Part 2
Peripherals for DSP Applications

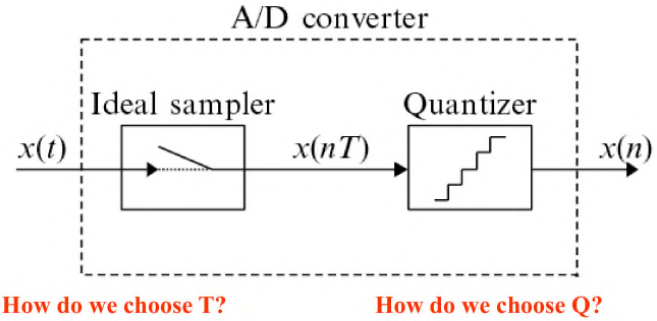
- Sampling
- Analog to digital converter
- Digital to analog converter
- Interrupts
- Real-time operation and scheduling

Analog signal to digital signal

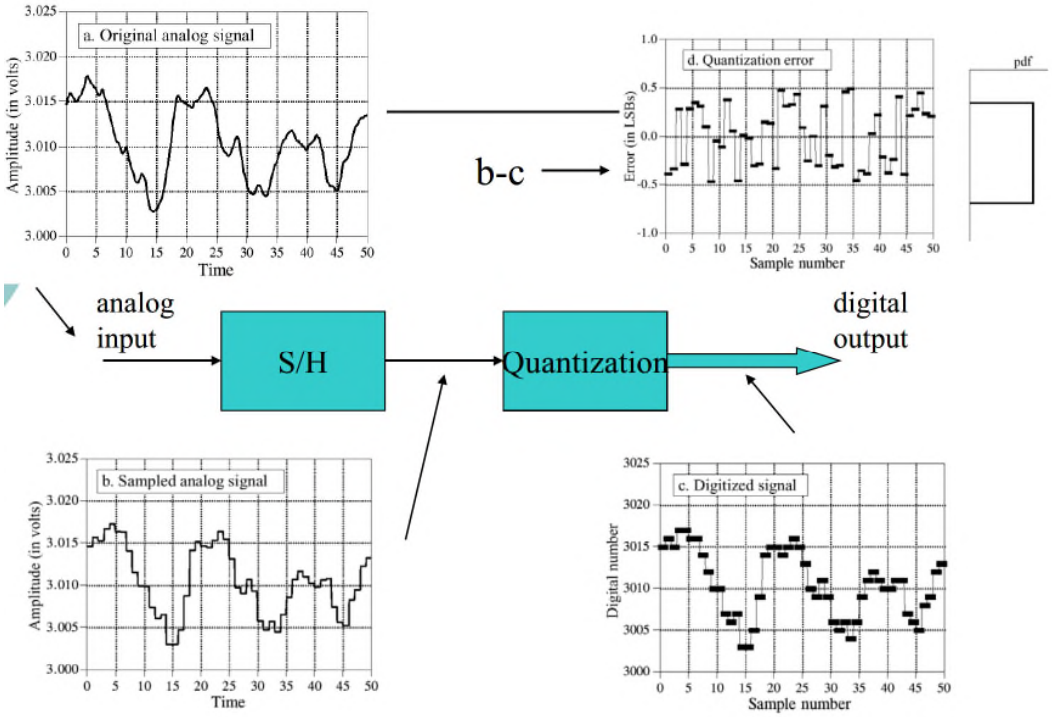
- Analog signal = continuous-time, continuous-value signal
- Digital signal = discrete-time, discrete-value signal

Analog to Digital Conversion (ADC):

- Sampling (convert continuous-time to discrete-time)
- Quantization (convert continuous-value to discrete-value)



How do we choose T? How do we choose Q?



Sample and Hold (S/H):

- Output b can change only at discrete times 1,2,3…
- At these times, b is equal to the instantaneous value of a

Signal values between these times are ignored ➔ error?

Quantization:
Signal c can take one of only a discrete set of possible values (finite precision)

Discrete values are different from original values ➔ error

Sampling and quantization are analyzed separately because

- Sampling and quantization degrade the signal differently
- Sampling and quantization are controlled by different parameters in the hardware

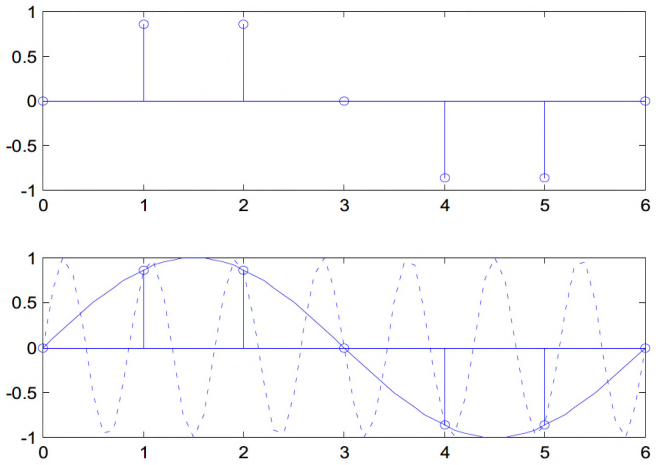
Sampling

Uniform sampling / Periodic sampling

- Sample a continuous-time signal at the interval of T
- Signal values in between the samples are lost
- Can the signal be reconstructed from its samples?

In general, reconstruction is not possible.

- Samples obtained from a sinusoid
  [0 0.866 0.866 0 –0.866 –0.866 0]
- While reconstructing, there is ambiguity



—— $\sin\left(2\pi\frac{1}{6}t\right)$
---- $\sin\left(2\pi\frac{7}{6}t\right)$

- Input sinusoid of frequency f_0 , $x_c(t) = \sin(2\pi f_0 t)$
- Sampling frequency f_s , sampling interval $T = \frac{1}{f_s}$
- Discrete-time output signal $x(n) = \sin(2\pi f_0 nT)$

- Input sinusoid of frequency $f_0 + kf_s$ for any integer k

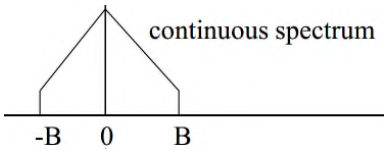
$$x_c(t) = \sin(2\pi(f_0 + kf_s)t)$$

- Same sampling frequency
- Same discrete-time output signal

➔ f_0 and $f_0 + kf_s$ are indistinguishable after sampling

Sampling low-pass signals

- $x_c(t)$ is a bandlimited low-pass signal,
  $X_c(f) = 0 \quad \text{for} \quad B < |f|$
- B = bandwidth

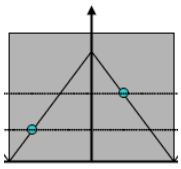


Nyquist sampling theorem

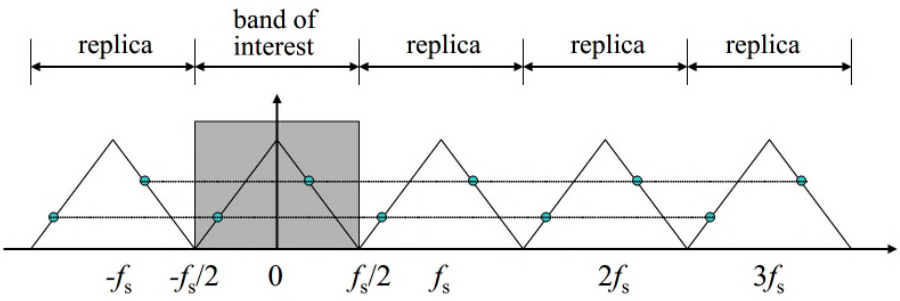
$x_c(t)$ is uniquely determined by its samples $x(n)$
($x_c(t)$ can be perfectly reconstructed from its samples $x(n)$)
if $f_s > 2B$

Effect of sampling on the spectrum of an input signal:

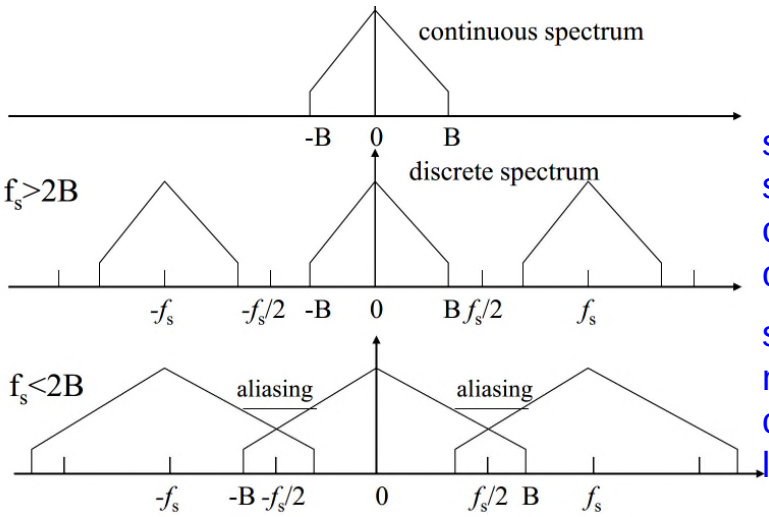
- Input spectrum



- Output spectrum = spectral copies at interval of f_s



Nyquist sampling theorem in spectral domain:



sampling theorem satisfied → copies don't overlap

sampling theorem not satisfied → copies overlap, leads to aliasing

Aliasing

Recall that $\sin(2\pi t/6)$ and $\sin(14\pi t/6)$ both have the same samples.

- Sampling frequency f_s = 1 Hz

- $\sin(2\pi t/6)$ has frequency 1/6 Hz, sampling theorem satisfied

- $\sin(14\pi t/6)$ has frequency 7/6 Hz, sampling theorem not satisfied
- 7/6 = $f_0 + kf_s$ = (1/6) + f_s , where f_0 = 1/6, k = 1
- So 7/6 Hz apparently looks like 1/6 Hz

- Frequency components higher than $f_s/2$ fold back into the frequency range 0 to $f_s/2$. This is aliasing.

Example: Consider the continuous-time signal

$$x_c(t) = 3\cos(2\pi \times 50 \times t)$$

a) Determine the minimum sampling rate required to avoid aliasing.

b) If it is sampled at 200 Hz, what discrete-time signal is obtained after sampling?

c) If it is sampled at 75 Hz, what discrete-time signal is obtained after sampling?

d) What is the frequency of a sinusoid that yields samples identical to those obtained in part (c)?

a) Since f_0 = 50 Hz, f_s > 100 Hz.

b)
$$f_s = 200\text{Hz}$$
$$\Rightarrow T = 0.005$$
$$x(n) = 3\cos(2\pi \times 50 \times 0.005n)$$
$$= 3\cos(0.5\pi n)$$

c)
$$f_s = 75\text{Hz}$$
$$\Rightarrow T = 1/75$$
$$x(n) = 3\cos(2\pi \times 50 \times n/75)$$
$$= 3\cos(4\pi n/3)$$

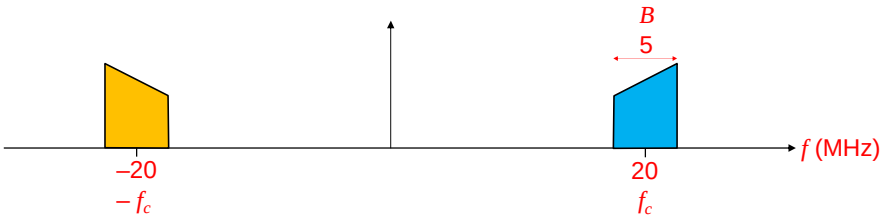
d) 50 = $f_0 + kf_s$ = (–25) + 75
So f_0 = –25, or it yields samples identical to 25 Hz

Sampling band-pass signals

Also known as IF sampling, harmonic sampling, sub-Nyquist sampling, undersampling

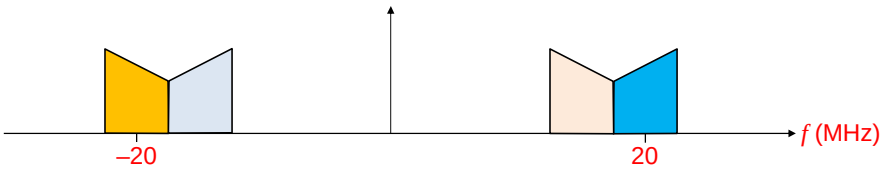
- $x_c(t)$ is centered around some non-zero frequency f_c
- Bandwidth B
- Requires less f_s than Nyquist low-pass sampling

Consider a band-pass signal of bandwidth B = 5 MHz centered around f_c = 20 MHz.



