Chapter 5 – System Design I: Functional Decomposition

1. **Describe the differences between *bottom-up* and *top-down* design. [R]**

   In a bottom-up design, the designer begins with basic components and synthesizes them to create the overall system. However, the top-down designer has a vision of the overall functionality of the final system and partitions the problem into components, or subsystems. These components work together to achieve the overall goal. In essences, bottom-up designers are given parts and then they build the system, while top-down designers envision the system and then build the parts.

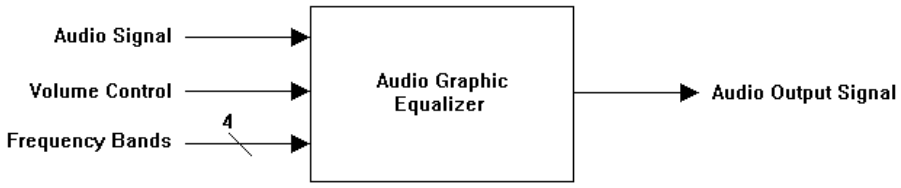
2. **Develop a functional design for an audio graphic equalizer. A graphic equalizer decomposes an audio signal into component frequencies bands, allows the user to apply amplification to each individual band, and recombines the component signals. The design can employ either analog or digital processing. Be sure to clearly identify the design levels, functional requirements, and theory of operation for the different levels in the architecture. [A]**

   The system must

   - **Accept an audio input signal source, with a source resistance of 1000Ω and a maximum input voltage of 1V peak-to-peak.**
   - **Have an adjustable volume control.**
   - **Deliver a maximum of 40W to an 8Ω speaker.**
   - **Have four frequency bands into which the audio is decomposed (you select the frequency ranges).**

   LEVEL 0

   The Level 0 functionality for the audio graphic equalizer, shown below, is fairly simple – the inputs are an audio signal, volume control, and four frequency band knobs, and the output is a reconstructed audio signal.



