situations that require the manual toggling of states.

Despite these advantages, SR flip-flops have some drawbacks as well. They are susceptible to a problem known as the "race condition," which can lead to unpredictable behavior. This issue arises when both S and R inputs change simultaneously. Additional logic may be required to modify this behavior, complicating the design.

Moreover, due to race conditions, SR flip-flops are not commonly used in modern digital circuit designs. Instead, edge-triggered flip-flops like D and JK are preferred for most applications. These flip-flops offer more predictable and reliable behavior under different conditions.

Fig. 1 shows an SR flip-flop, and its behavior is described in the truth table in Table 1.

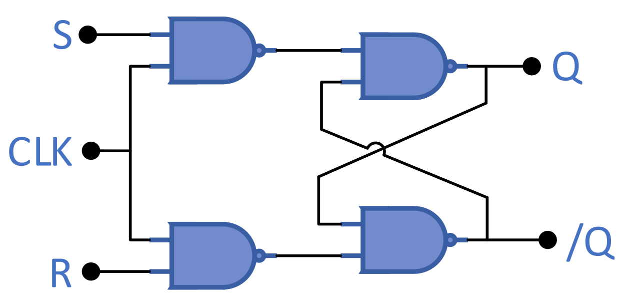

Fig. 1. SR flip-flop implemented using NAND Gates

S	R	Q_{t+1}	**Description**
0	0	Q_t	No change (hold state)
0	1	0	Reset (clear state)
1	0	1	Set (set state)
1	1	Undefined (X)	Invalid condition

Table 1. SR flip-flop truth table from [17]

- Q_t: Current state of the flip-flop.
- Q_{t+1}: Next state after evaluating the inputs.
- The truth table does not consider clock transitions, focusing instead on how the inputs determine the next state of the flip-flops.

1.1.2. T flip-flop

The T (Toggle) flip-flop is a type of digital circuit that has several advantages in specific applications [9]:

- Frequency Division: T flip-flops are commonly used in frequency division circuits. By toggling their state at each clock pulse, they effectively halve the frequency of the input signal. This is valuable in applications like clock generation.
- Simplicity: T flip-flops are simple in their construction. They can be built using basic logic gates, typically with just a few components.
- Toggle Capability: When the clock input triggers, the output will change to the opposite of its current state. This is especially useful in applications where a signal needs to alternate between two states.
- Divide-by-2 Counter: T flip-flops are often used to implement divide-by-2 counters, a fundamental component in many digital systems. Due to their toggling behavior, flip-flops are a natural choice.
- Synchronization: Sometimes, T flip-flops can be used for synchronization. They can synchronize an internal clock signal with an external one.
- Reduced Logic Complexity: When compared to JK or D flip-flops, T flip-flops may require fewer logic gates to implement. This can result in a more straightforward and potentially more efficient design.

However, it is essential to note that T flip-flops do have some limitations and considerations:

- Less Common: In practice, T flip-flops are less commonly used than other types, such as D or JK flip-flops. Their toggling behavior is specific and not always necessary for many applications.
- Potential for Glitches: The toggling behavior can sometimes introduce glitches or unwanted transitions if the input signal is noisy or has rapid changes. This can complicate the design and may require additional logic to mitigate.
- Limited Functionality: T flip-flops are excellent for frequency division and toggling applications. However, they may not be as versatile as other flip-flops in specific scenarios.

In summary, T flip-flops are valuable in applications that require frequency division and toggling. Their simplicity and specific functionality make them suitable for certain tasks. Still, they may not be the best choice for all digital circuit designs.

Fig. 2 shows a T flip-flop, and its behavior is described in the truth table in Table 2.

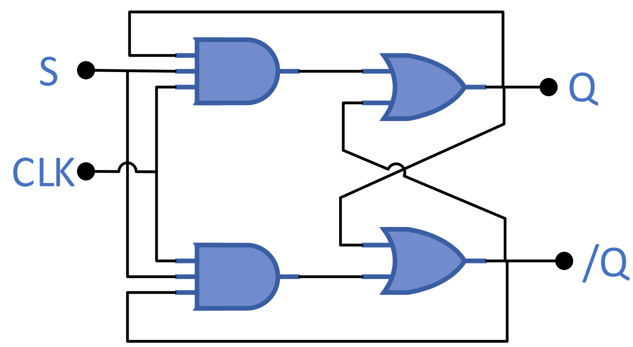

Fig. 2. T flip flop implemented using logic gates

T	Q_{t+1}	Description
0	Q_t	No change (hold state)
1	$\overline{Q_t}$	Toggle (invert state)

Table 2. T flip flop truth table from [17]

- Q_t: Current state of the flip-flop.
- Q_{t+1}: Next state after evaluating the inputs.
- The truth table does not consider clock transitions, focusing instead on how the inputs determine the next state of the flip-flops.

1.1.3. JK flip-flop

JK flip-flops have several advantages that make them a popular choice in digital circuit design [8], [10]:

- Toggle Capability: One critical advantage of a JK flip-flop is its ability to toggle its state. When both J and K inputs are set to 1, the flip-flop will toggle, changing its state from 0 to 1 or 1 to 0. This is a unique feature that D, T, and SR flip-flops do not have.
- Versatility: The JK flip-flop is very versatile. It can be used as a fundamental memory element. Still, it is also the building block for constructing more complex sequential logic circuits like counters and shift registers.
- Race Condition: Elimination: JK flip-flops sometimes help eliminate race conditions. A race condition occurs when two or more signals arrive at the flip-flop inputs simultaneously, and the output can become uncertain. The toggle capability of JK flip-flops can help resolve such issues.
- Self-Correcting Behavior: JK flip-flops exhibit a self-correcting behavior. Suppose both J and K inputs are held at 1 for an extended period. In that case, the flip-flop will eventually settle into a stable state.
- Equivalent to D and T flip-flops: A JK flip-flop can behave like a D or a T flip-flop. This means it can perform all the functions of these simpler flip-flops while having the additional toggle functionality.
- Symmetrical Operation: The JK flip-flop has a symmetrical operation, meaning that the behavior is the same regardless of whether the input signals are applied at the rising or falling edge of the clock signal (for edge-triggered flip-flops).
- Useful in Counters and Registers: JK flip-flops are commonly used in applications like counters and shift registers, where toggling is a fundamental operation due to their toggle capability.
- Reduced Circuit Complexity: In some cases, JK flip-flops can lead to simpler circuit designs than other flip-flops, especially when the toggle functionality is a critical requirement. JK flip-flops can be more complex to design and analyze than simpler flip-flops like D or T flip-flops. They also require more gates, increasing the overall circuit complexity and power consumption.

While JK flip-flops have these advantages, it is essential to note that they can be more complex to design and analyze compared to simpler flip-flops like D or T flip-flops. They also require more gates, increasing the overall circuit complexity and power consumption.

Therefore, while JK flip-flops have advantages, the choice of flip-flop type should always be based on the application's specific requirements.

Fig. 3 shows a JK flip-flop, and its behavior is described in the truth table in Table 3.

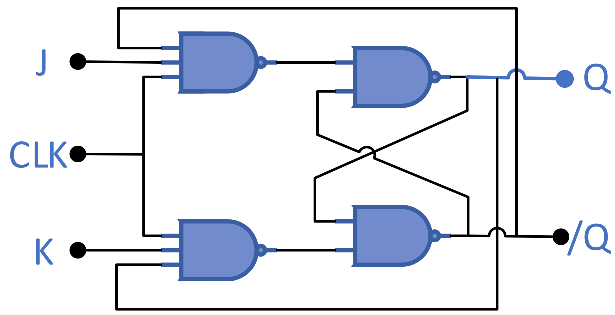

Fig. 3. JK flip flop implemented using logic gates

J	K	Q_{t+1}	**Description**
0	0	Q_t	No change (hold state)
0	1	0	Reset (clear state)
1	0	1	Set (set state)
1	1	$\overline{Q_t}$	Toggle (invert state)

Table 3. JK flip flop truth table from [17]

- Q_t: Current state of the flip-flop.
- Q_{t+1}: Next state after evaluating the inputs.
- The truth table does not consider clock transitions, focusing instead on how the inputs determine the next state of the flip-flops.

1.1.4. D flip-flop

The D flip-flop is a popular choice in digital circuit design for several reasons [5], [7]:

- Simplicity: The D flip-flop is more straightforward than other flip-flops like JK or T. It typically requires fewer logic gates to implement.
- Ease of Use: It has only one input, the data input (D), and one clock input. This simplicity makes it easy to understand and use in designs.
- Predictable Behavior: The D flip-flop changes its output state based solely on the data input and the clock signal. This predictable behavior makes designing and analyzing circuits using D flip-flops easier.
- Reduced Power Consumption: D flip-flops often consume less power than complex flip-flops like JK. This can be an essential consideration in low-power applications.
- Application in Edge-Triggered Systems: D flip-flops are commonly used in edge-triggered systems. They change their output state only on a specific clock signal transition (rising or falling edge), which can help synchronize operations.
- Memory Elements: The D flip-flop is a fundamental building block for constructing more complex memory elements like registers and RAM (Random Access Memory) cells.
- Waveform Shaping: D flip-flops are functional in waveform shaping applications, which can help synchronize or reshape signals before they are further processed.
- Ease of Cascading: They are easy to cascade. You can connect multiple D flip-flops to form more extended shift registers or other types of sequential circuits.
- Reduced Race Conditions: Compared to SR flip-flops, D flip-flops are less susceptible to race conditions. This makes them more reliable in synchronous systems.
- Useful in Digital Signal Processing (DSP): D flip-flops are used extensively in digital signal processing applications due to their predictable behavior and suitability for edge-triggered systems.

