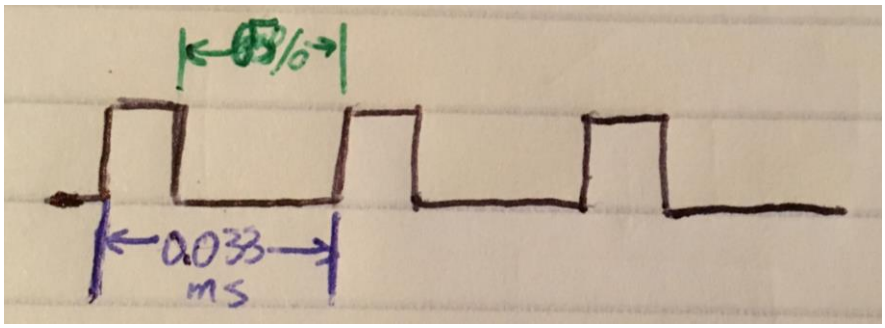
Lab: 6
Nicholas Imamshah
Section: 6957

b) Prelab Questions

1. Draw a 30 kHz square wave with 35% duty cycle. What is the period in ms?
    a. Period = 1/30kHz = 0.0333 ms



2. How does the prescaler affect the way the TC system counts per clock cycle? Where are the counts stored?
    a. The prescaler specifies when to increment the count for a given TC; for example, if the prescaler is set to CLK/64, then every 64th CLK tick will increment the TC's count which is stored in the respective CNT registers (CNTH, CNTL).
3. For part A, what is the limiting factor for the precision of your frequency generation? Can your XMEGA generate some frequency ranges with higher precisions than other frequency ranges? Explain.
    a. The size of the CNT registers is the limiting factor on the precision of the frequency generation. Since we have two 8-bit registers, we get 16 bits dedicated for counting, giving us a maximum count of 65,535. Using the prescaler allows for bigger ranges of frequencies, but also reduces precision since the CNT is only incremented every Nth iteration (N = prescaler value), as result the XMEGA can generate lower frequency ranges with higher precision.
4. Describe the difference(s) between the TC's Frequency Generation mode and its Single/Dual Slope PWM modes. Which mode(s) can be used to emulate the other(s)? How could you make a sine wave or other waveform using your XMEGA by using the timer system? Do you need to add any extra hardware? How can you produce these waveforms without extra hardware?
    a. From an implementation standpoint, the TC's FRQ mode uses CCA for match conditions, while the two slope modes use PER. The Single-slope PWM mode could be used to emulate the FRQ mode and vice versa, since each case counts up to a value, then resets to BOTTOM. FRQ mode could also be used to emulate Dual-slope PWM if on each match condition with CNT == CCA, the direction is changed to then count down.

b. To make a Sine wave or similar waveform using the XMEGA timer system, we do not need any extra hardware, we would need a table of sine values that could be iterated through while scaling up/down PWM duty cycle in accordance.

