```
// S100 Bus Interface for the Edison S100 board (John Monahan S100Computers.com)
//
//    V0.1              12/2/2016   Start of display port address working
//  V0.2                12/10/2016  Basic system working
//  V0.3             12/25/2016  Converted to an Edison .service
//  V1.0             1/21/2017   First released version (Works with V1.6P Board)
//
// Root password = 000 000 000 for Edison 10.0.0.106
//
// Notes:-
// You can use the printf() function to display feedback text messages in the Eclipse Console window.
// However if they are too rapid/long the S100 system seems to get out of sync.
// They are however very useful for debugging.
//
#include "mraa.h"
#include <stdio.h>
#include <unistd.h>

#define FALSE 0
#define TRUE 1
#define LOW 0
#define HIGH 1
#define ESC  0x1b
#define CR   0x0d
#define LF   0x0a
#define BS   0x08
#define BELL 0x07
#define SP   0x20
#define DEL  0x7f

                                                    // These three tests are used only initially to
check/debug the hardware
#define TEST1 FALSE                                   // Set TRUE to pulse D4 as a simple test
#define TEST2 FALSE                                     // Set TRUE to increase from 0 to FFFFF the S100 Bus
address lines (Can be monitored with SMB)
#define TEST3 FALSE                                 // Set TRUE to continuously send "3" to the S100 Bus
console out port (01H)
#define TEST4 FALSE                                   // Read a keyboard character and print it on screen

#define SerialBoardPresent TRUE                     // If serial board for speech synthesis is present
#define IOBYTE  0xEF                                // S100 Bus IOBYTE Port on V3 SMB Board
#define CON_OUT_PORT 1                              // Port 1 on Propeller driven Console I/O Board
#define CON_IN_PORT 1                               // Port 1 on Propeller driven Console I/O Board
#define CON_STATUS_PORT 0                           // Port 0 on Propeller driven Console I/O Board
#define BCTL 0xA0                                   // Serial board speaker CTL port (Zilog SCC Chip)
#define BDTA 0xA2                                   // Speaker data port
```

```
#define E_STOP_REQUEST 4                    // GP135, Input low to high stops S100 process on Edison
#define E_sINP 13                           // GP128, For S100 bus sINP
#define E_sOUT 14                           // GP13_PWM1
#define E_MEMR 15                           // GP165
#define E_MEMW 19                           // GP19
#define P24                              // GP12_PWM0    (Spare)
#define DATA_WR 21                                // GP183_PWM3
#define EDISON_READY 23                  // GP110
#define DATA_RD 24                       // GP114

#define ACTIVATE_EDISON 25                  // GP129
#define RW_PULSE 38                               // GP43
#define P17  44                                   // GP134  (Spare)
#define P18  55                          // GP81   (Spare)
#define P19  39                          // GP77   (Spare)
#define P23  40                                   // GP82   (Spare)
#define E_PSYNC 41                       // GP83

#define P45 45                                    // GP45   (Spare, Always HIGH)

#define bDO0 37                                   // GP40   (All set as OUTPUTS)
#define bDO1 46                                   // GP47
#define bDO2 47                                   // GP49
#define bDO3 48                          // GP15
#define bDO4 49                                   // GP48
#define bDO5 50                                   // GP42
#define bDO6 51                                   // GP42
#define bDO7 52                                   // GP78

#define E_RESET_CMD 53                      // GP79
#define ADDRESS3  54                        // GP80
#define ADDRESS2  31                        // GP44
#define ADDRESS1  32                        // GP46
#define E_sINTA   33                        // GP48
#define S4              35                        // GP131  (Spare)
#define S100_INT  36                        // GP14

#define bDI0 0                              // GP182_PWM2   (All set as INPUTS)
#define bDI1 26                                   // GP130
#define bDI2 6                                    // GP27
#define bDI3 7                                    // GP20
#define bDI4 8                                    // GP28
#define bDI5 9                                    // GP111
#define bDI6 10                                   // GP109
```

```
#define bDI7 11                                                    // GP115

extern void InterruptRoutine();
extern void SetRAMAddress(long);
extern void SetPortAddress(int);
extern void WritePort(int,int);
extern  int ReadPort(int);
extern void Send_pDBIN();
extern void Send_pSYNC();
extern void Send_sOUT();
extern void Send_pWR();
extern void Send_sINP();
extern void EndBusCycle();
extern void WriteRAM(long,int);
extern void Send_MWRT();
extern int  ReadRAM(long);
extern void Send_sMEMR();
extern void Send_Z80Reset();

extern void PrintString(char*);
extern void PutChar(char);
extern char GetChar();
extern  int GetStatus();
extern void PutCRLF();
extern void PrintSignon();
extern void PrintMenuOptions();
extern void ShowRAMMap();
extern void DisplayRAM();
extern long GetHexValue();
extern void GetHex2Values(long*,long*);
extern void GetHex3Values(long*,long*, long*);
extern int toupper(int);
extern void DisplayRAM_ASCII();
extern void Echo();
extern int isascii(int);
extern void FillRAM();
extern void MoveRAM();
extern void VerifyRAM();
extern void SubstituteRAM();
extern void QueryPort();
extern void S100Signals();
extern void InitilizeSerialPort(int);
extern int SpeakString(char*);
extern int SpeakOut(char);
extern void PutBinary(char);
```

```
extern long GetDecimalValue();                          // Return a long decimal value from keyboard
extern void StopChange();

int Stop_Flag;
int Activate_Interrupts_Flag;
int Interrupt_Flag;
mraa_gpio_context pin[56];
char buffer[1024];
int AbortFlag;

int main()
{
int  i;
char c;
mraa_init();

      for (i=0; i < 56; i++)                            // INITILIZE ALL EDISON PINS
            {
            switch(i)
                  {
                  case 1:                                       //skip these pins (Note these are MRAA library
pin numbers)
                  case 2:
                  case 3:
                  case 5:
                  case 12:
                  case 16:
                  case 17:
                  case 18:
                  case 22:
                  case 27:
                  case 28:
                  case 29:
                  case 30:
                  case 34:
                  case 39:                                  // P19 (GP77)
                  case 40:                                  // P23
                  case 42:
                  case 43:
                  case 44:                                  // P17 (GP134)
                  case 55:                                  // P18 (GP81)
                        break;
                  case ACTIVATE_EDISON:
                        pin[i] = mraa_gpio_init(i);                 // Set Slave Active Flag (U12-p19) as input
                        mraa_gpio_dir(pin[i], MRAA_GPIO_IN);
```

```
                        mraa_gpio_use_mmaped(pin[i],1);                 // For fast I/O
                        break;
                case 0:                                                 // S100 bus data inputs (U5)
                case 26:
                case 6:
                case 7:
                case 8:
                case 9:
                case 10:
                case 11:
                        pin[i] = mraa_gpio_init(i);                     // Set eight S100 data inputs  (From U5)
                        mraa_gpio_dir(pin[i], MRAA_GPIO_IN);
                        mraa_gpio_use_mmaped(pin[i],1);                 // For fast I/O
                        break;
                case E_STOP_REQUEST:
                        pin[i] = mraa_gpio_init(i);
                        mraa_gpio_dir(pin[i], MRAA_GPIO_IN);
                        mraa_gpio_isr(pin[i], MRAA_GPIO_EDGE_RISING, &StopChange,NULL);
                        mraa_gpio_use_mmaped(pin[i],1);
                        break;
                case S100_INT:
                        pin[S100_INT] = mraa_gpio_init(S100_INT);               // By Default S100 Interrupts are
not turned on
                        mraa_gpio_dir(pin[S100_INT], MRAA_GPIO_IN);
                        mraa_gpio_isr(pin[S100_INT], MRAA_GPIO_EDGE_RISING, &InterruptRoutine,NULL);
                        mraa_gpio_use_mmaped(pin[S100_INT],1);
                        break;
                default:
                        pin[i] = mraa_gpio_init(i);                                     // Default all outputs,
initially HIGH
                        mraa_gpio_mode(pin[i],MRAA_GPIO_STRONG);                // Note includes 8 Data outputs (U4)
and address lines
                        mraa_gpio_dir(pin[i], MRAA_GPIO_OUT_HIGH);
                        mraa_gpio_use_mmaped(pin[i],1);                                 // For fast I/O
                        }
        }

      mraa_gpio_write(pin[E_RESET_CMD],HIGH);                                   // Initialize all output pin
levels (just in case)
      mraa_gpio_write(pin[P45],HIGH);
      mraa_gpio_write(pin[ADDRESS1],HIGH);
      mraa_gpio_write(pin[ADDRESS2],HIGH);
      mraa_gpio_write(pin[ADDRESS3],HIGH);
      mraa_gpio_write(pin[RW_PULSE],HIGH);
      mraa_gpio_write(pin[DATA_WR],HIGH);
```

```
      mraa_gpio_write(pin[DATA_RD],HIGH);

      mraa_gpio_write(pin[E_sINP],HIGH);                                    // Start with S100 bus control
lines in NOP state.
