

Lab 4: UART Communications



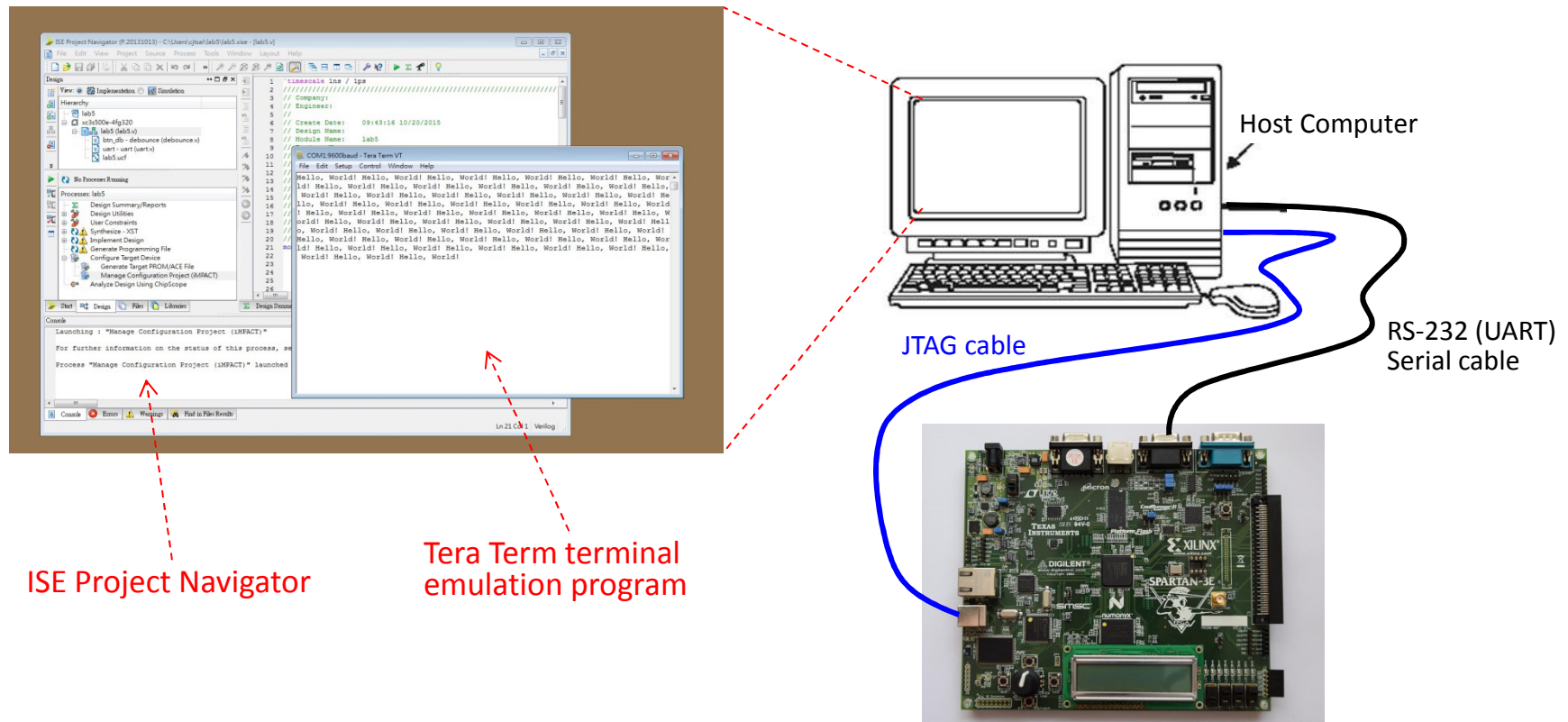
National Chiao Tung University
Chun-Jen Tsai
10/7/2016

Lab 4: UART Communications

- ❑ In this lab, you will design a circuit to perform UART I/O. Your circuit will do the following things:
 - Read a decimal number input from the UART port connected to a PC terminal window. The number ranges from 0 to 65535.
 - Convert the received number to a hexadecimal number, and print the number on the terminal window
- ❑ You will demo the design to your TA during the lab hours on 10/25