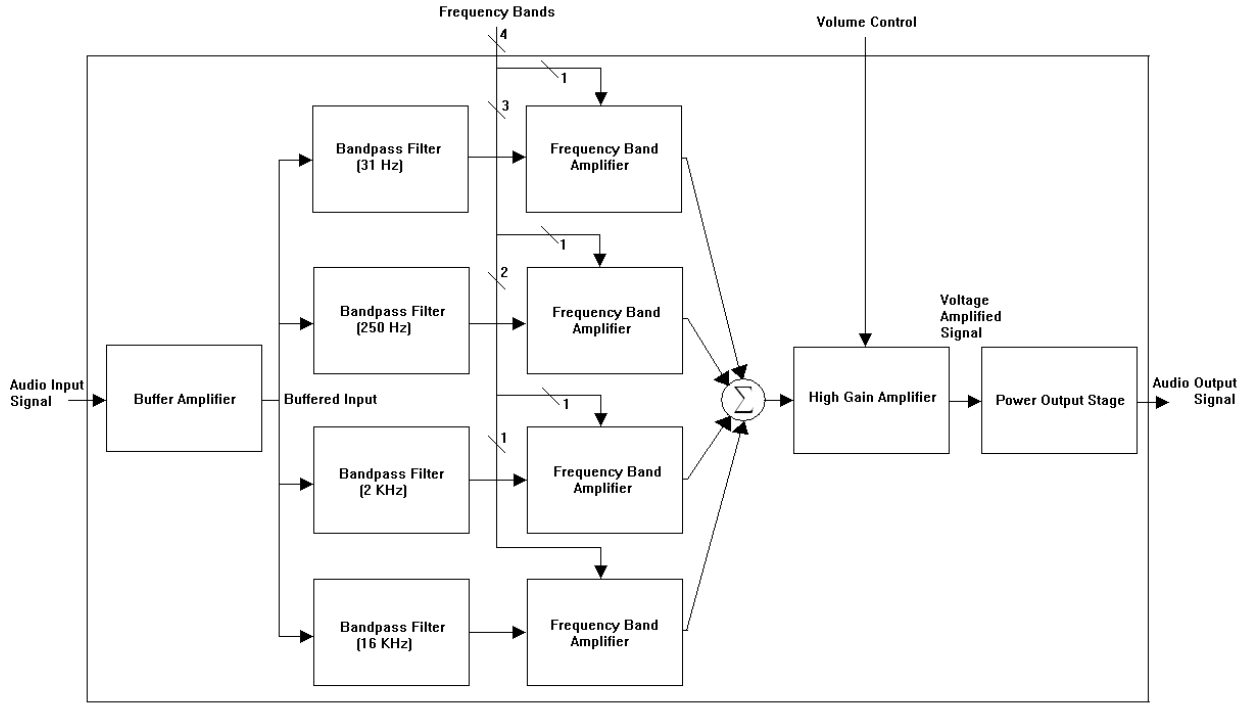
Audio Equalizer System Functionality

Module	Audio Graphic Equalizer
Inputs	- Audio input signal: 1Vpp , 1000Ω - User volume control: variable control - Frequency band knobs: 4 bands (31Hz, 250Hz, 2KHz, 16KHz) , variable control.
Outputs	- Audio output signal: ?V peak value
Functionality	Reconstructs the input audio signal by amplifying and/or deamplifying each of the 4 frequency bands to produce a 40W maximum output signal. The volume amplification and frequency knobs should have variable user control.

Not all values can be known on the first pass through the design. However, knowing the maximum power allows the maximum output voltage to be computed as

$V_{peak} = \sqrt{8\Omega * 40W} = 17.9\text{V}.$

LEVEL 1



Audio Graphic Equalizer Level 1 Architecture

There are several unknowns at this time, and that will be apparent as we investigate the individual modules. The functional requirements for the Level 1 architecture are as detailed as possible, starting with the buffer amplifier.

Module	Buffer Amplifier
Inputs	- Audio input signal: 1Vpp , 1000Ω
Outputs	- Audio output signal: 1Vpp , 1000Ω
Functionality	Buffer the input signal and provide unity voltage gain. it should have an input resistance of > 100kΩ and an output resistance <100Ω.

The input resistance is an educated guess, but is a realistically high value that will minimize any voltage losses with the source resistance. The output resistance is an educated guess, based on knowledge of what is achievable with the technology (bottom-up knowledge). These resistance values are later refined and adjusted, taking into account the input and output resistances for all stages.

Module	Bandpass Filters (0-31Hz, 31-250Hz, 250-2KHz, 2kHz-16KHz)
Inputs	- Buffered input signals: 1Vpp , 1000Ω
Outputs	- Audio output signal: 17.9V peak value
Functionality	This filters out each of the desired frequency ranges so that they may be adjusted independently. It should have an input resistance >1000Ω and an output resistance <100Ω.

Module	Frequency Band Amplifiers
Inputs	- Filtered audio signals: 1Vpp , 1000Ω - Frequency band knobs: 4 bands (31Hz, 250Hz, 2KHz, 16KHz) , variable control.
Outputs	- Audio output signal: 17.9V peak value
Functionality	Provide an adjustable voltage gain, between 1 and 35, for each of the 4 frequency bands, respectively. It should have an input resistance >10,000Ω and an output resistance <100Ω.

Module	Signal Summation
Inputs	- Audio input signals (31Hz, 250Hz, 2KHz, 16KHz): 1Vpp , 1000Ω
Outputs	- Audio output signal: 17.9V peak value
Functionality	This recombines each of the 4 frequencies that have be independently adjusted into one complete signal. It should have an input resistance >1000Ω and an output resistance <100Ω.

Module	High Gain Amplifier
Inputs	- Audio input signal: 1Vpp , 1000Ω - User volume control: variable control
Outputs	- Audio output signal: 17.9V peak value
Functionality	Provide an adjustable voltage gain, between 1 and 35. It should have an input resistance >10,000Ω and an output resistance <100Ω.

