IoT Application Development

Tassadaq Hussain Cheema

Professor EE Department NAMAL University

February 27, 2018

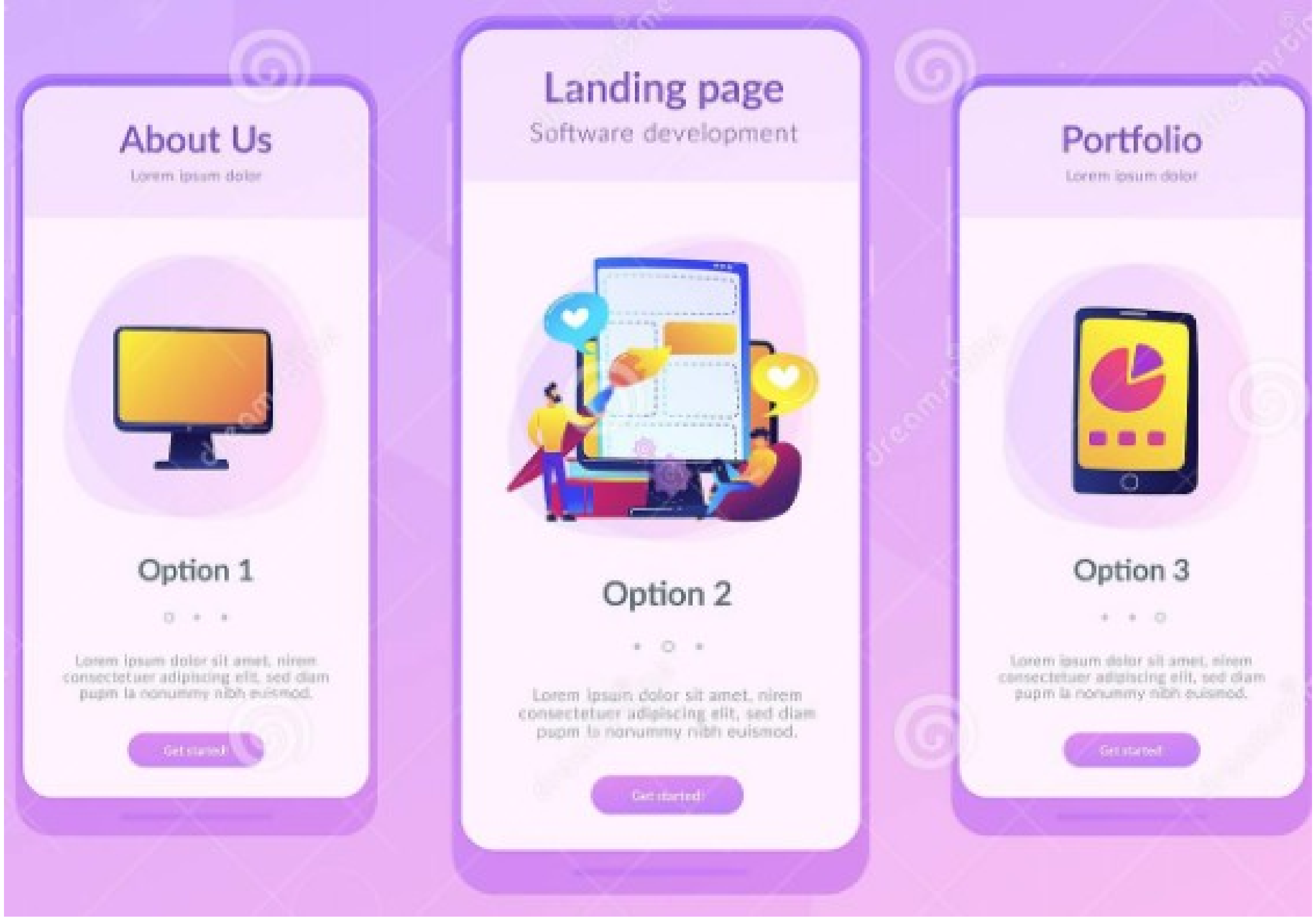
- Front-End User Interface
- Data Architecture
- Software Architecture
- Hardware Architecture

Front-end Interface (UI)

