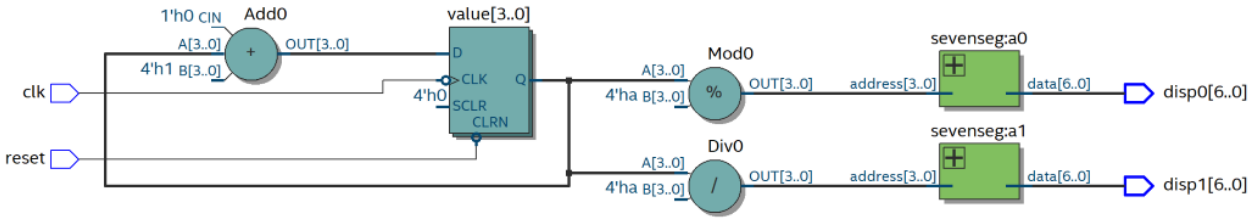
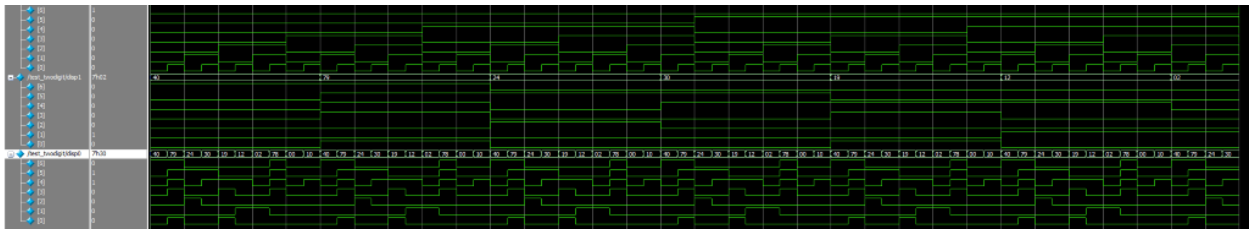
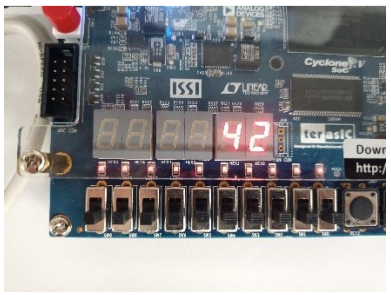
Using the path:
Tools -> Netlist Viewers -> RTL Viewer
I made Quartus create the circuit diagram of a conventional implementation of the device.

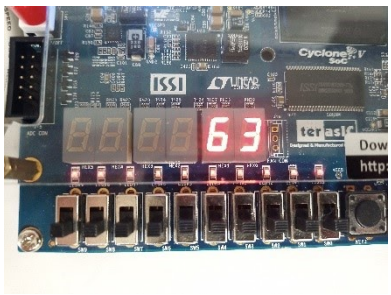


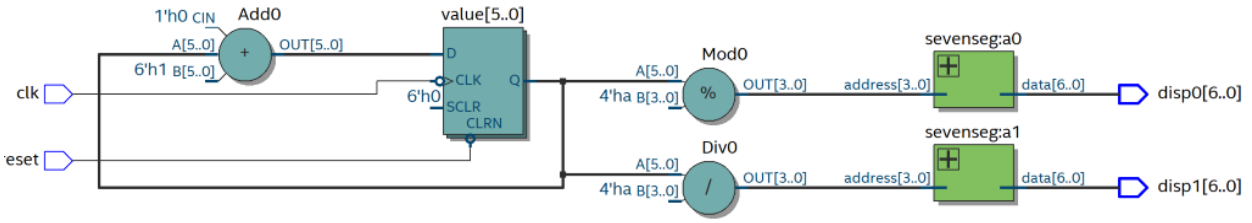
I then modified my code by increasing the size of the array value[3:0] to value[5:0] allowing it to count up to six bits. First I tested it in ModelSim, creating a testbench to do so.



Then I used Quartus to put it on the FPGA and count up to 63. I also made it create the circuit diagram for the counter.







Timers/Counters and Pulse-Width Modulation

8-bit resolution is not enough to cover the audio range with the resolution that would match the musical note. Hence, I will use timer 1, a 16-bit timer.

TTCR1A								
Bit	7	6	5	4	3	2	1	0
(0x80)	COM1A1	COM1A0	COM1B2	COM1B0	-	-	WGM11	WGM10
R/W	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Initial Val	0	0	0	0	0	0	0	0
Set Val	0	1	0	0	0	0	0	1
TTCR1B								
Bit	7	6	5	4	3	2	1	0
(0x81)	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Val	0	0	0	0	0	0	0	0
Set Val	0	0	0	1	0	0	1	0

Selecting a pre-scalar:

$$f_{oc} = \frac{f_{clk\_I/O}}{2 \times N_{max} \times TOP}$$

$$N_{max} = \frac{f_{clk\_I/O}}{2 \times f_{oc} \times TOP}$$

$$N_{max} = \frac{12000000}{2 \times 18000 \times 1}$$

$$N_{max} = 333$$

Most timers pre-scalars {1, 8, 64, 256, or 1024} would work. I will be using a pre-scalar of 8 in my code.

```
32  void init_tone(void)
33  {
34      DDRD |= _BV(PD5);     /* enable ouput driver for OC1A */
35      TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
36             | _BV(WGM10);  /* frequency (f) correct PWM, */
37      TCCR1B = _BV(WGM13)  /* varying f with OCR1A */
38             | _BV(CS11);   /* prescaler set to 8 */
39  }
```

```
void tone(uint16_t frequency)
{
    /*
    We rely on the compiler to substitute the constant part of the expression.
    1/2 for symmetric PWM & 1/2 for toggle output
    */
    OCR1A = (uint16_t)(F_CPU / (2 * 2 * TONE_PRESCALER) / frequency);
}
```

I will be using Timer2 for my amplitude control, as one of the two outputs for Timer0 is used with the usb interface.