Module	Power Output Stage
Inputs	- Audio input signal: : 17.9V peak
Outputs	- Audio output signal: 17.9V peak at up to ???A.
Functionality	Provide unity voltage gain with output current as required by a resistive load of up to ???A. It should have an input resistance >1000Ω and an output resistance <100Ω.

Note: *there are other architectures that can work here as well. For example, the high gain amplifier and power output stage can be combined into a single amplifier. In addition, the filter and associated amplifiers might also be combined together, depending upon the filter topology used.*

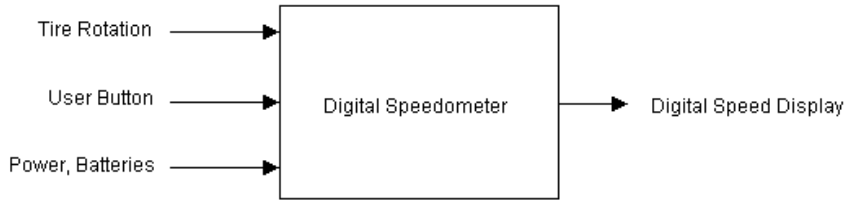
3. **Develop a functional design for a system that measures and displays the speed of a bicycle. Be sure to clearly identify the design levels, functional requirements, and theory of operation for each level. [A]**

The system must

- **Measure instantaneous velocities between zero and 75 miles per hour with an accuracy of 1% of full scale.**
- **Display the velocity digitally and include one digit beyond the decimal point.**
- **Operate with bicycle tires that have 19, 24, 26, 27 inch radii.**

LEVEL 0

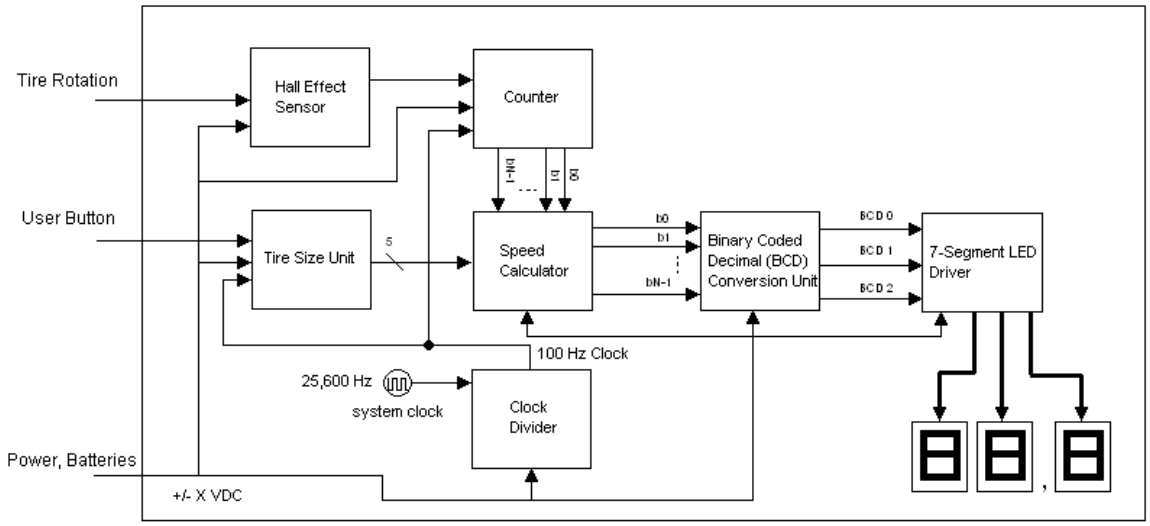
The overall goal is to convert a sensed speed to a digital speed reading (i.e. speedometer).



Digital Speedometer System Functionality

Module	Digital Speedometer
Inputs	- Tire Rotation (magnet) - User Button - Power. Batteries to power (_ VDC)
Outputs	- Digital speed display. 0-75 MPH. A total of three digits displayed, including one digit beyond the decimal point.
Functionality	Displays speed on a digital readout with 1% accuracy over the speed range.