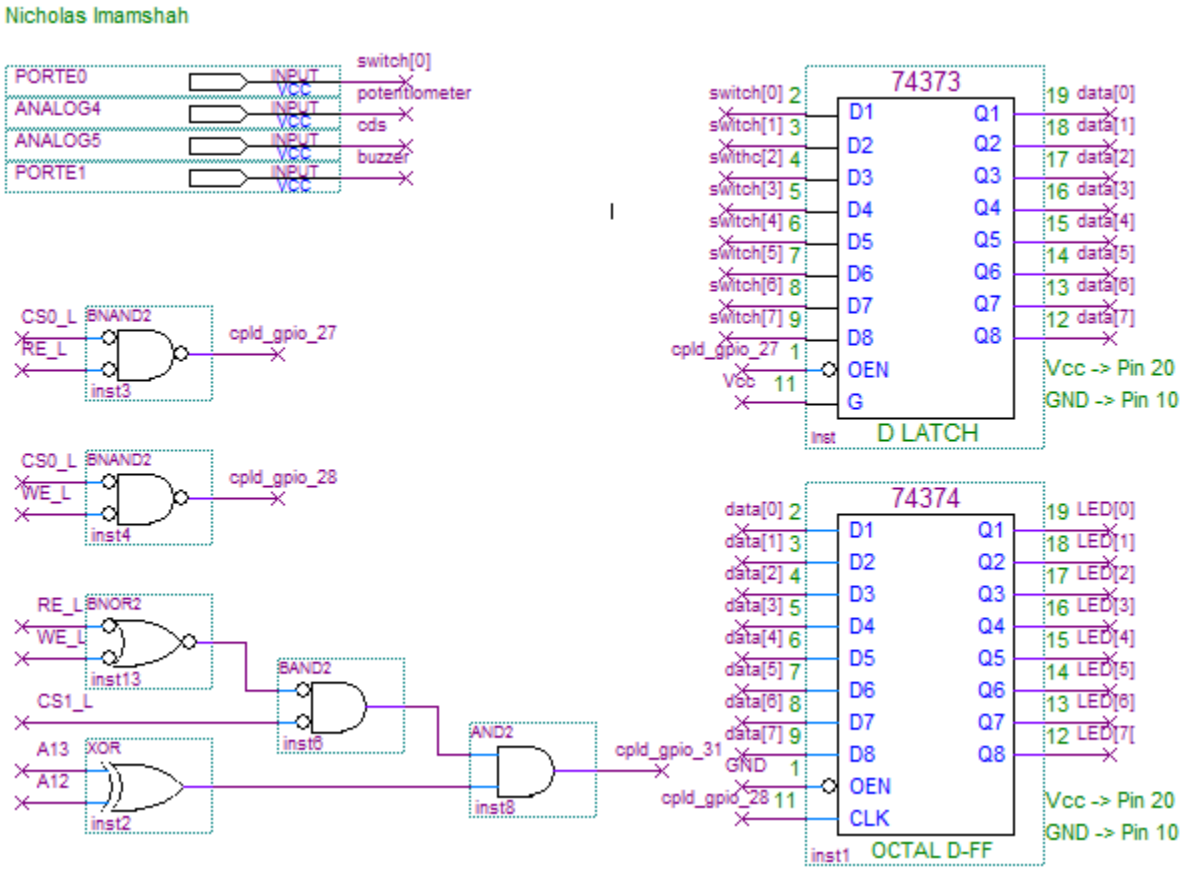
c) Problems Encountered

The most time consuming error I encountered in this lab was figuring out how to set the period for note durations, since the timers are only 16-bits and set millisecond durations. Eventually, I realized I needed to add a constant 1000 denominator to provide a range that would span into seconds.

d) Future Work/Applications

Using the timer/counter system to output a PWM could be used to control a motor/servo in designing a robot or similar mechanism.

e) Schematics



f) Decoding Logic

No new additions.

g) Pseudocode/Flowcharts

Part A

Inits
While (1)
    If (switch TRUE)
        Play(C6)
    Else
        // do nothing
END While

Part B

Inits
While (1)

Key = keyscan()

If key in 1 to D

Play(key)

Else if key == *

Play_sequence(song1)

Else

Play_sequence(song2)

END While

h) Program Code

Part A

```
/* Lab6_PartA.c
 *
 * Lab 6 Part A
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to generate the C6 note using the XMEGA's TC system.
 */
////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>
#include "ebi_init.h"
#include "ebi_driver.h"

////////////////////////////////INITIALIZATIONS/////////////////////////////////////
#define F_CPU 2000000

////////////////////////////////PROTOTYPES//////////////////////////////////////////
void tc_init(void);
void play(uint16_t freq);

////////////////////////////////MAIN FUNCTION///////////////////////////////////////
int main(void)
{
	ebi_init();
	tc_init();

	while (1)
	{
		play(1046.50);
		if (__far_mem_read(IO_PORT) & 0x01)      // If Switch0 True, play note
		{
			TCE0.CTRLB |= TC0_CCBEN_bm;
		}
		else                                                          // Else, turn off
		{
			TCE0.CTRLB = TC_WGMODE_FRQ_gc;
		}
	}
}

////////////////////////////////FUNCTIONS///////////////////////////////////////////
void tc_init(void)
{
	PORTE.DIRSET = 0x02;                    // Set Port E Pin 2 as output

	TCE0.CTRLA = TC_CLKSEL_DIV1_gc;  // Prescaler: CLK

	TCE0.CTRLB = TC0_CCBEN_bm |             // Enable CCB, FRQ mode
```

```
                                TC_WGMODE_FRQ_gc;
        TCE0.CTRLE = TC_BYTEM_NORMAL_gc;

}

void play(uint16_t freq)
{
        uint16_t cca = (F_CPU / (freq * 2)) - 1; // Uses fFRQ formula from doc8331
        TCE0.CCA = cca;
}
```

