Second assignment:
FSM design and RTL implementation with UART input:

1. Overview : Design, implement and verify a sequential digital circuit. Combine concepts of serial communication, shift registers, pattern detection and clock domain crossing.

1.1 Problem description : Implementing a pattern detector system that identifies a given 4-bit sequence on a serial input stream.

; this pattern is the last decimal digit of my ID.

converted into 4-bits binary like this:

ID reference : 100419 14 36 → $6_{10} \longleftrightarrow 0110_2$

1.2 implementation: The project it focuses on the implementation of the 4-bit pattern detector using verilog, integrating sequential and combinational logic with serial communication (UART) and multy-clock synchronization

2. Funcional overview: System receives a continuas serial stream at 115.200 bps via UART and detects the specific bit pattern inside the incoming data stream.

- Every time the sequence appears (even overlapping) the circuits assets a match (match = 1) pulse for one system clock cycle at 25 MHz.

e.g: Two overlapping pattern detected:

input : [0011 0110] as we see, there are overlapping.

match → input : [00010010] <=>  (wave) from

3. System architecture and module imp...

The complete system is built with five (5) modular blocks, connected hierarchically under a top-level entity.

· 3.1.

This design applies real-world digital principes such as:

· Baud rate generation (sequential)

Derived from the equation:

$$f_{baud} = \frac{f_{clk}}{DIV}$$

$$DIV = f_{Hz} / Baud\ rate$$

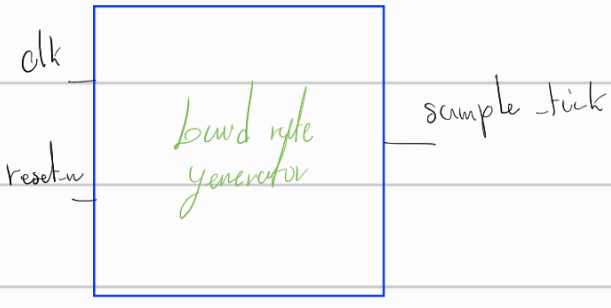
for our specific case:

$$DIV = 25.000.000 / 115.200 \approx 217$$

· His function is generates the baud sampling tick from the 25 MHz clock.

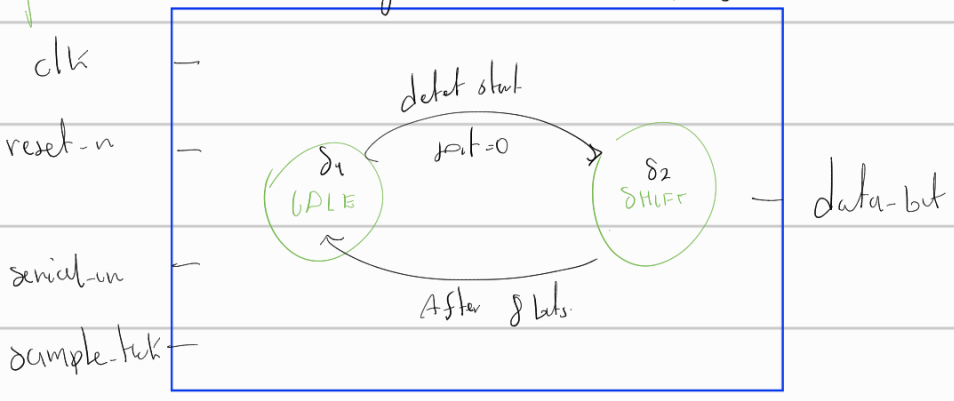
I/O: inputs: clk (25 MHz), reset_n (low active)

outputs: sample_tick; our sampling pulse that indicates when to capture data.



- FSM-based bit-sampler (sequential FSM or UART sampler) → start bit detection ensures proper bit alignment under asynchronous serial transmission.
  ↳ Detects start bit, shift 8 data bits.