      mraa_gpio_write(pin[E_sOUT],HIGH);
      mraa_gpio_write(pin[E_MEMR],HIGH);
      mraa_gpio_write(pin[E_MEMW],HIGH);
      mraa_gpio_write(pin[E_PSYNC],HIGH);
      mraa_gpio_write(pin[E_sINTA],HIGH);

      Activate_Interrupts_Flag = FALSE;                                     // Setup all flags (No
Interrupts for initial testing)
      Interrupt_Flag = FALSE;
      AbortFlag = FALSE;
      mraa_gpio_write(pin[EDISON_READY],LOW);                               // LOW from Edison board ->
LED D4 ON to indicate we are ready
                                                                                        // Inform
the CPLD code of status
      sleep(1);

      while(TEST1)                                                                  // <--- DIGNOSTIC
TEST (Loops forever if active)
            {
            mraa_gpio_write(pin[EDISON_READY],LOW);                         // LOW from Edison board ->
LED D4 ON to indicate we are ready
            sleep(1);
            mraa_gpio_write(pin[EDISON_READY],HIGH);                  // LOW from Edison board -> LED D4
ON to indicate we are ready
            sleep(1);
            printf("S100_Edison running. LED D4 should Flash On/Off\n");      // S100 bus has activated the board
            }


while(TRUE)                                                                             // <-- This is the
start of the main Edison software loop.
      {
      while(mraa_gpio_read(pin[ACTIVATE_EDISON]) == HIGH)               // Wait until Bus Master grants S100
access.
            {
            printf("Waiting for Activate command. \n");
            usleep(10000);
            }
      printf("S100_Edison running.\n");                                         // S100 bus has activated the
board (ACTIVATE_EDISON is LOW)
```

```
      while(TEST2)                                                    // <--- DIGNOSTIC TEST (Loops forever if
active)
            {
            printf("The S100 bus address lines should increase from 0H to FFFFFH\n");
            for(i=0;i < 0xFFFFF;i++)
                  {
                  SetRAMAddress((long)i);                                     // Set the S100 bus
address lines
                  Send_sMEMR();                                                     // Send pSync and
raise sMEMR status line on S100 bus (will stay up).
                  Send_pDBIN();                                                     // Send pDBIN
pulse to S100 bus
                  EndBusCycle();                                                    // Also Clear the
S100 Bus Status Line
//                printf("Address = %x\n",i);                               // Display current
address
                usleep(10000);
                  }
            }
      while(TEST3)                                                    // <--- DIGNOSTIC TEST (Loops forever if
active)
            {
            printf("The S100 bus console should continously display '3'\n");
            while(TRUE)
                  {
                  WritePort(CON_OUT_PORT, 0x33);                              // Write a 3's to
CON_OUT_PORT continously
                usleep(1000);
                  }
            }


      while(TEST4)                                                    // <--- DIGNOSTIC TEST (Loops forever if
active)
            {
            char c;
            printf("Read a keyboard character and print it on screen\n");
            while(TRUE)
                  {
                  while(!GetStatus());                                        // Check if a character
is available
                  c = ReadPort(CON_IN_PORT);                                  // If so get the
character
                  WritePort(CON_OUT_PORT, c);                                 // Write a 3's to
CON_OUT_PORT
```

```
                }
            }

      for (i=0; i<70; i++)                                                    // To test character
output to Console is OK
            PutChar(' ');
      PutCRLF();

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

      PrintSignon();
      PrintMenuOptions();
      if(SerialBoardPresent)
            InitilizeSerialPort(BCTL);                                  // Initialize Serial Port A0/A2 on serial
board (if present);

      while(mraa_gpio_read(pin[ACTIVATE_EDISON]) == LOW)          // Loop within main menu until 'Z' command
            {
            Stop_Flag = FALSE;
            if(AbortFlag)
                  {
                  PrintString("\r\nCommand Aborted\b\r\n\r\n");
                  PrintMenuOptions();
                  AbortFlag = FALSE;
                  }
            PutChar('>');
            c = toupper(GetChar());

            if(c != ESC)
                  PutChar(c);
            if(c != 'Q')                                                  // Because Q needs a second
character input
                  PutCRLF();

            switch(c)
                  {
                  case CR:
                  case LF:
                        PutChar(BELL);
                        break;
                  case ESC:
                        PrintString("\r\nTurn Edison CPUs Off. Reset S100 system. (Y/N):");
                        c = toupper(GetChar());
                        if(c != 'Y')
```

```
                        {
                        PrintString("N\r\n");
                        PrintMenuOptions();
                        break;
                        }
                Send_Z80Reset();
                PrintString("Y\r\nEdison CPUs Off (Until next system reboot). S100 system reset.\r\n");
                mraa_gpio_write(pin[EDISON_READY],HIGH);  // Edison board LED D4 OFF to indicate we are NOT
ready
                return MRAA_SUCCESS;
                break;

        case 'A':
                PrintString("S100 Bus Memory Map.\r\n");
                ShowRAMMap();                                           // Display Memory map
                break;
        case 'B':
                break;
        case 'D':
                DisplayRAM();                                           // Print HEX values in RAM
                break;
        case 'E':
                PrintString("\r\nEcho a character from keyboard. (ESC to quit)\r\n");
                Echo();                                                         // Keyboard test
                break;
        case 'F':
                FillRAM();                                              // Fill RAM area with a HEX value
                break;
        case 'G':
                S100Signals();
                break;
        case 'K':
                PutCRLF();
                PrintSignon();
                PrintMenuOptions();                                     // Show main menu
                break;
        case 'M':
                MoveRAM();                                              // Move RAM area
                break;
        case 'Q':
                QueryPort();                                            // Input or Output to a port
                break;
        case 'S':
                PrintString("\r\nSubstitute RAM.\r\n ");
                SubstituteRAM();                                    // Change RAM bytes one at a time
```

```
                        break;
                case 'T':
                        DisplayRAM_ASCII();                                     // Print ASCII values in RAM
                        break;
                case 'V':
                        VerifyRAM();                                            // Verify areas of RAM are the same.
