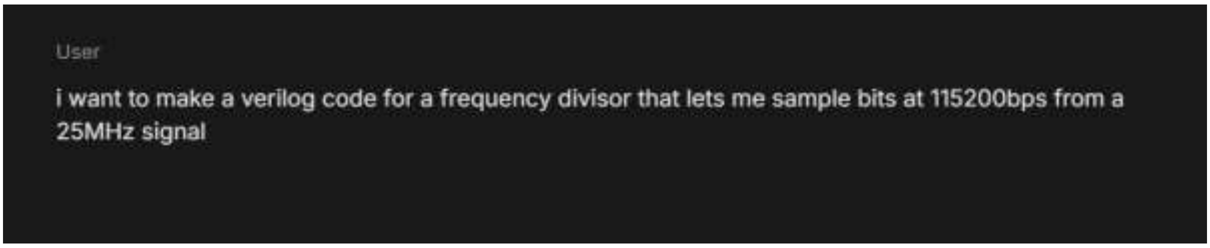
UART Pattern Detector

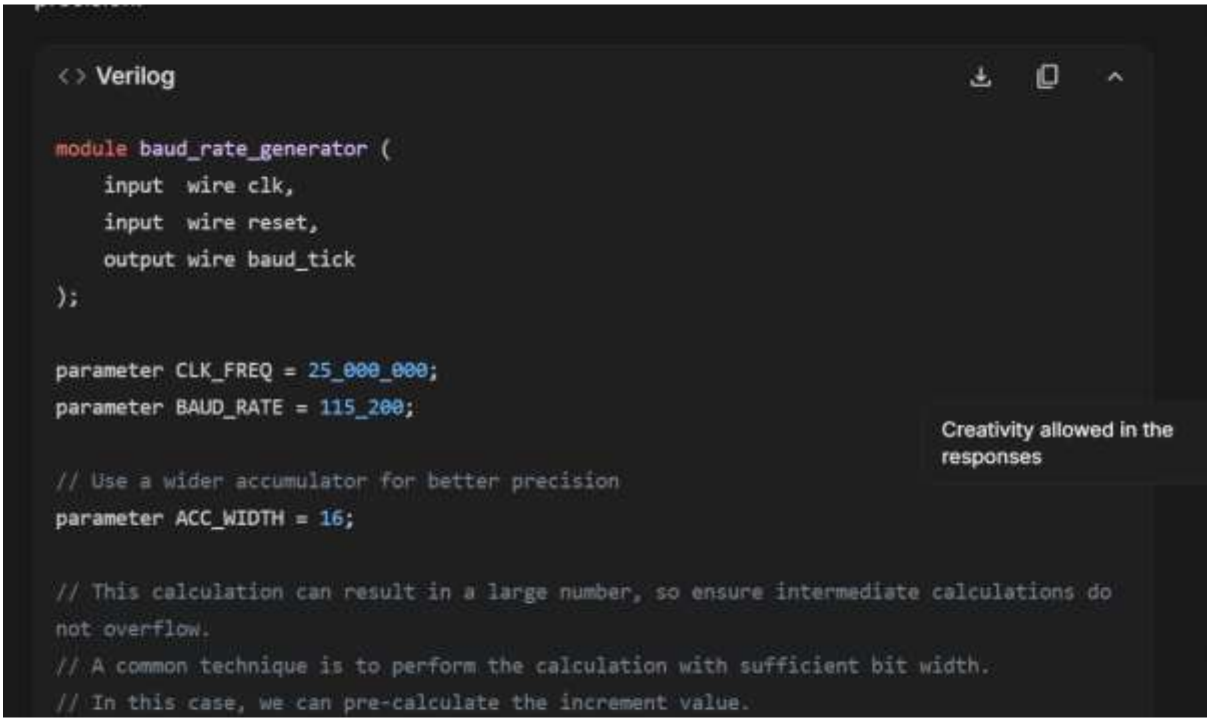
The Ai used as helper for this tas kwas Google Ai studio, the last id number is 8 (1000 in binary)

-Baud_Rate_Generator:



i want to make a verilog code for a frequency divisor that lets me sample bits at 115200bps from a 25MHz signal

