17-06-2025 12:00PM

RTL Design assignment: Synchronized Controlled Module

**Index**

1. **Top Module Overview**  
   1.1 Module Integration and Functional Summary  
   1.2 Signal Flow and Interconnections
2. **Synchronizer Module Explanation**  
   2.1 Clocks and Domain Crossing  
   2.2 Synchronization Using Toggle and Flip-Flop Based Logic  
   2.3 Simulation Demonstrations  
     Figure 1: Start Slot Synchronization  
     Figure 2: Analog Done Latching  
     Figure 3: Data Validity Windows
3. **Controller Module Explanation**  
   3.1 System Constraints and Design Considerations  
     Constraint 1: Programmable Delay Constraint  
     Constraint 2: Slot Delay Constraint  
     Constraint 3: Slot Overlap Constraint  
   3.2 Delay Measurement Techniques  
   3.3 Data Capture Mechanism  
   3.4 Analog Progress Flags – analog\_1, analog\_2  
   3.5 ANA State Management  
   3.6 Invalid Case Detection Logic  
   3.7 Data Handling with FIFO and Warning Flags  
   3.8 FSM States and Operational Flow
4. **Demonstration of Valid and Invalid Scenarios**  
     Figure 4: ADC Trigger Timing with and without Extra Delay  
     Figure 5: Slot Delay Violations and Invalid Detection  
     Figure 6: Multi-Cycle Slot Evaluation with Single Valid Data Acceptance  
     Figure 7: Delay + PD Constraint Violation and Marker-Based Invalid Case Identification  
     Figure 8: FIFO Full Behavior, Burst Read Impact, and Warning Flag Dynamics
5. Design Constraint Example

**Part 1: Top Module Overview**

The top module integrates the synchronizer and controller submodules to implement a robust dual-sampling data acquisition system. It handles asynchronous signal inputs from various domains, synchronizes them to a single clk domain, and manages control logic to capture and store analog and ADC data.

**Connections:**

* The synchronizer receives asynchronous inputs: start\_slot\_asyn, mid\_slot\_asyn, analog\_done\_asyn, adc\_eoc\_asyn, burst\_read\_asyn, and corresponding data values.
* These are synchronized to the clk domain and routed to the controller, which drives analog\_start, adc\_trigger, trigger\_burst\_read, and stores paired analog+ADC data.

**Part 2: Synchronizer Module Explanation**

**2.1: Clocks and Domain Crossing**

* **Input Clocks:** clk, time\_clk, ana\_clk, adc\_clk
* **Signals arriving with each clock:**
  + **time\_clk:** start\_slot\_asyn, mid\_slot\_asyn, delay\_from\_programmer\_ss\_asyn, delay\_from\_programmer\_ms\_asyn
  + **ana\_clk:** analog\_done\_asyn, analog\_data\_asyn
  + **adc\_clk:** adc\_eoc\_asyn, adc\_data\_asyn
  + **Other:** burst\_read\_asyn from potentially external asynchronous domain

