// 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)

// V1.3 4/9/2017 Modified S100\_Edison.c for TO\_BUS/FROM\_BUS capability

//

// 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>

#include <string.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

#define BUS\_CYCLES\_MAX 100 // Max number of Bus cycles captured

// The tests below 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 TEST5 FALSE // USB port testing

#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 P14 20 // GP12\_PWM0 (Unused, seems inactive)

#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 P15 44 // GP134 (Unused, seems inactive)

#define USB\_PC6 55 // GP81 USB Out Status port bit (Read only)

#define P16 39 // GP77 (Unused, seems inactive)

#define USB\_PC7 40 // GP81 USB In Status port bit (Read only)

#define E\_PSYNC 41 // GP83

#define USB\_SEL 45 // GP45 (USB Data & Status port select)

#define bDO1 46 // GP47 (All set as OUTPUTS)

#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\_WAIT 53 // GP79 Either adds wait states to S100 bus or resets back to Z80

#define ADDRESS3 54 // GP80

#define ADDRESS2 31 // GP44

#define ADDRESS1 32 // GP46

#define E\_sINTA 33 // GP48

#define TO\_BUS 35 // GP131 (Note, Edison Output only pin)

#define S100\_INT 36 // GP14

#define bDO0 37 // GP40 (Set as OUTPUT)

#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();

extern char ReadUSBPort();

extern void WriteUSBPort(char);

extern char GetUSBInStatus(); // Read Status In for USB Port

extern char GetUSBOutStatus(); // Read Status Out/Busy for USB Port

extern void USBPutChar(char);

extern char USBGetChar();

extern int SelectTrigger(); // Define an S100 bus signal that will trigger analysis

extern int CaptureData(); // Calls GetCurrentAddressLines(), GetCurrentStatusLine(),GetCurrentDataOut()

extern long GetCurrentAddressLines(int); // Capture Address line data. Note assumes S100 Bus is halted

extern int GetCurrentStatusLine(int); // Capture Status line data. Note assumes S100 Bus is halted

extern int GetCurrentDataIn(int); // Capture Data In lines. Note assumes S100 Bus is halted

extern int GetCurrentDataOut(int); // Capture Data In lines. Note assumes S100 Bus is halted

extern int HaltCPU(); // Halt the S100 bus CPU, display current state.

int Stop\_Flag;

int Activate\_Interrupts\_Flag;

int Interrupt\_Flag;

mraa\_gpio\_context pin[56];

char buffer[1024];

int AbortFlag;

int TO\_BUS\_FLAG = TRUE; // Initialize in TO\_BUS\* signal LOW mode i.e. the Edison controls the S100 bus.

long ADDRESS\_BREAKPOINT = 0;

int TRIGGER\_SIGNAL = -1;

int BUS\_CYCLES = 0;

int STATUS\_SIGNAL = -1;

char TriggerName[10];

struct BusData

{

long AddressLine;

char StatusLineName[10];

int DataIn;

int DataOut;

};

struct BusData CurrentCycle[BUS\_CYCLES\_MAX];

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: // GP27 (Unused, seems inactive)

case 12: // GP12\_PWM0 (P14)

case 16:

case 17:

case 18:

case 20: // GP12\_PWM0, P14, (Unused, seems inactive)

case 22:

case 27:

case 28:

case 29:

case 30:

case 34:

case 39: // GP77, P16, (Unused, seems inactive)

case 42:

case 43:

case 44: // GP134, P15, (Unused, seems inactive)

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:

case 55: // USB\_PC5 (GP81)

case 40: // USB\_PC6 (GP82)

pin[i] = mraa\_gpio\_init(i); // Set as data inputs (From U6 and status bits)

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

}

}

// Initialize all output pin levels (just in case)

mraa\_gpio\_write(pin[TO\_BUS],LOW); // Start mode = S100 bus control by Edison

mraa\_gpio\_write(pin[E\_WAIT],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[USB\_SEL],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); // Send 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 indicate we are ready

sleep(1);

mraa\_gpio\_write(pin[EDISON\_READY],HIGH); // HIGH from Edison board -> LED D4 OFF 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 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 continuously display '3'\n");

while(TRUE)

{

WritePort(CON\_OUT\_PORT, 0x33); // Write a 3's to CON\_OUT\_PORT continuously

usleep(1000);

printf("Write '3's to Port 01H\n"); // Display current address

}

}

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

}

}

while(TEST5) // <--- USB PORT DIGNOSTIC TEST (Loops forever if active)

