**INTERNET OF THINGS ASSIGNMENT RECORD**

**Subject code : BTCS-AMDS-009T**

|  |  |
| --- | --- |
| Name: | Rudra Prasanna Mishra |
| Registration Number: | FET-BAML-2022-2026-032 |
| Course: | Btech CSE AIML |
| Semester: | 5th |
| Faculty: | Mr. Biswajeeban Mishra &  Mr. Pritam Nanda |

|  |  |
| --- | --- |
| Remarks |  |
| Signature |  |



SRI SRI UNIVERSITY

Bidyadharpur, Cuttack, Odisha.

Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Experiment/Case Study** | **Page No.** | **Remark** |
| **1** | Prototyping and Arduino uno r3 |  |  |
| **2** | Encoding formats. |  |  |
| **3** | Basic Structure of an Arduino Program. |  |  |
| **4** | The Architecture of Modern Computers |  |  |
| **5** | Communication Protocol |  |  |
| **6** |  |  |  |
| **7** |  |  |  |
| **8** |  |  |  |
| **9** |  |  |  |
| **10** |  |  |  |
| **11** |  |  |  |
| **12** |  |  |  |
| **13** |  |  |  |
| **14** |  |  |  |
| **15** |  |  |  |
| **16** |  |  |  |

***Assignment No.:1***

**1. What is a Prototype?**

A [prototype](https://en.wikipedia.org/wiki/Prototype) is an early sample, model, or release of a product built to test a concept or process. It is a working model used to demonstrate and validate the design, functionality, and performance of a product before it is manufactured at scale. Prototyping is an essential phase in the product development cycle as it allows for the identification and rectification of design flaws and the gathering of user feedback.

**Open Source and Closed Source Prototype Platforms**

**Open Source Prototype Platforms:**

* These platforms offer designs and software that are publicly accessible, allowing users to view, modify, and distribute the code and design files.
* Examples include Arduino, Raspberry Pi, and BeagleBone.
* Opensource platforms encourage community collaboration and innovation.

**Closed Source Prototype Platforms:**

* These platforms have proprietary designs and software that are not freely accessible to the public. The source code and design files are controlled and protected by the company that developed them.
* Examples include platforms like Microsoft's Azure Sphere and some proprietary development boards from companies like Intel and TI.
* Closed source platforms often come with official support and documentation but limit customization and modifications.

**2. What is Arduino?**

[Arduino](https://en.wikipedia.org/wiki/Arduino) is an open-source electronics platform based on easy-to-use hardware and software. It is designed for anyone interested in creating interactive projects. Arduino boards are equipped with sets of digital and analog input/output (I/O) pins that can be interfaced with various expansion boards and other circuits. The platform uses the Arduino programming language, based on Wiring, and the Arduino Software (IDE), based on Processing, to write and upload code to the physical board.

**3. Arduino Uno R3 Key Specifications**

**Main Processor:**

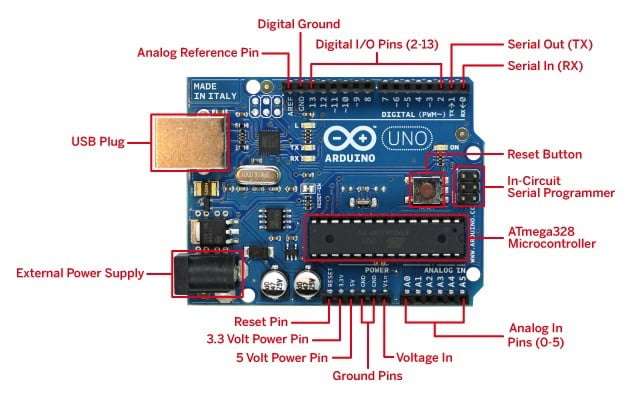
* **Microcontroller:** [ATmega328P](https://en.wikipedia.org/wiki/ATmega328) a  [8-bit](https://en.wikipedia.org/wiki/8-bit) [RISC](https://en.wikipedia.org/wiki/Reduced_instruction_set_computer) processor core.

