مدينة زويل للعلوم والتكنولوجيا
Zewail City of Science and Technology

COMMUNICATION AND INFORMATION ENGINEERING

CIE 314
Embedded Systems Fundamentals

Lecture #9
RTOS Basics

Instructor:
Dr. Ahmad El-Banna

Agenda

- Introduction
- RTOS vs. GPOS
- Defining RTOS
- Common components in an RTOS kernel
- Key Characteristics of an RTOS
- Design Tips

History of Operating Systems

- In the early days of computing, developers created software applications that included low-level machine code to initialize and interact with the system's hardware directly.
- This tight integration between the software and hardware resulted in non-portable applications.
- A small change in the hardware might result in rewriting much of the application itself.
- Obviously, these systems were difficult and costly to maintain.

History of Operating Systems..

- As the software industry progressed, operating systems that provided the basic software foundation for computing systems evolved and facilitated the abstraction of the underlying hardware from the application code.
- In addition, the evolution of operating systems helped shift the design of software applications from large, monolithic applications to more modular, interconnected applications that could run on top of the operating system environment.
- Over the years, many versions of operating systems evolved.
- These ranged from general-purpose operating systems (GPOS), such as UNIX and Microsoft Windows, to smaller and more compact real-time operating systems, such as VxWorks.

History of Operating Systems…

- In the 60s and 70s, when mid-sized and mainframe computing was in its prime, UNIX was developed to facilitate multi-user access to expensive, limited-availability computing systems.
- UNIX allowed many users performing a variety of tasks to share these large and costly computers. multi-user access was very efficient: one user could print files, for example, while another wrote programs.
- Eventually, UNIX was ported to all types of machines, from microcomputers to supercomputers.
- In the 80s, Microsoft introduced the Windows operating system, which emphasized the personal computing environment.
- Targeted for residential and business users interacting with PCs through a graphical user interface, the Microsoft Windows operating system helped drive the personal-computing era.

History of Operating Systems….

- Later in the decade, momentum started building for the next generation of computing: the post-PC, embedded-computing era.
- To meet the needs of embedded computing, commercial RTOSes, such as VxWorks, were developed.
- Although some functional similarities exist between RTOSes and GPOSes, many important differences occur as well.
- These differences help explain why RTOSes are better suited for real-time embedded systems.

RTOS vs. GPOS
Core functional similarities

- Some core functional similarities between a typical RTOS and GPOS include:
  - some level of multitasking,
  - software and hardware resource management,
  - provision of underlying OS services to applications, and
  - abstracting the hardware from the software application.

RTOS vs. GPOS..
Key functional differences

- On the other hand, some key functional differences that set RTOSes apart from GPOSes include:
  - better reliability in embedded application contexts,
  - the ability to scale up or down to meet application needs,
  - faster performance,
  - reduced memory requirements,
  - scheduling policies tailored for real-time embedded systems,
  - support for diskless embedded systems by allowing executables to boot and run from ROM or RAM, and
  - better portability to different hardware platforms.

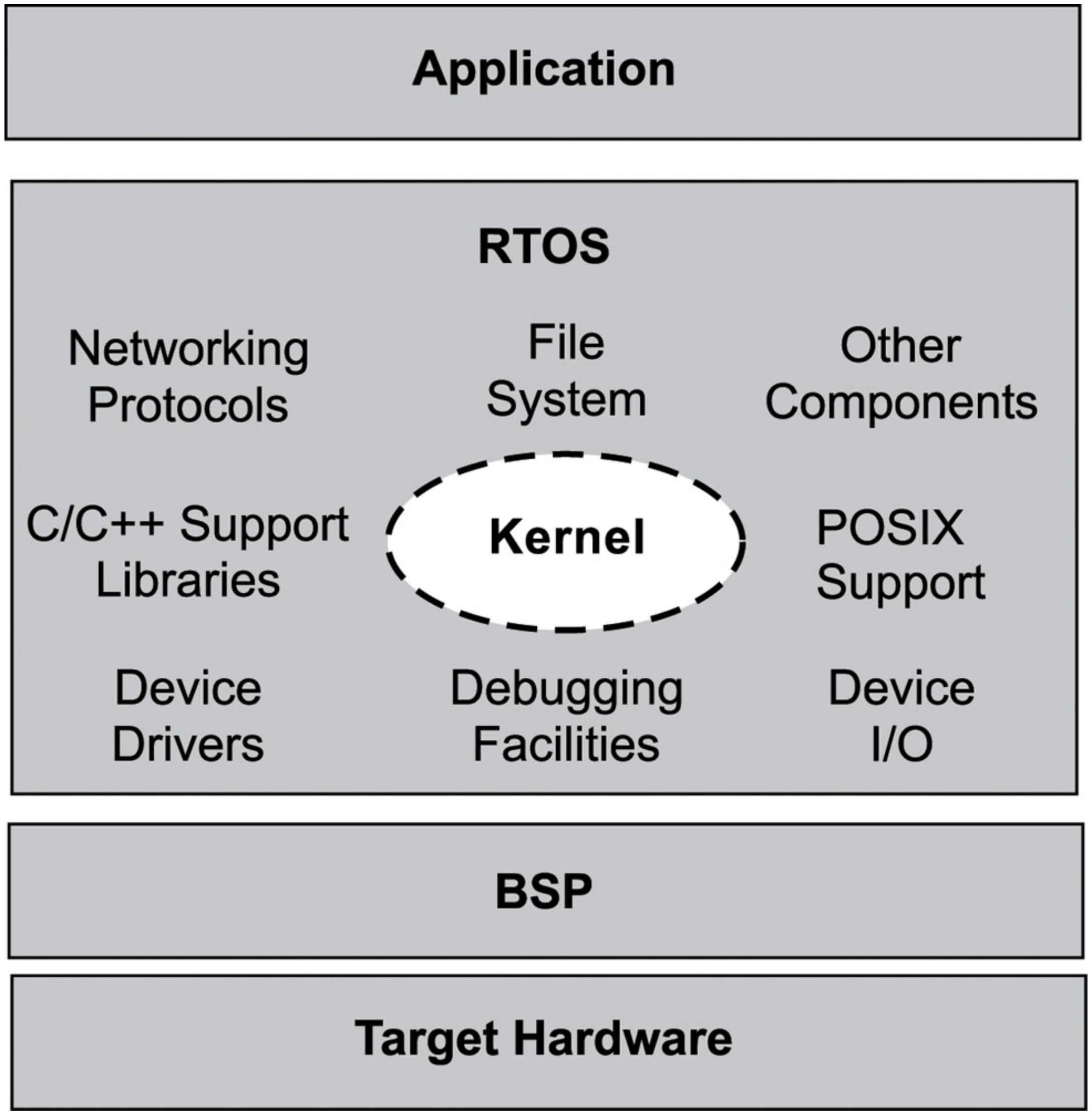
Defining an RTOS

- A real-time operating system (RTOS) is a program that
  - schedules execution in a timely manner,
  - manages system resources, and
  - provides a consistent foundation for developing application code.
- Application code designed on an RTOS can be quite diverse, ranging from a simple application for a digital stopwatch to a much more complex application for aircraft navigation.
- Good RTOSes, therefore, are scalable in order to meet different sets of requirements for different applications.

