Lab 5: UART Communications

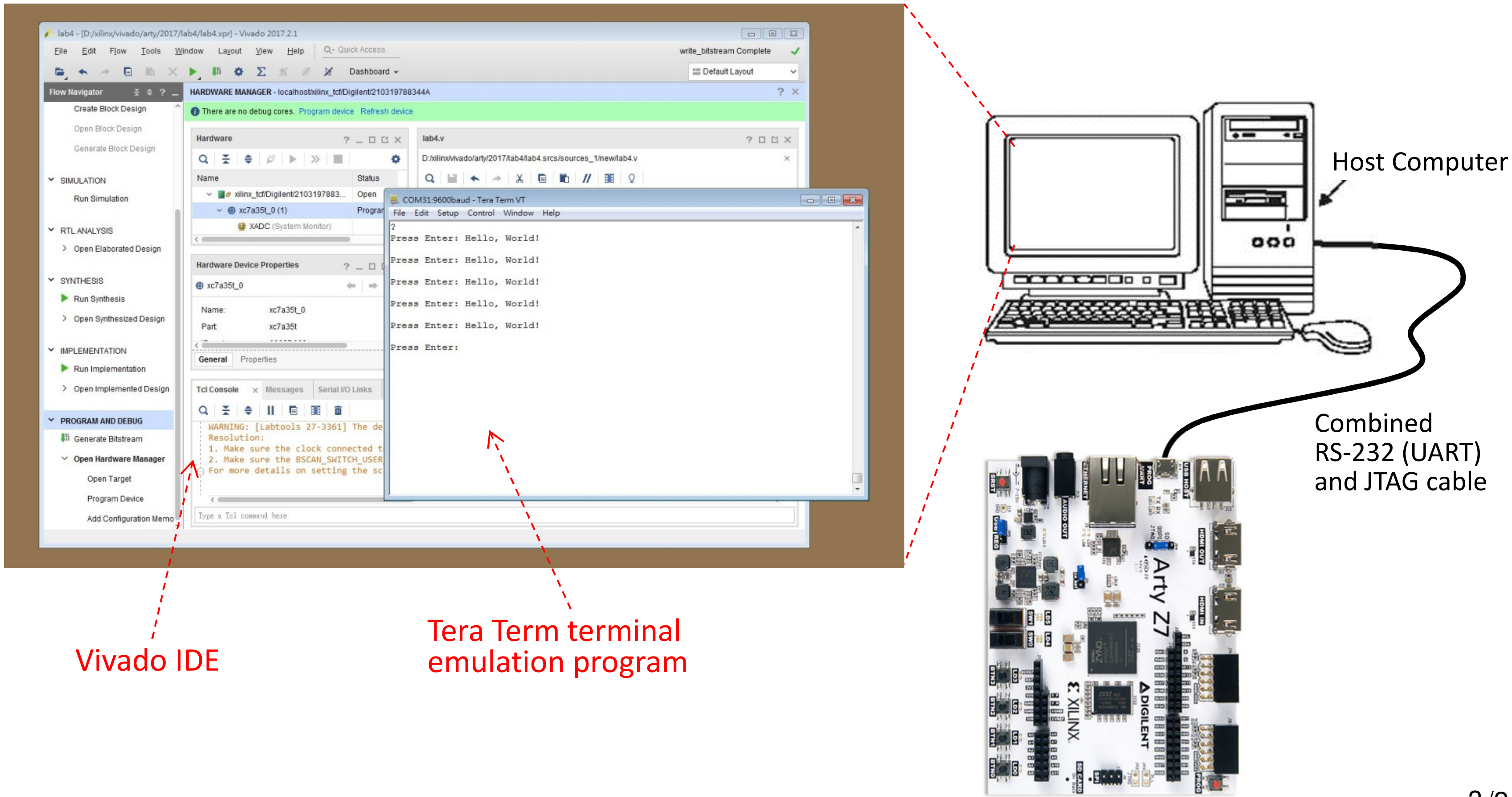
National Chiao Tung University
Chun-Jen Tsai
10/16/2018

Lab 5: UART Communications

- In this lab, you will design a circuit to perform UART I/O. Your circuit will do the following things:
  - Read two unsigned 16-bit decimal numbers from the UART port connected to a PC terminal window. The numbers range from 0 to 65535.
  - Compute the integer division of these two numbers, and print the quotient to the UART terminal in hexadecimal format
- The deadline of the lab is on 10/30, 5:00 pm