**Memory:**

* **SRAM:** 2 KB (ATmega328P) [SRAM](https://en.wikipedia.org/wiki/Static_random-access_memory)
* **Flash Memory:** 32 KB (ATmega328P) of which 0.5 KB is used by the bootloader. [Flash memory](https://en.wikipedia.org/wiki/Flash_memory)
* **EEPROM:** 1 KB (ATmega328P) [EEPROM](https://en.wikipedia.org/wiki/EEPROM)

**I/O Pins:**

* **Digital I/O Pins:** 14 (of which 6 provide PWM output)
* **Analog Input Pins:** 6
* **PWM Pins:** 6
* **UART:** 1
* **SPI:** 1
* **I2C:** 1
* **External Interrupts:** 2 (pins 2 and 3)



***Assignment No.:2***

**Q1: What is an encoding format?**

**A1:** An encoding format is a standardized method for converting data into a specific format that allows for efficient storage, transmission, and interpretation by computers.

**Q2: What are some common encoding formats for text?**

**A2:** Common text encoding formats include:

* [**ASCII (American Standard Code for Information Interchange)**](https://en.wikipedia.org/wiki/ASCII)
* [**UTF-8 (Unicode Transformation Format - 8-bit)**](https://en.wikipedia.org/wiki/UTF-8)
* [**UTF-16 (Unicode Transformation Format - 16-bit)**](https://en.wikipedia.org/wiki/UTF-16)
* [**UTF-32 (Unicode Transformation Format - 32-bit)**](https://en.wikipedia.org/wiki/UTF-32)
* [**ISO 8859-1 (Latin-1)**](https://en.wikipedia.org/wiki/ISO/IEC_8859-1)

**Q3: What are some encoding formats used for numbers?**

**A3:** Common number encoding formats include**:**

* [**Binary (Base-2)**](https://en.wikipedia.org/wiki/Binary_number)
* [**Decimal (Base-10)**](https://en.wikipedia.org/wiki/Decimal)
* [**Hexadecimal (Base-16)**](https://en.wikipedia.org/wiki/Hexadecimal)
* [**IEEE 754**](https://en.wikipedia.org/wiki/IEEE_754)
* [**BCD (Binary-Coded Decimal)**](https://en.wikipedia.org/wiki/Binary-coded_decimal)

**Q4: What are some encoding formats used for photos or images?**

**A4:** Common photo/image encoding formats include:

* [**JPEG (Joint Photographic Experts Group)**](https://en.wikipedia.org/wiki/JPEG)
* [**PNG (Portable Network Graphics)**](https://en.wikipedia.org/wiki/Portable_Network_Graphics)
* [**GIF (Graphics Interchange Format)**](https://en.wikipedia.org/wiki/GIF)
* [**BMP (Bitmap)**](https://en.wikipedia.org/wiki/BMP_file_format)
* [**TIFF (Tagged Image File Format)**](https://en.wikipedia.org/wiki/TIFF)

**Q5: What are some common audio encoding formats?**

**A5:** Common audio encoding formats include:

* [**MP3 (MPEG-1 Audio Layer III)**](https://en.wikipedia.org/wiki/MP3)
* [**WAV (Waveform Audio File Format)**](https://en.wikipedia.org/wiki/WAV)
* [**AAC (Advanced Audio Coding)**](https://en.wikipedia.org/wiki/Advanced_Audio_Coding)
* [**FLAC (Free Lossless Audio Codec)**](https://en.wikipedia.org/wiki/FLAC)
* [**OGG (Ogg Vorbis)**](https://en.wikipedia.org/wiki/Vorbis)

**Q6: What are some common video encoding formats?**

**A6:** Common video encoding formats include:

* [**MP4 (MPEG-4 Part 14)**](https://en.wikipedia.org/wiki/MPEG-4_Part_14)
* [**AVI (Audio Video Interleave)**](https://en.wikipedia.org/wiki/Audio_Video_Interleave)
* [**MKV (Matroska Video)**](https://en.wikipedia.org/wiki/Matroska)
* [**MOV (QuickTime Movie)**](https://en.wikipedia.org/wiki/QuickTime_File_Format)
* [**WMV (Windows Media Video)**](https://en.wikipedia.org/wiki/Windows_Media_Video)