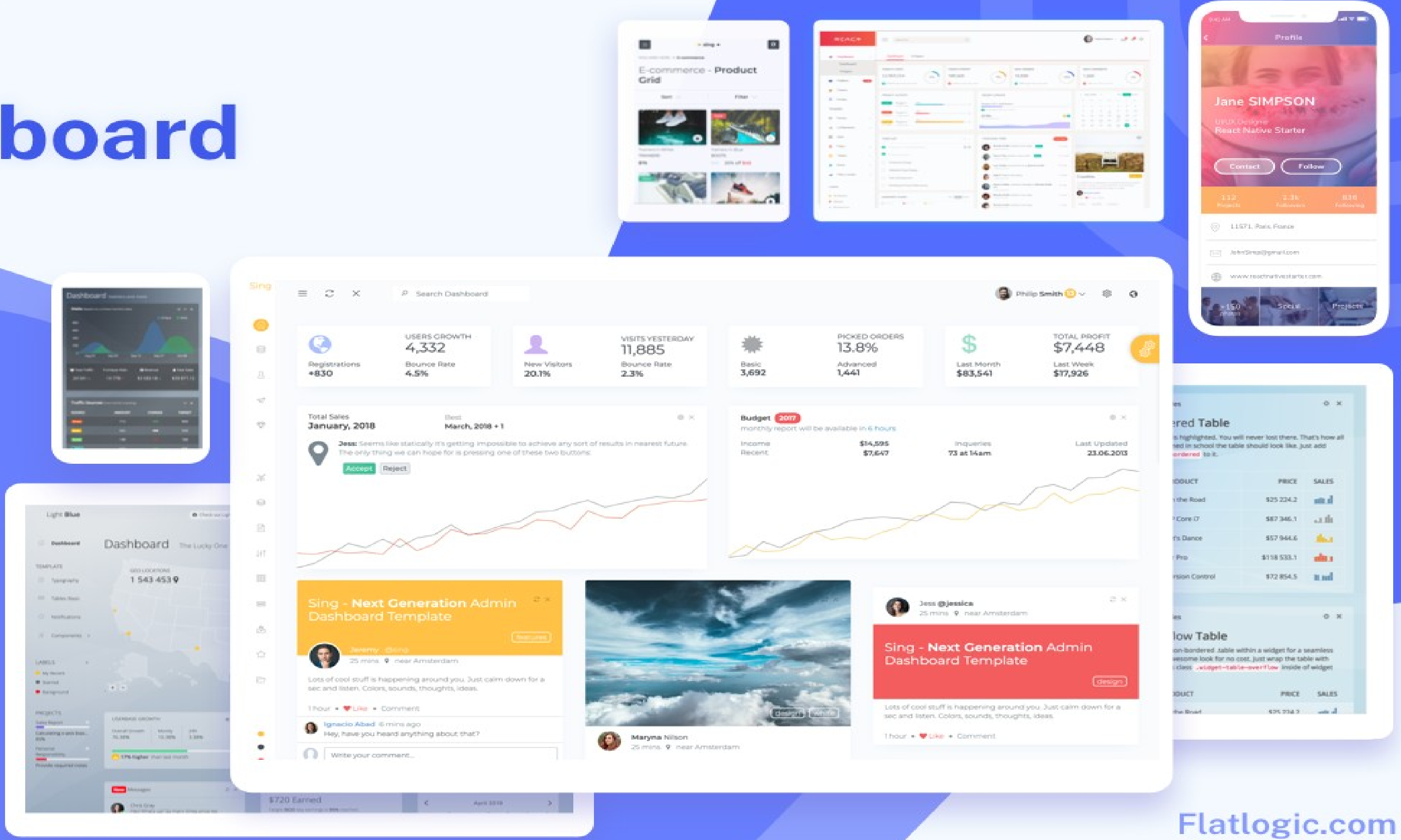
- Front-end interface architecture involves defining the visual and interactive elements of the user interface, such as buttons, menus, forms, icons, and other graphical elements. It also involves defining the layout and navigation of the user interface, including the organization of content, the flow of information, and the overall user experience.