**Updated Synchronizer Functionality:**

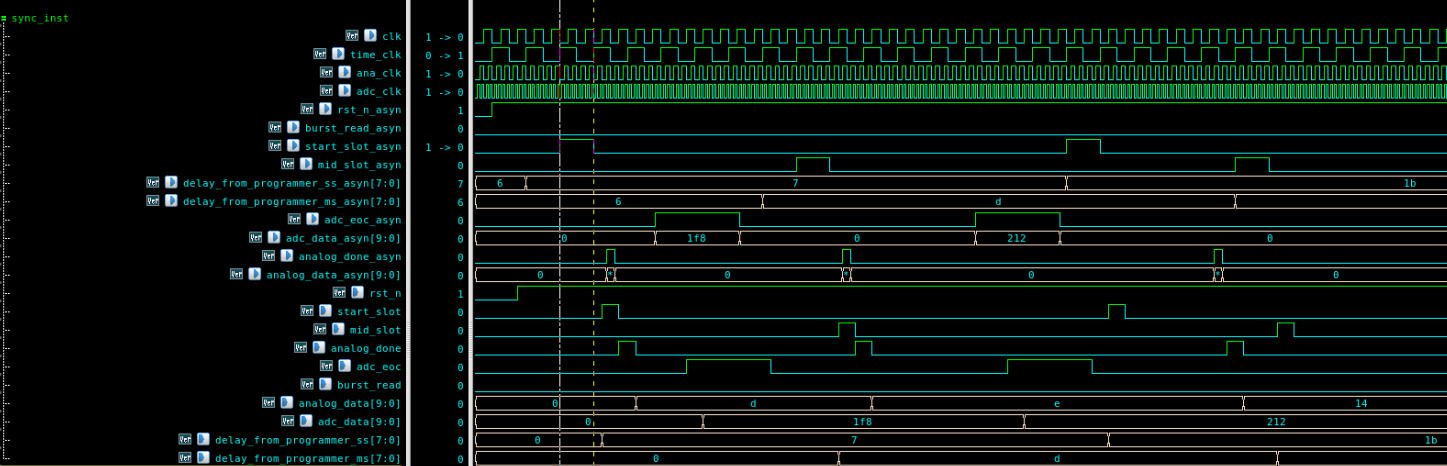
The updated synchronizer handles multiple asynchronous inputs from various domains and brings them safely into the main clk domain using dedicated synchronization structures:

* **Reset Synchronization:** Two-stage synchronizer ensures rst\_n\_asyn is safely brought into the system domain.
* **Pulse-Based Inputs (start\_slot, mid\_slot, analog\_done):**
  + These are short pulses in their respective domains.
  + A toggle mechanism is used: every time a pulse is generated in the source domain, a toggle bit flips.
  + In the clk domain, the change in this toggle value is detected using double flip-flop synchronizers.
  + The edge detection on the synchronized toggle produces a single-cycle pulse output.
* **Level-Based Inputs (burst\_read, adc\_eoc):**
  + Directly synchronized using two flip-flops.
  + No toggle or latching is required since these signals remain high long enough for safe sampling.
* **Data Capture:**
  + Analog and ADC data (analog\_data\_asyn, adc\_data\_asyn) are captured into temporary registers in their respective domains when analog\_done\_asyn or adc\_eoc\_asyn is high.
  + These temporary registers are then sampled into the main clk domain when the synchronized done/eoc pulse is detected.
* **Programmable Delay Capture:**
  + delay\_from\_programmer\_ss\_asyn and delay\_from\_programmer\_ms\_asyn are latched in the time\_clk domain.
  + Upon assertion of start\_slot or mid\_slot, these values are latched and passed into the clk domain for the controller to use.

