Before Class

- Install SDCC – Instructions in “Installing_SiLabs-SDCC-Drivers” document
- Install SiLabs – Instructions in “Installing_SiLabs-SDCC-Drivers” document
- Install SecureCRT – On LMS, also available online by going to the computer store site.
  LMS: Course Resources>>Software&Drivers>>securecrt-rpi_5-5.exe
  (or http://www.rpi.edu/dept/arc/web/software/sw_available.html)
- Download the c8051_SDCC.h header file – Available on the front page of the course LMS web page
- Install USB-serial driver – Link in Homework 1

Solutions to Number Systems Worksheet

1) 0000 1110
2) 1011 1101
3) 0001 0000 0001
4) 0001 1101 1000
5) 0x0E
6) 0xBD
7) 0x101
8) 0x1D8
9) 55
10) 171
11) 327
12) 10977
13) 0011 0111
14) 1010 1011
15) 0000 0001 0100 0111
16) 0010 1010 1110 0001
17) 0x2D
18) 0xAA
19) 0xE3
20) 0x29B5
21) 45
22) 170
23) 227
24) 10677

Announcements

- Homework 2 due next week (before 3rd class)
  - Considered late if turned in after class starts
- If you are still not registered for this class, please take care of it.
- TAs
  - We are still working on this – stay tuned

Overview

- Homework softcopy
- Compile, download, and run Homework #1
- Edit code, compile, download, and run
- Discussion of variable types
- Operators
- LOGIC Worksheet
- Labs 1 and 2 Overview
- Common digital gates
- Some basic circuitry

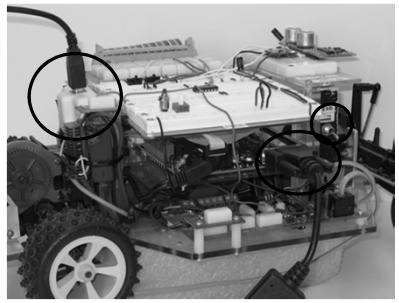
SiLabs IDE Software

- Microcontroller commands are converted to Assembly
  - We will be writing code in the C programming language and compiling it into Assembly using the SiLabs IDE and the SDCC compiler.
  - The SiLabs IDE will check for syntax errors, compile the code, and download it into the microcontroller flash memory
- The microprocessor will execute the programs stored in its memory, and does not need to be connected to any other computer to function – the processor is "embedded" in the system
  - Many of the programs we will write in this course, though, will have user interaction in the form of text printed on a screen and keyboard inputs
  - Feedback can be provided by other means such as turning on/off LEDs or buzzers or making adjustments to the motors

Communication Connections

- The serial port found on the C8051 board edge should be connected with a serial cable to the computer running SecureCRT or Hyperterminal
  - This is for text communication with the microcontroller (printf, getchar, etc.)
- The USB adaptor strapped behind the C8051 should be connected with a USB cable to the computer running the SiLabs IDE
  - This is for downloading the code
- Turn on power to C8051

TURN OFF AFTER DONE!!



Running Silicon Labs IDE

- Refer to instructions on LMS (also in Installing_SiLabs-SDCC-Drivers PDF)
  - Under Course Materials -- Lab1 -- "How to run codes"
    - Running the listed code listed in this handout is optional. It proves that everything is working. You are required to run the homework 1 code.
- Run SiLabs IDE, Click on Project – New Project
  - Device family – C8051F02x
  - Give it a name
  - Pick a location – strongly recommend creating a folder for projects
- Add Files to Project
  - Suggestion – right click on "Header Files" and remove, same for "Source Files"
  - Right click in white area on left and "Add files to Project ___
    - Load your .c file then right click the file to "Add *name.c* to build"
- Things to check
  - Options – connection options – USB Debug Adapter
- Click the Icons
  - Connect
  - Assemble/compile
  - Build/Make
  - Download
  - Go/Stop

Running HW1 Code

- Make sure you have installed the software completely
  - Read instructions from course website
  - Be sure to have saved the c8051_SDCC.h header file in C:\Program Files (x86)\SDCC\include\mcs51
- Connect the USB serial port adaptor – check device manager to get port numbers
  - A COM port will be used by SecureCRT or HyperTerminal to receive feedback
- Open the SecureCRT software, and set the COM port
- Compile/Build/Download
  - Note the error messages – remember the line numbers being flagged
  - Run the program for imax values of 18 and 270 by entering '1' or '2' at the prompt.
- Observe the results (what is wrong?)

Modify It, Run It Again!!

- Change variable count from *unsigned character* type to *integer* type.
  - Check lab manual index for printf syntax
  - %d is for printing signed decimal values
  - %u is for an unsigned decimal value
  - %x is for hexadecimal values (note: no '0x' is printed)
  - %c is for printing single byte character values
- Compile, download and run for the same values of imax (18 and 270).
- Note the differences.

