CS 466/566 Spring 2018

Miller Lowe

Mechanics of Course

- Lecture Tuesdays — VECS 122 5:45pm-7:40pm
- Lab Thursdays — VECS 221 5:45pm-8:15pm
- Course Syllabus

Books Used

- An Embedded Software Primer by David E. Simon
- Extra material as needed from other sources
  - Embedded System Journal
  - Modern Operating Systems by Tanenbaum

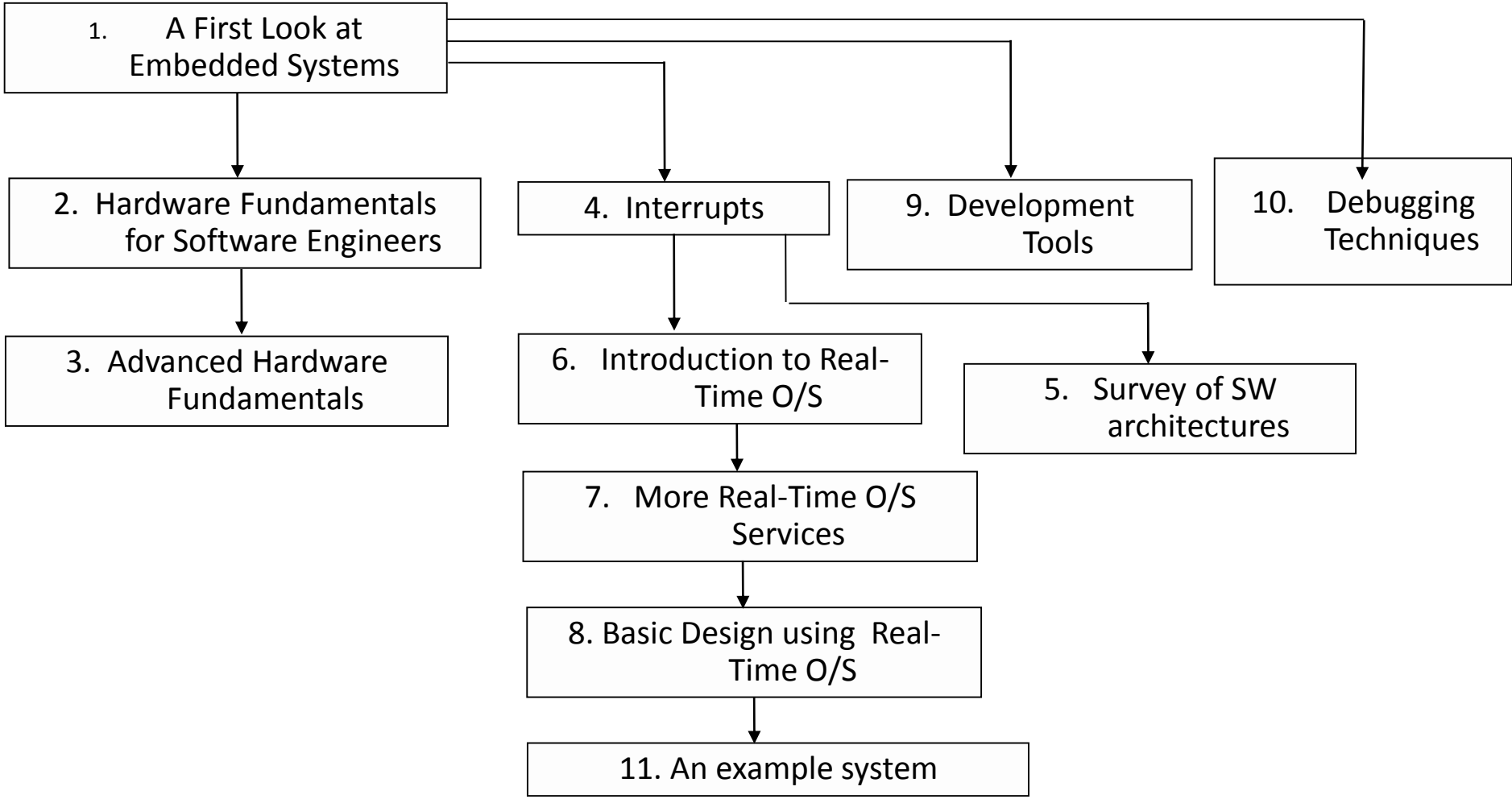
About the textbook

- Embedded Systems (Really Old)
  - Terminology is not very consistent
  - For every word, several subtly different meanings
  - Code sizes very from under 500 bytes to many megabytes
  - The textbook focuses on rules followed by 85% of engineers even though the other 15% of engineers need to break the rules to get their systems to work!