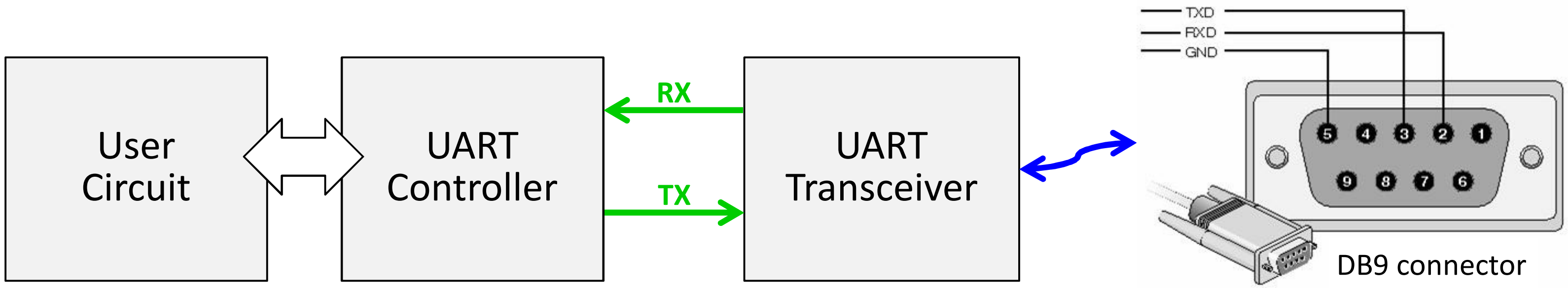
Setup of Lab5

- Lab 5 tests the communications between the PC and the FPGA through the UART devices (i.e., RS-232):



Traditional UART Devices

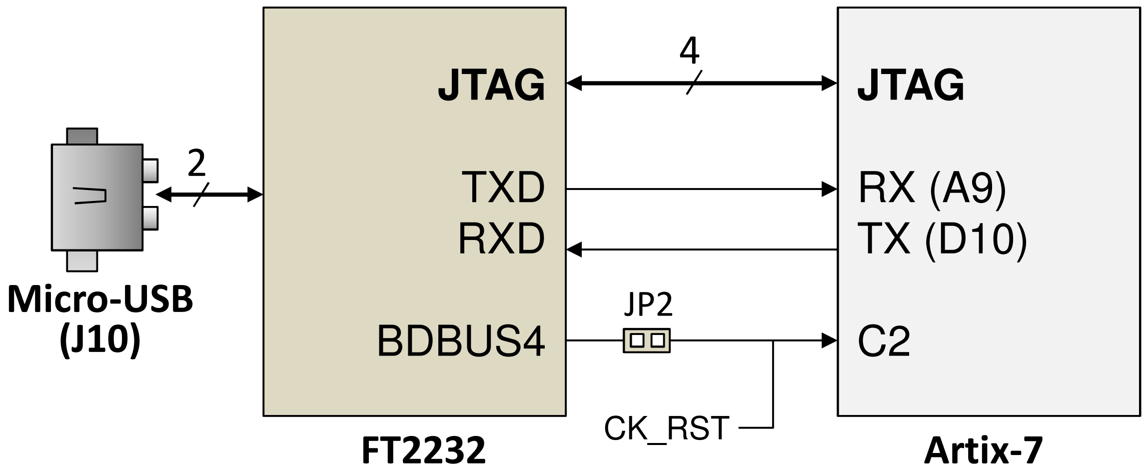
- Universal Asynchronous Receiver/Transmitter (UART) is one of the most popular I/O devices in small systems