Parameters

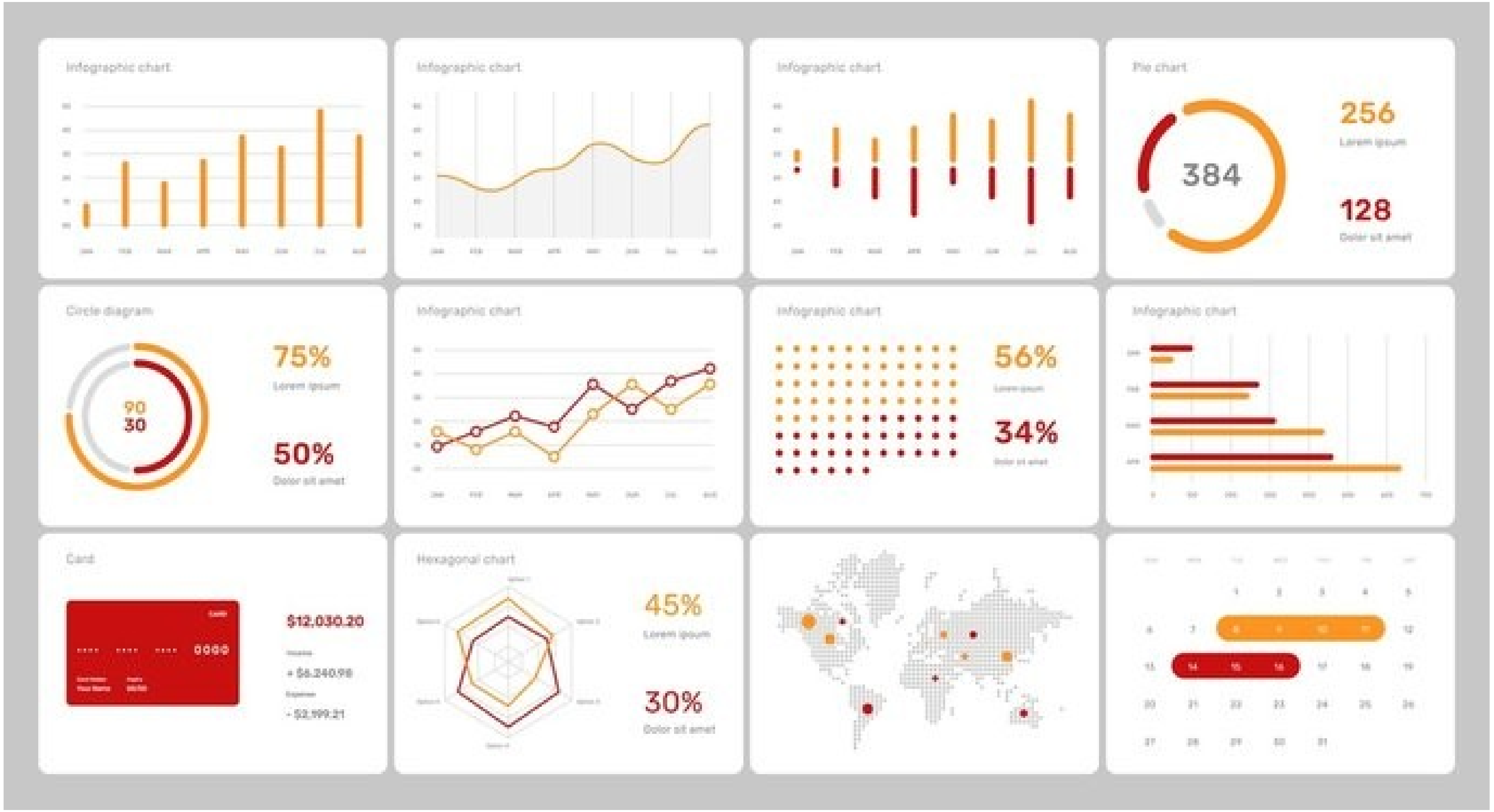
- User requirements: UI design begins with identifying the needs and preferences of the target audience, including their knowledge and experience, and designing the interface to meet their expectations.
- Visual design: UI design involves creating a visually appealing interface that is easy to read, navigate, and understand. This includes selecting appropriate colors, fonts, and images, and designing a layout that organizes the content effectively.
- Navigation: UI design includes designing an intuitive navigation system that enables users to find the information they need quickly and easily. This includes creating clear and concise menus, buttons, and links that guide users through the application.
- Interaction design: UI design involves creating an interaction design that enables users to interact with the application easily and efficiently. This includes designing forms, buttons, and other interactive elements that respond to user input and provide feedback.
- Accessibility: UI design includes designing an accessible interface that can be used by people with disabilities. This involves complying with accessibility guidelines and standards, such as WCAG (Web Content Accessibility Guidelines), and designing the interface to be compatible with assistive technologies.
- Performance: UI design includes designing the interface to perform efficiently and effectively. This involves optimizing the interface for speed, responsiveness, and scalability, and minimizing the impact on system resources.


About Us
Lorem ipsum dolor
Option 1
Lorem ipsum dolor sit amet, nirem consectetuer adipiscing elit, sed diam pupm la nonummy nibh euismod.
Get started
Landing page
Software development
Option 2
Lorem ipsum dolor sit amet, nirem consectetuer adipiscing elit, sed diam pupm la nonummy nibh euismod.
Get started
Portfolio
Lorem ipsum dolor
Option 3
Lorem ipsum dolor sit amet, nirem consectetuer adipiscing elit, sed diam pupm la nonummy nibh euismod.
Get started

IOT Dashboard



Flatlogic.com


Infographic chart
Infographic chart
Infographic chart
Pie chart
384
256
Lorem ipsum
128
Dolor sit amet
Circle diagram
90
30
75%
Lorem ipsum
50%
Dolor sit amet
Infographic chart
Infographic chart
56%
Lorem ipsum
34%
Dolor sit amet
Infographic chart
Card
0000
$12,030.20
+ $6,240.98
- $3,999.21
Hexagonal chart
45%
Lorem ipsum
30%
Dolor sit amet
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