About the textbook (cont)

- There are some dependencies in the text because it is intended to be read straight through.
- One can skip around if desired.

Dependencies



C++/C!!

- We will tend to ignore C++ and use C to focus on embedded systems because C++ is more complex and we have enough to learn.
- C!! is pushed by Simon as being the same as C but whatever is after the !! the computer will do without the details.

Variable naming conventions

- We will jump all over the place as far as variable naming..
- I do not care what you use as long as you:
  - Stay fairly consistent in your use.
  - Emulate that within an existing module.
  - Make the code readable, readable, readable

Brace Conventions

- Expect an adoption of a consistent style for this class.

- Typical choices are: *instructor preferred
  - Allman style (extended braces)

```
if(a  ==  b)
{
    cout << a;
    b++;
 }
```

  - One True Brace Style (K & R) (linesaver) * OK

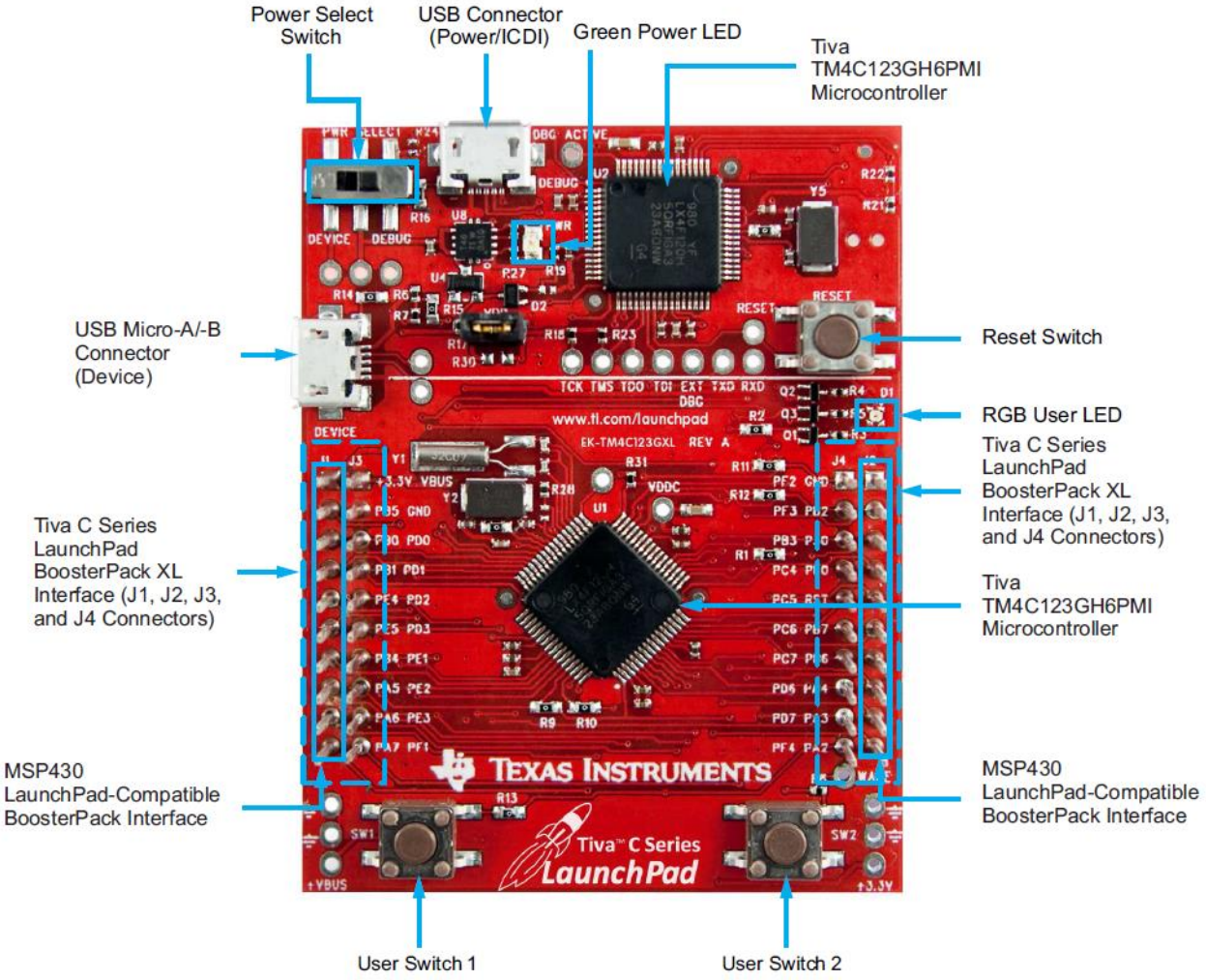
```
if(a  ==  b){
    cout << a;
    b++;
 }
```