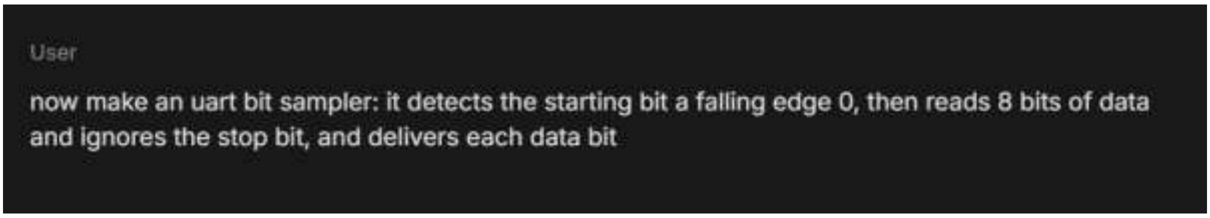
In the prompt it was specified that it should be a frequency divisor, in order to lean the ai more trowards the correct answer; the first prompt fullfilled the requirements so no addings were needed



```verilog
module baud_rate_generator (
    input  wire clk,
    input  wire reset,
    output wire baud_tick
);

parameter CLK_FREQ = 25_000_000;
parameter BAUD_RATE = 115_200;

// Use a wider accumulator for better precision
parameter ACC_WIDTH = 16;

// This calculation can result in a large number, so ensure intermediate calculations do not overflow.
// A common technique is to perform the calculation with sufficient bit width.
// In this case, we can pre-calculate the increment value.
```

