Project: Designing an asynchronous serial data receiver

A computer have to send data to a peripheral device like a teletext or a monitor using a simplex serial TX-RX interface (Fig. 1). Simplex means that the transmission takes place in one direction, unidirectional, from the computer to the terminal device. It is also possible for instance between a mouse (sender) and a computer (receiver).

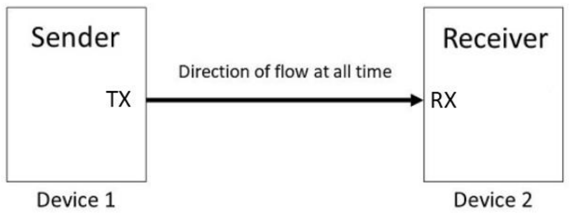

Fig. 1 Simplex communication interface.

Transmission parameters are as follows: Data length = 8 bit, parity bit = odd parity, stop bits = 1 bit, transmission speed = 600 b/s.

The receiver is always idle waiting for a transmission frame as represented in Fig. 2. The line RX is kept at '1' by the transmitter when it is not in use. When the transmitter has to send a byte, initialises the transmission with a start bit that is always '0', follows with the byte ordered from D7 .. D0, adds an odd parity bit (OP) and finally one stop bit that is always '1'. For instance, if the data byte is the ASCII character "K" = "01001011", the transmission frame looks like this 11-bit sequence: "00100101111".

