### **BUG list – 17/06/2025 to 23/06/2025**

### **Bug List Index**

**17-06-2025 -06-2025**  
 1. ADC data validity issue  
 2. FIFO write limitation issue   
 3. FIFO read overwrite issue   
 4. FIFO data skipping during write

**18-06-2025**  
 5. Invalid data stored despite timeout  
 6. ADC data capture issue (extension of point 1)  
 7. Analog delay misbehavior (extension of point 5)  
 8. Analog delay cycle mismatch  
 9. Timeout not triggered after analog\_done (extension of point 7)

**20-06-2025**  
 10. Timeout counter starts before start\_analog  
 11. Analog vs control clock ratio constraint  
 12. Missed start\_slot due to clock domain timing

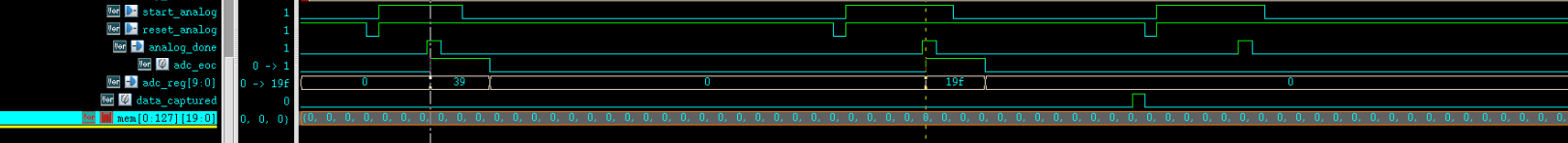
**23-06-2025**  
 13. Timeout counter not initialized properly  
 14. Minimum delay constraint between start and mid slots  
 15. FIFO skips values after 7th write  
 16. Timeout counter inactive during mid slot  
 17. Multiple start\_analog triggers before analog\_done  
 18. Timeout counter inactive after start slot  
 19. wr\_ptr not resetting without global reset

**17-06-2025**

**1. ADC Data Validity Note - SOLVED**

In the design, **ADC data is valid only when the EOC (End of Conversion) signal is high**.  
 This EOC signal remains high for **5 clock cycles**, during which the data output from the ADC is stable.

In the waveform below, you can see that the **ADC provides valid data during the EOC high window**.  
 However, due to incorrect capture timing, the data was **not sampled properly** during this window, and as a result, **it was not stored into memory**.

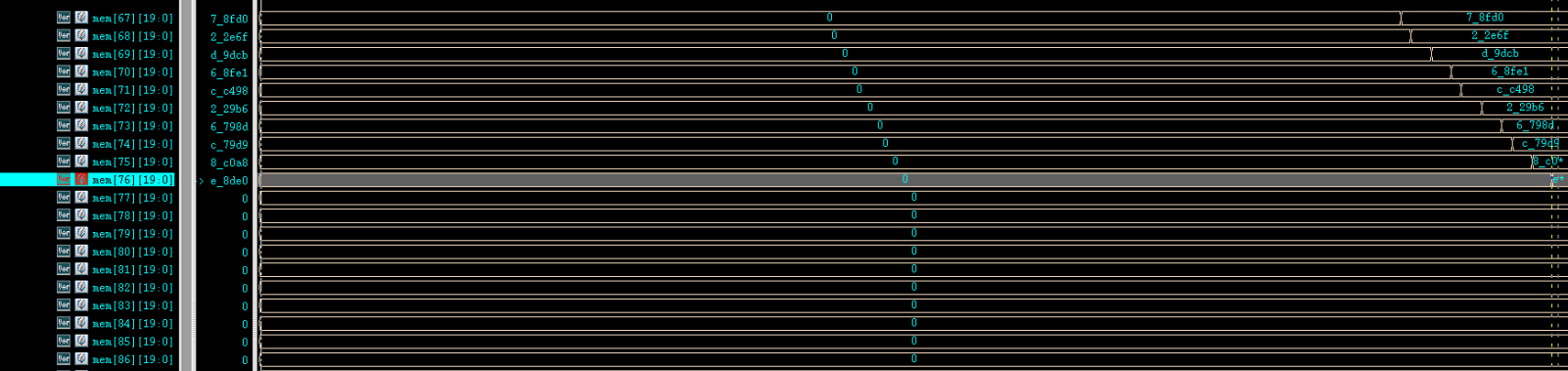
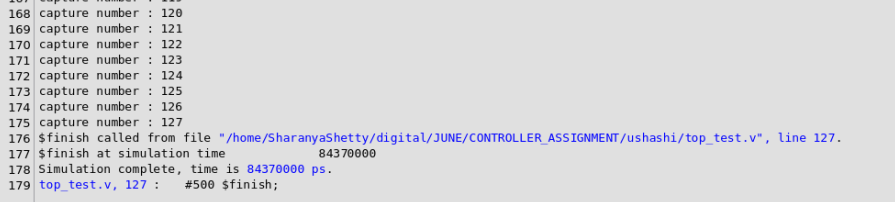


2. **FIFO Write Issue Observation - SOLVED**

Even though **ADC data remains valid for a long time** (since EOC is high for 5 cycles), when I attempt to write **128 values**, the FIFO only accepts **76 entries** — it doesn’t take more than that.