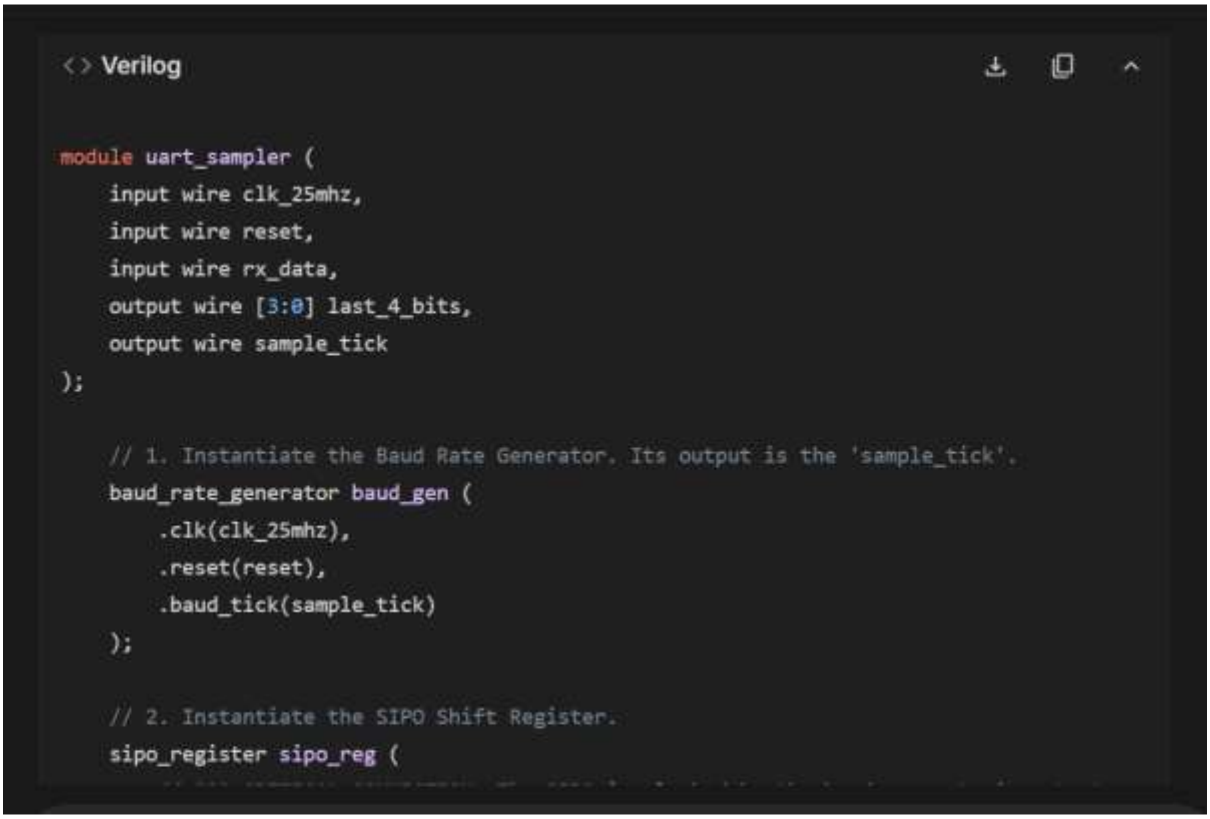
UART_Sampler

At first i made the ai define uart, and then asked to make the sampler acording



now make an uart bit sampler: it detects the starting bit a falling edge 0, then reads 8 bits of data and ignores the stop bit, and delivers each data bit

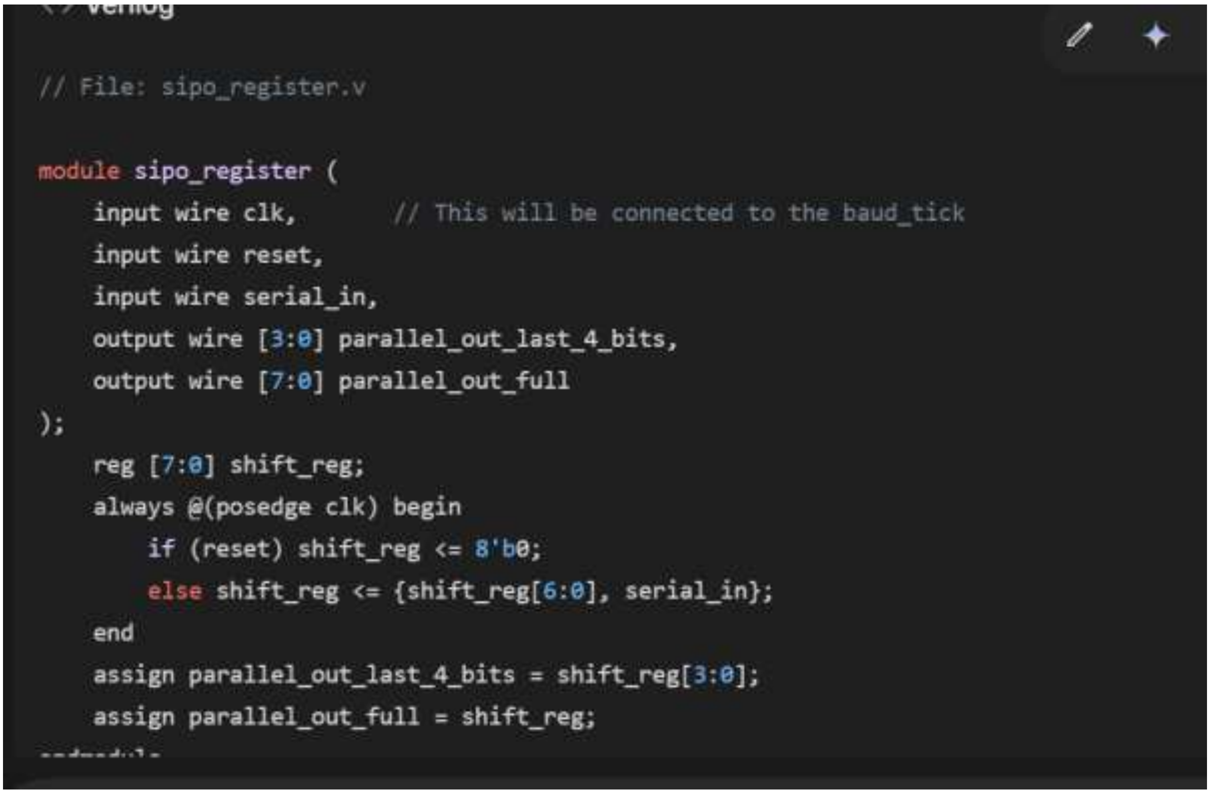
But it gave an over extense code, so later i asked it to make it taking into account the previous Baud_rate generator which got me a simpler code as follows:



```verilog
module uart_sampler (
    input wire clk_25mhz,
    input wire reset,
    input wire rx_data,
    output wire [3:0] last_4_bits,
    output wire sample_tick
);

    // 1. Instantiate the Baud Rate Generator. Its output is the 'sample_tick'.
    baud_rate_generator baud_gen (
        .clk(clk_25mhz),
        .reset(reset),
        .baud_tick(sample_tick)
    );

    // 2. Instantiate the SIPO Shift Register.
    sipo_register sipo_reg (
```

