CMPE 311: Programming & Embedded Systems

Course Description:

In this course, students learn about hardware and software aspects of embedded systems. Students learn C programming language through use in an embedded platform. The course builds on CMPE 310, introducing advanced topics including communication interfaces, advanced IO devices and other peripherals, multitasking, firmware, real-time operating systems/embedded operating systems and device drivers. The course will provide a hands-on experience in designing and programming an embedded system using a microcontroller based development platform

Prerequisites

Required: CMPE 310 - Systems Design and Programming

***Meeting Times and Locations*:**

Lecture

- M,W 5:30 - 6:45p
- Sherman Hall 151

Discussion

- 4 – 5:15p, M or W depending on section.
- Sherman Hall 006 (*may* change to: Information Technology 375?!?)

CMPE 311: Programming & Embedded Systems

Instructor

William H. Smith III (Bill)
Department of Computer Science and Electrical Engineering
Cell Phone: 484-554-3661 (texting works well!)
Email: whsmith003@hotmail.com
william.smith3@ngc.com

Office Hours: Because I am adjunct faculty, I won't be able to hold regular office hours. The plan is to be here in the classroom at 5p each evening and remain after the discussion class for as long as required to answer questions. You are welcome and encouraged to talk with me and ask questions during those times! I am also usually only a text away…

Teaching Assistant

Sadique Hasan email: mhasan5@umbc.edu Office Hours: TBD

Undergrad Teaching Assistant

Dominick Kroupa email: dkroupa1@umbc.edu Office Hours: TBD

Web

- Piazza may be setup. The teaching assistants will be the primary responders with backup by the instructor as needed.
- UMBC Blackboard will host most of the course materials for this lecture.

CMPE 311: Programming & Embedded Systems

Major Learning Objectives

At the conclusion of the course, students will:

- have developed programming skills in the C language;
- understand the hardware and software requirements for a microcontroller-based embedded system
- be familiar with, and will be able to use, built-in microcontroller peripherals, including coding and building external hardware
- be able to develop code for multi-tasking applications, and will be familiar with processes synchronization, resource sharing, and task scheduling
- be able to install and use a real-time operating system on embedded hardware
- be able to design and build a real-time system performing data capture, communications, and user interface

Text

James K. Peckol, 'Embedded Systems: A Contemporary Design Tool' First Edition, John Wiley & Sons, Inc. (2008), ISBN: 978-0471721802.

Supplementary text – note: K&R is STRONGLY recommended!!!:

Brian W. Kernighan and Dennis M. Ritchie 'The C Programming Language' 2nd edition: ISBN: 013-11036208 (Paperback), ISBN: 013-1103709 (hardback)

Jonathan W. Valvano 'Embedded Micro-computer Systems: Real Time Interfacing'3rd edition ISBN: 978-1111426255

CMPE 311: Programming & Embedded Systems

Grading Scheme

Evaluation of students will be made approximately according to the following scheme, though the instructor reserves the right to make adjustments as is fair and appropriate during the semester:

Grading of students includes the following:

– Midterm Exam: 15%
– Final Exam: 20%
– In-class Quizzes and assignments: 10% (2.5% each…)
– In-class Participation: 5%
– Discussion/Lab Assignments (programming projects): 45% (composed of several assignments with varying amounts of credit based on length and difficulty, to be used to generate a weighted average.)
– Discussion Participation: 5%

Late work policy

A two day delay policy for Discussion Homework will result in declining percentage awarded. 5% will be lost for one day overdue, 25% will be lost for two days overdue. Unless otherwise stated, assignments submitted later than two days after their due date will receive a zero. If it ever is ‘otherwise stated’ due to extenuating circumstances, assignment delays of more than a week will not be granted.

CMPE 311: Programming & Embedded Systems

Incomplete Grades

A grade of incomplete will be given only under exceptional circumstances described by the University policy for granting incompletes. Any such circumstance MUST be brought to the instructor's attention immediately as soon as it is known.

Failure to complete assignments on time is not a sufficient reason for an incomplete. If you feel you are falling behind, seek help ***immediately***. Any delay **MUST** be requested prior to the assignment due date.

