Embedded Systems & Internet of Things

Unit I: Embedded Computing

1.1 Introduction:

- Why we embed microprocessors in systems
- What is difficult and unique about embedding computing and cyber-physical system design
- Design methodologies
- System specification

→ In order to understand design process, first we need to understand how and why microprocessors are used for control, user interface, signal processing and many other tasks.

→ In this chapter we discuss the various uses of microprocessors, major reasons why microprocessors are used in system design.

1.2 Complex systems and microprocessors:

→ A computer is a stored program machine that fetches and executes instructions from a memory.

→ We can attach different types of devices to the computer, load it with different types of software, and build many different type of systems.

What is embedded computer system:

→ It is any device that includes a programmable

computer but is not itself intended to be a general purpose computer. Thus, a PC is not itself an embedded computing system. But a fax-machine or a clock built from a microprocessor is an embedded computing system.

→ Embedded computing system design is a useful skill for many types of product design.

Ex: Automobiles, cell phones and even house hold appliances make extensive use of microprocessors.

→ Designers in many fields must be able to identify where microprocessors can be used, design a hardware platform with I/O devices that can support the required tasks, and implement software that performs the required processing.

→ Embedded computing system design does not stand alone.

→ Many of the challenges encountered in the design of an embedded computing system are not computer engineering — for example they may be mechanical or analog electrical problems.

2.1 Embedding computers:

Whirlwind:

→ Computer designed at MIT in late 1940s to support real-time operations.

→ conceived as a mechanism for controlling an aircraft simulator.

→ Physical size was very large compared to today's computers.

→ A microprocessor is a single-chip CPU.

→ VLSI (Very large scale Integrated) technology has enabled to put a complete CPU on a single chip. but those CPUs were very simple.

→ Intel 4004, the first microprocessor was designed for embedded application, namely; a calculator.

→ The calculator was not a general purpose computer- it merely provided basic arithmetic functions.

→ Ted Hoff of Intel realized that a general purpose computer programmed properly could implement the required function, and that the computer-on-chip could then be reprogrammed for use in other products as well.

→ Because integrated circuit design was (and still) is an expensive and time-consuming process,

the ability to decide the hardware design by changing the software was a key break-through.
Ex: HP-35 was the first hand held calculator to perform transcendental functions.
Introduced in 1972, HP-35 used several chips to implement the CPU, rather than a single-chip microprocessor.

Microprocessors in Automobiles

→ Automobile designers started making use of the microprocessor soon after single-chip CPUs became available.

→ Important and sophisticated use of microprocessors in automobiles was to control the engine, determining when spark plugs fire, controlling the fuel/air mixture and so on.

→ The combination of low fuel consumption and low emissions is very difficult to achieve; to meet these goals without compromising engine performance, automobile manufacturers turned to sophisticated control algorithms that could be implemented only with microprocessors.

→ Microprocessors come in different levels of sophistication. They are usually classified by their word size.

→ An 8-bit microcontroller is designed for low-cost applications and includes on-board memory and

I/O devices; a 16-bit microcontroller is often used for more sophisticated applications that may require either longer word lengths or 'off-chip' I/O and memory; and a 32-bit RISC microprocessor offers very high performance for computation-intensive applications.

Household uses of microprocessors:

→ Microwave oven has atleast one microprocessor to control oven operation.

→ Thermostat systems which change the temperature level at various times during the day.

→ The modern camera is a prime example of the powerful features that can be added under microprocessor control.

→ Digital television makes extensive use of embedded processors.

→ A programmable CPU is used rather than a hardwired unit for two reasons.
i) it made the system easier to design and debug
ii) allows upgrades

→ A high-end automobile may have 100 microprocessors, but even inexpensive cars today use 40 microprocessors.

→ Some of these microprocessors do very simple things such as detect whether seat belts are in use.

others control critical functions such as the ignition and braking systems.

Design Example 1.1. BMW 850i Brake and stability control system.

2.2 Characteristics of embedded computing applications

→ Functionality is important in both general purpose computing and embedded computing, but embedded applications must meet many other constraints as well.

i) Embedded computing systems have to provide sophisticated functionality

a) Complex algorithms:

→ The operations performed by the microprocessor may be very sophisticated.

