Candle Simulator

Generated by Doxygen 1.6.1

Sun Dec 27 16:37:14 2009

Contents

1 File Index

1.1 File List

Here is a list of all files with brief descriptions:

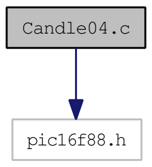
Candle04.c (Simulate a candle using three LEDs)	**1**

2 File Documentation

2.1 Candle04.c File Reference

Simulate a candle using three LEDs. `#include <pic16f88.h>`

Include dependency graph for Candle04.c:



Typedefs

- typedef unsigned int word

  Define an unsigned 16-bit type to be used for configuration bits.

Functions

- void isr ()

  Interrupt Service Routine.

- void Initialize (void)

  Initialization.

- void main ()

  Mainline for the candle simulator.

Variables

- word at CONFIG1

  Configuration word 1.

- word at CONFIG2

  Configuration word 2.

- static int Target

  Select pattern.

- static const unsigned char Patterns [32]

  Array of LED patterns.

2.1.1 Detailed Description

Simulate a candle using three LEDs. Program flashes the 3 PIC-EL LEDs in an erratic fashion.

This version uses the PIC16F88 with its internal oscillator set to 31.25 kHz. The LEDs, instead of being more or less randomly toggled, are fed from an array of allowable patterns, each of which ensures at least one LED is on at all times.

Ultimately, it is expected that the application will be ported to a board with transistors driving the LEDs, capacitors to soften the on-off flashing, and run from 3 volts.

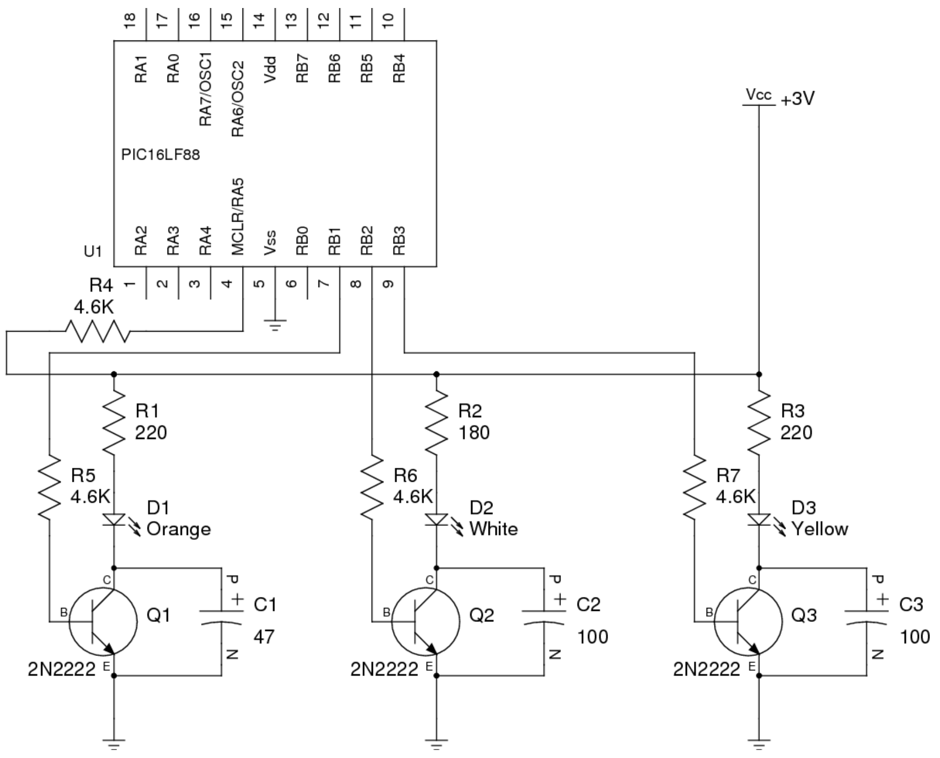


Figure 1: Target Board Schematic

The presumption is that the application is powered by a battery, so current consumption is an issue. The low clock speed ensures that the current consumed by the PIC is minimal.