RTOS

- For example, in some applications, an RTOS comprises only a kernel, which is the core supervisory software that provides minimal logic, scheduling, and resource-management algorithms.
- Every RTOS has a kernel.
- On the other hand, an RTOS can be a combination of various modules, including the kernel, a file system, networking protocol stacks, and other components required for a particular application

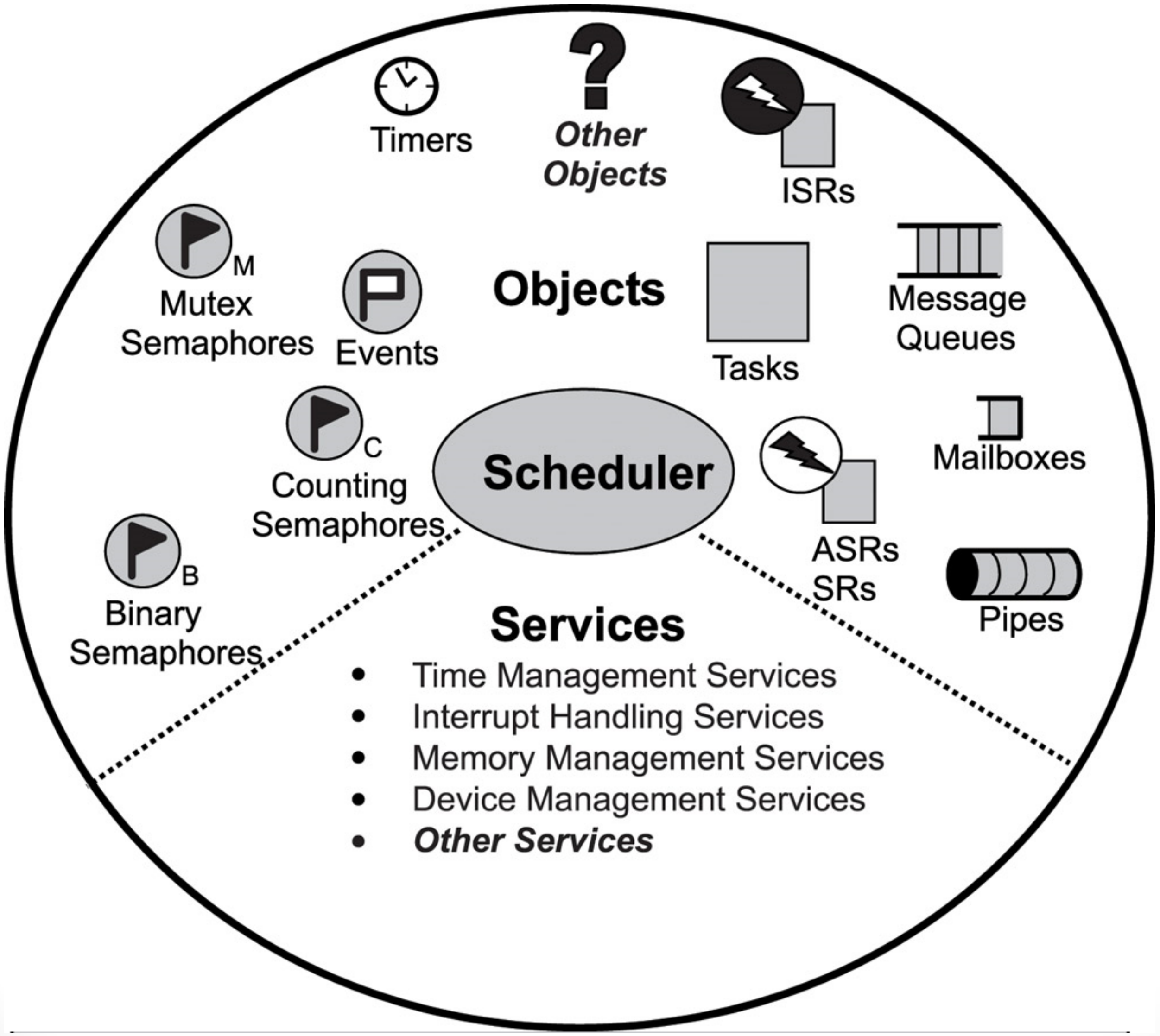
High-level view of an RTOS, its kernel, and other components found in embedded systems



(**POSIX**): Portable Operating System Interface

(BSP): board support package

Common components in an RTOS kernel that including objects, the scheduler, and some services.



Schedulers

- The scheduler is at the heart of every kernel.
- A scheduler provides the algorithms needed to determine which task executes when.
- Main items related to schedulers:
  - schedulable entities
  - multitasking
  - context switching
  - dispatcher
  - scheduling algorithms.

Schedulable entities & Context Switching

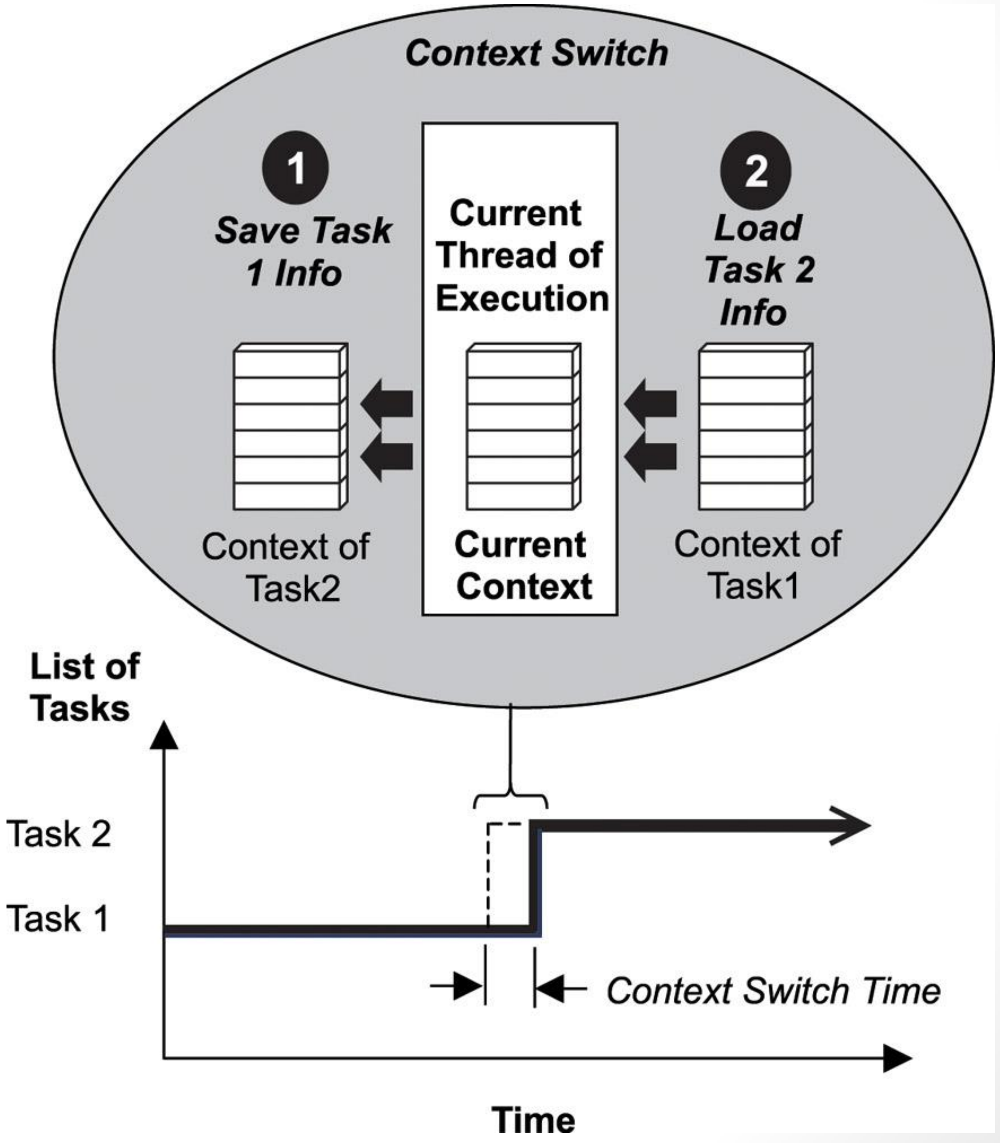
- Schedulable entities:
  - A schedulable entity is a kernel object that can compete for execution time on a system, based on a predefined scheduling algorithm.
  - Tasks and processes are all examples of schedulable entities found in most kernels.
- Task Context:
  - Each task has its own context, which is the state of the CPU registers required each time it is scheduled to run.
  - Every time a new task is created, the kernel also creates and maintains an associated task control block (TCB).
  - TCBs are system data structures that the kernel uses to maintain task-specific information.
  - TCBs contain everything a kernel needs to know about a particular task.
  - When a task is running, its context is highly dynamic. This dynamic context is maintained in the TCB.
  - When the task is not running, its context is frozen within the TCB, to be restored the next time the task runs.