**2.2: Synchronization Using Toggle and Flip-Flop Based Logic**

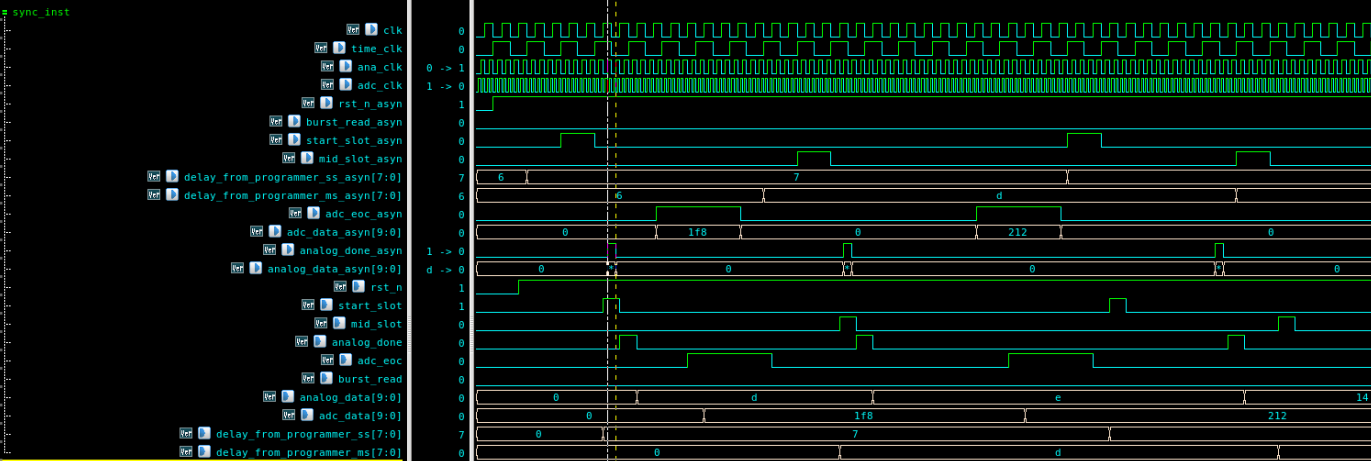
* **Why Toggles:**
  + Toggle synchronization is effective for short pulse signals that may not be reliably captured using level synchronization.
  + It allows detecting event edges rather than relying on pulse widths.
* **Why Flip-Flops:**
  + All signals are safely brought into the clk domain using double flip-flop sampling to avoid metastability.
  + This method ensures reliable signal transitions and avoids glitches.

**2.3: Demonstration via Simulation Screenshots**

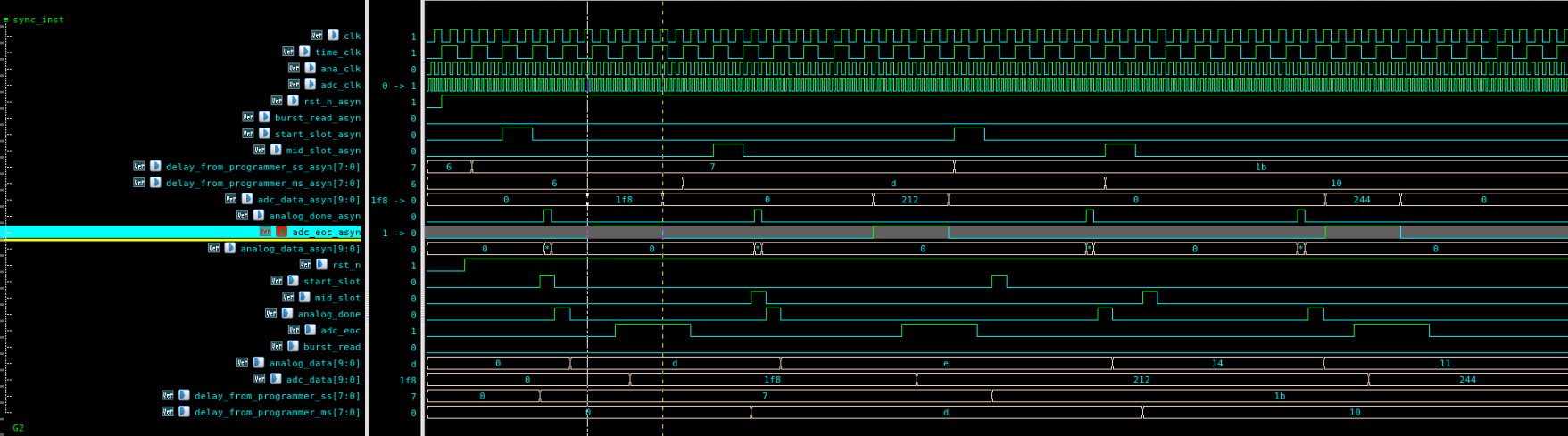
****

**Figure 1: Start Slot Synchronization**

The start\_slot signal is synchronized into the clk domain and converted into a single-cycle pulse. This ensures that even if the asynchronous input is held high for multiple cycles or too briefly, only one clean, deterministic pulse is generated and passed to the controller.