Because of this:

1. The **low storage warning** is **not triggered**.
2. The **FIFO does not reach full**, so we are **unable to test if it correctly prevents overwriting** when full.



**Note:**  
 This behavior is due to **Point 4** — only half the data is being written because of skipped writes (e.g., data\_2 is missed while data\_1 and data\_3 are captured).

However, when **256 values** are written continuously:

* The **FIFO does get filled**, and
* The **low storage warning is correctly raised**.

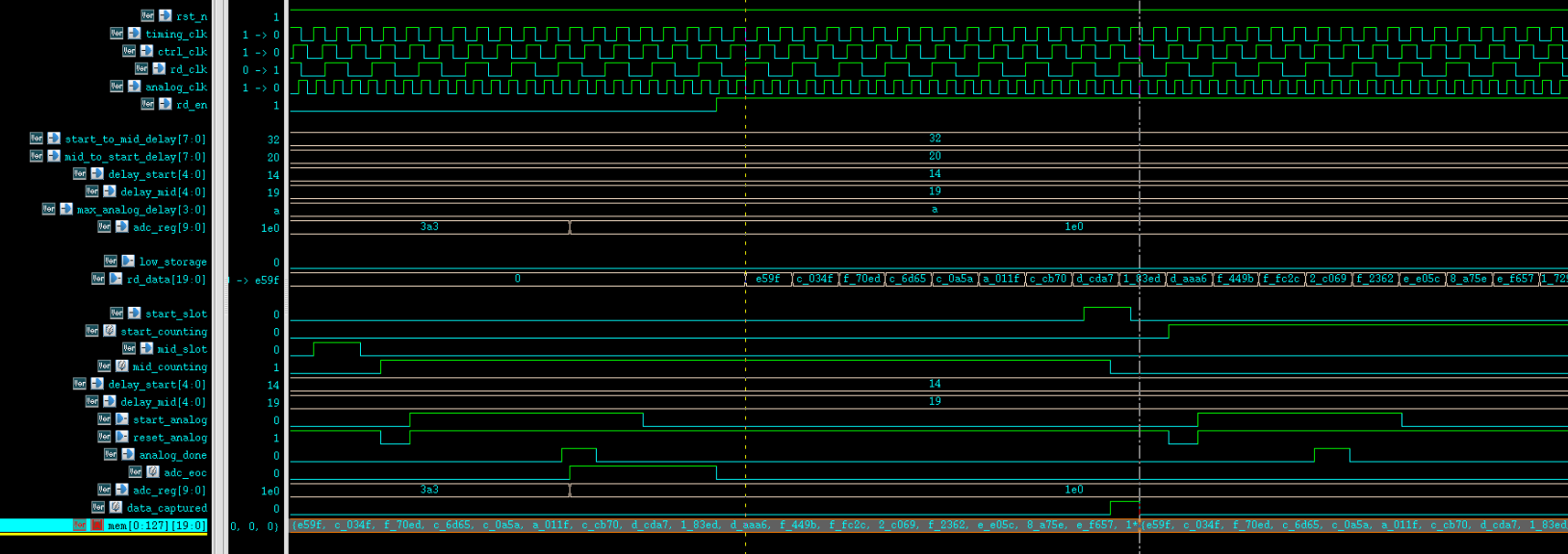
This confirms that the FIFO works properly, but **write logic needs to be checked** for data skipping in smaller transfers.

3. **FIFO Read Issue Observation – SOLVED (overwrite was allowed)**

Even though the **ADC data remains stable more than the entire duration of EOC**, and **only half the expected data is stored in the FIFO**, an issue occurs during reading.

When reading from the FIFO:

* **Data is not being removed (popped)** after each read. --> **overwrite is allowed**
* As a result, the same value appears repeatedly, and the FIFO content doesn’t update as expected.

