BECE403E-EMBEDDED SYSTEM DESIGN

MODULE-1

EMBEDDED SYSTEM PRODUCT DEVELOPMENT

Dr.S.Muthulakshmi
Asso. Professor, SENSE,
VIT Chennai

MODULE-1

Embedded System Product Development

- Characteristics of embedded systems, Classification of embedded systems, Embedded product development cycle, Embedded System Design Challenges, Performance and Benchmarking Tools.

EMBEDDED SYSTEM

WHAT IS EMBEDDED SYSTEM?

Embedded system = Hardware + Software

Designed for specific purpose

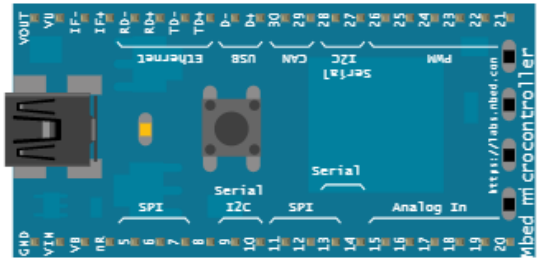
An embedded system is a combination of hardware and software, designed for a specific function or functions within a larger system.

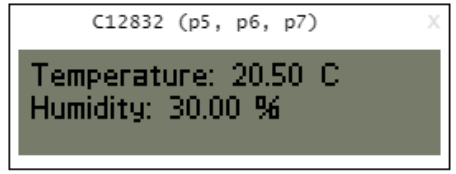
WHAT IS EMBEDDED SYSTEM?

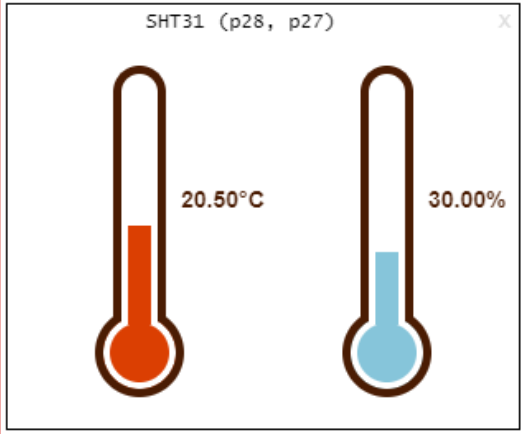
S/W

```
#include "mbed.h"
#include "C12832.h"
#include "Sht31.h"

C12832 lcd(SPI_MOSI, SPI_SCK, SPI_MISO, p8, p11);
Sht31 sht31(I2C_SDA, I2C_SCL);
DigitalOut led(LED1);

int main() {
    printf("Set the temperature above 25 degrees to trigger the warning LED\n");

    while (1) {
        lcd.cls();

        float temp = sht31.readTemperature();
        float humidity = sht31.readHumidity();

        lcd.locate(3, 3);
        lcd.printf("Temperature: %.2f C", temp);
        lcd.locate(3, 13);
        lcd.printf("Humidity: %.2f %%", humidity);

        // turn on LED if the temperature is above 25 degrees
        led = temp > 25.0f;

        wait(0.5f);
    }
}
```

H/W







Embedded System

EMBEDDED SYSTEM - EXAMPLES

Industrial Robots

GPS Receivers

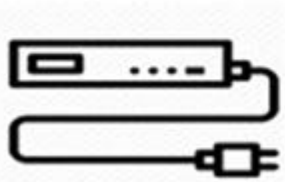
Digital Cameras

DVD Players

Wireless Routers

Embeeded System = *Computer Inside a Product*

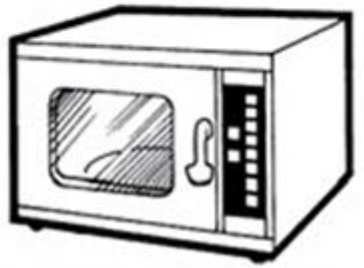
MP3 Players



Set top Boxes

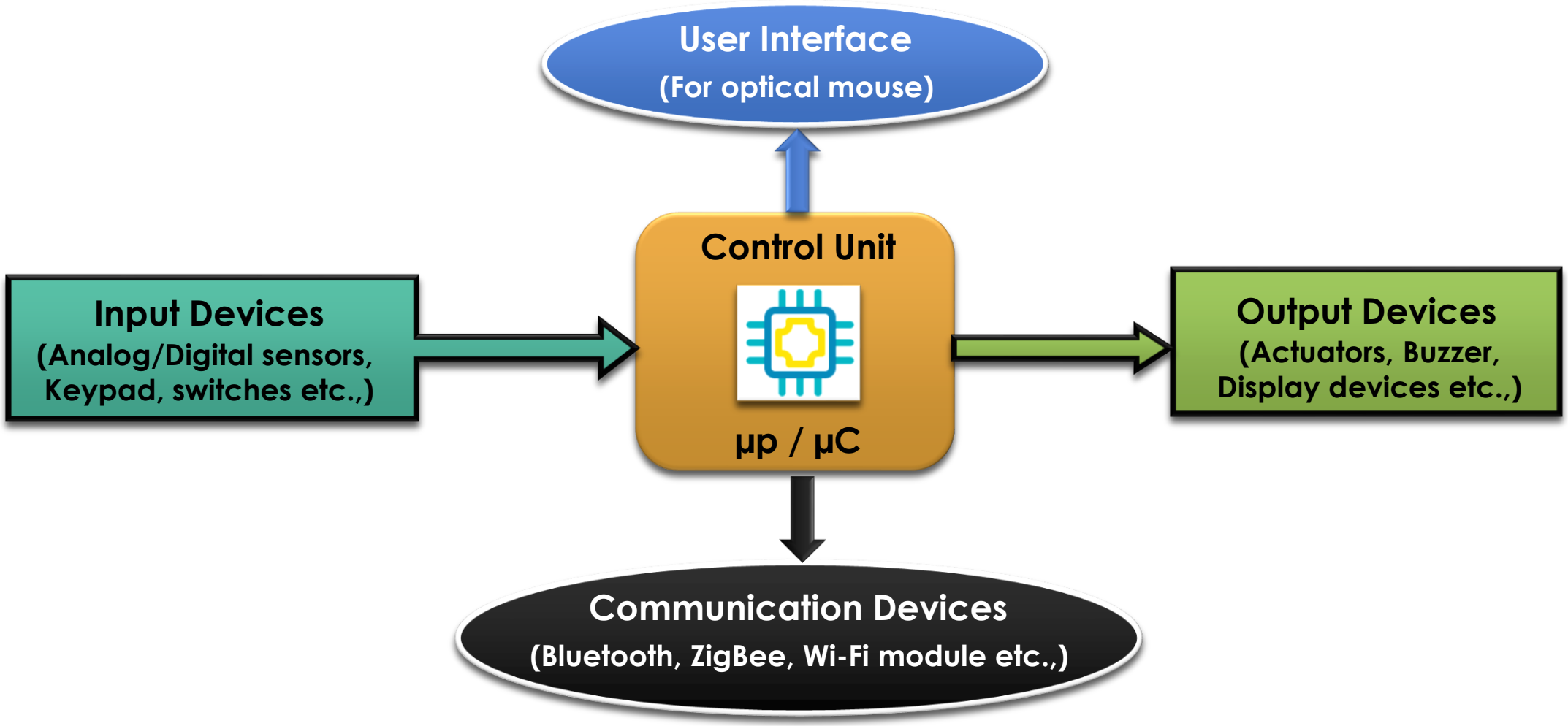
Gaming Consoles

Photocopiers



Microwave Ovens

EMBEDDED SYSTEM – BLOCK DIAGRAM



Embedded System Application Domain

Automotive

Home Appliances

Consumer Electronics

Industrial Automation

Healthcare

Military

Avionics

PURPOSE OF AN EMBEDDED SYSTEM

- Each embedded system is designed to serve the purpose of any one or combinations of following task.
  - Data collection / storage / representation
  - Data communication
  - Data (signal) processing
  - Monitoring
  - Controlling
  - Application specific user interface