                        break;
                case 'W':
                        SpeakString("This is a test of the Edison S 100 Board.$");      // Send test string to speech
synthesizer (Note the required '$')
                        PrintString("Spoke:- This is a test of the Edison S100 Board.\r\n\n");
                        break;
                case 'Z':
                        PrintString("\r\nEdison returning control back to S100 bus master. \r\n>");
                        Send_Z80Reset();                                        // Take easy way out (for now), just reset
the Z80
                        break;
                default:
                        PrintMenuOptions();
                        sprintf(buffer,"'%c' Menu option is not done yet.\r\n\n",c);
                        PrintString(buffer);
                        PutChar(BELL);
                        break;
                }
            }
        }
        return MRAA_SUCCESS;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////
////////////////////////////  HIGH LEVEL MONITOR SUPPORT ROUTINES
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////

void S100Signals()
{
char c;
long repeat;
long delay = 100000;                                                    // default pulse width
char char_buffer[100];
long port,address,value;

        while(TRUE)
```

```
{
PutCRLF();
PrintString(">>>>>>> S100 Bus Signal Testing Menu <<<<<<<\r\n\n");
PrintString("A    Turn Off all S100 bus Signals\r\n");
PrintString("B    Pulse sOUT    (S100 bus High, Pin 45) \r\n");
PrintString("C    Pulse sMEMR   (S100 bus High, Pin 47) \r\n");
PrintString("D    Pulse MEMW    (S100 bus High, Pin 68) \r\n");
PrintString("E    Pulse sINTA   (S100 bus High, Pin 96) \r\n");
PrintString("F    Pulse pDBIN   (S100 bus High, Pin 78) \r\n");
PrintString("G    Pulse pWR*    (S100 bus Low,  Pin 77) \r\n");
PrintString("H    Cycle the Address lines (0-FFFFFH)\r\n");
PrintString("I    Write to a Port \r\n");
PrintString("J    Read From a Port \r\n");
PrintString("K    Write To RAM location    \r\n");
PrintString("L    Read From RAM location   \r\n");
PrintString("M    Test Interrupt (S100 V1)\r\n");
sprintf(char_buffer,"N  Set Pulse width.  Currently set to ~%ld uSec.\r\n",delay);
PrintString(char_buffer);
PrintString("ESC to return to the main menu\r\n\n");
if(AbortFlag)
      {
      PrintString("\r\nCommand Aborted\b\r\n\r\n");
      AbortFlag = FALSE;
      }
PutChar('>');
c = toupper(GetChar());
if(c == ESC)
      {
      PrintMenuOptions();
      return;
      break;
      }
PutChar(c);

switch(c)
      {
      case 'A':
            PrintString("\r\nAll S100 Bus lines are now turned off.\r\n>");
            EndBusCycle();
            PutCRLF();
            PutCRLF();
            break;

      case 'B':
            PrintString("\r\nPulse sOUT, Pin 45.  Enter # of times to pulse (XXXXXH+CR) ");
```

```
                        repeat = GetHexValue();
                        if(AbortFlag)
                              return;
                        PrintString("\r\nTest running.....");
                        while(repeat--)
                              {
                              mraa_gpio_write(pin[DATA_RD],LOW);        // Activate DATA IN lines on U5 via CPLD
                              mraa_gpio_write(pin[E_sOUT],LOW);         // Activate the above command with a low pulse
to the CPLD
                              usleep(delay);
                              mraa_gpio_write(pin[DATA_RD],HIGH);       // Inactivate DATA IN lines on U5 via CPLD
                              mraa_gpio_write(pin[E_sOUT],HIGH);        // Inactivate the above command with a low pulse
to the CPLD
                              if(GetStatus())
                                    {
                                    if(GetChar() == ESC)
                                          {
                                          AbortFlag = TRUE;
                                          return;
                                          }
                                    }
                              }
                        PrintString("\r\nPulse sOUT test complete.\r\n\n");
                        mraa_gpio_write(pin[DATA_RD],HIGH);             // Inactivate DATA IN lines on U5 via CPLD
                        mraa_gpio_write(pin[E_sOUT],HIGH);              // Inactivate the above command with a low pulse
to the CPLD
                  break;

                  case 'C':
                        PrintString("\r\nPulse sMEMR, Pin47.   Enter # of times to pulse (XXXXXH+CR) ");
                        repeat = GetHexValue();
                        if(AbortFlag)
                              return;
                        PrintString("\r\nTest running.....");
                        while(repeat--)
                              {
                              mraa_gpio_write(pin[DATA_RD],LOW);        // Inactivate DATA IN lines on U5 via CPLD
                              mraa_gpio_write(pin[E_MEMR],LOW);         // Activate the above command with a low pulse
to the CPLD
                              usleep(delay);
                              mraa_gpio_write(pin[DATA_RD],HIGH);       // Inactivate DATA IN lines on U5 via CPLD
                              mraa_gpio_write(pin[E_MEMR],HIGH);        // Inactivate the above command with a low pulse
to the CPLD
                              if(GetStatus())
                                    {
```

```
                        if(GetChar() == ESC)
                              {
                              AbortFlag = TRUE;
                              return;
                              }
                        }
                  }
            PrintString("\r\nPulse sMEMR test complete.\r\n\n");
            mraa_gpio_write(pin[DATA_RD],HIGH);             // Inactivate DATA IN lines on U5 via CPLD
            mraa_gpio_write(pin[E_MEMR],HIGH);              // Inactivate the above command with a low pulse
to the CPLD
            break;

      case 'D':
            PrintString("\r\nPulse MWRT, pin 68.   Enter # of times to pulse (XXXXXH+CR) ");
            repeat = GetHexValue();
            if(AbortFlag)
                  return;
            PrintString("\r\nTest running.....");
            while(repeat--)
                  {
                  mraa_gpio_write(pin[DATA_WR],LOW);        // Activate DATA IN lines on U5 via CPLD
                  mraa_gpio_write(pin[E_MEMW],LOW);         // Activate the above command with a low pulse
to the CPLD
                  usleep(delay);
                  mraa_gpio_write(pin[DATA_WR],HIGH);       // Inactivate DATA IN lines on U5 via CPLD
                  mraa_gpio_write(pin[E_MEMW],HIGH);        // Inactivate the above command with a low pulse
to the CPLD
                  if(GetStatus())
                        {
                        if(GetChar() == ESC)
                              {
                              AbortFlag = TRUE;
                              return;
                              }
                        }
                  }
            PrintString("\r\nPulse MWRT test complete.\r\n\n");
            mraa_gpio_write(pin[DATA_WR],HIGH);             // Inactivate DATA IN lines on U5 via CPLD
            mraa_gpio_write(pin[E_MEMW],HIGH);              // Inactivate the above command with a low pulse
to the CPLD
            break;

      case 'E':
            PrintString("\r\nPulse sINTA, pin 96.   Enter # of times to pulse (XXXXXH+CR) ");
```

```
                repeat = GetHexValue();
                if(AbortFlag)
                      return;
                PrintString("\r\nTest running.....");
                while(repeat--)
                      {
                      mraa_gpio_write(pin[E_sINTA],LOW);        // pSYNC command to the CPLD
                      usleep(delay);
                      usleep(delay);
                      usleep(delay);
                      mraa_gpio_write(pin[E_sINTA],HIGH);
                      if(GetStatus())
                            {
                            if(GetChar() == ESC)
                                  {
                                  AbortFlag = TRUE;
                                  return;
                                  }
                            }
                      }
                PrintString("\r\nPulse sINTA test complete.\r\n\n");
                mraa_gpio_write(pin[E_sINTA],HIGH);
          break;

          case 'F':
                PrintString("\r\nPulse pDBIN, pin 78.   Enter # of times to pulse (XXXXXH+CR) ");
                repeat = GetHexValue();
                if(AbortFlag)
                      return;
                PrintString("\r\nTest running.....");
                while(repeat--)
                      {
                      mraa_gpio_write(pin[DATA_RD],LOW);        // Activate DATA IN lines on U5 via CPLD
                      mraa_gpio_write(pin[RW_PULSE],LOW);       // Activate the S100 pDBIN signal with a low
pulse to the CPLD
                      usleep(delay);
                      mraa_gpio_write(pin[RW_PULSE],HIGH);      // Activate the S100 pDBIN signal with a low
pulse to the CPLD
                      mraa_gpio_write(pin[DATA_RD],HIGH);       // Activate DATA IN lines on U5 via CPLD
                      if(GetStatus())
                            {
                            if(GetChar() == ESC)
                                  {
                                  AbortFlag = TRUE;
                                  return;
```

```
                              }
                        }
                  }
            PrintString("\r\nPulse pDBIN test complete.\r\n\n");
            mraa_gpio_write(pin[RW_PULSE],HIGH);            // Activate the S100 pDBIN signal with a low
pulse to the CPLD
            mraa_gpio_write(pin[DATA_RD],HIGH);             // Activate DATA IN lines on U5 via CPLD
            break;

      case 'G':
            PrintString("\r\nPulse pWR*, pin 77.   Enter # of times to pulse (XXXXXH+CR) ");
            repeat = GetHexValue();
            if(AbortFlag)
                  return;
            PrintString("\r\nTest running.....");
            while(repeat--)
                  {
                  mraa_gpio_write(pin[DATA_WR],LOW);        // Activate DATA IN lines on U5 via CPLD
                  mraa_gpio_write(pin[RW_PULSE],LOW);       // Activate the S100 pDBIN signal with a low
pulse to the CPLD
                  usleep(delay);
                  mraa_gpio_write(pin[RW_PULSE],HIGH);      // Activate the S100 pDBIN signal with a low
pulse to the CPLD
                  mraa_gpio_write(pin[DATA_WR],HIGH);       // Activate DATA IN lines on U5 via CPLD
                  if(GetStatus())
                        {
                        if(GetChar() == ESC)
                              {
                              AbortFlag = TRUE;
                              return;
                              }
                        }
                  }
            PrintString("\r\nPulse pDWR* test complete.\r\n\n");
            mraa_gpio_write(pin[RW_PULSE],HIGH);            // Activate the S100 pDBIN signal with a low
pulse to the CPLD
            mraa_gpio_write(pin[DATA_WR],HIGH);             // Activate DATA IN lines on U5 via CPLD
            break;

      case 'H':
            PrintString("\r\nTest running.....");
            address = 0;
            while(address < 0xffff)
                  {
```

```
                        SetRAMAddress(address++);                       // Note for the address lines to show up
on the SMB HEX display we need:-
                        Send_sMEMR();                                         // Send pSync and raise sMEMR status
line on S100 bus.