LEVEL 1



Digital Speedometer Level 1 Architecture

There are several unknowns at this time, and that will be apparent as we investigate the individual modules. The most important unknown is the voltage level that we will be supplying these devices with. This can not be determined until we know what devices we will be using.

Module	Hall Effect Sensor
Inputs	- Tire Rotation (magnetic flux) - Power. __ VDC to power this device
Outputs	- Vr = voltage due to rotation. ___ VDC represents logic '1', 0 VDC represents logic '0'.
Functionality	Outputs a ___ VDC to indicate that a complete rotation of the bicycle wheel has occurred.

Module	Counter
Inputs	- Vr from Hall Effect Sensor. Indicates complete tire rotation. - Power. ___ VDC to power electronics. - Clock. Increments counter.
Outputs	- ___ bit binary representation of the count between wheel rotations
Functionality	Counts in 10ms increments when Vr is low and resets when Vr is high.

The counter will only need to count in intervals of 10 ms (100Hz) because the fastest recording of speed will be at 75 MPH using a 19 inch tire. This equates to about 22.11 rotations per minute. Therefore 100ths of a second would be sufficient. Refer below.

75 miles / hour * hour / 3600 sec * 63360 inches / mile = 1320 inches/s (fastest)

NOTE: The time the Hall Effect sensor will activate would be shortest with a tire that has the shortest circumference. This tire would be the 19 inch tire

C = pi*d = 3.14 * 19 = 59.69 inches → 1320 inches/sec * rotation/59.96 inches
= 22.11 rotations/s

The size of the bit value passed should be based on the slowest a person could go (longest count time) considering the accuracy of our device. Refer below.

1% accuracy of 75 MPH = .01 * 75 = 0.75 MPH (the slowest recordable speed)

.75 miles / hour * hour / 3600 sec * 63360 inches / mile = 13.2 inches/s (fastest)

NOTE: The time the Hall Effect sensor will activate would be longest with a tire that has the longest circumference. This tire would be the 27 inch tire.

C = pi*d = 3.14 * 27 = 84.83 inches → 13.2 inches/sec * rotation / 84.83 inches
= 0.1556 rotations/s
or (by inversing)
= 6.5 sec / rotation

Therefore, the number of placements the counter needs to hold is 6.5 seconds (using 10 ms increments), which turns out to be 650 counts. This can be easily obtained with a 10-bit representation (1024 counts).

Module	Tire Size Unit
Inputs	- Button. Enable/reset counter. - Clock. Increment counter. - Power. ___ VDC to power the electronics.
Outputs	- 5 bit binary representation of the tire size chosen.
Functionality	Converts the length of a button press to its respective bicycle tire size.

The tire size unit also counts, but only counts the length of the button press by the user. The determined button press is mapped (decoded) to its corresponding association – tire size. Here a 1 second press would equate to 19 inch tire, 2 seconds would be 24 inch, 3 seconds would be 26 inch, and 4 seconds would be a 27 inch tire. I used 5 bits to represent the tire size, because $2^5 = 32$, and those values are easily represented in 5 bits.

Module	Speed Calculator
Inputs	- Vr. Latch timer value and tire size value. - 5 bit binary number. Represents the tire sizes 19, 24, 26, and 27. - 10 bit binary number. Represents the number of time units (10ms) that have completed a rotation.
Outputs	- ___ bit binary representation of the bicycle speed.
Functionality	Properly converts the speed of the bicycle depending on it's tire size.

Here the output of the Speed calculator is unknown in size. The accuracy of our device comes into play. The number of bits needed for the BCD converter is calculated from the maximum allowable error that the speed calculator can introduce (0.75 MPH – determined above). Below is the calculation.

$$\text{Max error} = \text{range} / \#\text{ or intervals} = 75 \text{ MPH} / 2^N \leq 0.75 \text{ MPH} \rightarrow N \geq 6.65 \text{ bits}$$

So the Speed Calculator needs to have at least 7 bits.