Python and Data Visualization

- Matplotlib
- Plotly
- Seaborn
- Ggplot
- Altair

Library	Description	Pros	Cons	Web and Mobile Support
Matplotlib	A widely-used plotting library for creating static plots.	- Offers extensive customization options	- Steeper learning curve compared to some other libraries	Limited
Plotly	An interactive visualization library with web-based features.	- Produces interactive and animated plots	- Some advanced features require a paid subscription	Extensive
Seaborn	Built on top of Matplotlib, provides high-level interface.	- Simplifies creating attractive statistical visualizations	- Limited customization options compared to Matplotlib	Limited
Ggplot	Based on the popular R library ggplot2.	- Familiar syntax for R users	- Less actively maintained compared to other libraries	Limited
Altair	Declarative library based on the Grammar of Graphics.	- Easy to use with a concise and intuitive syntax	- Limited support for some complex plot types	Limited

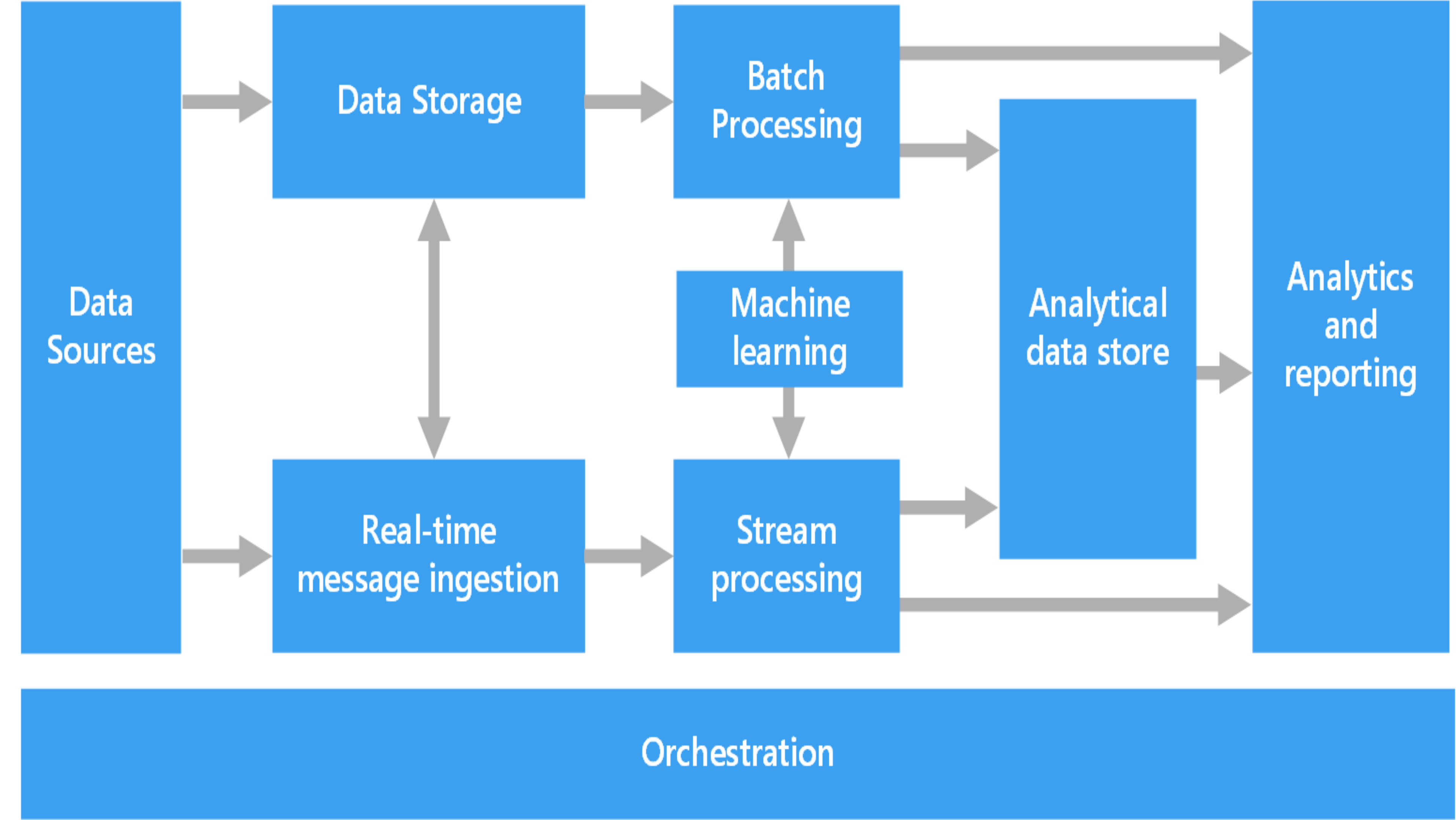
Data Architecture

- Data architecture is a set of principles, policies, standards, and models that govern how data is organized, stored, processed, and accessed by the IoT system.
- DA key component of IoT application architecture and plays a critical role in supporting complex operations and decision-making.

