Embedded Control Game Report

Michael Stark & David Melecio-Vázquez

Section 4, Side A (TA: Xuemei Gao)

Due: 8 October 2010

Table of Contents

[I. Introduction and Statement of Purpose 3](#_Toc274122655)

[II. System Development 3](#_Toc274122656)

[a. Hardware Details 3](#_Toc274122657)

[b. Software Details 3](#_Toc274122658)

[III. Results and Conclusions 3](#_Toc274122659)

[IV. References 3](#_Toc274122660)

[V. Appendices 4](#_Toc274122661)

[a. C code 4](#_Toc274122662)

[b. Hardware schematic 18](#_Toc274122663)

[c. Participation Summary 18](#_Toc274122664)

1. Introduction and Statement of Purpose
2. System Development
   1. Hardware Details
   2. Software Details
3. Results and Conclusions
4. References
5. Appendices
   1. C code

Following is a copy of our file lab2.c.

/\*

\* Names: Michael Stark + David Melecio-Vázquez

\* Section: 4 A

\* Date: 14 September 2010

\* File name: lab2.c

\* Description: Play a game of LITEC Memory with the user.

\*/

#include <c8051\_SDCC.h>// include files. This file is available online

#include <stdio.h>

#include <stdlib.h>

//-----------------------------------------------------------------------------

// Function Prototypes

//-----------------------------------------------------------------------------

// Initialization Functions

void Port\_Init(void); // Initialize ports for input and output

void Timer\_Init(void); // Initialize Timer 0

void ADC\_Init(void); // Initialize A/D Conversion

unsigned char Read\_Port\_1(void); // Performs A/D Conversion

void Interrupt\_Init(void); // Initialize interrupts

void Timer0\_ISR(void) interrupt 1; // Called at Timer0 overflow

// Game Functions

void play\_game(void); // Plays the LITEC Memory game.

int CalculateMaxCounts(unsigned char x); // Converts the port 1 result into a number of overflows to wait for.

unsigned char light\_LED(unsigned char LED\_to\_light, short on\_time, short off\_time); // Light the designated LED for on\_time, then wait for off\_time

unsigned char ReadPushbuttons(); // Wait for a pushbutton event, and return which one was pressed

unsigned char unique\_random(unsigned char last\_state); // Generates a random number different from the last one

unsigned char random(unsigned char N); // Generates a random number between 0 and N-1

void wait\_one\_second(void); // Waits 1 second

void light\_green(void); // Light BiLED green

void light\_red(void); // Light BiLED red

int CheckPushButton1(void); // function which checks push button 1

int CheckPushButton2(void); // function which checks push button 2

int CheckPushButton3(void); // function which checks push button 3

int CheckPushButton4(void); // function which checks push button 4

void PrintInputStatus(int push1, int push2, int push3, int push4); // Helper function to print the status of the inputs.

char\* newline(); // Helper function, used when printing (returns "\r\n")

//-----------------------------------------------------------------------------

// Global Variables

//-----------------------------------------------------------------------------

// Pushbuttons (Inputs)

sbit at 0xA0 PB3; // Push button 3, associated with Port 2, Pin 0 (second from left, red PB)

sbit at 0xA1 PB4; // Push button 4, associated with Port 2, Pin 1 (far left, black PB)

sbit at 0xA2 PB1; // Push button 1, associated with Port 2, Pin 2 (far right, red PB)

sbit at 0xA3 PB2; // Push button 2, associated with Port 2, Pin 3 (second from right, black PB)

// LEDs and Buzzer (Outputs)

sbit at 0xB2 LED2; // LED2, associated with Port 3, Pin 2

sbit at 0xB3 BILED0; // BILED0, associated with Port 3 Pin 3

sbit at 0xB4 BILED1; // BILED1, associated with Port 3 Pin 4

sbit at 0xB5 LED1; // LED1, associated with Port 3 Pin 5

sbit at 0xB6 LED0; // LED0, associated with Port 3 Pin 6

sbit at 0xB7 BUZZER; // Buzzer, associated with Port 3 Pin 7

unsigned int Counts = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void main(void) {

Sys\_Init(); // System Initialization

Port\_Init(); // Initialize ports 2 and 3

Interrupt\_Init();

Timer\_Init(); // Initialize Timer 0

ADC\_Init(); // Initialize A/D Conversion

putchar(' '); // the quote fonts may not copy correctly into SiLabs IDE

putchar('\r');

// Enable Timer 0

TR0 = 1;

while (!CheckPushButton4());

while (1) {

play\_game();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Port\_Init(void) {

// Port 1 Constant Masks

unsigned char P1MDIN\_LO = 0xFD; // 1111 1101, Set P1.1 as an analog input

unsigned char P1MDOUT\_LO = 0xFD; // 1111 1101, Set P1.1 as a input port bit

unsigned char P1\_HI = 0x02; // 0000 0010 Set P1.1 to a high impedance state

// Port 2 Constant Masks

unsigned char P2MDOUT\_LO = 0xF0; // 1111 0000

unsigned char P2\_HI = 0x0F; // 0000 1111

// Port 3 Constant Masks

unsigned char P3MDOUT\_HI = 0xFC; // 1111 1100

// Set Port 1 analog input bits

P1MDIN &= P1MDIN\_LO;

