

Microcontrollers

If you need more than a handful of logic gates to accomplish the task at hand, you likely should use a microcontroller instead of discrete logic gates¹. Microcontrollers (MCU: Microcontroller Unit) are essentially Central Processing Units (CPU; similar, but usually much leaner than the CPUs in your PC or MAC; sometimes also called MPU: Microprocessor Unit) that combine internal memory (RAM: Random-access memory for volatile data storage; FLASH memory to store program code and constants etc.) and a variety of peripherals (programmable input/output (I/O) hardware), such as digital inputs or output, serial or parallel interfaces etc. (see Fig 1). Sometimes you can also find analog input and output circuitry, Ethernet and USB ports, or in very powerful MCUs even camera and display interfaces. MCUs oftentimes also contain oscillators to drive the internal CPU core, so that only very few external components are required (usually just power-supply bypass capacitors and possibly a reference quartz crystal).

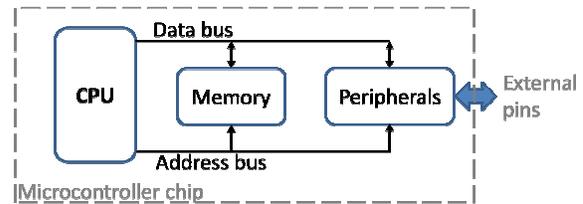


Fig. 1: Simplified diagram of a microcontroller. The CPU communicates with the outside world through the peripherals that are connected to pins on the chip. The memory contains the program and variables required to perform the desired function.

So, in essence you can regard an MCU as a complete mini-computer on a chip. For added convenience, you can typically program the internal, non-volatile FLASH memory without taking the chip out of your circuit. (This is called “in-circuit programming” and it is usually accomplished through a proprietary serial interface that connects to a few pins on the MCU chip). To make use of such a mini-computer you would likely store a program in the internal memory that instructs the CPU to read in your signals from the experiment (possibly through an internal analog-to-digital converter that could be part of the peripherals). Then your program could process this data and either store it in the internal memory, or control something in your experimental setup through the peripherals.

Memory, and address and data bus

As depicted schematically in Fig. 1, the CPU communicates with the memory and the peripherals through two buses: The address bus and the data bus². The two buses have very

¹ Besides MCUs you might also consider programmable logic devices (PLD). PLDs are great for creating fast and inexpensive networks of logic gates inside a single chip, but they are not particularly useful for complex control applications.

² The word ‘bus’ refers to a set of signaling lines that carry information between sub-units (here: between the CPU and the memory or peripherals).

different functions, but work in concert: The address bus ‘tells’ the memory (or the peripherals) *where* in the memory data should be read or written³.

Data is typically stored in consecutive blocks, which can be sub-divided into individual words (see Fig. 2). A word is typically 8 bits (= 1 Byte), 16 bit, or 32 bits wide⁴. Some very inexpensive microcontrollers sometimes use odd words, such as 14 bits for instruction storage. You can

think of a memory block as an array of data, say D_i for $i=0..N$. Each element (D_0 , D_1 , etc.) in the array is one word wide (8 bit for an 8 bit architecture, 16 bit for a 16 bit architecture etc.). The index of the array is called the *address* of the respective word. To read data from the memory, one has to apply the correct address through the address bus. The memory then presents the correct word on the data bus. Writing to the memory is similar: First assert the address at which you would like to store the data, and then write the data to the data bus.

The address and data bus’ are controlled by the CPU inside the microcontroller unit. The CPU has specific functions to transfer data between the CPU and the memory or peripherals. These instructions look somewhat like this “*MOVWF address*”. This stores the content from the working register ‘W’ inside the CPU to the memory word at the specified address. The hardware automatically applies the specified address and reads or writes the data from/to the data bus. Other instructions might add the value of a word stored in the memory to the working register. Again such instructions will automatically load the specified word from the memory to the processing unit and then perform the requested arithmetic function.

