ECE 3216: Computer Systems Design Lab

Lab 2: RS232 Signal Generation - Software

Anthony Lam, Robert Campbell

21 February 2020

Objective: The goal of this lab is to program a microprocessor to communicate with the BlueFruit BLE radio. The RS232 port of a UART was configured and an RS232 Signal was generated and observed using PuTTY and an oscilloscope.

Equipment:

-PuTTY terminal on Windows

-USB to USB micro cable

-USB to female RS232 cable

-BNC cable

-Oscilloscope

-Silicon Labs

-C8051F320 UART

Procedure:

The UART was configured to output an RS232 signal. A function was coded using SiLabs IDE. The function transmits one character from an array of characters loading it into the SBUF0. An oscilloscope was then used to validate the RS232 output and PuTTY was then used to monitor the string output. The outputs were then each observed and recorded.

Data and Observations:

main.c

```
extern void Init_Device(void);


void main()
{
      int i;
```

```
            unsigned char string[] = {'H', 'e', 'y', ' ', 't', 'h', 'e',
'r', 'e'};


            Init_Device();


            while (1)
            {
                  for (i = 0; i <9; i++)
                  {
                        SBUF0 = string[i];
                        //while (TI0 == 0) {};
                        //TI0 = 0;
                        while ((SCON0 & 0x02) == 0) {};
                        SCON0 = SCON0 & 0xFD;
                  }
          }
      }
```

The code initializes an array with the message “Hey there”. It then goes into an infinite loop of loading SBUF0 (serial interface buffer register 0) with the characters (can potentially use TI0 to check). The code then uses the and operation to check if SCON0 (serial control register 0) is equal to 0 to program the port.

Analysis and Discussion:

This lab was successful. The generated signals from the UART matched expectations, with the message “Hey there” being correctly generated and observed onto the PuTTY terminal. The output was verified on the oscilloscope as well before observing it using PuTTY, to ensure the process was safe and complete. Analysis is not applicable in this experiment.

Questions:

How are PC Baud rates historically generated?

Baud rates were historically generated by an integer clock division of the original IBM PC’s internal clock of 115200 Hz.

How are Baud rates generated in the C8051F320?

The baud rates are generated as half the clock from Timer1 divided by 256 (using the high byte of Timer1).

What is the percent difference in the PC baud rate and the baud rate of your designs?

There is a 0.16% difference in the 9600 PC baud rate and the UART baud rate.

What is the maximum allowed difference in Baud rates at the rate you choose, demonstrate your answer with a plot, annotation, and text? If the baud rate if the receiver is faster or slower than the transmitter baud rate, this causes the first data bit to be sampled slightly before or after its center by the baud rate differences. This continues on with each additional bit where the error essentially cumulatively becomes larger until a situation may occur in which the receiver samples a bit twice. A 25% error range on the center is generally safe for communication.

9600 N81

$9T_P + 0.5T_P - 9T_R - 0.5T_R <= 0.25T_P$

$1/T_R <= 9354$ Baud

$1/T_R <= 9859$ Baud

The baud rate range where an error would likely not occur is between 9354 baud and 9859 baud.

Conclusion:

The purpose of this lab was to gain familiarity with the UARTs role in interfacing between parallel and serial signals and utilizing a UART through C programming. On the UART side, the char is loaded nearly immediately and must wait to transmit the data serially, once the data is sent the next char can be loaded. Using C allowed us to skip over the formatting of the RS232 message that the UART transmitted. After performing this experiment, it is easy to see how this process could be scaled up to send larger messages, such as programming the initialization of a Bluefruit.