CHARACTERISTICS OF EMBEDDED SYSTEM

CHARACTERISTICS OF EMBEDDED SYSTEM

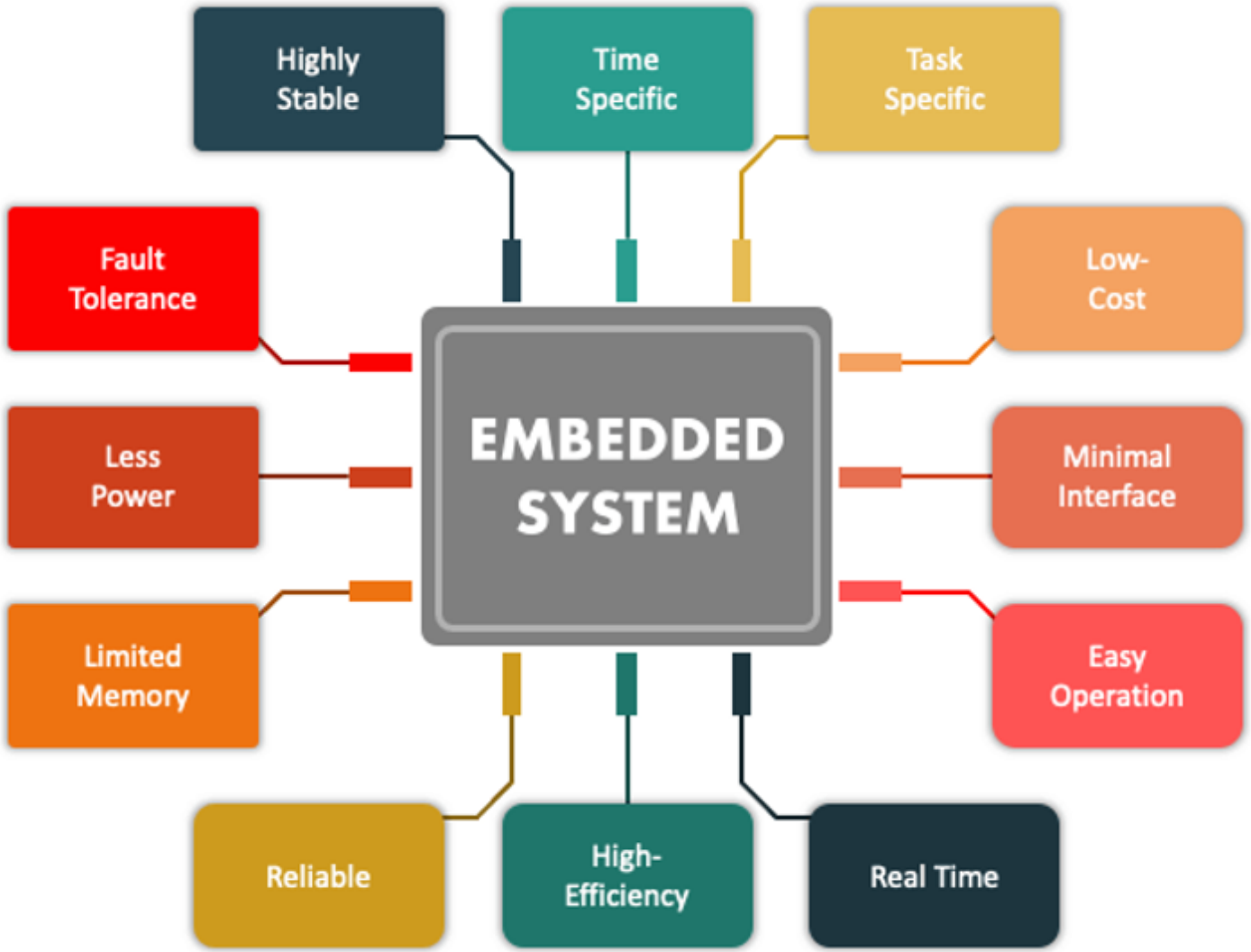


Image credit: sketchbubble, Ref. URL : https://www.sketchbubble.com/en/presentation-embedded-systems.html

CHARACTERISTICS OF EMBEDDED SYSTEM

1. **Task Specific:** Embedded Systems (ES) are optimized for the particular task they are intended to perform, which makes them more efficient and reliable
2. **Low Cost:** ES are typically designed to be cost-effective because they are often used in large volumes, and the cost per unit must be low to make the product economically viable
3. **Time Specific:** ES must operate within a specific time frame because applications such as industrial control systems, where timing is critical for safety and efficiency
4. **Low Power:** ES are designed to operate with minimal power consumption since the system needs to operate for extended periods on battery power
5. **Minimal User Interface:** Many ES do not require a complex user interface since they are often designed to operate autonomously or with minimal user intervention
6. **High Efficiency:** ES should be highly efficient in terms of processing power, memory usage, and energy consumption to perform their task with maximum efficiency & reliability

CHARACTERISTICS OF EMBEDDED SYSTEM

7. **Highly Reliability:** ES are in applications where failure is not an option, such as in medical devices or aviation where downtime can be costly or dangerous.
8. **High Stable:** ES are expected to operate consistently over long periods under extreme environmental conditions
9. **Real Time:** ES are designed to process data and respond to inputs instantly to ensures timely and accurate reactions to external events in time-critical applications
10. **Limited Memory:** ES utilize limited amount of memory tailored to their specific tasks, also it must have balanced between ROM for storing firmware and RAM for runtime operations
11. **Easy Operation:** ES are expected to have a user-friendly interface making them accessible to all users, even without any technical expertise.
12. **Fault Tolerance:** ES is designed to perform it's functionality despite faults, which can be caused by hardware failures, software errors, and environmental conditions.

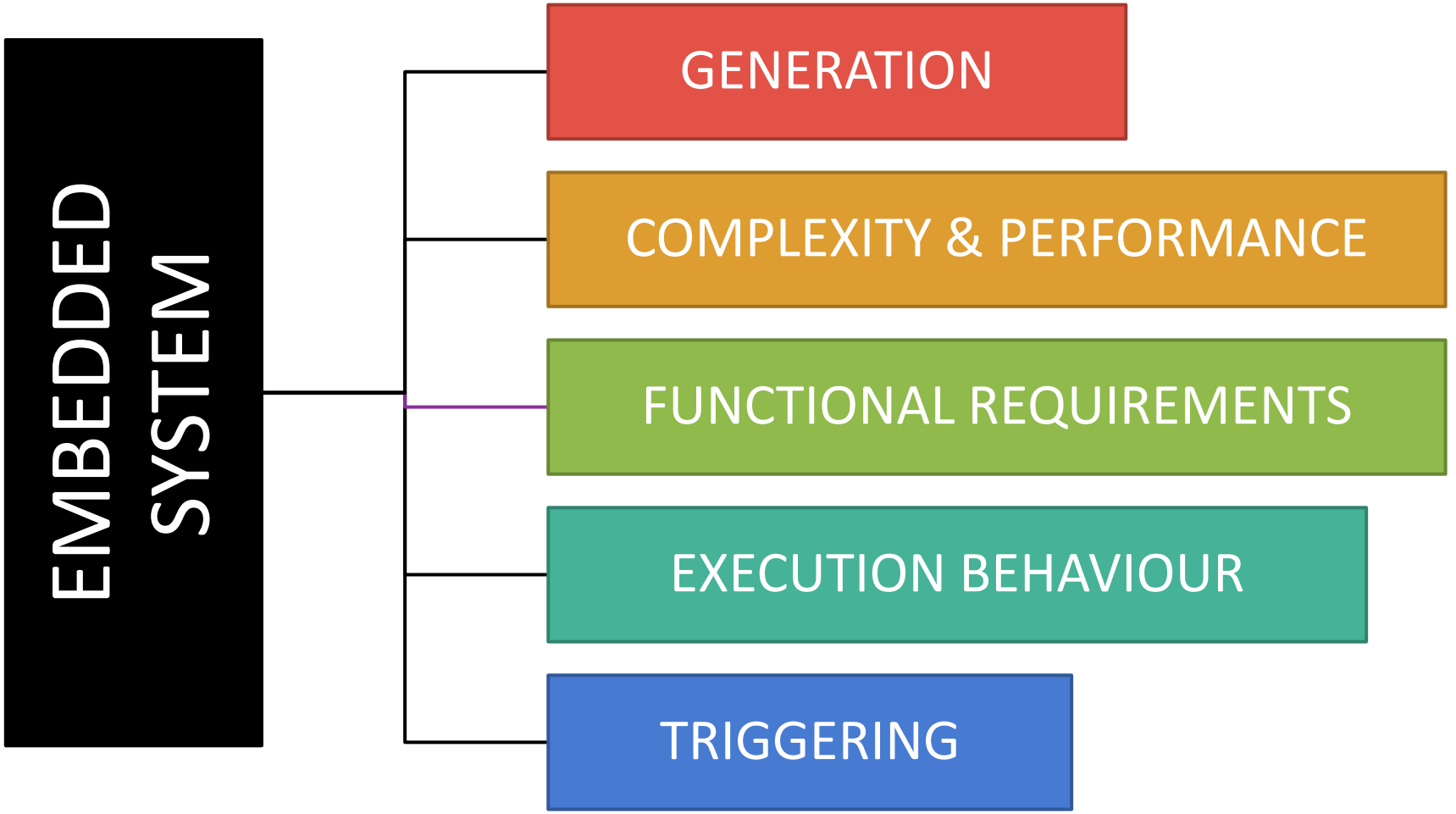
CHARACTERISTICS OF EMBEDDED SYSTEM

GENERAL PURPOSE vs EMBEDDED SYSTEM

S. NO.	GENERAL PURPOSE COMPUTING SYSTEM	EMBEDDED SYSTEM
1	A system which is a combination of generic hardware and general purpose operating system for executing verity of applications	A system which is a combination of special purpose hardware and embedded OS for executing a specific set of application
2	Contains General Purpose Operating System (GPOS)	May or may not contain operating system for function
3	Applications can be modified by user	The firmware of the embedded system is pre-programmed and it is non-modifiable by the end user
4	Performance is a key deciding factor in the selection of the system. Always “Faster is better”.	Application specific requirements (like performance, power, memory, cost) are key deciding factor
5	Need not to be deterministic in execution behavior	Execution behavior is deterministic for certain type of system like “hard real time systems” (Missile guidance or airbag system)
6	Personal computer, Laptops	DVD player, Digital camera

