Class Notes for Embedded Systems 1 - ENCE 361

Dr. S.J. Weddell

Department of Electrical & Computer Engineering (ECE)
University of Canterbury
Current address: Room: 470, Electrical Engineering *von Haast* Building
E-mail address: [1]`steve.weddell@canterbury.ac.nz`

Key words and phrases. computer architecture, embedded systems, microcontrollers, microprocessors

Abstract. This set of notes provides some of the information required for this part of your Embedded Systems course. The interested reader should review the relevant sections of books and papers listed in the reference section.

Contents

0.1. Preface

Sections of this set of course notes have been written and compiled from 3rd Year Computer Science and 2nd Professional Engineering courses over the last 12 years, such as COSC361, ENEL353, and ENEL323. This year, certain sections have been expanded and others have been completely removed.

Two points should be noted before using these notes. Firstly, background knowledge of number representation is useful, i.e., in terms of bits weightings. For example, does 1111_2 represent $-\frac{1}{8}$ or 15? Such topics are introduced in several courses, such as COSC221 and ENCE260.

Secondly, it is assumed that students have the ability to convert between Binary, Hexadecimal, and decimal numbers. Also, some experience using an 8-bit microprocessor is extremely useful. These topics are, however, briefly reviewed.

Since many non-engineering students may be new to electronics, Chapter 1 provides a brief overview on microcontroller fundamentals. Appropriate references are provided and this additional reading may prove useful for laboratory sessions.

If you are new to electronics I highly recommend you read the relevant sections and refer to these where and when they are introduced in the course. Due to the uniqueness of this course, no, one textbook was found to be entirely appropriate. However, the recommended text provides a wealth of material that will be referenced throughout these notes where appropriate. Our aim is to complement this text, not to rewrite it! Also, a wealth of other information pertaining to the Stellaris series of microprocessors can also be found on the world-wide web. Additional references will be provided throughout the course.

Lastly, care has been taken in preparing these notes but the author would be indebted to the reader for acknowledgement of any errors found. In addition, your continuous feedback throughout the course will be greatly appreciated.

I hope you enjoy the course!

Dr Steve Weddell
Dept. of Electrical & Computer Engineering
University of Canterbury
steve.weddell@canterbury.ac.nz
February 2015

CHAPTER 1

Foundations

This first section is designed for those who have little or no background in electronics. Such a background is required in the study of Digital Electronics and microprocessors, particularly in a course such as this. To have an understanding of two basic quantities used in electronics, namely voltage and current, and how they are related, this is considered an important prerequisite.

If you already have an understanding of basic electronics and can analyse a small, DC circuit in regards to voltage, current and resistance, you may wish to move on to the next section. For those who require a little more background the following is a summary from The Art of Electronics [**1**]. If more information is required students are encouraged to read Chapter 1 of The Art of Electronics for a more thorough coverage of the subject.

1.1. Electrical Parameters of Conducting Materials

Voltage between two points is the work done in moving a unit of charge from a more negative (lower potential) to a more positive (higher potential). The symbol used for voltage is V and the unit of measure is Volts. For small voltages of less than say 0.01 V, it is easier to use a lower scale unit such as *milli*, 10^{-3}. One millivolt is equivalent to 0.001 Volts. Thus, 0.01 Volts = 10 mV.

Current is the rate of flow of electric charge past a point. The unit of measurement is the Ampere and the symbol for current is, **I**. As with small Voltages, small Amperages also use the lower scale unit such as the *milli.* Thus, $0.001 Amps = 1$ mA. The next lower scale unit is the *micro.* The symbol used for the micro is μ, and this is 10^{-6} of the full-scale voltage or current i.e., $0.00001A = 10\mu$A.

Since a voltage is a potential difference, measurement is always between two points . For example, a small AA type battery is always measured between the positive and negative terminal. Current, on the other hand, is always refers to what is flowing *through* a device or circuit. A current is allowed to flow through a conductor by putting a voltage across two points. Without a voltage, a current cannot flow.

We can measure both voltages and currents using an instrument called a multimeter. Every bench in the electronics laboratory is equipped with a multimeter. If the multimeter has a digital display it is termed a Digital Multimeter (DMM). Rather than just measure voltages and currents we can actually see their effects by using another instrument called an Oscilloscope. As part of the first lab session, a tutorial will be provided to learn how to use these instruments.

When current flows through a conductive (or partially conductive) material, the current is proportional to the voltage across it. We refer to this proportionality as *resistance*. Some conductors, such as thick, multi-stranded copper wire for example, have a very low resistance and can be said to be highly conductive. Other materials, such as carbon, can be used to vary resistance. The unit of measure for resistance is the **Ohm** and the symbol used is Ω. Thus, a carbon resistor of value 100 Ohms can be written as $100\,\Omega$.

Since the scale units of milli (m) and micro (μ) are typically used for voltages and currents, resistance values are usually quite large. Thus, the Kilo ($\mathrm{K} = 10^3$) or Mega ($\mathrm{M} = 10^6$) symbols are used (note upper case), to signify scaling of 1000 and 1000000 Ohms, respectively. Rather than write 100 KΩ for a value of 100000 ohms, the Ω symbol is often dropped and the resistance is written as 100K.

1.2. Relationship between Electrical Parameters

Now that the qualities of voltage, current and resistance have been defined, the relationship between them can be stated. In the previous section the proportionality between voltage (V) and current (C) flowing in a conductor or partial conductor was defined as resistance, R. This can be formalised as:

$$V = I \cdot R. \tag{1.1}$$

Therefore, resistance is defined as,

$$R = \frac{V}{I}. \tag{1.2}$$

This relationship can be shown on what is known as a V-I Chart. Here, using the Euclidean plane, the actual slope of the line determines the conductivity. Conductivity (G) is the inverse of Resistance (R) i.e., $R = 1/G$. This is shown in Figure 1, below.

From Figure 1 we can see that resistances are characteristically *linear*. The response curves from other devices such as semiconductors and active components do not show linear characteristics.

1.3. Signals

A signal can be either a voltage or current value which can change with respect to time. Direct current (DC) ensures a flow of electrons in one direction (conventionally, this is positive to negative) and given a constant resistance, results in a

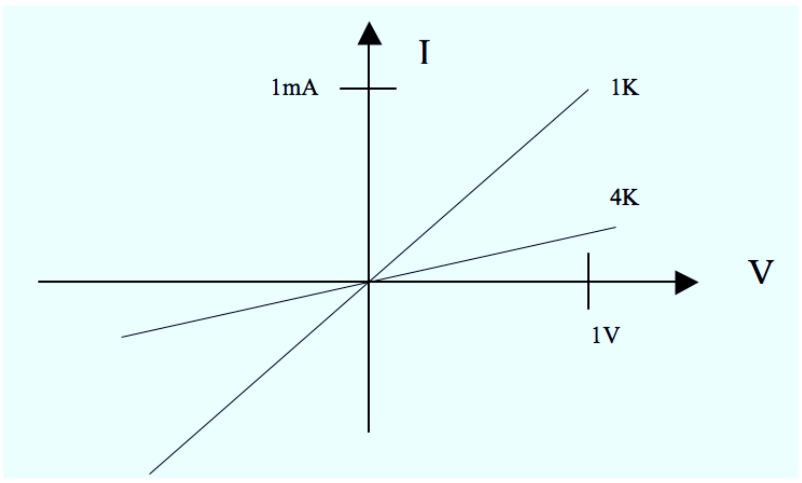


FIGURE 1. Plots for two resistors of values 1K and 4K.

voltage as shown on the left in Figure 2. Given a constant resistance, as current is depleted voltage is reduced as shown on the right of Figure 2.

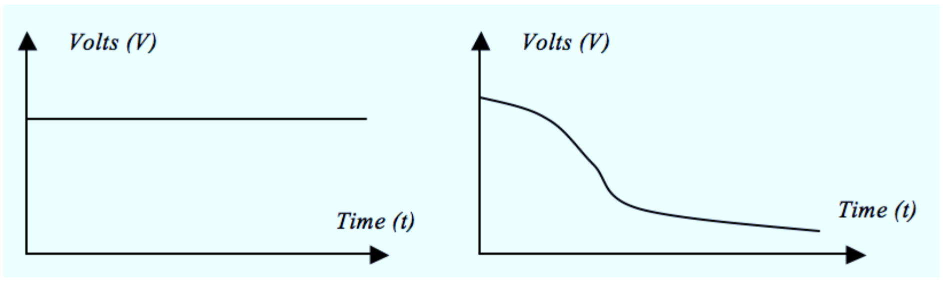


FIGURE 2. Examples of a constant supply (left) and depleted supply (right) voltage.

The curve on the left in Figure 2 could represent for example a DC, bench-top power supply. The curve on the right might represent the response of a battery used in a cellular phone.

In the case of the varying voltage signal, the pattern of variation can either repeat or not. Signals that repeat are referred to as periodic signals. Periodic signals that alternate between positive and negative voltages and currents are called AC signals.

The most common periodic signal is the Sine wave. As can be see in Figure 3 the magnitude of the waveform i.e. its amplitude (A), is shown, as well as its peak-to-peak (p-p) voltage (V_{pp}). The period of the waveform is measured in time (t). Time measurement is the time taken for the waveform to repeat i.e. between periods. Frequency measurement (f), is the number of cycles or periods that occur within one second and is measured in Hertz (Hz). Such a response is known as an alternating current (ac), waveform.

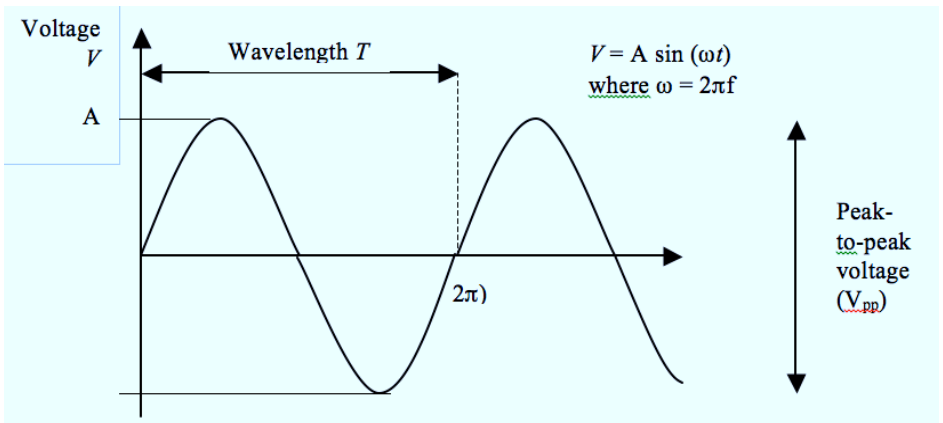


FIGURE 3. Sine wave of amplitude, A, and frequency, *f*.

The amplitude of a periodic waveform, A, is also used to characterise the voltage, in terms of root mean square (RMS), of a periodic signal. The RMS voltage is a useful parameter to calculate power. For a sinusoidal waveform, the RMS voltage is give by,

$$V_{RMS} = (1/\sqrt{2}) \cdot A. \tag{1.3}$$

It is important to understand that apart from sinusoid waveforms given in Equation 1.3, RMS will differ for other periodic waveforms, such as triangular and square waves. The general formula to calculate RMS voltage is given as,

$$V_{RMS} = \sqrt{\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} V^2 dt}, \tag{1.4}$$

where T_1 and T_2 are time constants specifying the period of the waveform, and V is the waveform function as defined in Figure 3.

1.4. Electronic Components

Components are used in order to change the voltage and current signals in a circuit. Resistors, as we have seen, produce a linear response, capacitors and inductors produce non-linear phase changes in ac signals. The symbols used in circuit diagrams to describe components are shown in Figure 4.

Components shown in the top row of Figure 4 are usually referred to as passive components and form the basic building blocks of electronic circuits. Semiconductor devices such as those shown in the last row are referred to as active devices. Active devices provide a very different response in terms of their V-I curves when compared with passives i.e., their resistances are not linear.

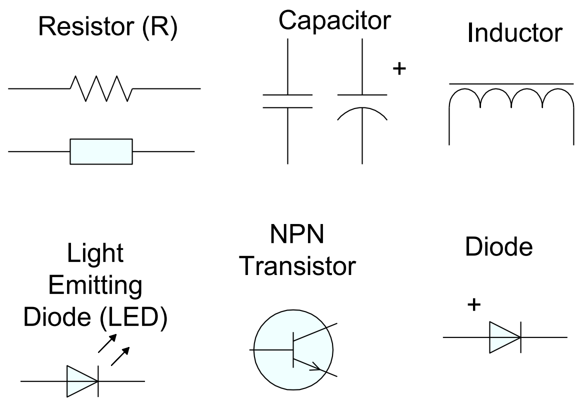


FIGURE 4. Typical symbols used for common electronic components.