- In simple systems, the UART transceiver can simply be a voltage translator IC

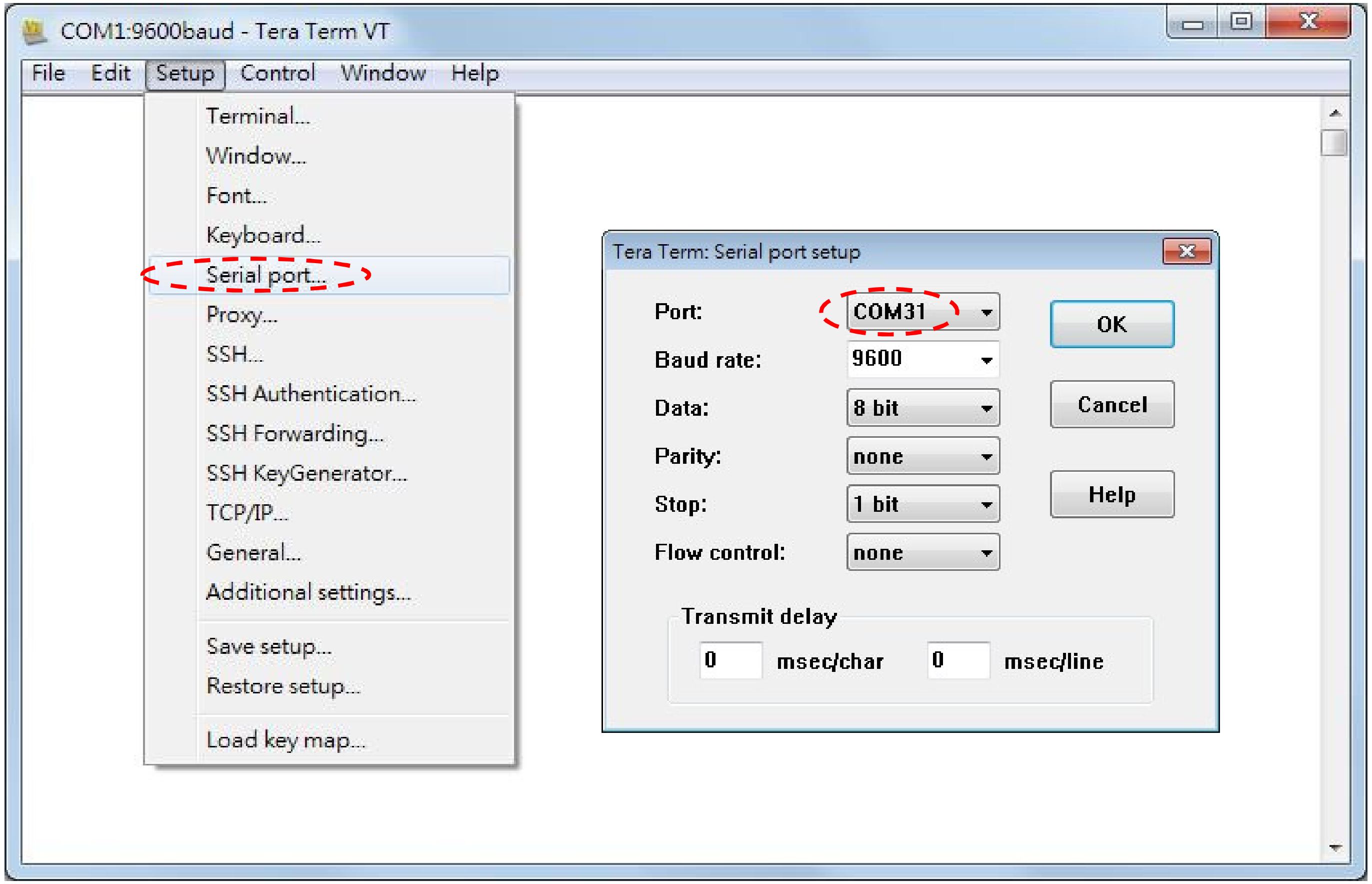
UART Devices on ARTY

- On Arty, the digital UART signals are converted to the USB data frames through a FTDI FT2232HQ USB-UART bridge IC
  - There is no traditional DB9 connector on ARTY!



