Internet of Things Security

Lecture 2: Introduction to Embedded Systems and Firmware

Mehmoona Jabeen

Mehmoona.jabeen@mail.au.edu.pk

Department of Cyber Security, Air University

Lecture Outlines

- Microcontroller Architecture
- Introduction to Embedded System
- Serial Communication
- Interrupt Handling (IH)
- Direct Memory Access (DMA)
- Serial Communication Protocols
- Introduction to Firmware
- Bootloading

Microcontroller vs Microprocessors

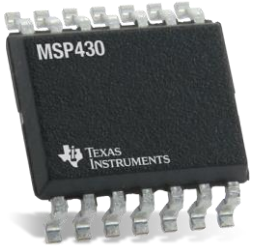
- Microprocessor consists of only a Central Processing Unit, whereas Micro Controller contains a CPU, Memory, I/O all integrated into one chip.
- Microprocessor is used in Personal Computers whereas Micro Controller is used in an embedded system.
- Microprocessor uses an external bus to interface to RAM, ROM, and other peripherals, on the other hand, Microcontroller uses an internal controlling bus.
- Microprocessors are based on Von Neumann model Micro controllers are based on Harvard architecture
- Microprocessor is complicated and expensive, with a large number of instructions to process but Microcontroller is inexpensive and straightforward with fewer instructions to process.

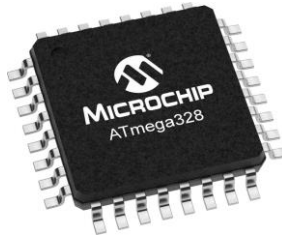
Types of Microcontroller

- Peripheral Interface Controller (PIC) Microcontroller
  - PIC is a kind of microcontroller used in the development of electronics, computer robotics, and similar devices.
  - PIC was produced by Microchip technology and based on hardware computing architecture,
- Advanced RISC Machine (ARM Microcontroller)
  - ARM is the most popular Microcontrollers Programming in the digital embedded system world
  - Industries prefer only ARM microcontrollers since it consists of significant features to implement products.
- 8051 Microcontroller
  - Intel created 8051 microcontrollers in 1981. It is an 8bit microcontroller. It's made with 40 pins DIP (Dual inline package), 4kb if ROM storage and 128 bytes of RAM storage, two 16 bit timer. It consists of are four parallel 8 bit ports, which are programmable as well as addressable as per the specification.
- Alf and Vegard's RISC (AVR) Microcontroller
  - AVR was developed by Atmel Corporation and is the modified Harvard architecture.
  - Its based on the RISC architecture
- Mixed Signal Processor (MSP) Microcontroller
  - MSP is the family from Texas Instruments and built around a 16 -bit CPU
  - MSP is designed for low cost and respectively, low power dissipation embedded statements.

Microcontroller Architecture

- Chip vendors either
  - Develop own CPU core or
  - License IP

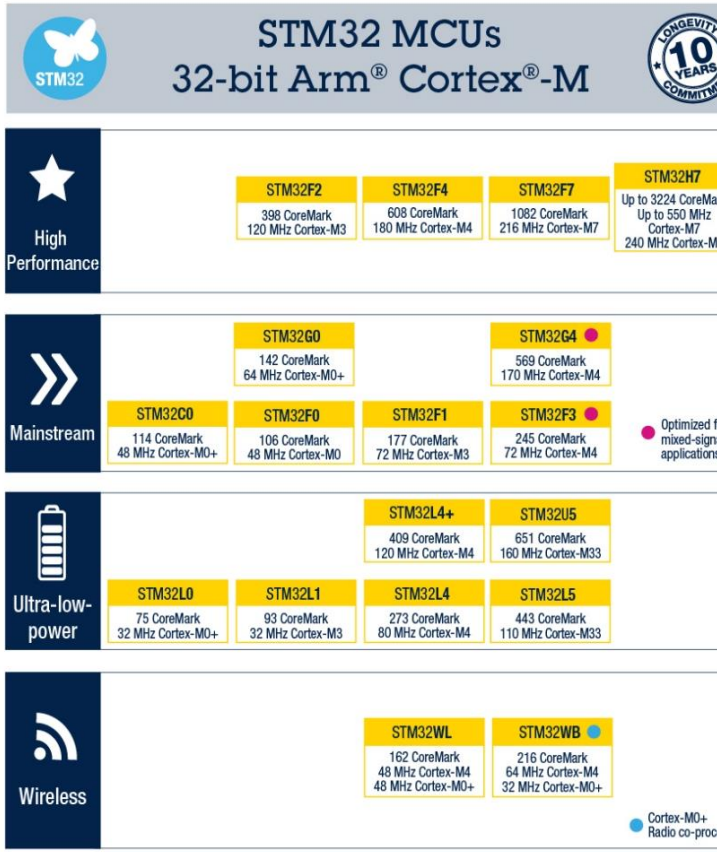




- Specific chips usually targeted toward a small set of applications
- Different from your regular desktop CPU
  - Smaller in size
  - Reduced instruction set
  - Less power consumption
  - Lower frequencies

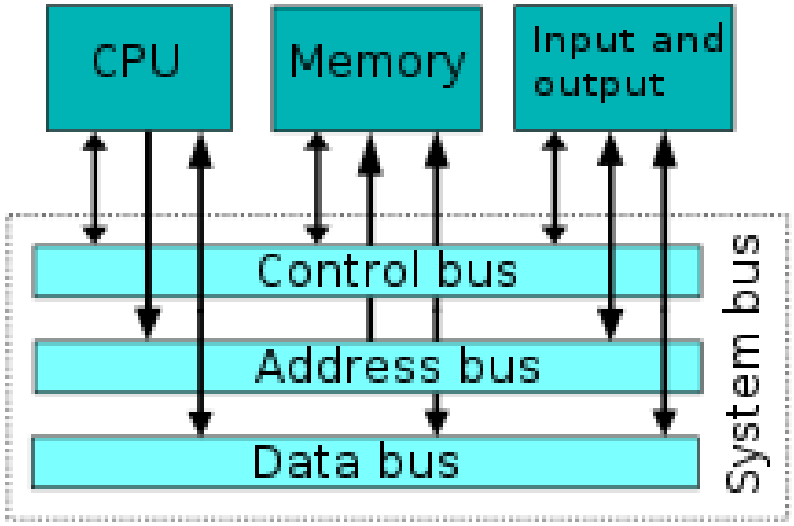
- Within microcontrollers,
  - large variation

Bus	Clock	RAM	ROM
8-Bit	1-8MHz	128-1KB	512-10KB
16-Bit	4-25MHz	1K-10KB	10KB-128KB
32-Bit	10-100MHz	10K-64MB	128KB-512MB