                        Send_pDBIN();                                         // Send pDBIN pulse to S100 bus
                        EndBusCycle();                                        // Also Clear the S100 Bus Status
Line
                        usleep(delay);
                        if(GetStatus())
                              {
                              if(GetChar() == ESC)
                                    {
                                    AbortFlag = TRUE;
                                    return;
                                    }
                              }
                        }
                  PrintString("\r\nAddress lines test complete.\r\n\n");
                  break;

            case 'I':
                  PrintString("\r\nEnter Port, Value, # of times (XXH,XXH,XXXXXH +CR) ");
                  GetHex3Values(&port, &value, &repeat);
                  if(AbortFlag)
                        return;
                  PrintString("\r\nTest running.....");
                  while(repeat--)
                        {
                        WritePort((int)port,(int)value);
                        if(GetStatus())
                              {
                              if(GetChar() == ESC)
                                    {
                                    AbortFlag = TRUE;
                                    return;
                                    }
                              }
                        }
                  if(AbortFlag)
                        return;
                  PrintString("\r\nWrite to port test complete.\r\n\n");
                  break;

            case 'J':
                  PrintString("\r\nEnter Port, # of times (XXH,XXXXXH +CR) ");
```

```
	GetHex2Values(&port,&repeat);
	if(AbortFlag)
		return;
	PrintString("\r\nTest running....");
	while(repeat--)
		{
		c = ReadPort((int)port);
		sprintf(char_buffer,"\r\nPort %02xH = %02xH",(int)port,(char)c);
		PrintString(char_buffer);
		if(GetStatus())
			{
			if(GetChar() == ESC)
				{
				AbortFlag = TRUE;
				return;
				}
			}
		}
	if(AbortFlag)
		return;
	PrintString("\r\nRead from port test complete.\r\n\n");
	break;

case 'K':
	PrintString("\r\nEnter Address, Value, # of times (XXXXXH,XXH,XXXXXH +CR) ");
	GetHex3Values(&address, &value, &repeat);
	if(AbortFlag)
		return;
	PrintString("\r\nTest running.....");
	while(repeat--)
		{
		WriteRAM(address,(int)value);
		if(GetStatus())
			{
			if(GetChar() == ESC)
				{
				AbortFlag = TRUE;
				return;
				}
			}
		}
	if(AbortFlag)
		return;
	PrintString("\r\nWrite to RAM test complete.\r\n\n");
	break;
```

```
		case 'L':
			PrintString("\r\nEnter RAM Address, # of read times (XXH,XXXXXH +CR) ");
			GetHex2Values(&address,&repeat);
			if(AbortFlag)
				return;
			PrintString("\r\nTest running....");
			while(repeat--)
				{
				c = ReadRAM(address);
				sprintf(char_buffer,"\r\nAddress %02xH = %02xH",(int)address,c);
				PrintString(char_buffer);
				if(GetStatus())
					{
					if(GetChar() == ESC)
						{
						AbortFlag = TRUE;
						return;
						}
					}
				}
			if(AbortFlag)
				return;
			PrintString("\r\nRead from RAM test complete.\r\n\n");
			break;

		case 'M':                                                    // Make sure the S100_INT pin
is configured above during pin initilization
			Activate_Interrupts_Flag = TRUE;
			PrintString("\r\nS100 Keyboard interrupt (V1, S100 pin 5) will now be recognized.\r\n");
			PrintString("\r\nType characters to test interrupt. (ESC to abort).\r\n");
			while(TRUE)
				{
				if(Interrupt_Flag)
					{
					PrintString("Keyboard Interrupted detected.\r\n");
					Interrupt_Flag = FALSE;
					if(GetChar() == ESC)
						break;
					}
				}
			Interrupt_Flag = FALSE;
			Activate_Interrupts_Flag = FALSE;
			PrintString("\r\nKeyboard test complete.\r\n\n");
			break;
```

```
                case 'N':
                      sprintf(char_buffer,"\r\nPulse Width = %ld uSec. Enter new value (XXXXXD) ",delay);
                      PrintString(char_buffer);
                      delay = GetDecimalValue();
                      if((AbortFlag) || (!delay))
                            return;
                      PutCRLF();
                      PutCRLF();
                      break;

                case CR:
                case LF:
                      PutChar(BELL);
                      break;

                default:
                      PrintString("\r\nInvalid menu Option.\r\n");
                      break;
                }
          }
      return;
}

void QueryPort()
{
char c;
long port,out_value;
int in_value;
char char_buffer[100];

      c = toupper(GetChar());

      switch(c)
            {
            case 'I':
                  PrintString("I\r\nQuery In Port (XXH) ");
                  port = GetHexValue();
                  if(AbortFlag)
                        return;
                  in_value = ReadPort(port);
                  sprintf(char_buffer," = %02x (",in_value);            // Print hex values
                  PrintString(char_buffer);
                  PutBinary((char)in_value);                                  // Print Binary value
                  PrintString(")\r\n");
```

```
                return;

        case 'O':
                PrintString("I\r\nQuery Out Port (XXH,XXH) ");
                GetHex2Values( &port,&out_value);
                if(AbortFlag)
                        return;
                WritePort((int)port,(int)out_value);
                PutCRLF();
                return;

        case ESC:
        default:
                AbortFlag = TRUE;
                return;
                break;
        }
}

void Echo()
{
      char c;

      while(TRUE)
            {
            c = GetChar();
            if (c == ESC)
                  {
                  AbortFlag = TRUE;
                  PutCRLF();

                  PrintMenuOptions();
                  break;
                  }
            if(isascii(c))
                  PutChar(c);
            else PutChar(BELL);
            }
}

void VerifyRAM()
{
      long start;
      long finish;
      long loc2;
```

```
	long p,q,temp;
	char c,k;
	int error_flag = 0;
	char char_buffer[200];

	PrintString("\rVerify RAM bytes. (XXXXXH,XXXXXH,XXXXXH+CR) ");
	GetHex3Values(&start, &finish, &loc2);
	if(AbortFlag)
		return;
	if(finish < start)                                          // Adjust so the order so first <
second
		{
		temp = start;
		start = finish;
		finish = temp;
		}
	PutCRLF();
	q = loc2;

	for(p = start; p <= finish; p++,q++)
		{
		c = ReadRAM(p);                                         // Add in fill byte
		k = ReadRAM(q);
		if (c != k)
			{
			if(error_flag++ == 6)
				{
				PrintString("\r\nMultiple mismatches. Will stop checking\r\n");
				PutCRLF();
				return;
				}
			sprintf(char_buffer,"\r\nMismatch at %lxH (%02x) and %lxH (%02x)\r\n",p,c,q,k);
			PrintString(char_buffer);
			}
		}
	if(!error_flag)
		{
		PrintString("\r\nNo mismatches were detected.\r\n");
		PutCRLF();
		return;
		}
}

void SubstituteRAM()
{
```

```
	long p;
	int c,k = 0;
	char char_buffer[200];

	PrintString("\r\nEnter RAM Location.(XXXXXH+CR) ");
	p = GetHexValue();
	if(AbortFlag)
		return;
	sprintf(char_buffer, "\r\n%05lx  ",p);
	PrintString(char_buffer);

	while(TRUE)
		{
		c = ReadRAM(p);
		if(AbortFlag)
			return;
		sprintf(char_buffer, "%02xH-",c);
		PrintString(char_buffer);
		c = GetHexValue();
		if(AbortFlag)
			{
			AbortFlag = FALSE;                                    // Use ESC just to end the
substitution process
			PutCRLF();
			PutCRLF();
			return;
			}
		if((c == CR) || (c == LF))
			{
			PutCRLF();
			PutCRLF();
			return;
			}
		WriteRAM(p++,c);
		PutChar(' ');
		if(k++ == 0x08)
			{
			sprintf(char_buffer, "\r\n%05lx  ",p);
			PrintString(char_buffer);
			k = 0;
			}
		}
	return;
}
```

```
void MoveRAM()
{
	long start;
	long finish;
	long new;
	long p,q,temp;
	char c;

	PrintString("\rMove RAM.(XXXXXH,XXXXXH,XXXXXH+CR) ");
	GetHex3Values(&start, &finish, &new);
	if(AbortFlag)
		return;
	if(finish < start)                                  // Adjust so the order so first <
second
		{
		temp = start;
		start = finish;
		finish = temp;
		}
	PutCRLF();
	q = new;

	for(p = start; p <= finish; p++)
		{
		c = ReadRAM(p);                                 // Add in fill byte
		WriteRAM(q++,c);
		}
	PutCRLF();
	return;
}

void FillRAM()
{
	long start;
	long finish;
	long fill_byte;
	long p,temp;

	PrintString("\rFill RAM.  (XXXXXH,XXXXXH,XXH+CR) ");
	GetHex3Values(&start, &finish, &fill_byte);
	if(AbortFlag)
		return;
	PutCRLF();
	if(finish < start)                                  // Adjust so the order so first <