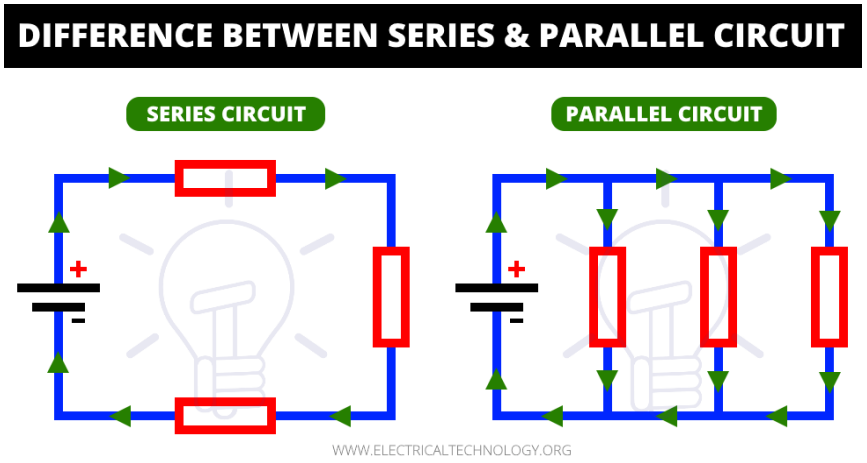
Maximum frequency at 8-bit resolution:

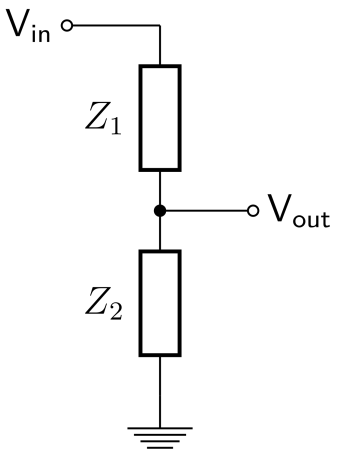
$$f_{oc} = \frac{f_{clk-I/O}}{N \times 256}$$
$$f_{oc} = \frac{12000000}{1 \times 256}$$
$$f_{oc} = 46.9KHz$$

TTCR2A								
Bit	7	6	5	4	3	2	1	0
(0xB0)	COM2A1	COM2A0	COM2B2	COM2B0	-	-	WGM21	WGM20
R/W	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Initial Val	0	0	0	0	0	0	0	0
Set Val	1	0	0	0	0	0	1	1
TTCR2B								
Bit	7	6	5	4	3	2	1	0
(0xB1)	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20
R/W	W	W	R	R	R/W	R/W	R/W	R/W
Initial Val	0	0	0	0	0	0	0	0
Set Val	0	0	0	0	1	0	0	1

```
116  void init_volume(void)
117  {
118      DDRD |= _BV(PD7);     /* enable ouput driver for OC1A */
119      TCCR1A = _BV(COM2A0) /* toggle OC2A on match */
120           | _BV(WGM21);    /* fast PWM, */
121           | _BV(WGM20);    /* fast PWM, */
122      TCCR1B = _BV(WGM22)  /* varying f with OCR2A */
123           | _BV(CS20);     /* prescaler set to 1 */
124  }
```

Signal	Port	Pin Number
Tone	D	5
Volume	D	7





Using the oscilloscope you can use the time base to find the time period of the signal, and then use f = 1/T to find the frequency.

Port Registers ($x \in$ {A,B,C,D})

PINx, PORTx, DDRx	Input, Output and Direction registers

Bit Manipulation ($n \in \{0,1,\ldots,7\}$, $r \in$ {I/O Registers})

uint8_t value;	Declare value as an 8-bit byte
#define _BV(n) (1 << (n))	Bit Value
value = 0xFF;	Set all 8-bits of byte value
value = 0x00;	Clear all 8-bits of byte value
value = ~value;	Invert all bits of byte value
value |= _BV(n);	Set bit n of byte value
value &= ~_BV(n);	Clear bit n of byte value
if bit_is_set(r, n) { ... }	Test if bit n of r is set
if bit_is_clear(r, n) { ... }	Test if bit n of r is clear
loop_until_bit_is_set(r, n);	Wait until bit n of r is set
loop_until_bit_is_clear(r, n);	Wait until bit n of r is clear

Input

DDRx = 0x00;	Set 8-bits of port x as inputs
PORTx = 0xFF;	Enable pull-ups on input port x
PORTx = 0x00;	Configure inputs as tri-state on port x
value = PINx;	Read value of port x
DDRx &= ~_BV(n);	Set bit n of port x as input
PORTx |= _BV(n);	Enable pull-up on bit n of port x
PORTx &= ~_BV(n);	Configure tri-state on bit n of port x
if (PINx & _BV(n)) { ... }	Test value of pin n on port x

Output

DDRx = 0xFF;	Set 8-bits of port x as outputs
PORTx = 0xFF;	Set all output bits on port x high
PORTx = 0x00;	Set all output bits on port x low
PINx = 0xFF;	Toggle all output bits on port x high
DDRx |= _BV(n);	Set bit n of port x as output
PORTx |= _BV(n);	Set bit n of port x high
PORTx &= ~_BV(n);	Set bit n of port x low
PINx |= _BV(n);	Toggle bit n of port x

Input/Output

DDRx = 0x0F;	High 4-bits input, low 4-bits output
PORTx |= 0x0F;	Set all output bits high
PORTx &= 0xF0;	Set all output bits low
value = PINx & 0xF0;	Read input bits on port x