Context Switching

- When the kernel’s scheduler determines that it needs to stop running task 1 and start running task 2, it takes the following steps:
  - The kernel saves task 1’s context information in its TCB.
  - It loads task 2’s context information from its TCB, which becomes the current thread of execution.
  - The context of task 1 is frozen while task 2 executes, but if the scheduler needs to run task 1 again, task 1 continues from where it left off just before the context switch.
- The time it takes for the scheduler to switch from one task to another is the context switch time.
- It is relatively insignificant compared to most operations that a task performs.

Multitasking using a context switch.

- Multitasking is the ability of the operating system to handle multiple activities within set deadlines.



Dispatcher

- The dispatcher is the part of the scheduler that performs context switching and changes the flow of execution.
- At any time an RTOS is running, the flow of execution, also known as flow of control, is passing through one of three areas:
  - through an application task,
  - through an ISR, or
  - through the kernel.
- When a task or ISR makes a system call, the flow of control passes to the kernel to execute one of the system routines provided by the kernel.
- When it is time to leave the kernel, the dispatcher is responsible for passing control to one of the tasks in the user's application.
- It will not necessarily be the same task that made the system call.
- It is the scheduling algorithms of the scheduler that determines which task executes next.
- It is the dispatcher that does the actual work of context switching and passing execution control.

Scheduling Algorithms

- The scheduler determines which task runs by following a scheduling algorithm (also known as scheduling policy).

<u>Types:</u>

- Non-preemtive Scheduling :
  - statically scheduled (task hold CPU until it completes)
  - Cooperative multitasking (task give up CPU by itself due to the lack of resources) e.g. pure round-robin.
- Priority-based Scheduling:
  - fixed priority scheduling algorithms
    - Static timing scheduling
    - Round-robin scheduling
    - Rate Monotonic Scheduling(RMS)
  - dynamic priority-based scheduling
    - Earliest Deadline First(EDF)

Objects

- Kernel objects are special constructs that are the building blocks for application development for real-time embedded systems.
- The most common RTOS kernel objects are:
  - **Tasks**—are concurrent and independent threads of execution that can compete for CPU execution time.
  - **Semaphores**—are token-like objects that can be incremented or decremented by tasks for synchronization or mutual exclusion.
  - **Message Queues**—are buffer-like data structures that can be used for synchronization, mutual exclusion, and data exchange by passing messages between tasks.

Services

- Along with objects, most kernels provide services that help developers create applications for real-time embedded systems.
- These services comprise sets of API calls that can be used to perform operations on kernel objects or can be used in general to facilitate timer management, interrupt handling, device I/O, and memory management.
- Other services might be provided; these services are those most commonly found in RTOS kernels.

Key Characteristics of an RTOS

- An application's requirements define the requirements of its underlying RTOS.
- Some of the more common attributes are
  - Reliability
    - Depending on the application, the system might need to operate for long periods without human intervention.
  - Predictability
    - The RTOS used in real-time systems needs to be predictable to a certain degree to meet time requirements .
    - The term deterministic describes RTOSes with predictable behavior, in which the completion of operating system calls occurs within known timeframes.

Key Characteristics of an RTOS

- Performance
  - This requirement dictates that an embedded system must perform fast enough to fulfill its timing requirements.
  - Typically, the more deadlines to be met-and the shorter the time between them-the faster the system's CPU must be.
  - Although underlying hardware can dictate a system's processing power, its software can also contribute to system performance.
- Compactness
  - Application design constraints and cost constraints help determine how compact an embedded system can be.
  - For example, a cell phone clearly must be small, portable, and low cost. These design requirements limit system memory, which in turn limits the size of the application and operating system.
- Scalability
  - Because RTOSes can be used in a wide variety of embedded systems, they must be able to scale up or down to meet application-specific requirements.
  - Depending on how much functionality is required, an RTOS should be capable of adding or deleting modular components, including file systems and protocol stacks.

DESIGN TIPS

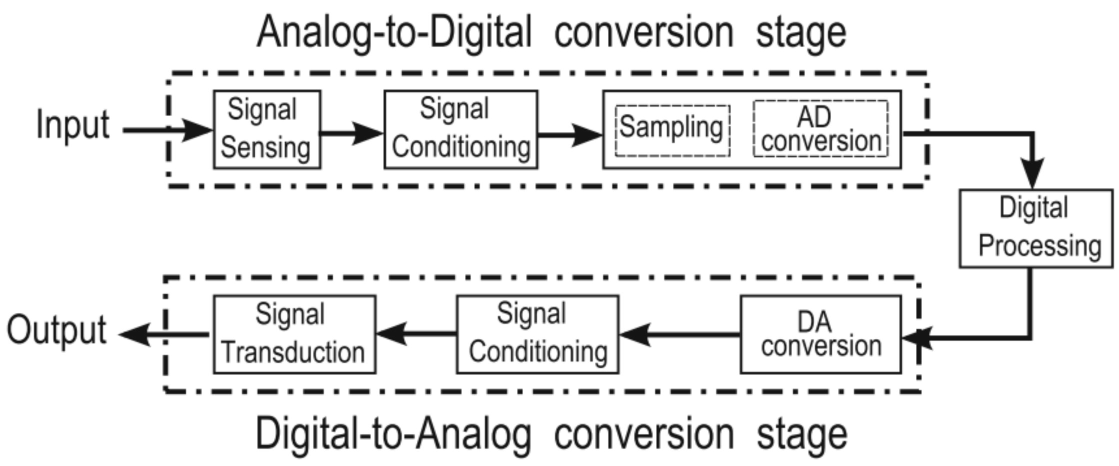


Fig. 10.1 Block description for mixed-signal processing chain

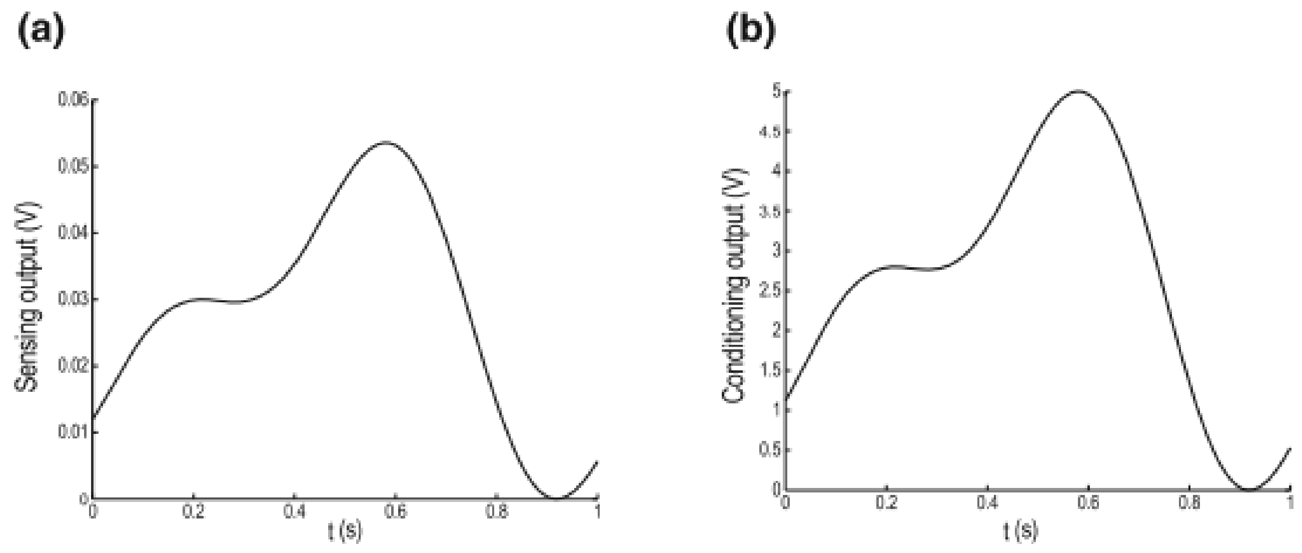


Fig. 10.22 From analog to the digital process: **a** a sensing output, **b** after conditioning