Academic Misconduct

The assignments in this class are for individual work. The instructor expect that you may minimally discuss some approaches to the projects, but you may not collaborate on writing code or share or copy it with others. You must never copy code or turn in anything that not representative of your learning and mastery of the material. Cheating or academic misconduct related to a assignment will make you subject to the maximum allowed penalty from the university. A zero on the assignment is only the minimum penalty. If you are stuck late and desperately decide you need to copy something to move on to complete the rest of an assignment, you must cite your source to only receive an grade reduction as appropriate and avoid being subject to academic misconduct penalties. See http://www.umbc.edu/undergrad_ed/ai/students.html for policies and definitions.

All students are expected to be knowledgeable regarding all University policies on academic misconduct. By enrolling in this course, each student assumes the responsibilities of an active participant in UMBC's scholarly community in which everyone's academic work and behavior are held to the highest standards of honesty. Cheating, fabrication, plagiarism, and helping others to commit these acts are all forms of academic dishonesty, and they are wrong. Academic misconduct could result in disciplinary action that may include, but is not limited to, suspension or dismissal. To read the full Student Academic Conduct Policy, consult the UMBC Student Handbook, the Faculty Handbook, or for graduate courses, the Graduate School website.

CMPE 311: Programming & Embedded Systems

Tentative Schedule of Topics:

1) Introduction
2) CH1_and_Review
3) Microcontrollers
4) AVR 8-bit Architecture
5) AVR CPU Registers
6) AVR IO Ports
7) AVR Addressing Modes
8) More AVR Assembler
9) C Basics
10) Functions, Separate Compilation, Macros
11) AVRI IO Examples - I think - Discussion IV
12) Arrays Argument Passing Promotion Demotion
13) C Strings
14) Memory Usage
15) C Pointer Variables
16) Debugging and logging with printf macros
17) Pointers And Arrays
18) Struct and Union
19) Advanced Pointers
20) Final C
21) Memory-Related Perils and Pitfalls.ppt
22) Interrupts
23) Timers, Counters
24) integers
25) RTOS
26) Tasks
27) Converters
28) Communications

Recommended Resource: http://code.geeksforgeeks.org/

Code:

Copy

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//returns negative if b%8 > a%8,
//          positive if a%8 > b %8, else 0
int Mod8Compare (const void * ptrA, const void * ptrB){
  int t1, t2, t3;
  const int *ptrX = (const int * ) ptrA;
  const int *ptrY = (const int * ) ptrB;
  t1 = (*ptrX) % 8;
  t2 = (*ptrY) % 8;
  t3 = t1 -t2;
  printf("ina: %2d  inb: %2d  ",*ptrX,*ptrY);
  printf("  t1: %2d  t2: %2d  t3: %2d\n",t1,t2,t3);
  return ((*ptrX)%8 -(*ptrY) % 8);
}
void main(){
  int a[]={5,6,7,8,9,10};
  int i;
  qsort((void *)a,(size_t)6,sizeof(int),Mod8Compare);
  printf("%d",a[0]);
  for (i = 1; i<6; ++i){
    printf(",%d",a[i]);
  }
}
```

Upload File

- **For the 'C' portion of the course**
- **Multiple languages**
- **Allows for quick check of code constructs.**
- **You can verify on class notes (in real time if needed…)**

Input:

Copy

C | C++ | C++14 | C# | Java | Perl | PHP | Python | Python 3 | Scala

Run | Run+URL(Also Generates URL) | Reset

Output:

Copy