SIPO:

Similar to uart i asked it for the definition and made the initial prompt which was later refined along with the uart

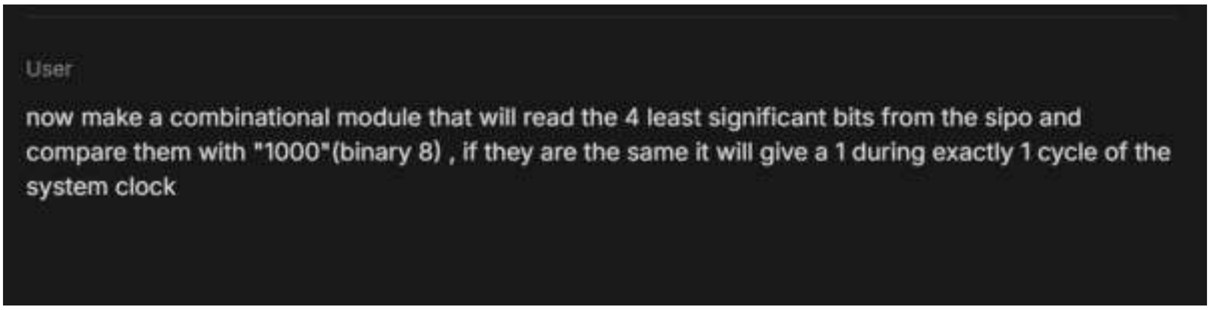


```verilog
// File: sipo_register.v

module sipo_register (
    input wire clk,       // This will be connected to the baud_tick
    input wire reset,
    input wire serial_in,
    output wire [3:0] parallel_out_last_4_bits,
    output wire [7:0] parallel_out_full
);
    reg [7:0] shift_reg;
    always @(posedge clk) begin
        if (reset) shift_reg <= 8'b0;
        else shift_reg <= {shift_reg[6:0], serial_in};
    end
    assign parallel_out_last_4_bits = shift_reg[3:0];
    assign parallel_out_full = shift_reg;
```

The code sucefully does what expected for the data signal, with a generic clk we can change attaching it to baud_rate generator