{

int c;

c = USBGetChar(); // Type a character at the USB input port (on your PC)

if(!c)

printf("USB Status Port Timeout (no character was typed at the USP serial input port)\n");

else USBPutChar(c);

}

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();

if(TO\_BUS\_FLAG) // Edison controls bus (TO\_BUS\_FLAG = TRUE)

{

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 S100 Board.$"); // Send test string to speech synthesizer

PrintString("Spoke:- This is a test of the Edison S100 Board.\r\n\n");

break;

case 'X':

PrintString("\r\nMode change to:- S100 Bus to Edison. Please use USB port for console I/O.\r\n\n");

TO\_BUS\_FLAG = FALSE;

mraa\_gpio\_write(pin[TO\_BUS],HIGH); // To switch direction we must RAISE to TO\_BUS

PrintMenuOptions();

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;

case 0x1A: // Ctrl-Z will abort the runtime S100\_Edison service

PrintString("\r\nExiting S100\_Edison. (Returning to the Linux root prompt).\r\n>");

Send\_Z80Reset();

sleep(1);

return MRAA\_SUCCESS;

break;

default:

PrintMenuOptions();

sprintf(buffer,"'%c' Menu option is not done yet.\r\n\n",c);

PrintString(buffer);

PutChar(BELL);

break;

}

}

else // Edison monitors S100 bus (TO\_BUS\_FLAG = FALSE)

{

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':

if(!SelectTrigger())

PutChar(BELL);

break;

case 'B':

if(!CaptureData(BUS\_CYCLES))

PutChar(BELL);

break;

case 'H':

if(!HaltCPU())

PutChar(BELL);

break;

break;

case 'K':

PutCRLF();

PrintSignon();

PrintMenuOptions(); // Show main menu

break;

case 'X':

PrintString("\r\nMode change to:- Edison to S100 Bus. Please use S100 bus for console I/O.\r\n");

TO\_BUS\_FLAG = TRUE;

mraa\_gpio\_write(pin[TO\_BUS],LOW); // To switch direction we must LOWER to TO\_BUS

PrintMenuOptions();

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");

PrintString("Please select menu option: ");

if(AbortFlag)

{

PrintString("\r\nCommand Aborted\b\r\n\r\n");

AbortFlag = FALSE;

}

c = toupper(GetChar());

if(c == ESC)

{

PutCRLF();

PutCRLF();

PrintMenuOptions();

return;

}

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 II S100 Bus Monitor V1.3 John Monahan (4/9/2017) ");

if(TO\_BUS\_FLAG)

{

c = ReadPort(IOBYTE);

sprintf(buffer, "IOBYTE = %x\r\n",c);

PrintString(buffer);

}

else PutCRLF();

}

void PrintMenuOptions()

{

if(TO\_BUS\_FLAG)

{

PrintString("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Mode = (Edison ----> S100 Bus) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\r\n\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 X=Change Mode Z=To Z80 ESC to abort \r\n\r\n");

}

else

{

PrintString("\_\_\_\_\_\_\_\_\_\_\_\_ Mode = (Edison <---- S100 Bus)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\r\n\n");

PrintString("A=Set Trigger Signal B=Capture Data H=Halt CPU \r\n");

PrintString("K=Menu X=Change Mode ESC to abort \r\n\r\n");

}

}

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

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

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

int HaltCPU()

{

char char\_buffer[100];

char c;

mraa\_gpio\_write(pin[E\_WAIT],LOW); // <----- Temporarily stop S100 bus cycles

mraa\_gpio\_write(pin[E\_MEMR],LOW); // Latch data into U7 (IO\_RAM\_RD)

mraa\_gpio\_write(pin[RW\_PULSE],LOW);

mraa\_gpio\_write(pin[RW\_PULSE],HIGH);

mraa\_gpio\_write(pin[E\_MEMR],HIGH);

GetCurrentAddressLines(0); // Get current address

GetCurrentStatusLine(0); // Get current bus status

GetCurrentDataIn(0); // Get current bus Data In

GetCurrentDataOut(0); // Get current bus Data Out

PrintString("\r\nS100 bus CPU is halted.\r\n");

sprintf(char\_buffer,"Address = %06lx %s Data in = %02x Data Out = %02x\r\n",CurrentCycle[0].AddressLine,CurrentCycle[0].StatusLineName,CurrentCycle[0].DataIn,CurrentCycle[0].DataOut);

