“Serial Terminal” using UART and SPI peripherals on EFM8BB2 Mictrocontroller

Yingzhi Hao and Nithilam Subbaian

***Abstract*- The EFM8BB2, “Busy Bee”, microcontroller receives data through UART and displays the received data on its LCD screen. The user can use the joystick to scroll the screen up or down to see the data that doesn’t show on the screen. All data showing on the screen can be switched between regular character and their corresponding ASCII hexadecimal number. A Raspberry Pi 3 Model B was also used to communicate with EFM8BB2 through UART.**

I. INTRODUCTION

The EFM8BB2, “Busy Bee”, microcontroller was set up to receive data through UART0 and display the received data on the EFM8’s 128*128-pixel LCD screen in the format of 16 characters per line. A total 22 lines of data can be stored and the whole screen can display 12 lines of characters at a time. The rest 10 lines can be accessed by using the joystick to scroll the screen up or down. All data can be displayed in either character format or their corresponding ASCII hexadecimal number. The left press-button PB1 on the EFM8 board is used to switch the display mode. The EFM8 can display data inputted through a keyboard of a computer, or the data outputted from the serial port from the raspberry pi when it turns on.

II. PROCEDURE

A. Raspian on Raspberry PI

First, Linux, or Raspbian, was imaged onto a micro-sd card by using a graphical SD card writing tool, “Etcher” [1]. The micro-sd card was inserted into the Raspberry Pi and used as memory and data storage. The default of Raspbian OS is outputting its console to its serial ports.

B. EFM8BB2 to display Serial Data

The EFM8BB2 was programmed to process and display the received data from UART. It was first tested by connecting it with USB cable to computer through the terminal “Putty”. Then it was connected with Raspberry Pi.

C. Connect EFM8BB2 with Raspberry Pi

The EFM8BB2 was connected with Raspberry Pi through UART and it displayed characters sent by the Raspberry Pi when the Raspberry Pi was booting.

III. DESCRIPTION OF CODE

For the device initialization, the watchdog timer is disabled. The interrupt of Timer 2, Timer 3, SPI0, ADC and Port-Match are enabled. The Internal High Frequency Oscillator 0 is set to be the system clock, and its frequency is 24.5 MHz. Crossbar XBR2 is set to enable crossbars and the weak pullup of every pin. XBR0 is set to enable the SPI0 and UART0. The ADC0 and Voltage Reference are also enabled. Peripheral pinout of the microcontroller is shown in the Table 1.

Pin 0.2, which connects the PB0, is set to compare with the high value. When the PB0 is pushed, the pin value will change from high to low, and the Port-Match interrupt will be triggered.

The Timer/display code works through files including draw.c, spi.c, tick.c, spi_0.c, disp.c and render.c file. These files set the configurations for the display. A function, “void DrawScreenText(char str, uint8_t y)” is used to draw characters on the screen. This function draws a char at the position y on the screen.

Since the microcontroller doesn’t have enough internal data memory space, external data memory on the chip was used to store char data. Two xdata char arrays, screenhex[] and screentotal[], were declared and used to store hexadecimal format of strings and char format strings.

When there is input from the keyboard, or the serial ports from the raspberry pi, the UART interrupt is called. An integer variable “Byte” is set to store the received data from receiver register SBUF0, and the LED1, which represents the register of Pin 1.5, is turned on.

In the main function, when the LED1 is on, the variable Byte is transformed to in using the format specifier “%x” into the array for hex values, screenhex[], and the same data is written using the char typecasting into the array screentotal[], and then the LED1 is changed to 1 again to prevent the same data being added into the array again.

Strings in the screenhex[] or screenhex[] are passed into the function “DrawScreenText” to draw the characters onto the screen.

In the Port-Match Interrupt function, the LED2, which represents the register of Pin 1.6 is toggled every time the interrupt is triggered. When PB0 is pressed, LED2 will toggle and the display will switch to hexadecimal numbers if LED2 becomes 0 and switch to regular characters if LED2 becomes 1. The Port-Match interrupt is disabled in the interrupt function and reenabled in the main function, if PB0 is 1, to ensure that if the button is pressed and held, the interrupt can only be triggered only once.

The ADC interrupt is used to get the voltage of P1.7 that connects the joystick. When the joystick is pushed up or down, different voltage will be on the P1.7. Thus, the moving direction of the joystick can be recognized by detecting the voltage on P1.7. A function "void JOYSTICK_convert_mv_to direction(uint8_t mv)" is used to convert the voltage to a number that represents the direction of the joystick. In our code, number 2 represents moving up and number 4 represents moving down. LED0, which represents the register of P1.4, is used to tell the main function to trigger the function "void increment(int i)" or "void incrementhex(int i)" to move the context on the screen.

The function "increment" and "incrementhex" put twelve arrays, whicdiffh contains characters of twelve lines, onto the screen.

IV. Different Iterations of The Program

While completing our project, there were numerous problems we encountered. These problems were then solved by trials and error, transferring files, learning more information and talking to peers.

A. *Interrupts and getchar function*

Originally the code was written in a way that the UART interrupt was not used, and the getchar() function was used. However, when the getchar() function was waiting for an input, the program is in a while loop and the following code in the main function cannot be executed. Thus, unless an interrupt was used, the code was always stuck at that line.