Data Architecture Parameters

- Data requirements: Data architecture begins with identifying the data requirements of the application program, including the types of data needed, their sources, and their intended use. Data requirements should be aligned with the application objectives and strategies.
- Data models: Data architecture involves creating data models that describe the structure and relationships of the data. Data models can be conceptual, logical, or physical, and they should be designed to support the applicatoin data requirements.
- Data storage and retrieval: Data architecture also involves defining how data will be stored and retrieved in the system. This includes selecting appropriate data storage technologies, defining data access methods, and establishing data management policies.
- Data integration: Data architecture includes designing and implementing data integration solutions to ensure that data from different sources can be combined and analyzed effectively. This involves defining data integration architectures and selecting appropriate tools and technologies for data integration.
- Data security: Data architecture includes defining data security policies and procedures to ensure that data is protected from unauthorized access, use, or disclosure. This involves establishing data security controls, such as access controls, encryption, and authentication, and implementing security best practices.
- Data governance: Data architecture includes defining data governance policies and procedures to ensure that data is managed effectively and efficiently. This involves establishing data quality standards, data management processes, and data stewardship roles and responsibilities.

- Data Structures: Choose appropriate data structures in Python based on the nature of your data and the operations you need to perform. Python provides built-in data structures like lists, tuples, sets, and dictionaries. Understanding the characteristics and performance implications of each data structure is crucial for efficient data handling.
- Data Validation and Cleaning: Implement data validation and cleaning techniques to ensure the integrity and quality of your data. Python offers various libraries and functions for data validation, such as regular expressions, data type checking, and input sanitization.
- Modular Code Organization: Organize your Python code into modular components to promote reusability and maintainability. Define separate functions or classes for specific data processing tasks, making it easier to handle different aspects of data architecture.
- Data Persistence: Choose appropriate methods for data persistence, such as writing data to files (e.g., CSV, JSON, or binary formats), interacting with databases using Python database APIs (e.g., SQLite, PostgreSQL), or using NoSQL databases (e.g., MongoDB) depending on the requirements of your application.
- Data Serialization: When transferring data between different components or systems, use serialization techniques to convert data into a portable format. Python provides modules like pickle, json, and msgpack for serializing and deserializing data.
- Data Processing and Analysis: Utilize Python libraries for data processing and analysis, such as NumPy, Pandas, and SciPy.
- Visualization: Leverage Python libraries like Matplotlib, Plotly, or Seaborn for data visualization.
- Data Pipelines and Workflow: Design data pipelines and workflows using Python libraries like Apache Airflow or Luigi. These tools help manage and orchestrate the flow of data between different stages of processing, making it easier to handle complex data transformations and dependencies.
- Error Handling and Logging: Implement error handling mechanisms and logging practices to capture and handle exceptions or errors during data processing. Proper error handling and logging allow for easier debugging and troubleshooting.
- Performance Optimization: (Parallel Processing Numba or Cython for faster execution)
- Scalability and Distributed Computing: (Dask or Apache Spark for distributed computing


Data Sources
Data Storage
Real-time message ingestion
Batch Processing
Machine learning
Stream processing
Analytical data store
Analytics and reporting
Orchestration

Software Architecture

Software architecture involves the design and organization of software systems. Key parameters involved in software architecture include:
Functionality: This refers to the software's ability to perform the intended tasks and meet the needs of its users.
Scalability: This parameter is about the software's ability to handle increased workload or larger amounts of data without losing performance.
Reliability: The software should be dependable and able to consistently perform its functions correctly, even under unexpected circumstances.
Reuseability: The software should be easy to modify or maintain over time, without requiring significant effort or causing downtime.
Performance: Deals with the software's speed and efficiency to optimize the best possible user experience.
Security: The software should be designed with appropriate security measures to protect against unauthorized access, data breaches, and other threats.
Portablity: The software should be able to work with other systems and software programs, using standard protocols and interfaces.
Usability: The software should be designed with a user-friendly interface and intuitive user experience, to make it easy for users to interact with and achieve their goals.
Architecture style: There are several architecture styles such as Client-Server, Microservices, Monolithic, Event-driven, etc., and choosing the right style is an important parameter in software architecture.
Technology stack: The technology stack includes the programming languages, frameworks, and tools used to develop the software. Choosing the right technology stack can affect the software's performance, scalability, and maintainability.