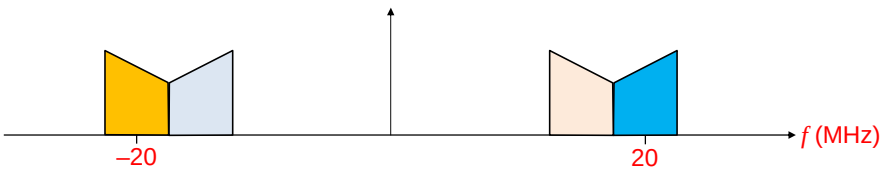
- Maximum frequency = 22.5 MHz
- Nyquist low-pass sampling frequency = 45 MHz

Sample at $f_s = 2f_c - B = 35$ MHz



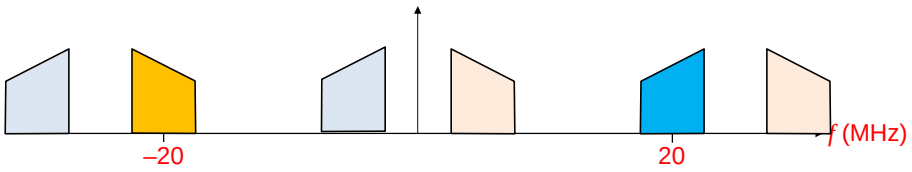
- Spectral copies just touch the original spectral components
- No aliasing

Reduce sampling frequency, $f_s \leq 2f_c - B = 35$ MHz



- Spectral copies move away as shown

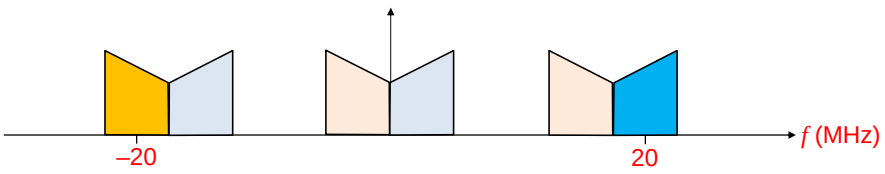
Can reduce up to $f_s \geq (2f_c + B)/2 = 22.5$ MHz



- Second spectral copies move in and just touch the original spectral components
- No aliasing

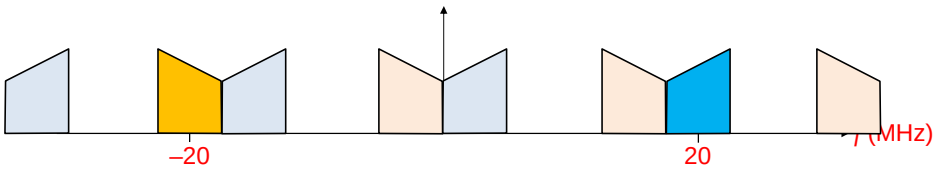
Thus, sampling at $(2f_c + B)/2 \leq f_s \leq 2f_c - B$ (in this example, 22.5 MHz $\leq f_s \leq$ 35 MHz) gives no aliasing

Sample at $f_s = (2f_c - B)/2 = 17.5$ MHz



- Second spectral copies just touch the original spectral components
- No aliasing

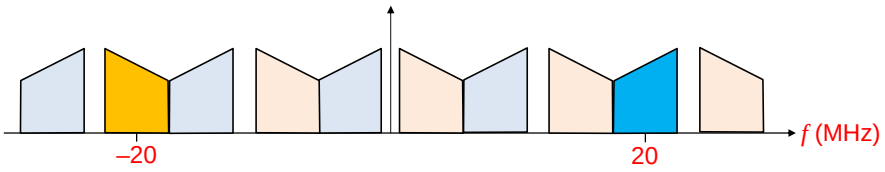
Can reduce up to $f_s \geq (2f_c + B)/3 = 15$ MHz



- Third spectral copies move in and just touch the original spectral components
- No aliasing

Thus, sampling at $(2f_c + B)/3 \leq f_s \leq (2f_c - B)/2$ (in this example, 15 MHz $\leq f_s \leq$ 17.5 MHz) gives no aliasing

Sampling at $(2f_c + B)/4 \leq f_s \leq (2f_c - B)/3$ or 11.25 MHz to 11.67 MHz



In general, for any integer m, sampling at

$$\frac{2f_c + B}{m+1} \leq f_s \leq \frac{2f_c - B}{m}$$

such that $f_s \geq 2B$, gives no aliasing

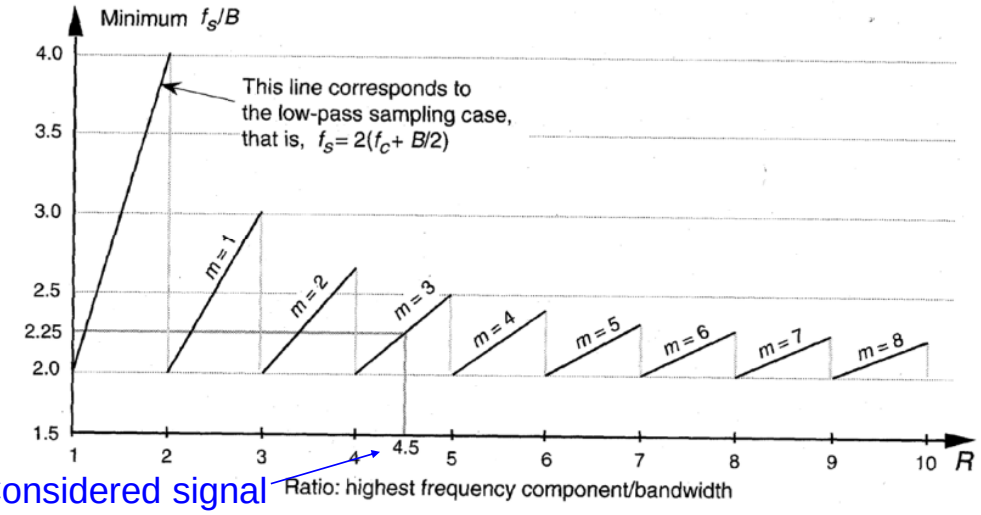
For the considered signal,

- m = 0 gives 45 MHz ≤ f_s ≤ ∞, this is low-pass sampling
- m = 1 gives 22.5 MHz ≤ f_s ≤ 35 MHz
- m = 2 gives 15 MHz ≤ f_s ≤ 17.5 MHz
- m = 3 gives 11.25 MHz ≤ f_s ≤ 11.67 MHz
- m = 4 and above does not satisfy $f_s \geq 2B$

These show all possible sampling frequencies.
The minimum possible f_s is 11.25 MHz.

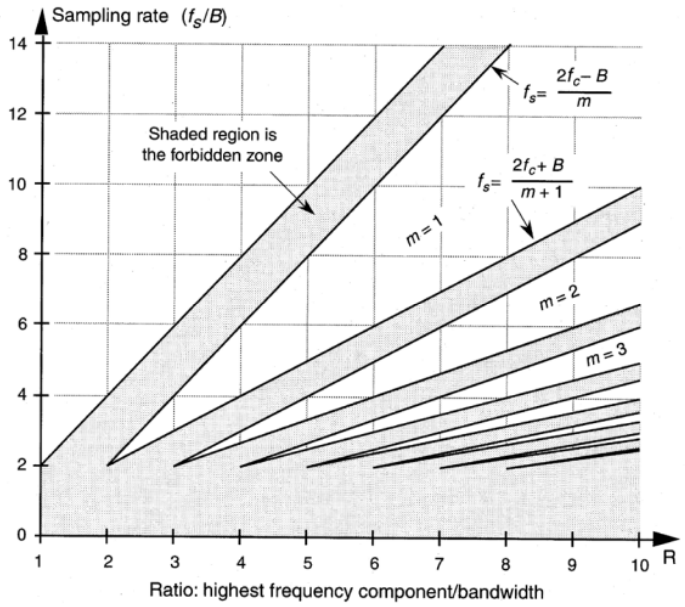
Normalized minimum sampling frequency = $\frac{2f_c + B}{B(m+1)}$ vs

$$R = \frac{\text{highest signal frequency component}}{\text{bandwidth}} = \frac{f_c + (B/2)}{B}$$

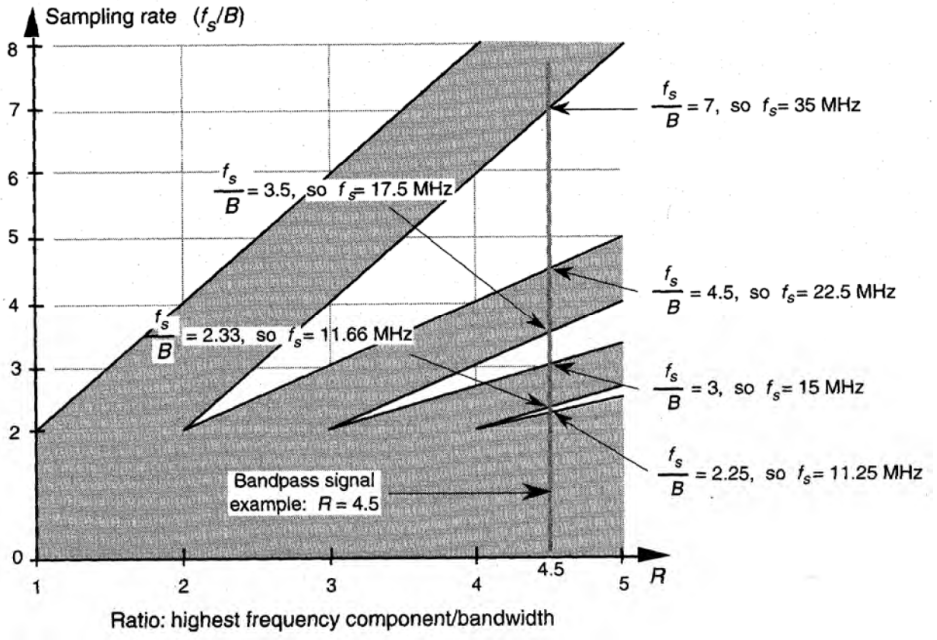


Considered signal

Regions of possible band-pass sampling rates

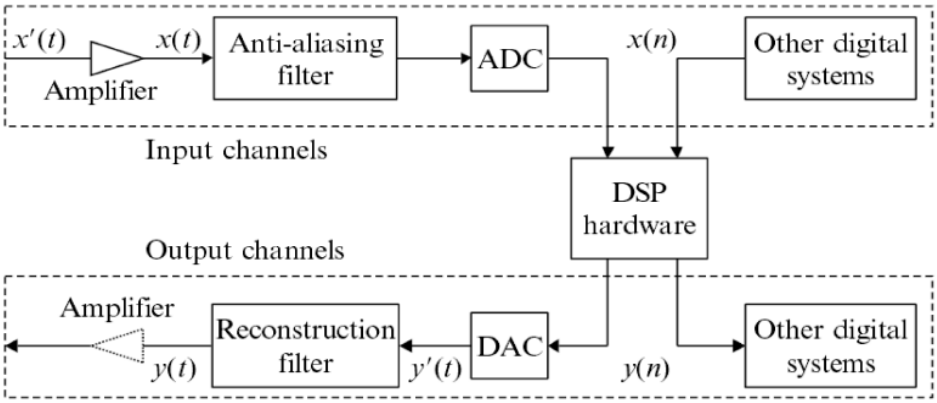


Possible sampling frequencies of the considered signal



Peripherals

Input and output channels of a DSP system for generic DSP application



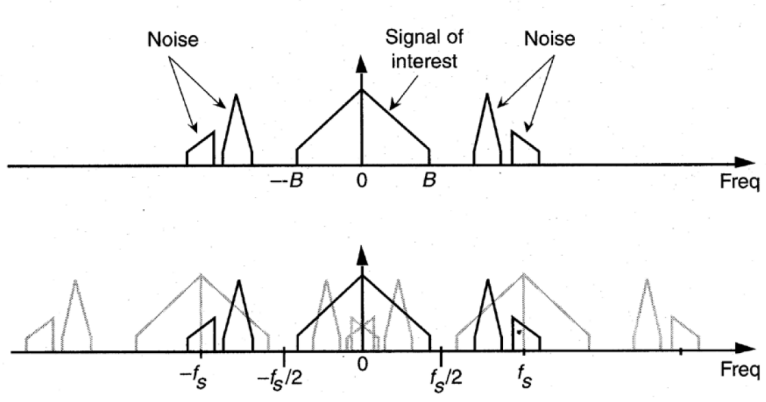
Amplifier

- The gain is determined to match the input dynamic range to the ADC dynamic range
- For unknown/time-varying input range, use an automatic gain controller (AGC) with time-varying gain

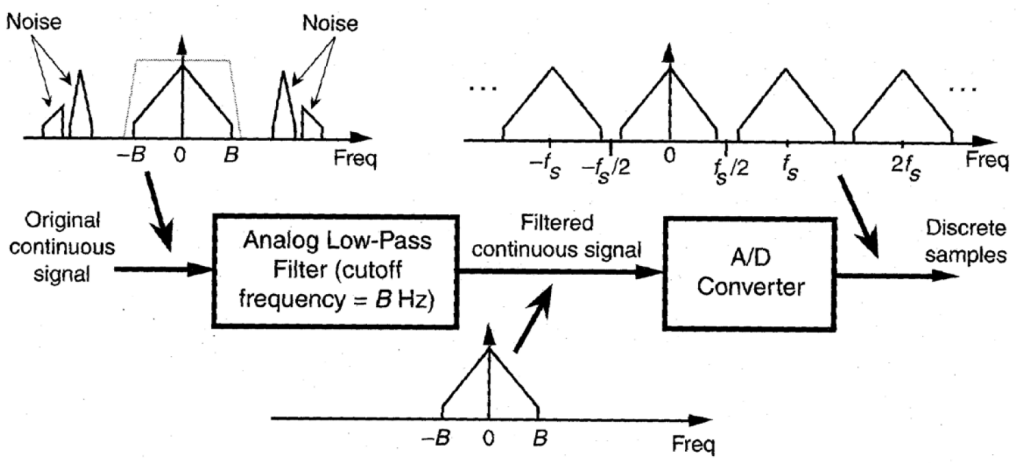
Anti-aliasing filter

- Analog low-pass filter with cut-off frequency B
- Aliasing occurs if analog signal is not bandlimited to $f_s/2$
- Anti-aliasing filter limits the bandwidth of the analog signal