Date:

2009-11-26 - Initial version
2009-12-27 - Latest revision

Author:

John J. McDonough, WB8RCR

Definition in file Candle04.c.

2.1.2 Typedef Documentation

2.1.2.1 typedef unsigned int word

Define an unsigned 16-bit type to be used for configuration bits.

Definition at line 37 of file Candle04.c.

2.1.3 Function Documentation

2.1.3.1 void Initialize (void)

Initialization. `Initialize()` sets the internal oscillator clock, sets up the timer and ports.

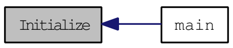
The oscillator is set to 31.25 kHz. The timer will use the internal oscillator with a 1:4 prescaler. `PORTB` is set to all outputs.

Definition at line 120 of file Candle04.c.

```
{
        /* Set the internal clock to 31.25 kHz */
        OSCCON = 0x0e;
        /* Mask all interrupts */
        INTCON = 0;
        /* Enable timer, use rising edge, prescaler to timer, 1:4 */
        OPTION_REG = 0xc1;
        /* PORTB all outputs */
        TRISB = 0;
        /* Just to put bank back to 0 to make asm easier to read */
        PORTB = 0;

}
```

Here is the caller graph for this function:



2.1.3.2 void isr ()

Interrupt Service Routine. The interrupt service routine first checks that it was the timer interrupt that brought us here. If not, nothing is done.

If it was the timer interrupt, the global `Target` is used to select a pattern from the `Patterns` array. This way a pattern is selected which ensures that at least one LED is always on.

An attempt was made to turn off the LEDs for a short time each interrupt to reduce the duty cycle, and hence the current consumption. This actually increased the power consumption, presumably because of the capacitors wanting to stay charged.

Definition at line 94 of file Candle04.c.

```
00095 {
00096      int i;
00097
00098         if ( TMR0IF )           /* Was it the timer that brought us here? */
00099         {
00100                TMR0IF = 0;      /* Turn off the timer interrupt flag */
00101
00102                /* Select the 3 bits connected to the LEDs
00103                   and change them based on the value in Patterns[] */
00104                PORTB = (PORTB & 0xf1) | Patterns[Target&0x1f];
00105
00106                /* Reload the timer register */
00107                TMR0 = 128;
00108         }
00109 }
```

2.1.3.3 void main ()

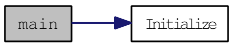
Mainline for the candle simulator. `main()` calls `Initialize()` and then enables the timer. `main()` then loops, establishing a somewhat random value for the global variable `Target` which will be used by the interrupt service routine to select the LED pattern.

Definition at line 144 of file Candle04.c.

```
00145 {
00146     int a,b,c;
00147
00148         /* Processor and port initializations */
00149         Initialize();
00150
00151         /* Initialize pattern counters */
00152         a = b = c = 0;
00153         Target = 0;
00154
00155         /* Enable timer interrupt and global interrupt */
00156         TMR0IE = 1;
00157         GIE=1;
00158
00159         while ( 1 == 1 )
```

```
        {
                /* Calculate the index into the Pattern[] array.  Really a
                   little redundant since all we are doing is setting an
                   index. Note that this will get executed many times, but
                   the result will only be used whenever a timer interrupt
                   occurs. */
                a += 38;
                b += 83;
                c += 134;
                Target = (a + b + c) & 0xff;
        }
}
```

Here is the call graph for this function:



2.1.4 Variable Documentation

2.1.4.1 word at CONFIG1

Initial value:

```
_WDT_OFF & _PWRTE_OFF & _INTRC_CLKOUT & _MCLR_ON &  _BODEN_OFF &
_LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF & _DEBUG_OFF & _CCP1_RB0 &
_CP_OFF
```

Configuration word 1. Watchdog timer off, Power-up timer on, internal RC timer with clock out of clock pins, `MCLR` not used as an IO, brown out detection off, low voltage programming off, EEPROM memory protection off, debug off, `CCP1` on `RB0` pin, code protection off.