CLASSIFICATION OF EMBEDDED SYSTEM

CLASSIFICATION OF EMBEDDED SYSTEM



CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON GENERATION

- **First Generation:**
  - build around 4-bit and 8-bit processor like 8085, Z80
  - Simple hardware & firmware developed in assembly code
  - Ex: Digital telephone keypads, calculator etc.,
- **Second Generation:**
  - build around 8-bit and 16-bit processor
  - complex and powerful instruction set
  - Ex: Data acquisition system, SCADA system etc.,

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON GENERATION

- **Third Generation:**
  - build around 16-bit and 32-bit processor, DSP & ASIC were introduced
  - complex and powerful instruction set with pipelining
  - dedicated embedded OS entered into market
  - Ex: robotics, industrial process control, networking etc.,
- **Fourth Generation:**
  - advent of SOC, reconfigurable & multicore processors
  - high performance, tight integration and miniaturization µP
  - High performance RTOS for functioning
  - Ex: smart phone, PDA etc.,

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON COMPLEXITY AND SYSTEM PERFORMANCE

- **Small Scale Systems:**
  - Entry-level embedded system with no design complexity involves
  - Work with 8-bit (8051) or 16-bit (80196) microcontrollers
  - Most of these systems are battery operated
  - Hardware and software complexity is very low due to the small size of µC
  - Easy to program using assembly language or C-programing language
  - Requires small memory as it deals with a small amount of data
  - Ex: robotic arm, electronic toys, automatic vending machines, thermometer etc.,

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON COMPLEXITY AND SYSTEM PERFORMANCE

- **Medium Scale Systems:**
  - System is designed using 16-bit or 32-bit microcontrollers
  - Offers better speed than small scale Embedded system
  - Hardware and software complexity is present
  - Run through Microcontrollers and digital signal processors
  - Requires comparatively more memory power than small scale ones
  - Along with microcontrollers, you need application-specific operating systems
  - Ex: Routers, ATMs, music systems etc.,

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON COMPLEXITY AND SYSTEM PERFORMANCE

- **Large Scale Systems:**
  - Make use of 32-bit or 64-bit microcontroller and multi-core processors
  - Requires very high memory power
  - Power consumption is the highest among rest types of embedded systems
  - Hardware and software complexities are enormous
  - Speed is a major concern
  - Some applications, like satellite systems, also involve (RTOS)
  - Ex: Missile navigation system, MRI scanner, Airbag System etc.,

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON FUNCTIONAL REQUIREMENTS

- **Real-Time Systems:**
  - Designed to perform a specific task within a pre-determined time
  - Offer quick responses under critical situations.
  - Further divided into two types : Soft and Hard real-time embedded system
  - Ex: Microwave oven, Traffic light control.
- **Stand-Alone Systems:**
  - Less complex
  - Independent of any system
  - Process digital/analog input signals and provide the output
  - Ex: Calculator, MP3 player

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON FUNCTIONAL REQUIREMENTS

- **Network Systems:**
  - Connected to a network and can communicate with other devices
  - Operate through a network interface
  - Communication to network happens through wired or wireless protocols
  - Ex: ATM machines, Home automation, weather monitoring systems
- **Mobile Systems:**
  - Small, portable and easy-to-carry
  - Work on restricted memory space
  - Optimized for low power consumption and high performance
  - Ex: Mobile phones, Digital camera, Smart watch

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON EXECUTION BEHAVIOUR

- Based on deterministic behaviour, ES is classified into Real-time and Non Real-time system

- **Real-time Systems:**
  - All deadlines are expected to meet without compromising safety or performance, regardless of any external disturbances like events from outside sources or changing environmental conditions.
  - Ex: Aircraft controller, Traffic light controller, Airbag system etc.,

- **Non Real-time Systems:**
  - There is no constraints to meet the deadline on-time.
  - Ex: MP3 player, Calculator etc.,

CLASSIFICATION OF EMBEDDED SYSTEM

BASED ON TRIGEERING

- **Event-Triggered Systems:**
  - Operate based on certain events within its operational environment as triggers
  - Events can be user commands or data transfers or activities related to the system
  - Need to awake all time to monitor for event action operation
  - Ex: Airbag system, Fire alarm system, Home automation etc.,
- **Time-Triggered Systems:**
  - Executes all activities based on predetermined schedules assigned at initialization.
  - Execute their tasks as per the assigned time frame regardless of any new input
  - Does not respond immediately when an event occurs
  - Ex: Data aquisation system, Weather monitoring station etc.,

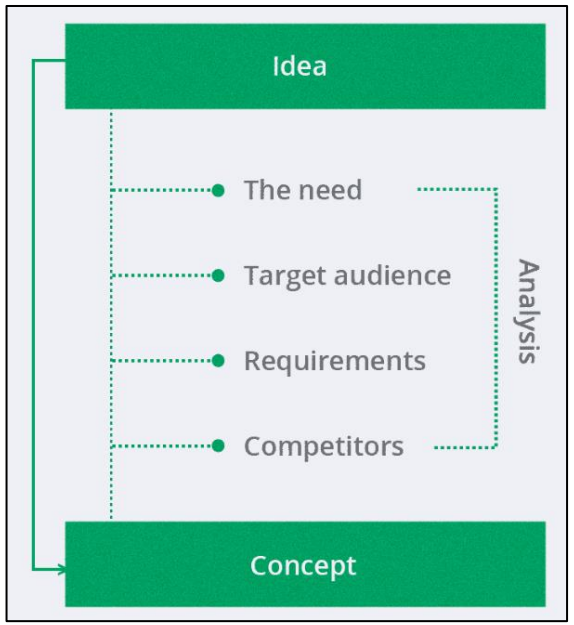
EMBEDDED PRODUCT DEVELOPMENT CYCLE

EMBEDDED PRODUCT DEVELOPMENT CYCLE

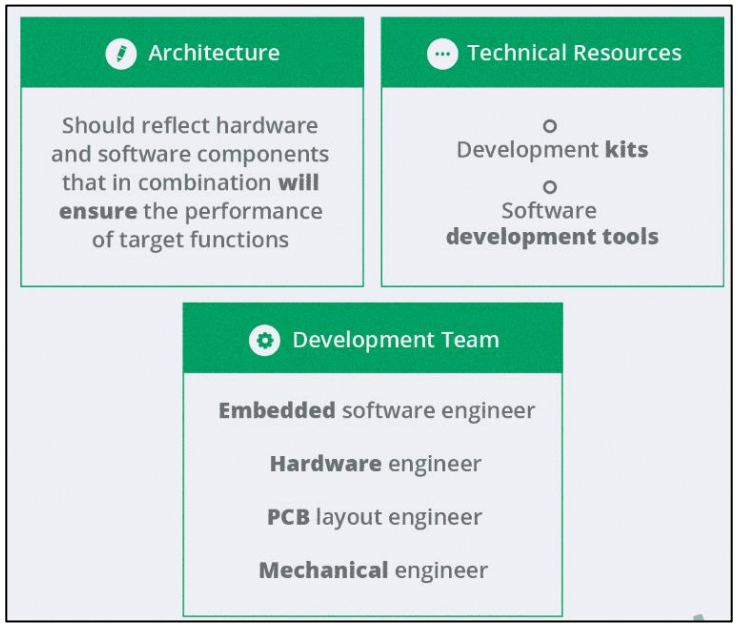
- Embedded systems play a crucial role in the proper functioning and easy access to modern devices and solutions.
- For building and developing a successful embedded product, you need to follow a well-defined Embedded Development Life Cycle (EDLC).
- EDLC is a step-by-step process that ensures high-quality products for end-users, defects prevention during development phase, and maximized productivity by identifying the
  - Cost-effective software and hardware solutions.
  - Risks involved in the development.
  - Shortest production time.
  - Methods to help with reducing timelines.
  - Approach to be taken for development.

EMBEDDED PRODUCT DEVELOPMENT CYCLE

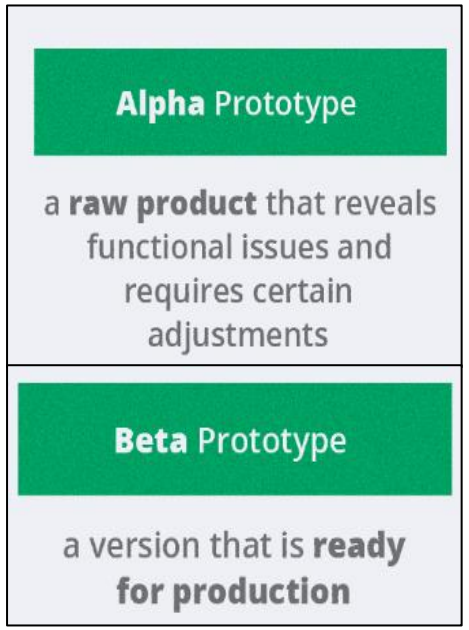
- EDLC approach comprises broadly FOUR stages and each stage is divided into sub-stages to break down the work into minor phases and milestones.



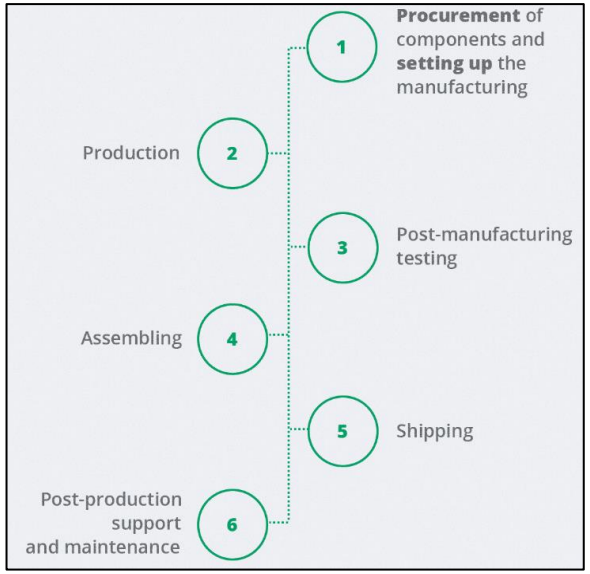
1. Planning and Analysis



2. Designing



3. Implementation



4. Deployment

EMBEDDED PRODUCT DEVELOPMENT CYCLE

STAGE-1: PLANNING AND ANALYSIS

- The first step of EDLC is to clearly define your business idea and transform it into a feasible concept and this phase can be break down into four deliverables to make it easier for the project – (i). Need (ii) Target Audience (iii) Requirements (iv) Competitors

(i) Need: You should realize the essence of the need behind your idea and the need may come from an individual, public, or company. Need can be visualized in any one of 3 types:

1) New/custom product development: the need for a product that does not exist in the market or will act as a competitor to an existing product.

2) Product re-engineering: the need to reengineer a product already available by adding new features or functionality.

3) Product maintenance: the need to launch a new version of a product followed by a failure due to non-functioning or to provide technical support for an existing product.

EMBEDDED PRODUCT DEVELOPMENT CYCLE

STAGE-1: PLANNING AND ANALYSIS

(ii) Target Audience: A crucial yet often overlooked component of the embedded product development process is to identify and define the target market of the product.

- Who will be the end-user of this product?
- What are the end user's demographics such as gender, age, profession, etc.?
- When is the product used by the end-user? And how often?
- What benefits users will get from your product

(iii) Competitors: It is essential to study the competitors and identify the gaps in the competitor's system and what you can do to improve it. you should have:

- A cost-benefit analysis for the project
- A thorough analysis of the product attributes and functions
- Product scope and feasibility

(iv) Requirements: Take the data you've collected during the research of your target audience, and use it to define the purpose of your product, its functional model, and the necessary hardware and software components.

EMBEDDED PRODUCT DEVELOPMENT CYCLE

STAGE-2: DESIGNING

- At this stage, you should choose a development approach that will allow you to implement your idea within budget.

(i) Architecture:

- The design process begins with the creation of the product's architecture, which should be based on the requirements collected in the previous (planning and analysis) step.
- In architecture, you should reflect software and hardware components that in combination will ensure the performance of target functions.

(ii) Development Team: At a minimum, an embedded product development team will need one each of the following:

- Embedded software engineer
- Hardware engineer
- Mechanical engineer
- PCB layout engineer

Depending on the project complexity and budget, you can decide on number of engineers.

EMBEDDED PRODUCT DEVELOPMENT CYCLE

STAGE-2: DESIGNING

(iii) Technical Resources: Based on the architecture, you need to choose technologies and tools to develop a proof-of-concept — a small model to prove the feasibility of your idea.

- Software Development Tools:
  - Operating systems (Linux, RTOS)
  - Languages (C, C++, Python, JavaScript, etc.)
  - IDEs, SDKs (Keil, PyCharm, Qt Creator), compilers, debuggers, and more.

- Development Kits: To quickly build embedded prototypes, you can use development kits or boards — out-of-the-box hardware platforms that can have integrated software, such as:
  - Processor modules
  - Microcontroller kits
  - Breakout boards.

- It is important to pay attention to several features when deciding on a development board for embedded processor, including the available peripherals, connectors, other communication peripherals, and onboard sensors, as well as the mechanical form factor and mounting style in a prototype enclosure.

EMBEDDED PRODUCT DEVELOPMENT CYCLE

STAGE-3: IMPLEMENTATION

- At this phase, you create a prototype — a working model of embedded product that can be tested and also improving the product quality and preparing it for production.

- Alpha Prototype:
  - When you fully integrate your device's hardware components on a PCB (printed circuit board), you get an alpha prototype.
  - This is a raw product that reveals functional issues and requires certain adjustments.

- Beta Prototype:
  - By addressing issues and extending the product by new features, you get a beta prototype — a version that is ready for production.
  - End users can test this version and provide feedback to help enhance the quality.
  - Moreover, you can apply a range of other tests to ensure your solution operates as expected.

- At this point of the EDLC, software engineers should check that the application meets regulatory standards, and ensure its security, scalability and maintainability.

EMBEDDED PRODUCT DEVELOPMENT CYCLE

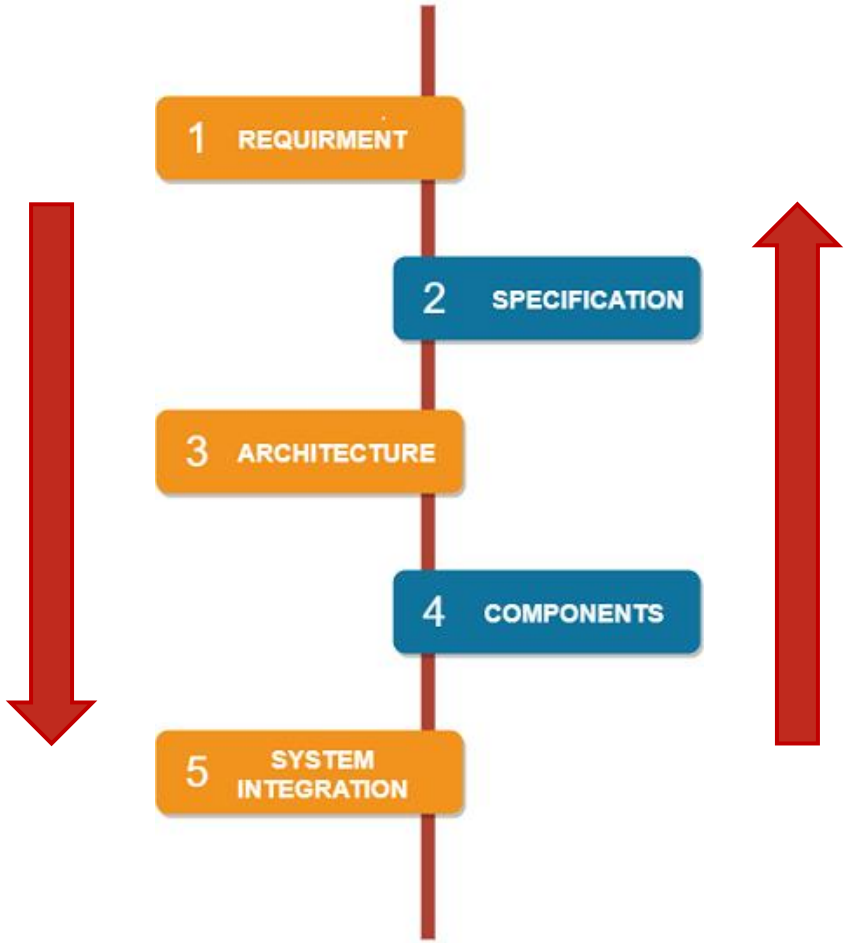
STAGE-4: DEPLOYMENT (PRODUCT LAUNCH)

- At this state, you have a real-life product that can be launched to mass production.
- You can't start the production immediately since it may take some time to procure the spare parts and set up the manufacturing process, up to three months.
- Perform post-manufacturing testing to reveal any defects in the production process and prevent defective products from going to customers.
- Once produced boards have been tried and tested, you can assemble them into their cover and deliver them to users.
- Post-production support and maintenance is a necessary activity during the embedded product development life cycle.

EMBEDDED SYSTEM DESIGN PROCESS

EMBEDDED SYSTEM DESIGN PROCESS

Top–down - From description of the system to concrete details.



bottom–up - Start with components to build a system.

