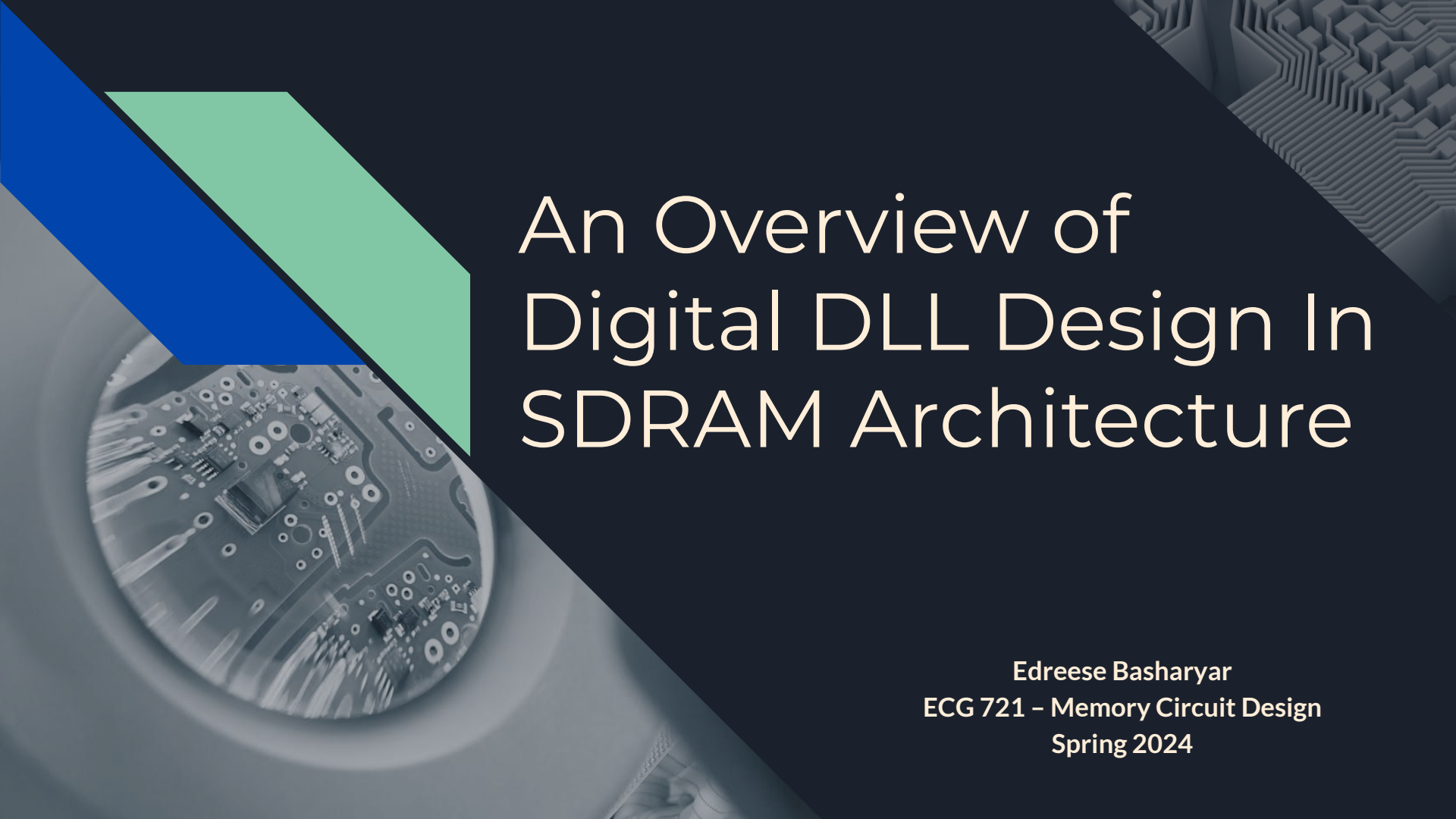


An Overview of Digital DLL Design In SDRAM Architecture

Edreese Basharyar

ECG 721 – Memory Circuit Design

Spring 2024

Table of Contents

What are DLLs ?

- A **Delay Locked Loop (DLL)** is an electronic control system that dynamically adjusts the phase of the output clock signal to match or synchronize with a given reference clock signal.

Why should these clock signals be synchronized anyways?

- In digital systems, synchronized clock signals ensure everything runs with precision, efficiency, and seamless coordination. When these signals are out of sync, they may act prematurely or delay their response, leading to confusion and errors within the system. This disruption can significantly degrade system performance. Therefore, as digital systems increase in speed, achieving highly precise synchronization becomes increasingly critical to maintain optimal function and performance.

So, what are the main purposes of the DLL?

- **DLLs** are extensively used for timing adjustment and synchronization in high-speed electronic systems.
  - The whole purpose of **DLLs** is to essentially minimize the **skew**, which is the difference in timing between when a clock signal is supposed to arrive and when it actually arrives.
  - They are able to synchronize the data output with the clock edges to improve the overall signal integrity and reliability of systems such as **DRAM/SDRAM**.
  - Another benefit of aligning the clock signals include reducing **jitter,** which is the variation in the timing of clock pulses.

Why DLLs over PLLs?

- DLLs do not contain any oscillators, which provides more stability and less jitter, and phase noise is no longer a worrying factor.
- Since DLLs operate on delaying an existing clock signal rather than generating a new one, the behavior is more controlled and less susceptible to possible variations.
- The DLL exhibits a first-order response in terms of controls, due to its singular feedback loop, steady-state behaviour without oscillation, and no accumulation of phase error.

SDRAM and Its Evolution

- **Synchronous Dynamic Random Access Memory (known as SDRAM)** is a type of DRAM that is favored in modern computer systems due to its high-speed capabilities, outpacing the earlier asynchronous DRAM models.
- Traditional forms of DRAM operate asynchronously, since they react to changes as the control inputs change, handling tasks one at a time, sequentially as they arrive.
- SDRAM was initially developed in the 1990s to overcome the limitations of processors with asynchronous interfaces that caused delays in input control signals, since SDRAM is synchronized to the processor's clock and the system bus.
- **If the SDRAM needs to synchronize with the processor's clock, what better innovation than to introduce the DLL into its architecture, whose function is to align clock signals together? This is why the DLL has become a critical component in most SDRAMs made today, as they are able to enhance their efficiency and performance by maintaining tight timing alignment with the processor's clock.**

DDR SDRAMs:

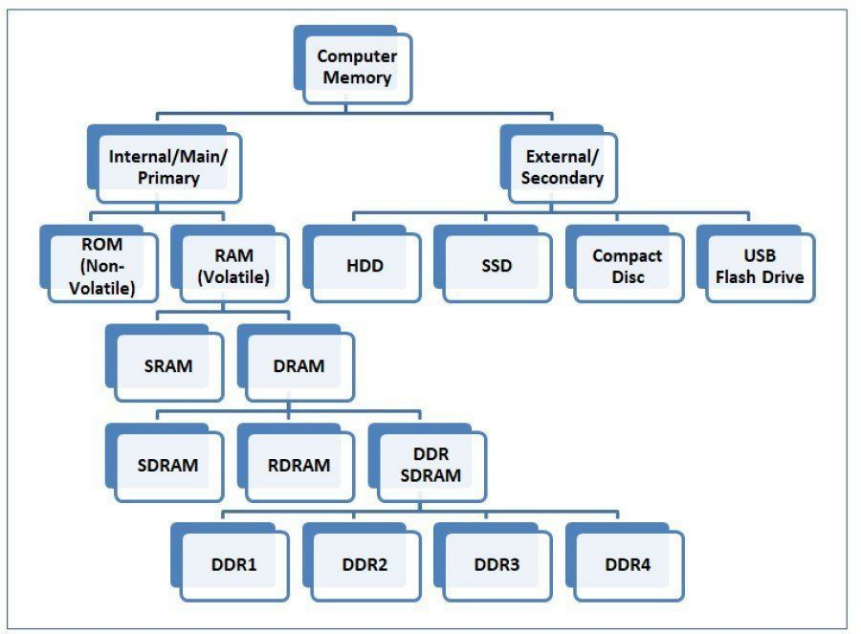
- The evolution of SDRAM technology culminated due to the concept of double data rate, also known as **DDR**, which enhanced data transfer speeds.
- Initially, **SDR (Single Data Rate) SDRAM**, which was introduced in 1993, was the foundation for SDRAM models, where data was transmitted on every single rising edge of each clock cycle.
- **DDR1 SDRAM** was introduced in 2000, and its name DDR signified its capability to double the data transfer rates of its SDR counterparts, since it was able to transmit data on both the rising and falling edges of each clock cycle.
- **DDR2 SDRAM** was introduced in 2003, which not only doubled the data transfer rate but also doubled the clock speed, which essentially quadruples the overall bandwidth of the SDR SDRAM.
- **DDR3 SDRAM** was introduced in 2007, and it continued to push the boundaries by continuing to double the transfer rate of its predecessor, while also reducing the amount of power consumption, making it energy efficient.
- **DDR4 SDRAM** was introduced in 2014, which set a new benchmark, as data transfer rates were at its highest, and voltage requirements continued to lower, and it became an ideal choice for high-end computing applications.

DDR5, the Newest Generation:

- **DDR5 SDRAM**, launched in 2021, represents the latest and most significant advancement in SDRAM technology.
- This generation has successfully doubled the data transfer rates of DDR4, operates at lower voltages, and offers increased memory capacity. As time progresses, DDR5 continues to push the boundaries of memory technology.

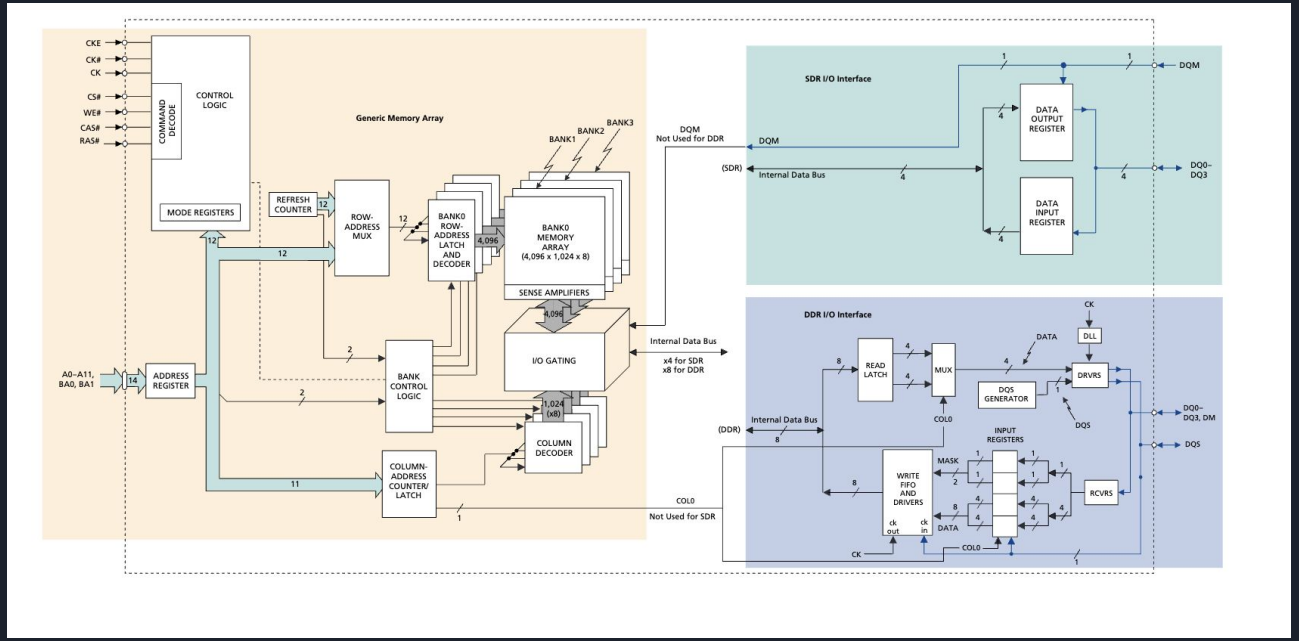
SDRAM and Its Evolution

- **Prefetch**: The amount of data bits fetched from the memory array in one memory access cycle.
- **Data Rate**: The speed at which data is read from or written to the memory, typically measured in MT/s (MegaTransfers per second), indicating millions of data units processed per second.
- **Transfer Rate**: The total rate at which data is transferred between the DDR SDRAM and the system.
- **Voltage**: The operational voltage level required by DDR SDRAM.



	DRAM	DDR	DDR2	DDR3	DDR4	DDR5
Prefetch	1 – Bit	2 - Bit	4 - Bit	8 - Bit	Bit per Bank	16 - Bit
Data Rate (MT/s)	100 - 166	266 - 400	533 - 800	1066 - 1600	2133 - 5100	3200 - 6400
Transfer Rate (GB/s)	0.8 - 1.3	2.1 - 3.2	4.2 - 6.4	8.5 - 14.9	17 - 25.6	38.4 - 51.2
Voltage (V)	3.3	2.5 - 2.6	1.8	1.35 - 1.5	1.2	1.1

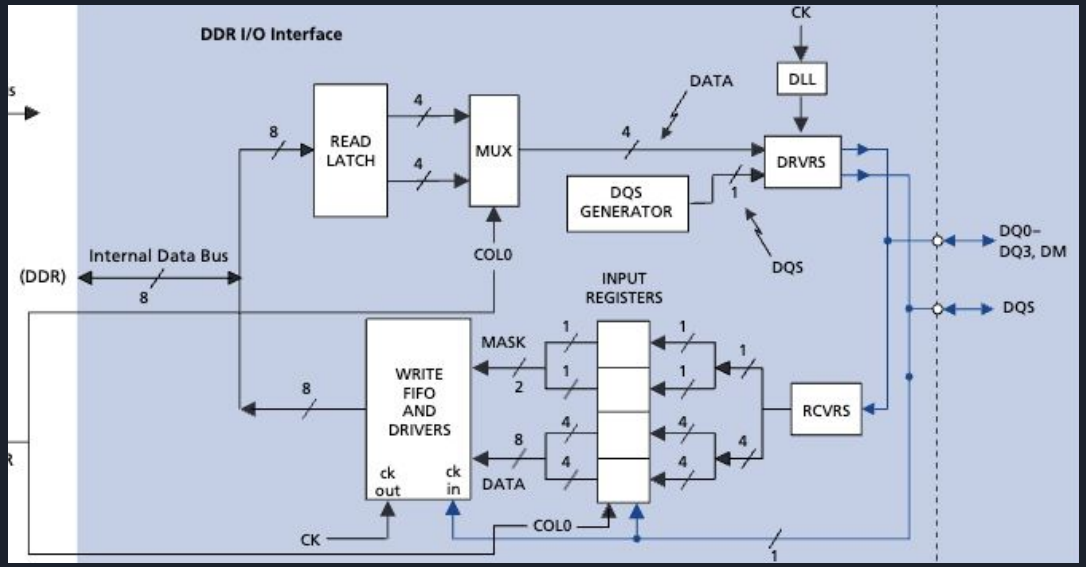
SDR SDRAM vs DDR SDRAM Architecture



The diagram on the left provides the visual representation of the architecture built for the SDRAM and the DDR SDRAM. The diagram is separated into three distinct colors for visual clarity:

- **Memory Core**: Both the SDRAM and DDR SDRAM share the same memory core architecture, which includes all of the addressing and command control interfaces, four-bank memory arrays and the same refresh requirements.
- **SDR Memory Data Interface**: This is a fully synchronous design where the internal and external bus widths match, the data latches only on the rising edge of the clock signal, and the memory supports a Data Mask Signal (DQM) for masking data during write operations and enabling output during read operations.
- **DDR Memory Data Interface**: This is a true source-synchronous design, since it uses a bidirectional data strobe to capture twice the data for every clock cycle on both the rising and falling edges of the clock, which essentially provides double the bandwidth compared to SDR SDRAM.

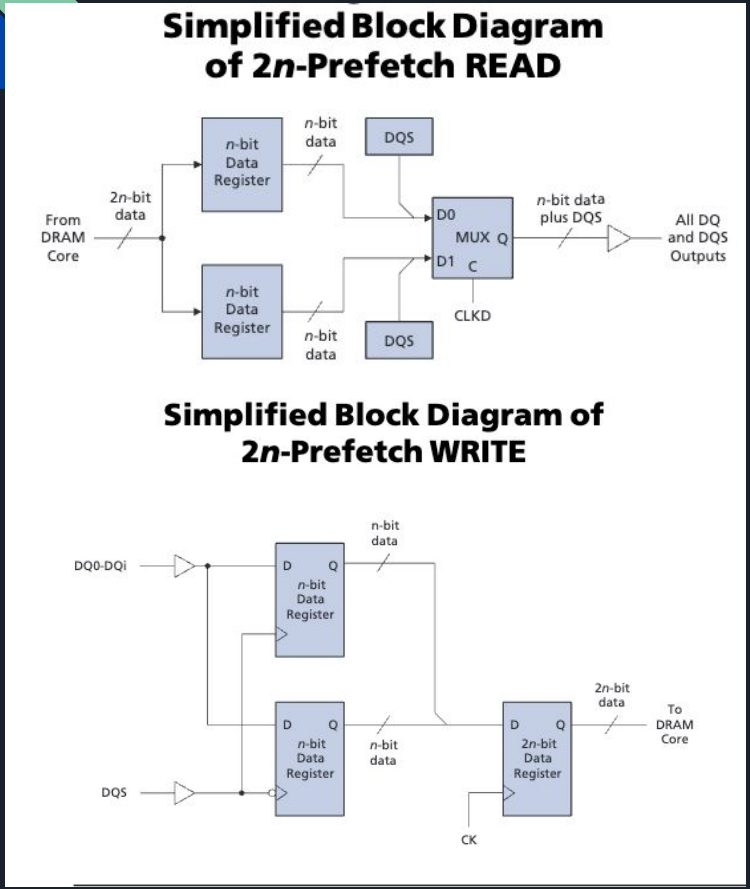
Focusing on DDR SDRAM Architecture



Although the DLL is an integral part of the SDRAM architecture, there are several other components that are crucial for its operation, as shown in the diagram:

- **Prefetch**: The DDR interface uses a 2n-prefetch architecture, which means that the internal data bus is twice the width of the external bus (unlike the SDR where the internal and external bus widths are equal), which allows the data from the internal memory cell to the I/O buffers to be sent in pairs.
- **Command Bus**: The DDR command bus includes components such as the clock enable, the chip select, row/column addresses, bank addresses, and a write enable.
- **Differential Clocks (CK/CK#)**: The DDR interface implements a differential pair for the system clock, which consists of a positive CK signal and its complementary CK# signal, allowing data transfers to happen on both the rising/falling clock edges.
- **No Output Enable**: While the SDR SDRAM uses an output enable signal for read operations, the DDR SDRAM interface does not; instead, it relies on BURST TERMINATE commands to quickly end a read operation, and a DM signal is used during write operations to allow invalid write data to be masked.

Prefetch Read/Write Operations in DDR SDRAM



Here's a visual diagram of the prefetch block diagram architecture for both read and write operations. Like stated before, a 2n-prefetch mechanism, allows for faster data **throughput** (or the amount of data that can be processed at one time) without increasing the frequency of the external data bus.

In the read block diagram, we see:

- **Two n-bit data registers,** that hold the data fetched from the DRAM core, which are read simultaneously with each holding an n-bit data word (in pairs, like previously stated).
- The **multiplexer (MUX)** is used to alternate between the two sets of data coming from the registers, and this is what provides an n-amount of bits of data per clock cycle to match the external bus width.
- The **DQS (Data Strobe Signal)** is used to signal when to capture the data on the data bus.

In the write block diagram, we see:

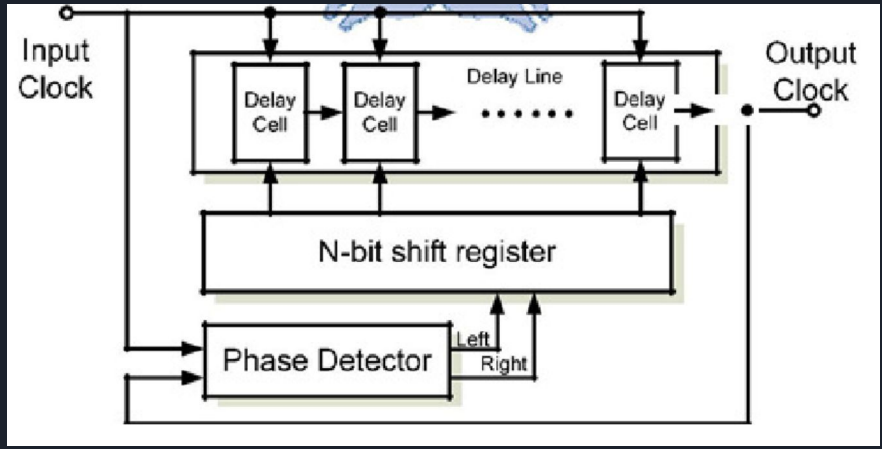
- Data from the system feeds into the **two n-bit data registers**, which temporarily store the information.
- Due to the **2n-prefetch mechanism**, the two n-bit data words are now combined to be written in one operation inside the 2n-bit data register.
- The combined **2n-bit data word** gets written to the DRAM core on either the rising/falling clock edges.

Categories of Digital DLLs

Due to variations in the Controller Circuit, the premise of the Digital DLL can be grouped into four different categories:

- **Register Controlled DLL (RDLL)**
- **Counter Controlled DLL (CDLL)**
- **Successive-Approximation Register (SAR)-Controlled DLL**
- **Time to Digital (TDC)-Controlled DLL**

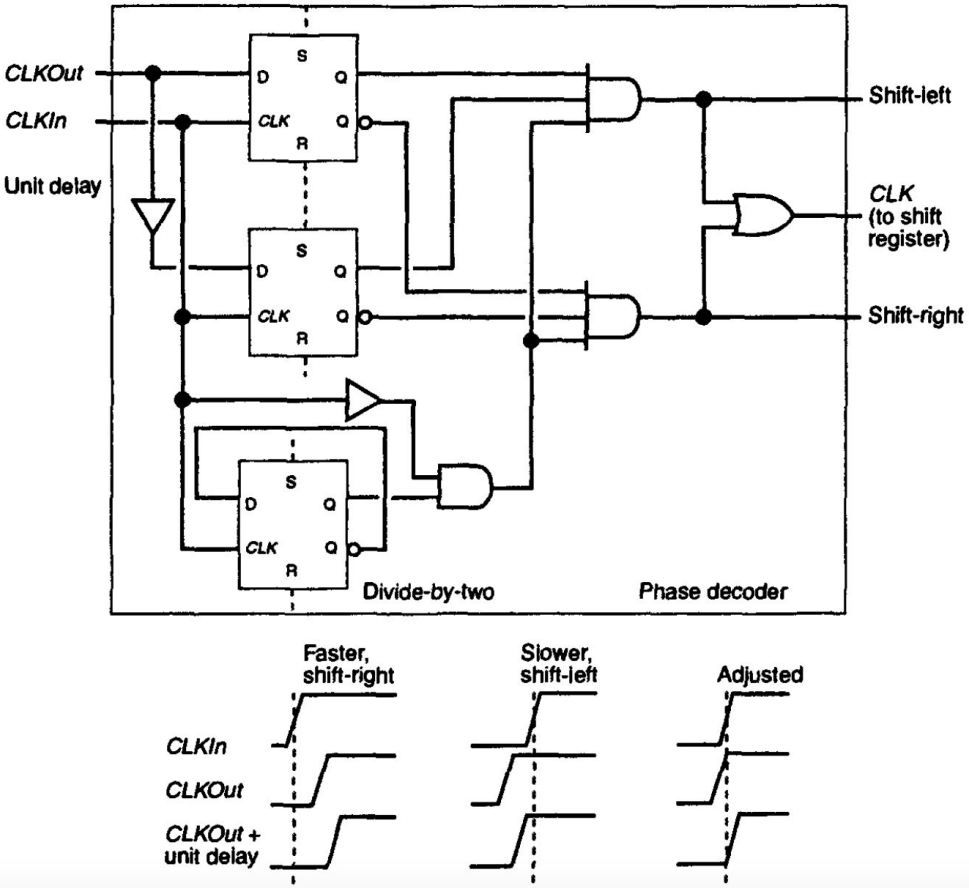
Register Controlled DLL Architecture



The figure above shows a diagram of the architecture that encompasses a standard RDLL. Let's go through an analysis of what's happening with the diagram:

- The **Input Clock** acts as the baseline timing reference used to synchronize the operations within the RDLL.
- The delay cell is used to align the phase of the output clock signal with the **Input Clock**.
- Each delay cell adds a small delay to the **Input Clock**. Cascading multiple delay cells (which forms a **Delay Line**) allows the RDLL to achieve a wide range of timing adjustments.
- The **Output Clock** is generated by passing the **Input Clock** through the **Delay Line.**
- The **Phase Detector** analyzes the relative timing of both the rising and falling edges of the **Input Clock** and the **Output Clock.**
- Two signals, **"Left"** and **"Right"**, are produced from the **Phase Detector**, which are used to indicate the timing difference between the two clock signals.
- **"Left"** indicates that the **Output Clock** is ahead of **Input Clock**, and therefore needs to be delayed.
- **"Right"** indicates the the **Output Clock** is behind of **Input Clock**, and therefore needs to catch up.
- The **N-bit shift register** receives the **Left** and **Right** signals from the **Phase Detector**, and in response, and it reacts by essentially shifting the bits left and right respectively.
- The goal is to reach a state where no further shifting is needed, which would signal that the phases of **Input Clock** and **Output Clock** are aligned, and would cause **Left** and **Right** to be low.
- As long as there is a phase mismatch, the phase detector will continue producing **Left** and **Right** signals for the shift register.

Register Controlled DLL Phase Detector Logic Circuit



Let's take a look at the inner circuitry of the phase detector seen in an RDLL and how it operates.

- The circuit uses flip flops to compare the leading edges of **CLKin** and **CLKout**.
- The outputs of the flip flops feed into the phase decoder circuitry, which ensures that only one of the **"shift-left"** or **"shift-right"** signals are active at a time.
- Looking at the bottom left waveform, we see that when **CLKout** is lagging, shift-right is induced, which reduces the delay and makes the signal faster.
- Looking at the bottom middle waveform,we see that when **CLKout** is leading, shift-left is induced, which increases the delay and makes the signal slower.
- Looking at the bottom right waveform, we see the phase adjustment in action and how **CLKout** responds to the feedback.
- In the diagram, we also see a divide-by two block implemented, which means that every other clock edge will be used in the operation. The reasoning for using a divide by two is to help mitigate the effects of noise, to provide a more stable output waveform, and to make sure that the shift register operates correctly.

Unwanted Side Effects due to Divide-by-Two Implementation:

- The consequence of this implementation is an increase in lock time.

Other Concerns:

- **Static Phase Error: Flip-flops (FFs)** have specific timing requirements, such as setup and hold times, which tell you how long before the clock edge that a data signal needs to be stable before it reliably latches onto the data. A non-zero setup time means that there is a delay from when the data signal is applied to when it is captured by the flip flop on the clock edge. Therefore, if our flip flops are not super fast, there is a chance that the setup and hold times will cause a consistent non-zero phase difference between **CLKin** and **CLKout**, even when they're locked. Although a zero setup time is not possible in a practical application, it is important to make sure our flip flops are as fast as possible in order to minimize the static phase error.
- **Metastability:** A phenomenon that can occur when the input signal changes state close to the FF's clock edge. This becomes a major concern as the DLL begins to lock, and some considerations to improve the overall operation would be to increase the dividing ratio which causes the phase detector to compare edges less and allow the flip flops more time to settle.

Register Controlled DLL Advantages / Drawbacks

Here are some of the main advantages that the RDLL provides:

Simplicity in Implementation:

- **RDLLs** are relatively simple to realize in silicon, which makes it a cost-effective choice to be used in manufacturing for a wide range of applications.

Scalability:

- **RDLLs** are easily scalable since the the value of the **N-bit shift register** can be increased in order to meet the requirements of the system.

Precise Delay Control:

- The use of the **N-bit shift register** allows the delay to be controlled in precise increments, which is very beneficial for systems that require accurate timing.

Here are the main drawbacks that the RDLL suffers from:

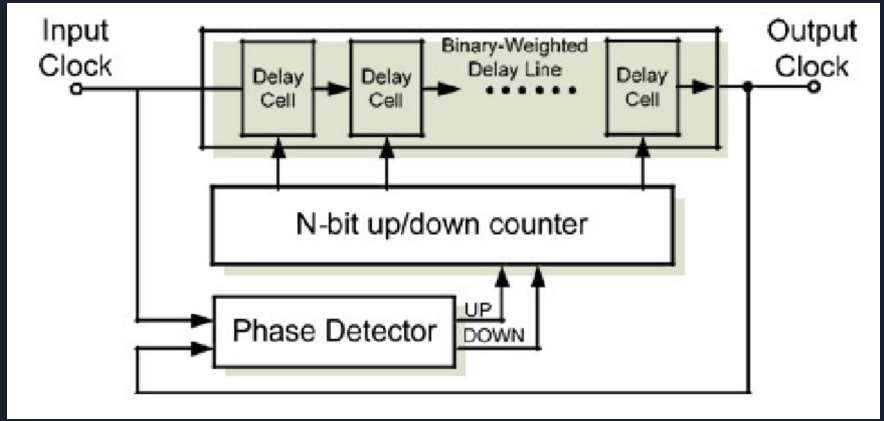
High Power Consumption Concerns:

- The active switching of multiple delay cells, and the frequent amount of adjustments made by the shift register leads to increased power consumption.

Long Locking Time:

- As the number of delay cells increases, the time it takes for the RDLL to cycle through all the potential states in order to find the correct phase alignment also increases, which can be even larger if the initial phase misalignment is high, or if the RDLL has to constantly re-lock due to changes in the system.

Counter Controlled DLL Architecture



The figure above shows a diagram of the architecture that encompasses a standard CDLL. We can see that most of the components are similar to the architecture of the RDLL, but with some slight differences:

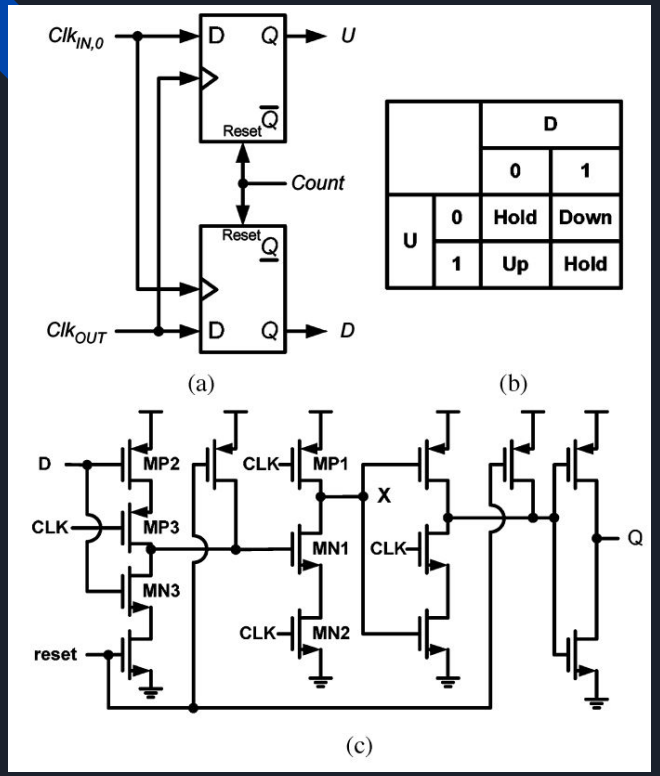
Just like the RDLL:

- The **Input Clock** acts as the baseline timing reference used to synchronize the operations within the RDLL.
- The delay cell is used to align the phase of the output clock signal with the **Input Clock**.
- Each delay cell adds a small delay to the **Input Clock**.
- The **Output Clock** is generated by passing the **Input Clock** through the **Delay Line**
- The **Phase Detector** analyzes the relative timing of both the rising and falling edges of the **Input Clock** and the **Output Clock.**

Here's the differences:

- Now we have a **Binary-Weighted Delay Line**, which means that each subsequent cell adds a delay with an integer multiple of the base delay unit (τ -> 2τ -> 4τ -> 8τ ->...). In this method, the **Input Clock** only needs to feed into the first delay cell, unlike the RDLL which connects to all delay cells.
- Instead of **"Left"** and **"Right"**, Instead of an **N-bit shift register**, we have an **N-bit Up/Down Counter,** which receives the **Up** and **Down** signals from the **Phase Detector**, and in response, and it functions by increasing or decreasing the value of the counter
- This approach allows for a wide range of delay times to be achieved with relatively few delay cells.

CDLL Special Case: Tri-State Phase Detector Logic Circuit



		D	
		0	1
U	0	Hold	Down
	1	Up	Hold

Let's take a look at the inner circuitry of a special case where a digital DLL adapts the functionality of a traditional N-bit Up/Down counter by adopting this Tri-state Phase Detector for more precise control:

Figure (a) shows the general schematic of the TSPD:

- Uses two D-type flip flops **(DFFs)**
- Two inputs, **CLKin**, and **CLKout**
- Three outputs instead of two: **Up, Down,** and **Hold**
- The **Hold** state is very useful in this case, as it is used when **CLKin** and **CLKout** are in sync, so it prevents any unnecessary toggling of the delay line.
- The **Count** signal controls the resetting of the **FFs** based on the phase comparison.

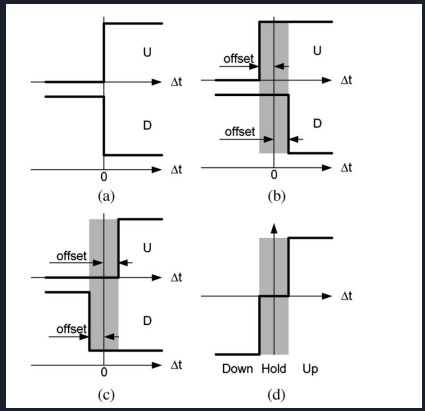
Figure (b) shows the truth table of the TSPD:

- We can see the implementation of the **Hold** State occurring when both **Up** & **Down** are '1' or '0', and then we see one or the other for **Up** & **Down** when there's a discrepancy in phases.

Figure (c) shows the inner circuitry of the DFF used in Figure (a):

Overall, this design helps suppress jitter caused by the dithering phenomenon, which is the variation in the counter output due to the input clock jitter and the DLL's delay lock process. The waveforms on the right depict the outputs of the TSPD in different variations:

- **Figure (a)** shows the outputs of the TSPD without any offsets in an ideal scenario (not practical)
- **Figure (b)** and **Figure (c**) shows the outputs of the TSPD with negative offsets and positive offsets respectively, where the **Hold** state is seen in the gray filled area.
- **Figure (d)** Is the final transfer characteristic of the TSPD between the two input clocks.



Counter Controlled DLL Advantages / Drawbacks

Here are some of the main advantages that the CDLL provides:

Simpler Implementation than RDLL

- The **CDLL** needs fewer elements than the **RDLL**, since it implements a binary weighted delay line where **CLKin** only needs to feed into the first delay cell for it to operate.

Less Power Consumption :

- **CDLLs** can be more power efficient than an **RDLL** since there is less switching activity and there only requires fewer delay cells.

Here are the main drawbacks that the RDLL suffers from:

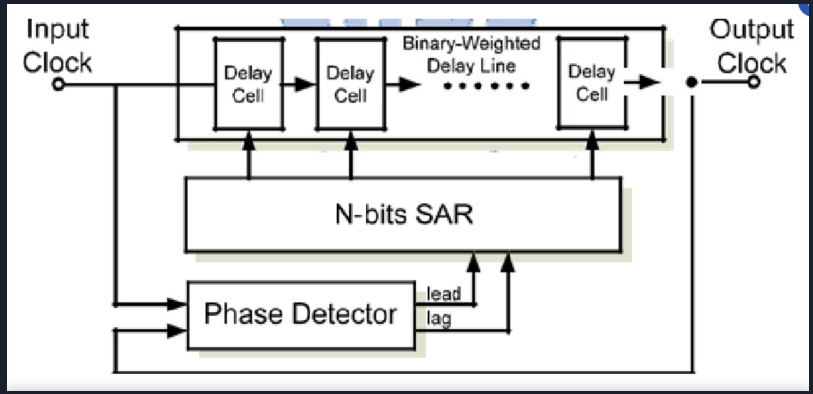
Limited Resolution:

- Since the resolution of the CDLL depends on the amount of bits in the counter, a larger counter can potentially add more complexity within the circuit.