second
```

```
            {
            temp = start;
            start = finish;
            finish = temp;
            }
      for(p = start; p <= finish; p++)
            WriteRAM(p,(char)fill_byte);                        // Add in fill byte
      PutCRLF();
      return;
}

void DisplayRAM()
{
      long start;
      long finish;
      long p,temp;
      int r;
      int i;
      char char_buffer[200];

      PrintString("\rDisplay RAM. (XXXXXH,XXXXXH+CR) ");
      GetHex2Values(&start, &finish);
      if(AbortFlag)
            return;
      PutCRLF();
      if(finish < start)                                        // Adjust so the order so first <
second
            {
            temp = start;
            start = finish;
            finish = temp;
            }

      sprintf(char_buffer, "\r\n%05lx  ",start);
      PrintString(char_buffer);

      for(p = start, i=0; p <= finish; p++,i++)
            {
            if(GetStatus())
                  {
                  if(GetChar() == ESC)
                        {
                        AbortFlag = TRUE;
                        return;
                        }
```

```
                }
            r =   ReadRAM(p);                                   // Get HEX value
            r &= 0xff;
            if(i == 0x10)
                {
                sprintf(char_buffer, "\r\n%05lx  ",p);
                PrintString(char_buffer);
                i = 0;
                }
            sprintf(char_buffer, "%02x ",r);
            PrintString(char_buffer);
            }
      PutCRLF();
      PutCRLF();
      return;
}

void DisplayRAM_ASCII()
{
      long start;
      long finish;
      long p,temp;
      int r;
      int i;
      char char_buffer[200];

      PrintString("\rDisplay RAM ASCII (XXXXXH,XXXXXH+CR) ");
      GetHex2Values(&start, &finish);
      if(AbortFlag)
            return;
      PutCRLF();
      if(finish < start)                                        // Adjust so the order so first <
second
            {
            temp = start;
            start = finish;
            finish = temp;
            }

      sprintf(char_buffer, "\r\n%05lx  ",start);
      PrintString(char_buffer);

      for(p = start, i=0; p <= finish; p++,i++)
            {
            if(GetStatus())
```

```
                {
                if(GetChar() == ESC)
                        {
                        AbortFlag = TRUE;
                        return;
                        }
                }
            r =   ReadRAM(p);                                   // Get HEX value
            r &= 0xff;
            if(r < ' ')                                         // Only printable characters
                r = '.';
            if(r > 0x7E)
                r= '.';
            if(i == 0x20)
                {
                sprintf(char_buffer, "\r\n%05lx  ",p);
                PrintString(char_buffer);
                i = 0;
                }
            sprintf(char_buffer, "%c",(char)r);
            PrintString(char_buffer);
            }
        PutCRLF();
        PutCRLF();
        return;
}

void ShowRAMMap()
{
        unsigned long k;
        char c1,c2,c3;

        for(k = 0; k < 0xffffff; k += 0x2000)
            {
            if(GetStatus())
                {
                if(GetChar() == ESC)
                        {
                        AbortFlag = TRUE;
                        return;
                        }
                }
            switch (k)
                {
                case 0:
```

```
            sprintf(buffer, "\r\n000000 ");
            PrintString(buffer);
            break;
      case 0x0080000:
            sprintf(buffer, "\r\n080000 ");
            PrintString(buffer);
            break;
      case 0x0100000:
      case 0x0180000:
      case 0x0200000:
      case 0x0280000:
      case 0x0300000:
      case 0x0380000:
      case 0x0400000:
      case 0x0480000:
      case 0x0500000:
      case 0x0580000:
      case 0x0600000:
      case 0x0680000:
      case 0x0700000:
      case 0x0780000:
      case 0x0800000:
      case 0x0880000:
      case 0x0900000:
      case 0x0980000:
      case 0x0a00000:
      case 0x0a80000:
      case 0x0b00000:
      case 0x0b80000:
      case 0x0c00000:
      case 0x0c80000:
      case 0x0d00000:
      case 0x0d80000:
      case 0x0e00000:
      case 0x0e80000:
      case 0x0f00000:
      case 0x0f80000:
            sprintf(buffer, "\r\n%4x ",(unsigned int)k);
            PrintString(buffer);
            break;
      }
c1 = ReadRAM(k);                                    //Read RAM
c2 = !c1;                                                 //Complement it
WriteRAM(k,c2);
c3 = ReadRAM(k);                                    //Read RAM again
```

```
            if(c3 == c2)                                                    //Must be RAM
                  {
                  WriteRAM(k,c1);                                           //Put back original data
                  PutChar('R');
                  }
            else if (c1 != 0xff)
                  {
                  PutChar('p');                                             //Must be PROM
                  }
            else
                  {
                  PutChar('.');                                             //Must be empty
                  }
            if(GetStatus())
                  {
                  if(GetChar() == ESC)
                        {
                        AbortFlag = TRUE;
                        return;
                        }
                  }
            }
      PutCRLF();
      PutCRLF();
}

void PrintSignon()
{
      int c;
      PrintString("Edison S100 Bus Monitor V1.12a John Monahan (2/21/2017) ");
      c = ReadPort(IOBYTE);
      sprintf(buffer, "IOBYTE = %x\r\n",c);
      PrintString(buffer);
}

void PrintMenuOptions()
{
      PrintString("\r\n-------------------------------------------------------------------\r\n");
      PrintString    ("A=Memmap      D=Disp RAM     E=Echo       F=Fill RAM    G=S100 Bus\r\n");
      PrintString    ("K=Menu        M=Move RAM     Q=Port I/O   T=Type RAM    S=Subs RAM\r\n");
      PrintString    ("V=Verify RAM  W=Speech Test  Z=To Z80     ESC to abort \r\n\r\n");
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////
//////////////////////////////  MEDIUM LEVEL MONITOR SUPPORT ROUTINES
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////


int ReadPort(int port_address)                                          // Read a byte from a  Port at 16 bit
address
{
      int i;
      int k;

      SetPortAddress(port_address);
      Send_sINP();                                                            // Send pSync and raise sINP status
line on S100 bus (will stay up).
      Send_pDBIN();                                                           // Send pDBIN pulse to S100 bus

      for(i=0; i < 8; i++)                                              // First set direction of Edison bits to
READ
                  {
                  switch(i)
                        {
                        case 0:
                         if(mraa_gpio_read(pin[bDI0]))
                               k = 1;
                         else k = 0;
                         break;
                        case 1:
                         if(mraa_gpio_read(pin[bDI1]))
                               k |= 0b00000010;
                         break;
                        case 2:
                         if(mraa_gpio_read(pin[bDI2]))
                               k |= 0b00000100;
                         break;
                        case 3:
                         if(mraa_gpio_read(pin[bDI3]))
                               k |= 0b00001000;
                         break;
                        case 4:
                         if(mraa_gpio_read(pin[bDI4]))
                               k |= 0b00010000;
                         break;
```

```
                        case 5:
                              if(mraa_gpio_read(pin[bDI5]))
                                    k |= 0b00100000;
                              break;
                        case 6:
                              if(mraa_gpio_read(pin[bDI6]))
                                    k |= 0b01000000;
                              break;
                        case 7:
                              if(mraa_gpio_read(pin[bDI7]))
                                    k |= 0b10000000;
                              break;
                        }
                  }
      EndBusCycle();                                          // Clear the S100 Bus Status Lines
etc.