While D flip-flops have these advantages, it is essential to remember that they may not be the best choice for all applications. For instance, a JK flip-flop might be more appropriate if you need a flip-flop with a toggle function. The selection of flip-flop type should always be based on the specific requirements and constraints of the application.

Fig. 4 shows a D flip-flop, and its behavior is described in the truth table in Table 4.

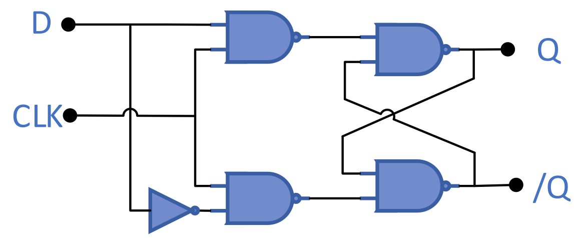

Fig. 4. D flip-flop implemented using NAND gates

D	Q_{t+1}	Description
0	0	Set to 0
1	1	Set to 1

Table 4. D flip-flop truth table from [17]

- Q_t: Current state of the flip-flop.
- Q_{t+1}: Next state after evaluating the inputs.
- The truth table does not consider clock transitions, focusing instead on how the inputs determine the next state of the flip-flops.

2. Optimization

Optimization algorithms are critical tools for finding the best solutions to problems by minimizing or maximizing an objective function. They are widely employed across various fields to enhance system performance, efficiency, or quality. These algorithms are applied in diverse domains such as:

- **Energy Systems:** Optimization frameworks enhance resource allocation and energy distribution, leading to cost reductions and better sustainability [12].
- **Engineering and Design:** Optimization techniques improve structural designs, products, and systems by meeting criteria such as weight, cost, or efficiency [13].
- **Machine Learning and Data Science:** Gradient-based methods, genetic algorithms, and other heuristic techniques are used to tune model parameters, improving predictive accuracy and computational efficiency [14].
- **Operations Research:** Algorithms like linear programming and integer optimization address logistical challenges, scheduling, and resource allocation, improving efficiency and reducing costs [15].
- **Traffic Control:** These methods optimize traffic signals and transportation systems, reducing congestion and improving flow [16].

Optimization algorithms include methods such as linear programming, genetic algorithms, simulated annealing, and gradient descent, each suited to specific challenges. By automating the search for optimal solutions in high-dimensional, complex spaces, these algorithms support better decision-making and improved outcomes across diverse applications.

2.1. Selected Topology

For the topology selected, we use the one described in [1], and corresponds to a typical SAR-ADC, as shown in Fig. 5. The selected topology contains four general blocks: Sample & Hold circuit, Comparator, N-bit SAR logic , and a Digital-to-Analog Converter (DAC) .

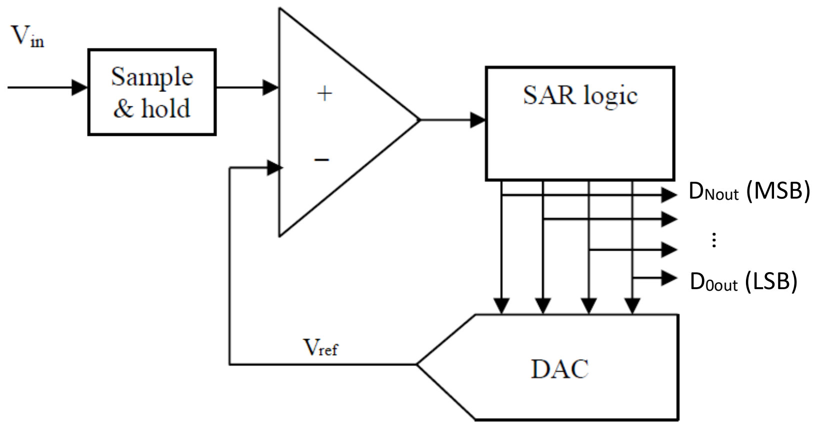

Fig. 5. Block diagram of a typical SAR-ADC [1]

The process to perform a conversion in a Successive Approximation Register Analog-to-Digital Converter (SAR-ADC) can be described as follows:

- A sample and hold circuit is employed by the system to capture the analog input signal, V_{in}. At the same time, the SAR logic resets the Digital-to-Analog Converter (DAC) to ensure proper initialization.
- The SAR logic sets the most significant bit (MSB) in the DAC. The output of the DAC is then referred to as the reference voltage ($V_{re.f}$). A comparator compares the sampled value of V_{in} with the reference voltage, $V_{re.f}$.
  - If V_{in} is greater than $V_{re.f}$, the comparator outputs a logic "1"
  - Otherwise, it produces a logic "0."
- The SAR logic stores the output of the comparator in the MSB position of the SAR output register.
- The next most significant bit (MSB-1) is set, and the partially converted value is fed into the DAC for further comparison.

This process continues, with the SAR logic repeating the steps for each successive bit (MSB-1, MSB-2, etc.), until it reaches the least significant bit (LSB). Once the LSB is evaluated, the conversion is complete, and the final digital output represents the sampled analog input.

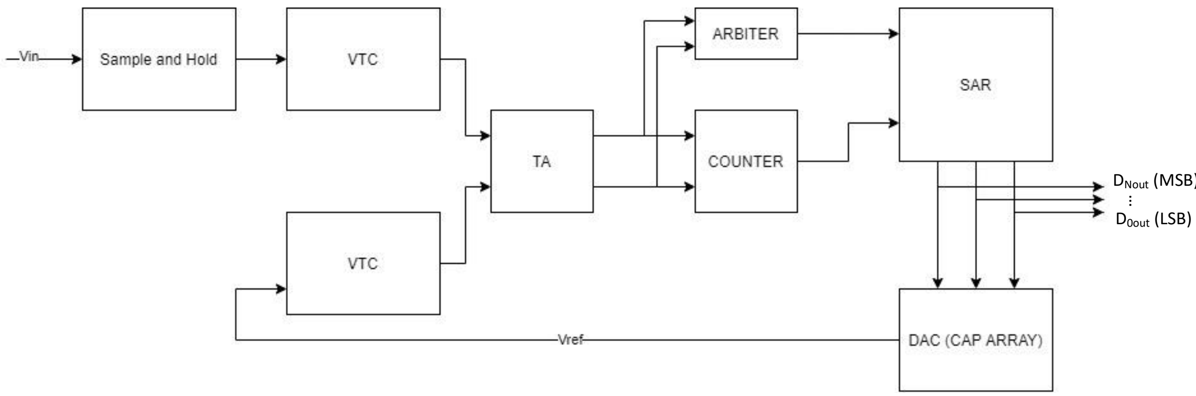

Fig. 6. Proposed SAR-ADC

The purpose of the proposed modification in [1] is to make it faster between conversions while using less power per conversion. This will be accomplished by forecasting consecutive logic “1” or “0”. This is due to the difference between V_{in} and $V_{re.f}$, The prediction code may function with 1-bit, 3-bit, and 9-bit sequentially equals bits in a single clock cycle.

The conversion process works as follows:

- The system captures the analog input, V_{in}, using a sample and hold circuit to retain the value for processing.
- Both, the analog input (V_{in}) and the reference voltage (V_{ref}), are converted into signals in the time domain through a Voltage-to-Time Converter (VTC).
- A time amplifier increases the time difference between these two signals, making the distinction more pronounced.
- The amplified time difference is passed to both, the arbiter and the counter.
- The role of the Arbiter is to determine which signal, V_{in} or $V_{re.f}$, occurred first in the time domain. Based on the information provided by the arbiter and the counter, the SAR logic can predict if consecutive bits will be the same and adjust the conversion cycles accordingly to improve the efficiency of the process.

The module developed in [1] is the “Arbiter, " which uses a TSPC D flip-flop. Its primary function is to detect whether the signal came first in the time domain. We decided to use a TSPC design because of its dynamic and high speeds, which are needed for this design. We also used a custom design because we wanted a setup time that was as short as possible. The design parameters of this custom design will be optimized in this work.

2.2. Optimization parameters

The only parameter requiring optimization is the Setup Time, as it determines the minimum duration required for the flip-flop to correctly respond to a signal at the D or Clock inputs. Both Setup Time and Hold Time are critical parameters for all flip-flops. However, for this design, Setup Time is of greater importance since the output timing from the time amplifier can be as fast as 1 ps.

2.3. Optimization process

The following describes a systematic process for optimizing a circuit designed in WinSpice and managed through MATLAB. This methodology integrates circuit simulation with numerical optimization techniques to achieve targeted design goals.

2.3.1. Problem Definition and Parameter Identification

The optimization process begins by defining the objective function, which quantifies the goal of the optimization, such as minimizing power consumption, reducing delay, or achieving specific output characteristics like gain, bandwidth, or rise time [12]. Next, design variables are identified as circuit parameters to be optimized, including transistor dimensions (e.g., W/L ratios), resistance values, capacitance values, or bias currents [13].

Finally, constraints are established to reflect physical and operational limitations, such as voltage, current, timing, or stability requirements. These constraints are essential for ensuring that the optimized solution remains feasible and functional within the real-world context [14]. By integrating these elements into the optimization process, designers can systematically improve circuit performance while adhering to practical limitations.

2.3.2. Integration of MATLAB and WinSpice

MATLAB is used to automate the optimization process and interface with WinSpice simulations. A MATLAB script is developed to perform the following tasks:

1. Generate the input files for the WinSpice simulation (e.g., .cir or .sp files) by modifying the design variables.
2. Execute the WinSpice simulation through system-level commands within MATLAB.

3. Parse the output files generated by WinSpice (e.g., .log or .raw) to extract performance metrics such as timing, power, or waveform characteristics.
4. Compute the objective function in MATLAB based on the extracted data. This setup enables an iterative loop between MATLAB and WinSpice during the optimization process.

