EE443 - Embedded Systems

Lecture 4
Microcontroller Hardware

Contents:

This lecture summarizes the common components that can be found in a microcontroller. It is better to combine as many as possible components in a single chip. The advantages of integration are:

Cost: IC packages, PCB area, and PCB assembly are all major cost items that can be reduced with integration of components in a single device.

Easy development: You don't need to worry about interfacing external components and the related noise coupling or interference problems. Accessing an ADC is as easy as writing and reading a few special function registers when the ADC is available as an on-chip component of the microcontroller. On the other hand, if you end up using an external ADC component, then you need to find solutions to several design requirements such as, ADC power supply, interface protocol, reserving I/O pins on the microcontroller, and routing I/O connections on the PCB.

Special function registers or **SFR**s in short, provide easy access to all on-chip microcontroller peripherals and interface units. SFRs can be read or written using simple machine instructions much like accessing the on-chip memory. In every microcontroller data sheet you will find the specifications and functional descriptions of several SFRs that are dedicated to peripheral functions.

4.1 Processor

The **instruction set architecture** (**ISA**) of the processor determines the main processing capabilities and efficiency of a microcontroller. Usually a microcontroller is labeled with a ***core*** name that identifies the processor ISA (e.g. **. . . . .** controller with *ARM core* or **. . . . .** controller with *8051 core*). Note that the ISA determines the main programming features and requirements of a processor. The hardware implementation may vary from an IC manufacturer to another. Following is a list of the common processor architectures used in microcontrollers.

AVR (Atmel): A large variety of microcontrollers manufactured by Atmel. One of these microprocessors will be used in the laboratory work of this course.

PICXXX (Microchip): PIC is a family of Harvard architecture microcontrollers manufactured by Microchip Technology. PIC processors are widely used for industrial applications and hobby purposes. Optimized processor efficiency, serial programming capability, free development tools, and availability of a large library of application notes made the PIC processors popular.

8051 (Intel): A Harvard architecture 8-bit microcontroller series developed by Intel in 1980 for use in embedded systems. Today, a large variety of faster and enhanced 8051-core microcontrollers are produced by more than 20 independent manufacturers.

68XX, 680X0 (Motorola): The 68xx is an 8-bit microprocessor family introduced by Motorola starting in 1974. The MC6801 and MC6805 were commonly used as microcontrollers in automotive industry. The Motorola 6809 introduced in 1977 had some 16-bit features. The Motorola 680x0 is a family of 32-bit processors that were popular in personal computers and workstations until the mid-1990s. Descendants of the 680x0 family processors are still widely used in embedded applications.

ARM (ARM7, ARM9, ARM11 - ARM Holdings): ARM architecture is the most widely used 32-bit ISA in terms of numbers produced. ARM Holdings itself does not manufacture any processors but they license ARM ISA to independent manufacturers. ARM processors dominate the mobile and embedded electronics market (90% of all embedded 32-bit RISC processors in 2009), because they are suitable for low power applications as relatively low cost, simple, and small microprocessors.

x86 (386, 486, Pentium, . . . - Intel): The term x86 refers to a family of ISAs based on the Intel 8086 CPU introduced in 1978. More specifically, x86 implies compatibility with the instruction set of 32-bit 80386. x86 processors are not widely used in embedded systems.

Choosing the appropriate processor for an application requires consideration of several factors.

ISA: The simplest ISA that can answer the requirements of an application is usually the best choice. A 32-bit ARM processor used for a simple industrial controller is an overkill that increases manufacturing costs and wastes development efforts. An 8-bit ISA with powerful built-in peripherals can handle many demanding tasks. Microcontrollers including dedicated I/O buffers, interface modules, and direct memory access controllers, significantly reduce the processor load.

Processor speed: It is often misleading to compare processors solely based on the clock frequency. There are different hardware implementations of the common ISAs

provided by several IC manufacturers. An instruction that takes 10 clock cycles to execute in processor-A may take only 3 clock cycles in processor-B. In this case, processor-B will run faster even if it uses half the clock frequency of processor-A. The number of instructions executed in unit time is a better measure of the processing speed for comparing processors that have the same ISA. A commonly used speed specification unit is **MIPS** (Million Instructions Per Second).

It is often misleading to compare performance of processors with different ISAs based on their MIPS specification. Incrementing a memory variable may take three instructions on a typical processor:

```
LOAD   A, Count;  Load counter variable to accumulator
ADD    A, #1;     Add 1 immediate to accumulator
STORE  A, Count;  Store the result back to memory
```

PIC16F87x processors can complete this operation in a single instruction:

```
INCF   Count, f;  Read Count, increment, and store back to memory
```

On the other hand, some other types of operations require more instructions on PIC16F87x. In some cases, benchmarks can provide more reliable results for evaluation of processor performance. A **benchmark** is the measure of time it takes to complete a certain procedure that requires execution of thousands or millions of instructions. The benchmark procedure can be the calculation of a PID controller output or calculation of a discrete Fourier transform on a sample sequence. Benchmarks can be obtained on previously developed systems or new designs running on simulation tools. Efficiency of the compilers and optimization of the test programs can be the other factors that can affect the benchmark results.

Availability of programming tools: Programming support available for the processor can be another important factor in selection of the microcontroller. Availability of free assemblers or compilers may significantly reduce the development cost of embedded systems.

4.2 Memory Organization

There are two common computer architectures that outline the memory organization in a computer:

1. The **Harvard architecture** is a computer architecture where instruction and data memories have distinct address spaces. The instruction address 0x0000 and the data address 0x0000 target different storage units. In the original Harvard architecture, instruction and data memories have physically separate address and data buses that allow simultaneous processor access to the two isolated memories. The instruction and data memories share the same address and data buses in the modified Harvard architectures that are more common today. The processor identifies the target address space using a single control signal. A computer with Harvard architecture that has 16-bit address bus and 8-bit data bus can have 64 KByte of instruction memory and another 64 KByte to store data.

2. The **von Neumann architecture** has a single address space. There is only one sequence of addresses where instructions and data occupy different address ranges. Addressing of instructions and data memories are the same from the processor's point of view. A von Neumann machine with 16-bit address bus and 8-bit data bus can have a total of 64 KByte memory that is shared between code and data.

Memory Requirement of an Embedded System:

A stand-alone computer requires two types of memory. First, a non-volatile memory is necessary to store the program instructions and other permanent information required for execution. The non-volatile memory does not lose the stored information when power is turned off. The second type of memory is the volatile memory to store temporary or dynamically changing information during program execution.

Non-volatile memory requirement:

- Program storage
- Read-only tables, such as calibration data, system control parameters, user settings that should be saved when power is turned off.
- At least 20% safety margin for the additional memory that may be required for future corrections and upgrades.

Volatile memory requirement:

- Variables
- I/O buffers
- Other data structures, such as stacks and queues.

Common Non-volatile Memory Devices:

ROM (Read-Only Memory): The term, ROM, strictly meant "*read-only memory*" as it was used some decades ago. The data stored in a ROM could not be modified by any means. But today, ROM may refer to all types of non-volatile memory included within an embedded computer system. The data stored in a ROM remain unchanged or "*read-only*" during the normal program execution, but the memory contents can be modified slowly or with difficulty when it is necessary. The following is a list of other related memory devices:

PROM (Programmable ROM): Memory contents can be written only once by fusing (blowing up) or anti-fusing the connections in the device applying high voltages.

EPROM (Erasable Programmable ROM): Memory contents can be written using a programming device. An EPROM can be erased by exposing it to strong ultraviolet light through the transparent window placed on top of the chip.

EEPROM (Electrically Erasable Programmable ROM): A version of EPROM where memory contents can be erased electrically instead of using ultraviolet light.

FLASH Memory: FLASH memory is a kind of EEPROM that is commonly used as non-volatile memory on the microcontrollers available in the market today. The EEPROMs other than FLASH are erasable in small blocks (i.e. byte by byte). Large blocks of data can be erased and written simultaneously in FLASH memories. This gives a significant speed advantage when writing large amounts of data.

Common Volatile Memory Devices:

RAM (Random Access Memory): *Random access* means ability to read or write at any memory address as opposed to restricted addressing methods, such as sequential access. The term, RAM, may refer to all types of volatile memory in a computer system. SRAM and DRAM are the two common types of RAM.

SRAM (Static RAM): Memory contents remain intact as long as the power is on. SRAM is the most common type of volatile memory included in a single-chip microcontroller. The cache memories on more powerful CPUs are also SRAM devices.

DRAM (Dynamic RAM): Unlike SRAM, a DRAM device needs to be periodically refreshed to keep the memory contents intact. DRAMs store each bit of data in a capacitor, and the voltage on the capacitor needs to be refreshed because the capacitor charge fades due to leakages. One bit of DRAM storage requires less silicon area compared to one bit of SRAM. DRAM is the choice for large memory requirements since the cost per unit of memory is less for DRAMs

SDRAM (Synchronous DRAM): Data transfers to/from SDRAM are synchronized with a clock signal. Once a memory address is sent to the SDRAM, a sequence of memory locations can be accessed starting at the specified address.

DDR SDRAM (Double Data Rate SDRAM): SDRAM that transfers data at a rate twice as high the synchronization clock frequency by receiving or sending data at both of the rising and falling edges of the clock.

4.3 Reset Circuit

Active electronic devices require all supply voltages to be within specified limits to operate properly. The device behavior is unpredictable if a supply voltage is outside the specified range. A power-on-reset (POR) signal is required to make sure that all components of a system are in a known idle state until the supply voltages reach the specified operation limits. Supply monitors (also called "power monitor" or "voltage monitor") are the dedicated circuits that generate a master reset signal according to the supply conditions. These are the functions of the master reset signal:

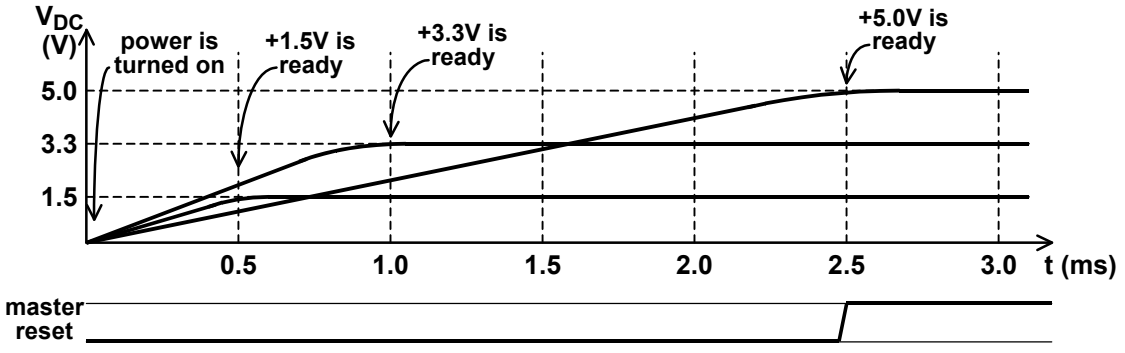
1. Suspend all operations while the voltage regulators are ramping up until the supply voltages reach the specified operation limits (power-on-reset).

2. Initialize state of sequential logic circuits.

3. Stop and re-initialize circuit operations if a fault condition occurs at the supplies.

Consider an embedded system that requires three voltage regulators to supply DC power for the following purposes:

- +1.5 V supply for processor core and memory
- +3.3 V supply for I/O with peripherals
- +5.0 V supply for sensors and drivers



All voltage regulators start to ramp-up after power is turned on as shown in the timing diagram given below. It takes time in the order of milliseconds for each supply voltage to reach its target level. In practice, voltage regulators should be designed to ramp-up slow enough to keep the current demand at a reasonable level while charging output and decoupling capacitors after power is turned on. The +1.5V supply for the processor core is ready 0.5 ms after power up. The master reset signal should be kept low until all supply voltages reach their target levels. Otherwise, the processor may try to communicate with the peripherals, sensors, and drivers that do not have the proper supply voltages. A typical processor can execute several thousand instructions in 2.0 ms, reading wrong sensor signals and generating unpredictable driver outputs.

The master reset is an active-low signal (0 means reset) because a valid logic signal at active-high level cannot be obtained while the voltage regulators are ramping up. Common supply monitors have open-collector reset outputs so that they can easily be merged with open-collector or open-drain outputs of other fault detection circuits. The following is a list of the common specifications for a supply monitor:

1. Valid voltage ranges for the supply inputs.

2. Response time and glitch filtering capabilities.

3. Duration of reset pulse after a reset condition.

4. Minimum supply voltage required to generate a valid signal level for the active-low reset output.

4.4 Clock Generator

Clock generator provides the clock signal for all sequential circuit operations. Most systems use a single clock source as a reference and necessary clock frequencies are derived from this reference clock using simple frequency dividers or phase lock loops (PLL). The precision of clock timing depends on the requirements of a particular application. Generation of a pulse-width modulation (PWM) output, for example, can tolerate large variations in the clock frequency. The effective value of a PWM output is determined by the duty cycle which is independent of the clock driving the PWM circuitry as long as the clock frequency remains relatively constant in each PWM cycle. The timing precision of a clock signal used for speed measurements on the other hand, should match the accuracy of the measurements. There are strict frequency matching requirements for I/O channels that rely on independent clock signals as frequency references.

A microcontroller may require an external clock generator or it may have on-chip circuitry that supports a frequency resonator. These are the typical specifications for a clock generator:

1. Maximum deviation from target frequency specified in +/- ppm (particle per million). If you purchase a 10 MHz resonator specified with +/-50 ppm accuracy, then you will obtain a resonance frequency between 9,999,500 and 10,000,500 Hz.

2. Temperature dependency of oscillation frequency specified in ppm/ºC. The maximum frequency deviation guaranties the oscillation frequency range at a constant temperature (usually 20 or 25ºC). The temperature dependency adds on to the frequency uncertainty according to the temperature variations.

3. Jitter, or the timing error between clock cycles: Having 10,000,000 clock cycles in one second does not necessarily mean that each clock cycle will be exactly 100 ns. Jitter is the measure of timing error from one clock interval to another, that can be an important factor when matching reference frequency of communication channels.

4.5 I/O Ports, Channels, Buffers

I/O ports on a microcontroller provide general-purpose access capabilities to external devices. These ports can be simple I/O pins that can be read or written by the core processor or they can be combined with more complex interface options. Some examples of these interface options are:

- Independent access to individual I/O pins as well as of 8 or 16 pin groups.
- Built in buffering functions for continuous data transfers.
- Direct memory access to or from the port pins for high-speed communication with external devices without using processor time.

One should be careful about determining the number of available I/O pins on a microcontroller. Almost all microcontrollers assign multiple functions to many of the I/O pins. An I/O pin required as an external interrupt input or as an analog interface pin will not be available for general-purpose I/O functions.

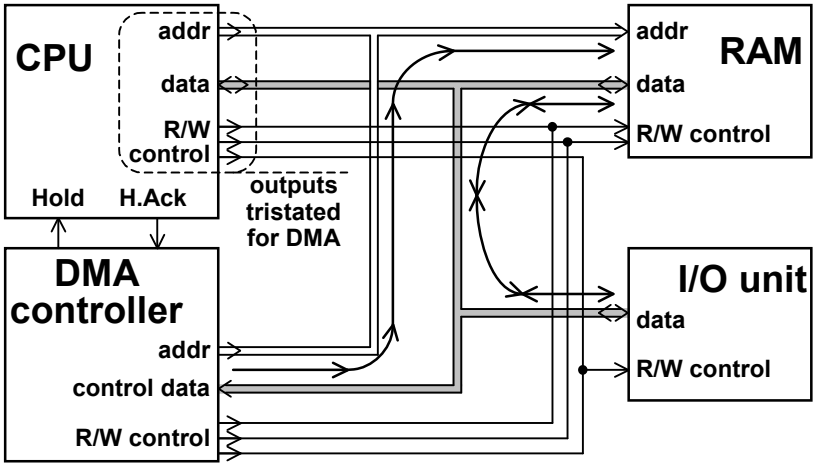
4.5.1 Direct Memory Access

Data transfers between the RAM and I/O units or other storage devices may take long processing times depending on the amount of data to be transferred. For example, if the CPU handles a sequence of data transfers from an I/O unit to the memory, then it performs several operations to transfer each word of data:

1. Read data from the I/O unit into the CPU register.
2. Write data from the CPU register to the RAM.
3. Increment the pointer for the target address in the RAM.
4. Decrement the byte/word counter.
5. Check the byte/word count and go back to step-1 if there is more data to be transferred.

A typical CPU executes a number of instructions to perform these operations. Every data transfer takes 10s of memory R/W cycles considering the operation codes and the address information loaded for each instruction. As an alternative, a **direct memory access** (DMA) controller can be used to speed up the data transfers. The following figure shows the organization of data and address busses with a DMA controller.

Address and Data Flow in Direct Memory Access



DMA controller takes over management of address, data, and R/W control lines after CPU tristates its outputs. DMA controller generates the required address sequence for memory and arranges timing of the control signals for memory and I/O unit. Data words are transferred directly between the memory and the I/O unit. It takes just one R/W memory cycle to complete the transfer of a data word. Direct memory access is an order of magnitude faster compared to the CPU operations that take10s of memory R/W cycles for each data transfer.

CPU programs the DMA controller writing starting memory address, number of data transfers, and other necessary information into the controller registers. DMA controller sends a "Hold" signal to the CPU asking for release of memory control when it is ready to start the transfer sequence. CPU replies with a "Hold Acknowledgement" after it tristates its outputs. DMA controller returns control of memory back to the CPU after it completes the data transfers.

The traditional DMA organization described above has been utilized widely in personal computers. DMA controllers handle data transfers between CPU cache, main memory, hard disk, and other storage devices. Microcontrollers designed for specialized I/O functions may have some parts of their data memory dedicated to DMA. These dedicated memory sections serve as buffers for high speed I/O interfaces. If uninterrupted CPU operation is crucial in a computer system then the CPU access to memory cannot be put to hold during DMA transfers. These systems use dual-port memories as I/O buffers. One of the memory ports is used for CPU access while the other port allows data transfers to/from the memory through a DMA controller or similar I/O hardware.

4.6 Serial I/O

A serial interface will be slower compared to a parallel bus operating at the same clock frequency. Whenever you don't need the combined speed of all the data bits in parallel, a serial interface is preferable because of these advantages:

- **It requires less number of pins.** A serial interface uses 2, 3, or 4 connections, so it will require 2-4 pins on transmitter and receiver ends. An 8-bit parallel interface on the other hand, would require 10 or more connections between the transmitter and receiver ends. Less number of pins means smaller and cheaper IC packages.

- **PCB design will be easier** for a serial interface, and it will save PCB area as well. Simply, you will route 3 connections instead of 10.
- **Serial interface will require less power.** A serial interface will eventually use more-less the same amount of average power to transmit the same number of bits, but the instantaneous power requirement will be different. If you decide to use a parallel interface, then you had to be prepared for the worst case where all data lines switch from 0 to 1 at the same instant.
- **Save on noise.** Just as simultaneously switching 8 data lines requires more power, it will also generate more noise on your PCB.

Two different types of serial I/O interfaces can be found on microcontrollers or embedded processors. The first type of serial interfaces are for "on-board" or "in-box" communication between the components of a single unit. The commonly used examples of on-board serial communication interfaces are **SPI**, **uWIRE**, and **I^2C**. The second type of serial I/O interfaces are used for box-to-box communication between the independent design units through cables. The typical examples of box-to-box serial interfaces are **USART** (Universal Synchronous Asynchronous Receiver Transmitter, RS-232, RS-422, etc.), **USB**, and **Firewire** (IEEE 1394). Another advantage of serial interfaces in box-to-box communication is the simplicity of the connectors and cables. Long serial transmission cables can be built at a lower cost compared to the cost of parallel data cables that support the similar data transfer rates.

4.7 Timers

Timers are counters that can be programmed to perform a variety of functions. Following are the typical operation modes and possible applications of timers:

1. Programmed operation: A timer can be used as an alarm clock to generate predetermined time delays. The microprocessor sets the count limit or initializes the counter and enables the count operation. The timer generates an interrupt when the count limit is reached indicating the end of the programmed delay period. This mode of operation utilizes the internal clock and it does not require an external connection.

2. Gated or triggered operation: The count operation is controlled by an external signal. There may be several options to start and to stop the counter. In gated mode, the counter is enabled while the external signal is active. A multi-purpose timer allows independent selection of events that start and stop the counter. These events can be a rising or falling edge of the external trigger signal or an internal start/stop command issued by the microprocessor itself. The timer can be programmed to generate an interrupt when the counter stops. The common applications of gated or triggered operation involve any kind of time measurements, such as, measuring revolution time of a motor to detect its rotation speed, or quantification of time-encoded signals.

3. Clocked operation: The counter clock is supplied by an external signal while the count operation is enabled by the microprocessor or another timer. The typical applications include quantization of frequency-encoded signals, or position information generated by linear or rotational encoders.

The specifications for a timer are directly related to the requirements of the application:

1. Maximum clock frequency determines the timing resolution. The internal clock frequencies available for timer operations depend on the microprocessor clock.

2. Number of counter bits determines the count range or the maximum time period that can be measured.

3. Functionality: Having a programmable timer does not necessarily mean that it will support all operation modes and gating or triggering options. You need to read the timer description to find out whether it is useful for your application or not. You may as well need additional features such as buffering of timer count results for motor speed measurements. A timer with buffered outputs can store the count result at the end of every motor revolution and re-start the counting process immediately.

Watchdog timers are special-purpose timers dedicated to ensure the proper execution of microcontroller functions. The processor is required to restart the watchdog timer before the preset timer period expires and it repeats this operation as long as the watchdog function is enabled. The program written for the processor includes the necessary statements to restart the watchdog timer periodically. If the processor fails to restart the watchdog timer, then this indicates a major functional failure due to corrupt program memory or some other reason. In this case, the watchdog timer resets the processor, forcing initialization of all microcontroller functions.

4.8 Pulse Width Modulators

Pulse width modulation (PWM) is an efficient way of synthesizing analog waveforms from digital signals. PWM signal is a square wave where the duty cycle (ratio of active pulse width to the full waveform period) in each PWM period corresponds to the relative amplitude of the analog signal. The analog waveform is obtained after taking the time average of the PWM output utilizing a low-pass filter.

Switched mode motor drivers are the most common applications for the PWM controllers. The average current flowing through a motor winding is controlled by the duty cycle of the PWM waveform switching on and off between the supply voltage and ground level. The inductance of the motor winding serves as a low-pass filter taking time average of the voltage across the winding. The power dissipation on the switched mode drivers is minimal since either the voltage drop or the current flow over the driving transistors is practically zero during the active and passive PWM cycles.

Pulse width modulators are basically special-purpose timers operating in programmed mode, that can repeat the counting sequence automatically. A common timer needs to be reprogrammed by the processor at the end of every counting sequence. A pulse width modulator is programmed only once, initializing the counter settings corresponding to the duty cycle and PWM period. The same active and passive PWM cycles are repeated automatically without any further commands from the processor. The processor updates the PWM duty cycle when a new amplitude setting is required.

Pulse width modulator specifications are similar to the timer specifications we have seen before. The maximum clock frequency and the number of counter bits

determine the amplitude resolution of the synthesized analog waveform. Number of clock cycles in each PWM period gives the number of possible amplitude settings.

4.9 ADC and DAC

Analog to digital converters (ADC) and digital to analog converters (DAC) are the bridges between digital and analog worlds. The analog input or output range is determined by a reference voltage, $\mathbf{V_{ref}}$. Typically for an N-bit converter with unsigned digital I/O and unipolar analog range ($\mathbf{0V .. +V_{ref}}$), one step at the analog end, $\mathbf{\Delta V_{LSB}}$, is given by:

$$\Delta V_{LSB} = \frac{V_{ref}}{2^N}$$

Similarly for a bipolar analog range ($\mathbf{-V_{ref} .. +V_{ref}}$), one step at the analog end is:

$$\Delta V_{LSB} = \frac{2\,V_{ref}}{2^N}$$

There is a large variety of ADC and DAC components that can have several options:

- Internal or external reference voltage supply.
- Unipolar ($\mathbf{0V .. V_{ref}}$) or bipolar ($\mathbf{-V_{ref} .. +V_{ref}}$) analog signal range.
- Unsigned ($\mathbf{0 .. 2^N-1}$) or signed ($\mathbf{-2^{(N-1)} .. 2^{(N-1)}-1}$) digital I/O range.
- Parallel or serial digital interface.
- Number of multiplexed analog inputs for an ADC or number of DAC circuits in a DAC component (number of analog channels).
- Sequential or simultaneous access to analog channels.
- Initialization options for a DAC after power-on.

Following are the critical specifications for an ADC or DAC that determine the performance of the converter. For simplicity, specifications related to analog accuracy are referred to the digital end, and expressed in terms of **LSB**s (least significant bit). One **LSB** error at the digital end corresponds to $\mathbf{\Delta V_{LSB}}$ inaccuracy in the analog domain that depends on $\mathbf{V_{ref}}$ used in the application.

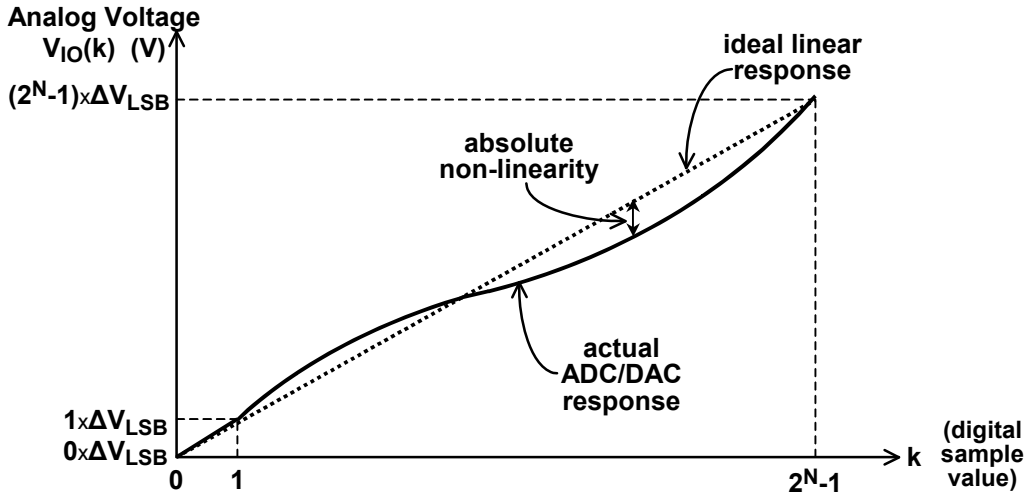
1. Resolution: Given by the number of bits at the digital interface. Note that, useable resolution of a converter is determined by the linearity of its response and the accuracy of analog reference.

2. Sampling rate: Is the maximum conversion speed expressed in **S/s** or **Sa/s** (samples per second). Converters supporting sampling rates from a few KS/s to several hundred MS/s are commercially available.

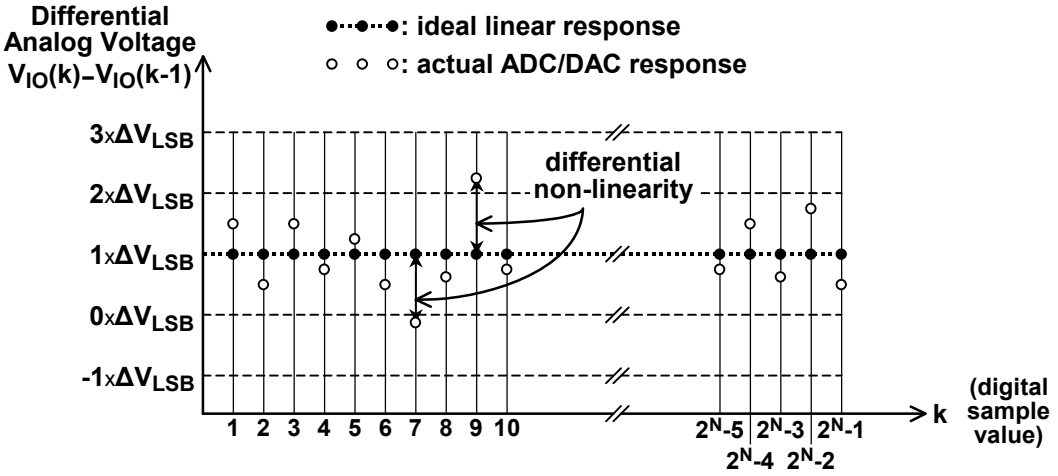
3. Settling time or slew rate for DAC: Sampling rate alone does not specify the actual analog bandwidth of a DAC output (you cannot get full range, 500 KHz square wave output from a 1 MS/s DAC). Settling time usually gives the time required for a DAC output to settle within $\mathbf{+/-1\ \Delta V_{LSB}}$ of the target value after changing digital input from 1/4 to 3/4 of the full range. The output slew rate expressed in V/s or V/μs is the measure of how fast the output changes under the specified load conditions.

4. Absolute non-linearity: The variation of analog voltage levels (vertical axis) corresponding to the digital sample values (horizontal axis) is plotted in the following figure as an example of the ADC or DAC response. Ideally, we expect a linear

variation of analog voltage given by $\mathbf{V_{IO}(k) = k x \Delta V_{LSB}}$ as a function of the digital sample values ($\mathbf{k = 0 .. 2^N-1}$) for an N-bit ADC or DAC. **Absolute non-linearity** is the maximum deviation of the actual converter response from the ideal linear conversion function.



5. Differential non-linearity: Difference between the analog voltage levels corresponding to successive pairs of digital sample values is plotted in the following figure. Ideally, all difference voltages should be exactly one $\mathbf{\Delta V_{LSB}}$ on the analog side or one **LSB** on the digital side throughout the conversion range. Differential non-linearity is the maximum deviations from this ideal behavior above and below the ideal response line.



Differential non-linearity is a measure of how smoothly the analog signal varies as we step through the entire range of digital sample values. Normally, differential non-linearity of an ADC or DAC should be within +/-1 LSB. If it is greater than **+1 LSB**, then the analog voltage increases by more than $\mathbf{2x\Delta V_{LSB}}$ between two successive digital values. If it is less than **-1 LSB**, then the analog voltage decreases while it is expected to increase.

4.10 Conversion Applications

We refer to the specifications included in device datasheets to determine whether each component is suitable for our purposes in an application. We need to find answers to several questions related to timing and amplitude specifications.

Timing specifications:

How fast: Bandwidth, slew rate, settling time for analog signals, clock frequency, response time, setup and hold time requirements for digital signals.

How accurate: Time resolution, jitter, variation in response time.

Amplitude specifications:

How big: Gain, I/O signal range for analog signals, I/O level compatibility (TTL, CMOS, LVDS, etc.) for digital signals.

How accurate: Amplitude resolution, linearity, noise figure, distortion for analog signals.

Driving capability: Receiver load, termination requirements, wiring or routing distance.

An embedded system interacting with the physical world converts continuous input variables of time and amplitude into numbers that can be processed in the microcontroller, and it converts the resulting numbers into continuous variables at the output. The conversion process introduces errors in addition to the errors that already exists in the continuous domain. The following table summarizes the error sources involved in the conversion process.

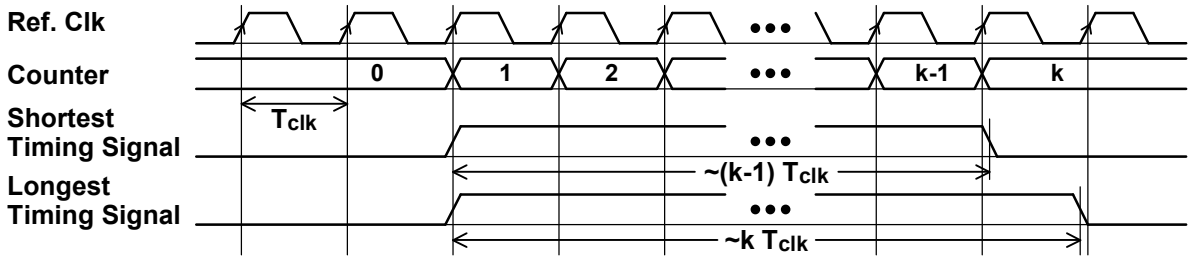
Type of error	Amplitude conversions using ADCs, DACs	Time conversion using timers, PWM modules
Error at the input source or output target	sensor/driver non-linearity, noise, distortion	timing accuracy, jitter, bandwidth restrictions
Quantization error	**+/- 1/2 ΔV_{LSB}**	**+/- 1/2 T_{clk}** or **+/- T_{clk}**
Reference error	**V_{ref}** accuracy	**F_{clk}** accuracy
Processing error	ADC/DAC non-linearity, settling time limitations	response time variation (if any)

Quantization error: The accuracy in any quantization process is limited by the minimum step size in the continuous domain. For example, the minimum step size is given by **ΔV_{LSB}** for an ADC. The numbers at the ADC output correspond to the input voltage levels quantized as **0**x**ΔV_{LSB}**, **1**x**ΔV_{LSB}**, **2**x**ΔV_{LSB}**, **. . .** If the actual sampled voltage at the ADC input is **1.5**x**ΔV_{LSB}**, then the ADC produces either **1** or **2** at the output. In any case, we have **+0.5**x**ΔV_{LSB}** or **-0.5**x**ΔV_{LSB}** error after the quantization.

The effect of quantization error in time-related measurements and timing control can be different depending on the synchronization of the timing signal with the

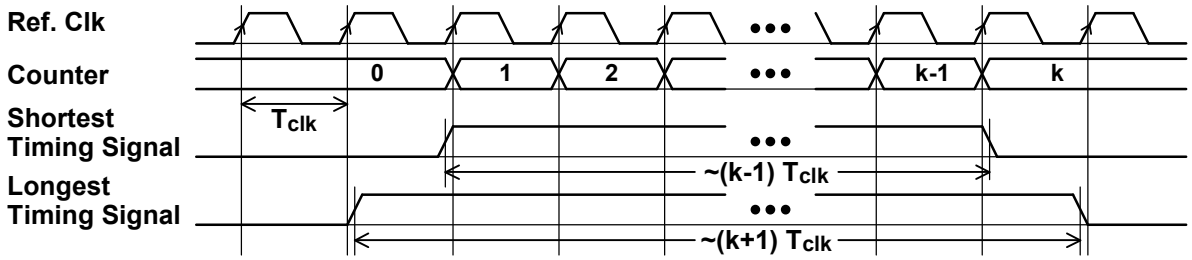
reference clock. The following two figures show the shortest and longest possible time periods that may produce the same quantized time value of $\mathbf{k}x\mathbf{T_{clk}}$ for the two cases. In the first case, beginning of the timing signal is synchronized with the clock. The counter stops at **k**, after the falling edge of the timing signal. The actual timing period may vary between $\mathbf{(k-1)}x\mathbf{T_{clk}}$ and $\mathbf{k}x\mathbf{T_{clk}}$, resulting in a random quantization error of $\mathbf{+/-1/2\ T_{clk}}$.

Case-1: Beginning of timing signal is synchronized with reference clock:



In the second case, there is no synchronization between the timing signal and the reference clock. There are timing uncertainties at the beginning and at the end of the measured period. The quantized value of the time period is $\mathbf{k}x\mathbf{T_{clk}}$ while the actual time in continuous domain may vary between $\mathbf{(k-1)}x\mathbf{T_{clk}}$ and $\mathbf{(k+1)}x\mathbf{T_{clk}}$, resulting in a quantization error of $\mathbf{+/-1\ T_{clk}}$.

Case-2: No synchronization between timing signal and reference clock:



The second case described above represents a typical situation where a processor receives a timing signal from an asynchronous source. The quantization error of time measurements increases by a factor of two when there is no common time reference for **t = 0**. Voltage measurements on the other hand, have a reference level for **V = 0** given by the signal ground.

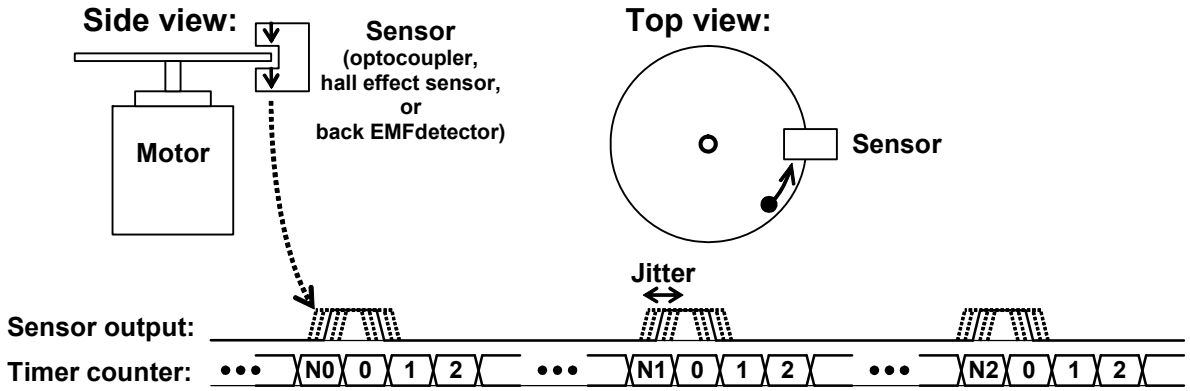
Reference error: Accuracy of a measurement cannot be any better than the accuracy of the reference source used in the measurement. All time conversions rely on the accuracy of the reference clock, and all amplitude conversions rely on the accuracy of the reference voltage or current. Generally, the maximum error due to reference variation occurs at the longest time duration or the highest signal amplitude involved in the conversions. If an ADC has a **1V +/-1%** reference voltage source, then this will cause **+/-1mV** error when the input signal is **100mV**, and **+/-5mV** error when the input signal is **500mV**.