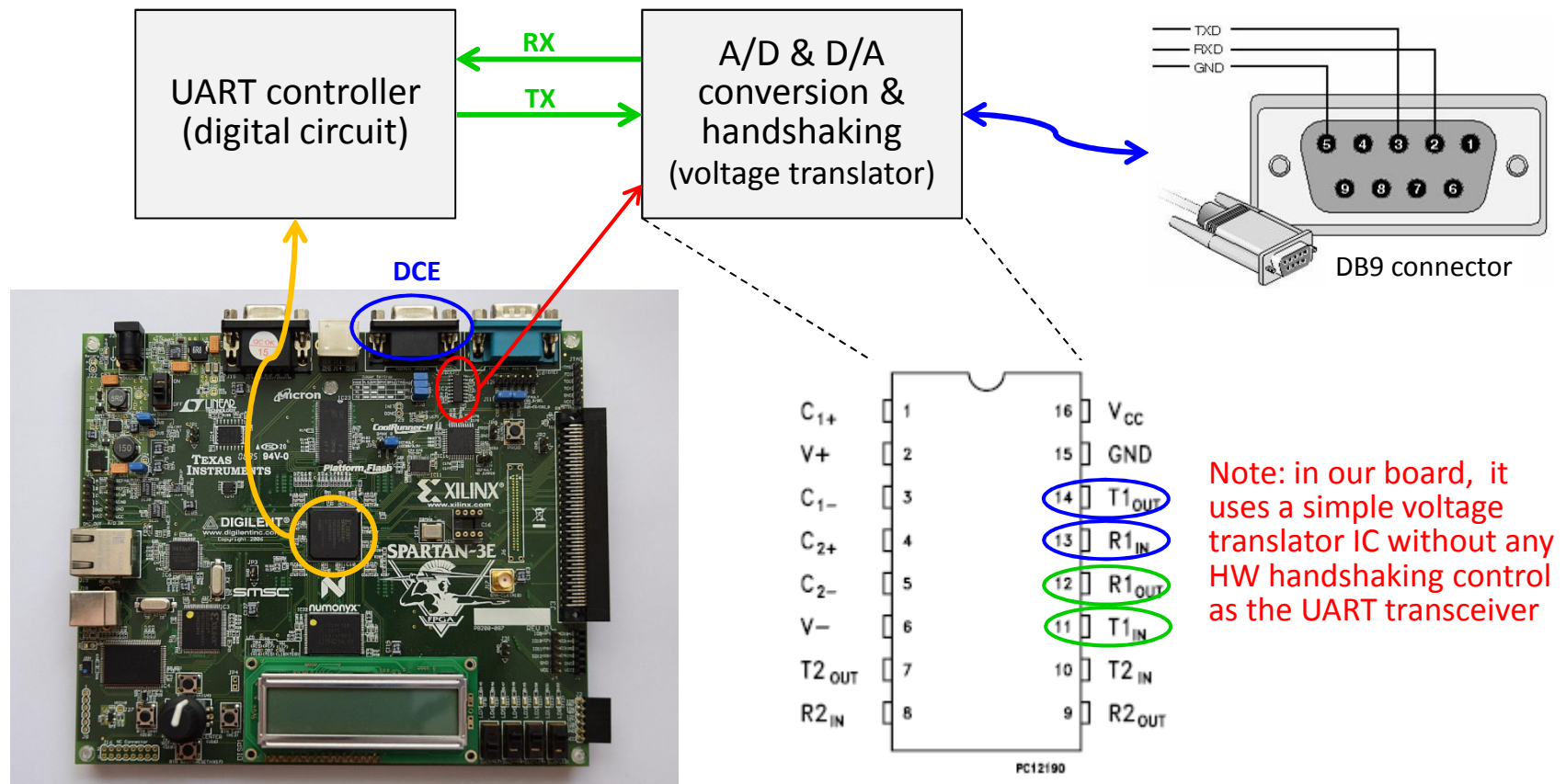
Setup of Lab4

- ❑ Lab 4 tests the communications between the PC and the FPGA through the UART devices (i.e., RS-232):



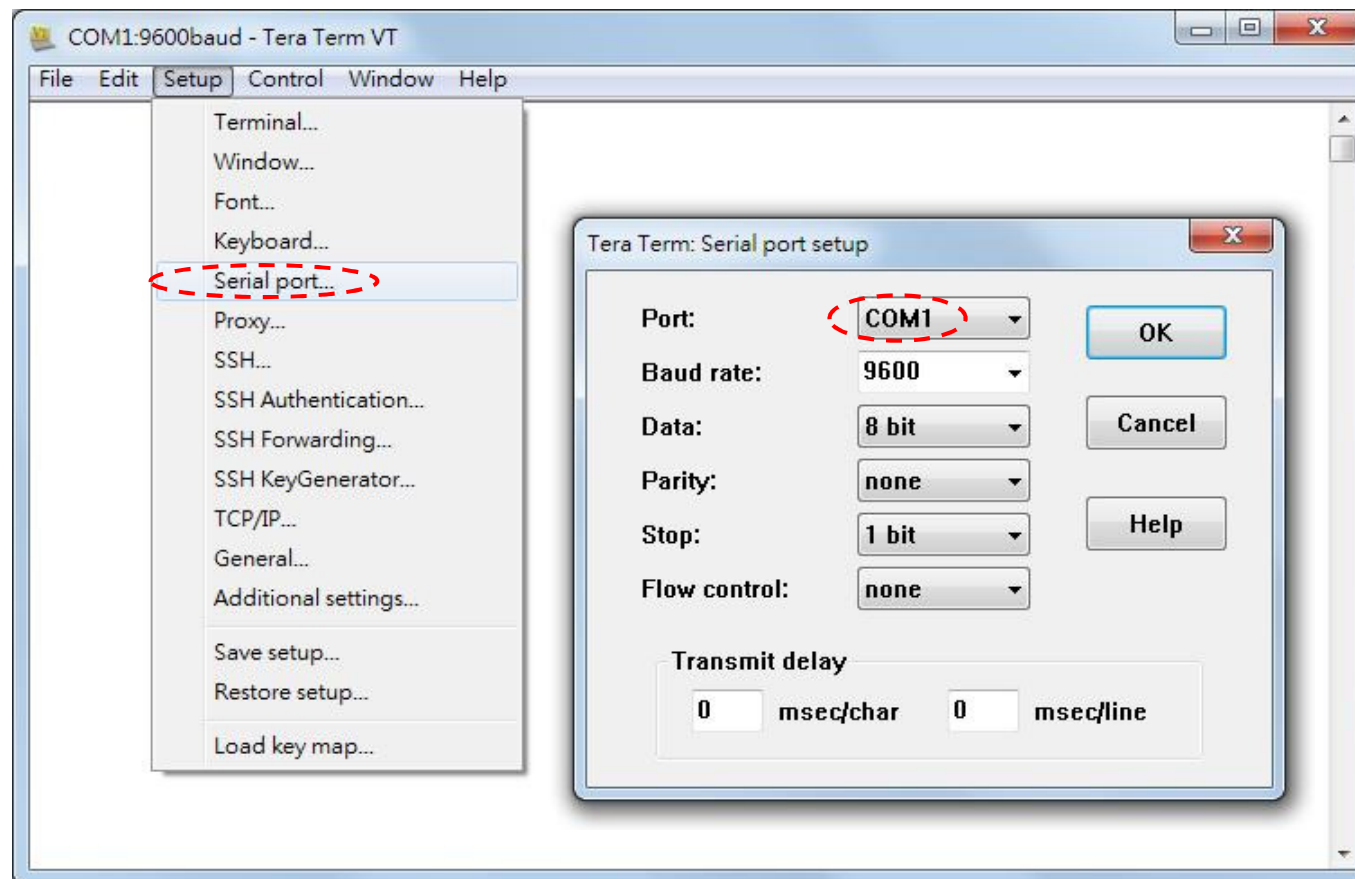
UART Communication

- ❑ Universal Asynchronous Receiver/Transmitter (UART) is one of the most popular I/O devices in small systems