2.3.3. Selection of Optimization Algorithm

An appropriate optimization algorithm is selected based on the characteristics of the problem:

- **Gradient-Based Methods**, such as steepest descent or quasi-Newton methods, are chosen for problems involving continuous and differentiable objective functions [12].
- **Global Optimization Techniques**, including genetic algorithms, particle swarm optimization, and simulated annealing, are suitable for tackling non-convex or multi-modal problems where local optimization methods may fail to find the global optimum [13], [15].
- For **multi-objective problems** that require balancing trade-offs, such as power versus speed, Pareto front optimization is employed to identify a set of optimal solutions offering various trade-offs among objectives [14], [16].

The optimization process begins by initializing the algorithm with feasible values for the design variables, ensuring a valid starting point for the iterative procedure. This approach enables a systematic search for optimal solutions while respecting problem-specific constraints and requirements.

2.3.4. Optimization Loop

The optimization process follows an iterative loop, as described below:

1. **Parameter Update**: The optimization algorithm updates the design variables based on the objective function and constraints.
2. **Simulation Execution**: MATLAB generates a new WinSpice input file with the updated parameters and runs the simulation.
3. **Result Extraction**: MATLAB parses the simulation output to extract performance metrics and computes the corresponding objective function value.

4. **Constraint Evaluation**: The algorithm evaluates whether the solution satisfies all constraints. If any constraints are violated, the optimization adjusts the parameters accordingly.
5. **Convergence Check**: The process continues until the objective function and design variables meet predefined convergence criteria, such as minimal objective improvement or a maximum number of iterations.

2.3.5. Post-Optimization Analysis

Once convergence is achieved, the optimized circuit is subjected to additional validation:

- **Performance Validation**: The optimized design is tested under various operating conditions, such as changes in temperature, supply voltage, or load.
- **Sensitivity Analysis**: The robustness of the design is evaluated by examining the impact of small variations in the optimized parameters on the circuit's performance.
- **Parameter Refinement**: If necessary, further adjustments are made to fine-tune the design based on observations during validation.

2.4. Implemented model from TSPC D flip-flop

Fig. 7, taken from [1], shows the proposed TSPC D flip-flop configuration, while Fig. 98 shows the corresponding performance.

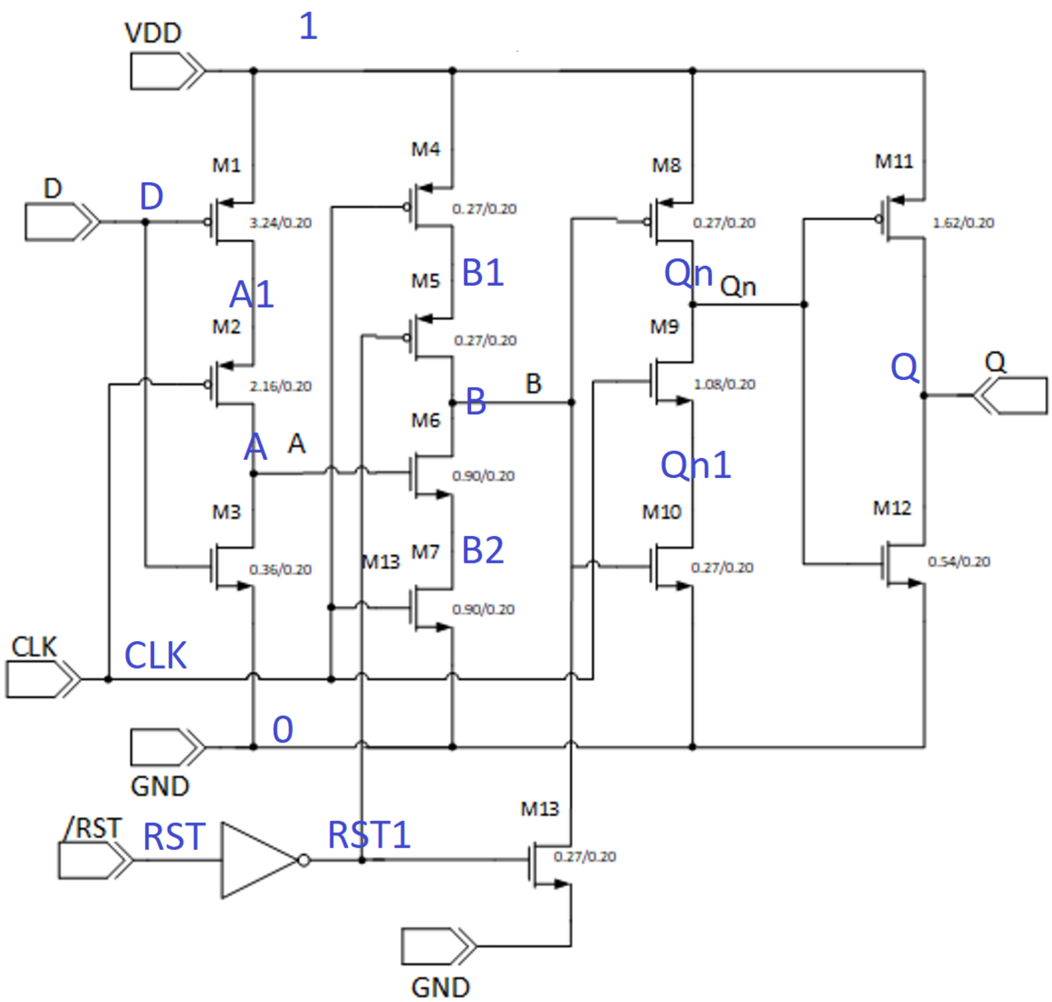

Fig. 7. Transistors model implemented in WinSpice with node identification. Taken from [1].

The corresponding performance from the implemented D flip-flop topology is shown next.

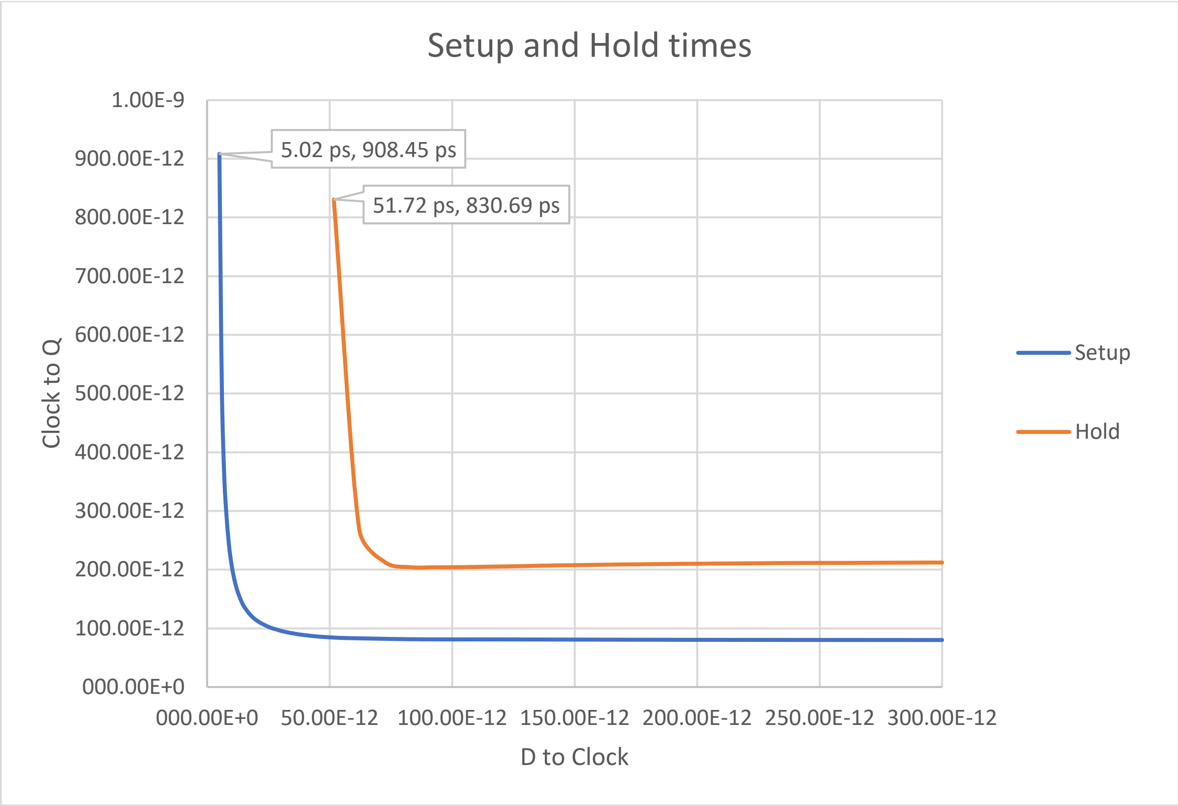

Fig. 8. Setup and Hold time for the schematic of Fig. 7. Taken from [1].

It can be seen from Fig. 8 that the original circuit in [1] has a Setup Time of 5.02 ps (x-axis), and a Hold Time of 51.72 ps (x-axis).