Fig. 10.23 Sampling and encoding with four bits the signal in Fig. 10.22b

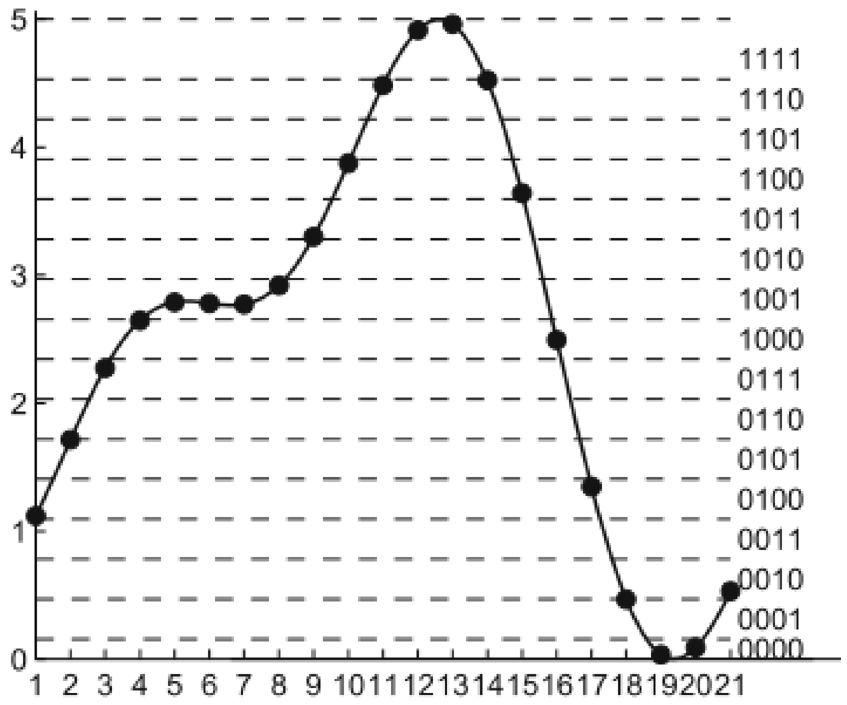
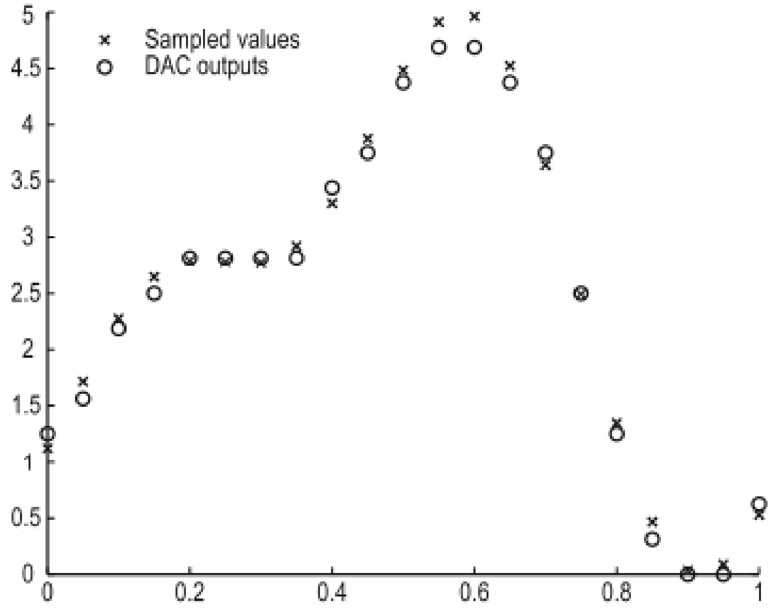


Fig. 10.24 Output values from DAC



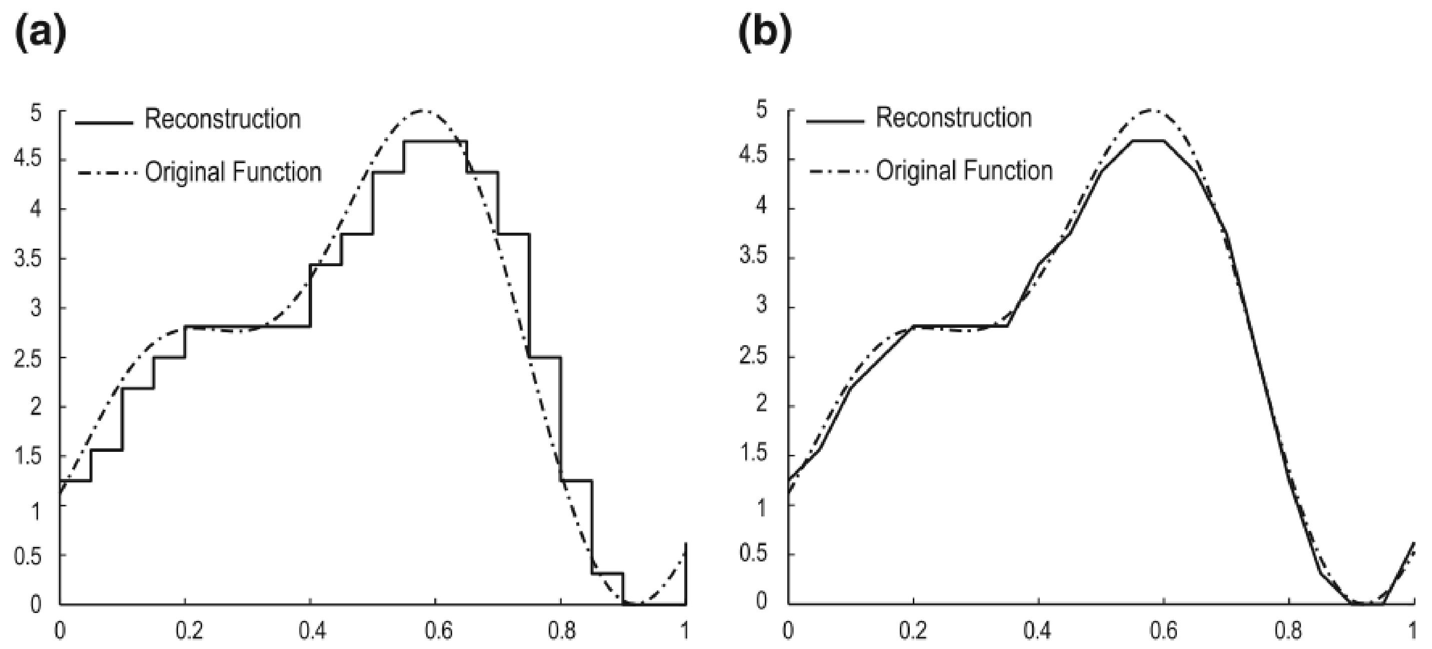


Fig. 10.25 Reconstructing the analog function of Fig. 10.22b using 21 samples and 4-bit words: **a** step-wise form; **b** piece-wise linear form

Features of an ADC

- Sampling rate – rate at which continuous analog signal is polled e.g. 1000 samples/sec
- Quantization – divide analog signal into discrete levels $N_q = 2^n$
  - where N_q = quantisation levels; and n is the number of bits.
- Resolution – depends on number of quantization levels