Module	BCD Conversion Unit
Inputs	- 7 bit binary number. Represents range 0.0 - 75.0 MPH. - Power. ___ VDC.
Outputs	- BCD0 = 4-bit BCD representation of tenths digit (after decimal). - BCD1 = 4-bit BCD representation of one's digit. - BCD2 = 4-bit BCD representation of ten's digit.
Functionality	Converts the 7 bit binary number to BCD representation of speed. Must refresh display at least twice per second.

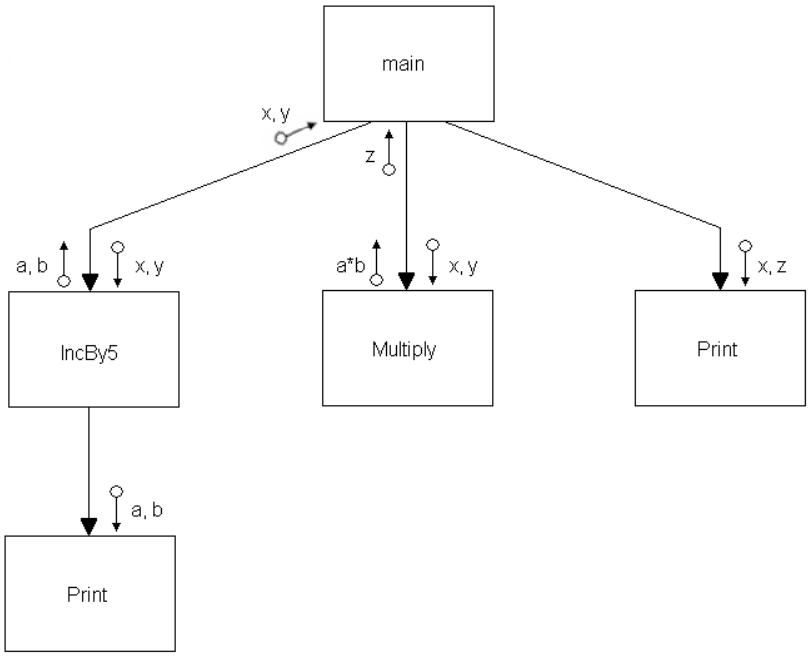
Module	7-Segment LED Driver
Inputs	- BCD0 = 4-bit BCD representation of tenths digit (after decimal). - BCD1 = 4-bit BCD representation of one's digit. - BCD2 = 4-bit BCD representation of ten's digit. - Power. ___ VDC.
Outputs	- Three 7-segment driver lines
Functionality	Converts the BCD for each digit into outputs that turn on LEDs in 7-segment package to display the speed.

Module	Clock Divider
Inputs	- System clock = 32,768 Hz
Outputs	- Internal clock = 10 Hz
Functionality	Divides the system clock by 3276 to produce 10 Hz clock

The value of 25,600 Hz was selected for the system clock for one main reason. This reason is due to the fact that 25600 is easily dividable by 256 (8 shifts right) to produce 100 Hz.

4. Draw a structure chart for the following C++ program:[A]

```
void INcBy5(int &a, int &b);
int Multiply(int a, int b);
void Print(int a, int b);

main(){
      int x=y=z=0;
      IncBy5(x,y);
      z=Mult(x,y);
      Print(x,z);
}

void IncBy5(int &a, int &b){
      a+=5;
      b+=5;
      Print(a,b);
}

int Multiply(int a, int b){
      return (a*b);
}

void Print(int a, int b){
      cout << a << ", " << b;

}
```