Ref. "Computers as components: principles of embedded computing system design" by Wayne Wolf

EMBEDDED SYSTEM DESIGN PROCESS

- The major goals of the design to be considered are :
  - Manufacturing cost
  - Performance
  - Power consumption

- Tasks which need to be performed at each step,
  - ✓ Analyze the design at each step - meet specification
  - ✓ Refine the design - add details
  - ✓ Verify the design - system goals

EMBEDDED SYSTEM DESIGN PROCESS

1. REQUIREMENTS

- Informal descriptions gathered from the customer are known as requirements which they are refined into a specification to begin the designing of the system architecture.
  - Functional requirements: Needed output as a function of input
  - Non-functional requirements: performance, cost, physical size, weight, power consumption

REQUIREMENTS – EXAMPLE (GPS Navigation System)

Product Name	GPS Navigation System
Purpose	Consumer grade moving map for driving use
Inputs	Power button two control button
Outputs	Back-lit LCD Display 400x600
Functions	Uses 5-receiver GPS system; three user-selectable resolutions; always displays current latitude and longitude
Performance	Updates screen within 0.25 seconds upon movement
Power	100mW
Cost	30 USD
Size And Weight	2" x 6" and 250 grams

EMBEDDED SYSTEM DESIGN PROCESS

2. SPECIFICATION

- Requirements gathered is refined into a specification.
- Specification serves as the contract between the customers and the architects.
- Specification is essential to create working systems with a minimum of designer effort.
- Specific, understandable and accurately reflect the customer's requirements.
- SPECIFICATION - EXAMPLE (GPS Navigation System)
  - Data received from the GPS satellite
  - Map data
  - User interface
  - Background actions
  - Operations that must be performed to satisfy customer requests

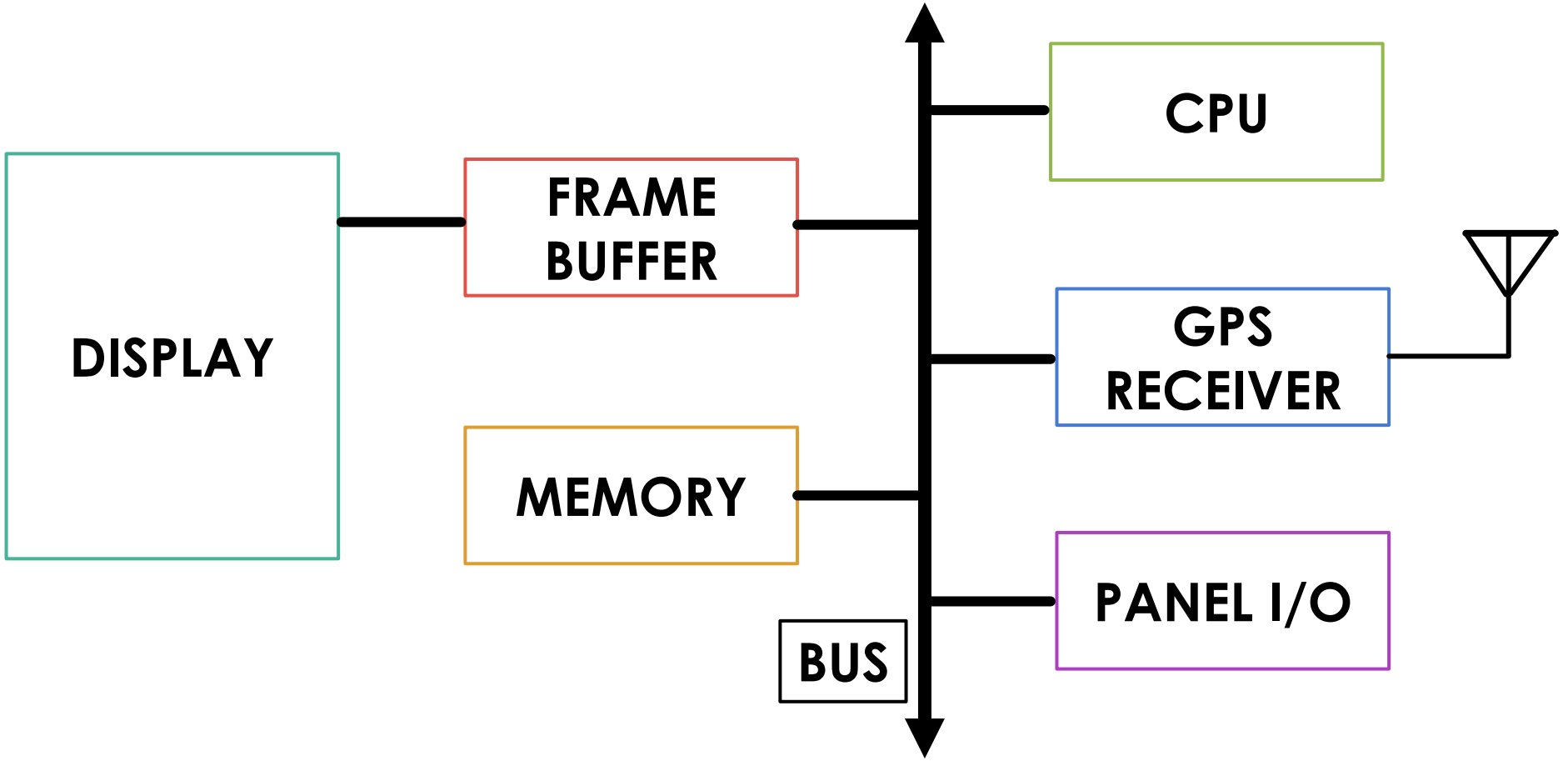
EMBEDDED SYSTEM DESIGN PROCESS

3. ARCHITECTURE

- The specification describes only the functions of the system
- Implementation of the system is described by the Architecture
- The architecture is a plan for the overall structure of the system
- It will be used later to design the components
- The system block diagram may be refined into two block diagrams - hardware and software

EMBEDDED SYSTEM DESIGN PROCESS

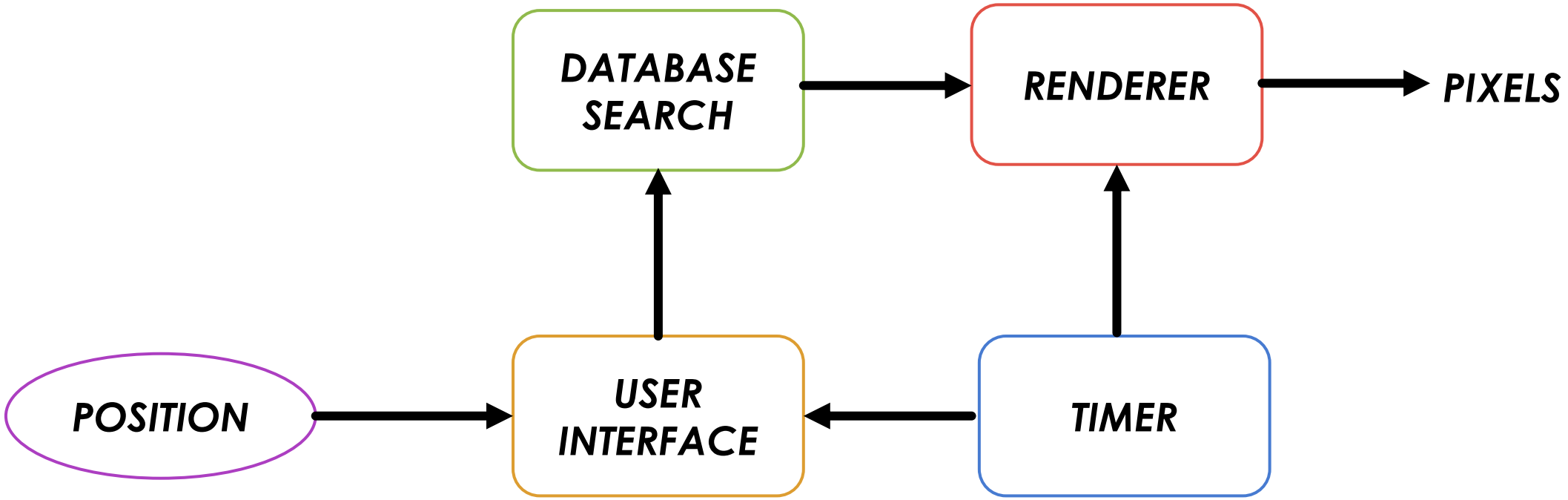
ARCHITECTURE - EXAMPLE (GPS Navigation System)



SYSTEM DESIGN - HARDWARE

EMBEDDED SYSTEM DESIGN PROCESS

ARCHITECTURE - EXAMPLE (GPS Navigation System)



SYSTEM DESIGN - SOFTWARE

EMBEDDED SYSTEM DESIGN PROCESS

4. HARDWARE AND SOFTWARE COMPONENTS

- Architectural description specify what components we need both hardware and software
- The component design effort builds those components in conformance to the architecture and specification
- Some of the components will be ready-made (example :CPU, memory chips)
- HARDWARE AND SOFTWARE COMPONENTS - EXAMPLE
  - In the moving map, GPS receiver is a predesigned standard hardware component.
  - Topographic software is a standard software module which uses standard routines to access the database.
  - Printed circuit board are the components which needs to be designed.
  - When creating these embedded software modules, ensure the system runs properly in real time and that it does not take up more memory space than allowed.

