Lecture 2: Introduction to Embedded Systems and Firmware

Instructor: Mehmoona Jabeen
Email: mehmoona.jabeen@mail.au.edu.pk
Department of Cyber Security, Air University
Course: Internet of Things Security – CYS, Spring 2024

Lecture Outline

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

Microcontroller vs Microprocessor

- Microprocessor = only CPU; Microcontroller = CPU + Memory + I/O in one chip
- Microprocessors → Personal Computers; Microcontrollers → Embedded Systems
- Microprocessor uses **external** bus; Microcontroller uses **internal** bus
- Microprocessors: Von Neumann model; Microcontrollers: Harvard architecture
- Microprocessor: complex, expensive, many instructions
- Microcontroller: simple, cheap, fewer instructions

Types of Microcontrollers

- **PIC (Peripheral Interface Controller)**
  - For electronics, robotics, etc.
  - Made by Microchip Technology (hardware computing architecture)
- **ARM (Advanced RISC Machine)**
  - Most popular in digital embedded systems
  - Preferred by industry for features
- **8051 Microcontroller**
  - Created by Intel (1981)
  - 8-bit, 40-pin DIP, 4KB ROM, 128B RAM, 2x 16-bit timers, 4x 8-bit ports
- **AVR (Alf and Vegard's RISC)**
  - Developed by Atmel
  - Modified Harvard architecture, RISC
- **MSP (Mixed Signal Processor)**
  - From Texas Instruments
  - 16-bit CPU, low cost and low power

Microcontroller Architecture

- Vendors:
  - Develop own CPU or license IP
- Targeted toward small set of applications
- Compared to desktop CPU:
  - Smaller, reduced instruction set, lower power, lower frequency

Microcontroller Specs Variation:

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

Von Neumann Architecture

- Single memory bus
- Same address space for instructions/data
- **Pros:**
  - Simple design
  - Reliable, flexible
  - RAM for data & code
  - Fewer CPU pins needed
- **Cons:**
  - Instructions/data treated the same
  - Bottlenecks

Harvard Architecture

- Separate buses for code and data
- **Pros:**
  - Faster execution & boot
  - More secure (no buffer overflow)
  - Code robustness
- **Cons:**
  - More on-chip lines
  - More difficult to code
- **Used in:** ARM, Atmel, PIC, TI

Embedded Systems

- Special-purpose systems with dedicated functions
- Features:
  - Small memory footprint
  - Highly optimized code
- Examples: Cell phones, mp3 players
  - Components tailored to need (e.g., no float point in mp3)

Clock Measurements

- Clock increments counter every 1/f sec
- Uses quartz crystal (XTAL)
- Can act as **timer** or **counter**
- **Example:**
  - Clk = 10ns
  - Count = 20,000 → 200µs
  - 16-bit counter: max = 655.35µs

Watchdog Timer

- Free running counter
- Must reset before timeout
- Use: detect failure, system self-reset
- E.g., ATM timeout

Registers

- **General use:** fast variable storage
- **Special purpose:**
    - Instruction register
    - Status
    - Program counter (PC)
    - Stack pointer (SP)
    - Accumulator
    - Capture/Compare registers

ARM Registers

- Total: 37 (40 with Security Extensions)
- 13 General-purpose (R0–R12)
- 1 Stack Pointer (SP)
- 1 Link Register (LR)
- 1 Program Counter (PC)
- 1 Application Program Status Register (APSR)
- Multiple banked SP/LR for modes:
    - Supervisor, Abort, Undefined, Interrupt, FIQ
- 2 Monitor mode (Security Extension only)

Serial/Parallel Communication

- Binary pulses: 1 = HIGH (5V), 0 = LOW (0V)
- Modes:
    - Simplex
    - Half Duplex
    - Full Duplex

Clock Synchronization

- Clock critical in serial communication
- **Synchronous serial Interface:**
  - Shared bus for clock/data
  - 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.
  - Faster, fewer I/O lines
  - No baud rate mismatch
  - E.g., I2C, SPI

- **Asynchronous serial Interface:**
  - does not have an external clock signal
  - Relies on four parameters:
    - Baud rate control
    - Data flow control
    - Transmission and reception control
    - Error control
  - Good for long distances
  - Asynchronous protocols are suitable for stable communication.
  - E.g., RS-232, RS-422, RS-485

Assignment 1 (For Practice)

- Compare 10 Microcontrollers (one from each category)
- Parameters:
  - Bus size, Clock speed, RAM, ROM
  - Architecture, Registers
  - Power, GPIOs, UART, Other ports
- Compare 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 maintained
- Not suitable for fast devices
- Works for slow devices
- Suggestion: use hardware for scanning

Interrupt Signaling

- 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
- Set event, do other tasks → CPU is signaled
- Needs hardware support
- ISR (Interrupt Service Routine):
    - Short subprogram
    - Save & restore all used registers
    - Context switch
- Latency impacts performance

Interrupt Working

- CPU finishes current instruction
- Flush pending instructions
- Saves PC, status regs
- CALL to ISR auto executed
- Jumps to ISR
- Fast, short code
- Used in real-time systems

ISR Types

- **Fixed Interrupt:** built-in pin, uses jump
- **Vectored Interrupt:** peripheral gives address
- **Interrupt Vector Table:**
    - Table holds ISR addresses
    - Peripheral gives index

Common Interrupts

- Input pin change
- Timer overflow/compare/capture
- UART RX/TX
- SPI/I2C transfer
- ADC complete
- Watchdog