I/O: inputs: clk, reset_n, serial_in, sample_tick
outputs: data_bit (goes to our sipo)

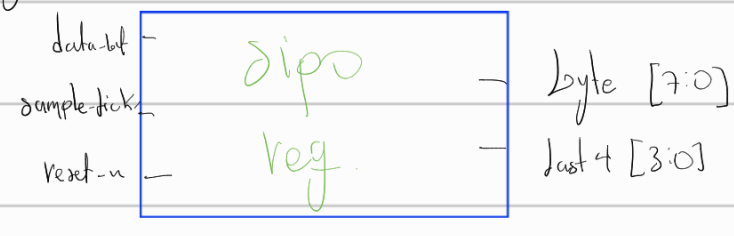


This is our two finite states machine, that detects a start bit (at low) goes to (S1) and shifts 8 serial bits, using sample_tick.
then, returns to (S1)

- Shift-register serial data handling: (sequential or sipo register)
  ↳ Stores 8 received bits and exposes last 4 bits.
  (convert serial data into parallel form)

  . With each sample_tick shifts a bit to inside.
  This give 8 bits y A less significative.

I/O inputs: data_bit, sample_tick, reset_n
outputs: byte [7:0] ~ last4 [3:0]



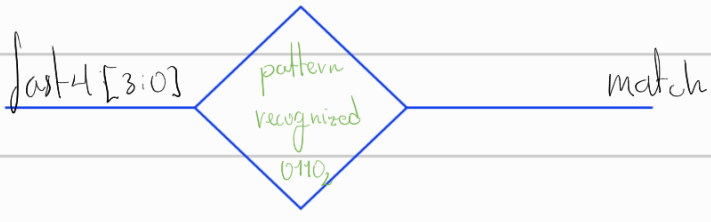
· Pattern matching with overlap detection.

(Combinational → detector) ↳ Digital comparator with a moving 4-bits window

↳ His function is compares last 4 bits with propose pattern (0110_2) → 6_{10}

I/O : input: last4 [3:0]

output: match



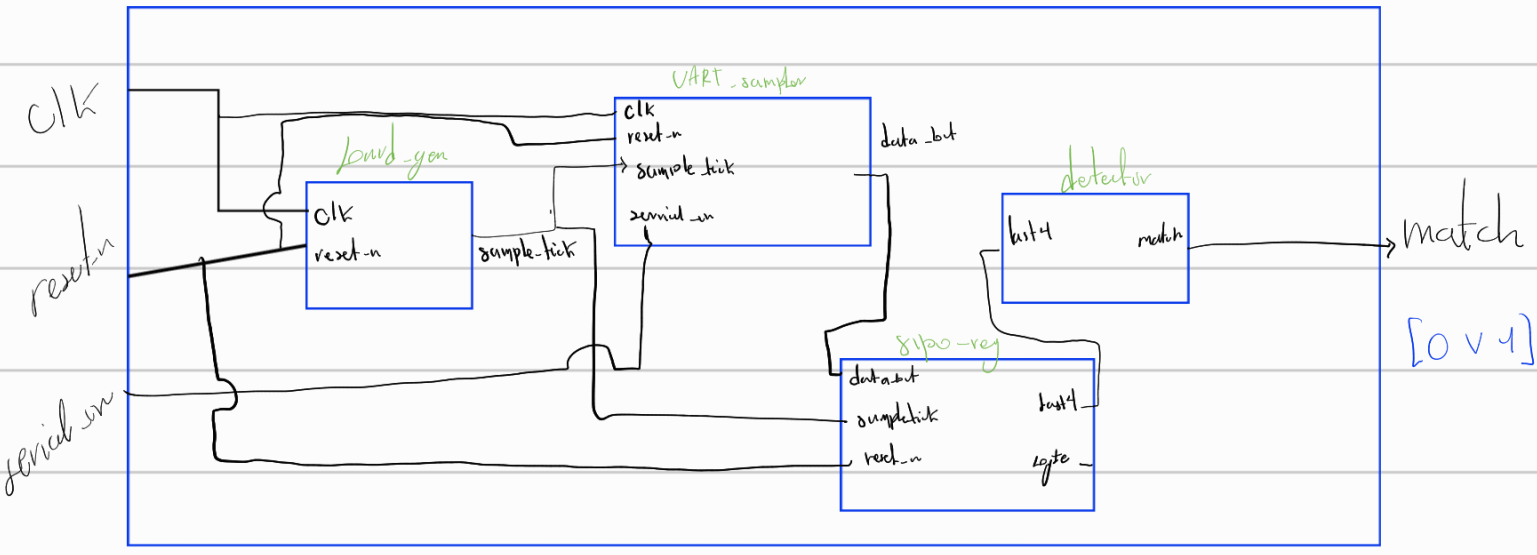
· Cross-domain synchronization (Structural and final module)

↳ Integrate all modules and synchronizes match.