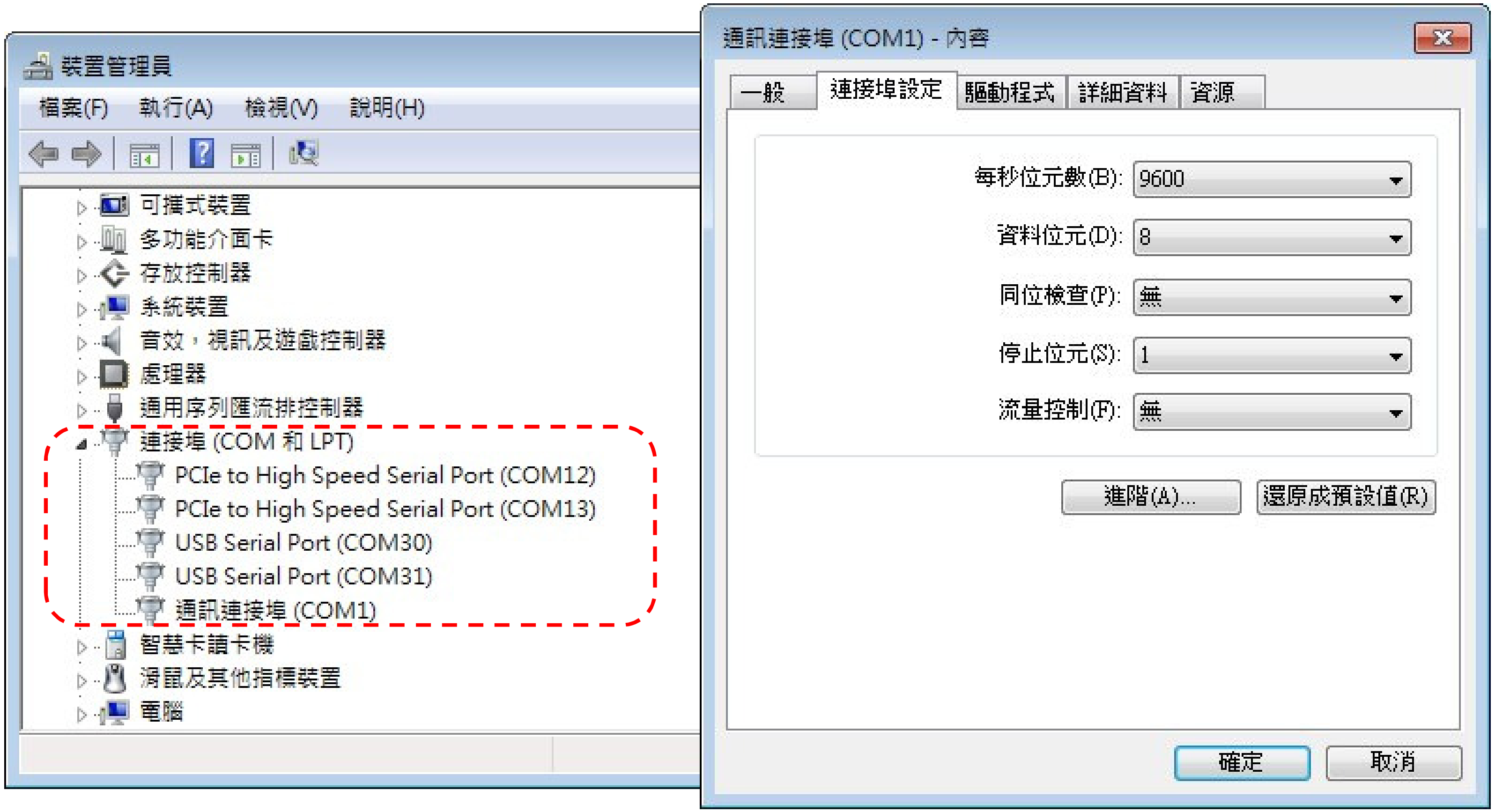
Running TeraTerm on the PC Side

- On the PC connected to the Arty 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

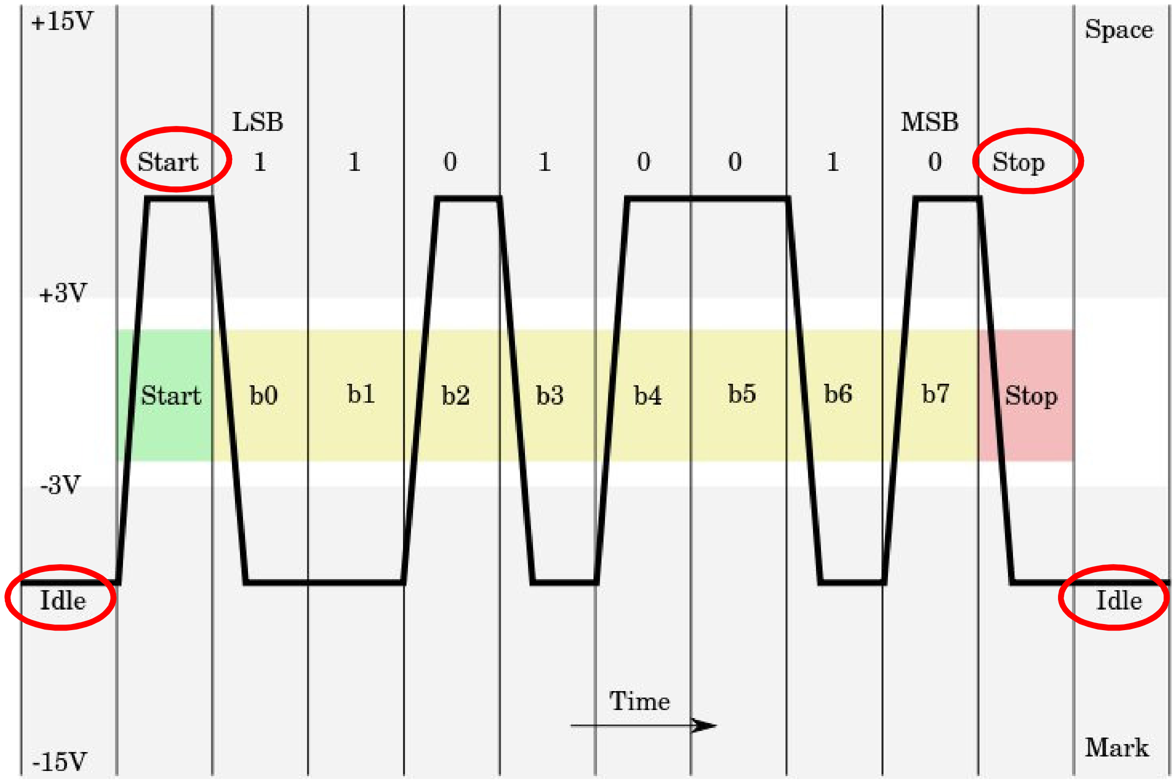
- UART is a asynchronous transmission standard, thus, there is no common clock signal for synchronization
- The most popular physical layer for the UART transmission line is the RS-232 standard
  - Common baud rates for RS-232 signals range from 4800 bps to 115200 bps
  - RS-232 voltages are (–15V, –3V)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