```
ina:  6  inb:  7    t1:  6  t2:  7  t3: -1
ina:  5  inb:  6    t1:  5  t2:  6  t3: -1
ina:  9  inb: 10    t1:  1  t2:  2  t3: -1
ina:  8  inb:  9    t1:  0  t2:  1  t3: -1
ina:  5  inb:  8    t1:  5  t2:  0  t3:  5
ina:  5  inb:  9    t1:  5  t2:  1  t3:  4
ina:  5  inb: 10    t1:  5  t2:  2  t3:  3
8,9,10,5,6,7
```

Additional Reading Material and Resources

- **Assembly**
  - AVR Assembly User Guide: http://www.atmel.com/Images/doc1022.pdf (includes condensed table of instructions)
  - List of commands sorted by function: http://www.avr-asm-tutorial.net/avr_en/beginner/COMMANDS.html
  - Gettng Started:
    - http://www.avr-asm-tutorial.net/avr_en/beginner/index.html many topics and tables
    - http://www.avr-asm-download.de/beginner_en.pdf Beginners introduction Beginners Introduction to the Assembly Language of ATMEL-AVR-Microprocessors by Gerhard Schmidt http://www.avr-asm-tutorial.net
- **C**
  - Arrays and Pointers: http://www.lysator.liu.se/c/c-faq/c-2.html
  - http://cslibrary.stanford.edu/101/EssentialC.pdf
  - http://publications.gbdirect.co.uk/c_book/

Relevant or Interesting Links

- https://www.mainframe.cx/~ckuethe/avr-c-tutorial/
- http://www.smileymicros.com/smileymicros_files/BrayTerminal.zip
- http://www.smileymicros.com/ avr butteryfly kits, books
- CMSC 313
  - Notes: http://www.csee.umbc.edu/courses/undergraduate/CMSC313/fall10/MiscPages/schedule.shtml
  - Resources mostly C: http://www.csee.umbc.edu/courses/undergraduate/CMSC313/spring11/Resources/resources.shtml
- CMSC 201, a C version, lecture notes: http://www.csee.umbc.edu/courses/undergraduate/201/spring09/lectures/
- A similar course at CMU: http://www.ece.cmu.edu/~ece348 Embedded Systems Engineering
- http://www.ece.cmu.edu/~koopman/pubs/koopman05_embedded_education.pdf

Systems Overview – CMPE310 Review

General Purpose Computing Systems

- Personal Computers, laptops, workstations, mainframes and servers

Systems for Dedicated Functions

- Usually embedded within larger electronic devices (referred to as embedded systems)
- Difficult to define exactly as they encompass a wide variety of electronic systems
- Definitions from several authors:
  - Any computing systems other than a general purpose computer
  - A system consisting of hardware, main application software and an optional real time operating systems (RTOS)
  - Loosely defined, it is any device that includes a programmable computer but is not itself intended to be a general-purpose computer
  - Electronic systems that contain a microprocessor or microcontroller, but we do not think of them as computers - the computer is hidden or embedded in the system
  - It is a system whose principal function is not computational, but which is controlled by a computer embedded within it, ...
  - And many more....

Lecture Slide Note:

