Assignment 1: Introduction to the M16C/62 Microcontroller

ES21R Digital Design

0013679

School of Engineering, University of Warwick

01/05/02

Abstract

The aim of this laboratory was familiarisation with the programming and operation of the Mitsubishi M16C/62 family of microcontrollers. The microcontroller is located on a M16C Development Board (PCB) containing a M30620FP microcontroller, 4 PTM switches, 2x 7-segment LED display and many other connections which were not used. The board was connected to the serial port of a PC. The *IAR Embedded Workbench* was used to program and debug the microcontroller. The C-Spy tool contained in this was used in ROM monitor mode to test the programs produced and monitor variables etc. using the debugging functions. Three programs were produced which implemented a simple switch press counter. Different methods were employed and the results compared. An interrupt method was found to be better method for the update of the display than putting it as a simple foreground task.

Introduction

The aim of this laboratory is familiarisation with the programming and operation of the Mitsubishi M16C/62 family of microcontrollers. The microcontroller is located on a M16C Development Board (PCB) containing the M30620FP microcontroller, 4 PTM switches, 2x 7-segment LED displays and many other connections which will not be used. The board was connected to the serial port of a PC. The *IAR Embedded Workbench* will be used to program and debug the microcontroller. This works by using software on the PC to monitor the operation of the board giving an accurate picture of the system operation. The board will be programmed in the high level 'c' language which is preferable to assembly code for this application. The microcontrollers I/O system, interrupt system and timer will be used primarily in this investigation.

Implementation

The code in Ass1i.c is nearly complete but needs some additional code to update the display. The hardware consists of two 7-segment LED displays connected to one output port, P0. The two displays are enabled independently using a low on pins P1.1 (LED2) and P1.0 (LED1) (See appendix for circuit diagram. The following code is added which enables each display separately and writes the information to it:

```
P0 = dis_code[count%10];
P1.1 = 0;
P1.1 = 1;
P0 = dis_code[count/10];
P1.0 = 0;
P1.0 = 1;
```

The full program is shown below.

Code for Ass1i.c - Simple Implementation

```
/* Ass1i.c - Simple Keypress Counter */

#define Chip_30602

#include "stdio.h" /* Standard 'c' libraries */
#include "iom16c62.h" /* Microcontroller Specific library */

/* Array containing bit patterns for seven segment display numbers 0-9*/
unsigned char dis_code [10] = {
  0x0C0, 0x0F9, 0x0A4, 0x0B0, 0x99,
  0x92, 0x82, 0x0F8, 0x80, 0x98},
  count, dir;

void wait (unsigned int delay) /* Delay function for switch debouncing */
{                    /* Counts down from a given no. to waste time */
      while (delay !=0)
      delay--;
    }

  void main(void)
    {
      PD0 = 0x0FF; /* Set Port 0 (LED Display) direction to out */
      PD1.0 = 1;   /* Set Port1.0 (LED Display enable) direction to output */
      PD1.1 = 1;   /* Set Port1.1 (LED Display enable) direction to output */
      P0 = 0x000;  /* Switch off both LED displays */
      PD8.2 = 0;   /* Set Port8.2 (Switch 1) direction to input */
      while (1)    /* Infinite Loop */
        {
          /* Switch Press Counter */
          if (P8.2 == 0) /* If switch one is being pressed */
            {
              wait (10000); /* wait */
              if (P8.2 == 0) /* If still pressed (prevents bounce) */
```

```
            {
              ++count%100; /* increment count modulo-100*/
              while (P8.2 ==0);
              wait (10000);
            }
        }

      /* Update routine for LED displays */
      P0 = dis_code[count%10]; /*set port 0 to correct bit pattern (ones)*/
      P1.1 = 0; /* Enable writing to led 1 (tens) */
      P1.1 = 1; /* Disable writing to led 1 (display updated) */
      P0 = dis_code[count/10]; /*set port 0 to correct bit pattern (tens)*/
      P1.0 = 0; /* Enable writing to led 2 (units) */
      P1.0 = 1; /* Disable writing to led 2 (display updated) */
    }
  }
```

The code works as expected but the display is slightly dim. This is because the display is constantly being updated, dimming the display. Extra code is now required to implement a reverse button. When a key (SW2) is pressed the direction of the counting is required to be changed. This can be done quite simply with a bit of extra code to check for the new keypress as shown below:

Code for Ass1i.c - Implementation with reverse key

```
/* Ass1i_2.c - Keypress Counter with direction change */

#define Chip_30602

#include "stdio.h" /* Standard 'c' libraries */
#include "iom16c62.h" /* Microcontroller Specific library */

/* Array containing bit patterns for seven segment display numbers 0-9*/
unsigned char dis_code [10] = {
  0x0C0, 0x0F9, 0x0A4, 0x0B0, 0x99,
  0x92, 0x82, 0x0F8, 0x80, 0x98},
  count, dir;

void wait (unsigned int delay) /* Delay function for switch debouncing */
{                   /* Counts down from a given no. to waste time */
      while (delay !=0)
      delay--;
    }

  void main(void)
    {
      PD0 = 0x0FF; /* Set Port 0 (LED Display) direction to out */
      PD1.0 = 1;   /* Set Port1.0 (LED Display enable) direction to output */
      PD1.1 = 1;   /* Set Port1.1 (LED Display enable) direction to output */
      P0 = 0x000;  /* Switch off both LED displays */
      PD8.2 = 0;   /* Set Port8.2 (Switch 1) direction to input */
      PD8.3 = 0;   /* Set Port8.3 (Switch 2) direction to input */
      dir = 1;     /* Sets variable dir (hold count direction) to 1 (fwd)*/


      while (1)  /* Infinite Loop */
        {
        /* Direction Changer */
          if (P8.3 == 0)  /* If switch two is being pressed */
            {
              wait (10000); /* wait */

              if (P8.3 == 0) /* If still pressed (prevents bounce) */
                {
                  dir = !dir; /* Change direction */
                  while (P8.3 ==0);
                  wait (10000);
                }
            }

          /* Switch Press Counter */
          if (P8.2 == 0) /* If switch one is being pressed */
            {
              wait (10000); /* wait */
              if (P8.2 == 0) /* If still pressed (prevents bounce) */
                {
                  if (dir == 1) ++count%100; /*If fwd direction increment count
```

```
modulo-100*/
                  else --count%100; /*If rev direction decrement count modulo-100*/
                  while (P8.2 ==0);
                  wait (10000);
                }
            }

        /* Update routine for LED displays */
          P0 = dis_code[count%10]; /*set port 0 to correct bit pattern (ones)*/
          P1.1 = 0; /* Enable writing to led 1 (tens) */
          P1.1 = 1; /* Disable writing to led 1 (display updated) */
          P0 = dis_code[count/10]; /*set port 0 to correct bit pattern (tens)*/
          P1.0 = 0; /* Enable writing to led 2 (units) */
          P1.0 = 1; /* Disable writing to led 2 (display updated) */
       }
     }
```

This code works as required but the display is still dim. Also no trap has been put in for counting down so an overflow can occur if an attempt is made to count down past zero. This is relitivly easy to fix with a simple if statement.
One was to stop the diplay being so dim during operation is to put the display update routine as a background job controlled by interrupts. This will mean the display routine is invoked much less than before, leading to a brighter display.
The program can be rewritten as shown below:

Code for Ass1ii.c - Interrupt Implementation

```
/* Ass1ii.c - Keypress counter using interupts*/
#define Chip_30602

#include "stdio.h" /* Standard 'c' libraries */
#include "iom16c62.h" /* Microcontroller Specific library */
#include "intrm16c.h" /* Microcontroller Specific library */

/* Array containing bit patterns for seven segment display numbers 0-9*/
unsigned char dis_code [10] = {
  0x0C0, 0xF9, 0x0A4, 0x0B0, 0x99,
  0x92, 0x82, 0x0F8, 0x80, 0x98, 0xBF},
  count;
bit dir, dig; /* Direction and display varibles */


void wait (unsigned int delay) /* Delay function for switch debouncing */
     {                         /* Counts down from a given no. to waste time */
      while (delay !=0)
      delay--;
     }


interrupt [21*4] void Int_TimerA0 (void)            /*Interrupt service routine
                                        for Timer A0*/
  {
  switch (dig) // Updates alternate displays
    {
    case 0:
      P1.0 = 1; /* Disable writing to led 1 (display updated) */
     P0 = dis_code[count%10]; /*set port 0 to correct bit patern (ones)*/
      P1.1 = 0; /* Enable writing to led 2 (ones) */
     dig = !dig; /* Set tens to be updated next interrupt */
   break;
    case 1:
      P1.1 = 1; /* Disable writing to led 2 (display updated) */
      P0 = dis_code[count/10]; /*set port 0 to correct bit patern (tens)*/
      P1.0 = 0; /* Enable writing to led 1 (tens) */
     dig = !dig; /* Set ones to be updated next interrupt */
   break;
    }
   }

  void main(void)
    {
      PD0 = 0x0FF; /* Set Port 0 (LED Display) direction to out */
      PD1.0 = 1;   /* Set Port1.0 (LED Display enable) direction to output */
      PD1.1 = 1;   /* Set Port1.1 (LED Display enable) direction to output */
      PD8.2 = 0;   /* Set Port8.2 (Switch 1) direction to input */
      PD8.3 = 0;   /* Set Port8.3 (Switch 2) direction to input */
      P0 = 0x0FF;  /* Switch off both LED displays */
```

```
    P1.0 = 1; /* Disable writing to led 2 (units) */
    P1.1 = 1; /* Disable writing to led 1 (tens) */


    TA0MR = 0x80; /* Set timer with clock speed */
    TA0 = 10000;  /* Reset Value */
    TA0IC = 4;    /* Timer interupt Priority set*/
    TABSR.0 = 1;  /* Start TimerA0 */
    write_ipl (0);/* Enable all interupts */
  /*enable_interrupt ();    Not needed - already enabled by CSPY ROM Monitor*/

    while (1)
      {
      /* Direction Changer */
        if (P8.3 == 0)  /* If switch two is being pressed */
          {
            wait (10000);  /* Debounce delay */

            if (P8.3 == 0) /* If still pressed (prevents bounce) */
              {
                dir = !dir; /* Change direction */
                while (P8.3 ==0); /* Loop until switch is released */
                wait (10000); /* Debounce delay */
              }
          }

        /* Switch Press Counter */
        if (P8.2 == 0) /* If switch one is being pressed */
          {
            wait (10000);  /* Debounce delay */
            if (P8.2 == 0) /* If still pressed (prevents bounce) */
              {
                if (dir) ++count%100;/*If fwd direction increment count modulo-100*/
                else {              /*If rev direction decrement count modulo-100*/
             if (count == 0)
               count = 99;
             else
               --count%100;

                while (P8.2 ==0);  /* Loop until switch is released */
                wait (10000);      /* Debounce delay */
              }
          }
      }
  }
```