EMBEDDED SYSTEM DESIGN PROCESS

5. SYSTEM INTEGRATION

- After the components are built, they are integrated
- Bugs are typically found during the system integration
- By debugging a few modules at a time, simple bugs can be discovered
- By fixing the simple bugs early, more complex bugs can be discovered
- Appropriate debugging facilities during design which can help to ease system integration problems.

EMBEDDED SYSTEM DESIGN PROCESS - EXAMPLE

AUTOMATIC CHOCOLATE VENDING MACHINE

1. REQUIREMENTS

- **Purpose:**
  - To sell chocolate through an ACVM from which children can automatically purchase
  - The payment is by inserting the coins into a coin-slot.
- **Inputs:**
  - Coins of different denominations through a coin slot.
  - User commands.
- **Outputs:**
  - Chocolate and signal to the system that subtracts the cost from the value of amount collected.
  - Display of the menus for GUIs, time and date, advertisements, welcome and thank messages.

AUTOMATIC CHOCOLATE VENDING MACHINE

1. REQUIREMENTS

- Functions of the system:
  - A child sends commands to the system using a GUI. It must consists of the LCD display and keypad units.
  - The child inserts the coins for cost of chocolate and the machine delivers the chocolate.
  - If the coins are not inserted as per the cost of chocolate in reasonable times then all coins are refunded. If the coin amount more, the excess amount is refunded along with chocolate.
  - The coins for the chocolates purchased collect inside the machine in a collector channel, so that owner can get the money, again through appropriate commands using the GUI.
  - USB wireless modem enables communication through Internet to the ACVM owner.

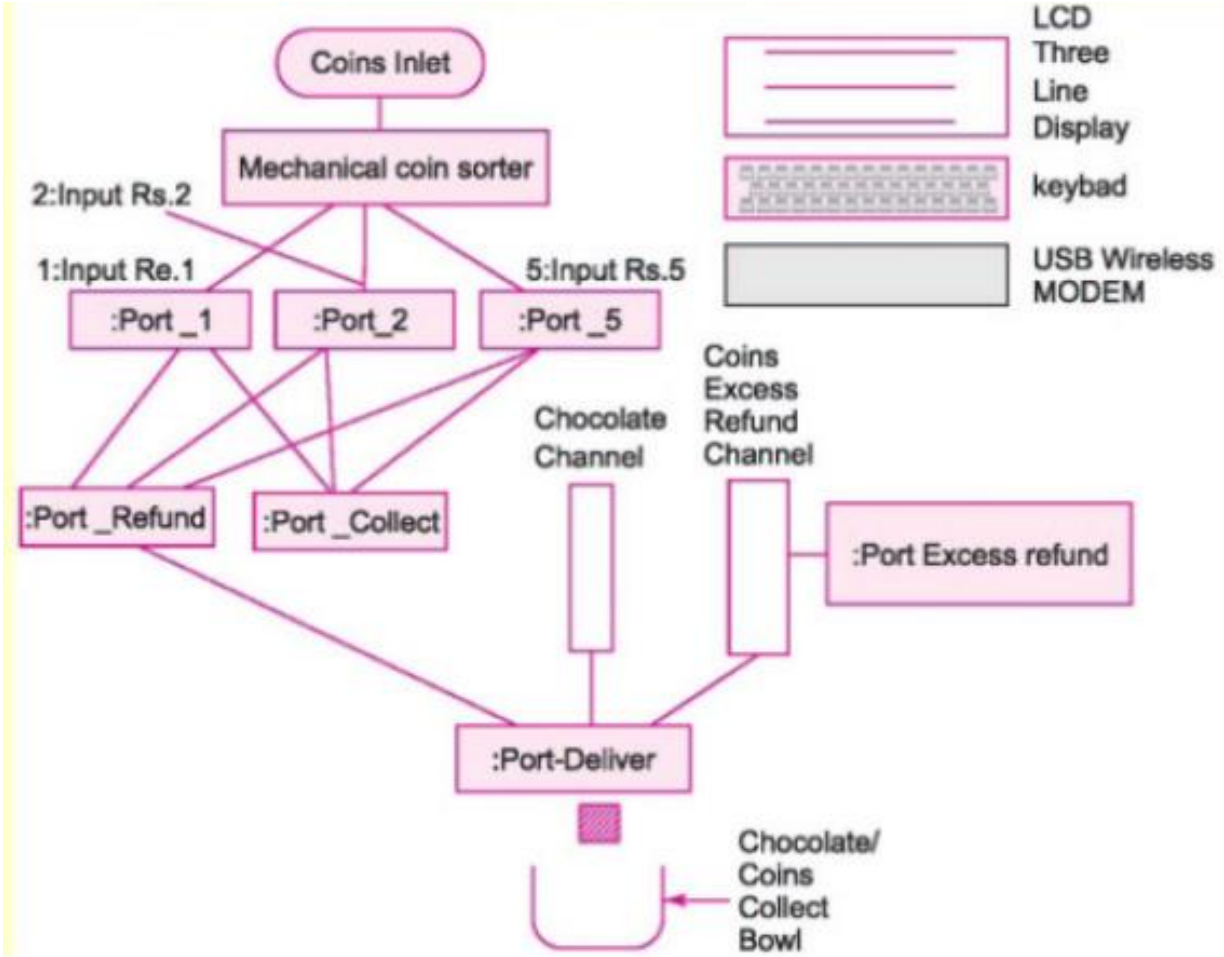
AUTOMATIC CHOCOLATE VENDING MACHINE

2. SPECIFICATIONS

- Alphanumeric keypad on the top of the machine. A child interaction with it when buying a chocolate. Owner commands and interaction with the machine.
- Four line LCD display unit on the top of the machine to display menus, entered text, pictograms, and welcome, thank messages, and time and date. Child as well as the ACVM owner GUIs with the machine using keypad and display.
- Coin insertion slot so that the child can insert the coins to buy a chocolate.
- Delivery slot to collect the chocolate, and coins if refunded.
- Internet connection port so that owner can interact with ACVM from remote.

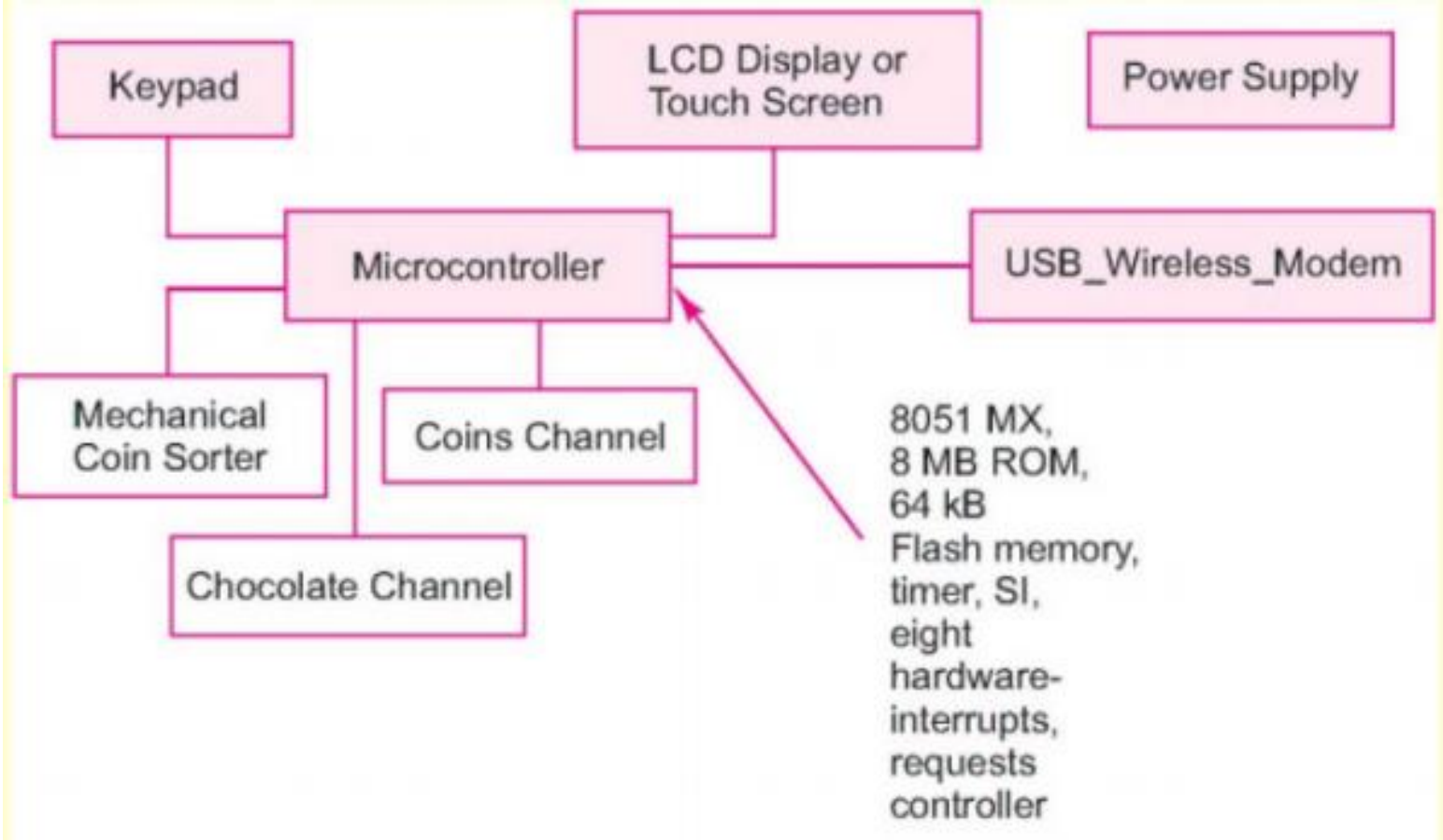
AUTOMATIC CHOCOLATE VENDING MACHINE

3. ARCHITECTURE – OVERALL SYSTEM DIAGRAM



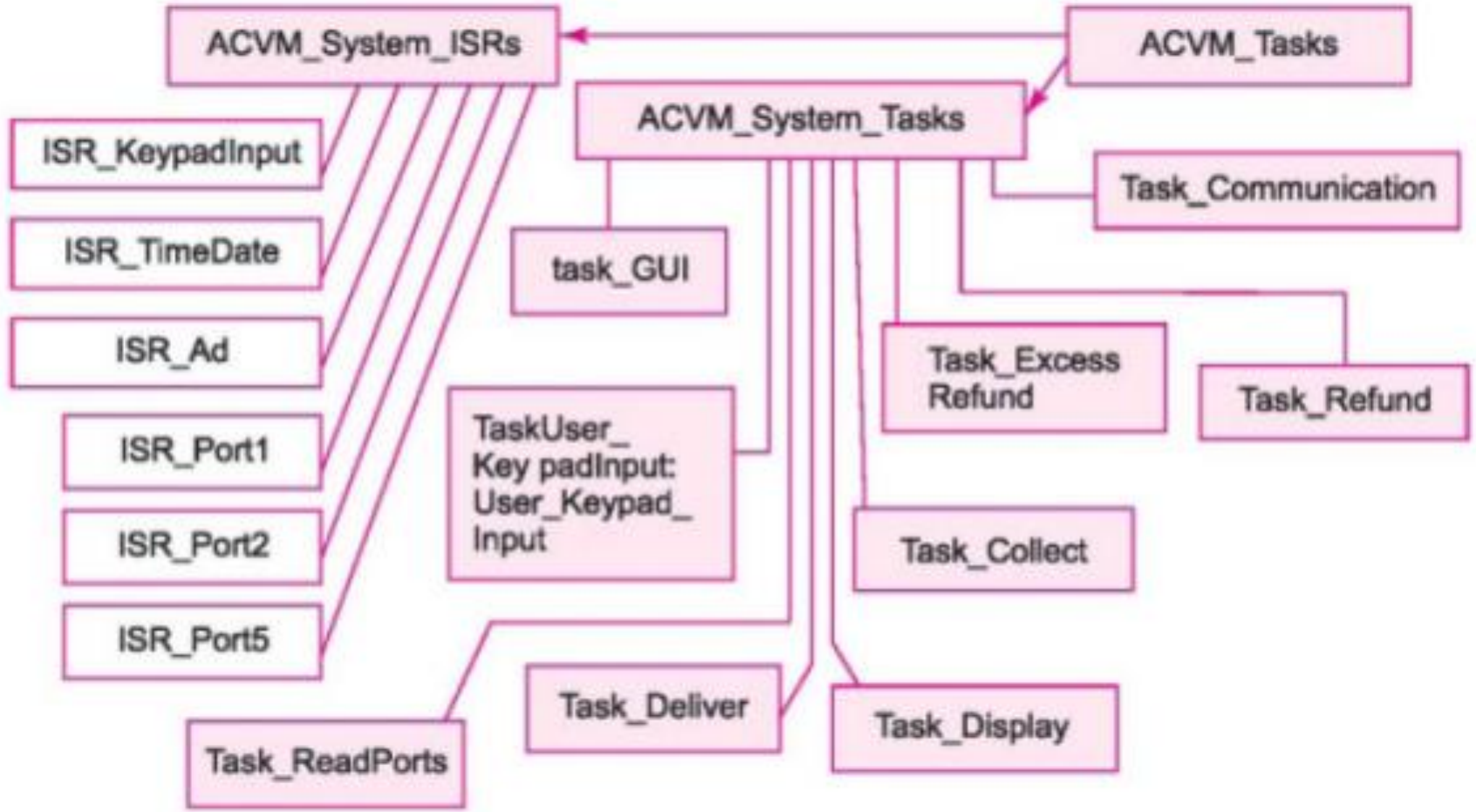
AUTOMATIC CHOCOLATE VENDING MACHINE

3. ARCHITECTURE – HARDWARE DIAGRAM



AUTOMATIC CHOCOLATE VENDING MACHINE

3. ARCHITECTURE – SOFTWARE DIAGRAM



AUTOMATIC CHOCOLATE VENDING MACHINE

4. COMPONENTS - HARDWARE

- ACVM specific hardware to sort the coins of different denomination.
- Main Power supply 220V 50 Hz or 110V 60 Hz. Internal circuits drive by supply of 5V 50mA for electronic and 12V 2A for mechanical systems.
- A TCP/IP port.
- A 1s resolution timer is obtained by programming 8051 timer.
- Flash memory part of ROM and RAM for storing temporary variables and stack.
- Microcontroller 8051 MX and 8 MB ROM for application codes.

AUTOMATIC CHOCOLATE VENDING MACHINE

4. COMPONENTS - SOFTWARE

- Task_ReadPorts: Waits for the coins and action as per coins collected.
- Task_Collect: Waits for coins = or > cost till timeout and act accordingly.
- Task_Deliver: Waits for Task_Readports, delivers chocolate, and decreases coins' amount after delivery.
- Task_Refund: Waits for refund event and calculate refund amount.
- Task_Excess Refund: Refunds the Excess amount
- Task_Display: Waits for the message and display as per message.

AUTOMATIC CHOCOLATE VENDING MACHINE

5. SYSTEM INTEGRATION

- Test and validation conditions:
  - All user commands must function correctly.
  - All graphic displays and menus should appear as per the program.
  - Each task should be tested with test inputs.
  - Tested for 60 users per hour.

EMBEDDED SYSTEM DESIGN CHALLENGES

EMBEDDED SYSTEM DESIGN CHALLENGES

- **Resource Constraints:**
  - Limited Memory: ES often have tight constraints on available memory therefore developers must optimize code and data storage to fit within these limitations.
  - Restricted Processing Power: Many ES operate with limited processing power hence designing efficient algorithms and optimizing code execution are crucial.
  - Power Consumption: Many ES are battery-powered, it will be very difficult to ensure they operate at low power to extend battery life and reduce heat generation.

- **Security Concerns:** As more ES connect to networks (IoT devices, for example), they become potential targets for security breaches hence ensuring the security of these systems is a growing challenge.

EMBEDDED SYSTEM DESIGN CHALLENGES

- **Reliability and Safety:**
  - Fault Tolerance: Many ES operate in environments where reliability is critical thus designing for fault tolerance and ensuring the system can recover from errors is a significant challenge.
  - Safety-Critical Considerations: In applications such as medical devices or automotive control systems, the system must comply with safety standards, adding an extra layer of complexity to the design process.

- **Integrating with Existing Systems:** Integrating embedded systems into pre-existing infrastructure is a significant challenge due to complexity arises from the necessity for compatibility and seamless communication between different systems.

EMBEDDED SYSTEM DESIGN CHALLENGES

- **Variability in Hardware Platforms:** ES are deployed on a wide range of hardware platforms with varying architectures henceforth developing software that can run seamlessly across different platforms is a challenge.

- **Development Tools and Environments:** Compared to general-purpose software development, ES have fewer development tools and debugging capabilities. Developers must work with tools optimized for specific microcontrollers or processors.

- **Testing and Debugging:** Debugging ES can be challenging due to limited visibility into the system's internal state hence techniques such as in-circuit debugging and simulation are often employed.

PERFORMANCE AND BENCHMARKING TOOLS

PERFORMANCE AND BENCHMARKING TOOLS

- Benchmarking is the process of comparing two or more systems to determine which is more efficient and/or provides better performance.
- For many professionals, benchmarking is almost synonymous with Dhrystones and MIPS.
- **MIPS:**
  - Originally defined in terms of the VAX 11/780 minicomputer which was the first machine that could run 1 million instructions per second (1 MIPS).
  - An instruction, however, is a one-dimensional metric that might not have anything to do with the way work scales on different machine architectures.
  - Executing 1,500 instructions on a RISC architecture or executing 1,000 instructions on a CISC architecture? Unless you are comparing VAX to VAX, MIPS doesn't mean much.
  - MIPS is not used commonly now, because it failed to consider other factors that can affect performance.
  - Instead of MIPS, many organizations and professionals rely SPEC CPU or Geekbench benchmarks.

PERFORMANCE AND BENCHMARKING TOOLS

Dhrystone:

- This benchmark is a simple C program that compiles to about 2,000 lines of assembly code and is independent of operating system services.
- The Dhrystone benchmark was also calibrated to the venerable VAX because a VAX 11/780 could execute 1,757 loops through the Dhrystone benchmark in 1 second, 1,757 loops became 1 Dhrystone.
- Dhrystone may represent a result more meaningfully than MIPS because instruction count comparisons between different instruction sets (e.g. RISC vs. CISC) can confound simple comparisons.
- For example, the same high-level task may require many more instructions on a RISC machine, but might execute faster than a single CISC instruction.
- Thus, the Dhrystone score counts only the number of program iteration completions per second, allowing individual machines to perform this calculation in a machine-specific way.
- The problem with the Dhrystone test is that It is susceptible to compiler optimizations, a crafty compiler designer can optimize the compiler to blast through the Dhrystone benchmark and do little else well.

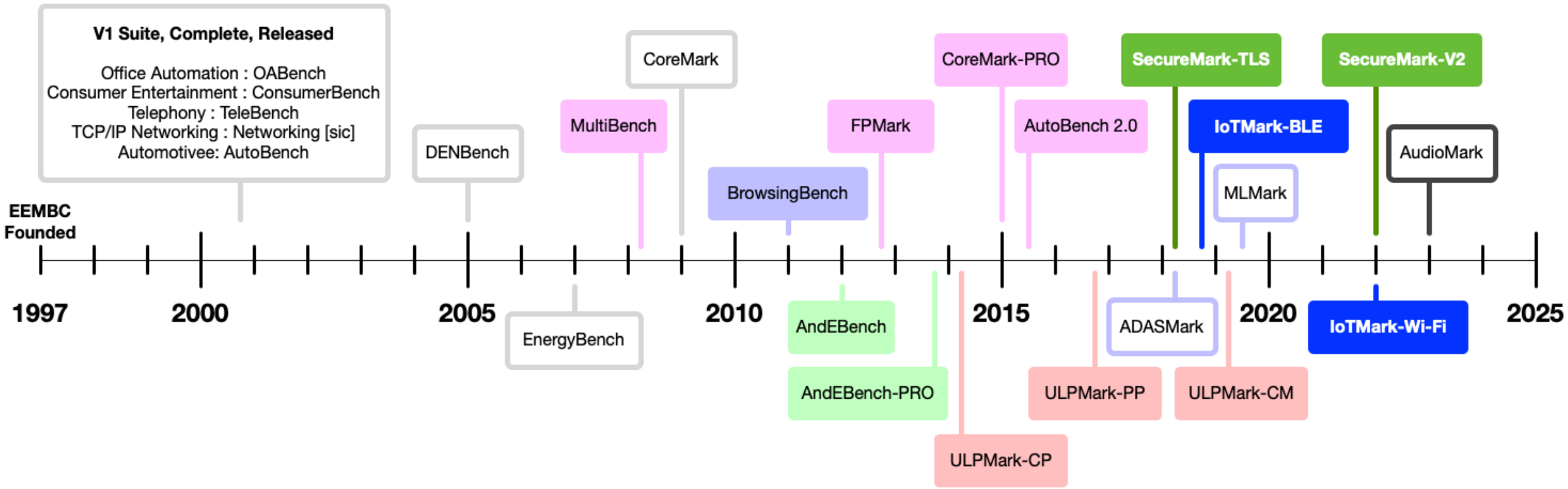
PERFORMANCE AND BENCHMARKING TOOLS

EEMBC (EDN Embedded Microprocessor Benchmark Consortium)

- Clearly, MIPS and Dhrystone measurements aren't adequate; designers still need something more tangible for their processor selection.
- To address this need, representatives of the semiconductor vendors, the compiler vendors, and their customers created a more meaningful benchmark i.e EEMBC.
- Benchmarks can be grouped into FIVE following categories:
  1. Single-core Processor Performance
  2. Symmetric Multi-core Processor Performance
  3. Ultra-Low Power and Internet of Things
  4. Heterogeneous Compute
  5. Phone and Tablet

PERFORMANCE AND BENCHMARKING TOOLS

EEMBC (EDN Embedded Microprocessor Benchmark Consortium)



A brief history of EEMBC benchmark releases

Image credit: EEMBC, Ref. URL : https://www.eembc.org/products/#ulp

PERFORMANCE AND BENCHMARKING TOOLS

EEMBC (EDN Embedded Microprocessor Benchmark Consortium)

1. Single-core Processor Performance:

- AudioMark: A modern audio pipeline with multiple datatypes, larger code size, DSP, and ML.
- CoreMark: A simple, yet sophisticated test of the functionality of a processor core; it produces a single-number score allowing users to make quick comparisons between processors.
- AutoBench: The algorithms in the benchmark analyze processor performance in automotive, industrial, and general-purpos applications.
- DENBench (Digital Entertainment): This benchmark addresses digital entertainment products such as smartphones, MP3 players, digital cameras, TV set-top boxes, and in-car entertainment systems.
- Networking 2.0: Routers and switches can benefit from this benchmark's analysis of performance associated with moving packets in networking applications.
- OABench (Office Automation): This benchmark approximates office automation tasks performed by processors in printers, plotters, and other systems that handle text and image processing tasks.
- TeleBench: The telecommunications suite approximates performance of processors in modem, xDSL, and related fixed-telecom applications.

PERFORMANCE AND BENCHMARKING TOOLS

EEMBC (EDN Embedded Microprocessor Benchmark Consortium)

2. Symmetric Multi-core Processor Performance:

- Built on the multi-instance test harness (MITH) which exploits the POSIX pthreads interface for testing both context- and worker-level parallelism.
- CoreMark-Pro: Builds on the original CoreMark benchmark by adding context-level parallelism and 7 new workloads which cover integer and floating-point performance.
- AutoBench-2.0: The original AutoBench benchmark is back in parallel form, with more workloads consistenting on aggregated tasks (combinations of AutoBench-1.1 kernels) that reflect the increase in compute demand of automotive electronic control units (ECU).
- FPMark: Focuses entirely on single- and double-precision floating-point workloads. From Gauss-Jordan Elimination to Black Schols computation, FPMark covers a broad range of intense floating point analysis.
- MultiBench: To analyze worker-level parallelism in addition to context-level, MultiBench contains integer workloads that scale in both directions.

PERFORMANCE AND BENCHMARKING TOOLS

EEMBC (EDN Embedded Microprocessor Benchmark Consortium)

3. Ultra-Low Power (ULP) and Internet of Things:

- The ULP subcommittee focuses on power and energy and the scores associated with the benchmarks are derived from measurements taking using the STMicroelectronics PowerShield.
- ULPMark-CP (CoreProfile): The benchmark runs an active workload for a period of time, then goes to sleep. The energy measurement during the duty cycle reflects a real-life test of embedded low power beyond a simple sleep number.
- The ULPMark-PP (PeripheralProfile): Examines the energy cost of four peripherals - real-time clock, pulse-width modulation, analog-to-digital conversion, and SPI communication.
- ULPMark-CM (CoreMark): It is first active-power embedded benchmark to measures the energy using CoreMark as the workload in a consistent environment.

PERFORMANCE AND BENCHMARKING TOOLS

EEMBC (EDN Embedded Microprocessor Benchmark Consortium)

- IoTMark: Builds on ULPMark by adding a sensor emulation module and a radio gateway. The execution profile incorporates the types of behavior an IoT edge node would perform.
- IoTMark-BLE: Uses a Bluetooth Low Energy (BLE) radio as the gateway and an I2C device as the sensor.
- IoTMark-Wi-Fi: Explores link- and application-layer energy efficiency for 802.11 devices.
- SecureMark: Security comes at a cost, both in programming complexity and energy. SecureMark provides security-specific profiles to assess the energy cost of the design.
- SecureMark-TLS (Transport Layer Security): Provides a method to account for the cost of a TLS handshake using elliptic curve key exchange and signing, as well as AES128 CCM & ECB for a ciphers and SHA256 for hashing.

PERFORMANCE AND BENCHMARKING TOOLS

EEMBC (EDN Embedded Microprocessor Benchmark Consortium)

4. Heterogeneous Compute:

- EEMBC's original benchmarks focused on single core 8 and 16-bit embedded processors which aren't suitable for todays mixed-core heterogeneous platforms deployed for high-performance computation.
- The benchmarks in this category differ from single-core or symmetric multi-core performance benchmarks of EEMBC, in that they provide more sophisticated frameworks for assymmetric compute.
- ADASMark: Focuses benchmarking a typical vision pipeline that may be used in advanced driver-assistance systems (ADAS) platforms. Built on OpenCL, the pipeline may be distributed among CPUs, GPUs and DSPs.
- MLMark: (Machine Learning): This benchmark strives to categories and analyze several broad classes of popular ML neural networks on embedded edge-compute platforms.

5. Phone and Tablet:

- BrowsingBench: Measure browser performance with BrowsingBench, a collection of webpages loaded using a local Nginx server over a wired LAN connection.

THANK YOU