CECS-412-01

Summer 2017

Project # 3

Copyright 2017, Jack Dorne, Sam DuPlessis, Matt Elmlinger, Aryan Ghazipour, Logan Schaufler

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Report  (20 Points) | Demo  (15 Points) | Quiz  (5 Points) |
| Group 8  Jack Dorne, Sam DuPlessis, Matt Elmlinger, Aryan Ghazipour, Logan Schaufler |  |  |  |

LAB 3

Group 8

CECS-412-01

Abstract

I/O communications through implemented drivers written in C are studied in this lab. This device onboard the A3BU that is used to accomplish this is through the Universal Synchronous Asynchronous Receiver Transmitted (USART), an I/O Peripheral device that permits serial I/O Communication. First, an RS232C serial interface circuit was built which allowed for asynchronous serial communication between the A3BU and a PC running terminal communication software, and an example driver was flashed onto the board to initialize the USART. If done correctly, ASCII characters would be communicated from the PC terminal to the A3BU, and then echoed back to be displayed on the terminal. Next, LCD synchronous communications were investigated through the PORT D Serial Peripheral Interface (SPI) and the LCD ST7565R MCU. Another example driver was flashed onto the A3BU to initialize the port. This time, lines were first drawn across the screen and then the screen was rolled. Finally, the two example drivers were integrated into one driver called SYSTEM, and a MAIN was written to receive ASCII characters from the keyboard to be displayed on the LCD. Through the successful completion of this lab, basic communication between an MCU and external hardware is understood.

Body

# Part 1

The Tera Term software was downloaded for aid in every portion of the lab. Tera Term was used as the chosen terminal program to display procedures that would be carried out to the ATxMEGA256A3BU Xplained board. To interface the A3BU board, a RS232C serial interface circuit (Schematic located in “Schematics” Section) was implemented. The circuit was then assembled and inspected. Once the circuit was set up, the program USART\_EXAMPLE was built and stored on the A3BU board. The function of USART was to communicate to the A3BU board by way of TxD and RxD asynchronously (of header J1). Pin PC3 served as the TxD pin and PC2 served as the RxD pin on the A3BU board.

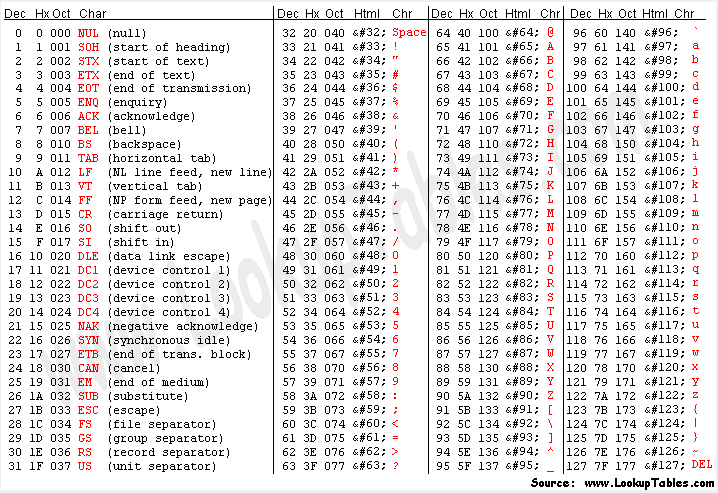
Serial port configuration settings next had to be configured. The number of data bits was set to 8, BAUD RATE to 9600, one stop bit used, and no parity bit was necessary in the C program (Tara Term would also follow these settings).

The program outputted a “Hello World” message on the Tera Term screen. This was possible by the translation of ASCII characters from the terminal back to the terminal by way of Tera Term.

The code was then re-written. The new program allowed for two single digit numbers to be added together. Again, ASCII values were translated, this time to single digit values. Since 0 has an ASCII value of 48, 48 had to be subtracted from the result of the addition of the two digits (any addition would result in a “c” output rather than the true sum value). This displays the true value of the new digit. This is an example of when digits “3” and a “6” are inputted through Tera Term, which results in their sum, 9.