In addition to the basic passive and active electronic components previously mentioned, symbols are used in circuit diagrams to represent dc sources such as batteries, ac sources such as alternators as well as power and ground. These symbols are shown in Figure 5.

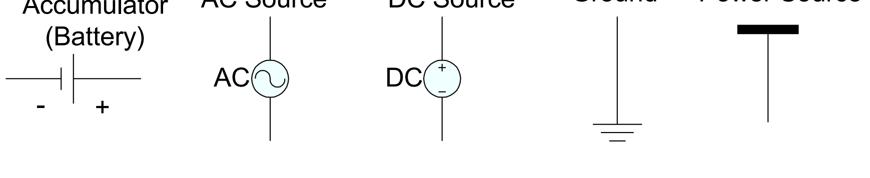


FIGURE 5. Power, ground, and source symbols.

1.5. Electronic circuits

An electronic circuit can be constructed by combining electronic components with either a voltage or current source. An example of such a circuit is shown in Figure 6.

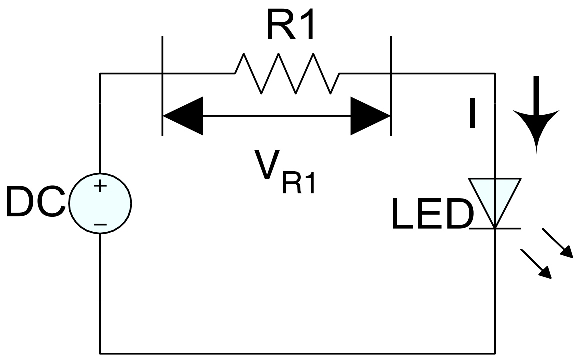


FIGURE 6. An LED circuit with load resistor.

The circuit shown in Figure 6 is a practical circuit that will illuminate an Light Emitting Diode (LED) when the DC power source provides sufficient voltage. The LED component typically needs a 1.7 voltage drop across its terminals in order to turn-on. In addition, approximately 20 mA of current is needed to flow through the LED to achieve the specified brightness. As a result, `R1` will regulate the required current as its resistance value will be directly proportional to this; this was discussed in Section 1.2.

Using the Equation (1.2) in Section 1.2, it is possible to calculate the resistance to provide the 20 mA required to flow through the LED. If a voltage drop of 1.7 exists across the LED and the supply voltage (DC) equals 5 Volts, then the voltage across `R1` must equal 3.3 Volts i.e.,

$$V_{DC} = V_{R1} + V_{LED}. \tag{1.5}$$

Now if $V_{R1} = 3.3$ Volts, then by using Equation (1.2), $R1 = V_{R1}/I = 3.3/0.02 = 165\Omega$. Since resistor values only come in a preferred series, the closest value would be $180\,\Omega$. Thus, we have now specified our complete circuit.

1.6. Signals and Signal Representation

Analog systems are associated with continuous representations of quantities such as temperature, voltage, current or time. For example, the daily temperature can be represented on a graph over continuous time. Digital systems however, are often used to approximate the values of these same quantities by using collections of just two, discrete symbols. For example, (a) in Figure 7 shows the temperature recorded over a 12-hour period. This system can be interpreted as an analog system for both temperature and time are represented by continuous values.

If, however, the continuous temperature signal were sampled every two hours and each value converted into the digital domain, the discrete signal representation shown in (b) of Figure 7 would result. Each binary bit provides a binary weighting (Radix 2), over each interval sampled every two hours.

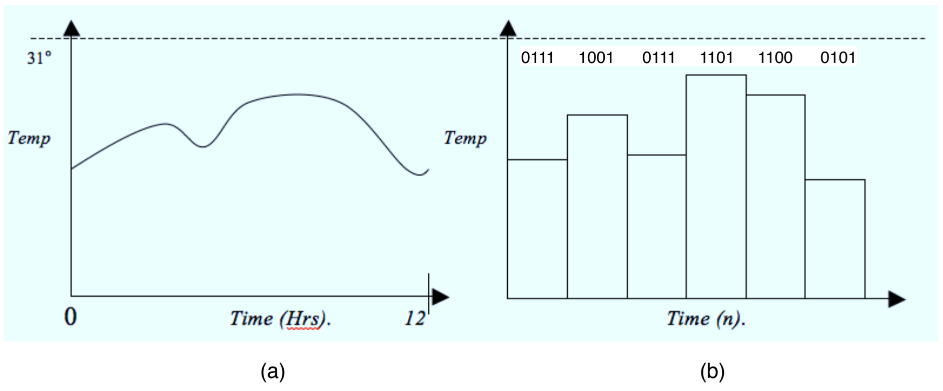


FIGURE 7. Signals: (a) a continuous time example; (b) a discrete time approximation of (a).

In this example, the temperature is taken every 2 hours and is represented by an unsigned, 4-bit binary value. The use of 4-bits provides 16 discrete levels that can be used to approximate the actual temperature. If each bit increment is represented as a 2-degree increase in temperature, then approximately 0 to 31 degrees could be represented in binary. As can be interpreted from the graphs, an infinite number of bits (resolution) in the digital system would be required in order to exactly represent the equivalent continuous system.

1.7. Sampling theory

Shannon's sampling theory was a significant contribution to the fields of communication and signal processing. It simply stated that to reproduce a function $x_1(t)$ completely, it was necessary to sample at twice the highest frequency of $x_1(t)$, or $2W$, where W is the highest frequency of the input spectrum.

The sampled waveform $x_1(t)$ in Figure 8 is shown in both the time and Fourier domain. However, when this waveform is sampled, as shown by the function $x_s(t)$,

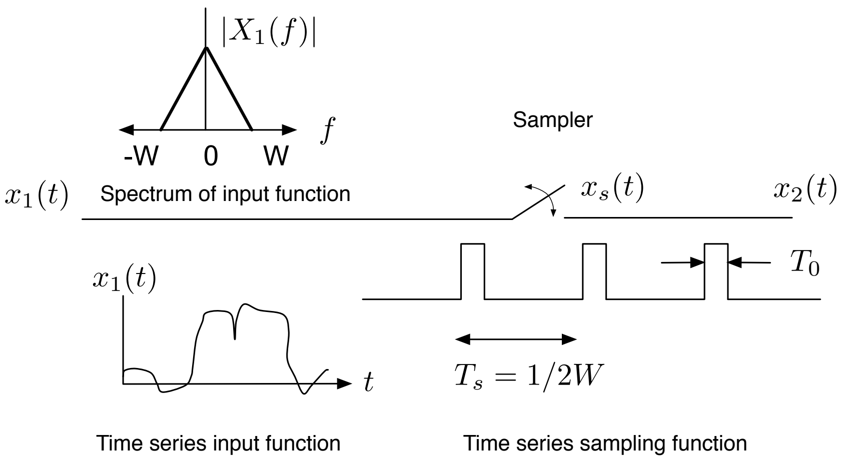

$|X_1(f)|$
f
-W
0
W
Spectrum of input function
Sampler
$x_1(t)$
$x_s(t)$
$x_2(t)$
T_0
$x_1(t)$
t
$T_s = 1/2W$
Time series input function
Time series sampling function

FIGURE 8. Sampling theory representation.

the resulting frequency spectrum of $x_2(t)$, $X_2(f)$, is repeated at $2W$ frequency spacings, as shown in Figure 9.

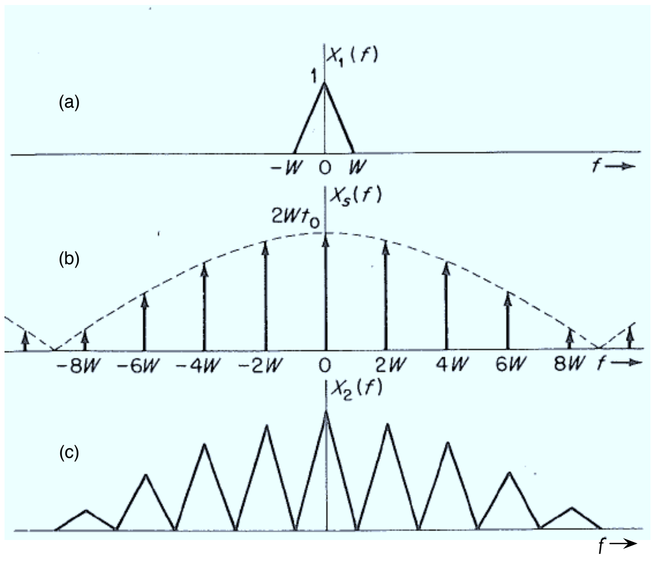


FIGURE 9. Frequency representation of sampling a time-series signal: (a) Spectra of the original signal; (b) Spectrum of the sampling function, $x_s(t)$; (c) resulting spectrum of the sampled original signal, $x_2(t)$. Adapted from [**2**].

In order to recreate the original signal, $x_1(t)$, a low-pass filter is required to remove the additional harmonics, e.g., at $2W, 4W, 8W$ etc., produced by the sampling function.

1.8. Binary signals

The basis for all digital logic is the existence and application of two states. This can be interpreted as a quantity being either "ON" or "OFF", "TRUE" or "FALSE", logic-1 or logic-0, "HIGH" or "LOW", Vcc[1] or Ground. In practice, by using just two symbols, any of these interpretations could describe the state of a binary logic condition.

The simplicity of using just two states means that only one of two states need to be assigned to a signal at any particular time. Due to noise, a range of voltages are used to represent a specific logic level. The assignment of two voltages, high and low, for both logic-1 and logic-0, is shown in Figure 10.

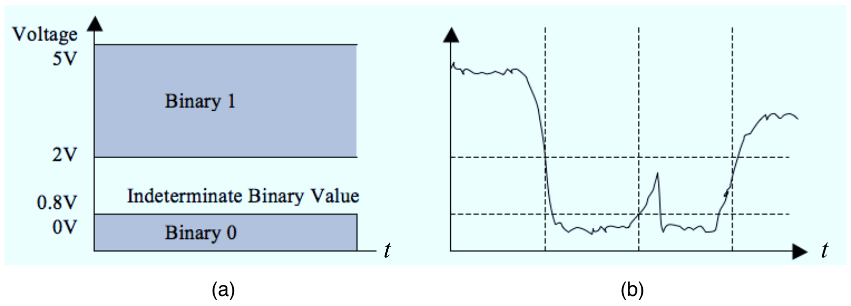


FIGURE 10. Signals: (a) discrete binary classification of signals; (b) continuous time-series signal representation.

Note that in Figure 10 the range of voltages between 0.8 and 2 Volts is referred to as the *noise margin*. The voltages that determine this range vary with technologies that are used to implement logic. It should be noted that with the use of 1.8 Volt logic used in some portable equipment, such as mp-3 players, noise margin is a major consideration. By using a consistant technology to implement a complete design, incompatibilities between logic levels are minimised.

1.9. Waveform charts

This section discusses the interpretation and construction of waveform charts. The ability to display and later analyse several waveforms with respect to time allows us to take a snap-shot of a digital or analog system.

[1]Vcc stands for *Voltage common collector* and is usually the positive supply voltage to a circuit; for most digital systems, this is 3.3 Volts

Typically waveforms are drawn on the Euclidian plane with the vertical axis showing signal amplitude and the horizontal axis showing time. Historically, a logic-1 represented 5 Volts and a logic-0, 0 Volts. However, even though 5 Volt technology is still available, digital circuits now use lower voltages. Therefore, waveform charts typically represent technology where 3.3 Volts is used to represent logic-1 and 0 Volts a logic-0.

A typical waveform chart used in analysing a digital system is shown in Figure 11.

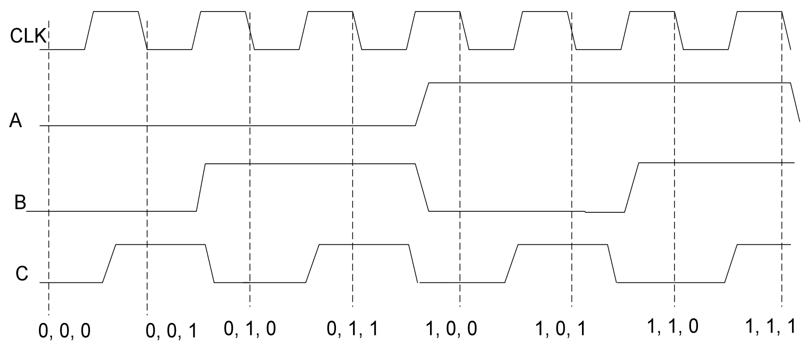


FIGURE 11. An example of a clock signal and 3 data signals.

Several points regarding Figure 11 can be made:

- A clock signal is typically used as a reference and can be either on the rising (left) edge, or falling (right) edge. In this example, the vertical dotted lines show each reference event.
- When a clock or data signal changes with respect to time, such changes as shown by using sloping lines. Ideal waveforms are shown as vertical lines. This example does not show ideal waveforms.
- The digital representation of all three combined signals is shown at the bottom of the diagram.

1.10. Integrated circuits

The use of Integrated Circuits has provided much higher levels of functionality in a relatively small package. Figure 12 shows the mechanical details of two popular packages, the 14-pin and 16-pin Dual In Line (DIL), package. Note the tab or dot that is used to distinguish Pin-1.

The terminals or pins that physically connect logic functions to your circuit are shown in Figure 12. In order to connect several gates together to form a circuit such as shown in Figure 13, pins 2, 13 and 5 would need to be connected together

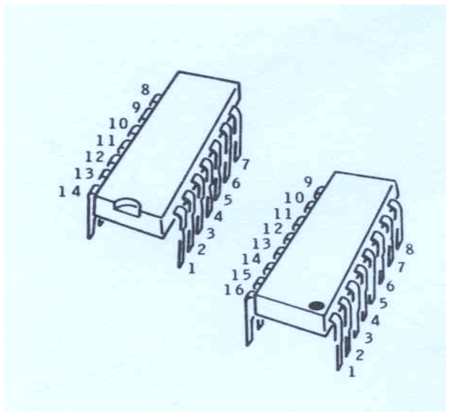
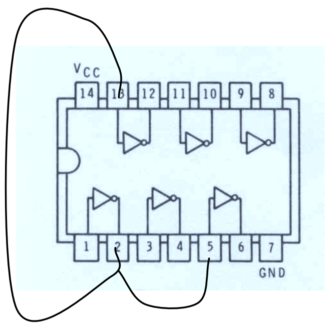
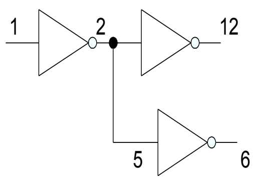


FIGURE 12. 14-Pin and 16-Pin DIL packages.

by a conductor (wire) via a breadboard or Printed Circuit Board (PCB). These wired pin connections are also shown. Pin 1 would form the input (possibly from a switch) and pins 12 and 6 would form outputs (possibly to LEDs).



(a)



(a)

FIGURE 13. Circuit: (a) a *windowed* depiction from a manufacturer's datasheet and associated wire connections; (b) schematic view.

The basic logic circuits that are encapsulated in modular, DIL packages are know as Transistor-Transistor Logic or TTL. This nomination confirms that several basic transistors are being used as a switch to implement several logic functions. The numbers "74" are used to specify the family of TTL functions. Typically several individual logic functions are grouped together. An example is shown in Figure 13 where six inverter gates are used in the one 14-pin package.

The "LS" used also in the part number of the package is an enhancement on the technology and stands for low-power Schottky. Faster and lower-powered logic has been driving the industry for over three decades. For our purposes the"LS" logic families are quite adequate for both digital logic and microprocessor interface applications.

1.11. Logic gates

Logic functions can be divided into two main groups. The first group consists of the so-called "basic" logic functions and comprised of OR, AND & NOT functions. These three gates, function diagrams and corresponding functional equivalents, are shown in Figure 14 below.

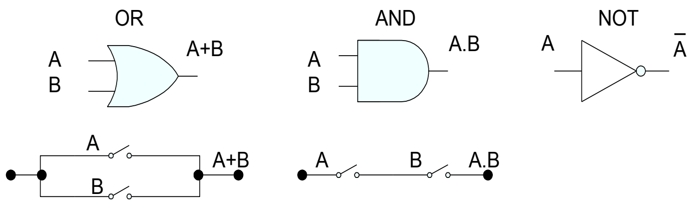


FIGURE 14. The basic logic gates & functional equivalents.

By connecting a NOT gate to the output of an OR gate a NOR gate is produced. Likewise if a NOT gate is connected to the output of a AND gate the functions are combined, a NAND gate results.

Note that the symbol + is used to represent an OR function and the period . is used to represent the AND function. A bar over the function represents a NOT function. Thus the function $\overline{A + B}$ is quite different from $\bar{A} + \bar{B}$ as shown in Figure 15.

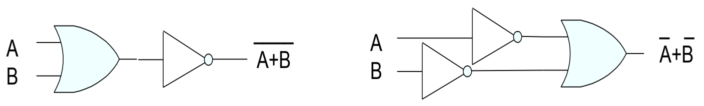


FIGURE 15. Examples of Inverted Output and Input logic Functions.

The NAND logic gate is considered a "Universal Gate" as all the basic gates and any function can be produced solely by using NAND gates.

As we have seen in this section the OR function supports a logic-1 or True result if one of the inputs is logic-1 or True, irrespective of the other input. Another function can be defined which results in a logic-1 or true output but only when one input is logic-1 and the other is logic-0. This function is called an Exclusive-OR function whereas the former is sometimes referred to as an Inclusive-OR. The Truth Table and symbol used to represent the Exclusive-OR function is shown in Figure 16.

Exercise: Implement a 2-input, Exclusive-OR function using five NAND gates.

A	B	$Z = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

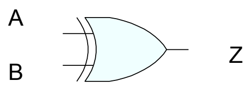


FIGURE 16. Truth Table and Symbol for Exclusive-OR Function.

1.12. Noise margins

1.13. Propagation delay

A delay exists for voltage signals to move or propagate through all logic gates. Ideally engineers would prefer zero propagation delay between the input and outputs however even with the best technology, we are still limited to around several nano (10^9), seconds per gate. This situation is shown in a timing diagram for an inverter gate in Figure 17.

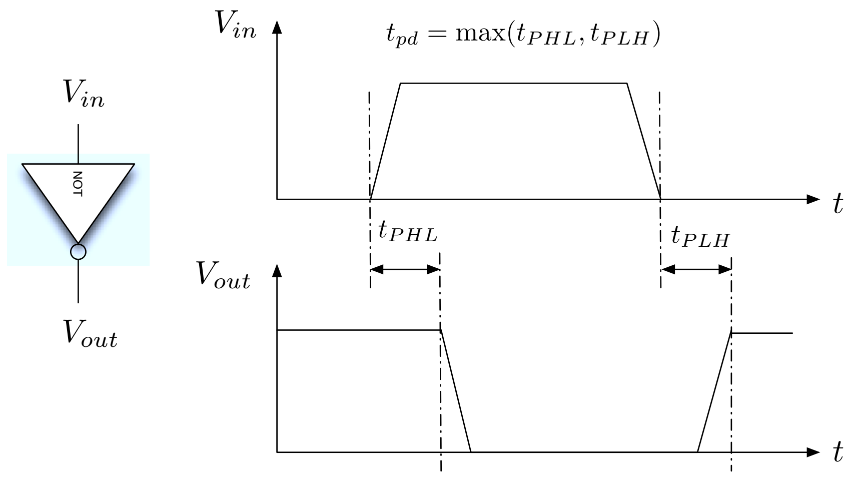


FIGURE 17. Propagation delay for an inverter gate.

From Figure 17 it can be seen that there are three delay parameters. The first parameter t_{PHL}, switches from a zero voltage (logic-0 for positive logic systems) to 3.3 Volts (logic-1 for positive logic systems). The parameter t_{PLH} is used to specify the opposite switching sequence. The propagation delay parameter is specified as the maximum between these two times i.e., if these parameters are different, then the propagation delay is the longest of these times. If logic gates are connected in series, each of their propagation delays forms part of the summation.

1.14. Number representations

The understanding and application of binary, hexadecimal and decimal number representations, including the ability to convert between bases, is of critical importance in the study of microprocessor systems. It is appropriate however to briefly review and expand on this topic as the application of this theory will be directly applicable, not only to digital logic, but more importantly micro-processors. Almost every text on digital logic covers this topic; Roth [**3**] and Mano & Kime [**4**] are two such examples.

1.14.1. Binary fractions. The binary numbering system is used extensively in computers to represent information. It forms the basis of other numbering schemes and allows us to encapsulate much more information in another radix or base such as octal (base 8) and hexadecimal (base 16).

Some details about the binary number system are:

- Only two discrete values, Logic-1 and Logic-0, are used to represent numbers.
- Each binary digit, commonly referred to as a bit, is one place in a binary number and represents an increasing power of 2.
- The least significant bit (LSB) is to the right and if it is set, i.e. logic-1, it has the value of $2^0 = 1$.
- Binary weightings are enabled by setting the appropriate 1s in a binary sequence.
- The number of bits used to represent a decimal (Base-10) number determines the range of a binary number or the resolution of a binary fraction.

Examples:

$0111_2 = (0 \times 8) + (1 \times 4) + (1 \times 2) + (1 \times 1) = 7_{10}$

$11110_2 = (1 \times 16) + (1 \times 8) + (1 \times 4) + (1 \times 2) + (0 \times 1) = 30_{10}$

1.14.2. One and two's complement numbers. The binary code used in the examples of the previous section cannot be used to represent signed numbers. The twos complement numbering system modifies the binary system to include negative numbers by interpreting the most significant bit (MSB), as negative.

Remember from your previous studies the 1s complement of a binary number is obtained by simply reversing the bit pattern i.e., for each 0 replace this with a 1 and visa-versa. To obtain the twos complement of a number you add 1 to the 1s complement.

Twos complement numbers:

- Follow the binary progression except that the MSB is interpreted as the highest ordered, negative number. All other bit orders to the right are interpreted as positive numbers.
- Can have any number of bits more bits allow a larger range of numbers to be represented.

- Allow adder circuits to subtract numbers by adding 2s complement numbers.

Examples:

$0111_2 = (0 \times -8) + (1 \times 4) + (1 \times 2) + (1 \times 1) = 7_{10}$

$11110_2 = (1 \times -16) + (1 \times 8) + (1 \times 4) + (1 \times 2) + (0 \times 1) = -2_{10}$

Two operations are useful in working with twos complement numbers:

- The ability to obtain an additive inverse of a value.
- The ability to load small numbers into larger registers (by sign extending).

More will be said on these points when we start our discussions on Stellaris microprocessor.
Examples:

Convert 27_{10} and 23_{10} to 2s complement numbers and subtract the latter from the former. Now subtract the former from the latter. Remember to re-complement and add one if you dont get a carry.

1.14.3. Binary fractions. Signed and unsigned binary fractions can be treated the same way as signed and unsigned binary integers however rather than the power increasing to the left, the reciprocal power increases to the right. In other words smaller fractional representations of numbers are provided as the exponent increases in size.

Combine this with the twos complement representation that was outlined in the preceding section and signed, binary fractions can be represented in a computers memory.

For example if our earlier example were used, 01112 as a binary fraction would be:

$$\begin{aligned} 0111_2 &= (0 \times -2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= 0 + 0.5 + 0.25 + 0.125 \\ &= 0.875_{10}. \end{aligned} \tag{1.6}$$

If the MSB were set this then becomes a negative binary fraction and the result would then be:

$$\begin{aligned} 1111_2 &= (-1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= -1 + 0.5 + 0.25 + 0.125 \\ &= -0.125_{10}. \end{aligned} \tag{1.7}$$

By multiplying a fraction by 32K (32768_{10}), a normalised fraction is created. The chart in Figure 18 shows how this normalisation is performed on 16-bit data. Also shown is the range of representation i.e., all binary, signed fractions are restricted to between $\approx +1_{10}$ and 1_{10}.

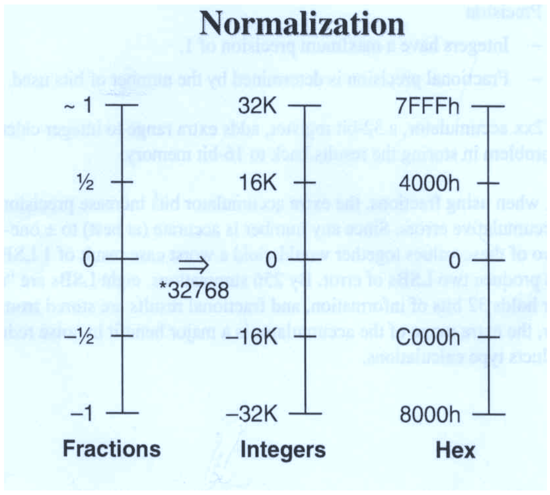


FIGURE 18. Signed, Binary Number Representation (Courtesy of Texas Instruments).

CHAPTER 2

Introduction

Embedded systems are everywhere - the average automobile, for example, supports around 100 individual microcontrollers. Each variant is selected to provide a range of capabilities, depending on their function. To adjust a wing-mirror a simple microcontroller is required. A simple 8-bit Atmel ATMega-8 microcontroller, for example, is ideal for this task. However, it would not make sense to use a 32-bit Cortex M-3 (CM-3) microprocessor for this task - in fact, it would be extremely wasteful of both resources and money. A far more appropriate use of a CM-3 would be for an engine management system.