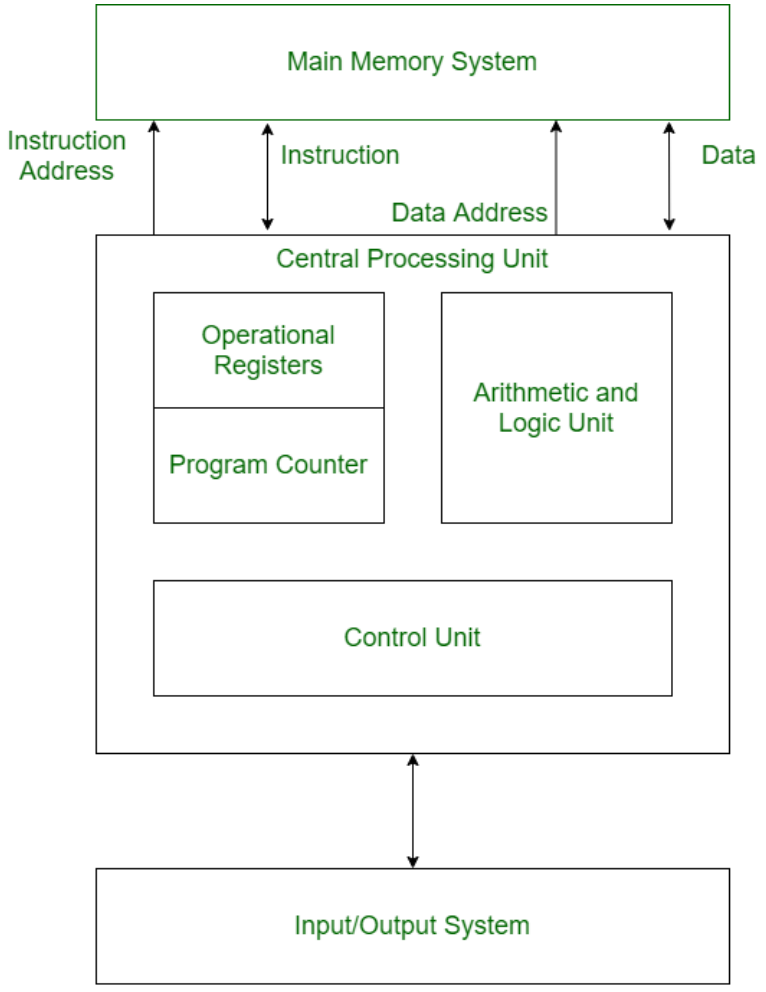
The Von Neuman Architecture

- All memory space on same bus
- Every location has unique address
- Benefits:
  - Simplifies processor design -- one memory interface
  - More reliable -- fewer things can fail
  - RAM can be used for both data and instruction storage
  - Greater flexibility in design of software (esp. real-time OS)
  - Fewer pins on the CPU for external interfacing (general purpose computers)
- Drawbacks
  - instructions and data treated the same way
  - Possible bottleneck between instruction and data fetches



Harvard Architecture

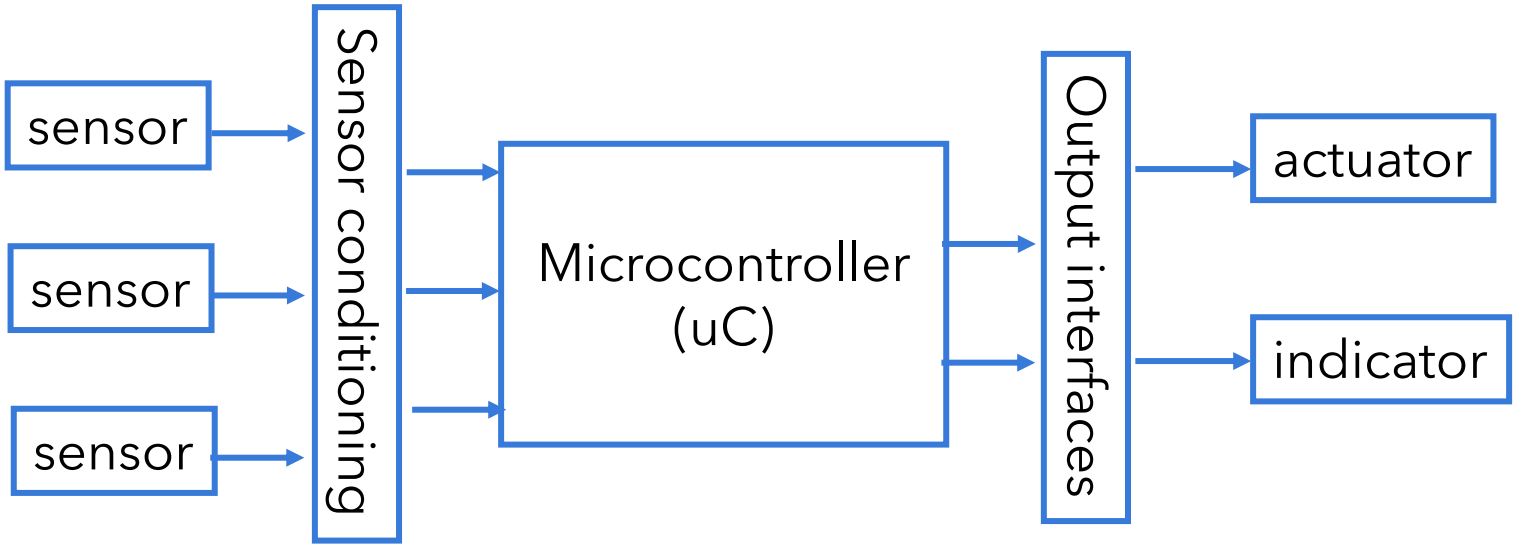
- Code and data address separate Using separate buses
- Benefits:
  - Programs execute in fewer cycles
  - Faster bootup from ROM
    - RAM is in known state
  - Memory within the same chip/die
    - No additional cost for external connection pins
  - Robust code
    - Program/data boundary cannot be crossed
    - Buffer overflow cannot cause system rooting
- Cons:
  - More on-chip lines
  - Difficult to program/code
- Example: most microcontrollers like ARM, Atmel, PIC, TI



Harvard Architecture

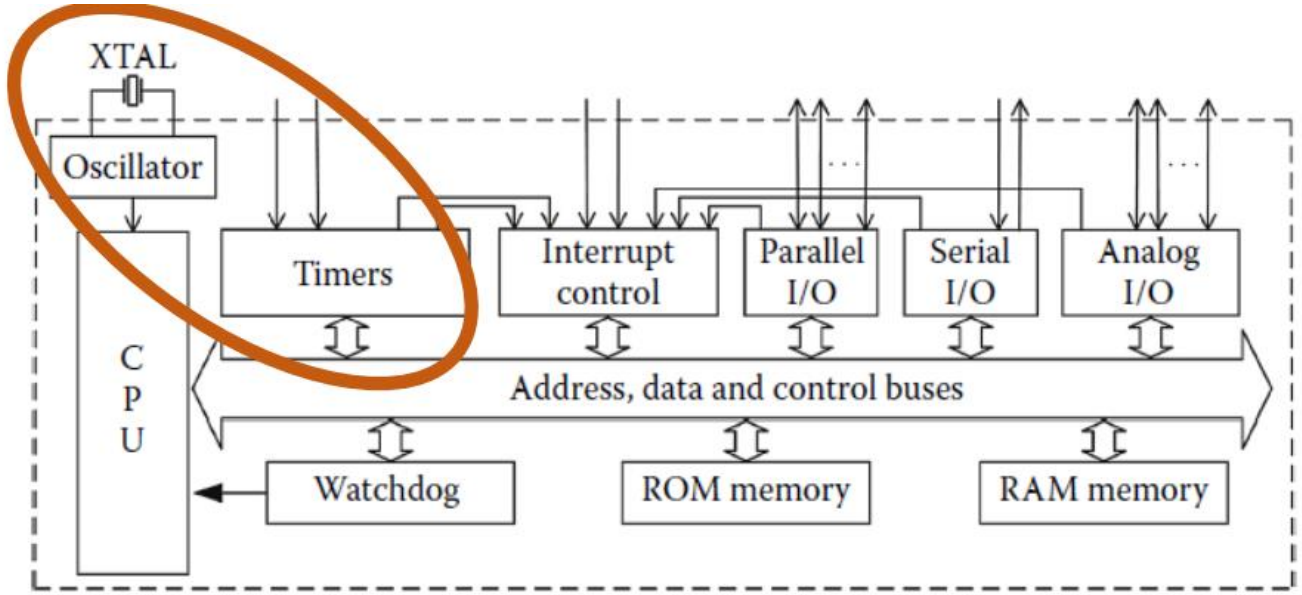
Embedded Systems

- It is a special purpose system designed to perform a few dedicated functions.
  - Small foot prints (in memory)
  - Highly optimized code
  - Cell phones, mp3 players are examples.
- The components in an mp3 player are highly optimized for storage operations. (For example, no need to have a floating point operation on an mp3 player!)