PrintString(char\_buffer);

PrintString("Press ESC to continue");

while(TRUE)

{

if(GetStatus())

{

c = toupper(GetChar());

if(c == ESC)

{

mraa\_gpio\_write(pin[E\_WAIT],HIGH); // <----- cRelease again S100 bus cycles, try again

PutCRLF();

PutCRLF();

PrintMenuOptions();

return 0;

}

}

}

}

int CaptureData()

{

char char\_buffer[100];

int i,k;

char c;

if(TRIGGER\_SIGNAL < 0)

{

PrintString("\r\nA Trigger signal has not yet been set.\r\n");

k = SelectTrigger();

if((AbortFlag) || (!k))

return 0;

}

sprintf(char\_buffer,"\r\nSet number of bus cycles to capture (0-%d): ",BUS\_CYCLES\_MAX);

PrintString(char\_buffer);

k = GetDecimalValue();

if((AbortFlag) || (!k))

return 0;

PrintString("\r\nWaiting for Trigger Signal to go active....(Esc to abort)\r\n");

while(TRUE)

{

if(GetStatus())

{

c = toupper(GetChar());

if(c == ESC)

{

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); // Release OE\* on U7 (BUS\_STATUS\_READ\*)

mraa\_gpio\_write(pin[E\_sOUT],HIGH);

mraa\_gpio\_write(pin[E\_WAIT],HIGH); // <----- cRelease again S100 bus cycles, try again

sprintf(char\_buffer,"\r\nData collecting with Trigger Signal (%s) was aborted.\b\r\n",TriggerName);

PrintString(char\_buffer);

PutCRLF();

PutCRLF();

PrintMenuOptions();

return 0;

}

}

mraa\_gpio\_write(pin[E\_WAIT],LOW); // <----- Temporarily stop S100 bus cycles

sleep(1); // Not clear why but without this. I get false triggering with a MWRT trigger

// No false triggers with other status signals!

mraa\_gpio\_write(pin[E\_MEMR],LOW); // Latch data into U7 (IO\_RAM\_RD)

mraa\_gpio\_write(pin[RW\_PULSE],LOW); //

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); //

mraa\_gpio\_write(pin[E\_MEMR],HIGH); //

mraa\_gpio\_write(pin[E\_sOUT],LOW); // Activate OE\* of U7 (BUS\_STATUS\_READ\*)

mraa\_gpio\_write(pin[RW\_PULSE],LOW);

if(mraa\_gpio\_read(pin[TRIGGER\_SIGNAL])) // Test if Trigger is high yet

break;

mraa\_gpio\_write(pin[E\_sOUT],HIGH); // Deactivate OE\* of U7 (BUS\_STATUS\_READ\*)

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); //

mraa\_gpio\_write(pin[E\_WAIT],HIGH); // Release wait on S100 bus cycles

}

mraa\_gpio\_write(pin[E\_sOUT],HIGH); // Get Trigger, deactivate OE\* of U7 (BUS\_STATUS\_READ\*)

mraa\_gpio\_write(pin[RW\_PULSE],HIGH);

sprintf(char\_buffer,"\r\nDetected Trigger signal (%s). Collecting data....\b\r\n",TriggerName);

PrintString(char\_buffer);

BUS\_CYCLES = k;

for(i=0; i < k; i++)

{

mraa\_gpio\_write(pin[E\_WAIT],LOW); // <--------- Halt (if not already halted), S100 bus cycles

GetCurrentAddressLines(i); // Get current address

GetCurrentStatusLine(i); // Get current bus status

GetCurrentDataIn(i); // Get current bus Data In

GetCurrentDataOut(i); // Get current bus Data Out

mraa\_gpio\_write(pin[E\_WAIT],HIGH); // <--------- Release again S100 bus cycles, we are done

}

sprintf(char\_buffer,"\r\nCaptured data for %d CPU cycles using (%s) Trigger:-\r\n ",BUS\_CYCLES,TriggerName);

PrintString(char\_buffer);

for(i=0; i < BUS\_CYCLES; i++)