2.5. Driver and Plotting Functions

The following code is the MatLab Driver for interfacing WinSpice. It also describes the schematic circuit and defines the circuit variables.

```
%   Matlab Driver

function [t,D,CLK,Q,RST1] = TSPC_D_FF_SPICE(x,TSTOP,TSTART,SETUP)
delete TSPC_D_FF.csv;
% WinSpice Executable File in Command Line Mode
WinSpiceExe = 'C:\WinSpice\wspice3';

% WinSpice Project File Name
WinSpiceProjectFileName = 'TSPC_D_FF_SPICE_m.cir';

%WinSpice Script
ws{1} = 'TSPC D-FF';
ws{2} = 'VDD 1 0 DC 1.8 AC 0';
ws{3} = '';
ws{4} = ['VD D 0        pulse(0v 1.8v ' num2str(SETUP) ' 1E-15 1E-15 5000E-12 10000E-12)'];
ws{5} = ['VCLK CLK 0     pulse(0V 1.8V 300E-12 1E-15 1E-15 3500E-12 7000E-12)'];
ws{6} = 'VRST RST1 0    pulse(1.8V 0v 10E-12 1E-12 1E-12 16E-9 16E-9)';
ws{7} = '';
ws{8} = ['M1 A1 D   1  1 TSMC180nmP L=' mat2str(x.L1) 'E-6 W=' mat2str(x.W1) 'E-6'];
ws{9} = ['M2 A  CLK A1 1 TSMC180nmP L=' mat2str(x.L2) 'E-6 W=' mat2str(x.W2) 'E-6'];
ws{10} = ['M3 A  D   0  0 TSMC180nmN L=' mat2str(x.L3) 'E-6 W=' mat2str(x.W3) 'E-6'];
ws{11} = '';
ws{12} = ['M4 B1 CLK  1  1 TSMC180nmP L=' mat2str(x.L4) 'E-6 W=' mat2str(x.W4) 'E-6'];
ws{13} = ['M5 B  RST1 B1 1 TSMC180nmP L=' mat2str(x.L5) 'E-6 W=' mat2str(x.W5) 'E-6'];
ws{14} = ['M6 B  A    B2 0 TSMC180nmN L=' mat2str(x.L6) 'E-6 W=' mat2str(x.W6) 'E-6'];
ws{15} = ['M7 B2 CLK  0  0 TSMC180nmN L=' mat2str(x.L7) 'E-6 W=' mat2str(x.W7) 'E-6'];
ws{16} = '';
ws{17} = ['M8  Qn  B   1   1 TSMC180nmP L=' mat2str(x.L8) 'E-6 W=' mat2str(x.W8) 'E-6'];
ws{18} = ['M9  Qn  CLK Qn1 0 TSMC180nmN L=' mat2str(x.L9) 'E-6 W=' mat2str(x.W9) 'E-6'];
ws{19} = ['M10 Qn1 B   0   0 TSMC180nmN L=' mat2str(x.L10) 'E-6 W=' mat2str(x.W10) 'E-6'];
ws{20} = '';
ws{21} = ['M11 Q Qn 1 1 TSMC180nmP L=' mat2str(x.L11) 'E-6 W=' mat2str(x.W11) 'E-6'];
ws{22} = ['M12 Q Qn 0 0 TSMC180nmN L=' mat2str(x.L12) 'E-6 W=' mat2str(x.W12) 'E-6'];
ws{23} = '';
ws{24} = ['C1 Q 0 ' mat2str(x.C1) 'f'];
ws{25} = '';
ws{26} = 'M13 B RST1 0 0 TSMC180nmN L=200E-9 W=0.27E-6';
ws{27} = '.lib C:\Users\ivano\OneDrive - imartinez\OneDrive - ITESO\TSPC_D_FF\Masters_Degree\Use_TSPC\TSMC180nm.txt';
ws{28} = '.control';
% ws{29} = ['TRAN 1E-12 ' mat2str(TSTOP) ' ' mat2str(TSTART)];
ws{29} = ['TRAN ' num2str(TSTART) ' ' num2str(TSTOP) ' 0'];
ws{30} = 'D = (V(D))';
ws{31} = 'CLK = (V(CLK))';
ws{32} = 'Q = (V(Q))';
ws{33} = 'RST1 = (V(RST1))';
ws{34} = 'write TSPC_D_FF.csv D CLK Q RST1';
ws{35} = 'quit';
ws{36} = '.endc';
ws{37} = '.end';

% Saving WinSpice Circuit File in Matlab Working Directory
ckt_file = char(ws);
[rows,~] = size(ckt_file);
fid = fopen(WinSpiceProjectFileName,'w+');
for i = 1:rows
    fprintf(fid, '%s', ckt_file(i,:));
    fprintf(fid, '%s\n', '');
end
fclose(fid);

% Run WinSpice
system([WinSpiceExe ' ' WinSpiceProjectFileName]);

% Read WinSpice Output File
RespAC = csvread('TSPC_D_FF.csv',1,0);% Read AC Response
t = RespAC(:,1);
D = RespAC(:,2);
CLK = RespAC(:,3);
Q = RespAC(:,4);
RST1 = RespAC(:,5);
```

Fig. 9. MatLab Driver for WinSpice

The following figure describes the TMSC 0.18 um technology used to implement the TSPC D flip-flop. This file was downloaded from MOSIS[1], the manufacturer selected for this technology.

```
* PSPICE TSMC180nm.lib file  RWN  04/18/2010
* library  file for transistor parameters for TMSC 0.18 micron process
* uses BIM parameters added 01/15/98
* can configure and attach to Nbreak and Pbreak transistors in PSpice
****
*****************  180nm TSMC parameters  *************
*T14B SPICE BSIM3 VERSION 3.1 PARAMETERS
* downloaded from MOSIS 04/18/10
*http://www.mosis.com/cgi-bin/cgiwrap/umosis/swp/params/
* tsmc-018/t92y_mm_non_epi_thk_mtl_params.txt
*SPICE 3f5 Level 8, Star-HSPICE Level 49, UTMOST Level 8
* DATE: Jun  8/01
* LOT: T14B                  WAF: 06
* Temperature_parameters=Default
*$
.MODEL TSMC180nmN NMOS (                              LEVEL   = 8
+VERSION = 3.1            TNOM    = 27             TOX     = 4.1E-9
+XJ      = 1E-7           NCH     = 2.3549E17      VTH0    = 0.354505
+K1      = 0.5733393      K2      = 3.177172E-3    K3      = 27.3563303
+K3B     = -10            W0      = 2.341477E-5    NLX     = 1.906617E-7
+DVT0W   = 0              DVT1W   = 0              DVT2W   = 0
+DVT0    = 1.6751718      DVT1    = 0.4282625      DVT2    = 0.036004
+U0      = 327.3736992    UA      = -4.52726E-11   UB      = 4.46532E-19
+UC      = -4.74051E-11   VSAT    = 8.785346E4     A0      = 1.6897405
+AGS     = 0.2908676      B0      = -8.224961E-9   B1      = -1E-7
+KETA    = 0.021238       A1      = 8.00349E-4     A2      = 1
+RDSW    = 105            PRWG    = 0.5            PRWB    = -0.2
+WR      = 1              WINT    = 0              LINT    = 1.351737E-8
*+XL      = -2E-8          XW      = -1E-8
+ DWG     = 1.610448E-9
+DWB     = -5.108595E-9   VOFF    = -0.0652968     NFACTOR = 2.4901845
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB   = 0              ETA0    = 0.0231564      ETAB    = -0.058499
+DSUB    = 0.9467118      PCLM    = 0.8512348      PDIBLC1 = 0.0929526
+PDIBLC2 = 0.01           PDIBLCB = -0.1           DROUT   = 0.5224026
+PSCBE1  = 7.979323E10    PSCBE2  = 1.522921E-9    PVAG    = 0.01
+DELTA   = 0.01           RSH     = 6.8            MOBMOD  = 1
+PRT     = 0              UTE     = -1.5           KT1     = -0.11
+KT1L    = 0              KT2     = 0.022          UA1     = 4.31E-9
+UB1     = -7.61E-18      UC1     = -5.6E-11       AT      = 3.3E4
+WL      = 0              WLN     = 1              WW      = 0
+WWN     = 1              WWL     = 0              LL      = 0
+LLN     = 1              LW      = 0              LWN     = 1
+LWL     = 0              CAPMOD  = 2              XPART   = 0.5
+CGDO    = 7.7E-10        CGSO    = 7.7E-10        CGBO    = 1E-12
+CJ      = 1.010083E-3    PB      = 0.7344298      MJ      = 0.3565066
+CJSW    = 2.441707E-10   PBSW    = 0.8005503      MJSW    = 0.1327842
+CJSWG   = 3.3E-10        PBSWG   = 0.8005503      MJSWG   = 0.1327842
+CF      = 0              PVTH0   = 1.307195E-3    PRDSW   = -5
+PK2     = -1.022757E-3   WKETA   = -4.466285E-4   LKETA   = -9.715157E-3
+PU0     = 12.2704847     PUA     = 4.421816E-11   PUB     = 0
+PVSAT   = 1.707461E3     PETA0   = 1E-4           PKETA   = 2.348777E-3    )
*
*$
.MODEL TSMC180nmP PMOS (                              LEVEL   = 8
+VERSION = 3.1            TNOM    = 27             TOX     = 4.1E-9
+XJ      = 1E-7           NCH     = 4.1589E17      VTH0    = -0.4120614
+K1      = 0.5590154      K2      = 0.0353896      K3      = 0
+K3B     = 7.3774572      W0      = 1E-6           NLX     = 1.103367E-7
+DVT0W   = 0              DVT1W   = 0              DVT2W   = 0
+DVT0    = 0.4301522      DVT1    = 0.2156888      DVT2    = 0.1
+U0      = 128.7704538    UA      = 1.908676E-9    UB      = 1.686179E-21
+UC      = -9.31329E-11   VSAT    = 1.658944E5     A0      = 1.6076505
+AGS     = 0.3740519      B0      = 1.711294E-6    B1      = 4.946873E-6
+KETA    = 0.0210951      A1      = 0.0244939      A2      = 1
+RDSW    = 127.0442882    PRWG    = 0.5            PRWB    = -0.5
+WR      = 1              WINT    = 5.428484E-10   LINT    = 2.468805E-8
*+XL      = -2E-8          XW      = -1E-8
+DWG     = -2.453074E-8
+DWB     = 6.408778E-9    VOFF    = -0.0974174     NFACTOR = 1.9740447
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB   = 0              ETA0    = 0.1847491      ETAB    = -0.2531172
+DSUB    = 1.5            PCLM    = 4.8842961      PDIBLC1 = 0.0156227
+PDIBLC2 = 0.1            PDIBLCB = -1E-3          DROUT   = 0
+PSCBE1  = 1.733878E9     PSCBE2  = 5.002842E-10   PVAG    = 15
+DELTA   = 0.01           RSH     = 7.7            MOBMOD  = 1
+PRT     = 0              UTE     = -1.5           KT1     = -0.11
+KT1L    = 0              KT2     = 0.022          UA1     = 4.31E-9
+UB1     = -7.61E-18      UC1     = -5.6E-11       AT      = 3.3E4
+WL      = 0              WLN     = 1              WW      = 0
+WWN     = 1              WWL     = 0              LL      = 0
+LLN     = 1              LW      = 0              LWN     = 1
+LWL     = 0              CAPMOD  = 2              XPART   = 0.5
+CGDO    = 7.11E-10       CGSO    = 7.11E-10       CGBO    = 1E-12
+CJ      = 1.179334E-3    PB      = 0.8545261      MJ      = 0.4117753
+CJSW    = 2.215877E-10   PBSW    = 0.6162997      MJSW    = 0.2678074
+CJSWG   = 4.22E-10       PBSWG   = 0.6162997      MJSWG   = 0.2678074
+CF      = 0              PVTH0   = 2.283319E-3    PRDSW   = 5.6431992
+PK2     = 2.813503E-3    WKETA   = 2.438158E-3    LKETA   = -0.0116078
+PU0     = -2.2514581     PUA     = -7.62392E-11   PUB     = 4.502298E-24
+PVSAT   = -50            PETA0   = 1E-4           PKETA   = -1.047892E-4   )
*
.ENDS
*$
```

