ECE342: Computer Hardware

Arnav Patil

University of Toronto

Contents

1 Introduction

1.1 Embedded Systems

- Much more common than what we call big computers.
- More formal definition – any physical system that employs computer control for a specific purpose.
- Internet-of-things (IoT) also a huge part of embedded systems.

1.2 Basic Embedded Peripherals

- Peripherals access each other through shared memory bus, components are generally on chip.
- We can also sue off-chip modules – connect buses to each other by using a bus interfaces.

2 Basic Peripherals

2.1 General Purpose Input/Output (GPIO)

- Single wire connected to CPU – reads logic to interface with the system bus.
- Basis of GPIO is a tri-state buffer, floating term used commonly in place of Hi-Z.
- GPIOs can be configured in different modes:
  - Input, Output
  - Analog
  - Alternate function
- Input/output can be further connected as:
  - Pull-up/pull-down – a resistor connects to VDD/GND to drive a weak 1 or 0.
  - Open-drain – for sinking current (accept current)
  - Push-pull – for sourcing current (output current)
- Analog – don't touch input, just read the analog signal.
- Alternate function – read as a 1-bit digital signal.

2.2 Measuring Time

- Using SysTick to generate periodic interrupts
  - Counts from $N - 1$ and generates an interrupt when it reaches 0
  - At 0, loads value again from SysTick reload register and counts down again
  - SysTick does not actually stop when the processor is halted
- Uses a 16-bit register as main timer – at 100 MHz means only 0.655 ms.
- Prescalar divides the clock to make a slower one – clkgen module.

2.3 Analog-Digital Converters

- Two stages – sample & hold then quantify.
- Will always be some quantization error due to finite sampling rate.
  - ADC resolutiion – number of bits
  - Sampling rate – per second samples
- Flash ADC – directly quantizes output, outputs then passed to a priority encoder.
  - Pro – can run at very high speeds
  - Con – requires a lot of hardware
- Successive approximation (SAR) – use binary search algorithm
  - Starting from MSB check is input is higher or lower and set each bit accordingly.
  - Capacitor used to take a snapshot in time.
  - N bits needs N steps, natural tradeoff between resolution and speed

2.4 Digital-Analog Conversion

- Binary-weighted DAC, set of N registers each is $R/2^i$.
- Each bit of digital inputs connects/disconnects one of the resistors.

$$V_o = -\left[\frac{V_A}{R_1} + \frac{V_B}{R_2} + \frac{V_C}{R_3}\right] \cdot R_4$$

- Two issues with the binary-weighted DAC
  - Output is negative – fixed by adding another inverting buffer
  - Output magnitude $> V_{DD}$, divide all resistors by 2
- R-2R DAC only uses two resistor values, easier to manufacture

$$V_o = \frac{V_A + 2V_B + 4V_C + 8V_D}{16}$$

2.5 Pulse-Width Modulation (PWM)

- Using DACs to produce specific analog signals can be quite slow
- PWM refers to quickly turning a signal ON/OFF so the average signal has a lower current
- ON time refers to duty cycle – to set any voltage, just set the duty voltage correctly

3 Binary Multiplication and Division

- Multiplication works exactly the same as in decimal
- Signed multiplication:
  - If multiplier is negative – sign extend every partial product to M+N
  - If multiplicand is negative, use 2's complement for just the last row
- Division is also the exact same
- Division is a lossy calculation – HW people hate it, integer division truncates the fractional part
- Compiler will optimize by replacing with a constant value

4 Fixed-Point Representation

- One easy way to represent non-integer values is just interpret it differently.
- There is no actual point stored in the number, it is on the programmer to remember it
- Like ADCs, we see some error introduced with fixed point quantization
  - We need a way to quantify the error in our representation
  - Format to communicate integer bits vs decimal bits

5 Floating-Point Representation

5.1 Biased Exponent

- Biased = actual + exp. bias (bias 127 for single-point FP)