This chapter firstly provides a brief history on digital computers. Secondly, the differences between computers, microcontrollers and microcomputers is outlined. Lastly,

A microprocessor is used as part of an embedded system when

(1) Other system components are required. For example, an EEPROM memory is required to periodically acquire nonvolatile data;
(2) a minimal delay is required to act on external (to the microprocessor) events;
(3) the end application does not change. For example, a toaster is always a toaster, and cannot be adapted (well, not easily anyway) as a hair dryer.

The remainder of this Chapter revises some material from earlier courses. The next chapter details the Cortex M-3 microprocessor that will be used for several projects in this course.

2.1. Basic Computer Structure

There is a huge variation in computer components. However, it is possible to define some common, highly complex modules, that are required for a stored program to be executed on a digital computer.

The basic components used in a microcontroller are shown in Figure 1.

The complex modules listed in Figure 1 form the basis for ongoing discussion.

2.2. Computers - An Historical Perspective

Computers originated in the 17 century with the proposal by Charles Babbage for a *difference machine*. Babbage's proposal was to make a machine that

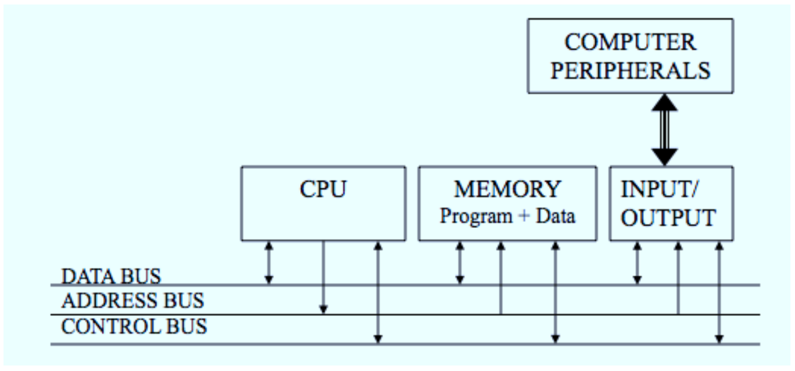


FIGURE 1. A block diagram of a microprocessor, adapted from [**5**]

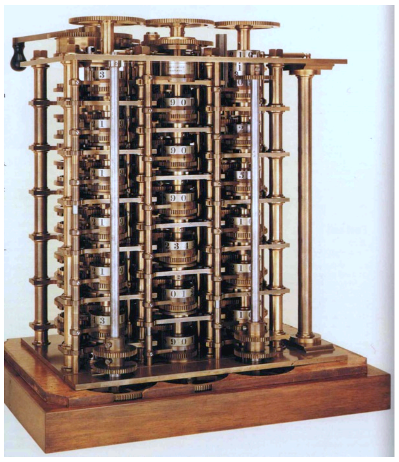
could compute astronomical and mathematical tables. Due to the complexity of the project, combined with limited manufacturing capabilities (and money) at the time, Babbage's engine was never built. However, the concept of this mechanical computer was demonstrated only recently by Doran Swade[1]. In fact, a practical, modern day use for Babbage's difference engine has only been realised[2].

With the development of the thermionic valve (vacuum tubes) in the 19th century, attention turned to electronic computation. However, rather than the valve being used in an analogue sense (combining and controlling various voltages rather than turning a series of mechanical wheels with a crank), switching to either an "on" state or an "off" state was deemed more efficient. This breakthrough occurred during the 20th century.

The world's first programmable, electronic, digital computer used a lot of valves, but rather than switching them "on" and "off", making them unreliable, a brilliant Post Office technician found a way to make the thermionic valve more reliable by keeping the values partially "on" and never turning them "off", i.e., keeping them warm. This significant contribution was make by Tommy Flowers and resulted in the design and construction of the Colossus computer [**6**]. Combined with the genius of Alan Turing as a cryptanalyst,Tommy Flowers and a hand-picked team at Bletchley Park were able to break the German's military encryption code, such as used in the Enigma machine, during World War 2. For more details on these historical events, refer to Jack Copeland's, "A Brief History of Computing" [**7**]. Images of early forms of computation are shown in Figure 2.

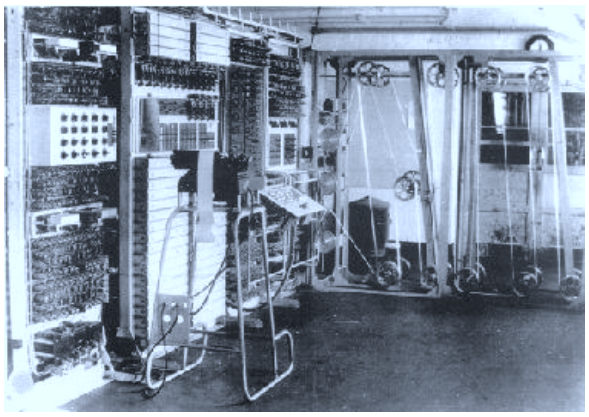
As technology improved, digital computers became more efficient (and reliable) when they were made out of transistors on circuit boards. In 1960, simple analogue circuits, such as operation amplifier μA709, could be integrated on silicon. This paved the way for the integration of digital circuits. Thus, printed circuit boards

[1]http://en.wikipedia.org/wiki/Doron Swade
[2]http://www.newscientist.com/article/dn19440-steampunk-chip-takes-the-heat.html



(a)

(b)



(c)

FIGURE 2. Early examples of computation: (a) Babbage's difference engine; (b) A German Enigma machine; (c) The Colossus: the World's first programmable electronic computer.

could be shrunk down several orders of magnitude forming a digital integrated circuit. Complex modules, in the form of discrete logic components, could be used to form more complex circuits without users knowing every fabrication detail. This was the age of the integrated circuit computer.

In 1971 the first 4-bit microprocessor, the Intel 4004, was announced. A microprocessor was the result of integrating more and more digital computational complexity on to a single silicon chip. This complexity comprised a central processing unit (CPU), a scratchpad area for operations, and input and output (IO) circuit. the Intel 4004 was a 4-bit microprocessor (essentially a CPU with IO), the clock speed was 740 kHz A block diagram shown in Figure 3 is for the Intel 4004.

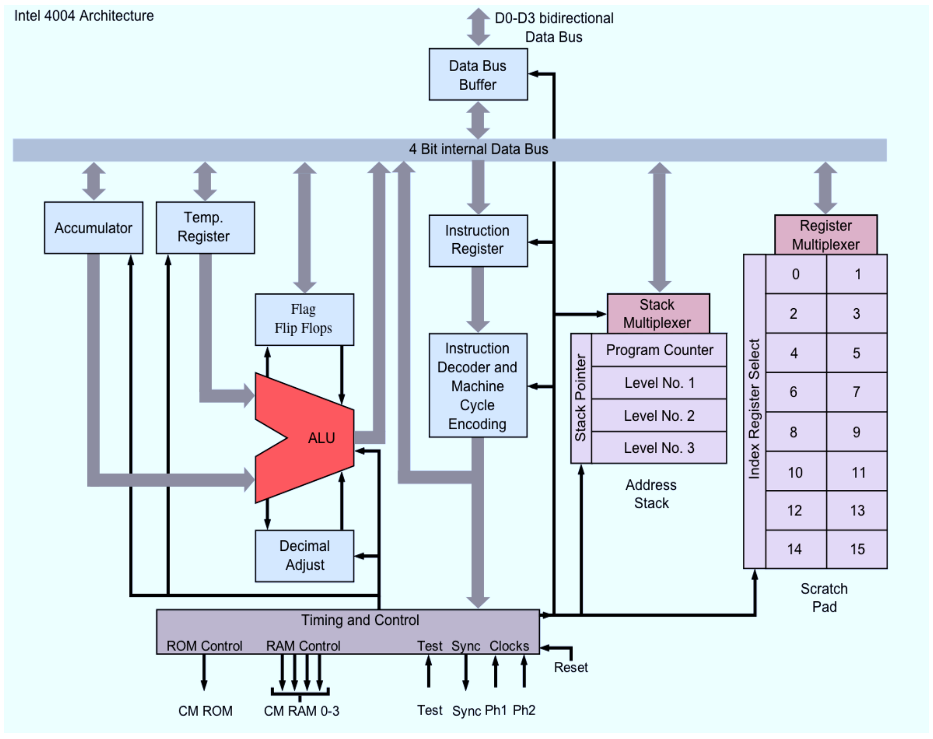


FIGURE 3. Block diagram of the World's first microprocessor, Intel 4004.

Steve Jobs and Steve Wozniak in 1976 released the Apple 1. This used an early MOS 6502 microprocessor. In 1983, the Apple-11e was released. This also used a 6502 microprocessor but was clocked at 1.023MHz and supported a 64KB memory. In 1981, the IBM PC was released. This supported an Intel 8088 microprocessor and was clocked at 4.77 MHz.

Between 1971 and 1977 chip manufacturers started to make a distinction made between embedded and reprogrammable computation. Even though the 4004 and 6502 could be used for embedded applications, these required a host of other integrated circuits (mostly logic devices) to interface to memory chips. Thus, with higher and higher levels of single-chip integration occurring at this time, read only and read/write memory were incorporated on the same substrate as the CPU and IO. Texas Instruments offered the TMS7000 in 1971, Intel introduced the 8048, and Motorola with their 6809 started targeting the new and emerging embedded system market. Thus, the microcontroller[3] was born.

[3]http://en.wikipedia.org/wiki/Microcontroller

2.3. Moore's Law

Moore's law has been a benchmark for the introduction of new architectures; where more transistors equate to supporting enhancements, such as fast, on-chip memory, fabrication methods have had to keep pace. Moore's law states that there is a doubling of transistors required to implement these additional hardware capabilities, every two years. As shown in Figure 4, the number of transistors used to implement popular microprocessor types have kept pace with the theoretical plot of plot shown as Moore's law.

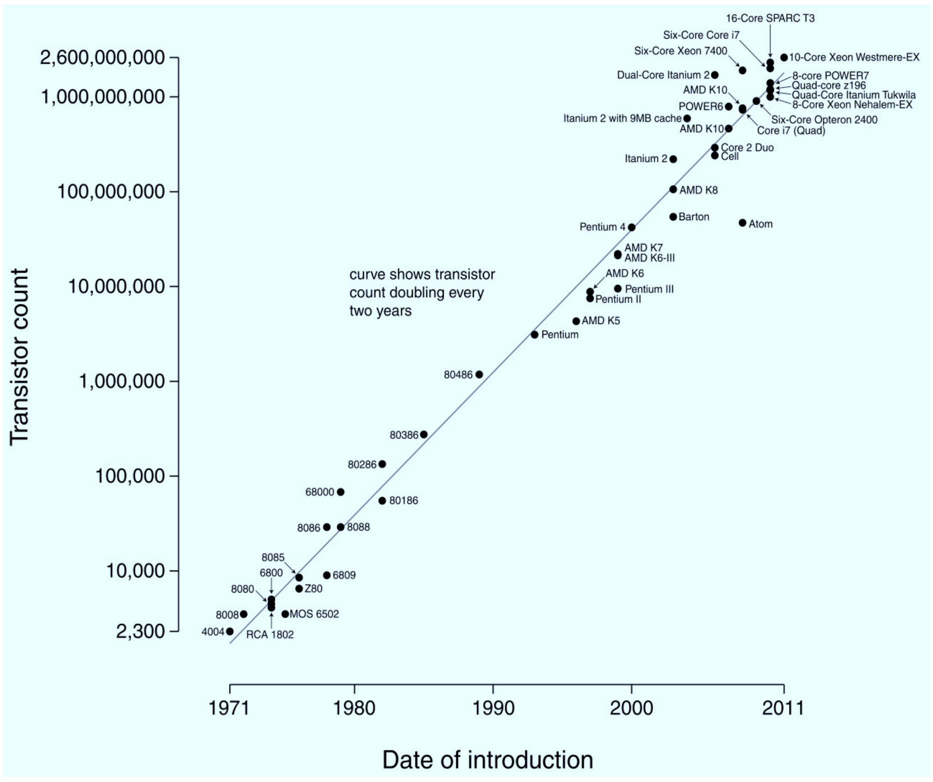


FIGURE 4. Microprocessor advancements according to Moore's law, [**8**].

2.4. Computers Microprocessors and Microcomputers

As an example of how early microprocessors were used, let's discuss a basic operation of adding two numbers together. Since we want to make the addition operation generic, i.e. we don't want a to always add the same pair of numbers, lets define a pair of *operands* and assign them letters, x and y. The addition *operation* can be defined as an instruction. A series of bits forming a *code* word is usually defined for this. The result is an *operation code* or *opcode* for short. An adder is can be defined as a module that takes two operands, x and y, adds them together, and provides a result, z, and may also generate a carry, c, if the result exceeds the data word (a set number of bits) each operand represents.

