CEG 3155: Digital Systems II
(Fall 2023)
Prof. Rami Abielmona
Project: *UART Design*

November 3, 2023

Contents

1 Objective

The objective of this project is to design and build a complete UART in VHDL.

Upon completion, the student must be able to:

- Design, realize and test transmitter and receiver modules;
- Design, realize and test a baud rate generator;
- Demonstrate a complete understanding for the design of a UART and its interface in a real-time system.

2 Pre-Pro

Implement the loopback test described in figure 6, to guarantee that the electrical signals of the serial port are of the right voltage and logic levels. Study the technical data sheet of the MAX232 chip, and in particular figure 5 on page 16, then build, utilising your own breadboard and the capacitors provided to you for this project, the circuit which aids in converting the CMOS signals to RS-232 signals in one direction, and the RS-232 signals to CMOS signals in the other direction. A +5V CMOS signal and a logic '1' level is converted to a -12V RS-232 signal and logic level '0', while a 0V CMOS signal and logic level '0' is converted to a +12V RS-232 signal and logic level '1'. This helps the signals in traveling long electrical paths without losing signal strength. The loopback test will aid you in guaranteeing that the logic levels and voltages that leave and enter the Cyclone chip are of the correct values and will not destroy the FPGA. Any character pressed on the keyboard of the PC will echo back and display on the terminal of the PC.

3 UART Overview

A UART is a *Universal Asynchronous Receiver-Transmitter*, which is used to communicate between two devices. Most computers and microcontrollers include one or more serial data ports utilisez to communicate with other serial I/O devices, such as keyboards and serial printers. Serial ports are also used to communicate between two computers (figure 1) using a UART in each computer and a *crossover* cable, which connects the transmitter (TxD) of one UART to the receiver (RxD) of the other, and vice versa. A common ground (GND) wire connects both computers to a common negative voltage source.

A UART provides the means to send information using a minimum number of wires. The data is sent bit serially, without a clock signal. The main function of a UART is the conversion of parallel-to-serial when transmitting and serial-to-parallel when receiving. The fact that a clock signal is not sent with the data complicates the design of a UART. The two systems (transmitter and receiver) contain separate and unsynchronised local clocks. A part of the function of a

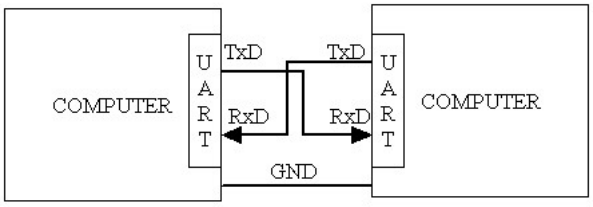

Figure 1: Communication between two computers

UART is to sample the incoming serial data at the right time to precisely capture the binary stream. This is accomplished by utilising a fast clock to sample the binary stream multiple times for each data bit. Thus, when transmitting, the UART receives the data in parallel from the application, and sends it serially on the TxD pin, and when receiving, the UART receives the data serially on the RxD pin, and provides the parallel data to the application.

4 Data Encoding

In order for the two systems to understand each other, we will need a commun data encoding. Since a clock does not exist, the data (D) is sent asynchronously, one byte at a time (figure 2). When no data is transmitted, D remains high. To signal the start of a new tranmission, D goes low for a bit period, which is known as the *start bit*. Then, eight data bits are sent, least significant bit (LSB) first, using ASCII code. The latter is a code which represents alphanumeric characters using 7 bits. The eighth bit can be used as the parity bit in order to better detect errors. In figure 2, the letter U, encoded as "1010101", is sent followed by a 0 parity bit, so that the number of 1s is even (even parity). After the transmission of the eight bits, D must climb back up to a logic '1' for at least one bit period, which is know as the *stop bit*. Hence, the transmission of another character can begin anytime after that. The number of bits transmitted per second is usually referred to as the *baud rate*.

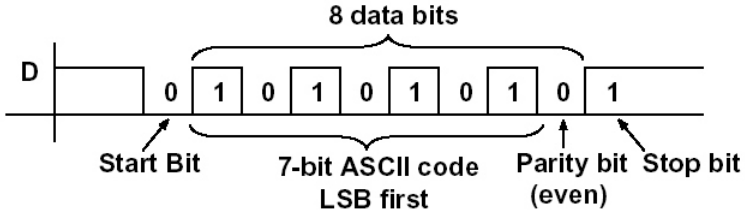

Figure 2: Standard data serial encoding

When sending, the UART takes the eight parallel data bits and converts them to a serial data bit stream which consists of one start bit, eight data bits

and one or more stop bits. When receiving, the UART detects the start bit, receives eight data bits, and converts the serial data into parallel data when it detects the stop bit. Since no clock is sent, the UART must synchronize the entering binary stream with its local clock.

5 UART Components

A UART is composed of four main components: the receiver, the transmitter, the baud rate generator and the UART registers. We suppose that the UART is connected to a microcontroller by a data bus and an address bus, to allow the CPU to read and write the register of the UART. Refer to figure 3 for the discussion of the following components.

do we even need receiver in this case, and what about debuging it

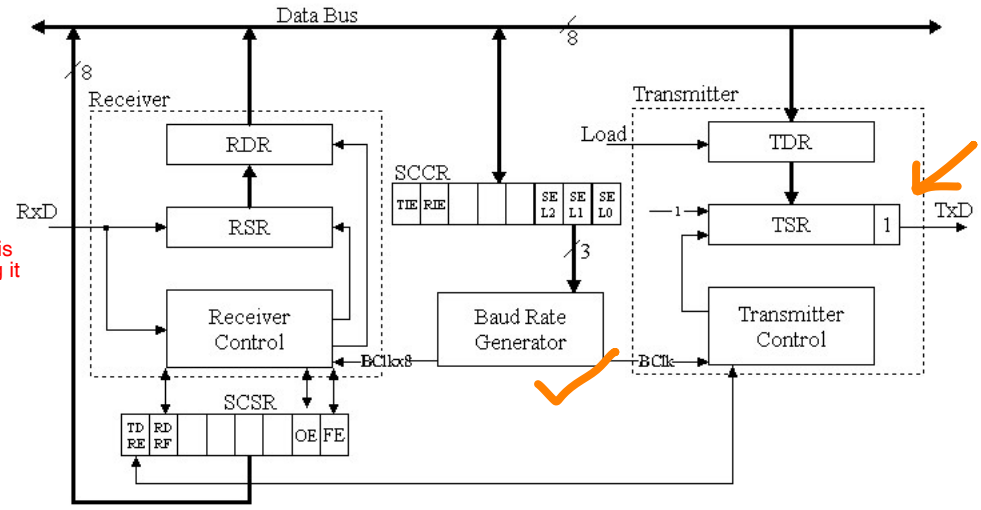

Figure 3: The UART components

5.1 UART Registers

There usually exists four registers for each UART: the receive data register (RDR), the transmit data register (TDR), the serial communications control register (SCCR), and the serial communications status register (SCSR).

RDR This 8-bit register receives the data from a 8-bit shift register (RSR) and contains the serially received byte;

TDR This 8-bit register transmits the data from the application to a 8-bit shift register (TSR) and contains the byte that has to be serially transmitted;

SCCR This 8-bit register contains the UART control signals: TIE, RIE and SEL[2:0]; and

SCSR This 8-bit register contains the UART status signals: TDRE, RDRF, OE and FE.

The TSR (Transmit Shift Register) receives the output of the TDR, and the RSR (Receive Shift Register) provides the inputs to RDR. The status and control signals are defined as shown in table 1.

some of signal never been used

Signal	*S/C*	*Description*
TDRE	Status	Transmit Data Register Empty
RDRF	Status	Receive Data Register Full
OE	Status	Overrun Error
FE	Status	Framing Error
TIE	Control	Transmit Interrupt Enable
RIE	Control	Receive Interrupt Enable
SEL[2:0]	Control	Baud Rate Selector

Table 1: UART status and control signals

5.2 Transmitter

The UART transmitter consists of register TDR and TSR and the transmitter control. The status bit $TDRE$ in the $SCSR$ is asserted by the controller when TDR is empty. When the microcontroller is ready to send data, the following occurs:

1. The microcontroller waits for $TDRE =' 1'$ then registers the byte into TDR. $TDRE$ is then reset back to 0.
2. The UART transfers the byte from TDR to TSR and asserts $TDRE =' 1'$.
3. The UART outputs a start bit for one bit period, and then shifts TSR right to transmit the eight data bits followed by a stop bit.

loop

5.3 Receiver

The UART receiver consists of registers RDR and RSR and the receiver controller. The status bit $RDRF$ in the $SCSR$ is asserted by the controller when RDR is full. When the UART detects a start bit, the following occurs:

1. The UART reads and shift in the 8 data bits serially into RSR.
2. When the eight data bits and the start bit have been received, the value in RSR is registered in RDR, and the status signal $RDRF$ is asserted.

3. The microcontroller verifes the status signal, and if asserted, reads RDR is read and $RDRF$ is reset back to 0.

The binary stream entering the RxD pin is not synchronized with the local clock ($BClk$). If we tried reading RxD at each positive edge of $BClk$, we would have problems if RxD changed values close to the edge. We can have additional problems if the speed of the entering binary bit stream differed from $BClk$, even by a small amount, since we would read certain bits at the wrong time. To avoid these potential issues, we will sample RxD eight times during each bit period (certain UARTs sample 16 times each bit). We will sample at the positivie edges of $BClkx8$, indicated by the arrows in figure 4. Ideally, we should read the bit value at the middle of each bit period for maximum robustness. Therefore, when RxD drops low, we wait four periods of $BClkx8$, so that we are near the middle of the start bit. Then, we continue reading each eight periods of $BClkx8$ to register each data bit, until we have detected and read the stop bit.

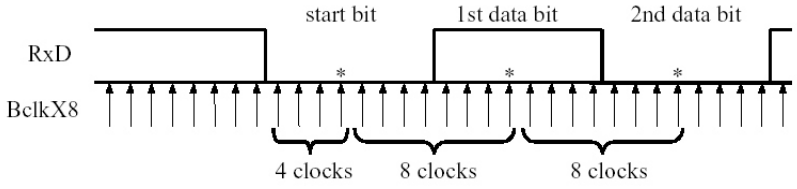

Figure 4: The sampling of RxD with $BClkx8$

5.4 Baud Rate Generator

The baud rate generator is programmable by utilizing the three control bits ($SEL[2:0]$) in $SCCR$. Since we are using three bits, we have the choice of 8 baud rates. Assume the system clock is 25.175 MHz, and that we require the baud rates of 300, 600, 1200, 2400, 4800, 9600, 19200 and 38400. The required maximum frequency of $BClkx8$ is $38400x8 = 307200$. To obtain this frequency, we divide 25.175 MHz by 81.95. Since we can only divide by integeres, we have to accept a small error in the baud rate or adjust the system clock frequency to 25.1904 MHz. We can see in figure 5 that the first divisor divides the clock by 41, then sends that clock to a 8-bit binary counter. Each flip-flop of the latter corresponds to a division by 2, hence the first corresponds to a division by 2, the second corresponds to a division by 4, until the eighth and last which corresponds to a division by 256. One of these eight clocks is chosen by using the multiplexer and the control signals $SEL[2:0]$.

Assuming that the system clock is 25.175 MHz, we obtain the frequencies shown in table 2.

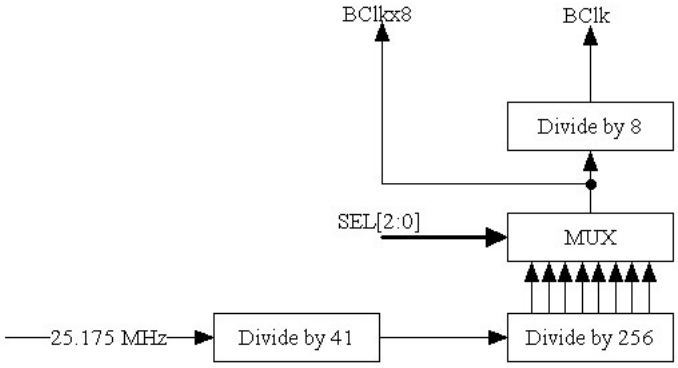

Figure 5: The programmable baud rate generator

SEL[2:0]	*Baud rate (BClk)*
000	38377
001	19188
010	9594
011	4797
100	2398
101	1199
110	600
111	300

Table 2: Frequencies of $BClk$

5.5 Interrupt Generator

The generation of the signal IRQ which interrupts the CPU when the UART receiver or transmitter require the CPU's attention. When RIE (receive interrupt enable) is set to 1 in $SCCR$, an IRQ is generated when $RDRE$ or OE is '1'. Meanwhile, if TIE (transmit interrupt enable) is asserted in $SCCR$, an IRQ is generated when $TDRE$ is '1'.

5.6 Address Decoder

The UART is connected to a computer or microcontroller so that we can read and write characters from/to the outside world. We will need an address decoder to choose which actions we would like to execute, and thus which registers we would like to read or write to. Firstly, we need an enable for the UART which we name $UART_Select$. When the latter is '1', we will be using the UART, otherwise, we have no need to access the UART. Secondly, we need an address decoding table, as shown in table 3.

seems unesssary? like the fsm only connect to the uart?

ADDR[1:0]	$R/\overline{W}$	*Action*
00	1	DATA_BUS ← RDR
00	0	TDR ← DATA_BUS
01	1	DATA_BUS ← SCSR
01	0	DATA_BUS ← Z
1d	1	DATA_BUS ← SCCR
1d	0	SCCR ← DATA_BUS

Table 3: Address Decoding Table

6 Project

In this project, you will, using the UART theory presented in the previous sections, proceed to implement a **UART**, by realizing the UART components (section 5), and by realizing the UART control circuits and finally by connecting up the lot to real sensors and actuators. From the given characteristics for each component, your goal is to design, build and test the sequential synchronous circuits,using the Quartus II software and the DE-2 Altera board (for more details, refer to section 7) which together make up a UART that will be incorporated with the traffic light controller, the latter realized in a previous laboratory, to produce a real-time debuggable system!

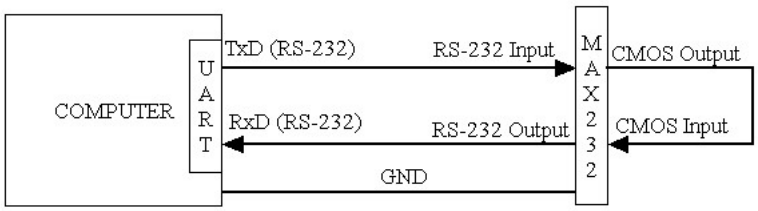

Figure 6: Loopback test of the pre-project

6.1 Problem Specifications

Your team has been hired to design a traffic light controller, which demonstrates the same characteristics as the one previously designed, except that the controller will produce debug messages. The messages correspond to a specific state in the finite state machine of the traffic light controller. We assume the presence of four main states: (1) main street green/side street red, (2) main street yellow/side street red, (3) main street red/side street green, and (4) main street red/side street yellow. When in the first state, the debug message must be "Mg_Sr", and when in the second state, the debug message must be "My_Sr", and when in the third state, the debug message must be "Mr_Sg", and finally

when in the fourth state, the debug message must be "Mr_Sy". All debug messages have to end with a carriage return to allow the computer to return to the start of the next line. Each debug message contains 6 bytes, including the carriage return. Typically, the debug messages have to be transmitted starting from the Cyclone's UART, to the computer's UART and finishing by posting the characters to the computer's screen, by using a serial port program (e.g. HyperTerminal or Minicom). Your task is to develop a traffic light controller which posts debug messages to a computer's screen. The input/output specifications of this traffic light controller are shown in table 4.

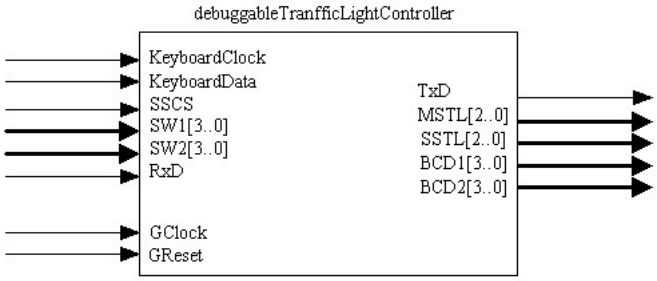

Figure 7: Debuggable traffic light controller

Type	*Nom*	*Description*
Input	GClock	Global clock needed to synchronize the circuitry
Input	GReset	Global reset needed to bring the internals to known states
Input	SSCS	Side street's car sensor input
Input	SW1[3..0]	Main street 4-bit DIP switch input
Input	SW2[3..0]	Side street 4-bit DIP switch input
Input	RxD	Serial port reception input
Output	TxD	Serial port transmission output
Output	MSTL[2..0]	Main street's traffic light output using one-hot encoding
Output	SSTL[2..0]	Side street's traffic light output using one-hot encoding
Output	BCD1[3..0]	BCD 4-bit output for left digit of current counter value
Output	BCD2[3..0]	BCD 4-bit output for right digit of current counter value

Table 4: Input/output specifications

6.2 Project Hints

This section is meant to present a potential solution to the problem at hand, while discussing the major parts of the solution.

Presented in figure 8 is the project network diagram. Included in the figure are the main system inputs and outputs.

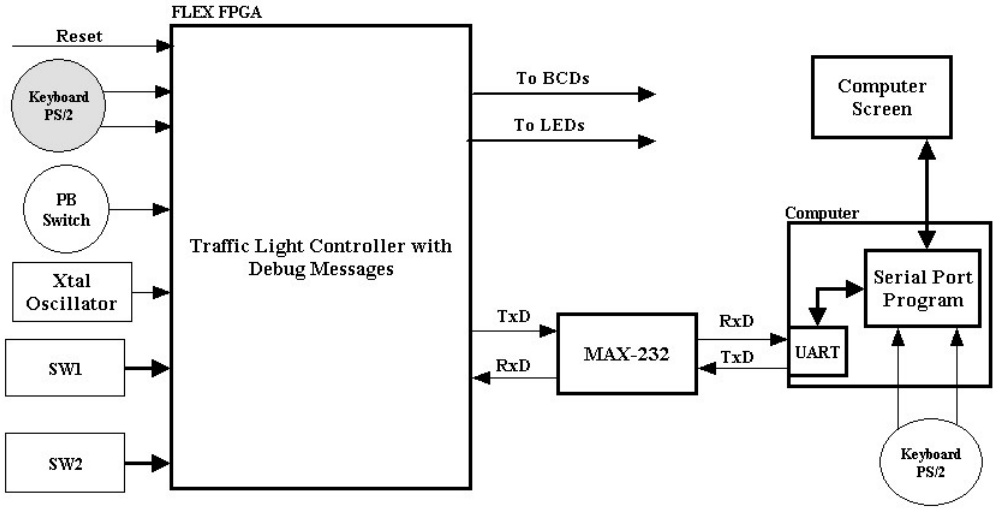

Figure 8: Project network diagram

The figure 9 shows a potential solution to the project including the blocs from the previous laboratory #3. Clearly note the simultaneous use of four different FSMs: (1) traffic light controller lab FSM, (2) UART FSM, (3) emitter FSM and (4) receiver FSM. The UART FSM is unique as it simulates the presence of a microcontroller which normally interfaces to the UART. The latter accepts state information (2 bits to represent 4 states) and produces the signals to send or receive bytes at the UART. Do not forget the *Select* and IRQ signals. The former allows access to the UART, while the latter requires the attention of the UART FSM, which occurs when the UART has sent or received a character. As well, let's not forget the address decoder (section 5.6) which is present in the UART module and decides which register writes to the data bus (write cycle) and which register reads from the data bus (read cycle). Finally, the UART module itself was previously discussed and shown in figure 3, therefore refer to that figure for you final implementation.

Finally, shown in figure 10 is the diagram for the previous laboratory, however modified for this project. The diagram is only shown to review the potential solution to the problem, and to view it as incorporated in figure 9.

6.2.1 Breakdown

We can break down the system diagram (figure 9) in five major components:

Lab #3 This component is the traffic light controller already built in laboratory #3. The internal FSM must be re-designed in order to output the

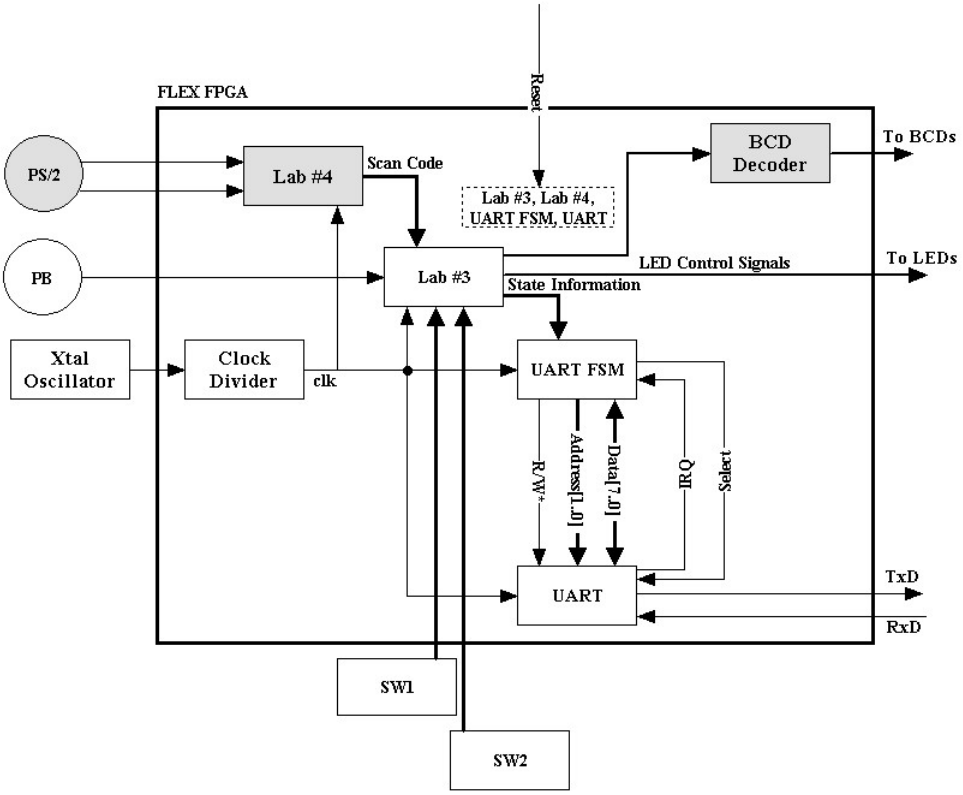

Figure 9: A potential system diagram

state information;

UART FSM This component simulates a microcontroller and receives the state information and interfaces to the UART in order to send debug messages at each state change. This component also receives the characters from the UART and decides to whether to invert the debug messages or not;

UART This component is the set of the UART register, the transmitter, the receiver and the baud rater generator, which is used to send or receive ASCII characters on the TxD and RxD wires;

Clock Divider This component divides the global clock into a slower one to control the different components. The goal is to provide the debug messages in human-readable time; and

BCD Decoders These components are used to visually show the current value of the counter on BCD 7-segment decoders.

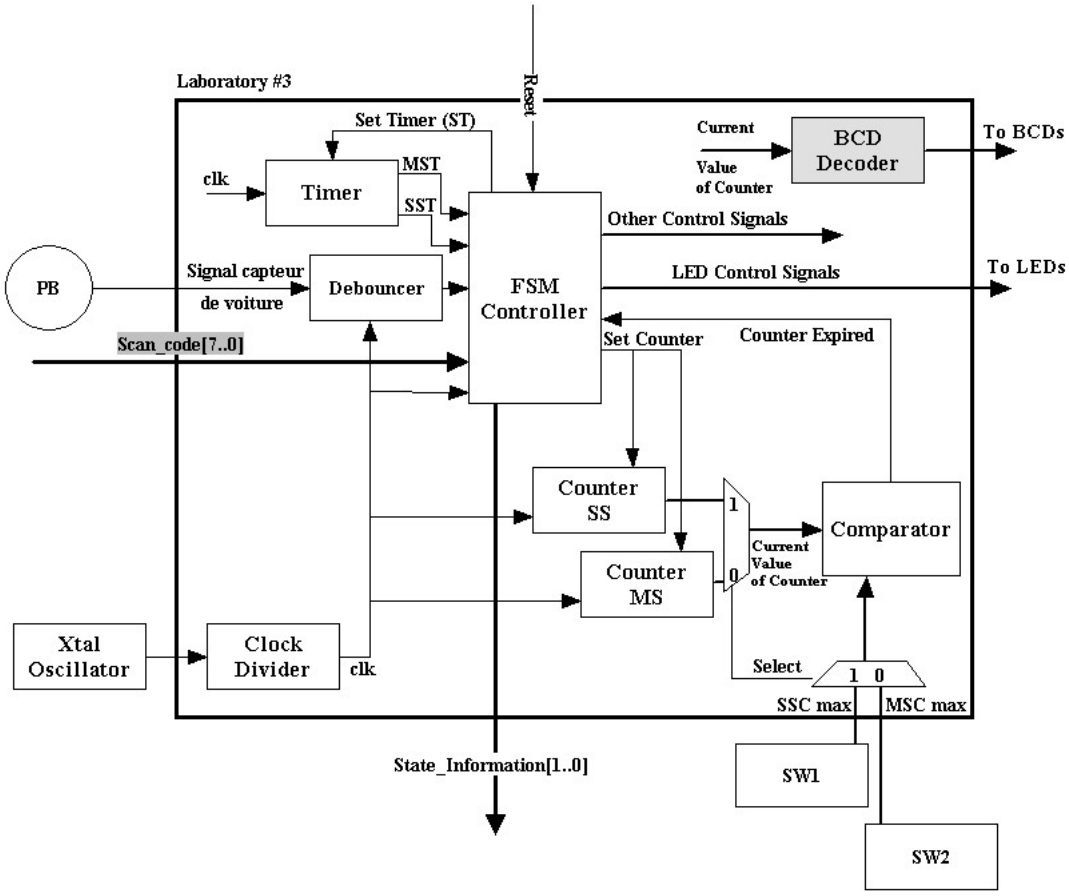

Figure 10: A system diagram for the traffic light controller modified for this project

6.2.2 System-Wide Hints

It is worthy to note that a **global reset** is provided to the FSM controller, which effectively resets all the state registers within the FSM, as well as any outputs of the FSM (going out of the system or to other building blocks within the system). This will cause the other blocks to enter their reset condition, and thus for example, cause the timer circuit to reset back to its initial count value or the shift register to reset back to 0. This *reset distribution* is not shown in the figures, but it is assumed that the designers will take care of this network.

~~**If you cannot implement the *interrupt-mode* of the UART by using *TIE* and *RIE* in *SCCR*, you can realize its counterpart solution: *polling-mode*. Hence, you can read the status register until a bit**~~

~~is asserted (either TDRE or RDRF), then read/write from/to the appropriate register (either RDR or TDR).~~

6.2.3 Bonus: BCD Decoders

For a **bonus of 3 marks**, include the **BCD decoder** to implement the posting of the current values for the counter on the BCDs in decimal format, as described in laboratory #3. You will have to demonsrate your new design to the TA.

7 Design Restrictions

- Verilog implementations will not be accepted. Perform all implementations in VHDL code only
- Behavioral level of modeling will not be accepted. Design should be done at the structural level of modeling
- Register Transfer Logic (RTL) design and coding will is mandatory
- Use graphical design for the top-level entity, and use your judgement for any other sub-blocks. However, all atomic modules have to be implemented in VHDL (i.e. D flip-flop, 1-bit adder, 1-bit comparator and so on)
- No core instantiations are allowed (i.e. LPMs from Altera or free IP cores from the Internet). All remaining building blocks have to be designed and realized by the group
- The group may use any core provided in the course in the design (i.e. debouncer, clock divider, register, shift register, adder, counter, comparator, etc...). Remember to reference all cores that the group itself did not develop
- If using the clock divider core in particular, its behavioral model is allowed
- The group can choose Mealy or Moore machine implemenations as the controller of any FSM
- Designers can re-use modules that were designed in previous laboratories
- The top level entity is given in input/output specification format, but the internals are left up to the group. A sample schematic solution for the debuggable traffic light controller has been given, but the group is free to design the system using another methodology
- The design has to be synchronous and globally reset-able. This means that global clock and reset signals are required in all functional blocks
- Simulate all designs and check your simulation results with your theoretical ones (e.g. press the push-button to simulate a car on the side street)

- Download the design to the Cyclone chip on your DE-2 boards, and use the on-board DIP switches to input SW1[3..0] and SW2[3..0], a push-button switch to input SSCS (the car sensor), and two sets of three LEDs to demonstrate correct output functionality of the traffic lights shown in figure 7. Finally, the signal TxD leaves from a hole of your prototyping header and the signal RxD enters from another hold of the same header, while the common GND is taken from hole #10 in the JTAG port of the DE-2 board and is wired to the GND of the computer's serial port (provided by the 3-wire DB-9 header). If implementing the bonus section, the two BCD decoders must be used to demonstrate the correct functionality of the counter (BCD1[3..0] and BCD2[3..0])
- Each group must demonstrate a working simulation of the project to the TA, before the due time of the report

8 Report Reminders

- Include timing simulations with explanation for all VHDL source files
- Describe and comment all your VHDL source files
- Include a flowchart representation of your solution to the problem
- Include a block diagram of your solution to the problem
- If using ASM design, include all appropriate charts and paths (control and data)
- If using FSM design, include all appropriate diagrams, tables and circuits
- Describe, in your own words, your solution to the problem
- Describe your design obstacles and how they were overcome
- Append all board design files (*.bdf) to your report
- Submit a soft copy of all VHDL and graphical design files with your report

9 Acknowledgements

Figures 2, 3, 4 and 5 and the description of the UART operation are extracted out of the textbook: *Digital Systems Design using VHDL* by Charles H. Roth, Jr.