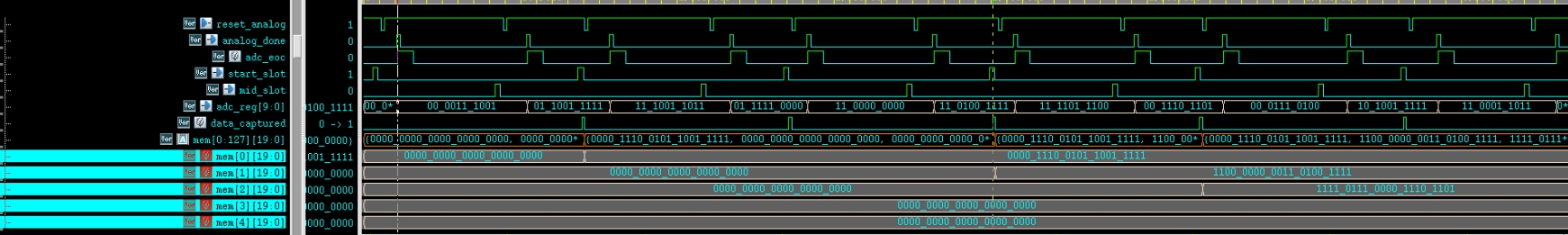


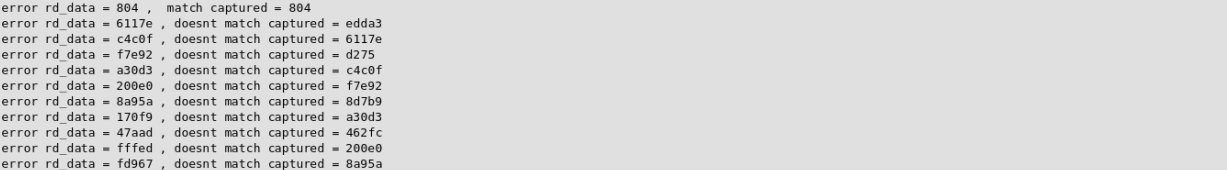
4. **FIFO Write Skipping Issue - SOLVED**

When writing data into the FIFO, I observe that **some values are getting skipped**.

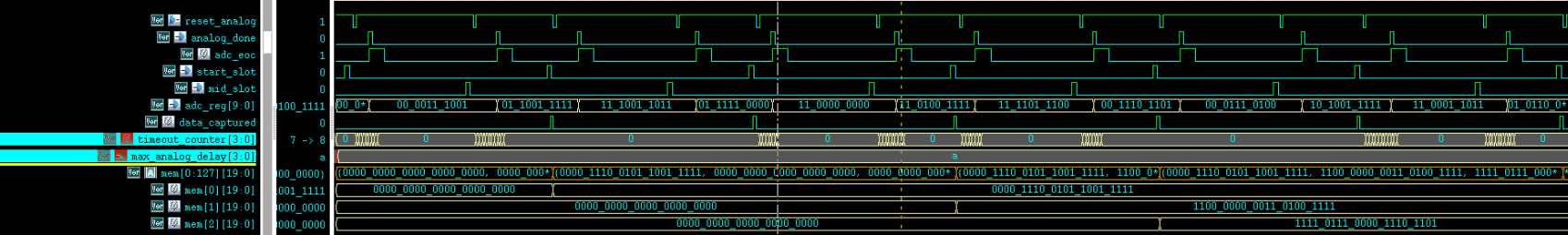
For example:

* I write data\_1, data\_2, data\_3 in consecutive cycles.
* But inside the FIFO, only data\_1 and data\_3 are stored — **data\_2 is missing**.





With the analog timeout

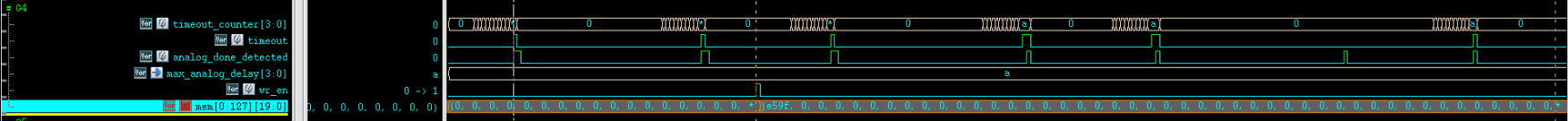


**18-06-2025**

### **5. Invalid Data Stored Despite Timeout**

Even though a **timeout occurred** — meaning the data was **marked as invalid** — the system still **stored this invalid data into the FIFO**.

**Here only the timeout reference is wrong the functionality is correct**



**19-06-2025**

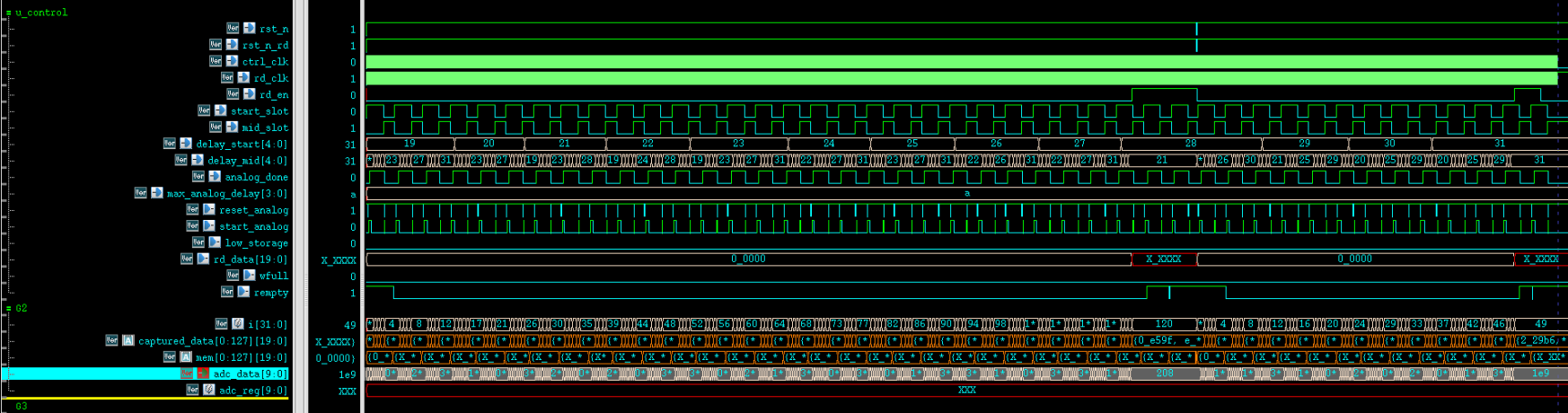
**6. Extension of Point 1 – ADC Data Capture Issue – SOLVED (top level design change wasn't informed )**

This issue is an **extension of Point 1**.

Earlier, it was assumed that **ADC data would remain stable for a longer time**, so capturing it directly was working.  
 Now, the code has been modified to **latch the ADC data internally** into a register (e.g., adc\_reg) before using it.

However, this new approach is **not working correctly**:

* The **actual ADC data is valid and correct**.
* But the internal register (adc\_reg) shows **XX values**



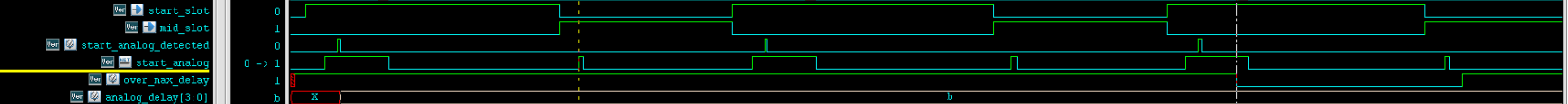
**7. Extension of Point 5 – Analog Delay Misbehavior - SOLVED**

This issue extends **Point 5**.

I was testing a case where the **analog delay is set differently for start\_slot and mid\_slot**.  
 The goal was to verify that if **either delay exceeds the maximum allowed analog delay**, the corresponding data should **not be written**.

However, while testing this:

* I noticed that the **analog start detection is triggered only by start\_slot**.
* The **mid\_slot does not generate its own analog start** — instead, it **reuses or copies the analog data** associated with start\_slot.
* As a result, the **ADC receives incorrect data** during the mid slot, defeating the purpose of having **separate analog delays**.



**8. Analog Delay Handling – SOLVED**

The **max\_analog\_delay** value is used for delay counting in **both the analog module and the controller module**.  
 However:

* In the **analog module**, the delay is calculated **per ana\_clk cycle**.
* In the **controller module**, the same delay is interpreted **per ctrl\_clk cycle**.

This creates a mismatch when:

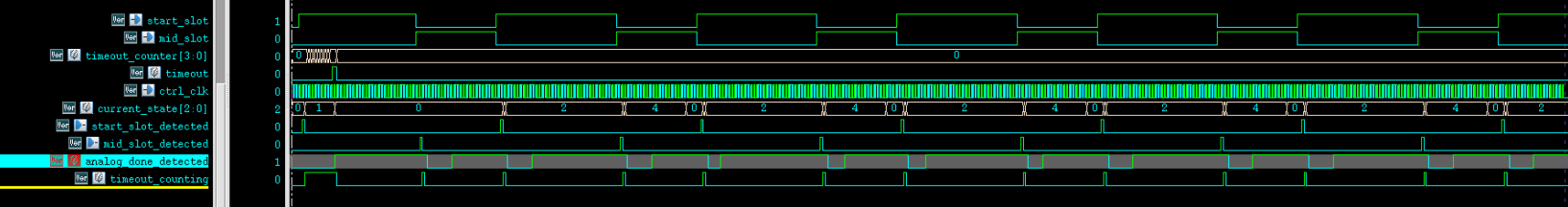
* **ana\_clk is faster than ctrl\_clk**  
   → The analog module believes it is delaying for a valid duration **less than or equal to max\_analog\_delay**,  
   → But from the controller's perspective, the effective delay **exceeds max\_analog\_delay**.

**9. Extension of Point 7 – Timeout Behavior - SOLVED**

This issue is an **extension of Point 7**.

From the waveform, we can see that:

* The **timeout condition occurs only at the beginning**.
* After that, **no further timeout is triggered**, even though it should be — because the system **immediately detects analog\_done**.



**20-06-2025**

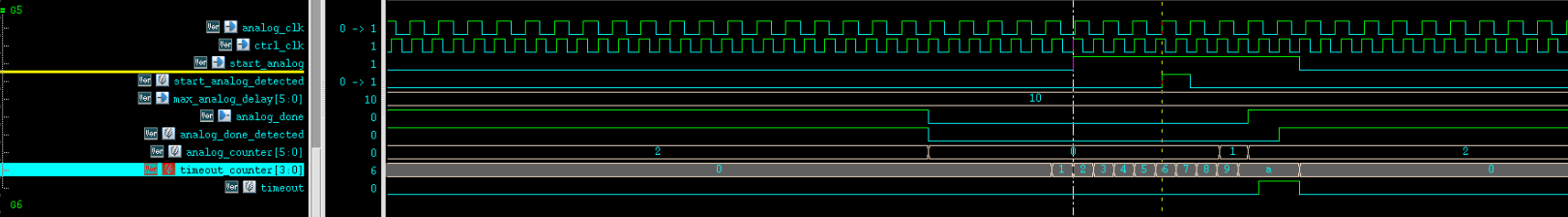
**10. Extension of Point 9 – Timeout Counter Error - SOLVED**

This is an **extension of Point 9**.

Another issue with the timeout logic is that the timeout counter starts counting even **before start\_analog is asserted**.

As a result:

* The timeout may expire **prematurely**, before the analog delay even begins.
* This leads to **false timeout detections**, even when the analog module is functioning correctly.



**11. Design Constraint – Maximum Ratio Between ana\_clk and ctrl\_clk**

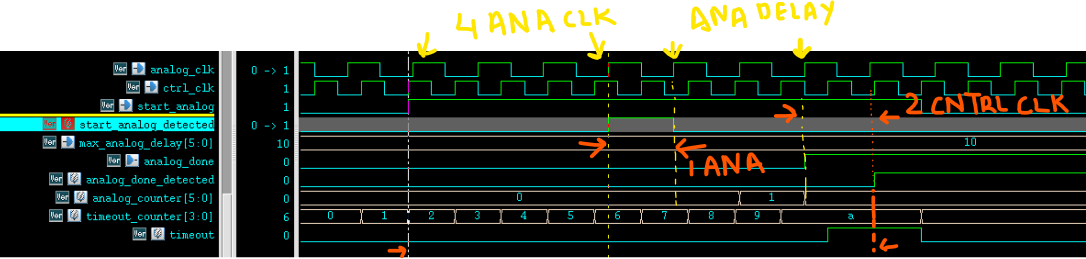
Due to the system’s functional behavior, **ana\_clk cannot be more than 1.33× slower than ctrl\_clk**.  
 This constraint comes from the way the timing is handled between start\_analog, analog\_done, and their respective detection points.

**Timing Breakdown:**

* From start\_analog to analog\_start\_detected → **5** **ana\_clk cycles**
* After that, analog delay = ana\_delay cycles (in ana\_clk)
* From analog\_done to analog\_done\_detected → **2 ctrl\_clk cycles**

So the total time for the analog transaction is:

**(5 + ana\_delay) \* ana\_clk\_period + 2 \* ctrl\_clk\_period**



To meet the constraint of maximum allowed analog delay (in controller time), we require:

(5 + ana\_delay) \* ana\_clk\_period + 2 \* ctrl\_clk\_period < max\_analog\_delay \* ctrl\_clk\_period

**Finding the Limit:**

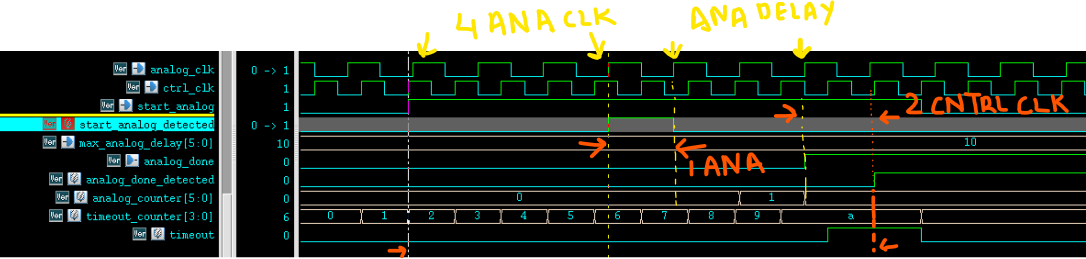
Give ana\_delay lower limit to find upper limit of ana\_clk\_period , max\_analog\_delay = 10

(5 + 1) \* ana\_clk\_period + 2 \* ctrl\_clk\_period < 10 \* ctrl\_clk\_period

6 \* ana\_clk\_period < 8 \* ctrl\_clk\_period

ana\_clk\_period < (8 / 6) \* ctrl\_clk\_period

ana\_clk\_period < 1.33 \* ctrl\_clk\_period

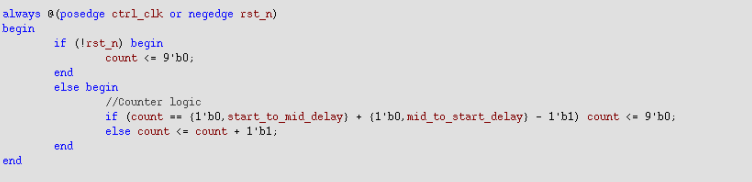
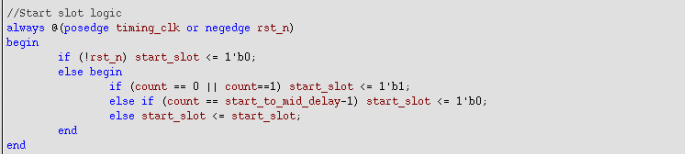
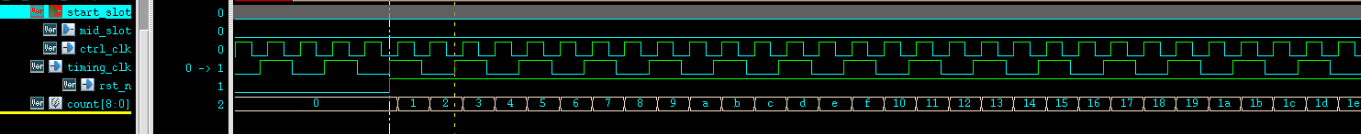


**12. Clock Domain Timing Issue – Missed start\_slot**

In this case, you can observe that the **counter runs on the ctrl\_clk domain**, while the **start\_slot signal is generated based on the rising edge of the timing\_clk**.

If **timing\_clk is slower than ctrl\_clk**, the following issue can occur:

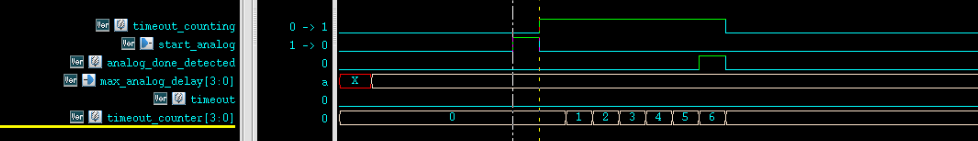
* The counter reaches specific values (e.g., 0, 1, etc.) **before the rising edge of timing\_clk**.



**23-06-2025**

**13. Incorrect Timeout Counter Initialization**

The timeout counter does not start correctly, causing the actual analog delay to exceed the user-specified max\_analog\_delay.



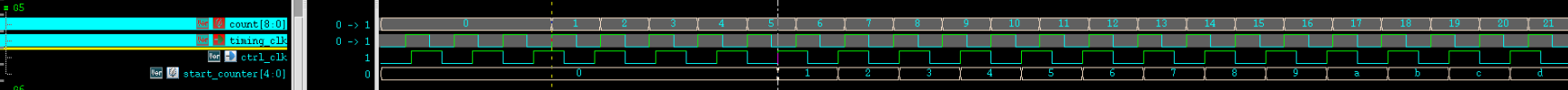
**14. Design Constraint: Minimum Delay Between Start and Mid Slot**

To ensure correct operation and prevent premature slot detection due to synchronization latency, the following design constraint must be enforced:

**The delay between start\_slot and mid\_slot must be greater than 36 controller clock cycles.**

**Rationale:**

* The start\_slot\_detected signal is generated after a synchronization delay of approximately **4 controller clock cycles**.
* The programmable delay counter begins after this detection.
* To guarantee that the next slot does not occur before the delay period completes, a **minimum of 36 controller clock cycles** is required between start\_slot and mid\_slot, including synchronization overhead.