Clock measurements

- The clock signal increments the counter every 1/f seconds (resolution)
  - Generally a quartz crystal (XTAL)
- Counter: counts pulses on a general input Signal
  - e.g., count cars passing over a sensor
  - Can often configure device as either a timer or counter
- Based on counting clock pulses
  - e.g., let Clk period be 10 ns
  - And we count 20,000 Clk pulses
  - Then 200 microseconds have passed
  - 16-bit counter would count up to 65,535*10 ns = 655.35 microsec., resolution = 10 ns
- Top: indicates top count reached, wrap-around



Watchdog Timer

- A free running counter
  - Must reset timer earlier than every X time unit, else timer generates a signal
- Common use: detect failure, self-reset
- Another use: timeouts
  - e.g., ATM machine

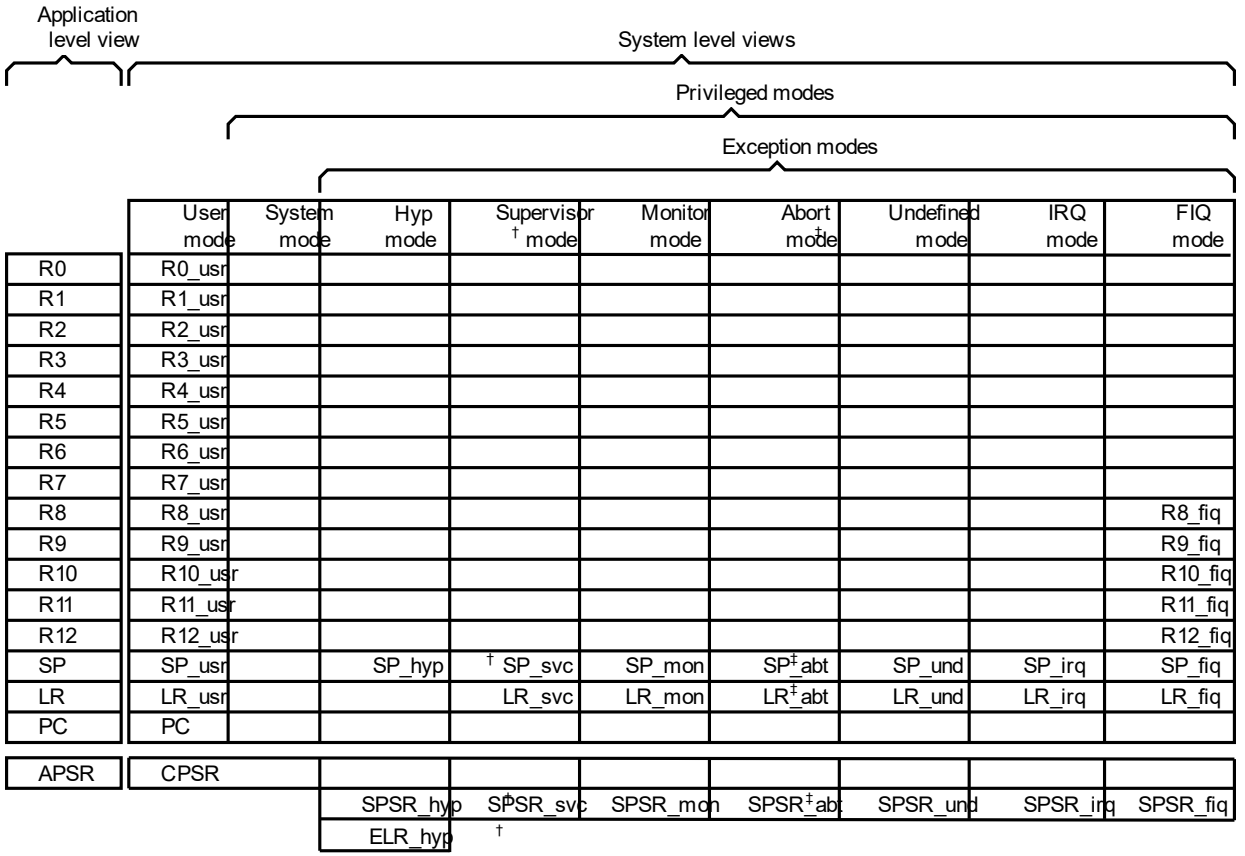
```
/* main.c */
main(){
  wait until card inserted
  call watchdog_reset_routine
 while(transaction in progress){
    if(button pressed){
      perform corresponding action
      call watchdog_reset_routine
    }

/* if watchdog_reset_routine not called every < 2
minutes, interrupt_service_routine is called */
}
watchdog_reset_routine(){
 watchdog_timer= MAX_VAL
}
void interrupt_service_routine(){
  eject card
  reset screen
}
```

13

Registers

- Special type of memory
- General use
  - Storing data variables for fast access
  - Special purpose registers
    - Instruction register
    - Status
    - Program counter
    - Stack pointer
    - Accumulator
    - Capture/Compare registers



† Hyp mode and the associated banked registers are implemented only as part of the Virtualization Extensions

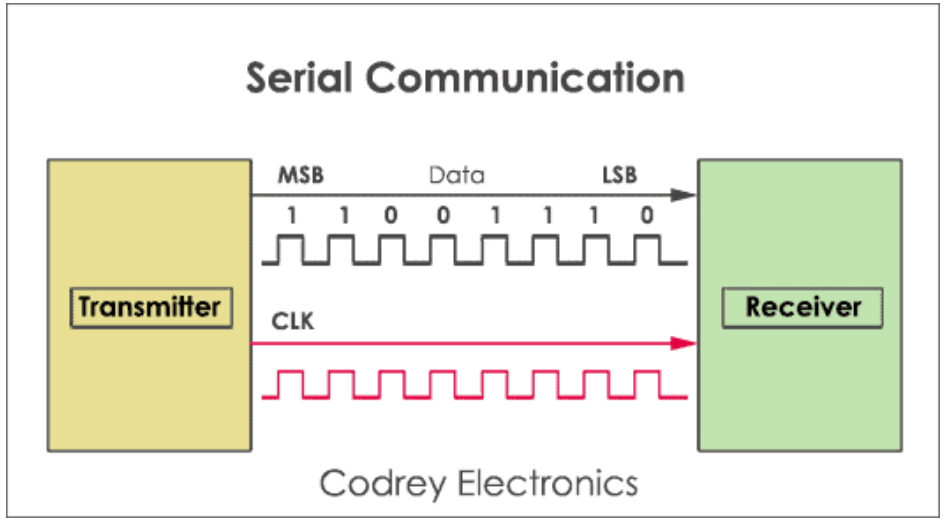
‡ Monitor mode and the associated banked registers are implemented only as part of the Security Extensions

