Week 5

- Function Pointers
- Lab 3
- Defining you own types
- Math and Floating point
- C++ and Embedded Systems
- Summary
- Wrap up
- Beyond C
- Tools of the Trade
- Your questions

Function pointers

- Pointers can point to functions as well as data. Function pointers are commonly used in embedded systems.

```
int  test_func(int a)
{
  printf("a = %x\n",a);

  return(a);
}

void testFptr(void)
{
  int (* funcp)(int);       /* this is a variable declaration,
                               really! */

  funcp = test_func;
  (*funcp)(0x55aa55); /* execute function indirectly */
}
```

Typedefs – Defining your own types

- Typedefs are used to create your own types. Can be useful in matching hardware up with hardware register sizes or structures.

```
typedef struct {
  int a;
  int b;
  char c;
} MYSTRUCTTYPE;

typedef unsigned long int u64;/* arch dependent*/
typedef unsigned int u32; /* arch dependent */
typedef unsigned short u16; /* guaranteed */
typedef char s8;

#define u64 unsigned long int /* arch dependent*/
#define u32 unsigned int /* arch dependent */
#define u16 unsigned short /* guaranteed */
#define s8 char
```

Floating point and embedded systems

- Using true floating point only as necessary. It is very CPU intensive and on some systems that have no floating point assistance in hardware, it greatly increases your code size and may require additional work such as saving extra registers.
- Some floating point emulation is done in software when the hardware does not support it.
- When possible and within reason work around it.
- Used scaled integers when possible.

Floating point and embedded systems (cont)

- How do a round up without floating point

```
c = a / b; /* rounded down by default */
c = (a + (b-1)) / b; /* is a / b rounded up */
c = (a + ((b-1) / 2)) / b; /* round if greater
                                than  half */
```

Order does matter for math

- Use parenthesizes to control order.
- There is a precedence order but, control what you want to happen.
- When you have space, multiply first then divide. This reduces rounding error.
- Be careful of rolling a signed value from positive to negative by adding a positive value.

Lab3

- Create the old standby Hello Word c program.
- Make the main program call two separate functions stored in an array. Each function prints “Hello “ and “World\n”.
- Compile it without errors and run it.
- Then do they same “Hello World” with a linked list of function pointers.

C++ and Embedded Systems

- When possible use your compiler in C++ mode. It is pickier but will catch problems before you have them.
- However use C++ constructs with care. C++ tends to do things that you may not expect.

Misc. Gotchas

- In embedded systems memory may not be initialized. Assume it is not. And initialize it within functions. Global level initializations such as those declared outside of functions may not occur.
- Just like pointers not all ***int***s are the same size. Use the ***sizeof*** function to determine their size.
- It is you job to make the hardware work.

Good Luck

Summary

- You now can manipulate individual bits within a value. Thereby meaning you can program a register.
- You understand pointers. Therefore you can address a hardware device or manipulate memory.
- Lastly your lab have allowed you to build a few examples which should last you years.

Code Used to Test The Samples


```
int main(int argc, char* argv[])
{
      printf("Hello C for Embedded Systems Class!\n");

      printf("----- AND ---\n");
      printf("testBAND(0,1) - %d\n",testBAND(0,1));
      printf("testLAND(0,1) - %d\n",testLAND(0,1));
      printf("testBAND(2,1) - %d\n",testBAND(2,1));
      printf("testLAND(2,1) - %d\n",testLAND(2,1));
      printf("testBAND(5,4) - %d\n",testBAND(5,4));
      printf("testLAND(5,4) - %d\n",testLAND(5,4));


      printf("----- OR ---\n");
      printf("testBOR(0,1) - %d\n",testBOR(0,1));
      printf("testLOR(0,1) - %d\n",testLOR(0,1));
      printf("testBOR(2,1) - %d\n",testBOR(2,1));
      printf("testLOR(2,1) - %d\n",testLOR(2,1));
      printf("testBOR(5,4) - %d\n",testBOR(5,4));
      printf("testLOR(5,4) - %d\n",testLOR(5,4));

      printf("----- NOT ---\n");
      printf("testBNOT(0) - %d\n",testBNOT(0));
      printf("testLNOT(0) - %d\n",testLNOT(0));
      printf("testBNOT(2) - %d\n",testBNOT(2));
      printf("testLNOT(2) - %d\n",testLNOT(2));
      printf("testBNOT(5) - %d\n",testBNOT(5));
      printf("testLNOT(5) - %d\n",testLNOT(5));

      printf("----- SHIFTING ---\n");
      printf("testLeft(1,0) - %d\n",testLeft(1,0));
      printf("testRight(1,0) - %d\n",testRight(1,0));
      printf("testLeft(2,1) - %d\n",testLeft(2,1));
      printf("testRight(2,1) - %d\n",testRight(2,1));
      printf("testLeft(5,3) - %d\n",testLeft(5,3));
      printf("testRight(5,3) - %d\n",testRight(5,3));


      printf("----- XOR ---\n");
      printf("testXOR(0,0) - %d\n",testXOR(0,0));
      printf("testXOR(1,0) - %d\n",testXOR(1,0));
      printf("testXOR(1,1) - %d\n",testXOR(1,1));
      printf("testXOR(2,1) - %d\n",testXOR(2,1));
      printf("testXOR(5,3) - %d\n",testXOR(5,3));

      printf("----- Function Pointers ---\n");
      testFptr();

      return 0;
}
```