Adder modules can be used for subtraction, and special structures can be used for repeated operations. Thus multiplication and division cover the basic arithmetic operations. Logical operations are also possible, such as `AND` and `OR` operations. All of these operations can be defined within a rather complex piece of hardware known as a *central processing unit* or CPU.

2.5. Selecting a Microcontroller

Selecting a microcontroller for a specific application is a simple process. There are many parameters to consider. For example, clock frequency is often an important parameter, however the number (typically in the millions) of instructions that can be executed is typically given. Another is the number of input and output ports (IO) is another, although this often depends on specific family members. The most general classification given to microcontrollers is the bus width. This describes the size of the maximum data word that can be processed by the microprocessor. Typical bus widths vary from 8-bits to 32-bits. The majority of small, embedded applications employ 8-bit data buses (these are still the very popular with designers). However, nowadays more engineers are using ARM-based processors that use a 32-bit with bus.

Some of these parameters over a wide range of microcontroller (MCU) and microprocessor units (MPUs) is given in Table 1.

There are a few points to note about the comparison chart in Table 1:

- Early microprocessors did not support many, if any, peripherals.
- Performance on early MPUs was measured in MIPs but nowadays is typically measured in MIPs/mW or MIPs/MHz.
- Segmented memory models has given way to a linear memory configurations.

Since their introduction in the 1970's, certain trends have emerged regarding microcontrollers (MCUs) and microcontrollers (MPUs). These trends are summarised in Table 2.

TABLE 1. Comparison chart of early microprocessors and late edition microcomputers used in embedded systems; an updated extract from [**9**].

Processor	Bus Width	Lagest External Memory	Internal Peripherals	Speed (MIPS)	Type	Year of Manuf.
Intel 8051 family	8	64 KB program + 64 KB data	3 Timers + 1 serial port	1	MPU	1980
Freescale MC9S08QB8	8	8 KB Flash + 512 B data	3 Timers + ADC, SPI	20	MCU	2010
Intel 8086 family	16	32KB program + 512 B data	All external	5	MPU	1978
Texas Instr. MSP430x3	16	8 KB Flash + 512 B data	3 Timers + ADC, SPI	16	MCU	2000
Motorola 68000 family	32	4GB	Various	10	MPU	1978
Texas Instr. Stellaris 1xxx	32	4GB	Timer/PWM ADC, SPI, +	60	MCU	2010

TABLE 2. Trends in microprocessor and microcomputer architectures over the last three decades.

Microprocessors (MPUs)	Microcomputers (MCUs)
CISC $\rightarrow$ RISC	CISC $\rightarrow$ RISC
Higher clock rates	Higher clock rates
single $\rightarrow$ multiscalar	Parallel I/O $\rightarrow$ serial I/O
Massively parallel structure	Larger on-chip FLASH and SRAM
Increases in data- and Address-widths	Increases in data- and Address-widths
The register file is increasing	The register file is increasing
More complex branch prediction schemes	Increase in on-chip peripherals
Multiple level caches	Support for distributed memory

CHAPTER 3

Microcontroller Architecture

In this section the basic microcontroller architecture is developed. The principal components are outlined in terms of computer architecture. These architectural descriptions are designed to be generic, i.e., not specifying a particular manufacture or part-number. For details on the Cortex-M3 CPU core, in addition to the peripherals that comprise the Stellaris series of microcontrollers, refer to Chapter 4.

3.1. The Central Processing Unit

The central processing unit (CPU) is basically the brains behind the controller. It controls how data is transferred between other modules, and, just as importantly, *when* it sequences data to be moved. As a result, two primary components of a CPU are the *datapath* and *control unit*. These are shown in Figure 1, including the feedback paths between each module and IO transfers.

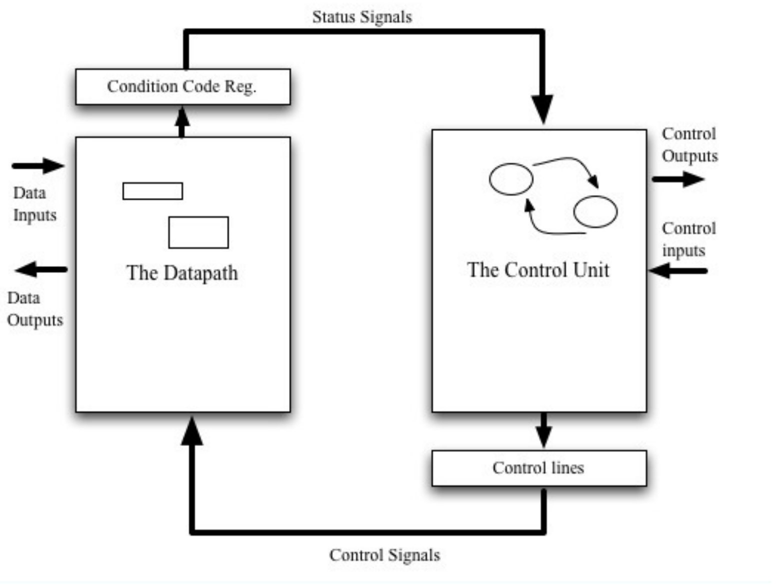


FIGURE 1. A block diagram of a central processing unit.

Further details on this is provided by [**9**].

3.2. The Hardware Software Interface

For a team of engineers working on a computer system, it is common to have different perspectives on development. For example, a software engineer will typically work with a programming model supplied by the manufacturer of the computer. Typically, a high-level of abstraction is adopted, where the programmer prefers to deal with concepts and ideas, rather than implementation details. However sometimes optimisation, particularly when working on a real-time embedded system, require a programmer to consider low-level programming requirements. This is when assembly language, i.e. the mnemonic language used to represent a set of basic computer instructions, may be used. Rarely, if at all, does the the native *machine* language of the computer hardware is used.

However, a hardware engineer is often more interested in the enhancements they can obtain from a given computer architecture. For example, the instruction set architecture (ISA) is a specification outlining how the computer operates. This specification describes how the computer operates and comprises a list of atomic operations that can be performed. Typically, a high-level language will use one or more of these operations to define a particular construct. The translation of the ISA into the signals that connect the hardware components together can itself be programmable. This is often referred to as *firmware*, were the actual instructions themselves are programmable. The term used to describe this process is *microprogramming.* A particular, and historic type of computer architecture that employs such translation into hardware is referred to as a complex instruction set computer (CISC). When this transfer is directly coupled to the hardware, the name given to this architecture is reduced instruction set architecture (RISC).

The schematic in Figure 2 shows two perspectives of computer architecture. From a programming perspective, on the right side of the diagram, the entry level can be either high-level or assembly level. However, in terms of the machine specification the computer architecture can be defined in terms of either RISC or CISC. This is shown on the left of the diagram, where the ISA of a computer can be interpreted as a flow of date between registers. The name given to this view is *register transfer language* (RTL). The programmability at the firmware level is a distinctive feature of CISC architecture.

3.3. Instruction Set Architectures

The instruction set architecture (ISA) is specific to each computer series and describes the processor that is usable by the programmer or development tool manufacturer. The ISA is essentially the interface between hardware and software.

There are several common types of ISA. These are typically classified in terms of the number of addresses they support. These include:

- Zero address: This is also referred to as a *stack* architecture.
- One address: Also known as *single accumulator* architecture.
- Two address: This architecture may have one or more accumulators, however two address buses are used. One address is used for a single operand, the other doubles as the other operand and destination operand.
- Three address: known also as a *load-store* or *register-to-register* architecture.

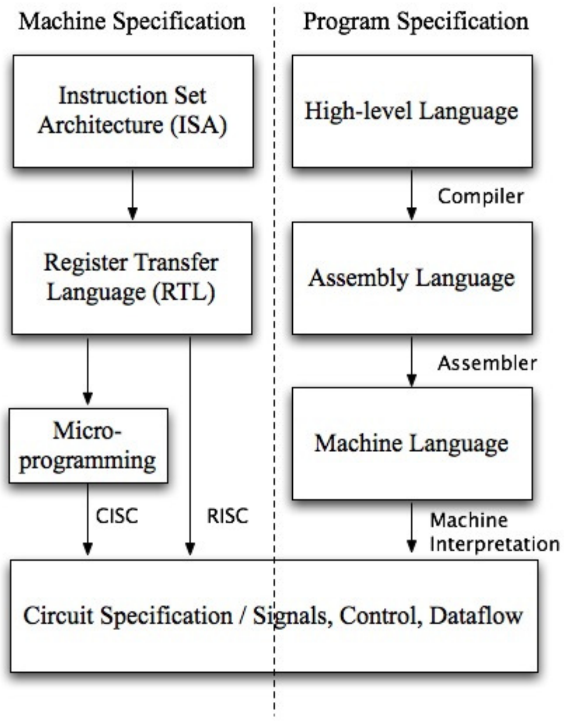


FIGURE 2. A block diagram of a central processing unit.

All ARM processors are three address architectures. This ISA is discussed in detail in Section 4.3.

3.4. Digital Input and Output (IO)

A computer, such as a microprocessor, is useless unless it has the ability to accept input or provide output. Ports are provided on microprocessors to allow information to be input and output. Information can be in the form of digital or analogue. This section focuses on digital information, where a logic-1 or logic-0 is represented by a specific voltage range. The microprocessor will accept or output a voltage signal if it is within a specific range. A datasheet for a microprocessor, such as the Stellaris range of micros, can be referenced to provide this information.

If several ports on a microprocessor supports a common function for group input and output, these are typically referred to as *parallel input output* (PIO) ports. If these port also have a non-specific, i.e. *general* purpose, these are typically referred to as *general purpose input output* ports (GPIO). A collection of GPIO ports can be used to input a data word. For example, if 8 ports are used to input one bit each, collectively this group of ports can simultaneously be used to input one 8-bit data word.

The problem with GPIO ports used for parallel input and output is that many ports are required - especially when dealing with 32-bit data words. As a result, information can also be input in serial form, i.e., one data bit at a time. Both of these configurations are discussed in this section.

3.4.1. Parallel IO. Microcontroller (MCU) ports can either provide or accept a limited supply of current for interfacing light-emitting diodes (LEDs) or switches. Depending on the circuit configuration, ports can be used to either source or sink current. Resistors are used to limit the amount of current flowing into (sinking), or out of (sourcing), a current source/sink such as a microcontroller input-output (IO) port.

Generally, most digital circuits can sink more current than they can source, however this may not apply to MCU ports. In addition to current limitations on GPIO ports, the total power dissipation of the MCU (as per the datasheet) should never be exceeded.

The circuits shown in Figure /reffig:gpio are commonly used for interfacing microcontrollers to LEDs and switches.

With reference to Figure 3 (a):

(1) The MCU supplies source current if a logic-1 is written to the output port.
(2) LED forward voltage drop (except Blue LEDs) is typically, 1.2 Volts.
(3) The value of the current limiting resistor, RL, is typically calculated to provide about 8 mA forward current through an LED; some IO ports on various MCUs may not provide this.

With reference to Figure 3 (b):

(1) In this configuration, Vcc supplies current that is sunk by the MCU, i.e., if a logic-0 is written to the output port.
(2) This is a common configuration for when a port can sink more current than it can source.
(3) Calculation for RL is identical to that of Item-3 in the aforementioned list, (a).

With reference to Figure 3 (c):

(1) The current limiting resistor RL is used to supply current to the MCU when switch SW1 is open circuit. The MCU then reads this input as a logic-1 as it is tied-high through RL.
(2) When SW1 is closed, the MCU reads the switch as a logic-0, since the input port is shorted to Ground through RS. The series resistor, RS, provides protection in the event the port is incorrectly configured as an output. Values for RS should not exceed 100 Ω.

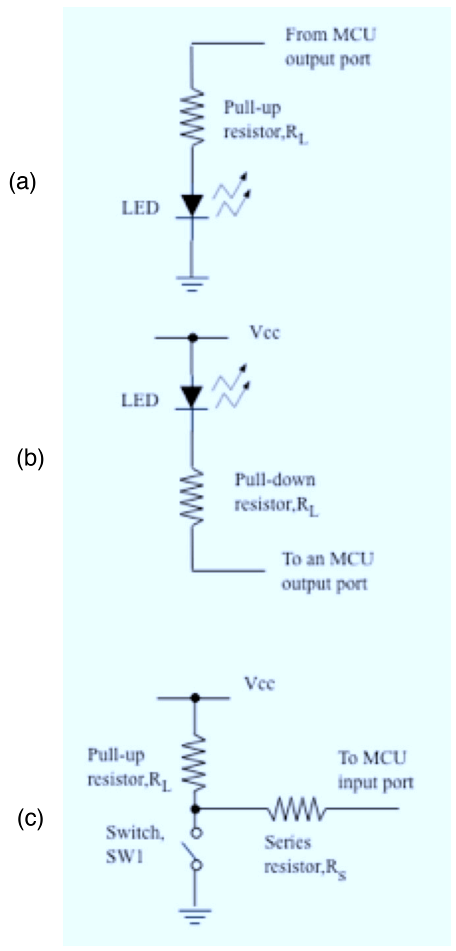


FIGURE 3. Examples of common GPIO interface circuits: (a) LED output using a *pull-up* resistor; (b) LED output using a *pull-down* resistor; (c) a switch input circuit.