The Peripherals

The availability of specific peripheral functions in a particular microcontroller depends fully on the make/model of the specific device. You would oftentimes choose a particular chip simply because it provides the I/O functionality that you require for your task. The CPU is oftentimes less important except for time critical or ‘memory-hungry’ applications. In nearly all MCU implementations the peripherals are configured and used by writing or reading data from particular address locations inside the address-map of the microcontroller. Fig. 3 depicts the memory map of a popular microcontroller series, the PIC 16F630/16F637. In this particular chip, reading (or writing) data from the addresses between 0x20 (= ‘hexadecimal 20h’ = 32 decimal)

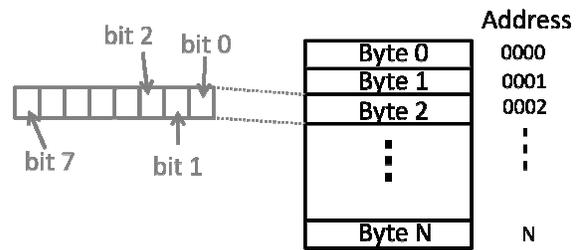


Fig. 2: 8 bit-wide memory block. The smallest addressable word is 1 Byte. Each Byte consists of 8 bits and has a unique address.

³ To be precise, there is an additional ‘control bus’ that is often intertwined with the address bus. The control bus tells the memory if the data should be written or read, or if a specific resource on the address/data bus should be shut-down to save power or to free up the data bus.

⁴ Some digital signal processing architectures are built on 256 and more bit-wide words.

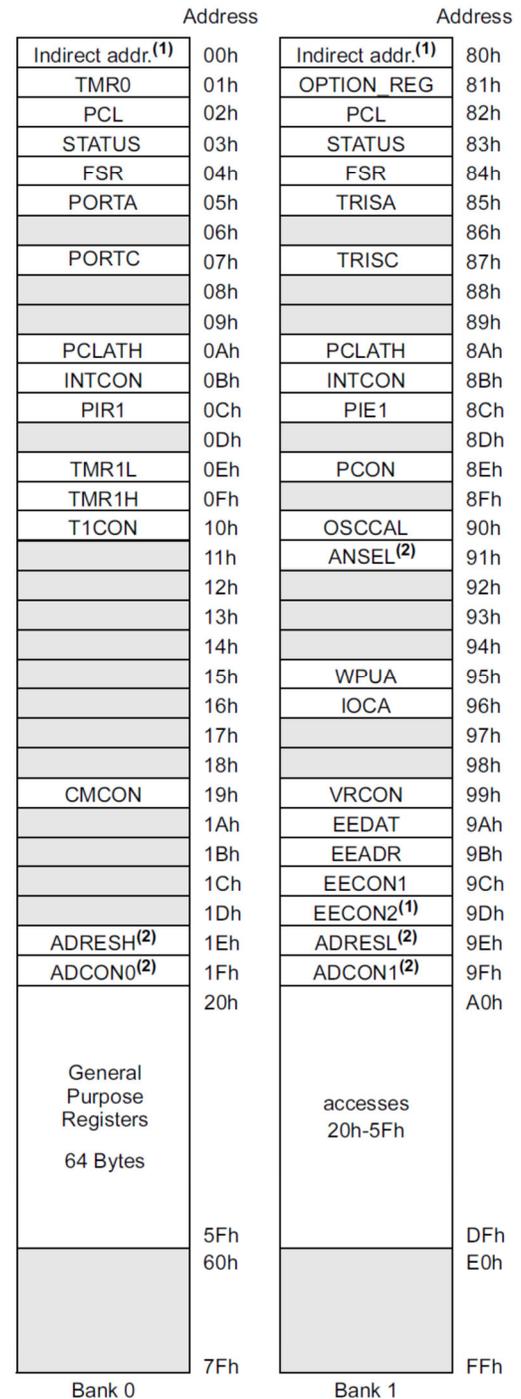
and 0x5F reads data from the internal volatile memory (see Fig. 3). However, writing or reading to any other valid address location will configure or read the status of a specific peripheral unit. For instance, reading address 0x05 will sample the signals that are currently present at the pins that are associated with PORT A and transfer these values to the working register in the CPU⁵. The value of pin A0 will be stored in bit 0 of the working register; the one of pin A1 will be stored in bit 1, etc. If the port A is configured as an output, writing to address 0x05 will change the voltages at the corresponding pins of the chip. A '1' represents logic 'high' and a '0' represents logic 'low'. Accessing other peripherals works in a similar way.