  
**Figure 2: Analog Done Latching**

The analog\_done signal has a very short pulse width, often less than one clk cycle. To prevent loss of this signal, a latch mechanism is used to retain the assertion until it can be safely sampled and converted into a single-cycle pulse in the clk domain.

****

**Figure 3: Data Validity Windows**

The input signals analog\_data and adc\_data are only considered valid during the high phases of their respective handshake signals — analog\_done and adc\_eoc. This ensures that data sampling occurs only when the upstream module has confirmed data validity.

**Part 3: Controller Module Explanation**

**3.1: System Constraints and Considerations**

The design must ensure the analog and ADC timing satisfies certain delay constraints for correctness:

**Constraint 1: Programmable Delay Constraint**

* PD >= α + β
* **α:** Time from analog\_start to analog\_done
* **β:** Time from adc\_trigger to adc\_eoc

**Constraint 2: Slot Delay Constraint**

* Δ >= α
* Ensures that one analog process finishes before another begins.

**Constraint 3: Slot Overlap Constraint**

* Δ + α2 + ED2 >= α1 + ED1
* Prevents a later slot from finishing analog+ADC operations before the earlier one does.

**3.2: Measurement of Delays**

* α is computed using internal counters (counter\_alpha\_1, counter\_alpha\_2) that track duration between analog\_start and analog\_done.
* Δ is tracked using counter\_delta\_1, counter\_delta\_2.
* Extra delays (extra\_delay\_1, extra\_delay\_2) are added if the programmable delay exceeds α + β + 5.

**3.3: Data Capture Mechanism**

* capture\_counter\_ss and capture\_counter\_ms are delay-based counters for start\_slot and mid\_slot respectively.
* When the counters reach the programmed delay values, flags capture\_start\_ss and capture\_start\_ms go high to indicate data should be latched.

**3.4: Analog Flags - analog\_1, analog\_2**

* These flags indicate if the current analog slot is still in progress.
* When analog\_start is asserted on start\_slot or mid\_slot, the respective analog\_1 or analog\_2 is cleared.
* Once analog\_done is received, the flags are set back.
* These are critical to ensure that slots don't overlap in analog activity.

**3.5: ANA State (ana)**

* A 3-bit state register that helps in timing and data validity management.
* Tracks which phases of analog and ADC have completed.
* Used to schedule adc\_trigger and control FSM transitions.

**3.6: Invalid Case Detection**

Invalid flags and markers identify problematic conditions:

* **Invalid Cases Include:**
  1. Starting mid\_slot when analog\_1 hasn’t completed.
  2. Triggering ADC before analog\_done.
  3. Delay overlap conditions (delta constraints not met).
  4. Slot ends but analog operation from earlier slot is still ongoing.

**How it's detected:**

* invalid, invalid\_next, and invalid\_flag logic detects specific FSM and timing violations.
* Conditions like:
  + delta\_2 + delay\_ss <= delay\_ms
  + capture\_counter\_ss + 2 == delay\_ss && !analog\_1
* Each invalid condition is marked using invalid\_mark with a unique ID to trace type of error.
* invalid\_flag is latched until the end of a capture to prevent using bad data.