B. *SBUF0 register*

In the EFM8 the ports 0.4 and 0.5, respectively are the UART0-TX and UART0-RX pins. In the second iteration, we eliminated the getchar() function, and used SBUF0 in its place. The SBUF0 is the UART Serial Port Data Buffer. This is a special function register which when data is written to it, and TXNF is 1, the data is put in the transmit FIFO for serial transmission. If TXNF is 0, the data written to SBUF0 will replace the most recent byte in the TX FIFO. On the other hand, when RI is 1, the SBUF0 will read the latest byte in the RX FIFO, whereas when RI is 0, it will return the most recent data byte in the RX FIFO. [4]

C. *Memory*

Another problem we reached in writing our code is the error "address space overflow". This error occurred because during the building phase, when the linker assigns all variables and functions to the respective memory segments, if the maximum size of that particular segment is exceeded, this error occurs. To solve this problem, we used XDATA. XDATA is the second slowest and compact memory segment. The phrase "SI_SEGMENT_VARIABLE (name, vartype, memseg)" is used to declare the name of the variable, data type, and the memory segment it will use. In the code, we used XDATA when declaring the two arrays used to store the data.

V. Inherent Problems

With the current code, when the Raspberry Pi is connected to the EFM8, it sends information to the EFM8 but the EFM8 displays broken information, (broken text with a lot of spaces, or information missing).

This occurs because when the raspberry pi sends data to the EFM8 through UART, the received value is stored in the SBF0 register, which is then put in the variable "Byte". Then in the main function the variable "Byte" is put into the array. However, in the circumstance when the code is running and hasn't reached that line yet to put the variable "Byte" into the array and if the UART gets called again, then that prior information is lost before it gets put into the array. This causes there to be broken output onto the LCD display on the EFM8BB2.

VI. Conclusion

The project was completed and turned in with all the required components. The EFM8BB2 took the input from the Raspberry Pi or the input from user keyboard through UART and displayed the data on the LCD screen. On the EFM8BB2, the joystick can be used to scroll up and down the lines of data, and the left push button can be used to switch the displayed data between char and hex.

VII. Additional Tables and Figures

The following tables and figures show the carious ports used in the program, and the output on the LCD screen of the EFM8 BB2.

A. *Figures and Tables*

EFM8 MictroController Main Ports Used

Port Name	***Connection***
P0.1	CS (Active High)
P0.2	Push Button 0
P0.3	Push Button 1
P0.4	UART0-TX
P0.5	UART0-RX
P0.6	SPI0-SCK
P0.7	SPI0-MISO
P1.0	SPI0-MOSI
P1.4	LED Green
P1.5	LED Blue

Port Name	***Connection***
P1.6	LED Red
P1.7	ADC-IN
P2.3	Display enable

Here is a link to our code:

https://github.com/1995hyz/ece251-project1/tree/master/ece251a-final

Here is a link to our project working:

https://www.youtube.com/watch?v=16mOPz714eo&feature=youtu

.be

Here is our commit number:

```
d2d161afa0792052342d682d9384e4560c288b3d
```

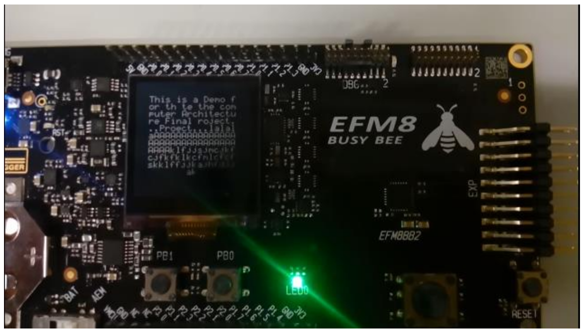
Fig. 1. Example of a LCD screen displaying input from Keyboard

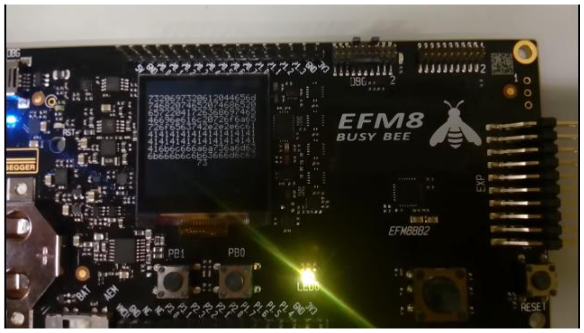
Fig. 2. Example of the Hex display of input from Keyboard

VIII. Acknowledgement

Y. H. and N. S. thanks Professor Alin Cosmanescu for spending extra class time with us on the weekends to help us with our projects. They would also like to thank their fellow classmates and other students in the J-Lab that helped them along their journey.

VIIII. Reference

[1] "Installing Operating System Images." Installing Operating System Images - Raspberry Pi Documentation. RASPBERRY PI FOUNDATION, n.d. Web. 07 May 2017.

[2] Hao, Yingzhi, and Nithilam Subbaian. "1995hyz/ece251-project1/Tryit." GitHub. N.p., 13 Apr. 2017. Web. 02 May 2017.

[3] Busy Bee Family UG237: EFM8BB2-SLSTK2021A User Guide (n.d.): n. pag. Silabs. Silicon Laboratories Inc. Web. 2 May 2017.

[4] Controllers. "EFM8 Busy Bee Family EFM8BB2 Reference Manual." (n.d.): n. pag. Keil. Silicon Laboratories Inc. Web. 2 May 2017.

[5] M, Chris. "Address Space Overflow Troubleshooting." Blog post. Silicon Labs Community Knowledge Base : Microcontrollers Knowledge Base :. Silicon Labs, 02 Dec. 2015. Web. 10 May 2017