5.2 Exceptions

Sign	**Exponent**	**Fractional**	**Number**
0 or 1	all 0	0	$+/-0$
0 or 1	all 0	$\neq 0$	Subnormal
0 or 1	all 1	0	$+/-\infty$
0 or 1	all 1	$\neq 0$	NaN

Table 1: Special Cases of FP Representation

5.3 Underflow and Overflow

- Underflow – if value is less than resolution, then we can't represent it properly
- Overflow – value exceeds the largest representable value
- Both lead to problems if not handled correctly

6 Buses

- A bus is a bundle of wires and a protocol
- Throughput – (data width x bits) - overhead
- Use controller and device terminology – move away from master and slave
- Buses defines what controller and device should do
- Controller initiates data transfer and device responds to initiation request

6.1 Types of Buses

- Serial vs parallel depends on the number os bits in transit at one point
- Sync vs async – separate clock and data signals or no clock
- On-chip vs off-chip – on connects cores within the device, and off connects modules on a PCB

6.2 Advanced Microcontroller Bus Architecture (AMBA)

- Widely used specs for buses in embedded systems
- Advanced eXtensible Interface (AXI) – high-speed components such as out-of-order CPUs and DRAM memory
- Advanced High-Performance Bus (AHB) – connects lower-speed in-order CPUs and SRAM memory
- Advanced Peripheral Bus (APB) – connects slow peripherals such as UART, timers, GPIOs, with the microcontroller
- Separate different tasks into 5 channels, each happens on its own time, writes and reads do their own things
- Supports multiple outstanding transactions
- AXI is considered full-duplex, device and controller can both talk to each other at the same time.
- APB and AHB is considered half-duplex, only one of device and controller can be talking at any point

7 Wired Protocols

7.1 I2C (Inter-Integrated Circuit)

- 2-wire serial protocol
  - Serial clock line (SCL)
  - Serial data line (SDA)
- Both lines use open drain configuration, unused bus reads as a weak 1
- Depending on the function, an I2C device can both transmit and receive or just either
- Only controllers create the clock, but devices can override this, called 'clock stretching'
- I2C protocol supports different speed modes
- Faster speeds require more complex hardware

7.1.1 I2C Tranmission

- Each byte is sent one bit at a time, MSB first
- SDA only changes when SCL is low, ensuring SDA is stable on both edges of SCL
- To start, SDA is pulled low
- To stop, SDA is released and floats high
- When another device sees that the line is low, it will not transmit (bus in use)
- Important that no device should hold the bus high, else nobody can use it

7.1.2 I2C Burst Transfer

- We can send as many bytes in a single transmission as we want
- Similarly, with chaining, we can also mix reading and writing multiple locations in a single transmission
- However, doing this means no one else can use the bus

7.1.3 I2C Clock Stretching

- We sometimes cannot guarantee that the device will be able to reply with a fixed latency
- The device might be busy and need some more time, so it can stretch the clock to cause the controller to wait
- The protocol does not specify a limit on how long the clock can stretch, but we should keep this down

7.2 I2S (Inter-IC Sound)

- Simple point-to-point protocol designed specifically for transmitting PCM audio data
- Pulse code modulation (PCM) is binarized version of audio output, basically, output from an ADC
- Signals:
  - Serial clock line (SCK)
  - Serial data line (SD) usually only one but there can be two to support full-duplex
  - Word select (WS)
  - Master clock (MCLK) an additional reference clock for high-fidelity audio, typically 256x or 512x the sample rate
  - Data is also sent MSB first, each bit is synchronized

7.2.1 Audio Sampling

- The same as ADC sampling we saw earlier
- Sampling rate – number of samples taken per second
- Audio bit depth – number of bits per sample

$$F_{SCK} = \text{ sample rate } \times \text{ bits per channel } \times \text{ number of channels}$$