- **Device Integration:**
- **Data Management Libraries**
- **Operating System Support**
- **Data Processing and Analytics:**
- **Communication and Protocols:**
- **Edge Computing:**
- **Scalability and Flexibility:**
- **Real-time Event-Trigger Architecture:**
- **API and Service Design:**.
- **Device Management and Monitoring:**
- **Cloud Integration:**
- **Redundancy and Failover:**

Software Structure

# Libraries

# Global variables

# Control Signals

# Initialization (Peripherals etc.)

# Data Structure (Read/Write)

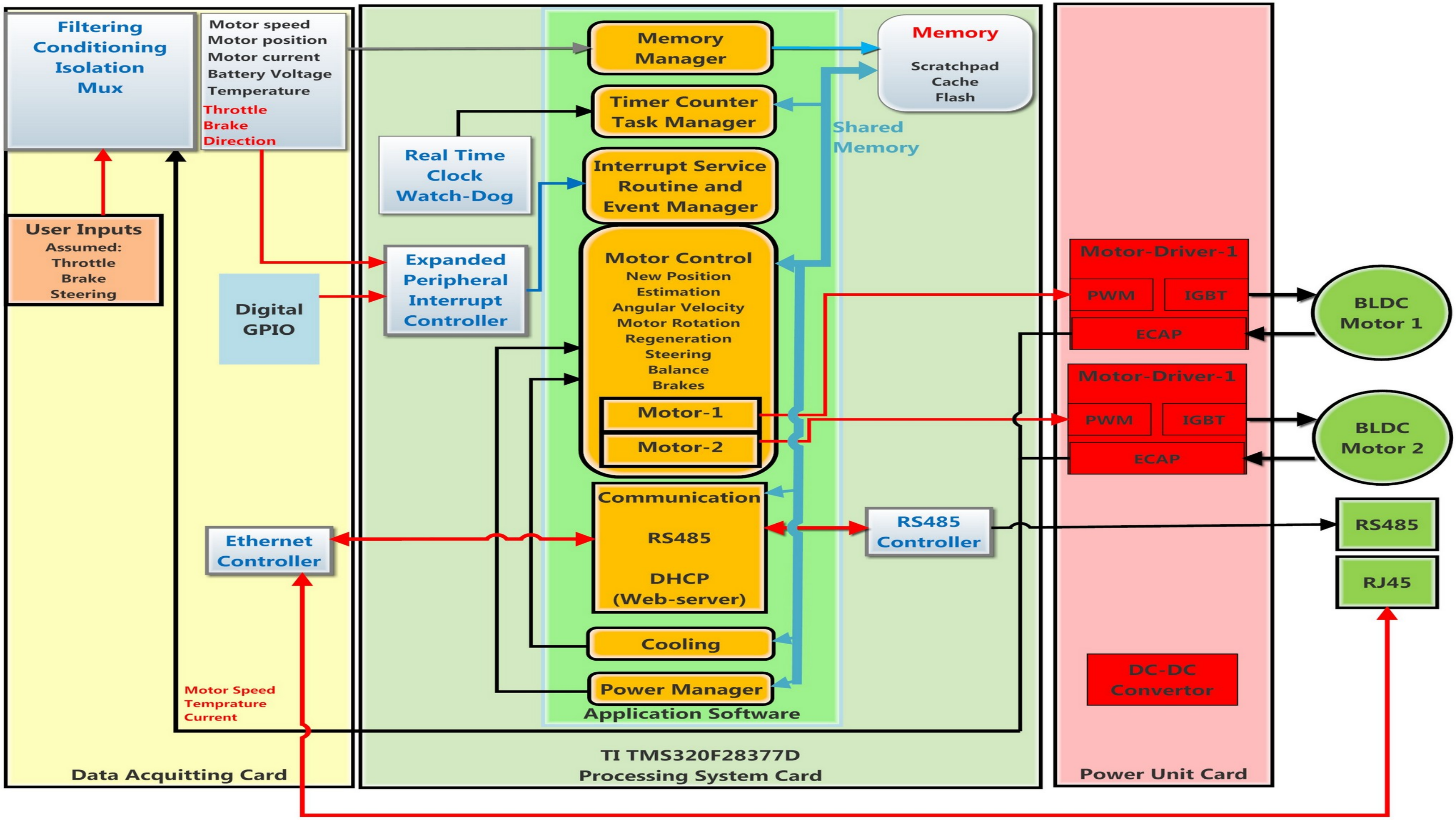
# Processing

#Function Calls (Read/Write, Pre-Process, Filter, Computation, etc.)