Throughout this course, lectures and provided slides do not encompass all material in the course. They are meant to be complementary and not a substitute for reading material and HW.

What is an Embedded System?

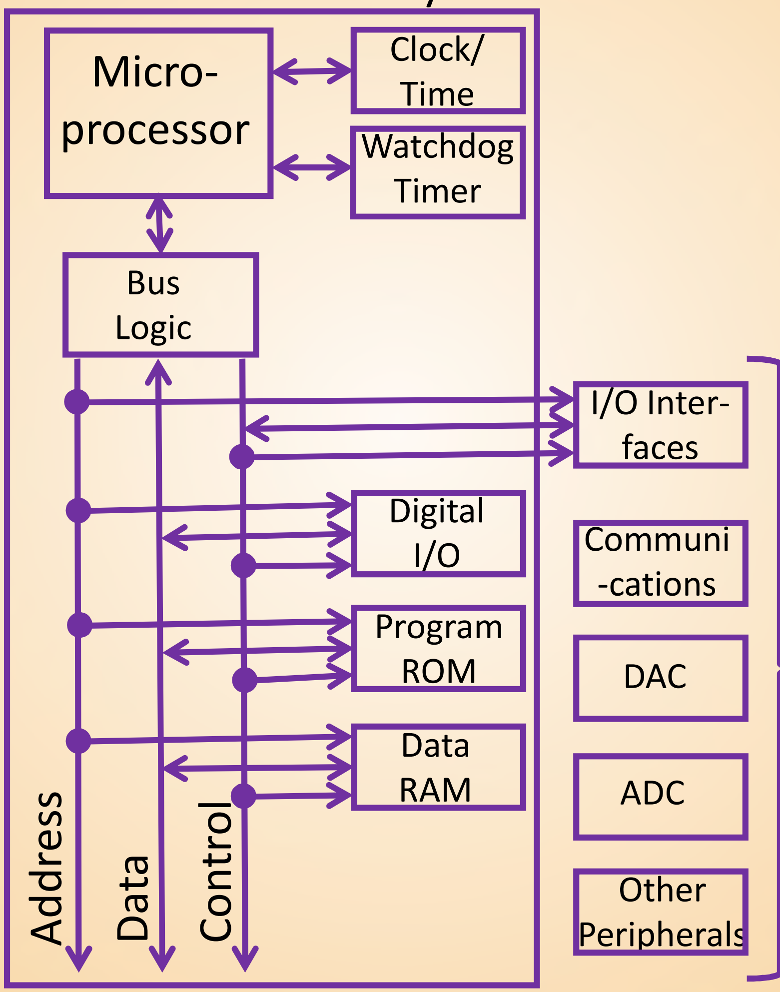
- Avoiding a formal definition, we can look at common characteristics:
  - A computer system embedded, collocated with hardware
  - A system with hardware (mechanical, electrical, etc...) and a processor to run software, often with a special set of constraints such as very low power, very small, reliable, real-time, etc...
  - To widely varying degrees, designed to perform a specialized task (reliably), in contrast to general purpose computer systems designed to be versatile and run a variety of software applications

A NOT so useful definition:

- An embedded system is a system embedded in a larger system.
  - In our context, "embedded system" will mainly refer to a microprocessor-based embedded system. It serves as the electronic control and processing unit for the system.

Diagram of a Microprocessor-based Embedded System

Minimal Embedded System:



Micro-controllers Embed these into the IC directly in many cases

Knowledge and Skill set required for Designers of Embedded Systems:

- Algorithm development
- Hardware-aware programming, need to know hardware resource implications of code, need to understand hardware to communicate with it
- Computer Architecture
- Hardware
- Circuits
- Processor-hardware interfaces
- User interfaces (hardware)
- Hardware and Embedded Software Debugging
- Designing embedded systems requires a range of skills from hardware skills to software skills.
- Tools -The diversity of hardware and software components means a verity of development tools are required. We will focus on software tools. The circuit components in this course should be simple enough that hardware design tools are not required.
- Successful system development comes from experience --knowing options and where problems will occur.

3 Levels of Programs:

- Machine Language
  - Numerical (opcode)
  - Menomics
- High Level Language
- Applications (and Games ;-)

Machine Level Language Hypothetical Example:

OPCODE		MNEMONIC	MEANING
000		ADD	Add memory to accumulator
001		SUB	Subtract memory from accumulator
010		LDA	Load accumulator from memory
011		STO	Store accumulator in memory
100		IOT	Input/Output Transfer: read or write from accumulator
101		PSH	Push accumulator to top of stack
110		POP	Pop top of stack to accumulator
111	000	INS	Initialize stack pointer
111	001	BRA	Unconditional branch
111	010	BMI	Branch if accumulator is negative
111	011	JSR	Jump to subroutine
111	100	RTS	Return from subroutine
111	101	HLT	Program halt, return to monitor

3 basic Programming Levels- Assembler, Machine Code: Fibonacci Example:

```
Fibonacci Numbers:  By definition, the first two
numbers in the Fibonacci sequence are either 1 and 1,
(or 0 and 1, depending on the chosen starting point
of the sequence,) and each subsequent number is the
sum of the previous two.

Example:
   1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
```

```
OPCODE    MNEMONIC  MEANING
-----------------------------------------------------
  000       ADD     Add memory to accumulator
  001       SUB     Subtract memory from accumulator
  010       LDA     Load accumulator from memory
  011       STO     Store accumulator in memory
  100       IOT     Input/Output Transfer:
                      0 = Input Int,   1 = Input Char,
                      2 = Output Int, 3 = Output Char
  101       PSH     Push accumulator to top of stack
  110       POP     Pop top of stack to accumulator
  111  000  INS     Initialize stack pointer
  111  001  BRA     Unconditional branch
  111  010  BMI     Branch if accumulator is negative
  111  011  JSR     Jump to subroutine
  111  100  RTS     Return from subroutine
  111  101  HLT     Program halt, return to monitor
-----------------------------------------------------
            BSS     Reserve Space
            END     Set Pointer to Label
            ORG     Set next address at this location
            OCT     Set an Octal Constant
            DEC     Number entered as decimal
```

```
LABEL OPCODE ADDRESS        ADDRESS     DATA
--------------------        ----------------
UNUSED EQU    2047
*
       ORG    20
TEMP   DEC    140             020      0214
NEXT   OCT    UNUSED          021      2047
LATEST BSS    1               022
LIMIT  BSS    1               023
SUM    BSS    1               024
*
       ORG    40
START  INS    777             040      7000  0777
       JSR    INPUT           042      7300  0045
*
       LDA    NEXT            044      2021
       IOT    2               045      4002
       LDA    LATEST          046      2022
       IOT    2               047      4002
LOOP   LDA    LATEST          050      2022
       ADD    NEXT            051      0021
       STO    SUM             052      3024
       IOT    2               053      4002
       LDA    LIMIT           054      2023
       SUB    SUM             055      1024
       BMI    STOP            056      7200  0066
       LDA    LATEST          060      2022
       STO    NEXT            061      3021
       LDA    SUM             062      2024
       STO    LATEST          063      3022
       BRA    LOOP            064      7100  0050
STOP   HLT                    066      7500
*
INPUT  IOT    0               067      4000
       STO    NEXT            070      3021
       IOT    0               071      4000
       STO    LATEST          072      3022
       IOT    0               073      4000
       STO    LIMIT           074      3023
       RTS                    075      7400
*
       END    START
```

3 basic Programming Levels - High Level Language

Example – C script:

```
      if (argv[1] == NULL)
           {
           printf("\nError:  The file name to be searched must be supplied\n\n");
           return;
           }

        if ((fpin = fopen(argv[1], "rb")) == NULL)
           {
           printf("Error:  the file to be searched, '%s',\n",argv[1]);
           printf("           could not be found.\n");
           return;
           }

        printf("       upper                  lower\n");
        printf("        byte   cm_crc inter    byte   cm_crc value\n");
        printf("       -----   ------------   -----   ------------\n");
        while ( fgets(line_buf, MAX_LINE_SIZE + 1, fpin) != (char *) NULL)
           {
           j = 0;
           i = 0;
           while (j = sgetw(line_buf, wordval[i], MAX_NAME_SIZE - 1, j)) { i++; }

           if ((line_buf[0] == '0') && (line_buf[1] == 'x'))
              {
              CRC_line_count++;

              if ( sscanf(line_buf,"%8x", &U16) == 1)
                 {
                 ch_upr = U16 >> 8;
                 ch_lwr = U16 & 0xff;
```

3 basic Programming Levels - High Level Language

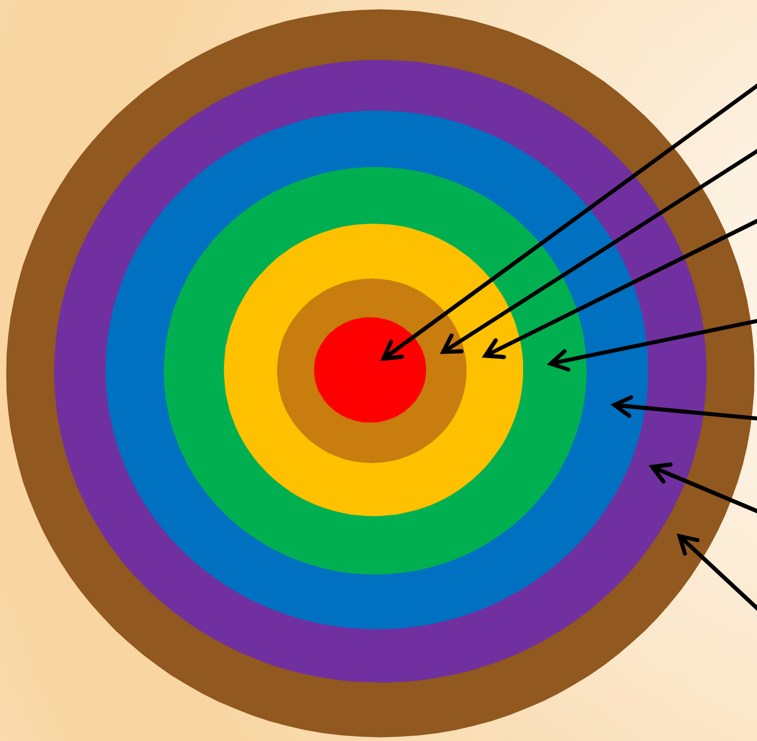
Example – UNIX operating script:

```
cd ../characterization
mkdir $1.dir
cd $1.dir

for A in ../../template_vlev/*_all ;
   do
      B=`echo $A | sed "s?../../template_vlev/??" | sed "s?_all??" | sed "s?qqq?$1?"`;
      echo "  ->>> Creating the new ibis file: $B" ;
      sed "s?qqq?$1?g" $A > $B;
   done
chmod +x x*.pl

for A in ../../template_vlev/*_"$2" ;
   do
      B=`echo $A | sed "s?../../template_vlev/??" | sed "s?_"$2"??" | sed "s?qqq?$1?"`;
      echo "  ->>> Creating the new ibis file: $B" ;
      sed "s?qqq?$1?g" $A > $B;
   done

ln -s ../../../../hspice/$1.inc $1.inc
#sed "s/\.DEVICE/\*\.DEVICE/" ../../../../adv/$1.adv > $1.adv
ln -s ../../process_files/$3/$4hc.dat $4hc.dat
ln -s ../../process_files/$3/$4lc.dat $4lc.dat
ln -s ../../process_files/$3/$4nc.dat $4nc.dat

chmod +x run_make_typ_max
chmod +x run_ibis_assemble
chmod +x run_sim
```

3 basic Programming Levels - Applications:

Hardware Application Onion Model - Expanded from 3 levels…

- Figure 6.0 from textbook



-------- - Hardware-Registers - Programmed by rewiring

1st GL - Machine Language - 0x001F, 0x0002, 0xB016

2nd GL - Assembler - ADD R1, R2
- Unique to the Machine

3rd GL - High-Level Language - a += b;
- C, Pascal

3rd GL - Higher Level Language - Advanced Programming concepts, e.g.OOP
- C++, Java

4th GL - e.g. CASE - (Computer Aided Software Engineering)

-------- - Application Programs

Examples of Embedded Systems – CMPE310 Review

- Consumer electronics
  Cell phones, pagers, digital cameras, camcorders, PDAs, DVD players, calculators
- Home Appliances
  Microwave ovens, answering machines, thermostats, home security systems
- Office Automation
  Fax machines, copiers, printers, scanners
- Business Equipment
  Cash registers, curbside check-in, alarm systems, card readers, ATMs, product scanners
- Automobiles
  Transmission control, cruise control, fuel injection, antilock brakes, active suspension
- Computing system peripherals and networking systems
  Networking equipment, routers, printers, network cards, monitors and displays
- …And many more
  E.g. In 1999, a typical American household had one PC and about 40-50 embedded computers, this has risen to about 300-400 by 2004. An average car can have more than 50 embedded computers.

System Characteristics – CMPE310 Review

Single-functioned

- Usually executes a specific program repeatedly
- Exceptions are in cases when a system's program is updated with a newer program
- Program can be swapped in and out of the system due to size limitations, depending on the function required at a specific time

Tightly Constrained

- Tight constraints for design metrics such as cost, size, performance and power

Reactive and Real Time

- Many systems must continually react to changes in system's environment and must compute certain results in real time without delay e.g. car cruise control
- Contrast to desktop systems that typically focus on computations with relatively infrequent reactions to input devices (from the computer's perspective).
- Delay in computations on desktop systems, while inconvenient to the computer user, typically does not result in a system failure

The Processing Unit – CMPE310 Review

Microprocessor

- The CPU is a unit that fetches and processes a set of general-purpose instructions
- The CPU instruction set includes instructions for data transfer, ALU operations, stack operations, input and output (IO) operations and program control, sequencing and supervising operations
- A microprocessor is a single VLSI chip that has a CPU and may also have other units (e.g. caches, floating point processing arithmetic unit, pipelining and super-scaling units) that are additionally present and result in faster processing of instructions.
- A system designer need not be concerned about the design of the microprocessor, only needs to understand the architecture related to the programming of the processor's memory to carry out the required functionality i.e. implement the software
- Examples: Intel 8085, Intel x86 processors, Motorola 68HCxxx, Sun Sparc, IBM PowerPC etc.
- Time-to-market and NRE costs are lower when systems are designed with microprocessors as the designer must only write a program. Flexibility is also high.
- Unit cost may be low in small quantities compared with designing a dedicated chip
- Performance varies by application, unit cost may be high for larger volumes, size and power might be higher due to unnecessary processor hardware

The Processing Unit – CMPE310 Review

Microcontroller

- A microcontroller is a single chip unit which, though having limited computational capabilities, possesses enhanced input-output capabilities and a number of on-chip functional units
- Particularly suited for use in embedded systems for real-time control applications with on-chip program memory and devices
- Common peripherals include serial communication devices, timers, counters, pulsewidth modulators, analog-to-digital and digital-to-analog convertors
- Enables single-chip system implementation and hence smaller and lower-cost products
- Examples: Motorola 68HC11xx, HC12xx, HC16xx, Intel 8051, 80251, PIC 16F84, PIC18, ARM9, ARM7, Atmel AVR etc.

Single-Purpose Processor

- A digital circuit designed to execute exactly one program
- Commonly referred to as coprocessor, accelerator or peripheral
- Examples JPGE codec, Serial-to-Ethernet convertor, etc.

The Processing Unit – CMPE310 Review

Digital Signal Processor

- Essential for systems that require large number of operations on digital signals, which are the digital encoding of analog signals like video and audio
- They carry out common signal processing tasks like signal filtering, tranformations or combinations
- Used widely in image processing applications, multimedia, audio, video, HDTV, DSP modem and telecommunication processing systems.
- They perform math-intensive operations, including operations like multiply and add or shift and add etc.
- Examples: TI TMS320Cxx, Analog Devices SHARC, Motorola 5600xx, etc.

Application Specific Instruction-Set Processors (ASIPs or ASSPs)

- A programmable processor optimized for a particular class of applications having common characteristics. Microcontrollers and DSPs can be considered as ASSPs.
- ASIPs are available for broad application classes (e.g. graphics processor) as well as very small application classes, some as small as a handful of programs
- Example: ASSP chip with TCP, UDP, IP, ARP and Ethernet 10/100 MAC logic.

The Processing Unit – CMPE310 Review

Programmable Logic Devices (PLD)/ Field Programmable Gate Arrays (FPGA)

- Contains general purpose logic elements that can be programmed to implement desired functionality, very flexible for implementing custom logic circuits
- PLD usually are smaller and contain programmable gates like AND/OR arrays
- FPGAs provide lot more functionality and can be used to implement complex designs
- FPGAs can have on-chip microprocessors, memory, DSP, communication devices
- Examples: Xilinx Virtex, Spartan series FPGAs, Actel, Altera, Lattice, QuickLogic

Application Specific Integrated Circuits (ASICs)/ System-on-a-chip (SOCs)

- Custom designed VLSI chips that perform the required function
- Functionality can be integrated using IP (Intellectual property) cores
- General purpose processors are also available as IP cores and can be integrated on the chip
- Embedded processors are available from ARM, Intel, Texas Instruments and various other vendors
- Only feasible for high volume, relatively high cost systems as NRE costs and time-to-market can be significant