An ASCII table was necessary for reference in part 1:



The rate at which bits are communicated between the PC and the A3BU is called “BAUD rate.” The RS232 aids by allowing both the PC and A3BU to know when bits are being communicated, and subsequently when this processes is stopping. A higher BAUD rate allows for data to transferred faster, however this comes with risk as data can be misinterpreted. A lower BAUD rate may run at a slower rate, but data is rarely, if ever, misinterpreted.

A chart of the USART registers that were involved in the program:

|  |  |
| --- | --- |
| Register | Purpose |
| UDR | 16-bit USART Data Register. The Transmit Data Buffer Register (TXB) is where UDR received data is stored. |
| UBRRH | Stands for USART Baud Rate Register High. The first 8 bits of the 16-bit BAUD rate register are stored here. |
| UBRRL | Stands for USART Baud Rate Register Low. The last 8 bits of the 16-bit BAUD rate register are stored here. |
| UCSRA | 8-bit USART Control and Status Register A. The flag bit is set by the CPU when unread data is in the Receive Buffer, and then cleared by the CPU when the Buffer is empty. |
| UCSRB | 8-bit USART Control and Status Register B. The UCSRB enables interrupts, receiver, transmitter, and third bit character size. |
| UCSRC | 8-bit USART Control and Status register C. Allows for stop bits, Asynchronous or Synchronous mode, parity type, selection of registers, and selection of two bits of character size. |

(The USART of the AVR » MaxEmbedded)

# Part 2

**LCD Synchronous Communication**

Part 2 introduces the concept of synchronous communication. This form of communication involves using the system clock rate as a reference to send signals between different devices on the A3BU board. Through using the same clock rate, these devices are considered in-sync. In synchronous communication, a continuous stream of data signals is transferred to the desired device. This allows for a more predictable outcome compared to asynchronous communication. On the A3BU board, the LCD ST7565R MCU is an example of a device that uses synchronous communication to receive data signals.

**Serial Peripheral Interface**

PORT D on the A3BU board is a Serial Peripheral Interface (SPI) device. SPI is an interface bus that allows the A3BU board to communicate with smaller peripheral devices such as the LCD module used in part 2. The ST7565R example program relates the SPI to the LCD. The SPI device on PORT D of the A3BU is considered the master because it generates the clock signal. The LCD module is considered the slave because it receives these signals. The SPI has registers that are capable of initiating data transfer.

The ST7565R code was rewritten to send and display alphanumeric text/characters. To accomplish this, the LCD module must first be initialized. The initialization code uses seven header files: board.h, sysclk.h, st7565r.h, gfx\_mono.h, gfx\_mono\_text.h, sysfont.h, and gpio.h. The file board.h configures the board by enabling the LCD screen. The file sysclk.h configures the system clock to be used for synchronous communication. The st7565r.h file configures a number of factors of the LCD module. It selects the SPI interface and sets the minimum clock period and the maximum frequency. It also sets the contrast and sets up the registers that will be used to hold information used by the LCD module during clock cycles. The two header files gfx\_mono.h, and gfx\_mono\_text.h initialize gfx functionality which allows the user to write text to the LCD. The gpio.h file enables the backlight on the LCD.

**Description of Code**

A function called InitializeLcdScreen is created that incorporates these header files to initialize the LCD. This function contains the initialization functions from most of the header files that were incorporated. It also clears the screen and lights it up. The function DrawWord is used to write the word hello on the screen. It takes as an argument a pointer to a string of characters and the length of that string. It then steps through each character in the string and displays it using the gfx function gfx\_mono\_draw\_char to place the char on the screen. The gfx function takes an X and a Y coordinate that defines a location on the screen. After each letter is displayed the X coordinate must be incremented so that the characters are not drawn on top of each other. The string “Hello” was passed into the DrawWord function. The text Hello, properly appeared on the LCD screen.