  - WhiteSmith's Style (indented braces) *deprecated

```
if(a  ==  b)
  {
      cout << a;
       b++;
    }
```

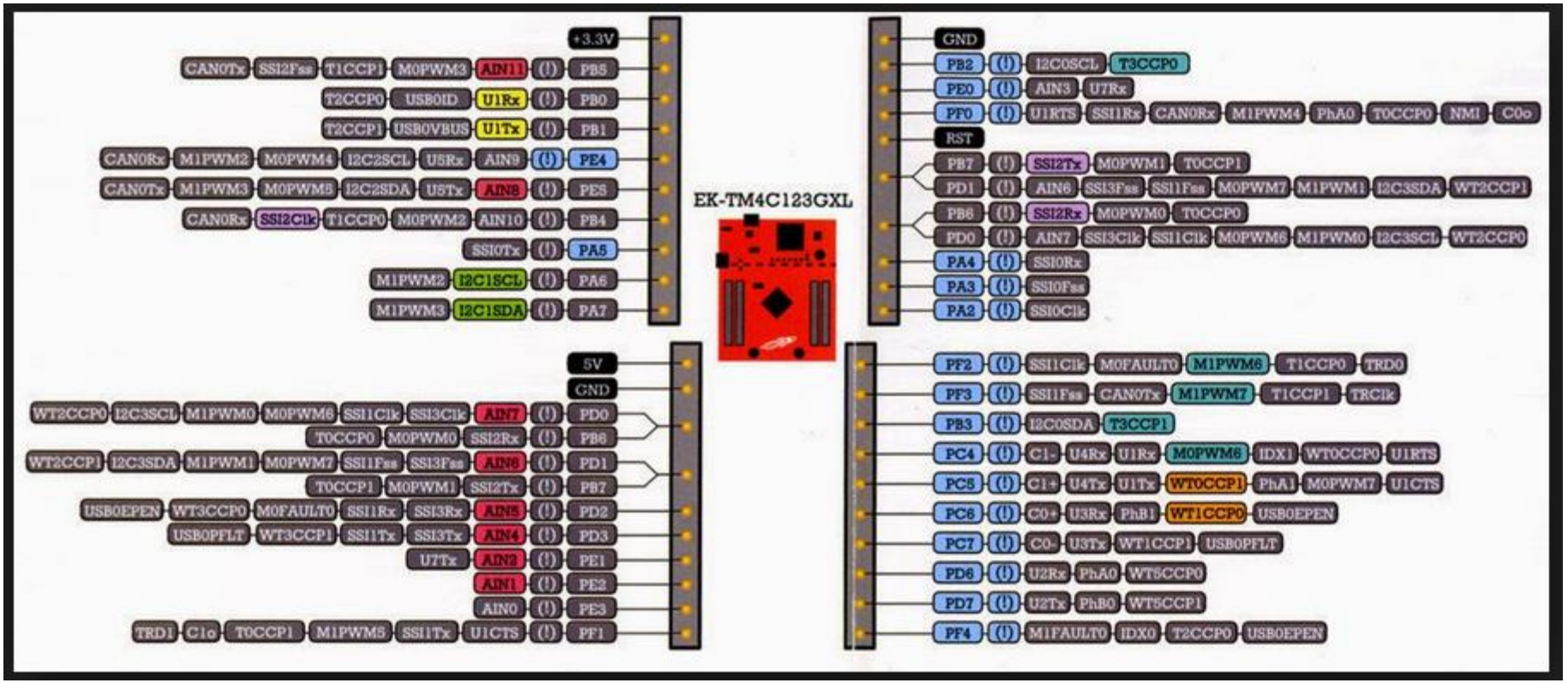
Lab Embedded CPU Board

Figure 1-1. Tiva C Series TM4C123G LaunchPad Evaluation Board



Tiva Pin-Muxing

(This will cause you heartburn, but it's real..)