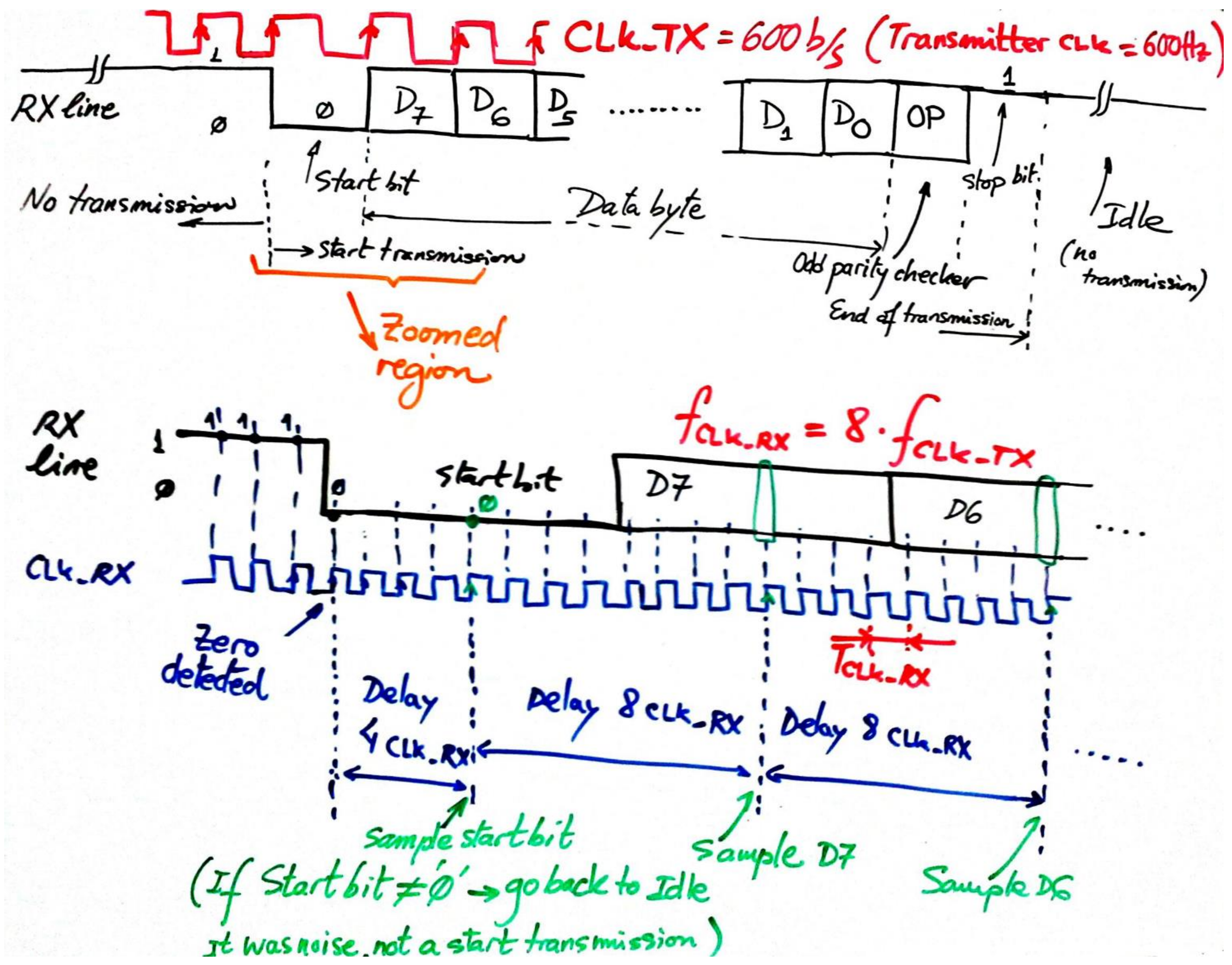

Fig. 2 A transmission frame and a zoomed region at the transmission start. The receiver circuit starts operating when a '0' is detected at the RX input line.

While receiving, if the start bit is not sampled to be '0' or the stop bit is not sampled to be '1', there is an error. There is also an error indication when the 9-bit parity checking is even instead of odd. The error signal is a pulse.

The symbol of the receiver subsystem is represented in Fig. 3. A pulse DA (data available) is generated once the sequence has been decoded and saved at the *Data_out* vector.

The receiver works with a 24 MHz quartz crystal oscillator, and, as indicated in Fig. 2 the *CLK_RX* is 8 times the transmitter frequency. In this manner, it is able to detect asynchronously and quickly the falling edge from '1' to '0' that starts the transmission and then to centre the samplings at the half of each bit being transmitted.

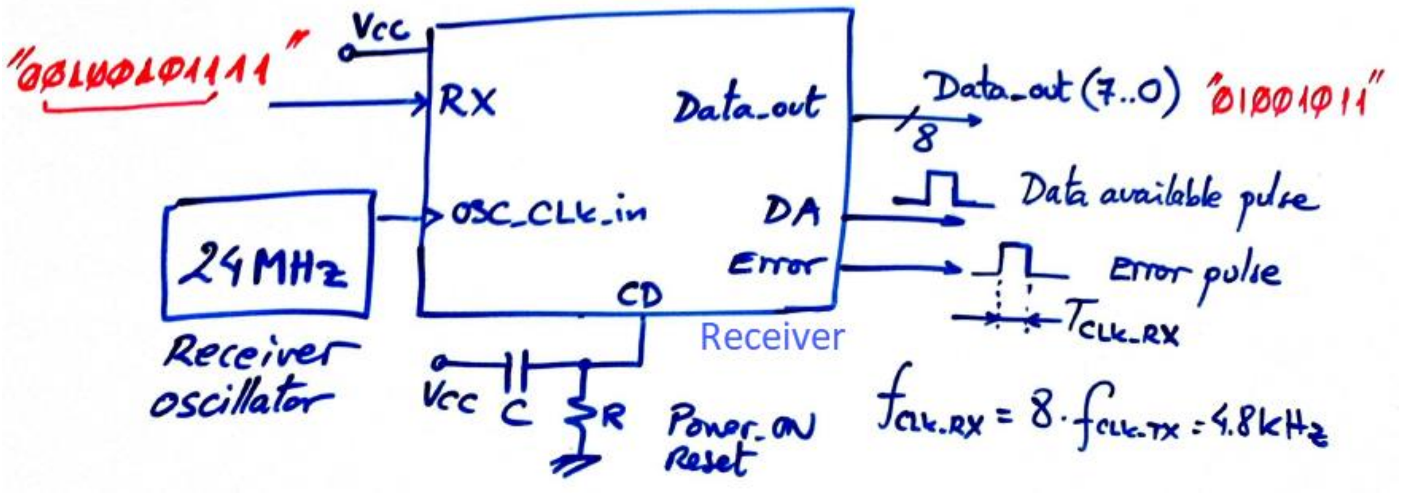

Fig. 3 Symbol

Fig. 4 gives you an idea of the components that may be included in the dedicated processor datapath. Basically, counters and data and shift registers and also the 9-bit parity checker.