→ For example, the microprocessor that controls an automobile engine must perform complicated filtering functions to optimize the performance of the car while minimizing pollution and fuel utilization.

→

b) User Interface:

→ Microprocessors are frequently used to control complex user interfaces that may include multiple menus and many options.

→ The moving map in Global Positioning System (GPS) navigation are good examples of sophisticated user interfaces

ii) Embedded computing operations must often be performed to meet deadlines.

a) Real time: Many embedded computing systems have to perform in real time - if the data isn't ready by a certain deadline, the system breaks.

→ In some cases, failure to meet a deadline is unsafe and can even endanger lives.

→ In other cases, missing a deadline doesn't create safety problems but does create unhappy customers - missed deadline in pointers, for example, can result in scrambled pages

b) Multirate:

→ Not only must operations be completed by deadlines, but many embedded computing systems have several real-time activities going on at the same time.

→ They may simultaneously control some operations that run at slow rates and others that run at high rates.

→ Multimedia applications are prime examples of multirate behavior.

→ The audio and video portions of a multimedia stream run at very different rates, but they must remain closely synchronized.

→ Failure to meet a deadline on either the audio or video portions spoils the perception of the entire presentation.

iii) Costs of various sorts are also very important

a) Manufacturing cost:

→ The total cost of building the system is very important in many cases.

→ Manufacturing cost is determined by many factors, including the type of microprocessor used, the amount of memory required, and the types of I/O devices.

b) Power and energy:

→ Power consumption directly affects the cost of the hardware, because a larger power supply may be necessary.

→ Energy consumption affects battery life, which is important in many applications, as well as heat consumption, which can be important even in desktop applications.

1.2.3 Why use Microprocessors?

→ There are many ways to design a digital system: custom logic, field programmable gate arrays (FPGAs) and so on.

→ Why use microprocessors? There are two reasons

i) Microprocessors are a very efficient way to implement digital systems

ii) Microprocessors make it easier to design families of products that can be built to provide various feature sets at different price points and can be extended to provide new features to keep up with rapidly changing markets.

i) CPUs are flexible:

→ The paradox of digital design is that using a predesigned instruction set processor may in fact result in faster implementation of application than designing our own custom logic.

→ overhead of fetching, decoding, and executing instructions is so high that it cannot be recouped.

ii) CPUs are efficient:

→ There are two factors that work together to make microprocessor-based designs fast.

a) Microprocessors execute programs very efficiently: Modern RISC processors can execute several instructions per cycle. Overhead can be hidden by clever utilization of parallelism within the CPU.

b) CPUs are highly optimized

b) Microprocessor manufacturers spend a great deal of money to make their CPUs run very fast.

iii) Programmability:

→ The programmability of microprocessors provide substantial advantages makes them the best choice in a wide variety of systems.

→ The programmability of microprocessors can be a substantial benefit during the design process.

→ It allows program design to be separated from design of the hardware on which programs will be run.

While one team is designing the board that contains the microprocessor, I/O devices, memory and so on, others can be writing programs at the same time.

→ Programmability makes it easier to design families of products. In many cases, high-end products can be created simply by adding code without changing the hardware.

→ This practice substantially reduces manufacturing costs. It may be possible to reuse software, reducing development time and cost.

iv. How many platforms?

→ How many different hardware platforms are required for embedded computing systems.

→ PCs are widely used and provide a very flexible programming environment.

v. Real time:

→ Real time performance is often best achieved by multiprocessors.

vi. Low power and low cost:

→ low power and low cost also drive us away from PC architectures and toward multiprocessors.

vii. Smartphones as platforms:

→ Smartphone is an important computing platform. Cell phones operate on batteries, so they must be very power efficient.

1.2.4 Cyber-physical systems

→ A cyber-physical system is one that combines physical devices, known as the plant with computers that control the plant.

→ The embedded computer is the cyber-part of the cyber-physical system.

→ Certain trade-offs need to be made between the design of the control of the plant and the computational system that implements that control.

Computing as a physical act:

→ Computing is a physical act: Computers in fact do their work by moving electrons and doing work. That's why programs take time to finish, consume energy and so on.

Physics of software:

→ Software performance and energy consumption are very important properties when we are connecting our embedded computers to the real world.

→ We need to understand the sources of performance and power consumption if we are to be able to design programs that meet our application's goals.