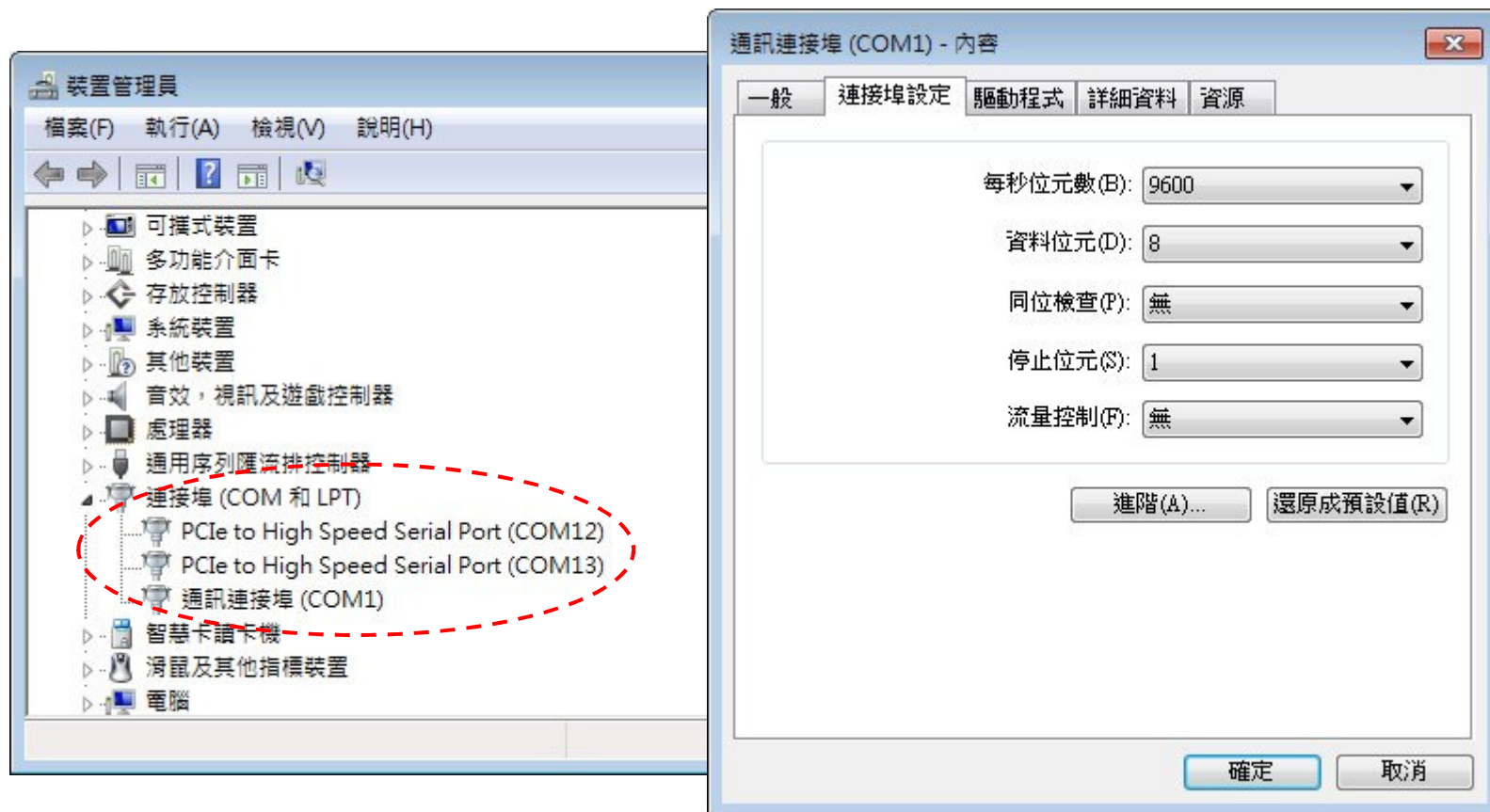
Tera Term on the PC Side

- ❑ On the PC connected to the Spartan-3E board, we run TeraTerm to send/receive data through RS-232:



Check COM Port Number

- ❑ The COM port number of your computer can be obtained from the device manager as follows:



UART Physical Layer

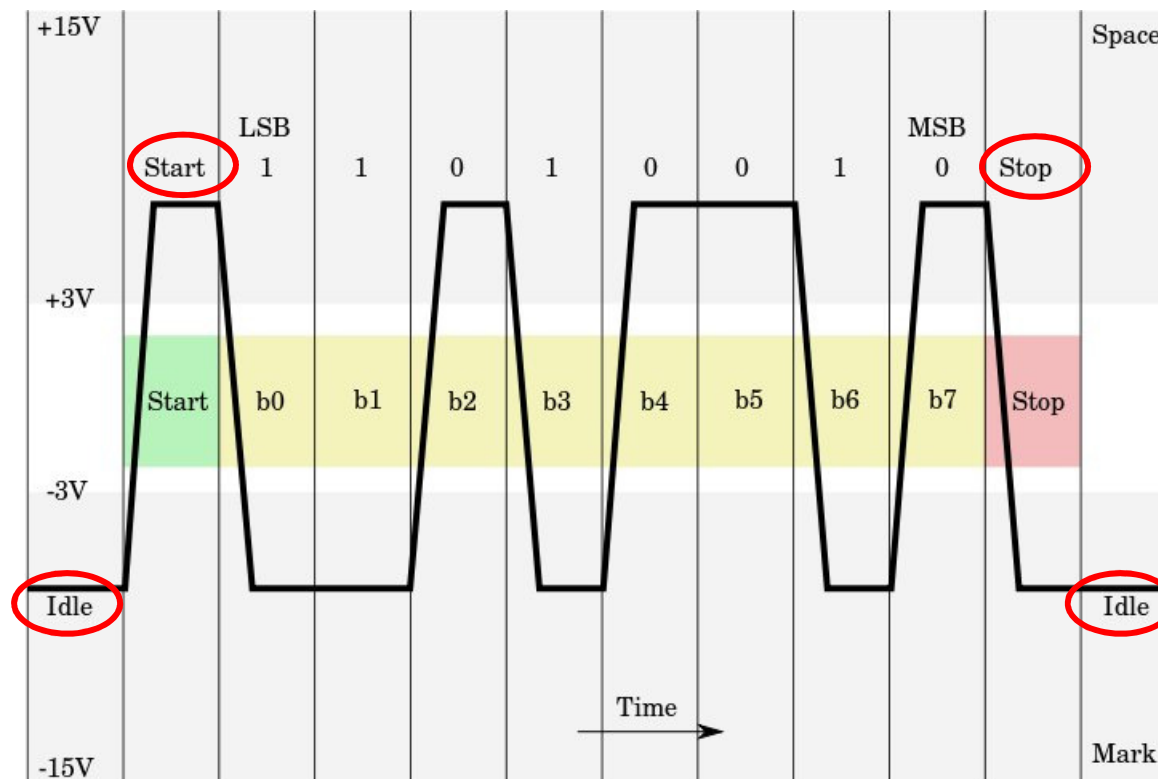
- ❑ UART is a **asynchronous** transmission standard, thus, there is no common clock signal for synchronization
- ❑ The most popular physical layer for the UART transmission line is the RS-232 standard
 - Common baud rates for RS-232 signals range from 4800 bps to 115200 bps
 - RS-232 voltages are $(-15V, -3V)$ for '1' and $(3V, 15V)$ for '0'