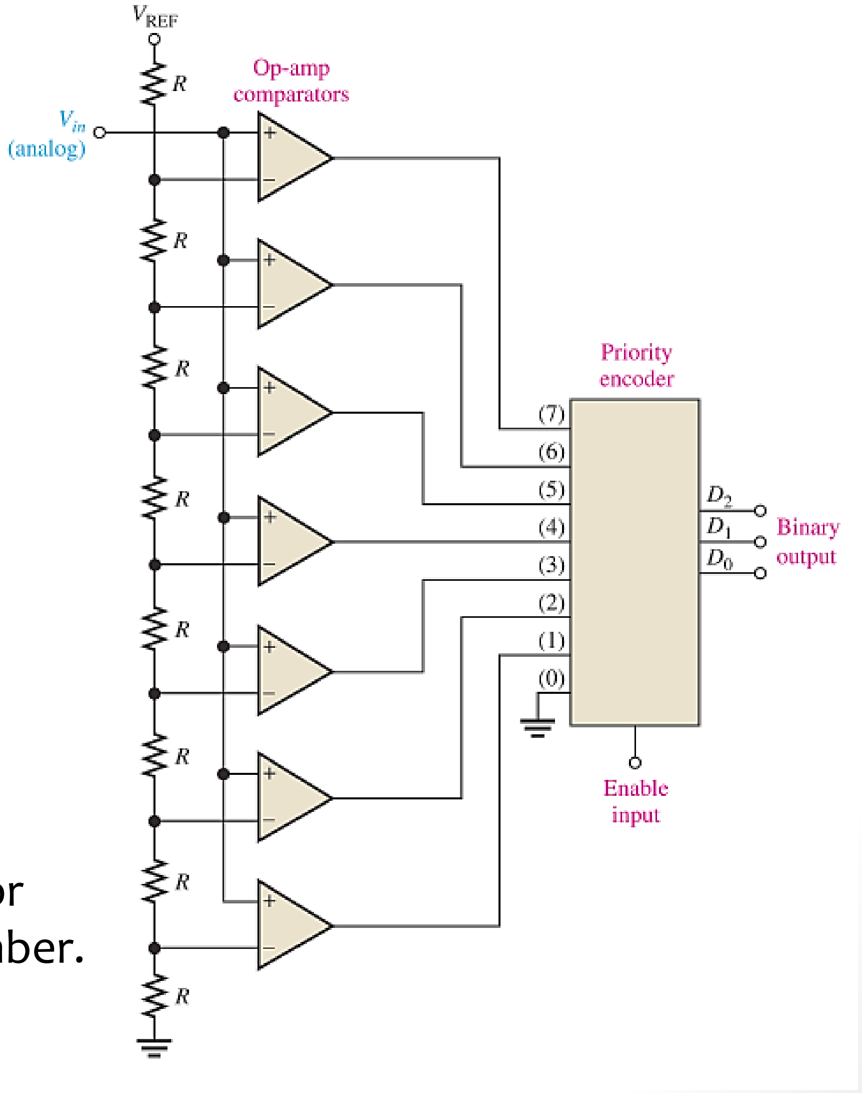
$$R_{ADC} = \frac{L}{N_q - 1} = \frac{L}{2^n - 1}$$

  - where R_{ADC} is the resolution of the ADC; L is the full-scale range of the ADC
- Conversion time – how long it takes to convert the sampled signal to digital code
- Conversion method – means by which analog signal is encoded into digital equivalent
  - Example – Successive approximation method & Flash

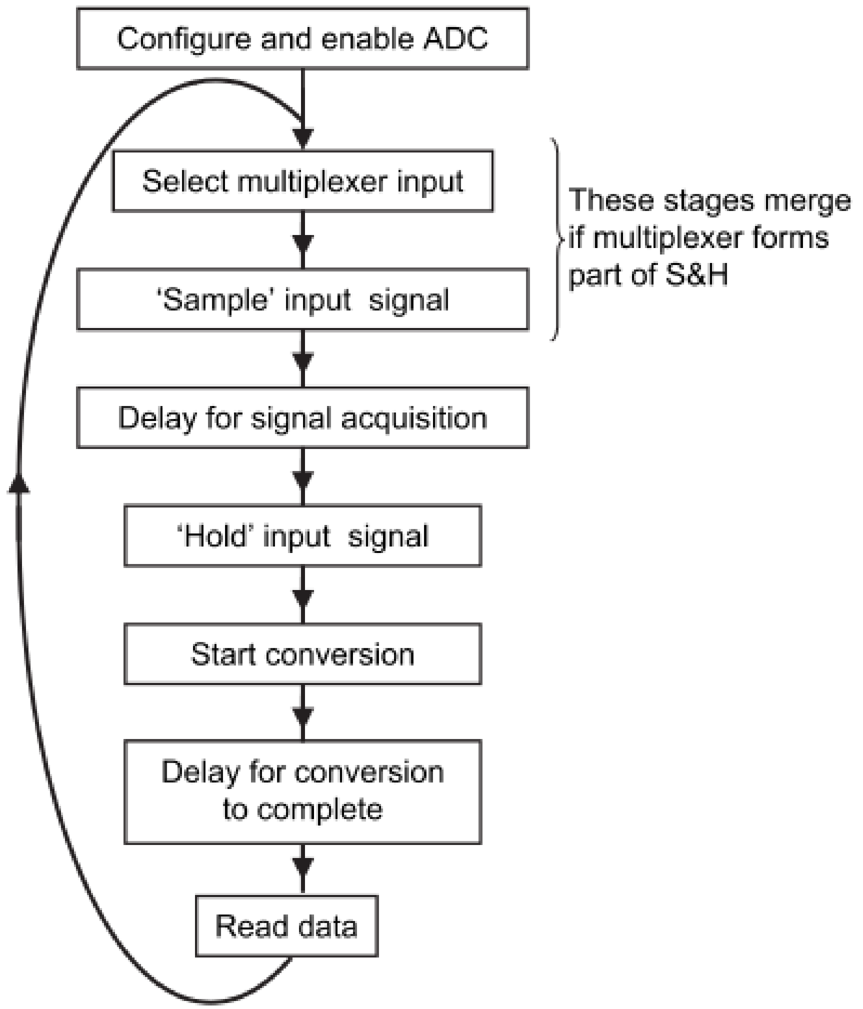
Flash ADC

- The **simultaneous**, or **flash,** method of A/D conversion uses **parallel comparators** to compare the linear input signal with **various reference** voltages developed by a voltage divider.
- When the **input** voltage **exceeds the reference** voltage for a given comparator, a **high level is produced** on that comparator's output.

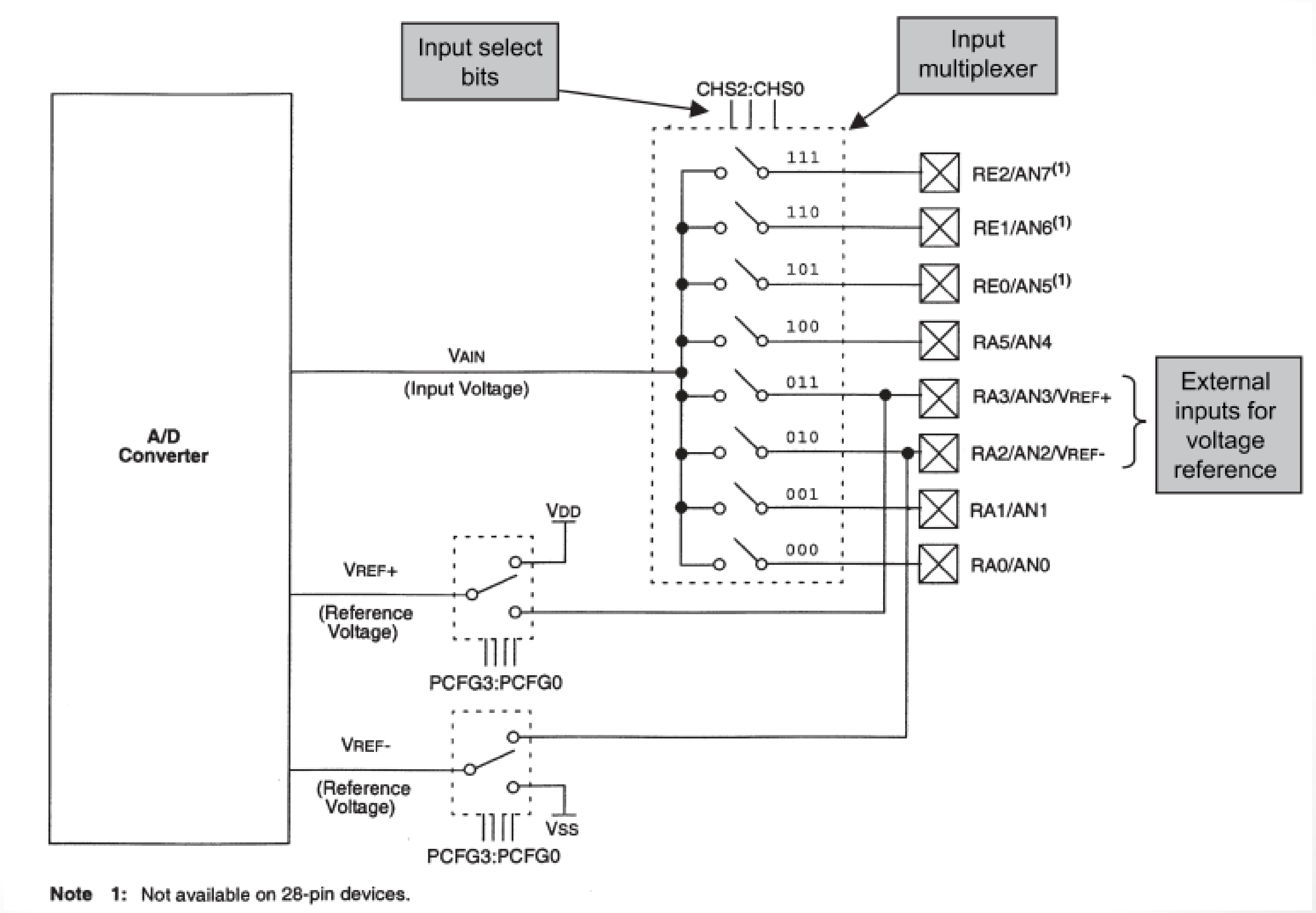
→**2^n- 1** comparators are required for conversion to an **n-digit** binary number.