{

sprintf(char\_buffer,"\r\nAddress = %06lx %s Data in = %02x Data Out = %02x ",CurrentCycle[i].AddressLine,CurrentCycle[i].StatusLineName,CurrentCycle[i].DataIn,CurrentCycle[i].DataOut);

PrintString(char\_buffer);

}

PutCRLF();

return k; // Return with number of cycles captured

}

long GetCurrentAddressLines(int j) // Capture Address line data. Note assumes S100 Bus is halted

{ // Note normally the S100 bus will be in a wait state at this point

int i;

long k;

mraa\_gpio\_write(pin[ADDRESS1],LOW); // Latch the data on U32, U33 & U34

mraa\_gpio\_write(pin[ADDRESS1],HIGH);

for(i=0; i < 24; i++) // Get 24 bits of address data

{

switch(i)

{

case 0:

mraa\_gpio\_write(pin[ADDRESS2],LOW); // Select U32 OE\* via CPLD code (BUS\_ADD1\_READ\*)

mraa\_gpio\_write(pin[ADDRESS3],HIGH);

mraa\_gpio\_write(pin[RW\_PULSE],LOW); // Activate this CPLD selection.

if(mraa\_gpio\_read(pin[bDI0]))

k = 1;

else k = 0;

break;

case 1:

if(mraa\_gpio\_read(pin[bDI1]))

k |= 0x2;

break;

case 2:

if(mraa\_gpio\_read(pin[bDI2]))

k |= 0x4;

break;

case 3:

if(mraa\_gpio\_read(pin[bDI3]))

k |= 0x8;

break;

case 4:

if(mraa\_gpio\_read(pin[bDI4]))

k |= 0x10;

break;

case 5:

if(mraa\_gpio\_read(pin[bDI5]))

k |= 0x20;

break;

case 6:

if(mraa\_gpio\_read(pin[bDI6]))

k |= 0x40;

break;

case 7:

if(mraa\_gpio\_read(pin[bDI7]))

k |= 0x80;

break;

case 8:

mraa\_gpio\_write(pin[ADDRESS2],HIGH); // Select U33 OE\* via CPLD code (BUS\_ADD2\_READ\*)

mraa\_gpio\_write(pin[ADDRESS3],LOW);

mraa\_gpio\_write(pin[RW\_PULSE],LOW); // Activate this CPLD selection.

if(mraa\_gpio\_read(pin[bDI0]))

k |= 0x100;

break;

case 9:

if(mraa\_gpio\_read(pin[bDI1]))

k |= 0x200;

break;

case 10:

if(mraa\_gpio\_read(pin[bDI2]))

k |= 0x400;

break;

case 11:

if(mraa\_gpio\_read(pin[bDI3]))

k |= 0x800;

break;

case 12:

if(mraa\_gpio\_read(pin[bDI4]))

k |= 0x1000;

break;

case 13:

if(mraa\_gpio\_read(pin[bDI5]))

k |= 0x2000;

break;

case 14:

if(mraa\_gpio\_read(pin[bDI6]))

k |= 0x4000;

break;

case 15:

if(mraa\_gpio\_read(pin[bDI7]))

k |= 0x8000;

break;

case 16:

mraa\_gpio\_write(pin[ADDRESS2],LOW); // Select U34 OE\* via CPLD code (BUS\_ADD3\_READ\*)

mraa\_gpio\_write(pin[ADDRESS3],LOW);

mraa\_gpio\_write(pin[RW\_PULSE],LOW); // Activate this CPLD selection.

if(mraa\_gpio\_read(pin[bDI0]))

k |= 0x10000;

break;

case 17:

if(mraa\_gpio\_read(pin[bDI1]))

k |= 0x20000;

break;

case 18:

if(mraa\_gpio\_read(pin[bDI2]))

k |= 0x40000;

break;

case 19:

if(mraa\_gpio\_read(pin[bDI3]))

k |= 0x80000;

break;

case 20:

if(mraa\_gpio\_read(pin[bDI4]))

k |= 0x100000;

break;

case 21:

if(mraa\_gpio\_read(pin[bDI5]))

k |= 0x200000;

break;

case 22:

if(mraa\_gpio\_read(pin[bDI6]))

k |= 0x400000;

break;

case 23:

if(mraa\_gpio\_read(pin[bDI7]))

k |= 0x800000;

break;

