Project 2 Individual Report

Andrew Leise

Project Description

Throughout the implementation of the second CpE 3150 project, I worked on the code for the serial communications between the terminal and the ATmega342PB. For the overall team project, this corresponded to displaying the current song that is being played, as well as the current mode (tune or keyboard) that the program is operating in. Thus, I created all of the serial transfer and receive functions, as well as a few extra additional functions for handling the output formatting. The first function that was created was the USART_Init function. This function takes in one parameter, a specified baud rate, and returns after initializing the corresponding USART registers for serial communication. See Figure 1 below.

```
// Function implmemtation for USART_Init
void USART_Init(unsigned long BAUDRATE)
{
	// Calculate baud prescaler value and set low/high bytes of baud rate register
	int BAUD_PRESCALER = (F_CPU / (BAUDRATE * 16UL)) - 1;
	UBRR1L = BAUD_PRESCALER;
	UBRR1H = (BAUD_PRESCALER >> 8);

	// Enable transmitter and receiver in status control register B
	UCSR1B |= (1 << RXEN) | (1 << TXEN);

	// Set bits corresponding to 8-bit data transfer, one stop bit, asychronous
	UCSR1C |= (1 << UCSZ0) | (1 << UCSZ1);
}
```

Figure 1: USART_Init function

The next functions that were created for the normal project interface handled the transmission of characters to the computer. First, the USART_TxChar function transmits a single character to the PC through the serial connection. This function takes the character to be transmitted as its only parameter, and simply sets the UDR1 to the character to be transmitted then returns when the process is complete. See Figure 2 below.

```
// Function implementation for USART_TxChar
void USART_TxChar(char data)
{
	// Set data to UDR1 register, then loop while data is being transferred
	UDR1 = data;
	while(!(UCSR1A & (1 << UDRE)));
}
```

Figure 2: USART_TxChar function

In addition to the USART_TxChar function, another function related to transmitting characters, USART_TxString was defined and created. This function takes in a character array (pointer) as the parameter and simply calls the USART_TxChar function for each character until the string is empty. This allowed us to send strings at a time to the interface without numerous lines of code for the process. See Figure 3 below.

```
// Function implementation for USART_TxString
void USART_TxString(char *str)
{
	// Loop while characters still exist in the string, and transmit each character individually
	while(*str != '\0')
	{
		USART_TxChar(*str++);
	}
}
```

Figure 3: USART_TxString function

Individual Project

In addition to the main serial communication functions, I helped form the main logic for the program with the assistance of my additional individual feature. My individual feature consisted of handling the working menu operations on the terminal serially, as well as displaying the characters being played on screen. The original grading document lists the serial menu as part of the normal project, but it also appears on the suggested individual projects, so I will discuss everything related to that here. It should be noted that much of this is used in the normal project program as well.

To create the menu and selection options serially, multiple new functions had to be implemented. The first of these was the USART_RxChar function, which returns a received character. It simply checks the value of RxC and if it equals 1, returns UDR1. This returns the character in the buffer if one has been received. See Figure 4 below.

```
// Function implementation for UART_RxChar
char USART_RxChar()
{
        // Check to see if character has been received
        if(UCSR1A & (1 << RXC))
        {
                // If char is received, return it
                return UDR1;
        }
        else
        {
                // If no char is received, return null
                return '\0';
        }
}
```

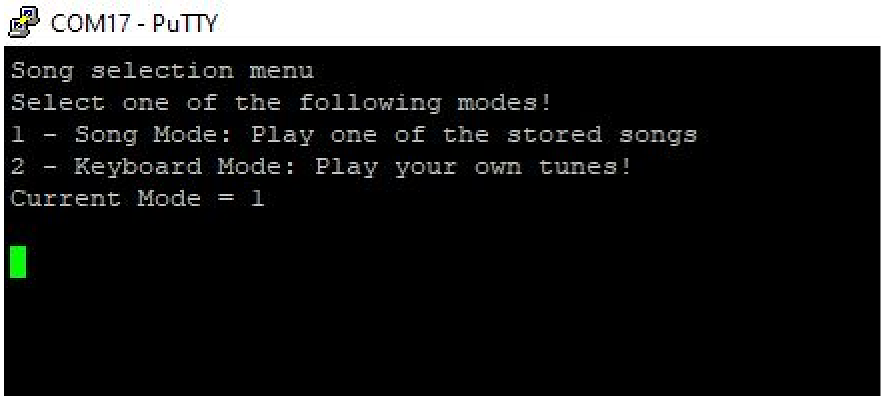
Figure 4: USART_RxChar function

Next, the ISR that executes when a character is received was implemented. The receiving functionality of the program was handled with an interrupt, both for ease of programming and to allow a user to change the mode in the middle of a tune. When a character is received in the buffer, this ISR is called. Inside the ISR, the received character is first determined. The only input that matters for this project are the numbers 1 and 2, as they correspond to the two modes of the program. If the user types 1 on the keyboard, the mode should change to mode 1 or the tune/song mode. If the user types 2 on the keyboard, the mode should change to mode 2 or the keyboard mode. Thus, the logic for handling that appears in this ISR. If a ‘1’ is received, the mode variable is set to 0 (meaning mode 1). If a ‘2’ is received, the mode variable is set to 1 (meaning mode 2). If any other value is found in the receive buffer, nothing happens. Lastly, after changing the mode, the new menu is displayed. This will be explained further below. It should also be noted that the current mode is also checked with the value passed. This is to ensure the menu is not updating when the mode did not change. This can be seen in the “if (mode != 0)” check after determining the received value to be ‘1’. The full routine appears below in Figure 5.

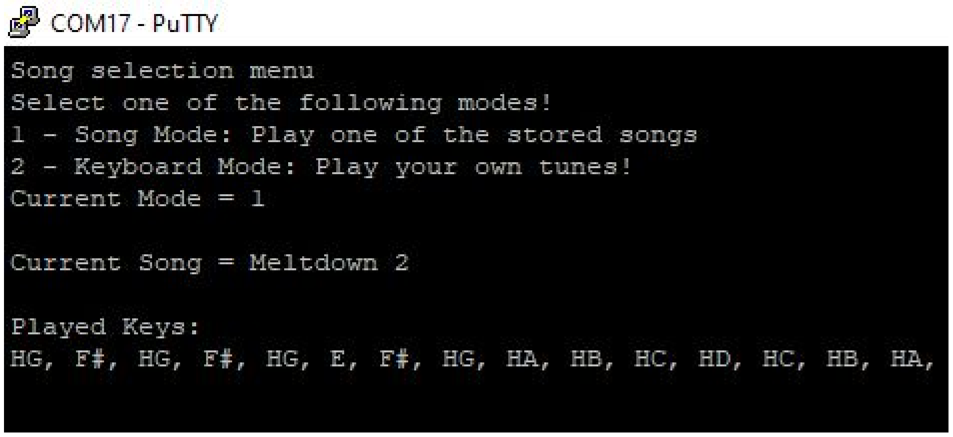
```
// Interrupt service routine implementation
ISR(USART1_RX_vect)
{
        // Set character to received value
        unsigned char val = UDR1;

        // If a 1 was received, change to mode 0 (if not already in mode 0) and output new menu
        if(val == '1')
        {
                if(mode != 0)
                {
                        mode = 0;
                        Display_Menu(mode, "");
                }
        }
        // If a 2 was received, change to mode 1 (if not already in mode 1) and output new menu
        else if(val == '2')
        {
                if(mode != 1)
                {
                                mode = 1;
                                Display_Menu(mode, "");
                }
        }
}
```

Figure 5: ISR for receiving a character

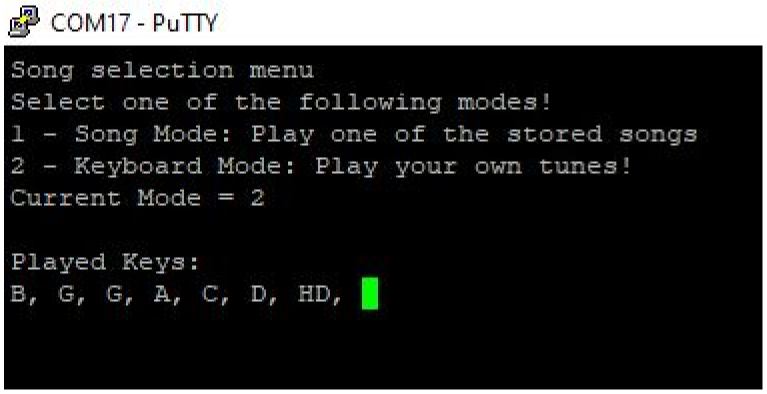
The last main function that was utilized in this system was the Display_Menu function. It takes two parameters, those being the selected mode and the current song being played (if there is one). The terminal used in this application was PuTTY, thus some commands specific to it were utilized. First, the terminal is cleared by writing to it using the specific command. Then the cursor is moved back to the first position by another PuTTY command. Now the menu can be properly displayed. First, a title is transmitted with newline characters to ensure proper output. The menu is then displayed. See Figure 6 below for a sample image of the menu with no current song playing.



```
Song selection menu
Select one of the following modes!
1 - Song Mode: Play one of the stored songs
2 - Keyboard Mode: Play your own tunes!
Current Mode = 1
```



```
Song selection menu
Select one of the following modes!
1 - Song Mode: Play one of the stored songs
2 - Keyboard Mode: Play your own tunes!
Current Mode = 1

Current Song = Meltdown 2

Played Keys:
HG, F#, HG, F#, HG, E, F#, HG, HA, HB, HC, HD, HC, HB, HA,
```



```
Song selection menu
Select one of the following modes!
1 - Song Mode: Play one of the stored songs
2 - Keyboard Mode: Play your own tunes!
Current Mode = 2

Played Keys:
B, G, G, A, C, D, HD,
```

Figure 6: Sample menu outputs

After the menu is displayed, logic statements to determine what else to display is executed. This consists of checking the current mode and determining if the passed string, ‘song’, is empty. If the current mode is mode 1 and no song is being played at the moment, then it just adds the current mode to the display. If the current mode is mode 1 and a song is being played, then the mode and song title are displayed. The header for the notes played is also displayed. If the current mode is mode 2, then the current mode and header for the notes played are displayed. See Figure 7 below for the full documentation.

```
// Function implementation for Display_Menu
void Display_Menu(unsigned char selected_mode, char* song)
{
	// Clear terminal by calling PuTTY specific command
	USART_TxChar(27);
	USART_TxString("[2J");
	USART_TxChar(27);
	USART_TxString("[H");

	// Output new menu to the terminal
	USART_TxString("Song selection menu\r\n");
	USART_TxString("Select one of the following modes!\r\n");
	USART_TxString("1 - Song Mode: Play one of the stored songs\r\n");
	USART_TxString("2 - Keyboard Mode: Play your own tunes!\r\n");

	// Display the current mode and song based on passed values
	// Song mode
	if (selected_mode == 0)
	{
		// No song is passed
		if (song[0] == '\0')
		{
			USART_TxString("Current Mode = 1\r\n\n");
		}
		Else // Song is being played
		{
			USART_TxString("Current Mode = 1\r\n\n");
			USART_TxString("Current Song = ");
			USART_TxString(song);
			USART_TxString("\r\n\n");
			USART_TxString("Played Keys:\r\n");
		}
	}
	// Keyboard mode
	else if(selected_mode == 1)
	{
		USART_TxString("Current Mode = 2\r\n\n");
		USART_TxString("Played Keys:\r\n");
	}
}
```

Figure 7: Display_Menu function

To handle the displaying of the notes being played, the USART_TxString function is called for each specific note. It should also be mentioned that USART_TxString is used here because some notes are longer than one character (A# for example). When displaying the notes, H and L were appended to the front to refer to high and low notes. The # character was appended to the end of notes to refer to sharp notes. So, LA# refers to a low A sharp note. As these functions are used in numerous different places, the specific code portions where they are used will not be pasted here to avoid confusion. Thus, please refer to the total program code pasted in the main report or the code files themselves to see all of the places these are utilized. For example, to display each note, the USART_TxString function was utilized in the Note function written by Tim, as well as the various if-conditions used for the keyboard written by Ben. Additionally, the Display_Menu function is used in multiple places in the main function to display the current song.

ATmega324PB

During the implementation of my individual portion of Project 2, I primarily utilized the registers, functions, and interrupts correlating to the serial communication functionality of the ATmega324PB. The registers that were utilized consisted of the following: UBRR1L, UBRR1H, UCSR1A, UCSR1B, UCSR1C, and UDR1. These are thoroughly described below.

The UBRR1H and UBRR1L registers were used when defining the specified baud rate of the transmissions. These were both utilized in the USART_Init function when setting up the serial communications. The desired baud rate is calculated during runtime based on a constant, BAUD_PRESCALER, that can be defined in the code. For this program, the baud rate was set to 9600, so the value passed to these registers is calculated by the following:

$$X = \frac{f}{BR*16} - 1 = \frac{16x10^6}{9600*16} - 1 = 103.167 = 103$$

Thus, the value passed to these registers is 103 and 103 shifted to the right by 8 bits (divided by 256). This sets the baud rate for the microcontroller correctly.

The USCR1A was primarily used for determining if a character has been received or checking to ensure a character has been fully transmitted. These occur in USART_RxChar and USART_TxChar respectively.

The USCR1B and USCR1C registers were both used in the serial communications initialization, the USART_Init function. USCR1B was used to enable the transmitter and receiver. USCR1C was used to set the serial port for 8-bit data transfer, one stop bit, asynchronous communications.

