Candle Simulator

Generated by Doxygen 1.6.1

Sun Dec 27 13:36:05 2009

Contents

1 File Index

1.1 File List

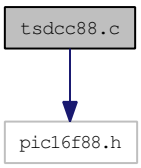
Here is a list of all files with brief descriptions:

2 File Documentation

2.1 tsdcc88.c File Reference

Little program to exercise SDCC. `#include <pic16f88.h>`

Include dependency graph for tsdcc88.c:



Typedefs

- typedef unsigned int word

Functions

- void isr ()

  Interrupt Service Routine.

- void Initialize (void)

  Initialization.

- void main ()

  Mainline.

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

Little program to exercise SDCC. Program flashes the 3 PIC-EL LEDs and an LED plugged into the transmitter port in an erratic fashion. The PIC-EL LEDs are red and get a different treatment than the xmit LED. The LED plugged into the xmit port is white.

This version uses the PIC16F88 with its internal oscillator set to 31.25 kHz. The red 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 form 3 vots. As a result, the sense of the LEDs is reversed compared to the PIC-EL.

Author:

jjmcd - 2009-11-26

Definition in file tsdcc88.c.

2.1.2 Typedef Documentation

2.1.2.1 typedef unsigned int word

Definition at line 27 of file tsdcc88.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 97 of file tsdcc88.c.

```
00098 {
00099         /* Set the internal clock to 31.25 kHz */
00100         OSCCON = 0x0e;
00101         /* Mask all interrupts */
00102         INTCON = 0;
00103         /* Enable timer, use rising edge, prescaler to timer, 1:4 */
00104         OPTION_REG = 0xc1;
00105         /* PORTB all outputs */
00106         TRISB = 0;
00107         /* Just to put bank back to 0 to make asm easier to read */
00108         PORTB = 0;
00109
00110 }
```

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 so, all the LEDs are turned off and the timer interrupt flag cleared. There is then a delay to give the LEDs some off time (to reduce current consumption).

Then the global Target is used to select a pattern from the LED pattern array. This way a pattern is selected which ensures that at least one LED is always on.

Definition at line 67 of file tsdcc88.c.

```
00068 {
00069     int i;
00070
00071         if ( TMR0IF )           /* Was it the timer that brought us here? */
00072         {
00073                 TMR0IF = 0;     /* Turn off the timer interrupt flag */
00074
00075                 // Turn off the LEDs
00076                 PORTB = (PORTB & 0xf1) | 0x0e;
00077                 /* Now hang around a while with the LEDs off to reduce the
00078                    average current consumption.
00079                    sdcc is (thankfully) not smart enough to optimize this
00080                    out of existence.  This results in 5 + 9 * loop count
00081                    instructions of wasted time with the LEDs off. */
00082                 for ( i=0; i<25; i++ ) ;
00083
00084                 /* Select the 3 bits connected to the LEDs
00085                    and change them based on the value in Patterns[] */
00086                 PORTB = (PORTB & 0xf1) | Patterns[Target&0x1f];
00087         }
00088 }
```

2.1.3.3 void main ()

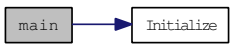
Mainline. main() calls Initialize() and then enables the timer. main() then loops, establishing a somewhat random value for the global variable Target which will be used by the interrupt service routine to select the LED pattern.

Definition at line 118 of file tsdcc88.c.

```
00119 {
00120     int a,b,c;
00121
00122         Initialize();
00123
00124         /* Initialize brightness counters */
```

```
00125         a = b = c = 0;
00126         Target = 0;
00127
00128         /* Enable timer interrupt and global interrupt */
00129         TMR0IE = 1;
00130         GIE=1;
00131
00132         while ( 1 == 1 )
00133         {
00134                 /* Somewhat leftover, really a little redundant
00135                    since all we are doing is setting an index.
00136                    Note that this will get executed many times,
00137                    but the result will only be used whenever a
00138                    timer interrupt occurs. */
00139                 a += 38;
00140                 b += 83;
00141                 c += 134;
00142                 Target = (a + b + c) & 0xff;
00143
00144         }
00145 }
```

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.

Definition at line 29 of file tsdcc88.c.