Disabling Interrupts

- Can disable interrupts per device or CPU-wide
- **Non-maskable Interrupts:**
    - Cannot be disabled
    - For catastrophic events (e.g., power failure)
- **While ISR running:**
    - Priority interrupts → allow higher priority
    - Or disable all temporarily

Issues with Interrupts

- CPU still handles every transfer
- High overhead for fast devices
- Software slower than hardware
- Repeated tasks like memory transfers are inefficient

Direct Memory Access (DMA)

- Fast data movement with minimal CPU help
- Transfers:
    - Peripheral ↔ Memory
    - Memory ↔ Memory
- Uses DMA Controller
- **Modes:**
    - Single-byte: slow
    - Block: fast but fixed
    - Demand: fast + flexible (while DMAREQ = high)

DMA Example: ESP32

- Without DMA: ~10Kbps
- With DMA: ~10Mbps (from ADC)

Buses in Embedded Systems

- **Bus = wires + communication rules**
- **Types:**
    - **Data Bus:** for transferring data
    - **Control Bus:** control signals (unidirectional)
    - **Address Bus:** maps memory locations (unidirectional)

Serial Communication Protocols

- Data sent bit-by-bit
- **Asynchronous:** no shared clock
- **Synchronous:** shared clock

UART/USART Protocol (Universal Asynchronous Receiver Transmitter)

- UART is a full-duplex asynchronous/synchronous serial communication protocol.
- RS-232 is a physical layer standard often used with UART.
- **Minimum wires (asynchronous mode):**
    - `TX` (Transmitter): Sends data.
    - `RX` (Receiver): Receives data.
    - `GND` (Ground): Completes the electrical circuit.

UART Configuration Options:

- Baud rate (e.g., 9600 – 115200 bits/sec)
- Data bits (usually 8)
- Stop bits (1, 0.5, 2)
- Parity (for error checking)
- Flow control

UART uses **start and stop bits** for synchronization since there is no shared clock.

UART Bootloader Protocol Format

Field	Size	Description
Guard	4 Bytes	Constant `0x5048434D`, validates packet
Data Size	4 Bytes	Number of data bytes in packet
Command	1 Byte	Type of command
Data 0-N	4 Bytes each	Payload data

The bootloader **starts by checking the Guard value** before processing.

I2C (Intra-Integrated Circuit) Protocol

- I2C is a Synchronous serial communication protocol with multiple **controllers** and **targets** on the same bus.
- Typically used **within a single circuit board**.
- **Two lines used:**
  - SCL (Serial Clock)
  - SDA (Serial Data - bidirectional)

Data Flow Modes:

- Controller-Transmitter → Target-Receiver
- Controller-Receiver ← Target-Transmitter
- **Combined format** allows changing direction mid-transfer.

SPI (Serial Peripheral Interface)

- Synchronous master-slave protocol.
- **Lines used:**
  - SCLK: Clock (from master)
  - MISO: Master In Slave Out
  - MOSI: Master Out Slave In
  - SS: Slave Select (Line for the master to select which slave to send data to.)

Pros:

- Fast, uninterrupted transmission
- No need for start/stop bits
- Multiple devices supported

Cons:

- Needs 4 wires
- No built-in error checking or acknowledgment

Firmware Basics

- Firmware is permanent software embedded in hardware to control device functions.

Key Components:

- **Filesystem**: Organizes data storage.
- **Kernel**: Manages system resources and hardware.
- **Bootloader**: Initializes hardware and loads OS.

Popular Embedded OS for Firmware:

- Embedded Linux
- Embedded Windows
- Windows IoT Core
- Real-Time OS (RTOS)

Firmware Filesystems

Filesystem	Description
SquashFS	Compressed, read-only, supports multiple compression algorithms. Used in routers and live Linux distros.
CramFS	Compressed per page. Max file size < 16MB, max FS ~256MB.
JFFS2	Log-structured FS for flash chips. Directly placed on flash.
YAFFS2	Another flash FS optimized for NAND.
Ext2	Standard Linux FS, used in some embedded systems.

Kernel (in Firmware)

- The kernel is the bridge between hardware and software.
- Manages CPU, memory, I/O, and network access.

Bootloader

- First software that runs at startup.

- Loads the OS into memory.
- Tasks include:
    - Hardware initialization
    - Mounting a temporary filesystem in RAM
    - Passing control to the main OS

BusyBox (for Embedded Systems)

- Combines multiple UNIX utilities in one binary.
- Highly modular and lightweight.
- Commonly used in IoT and embedded Linux systems.
- Needs:
    - Device nodes in `/dev`
    - Config files in `/etc`
    - Linux kernel

Firmware Analysis Goals

- Find critical components like:
    - Passwords
    - API tokens and endpoints
    - Backdoors
    - Configuration files
    - Source code
    - Private keys
    - Data storage methods

Firmware Analysis Methodology

1. **Obtain firmware**
2. **Analyze firmware**
3. **Extract filesystem**
4. **Mount filesystem**
5. **Analyze files**
6. **Emulate for dynamic analysis**

Binwalk Tool (Firmware Analysis)

- Open-source tool for firmware reverse engineering.
- Functions:
    - Scan for file signatures
    - Extract file types and compressed data
    - Identify CPU opcodes
    - Look for passwords, certificates, and vulnerabilities

Features:

- `--signature`: Scan for known file types
- `--extract`: Automatically extract
- `--matryoshka`: Recursively scan extracted files

Assignment 2 Tip

Use Binwalk to extract and analyze firmware:

- Look for compressed archives, bootloaders, kernels.
- Search for password files (`passwd`, `shadow`), try to crack hashes.
- Identify opcodes and embedded file formats (JPEG, PDF, etc.).