The UDR1 register was used when gathering a received value or transmitting a value. This was utilized in the USART_RxChar and USART_TxChar functions.

In addition to the different registers that were utilized, the interrupt for receiving characters was utilized. The USART_RX interrupt triggers when a character is received in the buffer and this was used to handle the menu operations upon receiving input from the keyboard.

Overall Contributions

Throughout the implementation of project 2, I contributed all of the code relating to the serial communications and terminal displays. This includes all of the functions defined below, as well as their different uses throughout the rest of the program. I also contributed to the main logic of the program through the use of the terminal menu. The mode variable is set and altered in the ISR for the received character, so much of the program relies on these functions. My contributions for the team and individual projects are fairly mixed as much of what I created is used throughout most of the program, however, I've tried to distinguish them as best as I can. My normal team contributions consisted of the logic for the main program, the code for sending/receiving data, and displaying the menu. My individual project contributions consisted of the menu operation through the keyboard and displaying the notes played on the terminal for both the tune and keyboard modes.

Commented Code

As stated before, this code consists of the main functions written for the individual project. The actual functions themselves are utilized extensively throughout the program, so every call of each function will not appear here. Reference the team report's commented code for that information.

```
// Function implmemtation for USART_Init
void USART_Init(unsigned long BAUDRATE)
{
		// Calculate baud prescaler value and set low/high bytes of baud rate register
		int BAUD_PRESCALER = (F_CPU / (BAUDRATE * 16UL)) - 1;
		UBRR1L = BAUD_PRESCALER;
		UBRR1H = (BAUD_PRESCALER >> 8);

		// Enable transmitter and receiver in status control register B
		UCSR1B |= (1 << RXEN) | (1 << TXEN);

		// Set bits corresponding to 8-bit data transfer, one stop bit, asychronous
		UCSR1C |= (1 << UCSZ0) | (1 << UCSZ1);
}

// Function implementation for UART_RxChar
char USART_RxChar()
{
		// Check to see if character has been received
		if(UCSR1A & (1 << RXC))
		{
				// If char is received, return it
				return UDR1;
		}
		else
		{
			// If no char is received, return null
			return '\0';
		}
}

// Function implementation for USART_TxChar
void USART_TxChar(char data)
{
		// Set data to UDR1 register, then loop while data is being transferred
		UDR1 = data;
		while(!(UCSR1A & (1 << UDRE)));
}




// Function implementation for USART_TxString
void USART_TxString(char *str)
{
		// Loop while characters still exist in the string, and transmit each character individually
		while(*str != '\0')
		{
			USART_TxChar(*str++);
		}
}

// Interrupt service routine implementation
ISR(USART1_RX_vect)
{
		// Set character to received value
		unsigned char val = UDR1;

		// If a 1 was received, change to mode 0 (if not already in mode 0) and output new menu
		if(val == '1')
		{
			if(mode != 0)
			{
				mode = 0;
				Display_Menu(mode, "");
			}
		}
		// If a 2 was received, change to mode 1 (if not already in mode 1) and output new menu
		else if(val == '2')
		{
			if(mode != 1)
			{
				mode = 1;
				Display_Menu(mode, "");
			}
```

```
        }
}

// Function implementation for Display_Menu
void Display_Menu(unsigned char selected_mode, char* song)
{
        // Clear terminal by calling PuTTY specific command
        USART_TxChar(27);
        USART_TxString("[2J");
        USART_TxChar(27);
        USART_TxString("[H");

        // Output new menu to the terminal
        USART_TxString("Song selection menu\r\n");
        USART_TxString("Select one of the following modes!\r\n");
        USART_TxString("1 - Song Mode: Play one of the stored songs\r\n");
        USART_TxString("2 - Keyboard Mode: Play your own tunes!\r\n");

        // Display the current mode and song based on passed values
        // Song mode
        if (selected_mode == 0)
        {
                // No song is passed
                if (song[0] == '\0')
                {
                        USART_TxString("Current Mode = 1\r\n\n");
                }
                // Song is being played
                else
                {
                        USART_TxString("Current Mode = 1\r\n\n");
                        USART_TxString("Current Song = ");
                        USART_TxString(song);
                        USART_TxString("\r\n\n");
                        USART_TxString("Played Keys:\r\n");
                }
        }
        // Keyboard mode
        else if(selected_mode == 1)
        {
                USART_TxString("Current Mode = 2\r\n\n");
                USART_TxString("Played Keys:\r\n");
        }
}
```