//    printf("EndBusCycle() DONE \r\n");
      return k;
}

void WritePort(int port_address, int value)                   // Write a byte to a Port at 16 bit address
{
      int i;
      char k=0;

//    printf("%c",value);
      SetPortAddress(port_address);

      for(i=0; i < 8; i++)
            {
            k = ((value >> i) & 1);
            switch(i)
                  {
                  case 0:
                        if(k)
                              mraa_gpio_write(pin[bDO0],HIGH);
                        else mraa_gpio_write(pin[bDO0],LOW);
                        break;
                  case 1:
                        if(k)
                              mraa_gpio_write(pin[bDO1],HIGH);
                        else mraa_gpio_write(pin[bDO1],LOW);
                        break;
                  case 2:
                        if(k)
```

```
                        mraa_gpio_write(pin[bDO2],HIGH);
                  else mraa_gpio_write(pin[bDO2],LOW);
                  break;
            case 3:
                  if(k)
                        mraa_gpio_write(pin[bDO3],HIGH);
                  else mraa_gpio_write(pin[bDO3],LOW);
                  break;
            case 4:
                  if(k)
                        mraa_gpio_write(pin[bDO4],HIGH);
                  else mraa_gpio_write(pin[bDO4],LOW);
                  break;
            case 5:
                  if(k)
                        mraa_gpio_write(pin[bDO5],HIGH);
                  else mraa_gpio_write(pin[bDO5],LOW);
                  break;
            case 6:
                  if(k)
                        mraa_gpio_write(pin[bDO6],HIGH);
                  else mraa_gpio_write(pin[bDO6],LOW);
                  break;
            case 7:
                  if(k)
                        mraa_gpio_write(pin[bDO7],HIGH);
                  else mraa_gpio_write(pin[bDO7],LOW);
                  break;
            }
      }
      Send_sOUT();                                    // Raise sOUT status line on S100
bus (will stay up).
      Send_pWR();                                     // Send pWR* pulse to S100 bus
      EndBusCycle();                                  // Also Clear the S100 Bus Status
Line
      return;
  }

void SetPortAddress(int location)                     // Set S100 bus port address lines A0 -A15 to a
value (16 bits wide)
    {
      int i,k=0;

      for(i=0; i < 16; i++)
            {
```

```
k = ((location >> i) & 1);
switch(i)
	{
	case 0:
		if(k)
			mraa_gpio_write(pin[bDO0],HIGH);
		else mraa_gpio_write(pin[bDO0],LOW);
		break;
	case 1:
		if(k)
			mraa_gpio_write(pin[bDO1],HIGH);
		else mraa_gpio_write(pin[bDO1],LOW);
		break;
	case 2:
		if(k)
			mraa_gpio_write(pin[bDO2],HIGH);
		else mraa_gpio_write(pin[bDO2],LOW);
		break;
	case 3:
		if(k)
			mraa_gpio_write(pin[bDO3],HIGH);
		else mraa_gpio_write(pin[bDO3],LOW);
		break;
	case 4:
		if(k)
			mraa_gpio_write(pin[bDO4],HIGH);
		else mraa_gpio_write(pin[bDO4],LOW);
		break;
	case 5:
		if(k)
			mraa_gpio_write(pin[bDO5],HIGH);
		else mraa_gpio_write(pin[bDO5],LOW);
		break;
	case 6:
		if(k)
			mraa_gpio_write(pin[bDO6],HIGH);
		else mraa_gpio_write(pin[bDO6],LOW);
		break;
	case 7:
		if(k)
			mraa_gpio_write(pin[bDO7],HIGH);
		else mraa_gpio_write(pin[bDO7],LOW);
		break;
	case 8:
		mraa_gpio_write(pin[ADDRESS1],LOW);                    // Send lower 8 Bits
```

```
                mraa_gpio_write(pin[ADDRESS1],HIGH);
                if(k)
                        mraa_gpio_write(pin[bDO0],HIGH);
                else mraa_gpio_write(pin[bDO0],LOW);
                break;
        case 9:
                if(k)
                        mraa_gpio_write(pin[bDO1],HIGH);
                else mraa_gpio_write(pin[bDO1],LOW);
        break;
        case 10:
                if(k)
                        mraa_gpio_write(pin[bDO2],HIGH);
                else mraa_gpio_write(pin[bDO2],LOW);
        break;
        case 11:
                if(k)
                        mraa_gpio_write(pin[bDO3],HIGH);
                else mraa_gpio_write(pin[bDO3],LOW);
                break;
        case 12:
                if(k)
                        mraa_gpio_write(pin[bDO4],HIGH);
                else mraa_gpio_write(pin[bDO4],LOW);
        break;
        case 13:
                if(k)
                        mraa_gpio_write(pin[bDO5],HIGH);
                else mraa_gpio_write(pin[bDO5],LOW);
        break;
        case 14:
                if(k)
                        mraa_gpio_write(pin[bDO6],HIGH);
                else mraa_gpio_write(pin[bDO6],LOW);
                break;
        case 15:
                if(k)
                        mraa_gpio_write(pin[bDO7],HIGH);
                else mraa_gpio_write(pin[bDO7],LOW);
                mraa_gpio_write(pin[ADDRESS2],LOW);                    // Send upper 8 bits
                mraa_gpio_write(pin[ADDRESS2],HIGH);
                break;
        }
}
```

```
debug/see the address lines displayed on the                                    // For testing, if you want to
SetPortAddress() routine then the following                                     // SMB using ONLY the
activated. Normally they are inactive.                                          // 3 lines need to be
//    Send_sINP();                                                          // Send pSync and raise sINP status
line on S100 bus (will stay up).
//    Send_pDBIN();                                                         // Send pDBIN pulse to S100 bus
//    EndBusCycle();                                                        // Also Clear the S100 Bus Status
Line
 }

void SetRAMAddress(long location)                                       // Set S100 bus address lines A0-A23 to a value
(24 bits wide)
    {                                                                           // Remember location is a long
      int i, k=0;

      for(i=0; i < 24; i++)
            {
            k = ((location >> i) & 1);
            switch(i)
                  {
                  case 0:
                        if(k)
                              mraa_gpio_write(pin[bDO0],HIGH);
                        else mraa_gpio_write(pin[bDO0],LOW);
                        break;
                  case 1:
                        if(k)
                              mraa_gpio_write(pin[bDO1],HIGH);
                        else mraa_gpio_write(pin[bDO1],LOW);
                        break;
                  case 2:
                        if(k)
                              mraa_gpio_write(pin[bDO2],HIGH);
                        else mraa_gpio_write(pin[bDO2],LOW);
                        break;
                  case 3:
                        if(k)
                              mraa_gpio_write(pin[bDO3],HIGH);
                        else mraa_gpio_write(pin[bDO3],LOW);
                        break;
                  case 4:
                        if(k)
```

```
            mraa_gpio_write(pin[bDO4],HIGH);
      else mraa_gpio_write(pin[bDO4],LOW);
      break;
case 5:
      if(k)
            mraa_gpio_write(pin[bDO5],HIGH);
      else mraa_gpio_write(pin[bDO5],LOW);
      break;
case 6:
      if(k)
            mraa_gpio_write(pin[bDO6],HIGH);
      else mraa_gpio_write(pin[bDO6],LOW);
      break;
case 7:
      if(k)
            mraa_gpio_write(pin[bDO7],HIGH);
      else mraa_gpio_write(pin[bDO7],LOW);
      break;
case 8:
      mraa_gpio_write(pin[ADDRESS1],LOW); // Send lower 8 Bits
      mraa_gpio_write(pin[ADDRESS1],HIGH);
      if(k)
            mraa_gpio_write(pin[bDO0],HIGH);
      else mraa_gpio_write(pin[bDO0],LOW);
break;
case 9:
      if(k)
            mraa_gpio_write(pin[bDO1],HIGH);
      else mraa_gpio_write(pin[bDO1],LOW);
break;
case 10:
      if(k)
            mraa_gpio_write(pin[bDO2],HIGH);
      else mraa_gpio_write(pin[bDO2],LOW);
break;
case 11:
      if(k)
            mraa_gpio_write(pin[bDO3],HIGH);
      else mraa_gpio_write(pin[bDO3],LOW);
      break;
case 12:
      if(k)
            mraa_gpio_write(pin[bDO4],HIGH);
      else mraa_gpio_write(pin[bDO4],LOW);
break;
```

```
case 13:
      if(k)
            mraa_gpio_write(pin[bDO5],HIGH);
      else mraa_gpio_write(pin[bDO5],LOW);
break;
case 14:
      if(k)
            mraa_gpio_write(pin[bDO6],HIGH);
      else mraa_gpio_write(pin[bDO6],LOW);
      break;
case 15:
      if(k)
            mraa_gpio_write(pin[bDO7],HIGH);
      else mraa_gpio_write(pin[bDO7],LOW);
      break;
case 16:
      mraa_gpio_write(pin[ADDRESS2],LOW);                    // Send next 8 bits
      mraa_gpio_write(pin[ADDRESS2],HIGH);
      if(k)                                                        // Now continue with A16 - A23
            mraa_gpio_write(pin[bDO0],HIGH);
      else mraa_gpio_write(pin[bDO0],LOW);