Lab 1 Introduction

- Bare-Metal Blinky (https://en.wikipedia.org/wiki/Bare_machine)
- Main objective is to get you targeting code, Some concepts are needed to answer the lab questions.
- Do prep-work before you get to lab
- I encourage you to get a development system setup on your own PC.
- We will use Linux in the Lab as our home development. I use Linux and Windows all the time

Lab 1 Some Information

- What's a register?
- How to locate a register?
- We have an address, How to use it
- Example
  - Want to output to a GPIO

Table 10-6. GPIO Register Map

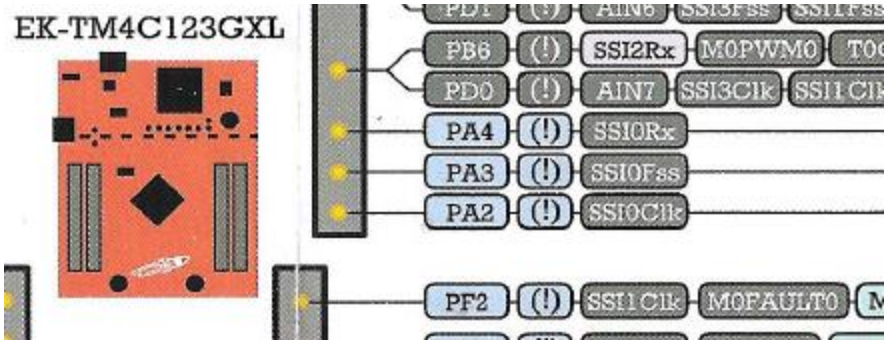
Offset	Name	Type	Reset	Description	See page
0x000	GPIODATA	RW	0x0000.0000	GPIO Data	662
0x400	GPIODIR	RW	0x0000.0000	GPIO Direction	663
0x404	GPIOIS	RW	0x0000.0000	GPIO Interrupt Sense	664
0x408	GPIOIBE	RW	0x0000.0000	GPIO Interrupt Both Edges	665

- GPIO Port A (AHB): 0x4005.8000
- GPIO Port B (APB): 0x4000.5000
- GPIO Port B (AHB): 0x4005.9000
- GPIO Port C (APB): 0x4000.6000
- GPIO Port C (AHB): 0x4005.A000
- GPIO Port D (APB): 0x4000.7000
- GPIO Port D (AHB): 0x4005.B000

Address I want is GPIO Port A GPIO Data Register 0x40058000 + offset(0x000)

Lab 1 Some Information

- Address I want is GPIO Port A GPIO Data Register 0x40058000 + offset(0x000)
- The Tiva PinMux shows a couple (but not all GPIOA ports)



- So PA2 is GPIO Port A2.
- To set the PA2 bit to a logical 1 you need to set the 0x00000004 bit in the GPIO Port A Data Register, The los level C code looks like

```
*((volatile uint32_t *)0x4005800)) = *((volatile uint32_t *)0x4005800)) |
0x00000004;

// …kinda ugly, non-portable…
```

- Would be nice to say
  - `PORTA_DATA |= (1<<2);`
  - `setBit(PA2, 2)`
  - `bitControl(PA2, BIT2, TRUE);`
  - `//Damn near anything is better than the first line`

LAB 1 Process

- I recommend that you get used to looking at the 'Lab Preparation' section of the lab and acting on it some before you show up in lab..
- I am a Cruel and Unfair instructor. I have added an issue with this lab that you will need to discover and overcome.
- Get a your Linux login working
- Install the git repo and build the lab-1 code provided
- Perform the modifications outlined in the lab handout. (available on Blackboard and in the git repo)
- I have a strict lab format, *(remember the 'Cruel and Unfair' comment above)* If you do not address and satisfy all the sections your grade will represent the omissions.

Recurring Examples of Embedded Systems used in the book.

- Telegraph
  - Allows a printer with only a high speed serial port to connect to a network
  - Looks like a little plastic box 2 to 3 inches by ½ inch thick
  - Has a pigtail connector for the serial connection and a network connector

Examples of Embedded Systems

- Telegraph
  - Allows a printer with only a high speed serial port to connect to a network
  - Looks like a little plastic box 2 to 3 inches by ½ inch thick
  - Has a pigtail connector for the serial connection and a network connector

Telegraph Requirements:

- On the network, data packets sometimes arrive out of order, data gets lost, and sometimes data arrives twice. Telegraph must make order out of chaos
- Multiple computers on the network may want to print at once. Telegraph must allow one computer to print and somehow hold off the others.
- Network printers must provide status info to any computer on the network that requires it, even if busy printing a job for some other computer.
- Telegraph has to work with different types of printers.
- Telegraph must respond rapidly to certain events, like various kinds of network frames.
- Telegraph must keep track of time. If a computer which is sending a print job crashes, telegraph must be able to give up on that print job and print from another computer on the network.