1.2.5 Challenges in embedded computing system design

→ External constraints are one important source of difficulty in embedded system design.

i) How much hardware do we need?

→ We have a great deal of control over the amount of computing power we apply to our problem

→ Not only can we select the type of microprocessor used, we can select the amount of memory, the peripheral devices and more.

→ It is essential to meet both performance deadlines and manufacturing cost constraints, the choice of hardware is important – too little hardware and the system fails to meet deadlines, too much hardware and it becomes too expensive.

ii) How do we meet deadlines:

→ The brute force way of meeting a deadline is to speed up the hardware so that the program runs faster

→ That makes the system more expensive.

→ It is also possible that increasing the CPU clock rate may not make enough difference to execution time because the program's speed may be limited by the memory system.

iii) How do we minimize power consumption?

→ In battery powered applications, power consumption

is extremely important.

→ Even in non battery applications, excessive power consumption can increase heat dissipation.

→ One way to make a digital system consume less power is to make it run more slowly, but naively slowing down the system can obviously lead to missed deadlines.

→ Careful design is required to slow down the non critical parts of the machine for power consumption while still meeting necessary performance goals.

iv) Upgradeability

→ The hardware platform may be used over several product generations, or for several different versions of a product in the same generation, with few or no changes

→ However we want to be able to add features by changing software.

→ How can we design a machine that will provide the required performance for software that we haven't yet written?

v) Does it really work?

→ Reliability is very important in safety critical systems.

Few ways in which, embedd computing machines makes their design more difficult. (4)

i. Complex testing: We may have to run a real machine in order to generate the proper data.

ii. Limited observability and controllability:

- Embedded computing systems usually do not come with keyboards and screens.
- This makes it more difficult to see what is going on and to affect the system's operation.
- Users are forced to watch the values of electrical signals on the microprocessor bus, for example, to know what is going on inside the system.
- In real-time applications we may not be able to easily stop the system to see what is going on inside.

iii) Restricted development environments:

→ The development environments for embedded systems are often much more limited than those available for PCs and workstations.

→ Generally, code is compiled on PC, and then it is downloaded onto embedded system.

→ To debug the code, again user need to rely on programs that run on the PC or workstation and then look inside the embedded system.

1.2.6 Performance of embedded computing systems:

Performance in general purpose computing:

→ Most programmers have a fairly vague notion of performance — they want their program to run "fast enough" and they may be worried about the asymptotic complexity of their program.

→ Most general-purpose programmers use no tools that are designed to help them improve the performance of their programs.

Performance in embedded computing systems:

→ Embedded system designers, in contrast, have a very clear performance goal in mind — their program must meet its deadline.

→ At the heart of embedded computing is real-time computing, which is the science and art of programming to meet deadlines.

→ The program receives its input data; the deadline is the time at which a computation must be finished.

→ If the program doesn't produce the required output by the deadline, then the program does not work, even if the output that is eventually produces is functionally correct.

→ This notion of deadline-driven programming is at once simple and demanding.

→ It is not easy to determine whether a large, complex program running on a sophisticated

microprocessor will meet its deadline.

Understanding real-time performance:

→ In order to understand the real-time behavior of an embedded computing system, we have to analyze the system at several different levels of abstraction.

Bottom-up approach

→ CPU: The CPU clearly influences the behavior of the program, particularly when the CPU is a pipelined processor with cache.

→ Platform: The platform includes the bus and I/O devices. The platform components that surround the CPU are responsible for feeding the CPU and can dramatically affect its performance.

→ Program: Programs are very large and a CPU sees only a small window of the program at a time. We must consider the structure of the entire program to determine its overall behavior.

→ Task: We generally run several programs simultaneously on a CPU, creating a multitasking system.

→ Multiprocessor:

→ Many embedded systems have more than one processor. They may include multiple programmable CPUs as well as accelerators.

→ The interaction between these processors add yet more complexity to the analysis of overall system performance.

1.3 The embedded system design process:

→ The embedded system design process is aimed at two objectives

i) It will give us an introduction to the various steps in embedded system before we delve into them in more detail.

ii) It will allow us to consider the design methodology.

Design methodology is important for three reasons.

a) It allow us to keep track of design process to ensure that we have done everything we need to do, such as optimizing performance or performing functional tests.

b) It allows us to develop computer-aided design tools.

c) A design methodology makes it much easier for members of a design team to communicate.