7.3 SPI (Serial Peripheral Interface)

- Higher-throughput interface than I2C
- Dedicated connection to each device
- Uses 4 wires
  - Serial clock line (SCLK)
  - Main in, sub out (MISO)
  - Main out, sub in (MOSI)
  - Sub select (SS) (active low)
- There are multidrop and daisychain ways to connect devices
- Controller selects a device by setting SSn low, then data is sent 1 bit at a time, MSB first
- No limit on how many bits can be sent at a time, but many devices support 8 bits for consistency

7.3.1 Limitations of SPI

- Limitmed to a single controller per bus
- In multidrop, devices are limited by the number of SS signals
- Can be slow if devices require different modes
- Both controller and deice are always sending information
- No ACK/NACK, so can't be sure if data is even getting to the right place

7.4 UART (Universal Asynchronous Receiver/Transmitter)

- 'Point to point' protocol, only goes between two devices, data is sent LSB first
- Asynchronous – without a clock both sides need to agree on a transfer rate, given as the baud rate
- Data is sent in frames which consist of a start bit, 5-9 data bits, 0 or 1 parity bits, and 1-2 stop bits
- Parity is a basic form of error checking, most used are odd and even parity

7.4.1 UART Modes

- Simplex – one direction only
  - RX cannot send info back to TX
- Full duplex – both devices can send and receive at the same time
- Half duplex – only one device can be sending or receiving at any time

7.4.2 UART Flow Control/Handshake

- Hardware flow control uses Request to Send (RTS) and Clear to Send (CTS)
  - RTS and CTS are cross-connected
  - Device keeps RTS high to indicate it can receive information
  - When it cannot, it sets RTS low. Sender then stops transmitting
- None of this is mandatory, can be emulated using ASCII characters

7.5 USB (Universal Serial Bus)

- Key features that USB stepped in to help solve
  - Plug and play – had to configure each other device manually, e.g. UART settings
  - Scalability – point to point only, multiple devices needed multiple ports
  - Power delivery – each point had a fixed power rating. devices had to work with that lose compatibility
- USB is a host to device protocol, where the host controls all communication
- Bust works like a tree, starting with the host as the root hub, can support up to 127 devices
- Only the host OR a device can be communicating at any time
- No device-to-device communication

7.5.1 USB Transfer Types

- Control – used for device enumeration and configuration
- Bulk – large burst-type data (mass storage)
- Interrupt – handling 'interrupt' like scenarios (mouse/keyboard)
- Isochronous – guaranteed latency but without error checking (mic, video)

7.5.2 Cyclic Redundancy Checks (CRC)

- We saw parity bits for UART, so why use CRC?
- Parity – detects single-bit errors, but misses many multi-bit errors
- CRC – detects all single-bit rrors, most multi-bit errors, and errors in subsequent bits up to the CRC size
- Mathematically uses polynomial division, modulo two

8 Wireless Protocols

8.1 Why Wireless?

- For IoT deployments, wireless is the cleanest or in some cases, only way of communicating with devices
- However, wireless communication is much more complicated than wired
- With wires, the interfaces are the biggest source os noise, e.g. where the wire connects to a pin on the board

8.2 Wireless Reliability

- With wireless, communication reliability scales with distance
- Worsened by interference from other devices and environmental obstructions
- Using parts of the EM spectrum is licensed by governments – need govt approval to use these for communications
- can vary from region to region, devices that work in one may not always work in another
- Parts of the spectrum are unlicensed

8.3 Wireless Communications

- Wireless transmissions modulate a high frequency sinusoid (the carrier signal) based on the messages which needs to be transmitted
- in digital communication, the analog carrier signal is modulated by a discrete signal

8.4 Link Budget

- Accounting of all gains and losses in a wireless link
- Must ensure that received power > receiver sensitivity, where RX power is:

$$\text{RX power} = \text{TX power} + \text{TX gain} + \text{RX gain} - \text{FPSL}$$