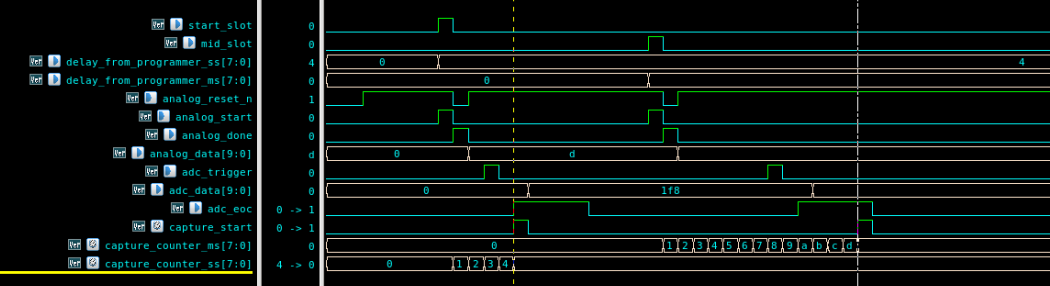
**3.7: Data Management using FIFO**

* Samples are stored in FIFO.
* When fifo\_full\_warning is high and fsm is in SINGLE, trigger\_burst\_read is asserted.
* If invalid\_flag is high during capture, data is discarded.

**3.8: FSM States**

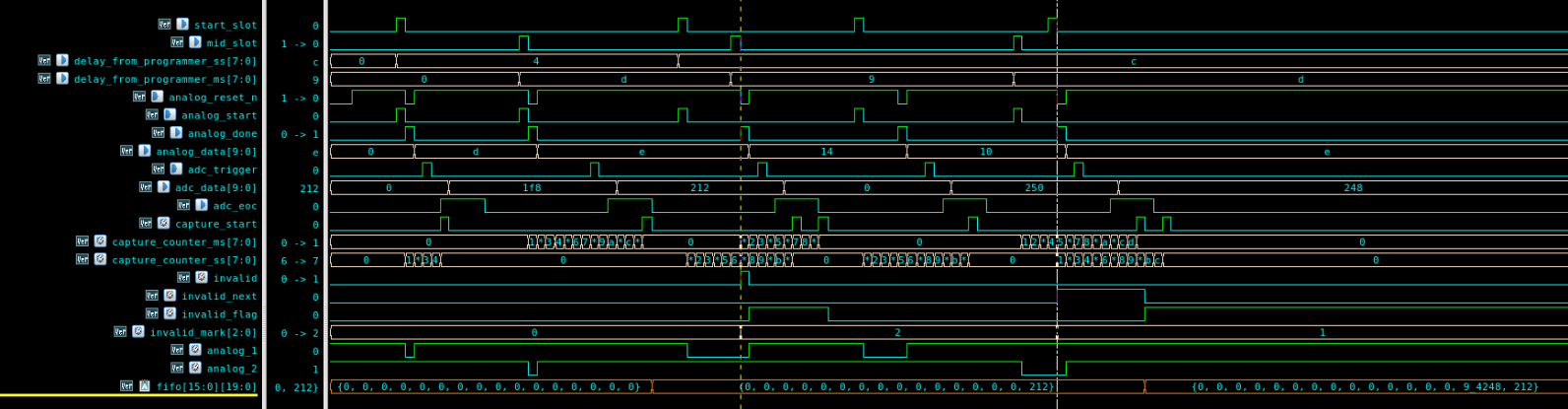
* IDLE: No slots active
* SINGLE: One analog-ADC pair in progress
* PAIR: Second analog-ADC pair in progress