→ By defining the overall process, team members can more easily understand what they are supposed to do, what they should receive from other team members at certain times, and what they are to hand off when they complete their assigned steps.

→ Because most embedded systems are designed by teams, coordination is perhaps the most important role of a well-defined design methodology.

→ Fig 1. summarizes the major steps in the embedded system design process. (10')

→ In this top-down view, we start with system requirements.

→ In the next step, specification, we create a detailed description of what we want.
Specification states only how the system behaves, not how it is built.

→ The details of the system's internals begin to take shape when we develop the architecture, which gives the system structure in terms of large components.

→ Once we know the components we need, we can design those components, including both software modules and any specialized hardware we need.

→ Based on those components, we can finally build a complete system.

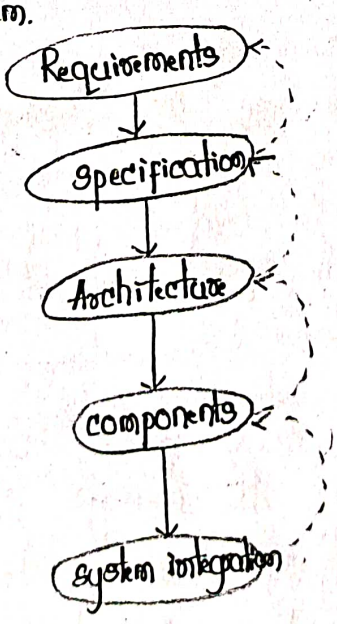


Figure 1. Major levels of abstraction in the design process.

→ In top-down approach we begin with the most abstract description of the system and conclude with concrete details.

→ In bottom-up approach we start with components to build a system

→ We need bottom-up design because we do not have perfect insight into how later stages of the design process will turn out.

→ Decisions at one stage of design are based upon estimates of what will happen later.

→ How fast can we make a particular function run? How much memory will we need? How much system bus capacity do we need?

→ Steps in the design process are only one axis along which we can view embedded system design.

→ The other major goals of the design are
- Manufacturing cost
- Performance (both overall speed and deadlines);
- Power consumption

→ We must also consider the tasks we need to perform at every step in the design process. At each step in the design, we add detail:
- Analyze the design at each step to determine how we can meet the specifications.

→ Refine the design to add detail.

→ Verify the design to ensure that it still meets all system goals such as cost, speed and so on

1.3.1 Requirements:

→ The initial stages of the design process capture this information for use in creating the architecture and components.

→ It is done in two phases.

i) First, we gather an informal description from the customers known as requirements

ii) Refine the requirements into a specification that contains enough information to begin designing the system architecture.

Requirements vs specifications:

→ Separating out requirements analysis and specification is necessary because of the large gap between what the customers can describe about the system they want and what the architects need to design the system.

→ Consumers of embedded systems are usually not themselves embedded system designers or even product designers.

→ Their understanding of the system is based on how they envision user's interaction with the system

→ They may have unrealistic expectations as to what can be done within their budgets, and they may also express their desires in a language very different from system architect's jargon.

→ Capturing a consistent set of requirements from the customer and then massaging those requirements into a more formal specification is a structured way to manage the process of translating from the consumer's language to the designer's.

→ Requirements may be functional or non functional

Typical nonfunctional requirements include

i) Performance: The speed of the system is often a major consideration both for the usability of the system and for its ultimate cost.

→ Performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.

ii) Cost:

→ The target cost or purchase price for the system is almost always a consideration.

→ Cost typically has two major components:
- Manufacturing cost: includes the cost of components and assembly.
- Nonrecurring engineering (NRE): costs include the personnel and other costs of designing the system

iii) Physical size and weight:

→ Physical size of the system vary greatly depending on the application

→ An industrial control system for an assembly line may be designed to fit into a standard-size rack with no strict limitations on weight.

→ A handheld device typically has tight requirements on both size and weight that can ripple through the entire system design

iv) Power-consumption:

→ is important in battery-powered systems and is often important in other applications as well.

→ Power can be specified in the requirements stage in terms of battery life

Validating requirements:

→ validating a set of requirements is ultimately a psychological task because it requires understanding both what people want and how they communicate those needs.