Since most microprocessor ports are fabricated using CMOS technology, the push-pull output circuits need to be held either at a logic-1 or logic-0 state by an internal MOSFET. Such a configuration is commonly called an active pull-up.

Most microprocessors provide internal, programmable pull-up resistors can be enable, on reset, for all GPIO ports. Note, however, that in terms of current capability, these tend to be weak alternatives. Additionally, IO ports on some microprocessors are not consistent, in terms of providing identical source or sink current capabilities. For example, on the SAM7, ports PA0-PA3 can provide 16mA, whereas PA4-PA16 and PA21-PA31 can supply 8mA.

To conserve power consumption, many microprocessors typically initialise all GPIO ports at reset to inputs, and enable pull-up resistors on all ports. This is done to provide a static condition for the MCU on power-up. Your software, however, should explicitly define all ports as either input or output ports.

In addition to the power consumption considerations, unused GPIO pins that are left floating can increase the microprocessors susceptibility to electrical noise. With the exception of the SAM7, MCUs that do not provide a start-up feature on all unused GPIO pins should be configured as inputs, and, if provided, enable the corresponding internal pull-up resistor. If internal pull-up resistors are not supported, configure the unused port as an input, and use a weak, external pull-up resistor to configure the port.

In summary, the following should be considered when using GPIO:

- Read the datasheet for internal configuration and initialisation.
- Do not exceed the datasheet specifications for sourcing or sinking current through a general-purpose input-output (GPIO) pin on MCUs such as the SAM7, current limits vary for individual pins.
- Consider the maximum power consumption of the device, especially when all GPIO pins are driven to their maximum specification.
- Debounce all switched inputs using a low-pass filter. This can be done in either hardware (capacitor, Schmitt trigger, or latch), or debounce in software by using a software delay routine. Typical delays should be between 10 20 milliseconds.
- Interrupt sources on GPIO are particularly susceptible to false triggering.

3.4.2. Serial IO.

3.5. Memory Structures

Memory structures evolve from combinational logic. The basic read-only memory (ROM) is non-volatile memory. This essentially means that power is not required to maintain the memory pattern.

Digital memory can be broadly grouped into three classifications. There are:

- Random Access memory (RAM).
- Read-Only memory (ROM).
- Programmable logic devices (PLD).

3.5.1. Random Access Memory. Random access memory (RAM) does not suggest your data is randomly accessed in memory locations. In fact, it is the complete opposite.

RAM can be defined as either static or dynamic. The dynamic is the simpler of the two so we will start with this.

3.5.1.1. *Dynamic RAM.* Dynamic RAM (DRAM) is volatile memory in the application of a voltage periodically for the memory cell to retain information, i.e., data. it requires a voltage to in terms of a single read/write memory bit. An example of a DRAM cell is shown in Figure 4.

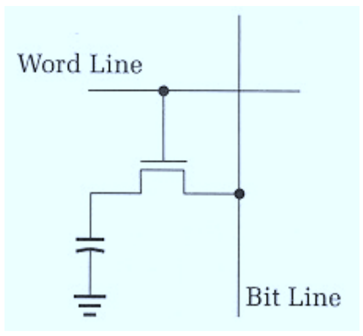


FIGURE 4. A single DRAM cell.

In its simplest form, we assume the capacitor shown in Figure 4 holds the value, i.e. either a logic-0 or a logic-1, of the memory cell. If the *Word Line* signal is high, then the transistor shown in the schematic is on. Whatever logic value is shown on the *Bit Line*, the input, is transferred to the output in the form of either maintaining the charge on the capacitor, thereby storing a logic-1, or discharging the capacitor, thereby storing a logic-0.

Due to their simple construction, DRAMs provide the maximum density of all semiconductor memory components. They are also very fast. However, they require a burst of voltage to maintain the charge on the capacitor.

DRAM cells can be connected as a data word. For example, 8 DRAM cells can form a byte of data. Show how this configuration can be constructed.

3.5.1.2. *Static RAM.* Like DRAM, static RAM (SRAM), is still volatile, i.e., it requires a voltage in order to retain data contents within each cell. However, unlike DRAM, SRAM is typically lower power. A disadvantage with SRAM is that it is more complex to make. For example, the schematic in Figure **??** describes how a single SRAM cell can be constructed using several digital logic gates and a flip-flop.

This can be sumarised in terms of a structure as shown in Figure 6:

The memory structure shown in Figure 6 shows three broad groups at the top-layer, where the random access memory is shown as *volatile.* This suggests that the contents of the RAM memory device are susceptible to modification. Indeed, this is the case, since a RAM device requires power in order to maintain the integrity

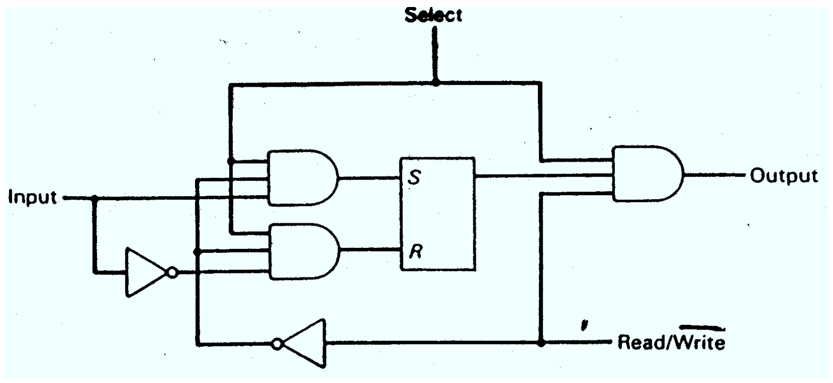


FIGURE 5. A single SRAM cell.

of its data contents.

However, the next group shown in read only memory. This is the simplest memory structure since it does not require power to retain its data contents. The term used for this is *non-volatile* memory.

3.5.2. Memory Maps. A memory map is simply a diagram showing how the memory of a complex device, such as a microprocessor, is arranged. For example, the memory requirements of a fictitious microprocessor, referred to as the *microbuz*, is defined as follows:

- A 1K[1] × 16-bit internal SRAM starting at memory location `0x0000`.
- An 8K × 16-bit internal (on-chip) FLASH memory, starting at memory address location `0x8000`.
- A 16K × 16-bit external static RAM, starting at memory address location `0xA000`.
- A 20 × 16-bit ROM vector table starting at `0xE000`.

A memory map of the Microbuz computer can be given as shown in Figure 7

3.5.3. Memory Structures.
3.5.3.1. *Stack.* A stack structure is simply a memory structure that can grow or reduce in size, depending on data or operations that are *pushed* or *popped* onto the *top of stack* (TOS), respectively. Stacks are also known as a *last-in first-out* (LIFO) abstract data type. Because stacks are read from and written to, memory allocated to implement a stack must be read/write memory, i.e., RAM.

Stack also provide a valuable aid in sequencing operations that are required for subroutine and/or interrupt control.

[1]Since the size of most memory is given in Base-2, 1K is actually short-hand for 1024

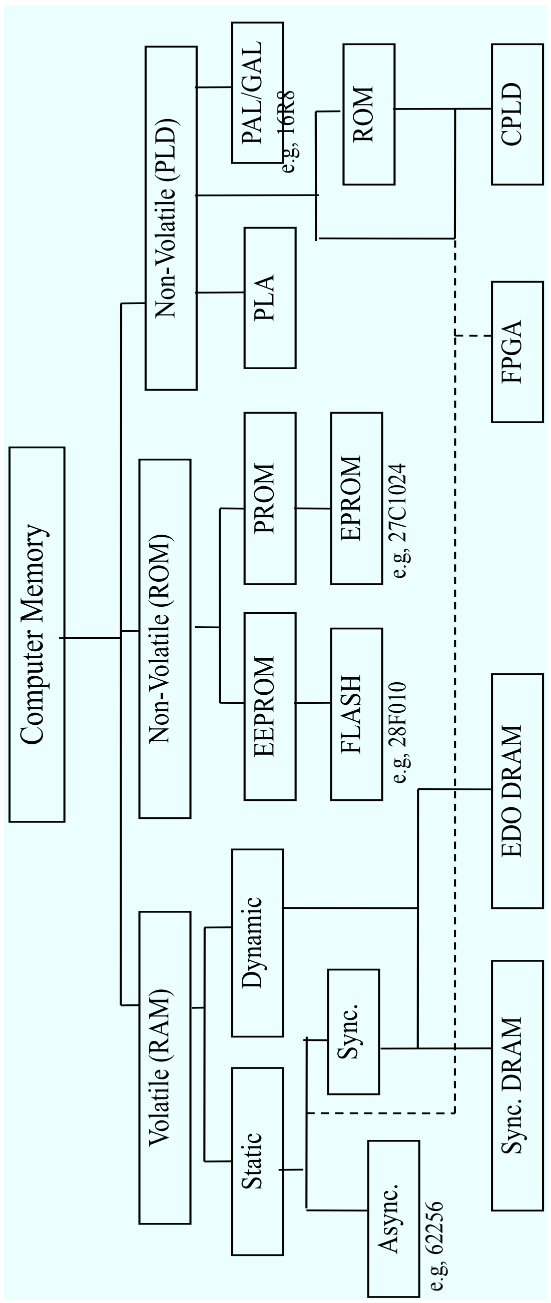


Figure 6. A memory hierarchy showing three broad groups of devices.

3.5.3.2. *First-In First-Out (FIFO).*
3.5.3.3. *Circular Buffers.*

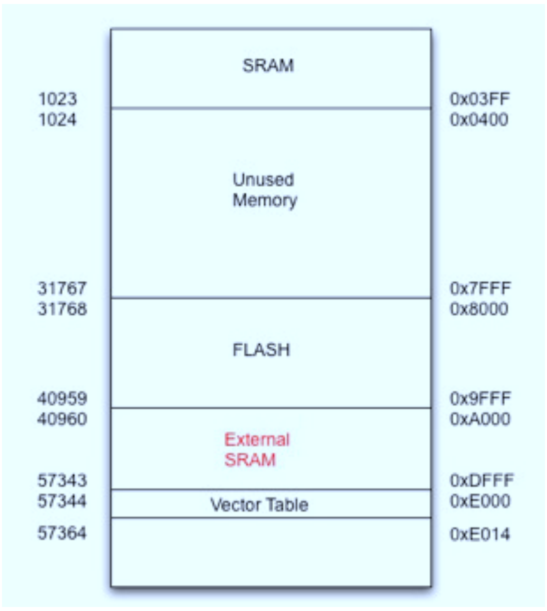


FIGURE 7. A memory map for the *microbuz* microcontroller.

3.6. Microprocessor Peripherals

3.6.1. Analogue-to-Digital Converter (ADC). Converting a continuous voltage to a discrete value, represented by a finite number of bits, is the job of an analogue-to-digital (ADC) converter. ADCs can be used as either a stand-alone monolithic integrated circuit, i.e. a separate chip, or be integrated on the same silicon substrate as a CPU and used as part of a single-chip microcontroller. An example of the latter is the Stellaris series LM3S1968 microcontroller by Texas Instruments. This section provides a general overview of ADC operation. Specific use of the ADC found on the LM3S1968 is provided in Section 4.5.1.

To help understand how the conversion process is undertaken, it is important to understand quantisation. To do this, a transfer function of an ideal 3-bit ADC is shown in Figure 8.

3.6.2. Timer Modules.

3.7. Software Tools

3.7.1. Assemblers.

3.7.2. Loaders.

3.7.2.1. *File Formats.*

3.7.3. Debuggers.

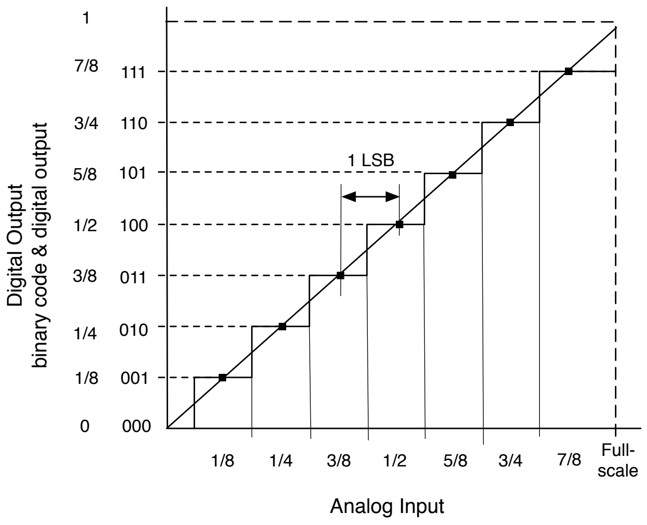


FIGURE 8. The transfer function of an ideal 3-bit ADC.