Structural Chart of C++ Program

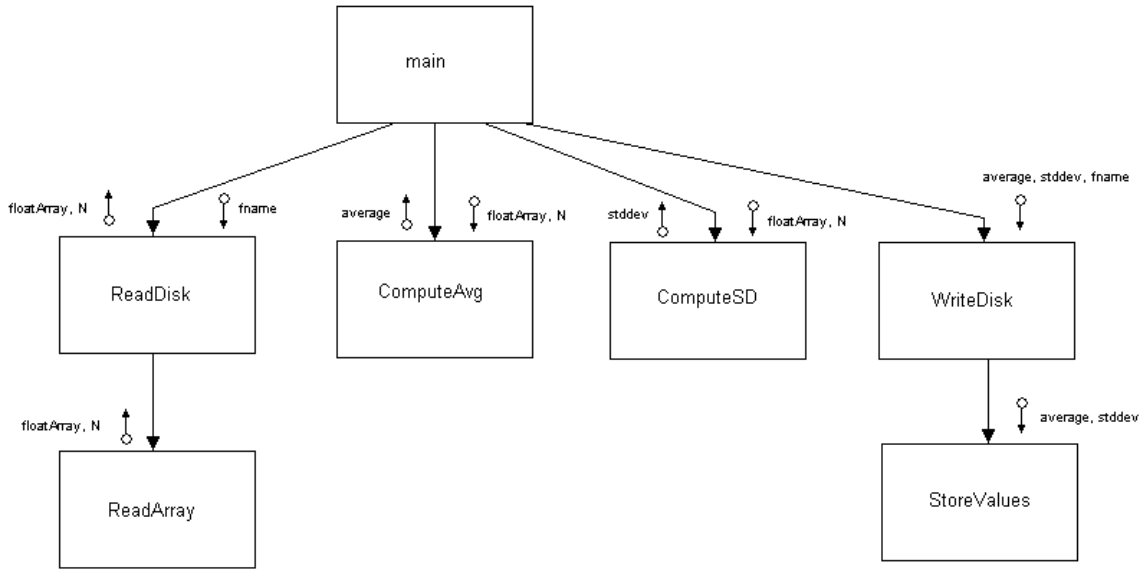
5. **Develop a functional design for software that meets the following requirements. [A]**

   The system must

   - **Read an array of floating point numbers from an ASCII file on disk.**
   - **Compute the average, median, and standard deviation of the numbers.**
   - **Store the average, median, and standard deviation values on disk.**

   The design should have multiple modules and include the following elements: (a) a structure chart, and (b) a functional description of each module.

Structure Chart



Structural Chart of Software Design

Functional Description

Module	main()
Module Type	Coordination
Input Arguments	None
Output Arguments	None
Description	The main function that calls ReadDisk() to read input file, ComputeAvg() to compute the average of the array values, ComputeSD() to compute the standard deviation of the array values, and WriteDisk() to store the average and deviate on disk.
Files Accessed	None
Files Changed	None
Modules Invoked	ReadDisk, ComputeAvg, ComputeSD, WriteDisk

Module	ReadDisk()
Module Type	Input and output
Input Arguments	- fname[] - character array with file name to read from
Output Arguments	- floatArray[] - floating point array with elements read from file. - N - number of elements within floatArray[]
Description	Opens the data file specified by the input argument. It returns the values by ReadArray().
Files Accessed	Input data file
Files Changed	None
Modules Invoked	ReadArray

Module	ReadArray()
Module Type	Input and output
Input Arguments	None
Output Arguments	- floatArray[] - floating point array with elements read from file. - N - number of elements within floatArray[]
Description	Reads the data from the currently file opened (by ReadDisk) and stores the elements in floatArray[]. The number of elements is placed in N.
Files Accessed	Input data file
Files Changed	None
Modules Invoked	None

Module	ComputeAvg()
Module Type	Input and output
Input Arguments	- floatArray[] - floating point array with elements read from file. - N - number of elements within floatArray[]
Output Arguments	- average - the average (mean) value of the floating point values in floatArray[]
Description	Computes the average (mean value) of the floating point elements in the array
Files Accessed	None
Files Changed	None
Modules Invoked	None