***Assignment No.:3***

**Basic Structure of an Arduino Program**

1. **Include Libraries (Optional):**
   * At the top of the sketch, you can include libraries that extend the functionality of your Arduino code. This is done using the #include directive.
   * Example: #include <DTH11.h> // Include DTH11 library for controlling sensor.
2. **Global Variables and Constants:**
   * You can define global variables and constants that will be used throughout the program. These are declared outside of any functions, usually right after the include statements.
   * Example: int ledPin = 13; // Define a variable for the LED pin
3. **setup() Function:**
   * The setup() function is called once when the Arduino board is powered on or reset. It is used to initialize variables, pin modes, start using libraries, and perform any other setup tasks.
   * Example: void setup() {   
     pinMode(ledPin, OUTPUT); // Set the LED pin as an output  
     }
4. **loop() Function:**
   * The loop() function runs continuously after the setup() function has completed. This is where the main logic of your program goes. The Arduino executes the code inside loop() over and over again until the board is powered off or reset.
   * Example:  
      void loop() {  
     digitalWrite(ledPin, HIGH); // Turn the LED on  
     delay(1000); // Wait for 1 second  
     digitalWrite(ledPin, LOW); // Turn the LED off  
     delay(1000); // Wait for 1 second(1000 ms)  
     }

**Example of a Complete Arduino Program:**

// Include Libraries (if needed)

#include <DTH11.h>

// Global Variables and Constants

int ledPin = 13; // Define the LED pin

// setup() function: Runs once at the start

void setup() {

pinMode(ledPin, OUTPUT); // Set the LED pin as an output

}

// loop() function: Runs repeatedly after setup() finishes

void loop() {

digitalWrite(ledPin, HIGH); // Turn the LED on (HIGH is the voltage level)

delay(1000); // Wait for 1 second (1000 milliseconds)

digitalWrite(ledPin, LOW); // Turn the LED off by making the voltage LOW

delay(1000); // Wait for 1 second

}

**Key Points:**

* setup() Function: Used to initialize settings; runs only once.
* loop() Function: Contains the main logic; runs continuously.
* Global Variables: Declared outside of any function, accessible throughout the sketch.
* Comments: Use // for single-line comments or /\* \*/ for multi-line comments.

This basic structure allows you to build increasingly complex Arduino programs by adding more code to the setup() and loop() functions, as well as including libraries and defining global variables as needed

**References:**

* [Arduino](https://en.wikipedia.org/wiki/Arduino): Overview of the Arduino platform, its history, and its uses.
* [Arduino IDE](https://en.wikipedia.org/wiki/Arduino_IDE): Information on the Integrated Development Environment used to write and upload sketches to the Arduino board.

**Assignment No.:4: The Architecture of Modern Computers**

**1. Parallelism in Modern CPU Architectures (e.g., x86-64 Architecture)**

Modern CPUs like those using the x86-64 architecture are designed to handle various forms of parallelism to enhance performance and efficiency:

* Instruction-Level Parallelism (ILP): Through techniques like *pipelining*, *superscalar execution*, and *out-of-order execution*, multiple instructions can be processed in parallel on a single core. This significantly boosts CPU efficiency by allowing multiple instructions to execute simultaneously.
* Data-Level Parallelism (DLP): Single Instruction, Multiple Data (SIMD) extensions, such as AVX in x86-64, enable the CPU to apply the same operation to multiple data points in parallel, which is especially useful in tasks like multimedia processing.
* Thread-Level Parallelism (TLP): Modern CPUs can process multiple threads simultaneously with multithreading and multicore designs, improving application responsiveness and system throughput.

Implications for Software Design: These parallelism techniques encourage software developers to utilize concurrency models and optimize for scalability to maximize performance on modern CPUs.

2**. Differences Between RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing) Architectures**

* CISC (e.g., x86):
  + *Complex Instruction Set*: A large set of varied instructions, some of which execute multi-step tasks, reducing the overall instruction count.
  + *Variable-Length Instructions*: Instructions vary in length, resulting in more complex decoding but fewer instructions overall.
  + *Memory Access*: Supports memory-to-register operations, which allows direct memory access within instructions.
* RISC (e.g., ARM):
  + *Reduced Instruction Set*: Uses a smaller set of simpler instructions, resulting in higher instruction counts but efficient decoding and execution.
  + *Fixed-Length Instructions*: Simplifies pipelining, enabling faster execution and increased instruction throughput.
  + *Load/Store Architecture*: Memory access is restricted to explicit load and store instructions, making the design simpler and energy-efficient.

