Introduction to Embedded System Design

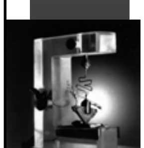
Babak Kia
Adjunct Professor
Boston University
College of Engineering
Email: bkia -*at*- bu.edu

Microcontrollers

- **A Microcontroller is essentially a small and self-sufficient computer on a chip, used to control devices**
- **It has all the memory and I/O it needs on board**
- **Is not expandable – no external bus interface**
- **Characteristics of a Microcontroller**
  - **Low cost, on the order of $1**
  - **Low speed, on the order of 10 KHz – 20 MHz**
  - **Low Power, extremely low power in sleep mode**
  - **Small architecture, usually an 8-bit architecture**
  - **Small memory size, but usually enough for the type of application it is intended for. Onboard Flash.**
  - **Limited I/O, but again, enough for the type of application intended for**

Microprocessors

- **A Microprocessor is fundamentally a collection of on/off switches laid out over silicon in order to perform computations**
- **Characteristics of a Microprocessor**
  - **High cost, anywhere between $20 - $200 or more!**
  - **High speed, on the order of 100 MHz – 4 GHz**
  - **High Power consumption, lots of heat**
  - **Large architecture, 32-bit, and recently 64-bit architecture**
  - **Large memory size, onboard flash and cache, with an external bus interface for greater memory usage**
  - **Lots of I/O and peripherals, though Microprocessors tend to be short on General purpose I/O**



Harvard Architecture

- Harvard Architecture refers to a memory structure where the processor is connected to two different memory banks via two sets of buses
- This is to provide the processor with two distinct data paths, one for instruction and one for data
- Through this scheme, the CPU can read both an instruction and data from the respective memory banks at the same time
- This inherent independence increases the throughput of the machine by enabling it to always prefetch the next instruction
- The cost of such a system is complexity in hardware
- Commonly used in DSPs

Von-Neumann Machine

- A Von-Neumann Machine, in contrast to the Harvard Architecture provides one data path (bus) for both instruction and data
- As a result, the CPU can either be fetching an instruction from memory, or read/writing data to it
- Other than less complexity of hardware, it allows for using a single, sequential memory.
- Today's processing speeds vastly outpace memory access times, and we employ a very fast but small amount of memory (cache) local to the processor
- Modern processors employ a Harvard Architecture to read from two instruction and data caches, when at the same time using a Von-Neumann Architecture to access external memory

Little vs. Big Endian

- Although numbers are always *displayed* in the same way, they are not *stored* in the same way in memory
- Big-Endian machines store the most significant byte of data in the lowest memory address
- Little-Endian machines on the other hand, store the least significant byte of data in the lowest memory address

A Big-Endian machine stores 0x12345678 as:		A Little-Endian machine stores 0x12345678 as:	
ADD+0:	0x12	ADD+0:	0x78
ADD+1:	0x34	ADD+1:	0x56
ADD+2:	0x56	ADD+2:	0x34
ADD+3:	0x78	ADD+3:	0x12

Little vs. Big Endian

- The Intel family of Microprocessors and processors from Digital Equipment Corporation use Little-Endian mode
- Whereas Architectures from Sun, IBM, and Motorola are Big-Endian
- Architectures such as PowerPC, MIPS, and Intel's IA-64 are Bi-Endian, supporting either mode
- Unfortunately both methods are in prevalent use today, and neither method is superior to the other
- Interfacing between Big and Little Endian machines can be a headache, but this is generally only a concern for TCP/IP stack and other interface developers

Program Counter (PC)

- The Program Counter is a 16 or 32 bit register which contains the address of the *next* instruction to be executed
- The PC automatically increments to the next sequential memory location every time an instruction is fetched
- Branch, jump, and interrupt operations load the Program Counter with an address other than the next sequential location
- During reset, the PC is loaded from a pre-defined memory location to signify the starting address of the code

Reset Vector

- The significance of the reset vector is that it points the processor to the memory address which contains the firmware's first instruction
- Without the Reset Vector, the processor would not know where to begin execution
- Upon reset, the processor loads the Program Counter (PC) with the reset vector value from a pre-defined memory location
- On CPU08 architecture, this is at location $FFFE:$FFFF
- A common mistake which occurs during the debug phase – when reset vector is not necessary – the developer takes it for granted and doesn't program into the final image. As a result, the processor doesn't start up on the final product.