Fig. 10.MOSIS TMSC 0.18 um technology used to implement the TSPC D flip-flop

[1] https://www.mosis.com/cgi-bin/cgiwrap/umosis/swp/params/tsmc-018/t92y_mm_non_epi_thk_mtl_params.txt

The following code is used to generate the setup time plot.

```
TSTART = 1.6E-9;
TSTOP = 2.4E-09;

contador = 1;

for SETUP = 300E-12:-10E-12:0E-12
    [t,D,CLK,Q,RST1] = TSPC_D_FF_SPICE(x,TSTOP,TSTART,-SETUP);

    t_cell{contador}    = t;
    D_cell{contador}    = D;
    CLK_cell{contador}  = CLK;
    Q_cell{contador}    = Q;
    RST1_cell{contador} = RST1;

    subplot(3,1,1);
    plot(t,D, 'LineWidth', 2);
    title('TSPC D Flip Flop');
    hold on;

    xlabel ('Time (us)');
    ylabel ('D (V)');

    subplot(3,1,2);
    plot(t,CLK, 'LineWidth', 2);
    xlabel ('Time (us)');
    ylabel ('CLK (V)');

    subplot(3,1,3);
    plot(t,Q, 'LineWidth', 2);
    xlabel ('Time (us)');
    ylabel ('Q (V)');
    hold on;

    max_Q = max(Q);
    mean_Q = max_Q/2;
    [C_Q index_Q_setup] = min(abs(Q-mean_Q)); % In index you have the cell where the midle value is

    max_D = max(D);
    mean_D = max_D/2;
    [C_D index_D_setup] = min(abs(D-mean_D)); % In index you have the cell where the midle value is

    max_CLK = max(CLK);
    mean_CLK = max_CLK/2;
    [C_CLK index_CLK_setup] = min(abs(CLK-mean_CLK)); % In index you have the cell where the midle value is
    % CLK TO Q
    CLK2Q_SETUP(contador) = abs(t(index_CLK_setup) - t(index_Q_setup));
    % D TO CLK
    D2CLK_SETUP(contador) = abs(t(index_D_setup) - t(index_CLK_setup));

    contador = contador + 1;
end

    figure
    plot(D2CLK_SETUP, CLK2Q_SETUP)
```

Fig. 11. Code used to generate a set up time plot.

2.6. Optimization Functions

MATLAB offers a variety of optimization algorithms that are well-suited for different types of design problems. Among these, ***fmincon***, ***fminsearch***, and ***GA*** (**Genetic Algorithms)** are commonly used for numerical optimization. Each algorithm is tailored to address specific problem types, ranging from constrained optimization to global search techniques. Below, the functionality, description, advantages, and disadvantages of each algorithm are discussed [12] – [15].

2.6.1. fmincon

The fmincon function in MATLAB is designed for solving constrained nonlinear optimization problems. It minimizes a scalar objective function, subject to equality and inequality constraints, as well as bound constraints on variables [12].

Description:

- fmincon uses gradient-based methods such as Sequential Quadratic Programming (SQP), Trust-Region-Reflective, and Interior-Point algorithms.
- It requires the objective function and constraints to be smooth and differentiable, as the algorithm relies on gradient calculations.
- The user can specify additional settings, such as tolerances and maximum iterations, to customize the optimization process.

Advantages:

1. Highly efficient for problems with smooth and differentiable objective and constraint functions.
2. Supports complex constraint formulations, including nonlinear equality and inequality constraints.
3. Offers multiple solver options, providing flexibility for different problem types.

Disadvantages:

1. Prone to converging to local optima in non-convex problems.
2. Requires gradient information, which may be computationally expensive to calculate for complex functions.
3. Not suitable for problems with discontinuous or non-differentiable functions.

The following figure shows the code used to implement the fmincon function. For this optimization process, lower bound values for L and W were set to 0.2 um, while upper bound for W and L were set to 10 um and 0.8 um, respectively.

```
lb = [0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2];
ub = [10 10 10 10 10 10 10 10 10 10 10 10 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8];
A = [];
b = [];
Aeq = [];
beq = [];
nonlcon = [];

options = optimset('MaxFunEvals',Inf,'MaxIter',Inf,'TolX',1e-6,'TolFun',1e-6);
[Xopt,FVAL,EXITFLAG,OUTPUT,LAMBDA,GRAD] = fmincon(@optimizacion,x,A,b,Aeq,beq,lb,ub,nonlcon,options);
```

Fig. 12. Optimization code used for implementing fmincon function.

2.6.2. fminsearch

The fminsearch function is a derivative-free optimization algorithm based on the Nelder-Mead simplex method. It minimizes unconstrained scalar objective functions [13].

Description:

- Unlike fmincon, fminsearch does not require gradient information and is particularly suited for unconstrained optimization problems.
- It evaluates the objective function at a series of points in the search space and updates the points iteratively to approach the minimum.

Advantages:

1. Simple to implement and does not require gradient information, making it suitable for non-differentiable or noisy functions.
2. Effective for small-scale problems and problems where gradient information is unavailable.
3. Computationally inexpensive for problems with a low number of variables.

Disadvantages:

1. Limited to unconstrained optimization, making it unsuitable for problems with constraints.
2. May perform poorly in high-dimensional spaces or for functions with many local minima.
3. Convergence can be slow for complex objective functions.

The following figure shows the code used for the fminsearch function. The values of W and L correspond to the initial design before optimization process begins.

```
%Values from W for each MOSFET
W1 = 3.24;
W2 = 2.16;
W3 = 0.36;
W4 = 0.27;
W5 = 0.27;
W6 = 0.90;
W7 = 0.90;

W8 = 0.27;
W9 = 0.18;
W10 = 0.27;
W11 = 1.62;
W12 = 0.54;

L1 = 0.2;
L2 = 0.2;
L3 = 0.2;
L4 = 0.2;
L5 = 0.2;
L6 = 0.2;
L7 = 0.2;
L8 = 0.2;
L9 = 0.2;
L10 = 0.2;
L11 = 0.2;
L12 = 0.2;

x  = [W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12];

options = optimset('MaxFunEvals',Inf,'MaxIter',Inf,'TolX',1e-6,'TolFun',1e-6);
[Xopt,fval,exitflag,output] = fminsearch(@optimizacion,x,options);
```

Fig. 13. Optimization code for implementing the fminsearch function.

2.6.3. GA (Genetic Algorithms)

Genetic Algorithms are global optimization techniques inspired by the principles of natural selection and evolution. MATLAB provides GA capabilities through its Global Optimization Toolbox [14], [15].

Description:

- GA uses a population-based search approach, where candidate solutions (individuals) evolve through operators such as selection, crossover, and mutation.
- It is well-suited for complex, non-linear, non-differentiable, and multi-modal objective functions.
- The algorithm is robust to noisy data and can handle discrete, mixed-integer, and continuous variables.

Advantages:

- Capable of escaping local optima, making it suitable for non-convex problems.
- Handles non-smooth, discontinuous, and noisy objective functions.

- Flexible in handling a mix of variable types (e.g., discrete and continuous).
- Does not require gradient information.

Disadvantages:

- Computationally expensive, as it requires multiple evaluations of the objective function for a large population.
- Slower convergence compared to gradient-based methods for smooth and convex problems.
- Requires careful tuning of parameters (e.g., population size, mutation rate) for optimal performance.
- Does not guarantee finding the exact global optimum.