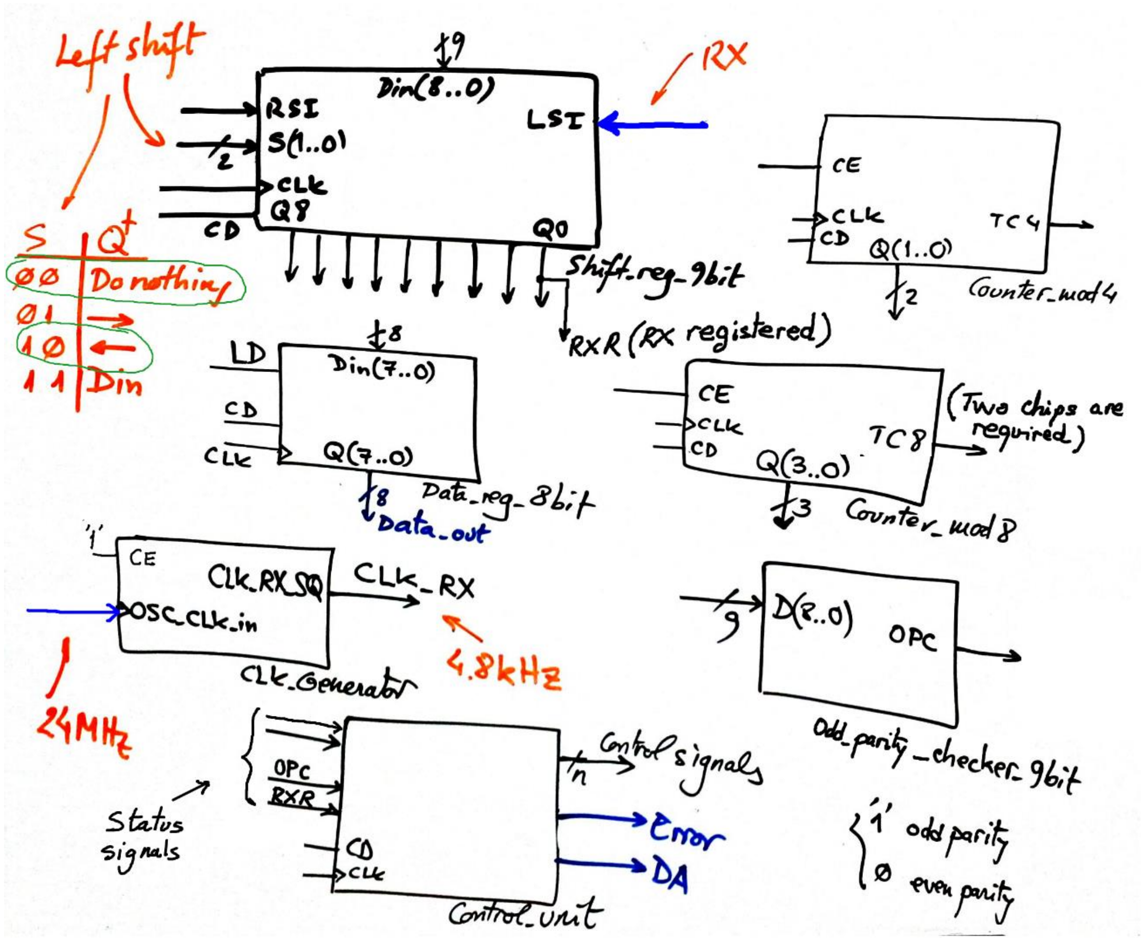

Fig. 4 Example of some components that may be used in the design of the receiver.

Fig. 5 gives you an idea of possible states for the control unit of the dedicated processor.