Impact on Operating Systems:

* *CISC OS Design*: CISC-based systems require fewer instructions per task but may involve complex decoding steps, which OS developers manage to optimize overall performance.
* *RISC OS Design*: RISC-based systems benefit from a straightforward pipeline and simpler design, which generally leads to higher throughput and modular OS designs.

**3. Comparing CISC vs. RISC Architectures and John von Neumann vs. Harvard Architectures**

* Von Neumann Architecture: Uses a unified memory for data and instructions, making it simpler and cost-effective but can lead to bottlenecks due to bus contention.
* Harvard Architecture: Separates memory for data and instructions, which reduces contention and allows parallel access, making it efficient for embedded and real-time processing.

Impact on Processor Performance: The Harvard architecture typically offers faster data handling and is ideal for specialized tasks, while the Von Neumann model is more widely used in general-purpose systems due to its simplicity.

Microcontrollers (e.g., Arduino Uno R3)

AVR vs. ARM Cortex-M Architecture

* AVR Architecture: Used in the Arduino Uno R3, an 8-bit RISC architecture known for low power consumption and a simple instruction set, ideal for basic applications.
* ARM Cortex-M Architecture: A 32-bit RISC architecture used in more advanced microcontrollers, which provides more processing power, larger memory access, and additional features suitable for complex applications.

8-bit vs. 16/32-bit Microcontrollers

* Memory Addressing: 8-bit microcontrollers have smaller address space, limiting memory access compared to 16/32-bit microcontrollers.
* Processing Power: Higher-bit microcontrollers, like 16- or 32-bit, offer more processing power and are suited for more demanding applications.
* Application Suitability: 8-bit microcontrollers are best for light-duty tasks, while 16- and 32-bit microcontrollers excel in more complex environments.

Memory Segmentation

Memory Segmentation in x86 Architecture

Memory segmentation divides the address space, facilitating backward compatibility with 16-bit applications, though it adds complexity compared to the flat memory models in modern 64-bit systems.

User Space vs. Kernel Space

* User Space: Where applications run, isolated to prevent direct access to system-critical code.
* Kernel Space: Reserved for the OS kernel to manage critical system processes and protect system stability and security.

Endianness

* Big-Endian: Stores the most significant byte first.
* Little-Endian: Stores the least significant byte first.

These differences are essential for data interpretation and correct system operation.

Related Wikipedia Articles:

* [Instruction-level parallelism](https://en.wikipedia.org/wiki/Instruction-level_parallelism)
* [RISC](https://en.wikipedia.org/wiki/Reduced_instruction_set_computing)
* [CISC](https://en.wikipedia.org/wiki/Complex_instruction_set_computing)
* [Von Neumann architecture](https://en.wikipedia.org/wiki/Von_Neumann_architecture)
* [Harvard architecture](https://en.wikipedia.org/wiki/Harvard_architecture)
* [Endianness](https://en.wikipedia.org/wiki/Endianness)

**Assignment No.:5: Communication Protocols**

**UART (Universal Asynchronous Receiver-Transmitter)**

* *Simple, two-way communication* with only two wires (TX, RX), typically operating at low speeds (9600-115200 baud rate).
* Used in basic setups like connecting Arduinos to computers.

**I²C (Inter-Integrated Circuit)**

* A *two-wire protocol* (SDA and SCL) that can connect multiple devices (up to 127) and operates at medium speeds (100kHz to 400kHz).
* Commonly used for multiple sensors and chips on one board.

**SPI (Serial Peripheral Interface)**

* A fast protocol with multiple wires (MOSI, MISO, CLK, SS), ideal for high-speed data transfers between devices like SD cards or displays.

**CAN (Controller Area Network)**

* Uses *differential signaling* with two wires (CAN-H, CAN-L) to ensure reliable communication in noisy environments.
* Commonly found in automotive and industrial applications.

**USB (Universal Serial Bus)**

* A versatile, high-speed protocol supporting both data transfer and power supply, widely used in computers, phones, and other devices.

Related Wikipedia Articles:

* [UART](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter)
* [I²C](https://en.wikipedia.org/wiki/I%C2%B2C)
* [SPI](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface)
* [CAN bus](https://en.wikipedia.org/wiki/CAN_bus)
* [USB](https://en.wikipedia.org/wiki/USB)