The following figure shows the implemented code for GA algorithm. Lower and upper bounds are specified as well as the initial circuit design.

```
%Values from W for each MOSFET
W1 = 3.24;
W2 = 2.16;
W3 = 0.36;
W4 = 0.27;
W5 = 0.27;
W6 = 0.90;
W7 = 0.90;

W8 = 0.27;
W9 = 0.18;
W10 = 0.27;
W11 = 1.62;
W12 = 0.54;

L1 = 0.2;
L2 = 0.2;
L3 = 0.2;
L4 = 0.2;
L5 = 0.2;
L6 = 0.2;
L7 = 0.2;
L8 = 0.2;
L9 = 0.2;
L10 = 0.2;
L11 = 0.2;
L12 = 0.2;

x  = [W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12];

lb = [0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2];
ub = [10 10 10 10 10 10 10 10 10 10 10 10 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8];
A = [];
b = [];
Aeq = [];
beq = [];
nonlcon = [];
intcon = [];
nvars = 24;

options = optimset('MaxFunEvals',Inf,'MaxIter',Inf,'TolX',1e-6,'TolFun',1e-6);
[Xopt,fval,exitflag,output] = ga({@optimizacion},nvars,A,b,Aeq,beq,lb,ub,nonlcon,intcon,options);
```

Fig. 14. Optimization code for implementing the Genetic Algorithm

3. Results of optimization

The following figures show the corresponding optimization process results for each of the three implemented optimization algorithms.

3.1. fmincon

Figure 15 shows the corresponding optimization result when using *fmincon* algorithm. It can be seen that there is a not valid region between 0 and 250 ps. Therefore, the starting point is located around 300 ps and the optimal model is located around 250 ps (x-axis).

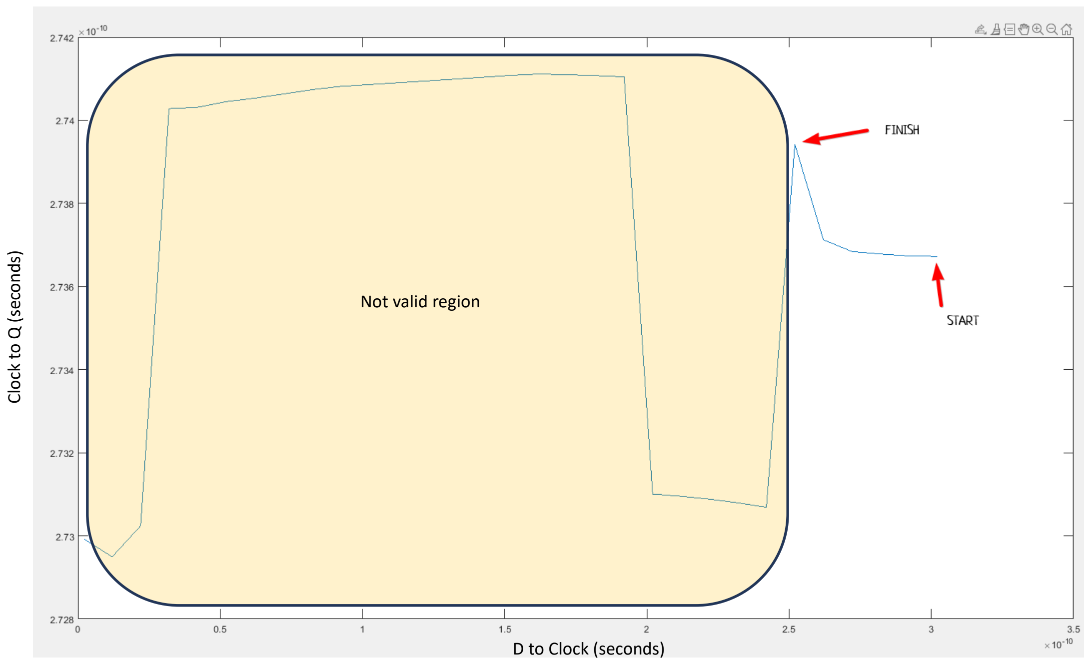

Fig. 15. Setup Time optimization plot using fmincon.

3.2. fminsearch

Figure 16 shows the corresponding optimization result when using *fminsearch* algorithm. It can be seen that also the starting point is located around 300 ps but in this case the optimal model response is located around 50 ps (x-axis), which is better than the resulting model using *fmincon*, since it is closer to 0 ps (x-axis).

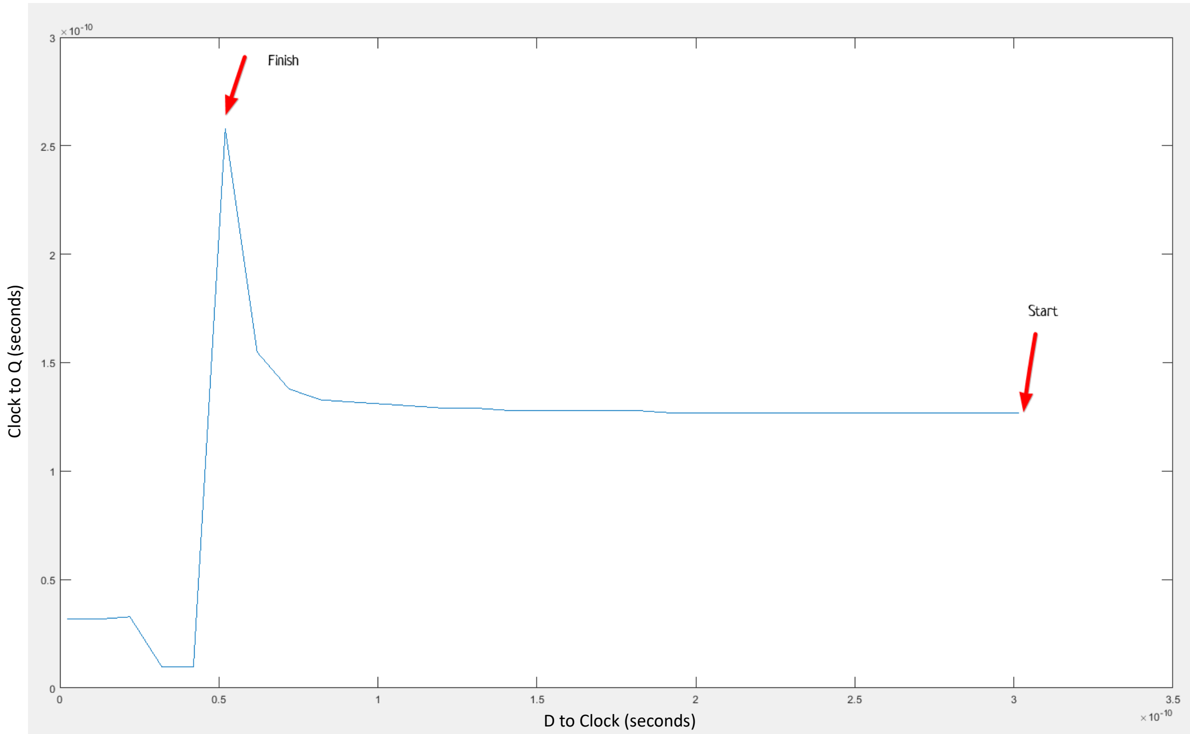

Fig. 16. Setup Time optimization plot using fminsearch.

3.3. Genetic Algorithm

Figure 17 shows the corresponding optimization result when using *GA* algorithm. It can be seen that the starting point is located around 300 ps and the optimal model response is located near to 0 ps (x-axis). Therefore, this response is the best among the three optimization models. The only downside is the time it takes to complete the optimization process, which is about 24 hours.

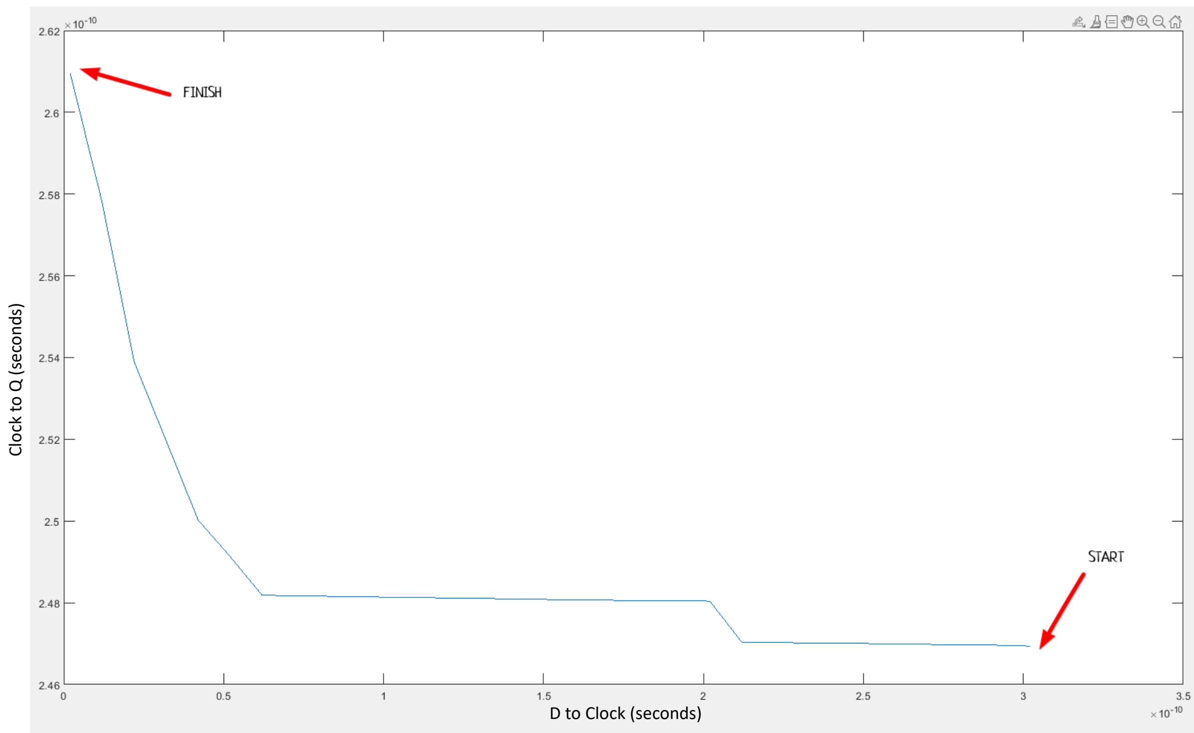

Fig. 17. Setup Time optimization plot using GA.