The operation of this program is more satisfactory than Ass1i.c. The display is updated considerable less often, but still often enough to seem instant to the user. The display is therfore at a considerably higher instensity than Ass1i.c, and therfore easier to read. Also an extra piece of code has been added to stop the counter overflowing when the number is decremented below zero. (the count variable is unsigned char).

Design of a Random Number Generator

A design for a random number generator can be produced by altering a previous program. The program can be made to count very quickly from 0-99 when a button is pressed and stop on one number when it is released. If the numbers cycle many times before the switch is released then the number produced should be random. A possible solution to this problem is shown below:

```
/* Ass1Rand.c - Random number generator 0-99 */

#define Chip_30602

#include "stdio.h" /* Standard 'c' libraries */
#include "iom16c62.h" /* Microcontroller Specific library */

/* Array containing bit patterns for seven segment display numbers 0-9*/
unsigned char dis_code [10] = {
  0x0C0, 0x0F9, 0x0A4, 0x0B0, 0x99,
  0x92, 0x82, 0x0F8, 0x80, 0x98},
  count;

  void main(void)
```

```
    {
      PD0 = 0x0FF; /* Set Port 0 (LED Display) direction to out */
      PD1.0 = 1;   /* Set Port1.0 (LED Display enable) direction to output */
      PD1.1 = 1;   /* Set Port1.1 (LED Display enable) direction to output */
      P0 = 0x0AF;  /* Switch on g on both LED displays (shows active and ready by --
on display)*/
      PD8.2 = 0;   /* Set Port8.2 (Switch 1) direction to input */


      while (1)  /* Infinate Loop */
        {

          /* Switch Press Counter */
          while (P8.2 == 0) /* If switch one is being pressed */
            {
              if (count != 99) ++count; /* Check for overflow if non increase count */
              else count = 0; /* Reset Count */
              /* Update Display */
              P0 = dis_code[count%10]; /*set port 0 to correct bit patern (ones)*/
              P1.1 = 0; /* Enable writing to led 1 (tens) */
              P1.1 = 1; /* Disable writing to led 1 (display updated) */
              P0 = dis_code[count/10]; /*set port 0 to correct bit patern (tens)*/
              P1.0 = 0; /* Enable writing to led 2 (units) */
              P1.0 = 1; /* Disable writing to led 2 (display updated) */
            }
    }
  }
```

This program will require testing to confirm its operation and to remove any bugs. However the concept is a good way of producing a random number but it does require some user interaction.

Conclusion

All the tested programs produced worked correctly, (after simple debugging when required) The key-press count using the interrupt method worked the best. The constant updating of the LEDs in the Ass1i.c program caused a noticeable dimming of the display. This however did not occur with the Ass1ii.c which updates the display less frequently. The laboratory provided a useful introduction to the use of the Mitsubishi M16C/62 family of microcontrollers and the used of a prototyping board with ROM monitoring software for use in system development. It also proved the usefulness of interrupts in program design, especially for timing. The use of ‘c’ for this application was suitable, as processing time was not an issue. Assembly is only generally used when fast code is required.

Bibliography

Lecture Notes - ES21R Digital Design
Assignment Sheet - ES21Q Digital Design
A. Fischer, D. Eggert, S. Ross, *Applied C: An introduction and more*, *McGraw-Hill*, 2001.
Lecture Notes - ES153 Engineering Software

Appendix A: Assignment Sheet

Including PCB circuit diagrams.