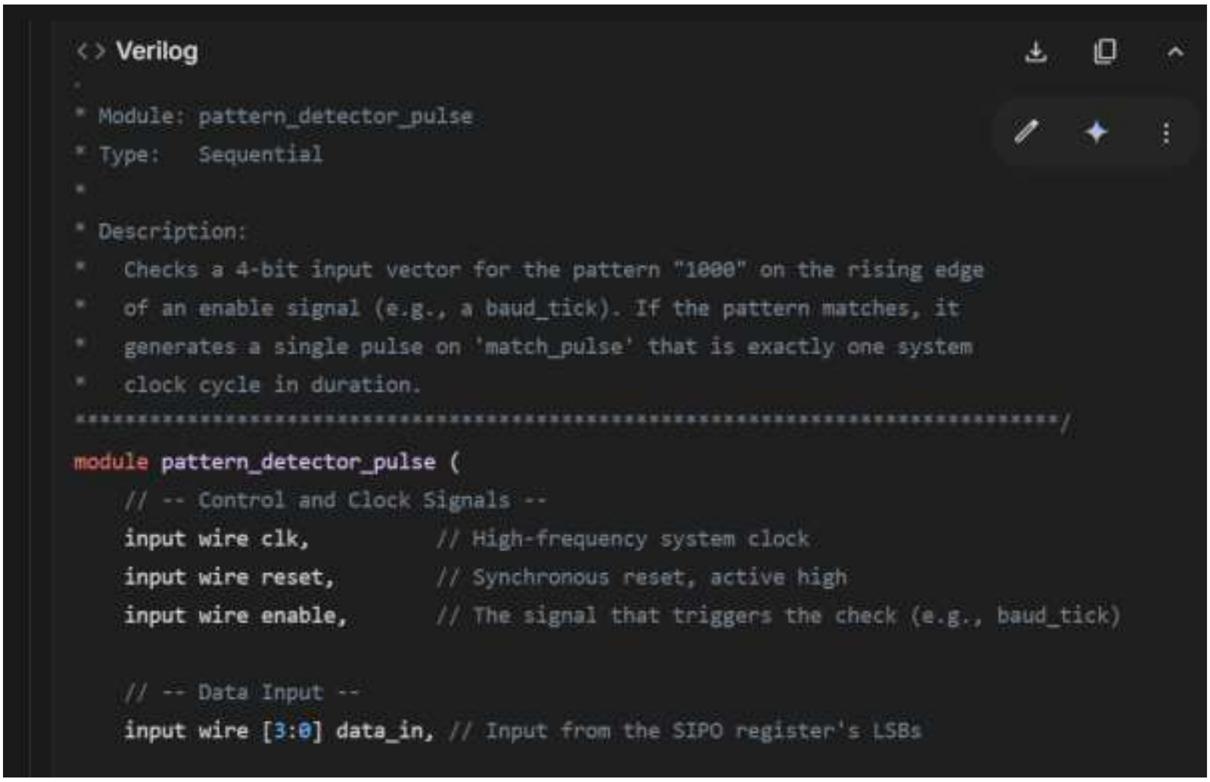
Detector:

This one is the simplest as it has combinational logic



now make a combinational module that will read the 4 least significant bits from the sipo and compare them with "1000"(binary 8) , if they are the same it will give a 1 during exactly 1 cycle of the system clock

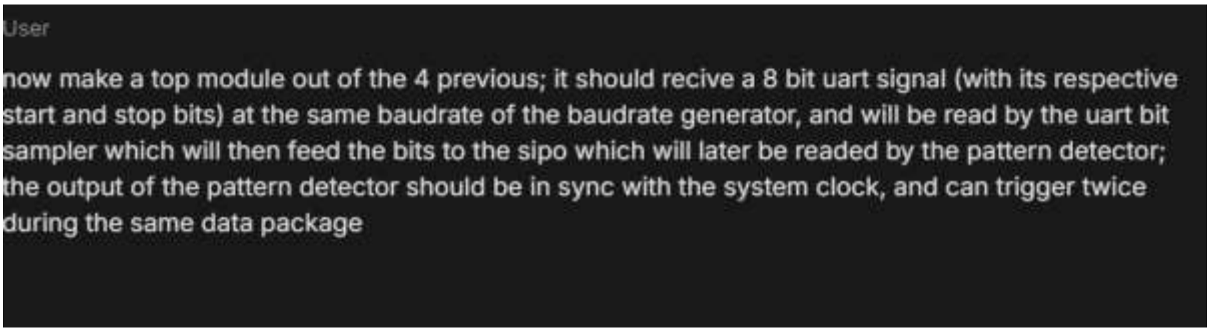
From this the Ai gave me 2 codes, once which was purely combinational and other that was synchronous, since we need it to have exactly 1 tick, i selected the synchronous