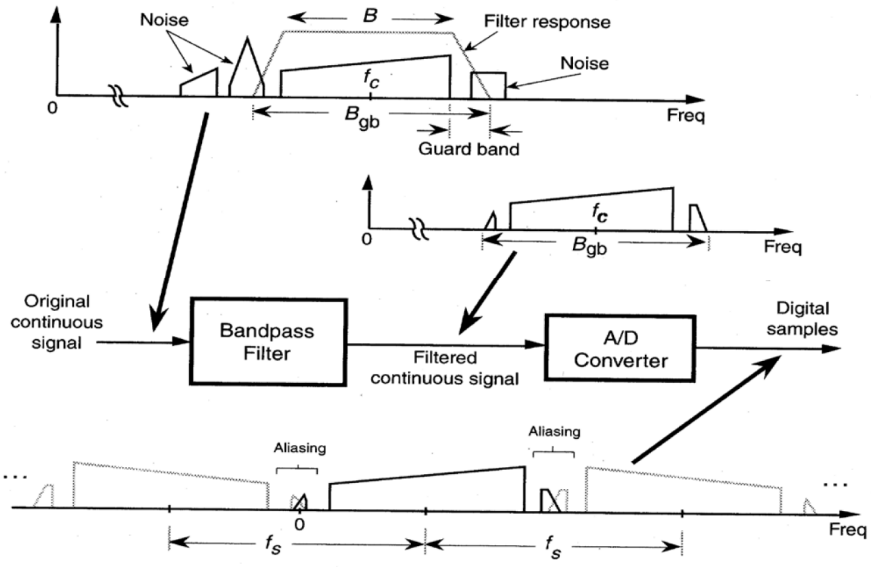
- Anti-aliasing filter is used even for bandlimited signals
- Since noise is not bandlimited, aliased noise energy corrupts the signal spectrum



- Anti-aliasing filter removes any noise outside $-B$ to B

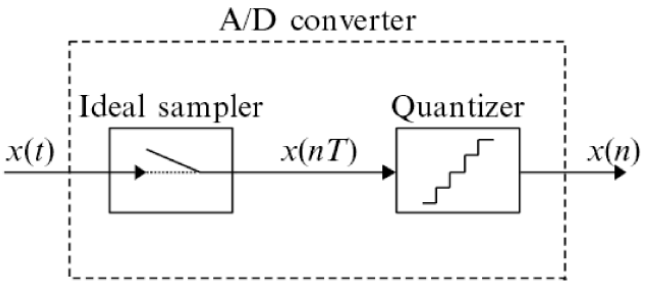


- For band-pass sampling, the anti-aliasing filter is a band-pass filter centered around f_c with width B



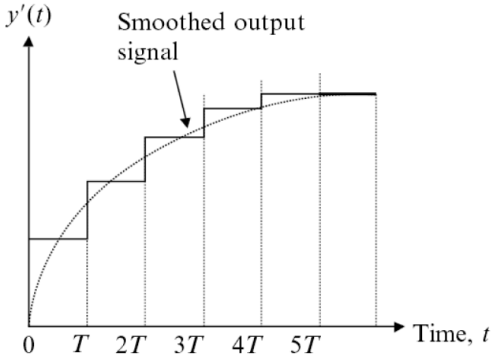
Analog to digital converter (ADC)

- Sampling (convert continuous-time to discrete-time)
- Quantization (convert continuous-value to discrete-value)



Digital to analog converter (DAC)

- Converts the binary sample to the analog voltage
- Holds the analog value until the next sample, thus producing a staircase waveform

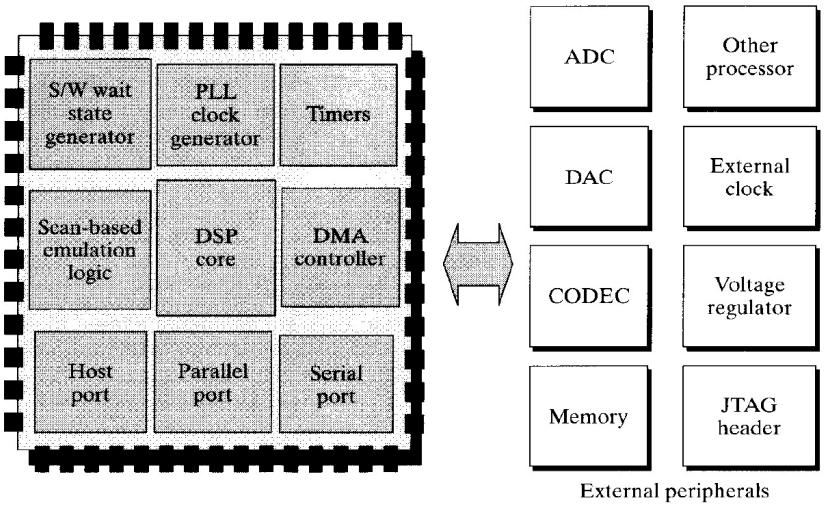


Reconstruction filter

- Also known as anti-imaging filter / smoothing filter
- Smoothens the staircase waveform
- Analog low-pass filter with cut-off frequency B
- Removes spectral copies but retains original spectrum
- In practical ADC, f_s is chosen greater than $2B$ to be able to separate spectral copies by the reconstruction filter

Internal and external peripherals

- Commonly used hardware interfacing blocks



DSP processor on-chip peripherals

- Host port interface: To interface a host processor (such as a microprocessor) with the DSP processor. Both processors exchange information via the DSP processor's on-chip memory, which can be programmed for on-chip data or program memory.

- Direct memory access (DMA) controller: To control data transfer in on/off-chip memory and peripherals.

- Serial port: To transmit/receive data slowly, typically 16 bits. Synchronization pulse or clock signal is deduced from the data itself.

- Parallel port: To transmit/receive data faster, typically 8 or 16 bits. Requires more pins, and handshakes or strobe lines for synchronization. To save the number of pins, a DSP processor uses the main data bus as a parallel port.

- Hardware timer: Programmable, for periodic interrupts for the processor. Can also be used as software controlled signal generators.

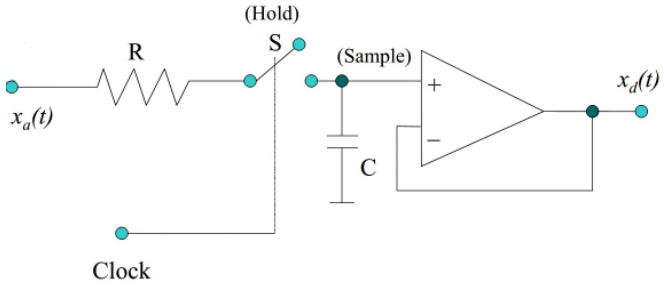
- Clock generator: The DSP processor's master clock either comes from externally supplied clock, or is generated from an external crystal plus the internal oscillator. (If no internal oscillator is included, an external clock circuit must be implemented.)

- Phase-locked loop (PLL): Some DSP processors include a PLL that can generate a higher frequency internal clock from a low frequency external clock. It has the advantages of reducing electromagnetic interference and of using a cheap external clock circuit.

Analog to Digital Converters

Sample and hold circuit

- Output $x_d(t)$ can change only at discrete times
- At these times, $x_d(t)$ is equal to the instantaneous value of $x_a(t)$
- Operarion of the switch is controlled by the clock

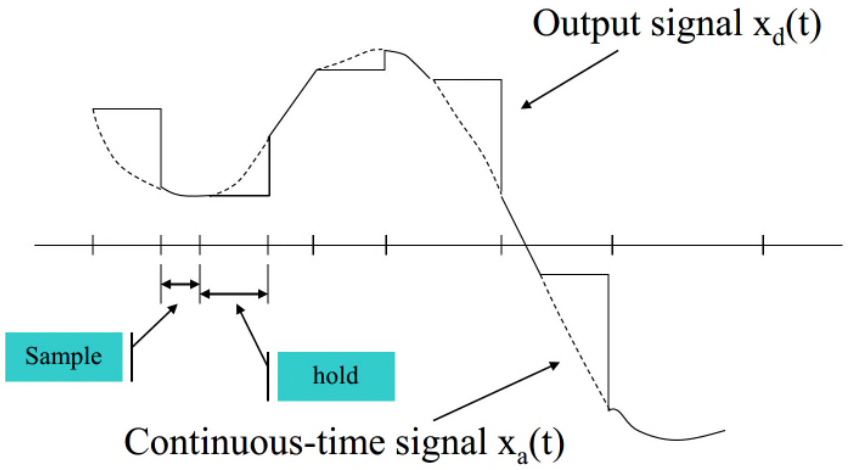


Sampling phase:

- Switch S is closed, capacitor C charges equal to $x_a(t)$

Hold phase:

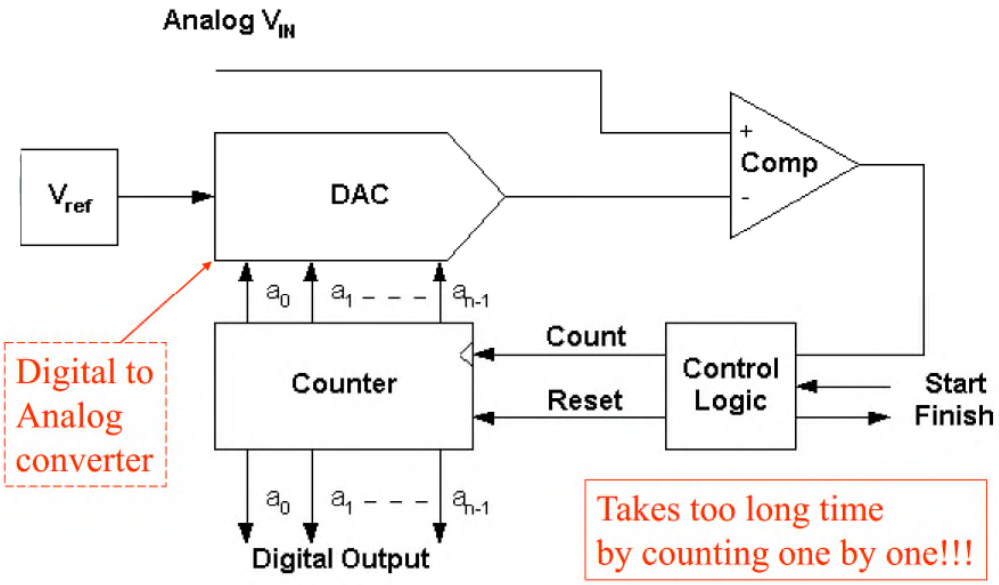
- Switch S is open, charged capacitor C holds the voltage through the unity gain opamp



Types of ADC:

- Counter ADC
- Successive approximation ADC
- Dual slope integrating ADC
- Parallel (flash) ADC
- Sigma-delta ADC

Counter ADC

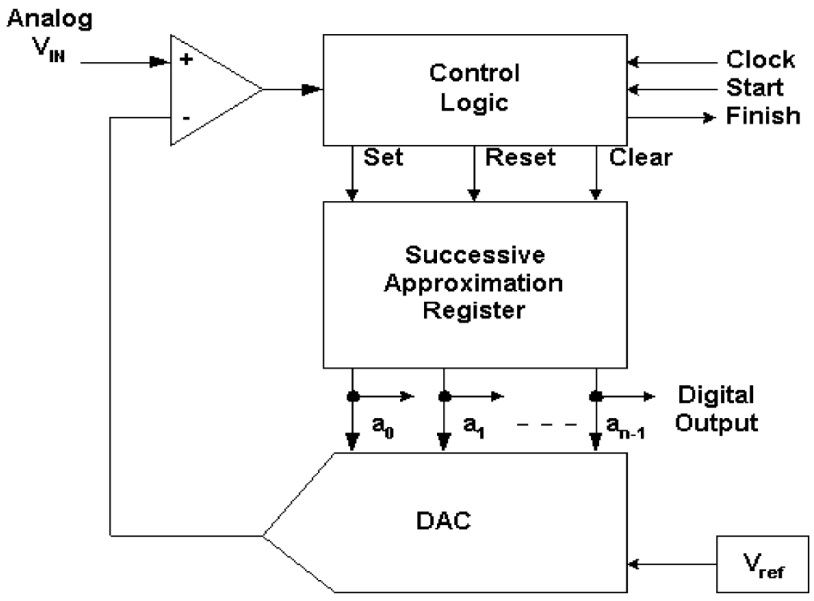


Example: V_{ref} = 1 v (input range 0 to 1 v), V_{IN} = 0.51 v, 4-bit ADC

a_3 a_2 a_1 a_0	V_{IN}	DAC output	Comp
0000	0.51 v	0.0625 v	+ (count)
0001	0.51 v	0.125 v	+ (count)
0010	0.51 v	0.1875 v	+ (count)
0011	0.51 v	0.25 v	+ (count)
0100	0.51 v	0.3125 v	+ (count)
0101	0.51 v	0.375 v	+ (count)
0110	0.51 v	0.4375 v	+ (count)
0111	0.51 v	0.5 v	+ (count)
1000	0.51 v	0.5625 v	– (finish)