3.7.4. JTAG. JTAG is an acronym for the Joint Test Action Group and is the Standard Test Access Port & Boundary-Scan Architecture (IEEE 1149.1) for testing circuit boards [3]. More recently, manufacturers of VLSI and ULSI microchips have incorporated JTAG into the firmware of each device. JTAG uses a simple serial interface, much like that of a synchronous serial port used on many microcontrollers. Information about the unit under test is extracted, including the state information of registers and memory locations of complex microchips. When a specific scan code is sent to the microprocessor using a hardware interface, information about specific hardware components, for example, the contents of a memory location or register, is returned for evaluation via the same hardware interface.

A debugger, such as the gdb or Insight can be used to send the appropriate scan chains to the microprocessor, and by incorporating the JTAG protocol and interface scan the internal modules within microprocessors for the state of internal registers or memory locations. This information is then send back to the debugger running on a host computer and the results so they can be displayed on either a console or a window if the GUI version of gdb, i.e., the Insight debugger, is used.

Since PCs do not support a direct JTAG interface, a toolchain such as OpenOCD discussed in Section 4.6.2.1, must operate through an existing interface. Popular examples have included serial and parallel port interfaces. More recently, USB has been the interface of choice due to its higher speed capabilities.

The gdb debugger has always supported a software simulator and serial interface for hardware debugging. Before JTAG a small monitor ROM was run in the processor that provided information on the status of registers and memory locations. As devices became more complex, sophisticated methods to test parts were introduced. JTAG became the default standard and gained popularity in extending its original functionality for the purposes of debugging.

Special pins were supported on microprocessors that allowed a serial interface for JTAG connections. Since the JTAG interface has been built into the chip as an aid to manufacture, JTAG support and an interface to either parallel or USB is a more responsive method and provides a more flexible method of debugging.

3.7.4.1. *The Hardware Interface.* OpenOCD supports either parallel port, or USB interface to a host PC. Typically, a PC is used as a host development environment, running a development software application such as gdb or Insight. Converting the JTAG bi-directional signals to an interface supported by most current PCs has traditionally been challenging. Over the last few years a simplified parallel port interface has been detailed.

3.7.4.2. *The JTAG to Parallel Port Interface.* The JTAG to parallel port interface consists of a buffer and software driver that allows the debugger running on a host PC to communicate to the microcontroller. Some chip manufactures, however, have limited the functionality of their JTAG interface and only offer download capabilities - this does not apply to ARM devices. Low-cost parallel port modules can be purchased for under $30 or constructed for less that $10. Performance has not been an issue, however, with the convenience of USB, in addition to the removal of parallel ports on many laptops, an alternative has been developed the JTAG to USB interface.

3.7.4.3. *The JTAG to USB Interface.* Over the last few years a company known as FTDI has manufactured a JTAG to USB device known as the FT2232L [?]. This device greatly simplified the design of a JTAG to USB interface and provided what many would argue as a more convenient method of embedded software development.

A dual serial interface provided translation between JTAG on the target end, and USB on the host PC side. This flexibility requires the manufacture of a small interface board. This interface board can be seen in the labs on benches 15-20 and has been manufactured in ECE specifically for Hardware Engineering courses.

Figure 9 shows in schematic form the philosophy behind JTAG and the ability to daisy-chain complex devices together for either testing or debugging. The pin definitions used for the JTAG interface are as follows:

- TDI (Test Data In)
- TDO (Test Data Out)
- TCK (Test Clock)
- TMS (Test Mode Select)
- TRST (Test ReSeT) optional.

3.7.5. Compilers.

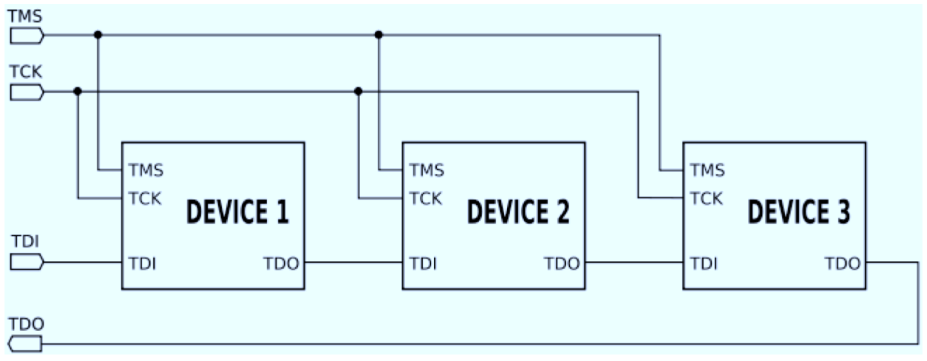


Figure 9. The JTAG scan chain.

CHAPTER 4

Stellaris Architecture

4.1. Introduction

Back in 2009 Texas Instruments (TI) acquired Luminary micro, giving TI access to one of the first manufacturers in the World of Cortex-M3 (CM3) CPU cores. TI call their 32-bit family of CM3 processors the *Stellaris* microprocessor series. The LM3s1968 microprocessor from this series will be used in undergraduate teaching in 2011. These pages will focus on the build tools and OpenOCD toolchain used for general development. It is hoped that this information will also be useful to those students, both undergraduate and postgraduate, who are considering or are actually using this microcontroller series for project work.

An extension to these pages for Stellaris use in specific course work, such as for Software Engineering 1, is provided by linking to the ENEL323 Page.

4.2. Overview

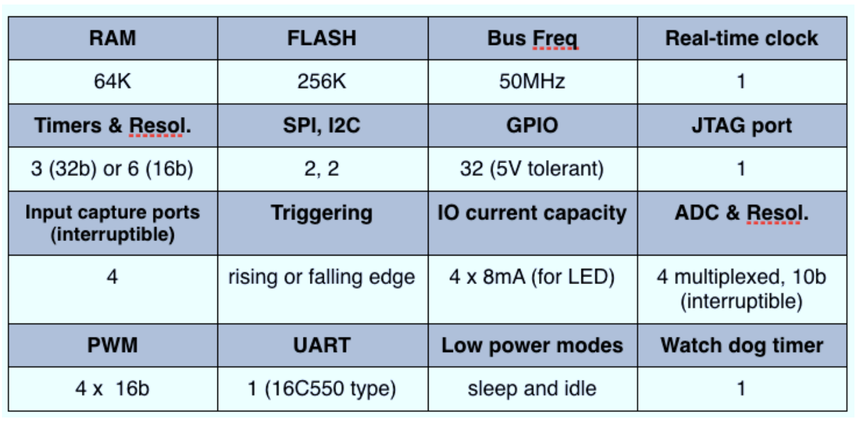
Similar to ARM7 and ARM9 cores, the Cortex-M3 (CM3) architecture is based on a standardised 32-bit datapath, ARM core. However, the CM3 CPU has been extended to include an interrupt structure, system tick (SysTick) timer, enhanced debug capabilities, and a 4GB memory map. The CM3 supports a Harvard architecture, implemented using a multiple internal bus structure, yet it provides a simplified ARM7/9 programming model. However unlike ARM7/9 cores, the CM3 allows unaligned data access; this provides efficiency in terms of code space, without compromising performance.

The CM3 interrupt system is based on the Nested Vector Interrupt Controller (NVIC). Since this structure is now standardised, the NVIC is consistent for all CM3 manufacturers, thus providing users with a more consistent programming environment.
A list of specificaitions supported by the Stellaris LM3S1968 microprocessor is shown in Figure 1.

4.3. Instruction Set Architecture

The instruction set architecture (ISA) used on all ARM microprocessors and microcomputers is referred to as a load-store architecture.

4.3.1. ARM Instruction Set.



RAM	**FLASH**	**Bus Freq**	**Real-time clock**
64K	256K	50MHz	1
Timers & Resol.	**SPI, I2C**	**GPIO**	**JTAG port**
3 (32b) or 6 (16b)	2, 2	32 (5V tolerant)	1
Input capture ports (interruptible)	**Triggering**	**IO current capacity**	**ADC & Resol.**
4	rising or falling edge	4 x 8mA (for LED)	4 multiplexed, 10b (interruptible)
PWM	**UART**	**Low power modes**	**Watch dog timer**
4 x 16b	1 (16C550 type)	sleep and idle	1

FIGURE 1. Principal specifications relating the Stellaris LM3S1968 microprocessor.

4.3.2. Programmers Model. The programmers model of an ARM7 is shown in Figure 2

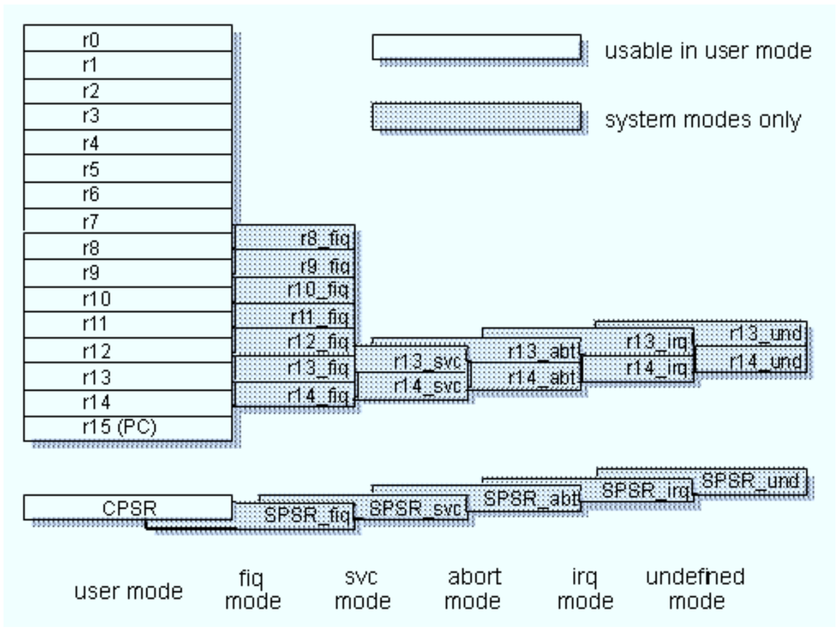


FIGURE 2. The register file for the ARM7 microprocessor, from [**10**].

Differences between the ARM7 architecture and the Cortex M3 architecture can be summarised in Table 1:

TABLE 1. Differences between ARM7 and Cortex M3 architectures, an extract from [**11**].

Features	ARM7TDMI	Cortex-M3
Architecture	ARMv4T (von Neumann)	ARMv7-M (Harvard)
ISA	Thumb / ARM	Thumb / Thumb-2
Interrupts	FIQ / IRQ	NMI + 1-240 specialised
Interrupt Latency	24-42 cycles	12 cycles
Power Consumption	0.28mW/MHz	0.19mW/MHz

4.4. Stellaris LM3S1968 Memory Structure

In theory, since the Stellaris is an ARM7-based core, the processor can support a total addressing range of 4GB. However, in practice, only a very small portion of this memory is supported. This is not a problem, since these microcontrollers have been designed for embedded systems. As a result, they do not require large memory structures to support program storage (Flash), or data (SRAM).

Two types of memory are supported by the LM3S1968 microcontroller. The first is 256K Flash block that provides for program storage. The second is the 64KB SRAM block and this is used for data storage. A data structure, such as an 1-dimensional array, defined in program such as shown below, would be stored in the latter.

A simplified memory structure used on the LM3S1968 is shown in Figure 3.

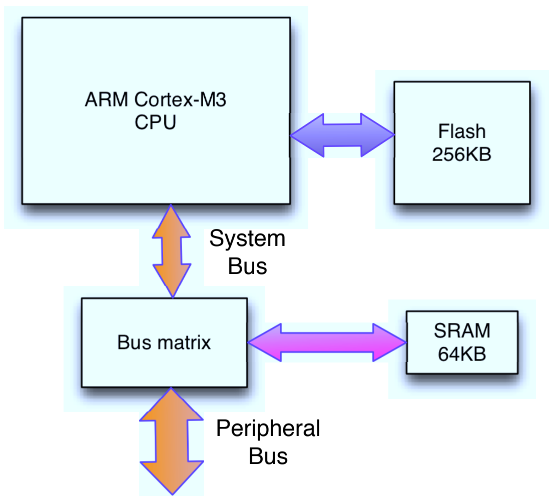


FIGURE 3. Simplified memory structure supported by the LM3S1968 microcontroller.