UART Link Layer

- ❑ The serial line is 1 when it is idle
- ❑ The transmission starts with a start bit, which is 0, followed by data bits and an optional parity bit, and ends with stop bits, which are 1
- ❑ The number of data bits can be 6, 7, or 8
- ❑ The optional parity bit is used for error detection
 - For odd parity, it is set to 0 when the data bits have an odd number of 1's
 - For even parity, it is set to 0 when the data bits have an even number of 1's
- ❑ The number of stop bits can be 1, 1.5, or 2

RS-232 Transmission Example

- ❑ An example of the RS-232 transmission signals:



Out-of-Band Parameter Setting

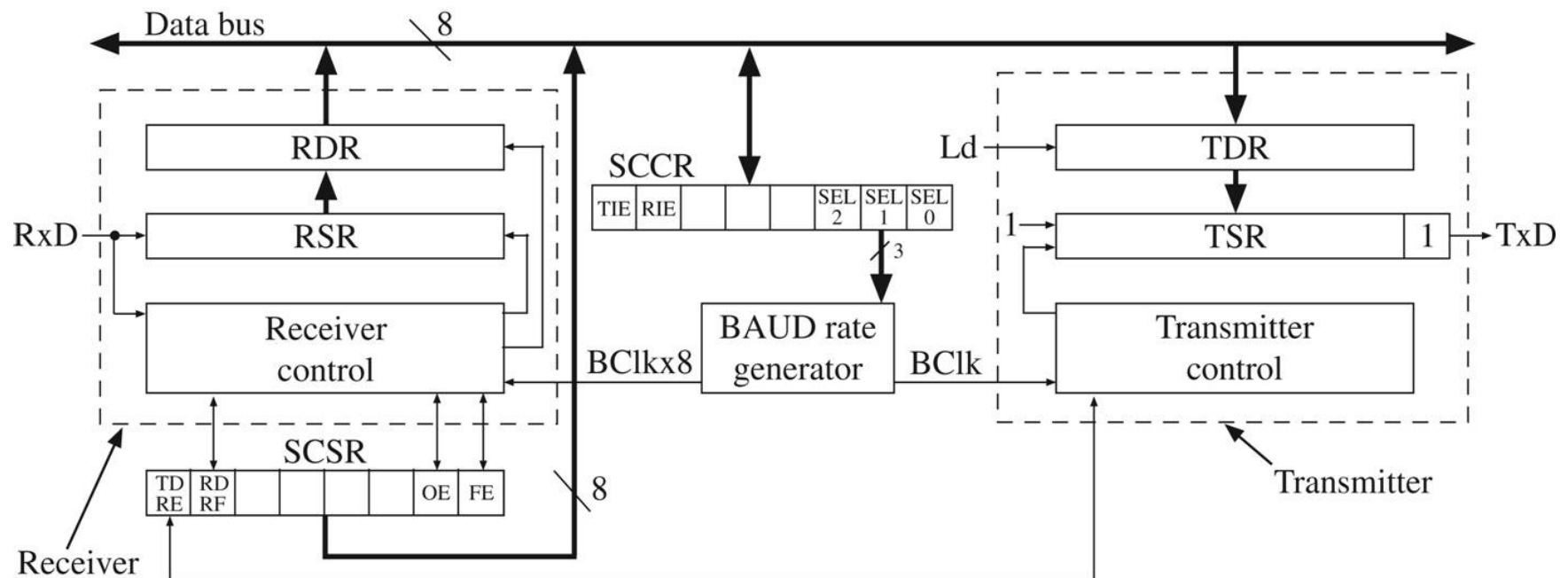
- ❑ UART control parameters such as: bit-rate, #data bits, #stop bits, and types of parity-check must be set on both side of the serial transmission line before the communication begins
- ❑ Implicit clocks must be generated on both sides for correct transmission
 - Bit rate per second (bps), or baud rate, is used to imply the clock on both end of the transmission line
 - Baud rates must be two's multiples of 1200, e.g., 2400, 4800, 9600, ..., 115200, etc.
 - This clock is often called the baud rate generator

UART Controller

- ❑ A UART controller performs the following tasks
 - Convert 8-bit parallel data to a serial bit stream and vice versa
 - Insert (or remove) start bit, parity bit, and stop bit for every 8 bits of data
 - Maintain a local clock for data transmission at correct rate
- ❑ A UART controller includes a transmitter, a receiver, and a baud rate generator
 - The transmitter is essentially a special shift register that loads data in parallel and then shifts it out bit by bit at a specific rate
 - The receiver, on the other hand, shifts in data bit by bit and then reassembles the data