Output = 1000

Successive approximation ADC



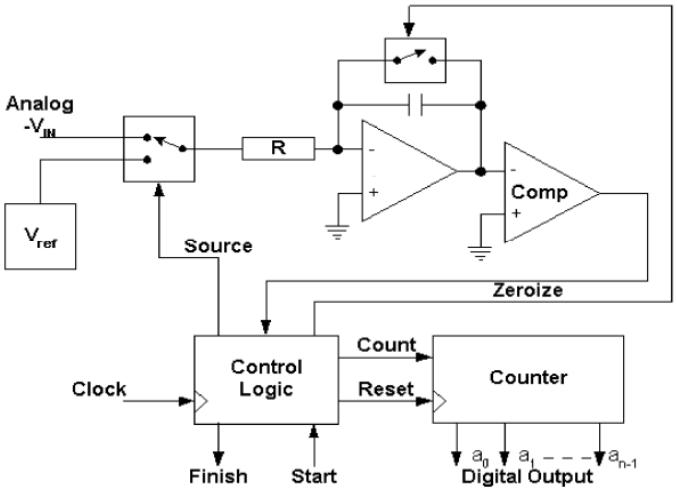
Example: V_{ref} = 1 v (input range 0 to 1 v), V_{IN} = 0.51 v, 4-bit ADC

a_3 a_2 a_1 a_0	V_{IN}	DAC output	Comp
1000	0.51 v	0.5 v	+ (a_3 = 1)
1100	0.51 v	0.75 v	– (a_2 = 0)
1010	0.51 v	0.625 v	– (a_1 = 0)
1001	0.51 v	0.5625 v	– (a_0 = 0)

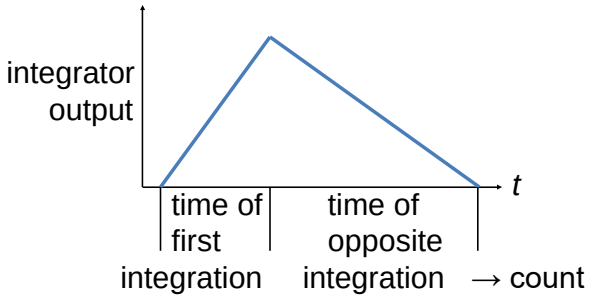
Output = 1000

- Successive approximation ADC takes only n cycles
- Proceeds from MSB to LSB
- Popular because it is fast (2μsec for 12-bit ADC), accurate (high-resolution), low cost

Dual slope integrating ADC



- Reset integrator to 0
- Connect switch to V_{IN} , integrate for a fixed time
- Connect switch to V_{ref} (opposite polarity), integrate in opposite direction until 0
- Time taken is noted by the counter, the count is proportional to V_{IN}



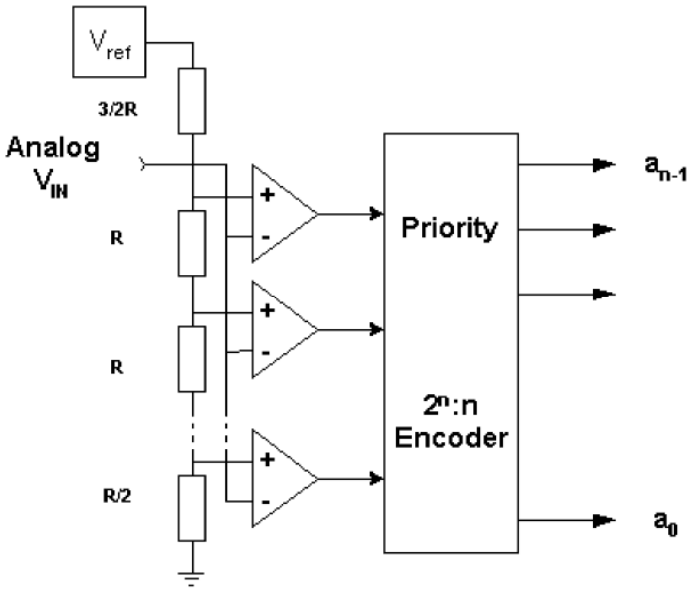
- Precise (high resolution), insensitive to temperature and aging, slow

Parallel (flash) ADC

• Voltage divider sets reference voltages

• Very fast: parallel operations

• More area due to $2^n - 1$ comparators, so low resolution



For symmetry of levels, sometimes V_{max} and V_{min} in mid-tread quantizer are chosen asymmetric similar to 2's complement numbers

Example: for –1 to 1v range, 4-bit ADC, V_{max} = 0.882 v, V_{min} = –1 v,

$$Q = \frac{V_{max} - V_{min}}{2^4} = 0.118v$$

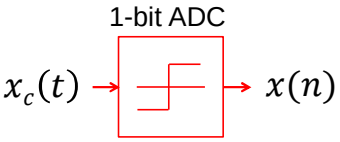
V_{ref} = 0.941v, ground = –0.941v, V_{IN} = 0.51 v

Output = 0100

V_{IN}	voltage divider	comp	priority encoder
0.51 v	0.765 v	+	
0.51 v	0.647 v	+	
0.51 v	0.529 v	+	
0.51 v	0.412 v	–	
0.51 v	0.294 v	–	
0.51 v	0.176 v	–	
0.51 v	0.059 v	–	
0.51 v	–0.059 v	–	$(0100)_{2's}$ = 4
0.51 v	–0.176 v	–	
0.51 v	–0.294 v	–	
0.51 v	–0.412 v	–	
0.51 v	–0.529 v	–	
0.51 v	–0.647 v	–	
0.51 v	–0.765 v	–	
0.51 v	–0.882 v	–	

Sigma-delta ADC

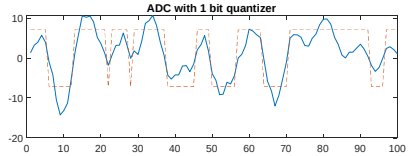
1) 1-bit quantizer

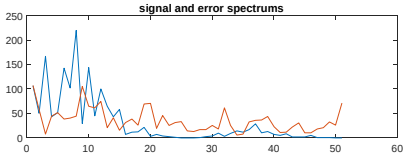


Reconstruction from $x(n)$ to $x_c(t)$ by lowpass filtering

- $SQNR = 6.02n + 4.77 + 20\log(\sigma_x/V_{max})$ for n = 1

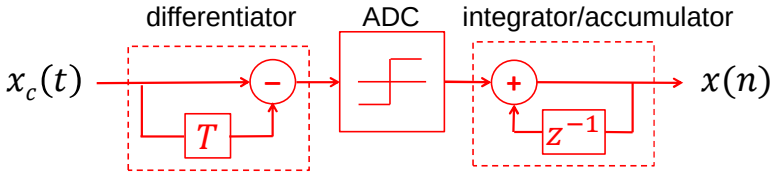
Matlab demo:
- large error
- flat spectrum



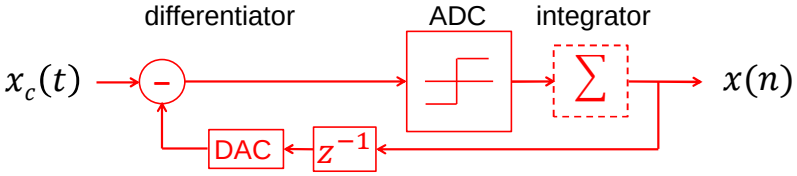


2) Strategies to reduce quantization error

a) Differentiation of the signal



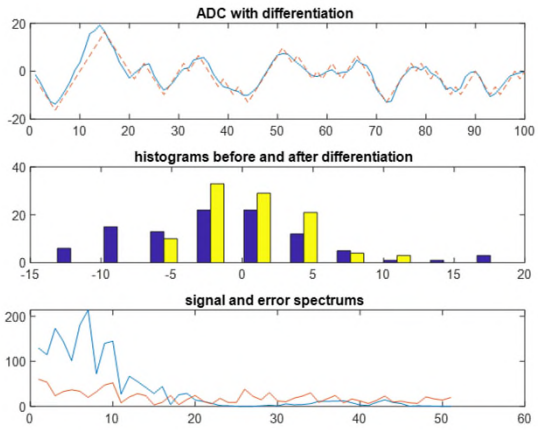
To avoid error propagation, use reconstructed signal



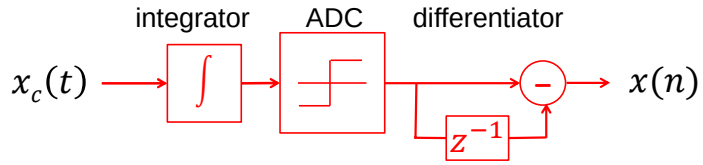
Also known as delta modulator, differential quantization

Matlab demo:

- Positive: smaller error, smaller range
- Negative: lowpass error spectrum (integrator = lowpass)

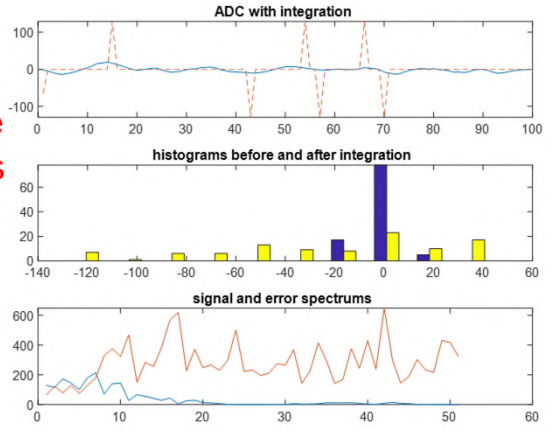


Integration of the signal



Matlab demo:

- Negative: larger error, larger range
- Positive: highpass error spectrum (differentiator = highpass)



Analysis of differentiation of the signal:

- Auto-correlation of $x_c(t)$ is $R(\tau) = E\{x_c(t)x_c(t-\tau)\}$
- Variance of $x_c(t)$ is $R(0) = \sigma_x^2$, variance of error is σ_1^2
- SQNR$_1$ $= 10\log(\sigma_x^2/\sigma_1^2) = 10\log(3\sigma_x^2 2^{2n}/V_{max_x}^2)$

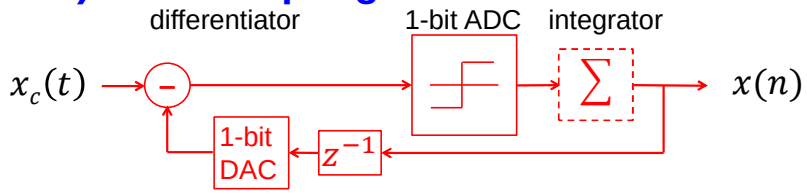
- Differentiation $d(t) = x_c(t) - x_c(t-T)$
- Variance of $d(t)$ is $\sigma_d^2 = E\{[x_c(t)-x_c(t-T)]^2\} = 2R(0)-2R(T)$
- Variance of error is σ_2^2, SQNR$_2 = 10\log(\sigma_d^2/\sigma_2^2)$

- If $\sigma_x/V_{max_x} = \sigma_d/V_{max_d}$ then SQNR$_1$ = SQNR$_2$, both quantizers perform equal, $\sigma_x/\sigma_1 = \sigma_d/\sigma_2$, or $\sigma_2 = \sigma_1(\sigma_d/\sigma_x)$

- If $R(T) > R(0)/2$ then $\sigma_d < \sigma_x$, the variance decreases = range of values decreases = better to quantize $d(t)$
- SQNR $10\log(\sigma_x^2/\sigma^2)$ increases by $10\log(\sigma_x^2/\sigma_d^2)$ dB

2) Strategies to reduce quantization error

b) Oversampling

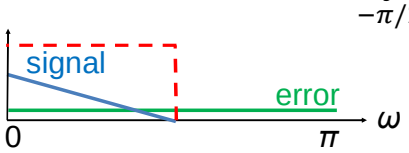


Analysis:

- Power spectral density of $x(n)$ is $S(\omega)$
- Variance $\sigma_x^2 = \int_{-\pi}^{\pi} S(\omega) \frac{d\omega}{2\pi}$
- Similarly error variance σ_e^2

Oversample by 2

- $S_2(\omega)$ has half width, double height
- Same variance $\sigma_x^2 = \int_{-\pi/2}^{\pi/2} S_2(\omega) \frac{d\omega}{2\pi}$



- After lowpass filter, out-of-band error will be removed
- 2 times sampling = σ_e^2 becomes half = 3dB more SQNR (each doubling of sampling gives 3dB more)
- Also, oversampling = lower T = higher $R(T)$ = lower σ_d^2

Oversample by 3

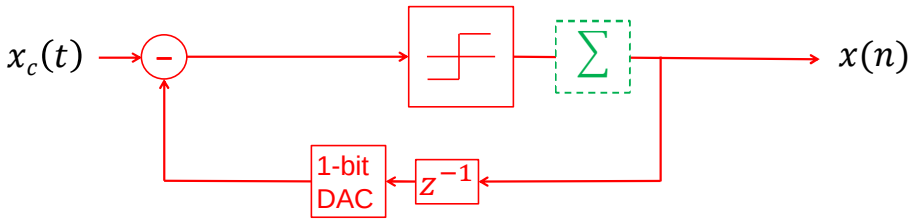
- $S_3(\omega)$ has one-third width, triple height
- Same variance $\sigma_x^2 = \int_{-\pi/3}^{\pi/3} S_3(\omega) \frac{d\omega}{2\pi}$

Matlab demo:

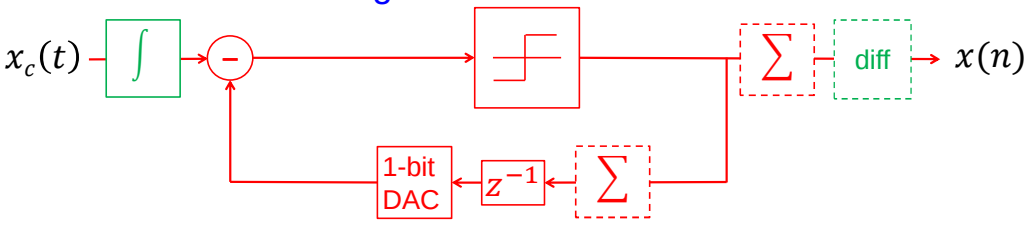
- Smaller error

2) Strategies to reduce quantization error:
c) Integration / noise shaping

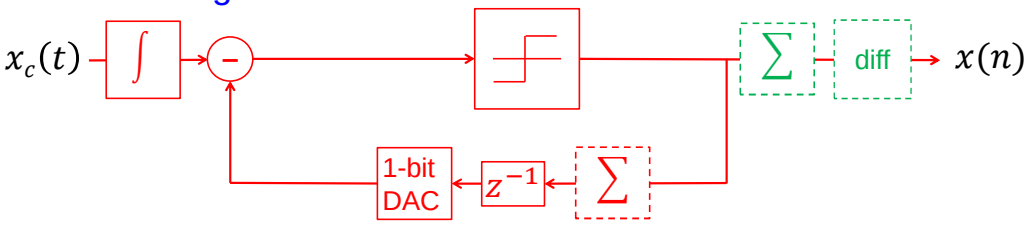
- Move integrator



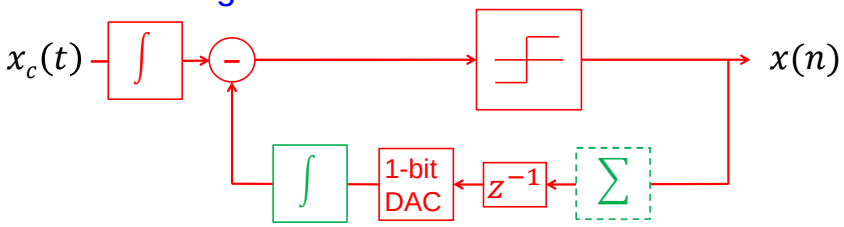
- Introduce integrator and differentiator



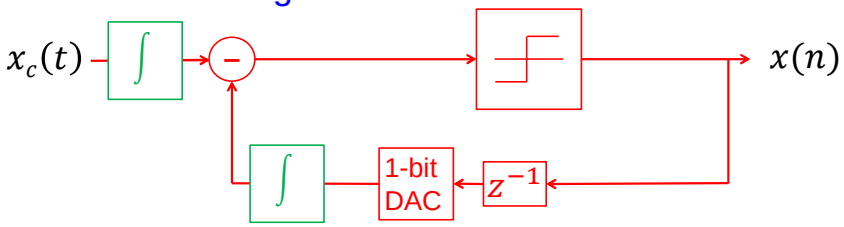
- Integrator cancels differentiator



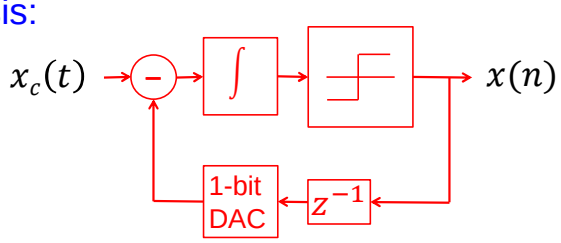
- Move integrator before DAC



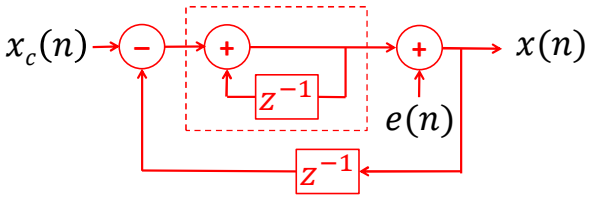
- Combine integrators



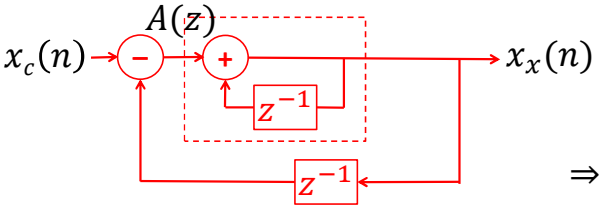
Analysis:



- Model all signals as digital signals
- Model 1-bit ADC as adding noise



Consider only the signal:



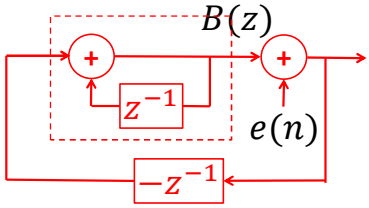
$$A(z) = X_c(z) - z^{-1}X_x(z)$$

$$X_x(z) = \frac{A(z)}{1 - z^{-1}}$$

$$\Rightarrow X_x(z) = X_c(z)$$

- No change in signal (differentiator cancels integrator)

Consider only the error:



$$B(z) = \frac{-z^{-1}X_e(z)}{1 - z^{-1}}$$

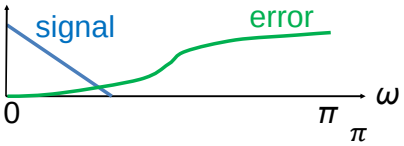
$$X_e(z) = B(z) + E(z)$$

$$\Rightarrow X_e(z) = (1 - z^{-1})E(z)$$

- Highpass error (differentiator)

Error is reduced because

- Highpass error $X_e(z) = (1 - z^{-1})E(z)$
- Magnitude of frequency response $\left|H\left(e^{j\omega}\right)\right| = 2\sin\frac{\omega}{2}$
- $e(n)$ power spectral density = N



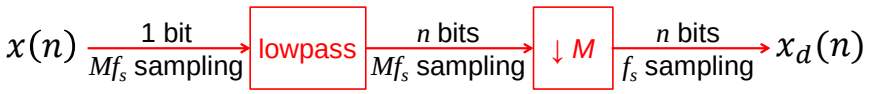
- $x_e(n)$ variance $\sigma_{x_e}^2 = \int_{-\pi}^{\pi} \left(2\sin\frac{\omega}{2}\right)^2 N\frac{d\omega}{2\pi} = 2N$
- $x_e(n)$ variance after lowpass filter

$$\sigma_{x_e}^2 = \int_{-B}^{B} \left(2\sin\frac{\omega}{2}\right)^2 N\frac{d\omega}{2\pi} \approx \int_{-B}^{B} \omega^2 N\frac{d\omega}{2\pi} = \frac{B^3 N}{3\pi}$$

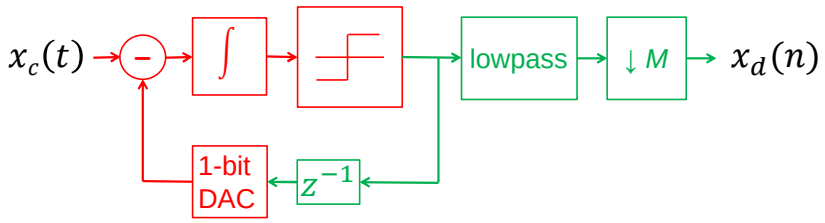
- 2 times sampling = B becomes half = $\sigma_{x_e}^2$ becomes 1/8 = 9dB more SQNR

3) Decimator

Converts from 1-bit to n-bits



4) Block diagram of sigma-delta ADC



first-order sigma-delta ADC (red = analog, green = digital)

5) SQNR

Since any quantizer SQNR = 6.02n + .., adding 1 bit resolution = 6dB more SQNR.

For oversampling:

- Each doubling of sampling = 3dB more SQNR = adding ½ bit

For oversampling and first order noise shaping:

- Each doubling of sampling = 9dB more SQNR = adding 3/2 bits

For oversampling and second order noise shaping:

- Each doubling of sampling = 15dB more SQNR = adding 5/2 bits

Example:
What should be the internal sampling frequency of a 16-bit first order sigma-delta ADC operating at an effective sampling frequency of 8KHz? How would it change if instead a second order loop is used?

For 16-bit ADC using 1-bit quantizer, add 15 bits
we need 15/1.5 = 10 doubling of sampling
Internal sampling = 2^{10} x 8KHz = 8.192MHz

For second order, 15/2.5 = 6 doubling of sampling
Internal sampling = 2^{6} x 8KHz = 512KHz

Real sigma-delta ADC Specifications:

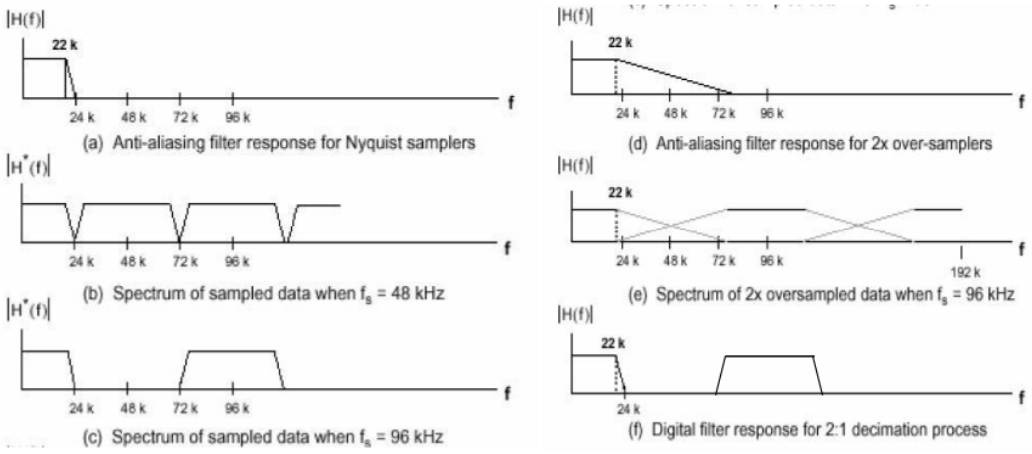
- Multiple effective sampling frequencies, such as 16-bit 6.4MHz/100KHz and 12-bit 6.4MHz/400KHz, second order
- Higher oversampling factor than required, such as for voice, 16-bit, 2.048MHz/8KHz, second-order
- Oversampling factor not 2^n, such as for mp3/CD audio, 12-bit, 3.1752MHz/44.1KHz

6) Advantages and disadvantages

Advantages of sigma-delta ADC

- Low cost because 90% of silicon die area is digital processing
- No need of high-precision analog component
- High resolution, good noise characteristics
- Anti-aliasing filter requirement is relaxed

Sigma-delta ADC: anti-aliasing filter requirement



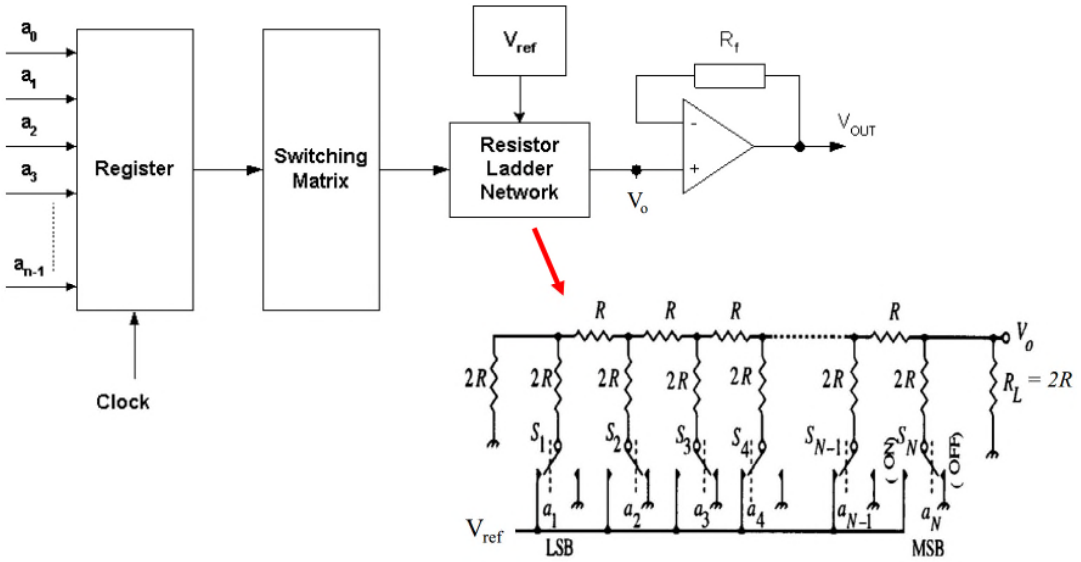
Single-stage RC lowpass filter is sufficient

Disadvantages of sigma-delta ADC

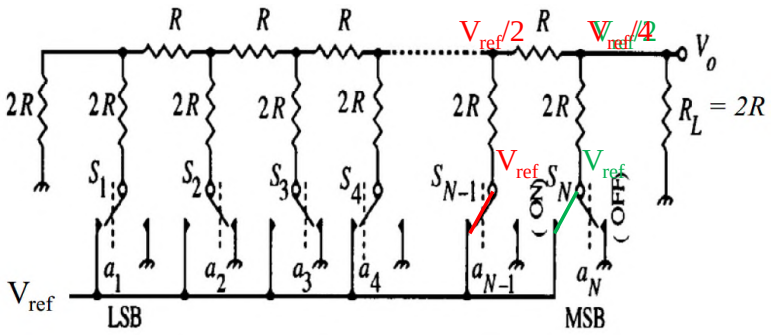
- Low speed (due to oversampling), used for voice and audio
- Slowly varying inputs cause low frequency artifacts