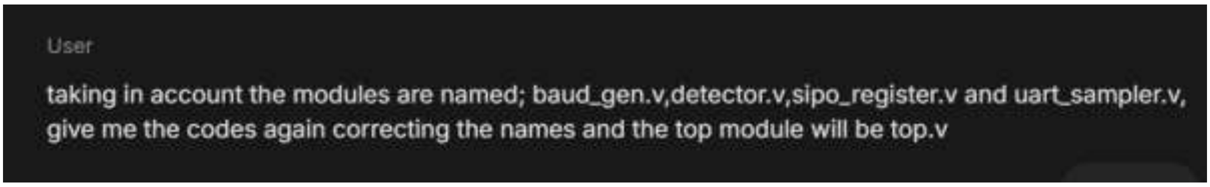
```verilog
*
* Module: pattern_detector_pulse
* Type:   Sequential
*
* Description:
*   Checks a 4-bit input vector for the pattern "1000" on the rising edge
*   of an enable signal (e.g., a baud_tick). If the pattern matches, it
*   generates a single pulse on 'match_pulse' that is exactly one system
*   clock cycle in duration.
******************************************************************************/
module pattern_detector_pulse (
    // -- Control and Clock Signals --
    input wire clk,          // High-frequency system clock
    input wire reset,        // Synchronous reset, active high
    input wire enable,       // The signal that triggers the check (e.g., baud_tick)

    // -- Data Input --
    input wire [3:0] data_in, // Input from the SIPO register's LSBs
```

Top_module:

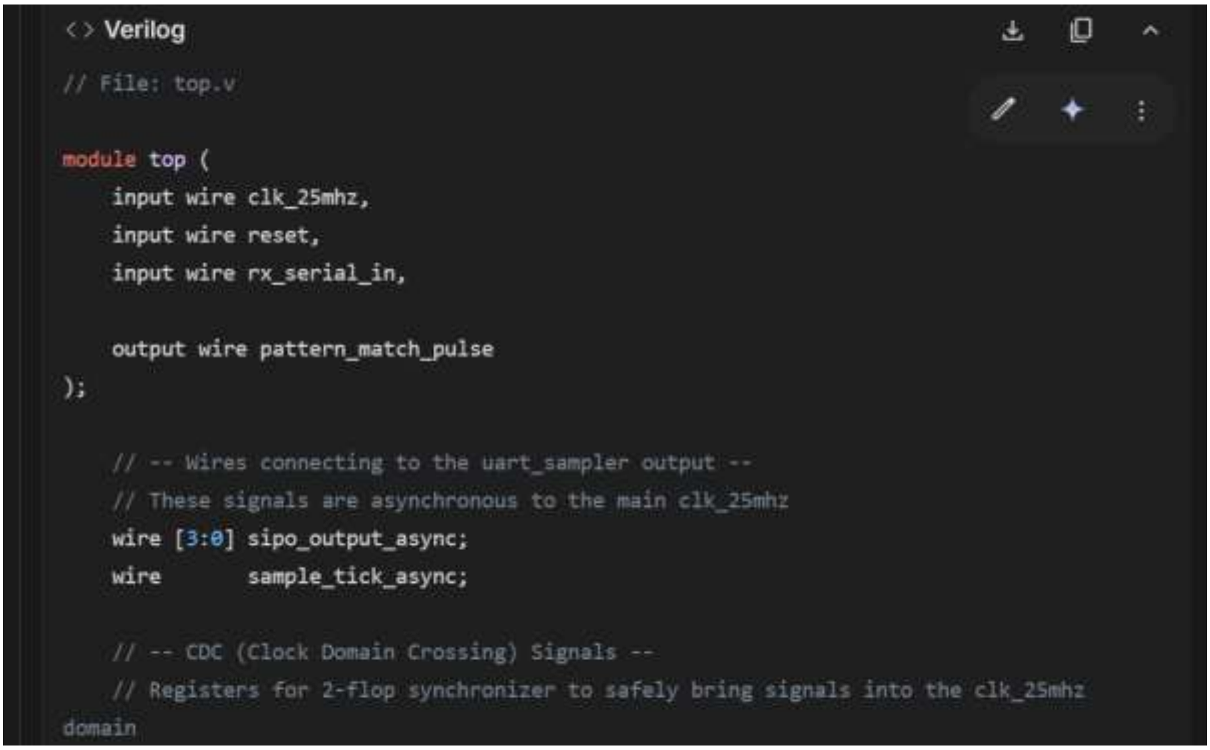
For this one there were some issues since the ai forgot to use certain of the others modules despite being told to, so after some tries, I got a prompt that implemented each module correctly, and also made it correct the names as I were at it



now make a top module out of the 4 previous; it should recive a 8 bit uart signal (with its respective start and stop bits) at the same baudrate of the baudrate generator, and will be read by the uart bit sampler which will then feed the bits to the sipo which will later be readed by the pattern detector; the output of the pattern detector should be in sync with the system clock, and can trigger twice during the same data package