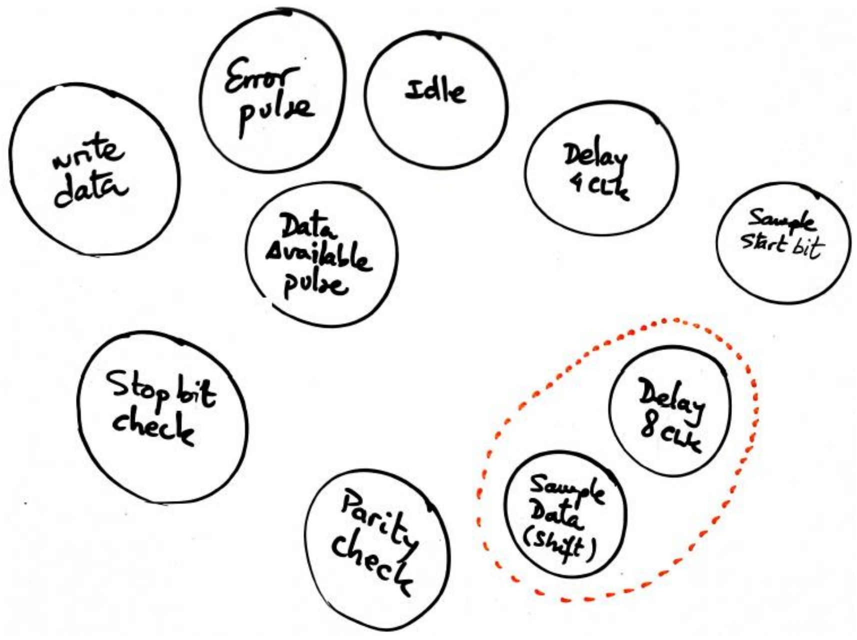

Fig. 5
Example of feasible states for the control unit. Naturally, you can invent your own states and FSM organisation.

Questions (chapter II)

1. Invent an internal architecture for the *CLK_Generator* in Fig. 4 following a plan C2 based on *T_FF* and *freq_div_N* components where *N* is an ajustable parameter. (1p)

2. Using a plan C2 strategy and the universal chip in Fig. 6 design the internal architectures of the *Counter_mod4* and the *Counter_mod8* required in the project for delaying four CLK_RX, eight CLK_RX and also for counting the eight data bits. (1p)

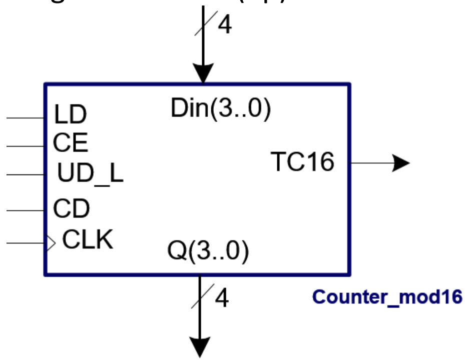

Fig. 6
Symbol of the universal 4-bit synchronous binary counter.

3. Infer the truth table of a 4-bit odd parity checker combinational circuit represented in Fig. 7 so that when the input data has an odd number of '1', the output OPC = '1'. Explain how several chips of the same kind can be connected to produce the *Odd_parity_checker_9bit* required in the project. (1p)

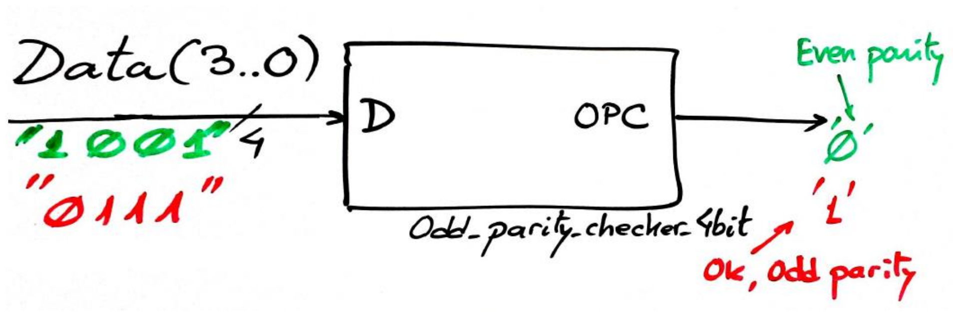

Fig. 7
The idea and circuit for checking the odd parity of a transmitted frame.

4. Invent the internal architecture of the *Data_reg_8bit* circuit using only components *Counter_mod16* represented in Fig. 6. (0.5p)

5. Invent a datapath for the *Receiver* in Fig. 3 using components like in Fig. 4 or others. Explain your circuit and justify your signals and connections. (1.5p)

6. Invent a state diagram for the *Receiver*'s control unit using states like in Fig. 5 or any other that you may find necessary. Justify your signals, state transitions and loops. The state diagram also includes the outputs (external outputs and control signals) at each state in parenthesis. (1p)

7. How many *D_FF* will include your *Receiver* design? Justify your answer. (0.5p)

Questions (chapter III)

We like to design the circuit in an alternative way using a µC PIC18F4520 as represented in Fig. 8 and C language. Hardware tools are ICD3 in-circuit debugger-programmer and Proteus-VSM virtual laboratory. Software tools are MPLABX integrated development environment and XC8 C compiler.