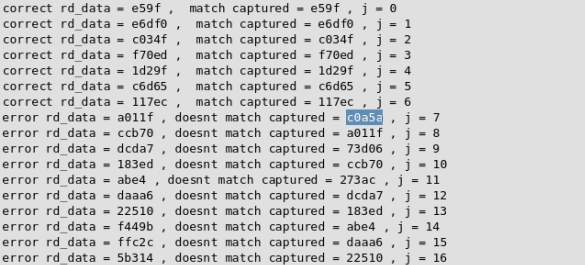


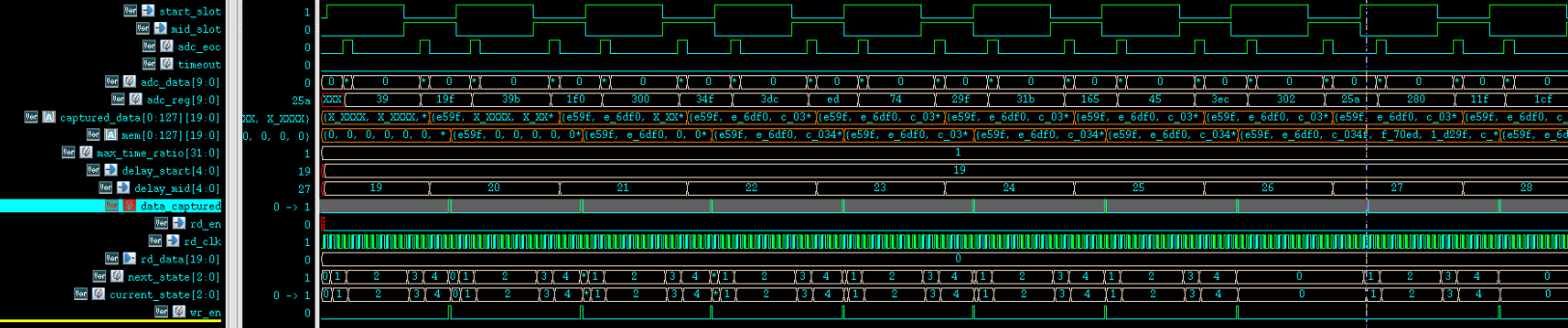
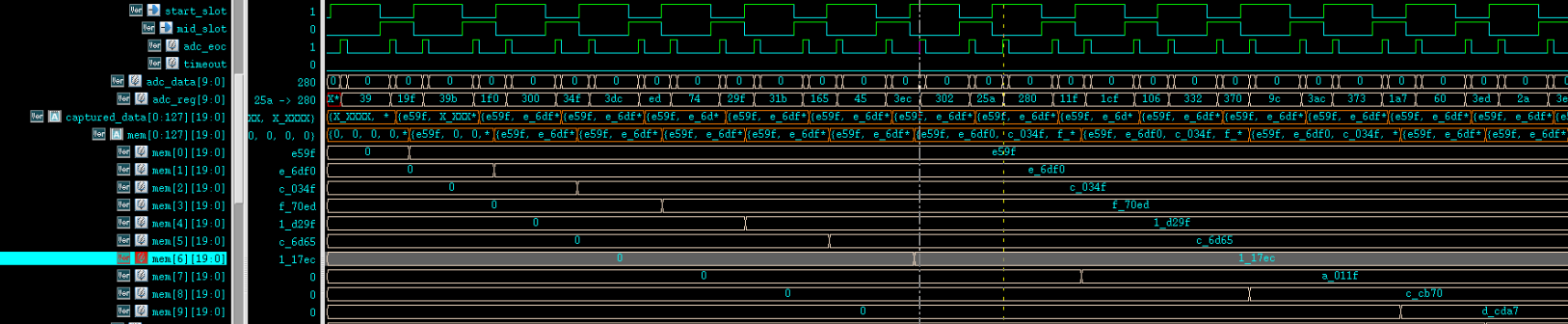
**15. FIFO Skipping Values After 7th Write – solved**

In this case, the FIFO begins to **skip values starting from the 8th write**, even though the ADC data is valid and the programmable delays are within the specified range.

To demonstrate this, a timeout trace has been plotted alongside the captured data. The trace confirms that each data point falls within a **valid programmable delay window**, and the ADC is triggered correctly.

This indicates that the issue is not with analog timing or timeout logic

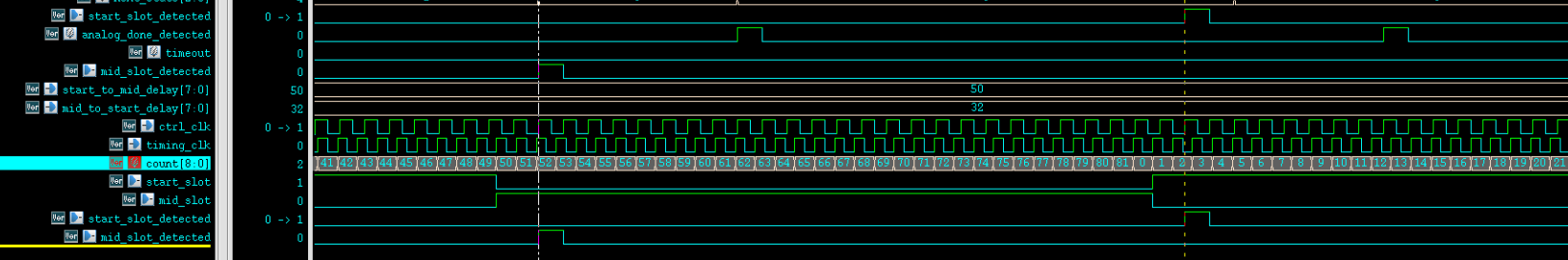


  
Debug: **Checker Mismatch Due to Timing Clock Change**

Following recent changes to the **timing clock period**, a few test checkers did not correctly account for the updated clock behavior. As a result:

* The computed **delta between start\_slot and mid\_slot** became **smaller than the intended valid range**.
* This led to incorrect behavior in the test, including **false failures or unexpected results**.

The issue was not in the design itself but in the **testbench assumptions and setup**.  
 It has now been resolved by updating the relevant test cases to align with the new timing configuration, ensuring that all slot intervals fall within valid limits.