4.5. Stellaris LM3S1968 Peripherals

A key advantage with using single-chip microcontrollers is that many of them support a wealth of on-chip peripheral modules. Most of these modules are designed for digital signals, however some are referred to as mixed signal modules. This term is used to describe modules that combine analogue and digital signals.

Figure 4 provides a simplified view of the on-chip peripheral modules supported by the LM3S1968.

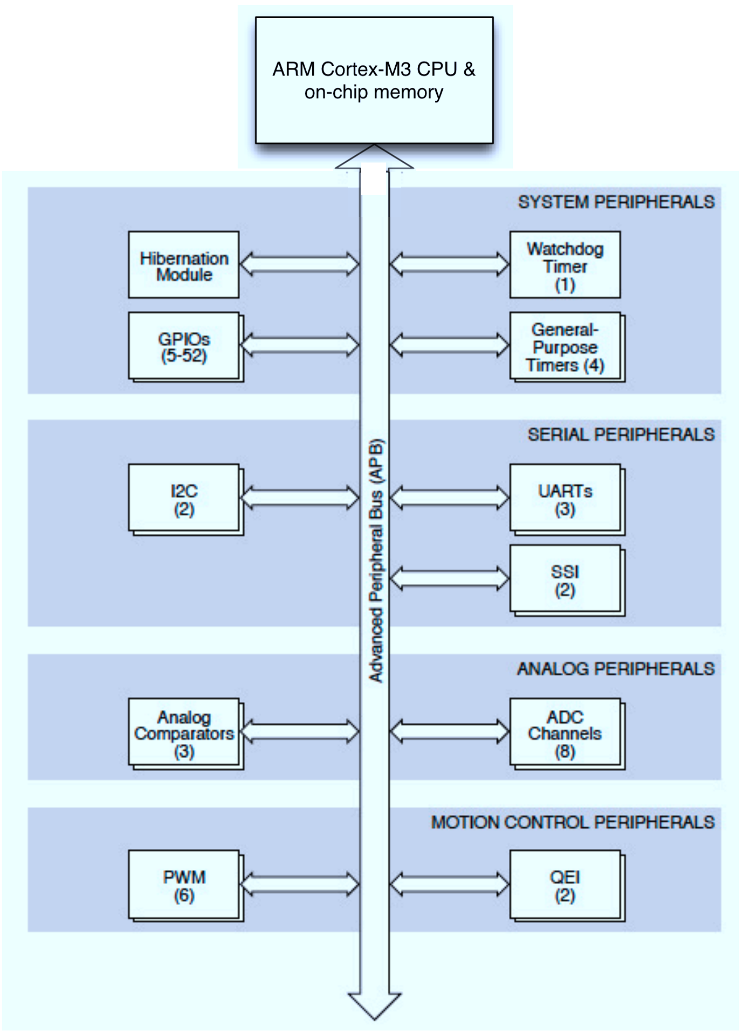


FIGURE 4. System peripherals supported by the LM3S1968 microcontroller.

4.5.1. Analogue to Digital Converter. The analogue-to-digital converter (ADC) supported on the Stellaris series of microcontrollers is a 10-bit, successive approximation converter. It has a minimum conversion time of 1 μ Sec (1 million samples per Sec) and can be configured in a multitude of different channels and modes.

A typical example of an ADC application is shown in Figure 5. Here, the ADC is taking a voltage through

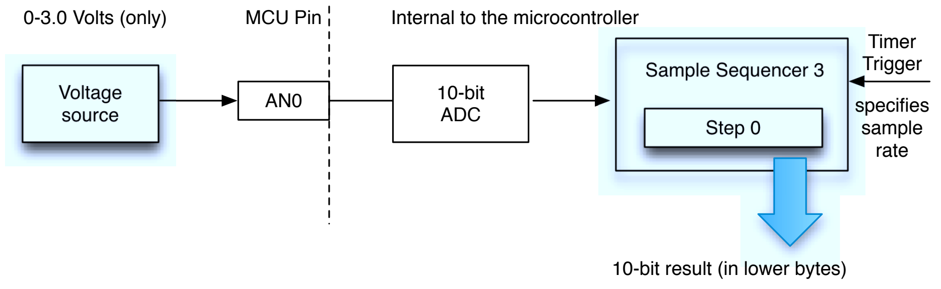


FIGURE 5. Simplified example of a single-channel ADC conversion using the Stellaris microcontroller.

4.5.2. Timers. Four, 32-bit timer modules are supported on the LM3S1968, each of which can be configured as two, 16-bit timers. However, not all the timer modes are supported in both configurations. Careful reading of the LM3S1968 datasheet is required to determine what modes are available in what configuration.

Several types of timer functions are supported on the Stellaris series of microcontrollers. These include:

(1) Input capture;
(2) output compare;
(3) programmable one-shot;
(4) ADC event trigger.

4.5.2.1. *Input Capture.* With the Stellaris series of microprocessors, there are a couple of different methods you can use to measure the frequency of a waveform. Both methods are broadly classified as input capture.

Edge Triggered Timer Capture Mode

Capture pins not available on all four timers. For example, only Timer 0 and Timer 1 provide four capture pin inputs (two per timer in 16-bit mode, labelled A and B): CCP0 (Timer 0A), CCP1 (Timer 0B), CCP2 (Timer 1A), and CCP3 (Timer 1B). For specific pin references, refer to the LM3S1968 EVK documentation.

Limitations

This mode is only available for 16-bit timer resolution. Also, the prescaler cannot be used in this mode. A prescaler is essentially a clock divider. By employing a prescaler, more "clock ticks" be counted between edges. Typically, the use of a prescaler means that the programmer can select a range of frequencies allowing wider frequency measurements. For lower frequency measurement higher prescaler divisions are typically required.

The inability to use the prescaler in this timer mode is somewhat annoying because if you are trying to measure low frequencies, e.g. ¡ 150Hz at a CPU clock speed of 6MHz, it is not possible and another mode, such as free-running timer capture is required (see below). A possible alternative is to allocate a pin for input and enable an interrupt on either a rising or falling edge; the free-running timer, with prescaler, can be taken on subsequent edges and the period simply calculated by comparing the timer values captured at each edge. The name for this mode is general-purpose timer function with 8-bit prescaler.

Edge Triggered Free-Running Timer Capture Mode

Select a GPIO pin to interrupt on an edge (rising, falling, or both). Start a timer so that it counts upwards freely and wraps automatically when it reaches a value. Note that the prescaler can be used in this mode to extend timer "ticks". Each time the pin interrupt fires, read the timer and reset it to 0. The timer value you read represents 1 period of your signal and you can derive the frequency easily from this.

This mode is easy to implement and should work well for low frequency signals. The measurement will contain jitter dependent upon the interrupt configuration and relative priority of the GPIO interrupt and any other interrupts you are running.

4.6. Software Development

4.6.1. Integrated Development Environment (IDEs).

4.6.2. Tool-Chains. Several tool-chains are supported. The most common build tool is GNU G++. In terms of debugging, either integrated or stand alone command line tools are available. In support of other ARM-based microcontrollers, a discussion on the OpenOCD tool-chain is provided.

The important requirements for the tool-chain include:

(1) A set of low-cost, low-maintanence build tools (compilers, linkers, loaders);
(2) debugging support for common interfaces, i.e. USB, serial, etc.;
(3) operating system independence.

4.6.2.1. *The Open On-Chip Debugger (OpenOCD).* The Open OCD development environment was presented as a diploma thesis written by Dominic Rath at the University of Applied Sciences in Augsburg, Germany [**12**]. The results of this work provided developers with a reliable, low-cost development environment that employ existing, open-source tool-chains.

Effectively, a USB or parallel port hardware interface and software modules were developed to simplify the use of open source tool-chains and firmware implementations. For example, JTAG is an IEEE standard used by many chip manufacturers to provide device-testing facilities. Chip manufacturers of ULSI devices, such as microprocessors and DSPs have been using the JTAG interface for device testing for over the last decade. Until relatively recently, proprietary debug tool chains have used JTAG as the preferred method of debugging within a graphical user interface (GUI) environment. This has historically resulted in high cost for development. The Open OCD tool-chain has allowed users of open source tools to employ a JTAG interface. The result has been the provision of high-performance, low-cost development tools for hobbyists, students, and professional engineers.

4.6.3. Application Programming Interface (API). To help programmers build applications, many chip manufacturers provide an application programming interface (API) for specific families of devices. Often, these comprise a set of routines, protocols, and tools for integration into your own development. Essentially, a programmer can use existing API routines and contribute very little, in terms of developing their own routines. However the key responsibility of the programmer is to:

- Understand what each API function does.
- Consider the appropriate use of an API function, as distinct to writing their own routine.
- Know when and where within their program to call specific API functions.

For large operating systems, such as MS-Windows, APIs have been around for a long time. More recently, chip manufacturers have started to include APIs to support embedded software development. This is especially the case when microcontroller architectures become more complex. For example, there are typically several steps required to initialise a peripheral device on a microcontroller. Timer and analogue-to-digital converter peripherals need several functions to initialise and ensure reliable operation. APIs provide help in this regard.

Most manufacturers of ARM microcontrollers provide an API. An example is StellarisWare that is used in this course.

CHAPTER 5

Software Design

5.1. Abstract Data Types

In software we represent real-world objects in terms of collections of data: numbers, codes, words, characters, and so on. In C the struct allows such a collection to be formalised in one contiguous block of memory. In order to pass the collection around we can either copy the block of memory in its entirety or we can pass just the address of the first byte occupied by the block.

If we want to do arithmetic on an integer (say), we know the tools are readily available in the programming language. Thus in the C language the type int has a whole lot of standard operations defined for it. But what if we want to manipulate complex numbers?

We could define a data type (structure) to store a complex number, thus

```
typedef struct
{
   double real;
   double imag;
} complex;
```

Clearly the standard operators that wed use with integer variables: +, *, /, etc. are not going to work for variables of type complex. Therefore we need to define functions to provide a set of operations on the new type. For example,

```
complex new_complex(double real, double imag);
complex complex_add(complex c1, complex c2);
complex complex_product(complex c1, complex c2);
double real_part(complex c);
complex complex_conjugate(complex c);
```

. . . and so on.

Together, the data structure and the set of operations comprise an abstract data type (ADT).

The ADT allows complex numbers to be incorporated into programs in an analogous manner to a built-in types like int and double. Importantly, just as you do not have to consider the possibility that the + operator used on two variables of type int can fail, neither need you be concerned that the $complex_add()$ function will fail, once it has been thoroughly tested. This can lead to reusable and safer code.

What if the collection of data is actually a list of books and we want to insert another book in that list? That scenario conjures declarations and function calls in C something like:

```
booklist_type abooklist;

book_type abook = new (author, title, publisher, year);

insert (abook, abooklist);
```

We might also want to find out if a book is already present, say:

```
if (present(abook, abooklist))
{
   . . .
}
```

Bibliography

[1] P. Horowitz and W. Hill, *The Art of Electronics*. Cambridge University Press, 2 ed., 1989.

[2] C. McGillem and G. Cooper, *Continuous and Discrete Signal and System Analysis*. CBS Publishing, Japan, 1984.

[3] C. H. J. Roth, *Fundamentals of Logic Design*. West Publishing Company, 4 ed., 1992.

[4] M. Mano and C. Kime, *Logic and Computer Design Fundamentals*. Prentice Hall, 2 ed., 2000.

[5] F. Cady and J. Sibigtroth, *Software and Hardware Engineering: Motorola M68HC12*. Oxford University Press, 2000.

[6] J. Copeland, *Colossus: The secrets of Bletchley Park's Codebreaking Computers*. Oxford University Press, 2006.

[7] J. Copeland, “A brief history of computing.” `http://www.alanturing.net/turing_archive/pages/Reference%20Articles/BriefHistofComp.html#Col`, 2000. [Online; accessed 29-Jan-2012].

[8] Wgsimon, “Transistor count and moore's law.” `http://en.wikipedia.org/wiki/Moore's_law`, 2011. [Online; accessed 23-Jan-2012].

[9] D. Simon, *An Embedded System Primer*. Addison-Wesley, 1999.

[10] S. Furber, *ARM system-on-chip architecture*. Addison-Wesley, 2 ed., 2000.

[11] S. Sadasivan, “An introduction to the arm cortex-m3 processor,” *ARM Holdings*, 2006.

[12] D. Rath, “Open on-chip debugger,” diploma thesis, University of Applied Sciences Augsburg, Dept. of Computer Science, 2005.