}

} // All 24 lines done

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); // Clear all signals

mraa\_gpio\_write(pin[ADDRESS1],HIGH); // we now have the data

mraa\_gpio\_write(pin[ADDRESS2],HIGH);

mraa\_gpio\_write(pin[ADDRESS3],HIGH);

CurrentCycle[j].AddressLine = k; // Save the address info (globally) here

return k; // Return with the S100 bus still on hold (may want more info)

}

int GetCurrentStatusLine(int j) // Capture Status line data. Note assumes S100 Bus is halted

{

mraa\_gpio\_write(pin[E\_MEMR],LOW); // Latch data into U7 (IO\_RAM\_RD)

mraa\_gpio\_write(pin[RW\_PULSE],LOW); //

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); //

mraa\_gpio\_write(pin[E\_MEMR],HIGH); //

mraa\_gpio\_write(pin[E\_sOUT],LOW); // Activate OE\* of U7 (BUS\_STATUS\_READ\*)

mraa\_gpio\_write(pin[RW\_PULSE],LOW);

if(mraa\_gpio\_read(pin[bDI7])) // Put the most common one first

{

STATUS\_SIGNAL = bDI7; // This is the relevant data input pin to the Edison

strcpy(CurrentCycle[j].StatusLineName,"sMEMR");

}

else if(mraa\_gpio\_read(pin[bDI0]))

{

STATUS\_SIGNAL = bDI0;

strcpy(CurrentCycle[j].StatusLineName," MWRT");

}

else if(mraa\_gpio\_read(pin[bDI6]))

{

STATUS\_SIGNAL = bDI6;

strcpy(CurrentCycle[j].StatusLineName," sINP");

}

else if(mraa\_gpio\_read(pin[bDI2]))

{

STATUS\_SIGNAL = bDI2;

strcpy(CurrentCycle[j].StatusLineName,"sINTA");

}

else if(mraa\_gpio\_read(pin[bDI5]))

{

STATUS\_SIGNAL = bDI5;

strcpy(CurrentCycle[j].StatusLineName," sOUT");

}

else

{

STATUS\_SIGNAL = 0;

strcpy(CurrentCycle[j].StatusLineName," ");

}

mraa\_gpio\_write(pin[E\_sOUT],HIGH); // Clear all signals

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); // we now have the data

return STATUS\_SIGNAL; // Return with 0 if no status lines active

}

int GetCurrentDataIn(int j) // Capture Data In Lines. Note assumes S100 Bus is halted

{

int i;

int k;

mraa\_gpio\_write(pin[E\_MEMR],LOW); // Latch data into U22 (IO\_RAM\_RD)

mraa\_gpio\_write(pin[RW\_PULSE],LOW); //

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); //

mraa\_gpio\_write(pin[E\_MEMR],HIGH); //

mraa\_gpio\_write(pin[E\_sINTA],LOW); // Activate OE\* of U22 (BUS\_DI\_READ\*)

mraa\_gpio\_write(pin[RW\_PULSE],LOW);

for(i=0; i < 8; i++) // Get 8 bits from current data in lines data

{

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 |= 0x2;

break;

case 2:

if(mraa\_gpio\_read(pin[bDI2]))

k |= 0x4;

break;

case 3:

if(mraa\_gpio\_read(pin[bDI3]))

k |= 0x8;

break;

case 4:

if(mraa\_gpio\_read(pin[bDI4]))

k |= 0x10;

break;

case 5:

if(mraa\_gpio\_read(pin[bDI5]))

k |= 0x20;

break;

case 6:

if(mraa\_gpio\_read(pin[bDI6]))

k |= 0x40;

break;

case 7:

if(mraa\_gpio\_read(pin[bDI7]))

k |= 0x80;

break;

}

} // All 8 lines done

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); // Clear all signals

mraa\_gpio\_write(pin[E\_sINTA],HIGH); // we now have the data

CurrentCycle[j].DataIn = k; // Save the info (globally) here

return k; // Return with the S100 bus still on hold (may want more info)

}

int GetCurrentDataOut(int j) // Capture Data In Lines. Note assumes S100 Bus is halted

{

int i;

int k;

mraa\_gpio\_write(pin[E\_MEMR],LOW); // Latch data into U22 (IO\_RAM\_RD)

mraa\_gpio\_write(pin[RW\_PULSE],LOW); //

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); //

mraa\_gpio\_write(pin[E\_MEMR],HIGH); //

mraa\_gpio\_write(pin[E\_PSYNC],LOW); // Activate OE\* of U22 (BUS\_DI\_READ\*)

mraa\_gpio\_write(pin[RW\_PULSE],LOW);

for(i=0; i < 8; i++) // Get 8 bits from current data in lines data

{

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 |= 0x2;

break;

case 2:

if(mraa\_gpio\_read(pin[bDI2]))

k |= 0x4;

break;

case 3:

if(mraa\_gpio\_read(pin[bDI3]))

k |= 0x8;

break;

case 4:

if(mraa\_gpio\_read(pin[bDI4]))

k |= 0x10;

break;

case 5:

if(mraa\_gpio\_read(pin[bDI5]))

k |= 0x20;

break;

case 6:

if(mraa\_gpio\_read(pin[bDI6]))

k |= 0x40;

break;

case 7:

if(mraa\_gpio\_read(pin[bDI7]))

k |= 0x80;

break;

}

} // All 8 lines done

mraa\_gpio\_write(pin[RW\_PULSE],HIGH); // Clear all signals

mraa\_gpio\_write(pin[E\_sINTA],HIGH); // we now have the data

CurrentCycle[j].DataOut = k; // Save the info (globally) here

return k; // Return with the S100 bus still on hold (may want more info)

}

int SelectTrigger()

{

char c;

while(TRUE)

{

PutCRLF();

PrintString(">>>>>>> Select Trigger Menu <<<<<<<\r\n\n");

PrintString("1 sINP (S100 bus High, Pin 46) \r\n");

PrintString("2 sOUT (S100 bus High, Pin 45) \r\n");

PrintString("3 sMEMR (S100 bus High, Pin 47) \r\n");

PrintString("4 MEMW (S100 bus High, Pin 68) \r\n");

PrintString("5 sINTA (S100 bus High, Pin 96) \r\n");

PrintString("ESC To return to the main menu\r\n\n");

PrintString("Please select menu option: ");

if(AbortFlag)

{

PrintString("\r\nCommand Aborted.\b\r\n\r\n");

AbortFlag = FALSE;

}

c = toupper(GetChar());

if(c == ESC)

{

PutCRLF();

PutCRLF();

PrintMenuOptions();

TRIGGER\_SIGNAL = -1;

return -1;

}

PutChar(c);

switch(c)

{

case '1': // sINP (S100 bus High, Pin 46)

PrintString("\r\nS100 bus line trigger = sINP (LOW -> HIGH).\r\n>");

TRIGGER\_SIGNAL = bDI6; // This is the relevant data input pin to the Edison

strcpy(TriggerName,"sINP"); // Used in CaptureData() etc (U18).

PutCRLF();

PrintMenuOptions();

return TRIGGER\_SIGNAL;

case '2': // sOUT (S100 bus High, Pin 45)

PrintString("\r\nS100 bus line trigger = sOUT (LOW -> HIGH).\r\n>");

TRIGGER\_SIGNAL = bDI5;

strcpy(TriggerName,"sOUT");

PutCRLF();

PrintMenuOptions();

return TRIGGER\_SIGNAL;

case '3': // sMEMR (S100 bus High, Pin 47)

PrintString("\r\nS100 bus line trigger = sMEMR (LOW -> HIGH).\r\n>");

TRIGGER\_SIGNAL = bDI7;

strcpy(TriggerName,"sMEMR");

PutCRLF();

PrintMenuOptions();

return TRIGGER\_SIGNAL;

case '4': // MEMW (S100 bus High, Pin 68)

PrintString("\r\nS100 bus line trigger = MEMW (LOW -> HIGH).\r\n>");

TRIGGER\_SIGNAL = bDI0;

strcpy(TriggerName,"MEMW");

PutCRLF();

PrintMenuOptions();

return TRIGGER\_SIGNAL;

case '5': // sINTA (S100 bus High, Pin 96)

PrintString("\r\nS100 bus line trigger = sINTA (LOW -> HIGH).\r\n>");

TRIGGER\_SIGNAL = bDI2;

strcpy(TriggerName,"sINTA");

PutCRLF();

PrintMenuOptions();

return TRIGGER\_SIGNAL;

default: // Default return to the main menu

PrintString("\r\nInvalid menu Option.\r\n");

TRIGGER\_SIGNAL = -1;

break;

}

}

return TRIGGER\_SIGNAL;

}

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++)

{

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;

}

}

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

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

// 3 lines need to be activated. Normally they are inactive.