Long Locking Time similar to RDLL:

- Even the N-bit Up/Down Counter still adjusts the delay in fixed increments, it could potentially take more time to lock onto the correct phase, and overall ends up having around the same slow locking time as the RDLL.

SAR Controlled DLL Architecture



The figure above shows a diagram of the architecture that encompasses a standard SAR Controlled DLL. We can see that most of the components are similar to the architecture of the RDLL, and it adopts the binary-weighted Delay Line from the CDLL, but there are still some differences:

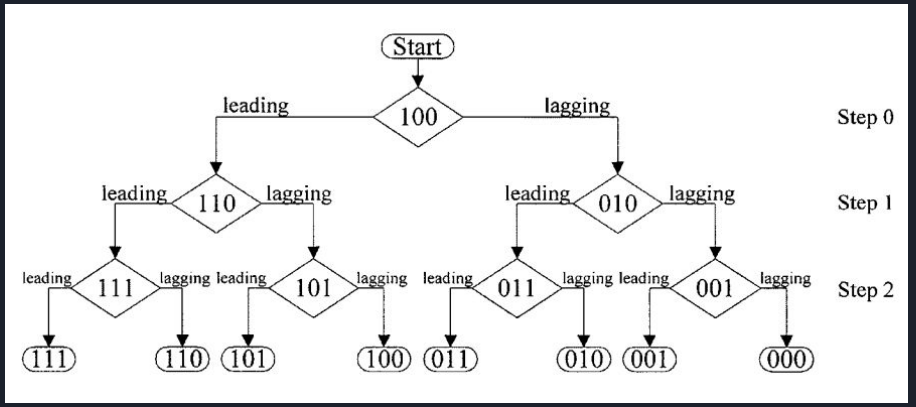
Just like the RDLL & CDLL:

- The **Input Clock** acts as the baseline timing reference used to synchronize the operations within the RDLL.
- The delay cell is used to align the phase of the output clock signal with the **Input Clock**.
- Each delay cell adds a small delay to the **Input Clock**.
- The **Output Clock** is generated by passing the **Input Clock** through the **Delay Line**
- The **Phase Detector** analyzes the relative timing of both the rising and falling edges of the **Input Clock** and the **Output Clock.**
- Now we have a **Binary-Weighted Delay Line**, which means that each subsequent cell adds a delay with an integer multiple of the base delay unit (τ -> 2τ -> 4τ -> 8τ -> ...)

Here's the differences:

- Instead of **"Up"** and **"Down"**, we have two signals, **"Lead"** and **"Lag"**, are produced from the **Phase Detector**, which are still used to indicate the timing difference between the two clock signals.
- **"Lead"** indicates that the **Output Clock** is ahead of **Input Clock**, and therefore needs to be delayed.
- **"Lag"** indicates the the **Output Clock** is behind of **Input Clock**, and therefore needs to catch up.
- Instead of an **N-bit shift register** or an **N-bit Up/Down Counter**, we have an **N-bit Successive Approximation Register (SAR)**, which receives the **Lead** and **Lag** signals from the **Phase Detector**, and in response, and it functions by successively approximating the correct delay, which is done by starting with the MSB and either sets or clears based on if the delay needs to be increased / decreased, and then moves onto the next bit, refining the binary pattern search until the best match is found.

SAR Flowchart Binary Search Algorithm



Let's look at the flowchart of the binary search algorithm used by the SAR DLL using a 3-bit binary-weighted delay line.

Initialization:

- The algorithm first starts by setting the **Most Significant Bit (MSB)** to '1' and all other bits to '0', giving us **'100'.**

Step 0:

- The algorithm checks the phase difference between the input and output clocks and determines whether **CLKout** is leading or lagging **CLKin.**
  - **If Leading: MSB remains 1, and we move to the next bit and set it to a value of '1'**
  - **If Lagging: MSB changes to '0', and we move to the next bit and set it to a value of '1'**

Step 1:

- The algorithm repeats this check of determining whether the phase is leading or lagging::
- **For '110' :**
  - **If Leading: MSB and the 2nd bit remains 1,, and we move to the next bit and set it to a value of '1'**
  - **If Lagging: MSB is 1, but the 2nd bit remains to 0, and we move on to the next bit and set it to '1'**
- **For '010':**
  - **If Leading: MSB is 0, second bit remains 1, and we move to the next bit and set it to a value of '1'**
  - **If Lagging: MSB is 0, second bit changes to 0, and we move on to the next bit and set it to '1'**

Step 2:

- The process is exactly like **Step 1**, with more variations. This is the **LSB (Least Significant Bit)** we are working with, which is the final bit used before the binary search algorithm is finished. This final 3-bit value represents the delay that will align **CLKout** with **CLKin**.

We can clearly see how the flowchart reduces the number of steps required with every step as half of the possibilities are eliminated every time.

SAR Controlled DLL Advantages / Drawbacks

If both the RDLL and CDLL suffer from Long Locking Time due to their linear approach in obtaining the control code, how does the SAR overcome this issue?

The SAR is based off a Binary Search Algorithm:

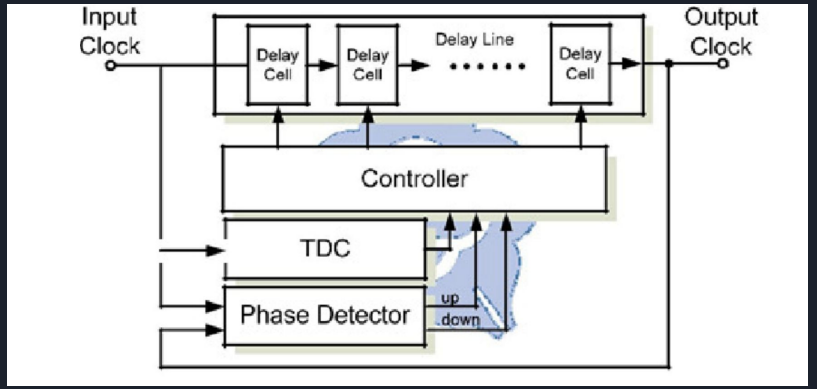
- This method is much faster than a linear search since it essentially halves the search space with each step until it finds the correct one.
- With every iteration, the **SAR** gets closer and closer to the correct delay setting, starting with the **MSB (most significant bit)** and successively approximates the delay until the desired phase alignment is achieved.
- It makes a comparison at each bit level, which allows it to skip over large ranges and doesn't have to search through each delay sequentially.
- The rate at which the SAR is able to find the desired delay is pretty much exponential compared to the RDLL and CDLL.
- **Example: Guessing from 1-100 one by one? Or dividing the range half each time to find the answer?**

Overall, the use of a binary search algorithm reduces locking time as well as hardware complexity, but there's a main drawbacks that the SAR-controlled DLL suffers from.