      break;
case 17:
      if(k)
            mraa_gpio_write(pin[bDO1],HIGH);
      else mraa_gpio_write(pin[bDO1],LOW);
      break;
case 18:
      if(k)
            mraa_gpio_write(pin[bDO2],HIGH);
      else mraa_gpio_write(pin[bDO2],LOW);
      break;
case 19:
      if(k)
            mraa_gpio_write(pin[bDO3],HIGH);
      else mraa_gpio_write(pin[bDO3],LOW);
      break;
case 20:
      if(k)
            mraa_gpio_write(pin[bDO4],HIGH);
      else mraa_gpio_write(pin[bDO4],LOW);
      break;
case 21:
      if(k)
            mraa_gpio_write(pin[bDO5],HIGH);
```

```
                        else mraa_gpio_write(pin[bDO5],LOW);
                        break;
                  case 22:
                        if(k)
                              mraa_gpio_write(pin[bDO6],HIGH);
                        else mraa_gpio_write(pin[bDO6],LOW);
                        break;
                  case 23:
                        if(k)
                              mraa_gpio_write(pin[bDO7],HIGH);
                        else mraa_gpio_write(pin[bDO7],LOW);
                        mraa_gpio_write(pin[ADDRESS3],LOW);                  // Send top 8 bits
                        mraa_gpio_write(pin[ADDRESS3],HIGH);
                              break;
                  }
            }

                                                                                  // For testing, if you want to
debug/see the address lines displayed on the

                                                                              // SMB using ONLY the
SetRAMAddress() routine then the following

                                                                                  // 3 lines need to be
activated. Normally they are inactive.
//Send_sMEMR();                                                               // Send pSync and raise sMEMR status
line on S100 bus (will stay up).
//Send_pDBIN();                                                               // Send pDBIN pulse to S100 bus
//EndBusCycle();                                                            // Also Clear the S100 Bus Status Line
 }

void WriteRAM(long RAM_address, int value)
{
      int i;
      int k=0;

      SetRAMAddress(RAM_address);                                             // First set the RAM location

      for(i=0; i < 8; i++)
            {
            k = ((value >> i) & 1);
            switch(i)
                  {
                  case 0:
                        if(k)
                              mraa_gpio_write(pin[bDO0],HIGH);
                        else mraa_gpio_write(pin[bDO0],LOW);
                        break;
```

```
                case 1:
                      if(k)
                            mraa_gpio_write(pin[bDO1],HIGH);
                      else mraa_gpio_write(pin[bDO1],LOW);
                      break;
                case 2:
                      if(k)
                            mraa_gpio_write(pin[bDO2],HIGH);
                      else mraa_gpio_write(pin[bDO2],LOW);
                      break;
                case 3:
                      if(k)
                            mraa_gpio_write(pin[bDO3],HIGH);
                      else mraa_gpio_write(pin[bDO3],LOW);
                      break;
                case 4:
                      if(k)
                            mraa_gpio_write(pin[bDO4],HIGH);
                      else mraa_gpio_write(pin[bDO4],LOW);
                      break;
                case 5:
                      if(k)
                            mraa_gpio_write(pin[bDO5],HIGH);
                      else mraa_gpio_write(pin[bDO5],LOW);
                      break;
                case 6:
                      if(k)
                            mraa_gpio_write(pin[bDO6],HIGH);
                      else mraa_gpio_write(pin[bDO6],LOW);
                      break;
                case 7:
                      if(k)
                            mraa_gpio_write(pin[bDO7],HIGH);
                      else mraa_gpio_write(pin[bDO7],LOW);
                      break;
                }
          }
      Send_MWRT();
      Send_pWR();                                   // Send pWR* pulse to S100 bus
      EndBusCycle();                                // Also Clear the S100 Bus Status
Line
      return;

}
```

```
int ReadRAM(long RAM_address)                                   // Read a byte from a  Port at 16 bit address
{
      int i;
      int k=0;

      SetRAMAddress(RAM_address);

      Send_sMEMR();                                                   // Raise sMEMR status line on S100
bus (will stay up).
      Send_pDBIN();                                                   // Send pDBIN pulse to S100 bus

      for(i=0; i < 8; i++)                                          // First set direction of Edison Bits to
READ
                  {
                  switch(i)
                        {
                        case 0:
                         if(mraa_gpio_read(pin[bDI0]))
                               k = 1;
                         else k = 0;
                         break;
                        case 1:
                         if(mraa_gpio_read(pin[bDI1]))
                               k |= 0b00000010;
                         break;
                        case 2:
                         if(mraa_gpio_read(pin[bDI2]))
                               k |= 0b00000100;
                         break;
                        case 3:
                         if(mraa_gpio_read(pin[bDI3]))
                               k |= 0b00001000;
                               break;
                        case 4:
                         if(mraa_gpio_read(pin[bDI4]))
                               k |= 0b00010000;
                         break;
                        case 5:
                         if(mraa_gpio_read(pin[bDI5]))
                               k |= 0b00100000;
                         break;
                        case 6:
                         if(mraa_gpio_read(pin[bDI6]))
                               k |= 0b01000000;
                         break;
```

```
                        case 7:
                              if(mraa_gpio_read(pin[bDI7]))
                                    k |= 0b10000000;
                              break;
                        }
                  }
      EndBusCycle();                                          // Also Clear the S100 Bus Status
Line
      return k;
}

void PrintString(char* TextString)                      // Print a string on the console
{
      while(*TextString)
      {
            char p;
            p = *TextString++;
            PutChar(p);
      }
}

void PutCRLF()                                                // Send CR + LF to Console
{
      PutChar(CR);
      PutChar(LF);
}

void PutChar(char c)                                    // Print a character on the Console
{
      while(!(ReadPort(CON_STATUS_PORT) & 0x04));
      WritePort(CON_OUT_PORT,c);
}

char GetChar()                                                // Get a keyboard character from the
Console
      {
      char c;
      while(!GetStatus());                                    // Check if a character is available
      c = ReadPort(CON_IN_PORT);                              // Check if a character is available
      return c;
      }

int GetStatus()                                               // See if the is a character at the
Console Status port
      {
```

```
	int c;
	c = ReadPort(CON_STATUS_PORT);                                  // Check if a character is available
	return(c &= 0x02);
	}

void GetHex3Values(long* first, long* second, long* third)
{
	*first = 0;
	*second = 0;
	*third = 0;

	*first = GetHexValue();
	if(AbortFlag)
		return;
	*second = GetHexValue();
	if(AbortFlag)
		return;
	*third = GetHexValue();
	return;
}

void GetHex2Values(long* first, long* second)
{
	*first = 0;
	*second = 0;

	*first = GetHexValue();
	if(AbortFlag)
		return;
	*second = GetHexValue();
	return;
}

long GetHexValue()                                                  // Return a long HEX value from
keyboard
{
	int i = 0;
	char c;
	char char_buffer[256];
	long hex_value;
	while(TRUE)
		{
		c = GetChar();
		c = toupper(c);
		switch(c)
```

```
                {
                case ESC:
                      AbortFlag = TRUE;
                      return(0);
                case ' ':
                case ',':
                case '\n':
                case '\r':
                      if(c == ',')
                            PutChar(c);
                      char_buffer[i++] = 0;
                      sscanf(char_buffer,"%lx",&hex_value);
                      return(hex_value);
                      break;