2.1.4.2 word at CONFIG2

Initial value:

```
_FCMEN_ON & _IESO_ON
```

Configuration word 2.

Definition at line 34 of file tsdcc88.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 indek 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, so 0x0e is the not allowed pattern. The allowed patterns are 0x00, 0x02, 0x04, 0x06, 0x08,0x0a and 0x0c.

Definition at line 50 of file tsdcc88.c.

2.1.4.4 int Target [static]

Select pattern. Target will be calculated more or less randomly by the mainline. The interrupt service routine will use Target as an index into Patterns[].

Definition at line 42 of file tsdcc88.c.

2.2 tsdcc88.c

```
00001
00022 /* Include processor file */
00023 #include <pic16f88.h>
00024
00025 /* ------------------------------------------------------------------------ */
00026 /* Configuration bits */
00027 typedef unsigned int word;
00029 word at 0x2007 CONFIG1 =
00030         _WDT_OFF & _PWRTE_OFF & _INTRC_CLKOUT & _MCLR_ON &  _BODEN_OFF &
00031         _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF & _DEBUG_OFF & _CCP1_RB0 &
00032         _CP_OFF;
00034 word at 0x2008 CONFIG2 =
00035         _FCMEN_ON & _IESO_ON;
00036
00037 /* Global variables */
00039
00042 static int Target;
00044
00050 static const unsigned char Patterns[32] = {
00051     0x06, 0x08, 0x00, 0x08, 0x0a, 0x02, 0x02, 0x04,
00052     0x0a, 0x06, 0x02, 0x08, 0x06, 0x0c, 0x00, 0x02,
00053     0x04, 0x04, 0x08, 0x04, 0x02, 0x06, 0x04, 0x00,
00054     0x02, 0x00, 0x06, 0x08, 0x00, 0x00, 0x06, 0x00
00055     };
00056
00058
00067 void isr() interrupt 0  /* interrupt service routine */
00068 {
00069      int i;
00070
00071         if ( TMR0IF )           /* Was it the timer that brought us here? */
00072         {
00073                 TMR0IF = 0;     /* Turn off the timer interrupt flag */
00074
00075                 // Turn off the LEDs
00076                 PORTB = (PORTB & 0xf1) | 0x0e;
00077                 /* Now hang around a while with the LEDs off to reduce the
00078                    average current consumption.
00079                    sdcc is (thankfully) not smart enough to optimize this
00080                    out of existence.  This results in 5 + 9 * loop count
00081                    instructions of wasted time with the LEDs off. */
00082                 for ( i=0; i<25; i++ ) ;
00083
00084                 /* Select the 3 bits connected to the LEDs
00085                    and change them based on the value in Patterns[] */
00086                 PORTB = (PORTB & 0xf1) | Patterns[Target&0x1f];
00087         }
00088 }
00089
00091
00097 void Initialize( void )
00098 {
00099         /* Set the internal clock to 31.25 kHz */
00100         OSCCON = 0x0e;
00101         /* Mask all interrupts */
```

```
00102         INTCON = 0;
00103         /* Enable timer, use rising edge, prescaler to timer, 1:4 */
00104         OPTION_REG = 0xc1;
00105         /* PORTB all outputs */
00106         TRISB = 0;
00107         /* Just to put bank back to 0 to make asm easier to read */
00108         PORTB = 0;
00109
00110 }
00111
00113
00118 void main()
00119 {
00120     int a,b,c;
00121
00122         Initialize();
00123
00124         /* Initialize brightness counters */
00125         a = b = c = 0;
00126         Target = 0;
00127
00128         /* Enable timer interrupt and global interrupt */
00129         TMR0IE = 1;
00130         GIE=1;
00131
00132         while ( 1 == 1 )
00133         {
00134                 /* Somewhat leftover, really a little redundant
00135                    since all we are doing is setting an index.
00136                    Note that this will get executed many times,
00137                    but the result will only be used whenever a
00138                    timer interrupt occurs. */
00139                 a += 38;
00140                 b += 83;
00141                 c += 134;
00142                 Target = (a + b + c) & 0xff;
00143
00144         }
00145 }
```

Index