Simple requirements form:

→ Requirements analysis for big systems can be complex and time consuming.

→ However, capturing a relatively small amount of information in a clear, simple format is a good start toward understanding system requirements.

→ Fig. below shows a sample requirement form that can be filled at the start of the project.

→ This form is used as a checklist in considering the basic characteristics of the system.

→ Let's consider the entries in the form

Name GPS moving map

Purpose

Inputs

Outputs

Functions

Performance

Manufacturing cost

Power

Physical size and weight

Figure: Sample requirements form.

1. Name: This is simple but helpful. Giving a name to the project not only simplifies talking about it to other people but can also crystalize the purpose of the machine.

2. Purpose: This should be a brief one or two line description of what the system is supposed to do

→ If you cannot describe the essence of your system in one or two lines, chances are that you don't understand it well enough.

3. Inputs and Outputs:

→ These two entries are more complex than they seem. The inputs and outputs to the system encompass a lot of detail.

i) Types of data:

→ Analog electronic signals? Digital data? Mechanical inputs

ii) Data characteristics:

→ Periodically arriving data, such as digital audio samples? occasional user inputs? How many bits per data element?

iii) Types of I/O devices:

→ Buttons? Analog / digital converters? video display?

iv) Functions:

→ This is a more detailed description of what the system does.

→ A good way to approach this is to work from the inputs to the outputs

→ When the system receives an input, what does it do?

→ How do user interface inputs affect these functions?

→ How do different functions interact?

4. Performance:

→ It is essential that the performance requirements be identified early because they must be carefully measured during implementation to ensure that system works properly.

5. Manufacturing Cost:

→ This includes primarily the cost of the hardware components.

→ cost has a substantial influence on architecture.

6. Power:

→ Typically, the most important decision is whether the machine will be battery powered or plugged into the wall.

→ Battery-powered machines must be more careful about how they spend energy.

7. Physical size and weight:

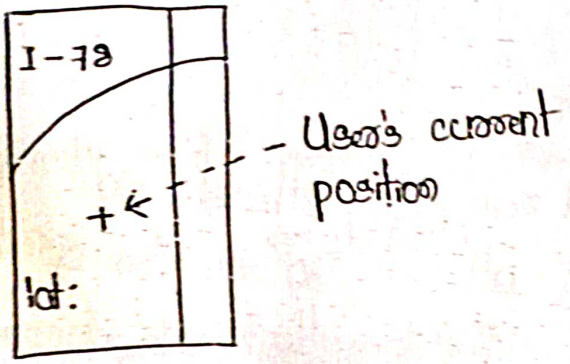
→ You should give some indication of the physical size of the system to help guide certain architectural decisions.

Example 1.1: Requirements Analysis of a GPS Moving Map.

→ The moving map is a handheld device that displays for the user a map of the terrain around the user's current position; the map display changes as the user and the map device change position.

→ The moving map obtains its position from the GPS, a satellite based navigation system.

→ The moving map is as shown in figure.



Functionality:

→ This system is designed for highway driving and similar uses. The system should show major roads and other landmarks available in standard topographic databases.

User interface:

→ The screen should have at least 400 x 600 pixel resolution.

→ The device should be controlled by no more than three buttons.

→ A menu system should pop up on the screen when buttons are pressed to allow the user to make selections to control the system.

Performance:

→ The map should scroll smoothly.

→ Upon power up a display should take no more than one second to appear, and the system should be able to verify its position and display the current map within 15 seconds.

Cost:

→ The selling cost of the unit should be no more than $ 320.

Physical size and Weight:

→ The device should fit comfortably in the palm of the

hand.

Power Consumption:

→ The device should run for atleast eight hours on four A4 batteries, with atleast 30 minutes of those eight hours comprising operation with the screen on.

Requirement chart for moving map system:

Name	GPS moving map
Purpose	Consumer-grade moving map for driving use
Inputs	Power button, two control buttons
Outputs	Back-lit LCD display 400×600
Functions	Uses 5-receiver GPS system; three user-selectable resolutions; always displays current latitude and longitude.
Performance	Updates screen with 0.25 seconds upon movement
Manufacturing cost	$40
Power	100mW
Physical size and weight	no more than 2" × 6", 12 ounces.

→ This chart adds some requirements in engineering terms by providing actual dimensions of the device.

3.2 Specification:

→ It is more precise – it serves as the contract between the customer and architects.

→ Specification must be carefully written so that it accurately reflects the customer's requirements and does so in a way that can be clearly followed during design.

→ The specification should be understandable enough so that someone can verify that it meets system requirements and overall expectations of the customer.

A specification of the GPS system would include several components:

→ data received from the GPS satellite constellation;

→ map data;

→ user interface;

→ operations that must be performed to satisfy customer requests;

→ background actions required to keep the system running, such as operating the GPS receiver.

3.3.3 Architecture design:

→ The spec specification does not say how the system does things, only what the system does.

→ Describing how the system implements those functions is the purpose of the architecture.

→ The architecture is a plan for the overall structure of the system that will be used later to design the

components that make up the architecture.

→ Figure below shows a sample system architecture in the form of a block diagram that shows major operations and data flows among them.

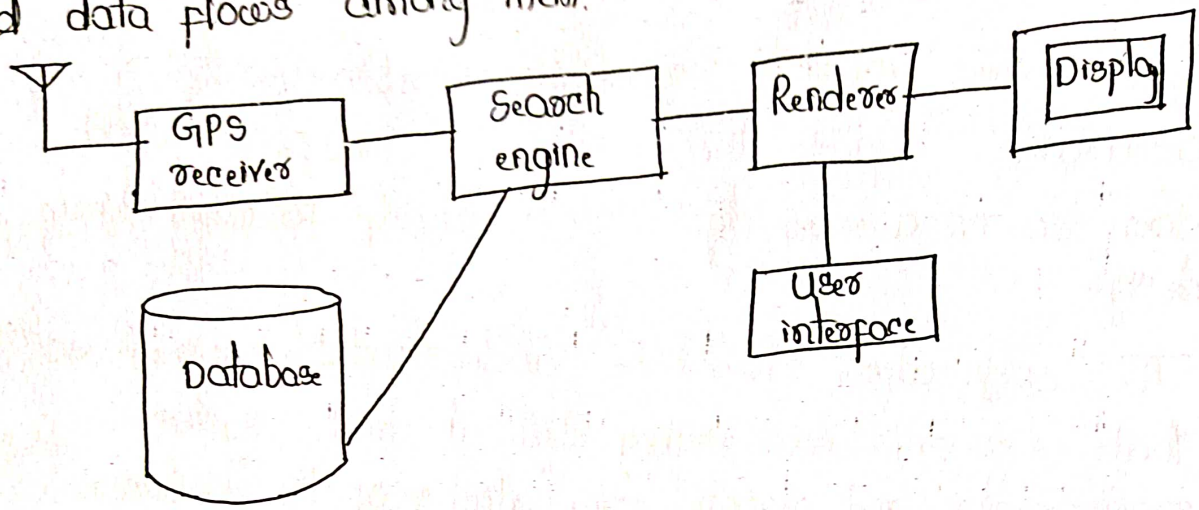


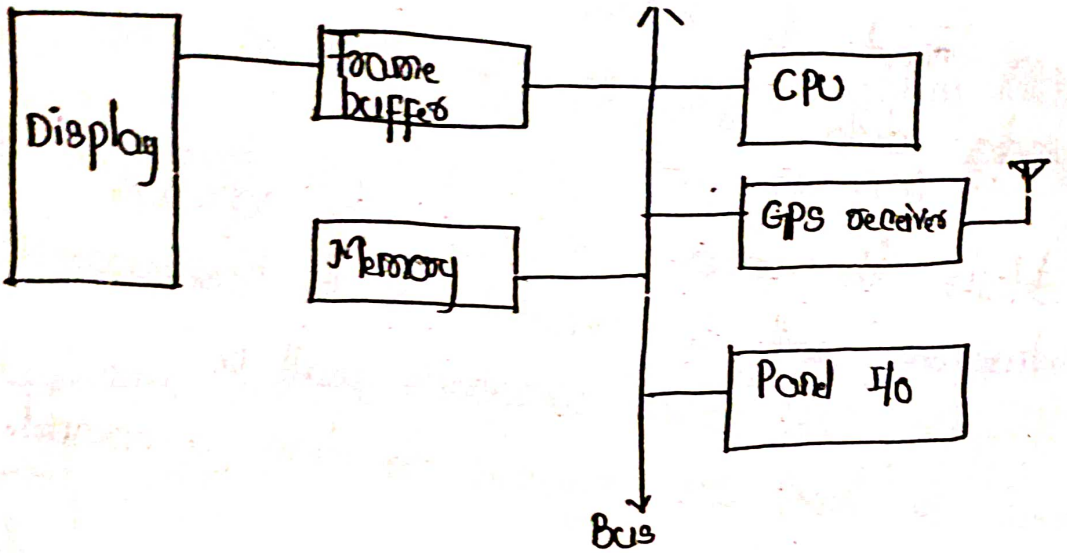
Figure: Block diagram for the moving map

→ This block diagram is still quite abstract. It does not specifies which operations will be performed by software running on a CPU, what special hardware does and so on.

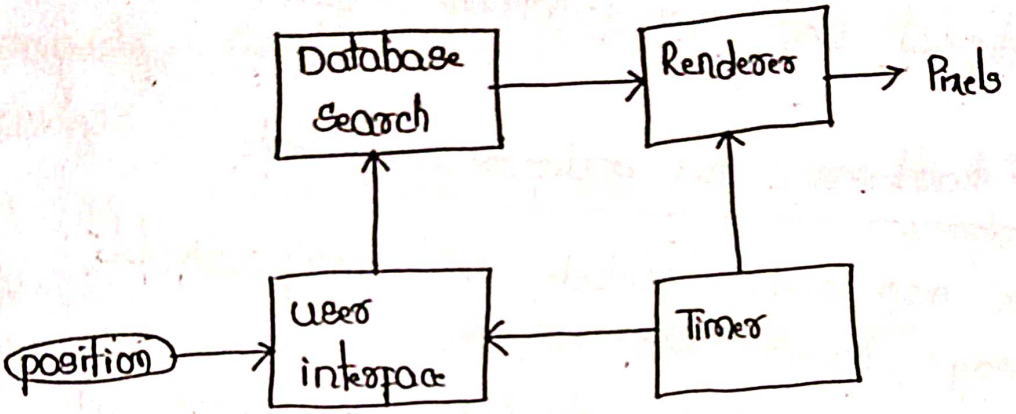
→ It just explains how to implement the functions described in the specification.

→ We see, we need to search the topographic database and to draw (render) the results for display.

→ These two functions are separated to do them in parallel – performing rendering separately from searching the database may help us update the screen more fluidly.



Hardware



Software

Figure: Hardware and software architectures for the moving map.

→ The hardware block diagram clearly shows that we have one central CPU surrounded by memory and I/O devices.

→ Two memories are used: Frame buffer for the pixels to be displayed and a separate program/data memory for general use by the CPU.

→ The software block diagram follows the system block diagram, but timer is added to control

when we read the buttons on the user interface and render data onto the screen.

→ More details like where units in the software block diagram will be executed in the hardware block diagram and when operations will be performed in time, are required to have a complete architectural description.

→ Architectural descriptions must be designed to satisfy both functional and non functional requirements.

1.3.4 Designing hardware and software components

→ Hardware components include field programmable gate arrays, boards and so on.

→ CPU is an example for standard component. Shared

→ Topographic data-base is an example for standard software module. This can be accessed by standard routines.

→ Using standard software for these access functions not only saves design time, also gives us a faster implementation for specialized functions such as the data decompression phase

→ For hardware components even if we are using only standard ICs, printed circuit board need to be designed to connect them.

1.3.5 System integration: only after the components are built, put them together to see a working system

→ Bugs are typically found during system integration and a good planning can help us find the bugs quickly.

→ By building up the system in phases and running properly chosen tests, bugs can be identified more easily.

→ By building up the system in phases and running properly chosen tests, we can often find bugs more easily.

→ If we debug only a few modules at a time, we are more likely to uncover the simple bugs and be able to easily recognize them.

→ only by fixing the simple bugs will we be able to uncover the more complex bugs?

→ We need to ensure during the architectural and component design phases that we make it as easy as possible to assemble the system in phases and test functions relatively independently.

1.3.6. Formalisms for system design:

→ Unified Modeling Language (UML) was designed to be useful at many levels of abstraction in the design process.