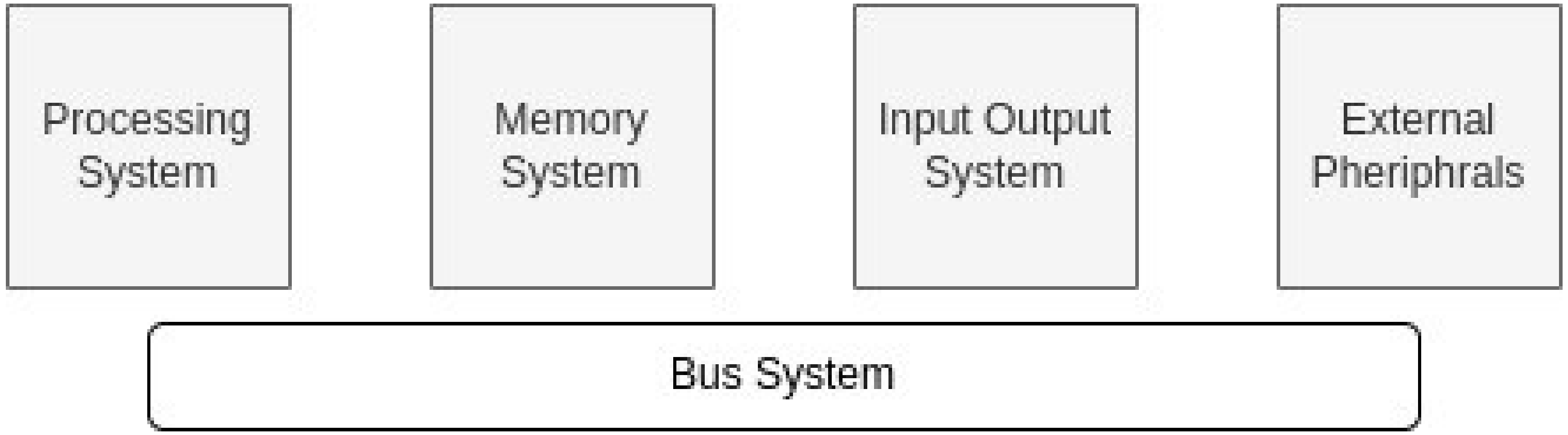
# Transfer / Store / Display

Software Architecture Representation

- Flow Diagram: It is used to describe the flow of control or data between different components or subsystems within the software system. They can help to illustrate how the different components interact with each other and how the data flows between them.
- Block diagrams: Block diagrams are used to illustrate the high-level structure of a system or software architecture. They use blocks to represent different components or subsystems and arrows to show the relationships and interactions between them.
- State diagrams: State diagrams are used to represent the behavior of a system or software architecture. They show the different states that a system can be in and the events that trigger transitions between these states.
- Entity-relationship diagrams: Entity-relationship diagrams are used to represent the relationships between different entities in a system. They are commonly used in database design to illustrate the relationships between tables.

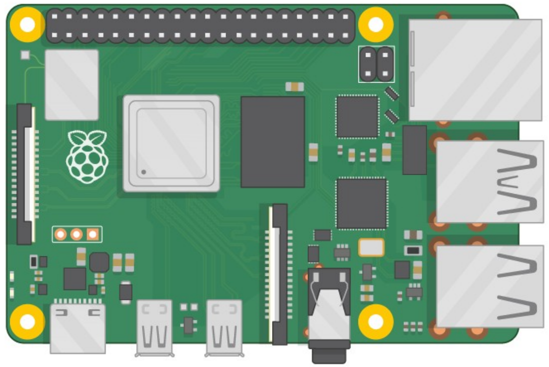

Filtering
Conditioning
Isolation
Mux
Motor speed
Motor position
Motor current
Battery Voltage
Temperature
Throttle
Brake
Direction
User Inputs
Assumed:
Throttle
Brake
Steering
Digital GPIO
Ethernet Controller
Motor Speed
Temprature
Current
Data Acquitting Card
Memory Manager
Timer Counter Task Manager
Real Time Clock Watch-Dog
Interrupt Service Routine and Event Manager
Expanded Peripheral Interrupt Controller
Motor Control
New Position Estimation
Angular Velocity
Motor Rotation
Regeneration
Steering
Balance
Brakes
Motor-1
Motor-2
Communication
RS485
DHCP
(Web-server)
Cooling
Power Manager
Application Software
Memory
Scratchpad
Cache
Flash
Shared Memory
RS485 Controller
TI TMS320F28377D
Processing System Card
Motor-Driver-1
PWM
IGBT
ECAP
Motor-Driver-1
PWM
IGBT
ECAP
BLDC Motor 1
BLDC Motor 2
RS485
RJ45
DC-DC Convertor
Power Unit Card