# Part 3

For this part of the project it was necessary to combine the functionalities of the last two parts of the project. A program that receives text sent from the terminal on the computer and displays them on the LCD screen will be written. The serial interface circuit must be able to collect data from the computer and send it along to the ST7565R MCU through the SPI.

**Description of Code**

The ASF wizard was used to incorporate all of the libraries that were needed to control both the Serial interface and the SPI interface to the LCD display. As in part 2, GFX functions were utilized that made displaying text on the screen very simple. The GFX functions display text in a certain location. An X and a Y coordinate must be passed to the GFX functions. The starting location is X=0 Y=0. After each character was displayed the X coordinate had to be incremented so that the characters were not written on top of each other. Then if the characters filled up a whole line of the LCD screen, the X coordinate was reset to 0 and the y coordinate was incremented. This made a new line of text appear. If the enter key was ever pressed the whole screen had to be cleared and the X and Y coordinates were both set to 0 so that the next characters were displayed at the starting location again. The code can be viewed in the source code section.

Source Code (Software)

**Part 1 Source code**

/\* Lab 3 part 1 \*/

#include <conf\_usart\_example.h>

#include <asf.h>

// some variables to store simple characters

static *uint8\_t* newLine = '\n';

static *uint8\_t* carriageReturn = '\r';

static *uint8\_t* numberOne = '1';

static *uint8\_t* ConvertToInt(*uint8\_t* numberToConvert) {

return numberToConvert - '0';

}

static *uint8\_t* ConvertToAscii(*uint8\_t* numberToConvert) {

return numberToConvert + '0';

}

static void DisplayNumber(*uint8\_t* numberToDisplay\_integer) {

*uint8\_t* numberToDisplay\_Ascii;

// Start a new Line

usart\_putchar(USART\_SERIAL\_EXAMPLE, newLine);

// Make sure the cursor is at the beginning of that line

usart\_putchar(USART\_SERIAL\_EXAMPLE, carriageReturn);

// If the number is greater than 9 the code needs to display 2 digits

if (numberToDisplay\_integer >= 10) {

// Convert integer to ASCII

// Only save the last digit

// (subtracting 10 removes the first digit)

numberToDisplay\_Ascii = ConvertToAscii(numberToDisplay\_integer) -10;

// Display leading 1 as first digit

usart\_putchar(USART\_SERIAL\_EXAMPLE, numberOne);

// Display second digit

usart\_putchar(USART\_SERIAL\_EXAMPLE, numberToDisplay\_Ascii);

}

else

{

// Convert integer to ASCII

numberToDisplay\_Ascii = ConvertToAscii(numberToDisplay\_integer);

// Display the digit

usart\_putchar(USART\_SERIAL\_EXAMPLE, numberToDisplay\_Ascii);

}

}

int main(void)

{

// variables for storing data at different

// stages of the conversion process

*uint8\_t* receivedByte\_Asc;

*uint8\_t* return\_num;

*uint8\_t* convertedNum\_prev;

*uint8\_t* convertedNum\_rec;

/\* Initialize the board.

\* The board-specific conf\_board.h file contains the configuration of

\* the board initialization.

\*/

board\_init();

sysclk\_init();

// USART options.

static usart\_rs232\_options\_t USART\_SERIAL\_OPTIONS = {

.baudrate = USART\_SERIAL\_EXAMPLE\_BAUDRATE,

.charlength = USART\_SERIAL\_CHAR\_LENGTH,

.paritytype = USART\_SERIAL\_PARITY,

.stopbits = USART\_SERIAL\_STOP\_BIT

};

// Initialize usart driver in RS232 mode

usart\_init\_rs232(USART\_SERIAL\_EXAMPLE, &USART\_SERIAL\_OPTIONS);

// This loop runs forever

while (true) {

// Receive characters asynchronously from Tera Term

receivedByte\_Asc = usart\_getchar(USART\_SERIAL\_EXAMPLE);

// Convert ASCII number to an integer

convertedNum\_prev = ConvertToInteger(receivedByte\_Asc);