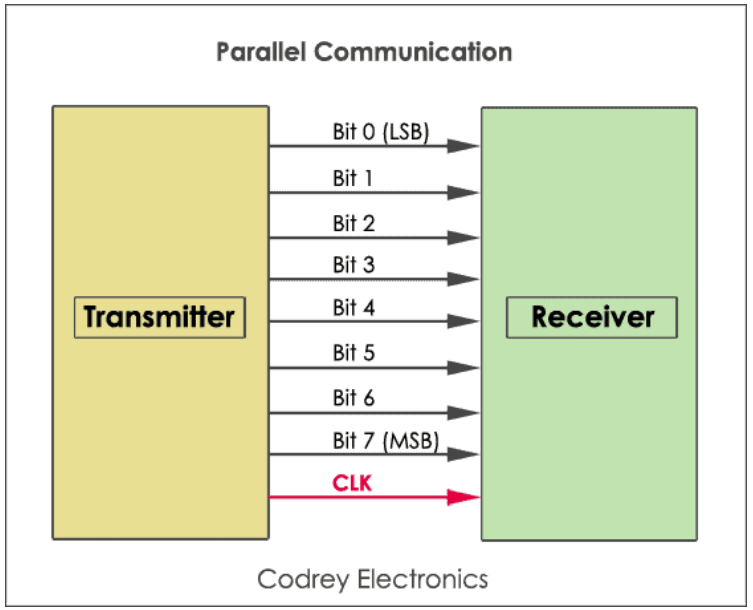
ARM Registers

- ARM have a total of 37 registers, with 3 additional registers if the Security Extensions are implemented
  - 13 general-purpose registers R0-R12.
  - One Stack Pointer (SP).
  - One Link Register (LR).
  - One Program Counter (PC).
  - One Application Program Status Register (APSR).
  - Two Supervisor mode registers for banked SP and LR.
  - Two Abort mode registers for banked SP and LR.
  - Two Undefined mode registers for banked SP and LR.
  - Two Interrupt mode registers for banked SP and LR.
  - Seven FIQ mode registers for banked R8-R12, SP and LR.
  - Two Monitor mode registers used only in Security Extension mode

Serial/Paralell Communication

- Data is in the form of binary pulses.
  - Binary One represents a logic HIGH or 5 Volts, and zero represents a logic LOW or 0 Volts.
- Serial communication can take many forms depending on the type of transmission mode and data transfer.
  - The transmission modes are classified as Simplex, Half Duplex, and Full Duplex.





Clock Synchronization

- For Serial Communication, the clock is the primary source.
  - Malfunction of the clock may lead to unexpected results.
  - The clock signal is different for each serial device
- Synchronous serial interface
  - All the devices on Synchronous serial interface use the single CPU bus to share both clock and data.
  - Due to this fact, data transfer is faster.
  - The advantage is there will be no mismatch in baud rate. Moreover, fewer I/O (input-output) lines are required to interface components.
  - Examples are I2C (inter integrated circuit) , SPI (Serial peripheral interface) etc.
- Asynchronous serial interface
  - The asynchronous interface does not have an external clock signal, and it relies on four parameters namely
    - Baud rate control
    - Data flow control
    - Transmission and reception control
    - Error control.
  - Asynchronous protocols are suitable for stable communication.
  - These are used for long distance applications.
  - Examples of asynchronous protocols are RS(Recommended standard) -232, RS-422, and RS-485.

Assignment 1

- Make comparison of 10 Microcontrollers at least one from each category.
- Comparison parameters:
  - Bus size
  - Clock speed
  - RAM
  - ROM
  - Architecture
  - Registers
  - Power consumption
  - GPIOs
  - UART
  - Other ports
- Comparison of Serial Protocols (UART, I2C, SPI, RS-485, RS-232)

Polling (Program Driven I/O)

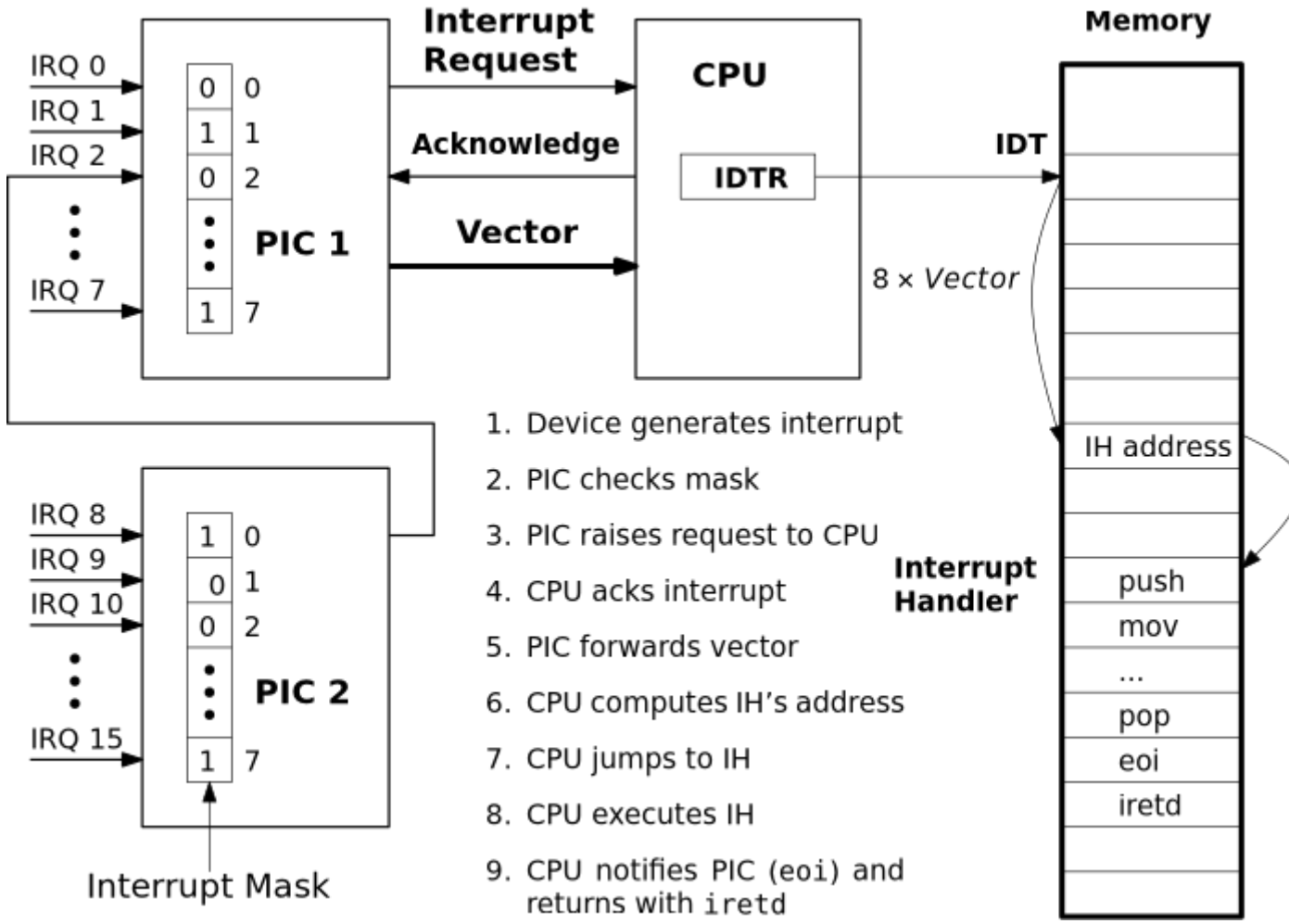
- CPU polls the device addresses and take action when needed
  - Simple to build HW, but CPU needs to poll often – inefficient and wasteful
  - Sequential program flow is maintained
- Poll for multiple events by merging local loops into larger one.
  - Leads to 'grand loop' designs
  - Works only if devices are slow compared to CPU
  - Faster devices lead to dropout of data/events
  - If devices are really slow– wastes CPU power
- Thought: Can we use hardware to do the 'scan' for change of I/O state?
  - What would be the benefit?