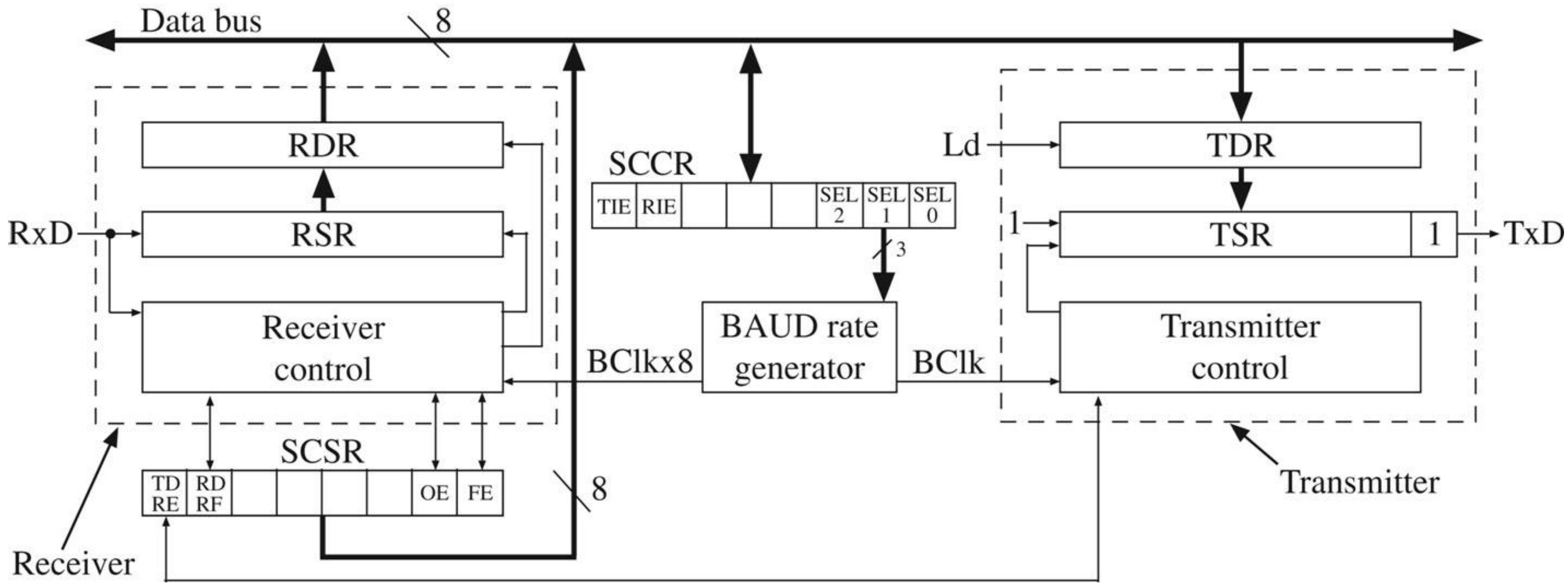
- Implicit clocks must be generated on both sides for correct transmissions
  - Bit rate per second (bps), or baud rate, is used to imply the clock on both end of the transmission line
  - Common baud rates are 4800, 9600, …, 57600, 115200, etc.
  - This clock is often called the baud rate generator

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

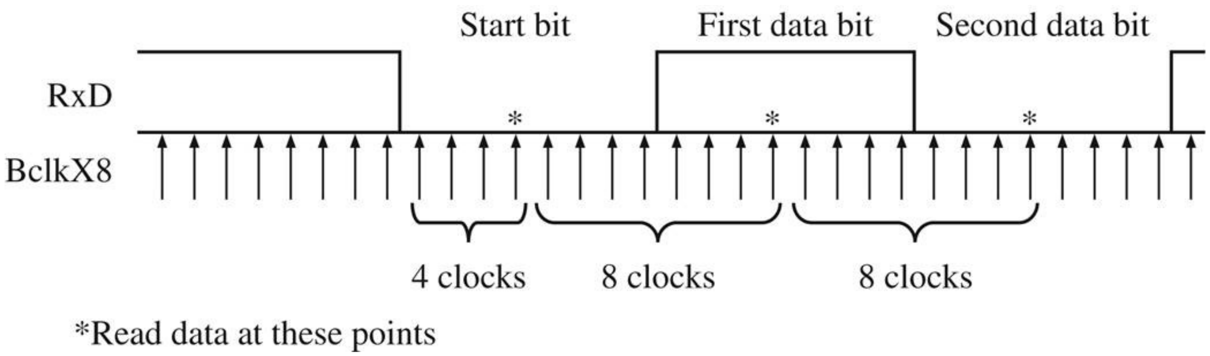
- The following diagram shows a typical UART controller:



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 4x, 8x, or 16x sampling
- Take 8× sampling for example, the fourth sample of each bit time will be read as a data bit



Baud Rate Generator

- Since the system clock rate is often much higher than the baud rate, we must slow down the system clock to generate a UART clock.
- For example, if a 16x baud rate clock is used:
  - If baud rate is 9600 bps, $9600 \times 16 = 153600$
  - If the system clock is 100 MHz, the clock divisor should be: $100\text{ M} / 153600 = 651.041 \approx 651$
- In the UART controller (uart.v) in Lab5, 651 is used as the system clock divisor to generate a baud rate clock @ 153.6 kHz