// 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++)

{

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;

}

char ReadUSBPort() // Read a byte from the on-board USB Port

{

int i;

char k;

mraa\_gpio\_write(pin[USB\_SEL],LOW); // Select on-board USB port

mraa\_gpio\_write(pin[E\_sINP],LOW); // Activate with a low pulse

for(i=0; i < 8; i++)

{

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;

}

}

mraa\_gpio\_write(pin[E\_sINP],HIGH); // Deactivate with a high to the CPLD

mraa\_gpio\_write(pin[USB\_SEL],HIGH);

return k;

}

void WriteUSBPort(char value) // Write a byte to the on-board USB Port

{

int i;

int k;

mraa\_gpio\_write(pin[USB\_SEL],LOW); // Select on-board USB port

mraa\_gpio\_write(pin[E\_sOUT],LOW); // Activate with a low pulse

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;

}

}

mraa\_gpio\_write(pin[E\_sOUT],HIGH); // Deactivate with a High to the CPLD

mraa\_gpio\_write(pin[USB\_SEL],HIGH);

}

char USBGetChar()

{

char c;

long i;

for(i=0; i< 10000000; i++) // Try 10000000 times

{

if(!GetUSBInStatus()) // Wait for a character (RXF#, HIGH=empty,LOW when a character arrives)

{

c = ReadUSBPort(); // Then get the character

return c;

}

}

return 0;

}

void USBPutChar(char c)

{

long i;

for(i=0; i< 10000000; i++) // Try 10000000 times

{

if(!GetUSBOutStatus()) // Wait until not busy (TXE#, HIGH while busy, do not write until LOW)

{

WriteUSBPort(c); // If so send the character

return;

}

}

return;

}

char GetUSBInStatus() // RXF#, Read Status In for USB Port

{ // >>>> LOW/0 if byte available <<<

char i;

i = mraa\_gpio\_read(pin[USB\_PC7]);

return i;

}

char GetUSBOutStatus() // TXF#, Read Status Out for USB Port

{ // >>>> HIGH/1 if busy <<<<

char i;

i = mraa\_gpio\_read(pin[USB\_PC6]);

return i;

}

void PrintString(char\* TextString) // Print a string on the console or USB port

{

while(\*TextString)

{

char p;

p = \*TextString++;

PutChar(p);

}

}

void PutCRLF() // Send CR + LF to Console or USB port

{

PutChar(CR);

PutChar(LF);

}

void PutChar(char c) // Print a character on the Console OR USB PORT

{

if(TO\_BUS\_FLAG)

{

while(!(ReadPort(CON\_STATUS\_PORT) & 0x04));

WritePort(CON\_OUT\_PORT,c);

}

else

{

while (GetUSBOutStatus()); // Wait until not busy (TXE#, HIGH while busy, do not write )

WriteUSBPort(c); // If so get the character

}

}

char GetChar() // Get a keyboard character from the Console or USB port

{

char c;

if(TO\_BUS\_FLAG)

{

while(!GetStatus()); // Check if a character is available

c = ReadPort(CON\_IN\_PORT); // Check if a character is available

return c;

}

else

{

while (GetUSBInStatus()); // Wait for a character (RXF#, HIGH=empty,LOW when a character arrives)

c = ReadUSBPort(); // Then get the character

}

return c;

}

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

{ // Return Non ZERO if something there

int c;

if(TO\_BUS\_FLAG)

{

c = ReadPort(CON\_STATUS\_PORT); // Check if a character is available (Non zero if char. available)

return(c &= 0x02);

}

else

{

if(!GetUSBInStatus())

return 1;

else return 0;

}

}

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()

{

if(TO\_BUS\_FLAG) // If Edison is controlling bus this command is active as a reset CMD

{ // If a monitor it is ignored here.

WritePort(0xEE,0); // Reset the TMA lines 1,2 & 3.

mraa\_gpio\_write(pin[E\_WAIT],LOW); // Activate the above command with a low pulse to the CPLD

mraa\_gpio\_write(pin[E\_WAIT],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 ToBusChange() // For edge triggered single step request

// {

// TO\_BUS\_FLAG = mraa\_gpio\_read(pin[TO\_BUS]);

// printf("ToBus\_Flag = %x \n",TO\_BUS\_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");

}

}