// If the number is between 0 and 9 then enter the next loop

if (convertedNum\_prev >= 0 && convertedNum\_prev <= 9) {

while(true) {

// Receive characters asynchronously from Tera Term

receivedByte\_Asc = usart\_getchar(USART\_SERIAL\_EXAMPLE);

// Convert ASCII number to an integer

convertedNum\_rec = ConvertToInt(receivedByte\_Asc);

// Add the two numbers

return\_num = convertedNum\_rec + convertedNum\_prev;

// Display the number on the LCD screen

DisplayNumber(return\_num);

// After the number has been displayed break

// out of second loop and enter the top level loop

break;

}

}

}

}

**Part 2 Source code**

/\* Lab 3 Part 2 \*/

#include <board.h>

#include <sysclk.h>

#include <st7565r.h>

#include <gfx\_mono.h>

#include <gfx\_mono\_text.h>

#include <sysfont.h>

#include <gpio.h>

static void ClearLcdScreen(void) {

// This function clears the screen by writing 0x00 over

// and over again which turns off every pixel

for (int page\_address = 0; page\_address <= 4; page\_address++) {

st7565r\_set\_page\_address(page\_address);

for (int column\_address = 0; column\_address < 128; column\_address++) {

st7565r\_set\_column\_address(column\_address);

st7565r\_write\_data(0x00);

}

}

}

static void InitializeLcdScreen(void) {

// Initialize the board and clock

board\_init();

sysclk\_init();

// Initialize the interface (SPI), ST7565R LCD controller and LCD

st7565r\_init();

// Set addresses at beginning of display

st7565r\_set\_page\_address(0);

st7565r\_set\_column\_address(0);

// Initialize gfx functionality

gfx\_mono\_init();

// Turn on backlight

gpio\_set\_pin\_high(LCD\_BACKLIGHT\_ENABLE\_PIN);

ClearLcdScreen();

}

static void DrawWord(char \*word, int wordLength) {

// The starting location is (10,10)

int x = 10;

int y = 10;

for (int i = 0; i < wordLength; i++) {

gfx\_mono\_draw\_char(\*word, x, y, &sysfont);

// Increment the X coord

x += 7;

// Increment the pointer so that it points to the next char

word = word + 1;

}

}

int main(void)

{

InitializeLcdScreen();

// Declare a the word to display

char word[5] = {'H','e','l','l','o'};

// Display the word to the LCD screen

DrawWord(word, 5);

}

**Part 3 Source code**

/\* Lab 3 Part 3 \*/

#include <board.h>

#include <sysclk.h>

#include <conf\_usart\_example.h>

#include <st7565r.h>

#include <gfx\_mono.h>

#include <gfx\_mono\_text.h>

#include <sysfont.h>

#include <gpio.h>

static void ClearLcdScreen(void) {

// This function clears the screen by writing 0x00 over

// and over again

for (int page\_address = 0; page\_address <= 4; page\_address++) {

st7565r\_set\_page\_address(page\_address);

for (int column\_address = 0; column\_address < 128; column\_address++) {

st7565r\_set\_column\_address(column\_address);

st7565r\_write\_data(0x00);

}

}

}

static void FlashScreen(void) {

// Turn the backlight off

gpio\_set\_pin\_low(LCD\_BACKLIGHT\_ENABLE\_PIN);

// Delay for 4 milliseconds

delay\_ms(4);

// Turn the backlight back on

gpio\_set\_pin\_high(LCD\_BACKLIGHT\_ENABLE\_PIN);

}

static void InitializeLcdScreen(void) {

// initialize the board and clock

board\_init();

sysclk\_init();

// Initialize the interface (SPI), ST7565R LCD controller and LCD

st7565r\_init();

// Set addresses at beginning of display

st7565r\_set\_page\_address(0);

st7565r\_set\_column\_address(0);

// Initialize gfx functionality

gfx\_mono\_init();

// Turn on backlight

gpio\_set\_pin\_high(LCD\_BACKLIGHT\_ENABLE\_PIN);

ClearLcdScreen();

}

int main(void)

{

char receivedByte\_Ascii;

InitializeLcdScreen();

static usart\_rs232\_options\_t USART\_SERIAL\_OPTIONS = {

.baudrate = USART\_SERIAL\_EXAMPLE\_BAUDRATE,

.charlength = USART\_SERIAL\_CHAR\_LENGTH,

.paritytype = USART\_SERIAL\_PARITY,

.stopbits = USART\_SERIAL\_STOP\_BIT

};

// Initialize usart driver in RS232 mode

usart\_init\_rs232(USART\_SERIAL\_EXAMPLE, &USART\_SERIAL\_OPTIONS);

// The first location for the text to appear

// is (0,0) or the top left of the LCD screen

int x = 0;

int y = 0;

while (true) {

// Asynchronously receive ASCII character

receivedByte\_Ascii = usart\_getchar(USART\_SERIAL\_EXAMPLE);

// Flash the screen once after a character is received

FlashScreen();

// If the enter key was pressed then clear the screen

// and put the location for text back to (0,0)

if (receivedByte\_Ascii == '\r') {

ClearLcdScreen();

x=0;

y=0;

}

// If any other key was pressed then display it

else {

gfx\_mono\_draw\_char(receivedByte\_Ascii, x, y, &sysfont);

// Then increment x so that there is room

//for the next character

x += 7;

}

// The screen only fits 18 characters

// So if x gets above 18\*7 then go to the next line

// by incrementing y

if ( x == 18 \* 7)

{

x = 0;

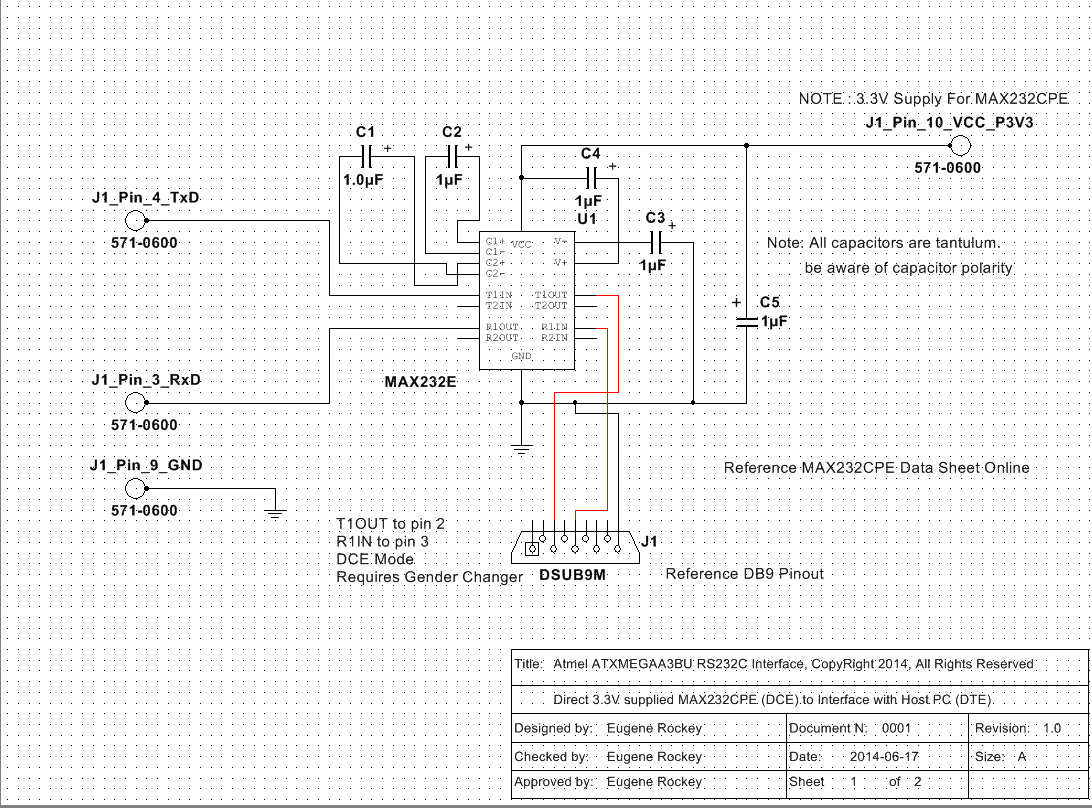
y += 8;