Typical timing requirement of one analog-to-digital conversion



Example: PIC 16F87XA ADC module

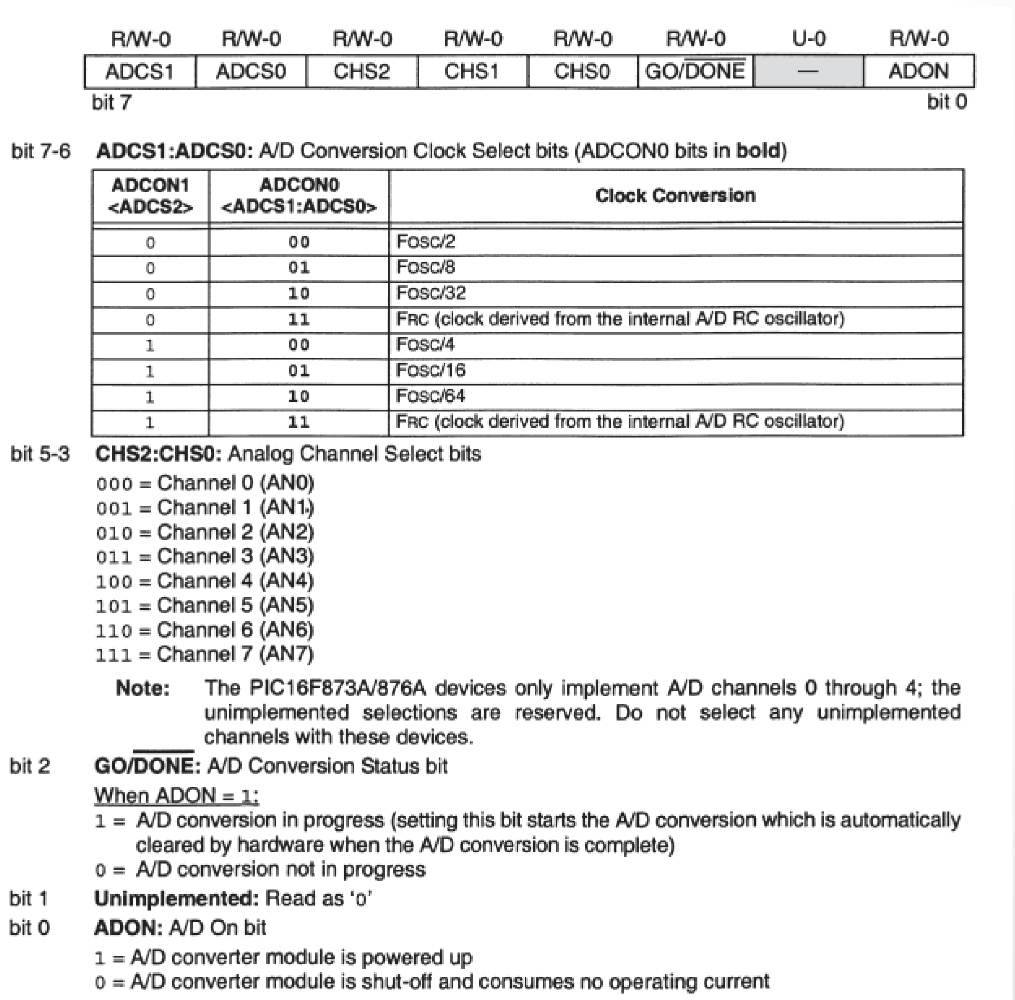


Note 1: Not available on 28-pin devices.

Controlling the ADC

- The ADC is controlled by two SFRs, ADCON0 and ADCON1).
- The result of the conversion is placed in two further SFRs, ADRESH and ADRESL

The ADCON0 register (address 1F H)



R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	**00**	FOSC/2
0	**01**	FOSC/8
0	**10**	FOSC/32
0	**11**	FRC (clock derived from the internal A/D RC oscillator)
1	**00**	FOSC/4
1	**01**	FOSC/16
1	**10**	FOSC/64
1	**11**	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

Note: The PIC16F873A/876A devices only implement A/D channels 0 through 4; the unimplemented selections are reserved. Do not select any unimplemented channels with these devices.

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

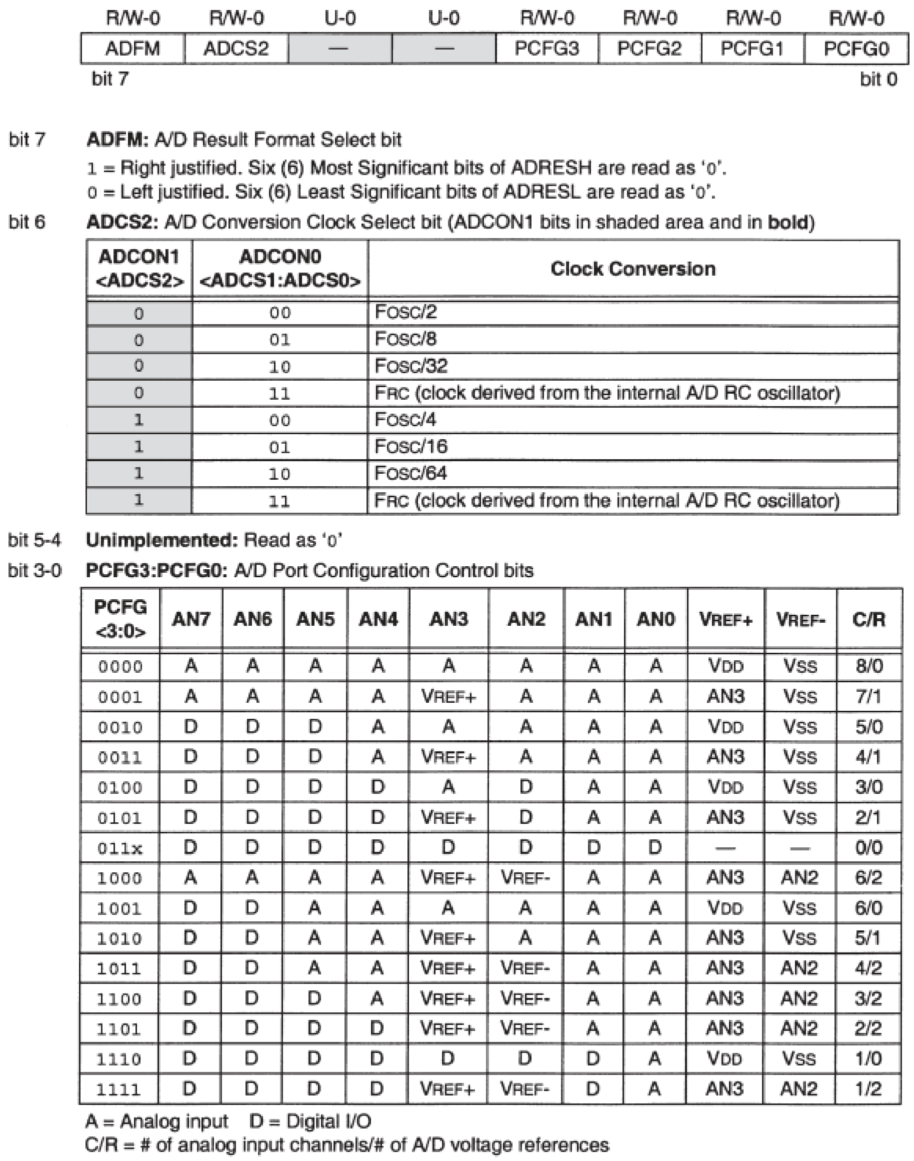
When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit

1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

The ADCON1 register (address 9F H)



R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

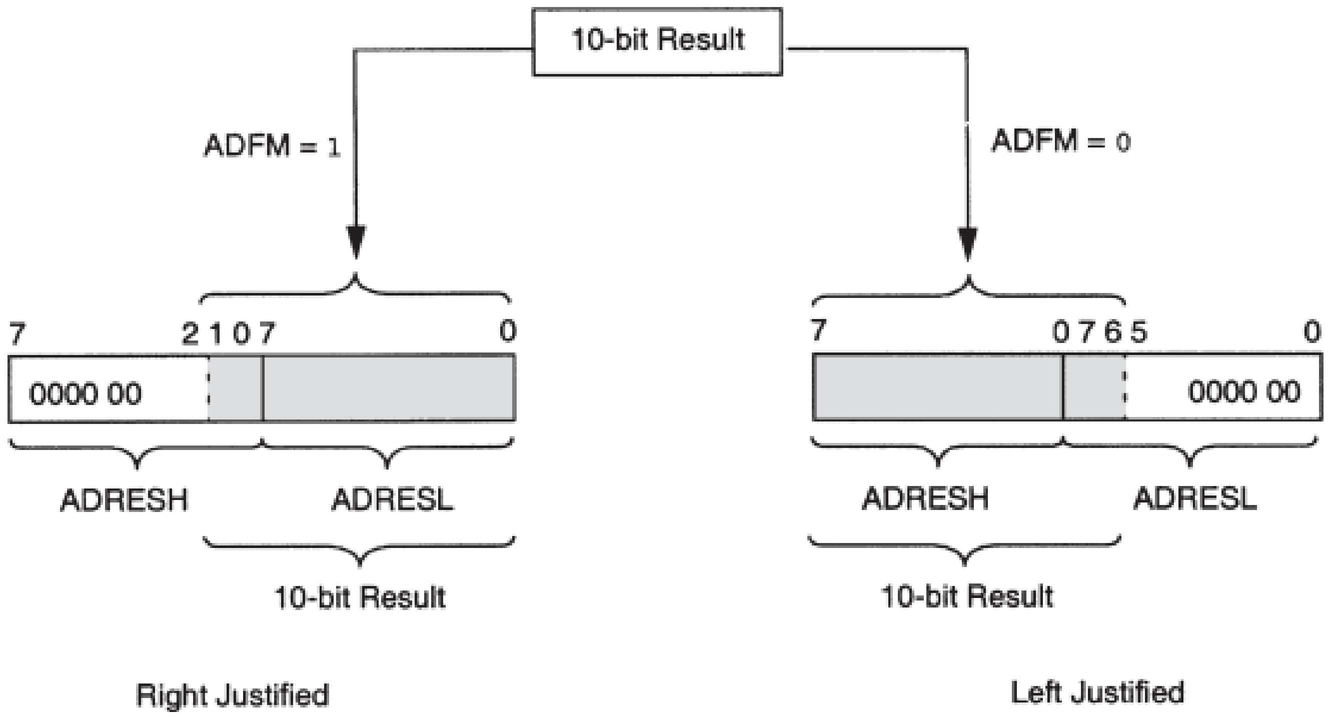
bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Formatting the analog-to-digital converter conversion result



DAC

- The reverse function of ADC.
- Usually needs external interface circuit.
- Convert digital values into continuous analogue signal
  - Decoding digital value to an analogue value at discrete moments in time based on value within register

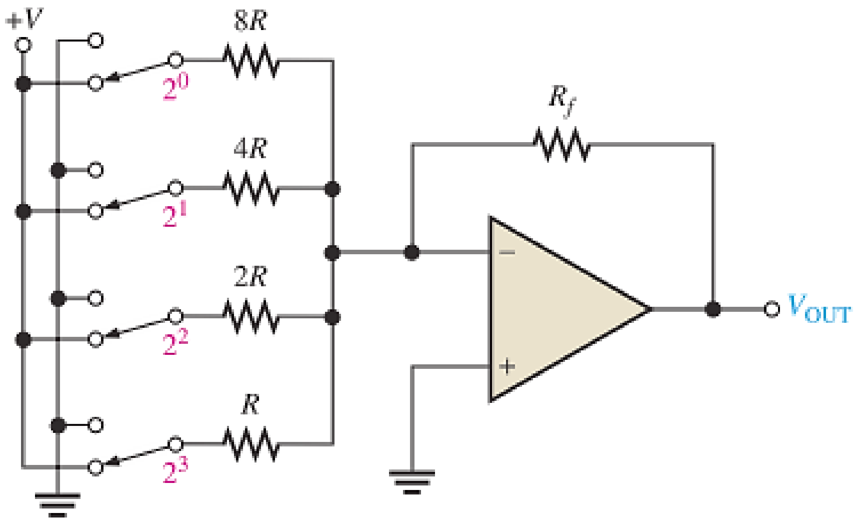
$$E_0 = E_{ref}\left\{0.5B_1 + 0.25B_2 + \cdots + \left(2^n\right)^{-1} B_n\right\}$$

Where E_o is output voltage; E_{ref} is reference voltage; B_n is status of successive bits in the binary register