Conclusions

The objective of this work was to explore whether it is feasible to improve the setup time of a True Single-Phase Clock (TSPC) D flip-flop (D FF) using numerical methods. The significance of reducing setup time becomes evident in the context of a previous design involving a Successive Approximation Register Analog-to-Digital Converter (SAR ADC), which was converted from a traditional voltage-based architecture to a time-based design. While most designs follow a conventional voltage-based approach, this transition to a time-domain architecture introduced new challenges, with setup time being a critical parameter affecting overall performance.

In the course of this investigation, one of the most crucial components was the SPICE simulation driver. For the purposes of this study, WinSpice was employed as the simulation environment. The correct execution of the SPICE driver was essential, as it provided the foundation for building a reliable model capable of accurately predicting the output states based on varying input conditions. The SPICE model utilized nodes to represent the physical connections between transistors, thereby enabling the simulation of the circuit's behavior. Once the driver was correctly configured, an ASCII-based SPICE script was written to facilitate the simulation process.

In contemporary circuit design, a wide variety of transistor models are readily available from different manufacturers. However, for the purposes of this research, a custom design was developed. This decision was made to ensure the design could meet specific requirements, particularly in terms of setup time and power consumption. The custom model provided greater control and flexibility, allowing the investigation to focus on optimizing these parameters.

Three different optimization algorithms were considered during the course of this work. Among them, the Genetic Algorithm yielded the best results, outperforming the other methods, including fmincon and fminsearch, in terms of accuracy. The Genetic Algorithm proved to be the most effective in refining the setup time of the TSPC D FF.

Furthermore, the developed code and methodologies have broader applications beyond this specific case study. The optimization techniques used here can be applied to other areas of circuit design, particularly in the context of optimizing transistor sizing for various applications.

This research not only highlights the potential of numerical methods in improving setup time but also demonstrates their utility in enhancing the overall efficiency and performance of digital circuits in a wide range of applications.

Future areas of opportunity

Based on the work described, several future research lines can be explored to build on the findings of this investigation:

1. Optimization of Setup Time for Other flip-flop Architectures. Extending the optimization techniques, such as the genetic algorithm, to other flip-flop designs (e.g., C2MOS or TG-based FFs) could further enhance performance in different digital circuits, particularly in high-speed or low-power applications.
2. Time-Based ADC Design Innovations. Investigating alternative time-based ADC architectures that leverage the optimizations discovered in this work could lead to more efficient SAR ADC designs, focusing on minimizing power consumption and improving accuracy.
3. Integration of Machine Learning for Circuit Optimization. Exploring the application of machine learning algorithms to optimize setup time, power consumption, and transistor sizing in custom circuits could further automate and improve the design process beyond traditional numeric methods.
4. Development of Advanced SPICE Models for Emerging Technologies. Creating enhanced SPICE models tailored to new transistor technologies such as FinFET, GAAFET, or nanosheet transistors could provide more accurate simulations and optimizations for next-generation circuits.
5. Multi-objective Optimization for Power-Performance Trade-offs. Expanding the optimization to consider not only setup time but also other critical factors such as power efficiency, area, and signal integrity in a multi-objective framework would benefit applications requiring balanced trade-offs.

6. Adapting Time-Domain Circuit Design for IoT Devices. Investigating how time-based design techniques can be adapted to ultra-low-power devices, particularly in the context of the Internet of Things (IoT), where energy efficiency and compactness are critical.
7. Exploration of Quantum and Neuromorphic Computing Circuits. Extending the methodologies for optimizing flip-flop setup time to circuits used in quantum or neuromorphic computing could address the unique demands of these cutting-edge fields, where timing accuracy and energy consumption are of utmost importance.

References

1. Martínez-Flores, I. (2020). *A 0.18um CMOS TSPC D flip-flop as an Arbiter for a low power 10-bits 200kS/s ADC with Adaptive Conversion Cycle Oriented to Audio Applications*. Retrieved from https://rei.iteso.mx/server/api/core/bitstreams/1b3ddf9b-bae5-4a0a-9c71-d7ed58873b3b/content
2. Thorn EMI Television Rentals. (1984). *An Introduction to Digital Techniques: Logic Gates, Flip-Flops, Counters and Shift Registers*. Thorn EMI Macmillan series. Macmillan Education.
3. Rayas-Sánchez, J. E. (s. f.). *Driving WinSpice from MATLAB - Simple Example*. Retrieved from https://desi.iteso.mx/erayas/documents/cad_course/lectures/SPICE/WinSpice_from_Matlab_ex_RLC.pdf
4. Niknejad, P. A. M. (2010). *Lecture 24: Latches and Flip Flops*. Retrieved from https://studylib.net/doc/11094727/electronics-ee-42-100-lecture-24--latches-and-flip-flops
5. Floyd, T. L. (2015). *Digital Fundamentals* (11th ed.). Upper Saddle River, NJ: Pearson.
6. Kinney, L. L., & Roth Jr., C. H. (2013). *Fundamentals of Logic Design* (7th ed.). Boston, MA: Cengage Learning.
7. Givone, D. D. (2003). *Digital Principles and Design*. New York, NY: McGraw-Hill.
8. Mano, M., & Ciletti, M. D. (2017). *Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog* (6th ed.). Boston, MA: Pearson.
9. Hayes, J. P. (1993). *Introduction to Digital Logic Design*. Boston, MA: Addison-Wesley.
10. Wakerly, J. F. (2005). *Digital Design: Principles and Practices* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

11. Rabaey, J. M. (2003). *Digital Integrated Circuits: A Design Perspective* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
12. Bertsimas, D., & Tsitsiklis, J. N. (1997). *Introduction to Linear Optimization*. Belmont, MA: Athena Scientific.
13. Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY: Wiley.
14. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.
15. Hillier, F. S., & Lieberman, G. J. (2021). *Introduction to Operations Research* (11th ed.). New York, NY: McGraw-Hill.
16. Sheffi, Y. (1985). *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Englewood Cliffs, NJ: Prentice Hall.
17. Tocci, R. J., Widmer, N. S., & Moss, G. L. (2017). *Digital systems: Principles and applications* (12th ed.). Pearson.

Appendix A

Complete Code of TEST_TSPC_D_FF_SPICE.m

```
clear
clc
% close all

% format short g;

ti = clock;

%Values from W for each MOSFET
W1 = 3.24;
W2 = 2.16;
W3 = 0.36;
W4 = 0.27;
W5 = 0.27;
W6 = 0.90;
W7 = 0.90;

W8 = 0.27;
W9 = 0.18;
W10 = 0.27;
W11 = 1.62;
W12 = 0.54;

L1 = 0.2;
L2 = 0.2;
L3 = 0.2;
L4 = 0.2;
L5 = 0.2;
L6 = 0.2;
L7 = 0.2;
L8 = 0.2;
L9 = 0.2;
L10 = 0.2;
L11 = 0.2;
L12 = 0.2;

x  = [W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 L1 L2 L3 L4 L5 L6 L7 L8 L9
L10 L11 L12];

lb = [0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2];
ub = [10 10 10 10 10 10 10 10 10 10 10 10 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8
0.8 0.8 0.8 0.8];
A = [];
b = [];
Aeq = [];
beq = [];
nonlcon = [];
intcon = [];
nvars = 24;

figure1 = figure('Position', [0, 0, 1800, 1000]);

% Para generar 'setup' numero de Ds que a su vez genera diferentes
valores
```

```
% de Q

TSTART = 1.6E-9;
TSTOP = 2.4E-09;

contador = 1;

for SETUP = 300E-12:-10E-12:0E-12
    [t,D,CLK,Q,RST1] = TSPC_D_FF_SPICE(x,TSTOP,TSTART,-SETUP);

    t_cell{contador}    = t;
    D_cell{contador}    = D;
    CLK_cell{contador}  = CLK;
    Q_cell{contador}    = Q;
    RST1_cell{contador} = RST1;

    subplot(3,1,1);
    plot(t,D, 'LineWidth', 2);
    title('TSPC D flip Flop');
    hold on;

    xlabel ('Time (us)');
    ylabel ('D (V)');

    subplot(3,1,2);
    plot(t,CLK, 'LineWidth', 2);
    xlabel ('Time (us)');
    ylabel ('CLK (V)');

    subplot(3,1,3);
    plot(t,Q, 'LineWidth', 2);
    xlabel ('Time (us)');
    ylabel ('Q (V)');
    hold on;

    max_Q = max(Q);
    mean_Q = max_Q/2;
    [C_Q index_Q_setup] = min(abs(Q-mean_Q)); % In index, you have the
cell where the middle value is

    max_D = max(D);
    mean_D = max_D/2;
    [C_D index_D_setup] = min(abs(D-mean_D)); % In index, you have the
cell where the midle value is

    max_CLK = max(CLK);
    mean_CLK = max_CLK/2;
    [C_CLK index_CLK_setup] = min(abs(CLK-mean_CLK)); % In index, you
have the cell where the midle value is
    % CLK TO Q
    CLK2Q_SETUP(contador) = abs(t(index_CLK_setup) - t(index_Q_setup));
    % D TO CLK
    D2CLK_SETUP(contador) = abs(t(index_D_setup) - t(index_CLK_setup));

    contador = contador + 1;
end

    figure2 = figure('Position', [0, 0, 1800, 1000]);
    plot(D2CLK_SETUP, CLK2Q_SETUP)
```

```
options = optimset('MaxFunEvals',Inf,'MaxIter',Inf,'TolX',1e-
6,'TolFun',1e-6);

% [Xopt,FVAL,EXITFLAG,OUTPUT,LAMBDA,GRAD] =
fmincon(@optimizacion,x,A,b,Aeq,beq,lb,ub,nonlcon,options);
% [Xopt,fval,exitflag,output] = fminsearch(@optimizacion,x,options);
[Xopt,fval,exitflag,output] =
ga({@optimizacion},nvars,A,b,Aeq,beq,lb,ub,nonlcon,intcon,options);

figure3 = figure('Position', [0, 0, 1800, 1000]);

TSTART = 1.6E-9;
TSTOP = 2.4E-09;

contador = 1;

for SETUP = 300E-12:-10E-12:0E-12
    [t,D,CLK,Q,RST1] = TSPC_D_FF_SPICE(Xopt,TSTOP,TSTART,-SETUP);

    t_cell{contador}    = t;
    D_cell{contador}    = D;
    CLK_cell{contador}  = CLK;
    Q_cell{contador}    = Q;
    RST1_cell{contador} = RST1;

    subplot(3,1,1);
    plot(t,D, 'LineWidth', 2);
    title('TSPC D flip Flop Opt');
    hold on;

    xlabel ('Time (us)');
    ylabel ('D (V)');

    subplot(3,1,2);
    plot(t,CLK, 'LineWidth', 2);
    xlabel ('Time (us)');
    ylabel ('CLK (V)');

    subplot(3,1,3);
    plot(t,Q, 'LineWidth', 2);
    xlabel ('Time (us)');
    ylabel ('Q (V)');
    hold on;

    max_Q = max(Q);
    mean_Q = max_Q/2;
    [C_Q index_Q_opt] = min(abs(Q-mean_Q)); % In index, you have the cell
where the midle value is

    max_D = max(D);
    mean_D = max_D/2;
    [C_D index_D_opt] = min(abs(D-mean_D)); % In index, you have the cell
where the midle value is

    max_CLK = max(CLK);
    mean_CLK = max_CLK/2;
    [C_CLK index_CLK_opt] = min(abs(CLK-mean_CLK)); % In index, you have
the cell where the midle value is
    % CLK TO Q
```

```
    CLK2Q_opt(contador) = abs(t(index_CLK_opt) - t(index_Q_opt));
    % D TO CLK
    D2CLK_opt(contador) = abs(t(index_D_opt) - t(index_CLK_opt));

    contador = contador + 1;
end

    figure4 = figure('Position', [0, 0, 1800, 1000]);
    plot(D2CLK_opt, CLK2Q_opt)
```