Telegraph development challenges

- To satisfy the previous requirements, telegraph has an embedded microprocessor. It must have logically correct performance.
- Throughput – telegraph must not be a bottleneck for print data flowing through it.
- Response – telegraph must respond to certain network frames within 200 usec, even if it is doing something else. We will talk about throughput and response and avoid speed…
- Testability – telegraph must deal with everything without human intervention. It is hard to test "everything" and how do you make something happen to see if the code handles it.

Telegraph Development (cont)

- Debugability – telegraph has no screen, no keyboard, no mouse, no speaker, no lights. When a bug occurs, it typically stops working…
- Reliability – telegraph is not allowed to crash, which is typical of embedded systems. The sw must function without human intervention.
- Memory Space – telegraph has limited memory, typically 32 kbytes of memory for its program and another 32 kbytes for its data.
- Program Installation – we need special tools to install sw on these kinds of systems.

Cordless Bar Code Scanner

- No problem with throughput…there just isn't that much data
- Power consumption – since the scanner is cordless, its battery is its only source of power. It is intended to be handheld, which limits its weight. How long must the battery last? An eight hour shift is probably the least we can tolerate.
- What are some ways that the battery life could be mitigated/extended?

Laser Printer

- Most have fairly substantial microprocessors embedded in them.
- The microprocessor is responsible for getting data from any of various communication ports, sensing when users press front panel buttons, presenting messages to the user on the front panel, sensing paper jams, noticing out of paper, etc.
- It also must deal with the laser engine, which is that part of the printer responsible for putting marks on the paper.

Laser Printer (continued)

- Users expect a nearly instantaneous response when pressing a front panel button, no matter what the microprocessor is doing.

Underground Tank Monitor

- This system watches the levels of gasoline in underground tanks at a gas station.
- Its principal purpose is to detect leaks before the gas station turns into a toxic waste dump by mistake and to set off a loud alarm if it discovers a leak.
- The system has a panel of 16 buttons, a 20 character LCD as a display, and a thermal printer.
- The user interacts with various buttons to tell the system to display or print various information such as the gasoline levels or time of day or system status.

Underground Tank Monitor

- The system has two float levels in each tank, namely one that indicates the level of gasoline and the other that indicates the level of water in the tank. (Water always accumulates in such tanks).
- It also reads the temperature at various levels in the tank because gasoline expands and contracts considerable with variations in temperature.
- Cost – A gas station owner buys one of the systems because some government agency tells him that he has to. Therefore, we might use a cheap 8 bit microprocessor that can barely add two numbers let alone deal with the expansion coefficient of gasoline in a reasonable way.

Nuclear Reactor Monitor

- A very simple example in which we can learn from a system controlling a hypothetical nuclear reactor.
- It monitors two temperatures, which are always supposed to be equal. It they ever differ, it is supposed to set off the alarm.

Death Ray

- Requirements:
  - Operation:
  - Physical:
  - Power:
  - Safety:

Typical Hardware

Processor	Bus Width	Ext Memory	Int Memory	Periphials	Speed(MIPS)
Zilog Z8 fam	8	64K	0	2 Timers	1
Intel 8051 fam	8	64K	64K	3 Timers	1
Zilog Z80 fam	8	64K	1MB	Various	2
Intel 80188	8	1MB	0	3 Timers 2 DMA	2
Intel 80386	16	64MB	0	3 Timers 2 DMA Ch	5
Motorola 68000	32	4GB	0	Varies	10
Motorola PowerPC	32	64MB	0	Many	75
ARM M4F TM4C123G	32	4GB	256K/32K F/R	Lots	

Embedded Systems

- Embedded systems are known for what they do NOT have – note the following
  - Keyboard
  - Screen
  - Disk drive
  - Compact discs, speakers, microphones,diskettes, modems, etc.

Embedded Systems

- Embedded systems are known for what they do NOT have – note the following
  - Keyboard
  - Screen
  - Disk drive
  - Compact discs, speakers, microphones,diskettes, modems, etc.

Chapter 1 summary

- An embedded system is any computer system hidden inside a product other than a computer.
  - Throughput
  - Response time
  - Testability
  - Debugability
  - Reliability
  - Memory space
  - Program installation
  - Power consumption
  - Processor hogs
  - Cost