                case ':':
                case ';':
                case '<':
                case '=':
                case '>':
                case '?':
                case '@':
                      PutChar(BELL);
                      continue;
                default:
                      if((c < '0') || (c > 'F' ))
                            {
                            PutChar(BELL);
                            continue;
                            }
                      char_buffer[i++] = c;
                      PutChar(c);
                }
          }
}

long GetDecimalValue()                                    // Return a long decimal value from
keyboard
{
      int i = 0;
      char c;
      char char_buffer[256];
      long dec_value;
      while(TRUE)
            {
            c = GetChar();
```

```
            c = toupper(c);
            switch(c)
                  {
                  case ESC:
                        AbortFlag = TRUE;
                        return(0);
                  case ' ':
                  case ',':
                  case '\n':
                  case '\r':
                        if(c == ',')
                              PutChar(c);
                        char_buffer[i++] = 0;
                        sscanf(char_buffer,"%ld",&dec_value);
                        return(dec_value);
                        break;
                  default:
                        if((c < '0') || (c > '9' ))
                              {
                              PutChar(BELL);
                              continue;
                              }
                        char_buffer[i++] = c;
                        PutChar(c);
                  }
            }
}

void PutBinary(char c)
{
int n;

      for(n = 8; n ;n--)                                          //Print binary values of a byte
            {
          if (c & 0x80)
              PutChar('1');
          else
              PutChar('0');
          c <<= 1;
            }
}

void InitilizeSerialPort(int base_port)
{
            WritePort(base_port,0x04);                          //Point to WR4
```

```
            WritePort(base_port,0x44);                          //X16 clock,1 Stop,NP
            WritePort(base_port,0x03);                          //Point to WR3
            WritePort(base_port,0x0C1);                         //Enable reciever, Auto Enable, Recieve 8
bits
            WritePort(base_port,0x05);                          //Point to WR5
            WritePort(base_port,0x0EA);                         //Enable, Transmit 8 bits
            WritePort(base_port,0x0B);                          //Set RTS,DTR, Enable. Point to WR11
            WritePort(base_port,0x56);                          //Recieve/transmit clock = BRG
            WritePort(base_port,0x0C);                          //Point to WR12
            WritePort(base_port,0x06);                          //Low byte 19,200 Baud
            WritePort(base_port,0x0D);                          //Point to WR13
            WritePort(base_port,0x00);                          //High byte for Baud
            WritePort(base_port,0x0E);                          //Point to WR14
            WritePort(base_port,0x01);                          //Use 4.9152 MHz Clock. Note SD Systems
uses a 2.4576 MHz clock, enable BRG
            WritePort(base_port,0x0F);                          //Point to WR15
            WritePort(base_port,0x00);                          //Generate Int. with CTS going high
            return;
}

int SpeakOut(char character)
{
      int c;
      int retry_count = 100;

      if(ReadPort(BCTL) == 0xff)
            {
            PrintString("Speech Synthesizer not detected.\r\n");
            PutChar(BELL);
            return 0;
            }
      while(retry_count--)
            {
            c = ReadPort(BCTL);
            if((c &= 0x04))
                  {
                  WritePort(BDTA, character);
                  return 1;
                  }
            }
      PrintString("Speech Synthesizer timed out.\r\n");
      PutChar(BELL);
      return 0;
}
```

```
int SpeakString(char* SpeechString)
{
	char p;

	while((p = *SpeechString++) != '$')
		{
		if((p < SP ) || (p == DEL) || (p == '\n') || (p == '\r'))    // Send speech if CR as well
			break;
		if(!SpeakOut(p))
			return 0;
		}
	if(!SpeakOut(CR))
		return 0;
	return 1;
}


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////
////////////////////////////  LOW LEVEL MONITOR SUPPORT ROUTINES
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////

void Send_pSYNC()
	{
	mraa_gpio_write(pin[E_PSYNC],LOW);                              // pSYNC command to the CPLD
	mraa_gpio_write(pin[E_PSYNC],HIGH);
//	printf("Sent pSYNC\n");
	}


void Send_MWRT()
	{
	mraa_gpio_write(pin[DATA_WR],LOW);                              // Activate DATA OUT lines on U4 via CPLD
	mraa_gpio_write(pin[E_MEMW],LOW);                               // Activate the above command with a low pulse
to the CPLD
	mraa_gpio_write(pin[E_PSYNC],LOW);                              // pSYNC command to the CPLD
	mraa_gpio_write(pin[E_PSYNC],HIGH);
//	printf("Sent MWRT\n");
	}

void Send_sMEMR()
	{
	mraa_gpio_write(pin[DATA_RD],LOW);                              // Activate DATA OUT lines on U4 via CPLD
```

```
      mraa_gpio_write(pin[E_MEMR],LOW);                    // Activate the above command with a low pulse
to the CPLD
      mraa_gpio_write(pin[E_PSYNC],LOW);                   // pSYNC command to the CPLD
      mraa_gpio_write(pin[E_PSYNC],HIGH);
//    printf("Sent sMEMR\n");
      }

void Send_sOUT()
      {
      mraa_gpio_write(pin[DATA_WR],LOW);                   // Activate DATA OUT lines on U4 via CPLD
      mraa_gpio_write(pin[E_sOUT],LOW);                    // Activate the above command with a low pulse
to the CPLD
      mraa_gpio_write(pin[E_PSYNC],LOW);                   // pSYNC command to the CPLD
      mraa_gpio_write(pin[E_PSYNC],HIGH);
//    printf("Sent sOUT\n");
      }

void Send_sINP()
      {
      mraa_gpio_write(pin[DATA_RD],LOW);                   // Activate DATA IN lines on U5 via CPLD
      mraa_gpio_write(pin[E_sINP],LOW);                    // Activate the above command with a low pulse
to the CPLD
      mraa_gpio_write(pin[E_PSYNC],LOW);                   // pSYNC command to the CPLD
      mraa_gpio_write(pin[E_PSYNC],HIGH);
//    printf("Sent sINP\n");
      }

void Send_pWR()
      {
      while(Stop_Flag);
      mraa_gpio_write(pin[RW_PULSE],LOW);                  // Activate the S100 pWR* signal with a low
pulse to the CPLD
//    printf("Sent pWR*\n");                                     // Important! PW_PULSE does not go high
until EndCycle()
      }

void Send_pDBIN()
      {
      while(Stop_Flag);
      mraa_gpio_write(pin[RW_PULSE],LOW);                  // Activate the S100 pDBIN signal with a low
pulse to the CPLD
//  Printf("Sent pDBIN\n");                                    // Important! PW_PULSE does not go high
until EndCycle()
      }
```

```
void Send_Z80Reset()
      {
      WritePort(0xEE,0);                                  // Reset the TMA lines 1,2 & 3.
      mraa_gpio_write(pin[E_RESET_CMD],LOW);            // Activate the above command with a low pulse
to the CPLD
      mraa_gpio_write(pin[E_RESET_CMD],HIGH);
//  Printf("Sent Z80 Reset\n");
      }

void EndBusCycle()
      {                                                       // Terminate the current S100
bus cycle
      mraa_gpio_write(pin[RW_PULSE],HIGH);              // Deactivate the pDBIN or pWR line with a low
pulse to the CPLD
      mraa_gpio_write(pin[DATA_RD],HIGH);               // Deactivate DATA IN lines on U5 via CPLD
      mraa_gpio_write(pin[DATA_WR],HIGH);               // Deactivate DATA OUT lines on U4 via CPLD
      mraa_gpio_write(pin[E_PSYNC],HIGH);
      mraa_gpio_write(pin[E_sINP],HIGH);                // Deactivate Status lines
      mraa_gpio_write(pin[E_sOUT],HIGH);                // Deactivate Status lines
      mraa_gpio_write(pin[E_MEMR],HIGH);                // Deactivate Status lines
      mraa_gpio_write(pin[E_MEMW],HIGH);                // Deactivate Status lines
      }

void StopChange()                                           // For edge triggered single step request
    {
      Stop_Flag = !Stop_Flag;
//    printf("Stop_Flag = %x \n",Stop_Flag);
      sleep(1);                                                 // Delay 1 sec. to stop switch
bounce
      }

void InterruptRoutine()                                     // For edge triggered interrupt to
activate S100 interrupt
    {
      if(!Interrupt_Flag)
            {
            Interrupt_Flag = TRUE;
            printf("Interrupt_Flag = TRUE \n");
            }
    }
```