Interrupt Signalling

- Interrupts (Event Driven I/O)
  - Set up event, then go off and do other things until signaled
  - On signal, drop everything, service need and resume other thing
  - Preempt CPU as events dictate, but
  - Breaks sequential program flow
- Hardware/Peripheral must be able to request action from computer
- Generally a pin made to go high
  - Determining the source of interrupt and corresponding action single
- Interrupt handler, Interrupt service routine (ISR)
  - Very short, fast code
  - Implemented like a subprogram
  - All used registers must be saved and restored (Saving the context)
  - Any latency in service routine shows up in every event response.

How interrupt works?

- CPU stops doing what it was doing (executing instructions)
  - Completes execution of the instruction that is executing
  - flush the instructions currently pending execution
- Create new stack frame (after any required context switch)
  - Saves the next program address on the stack (the PC)
  - Also other status register (depends on architecture e.g. SP)
  - Effectively the 'CALL ' instruction is executed on its own!
- Jumps to interrupt routine
  - Interrupt handler or service routine (ISR)
  - Very short, fast subprogram
  - Interrupts live in real-time, often on system SW



Interrupt Service Routine (ISR) Types

- Fixed interrupt
  - ISR location for a pin built into the processor
  - If limited, have a JUMP to different place
- Vectored Interrupt
  - Peripheral provides the address while interrupting
  - Common for multiple peripherals with shared system bus
  - Requires an ack pin
- Interrupt vector tables = a compromise
  - Table in memory holding ISR address
  - Peripheral provides just index
  - Can move ISR location without changing peripheral

Common Interrupts

- ✓ Input pin state change
- ✓ Timer overflow
- ✓ Timer compare/match
- ✓ Timer capture
- ✓ UART RX char ready
- ✓ UART TX ready
- ✓ UART TX complete
- ✓ SPI transfer
- ✓ I2C transfer
- ✓ ADC conversion complete
- ✓ Watchdog

Disabling Interrupts

- Every system allows interrupts to be disabled in some way
  - Devices can be told not to interrupt
  - CPU can be told to ignore interrupts
  - In addition, the CPU can ignore some subset of interrupts
- Non-maskable interrupt
  - An interrupt that can never be turned off
    - Used for exceptional circumstances
    - Catastrophic event
    - Power failure
    - Watchdog timer interrupt!
- When an interrupt occurs while an interrupt handler is executing; two options
  - Priority based interrupts
    - only a higher priority interrupt can interrupt the handler
  - Otherwise disable all interrupts automatically whenever an interrupt handler is invoked

Issue in Interrupt Driven Control

- CPU is still involved in every data transfer
- Interrupt handling overhead is high
  - Overhead is too high for fast devices
- Assembly code is software, still slower than hardware
  - Especially when some high throughput peripherals behave in a common fashion
    - Transfer data from device to memory and vice versa
- Possible energy savings if data transfer to memory w/o interrupting processor

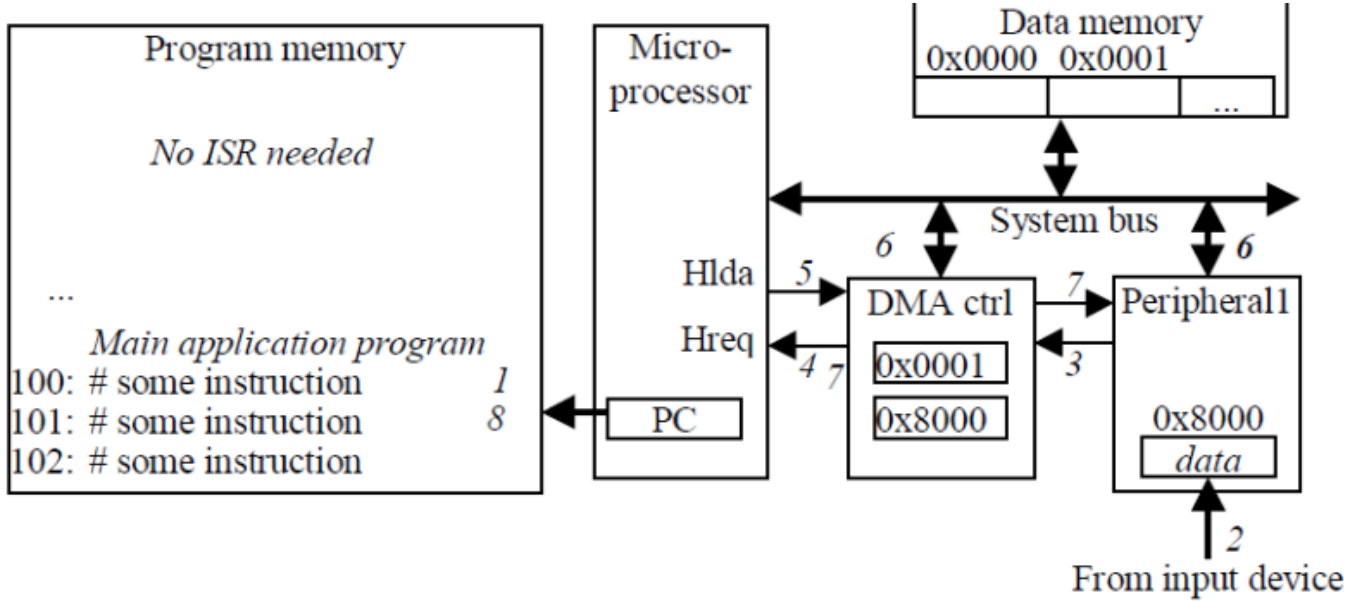
Solution: Direct Memory Access (DMA)

Direct Memory Access (DMA)

- Goal: Get data in and out of systems quickly with minimum MCU intervention
  - Allow direct peripheral-to-memory writes (and vice versa)
  - Also, possible to move data between memory locations
- How, without software and MCU intervention?
  - Using an ancillary processor (DMA controller)
- Potential problems
  - Must not interfere with MP on the bus (address/data lines)
  - Often does, of course– idea is to keep the overhead low…
- DMA Modes
  - Single one byte transfer - Slowest
  - Block Transfer: pre-programmed block of data - Fast but fixed
  - Demand: as much as device requires - Fast and flexibly
    - Keep transfer until DMAREQ(DMA Request) is high