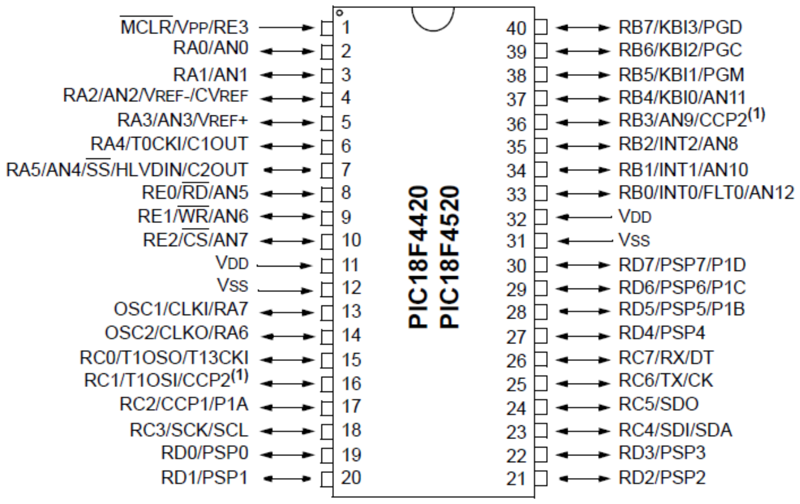

Fig. 8
PIC18F4520 40-Pin PDIP.

8. Draw the hardware schematic including the reset circuit MCLR_L and an external quartz crystal oscillator of **15.36 MHz**[1]. Explain how to configure the inputs and outputs in the *init_system().* Suppose that at this initial design phase, an external CLK signal of 4.8 kHz is used as the RX_CLK. (0.5p)
9. Draw the hardware/software diagram indicating the required RAM variables and how the FSM is solved in software. (1p)
10. Draw the truth table and its equivalent flowchart for the *state_logic()* function. (0.5p)
11. Draw the truth table and its equivalent flowchart for the *output_logic()* function. (05p)
12. If in the next design phase the external CLK is replaced by the internal peripheral Timer0 (represented in Fig. 9) in 8-bit mode, calculate **N1** (prescaler), **N2** (TMR0L) and the configuration bits required to generate a *var_CLK_RX_flag* interrupt every CLK_RX period (208.33 µs) to run the machine. (0.5p)
13. How to use the Timer0 to generate four (833.33 µs) and eight (1.66 ms) CLK_RX period delays? (0.5p)

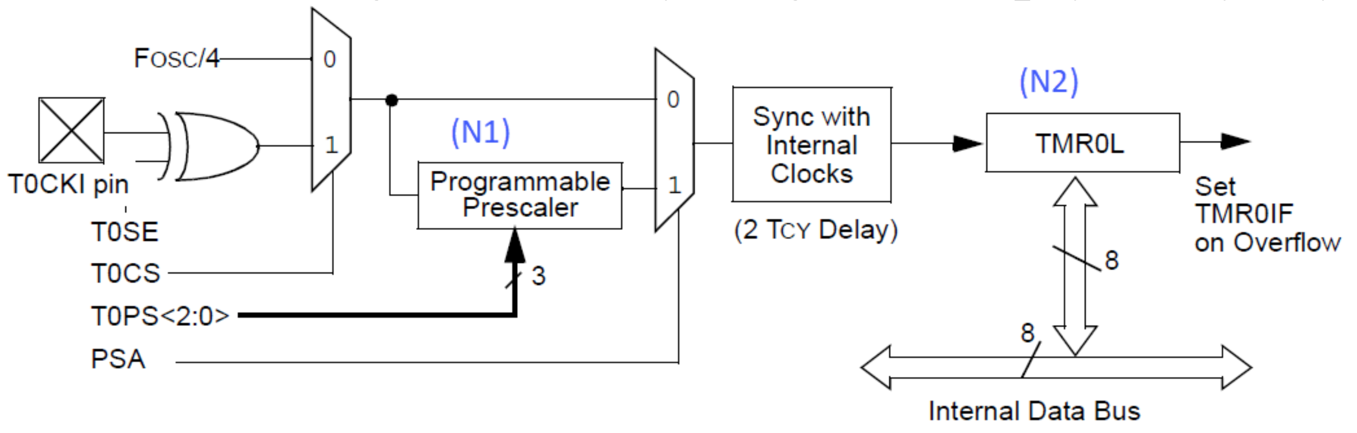

Fig. 9 Architecture of the Timer0 peripheral.

[1] https://en.wikipedia.org/wiki/Crystal_oscillator_frequencies