Digital to Analog Converters

Ladder DAC

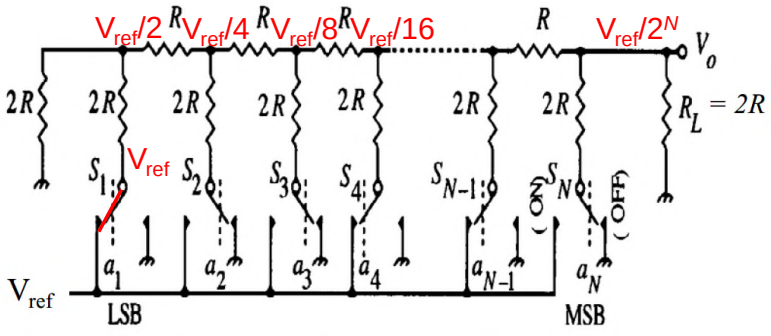


Operation of DAC



- a_N sees resistances $2R + 2R$, so V_{ref} is halved

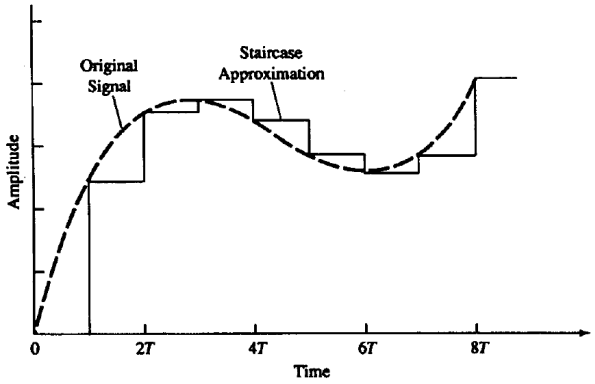
- a_{N-1} sees resistances $2R + 2R$, so V_{ref} is halved
- $V_{ref}/2$ sees resistances $R + R$, so it is halved



- a_1 sees resistances $2R + 2R$, so V_{ref} is halved
- $V_{ref}/2$ sees resistances $R + R$, so it is halved
- $V_{ref}/4$ sees resistances $R + R$, so it is halved
  :

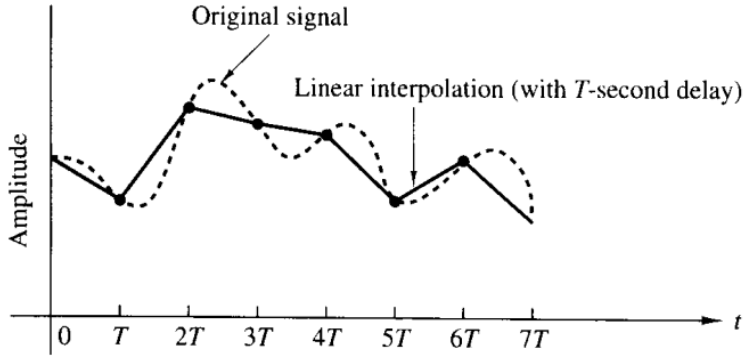
- Input = $a_N \ldots a_2\, a_1$, N-bit fraction from 0 to 1
- V_{ref} = reference (maximum) voltage
- Due to linearity, voltages at the output add up
- Output voltage $V_o = V_{ref}\left(a_N 2^{-1} + a_{N-1}2^{-2} + \cdots + a_1 2^{-N}\right)$ in the range $0 \le V_o < V_{ref}$
- The resistor ladder network requires high precision component matching
- Unity gain opamp acts as a buffer

Zero-order hold



- Zero-order hold or zero-order interpolation = holds the value until the next sample
- Constant = order zero
- Staircase output waveform

First-order (linear) interpolation

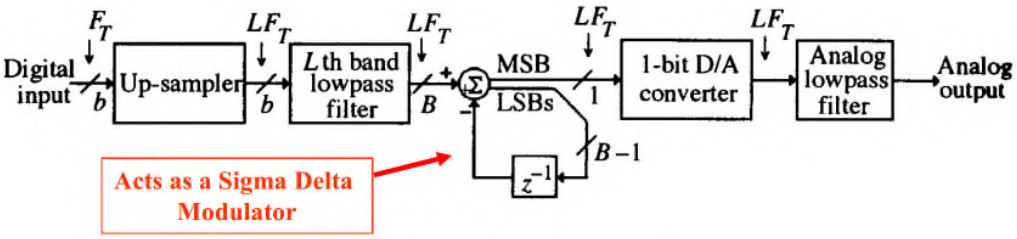


- First-order interpolation = connects successive samples with straight-line segments
- Straight line = order one

Reconstruction filter

- Suboptimal interpolation = frequencies above cut-off
- Remove using a lowpass analog filter at the output (same specification as the anti-aliasing filter)
- Switched-capacitor analog filter
  - Programmable cut-off
  - Physically compact

Sigma-delta based DAC



- Input upsampled to higher sampling frequency
- Quantized by 1-bit quantizer
- B–1 LSBs = quantization error
- Loop subtracts quantization error from next sample
- Downsampling not necessary since analog output
- Advantages: low-cost digital processing, simple analog filter

Example of ADC Chips

Analog Devices AD9280
- 8 bit, 32 MSPS
- Multistage differential pipeline architecture
- Monolithic, single-supply, on-chip sample-and-hold amplifier and voltage reference

Analog Devices AD1877
- 16 bit, for audio
- Sigma-delta, fourth-order
- Stereo (2 channel), output time-multiplexed to a single stream
- Single 5V supply, on-chip voltage reference
- Stable over temperature and time

Example of DAC Chips

Analog Devices AD7801

- 8 bit parallel input
- Single 2.7V to 5.5V supply, on-chip precision output buffer
- High-speed registers, parallel microprocessor and DSP compatible interface
- Power consumption 5mW (3μW in power-down mode)

Analog Devices AD1833A

- 24 bit, 192 KHz, 6 channel, for audio
- Sigma-delta
- On-chip programmable (through a serial port) per channel clickless attenuator, mute capability

Real-Time Operations: Interrupts

Interrupt

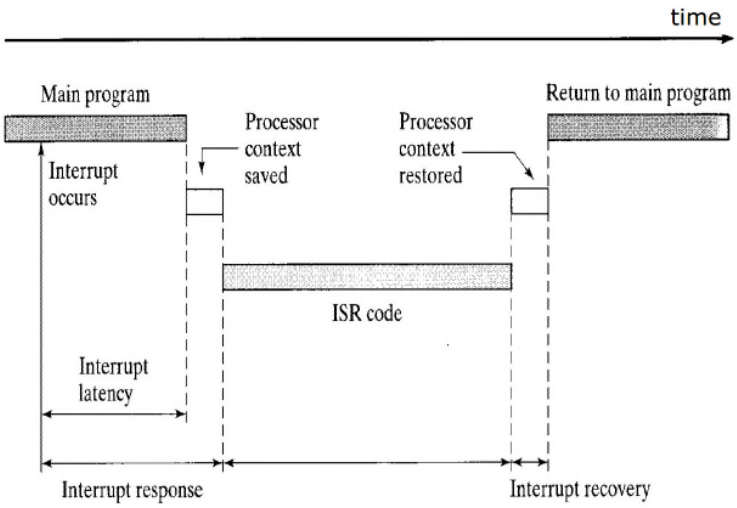
- Causes CPU to temporarily transfer control from its current program to the interrupt handler
- Primary means of communication between most DSP processors and their peripherals
- Caused by events external to the current program, so there is no relationship between the current program and the occurrence of the interrupt
- When a system reacts to an interrupt, it must preserve the status of the program that was interrupted

The sequence of events

- Hardware/software activates the interrupt request control line
- Request should be kept high until the CPU acknowledges the interrupt
- Interrupt indicator is stored into CPU register, checked at the end of every instruction cycle
- CPU identifies the source of the interrupt, perhaps by polling IO devices

- With multiple interrupts, interrupt management prioritizes the interrupts, and the request with the highest priority is selected for servicing
- CPU sends the interrupt acknowledgement
- CPU obtains the memory address of the interrupt handler. (Address can be provided by the interrupting device along with the interrupt request.)
- CPU suspends its current program, saves the program counter and other CPU status information
- CPU loads the program counter with the address of the interrupt handler, and performs the interrupt service routine

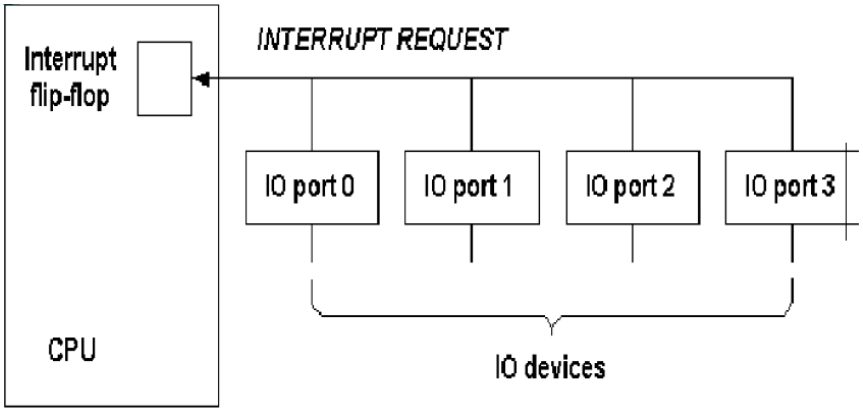
- On the return instruction, CPU transfers back the control to the interrupted program



Sources of interrupt

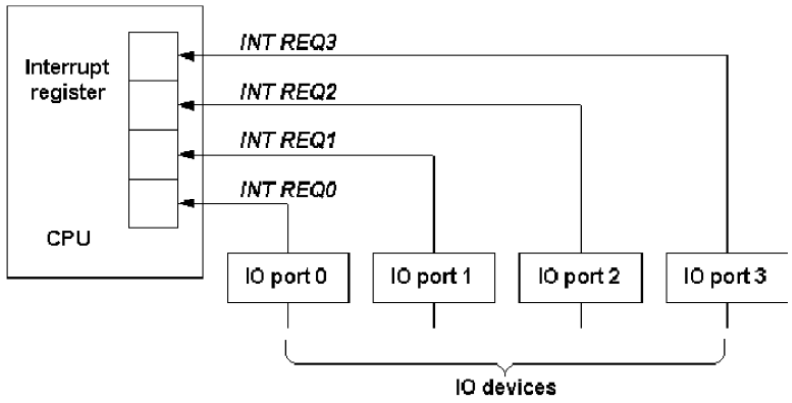
- Internal (on-chip) interrupts
  - Serial port
  - Parallel port
  - Timer
- External interrupt lines
  - One to four external lines for external peripherals
- Software interrupts (exceptions or traps)
  - Division by zero
  - Overflow
  - Underflow

Single-line interrupt system



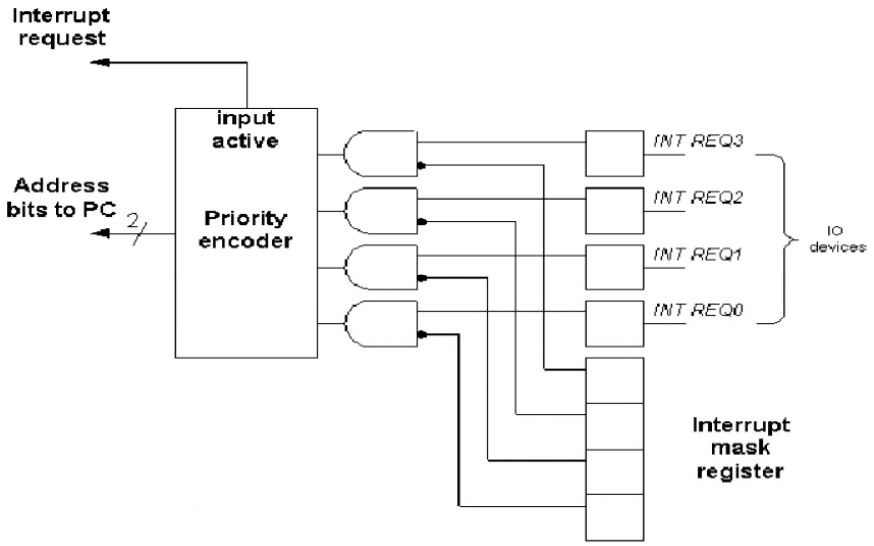
- On interrupt request, CPU must poll each IO device
- Polling can be priority programmed
- Polling is slow

Multiple-line/multilevel interrupt system

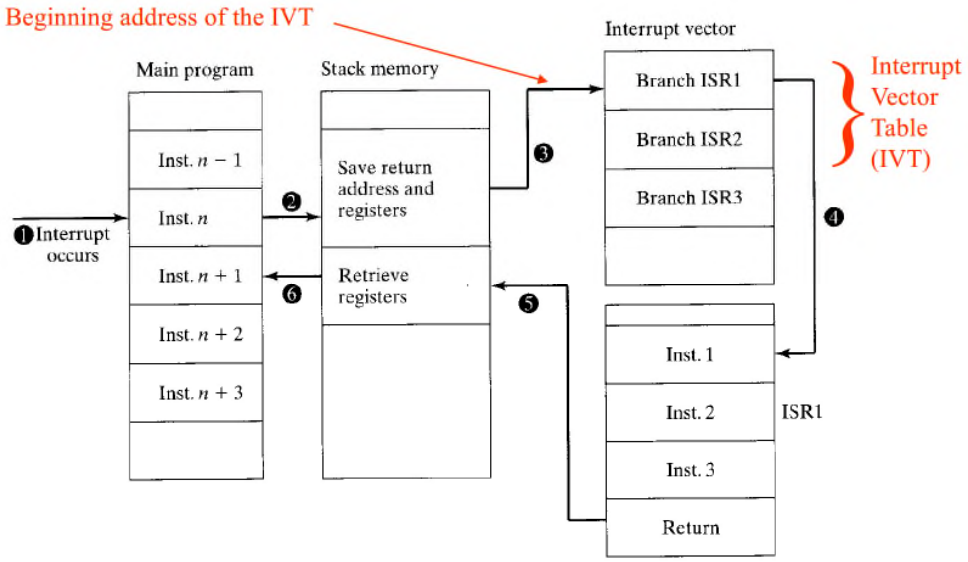


- Polling not necessary, source of interrupt is known
- Interrupt priority can be resolved in software
- Faster, CPU may still have to fetch the address of the interrupt service routine

Vectored interrupt scheme



The sequence of events



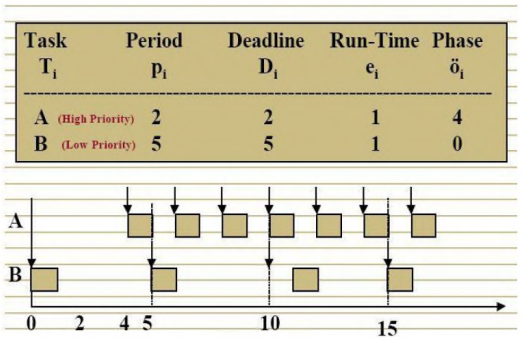
- Interrupt signals are fed into a priority encoder
- Priority encoder sends interrupt request and produces a $\lceil \log_2 k \rceil$ bit address
- Interrupt vector = memory address of interrupt service routine of each device
- Interrupt vector table = table of interrupt vectors. CPU loads the starting address of this table in the program counter.
- Interrupt vector is neither computed by the CPU nor supplied by the IO device. Bits from priority encoder modify the program counter. Thus, an interrupt causes a direct hardwired transition to its service routine.