- Free-space path loss (FSPL) is a formula to quantify all gains and losses along a transmission path

$$FPSL = \left(\frac{4\pi df}{c}\right)^2 = 20\log d + 20\log f + 20\log\frac{4\pi}{c}$$

- The last constant term depends on the units

8.5 Wireless Protocols

8.5.1 Bluetooth

- Short range (10m – 100m) communication, data rate of 1-3 Mpbs
- Controller device architecture
- Uses time division multiplexing
- Bluetooth uses frequencies between 2.4 GHz and 2.48 GHz (ISM band), split into 79 channels

8.5.2 Bluetooth Low Energy

- Introduced to address the limitations of Bluetooth classic
- Supports mesh networks
  - Devices can talk to each other and relay information to each other
  - Removes the 7 active device limit of Bluetooth Classic
  - BLE supports 100s of devices

8.5.3 Others

- There is also Wi-Fi, which is the most used wireless protocol, uses 2.4 GHz and 5 GHz bands with 6 GHz bands under development
- By far the highest power consumption but also the highest data rate
- Also LoRA and NFC

9 Clock, Power, and Energy

- Crystal oscillators produce only a single clock – how to get multiple frequencies?
- Using phase-locked loops – goes from slow to fast
- Remember that power is NOT energy
- Power does matter but we care more so about energy consumption

9.1 Power in Digital Circuits

- Since we now use transistors we consider static and dynamic power
- Static is power from transistor leakage, as always some amount of leakage current
- Dynamic power is power spent from turning on and off very often

$$P_{DD} = \alpha \cdot V_{DD}^2 \cdot C_L \cdot f$$

- α is the switching activity factor, measured as a percentage
- To get accurate values of α we need to run extensive simulations
- Capacitive load C_L – larger circuit – more charge required
- As embedded designers we have most control over V_{DD} and frequency

9.2 Circuit Techniques to Reduce Power

- Reducing frequency – lower frequency of blocks when they're not being used
- Power gating – power off parts of the circuit
- In most circuits, the clock consumes 10 to 40% of the total power
- Can also gate clock – send no signal to parts of the chip not in use

9.3 Lowe Power Modes

- System can't operate like normal when in sleep
- Need to wake up the system up, could be from EXTI
- Deeper sleep modes means we need longer to enter and exit from a low-power state
- Need to restore volatile data from Flash memory
- Often given time and not the number of cycles needed to enter and exit status
- If we're accounting for wake up time, a conservative estimate assumes the wake up time is at the full-speed of the CPU
- If we increase sampling rate, we could see at some point, waking up and sleeping is too much so we just leave on for that long
- Also keep in mind that batteries do not always provide the same voltage for change

9.4 Voltage Regulators

- Takes variable voltage input and creates a regular noise-less output
- Low-dropout regulator (LDO)
- We typically measure energy stored in batteries as watt-hours
- Battery energy is typically given in amp-hours, which measures charge and not energy

10 Memory

10.1 Random Access Memory

- Fast but limited, volatile so power-dependent
- Major types are SRAM and DRAM
  - SRAM – retains value as long as power provided
  - DRAM – dynamic because just a capacitor, needs refreshing

10.2 SRAM Cell

- To read – set WL to 1
- To write – set WL to 1, drive BL and !BL to values desired, then when WL is lowered, the value is retained
- Don't forget address decoders and sense amps
- And also drivers for data incoming

10.3 DRAM Cell

- Significantly better density, but requires a completely differenty fabrication process
- Write – set BL to desirted value, activate WL
- To read – precharge BL to $V_{DD}/2$, then the sense amp will detect a small change in V. Direction of the change was the value stored.
- Every few ms, needs a refresh of voltage and rewrite values
- DRAM is about 20x denser than SRAM but about 10x slower.

10.4 Read-Only Memory

- For code that shouldn't change
- Used for program once PROM or masked ROM
- Nowadays we typically use Flash memory

10.5 Flash Cell

- To read, set both s and G to V_{DD}
  - If no charge of FG, acts like a regular MOSFET and would read 1 at the gate
  - If there is a charge on the FG then the fgate would readf 0 at the drain
- Changing value of a cell is more complex, if FG is high then we need to erase that charge
- To speed up erases we need to erase whole blocks at a time, same goes for writing.
- Means when we write even a single bit we have to rewrite the whole block
- Flash memory will wear out if used too much, typically 10000s of write cycles

10.6 NAND and NOR Flase

- NOR is in parallel
  - Fast reading, slow writing, byte granularity
  - Good for code
- NAND is in series
  - Slow reading, but ultrafast writing
  - Far denser, better for storage

11 Embedded CPUs

- For more systems, the CPU is only a small portion of the chip
- Most things boil down to the Instruction Set Architecture
- Cortex M4 uses the ARMv7 ISA – 16-bit instructions

11.1 Single-Instruction Multiple Data

- Instructions meant for video/audio processing
- SIMD instructions allow you to operate on each separately
- SIMD is how GPUs can do so many operations at once
- ARM-M4 has special instructions for FP adds
- Often best to use the provided libraries

11.2 Cycles Per Instruction (CPI)

- Number of cycles taken by an instruction is known as CPI
- Total cycles is determined by the number of instructions x CPI average
- Factors affecting CPI
  - Hardware
    - ISAs – as we saw earlier
    - Architecture – how the CPU is designed at an RTL level
  - Software
    - More efficient algorithms – less lines of C and assembly
    - Why we prefer lower level programming languages
    - Compiler – how efficiently binaries are produced

11.3 Using Benchmarks

- Use a representative set of programs that mimic real world
- Embench is a popular benchmark suite

12 Finite State Machines

- FSMs are a way to make a parallel system work i a sequential way
- Same applies for embedded systems, where tasks can happen in parallel with the CPU
- Biggest advantage is clarity
- Could use tyedef enum to declare states – enumerates automatically
- current state is declared static because we want it to remember value even after the function is exited
- For clarity we could combine output with the next state but generally we don't do that, just like in Verilog

13 Pipelining

- Separating each stage of a program so they can all run in parallel
- Latency – time between an input arriving and corresponding output being produced
- Throughput – total number of inputs that can be produced per unit time
- Bandwidth – amount of data transferred per unit time

13.1 Hazards

- Structural hazards are when two stages need to use the same hardware at the same time
- Pipeline stall – a period of time where the entire pipeline is halted due to a hazard
- Data hazard – when a stage needs data that is not yet ready
- Bubble – paritcular unit of hardware is idle due to a hazard, but the pipeline is not stalled

13.2 Double Buffering

- Create 2 buffers and swap them between two ends of the pipeline
- No need to disable IRQs, but we double memory requirement
- Without synchronization, incorrect data (due to hazard) and with over synchronization, get stalls (structural hazards)
- Good pipelining avoids both, while maximizing performance

14 Real-Time Operating Systems

- In many embedded systems, a single CPU must monitor multiple things
- Doing multiple things on one system introduces new issues
- Real-time means meeting strict deadlines, not just fast
- Need a way to handle scheduling different critical operations

14.1 GPOS vs RTOS

- GPOS focusses on high throughput, which is important for consumer devices hwere OS has to handle applications are running at the same time
- RTOS focusses on predictable latency
- The biggest diffwerence lies in the scheduler

14.2 OS Scheduler

- With GPOS schedulers, it is common to use round robin scheduling to ensure tasks can execute
- Scheduler runs when a task finishes
- RTOS scheduling uses priority-based pre-emptive scheduling – higher priority tasks always take precedence

14.3 Context Scheduling

- In an RTOS, tasks may not complete in one go so we have what's called context switching
- Is not free, just like how we have to spend special jump instructions in assembly