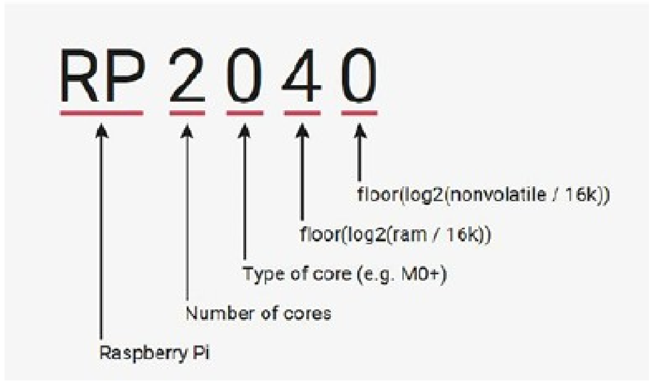
Hardware Architecture

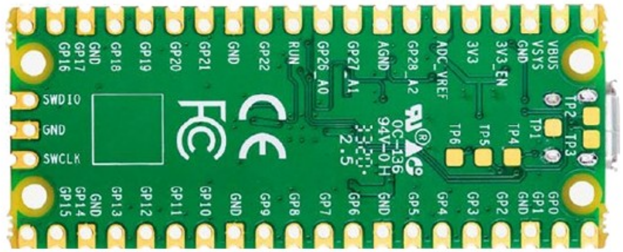


Hardware Architecture

- Single Board Computer
  - RasppberyPi
  - RasppberyPi Pico



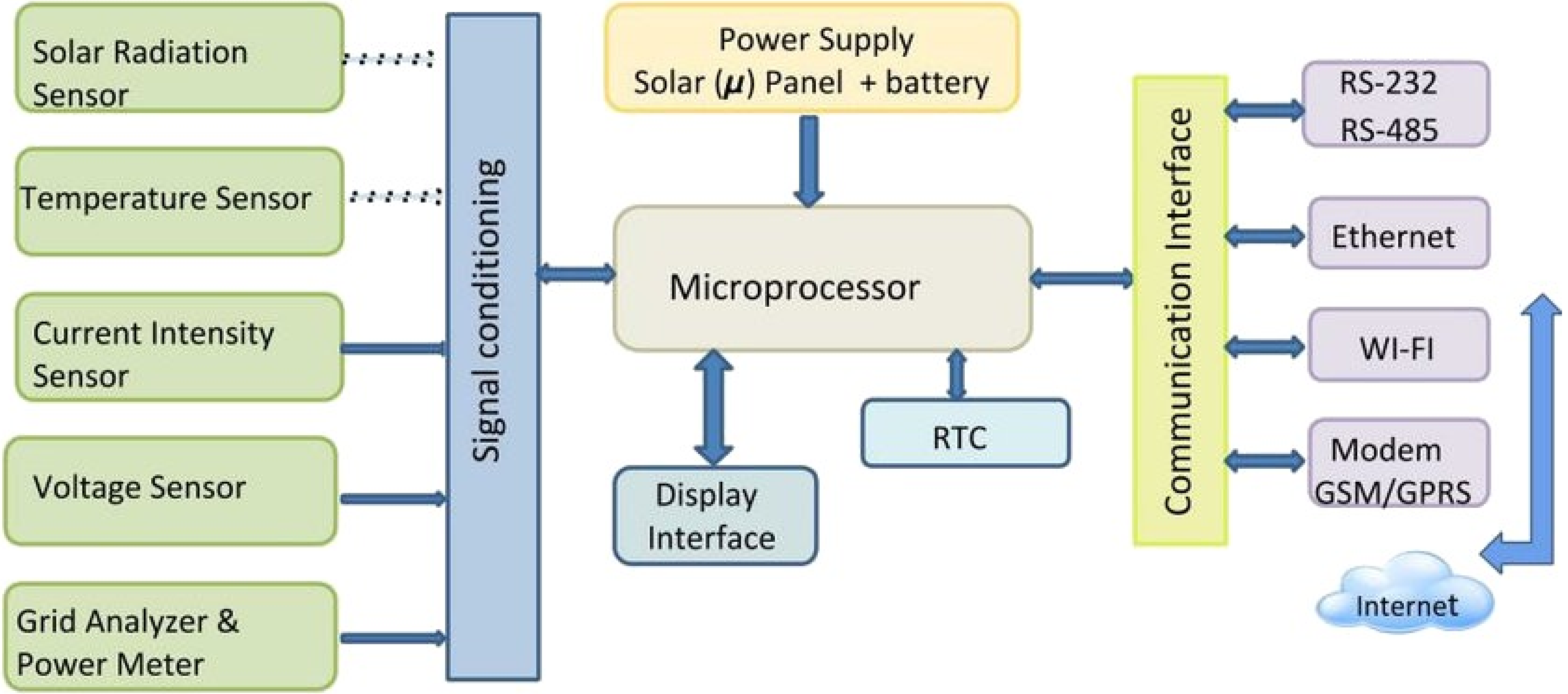




The Raspberry Pi Pico – bottom view (courtesy of raspberrypi.org)

Hardware Specifications

- Processor Architecture
  - Number of Cores (ALU) and Operating Frequency
- Processor Local Bus
  - Instruction and Data Bus
  - Program and Data Memory
  - Local Memory (Cache and ScratchPad)
- Processor External Bus (Peripheral Bus)
  - Main Memory
  - DMA, PWM, ADC, DAC
  - I/O Interfaces



Solar Radiation Sensor
Temperature Sensor
Current Intensity Sensor
Voltage Sensor
Grid Analyzer & Power Meter
Signal conditioning
Power Supply Solar (μ) Panel + battery
Microprocessor
Display Interface
RTC
Communication Interface
RS-232 RS-485
Ethernet
WI-FI
Modem GSM/GPRS
Internet

Assignment 2

- Select a sensor available in the lab, and make a project title
- Draw/make data architecture, Front-end Architecture, Software Architecture and Hardware Architecture for the Project-title.