An Example of UART Controller

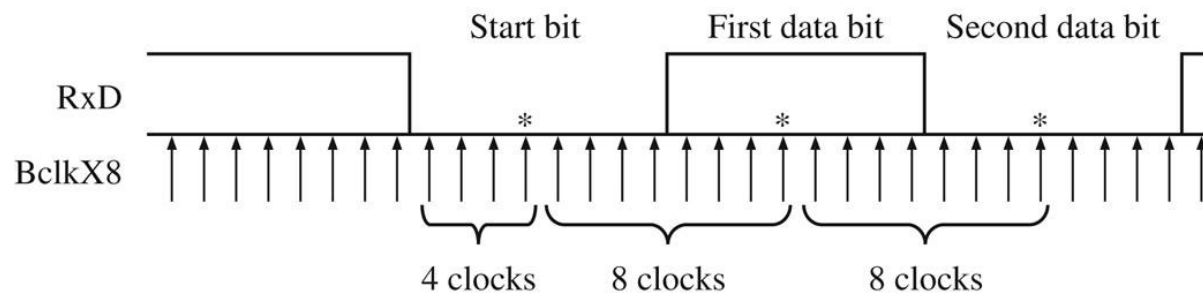
- The following diagram shows a typical UART controller:



RSR – Receive shift register
TSR – Transmit shift register
RDR – Receive data register
TDR – Transmit data register
SCCR – Serial communications control register
SCSR – Serial communications status register

Clock Synchronization Problem

- ❑ Since there is no explicit clock signal between the transmitter and the receiver, the receiver can not simply read incoming bits based on its system clock
- ❑ To solve this problem, we sample the incoming data multiple times per baud rate clock cycle
 - Typical up-sampling rates are 8- or 16-time sampling
- ❑ Take 8× sampling for example, the fourth sample of each bit time will be read as a data bit

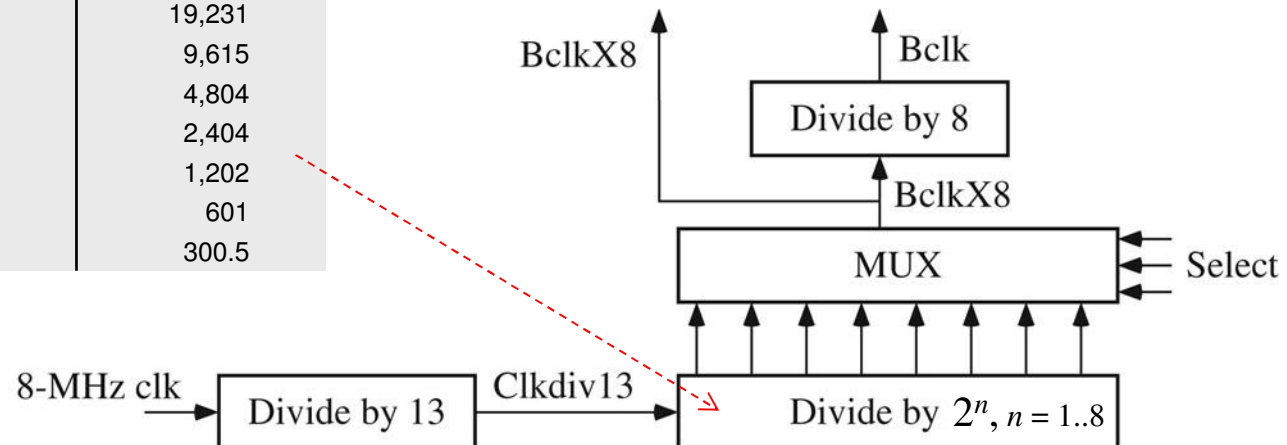


*Read data at these points

Baud Rate Generator

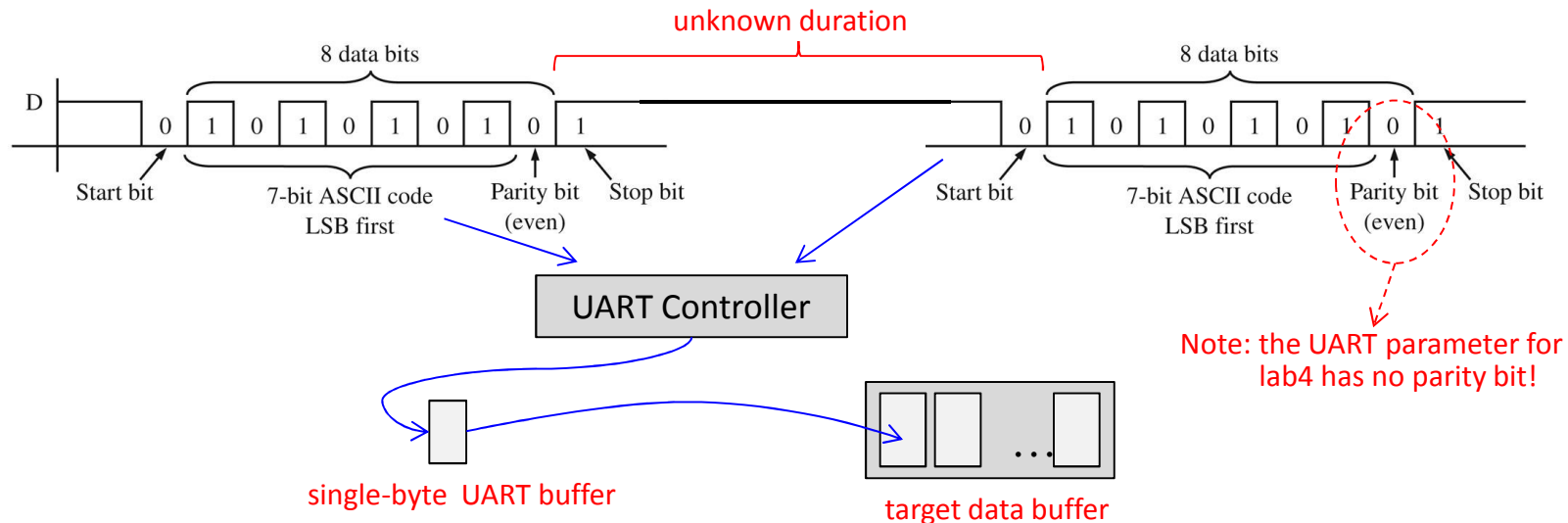
- ❑ We must generate a baud rate clock ($bclk$) and a 8- or 16-time baud rate clock from the system clock, clk
 - If highest baud rate = 38400, $38400 \times 8 = 307200$
 - If the system clock is 8MHz, the divider is $8M / 307200 = 26.04$
 - If a divider of 26 is used, the actual baud rate is as follows