DMA based Memory Transfer

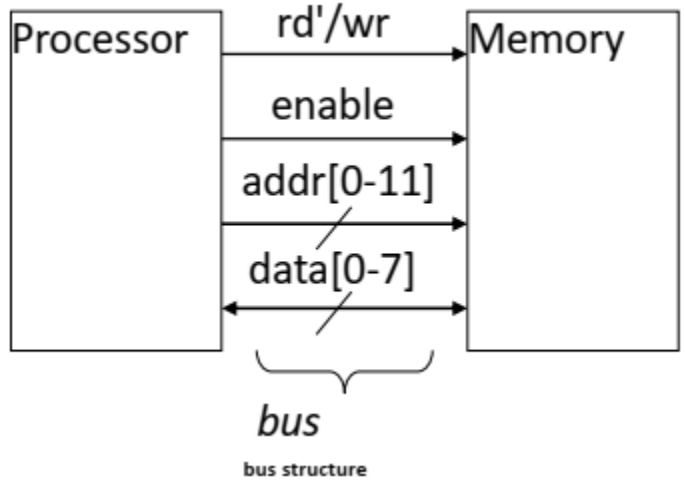
- ESP32 has DMA
  - Without DMA it can only take about 10Kbit/sec
  - ESP32 can take 10Mbit/sec from ADC(Analog to digital converter) with DMA



1. Microprocessor is executing its program, say current program counter (PC)=100.
2. Peripheral1's input device stores new data in register at 0x8000.
3. Peripheral1 asserts *Dreq* to request servicing of the new data.
4. DMA controller asserts *Hreq* to request control of the system bus.
5. Microprocessor relinquishes system bus, perhaps stopping after executing statement 100.
6. **DMA controller reads data from 0x8000 and writes that data to 0x0001 in data memory.**
7. DMA controller deasserts Hreq and completes handshake with peripheral
8. Microprocessor resumes executing its program, perhaps starting with statement 101.

Busses in Embedded Systems

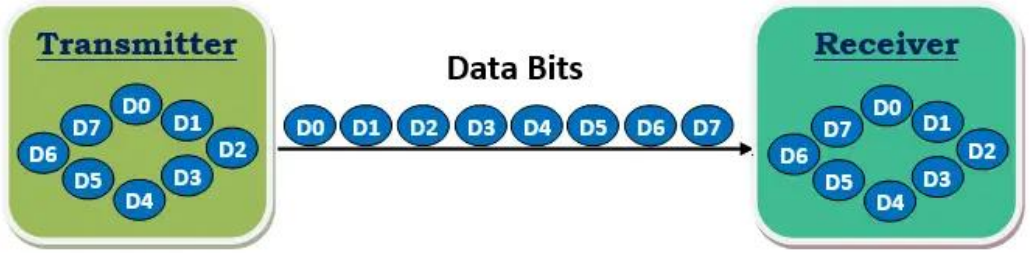
- A Bus is set of wires with a single function
  - Associated protocol: rules for communication
- Three types of Buses
  - Data Bus
    - To move actual data to and from peripherals
    - These include memory (RAM/ROM), ADC, serial port etc
  - Control bus
    - To control signal to each peripheral
    - Sometimes part of address bus, as a few special lines
    - Unidirectional
  - Address Bus
    - Unidirectional
    - signals necessary to map any memory locations of RAM & I/O



bus structure

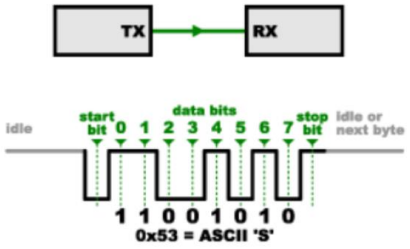
Serial Communication Protocols

- In a serial communication protocol the bits are transferred one by one over the wire.

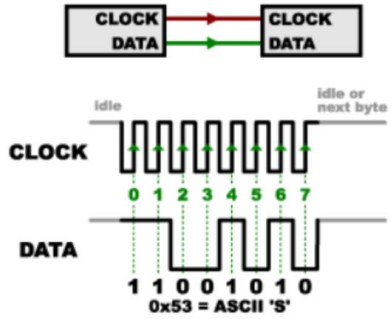


- Asynchronous:
  - In asynchronous communication protocol, sender and receive do not share a common clock.
- Synchronous
  - In a synchronous communication protocol, sender and receive share a common clock.

Asynchronous



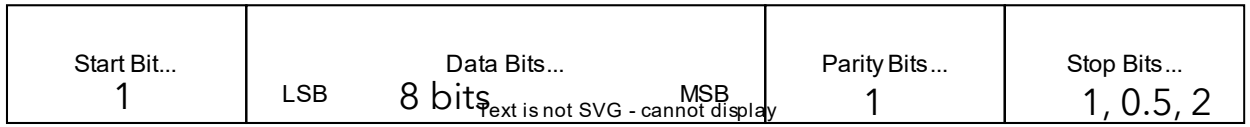
Synchronous



Universal Asynchronous(Synchronous) Receiver Transmitter (UART/USART) Protocol

- UART is a full-duplex asynchronous (synchronous) serial data transmission protocol
- RS-232 (Recommended Standard) is a serial communication transmission standard can be seen as a physical layer to UART.
  - The asynchronous variant needs minimum 3 wires to receive and transmit:
  - TX (transmitter): This is the line used by the transmitting device to send data.
  - RX(receiver): This is the line used by the receiving device to send data.
  - GND(ground): This is the ground reference or ground path that completes the electrical circuit.

- If you are configuring a UART device, you normally have the following options:
  - Baud rate
  - Data bits
  - Stop bits
  - Parity
  - Flow Control



- The baud rate (Bd) is the number of symbols per second. In UART one symbol is one bit, so the baud rate is the same as the bit rate.
- Common baud rates are 9600 bits/sec upto 115,200.
- UART Receiver and Transmitter do not share a clock, start and stop bits are used for synchronization.

Serial Programming in Aurdino

Serial Read

```
void setup() {
  pinMode(8, INPUT_PULLUP); // set push button pin as input
  pinMode(13, OUTPUT);      // set LED pin as output
  digitalWrite(13, LOW);    // switch off LED pin
  Serial.begin(9600);       // initialize UART with baud rate
of 9600 bps
}
void loop() {
  if(Serial.available()) {
    char data_rcvd = Serial.read();   // read one byte from
serial buffer and save to data_rcvd
    if(data_rcvd == '1') digitalWrite(13, HIGH);  // switch
LED On
    if(data_rcvd == '0') digitalWrite(13, LOW);   // switch
LED Off
  }
}
```

Serial Write

```
void setup() {
  pinMode(8, INPUT_PULLUP); // set push button pin as input
  pinMode(13, OUTPUT);      // set LED pin as output
  digitalWrite(13, LOW);    // switch off LED pin
  Serial.begin(9600);       // initialize baud rate of 9600
 }
void loop() {
  if (Serial.available()) {
    char data_rcvd = Serial.read();   // read one byte
    if (data_rcvd == '1') digitalWrite(13, HIGH); // LED On
    if (data_rcvd == '0') digitalWrite(13, LOW);  // LED Off }
if (digitalRead(8) == HIGH) Serial.write('0');    // send '0'
to serial if button is not pressed.
else Serial.write('1');                           // send the
char '1' to serial if button is pressed.
}
```