Drawbacks:

- Due to its open-loop characteristics, it cannot track **PVTL (Process, Voltage, Temperature, and Load)** variations. This is in contrast to closed-loop systems, which continuously monitor and adjust to maintain the correct phase despite variations in temperature, supply voltage, semiconductor manufacturing, or loads on the output of a circuit.

TDC Controlled DLL Architecture



The figure above shows a diagram of the architecture that encompasses a standard TDC-based DLL. We can see that most of the components are similar to the architecture of the RDLL, some components are adopted from the CDLL, but this architecture has the most components out of the rest:

The TDC-based architecture uses two entire blocks, the “TDC” block and the “Controller” block.

- The **TDC** block stands for **Time-To-Digital**, and it’s used to measure the timing difference between the **Input Clock** and **Output Clock** from the delay line. It takes this information and converts the data into “digital” code which quantifies the amount that the **Output Clock** is lagging / leading.
- The **Phase Detector** compares the phase of the **Output Clock** and **Input Clock**, and both the **TDC** and the **Up/Down** signals feed into the **Controller.**
- The **Controller** is the main logic block that receives the digital code from the **TDC** and the **Up/Down** phase information, and uses these to activate or deactivate certain delay cells in order to adjust the total delay based on what it was provided.
- The process works continuously with the feedback loop, until the DLL is considered to be locked.

TDC Controlled DLL Advantages / Drawbacks

The TDC method doesn't rely on phase comparison, but directly measures the time difference:

- All the other DLLs we've talked about compare the phase difference between **CLKin** and **CLKout**, and take multiple steps to adjust it in either fixed increments or successive approximations, but TDC is able to perform the phase alignment in a one-time correction.
- Like we talked previously, it measures the time interval between the two clock signals, translates that information into digital code, which then can be used to bring the clocks into alignment.
- Due to this method, it provides the fastest locking time out of the **RDLL, CDLL,** and even the **SAR-DLL.**

If it has the fastest locking time out of the rest of the DLLs, what's the catch? Unfortunately, the cost of having the fastest locking time has a lot of significant drawbacks that come along with it.

Power Consumption:

- One of the biggest concerns of TDC-based DLLs is their considerable power consumption compared to the other DLLs, which is a main factor as modern technologies push for energy efficiency.

Hardware Complexity:

- In order to determine the time difference between the two clock signals, the DLL uses extremely advanced hardware implementations including high-resolution TDC converters and other relevant units. The complexity and amount components required for this DLL leads to a considerably large area on the chip, which is undesirable to have.

References

Baker, R. "DRAM Circuit Design: A Tutorial." 2000.

Bayram, E., A. F. Aref, M. Saeed, and R. Negra. "1.5–3.3 GHz, 0.0077 mm2, 7 mW All-Digital Delay-Locked Loop With Dead-Zone Free Phase Detector in $0.13~μm CMOS." *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, Jan. 2018, pp. 39-50. doi: 10.1109/TCSI.2017.2715899.

Dehng, Guang-Kaai, June-Ming Hsu, Ching-Yuan Yang, and Shen-Iuan Liu. "Clock-deskew buffer using a SAR-controlled delay-locked loop." *IEEE Journal of Solid-State Circuits*, vol. 35, no. 8, Aug. 2000, pp. 1128-1136. doi: 10.1109/4.859501.

"Difference Among DDR2, DDR3, DDR4, and DDR5 Memory." *Crucial*. https://www.crucial.com/articles/about-memory/difference-among-ddr2-ddr3-ddr4-and-ddr5-memory. Accessed 15 April 2024.

Gomm, T. J. et al. "DESIGN OF A DELAY-LOCKED LOOP WITH A DAC-CONTROLLED ANALOG DELAY LINE: A Thesis Presented in Partial Fulfillment of the Requirements for the Degree of Master of Science with a Major in Electrical Engineering." 2001.

Lin, Feng, J. Miller, A. Schoenfeld, M. Ma, and R. J. Baker. "A register-controlled symmetrical DLL for double-data-rate DRAM." *IEEE Journal of Solid-State Circuits*, vol. 34, no. 4, April 1999, pp. 565-568. doi: 10.1109/4.753691.

Nagpara, B.H. et al. “A Review of Different Types of Digital Delay-Locked Loop in 65nm CMOS technology.” *International Journal of Modern Trends in Engineering and Research*, vol. 3, 2016.

Quchani, M. E., and M. Maymandi-Nejad. "Design of a Low-Power Linear SAR-Based All-Digital Delay-Locked Loop." *2019 27th Iranian Conference on Electrical Engineering (ICEE)*, Yazd, Iran, 2019, pp. 118-124. doi: 10.1109/IranianCEE.2019.8786365.

"Technical Note." Studylib. https://studylib.net/doc/18113990/technical-note. Accessed 13 April 2024.

"What Is SDRAM MEMORY: DDR, DDR2, DDR3, DDR4, DDR5." Electronics Notes. www.electronics-notes.com/articles/electronic_components/semiconductor-ic-memory/sdram-synchronous-dram-what-is.php. Accessed 11 Apr. 2024.

Yang, Rong-Jyi, and Shen-Iuan Liu. "A 2.5 GHz all-digital delay-locked loop in 0.13 μm CMOS technology." *IEEE Journal of Solid-State Circuits*, vol. 42, 2007, pp. 2338-2347. doi: 10.1109/JSSC.2007.906183.

Yang, R.-J., and S.-I. Liu. "A 40–550 MHz Harmonic-Free All-Digital Delay-Locked Loop Using a Variable SAR Algorithm." *IEEE Journal of Solid-State Circuits*, vol. 42, no. 2, Feb. 2007, pp. 361-373. doi: 10.1109/JSSC.2006.889381.