Programming for Embedded Systems
Lecture 1

Bernhard Firner

January 22, 2013

Introduction

- Office: ECE 218
- Email: bfirner@winlab.rutgers.edu

Resources

- All slides and code examples will be posted on sakai
- Use sakai forum for programming questions

Objectives

- Learn embedded programming in C
- Learn how to read and understand device datasheets
- Gain experience with a standard MCU (TI's MSP430)
- Complete a few simple projects and one larger final project

Required Material

- No book, materials will be on sakai
- Every students needs a launchpad ($10)
  - Model: MSP-EXP430G2
- Code Composer Studio
  - CD comes with launchpad or you can download it
  - Should also be installed in ECE lab computers
- Download g2553 manual and family user guide from TI

Tools and Technology

- Breadboards, resistors, LEDs, etc
- Soldering iron (can't breadboard everything)
- Oscilloscopes (for debugging)

Course Outline

- Reintroduction to C
- Introduction to the MSP430
- Output and basic timing
- Advanced timer controls
- Digital to analog conversion
- Input (Digital and Analog)
- Low Power Mode
- Wire communiction

Evaluation

- Labs (30%)
- Four small projects (20%)
- Four quizzes, one for each project (20%)
- One final project (30%)

Today's Topics

- Review of C
- Introduction to Embedded Programming

Some Examples of Embedded Systems

- Avionics systems
- Automotive systems
- Smartphones, watches, homes
- Calculators

Embedded Systems

- Single-purpose systems
- Resource constrained
  - Little memory
  - Usually no OS

No OS Means...

- No memory management
- No threads
- Programmer directly interacts with the hardware

Warning

- Embedded programming is fun, but be warned
- You will need to learn many little details
- Looking things up will be harder than you are used to
- Debugging mysterious errors can be a trial

C and C++

- C and C++ are well-suited to embedded programming
  - Fast and efficient binary operators
  - Directory control of memory through pointers
  - Compiler support for low-level features
- Most C++ features not used due to memory constraints

C++ Review: the Main Function

- All programs start with the *main* function
- This is the only function to run
- Returning from main means that everything stops

Binary Operators

- AND (&), OR (|), and XOR (^)
  - These operators have assignment forms (&=, |=, ^=)
- Binary negation: ~
- Binary operators work bit by bit