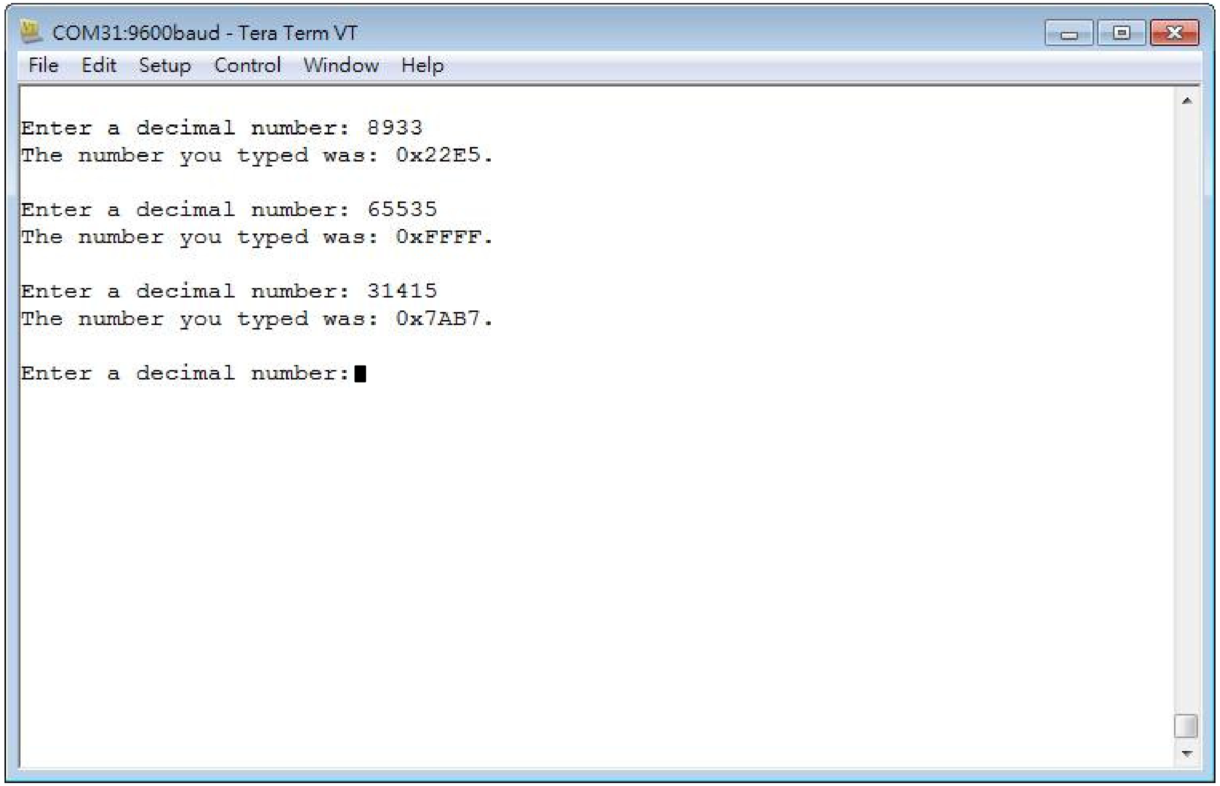
About the Lab 5 Sample Package

- The package contains a Vivado project files that shows you how to use a circuit to read a decimal number from the user keyboard inputs and then print the number in hexadecimal base through the UART controller
- The source files are as follows:
  - `lab5.v` → Top-level module, with two FSMs for flow control
  - `uart.v` → An UART controller†
  - `lab5.xdc` → the constraint file

† The UART controller is based on a controller by Timothy Goddard downloaded from www.opencores.org.