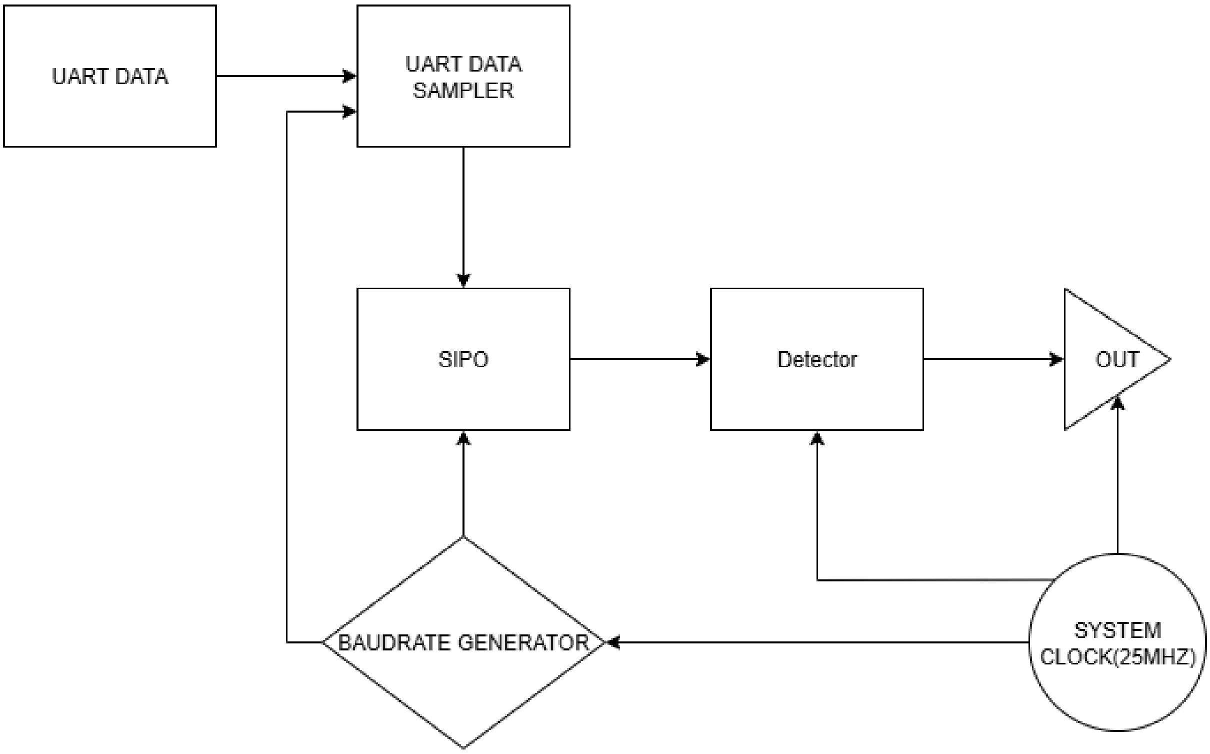


taking in account the modules are named; baud_gen.v,detector.v,sipo_register.v and uart_sampler.v, give me the codes again correcting the names and the top module will be top.v



```verilog
// File: top.v

module top (
    input wire clk_25mhz,
    input wire reset,
    input wire rx_serial_in,

    output wire pattern_match_pulse
);

    // -- Wires connecting to the uart_sampler output --
    // These signals are asynchronous to the main clk_25mhz
    wire [3:0] sipo_output_async;
    wire       sample_tick_async;

    // -- CDC (Clock Domain Crossing) Signals --
    // Registers for 2-flop synchronizer to safely bring signals into the clk_25mhz
domain
```

Flow Diagram


UART DATA
UART DATA SAMPLER
SIPO
Detector
OUT
BAUDRATE GENERATOR
SYSTEM CLOCK(25MHZ)