Module	ComputeSD()
Module Type	Input and output
Input Arguments	- floatArray[] - floating point array with elements read from file. - N - number of elements within floatArray[]
Output Arguments	- stddev - the standard deviation value of the floating point values in floatArray[]
Description	Computes the standard deviation of the floating point elements in the array
Files Accessed	None
Files Changed	None
Modules Invoked	None

Module	WriteDisk()
Module Type	Input and output
Input Arguments	- fname[] - character array with file name to write to - average - the average (mean) value of the floating point values in floatArray[] - stddev - the standard deviation value of the floating point values in floatArray[]
Output Arguments	None
Description	Opens the data file specified by the input argument.
Files Accessed	Output data file
Files Changed	None
Modules Invoked	StoreValues

Module	StoreValues()
Module Type	Input and output
Input Arguments	- average - the average (mean) value of the floating point values in floatArray[] - stddev - the standard deviation value of the floating point values in floatArray[]
Output Arguments	None
Description	Writes the values of the input arguments to the file currently opened.
Files Accessed	Output data file
Files Changed	Output data file
Modules Invoked	None

6. **Describe in your own words what is meant by coupling in design. Describe the advantages of both loosely and tightly coupled designs. [R]**

   Coupling describes a particular individual module's dependence upon the interconnectivity of various modules for proper functionality. A module can be loosely coupled or tightly coupled. The advantages of these are as follows:

   Loosely Coupled
   (+) The maximum degree of impact one module can have is limited
   (+) May allow for continued functionality upon module failures
   (+) Maximizes the cohesion of a design
   (+) Allows for independent testing of modules

   Tightly Coupled
   (+) Better performance (i.e. software)
   (+) Quicker solutions (not necessarily better)

   It should be noted that the number of advantages of a loosely coupled design usually outweigh that of a tightly coupled design.

7. **Project Application. Develop a functional design for your project. Follow the presentation guidelines in Section 5.9 for communicating the results of the design. [P]**

   Note: *this is a major deliverable that is due for the project. It is also tied closely in with Chapter 6 which also provides modern design techniques. The information that we use in our assignment is below.*

System Design Deliverable

CENBD 480/EE BD 480 - Engineering Design Concepts

Penn State Erie, The Behrend College Fall 200X

The capstone project system design is due on Friday. The team needs to demonstrate that is has done the following:

- Analyzed and evaluated different options/concepts for the design. This means that you should have examined different alternatives and be able to justify the choices the team made. The choices could be in terms of different design architectures and/or different decisions for elements of the overall system. Apply the methods from Chapter 4 that are appropriate for the problem. The results can be presented in terms of a list of brainstormed ideas, Concept Tables/Fans and Decision matrices. If you do use decision matrices, you can use them to benchmark or compare different technical solutions considered. Only use the highly quantitative matrix (i.e. Example 4.1) if it realistic and applicable to the problem.

- Developed a clear and feasible system design. The objective is to describe the major systems and the subsystems of the design. My advice:

    - Describe the behavior of the system using a combination of the following techniques:
        - Functional Decomposition.
        - Flow Charts.
        - State Diagrams.
        - Data Flow Diagrams.
        - Entity Relationship Diagrams.
        - UML – different views.
        - Other methods of describing behavior appropriate for the problem.

    - Fill in as many of the details as possible. If you don't know the exact value, leave it as a question mark (it is best to identify as something that needs to be determined). The idea is to start answering the important questions and designs that you will need for the system. Make sure you go down as deep as possible in the hierarchy (you can stop at the component level, but can go further if you like).

    - Have a narrative description of how the system works. This is a description of the how all of the architectural pieces work together. Keep in mind that the design needs to meet the engineering requirements, so indicate how it meets them (as necessary).

    - Software & algorithms. If you have a significant software component, make sure that it is properly designed using the above tools.

- Mechanical systems. If you have a mechanical aspect to the system, say a robot, create mechanical drawings of the system. Detailed drawings will be needed at the start of the spring semester. It is best to submit the work orders to the mechanical technicians early as they often have a long backlog.