Part B

```
/* Lab6_PartB.c
 *
 * Lab 6 Part B
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to provide a keypad interface mapped to various sounds.
 */
////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>
#include <avr/interrupt.h>
#include "ebi_driver.h"
#include "ebi_init.h"
#include "lcd_init.h"
#include "keypad.h"

////////////////////////////////INITIALIZATIONS/////////////////////////////////////
#define F_CPU 2000000
#define SCALE 128000

typedef struct note {
        char *name;
        char *ascii_freq;
        uint16_t freq;
} note;

typedef struct beat {
        note n;
        uint16_t d;
} beat;

#define w 4000
#define h 2000
#define q 1000
#define e 500

#define k0 { "A6", "1760.00 Hz", 1760.00 }
#define k1 { "C6", "1046.50 Hz", 1046.50 }
#define k2 { "C#6/Db6", "1108.73 Hz", 1108.73 }
#define k3 { "D6", "1174.66 Hz", 1174.66 }
#define k4 { "D#6/Eb6", "1244.51 Hz", 1244.51 }
#define k5 { "E6", "1318.51 Hz", 1318.51 }
#define k6 { "F6", "1396.91 Hz", 1396.91 }
#define k7 { "F#6/Gb6", "1479.98 Hz", 1479.98 }
#define k8 { "G6", "1567.98 Hz", 1567.98 }
#define k9 { "G#6/Ab6", "1661.22 Hz", 1661.22 }
#define ka { "A#6/Bb6", "1864.66 Hz", 1864.66 }
#define kb { "B6", "1975.53 Hz", 1975.53 }
#define kc { "C7", "2093.00 Hz", 2093.00 }
#define kd { "C#7/Db7", "2217.46 Hz", 2217.46 }
#define knull { "0", "0", 0 }
#define e7 { "E7", "2637.02 Hz", 2637.02 }
#define d7 { "D7", "2349.32 Hz", 2349.32 }
#define wa { "W", "0", 1 }

note notes[] = { k0, k1, k2, k3, k4 , k5, k6, k7, k8, k9, ka, kb, kc, kd, knull };
```

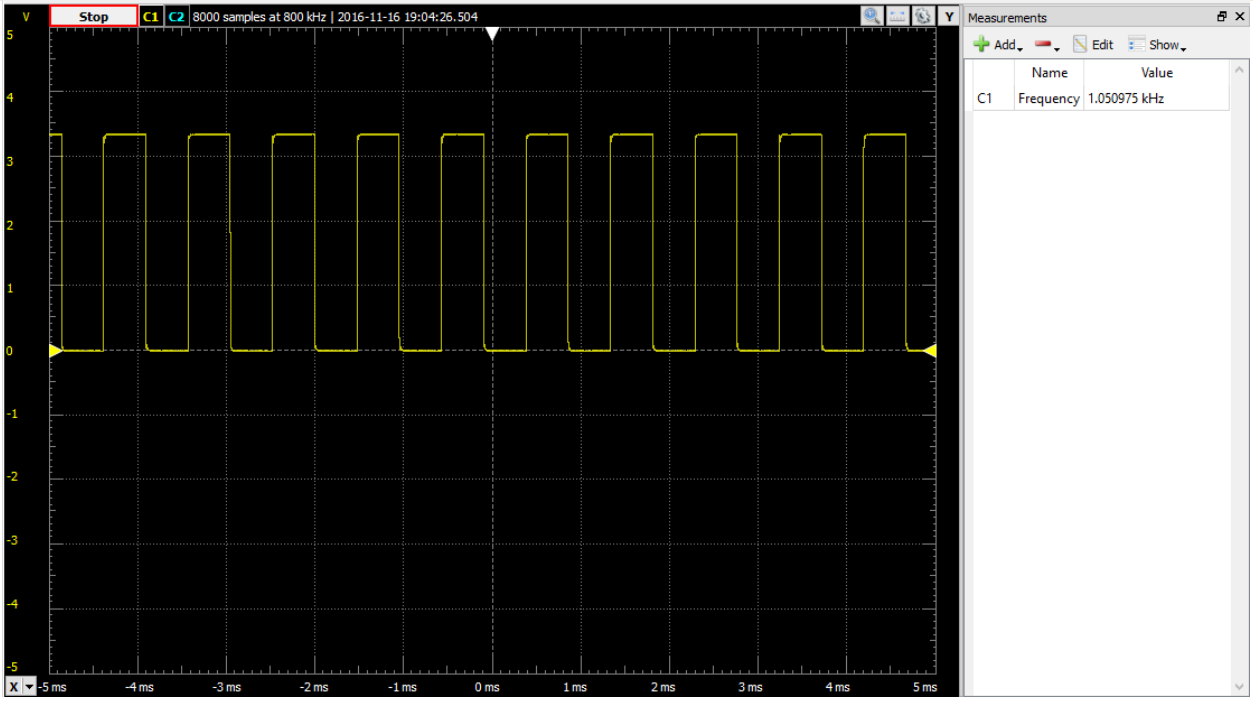
```
beat arp_beats[] =
{
        { k1, q },
        { k5, q },
        { k8, q },
        { kc, q },
        { k8, q },
        { k5, q },
        { k1, q },
        { knull, w }
};

beat lavender[] =
{
        { k1, q },
        { k8, q },
        { kb, q },
        { k7, q },
        { k1, q },
        { k8, q },
        { kb, q },
        { k7, q },
        { k1, q },
        { k8, q },
        { kb, q },
        { k7, q },
        { knull, w }
};

beat green_hill[] =
{
        { kc, e },
        { k0, q },
        { kc, e },
        { kb, q },
        { kc, e },
        { kb, e },
        { kb, e },
        { k8, q },
        { k8, e },
        { k5, e },
        { k8, e },
        { e7, e },
        { d7, q },
        { kc, e },
        { kb, q },
        { kc, e },
        { kb, e },
        { kb, e },
        { k8, q },
        { k8, e },
        { kc, e },
        { k0, q },
        { kc, e },
        { kb, q },
        { kc, e },
        { kb, e },
        { kb, e },
        { k8, q },
        { k8, e },
        { k0, e },
        { k0, e },
        { k6, q },
        { k0, e },
        { k8, q },
        { k0, e },
        { k8, e },
```

```
        { k8, e },
        { k1, q },
        { k1, e },
        { knull, w }
};


////////////////////////////////////PROTOTYPES/////////////////////////////////////
void play(note n);
void play_note(note n, uint16_t d);
void play_sequence(char *name_top, char *name_bottom, beat *b);
void tc_init(void);
uint16_t calc_ffrq(uint16_t freq);
void calc_per(uint16_t period);

////////////////////////////////////MAIN FUNCTION/////////////////////////////////////
int main(void)
{
        ebi_init();
        lcd_init();
        keypad_init();
        tc_init();

        uint8_t key;
        while (1)
        {
                do
                {
                        key = keyscan();        // Get Key press
                } while (key == 0xFF);
                keyhold();
                TCE1.CTRLFSET = TC_CMD_RESTART_gc;

                if (key <= 0x0D)                // Keys 0-D
                {
                        calc_per(567.8);
                        note n = notes[key];
                        play(n);
                }
                else if (key == 0x0E)           // Key *
                {
                        play_sequence("Sonic", "Green Hill Zone", green_hill);
                }
                else                                            // Key #
                {
                        play_sequence("Pokemon", "Lavender Town", lavender);
                }
        }
}

////////////////////////////////////FUNCTIONS/////////////////////////////////////
void play(note n)
{
        CLEAR_SCREEN();

        OUT_STRING(n.name);                                     // Print note and frequency
        OUT_COMMAND(0xC0);
        OUT_STRING(n.ascii_freq);

        TCE0.CCA = calc_ffrq(n.freq);           // Calculate CCA for given note frequency
        TCE0.CTRLB |= TC0_CCBEN_bm;                     // Enable TCs
        TCE1.CTRLB |= TC1_CCAEN_bm;
}

void play_note(note n, uint16_t d)
{
        calc_per(d);                                            // Set PER register for note duration
```

```
        if (n.freq > 1)                                         // If freq < 1, do not play (used for
waits)
        {
                TCE0.CCA = calc_ffrq(n.freq);
                TCE0.CTRLB |= TC0_CCBEN_bm;
        }
        TCE1.CTRLB |= TC1_CCAEN_bm;
}

void play_sequence(char *name_top, char *name_bottom, beat *b)
{
        CLEAR_SCREEN();

        OUT_STRING(name_top);                           // Print name of song selection
        OUT_COMMAND(0xC0);
        OUT_STRING(name_bottom);

        int i = 0;
        while (b[i].n.freq != 0)                // Loop until end of sequence reached (knull)
        {
                TCE1.CTRLFSET = TC_CMD_RESTART_gc;
                play_note(b[i].n, b[i].d);
                while(TCE1.CTRLB & TC1_CCAEN_bm);
                i++;
        }
}

void tc_init(void)
{
        PORTE.DIRSET = 0x12;

        TCE0.CTRLA = TC_CLKSEL_DIV1_gc;                 // Prescaler: CLK
        TCE0.CTRLB = TC_WGMODE_FRQ_gc;                  // FRQ Mode
        TCE0.CTRLE = TC_BYTEM_NORMAL_gc;

        TCE1.CTRLA = TC_CLKSEL_DIV64_gc;        // Prescaler: CLK/64
        TCE1.CTRLB = TC_WGMODE_NORMAL_gc;               // Normal Mode
        TCE1.CTRLE = TC_BYTEM_NORMAL_gc;

        TCE1.INTCTRLA = TC_OVFINTLVL_LO_gc;             // Enable low-level interrupts on overflow
        calc_per(567.8);

        PMIC.CTRL |= PMIC_LOLVLEN_bm;

        sei();
}

uint16_t calc_ffrq(uint16_t freq)
{
        return (F_CPU / (freq * 2)) - 1;        // Formula from Doc 8331.
}

void calc_per(uint16_t period)
{
        TCE1.PER = ((period * F_CPU) / SCALE) - 1;      // Formula for FRQ from Doc 8331,
                                                                                        //
        rearranged to determine PER.
}


//////////////////////////////////////ISRs////////////////////////////////////////
ISR(TCE1_OVF_vect)
{
        TCE1.INTFLAGS = TC1_OVFIF_bm;                   // Clear interrupt flag
        TCE1.CTRLB = TC_WGMODE_NORMAL_gc;               // Disable CCs
        TCE0.CTRLB = TC_WGMODE_FRQ_gc;
}
```

i) Appendix

DAD Frequency Verification



keypad.h

```
/* keypad.c
 *
 * Keypad in C
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to interface the XMEGA processor
 *              with an external Keypad.
 */
////////////////////////////////////INCLUDES/////////////////////////////////////
#include <avr/io.h>

//////////////////////////////////INITIALIZATIONS////////////////////////////////////
#define F_CPU 2000000

uint8_t keys[] = {0x1, 0x4, 0x7, 0xE, 0x2, 0x5, 0x8, 0x0, 0x3, 0x6, 0x9, 0xF, 0xA, 0xB, 0xC, 0xD};

////////////////////////////////////PROTOTYPES/////////////////////////////////////
void keypad_init(void);
uint8_t keyscan(void);
void keyhold(void);

////////////////////////////////////FUNCTIONS/////////////////////////////////////
void keypad_init(void)
{
        PORTF.PIN7CTRL = PORT_OPC_PULLUP_gc;            // Set OPC to Pull-Up for all Keypad pins
        PORTF.PIN6CTRL = PORT_OPC_PULLUP_gc;
        PORTF.PIN5CTRL = PORT_OPC_PULLUP_gc;
        PORTF.PIN4CTRL = PORT_OPC_PULLUP_gc;
        PORTF.PIN3CTRL = PORT_OPC_PULLUP_gc;
        PORTF.PIN2CTRL = PORT_OPC_PULLUP_gc;
        PORTF.PIN1CTRL = PORT_OPC_PULLUP_gc;
```

```
        PORTF.PIN0CTRL = PORT_OPC_PULLUP_gc;

        PORTF.DIRSET = 0x0F;                    // Set LSNibble of PortF as Output
}

uint8_t keyscan(void)
{
        uint8_t input, index, line, key = 0xFF, i = 0;
        for (i = 0; i < 4; i++)  // Iterate columns
        {
                line = ~(0x01 << i) & 0x0F;             // Iterate shift 0x08 by i and not to hit each
col
                PORTF.OUT = line;                       // Output value for col
                asm volatile ("nop");
                input = PORTF.IN & 0xF0; // Read Input and bitmask off Output bits

                if (input < 0xF0)
                {
                        switch (input)
                        {
                                case 0xE0:
                                        index = 0x00;
                                        break;
                                case 0xD0:
                                        index = 0x01;
                                        break;
                                case 0xB0:
                                        index = 0x02;
                                        break;
                                case 0x70:
                                        index = 0x03;
                                        break;
                        }
                        key = keys[index+4*i];
                        return key;
                }
        }
        return key;
}

void keyhold(void)
{
        while ((PORTF.IN & 0xF0) < 0xF0);
}
```