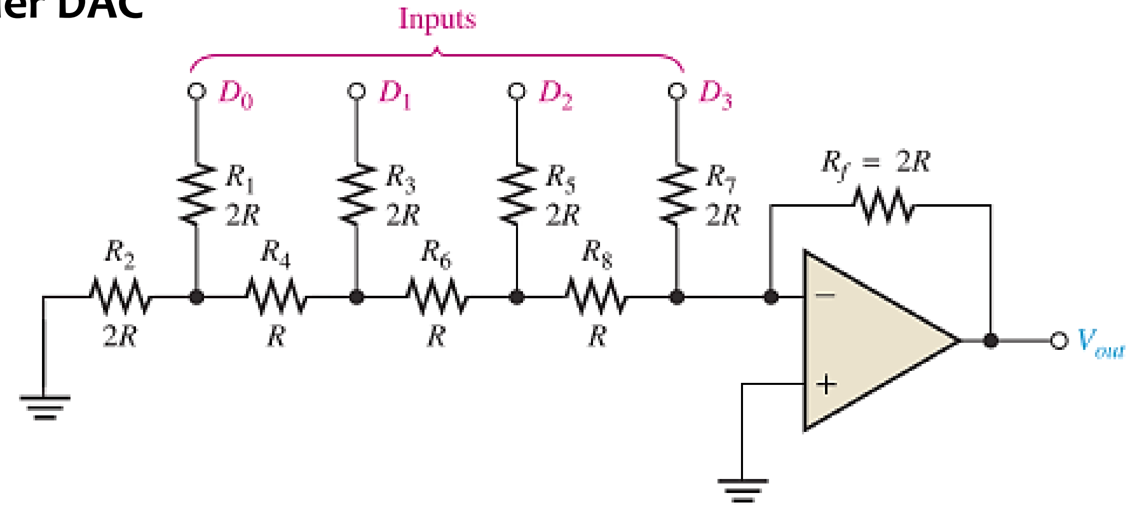
Examples of DAC Circuits

- **Scaling Adder as a four-digit DAC**

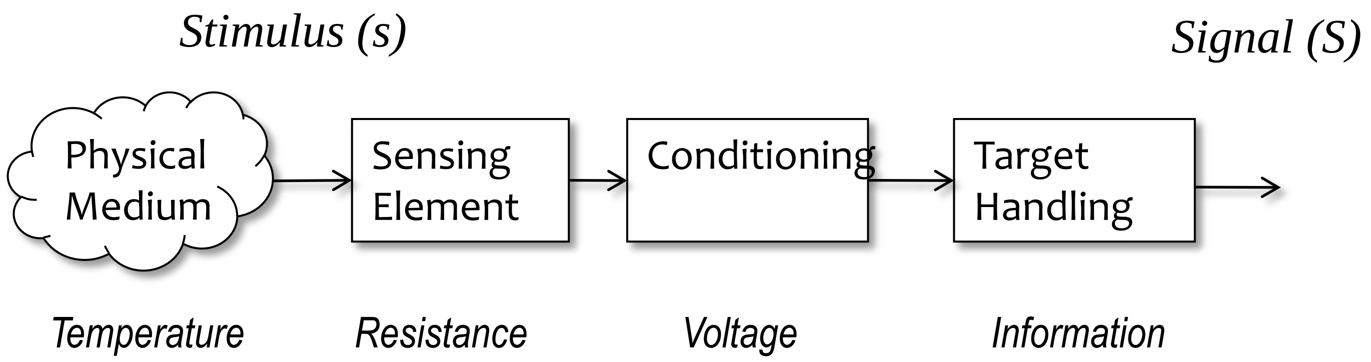
I_0=+V/8R	$V_{out(D0)}$=-R_f I_0
I_1=+V/4R	$V_{out(D1)}$=-R_f I_1
I_2=+V/2R	$V_{out(D2)}$=-R_f I_2
I_3=+V/R	$V_{out(D3)}$=-R_f I_3



- **An R/2R ladder DAC**

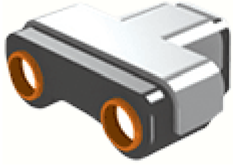


Sensors



- A sensor is a transducer that converts a physical stimulus from one form into a more useful form to measure the stimulus.
- Two basic categories:
  1. Analog
  2. Discrete
     - Binary
     - Digital (e.g., pulse counter)

Sensors..



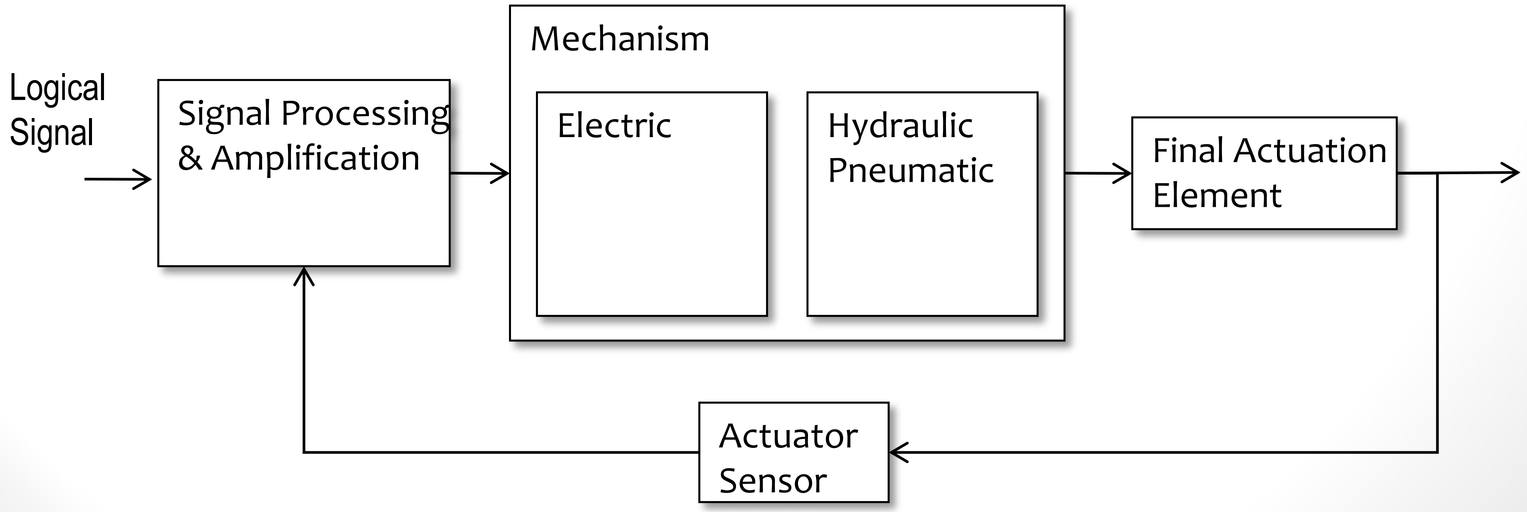
- Main categories
  - Any energy radiated? Passive vs. active sensors
  - Sense of direction? Omidirectional?

  - Passive, omnidirectional
    - Examples: light, thermometer, microphones, hygrometer, …
  - Passive, narrow-beam
    - Example: Camera
  - Active sensors
    - Example: Radar

- Important parameter: Area of coverage
  - Which region is adequately covered by a given sensor?

Actuators

- **Actuators** are hardware devices that convert a controller command signal into a change in a physical parameter
- The change is usually *mechanical* (e.g., position or velocity)
- An actuator is also a *transducer* because it changes one type of physical quantity into some alternative form
- An actuator is usually activated by a low-level command signal, so an *amplifier* may be required to provide sufficient power to drive the actuator

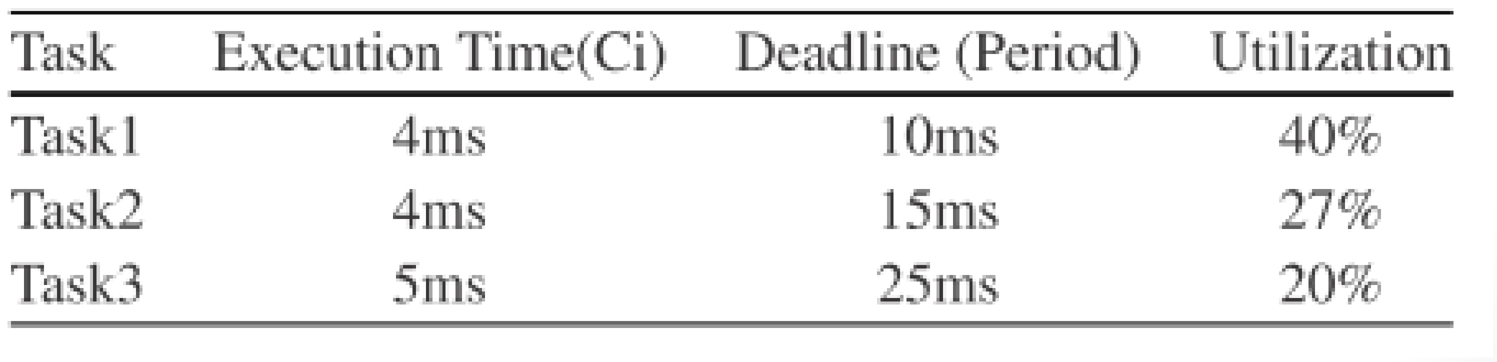


Types of Actuators

1. Electrical actuators
   - Electric motors
     - DC servomotors
     - AC motors
     - Stepper motors
   - Solenoids
2. Hydraulic actuators
   - Use hydraulic *fluid* to amplify the controlle command signal
3. Pneumatic actuators
   - Use compressed *air* as the driving force

Assignment#4

- Write a code that implements the *cyclic scheduling* algorithm for the below scenario and test it.
  - **Task1:** toggles a led status between on & off.
  - **Task2:** reads the status of a button and turn a led on or off depending on the button status.
  - **Task3:** increments a 7-segment display from 0 to 9 then 0 and so on.



Task	Execution Time(Ci)	Deadline (Period)	Utilization
Task1	4ms	10ms	40%
Task2	4ms	15ms	27%
Task3	5ms	25ms	20%

- For more details, refer to:
  - Chapter 4 at **Real-time concepts for embedded systems**, CMP Books, 2003 by Qing Li and Carolyn Yao (ISBN:1578201241).
  - Chapter 5 at **Embedded Software Development with C,** Springer 2009 by Kai Qian et al.
  - Chapter 10 at **Introduction to Embedded Systems,** Springer 2014 by Manuel Jiménez et al.
- The lecture is available online at:
  - *http://bu.edu.eg/staff/ahmad.elbanna-courses*
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg