Ecodesign of electronic devices

UNIT 8: Microcontroller systems part 1

Author: Andrej Sarjaš

Chapter summary:

- Real-time systems
- Components of real-time
- Designing of programs in real-time systems

Funded by the Erasmus+ Programme of the European Union

Ecoinnovation Skills for European Designers, Project number: 562573-EPP-1-2015-1-SI-EPPKA2-SSA. Note: The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Commision. The European Commision may not be held responsible for the use which may be made of the information contained therein.

8.1. Introduction to microcontroller systems

When Intel presented the first microcontroller 4004, the era of microcontroller development has started. The modern structure of the TMS1802 microcontroller by Texas Instruments was developed to be used in calculators and until 1971 it was used in many real-time systems, such as clocks, measuring devices, cash registers, etc. With the development of new series TMS100 which has started in 1974, the microcontroller contained peripheral units that are the basic components also in modern microcontrollers (RAM, ROM, I/O). TMS100 was then known under the nickname microcomputer. The first controller that has been a real success and in broad use was Intel's 8048 that had integrated keyboard. That era has continued with Intel model 8051, as well as Motorola 68HCxx.

Today's microcontroller production reaches billions of pieces yearly with numerous different manufacturers. Today's electronic systems are hard to imagine without a microcontroller, and their use is constantly rising. Here are a few groups of devices that use microcontroller systems:

- Household appliances (microwaves, washing machines, coffee machines, etc..).
- Telecommunication (telephones, smartphones, modems, routers, etc..).
- Consumer electronics (television sets, music and video players, gaming consoles, etc..)
- Industry (management and control of devices and production processes).
- Automotive industry (engine control, safety driving system, etc..).
- Space technology.

Microcontroller systems have flourished and are a great substitution for analog systems and circuits. With the inclusion of microcontroller systems into device design we can greatly decrease device sizing, increase efficiency, reliability and make the upgrade easier. From the ecological perspective, devices with microcontroller system are significantly more economical, use of materials is lower, and recycling is easier. Modern microcontrollers contain basic peripheral units (RAM, FLASH, I/O, communication modules) and also separate units which we can use to enable lower energy consumption when we do not use the system, or it is inactive (standby mode, wake-up mode, etc.). with a good understanding of microcontroller architecture and the concept of program in the system, we can develop an efficient system or device that is environmentally friendlier. Ecodesign has an important role in microcontroller systems as there are many options how to provide reliable functioning, low consumption, and low material use.

So, what is microcontroller? What is the difference between microcontroller and processor? Why do we need a microcontroller and what tasks does it serve? For

example, let us take a closer look at a device for water heating with adjustable temperature. The operating principle of this device is following:

- Periodical temperature measurement.
- Heater management depending on the current and set temperature (ON/OFF).
- Interfaces for device management (buttons, keyboard).
- Temperature display.

First, the process begins with the design of printed circuit board PCB and processor Zilog's Z80 is used. There are also added two input-output units – PIO, serial interface SIO, timer, SRAM, FLASH, EEPROM. The final printed circuit board is shown in image 1.

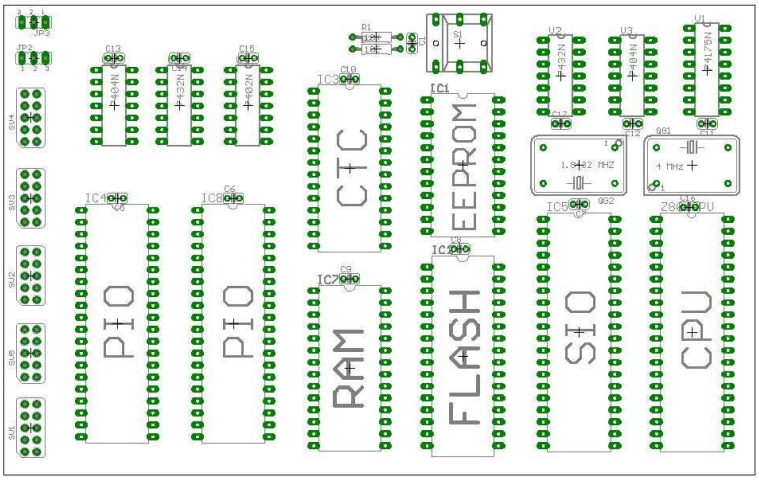

IMAGE 1: PRINTED CIRCUIT BOARD WITH PROCESSOR Z80.

The problem can be solved with microcontroller ATmega16. Printed matter design for the same system is shown in image 2.

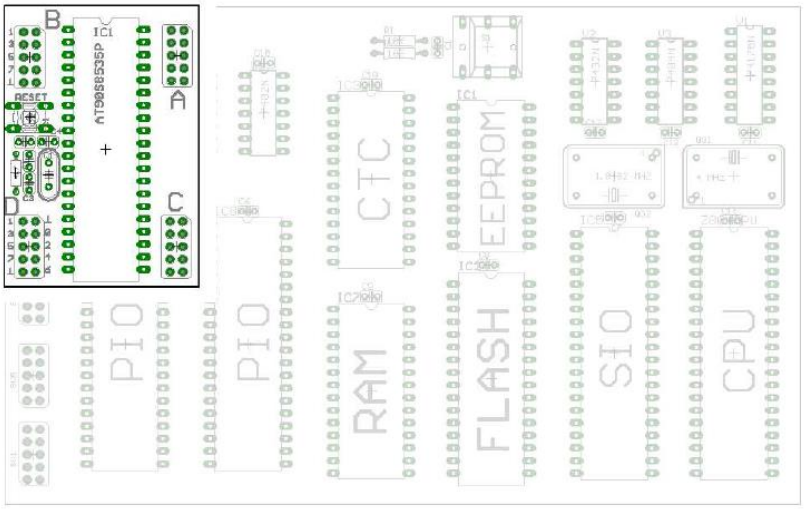

IMAGE 2: PRINTED MATTER WITH CONTROLLER ATMEGA16.

From both printed matters it is visible that the printed matter in image 2 is smaller than the one in image 1. Briefly speaking, the microcontroller is a microprocessor with peripheral units in one chip. For example, we have chosen Z80 and ATmega16 because microcontroller ATmega16 was developed based on Z80. This means it includes Z80 and has integrated peripheral units, such as serial and I/O interface, ADC, SRAM, ROM, FLASH, and EEPROM. In this case, we can also see the main usefulness and advantages of the microcontroller.

In device development, we have many options, for example when choosing manufacturer and microcontroller family. The microcontroller family is usually associated with efficiency of the arithmetic logic unit (8, 16, 32, 64, 128 bit). For the given application the chips size needs to be determined. Microcontrollers from the same family are differentiated based on chip size or by free pins. This leads to 48, 100, 144 or 176 pin housings in the same family. If we do not need many external interfaces, it is sensible to choose chips with fewer pins, because we can save on device price and size of printed matter.

8.2. Microcontroller structure

The current development of microcontroller systems is steep. Currently, the most often used are 16-32 bit systems with clock speed from 10 MHz to 300 MHz. The internal controller structure is the same. Image 3 presents the basic structure of microcontroller.

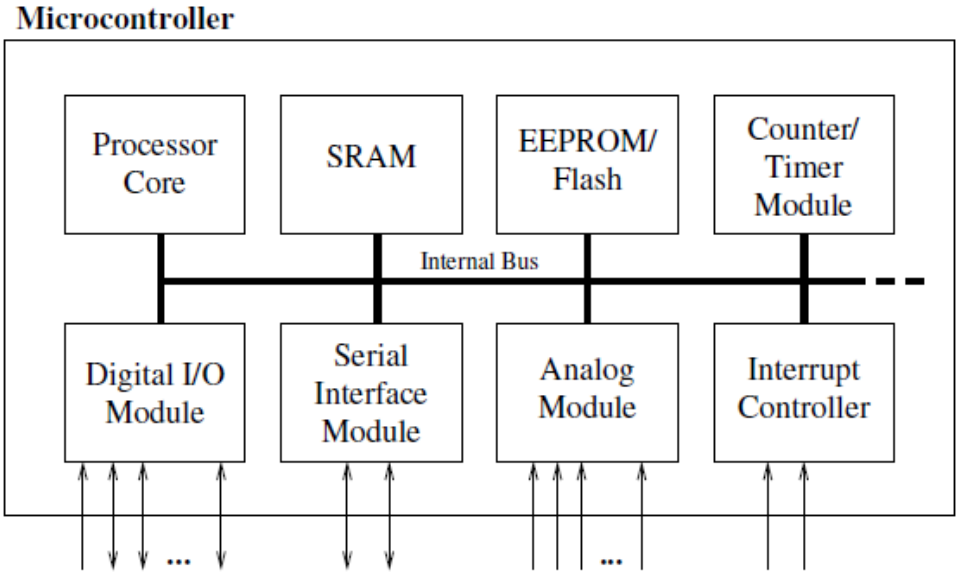

IMAGE 3: MICROCONTROLLER STRUCTURE

All peripheral units of the microcontroller as seen in image 3, are interconnected with the central “internal bus”. The typical modules that are common in most microcontrollers are:

- **Processor CPU:** Processor or CPU (central processing unit) is arithmetic logic unit that is capable of mathematical operations (addition and subtraction) and leads internal registers (memory register, program register, battery, etc.).
- **Memory:** Memory is used for saving of final or intermediate data and variables. Microcontroller memory is divided into program and data part. Amongst more powerful microcontrollers we know DMA (direct memory access) that allows the peripheral units to communicate with memory directly and not interrupt processor operations.
- **Interrupt controller:** Interruptions are key for efficient program functioning in real-time. The interruptions interrupt the basic program functions in case of events in the peripheral units. Interruptions are also useful when we want to decrease energy consumption in sleep mode.
- **Counters and timers:** Most controllers have at least two counters that are intended for counting time, counting external events and intervals. Many timers have encoding mode of counting and PWM modulation (pulse width modulation). PWM modulation is often used as DAC (digital to analog conversion). The timer/counter resolution depends on the system clock that is determined by the external or internal oscillator and length of counting registrar. The registrar, length is determined by registrar’s number of bits. Therefore, we can have 8, 16 or 32-bit counters.
- **Digital inputs/outputs:** We often see them labeled as GPIO (general purpose input and output). The number of inputs and outputs is limited by controller type and its version.
- **Analog inputs/outputs:** Amongst smaller controllers, most have analog converters ADC (analog to digital conversion). More powerful series of microcontrollers have multiple ADCs, up to four separate units. These units are connected to external pins through the multiplexer. In practice, this means that we can use several multiplexed analog inputs. The number of multiplexed physical analog inputs on controller pins is ,therefore, higher than the number of ACD units in the controller. ACD units are differentiated according to conversion resolution, and we know from 6 to 12-bit converters. ACD resolution is determined by control register of ADC unit that is set up by program code. In the newest controller series, the trend is also to integrate DAC units that are key for processing of outputs signals.

- **Communication interfaces:** For peripheral devices in printed matter that are not part of the chip are the most commonly use three types of communication standards. These are serial connections USART, I2C, and SPI. All these connections have in common that the distance between devices is between a few centimeters to the maximum of one meter.
- '**Watchdog' timer:** Watchdog timer is a special timer that triggers independently from controller functioning. This timer ensures that in case of functioning error the controller is reset.
- **Debugging unit:** In designing a program for the controller it is practical to use debugging interface. This interface can access internal controller's register while the program is running. This way, it is easier to search for errors in the program code and resolve possible problems in program implementation. Often it is called JTAG (joint test action group).

To sum it up: microcontroller is a limited processor that has integrated previously listed peripheral units. The biggest advantage of the microcontroller is the price. We can save money, decrease device size and add certain calculation power to the system. The microcontroller's disadvantage is that it can not meet the speed of analog circuits. In systems with very short reaction time, we need to substitute a certain part of the system with analog circuit. In most cases, the reaction time is not key for device functioning. The new microcontrollers already reach the reaction time in the range of several 10 microseconds.

8.2.1. Common terms

We often encounter different terms that are misleading or are used incorrectly.

- **Microprocessor:** This is normal CPU that can be found on personal computers. Communication between peripheral devices is done through the main bus. All external units (memory, USB, timers) are connected to the main bus. Microprocessor cannot operate on its own because it needs peripheral devices, such as memory, input/output units, etc. Microprocessor is not a microcontroller.
- **Microcontroller**: It includes all peripheral units and can function on its own. It is designed for systems management. Depending on the microprocessor, the microcontroller has integrated peripheral units in the chip itself.
- **Mixed signal controller:** Mixed signal controller can process analog as well as digital signals.
- **Built-in system:** Important area of microcontroller are built-in systems. The term built-in systems means that controller is built into the system or

device. Such examples are phones, robots, cars, etc. These are systems that are controlled by these controllers.

- **Real-time system:** It means that the system is controlled and managed at the defined time. The system reaction is executed in defined time intervals – in real-time. The time interval is often evaluated as system time that is determined by microcontroller or processor.
- **Digital signal processor DSP:** These are processors that are apart from an arithmetic logic unit of addition and subtraction capable of executing multiplication and division in one cycle. Such processors are intended for digital signal processing (Fourier transformation, correlation calculation, autocorrelations, and convolution). The leading manufacturers of DSP controllers are:

Texas Instruments, STMicroelectronics, Freescale, Microchip and Nxp Semiconductors, etc.

8.3. Microcontroller structure

In this chapter, we will take a look at structure and functioning of the main microcontroller elements.

8.3.1. Processor core

Processor core CPU is the main part of each microcontroller. Image 4 presents the basic processor structure.

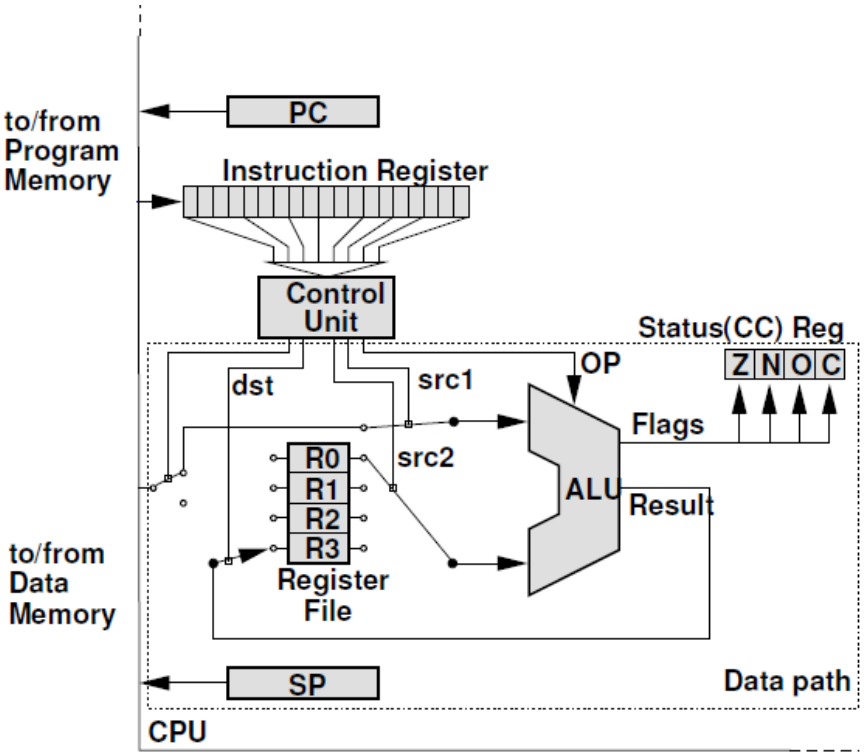

IMAGE 4: PROCESSOR STRUCTURE

The CPU core consists of a data path that executes instructions from the control unit that controls the data path. The CPU core is arithmetic logic unit ALU that is only capable of executing basic computing operations, such as addition, subtractions, and bitwise complement. Basically, ALU takes two values and returns them to the output. ALU also saves in the status registrar. The labels in status registrar mean that Z result is zero, N is negative, O operation has caused overflow, C operation delivers data ("curry").

Control unit's task is to execute instructions and determine which instruction will be executed at the given time. This unit also saves index of instructions in the program counter and the content of these instructions to the instruction register. The instruction defines which value from the register will be sent to ALU and where it will be saved. Depending on the control unit structure, we know two architectures:

- RISC: Reduced instruction set computer architecture is simpler because the execution only takes a few hour cycles. The advantages of RISC systems are that they use a certain length of micro-code for addressing and instructions. As a result, the instructions are executed quickly but are fairly small.
- CISC: Complex instruction set computer is a structure that is managed by complex microcoded instructions. Such instructions need several hour cycles for the execution. The instruction has variable length and enables many efficient instructions in comparison to the RISC structure.

The choice of the structure depends on the application for which we need the controller. If we need more complex instructions, then we need to use CISC structure. The speed at which the instructions are executed is connected to the main clock in CPU. The microcontroller core is often based on RISC structure, and the computer processors are mainly based on CISC structure.

In image 4 we can see instruction and data memory as two separate subjects. This is not always the case, but both memories can use the same memory. Depending on whether the memory is separate or common, we differentiate two types of processor architecture:

- **Von Neumann architecture:** In this architecture, the instruction and data memory is common. The bus that is used to access the memory is also common. Unfortunately, in this architecture, it can lead to a conflict between the instruction and data which can cause unwanted system delay. This disadvantage is generally known as Von Neumann bottleneck.
- **Harvard architecture:** In this architecture, the memories, as well as bus, are separate. The conflict between data and instruction is not possible which consequently improves processor efficiency. The system's disadvantage is that the architecture needs several components, such as double memory, double bus and a unit that enables simultaneous access to data and instruction memory.

The speed of task execution in CPU depends on several factors. It is mainly influenced by the CPU architecture which means whether the system is RISC or CISC. The task execution speed also depends on instruction length (8, 16, 32, 64 bit). Let us take a look at the 32-bit instruction that is executed only in one cycle on 32-bit CPU, which is faster than if the instruction would be executed on 8-bit CPU. 8-bit CPU needs four cycles for instruction execution. In the end, the task execution speed is dependent on the main clock (external or internal crystal), which means that the instructions are executed faster with 160 MHz clock than with 10 MHz clock.

8.3.2. Memory

In the previous chapter, we have presented the structure of processors that need memory for their functioning. Depending on the memory task and structure we know three types of memories:

- **Register:** Register memory is a relatively small memory built-in to CPU. The CPU needs memory to write CPU condition and the settings of peripheral units, such as ADC, DAY, I2C, etc. This memory is also named short-term memory because all values in the memory are lost when the charging source is disconnected.

- **Data memory:** In the data memory, we can save long term, and it is usually added to CPU as a peripheral unit. Data memory is significantly larger than register.
- **Instruction memory:** It is relatively large, similarly as data memory and consequently it is implemented besides CPU as a peripheral device. In Von Neuman architecture, the instruction and data memory are one peripheral device with a common bus. In microcontroller systems, the memory is implemented in the single chip memory, but it is divided into sectors.

Above mentioned memory types are the most common types of use for CPU memory. There are other memory types and registers, such as pipeline registers, cache memory, and different buffers. Depending on the microcontroller structure the memory size is integrated into the chip and is fix. The memory cannot be added or increased. Due to this, microcontrollers are only intended for execution of simpler tasks.

Until now, we have treated memory as a unit that does certain CPU tasks. From the programming viewpoint, the memory can be characterized as non-volatile and volatile memory. Volatile memory can be classified as dynamic or static. For non-volatile memory, we use abbreviations depending on structure and type of functioning: ROM, EPROM, EEPROM, FLASH. Memory classification is presented in image 5.

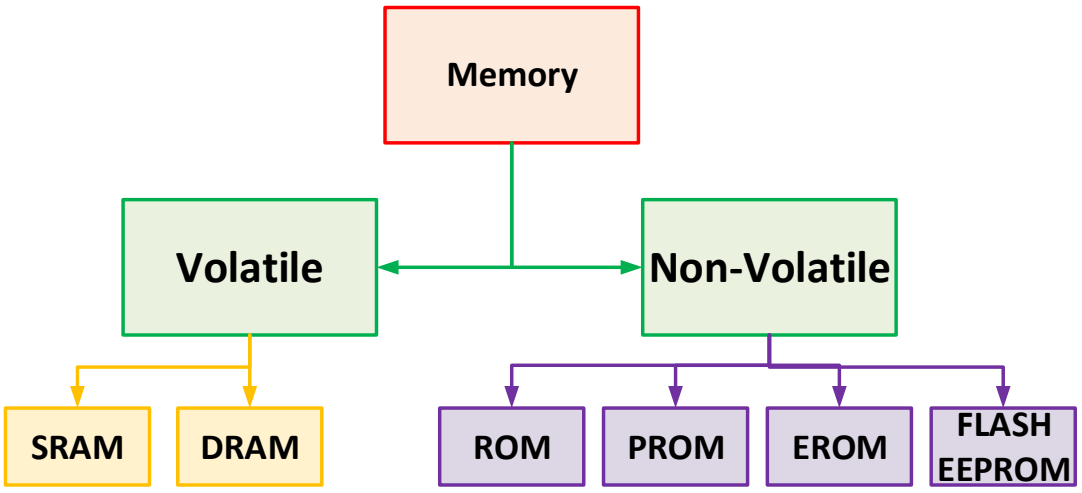

IMAGE 5: MEMORY TYPE

Volatile memory is the type of memory that saves data until it is not plugged off or the system is reset. So, why do not we only use non-volatile memory? The answer is simple. Non-volatile memory is slower than volatile memory and requires more operational time for saving, deletion, and reading. If we make a rough comparison: reading on volatile memory is measured in nanoseconds and reading on non-volatile memory is measured in milliseconds.

- **Static RAM-SRAM:** Word RAM comes from the term random access memory, which means that we can access the memory locations randomly.

Each location is accessible with the memory address. Contrary to RAM, we know shift registers where data can be written/read only sequentially. Such memory is named shift registers type FIFO (First in first out), FILO, LIFO (Last in first out), LILO, etc. SRAM memory for saving uses D-flip-flop cells. D-flip-flop is composed of at least six transistors. Each D-flip-flop cell can save one bit (0 or 1), and each cell has address. Image 6 shows 16-bit matrix memory with D-flip-flops. $A_{0,1}$ in $A_{2,3}$ mean row/column address.

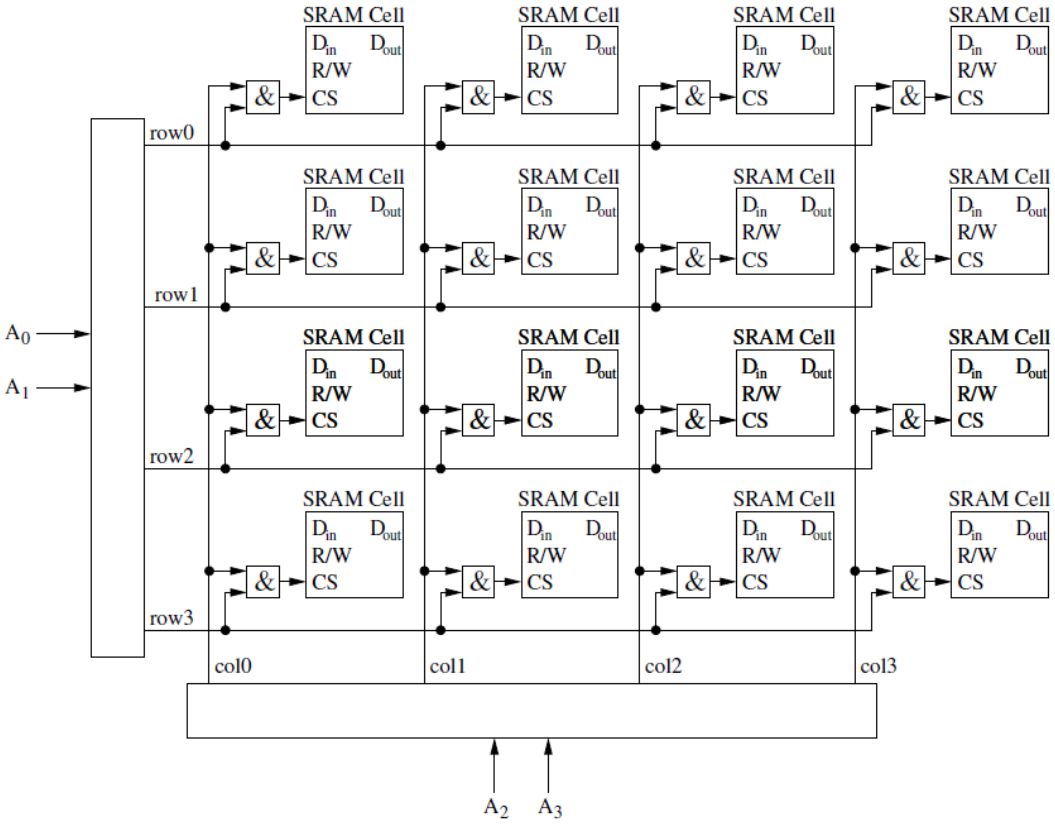

IMAGE 6: 16-BIT RAM MEMORY

- **Dynamic RAM – DRAM:** In comparison to SRAM, dynamic RAM reaches significantly higher capacities. Now we know that SRAM needs at least six transistors for saving 1 bit. For the higher capacity of SRAM, we need more cells which significantly increases the physical chip size and price. If the memory can be reduced, we can use a simple way of saving bit value. DRAM saves data in electric charge storage (capacitor). This way the number of elements for saving one bit is reduced and consequently the capacity of memory is increased at significantly smaller chip size. If we want to save the logic value in DRAM memory, we need to charge the capacitor at a certain location. It uses addresses for addressing each cell, same as SRAM and DRAM. Due to a simpler structure and lower price,

DRAM memories have some disadvantages in comparison to SRAM memory. DRAM is, for example, slower. For maintenance, it is necessary to refresh the memory because the capacitor is depleted over time.

Contrary to SRAM and DRAM, we also use non-volatile memory. It is used when we want to save data for longer period of time even when the system is not connected to power supply.

- **ROM:** ROM stands for read-only memory. Read-only means that we cannot save any data. ROM usually contains data saved by the manufacturer and are key for device functioning. In ROM is saved BIOS for personal computers or register values for microcontroller settings.
- **PROM:** ROM is used only in mass production because its production for smaller quantities or pilot research is too expensive. PROM (programmable read-only memory) is an alternative for ROM. It can be programmed only once because it contains a fuse that prevents repeated writing. PROM is also known as OTP (one-time programmable memory). It is not suitable for development when memory content and the program often change.
- **EPROM:** EPROM is an erasable programmable read-only memory. Programming of EPROM requires a complex procedure. Hence it usually cannot be deleted by the microcontroller. Value of EPROM is saved in FET (field effect transistors) that is managed by pin-gate. High voltage in pin-gate closes the transistor. When these gates are closed, they stay closed until they are not under voltage anymore. This way we can save digital values 1 or 0. Due to issues in FET, the gates can open over time. Manufacturers usually state how long data can be saved in EPROM. This period is usually ten years - each EPROM has a window that enables deletion. This is executed through the window with ultraviolet light. UV light opens FET, meaning the memory is deleted. Deletion of EPROM takes approximately 30-40 minutes.
- **EEPROM:** EEPROM is electrically erasable and programmable ROM, meaning it is an electronically programmable memory. Essentially, EEPROM works the same way as EPROM; the only difference is that for memory deletion we do not need any special external high voltages and UV light. High voltage for the awakening of FET is built-in inside chip and is known as charge pump. Generally, EEPROM has, same as EPROM, a lifecycle that is often limited to 100.000 deletion cycles.
- **FLASH:** FLASH is a limited version of EEPROM memory and works the same way as EEPROM. The difference between FLASH and EEPROM is that we cannot delete each cell separately, but only complete memory of a certain sector. The reason for the implementation of FLASH memory is the high price of EEPROM memory. FLASH also has a limited number of sign-ins, same as EEPROM.
- **NVRAM:** NVRAM (non-volatile RAM) is a combination of volatile and non-volatile memory. This memory has the same working principle as SRAM,

but it has added power supply battery. We also know a version of memory, where EEPROM and SRAM memory are joined in the same chip.

When we operate with data from memory, it is important to acknowledge the writing type. For writing data, we know two approaches:

- **Big Endian:** It is a type of writing or reading of data where higher bytes are saved at the start of the memory location. For example, if we save data '0x7799' on memory address '0x00' and '0x01'. In Big Endian the value '0x77' is saved at address '0x00' and value '0x99' is saved at address '0x01'.
- **Little Endian:** Is a type of writing where the higher bytes are saved at higher locations. If we continue with the prior example: from the value '0x7799', the part '0x77' is saved at location '0x01' and '0x99' is saved at '0x00'.

8.4. Peripheral units

In the previous chapters, we have presented the difference between microcontroller and processor. In this chapter, we will present peripheral units, that most commonly integrated into a chip of a modern microcontroller.

8.4.1. Digital inputs and outputs

Digital inputs and outputs – I/O are used for controlling and management of external devices and are the main units inside the microcontroller. Consequently, each microcontroller has at least 1 to 2 digital I/O pins that can be connected to external devices. Generally, microcontrollers have more than 32 I/O that can be used for various purposes. When connecting I/O it is necessary to be cautious of permissible voltage levels that are allowed by a certain microcontroller. The most common voltages are 5 and 3.3 volt.

Digital inputs and outputs are often grouped into ports. Each port can contain 8, 16 or 32 I/O pins. Grouping of a number of pins to individual ports depends on the manufacturer and length of control microcontroller registers. Most often all I/O pins are two-way, meaning they can be used as inputs or outputs. Most I/O pins also have additional functionalities, such as counters, external disruptions, a communication interface and analog-digital ADC or digital-analog DAC conversion. All functionalities of I/O port can be set up in the program through control registers of each port.

Digital inputs/outputs are named this way because the voltage potential in the pin entrance is converted to a logical value “1” or “0”. Voltage levels of logical values are presented in image 7. Generally, we can say that logical zero is voltage potential under 0-1V. Logical “1” is voltage potential above 2.4V-Vcc. Vcc is supply voltage of

microcontroller. Voltage potential variates depending on the manufacturers and the type of I/O pins and also whether it is used as input or output [2].

GPIO input/output pin electrical characteristics	
Output low voltage V_{OL}	< 0.40 V [1] < 0.66 V [2] < 0.40 V [3] < 0.40 V [4]
Output high voltage V_{OH}	> 2.40 V [5] > 2.64 V [6] > 2.90 V [7]
Input low voltage V_{IL}	< 0.80 V [8] < 0.54 V [9] < 1.15 V [10]
Input high voltage V_{IH}	> 2.00 V [11] > 2.31 V [12] > 2.15 V [13]
Hystereses	> 0.25 V [14] 0.66 – 2.08 V [15]
Schmitt trigger input low threshold V_{T-}	1.09 - 1.16 V [16] 0.9 [17]
Schmitt trigger input high threshold V_{T+}	2.24 - 2.74 V [18] 0.90 V [19]
Pull-up/down resistance	40 – 65 KΩ [20] 100 KΩ [21]
Pull-up/down current	< 50 uA [22] < 28 uA [23]
Pin capacitance	5 pF [24]
Bus hold resistance	5-11 KΩ [25]

IMAGE 7: VOLTAGE POTENTIALS I/O PINS BY DIFFERENT MANUFACTURERS

Microcontrollers most often use positive logic, meaning that logical “1” is potential above 2.4V and logical “0” is under 1V. In negative logic logical “1” is under 1V and logical “0” is above 2.4V. Mixed systems are often used when we want to connect microcontrollers with different supply voltages. For example, we have a system powered by 5V and 3.3V. Often microcontroller manufacturers with lower supply voltage manufacture I/O that are 5V tolerant. The characteristics of pins and electrical characteristics of pins are described in datasheets for each group and manufacturer separately.

Often we can encounter the terms “pull-up”, “pull-down” or “floating”. The term pull means that we determine default voltage potential to the I/O pins when we do not have external potential on the pin. In mode pull-up, the default potential is supply voltage, and in pull-down the default potential is GND. The term floating means that we leave the pin floating (we do not have default potential). In this case, the pin is subject to external effects, which can cause unwanted interruptions in the system if the pin is used as discrete input or output. The floating mode is often used in ADC or DAC conversion. Image 8 presents pull-up, pull-down, and floating modes.

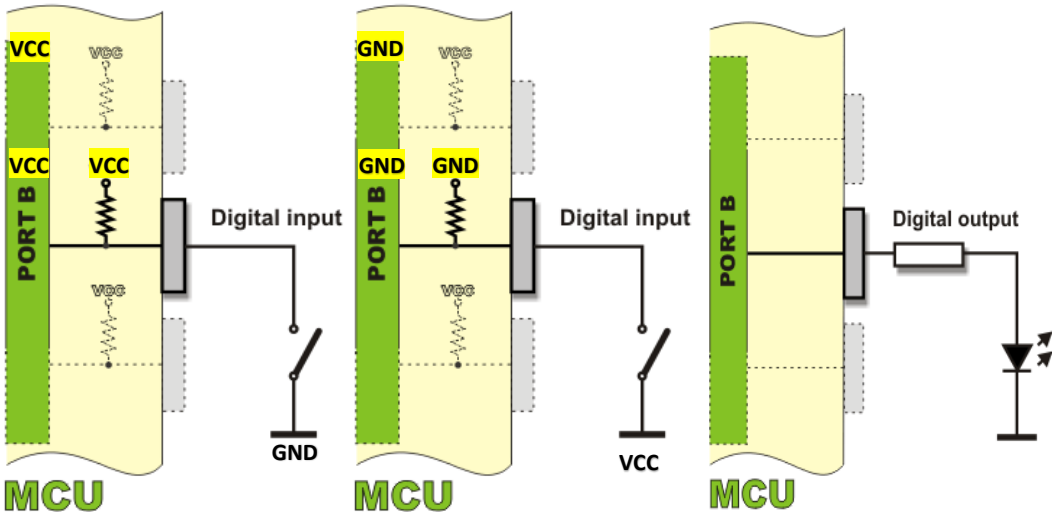

IMAGE 8: I/O PIN MODES; A) PULL-UP, B) PULL-DOWN, C) FLOATING.

8.4.2. Analog inputs and outputs

In comparison to the digital inputs and outputs, it is possible to read or process different voltage levels with analog inputs/outputs. Reading of analog values is related to ADC peripheral unit in the controller. The basic data of ADC converter are resolution, conversion speed, and conversion type. ADC resolution is given with the bit number. From the number of bits, we can determine how many voltage levels are divided into voltage range of ADC. The default voltage range of ADC is usually supply voltage of ADC. Most microcontrollers have an adjustable voltage range that can come from an external reference circuit. The microcontrollers also have adjustable ADC resolution. This means we can choose between 6, 8, 10 or 12-bit converter resolution. Let us take a look at 12 bit ADC and supply voltage 3.3V. The lowest voltage that can be measured by ADC is given by the following formula:

$$V_{RES} = \frac{V_{REF}}{2^N - 1},$$

where V_{REF} is voltage range of ADC, N is number of bits and V_{RES} is voltage resolution of the converter. For the given example (12 bit, 3.3V), the converter voltage resolution is V_{RES}= 0.805mV. Many converters enable setting of the highest and lowest voltage reference.

This means that the full converter resolution can be used between two potentials. For example, take a look at measurements of voltage potentials between 1.5V and 2.2V. with the setting V_{REF-}=1.5V and V_{REF+}=2.2V we can distribute 12-bit resolution only between 0.7V (V_{REF+} - V_{REF-}). The converter resolution between potentials 1.5-2.2.V is now $V_{RES} = \frac{V_{REF+} - V_{REF-}}{2^N - 1} = \frac{2.2 - 1.5}{2^{12} - 1} = 0.171mV$.

The next key data is conversion speed that is usually given in the number of samples per seconds. The speed of conversion in microcontrollers usually ranges between several 100ksps to several 3Msps, depending on the manufacturer and controller family. 3Msps means that ADC creates 3 million samples in one second. The samples speed is key data in digital signal processing.

Capturing of ADC signal is divided into three phases. At the beginning, we use signal holder (sample & hold). Signal holder holds the signal for as long as the conversion takes. In image 9, the signal holder is presented as switch and capacitor in which we save the voltage of the sampled signal.

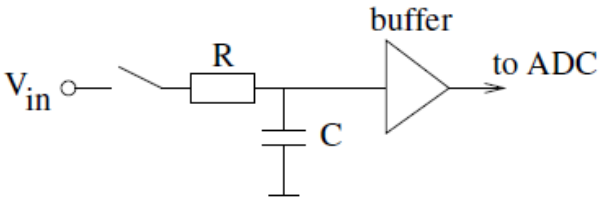

IMAGE 9: SAMPLE AND HOLD SCHEME

The second conversion phase is quantification. This is a process where we divide voltage levels to measuring ranges. One ADC quantum presents ADC resolution that has been presented with the formula above. The quantification process is rounding of real analog value; this means that we enter measurement error. This error is called quantification error of ADC converter. The last step in conversion is encoding where every quantification level has assigned a binary value. In image 10 is presented the quantification process for 3 bit DC with samples time τ_s.

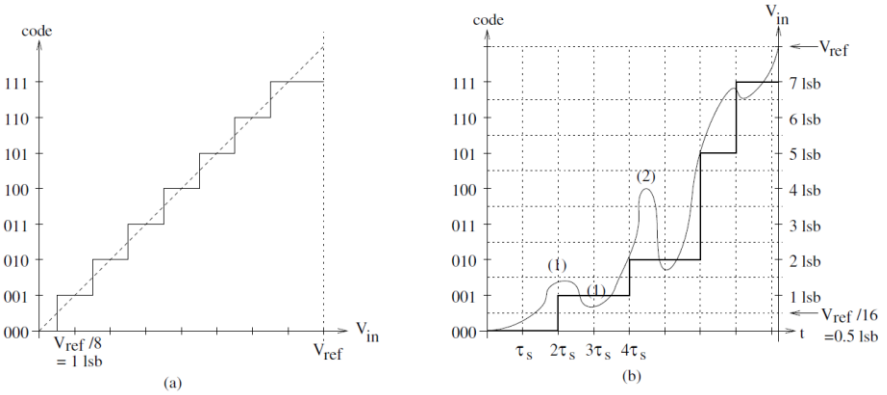

IMAGE 10: QUANTIFICATION PROCESS: A) VOLTAGE LEVELS OF ADC *0-V_{REF}*, B) SIGNAL QUANTIFICATION

In a microcontroller, ADC unit is connected to I/O controller pins. This enables us to read multiple analog signals with only one or two ADCs. Inputs in ADC are multiplexed. Multiplexed pins are also named ADC channels. Advanced microcontrollers have 3-4

separate ADCs, whereas the simplest ones only have 1 ADC. Image 11 presents multiplexing of input pins in ADC.

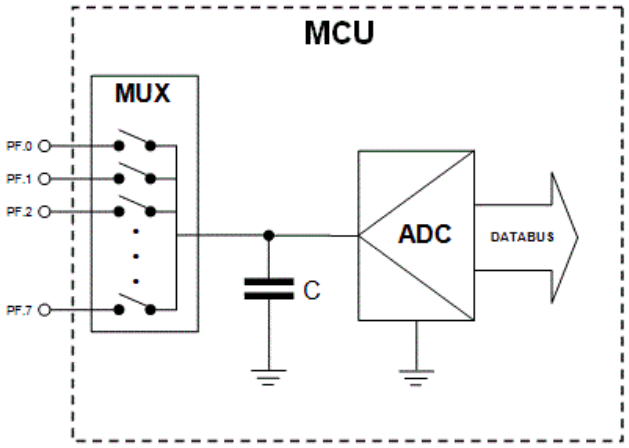

IMAGE 11: MULTIPLEXED INPUT PINS IN ADC

The function of digital-analog converter DAC is to convert the binary value to analog signal. DAC is most often used for processing of any output signals from the controller. DAC has similar characteristics than ADC. DAC resolution is defined by resolution (bit number) and samples per second speed. The voltage level of output signal is determined by voltage reference that is by default set to the controller supply voltage. DAC units are only found in the most advanced controllers, and the number of units is generally smaller than the number of ADCs. If the controller does not have DAC unit, the analog signal can be processed with pulse width modulation (PWM). PWM signal is generated with a fixed frequency and adjustable duty cycle. With duty cycle, we generate the wanted voltage potential on the output pin. Average value of one period of PWM signal presents output analog value.

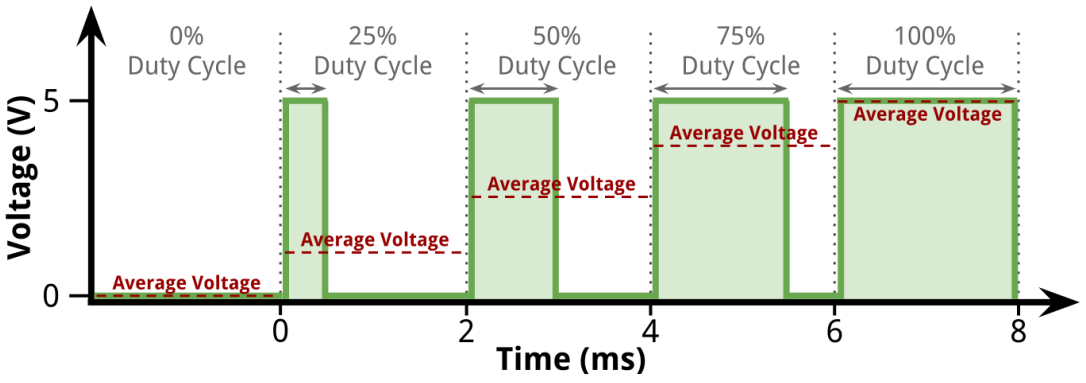

IMAGE 12: PULSE WIDTH MODULATION

8.4.3 Interrupt routine

Programs that are executed on microcontroller systems have to react to given events. The events differ by duration, whether they repeat and by complexity. The

controllers have to react to each potential change on the input pin, as well as receiving or transmission of data through communication interfaces.

Interrupt routine means that the controller interrupts execution of the main program and is dedicated to the interruption function. If we use multiple interrupt routines, priorities need to be set. The priorities determine which routine has the advantage in execution. When designing a program, it is sensible to classify interrupt routines by priorities and how they will be executed. If we use routines, that need to be executed at a specific time, it makes sense to set high priority. Below we will present some most common interrupt routines:

- **Time interruption:** Time interruption is related to the timer and is executed in accordance with the timer counter. Time interruption is executed at the exact time. Interruption can be periodical or controlled through interruption status register. Interruption control means that we enable it when needed. Time interruptions are often used for measuring time and synchronization of different periodical tasks.
- **Counter interruption:** Counter interruption is essentially the same as time interruption, but it is not strictly related to time. The interruption is executed when the counter reaches a certain value. Counting of the counter is generally triggered by external devices through discreet inputs. Counter interruption is usually used for counting pulses.
- **External interruption:** External interruption is triggered when a voltage potential is changed at the input. The external interruption only reads logical values 0 or 1. Interruption detects positive or negative front of discrete input.
- **Receive interrupt:** This interruption is used for synchronization with the sender. This means that the routine detects when data is on the bus. The interruption routine is often used in asynchronous receiving of data. Asynchronous data receiving means that transmission of data is not periodically at a certain time, but random. The interruption guarantees that we do not miss data receiving.
- **Transmit interrupt:** Interruption at data transmission enables that data package in sent in the given moment. The interruption enables that the controller does not perform other tasks other than data transmission.
- **ADC interruption:** ADC interruption is the time ADC needs for reading analog value. During the interruption, the controller waits until ADC conversion and then continues with other tasks. ADC interruption is useful when we need data from ADC for next tasks.
- **DAC interruption:** Similarly as in ADC interruption, DAC interruption has a similar role. DAC triggers interruption for a short period that is needed for conversion of binary value to analog output.
- **Interruption of sleep mode:** Interruption of sleep mode is useful when the controller does not execute the operations constantly. Then we can set the controller to go into sleep mode. In this mode, it has low energy

consumption. Interruption of sleep mode is used so the controller uses the majority of energy for monitoring of a certain event. All other peripheral devices are turned off. At the set event, the interruption wakes up the controller that works with all needed peripheral units.

Interruption routine is work of programme code that is executed at the interruption. In this routine, it is important that we delete interruption flag. The interruption flag is a bit in status register that is set to value “1” when the interruption occurs. If we want the interruption to be executed again, the flag needs to be reset to “0”. When writing the program of interruption routines, we need to be especially cautious that the code does not perform for longer than the interruption can repeat. This is especially important at interruptions that are executed periodically and quickly.

Here are a few factors that influence when it makes sense to use interruption routines:

- Tasks need to be executed randomly.
- Long intervals between two tasks.
- Change of condition is important.
- Monitoring of short pulses.
- For the full functioning of controller, only a few tasks need to be executed.
- The events are generated by both devices. There are no peaks in signal and no mechanical bouncing effect.

A few guidelines when we do not need interruption routine and when the tasks are executed in the main program:

- The operator is human.
- Event execution is not time-critical.
- Input impulses are long.
- The signal is covered by noise.
- The main program is not long.

When designing a program for the microcontroller, it is important to analyze program execution comprehensively. It needs to be studied what is executed in the interruption routine and what will be executed in the main program. When multiple interruptions are used, we need to be cautious if the interruption execution requires additional use of RAM memory. When multiple interruptions are triggered, the controller can run out of memory, and we can lose certain information from the interruption routines. Debugging of similar problems is very difficult and time-consuming.

8.4.3. Counters and timers

Counters and timers are key peripheral units of each microcontroller that are often connected to other units. Microcontrollers have integrated multiple different counters that differ by number of bits (8, 16, 32-bit). Timers are used for different tasks, such as delay, measurement of time, measurement of frequency. The most basic use of timers is use of the counter only. The timers can generate different events, interruption or process modulated PWM signal.

Each timer is a counter. The counter of the timer can count up or down. Count mode can be configurated inside the control register. The counter condition can be read in counter's data register. Each counter has own control, status and data register. Its size is determined by its bit length. Bit length is the largest possible number of counter 2^N-1, where N is bit number. For example, the 8-bit counter can count between 0 and 255, where 16-bit counter can count from 0 to 65535. If the counter is connected to the clock, we get a timer. The timer triggers counting depending on the input tack (clock). Timer counting can be triggered by the positive or negative edge of clock tact. Clock tact that is used in timer can be external or internal. In case of internal tact, we use CPU clock although it comes from an external crystal. This mode of clock use is synchronous mode. External clock tact means that we provide a new clock specifically for the timer. External timer clock is not related to CPU tact that is known as an asynchronous mode. The external clock is often used for calendar (time, days, month, etc.). For calendar, clock tact is precisely 32.768kHz. Timer for the calendar is known as RTC counter (RTC is real time clock). The timer can be configurated by several counting modes and time frame for counting. The timer's time increment is the time of one counter's increment. The time increment and time length can be set in timer registers.

Main timer parameters:

- **Clock:** Timer clock is determined by the speed of peripheral bus for the given timer. The speed of peripheral bus is determined by controller's system settings. Some simpler controllers do not have separate peripheral buses for individual units, and the clock is same for all peripheral units. We also need to determine clock source which is asynchronous or synchronous mode.
- **Scaling factor (prescaler):** Scaling factor determines timer clock split. With this factor, we determine timer increment and duration of counting when we count until the end of counters data register. Timer increment is time resolution that determines how precisely we measure time. Scaling factor value is saved in PSC register.
- **Period:** Timer period determines counting time frame. This means that the longest period equals the length of the counter data register. Counter data length is determined by a number of bits in the timer. Timer period is saved in ARR (auto reload register). Counter period determines the number of counted units inside the length of full ARR register. This means that the

counter does not only count until the maximum length of ARR but also until the period value that is set in the program.

Counters have two more functioning modes. These are known as input capture and output capture. Input capture is used when we want to count certain events on controller input pins. Each change (event) of the logic values “0” or “1” on a specific input pin is transferred from the counter to input capture register. This register can be read in program code. Input capture mode is often used for reading of different encoders. Output comparison mode is similar to input capture, but here we monitor output pins. Changes on the output pins increase or lower counter condition. Output comparison mode can also trigger interruption when the counter has reached the set increment number.

Apart from widely useful counters, microcontrollers also contain special counters that are not intended for above-mentioned uses. These are watchdog timer (WT) and are intended for monitoring of controller functioning and program code flow. WT timer has own separate clock that is not linked to CPU. Its functioning is relatively simple. WT is set to a time frame that can be in the range of milliseconds to seconds. At the start, the counter counts down to zero. Each task, function, interruption routine resets WT counter, meaning the timer has the start value after each reset. If we are not able to reset the counter before it reaches zero, the hardware controller reset will be triggered (“kick the dog”). The controller is restarted. With WT timer we prevent the controller would stay stuck in cycles or become unresponsive.

8.4.5. Communication interfaces

Communication interfaces enable communication between the microcontroller and other external devices, such as other controllers, PCs, sensors, etc. In this unit, we will present only interfaces that are directly implemented in controller chip. Communication interfaces differ by different characteristics, such as parallel or serial interface, synchronous or asynchronous communication, point-to-point or network mode, half-duplex or full-duplex mode, wire or wireless mode. Below will be presented only wire interfaces.

Serial interface transmits one bit in one clock tact. This means it is dependent on the clock speed. Depending on the clock frequency, we determine the number of transmitted bits per time unit (bit/s), which is named communication speed. Due to this, serial communication interface needs less physical data lines. Parallel interface transmitted each bit through the physically separated bus. If we transmit 8-bit data, we need at least 8 physical connection between two interfaces. The parallel connection is faster than serial. Its disadvantage is that it requires too many physical buses and more controller pins. Parallel communication is often used for very short distances, LCD displays, reading of fast ADCs and DACs, different sensors, etc. The serial connection is often used for shorter, as well as longer distances due to its simplicity.

In many cases, the communication between two systems is two-sided. This means that both devices can receive or transmit data. The next question is, when can a device transmit or receive data. If we use full-duplex mode, then both devices can receive or transmit data at the same time. Physically this means they have two separate lines for transmitting (Tx) and receiving (Rx). In serial connection this means that we have two wires; one for transmitting, one for receiving. In the 8-bit parallel mode, this means 8 wires for receiving and 8 additional wires for transmitting. If we use half-duplex mode, then both devices receive and transmit through the same line, meaning concurrency is not allowed. This would mean collision and loss of data. This mode requires less physical connections but increases the complexity of communication protocol.

In communication interfaces, we often come across the term master-slave. This mode is often used in serial mode. Device master decides when the slave can access the bus and what he can do on the bus (receive or transmit data). If devices are in the network, then it is permissible that only one device is the master and others are subordinates.

The most common communication interfaces for microcontrollers are USART, SPI, I2C, etc. Some advanced controllers have also integrated complex interfaces, such as CAN, USB and TCP/IP.

- ***USART*:** It is a serial connection (universal synchronous asynchronous receiver transmitter) that uses only two lined for communication. One line is for transmitting Tx and other for receiving Rx. The difference between USART and UART is that USART needs a separate line for clock. UART has a preset clock with which are familiar all other devices in the communication path. In USART the receiving device uses sender's clock. Image 13 presents UART communication interface.

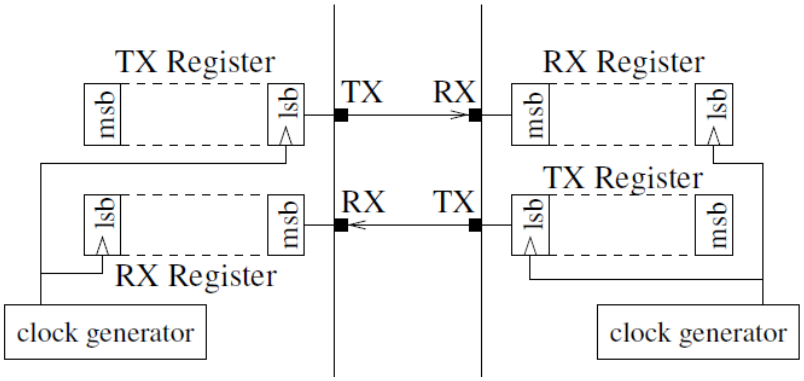

IMAGE 13: UART COMMUNICATION INTERFACE

Adjustable parameters of USART serial interface are:

- ***Number of data bits:*** Number of data bits determines how many bits will be captured in the sent data. This can range from 5 to 9-bit data. The most commonly are used 8-bit data.

- ***Parity bit:*** The user can decide if parity will be used or not. Parity can be an even "1" or odd "0" bit. It is used as simple control over communication regularity. One example for the use of even and odd parity in 7-bit data is in image 14.

7 bits of data	(count of 1-bits)	8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

IMAGE 14: EVEN AND ODD PARITY

- ***Stop bit:*** Stop bit announces when the transfer is finished. It can be 1 or 2 bits long.
- ***Transfer speed (baud rate):*** Transfer speed is determined by clock tact. The higher the clock tact, the faster the transfer is. Most often are used the following connection speeds: 9.6kb/s, 38kp/s, 115.2kb/s, 1Mb/s,10Mp/s.

The serial connection is known under standard RS232, RS422, RS485, etc. RS232 is a connection between two points where the voltage level of the positive front is determined between 3-15 V. Both devices connected to RS232 have to be on the same potential and have common GND. For the use of RS232 and UART on the microcontroller are used interfaces that increase or decrease voltage level for standard RS232. Often is also used interface MX232.

RS422 is the same communication as RS232, but it uses differential voltage levels. These enable transfer on long distances. Devices do not need common GND.

RS485 is similar standard to RS422, but it allows 32 devices on the same bus, while RS232 and RS422 only allow one pair. In both standards RS422 and RS485 the connection speed decreases with connection length.

- **SPI:** SPI (serial peripheral interface) is a serial connection that is intended for communication between near-by devices. Allowed distances are from only several centimeters to the maximum of one meter. Communication is provided by full-duplex mode between the master and slave. The interface also supports 3,2,1 wire mode.

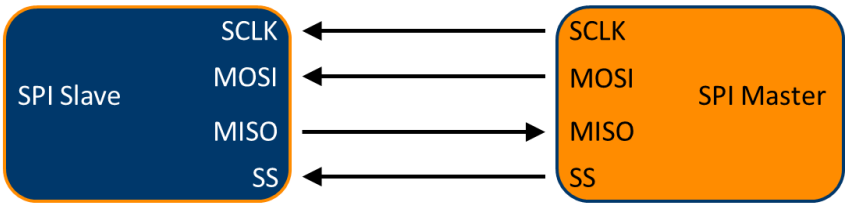

IMAGE 15: SPI COMMUNICATION

SPI communication in full extent uses four lines:

MOSI: (master out slave in): Connection for transmitting data from master to slave.
MISO: (master in slave out): Connection for receiving data from slave to master.
SCK: (system clock): Generated clock-communication speed.
SS or CS: (slave select or chip select): Addressing device by the master.

SPI communication enables connection of several devices to the same SPI bus (MOSI, MISO, SCK). Each device only has separate SS/CS pin, as seen in image 16. SPI communication sends packets of 8 bit. Communication speed is related to CPU clock of the master and clock speed of the slave.

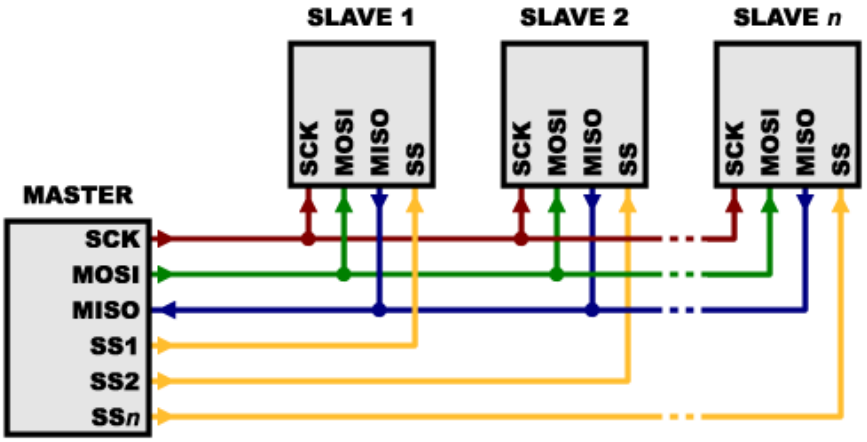

IMAGE 16: SEVERAL DEVICES ON SPI BUS

- **IIC (I^2C):** Is serial communication that enables half-duplex mode. It also uses principle master-slave. Same as SPI, IIC is also intended for communication between near-by devices. The speed of IIC bus is determined by three factors. The slow mode is 100 kbit/s, fast mode 400 kbit/s and the highest speed 3.4 Mbit/s. The communication interface only requires two lines. One line is data SDA and the second is generated clock SCL by the master. The bus enables 10 or 7-bit data transfer. The device on IIC bus has 7-bit address. At the beginning of communication, the master addresses device with the address, then begins receiving and transmitting of data. The interface enables several devices on one bus, as seen in image 17.

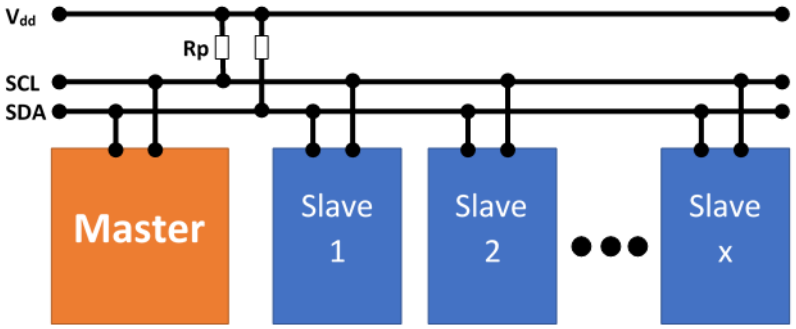

IMAGE 17: IIC COMMUNICATION INTERFACE

SPI communication is faster than I2C communication and physically needs more pins than the I2C bus. SPI communication is often used for management of graphical displays for SD cards and for capturing images from the optical chip. I2C is often used interface where there are no strictly defined time limitations.

Microcontrollers often contain several communication interfaces. For example, take a look at microcontroller STM32F407 where we have 6 USART interfaces, 3 SPI interfaces, and 3 IIC interfaces.

8.3. Guidelines for low energy consumption of microcontrollers and ecological aspects

Each controller can be configurated in order to have the lowest possible consumption. The first step to ecological design requires choosing a controller depending on its characteristics and chip size. The chip size is directly related to used chip materials, such as housing plastic, metal pins or silicon. Used material at the end of lifecycle also affects recycling. When choosing a controller, we will not choose 144 pin chip and only use a few input and output pins. For this, it makes sense to choose a smaller version from the same family. Another example are highly efficient controllers. Those use more energy than simple controllers. For each device or application, it is important to evaluate which controller we will use. On the market are several controllers that are intended for low consumption and long autonomy.

When choosing a controller, it is sensible to consider the following points:

- **Controller clock:** Setting of controller clock is closely related to energy consumption and with system autonomy. Frequency is proportionally related to energy. In each controller, it is possible to set CPU in certain limits that are recommended by the manufacturer. If controller tasks do not have high time demand, then the controller tact can be reduced until it meets the time criteria for the tasks. Most controllers have a program-managed clock, meaning that during execution of complex tasks we

increase the clock tact and provide fast execution. When the controller is working on simpler and non-time-demanding tasks, the clock is lowered to the extent where the tasks are still working normally.

- **Supply voltage:** Most controllers have adjustable supply voltage in a certain area. The supply voltage is also proportionately related to energy consumption. In most cases, the supply voltage is also related to controller efficiency and CPU clock tact. We often meet the limitation that at lower voltage we cannot set higher clock tact. When choosing supply voltage, it is needed to compromise between efficiency and energy consumption.
- **Switching off unused modules:** Many times when designing program code we do not use many peripheral devices. The controller enables that peripheral devices are switched off, meaning they do not consume additional energy when not in use. Use of peripheral devices can also be managed through program code.
- **Optimal code design:** When designing code, it is important to consider the execution. The shorter the code, less instruction the controller has to execute, meaning the controller can be in standby mode for longer. When designing code we also need to be aware of the type of selected variables and the number of these. In larger programs, it can happen that the variable is declared and then it is rarely used or even not used at all.

- **Standby mode:** When the controller enables different standby modes it is sensible to use them, especially if we have long pauses between task executions.

References:

[1] Günther Gridling, Bettina Weiss, 'Introduction to Microcontrollers', Vienn University of Technology press, 2007.

[2] http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications