```
/*
 * duoshft.c
 *
 * Created: 21-Nov-12 11:39:39
 *  Author: devtech
 */


#include <avr/io.h>
#define  F_CPU              1000000
#include <avr/interrupt.h>
#include <avr/sleep.h>

//define functions
void send_hi(void);
void send_lo(void);
void send_byte(unsigned char byte);
void send_data(unsigned char data, unsigned char digit);
void chip_init(void);
void set_timer();
//
                    //active low data
                    unsigned int data_buffer[11] =
{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0xff};
                    unsigned int digitz[4] ={0x01,0x02,0x04,0x08};
                    unsigned int result = 1009;
                    unsigned int displaybuffer[4];
                    unsigned int digit_count = 0;
```

```
		unsigned int displaybuffer_cnt = 0;
		unsigned int blank_digit = 10;
		unsigned int ones = 0;
		unsigned int tens = 0;
		unsigned int hundreds = 0;
		unsigned int thousands = 0;



//function implementation
////////////////////////////////////
void send_hi()
{
	PORTB |= (1<<PB1); // SER(data) send a one and then clock
	PORTB |= (1<<PB3); // SRCLK (clock)
	PORTB &= ~(1<<PB3);// SRCLK (clock)
}

void send_lo()
{
	PORTB &= ~(1<<PB1); // SER(data) send a zero and then clock
	PORTB |= (1<<PB3); // SRCLK (clock)
	PORTB &= ~(1<<PB3);// SRCLK (clock)
}
////////////////////////////////////
void send_byte(unsigned char byte)
{
	unsigned char shifter =0x80;
	unsigned char result =0;
```

```
	for(char x= 0; x<8;x++)
	{
		result = (byte & shifter);
		if (result ==shifter)
		{
			send_hi();
		}
		else
		{
			send_lo();
		}
		shifter >>= 1;
	}
}
////////////////////////////////////
////////////////////////////////////
void send_data( unsigned char data, unsigned char digit)
{
				send_byte(data);
				send_byte(digit);
				PORTB |= (1<< PB2);  // store to register latch
				PORTB &= ~(1<< PB2); // store to register latch
}
////////////////////////////////////
void display_result(unsigned int result)
{
	if((result >= 0) && (result < 10))
```

```
{
        thousands       = blank_digit;
        hundreds        = blank_digit;
        tens            = blank_digit;
        ones            = result;
        // fill the display buffer
        displaybuffer[0] = data_buffer[thousands];
        displaybuffer[1] = data_buffer[hundreds];
        displaybuffer[2] = data_buffer[tens];
        displaybuffer[3] = data_buffer[ones];
}
else if((result > 9) && (result < 100))
{
        thousands       = blank_digit;
        hundreds        = blank_digit;
        tens            = result / 10;
        ones            = result % 10;

        displaybuffer[0] = data_buffer[thousands];
        displaybuffer[1] = data_buffer[hundreds];
        displaybuffer[2] = data_buffer[tens];
        displaybuffer[3] = data_buffer[ones];
}
else if((result > 99) && (result < 1000))
{
        thousands       = blank_digit;
        hundreds        = result / 100;
        tens            = (result % 100) / 10;
```

```
		ones		= (result % 100) % 10;

		displaybuffer[0] = data_buffer[thousands];
		displaybuffer[1] = data_buffer[hundreds];
		displaybuffer[2] = data_buffer[tens];
		displaybuffer[3] = data_buffer[ones];
	}
	else if((result > 999) && (result < 10000))
	{
		thousands	= result / 1000;
		hundreds	= (result % 1000) / 100;
		tens		= ((result % 1000) % 100) / 10;
		ones		= ((result % 1000) % 100) % 10;

		displaybuffer[0] = data_buffer[thousands];
		displaybuffer[1] = data_buffer[hundreds];
		displaybuffer[2] = data_buffer[tens];
		displaybuffer[3] = data_buffer[ones];
	}

	//return displaybuffer[result];
}

////////////////////////////////////

void set_timer()
{
	TCCR1B |= (1<< WGM12); // Configure timer1 in CTC mode
```

```
	TIMSK1 |= (1<< OCIE1A); // Enable ctc interrupts
	sei (); //Enable global interrupts
	//OCR1A = (78124);
	OCR1A = (60);
	TCCR1B |=(1<< CS10) | (1<< CS11);  // Prescale by 64
}
////////////////////////////////////
ISR(TIMER1_COMPA_vect)
{
		display_result(result);
		send_data (displaybuffer[displaybuffer_cnt], digitz[digit_count]);
		digit_count++; // Increment digit_count
		digit_count &= (3); //  check increament against maximum number of digits
		displaybuffer_cnt++; // Increment display_count
		displaybuffer_cnt &= (3); //  check increament against maximum number of display data
}

////////////////////////////////////
void chip_init()
{
	DDRB |= (0x0e);
	DDRD |= (0x0f);
	PORTB = (0x00);
	PORTD = (0x00);
}
////////////////////////////////////
////////////////////////////////////
int main(void)
```

```
{
        chip_init();
        set_timer();

        while(1)
        {

        }
        return 0;
}
```