- Interrupt mask register can enable/disable any or all of the interrupt request lines under program control by setting the corresponding bits to 0/1
- Simplified programming: the programmer does not have to be concerned with which device interrupted, this is implicitly contained in the interrupt vector
- Smaller interrupt response time
- Most flexible response to interrupts

Real-Time Operations: Scheduling

Real-time operation:

- Real-time operation = the maximum time between an interrupt request (initiation) and the completion of the corresponding task (resource) is bounded
- Priority based interrupt management is insufficient for real-time operation
- Real-time operation requires task/process scheduling
- Scheduling considers resource run times, requirements of resources, deadlines, avoidance of deadlocks, etc.

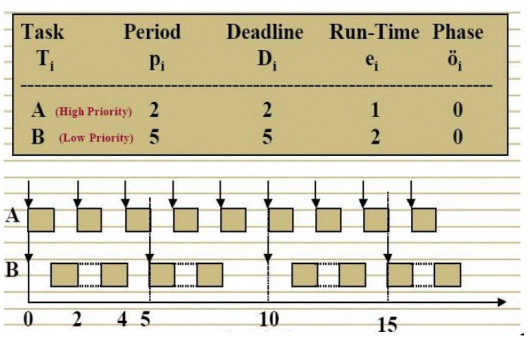


Scheduling

Example:

- A run-time 1, periodic with period 2, starts at 4
- B run-time 1, periodic with period 5, starts at 0

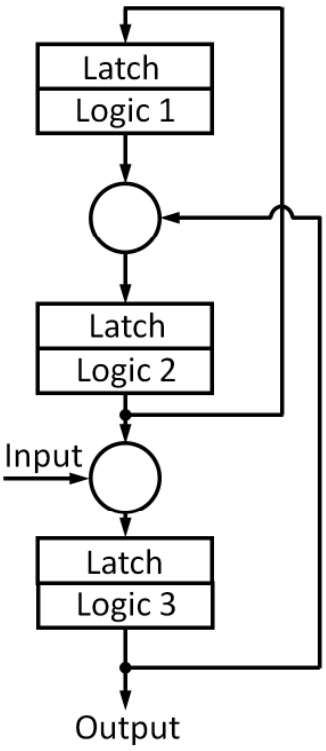
Both deadlines met



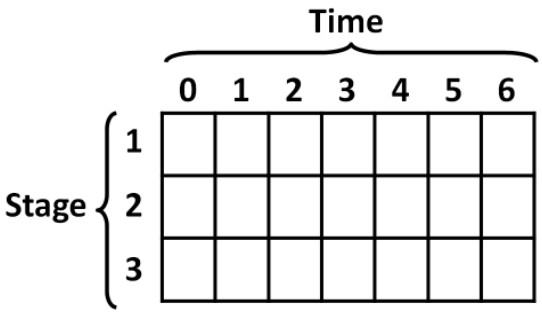
Example:

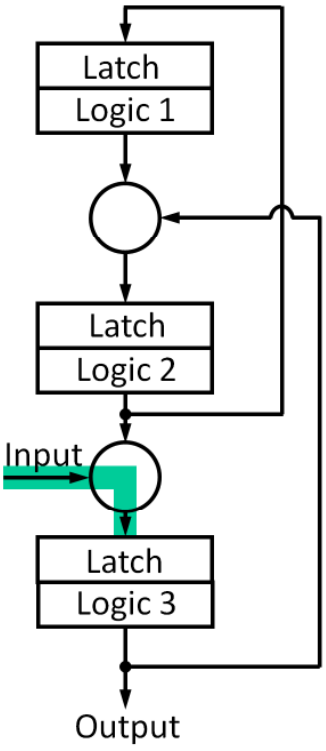
- A run-time 1, periodic with period 2, starts at 0
- B run-time 2, periodic with period 5, starts at 0

Both deadlines met

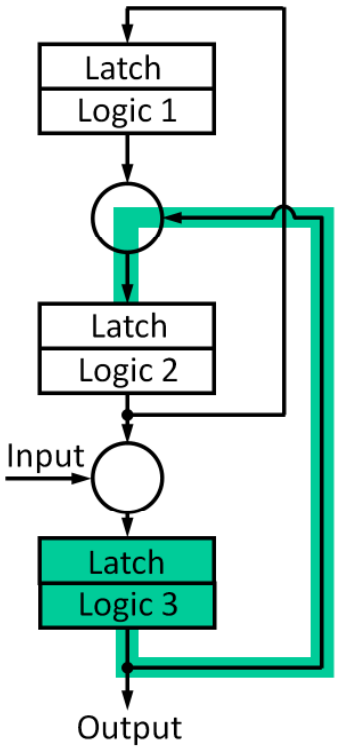


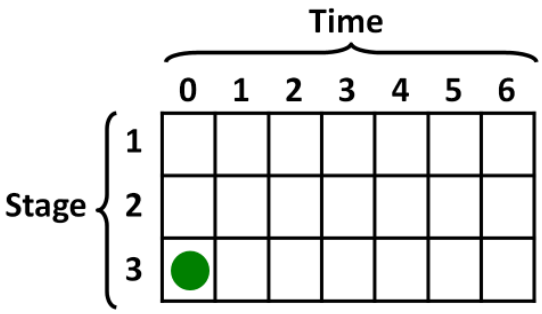
The same resource (stage) may be used multiple times in pipelining.