UART Bootloader Protocol

- GUARD: The Guard value must be a constant value of 0x5048434D
  - This value provides protection against spurious commands
  - Bootloader always checks for the Guard value at start of packet reception and proceeds further accordingly
- Data Size: This field indicates the number of data bytes to be received
- Command: Indicates the command to be processed. Each command is of 1 Byte width

Command Type	Command Code	Description
Unlock	0xA0	Used to calculate application start address and end address
Data	0xA1	Used to send the image data
Verify	0xA2	Used to verify the image data sent and programmed
Reset	0xA3	Used to trigger a reset to run the application
Bank Swap and reset	0xA4	Used to Swap the bank and trigger a reset to run the application
Device Configuration	0xA5	Used to send the device configuration bits (Fuse Settings)
Read Version	0xA6	Used to read the bootloader version running

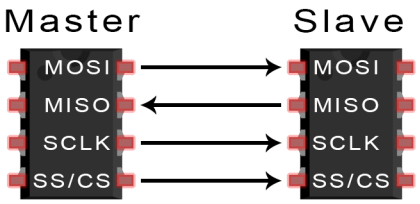
https://microchip-mplab-harmony.github.io/bootloader/GUID-8828D474-F227-4FE0-88EE-135AA591750F.html

Intra-Integrated Circuit (I2C) Protocol

- I2C is a synchronous, serial communication protocols with multiple controllers and targets on the same bus.
- It is mainly used for intra-board connections, meaning for short-distance communication on the same board between the microcontroller and peripheral integrated circuits (ICs).
- Two wires carry the information from the controller to the target, or vice-versa:
- Serial clock-This line carries the clock signal that synchronize the data transfer
- Serial data- The bidirectional lines carries the actual data.
- The controller can also be the receiver, but the controller always initiates the transfer and generates the clock signal.
  - controller-transmitter transmits to target-receiver
  - controller-receiver receives from target-transmitter
- The controller can change the direction of data within a transfer. The capability is referred to as the combined format.

Serial Peripheral Interface (SPI)

- It's a synchronous protocol and has two types of devices namely master and slave
- Types of signals
  - Serial Clock_ set by master
  - Master In Slave Out (MISO
  - Master Out Slave In (MOSI)
  - Slave Select
    - Line for the master to select which slave to send data to.
- Prons
  - No start and stop bits, transmission without interruption
  - No complicated slave addressing system like I2C
  - Multiple devices can communicate unlike UART
- Cons
  - Uses four wires (I2C and UARTs use two), No acknowledgement
  - No form of error checking



Firmware

What is Firmware?

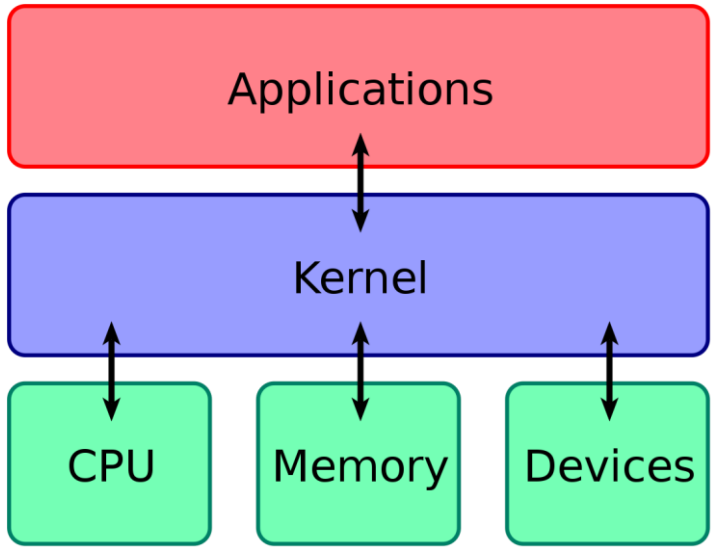
- Firmware is a software permanently embedded into a device's read-only memory, which provides instructions on how the device is supposed to operate.
- Simply to say firmware is software that is programmed on a hardware device to perform its function.
- Firmware components
  - Filesystems---Manage how data is stored and retrieved by the firmware.
  - Kernel---interact with the hardware and manage system resources
  - Bootloader---SOFTWARE that initialize the hardware and loads the OS into during boot process.
- Different types of OS to build firmware
  - embedded Linux
  - embedded Windows
  - Windows IoT core
  - Real Time Operating Systems (RTOS)-resource constrained applications

Filesystems in Firmware

- Some of the common files systems of the IoT Devices are as follows
  - Squashfs: [https://www.kernel.org/doc/html/v5.18/filesystems/squashfs.html]
    - Squashfs is a compressed read-only file system for Linux.
    - It compresses files, inodes and directories, and supports block sizes from 4 KB up to 1 MB
    - Several compression algorithms are supported zlib, lz4, lzo, or xz compression
    - It is used by the Live CD versions of Arch Linux, Debian, Fedora, openSUSE, Ubuntu, Kali Linux and on embedded distributions such as the OpenWrt and DD-WRT router firmware.
  - Cramfs (compressed RAM/ROM file system)
    - It uses the zlib routines to compress a file one page at a time, and allows random page access
    - File sizes are limited to less than 16MB but Maximum filesystem size is a little over 256MB.
  - JFFS2
    - JFFS2 (Journalling Flash File System Version 2) is a log-structured file system designed for use on flash devices in embedded systems.
    - it places the filesystem directly on the flash chips. JFFS2 was developed by Red Hat
  - Yaffs2 (Yet Another flash file system
  - Ext2 (Second extended file system)

Kernel

- The kernel is a core component of an operating system.
- It serves as the main interface between the computer's physical hardware and the processes running on it.
- The kernel enables multiple applications to share hardware resources by providing access to CPU, memory, disk I/O, and networking.



Bootloader

- A bootloader is responsible for loading and initializing an operating system kernel.
- Data of an operating system must be loaded into the working memory during device start-up. This is made possible by a so-called bootloader.
- immediately after a device starts, a bootloader is generally launched by a bootable medium like a hard drive, a CD/DVD, or a USB stick.
- The boot medium receives information from the computer's firmware (e.g. BIOS) about where the bootloader is. The whole process is also described as "booting".
- Typical tasks performed by the bootloader include
  - selecting the and loading an initial file system to RAM.
  - The filesystem on RAM contains a minimal environment to mount the root filesystem and start the normal boot process.
  - It initializes the hardware by choosing the correct image. Usually placed in a part of flash that is protected from accidental deletion or data corruption.