Processing error: This is a generic term that refers to the other errors introduced during the conversion process. If a time measurement relies on the immediate response of the processor to some external events, then the accuracy of the produced result depends on the delay time between the actual event and the processor's response. The deterministic or repeatable part of the error can be corrected by using compensation techniques. For example, the error due to the

average response time of a processor can be compensated by applying an offset to the measured values. On the other hand, random variations in the response time are not predictable, and they cannot be corrected. Similarly, if the non-linear response of a sensor or an ADC can be characterized thoroughly, then the non-linearity errors can be corrected to some extend by using look-up tables and interpolation methods. Such correction methods demand processor time which may not be available in simple microcontrollers.

4.10.1 Design Example

A time measurement setup is required to detect speed of a motor with **0.1 %** accuracy. Motor speed will be determined by measuring the revolution time of the rotor.



The sensor generates a short pulse every time it detects the marker attached to the rotor. A timer is necessary to count the number of clock cycles between the sensor pulses.

The counter result is inversely related to the motor speed in rpm, or the frequency of the reference clock. These nonlinear **1/x** functions can be approximated by a linear relation for the analysis of small variations around a target value. If the motor speed varies **+/- 1 %** around **1200 rpm** then this will result in approximately **-/+ 1 %** variation in the count result. Similarly, if the reference clock frequency changes by **-/+ 0.1 %**, then this will result in **-/+ 0.1 %** error in the count result.

Related MCU Hardware

Clock Generator:

The clock generator will provide the timing reference for the measurements. Precision and stability of an RC oscillator is not good enough to obtain the required **0.1 %** accuracy. For example, precision of the internal RC oscillator provided with the Atmel ATmega328 microcontroller is limited by **10 %**. Even if the oscillator is tuned exactly at the desired frequency, it may still have **1 %** variation due to temperature changes according to the ATmega328 datasheet. Therefore, a crystal oscillator should be used to obtain the required **0.1 %** accuracy.

Timer:

The timer clock frequency and the number of bits in the timer counter are the critical specifications that will be determined in this design example. The functionality of the timer module may introduce additional processing error. In practice, we should refer to the timer module description and determine whether any CPU time is

required to 1) start / stop the timer, and 2) read the count result. If that is the case, then any random variations in the CPU response time will cause additional errors. In this example, we will assume that such processing errors are zero for simplicity.

We should first determine some of the design specifications before starting any calculation:

Motor speed control range: Minimum **1200 rpm**, maximum **2400 rpm** (determined according to the application requirements).

Jitter at the sensor output: +/-10 µs (given by the sensor specifications).

Reference clock precision: +/-100 ppm over the operating temperature range (given by the crystal oscillator specifications).

The following table summarizes the design parameters and the error components for the minimum and maximum motor speeds.

	at 1200 rpm	**at 2400 rpm**
Revolutions per second	20 rev/s	40 rev/s
Revolution time	50 ms	25 ms
Maximum allowed error in time measurement (+/-0.1 % of the revolution time)	**+/-50 µs**	**+/-25 µs**
Error at the sensor output (jitter specification)	+/-10 µs	+/-10 µs
Quantization error (to be determined according to the timer clock frequency)	should be less than +/-35 µs	should be less than +/-12.5 µs
Reference error (+/-100 ppm of the measured time period according to the oscillator specifications)	+/-5.0 µs	+/-2.5 µs
Processing error (given by the requirements of the timer)	0 µs	0 µs

We must add up all error components in order to consider the worst case scenario among all possibilities. The quantization error should be small enough to keep the total error less than the maximum allowed error over the entire speed range. The quantization error will be given by the **+/- T_{clk}** (clock period) of the timer clock. The timer clock frequency should be greater than **80 KHz** to have the quantization error less than **+/-12.5 µs**, and the total error less than **+/-25 µs** at **2400 rpm**.

We can determine the number of timer bits after we choose the timer clock frequency. The timer should be able to count through the longest measurement period without an overflow. In this case, the longest revolution time is **50 ms** at **1200 rpm**. If the counter clock input is exactly **80 KHz** then the corresponding counter result is **4000** (50 ms / 12.5 µs). At least a **12-bit** counter (2^{12} = 4096) is required for the revolution time measurements.

Practical Considerations

In practice, we cannot get the exact values required for our design. Instead we are forced to choose from a set of available predefined options according to the MCU hardware we are using.

Number of timer bits:

We found that the minimum required number of timer bits is **12** when the timer clock frequency is **80 KHz**. We should note that a 12-bit counter will overflow when the motor speed drops down to **1172 rpm** (revolution time exceeds **51.19 ms**). Depending on the load conditions of the motor, an overflow may easily occur since **1172 rpm** is very close to **1200 rpm**. It will be wise to use a **13-bit** timer that can provide sufficient safety margin to prevent an overflow.

The most of the common MCUs have either 8-bit or 16-bit timers. If a 16-bit timer is available, then the timer clock frequency can be as high as **640 KHz** (8 x 80 KHz). This will reduce the quantization error by a factor of eight while the overflow condition remains the same since three more counter bits are available.

Timer clock frequency:

Timer modules are usually equipped with a ***clock prescaler*** providing a variety of timer clocks to choose from. The most common type of prescaler is a simple counter that can reduce the main system clock frequency by a factor of $\mathbf{2^N}$. The divider power, **N**, is programmable through the SFRs that control the timer functions. For example, in an MCU that has 4 MHz main system clock, the timer clock frequency at the prescaler output can be 4 MHz, 2 MHz, 1 MHz, 500 KHz, 250 KHz, 125 KHz, 62.5 KHz, or 31.25 KHz. If these are the available frequency settings for our design, then the lowest frequency we can choose is **125 KHz** for a **13-bit** timer. Using a frequency setting lower than 80 KHz will result in too much quantization error.