**Part 4: Demonstration of Valid and Invalid Scenarios**

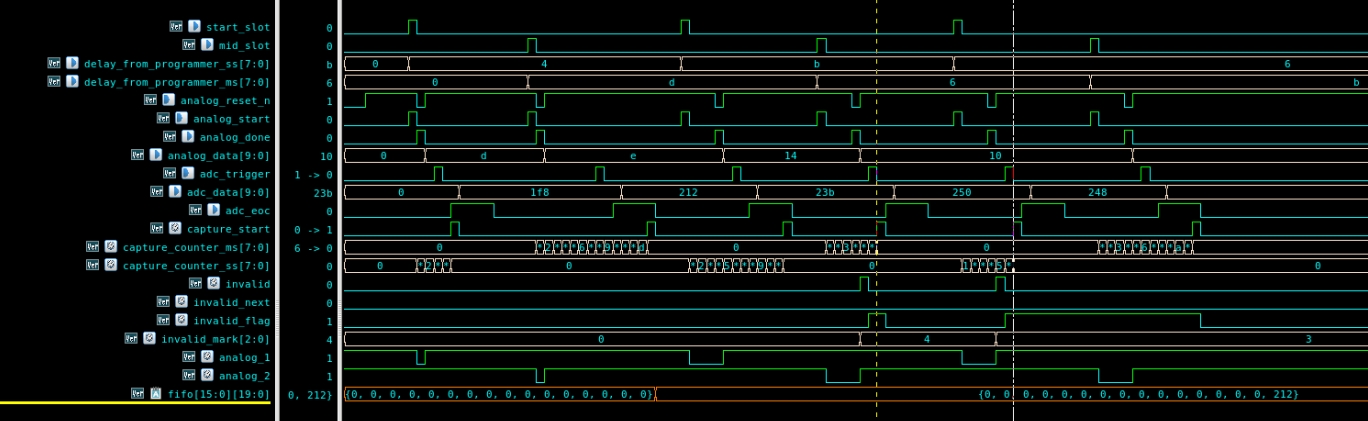


**Figure 4: ADC Trigger Timing with and without Extra Delay**

This figure illustrates two scenarios of programmable delay (PD) relative to α + β. In the first case, PD = α + β, so the ADC trigger aligns with the first slot of adc\_eoc, capturing data immediately. In the second case, PD > α + β + 5, which invokes an additional delay (extra delay) between the end of analog and the start of ADC trigger. This extra delay ensures the capture still happens within the adc\_eoc window but is positioned toward the end, using the fewest possible cycles. This placement accelerates readiness for the next slot