Binary Operator Review

- Let A = 01010101 = 0x55
- Leb B = 00001111 = 0x0F

$$A \;\&\; B = 00000101$$

$$A \mid B = 01011111$$

$$A \;\hat{}\; B = 01011010$$

$$\tilde{}A = 10101010$$

Speaking of Memory

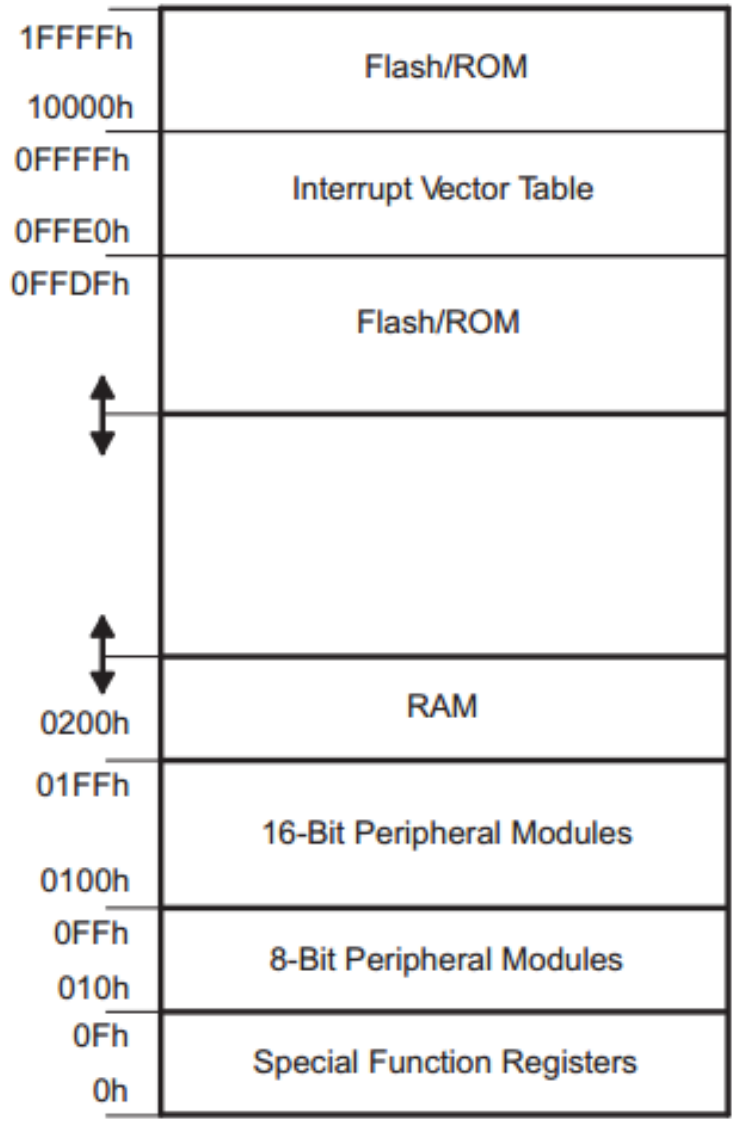
- All those bits need to go somewhere
- Many embedded systems have a simple memory space
  - Usually start at offset 0 and go up to the memory's size
  - No virtual memory and memory space protection
- In a sense, all variables are "global"

Controlling Memory Use

- You control memory use (through IDE or config files)
- Specify heap space (for new operator), set variables locations
- Can easily get confusing, leads to surprising errors

Memory on the MSP430

- We will be using the MSP430G2553
- 16kB memory (addresses 0xFFFF to 0xC000)
  - Your program (variables and functions) must fit here
- 512B RAM (addresses 0x03FF to 0x0200)



Function calls

- Every function call consumes memory on the *stack*
  - Arguments consume memory
  - Function variables consume memory
- What happens when you run out of RAM?

Embedded Systems and Recursive Calls

- Which is better in an embedded system, quicksort or insertion sort?

Embedded Systems and Recursive Calls

- Which is better in an embedded system, quicksort or insertion sort?
- Quicksort's recursive call may consume too much RAM and halt your program
- Safer to use a sorting algorithm with known memory requirements

Interrupts

- Interrupts are a special kind of function called by the hardware
- They halt the execution of whatever code is currently running
- When the interrupt is complete, control returns to the interrupted function
- Disable interrupts if you don't want a segment of code to be interrupted!

Vector Interrupts

- The MSP430 uses *vector interrupts*
- A block of memory called the *interrupt vector table* stores the locations of *interrupt handlers*
- When you write code to handle an interrupt, you (really your compiler) adds the location of that function into that interrupt vector table

Interrupt Handling

- When an interrupt occurs the hardware checks for a memory address in the interrupt vector table
- The hardware loads that memory address into the program counter (PC)
    - The PC is the address of the next instruction to run
- When the interrupt handler ends, execution returns to the previous function

Scope, Variables, and Registers

- Local variables are in the scope of a single function
- Global variables can be accessed from any function
- Registers are global variables that represent actual hardware registers

More on Registers

- You interact with hardware by changing register values
  - For example, writing a 1 to a register may turn on an LED
- Hardware may also write into registers to indicate changes in status

Documentation

- There is a link to TI's literature on the MSP430 in sakai
- It is not easy to read, but download the pdf files and take a look
- We will be referring to these documents later

Next Week

- Form groups!
- Make sure one group member has code composer!
- Next time we will start working with the MSP430