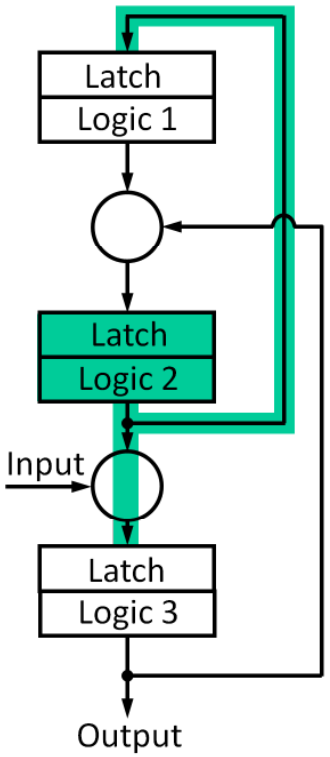


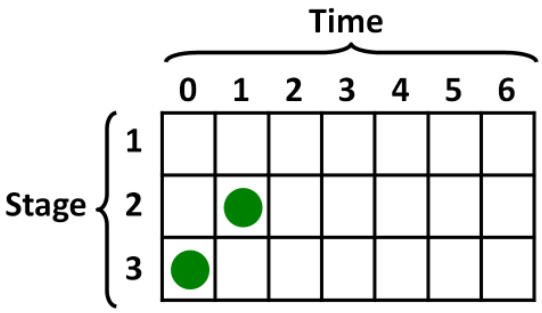
Time 0


Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output

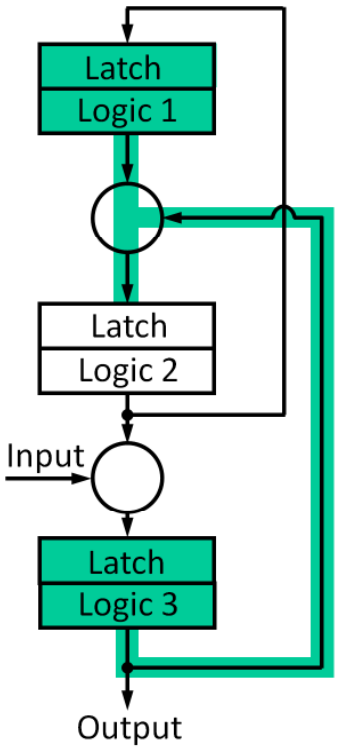

Time
0 1 2 3 4 5 6
Stage
1
2
3

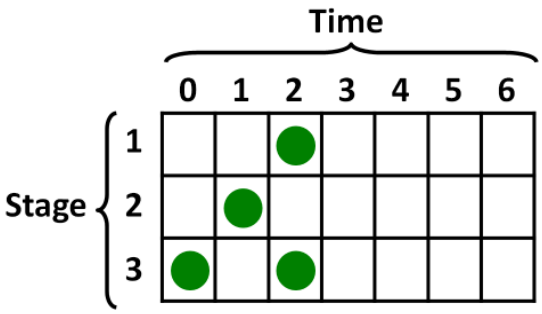
Time 1


Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output

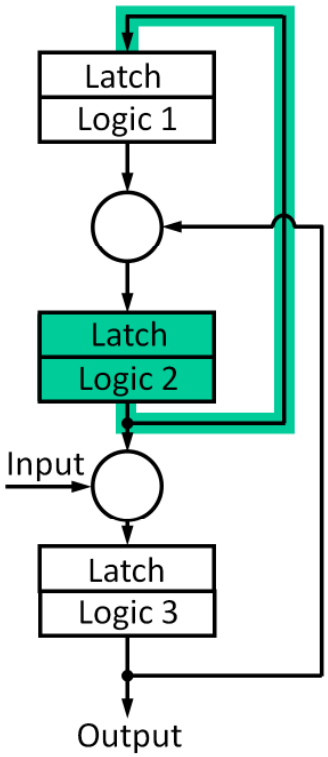

Time
0 1 2 3 4 5 6
Stage
1
2
3

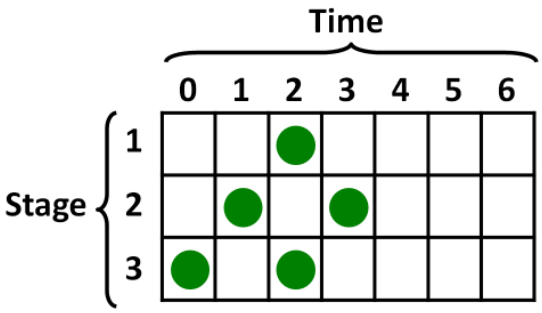
Time 2


Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output

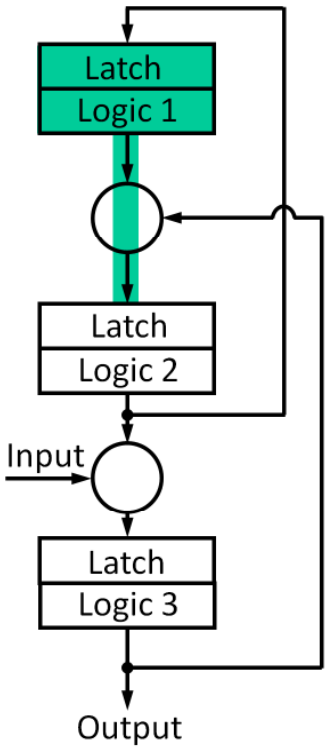

Time
0 1 2 3 4 5 6
Stage
1
2
3

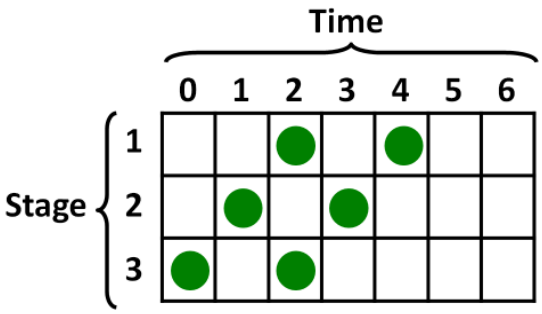
Time 3


Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output

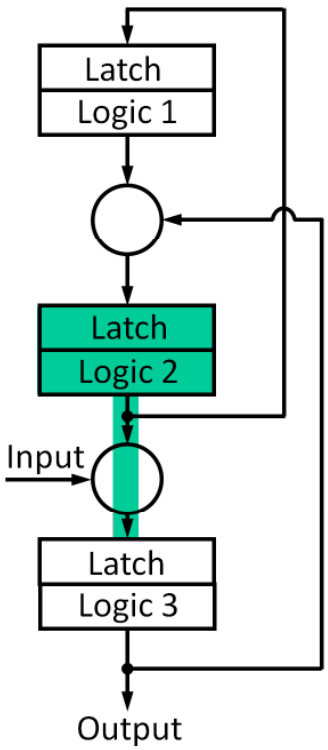

Time
0 1 2 3 4 5 6
Stage
1
2
3

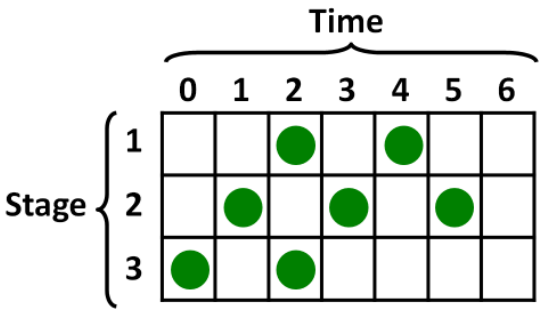
Time 4


Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output

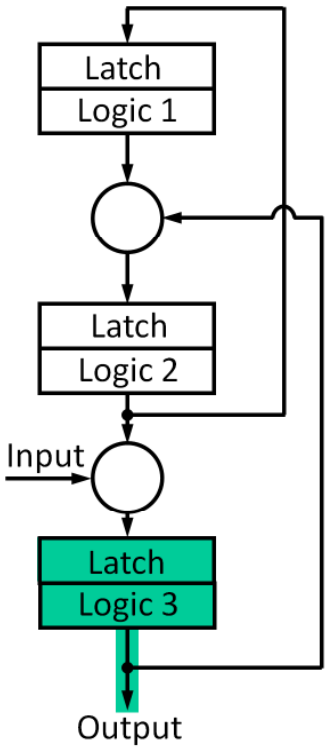

Time
0 1 2 3 4 5 6
Stage
1
2
3

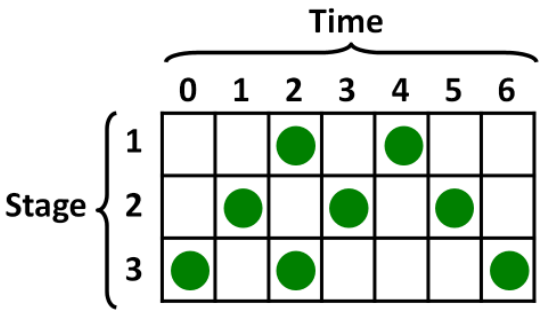
Time 5


Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output

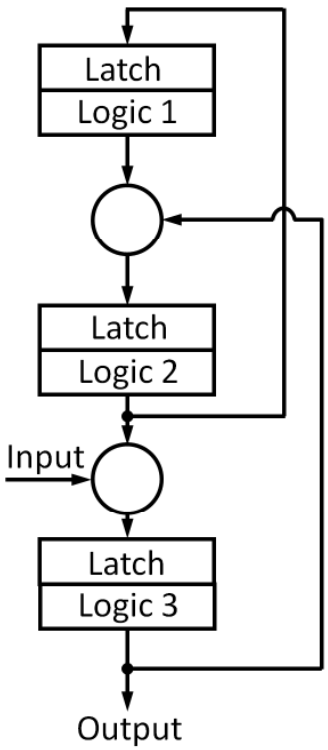

Time
0 1 2 3 4 5 6
Stage
1
2
3

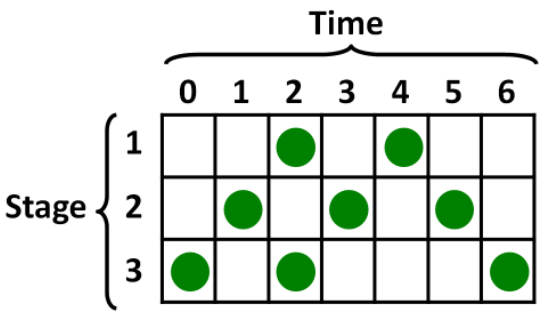
Time 6

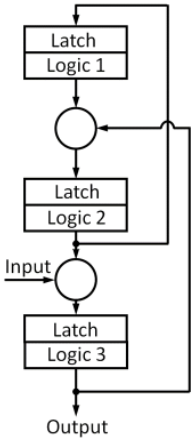

Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output


Time
0 1 2 3 4 5 6
Stage
1
2
3

Time 7


Latch
Logic 1
Latch
Logic 2
Input
Latch
Logic 3
Output


Time
0 1 2 3 4 5 6
Stage
1
2
3



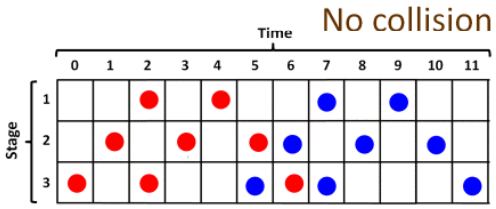
A reservation table

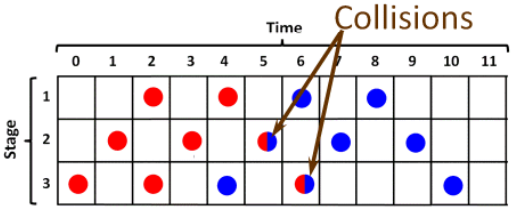
Stage	0	1	2	3	4	5	6
1			A		A		
2		A		A		A	
3	A		A				A

Time

Latency = 7 clocks

Collision: A collision is said to have occurred if two or more initiations attempt to use the same stage at the same time.

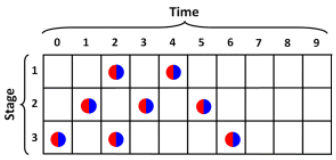




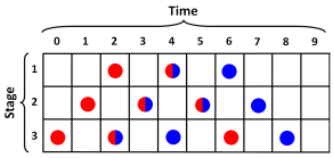
Permissible initiation: An initiation that does not cause collision.

Forbidden initiation: An initiation that causes collision.

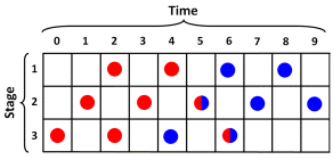
Forbidden set



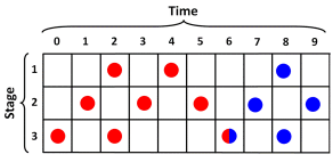
Collision after 0 clock



Collision after 2 clock



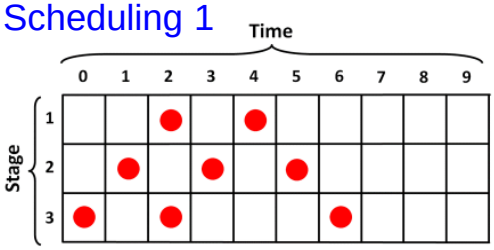
Collision after 4 clock



Collision after 6 clock

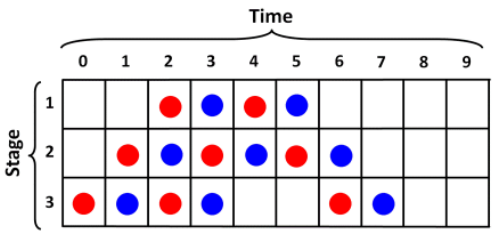
Thus, forbidden set = {0,2,4,6}

Scheduling 1

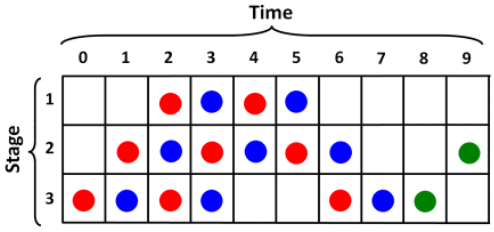


		A		A		
	A		A		A	
A		A				A

Input sample 1



Input sample 2



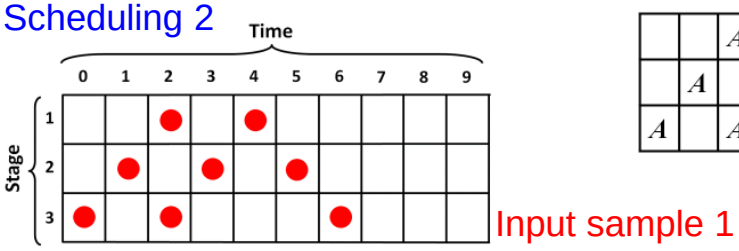
Input sample 3

Scheduling 1

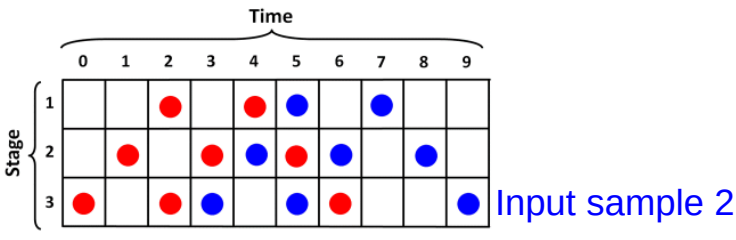
time / stage	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1			A_1	A_2	A_1	A_2					A_3		A_3		
2		A_1	A_2	A_1	A_2	A_1	A_2			A_3		A_3		A_3	
3	A_1	A_2	A_1	A_2			A_1	A_2	A_3		A_3				A_3

Initiation sequence: 1,7,1,7,..
Average initiation = 4 clocks/sample
Average throughput = ¼ input/clock

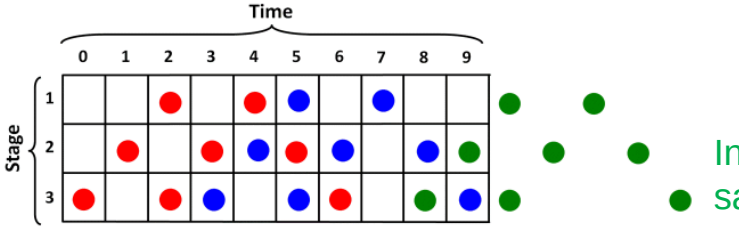
Scheduling 2



Input sample 1



Input sample 2



Input sample 3

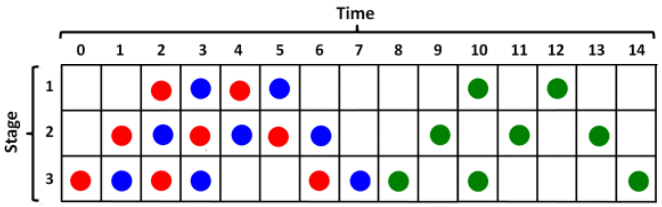
		A		A		
	A		A		A	
A		A				A

Scheduling 2

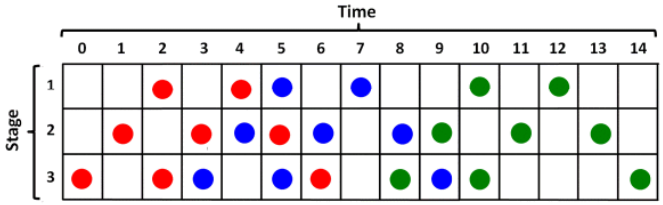
time / stage	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1			A_1		A_1	A_2		A_2			A_3		A_3		
2		A_1		A_1	A_2	A_1	A_2		A_2	A_3		A_3		A_3	
3	A_1		A_1	A_2		A_2	A_1		A_3	A_2	A_3				A_3

Initiation sequence: 3,5,3,5,..
Average throughput = ¼ input/clock

Permissible sequence



1, 7, 1, 7, …



3, 5, 3, 5, …

Stage utilization = the fraction of time a given stage is being utilized

Example: stage 1 utilization = 2/7

Number of utilized stage 1 = 2

		A		A		
	A		A		A	
A		A				A

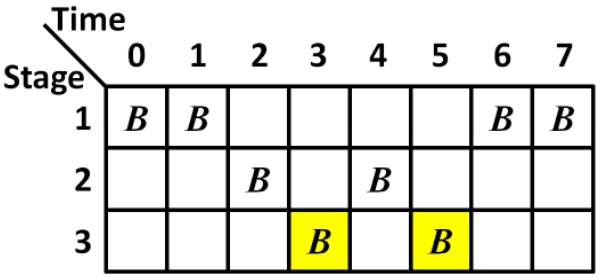
Maximum achievable (average) throughput
≤ the reciprocal of the maximum number of utilized stages in a single row of the reservation table.

Example: maximum achievable throughput ≤ 1/3

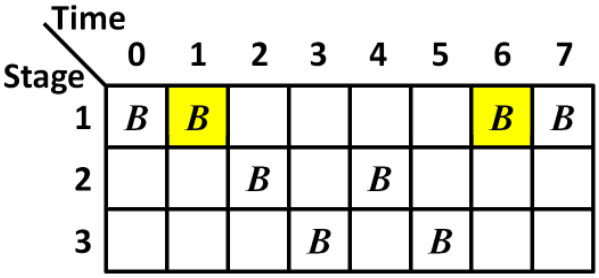
Stage \ Time	0	1	2	3	4	5	6	7
1	B	B					B	B
2			B		B			
3				B		B		

Example B: reservation table

Two yellow boxes in stage 1 show that initiation at time 1 is forbidden.
Two blue boxes show the same.



Two yellow boxes in stage 3 show that initiation at time 2 is forbidden.



Two yellow boxes in stage 1 show that initiation at time 5 is forbidden.

Forbidden set = {0,1,2,5,6,7}
Stage 1 utilization = ½
Maximum achievable throughput ≤ ¼