Hardware Platforms and Sensors

Tom Spink

Including material adapted from Bjoern Franke and Michael O’Boyle
A hardware platform describes the physical components that go to make up a particular device.
Designing a hardware platform

- Application size/complexity
  - Type of microcontroller
    - Clock speed
  - Additional processors
    - GPUs or DSPs?

- I/O
  - GPIO
  - Serial (I²C, SPI, etc)

- Connectivity
  - Wired/Wireless

- Power/energy constraints
  - Battery powered
  - Solar powered
  - Grid connected - reliable/unreliable
Microcontrollers

A microcontroller is an integrated circuit, containing one or more processing units, various memories, and a number of functional units for computation and I/O. Typical microcontrollers implement the Harvard architecture.
Types of Memory

The embedded memories present in microcontrollers come in different flavours, and are used for different purposes.

In the Harvard architecture, systems have separate program and data memories, usually backed by different technologies, and with different capacities.

- **Program Memory**
  - Flash

- **Volatile Data Memory**
  - SRAM

- **Non-volatile Data Memory**
  - EEPROM

- **Volatile Data Memory**
  - DRAM
Memory Constraints

Microcontrollers are **memory constrained** devices, e.g. the AVR ATmega328 has:

32k flash, 1k EEPROM and 2k SRAM

Because of small program memory, it is important to try to reduce code size.

Software techniques (and the use of -Os) help, but architectural techniques can also help (e.g. CISC, or “compressed” instruction sets (**Thumb**))
Power and Energy

Power and energy are first-class issues in embedded systems:

- Battery Life
- Energy Density
- Environment

A processor that uses more power, but takes less time may use less energy:

\[ E = \int P(t)\,dt \]

Important to decide what to optimise for.
Power and Energy

Amps \( A = \text{Cs}^{-1} \) \hspace{3cm} V = I \cdot R
Volts \( V = \text{JC}^{-1} \) \hspace{2cm} P = I \cdot V
Watts \( W = \text{Js}^{-1} \)
(Ohms \( \Omega = \text{JsC}^{-2} \))

3.5V (nominal) battery with 3Ah of storage = 3 Cs\(^{-1}\)h \cdot 3.5 JC\(^{-1}\) = 37.8kJ

Processor running continually at 1W => 37.8kJ / 1 Js\(^{-1}\) = 10.5 hours
Power Saving Techniques

Dynamic adjustment of Voltage/Frequency: Dynamic Voltage and Frequency Scaling

Usage of sleep modes for microcontroller: Turn off internal functional units when not required.

Power down external peripherals: Turn off radio when not required.

Software optimisation!
Dynamic Voltage and Frequency Scaling

Power Consumption for CMOS circuits

\[ P = \alpha C_L V_{dd}^2 f \]

- **P**: Power
- **\( \alpha \)**: Switching activity
- **\( C_L \)**: Load capacitance
- **\( V_{dd} \)**: Supply voltage
- **\( f \)**: Clock frequency

Delay for CMOS circuits

\[ \tau = k C_L \frac{V_{dd}}{(V_{dd} - V_t)^2} \]

- **\( \tau \)**: Delay time (upper limit of 1/f)
- **\( V_t \)**: Threshold voltage (< \( V_{dd} \))

Decreasing voltage slows down linearly, but quadratic power saving.

Processors offer valid “pairs” of valid voltage/frequency choices, e.g. Intel Speedstep has 6 choices, ARM big.LITTLE has 18!
Sleep Modes

Microcontrollers may provide a way to manage power by enabling different sleep modes, with different wake-up sources, and quiescent power usage.

**Table 10-1.** Active clock domains and wake-up sources in the different sleep modes.

<table>
<thead>
<tr>
<th>Sleep mode</th>
<th>Active clock domains</th>
<th>Oscillators</th>
<th>Wake-up sources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>clk_cpu</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clk_FLASH</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clk_GPIO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clk_ADC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clk_MSR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Idle</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x(2)</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>ADC noise reduction</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x(2)</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x(3)</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Power-down</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x(3)</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Power-save</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x(3)</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Standby(1)</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x(3)</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Notes:
1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.
Sensors

A sensor captures a **physical** quantity, and converts it to an **electrical** quantity.

Many physical effects are used for constructing sensors:

- **Law of induction** (generation of voltages in a magnetic field)
- **Mechanical properties** leading to varying electrical resistances
- **Photo-electric effects**
Types of sensors

- Push-button/switch
- Acceleration
- Gyroscope
- Magnetometer
- Temperature
- Pressure
- Image/Optical
- Rain
- Proximity
- Hall-effect

All deliver an electrical representation of a physical quantity
Input/Output

I/O is how the device interacts with, and senses, the real world. I/O can be as simple as an on/off electrical signal (GPIO), or a more complicated signalling protocol for data transfer, such as I²C, SPI or CAN.

I/O is required for interfacing with sensors.
General Purpose Input/Output (GPIO)

GPIO are simple on/off voltage signals, that can drive outputs, or signal inputs to the microcontroller. They appear as physical pins on the microcontroller.

Microcontrollers generally have current sinking and sourcing limitations (e.g. in the range of 50mA), meaning that GPIOs are limited to low-current applications. They can be used to drive higher current loads through transistors and relays.

Input pins can be configured to signal interrupts - important for waking up from sleep modes, and real-time applications.
Pulse-width Modulation

If you switch a GPIO on and off fast enough, you can generate a **pulse-width modulation** (PWM) signal, to **approximate** varying output voltages. Microcontrollers typically have **dedicated** configurable PWM drivers.

Commonly used technique for varying power to **inertial** loads. The choice of switching frequency varies depending on the target load.

The greater the **duty cycle**, the more **power** is transferred to the load.

Made practical by modern electrically controlled (solid-state) power switches, e.g. MOSFETs. Very efficient power transfer, as typically **zero current** flows when the switch is **open**, and very little **voltage drop** ($R_{DS,ON}$ is low) when switch is **closed**.
Pulse-width Modulation

**Duty Cycle (%)**: Percentage of time on during a period

**Frequency (Hz)**: Rate of periods

**Period (s)**: 1/f

**Amplitude (v)**: Output voltage

\[
V_{avg} = V_{amp} \cdot \left( \frac{DutyCycle}{100} \right)
\]
Analogue-to-Digital

Another form of simple I/O are **analogue-to-digital inputs**. These inputs read a voltage level (usually between ground and some reference voltage), and report back a **discrete number** indicating the level of this voltage.

Digital computers require a **discrete mapping** from the **time domain**, to the **value domain**. Taking a reading at a particular point in time is called **sampling**.

\[ s : D_t \rightarrow D_v \]

The **sampling rate** relates to how fast the input can read a (stable) voltage level.

Regular sampling allows recording incoming waveforms.
Analogue-to-Digital (Sampling)

**Sample-and-hold:** clocked transistor and capacitor; the capacitor holds the sequence values

\[ e(t) : \mathbb{R} \to \mathbb{R}, \quad h(t) : \mathbb{Z} \to \mathbb{R} \]
Analogue-to-Digital (Sampling)
Aliasing

\[ \sin\left(\frac{2 \cdot \pi \cdot x}{8}\right) + \frac{1}{2} \sin\left(\frac{2 \cdot \pi \cdot x}{4}\right) \]
Sampling Theorem

Reconstruction is **impossible**, if not sampling frequently enough

How frequently do we have to sample?

**Nyquist criterion (sampling theory):**

Aliasing can be avoided if we restrict the frequencies of the incoming signal to less than half of the sampling rate.

\[ p_s < \frac{1}{2} p_n \]  : where \( p_n \) is the period of the “fastest” sine wave

\[ f_s > 2 f_n \]  : where \( f_n \) is the frequency of the “fastest” sine wave

\( f_n \) is the Nyquist Frequency, \( f_s \) is the sample rate.
Analogue-to-Digital (Resolution)

The resolution determines how precise the reading can be for a given voltage range. The resolution of an ADC converter is typically stated in bits.

\[ Q = \text{resolution in volts-per-step} \]
\[ V_{FSR} = \text{difference between largest and smallest voltage} \]
\[ n = \text{number of voltage intervals} \]

For example, 10-bit resolution means \( 2^{10} = 1024 \) discrete voltage levels. With a voltage range of 0-5V, this means \( Q = \frac{5}{1024} = 0.0049V \) per step.
Resolution Example (3-bit)

\[ \frac{5V}{2^3} \text{ steps} = 0.625 \, \text{V step}^{-1} \]
Signal-to-noise Ratio

Typically expressed in dB

$$SNR = 20 \log_{10} \left( \frac{V_{signal}}{V_{noise}} \right)$$

e.g. $$SNR = 20 \log_{10} \left( \frac{5}{0.5} \right) = 20 dB$$

SNR for ideal n-bit converter is:

$$n \cdot 6.02 + 1.76$$

e.g. 10-bit converter:

$$10 \cdot 6.02 + 1.76 = 61.96 dB$$
Digital-to-analogue

The reverse of analogue-to-digital is digital-to-analogue. In this case, you tell the driver what voltage to produce, and it will generate the requested voltage, or even a (possibly complex) waveform.

Some applications for digital-to-analogue conversion can be approximated with PWM, e.g. dimming an LED.

Quality of DAC (frequency, resolution) important for application, e.g. audio.
Interfacing with other sensors

Non-trivial sensors may have a dedicated communications interface, or it may simply be convenient to use a communications bus if using many sensors.

Retrieving a value from the sensor requires interrogating the sensor’s internal registers, by using communication channels such as: I²C, SPI, One-wire-bus, 4-20mA+HART, CAN, UART.

Example: MCP9808 I²C Temperature Sensor: send a command to retrieve current temperature reading.

Benefit: Sensor does all the heavy-lifting for A-to-D conversion (+ ability to easily multiplex sensors)
Actuators

Actuators turn a digital signal into a physical effect.

- Indicators (LEDs, bulbs, LCDs)
- Motors
- Relays
- Speakers/Buzzers
- Heaters

Embedded systems typically can’t drive high-power loads directly, so must use power switches (MOSFETs, IGBTs, relays, contactors, etc) to operate them.
Application: Heating System

**Sensors:** Temperature, Up/Down Buttons

**Actuators:** Heating Element (via relay), LCD

**Microcontroller:** 1 analogue input, 2 digital inputs, 1 digital output, SPI interface for LCD

**Implementation:** PID control loop
Application: Heating System

Sensors: Temperature, Up/Down Buttons

Actuators: Heating Element (via relay), LCD

Microcontroller: 1 analogue input, 2 digital inputs, 1 digital output, SPI interface for LCD