Stack Pointer (SP)

- The Stack Pointer (SP), much like the reset vector, is required at boot time for many processors
- Some processors, in particular the 8-bit microcontrollers automatically provide the stack pointer by resetting it to a predefined value
- On a higher end processor, the stack pointer is usually read from a non-volatile memory location, much like the reset vector
- For example on a ColdFire microprocessor, the first sixteen bytes of memory location must be programmed as follows:

  0x00000000: Reset Vector
  0x00000008: Stack Pointer

COP Watchdog Timer

- The Computer Operating Properly (COP) module is a component of modern processors which provides a mechanism to help software recover from runaway code
- The COP, also known as the Watchdog Timer, is a free-running counter that generates a reset if it runs up to a pre-defined value and overflows
- In order to prevent a watchdog reset, the user code must clear the COP counter periodically.
- COP can be disabled through register settings, and even though this is not good practice for final firmware release, it is a prudent strategy through the course of debug

The Infinite Loop

- Embedded Systems, unlike a PC, never “exit” an application
- They idle through an Infinite Loop waiting for an event to happen in the form of an interrupt, or a pre-scheduled task
- In order to save power, some processors enter special sleep or wait modes instead of idling through an Infinite Loop, but they will come out of this mode upon either a timer or an External Interrupt

Interrupts

- Interrupts are mostly hardware mechanisms which tell the program an event has occurred
- They happen at any time, and are therefore asynchronous to program flow
- They require special handling by the processor, and are ultimately handled by a corresponding Interrupt Service Routine (ISR)
- Need to be handled quickly. Take too much time servicing an interrupt, and you may miss another interrupt.

Designing an Embedded System

- Proposal
- Definition
- Technology Selection
- Budgeting (Time, Human, Financial)
- Material and Development tool purchase
- Schematic Capture & PCB board design
- Firmware Development & Debug
- Hardware Manufacturing
- Testing: In-Situ, Environmental
- Certification: CE
- Firmware Release
- Documentation
- Ongoing Support

System Design Cycle

- Writing code conjures up visions of sleepless nights and stacked up boxes of pizza
- And if not done correctly, that is exactly what the outcome will be!
- The purpose of the design cycle is to remind and guide the developer to step within a framework proven to keep you on track and on budget

System Design Cycle

- There are numerous design cycle methodologies, of which the following are most popular
  - The Spaghetti Model
  - The Waterfall Model
  - Top-down versus Bottom-up
  - Spiral Model
  - GANTT charts

The Spaghetti Model

Not in this course, thank you.

The Waterfall Model

- Waterfall is a software development model in which development is seen flowing steadily through the phases of
  - Requirement Analysis
  - Design
  - Implementation
  - Testing
  - Integration
  - Maintenance
- Advantages are good progress tracking due to clear milestones
- Disadvantages are its inflexibility, by making it difficult to respond to changing customer needs / market conditions

Top-down versus Bottom-up

- **The *Top-Down Model* analyses the overall functionality of a system, without going into details**
  - **Each successive iteration of this process then designs individual pieces of the system in greater detail**
- **The *Bottom-Up Model* in contrast defines the individual pieces of the system in great detail**
  - **These individual components are then interfaced together to form a larger system**

The Spiral Model

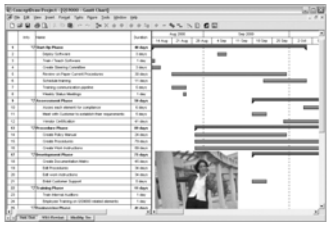
- **Modern software design practices such as the Spiral Model employ both top-down and bottom-up techniques**
- **Widely used in the industry today**
- **For a GUI application, for example, the Spiral Model would contend that**
  - **You first start off with a rough-sketch of user interface (simple buttons & icons)**
  - **Make the underlying application work**
  - **Only then start adding features and in a final stage spruce up the buttons & icons**

The Spiral Model

- **In this course, we won't focus on the aesthetic qualities of the application - only the underlying technology**

GANTT Chart

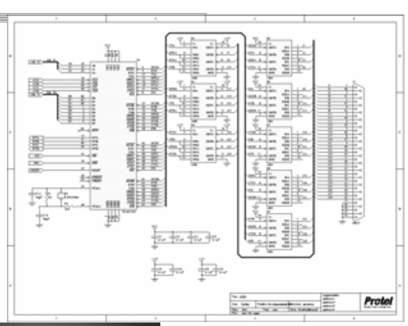
- GANTT Chart is simply a type of bar chart which shows the interrelationships of how projects and schedules progress over time



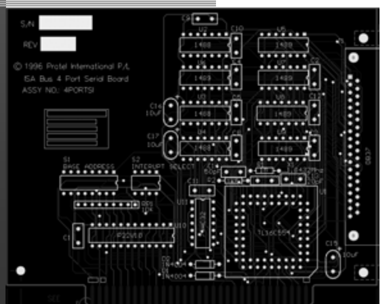
Design Metrics

- Metrics to consider in designing an Embedded System
  - Unit Cost: Can be a combination of cost to manufacture hardware + licensing fees
  - NRE Costs: Non Recurring Engineering costs
  - Size: The physical dimensions of the system
  - Power Consumption: Battery, power supply, wattage, current consumption, etc.
  - Performance: The throughput of the system, its response time, and computation power
  - Safety, fault-tolerance, field-upgradeability, ruggedness, maintenance, ease of use, ease of installation, etc. etc.

Schematic Capture



PCB Layout



PCB Board