Definition at line 45 of file Candle04.c.

2.1.4.2 word at CONFIG2

Initial value:

```
_FCMEN_ON & _IESO_ON
```

Configuration word 2. Internal/external switchover mode enabled, fail-safe clock monitor enabled.

Definition at line 52 of file Candle04.c.

2.1.4.3 const unsigned char Patterns[32] `[static]`

Initial value:

```
{
    0x06, 0x08, 0x00, 0x08, 0x0a, 0x02, 0x02, 0x04,
    0x0a, 0x06, 0x02, 0x08, 0x06, 0x0c, 0x00, 0x02,
    0x04, 0x04, 0x08, 0x04, 0x02, 0x06, 0x04, 0x00,
    0x02, 0x00, 0x06, 0x08, 0x00, 0x00, 0x06, 0x00
    }
```

Array of LED patterns. The array is long enough that randomly selecting an index will lead to an apparently random pattern. These patterns never allow all 3 LEDs to be off. The only relevant bits are 1, 2, and 3, and a true bit represents an LED off, so 0x0e is the not allowed pattern. The allowed patterns are 0x00, 0x02, 0x04, 0x06, 0x08, 0x0a and 0x0c.

Definition at line 70 of file Candle04.c.

2.1.4.4 int Target `[static]`

Select pattern. `Target` is calculated more or less randomly by the mainline. The interrupt service routine uses Target as an index into `Patterns[]`.

Definition at line 60 of file Candle04.c.

2.2 Candle04.c

```
/* Include processor file */
#include <pic16f88.h>

/* ---------------------------------------------------------------------- */
/* Configuration bits */
typedef unsigned int word;

word at 0x2007 CONFIG1 =
        _WDT_OFF & _PWRTE_OFF & _INTRC_CLKOUT & _MCLR_ON &  _BODEN_OFF &
        _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF & _DEBUG_OFF & _CCP1_RB0 &
        _CP_OFF;

word at 0x2008 CONFIG2 =
        _FCMEN_ON & _IESO_ON;

/* Global variables */

static int Target;

static const unsigned char Patterns[32] = {
    0x06, 0x08, 0x00, 0x08, 0x0a, 0x02, 0x02, 0x04,
    0x0a, 0x06, 0x02, 0x08, 0x06, 0x0c, 0x00, 0x02,
    0x04, 0x04, 0x08, 0x04, 0x02, 0x06, 0x04, 0x00,
    0x02, 0x00, 0x06, 0x08, 0x00, 0x00, 0x06, 0x00
    };



void isr() interrupt 0
{
     int i;

        if ( TMR0IF )           /* Was it the timer that brought us here? */
        {
                TMR0IF = 0;     /* Turn off the timer interrupt flag */

                /* Select the 3 bits connected to the LEDs
                   and change them based on the value in Patterns[] */
                PORTB = (PORTB & 0xf1) | Patterns[Target&0x1f];

                /* Reload the timer register */
                TMR0 = 128;
        }
}


void Initialize( void )
{
        /* Set the internal clock to 31.25 kHz */
        OSCCON = 0x0e;
        /* Mask all interrupts */
        INTCON = 0;
        /* Enable timer, use rising edge, prescaler to timer, 1:4 */
        OPTION_REG = 0xc1;
        /* PORTB all outputs */
        TRISB = 0;
        /* Just to put bank back to 0 to make asm easier to read */
        PORTB = 0;

}


void main()
{
    int a,b,c;

        /* Processor and port initializations */
        Initialize();

        /* Initialize pattern counters */
        a = b = c = 0;
        Target = 0;

        /* Enable timer interrupt and global interrupt */
        TMR0IE = 1;
        GIE=1;

        while ( 1 == 1 )
        {
                /* Calculate the index into the Pattern[] array.  Really a
                   little redundant since all we are doing is setting an
                   index. Note that this will get executed many times, but
                   the result will only be used whenever a timer interrupt
                   occurs. */
                a += 38;
                b += 83;
                c += 134;
                Target = (a + b + c) & 0xff;
        }
}
```

Index