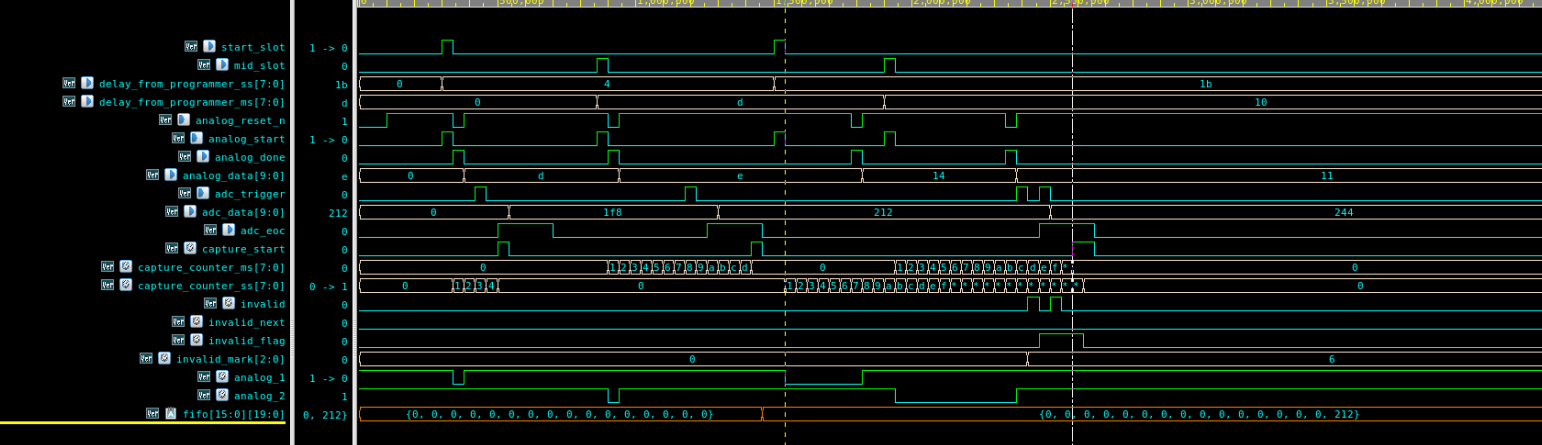


**Figure 5: Slot Delay Violations and Invalid Detection**

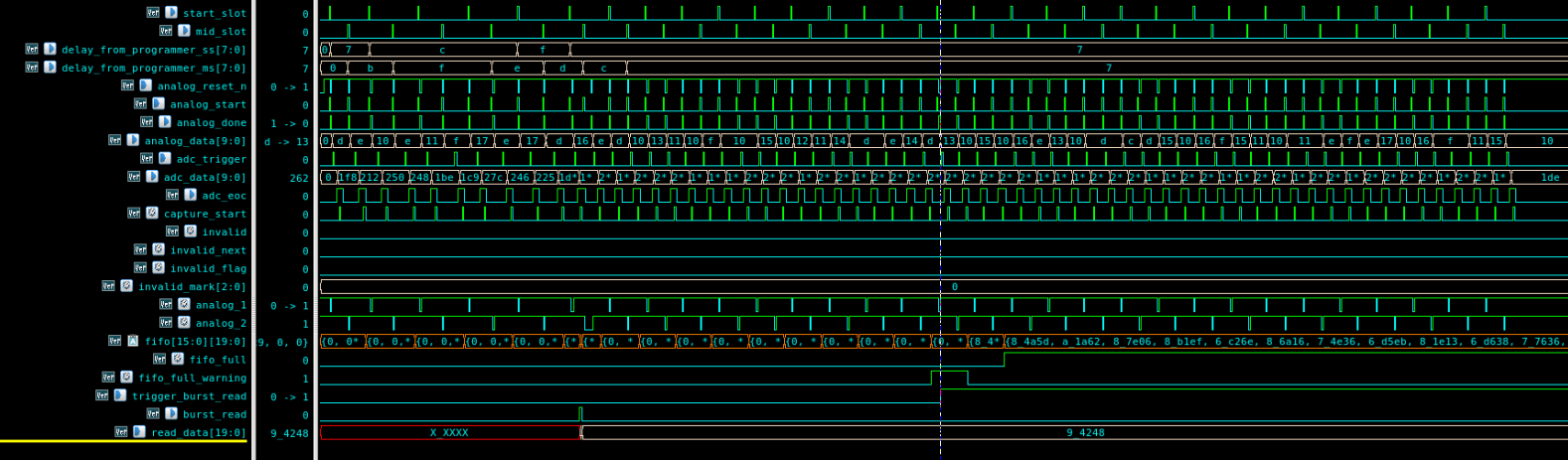
This figure demonstrates a mix of valid and invalid capture scenarios. The first instance shows a valid capture, where start\_slot and mid\_slot occur with proper delays, and the resulting data is successfully stored in the FIFO. In the second scenario, the mid\_slot occurs too early, causing the slot delay Δ to be less than α, violating the required constraint. As a result, the invalid signal is asserted and the data is not stored. The third scenario starts with a valid sequence where both start\_slot and mid\_slot have sufficient programmable delay, ensuring proper capture. However, the subsequent start\_slot appears too soon — the delay between the previous mid\_slot and this start\_slot is less than α, which triggers invalid\_next to be high, preemptively flagging the upcoming data as invalid.  
  


**Figure 6: Multi-Cycle Slot Evaluation with Single Valid Data Acceptance**

This figure presents a comprehensive sequence illustrating both valid and invalid behaviors across multiple slot captures. The first start\_slot and mid\_slot pair is configured correctly, and the resulting analog and ADC data is stored in the FIFO. This case serves as a reference for evaluating subsequent captures. In the second scenario, while the start\_slot is valid, the mid\_slot comes too early, with a programmable delay less than α + β, making the capture invalid. In the third case, the start\_slot itself has a PD less than α, violating the constraint. As seen in the waveforms, when an invalid condition is detected at start\_slot, the invalid flag is raised, which also impacts the validity of its corresponding mid\_slot. Ultimately, only the data from the first, valid case is accepted and stored.



