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:

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

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

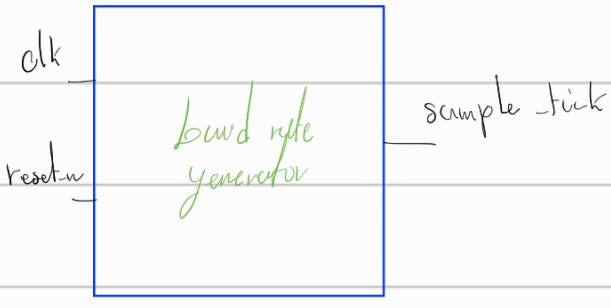
for our specific case:

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

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

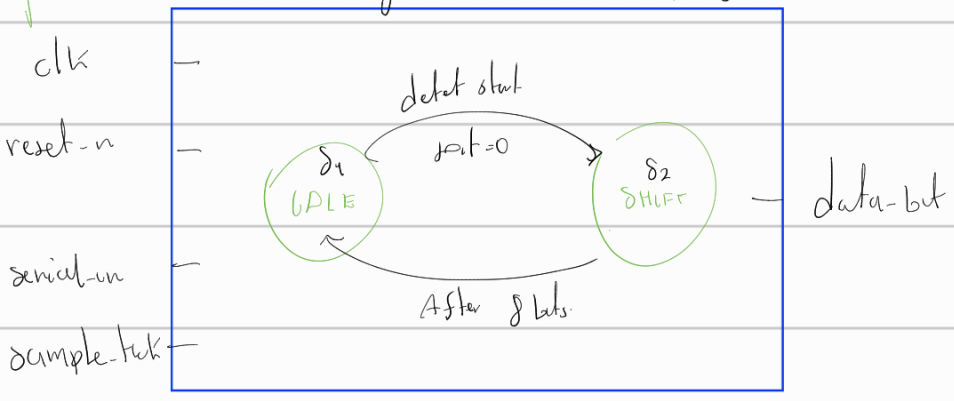
I/O: inputs: clk (25 MHz), reset_n (low active)

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)

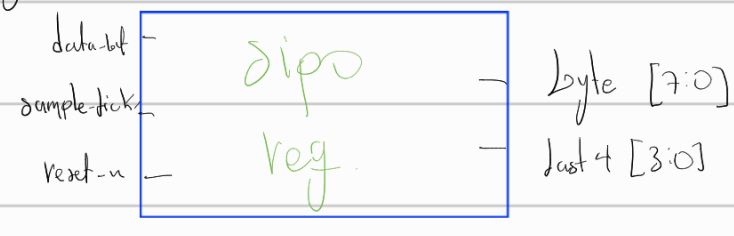


This is our two finite states machine, that detects a start bit (at low) goes to (S1) and shifts 8 serial bits, using sample_tick.
then, returns to (S1)

- 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]



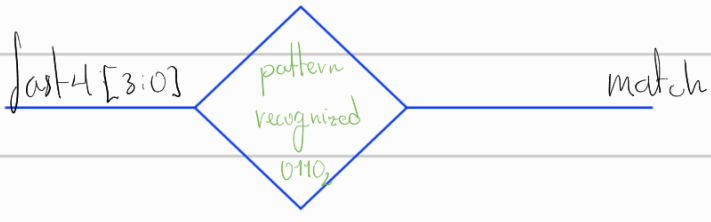
· Pattern matching with overlap detection.

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

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

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

output: match



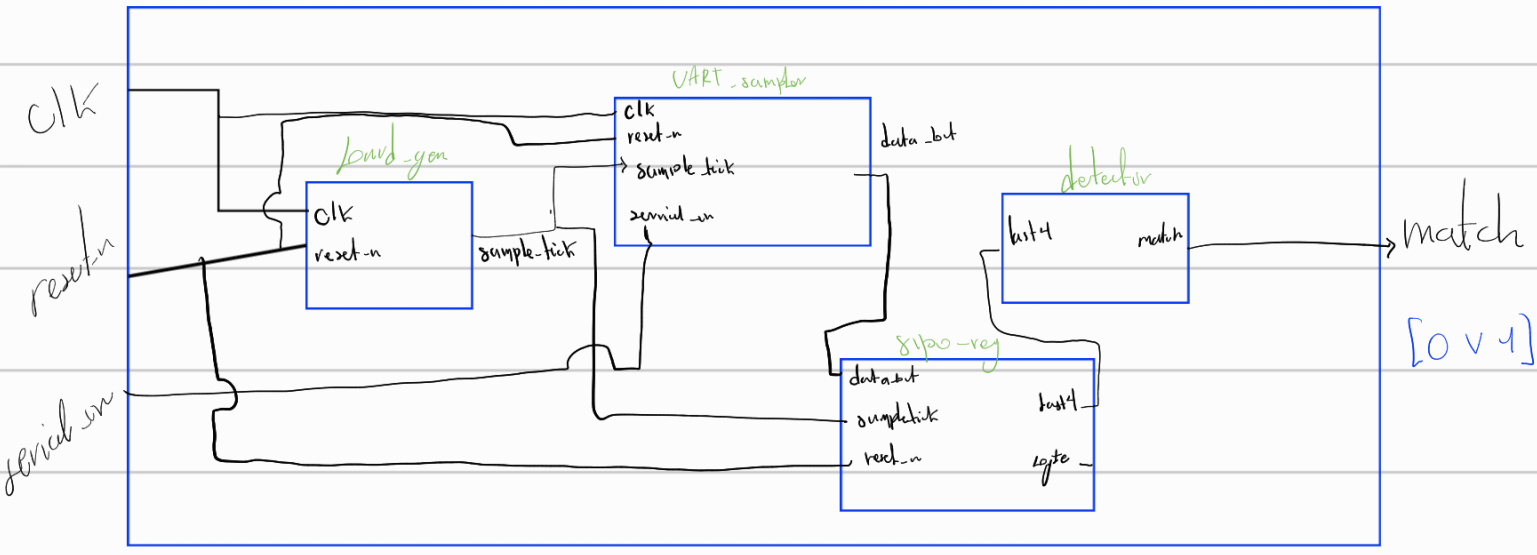
· Cross-domain synchronization (Structural and final module)

↳ 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: LP is 1004194436, last digit: $6_{10} \Leftrightarrow 0110_2$
  - Valid UART stream containing pattern
  - Overlapping patterns (e.g. 01100110)
  - 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:

6. Summary and conclusions