ebi_init.h

```
/* ebi_init.h
 *
 * EBI Initialization Header
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to configure the EBI for I/O Ports and LCD.
 */
////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>

////////////////////////////////////INITIALIZATIONS//////////////////////////////////
#define F_CPU 2000000
#define CS0_Start 0x288000
#define CS0_End  0x289FFF
#define IO_PORT 0x288000
#define CS1_Start 0x394000
#define CS1_End 0x397FFF

////////////////////////////////////PROTOTYPES///////////////////////////////////////
void ebi_init();
```

```
////////////////////////////////////FUNCTIONS/////////////////////////////////////////
void ebi_init()
{
	PORTH.DIR = 0x37;
	PORTH.OUT = 0x33;
	PORTK.DIR = 0xFF;

	EBI.CTRL = EBI_SRMODE_ALE1_gc | EBI_IFMODE_3PORT_gc;

	EBI.CS0.BASEADDRH = (uint8_t) (CS0_Start>>16) & 0xFF;
	EBI.CS0.BASEADDRL = (uint8_t) (CS0_Start>>8) & 0xFF;
	EBI.CS0.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_8KB_gc;

	EBI.CS1.BASEADDR = (uint16_t) (CS1_Start>>8) & 0xFFFF;
	EBI.CS1.CTRLA = EBI_CS_MODE_SRAM_gc | EBI_CS_ASPACE_16KB_gc;
}
```

lcd_init.h

```
/* lcd_init.h
 *
 * LCD Initialization Header
 * Name: Nicholas Imamshah
 * Section: 6957
 * TA Name: Daniel Gonzalez
 * Description: The purpose of this program is to initialize the LCD.
 */
////////////////////////////////////INCLUDES/////////////////////////////////////////
#include <avr/io.h>

////////////////////////////////////INITIALIZATIONS/////////////////////////////////////
#define F_CPU 2000000
#define LCD_BASEADDR 0x395000

////////////////////////////////////PROTOTYPES/////////////////////////////////////////
void lcd_init();
void wait_busy();
void OUT_CHAR(char character);
void OUT_STRING(char *string);
void OUT_COMMAND(char command);
void CLEAR_SCREEN(void);
void lcd_toggle(void);

uint8_t lcd_disp = 0x07;

////////////////////////////////////FUNCTIONS/////////////////////////////////////////
void lcd_init()
{
	wait_busy();
	__far_mem_write(LCD_BASEADDR, 0x38);	// Two lines
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();

	__far_mem_write(LCD_BASEADDR, 0x0F);	// Display on; Cursor on; Blink on
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();

	__far_mem_write(LCD_BASEADDR, 0x01);	// Clear screen; Cursor home
	asm volatile ("nop");
	asm volatile ("nop");
	wait_busy();
}

void wait_busy()
{
	uint8_t result = 0;
	do
```

```
        {
                result = __far_mem_read(LCD_BASEADDR);
        } while (result & 0x80);                        // Poll the BF of the LCD
}

void OUT_CHAR(char character)
{
        wait_busy();
        __far_mem_write(LCD_BASEADDR+1, character);
        asm volatile ("nop");
        asm volatile ("nop");
        wait_busy();
}

void OUT_STRING(char *string)
{
        while (*string != '\0')                                 // Loop until null character is
encountered
        {
                OUT_CHAR(*string);
                string++;
        }
}

void OUT_COMMAND(char command)
{
        wait_busy();
        __far_mem_write(LCD_BASEADDR, command);
        asm volatile ("nop");
        asm volatile ("nop");
        wait_busy();
}

void CLEAR_SCREEN(void)
{
        wait_busy();
        __far_mem_write(LCD_BASEADDR, 0x01);
        asm volatile ("nop");
        asm volatile ("nop");
        wait_busy();
}

void lcd_toggle(void)
{
        lcd_disp = lcd_disp ^ 0x07;
        uint8_t disp_comm = 0x08 | lcd_disp;

        wait_busy();
        OUT_COMMAND(disp_comm);
}
```