Also, the output synchronization (2 flip-flops) avoid METASTABILITY, when the UART tick and system clock are asynchronous.

4. Lets see the diagram:

I/O inputs: clk, reset_n, serial_in
output: match



5 Verify modules:

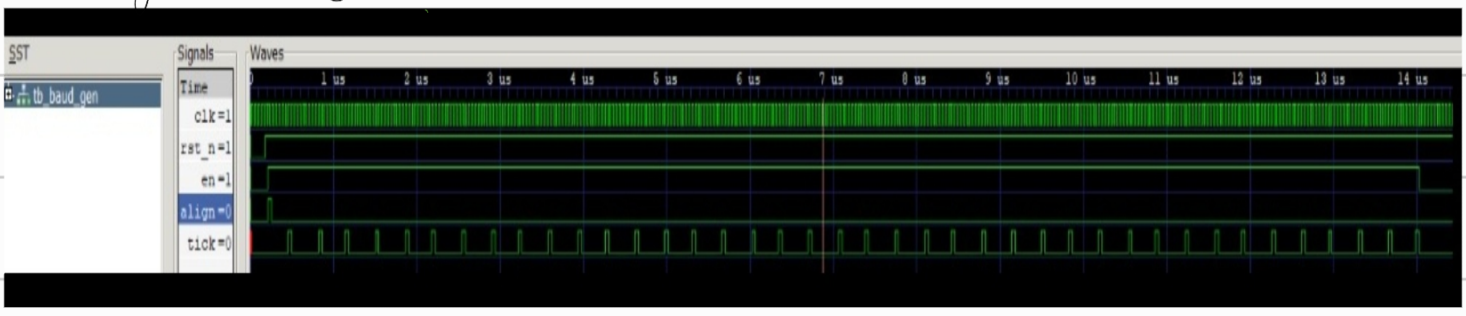
- Once we define our modules and their respective TB on verilog, we can analyse the simulate case, where: ID is 1004194436, last digit: $6_{10} \Leftrightarrow 0110_2$
  - Valid UART stream containing pattern
  - Overlapping patterns (e.g. 01100110 ✓ 00110110
  - No pattern present
  - Noise or start/stop bits
  - Reset mid / transmission

Expecting the following results:

- match pulses are single-cycle, precisely alligned.
- Pattern overlap generate multiple pulses.
- Reset clears internal registers

i) Baud rate generator:

The "baud_gen" module divides the system clock frequency to produce a single-cycle "tick" pulse that defines the sampling rate for the UART system. It ensures synchronization between the serial input line and the receiver internal logic sampled by UART.



- Simulation setup:

clk (system clock input): 1MHz ~ Baud rate: 115200 bps

, given a expected divisor 868. ie, the generator alternates between 8 and 9 clock cycles per tick.

. keeping an average frequency close to 115.2 KHz.

Signals on time: en: Enable signal counting while high.

· align: Allignment pulse (reset dividers), single short pulse at start bit detection.