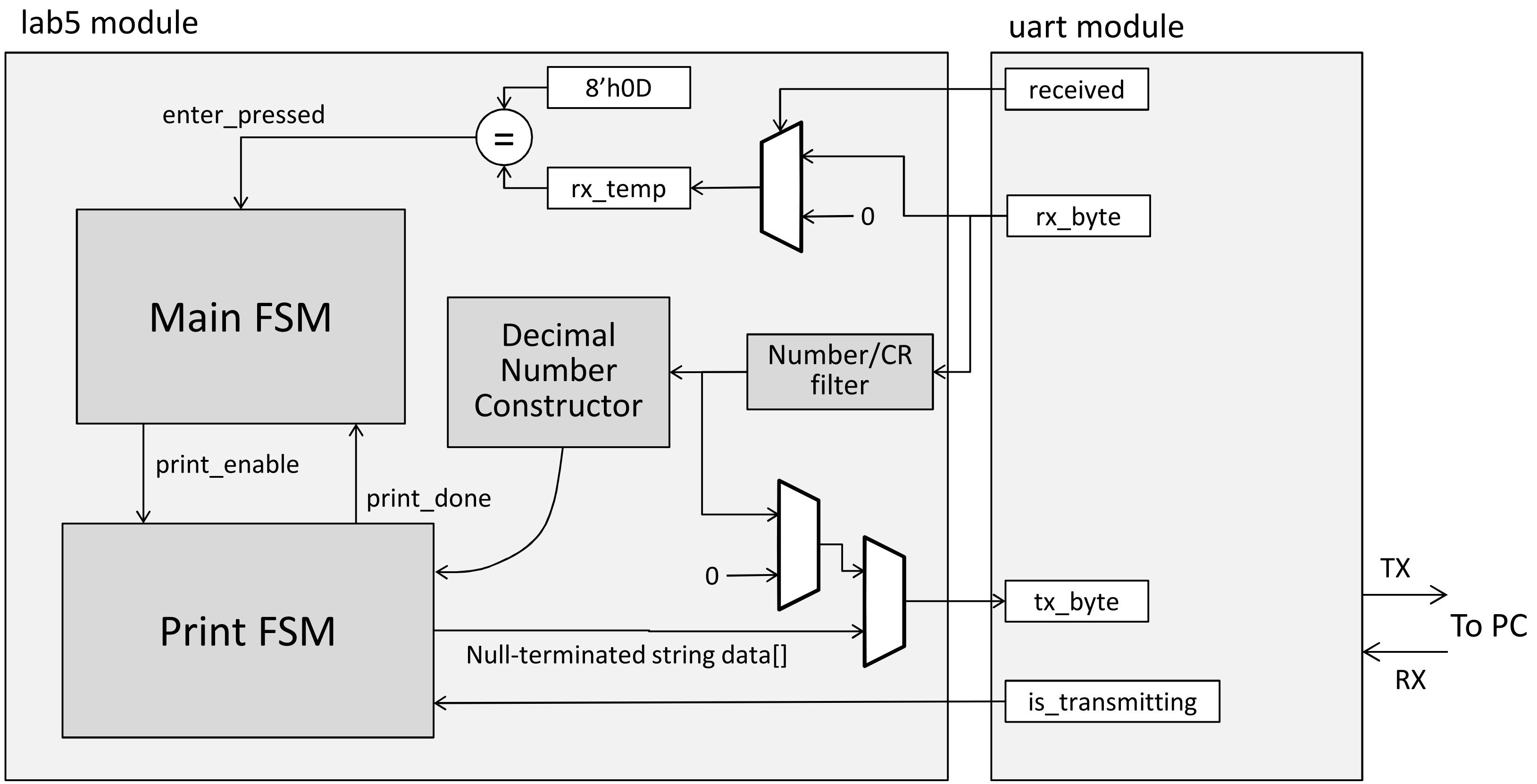
Screen Shot of Lab 5 Sample Code

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



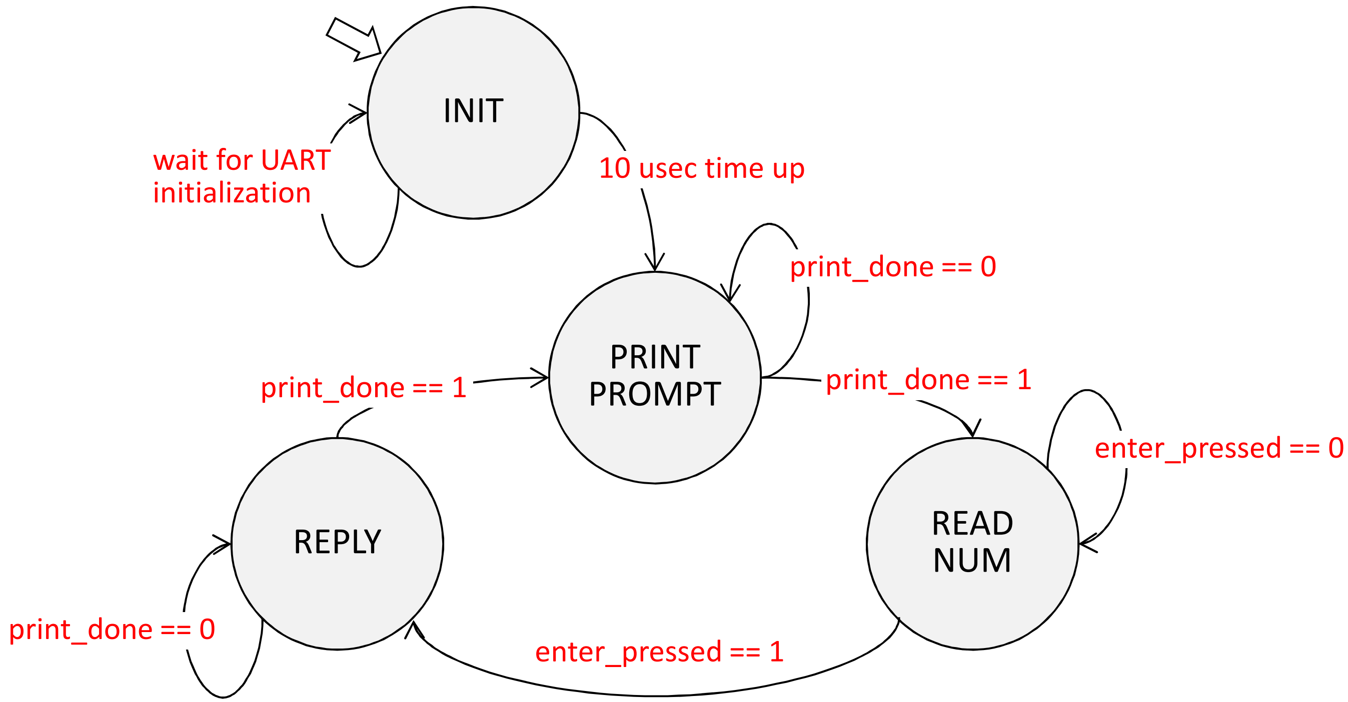
Top-Level Block Diagram of Lab 5

- The block diagrams of the different circuit blocks in the lab 5 sample code:



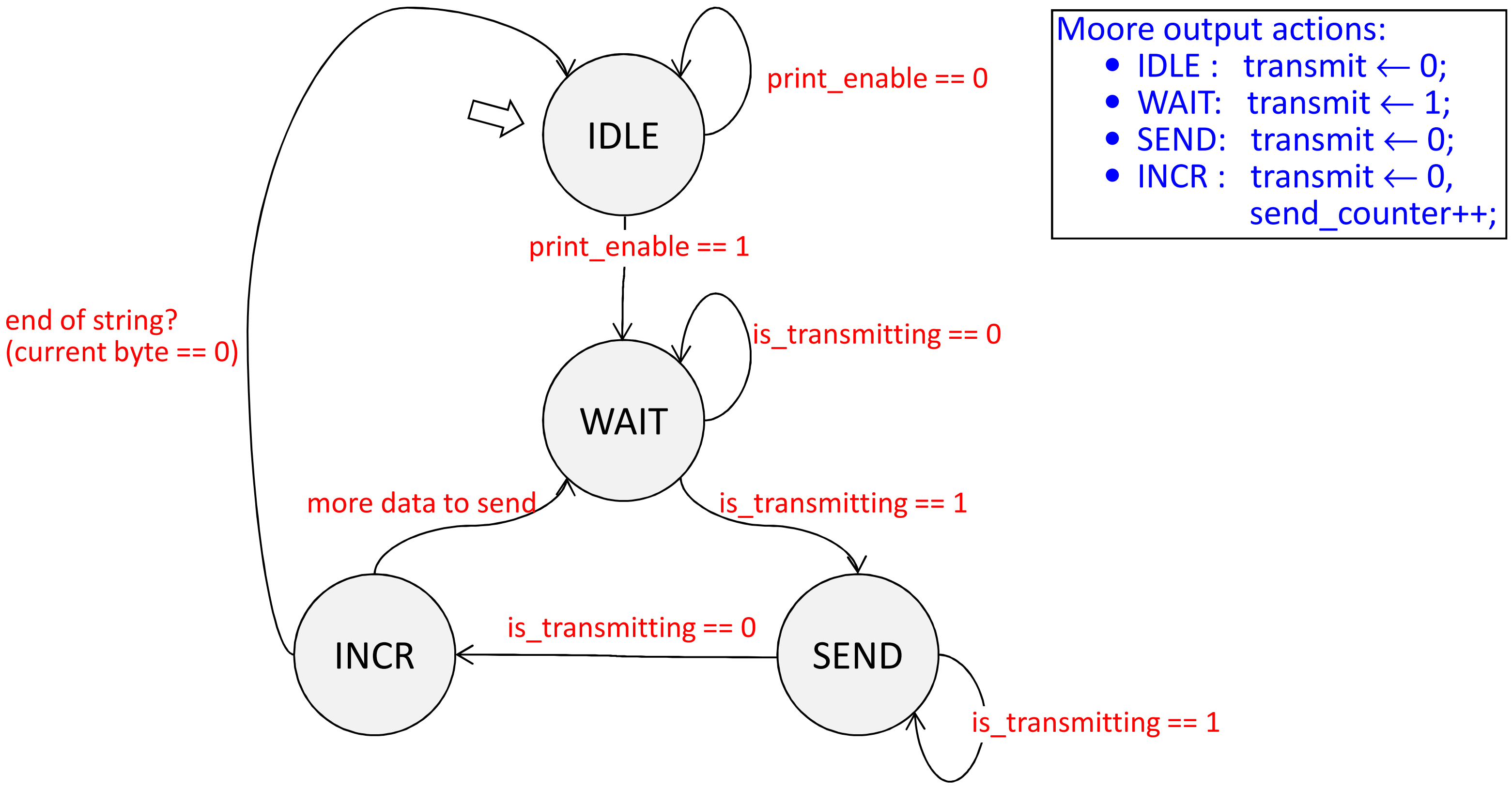
FSMs of the Sample Code

- 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 Lab5

- Design a circuit to read two decimal numbers from the UART terminal, compute the integer quotient, then print it in hexadecimal format to the UART terminal
  - The first input number is the dividend, and the second one is the divisor
- Your screen outputs may look as follows:

```
Enter the first decimal number: 8976
Enter the second decimal number: 96
The integer quotient is: 0x005D

Enter the first decimal number: _
```

About Division Circuit

- The simplest way to compute 16-bit unsigned integer divisions is Successive Subtraction:
  - Too slow to use! DO NOT use this method!

```
// Division of A/B
// Q: quotient
// R: remainder

Q ← 0, R ← 0;
while (A ≥ B)
  A ← A – B;
  Q++;
end
R ← A;
```

Long Division

- A more efficient way is to use a shift-and-subtraction algorithm
  - Much more efficient than successive subtraction
  - Need a shift register

```
// A is an N-bit number,
// B is an M-bit number, compute A / B

Q = 0, R = 0;
for (i = N-1; i ≥ 0; i--) {
  R = R << 1;   // left-shift R by 1 bit
  R(0) = A(i); // R(0) is 0th bit of R,
               // A(i) is i-th bit of A
  if (R ≥ B) {
    R = R - B;
    Q(i) = 1;
  }
}
```

Requirements for Lab 5

- There are two requirements for lab 5:

1. You must use long division to implement your division circuit
2. Your integer divisor must be implemented as a module
   $\rightarrow$ You will need it for your midterm exam!