Ethan Nagelvoort
821234668

Lab 6 report

Video:
https://drive.google.com/file/d/1JpMFCrfzEXQnEtRLcaOmZcUzUCCbqKwl/view?usp=sharing

Description:

In this lab, we were supposed to have our keyboard generate different notes of frequencies. To do this, I set all the columns of the keyboard to inputs using the PortB and all the rows of the keyboard to outputs using PortD. I also set pin C1 as output because I used this pin to connect the Aux to the AVR board. I also set my timer to CTC mode as well. Inside the infinite while loop, I declare a 2D int array that is called key. This will represent every button on the keyboard as an int with button 1 being int 0 and button D being int 15. Then for the following code, I used the same process as the keyboard scan function from lab 4, where you use 2 for loops, one for row and one for column to check if the button is being pressed. Then if the button is pressed, the program will go through the function keypad with that button as the parameter. In the keypad function, I use a switch statement to show what would happen for each button being pressed. Each button will be represented in a case and if that button is pressed, the function will go to that case. In each case, I set the OCR0A value to represent the frequency of that button. I calculated the frequencies of each button through the equation presented in the lab 6 description/slides. Then I set the OCR0B value to represent 50% duty cycle, which will just be OCR0A * (½). I then set the prescaler for the button that I evaluated through calculation as well. I do this by manipulating the TCCR0B register. Cases 0 through 13 will use a prescaler of 256 but cases 14 and 15 will use a prescaler of 64. Then in each case, I turn C1 on and use a busy while loop to wait for the overflow event of OCR0B. I then reset the overflow flag for OCR0B. This process of using this busy while loop and resetting the overflow flag checks if the timer finishes. Then I turn C1off and do the whole process again but with OCR0A. This process will utilize the overflow process for the whole cycle. Then I use a break statement to end the case and leave the switch statement.

Result:

After hooking up my keypad to my AVR and hooking up the Aux to my AVR, I plugged my headphones into the Aux. Then I soon pressed the button on the keypad which generated sound at different frequencies. Hence, all the buttons generated slightly different sounds from one another.

Source code:

```
/*
 * LAB6.c
 *
 * Created: 3/20/2020 10:49:34 PM
```

```
 * Author : Ethan
 */

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

#define BAUD 9600
#define ON PORTC |= (1<<1);//
#define OFF PORTC &=~(1<<1);//
#define MYUBBR ((F_CPU)/(BAUD * 16UL) - 1) //Use UBBR for USART baud rate
//void init_uart();
void keypad (int key);

/*void init_uart() {
	UBRR0H = (MYUBBR >> 8); //set UBBR high
	UBRR0L = (MYUBBR); //set UBBR low
	//UCSR0C = (1 << UCSZ00) | (1 << UCSZ01) | (0 << USBS0); // 8-bit 1-stop bit mode
	UCSR0B = (1 << TXEN0); //enable transmitter
}*/

int main(void)
{
   //init_uart();
	DDRB &=~ (1<<0)&~(1<<1)&~(1<<2)&~(1<<3);//inputs
	DDRD |=	(1<<4|1<<5|1<<6|1<<7);//output
	PORTB  |= (1<<0) | (1<<1) | (1<<2) |(1<<3);//pullups
	DDRC|=(1<<1);//set C1 as output
	TCCR0A &= ~(1<<WGM00) &~(1<<WGM02);//set timer to CTC mode
	TCCR0A |=(1<<WGM01);//set timer to CTC mode
   while (1)
   {
		int key[4][4] = {{0, 1, 2, 3 }, {4, 5, 6, 7}, {8, 9, 10, 11}, {12, 13, 14,
15}};//keypad buttons
		PORTD	|=	(1<<4)|(1<<5)|(1<<6)|(1<<7);//set rows to high
		for(int r=4; r<8; r++)//loop through rows
		{
			PORTD &= ~(1<<r);//set this row to low, to check row
			for(int c=0; c<4; c++)//loop through column
```

```
                        {
                                if((PINB & (1<<c))==0)//find key that is pressed
                                {
                                        keypad(key[r-4][c]);//transmit key to GPIO
                                }
                        }
                        PORTD |= (1<<r);//set row back to high
                }

    }
}

void keypad (int key)
{
        switch(key)
        {
                case 0:
                OCR0A = 0x8D; //represents 440MHz
                OCR0B = 0x46;//50% duty cycle
                TCCR0B &= ~(1<<CS01) &~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while(TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 1:
                OCR0A = 0x85; //represents 466.16MHz
                OCR0B = 0x42;//50% duty cycle
                TCCR0B &= ~(1<<CS01) &~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
```

```
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while(TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 2:
                OCR0A = 0x7D; //represents 493.88MHz
                OCR0B = 0x3E;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 3:
                OCR0A = 0x76; //represents 523.3 MHz
                OCR0B = 0x3B;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 4:
                OCR0A = 0x6F; //represents 554.4 MHz
```

```
                OCR0B = 0x37;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 5:
                OCR0A = 0x69; //represents 587.3 MHz
                OCR0B = 0x34;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 6:
                OCR0A = 0x63; //represents 622.3 MHz
                OCR0B = 0x31;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
```

```
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 7:
                OCR0A = 0x5D; //represents 659.25 MHz
                OCR0B = 0x2E;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 8:
                OCR0A = 0x58; //represents 698.46 MHz
                OCR0B = 0x2C;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 9:
                OCR0A = 0x53; //represents 739.99 MHz
                OCR0B = 0x29;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
```

```
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 10:
                OCR0A = 0x4E; //represents 783.99 MHz
                OCR0B = 0x27;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 11:
                OCR0A = 0x4A; //represents 830 MHz
                OCR0B = 0x25;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
```

```
                break;

                case 12:
                OCR0A = 0x46; //represents 800 MHz
                OCR0B = 0x23;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 13:
                OCR0A = 0x42; //represents 932 MHz
                OCR0B = 0x21;//50% duty cycle
                TCCR0B &= ~(1<<CS01) & ~ (1<<CS00);
                TCCR0B |=(1<<CS02);//256 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 14:
                OCR0A = 0xFC; //represents 988 MHz
                OCR0B = 0x7E;//50% duty cycle
                TCCR0B |= (1<<CS01) & ~ (1<<CS00);
                TCCR0B &=~(1<<CS02);//64 prescale
                ON;
```

```
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;

                case 15:
                OCR0A = 0xED; //represents 1047 MHz
                OCR0B = 0x76;//50% duty cycle
                TCCR0B |= (1<<CS01) & ~ (1<<CS00);
                TCCR0B &=~(1<<CS02);//64 prescale
                ON;
                while((TIFR0 & ~(1<<OCF0B))==0){};//busy while loop, wait for OCR0B
overflow event
                TIFR0|= (1<<OCF0B);//reset OCR0B overflow flag
                OFF;
                while( TIFR0 & ~(1<<OCF0A)==0){};//busy while loop, wait for OCR0A
overflow event
                TIFR0|= (1<<OCF0A);//reset OCR0A overflow flag
                break;
        }
}
```