Sample Compilation Results


```
Deleting intermediate files and output files for project 'c4embedded - Win32 Debug'.
--------------------Configuration: c4embedded - Win32 Debug--------------------
Compiling...
StdAfx.cpp
Compiling...
c4embedded.cpp
Linking...

c4embedded.exe - 0 error(s), 0 warning(s)
```

Sample Output


```
Hello C for Embedded Systems Class
----- AND ---
testBAND(0,1) - 0
testLAND(0,1) - 0
testBAND(2,1) - 0
testLAND(2,1) - 1
testBAND(5,4) - 4
testLAND(5,4) - 1
----- OR ---
testBOR(0,1) - 1
testLOR(0,1) - 1
testBOR(2,1) - 3
testLOR(2,1) - 1
testBOR(5,4) - 5
testLOR(5,4) - 1
----- NOT ---
testBNOT(0) - 255
testLNOT(0) - 1
testBNOT(2) - 253
testLNOT(2) - 0
testBNOT(5) - 250
testLNOT(5) - 0
----- SHIFTING ---
testLeft(1,0) - 1
testRight(1,0) - 1
testLeft(2,1) - 4
testRight(2,1) - 1
testLeft(5,3) - 40
testRight(5,3) - 0
----- XOR ---
testXOR(0,0) - 0
testXOR(1,0) - 1
testXOR(1,1) - 0
testXOR(2,1) - 3
testXOR(5,3) - 6
----- Function Pointers ---
a = 55aa55
```

Past this Point is the end of the Class Requirements

- Beyond C
- Good Practices
- Tools of the Trade

Beyond C

- Tasks
- ISRs – Interrupt Service Routines
- Semaphores
- Messaging

Task


```
int task()
{
    int errorcode;
    while(1)
    {
        errorcode = doSomething();
        if (errorcode)
        {
                break;
        }
    }
    return errorcode;
}
```

ISR – Interrupt Service Routine


```
void myISR(void)
{
  doSomeThingQuick();
}
```

Note: If directly called from interrupt vector, probably requires the attribute “interrupt”.

Semaphore


```
void funct1(void)
{
  getSem();
  doSomethingCriticalHere()
  putSem();
}
void funct2(void)
{
  getSem();
  doSomethingElseCriticalHere()
  putSem();
}
```

Note: Do not perform any operation within an ISR which will cause execution to block. If you have something critical that you do not want interrupted, then block that interrupt temporarily.

Messaging

- Comes in many forms and flavors on different OSes but is allows sending a message from an ISR to a task or a task to task. In the simplest form this may be a linked list of messages.
- Great for synchronization and to eliminate polling.

Good Practices

- Write code as if your code will be delivered to your friends who will give you grief if they can't maintain it.
  1. Comment
  2. Use white space
  3. Write readable code
  4. Pick a standard and stick with it, be consistent.
- Version Control is your friend

Tools of the Trade

- RTOSes – Real Time Operate Systems
- Debuggers
- Emulators
- EPROM programmers
- Simulators
- Co-Simulation

Ask any Questions

- Did you forget to ask any questions?
- If you realize now or six months from now you have my email address email me.

Once I am you teacher I am always your teacher

Live Long and Prosper Spock