**Figure 7: Delay + PD Constraint Violation and Marker-Based Invalid Case Identification**  
In this scenario, the first case is a valid one and serves as a reference. In the next capture, the mid\_slot arrives after some delay from the start\_slot, but the programmable delay for the mid\_slot is less than the one used for start\_slot. This leads to a violation of the Δ + PD < PD\_prev condition — a type of invalid case #3. When ADC is triggered, this inequality is detected, and the invalid signal is raised. The system uses invalid\_mark to identify exactly which type of invalid case has occurred, enabling robust classification and debugging.



**Figure 8: FIFO Full Behavior, Burst Read Impact, and Warning Flag Dynamics**  
In this figure, a few key observations are highlighted. First, when a burst\_read signal is asserted and held high for multiple cycles, a corresponding number of entries are read from the FIFO. Second, when the FIFO has only three spaces remaining, the fifo\_full\_warning signal is asserted. This warning stays high until a burst\_read operation is completed, ensuring that the system is aware of the near-full status. Lastly, once the FIFO is full, no new data is written into it until space becomes available. This behavior protects data integrity and prevents overwriting.

**link to rtl\_filelist:** /home/SharanyaShetty/digital/JUNE/CONTROLLER\_ASSIGNMENT/script/ rtl\_sample.flist

**Part 5: Design Constraint for VALID Example**

To understand how design constraints impact the programmable delay (PD) range and slot timing, consider the following illustrative example:

Let:

* α (analog delay) = 10 clock cycles
* β (ADC latency) = 3 clock cycles

Then α + β = 13 cycles.

**Constraint 1: Programmable Delay Validity**

To satisfy the first constraint:

PD ≥ α + β

⇒ PD ≥ 13

So, any PD less than 13 will be considered invalid. For a valid scenario, PD must be **13 or more**.

**Constraint 2: Slot Delay Bound**

Slot-to-slot gap (Δ) must be strictly greater than α:

Δ > α

⇒ Δ > 10

This ensures that the analog phase of the current slot completes before the next slot can begin, preventing overlapping use of the analog block.

**Constraint 3: Overlap Prevention**

To ensure that the current slot does not end before the previous slot finishes, we require:

Δ + PD\_current > PD\_previous

For example, if:

* Δ = 11 (just greater than α)
* PD\_current = 13 (minimum valid PD)
* Then:

11 + 13 = 24

So this combination is only valid if the previous slot’s PD was ≤ 24.

Now suppose a previous slot had PD = 32. Then:

11 + 13 = 24 < 32 ⇒ Invalid

This would violate Constraint 3 and mark the slot as invalid.

**Conclusion and Acceptable PD Range**

For a safe system design with α\_max = 10 and β = 3, and assuming:

Δ > α\_max ⇒ Δ ≥ 11  
**Note:** additional 5 for EOC

The **valid programmable delay (PD)** must satisfy:

**PD ∈ [α\_max + β + 5, Δ + α\_max + β + 5)**

Lower bound: PD ≥ α\_max + β + 5 = 18

Upper bound: PD < Δ + α\_max + β + 5= Δ + 18

So the valid range of PD becomes:

PD ∈ [18, Δ + 18) where Δ > 10

This ensures all constraints are met:

1. Enough delay after analog before ADC
2. Sufficient separation between successive slots
3. No overlapping between adjacent slot data collections

By bounding PD within this calculated window, both correctness and timing reliability are guaranteed.

**Note:** These constraints provide a guaranteed safe operating window where all captured data will be valid. However, even if these conditions are not strictly met, the system is still designed to handle such scenarios. An invalid flag will be raised internally, and data from those cycles will either be discarded or handled appropriately to maintain system integrity without causing breakdown or data corruption.

**Fixed Design Constraint:**

Due to the way start\_slot and mid\_slot events are synchronized and tracked, **a new mid\_slot cannot be initiated until the previous mid\_slot process has completed**, and the same applies to start\_slot. This enforces strict sequencing and prevents overlapping analog capture operations. If this rule is violated **design will fail**.