// Set Port 1 output (low) bits

P1MDOUT &= P1MDOUT\_LO;

// Set Port 1 impedence (high) bits

P1 |= P1\_HI;

// Set Port 2 MDOUT high bits

// P2MDOUT |= P2MDOUT\_HI;

// Set Port 2 MDOUT low bits

P2MDOUT &= P2MDOUT\_LO;

// Set Port 2 impedence (high) bits

P2 |= P2\_HI;

// Set Port 3 MDOUT high bits

P3MDOUT |= P3MDOUT\_HI;

// Set Port 3 MDOUT low bits

// P3MDOUT &= P3MDOUT\_LO;

// Set Port 3 impedence (high) bits

// P3 |= P3\_HI;

}

void Interrupt\_Init(void) {

IE |= 0x82; // 1000 0010

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Timer\_Init(void) {

CKCON |= 0x08; // 0000 1000 (use SYSCLK)

TMOD &= 0xF0; // clear the 4 least significant bits (1111 0000)

TMOD |= 0x01; // 0000 0001 (Timer0 in mode 1, for 16-bit)

TR0 = 0; // Stop Timer0

TL0 = 0; // Clear low byte of register T0

TH0 = 0; // Clear high byte of register T0

}

void ADC\_Init(void) {

REF0CN &= 0xF7; // 1111 0111 Configure ADC1 to use VREF

REF0CN |= 0x03; // 0000 0011

ADC1CF = 0x01; // 0000 0001 Set a gain of 1

ADC1CN |= 0x80; // 1000 0000 Enable ADC1

}

unsigned char Read\_Port\_1(void) {

AMX1SL = 0x01; // 0000 0001 Set the Port pin number

ADC1CN &= 0xDF; // 1101 1111 Clear the flag from the previous ADC1 conversion

ADC1CN |= 0x10; // 0001 0000 Start A/D Conversion

while ((ADC1CN & 0x20) == 0x00); // Wait for conversion to be complete

return ADC1; //Assign the A/D conversion result

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Timer0\_ISR(void) interrupt 1 {

TF0 = 0; // clear interrupt request

Counts++;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Play the LITEC Memory game. Returns if PB4 is ever pressed.

\*/

void play\_game(void) {

unsigned char turns[5], inputs[5], port1\_result, i, correct;

unsigned short on\_max\_counts, off\_max\_counts;

// Turn off all outputs

LED0 = 1;

LED1 = 1;

LED2 = 1;

BUZZER = 1;

BILED0 = 1;

BILED1 = 1;

port1\_result = Read\_Port\_1();

on\_max\_counts = CalculateMaxCounts(port1\_result);

off\_max\_counts = (on\_max\_counts / 2);

turns[0] = unique\_random(3);

for (i = 1; i < 5; i++) {

turns[i] = unique\_random(turns[i - 1]);

}

// Light LED sequence

for (i = 0; i < 5; i++) {

// Restart the game if the player presses PushButton4.

if (light\_LED(turns[i], on\_max\_counts, off\_max\_counts) == 1) {

return;

}

}

light\_green();

correct = 1;

for (i = 0; i < 5; i++) {

inputs[i] = ReadPushbuttons();

if (inputs[i] == 3) { // Pushbutton 4 pressed, restart game

return;

} else if (inputs[i] != turns[i]) {

correct = 0;

printf("Input %d was incorrect!%s", i + 1, newline());

break;

} else {

printf("Input %d was correct!%s", i + 1, newline());

}

}

if (correct) {

// flash LEDs 3 times

for (i = 0; i < 3; i++) {

// Turn LEDs on

LED0 = 0;

LED1 = 0;

LED2 = 0;

// Wait about a quarter of a second

Counts = 0;

while (Counts < 75);

// Turn LEDs off

LED0 = 1;

LED1 = 1;

LED2 = 1;

// Wait about a quarter of a second

Counts = 0;

while (Counts < 75);

}

} else {

// sounds buzzer for 1.5 seconds

BUZZER = 0;

light\_red();

Counts = 0;

while (Counts < 506);

BUZZER = 1; // Turn buzzer back off (thank God)

}

// Turn off BILED

BILED0 = 1;

BILED1 = 1;

while (!CheckPushButton4());

}

/\*

\* Returns the number of overflows to wait for in on\_time.

\*

\* NOTE: off\_time is simply (on\_time / 2).

\*/

int CalculateMaxCounts(unsigned char x) {

int on\_time\_millis = (x \* 5) + 200;

// This gives us the milliseconds to wait for. In 16 bit counting mode,

// based on SYSCLK, there are exactly 337.5 overflows per second, or

// 0.3375 overflows per millisecond. We multiply by this constant

// to convert from milliseconds to overflows.

return (on\_time\_millis \* 0.3375);

}

/\*

\* Light the LED for on\_time, then wait for off\_time. Returns 0 if successful, or 1 if PushButton4 was pressed.

\*/

unsigned char light\_LED(unsigned char LED\_to\_light, short on\_time, short off\_time) {

switch(LED\_to\_light) {

case 0:

// Light LED0

LED0 = 0;

break;

case 1:

// Light LED1

LED1 = 0;

break;

case 2:

// Light LED2

LED2 = 0;

break;

default:

printf("Invalid LED input: %d%s", LED\_to\_light, newline());

}

// Wait for on\_time

Counts = 0;

while (Counts < on\_time) {

if (CheckPushButton4()) {

return 1;

}

}

// Turn LEDs off again

LED0 = 1;

LED1 = 1;

LED2 = 1;

// Wait for off\_time

Counts = 0;

while (Counts < off\_time) {

if (CheckPushButton4()) {

return 1;

}

}

return 0; // Pushbutton 4 not pressed

}

/\*

\* Waits for a pushbutton to be pressed, then returns which button was pressed (an integer between 0 and 3).

\*/

unsigned char ReadPushbuttons() {

while (1) {

if (CheckPushButton1()) {

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely pressed.

if (CheckPushButton1()) { // Check again, to make sure it was no accident

while (CheckPushButton1()); // Wait until the button is released.

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely released.

return 0;

}

}

if (CheckPushButton2()) {

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely pressed.

if (CheckPushButton2()) { // Check again, to make sure it was no accident

while (CheckPushButton2()); // Wait until the button is released.

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely released.

return 1;

}

}

if (CheckPushButton3()) {

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely pressed.

if (CheckPushButton3()) { // Check again, to make sure it was no accident

while (CheckPushButton3()); // Wait until the button is released.

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely released.

return 2;

}

}

if (CheckPushButton4()) {

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely pressed.

if (CheckPushButton4()) { // Check again, to make sure it was no accident

while (CheckPushButton4()); // Wait until the button is released.

Counts = 0;

while (Counts < 25); // Wait for PushButton to be completely released.

return 3;

}

}

}

}

/\*

\* Returns a random number different from last\_state.

\*/

unsigned char unique\_random(unsigned char last\_state) {

unsigned char return\_value;

do {

return\_value = random(3);

} while(return\_value == last\_state);

return return\_value;

}

/\*

\* Returns a random integer between 0 and N-1 (a range of N numbers).

\*/

unsigned char random(unsigned char N) {

return (rand() % N);

}

/\*

\* Waits 1 second.

\*/

void wait\_one\_second(void) {

Counts = 0;

while (Counts < 338);

}

/\*

\* Light BiLED green.

\*/

void light\_green(void) {

BILED0 = 0;

BILED1 = 1;

}

/\*

\* :Light BiLED red.

\*/

void light\_red(void) {

BILED0 = 1;

BILED1 = 0;

}

/\*

\* Returns a 0 if CheckPushButton1 not activated

\* or a 1 if CheckPushButton1 is activated.

\* This code reads a single input only, associated with PB1

\*/

int CheckPushButton1(void) {

// !PB1 will evaluate to 0 if PB1 is off, or 1 if PB1 is on. These are the

// desired return values, so we'll just return the statement.

return !PB1;

}

/\*

\* Returns a 0 if push button 2 not activated

\* or a 1 if push button 2 is activated.

\* This code reads a single input only, associated with PB2

\*/

int CheckPushButton2(void) {

// !PB2 will evaluate to 0 if PB2 is off, or 1 if PB2 is on. These are the

// desired return values, so we'll just return the statement.

return !PB2;

}

/\*

\* Returns a 0 if push button 3 not activated

\* or a 1 if push button 3 is activated.

\* This code reads a single input only, associated with PB3

\*/

int CheckPushButton3(void) {

// !PB3 will evaluate to 0 if PB3 is off, or 1 if PB3 is on. These are the

// desired return values, so we'll just return the statement.

return !PB3;

}

/\*

\* Returns a 0 if push button 4 not activated

\* or a 1 if push button 4 is activated.

\* This code reads a single input only, associated with PB4

\*/

int CheckPushButton4(void) {

// !PB4 will evaluate to 0 if PB4 is off, or 1 if PB4 is on. These are the

// desired return values, so we'll just return the statement.

return !PB4;

}

/\*

\* Prints a message to standard output indicating the status of the Slide

\* switch and Pushbuttons 1 and 2.

\*/

void PrintInputStatus(int push1, int push2, int push3, int push4) {

LED0 = 1;

LED1 = 1;

LED2 = 1;

BILED0 = 1;

BILED1 = 1;

BUZZER = 1;

if (push1) {

printf("Pushbutton 1 enabled, ");

LED0 = 0;

}

if (push2) {

printf("Pushbutton 2 enabled, ");

LED1 = 0;

}

if (push3) {

printf("Pushbutton 3 enabled, ");

LED2 = 0;

}

if (push4) {

printf("Pushbutton 4 enabled");

light\_red();

BUZZER = 0;

}

printf("%s", newline());

}

/\*

\* Helper function to return a string containing a newline character and return

\* character.

\*/

char\* newline() {

char\* retval = "\r\n";

return retval;

}

* 1. Hardware schematic
  2. Participation Summary