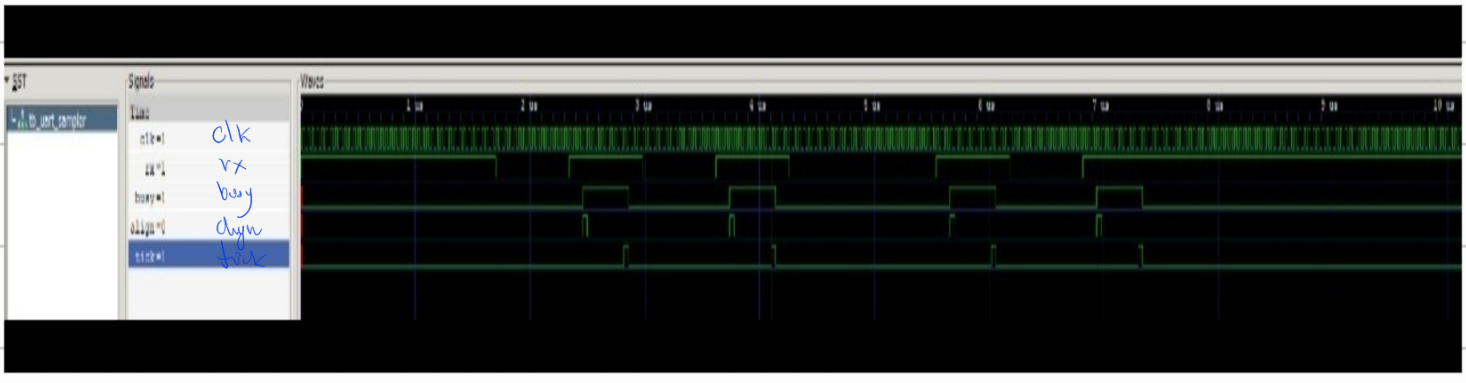
· tick: Output pulse at baud freq. We observe a regular single-clock pulses spaced by 8-9 cycles of clock.

Conclusion: As we see on the GTKWave simulation that, "align" resets the internal counter, ensuring the first tick aligns with the UART start bit.

· "Tick" pulses are uniform and 1 clock wide, confirming clean generation without jitter. Besides, when "en" is deasserted, "tick" immediately stops, demonstrating correct enable control. Given a fully validated module.

(i) UART sampler.

The uart_sampler is responsible for detecting the start bit on the UART serial line "rx", aligning the baud-rate generator "align", and generating one-bit samples (bit_valid_pulses) for each received bit during a data frame.



Setup: rx: Serial input line, align: sync pulse to baud_gen, tick as baud retrence pulses and busy to detect the frame active flag.

As we see, simulation bridges the asynchronous UART input with the synchronous system clock domain.

- "Rx" falling edge correctly produces an "align" pulse, "tick" pulses are periodic and properly synchronized, and "busy" stays high during the UART frame, confirming correct state control.

Thus, gave use a fully validated module to integrate with the detector and TDF module.

iii) Detector

This validate that the module asserts "match" when window [3:0] equals pattern [3:0]

Setup: pattern [3:0]: Expected 4-bit sequence as fixed reference value ($b_{10} = 0110_2$)

window [3:0]: Current input bits that cycles through multiple 4 bit values.

finally match: as detection flag, high only when window = pattern.

As we see, signal transition timing is stable, and no false matches occur, working correctly ready for use within the top level pattern recognition chain.

iv) Top integration.

On this case, we try to validate complete UART based-pattern detection through three test cases.

Case 1: About 200 us, we send test byte as: 0x FF (no pattern to match)

Case 2: Next, send 0x 66 (two occurrences of 0110_2 overlapped) → system works.

Case 3: Finally, send 0x 06 and 0x 60 (match at start and end) → two separate pulses.

- As we see, at this point, we have a full system verified where, the integrated modules (baud_gen, uart_sampler, detector) operate coherently.

Finally, we have this final full-ID verification:

1. Send a UART input sequence ASCII: "1004191436"
2. Each digit transmitted as an 8-bit UART frame.
3. Pattern to detect is: $0110_2 = 6_{10}$

given clk = 1.6 MHz ~ Baud = 100 kbds.

- As we can see, all ten characters of the ID were receiving through UART.

Also, only the final digit "6_{10}" contained the target 4-bit pattern. Finally, the detector generated a single match pulse at the end of the sequence, proving end-of-ID validation works.

6. Summary and conclusions

- The detector module works correctly - it performs a clean 4-bit equality check, setting match high exclusively when the pattern and input window are identical.

- The complete design now demonstrates full functional integration: UART reception, bit sampling, serial to parallel conversion and pattern detection aligned with the expected LED-based rule.

- This confirms functional correctness and timing-alignment of the complete FSM design with UART input.