The CPU core

Now we know how the CPU loads and stores information in the memory or how it controls the peripherals. Once the data is in the CPU, it can apply a variety of logic operations (such bit-wise AND, OR, NOT, XOR etc.) to the data stored in the working register. It can typically also perform arithmetic operations, such as add, subtract and sometimes multiply and divide. Modern controllers might even implement floating point arithmetic or digital signal processing functions (DSP) that are particularly well suited for efficient implementation of high, low, or band-pass filters or to perform the fast Fourier transformation.

Other important functions are comparisons between values and conditional branching. Latter refers to the possibility to 'decide' which part of the code should be executed based on a comparison between two values. This allows you for instance to implement simple but very accurate delays (by

⁵ Port A has to be configured as inputs beforehand.



■ Unimplemented data memory locations, read as '0'.

Fig. 3: Address space of a popular microcontroller series (PIC 16F630/16F637). Writing data to addresses between 20h and 5fh stores it in the internal RAM. Writing to other address locations accesses the specified peripheral function.

decrementing a value stored in the memory and continuing doing so until that value reaches zero), or to perform a certain action if one or more conditions occurred (for instance the analog input exceeded a certain value). Certainly, it is impossible to teach good programming skills in this short lecture, but with the above you will be equipped with the most basic concepts to complete the microcontroller lab.

Appendix

Hexadecimal and binary numbers

The CPU is built on a large number of logic gates and the memory stores information in bits that are typically grouped into words (see above). So it seems tempting to express all numbers in form of individual bits, i.e. by a number system based on “base 2”. Such a counting system is called ‘binary’ number system. However, for an 8-bit word one would have to write 8 ones or zeros; or for a 16 bit word, one would have to write 16 ones and zeros etc. It does not require much explanation that this scheme is not very practical for most situations, but it is still sometimes handy when interacting with specific peripherals (for example to turn on or off an LED that’s connected to one of the digital I/O pins).

It turns out that our trusty decimal system (“base 10”, hence the name) is not very handy for programming, since it does not line-up with typical word boundaries. These boundaries are typically a multiple of four or eight bits. A not very scientific but non-the-less extremely useful choice is “base 16” called the “hexadecimal” system⁶. Base 16 can represent 4 bits in one digit and therefore nicely lines up at half-byte and byte boundaries. Fig. 4 shows an example of the decimal numbers from 0 to 18 for binary and hexadecimal representation. Please note that the right digit represents the lower four bits and the left digit represents the upper four bits of the byte. There is no such correspondence in the decimal representation.

DECIMAL	BINARY	HEXADECIMAL
0	00000000b	0x00
1	00000001b	0x01
2	00000010b	0x02
3	00000011b	0x03
4	00000100b	0x04
5	00000101b	0x05
6	00000110b	0x06
7	00000111b	0x07
8	00001000b	0x08
9	00001001b	0x09
10	00001010b	0x0A
11	00001011b	0x0B
12	00001100b	0x0C
13	00001101b	0x0D
14	00001110b	0x0E
15	00001111b	0x0F
16	00010000b	0x10
17	00010001b	0x11
18	00010010b	0x12
...

Fig. 4: A comparison of decimal, binary and hexadecimal systems. Hexadecimal numbers are sometimes also written as 00h, 01h, 02h ...

⁶ “Base 8” (called “octal” system) can also be a useful but has not gained the same traction since most words are multiples of 8 bits.