Busybox for embedded systems

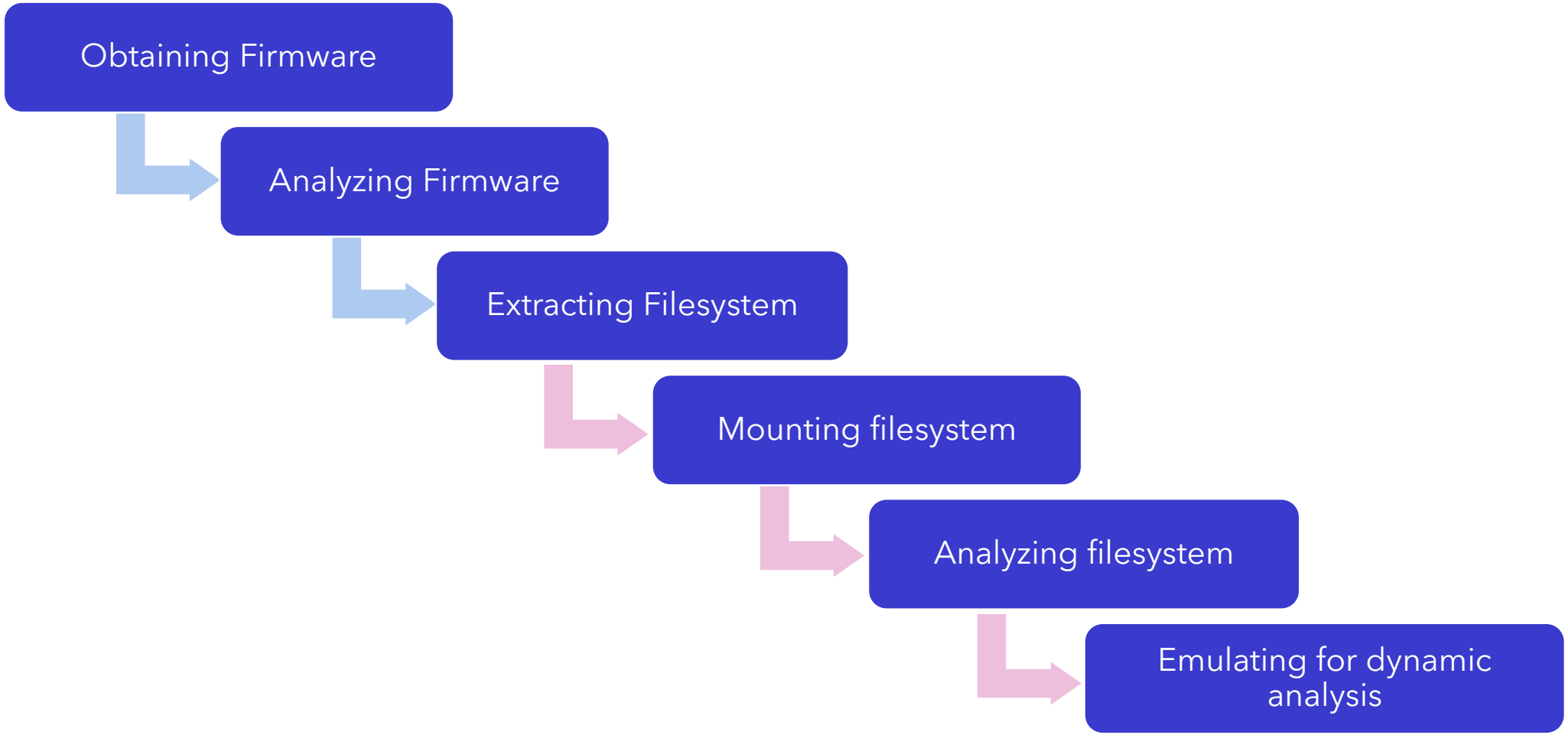
- It combines tiny versions of many common UNIX utilities into a single small executable.
- BusyBox provides a fairly complete environment for any small or embedded system.
- BusyBox has been written with size-optimization and limited resources in mind. It is also extremely modular so you can easily include or exclude commands (or features) at compile time. This makes it easy to customize your embedded systems.
- To create a working system, just add some device nodes in /dev, a few configuration files in /etc, and a Linux kernel.
- BusyBox has a minimal init program that uses a configuration file, /etc/inittab, to define rules to start programs at boot up and to stop them at shutdown.

Firmware Analysis

- The most common goals of an assessor will be to locate the following:
  - Passwords
  - API tokens
  - API endpoints (URLs)
  - Vulnerable services
  - Backdoor accounts
  - Configuration files
  - Source code
  - Private keys
  - How data is stored

Methodologies for Analysis

- Following is a list of the basic methodologies for analyzing IoT firmware:



Firmware Analysis Tools: BinWalk

- Binwalk is an open-source tool for analyzing, reverse engineering and extracting firmware images.
  - Used to reverse engineer a firmware image to understand how it works.
  - Used to reverse engineer binaries inside filesystem images to look for vulnerabilities.
  - Can extract files from the image and search for backdoor passwords or digital certificates.
  - Can identify opcodes for a variety of CPU architectures.
- The main feature of binwalk is its signature scanning.

```
$ binwalk
Binwalk v2.2.0 Craig Heffner, ReFirmLabs https://github.com/ReFirmLabs/binwalk
Usage: binwalk [OPTIONS] [FILE1] [FILE2] [FILE3] ...
Signature Scan Options:
-B, --signature Scan target file(s) for common file signatures
-R, --raw=<str> Scan target file(s) for the specified sequence of bytes
-A, --opcodes Scan target file(s) for common executable opcode signatures
Extraction Options:
    -e, --extract               Automatically extract known file types
    -D, --dd=<type:ext:cmd>     Extract <type> signatures, give the files an extension of <ext>, and execute <cmd>
    -M, --matryoshka            Recursively scan extracted files
```

BinWalk Analysis

```
$ binwalk --signature --term archer-c7.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
21876         0x5574          U-Boot version string, "U-Boot 1.1.4-g4480d5f9-dirty (May
                              20 2019 - 18:45:16)"
21940         0x55B4          CRC32 polynomial table, big endian
23232         0x5AC0          uImage header, header size: 64 bytes, header CRC:
                              0x386C2BD5, created: 2019-05-20 10:45:17, image size:
                              41162 bytes, Data Address: 0x80010000, Entry Point:
                              0x80010000, data CRC: 0xC9CD1E38, OS: Linux, CPU: MIPS,
                              image type: Firmware Image, compression type: lzma, image
                              name: "u-boot image"
23296         0x5B00          LZMA compressed data, properties: 0x5D, dictionary size:
                              8388608 bytes, uncompressed size: 97476 bytes
64968         0xFDC8          XML document, version: "1.0"
78448         0x13270         uImage header, header size: 64 bytes, header CRC:
                              0x78A267FF, created: 2019-07-26 07:46:14, image size:
                              1088500 bytes, Data Address: 0x80060000, Entry Point:
                              0x80060000, data CRC: 0xBB9D4F94, OS: Linux, CPU: MIPS,
                              image type: Multi-File Image, compression type: lzma,
                              image name: "MIPS OpenWrt Linux-3.3.8"
78520         0x132B8         LZMA compressed data, properties: 0x6D, dictionary size:
                              8388608 bytes, uncompressed size: 3164228 bytes
1167013       0x11CEA5        Squashfs filesystem, little endian, version 4.0,
                              compression:xz, size: 14388306 bytes, 2541 inodes,
                              blocksize: 65536 bytes, created: 2019-07-26 07:51:38
15555328      0xED5B00        gzip compressed data, from Unix, last modified: 2019-07-26
                              07:51:41
```

Assignment 2

- Download any firmware and extract the following information using binwalk
  - Can a firmware image and search for file signatures to identify and extract filesystem images, executable code, compressed archives, bootloader and kernel images, file formats like JPEGs and PDFs, and configuration files
  - Identify opcodes for a variety of CPU architectures.
  - Decompress filesystem images to search for specific password files (passwd, shadow, etc) and try to break password hashes.

https://sergioprado.blog/reverse-engineering-router-firmware-with-binwalk/