Appropriate Variable Choices

- Why do we care?
  - Embedded controllers become more expensive as their memory requirements increase.
  - The system also becomes more complicated.
- What should we do to help ourselves?
  - Always use the smallest variable type consistent with our objective.
- Recommended practices:
  - Use unsigned char for digital input and output (I/O) and the analog-to-digital conversion subsystem
  - Can use the bit variable type for simple T/F variables
  - Use int or unsigned int when larger variables needed
  - Use long and float types as a last resort

Appropriate Variable Choices

(Cx51 C compiler manual & in LITEC Lab Manual)

Data Type	Bits	Bytes	Value Range
bit †	1		0 to 1
signed char	8	1	-128 to +127
unsigned char	8	1	0 to 255
enum	8 / 16	1 or 2	-128 to +127 or -32768 to +32767
signed short	16	2	-32768 to +32767
unsigned short	16	2	0 to 65535
signed int	16	2	-32768 to +32767
unsigned int	16	2	0 to 65535
signed long	32	4	-2147483648 to 2147483647
unsigned long	32	4	0 to 4294967295
float	32	4	±1.175494E-38 to ±3.402823E+38
sbit †	1		0 to 1
sfr †	8	1	0 to 255
sfr16 †	16	2	0 to 65535

† The **bit**, **sbit**, **sfr**, and **sfr16** data types are not provided in ANSI C and are unique to **Cx51**.

Operators

- There are many operators available in C to perform operations on numerical values
  - mathematical: +, -, *, /, % (mod – remainder)
  - relational tests: <, >, <=, >=
  - equality tests: ==, !=
  - logical tests: || (OR), && (AND)
  - bitwise: & (AND), | (OR), ^ (XOR), << (left shift), >> (right shift)
  - unary: ++ (increment), -- (decrement), ~ (ones complement), ! (logical negation)
  - others listed in lab manual Chapter 3

True/False Logic

- When the computer performs decision making, it evaluates whether a given argument is True or False
  - For binary operations, we are familiar with the following concept
    - 0 → False
    - 1 → True

  For example, the following lines of code will only print "Goodbye" (and return to the next line)

```
if (0) printf("Hello \r\n");
if (1) printf("Goodbye \r\n");
```


True/False Logic

- The C code is a little more flexible than simple binary inputs for the argument. Anything that is not a "0" is interpreted as a "true"

  For example, the following lines of code will print both "Hello" and "Goodbye"

```
if (1) printf("Hello \r\n");
if (9) printf("Goodbye \r\n");
```


Operators

- Examples of usage in programming:

```
A = 0x09; B = 0x05; C = 0x01;
if (A < B) C++;  /* C++ is short form of C=C+1; */
else C=C<<2;
```

  - If A were less than B, then C would be incremented by 1 (C=0x02),
  - Since A is greater than B, C all of the bits in C are shifted to the left by 2 bits (C=0x04)

C = 0000 0001 << 2

C = 0000 0100 = 0x04

Note: Values of added bits are always '0'

Operators

- Examples of usage in programming:

```
A = 0x09; B = 0x05; C = 0x01;
if (A == B) C+=3;  /* C+=3; is short form of C=C+3; */
else C=~C;
```

  - If A were equal to B, then 3 would be added to C
    - C += 3; is same as C = C + 3;
  - Since A is not equal to B, the ones complement is taken of C (C = 0xFE)

~C = ~(0000 0001)

~C = 1111 1110 = 0xFE

Note: each bit is 'flipped'

Operators

- Examples of usage in programming:

```
A = 0x09; B = 0x05; C = 0x00;
(Using digital logic, the above variables may
   be considered as: A=true, B=true, C=false)
D = A || C;    \\ Byte operation,
E = A | B;     \\ Bit operation
F = A && C;    \\ Byte operation
G = A & B;     \\ Bit operation
```

  In this example, both D and F are single bit variables whereas E and G are byte variables
- D and F use LOGICAL operations:
  - Note: 0 is false; 1 (or anything else) is true

  D: A is true, C is false, therefore, D is true

  F: A is true, C is false, therefore, F is false

Truth Table: OR
X | Y = Q

X	Y	Q
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table: AND
X & Y = Q

X	Y	Q
0	0	0
0	1	0
1	0	0
1	1	1

Operators

- Examples of usage in programming:

```
A = 0x09; B = 0x05; C = 0x00;
D = A || C; \\ Byte operation
E = A | B;   \\ Bit operation
F = A && C; \\ Byte operation
G = A & B;  \\Bit operation
```

- E and G use BITWISE operations:

```
    0000 1001
  | 0000 0101   <- Conduct operation on bits in each column
E = 0000 1101  = 0x0D

    0000 1001
  & 0000 0101
G = 0000 0001  = 0x01
```

Truth Table: OR
X | Y = Q

X	Y	Q
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table: AND
X & Y = Q

X	Y	Q
0	0	0
0	1	0
1	0	0
1	1	1

Application to Microcontroller

- You can use these operators to read or change the state of a bit or groups of bits in an I/O port.
  - Make pin 2 of Port 2 (P2.2) high and leave others as they were: `P2 = P2 | 0x04;`

```
P2:  XXXX XXXX   (X can be 0 or 1)
0x04: 0000 0100  conduct a bitwise OR operation
      XXXX X1XX  Note: P2.2 is high, rest are unchanged
```

  - Alternate form of writing this: `P2 |= 0x04;`

Application to Microcontroller

- Another application – testing the status on one or more input pins.
  - Test if pin 4 of Port 3 (P3.4) is high or low:
  - Statement `if(P3)` will check status of *entire* port
    - If P3 = 0000 0000, statement is false; anything else is true
  - To check one pin (P3.4), use: `test = P3 & 0x10;`

```
P3:  XXXX XXXX  (X can be 0 or 1)
0x10: 0001 0000  conduct a bitwise AND operation
test = 000X 0000  Note: just status of pin 4 is preserved
```

  test will be true if P3.4 was high, or false if it was low
  - This is called *bit-masking* (the 0x10 is the mask value)

Lab 1-1 Overview

- Lab 1 focuses on the hardware and the functions used to interface the hardware to the C8051
- Lab 1-1 specifically introduces the use of digital inputs and outputs
  - Acquire digital input from external source
  - Use this to determine a digital output
- It is important to develop reusable code
  - Use of functions in your programming
- GOAL: Control 2 LEDs and a buzzer with two switches

Lab 1-2 Overview

- Lab 1-2 presents the use of interrupts and the system timer
- Added components and code
  - More switches
  - Random number generator
- GOAL: Turn on a bi-color LED to red or green after a random delay. And then to measure a reaction time.
  - Use timer overflows to control the delay length of the LEDS and to determine the reaction time.

Lab 2 Overview

- Lab 2 focuses on the development of an interactive game using the hardware and the primitive functions developed in Lab 1.
- The completion of Lab 2 should give you enough experience to get started on the *Smart Car* and *Gondola* systems.
- Now, let's look further into the hardware aspects of the labs.

Preview Lab 1-1

- Refer to the Sample code for Lab 1-1, available on LMS
  - This code will control one LED with one switch
  - You need to modify this code to control 2 LEDs and a buzzer with two switches (as described previously)
  - Things to note as a review of C programming
    - Include header files – we need to include c8051_SDCC.h (website)
    - Function prototypes – note what is returned/passed
    - Variable declarations – must declare at beginning of function
    - Use of indentation & brackets – proper use is very helpful
    - Function calls – need () for functions, not variables

Pseudo-Code

- Homework 2 (see calendar due date) consists of 2 parts:
  - Study provided Lab 1-1 Pseudo-Code and carefully note the structure and formatting
  - Make syntax error corrections on a separate C file
- Pseudo-Code is an outline for the actual code.
  - Read Lab 1, part 1 on LMS and compare the requirements to the Pseudo-Code to complete the lab. It is expected that this structure be followed all semester.
- Readability, Structure
  - Indentation
  - Sequence

Pseudo Code, General Form

```
compiler directives
declare global variables
function prototypes
main function
   declare local variables
   initialization functions
   while (TRUE)
         get sensor information
         act on sensor information
   end while
end main function
```


Pseudo Code, Subroutines

```
function sensor_0
   read and return appropriate value

function sensor_1
   read and return appropriate value

function physical_action
   change system state
```

- Suggestion: Make choices that will keep your code simple and elegant. There is no single best answer. Refer to the provided pseudo code handout.

Logic Worksheet Exercise!!

- Use this worksheet to test your knowledge of how bitwise and logical operators work.
  - Work on this now
  - Save the worksheet and put it in your lab notebook.
  - What no lab notebook yet? Get one. One per group.

Rest of Today

- Finish Worksheet 2
- Have your name checked off on HW1 list & Optionally submit HW1 .c file on LMS
  - Each individual should be able to upload the file on LMS, a hardcopy is only for your own use
- Before you leave
  - Turn off power to the C8051 on car
  - Save any changes to code
  - Take all your possessions with you

Before Next Class

- Finish reading all assigned chapters listed in Homework 1
- Print out the Pseudocode for Lab 1-1 that has been provided to you on LMS
  - Look it over and make sure you understand it
  - This will be added to your Lab Notebook
  - Use this template for the pseudocode assignments required for the rest of the labs this semester
- LMS: Lecture, Labs, & References>Laboratories & Worksheets>Lab 1>Pseudocode Specifications & Examples