}

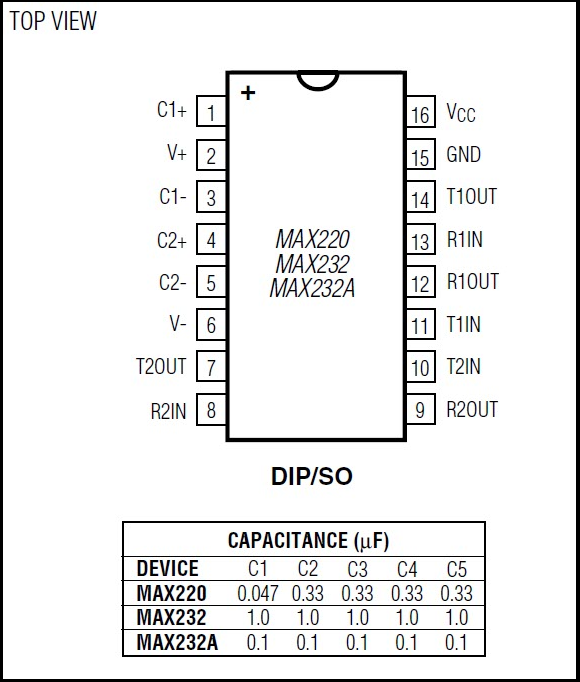
}

}

Schematics (Hardware)



This is a schematic of the circuit used to create the MAX232CPE Serial interface



This is a schematic of the chip used in the circuit above.

Analysis

In lab 3 several new capabilities of the A3BU were demonstrated. This lab explored the concepts of Asynchronous and Synchronous serial I/O communication with a PC terminal. The lab also explored communication techniques of SPI with the LCD module. When working with microcontrollers in the field, communication between devices is incredibly important. Systems need to fine tune the type of communication that is used based on the resources, money, and time that are available. This lab introduced a lot of the types of communication and how they work. This knowledge is very helpful in understanding and designing communication frameworks for large embedded systems.

Part 1 introduces asynchronous communication which uses Tx and Rx lines. Through this type of communication there is no control over when the data is sent. As a result, there is no guarantee that both sides of the communication are running at the same rate. This makes the possibilities of errors and packets being lost greater. Part 2 introduces synchronous communication which uses the system clock for timing. This ensures that both sides of the communication are running at the same rate. Part 3 combines both types of communication. Asynchronous communication is used for receiving characters and synchronous communication is used to write these characters on the LCD.

An example of asynchronous communication is a graphical user interface (GUI). A GUI does not know when an input is going to be received so it needs to use asynchronous communication to account for this occurrence. An example of synchronous communication is a phone call when data is needed to be transmitted in real time.

Conclusion

The main purpose of lab 3 was to study I/O communications through implemented drivers written in C. This was accomplished through the Universal Synchronous Asynchronous Receiver Transmitted (USART). The USART is an I/O Peripheral device, found onboard the A3BU, that allows for serial I/O communication. An RS232C serial interface circuit was constructed to allow for asynchronous communication between the A3BU board and a PC running terminal communication software. Next, LCD synchronous communications was used through the PORT D Serial Peripheral Interface (SPI) and the LCD ST7565R MCU. Finally, both parts were combined into one program that receives ASCII characters from the terminal communication software to be displayed on the LCD. Through the successful completion of this lab, basic I/O communication techniques were understood.

References

1. "The USART of the AVR » MaxEmbedded." *MaxEmbedded*. N.p., 04 Nov. 2015. Web. 23 June 2017.