Select Bits of clock divider	Baud Rate (Bclk)
000	38,462
001	19,231
010	9,615
011	4,804
100	2,404
101	1,202
110	601
111	300.5



Notes on Large Data for Receiver

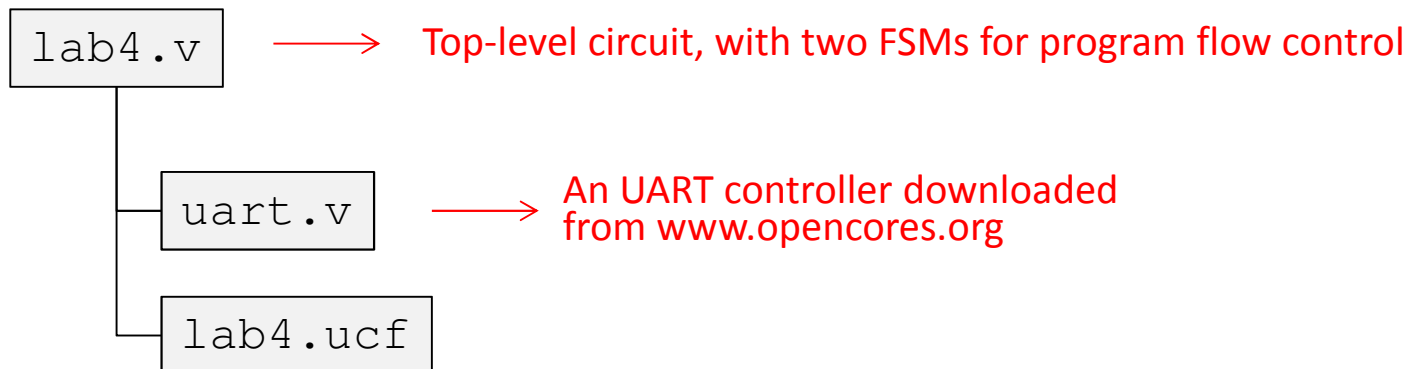
- ❑ FPGA runs at 50MHz, but data arrives **at** 9600 bps
 - The “received” notification from the UART controller only lasts for one system clock cycle (20 nanoseconds)



- ❑ For this lab, we don't have to worry about this problem because keystrokes are slow

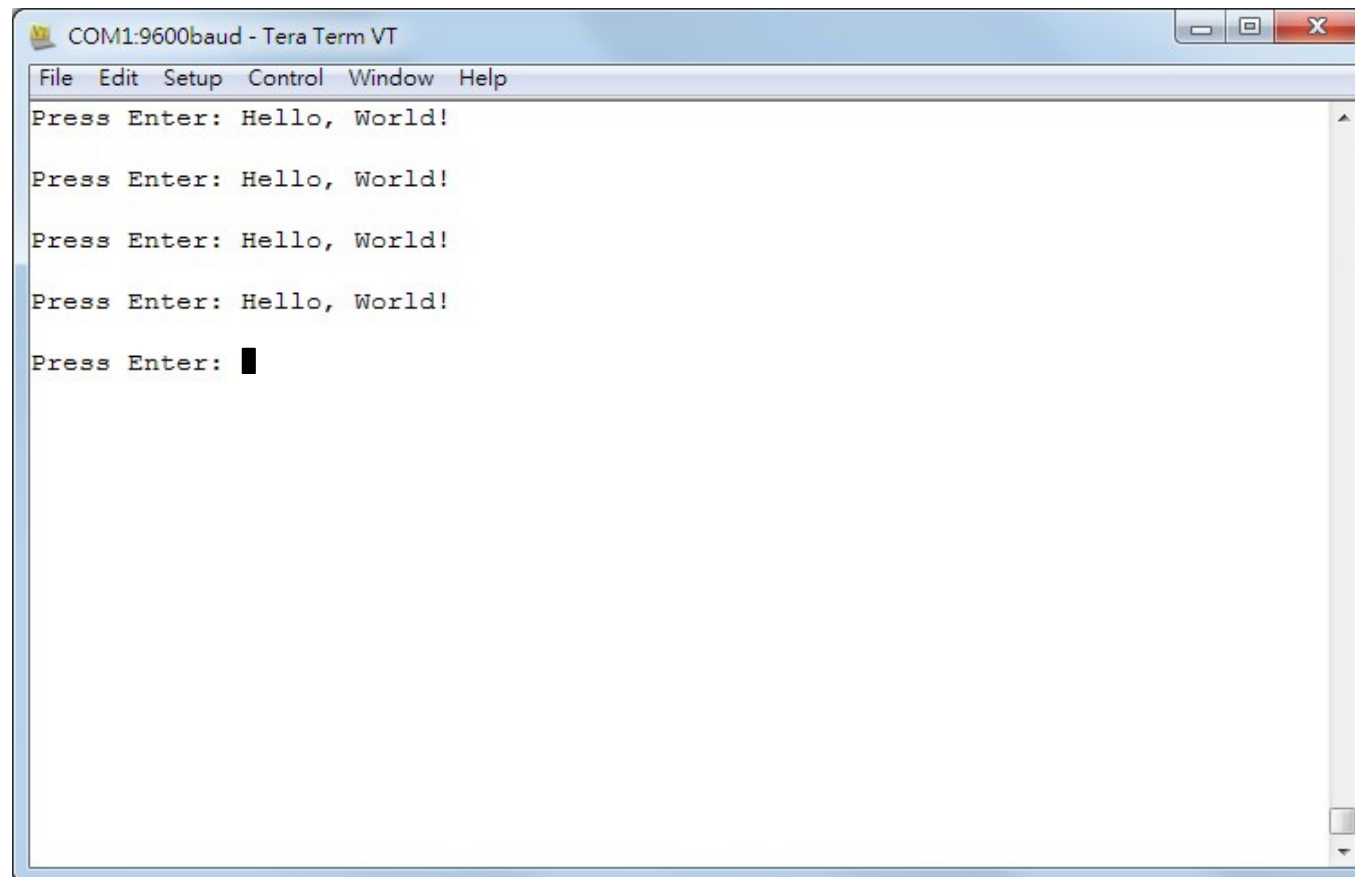
About the Lab4 Sample Package

- ❑ The package contains a project that shows you how to use a circuit to read the user keyboard inputs and then print “Hello, World!” through the UART controller
- ❑ The source tree structure is as follows:



Screen Shot of Lab 4 Sample Code

- ❑ The TeraTerm window prints “Hello, World!” every time an “Enter” key is pressed:

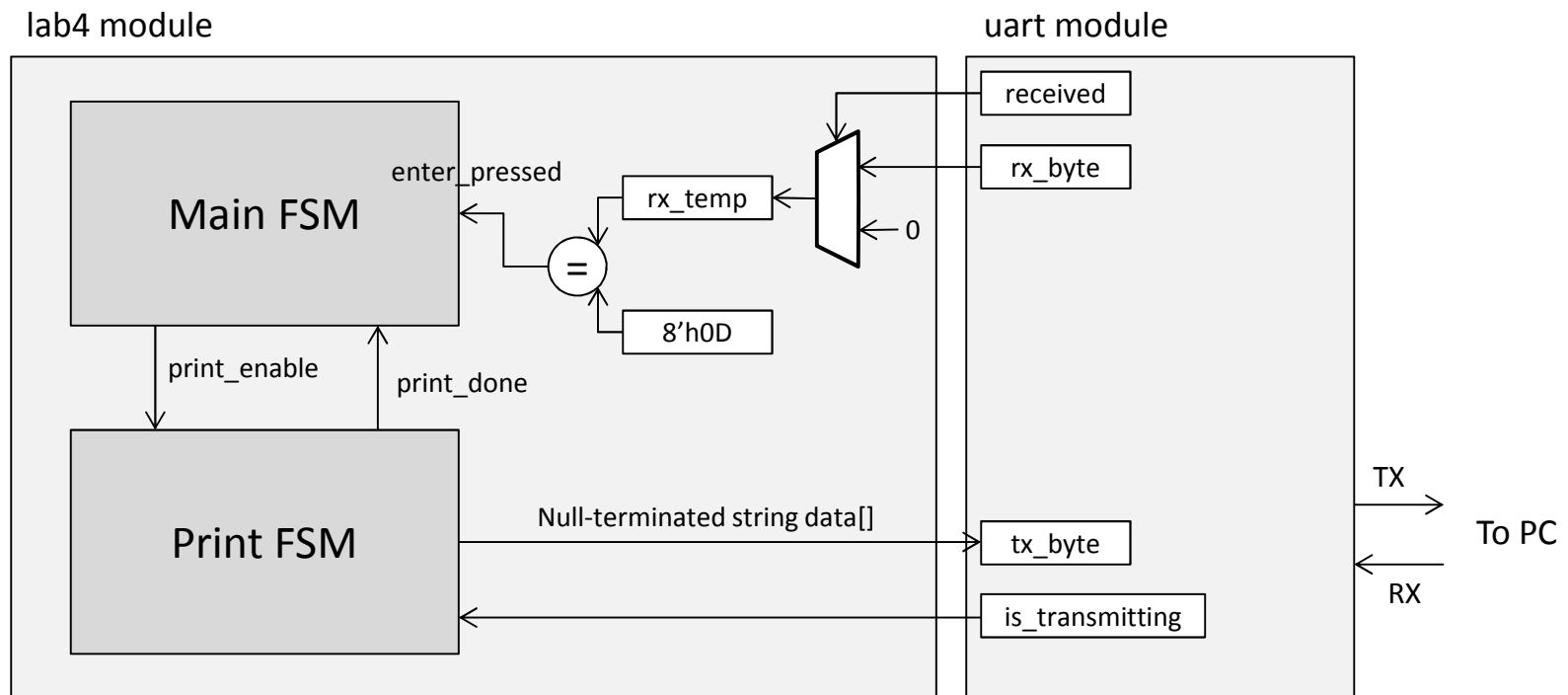


The screenshot shows a TeraTerm window titled "COM1:9600baud - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main text area displays the following output:

```
Press Enter: Hello, World!  
Press Enter: Hello, World!  
Press Enter: Hello, World!  
Press Enter: Hello, World!  
Press Enter: █
```

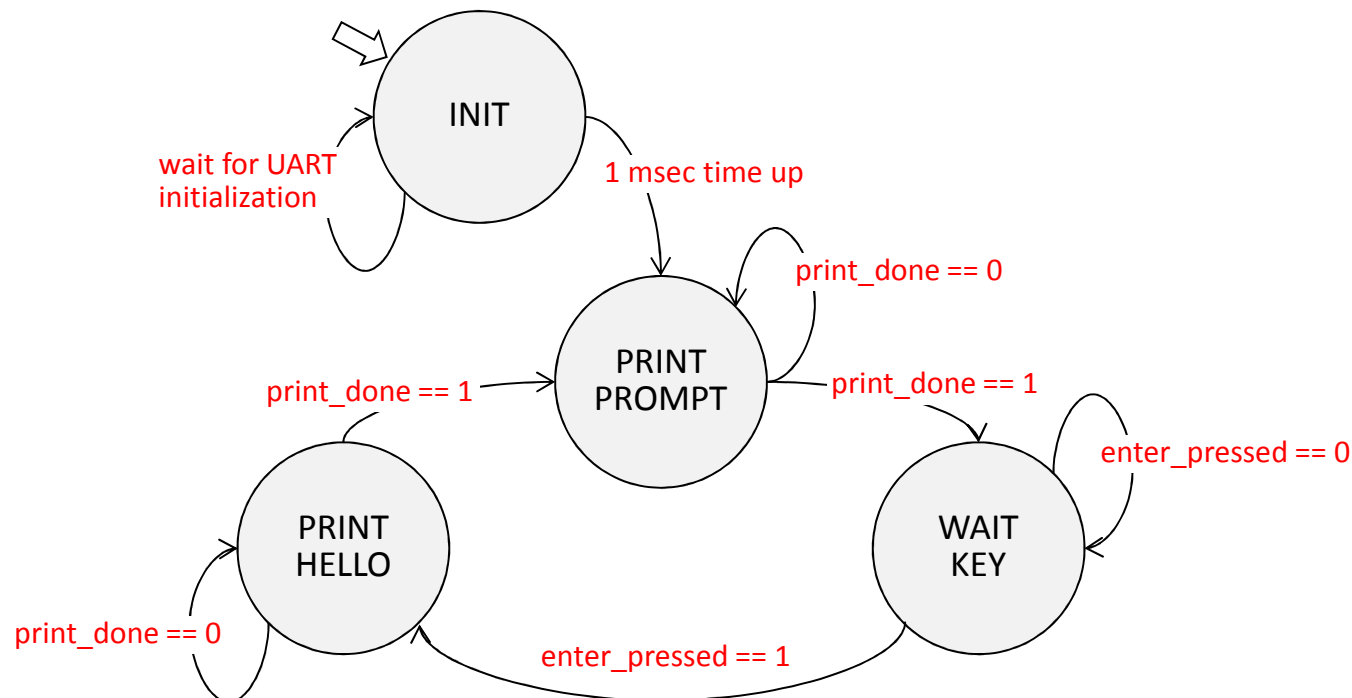
Top-Level Block Diagram of Lab4

- ❑ The block diagrams of different circuit blocks in the lab4 sample code:



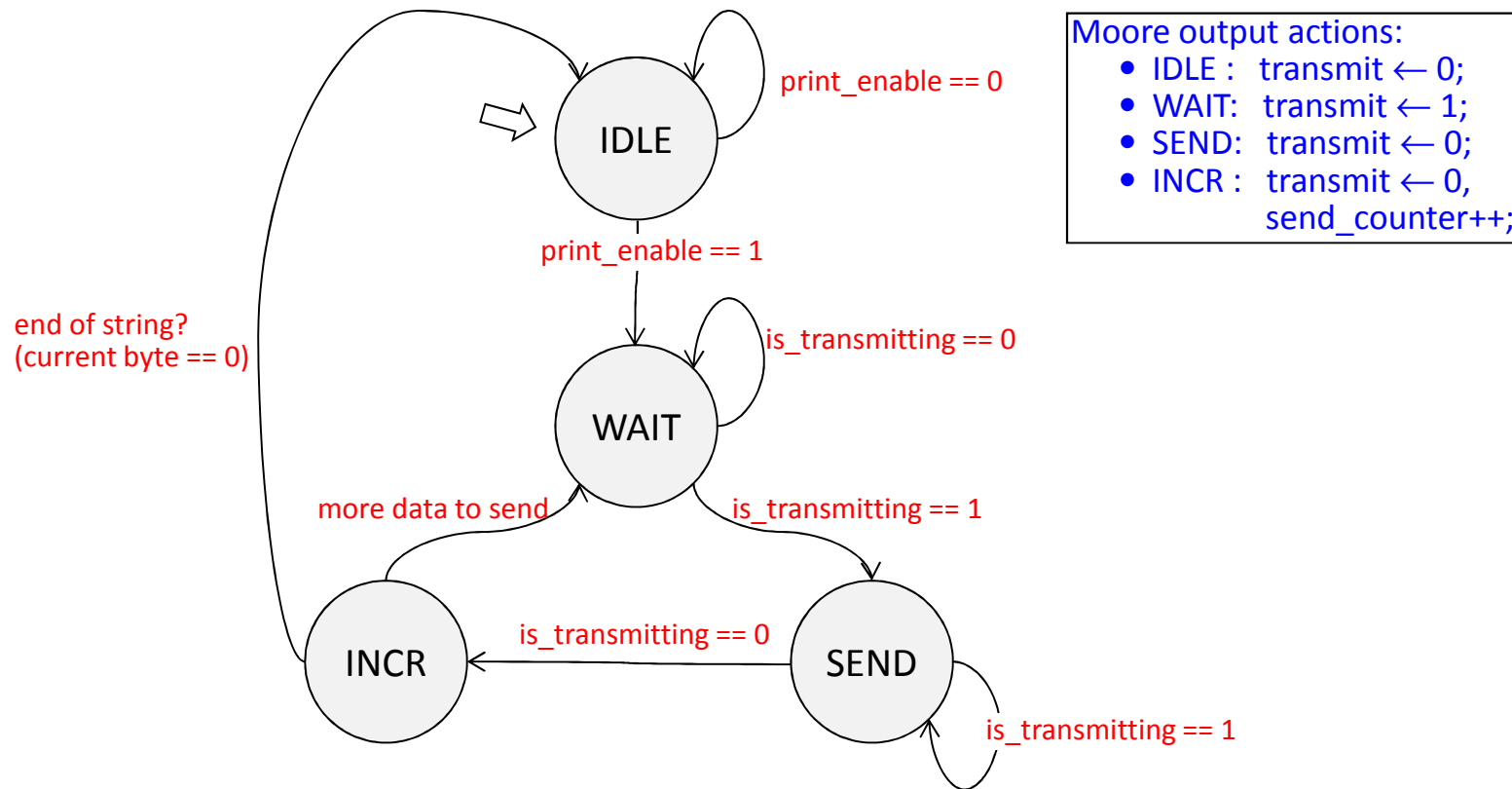
FSMs of the Sample Code

- ❑ There are two FSMs in the sample code
 - The first one controls the main program flow
 - The second one controls the print string function
- ❑ The main FSM is as follows:



The Print String FSM

- ❑ We can send data to the UART controller when it is not busy:



Your Task in Lab4

- ❑ Write Verilog code to read a decimal number input from the TeraTerm window and convert it to a hexadecimal number and print the number.
- ❑ Your screen outputs may look as follows:

```
Enter a decimal number: 1759  
The hexadecimal number is: 06DF  
  
Enter a decimal number: ■
```

Note that the sample code does not echo your input keystrokes (e.g., 1759) to the terminal window. You must add code to do that.

Reference

- ❑ Chapter 8 of P. Chu's *FPGA Prototyping by Verilog Examples*, Wiley, 2008.