Full Code of TSPC_D_FF_SPICE.m

```
%   Matlab Driver
function [t,D,CLK,Q,RST1] = TSPC_D_FF_SPICE(x,TSTOP,TSTART,SETUP)

% Matlab Driver modified from
% desi.iteso.mx/erayas/documents/cad_course/lectures/
SPICE/WinSpice_from_Matlab_ex_RLC.pdf

% WinSpice Executable File in Command Line Mode
WinSpiceExe = 'C:\WinSpice\wspice3';

% WinSpice Project File Name
WinSpiceProjectFileName = 'TSPC_D_FF_SPICE_m.cir';

% Design Variables

W1 = x(1);
W2 = x(2);
W3 = x(3);
W4 = x(4);
W5 = x(5);
W6 = x(6);
W7 = x(7);
W8 = x(8);
W9 = x(9);
W10 = x(10);
W11 = x(11);
W12 = x(12);
L1 = x(13);
L2 = x(14);
L3 = x(15);
L4 = x(16);
L5 = x(17);
L6 = x(18);
L7 = x(19);
L8 = x(20);
L9 = x(21);
L10 = x(22);
L11 = x(23);
L12 = x(24);

%WinSpice Script
ws{1} = 'TSPC D-FF';
ws{2} = 'VDD 1 0 DC 1.8 AC 0';
ws{3} = '';
ws{4} = ['VD D 0 PWL(0 0V ' mat2str(SETUP + 1.998E-9) ' 0V '
mat2str(SETUP + 1.999e-9) ' 1.8V)'];
```

```
% pulse(0V 1.8V 1E-9 1E-12 1E-12 3800E-12 8E-9)';
ws{5} = 'VCLK CLK 0 pulse(0V 1.8V 0E-12 1E-12 1E-12 950E-12 2E-9)';
ws{6} = 'VRST RST1 0 pulse(1.8V 0v 1E-9 1E-12 1E-12 16E-9 16E-9)';
ws{7} = '';
ws{8} = ['M1 A1 D   1  1 TSMC180nmP L=' mat2str(L1) 'E-6 W=' mat2str(W1)
'E-6'];
ws{9} = ['M2 A  CLK A1 1 TSMC180nmP L=' mat2str(L2) 'E-6 W=' mat2str(W2)
'E-6'];
ws{10} = ['M3 A  D   0  0 TSMC180nmN L=' mat2str(L3) 'E-6 W=' mat2str(W3)
'E-6'];
ws{11} = '';
ws{12} = ['M4 B1 CLK  1  1 TSMC180nmP L=' mat2str(L4) 'E-6 W='
mat2str(W4) 'E-6'];
ws{13} = ['M5 B  RST1 B1 1 TSMC180nmP L=' mat2str(L5) 'E-6 W='
mat2str(W5) 'E-6'];
ws{14} = ['M6 B  A    B2 0 TSMC180nmN L=' mat2str(L6) 'E-6 W='
mat2str(W6) 'E-6'];
ws{15} = ['M7 B2 CLK  0  0 TSMC180nmN L=' mat2str(L7) 'E-6 W='
mat2str(W7) 'E-6'];
ws{16} = '';
ws{17} = ['M8  Qn  B   1   1 TSMC180nmP L=' mat2str(L8) 'E-6 W='
mat2str(W8) 'E-6'];
ws{18} = ['M9  Qn  CLK Qn1 0 TSMC180nmN L=' mat2str(L9) 'E-6 W='
mat2str(W9) 'E-6'];
ws{19} = ['M10 Qn1 B   0   0 TSMC180nmN L=' mat2str(L10) 'E-6 W='
mat2str(W10) 'E-6'];
ws{20} = '';
ws{21} = ['M11 Q Qn 1 1 TSMC180nmP L=' mat2str(L11) 'E-6 W=' mat2str(W11)
'E-6'];
ws{22} = ['M12 Q Qn 0 0 TSMC180nmN L=' mat2str(L12) 'E-6 W=' mat2str(W12)
'E-6'];
ws{23} = '';
ws{24} = 'C1 Q 0 10F';
ws{25} = '';
ws{26} = 'M13 B RST1 0 0 TSMC180nmN L=200E-9 W=0.27E-6';
ws{27} = '.lib C:\Users\ivano\OneDrive - imartinez\OneDrive -
ITESO\TSPC_D_FF\Masters_Degree\TSMC180nm.txt';
ws{28} = '.control';
ws{29} = ['TRAN 1E-12 ' mat2str(TSTOP) ' ' mat2str(TSTART)];
ws{30} = 'D = mag(V(D))';
ws{31} = 'CLK = mag(V(CLK))';
ws{32} = 'Q = mag(V(Q))';
ws{33} = 'RST1 = mag(V(RST1))';
ws{34} = 'write TSPC_D_FF.csv D CLK Q RST1';
ws{35} = 'quit';
ws{36} = '.endc';
ws{37} = '.end';

% Saving WinSpice Circuit File in Matlab Working Directory
ckt_file = char(ws);
[rows,~] = size(ckt_file);
fid = fopen(WinSpiceProjectFileName,'w+');
for i = 1:rows
    fprintf(fid, '%s', ckt_file(i,:));
    fprintf(fid, '%s\n', '');
end
fclose(fid);

% Run WinSpice
system([WinSpiceExe ' ' WinSpiceProjectFileName]);

% Read WinSpice Output File
```

```
RespAC = csvread('TSPC_D_FF.csv',1,0);% Read AC Response
t = RespAC(:,1);
D = RespAC(:,2);
CLK = RespAC(:,3);
Q = RespAC(:,4);
RST1 = RespAC(:,5);

% Erase Previous CSV and txt Data Files
% delete TSPC_D_FF.csv;
```

Full code of optimization.m

```
function u = optimizacion(x)
TSTART = 1.6E-9;
TSTOP = 2.4E-09;
contador = 1;

for SETUP = 240E-12:1E-12:245E-12
    [t,D,CLK,Q,RST1] = TSPC_D_FF_SPICE(x,TSTOP,TSTART,SETUP);

    t_cell{contador}    = t;
    D_cell{contador}    = D;
    CLK_cell{contador}  = CLK;
    Q_cell{contador}    = Q;
    RST1_cell{contador} = RST1;

    max_Q = max(Q);
    mean_Q = max_Q/2;
    [C_Q index_Q] = min(abs(Q-mean_Q)); %en index tienes la celda en la
cual se encuentra el valor medio

    max_D = max(D);
    mean_D = max_D/2;
    [C_D index_D] = min(abs(D-mean_D)); %en index tienes la celda en la
cual se encuentra el valor medio

    max_CLK = max(CLK);
    mean_CLK = max_CLK/2;
    [C_CLK index_CLK] = min(abs(CLK-mean_CLK)); %en index tienes la celda
en la cual se encuentra el valor medio

    % CLK TO Q
    CLK2Q(contador) = t(index_CLK) - t(index_Q);

    % D TO CLK
    D2CLK(contador) = t(index_D) - t(index_CLK);

    contador = contador + 1;

end

u = min(D2CLK); % Norma Euclidiana
```

Index