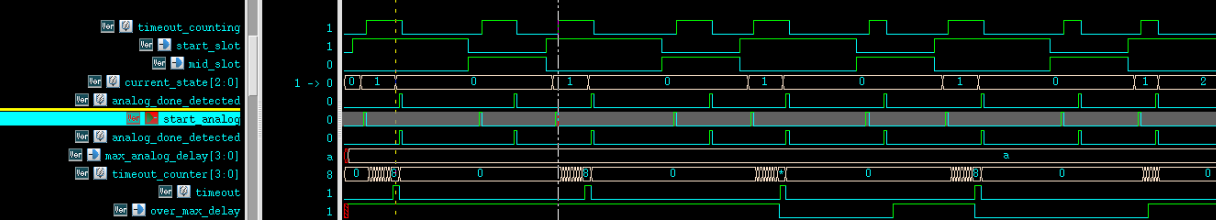


16. **Timeout Counter Not Active During Mid Slot - SOLVED**

Due to the current implementation, the **timeout counter is only active during the START slot** and not during the MID slot. This behavior occurs because:

* The counter is **reset as soon as analog\_done is detected**, which typically happens at the end of the START state.
* The counter **only starts again in the START state** of the next capture cycle.
* As a result, during the MID state, the timeout counter is not running, and **no timeout detection** occurs for that phase.

This causes the design to **miss timeout conditions** during the MID slot, leading to **incorrect acceptance of delayed or invalid data**. The timeout logic needs to be extended or duplicated for the MID slot to ensure proper coverage and correctness.

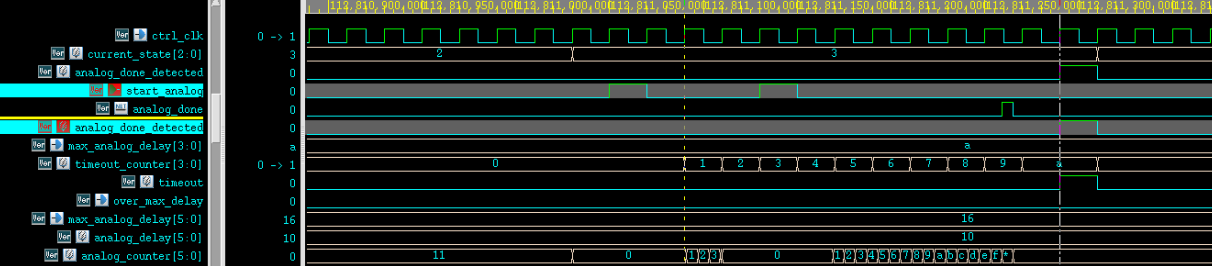
Debug: **Designer didn't realize it was a feature of their FSM**

17. **Multiple start\_analog Triggers Before analog\_done Causes False Timeout - SOLVED**

In the current behavior, **multiple start\_analog signals are being generated before a single analog\_done is detected**. This leads to a critical issue:

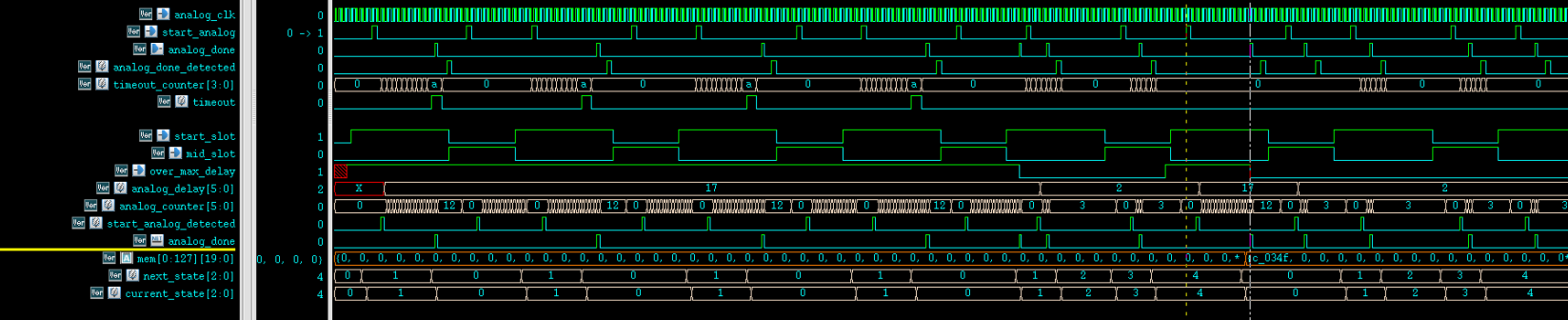
* Each new start\_analog **resets or restarts** the timeout counter.
* If analog\_done does not arrive in time relative to the latest start\_analog, the system incorrectly flags a **timeout**, even though the data may still be valid from the initial trigger.

This results in **false timeout detections**, which can cause valid ADC data to be incorrectly discarded or skipped.

Debug: **A design constraint was violated**

18. **Timeout Counter Not Active After Start Slot - SOLVED**

In the highlighted case, the **start\_slot signal has arrived**, but the **timeout counter is not running** as expected. This leads to missed timeout detection.

Debug: **A design constraint was violated**

**19. wr\_ptr Does Not Reset to Zero Without Global Reset**

The write pointer (wr\_ptr) logic in the design is structured such that **it does not return to 0 unless a full system reset is applied**.

As a result:

* The condition wr\_ptr\_bin < rd\_ptr\_bin is **never satisfied** during normal operation.
* This affects **code coverage**, as any logic depending on wr\_ptr wrapping or resetting is never exercised

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.