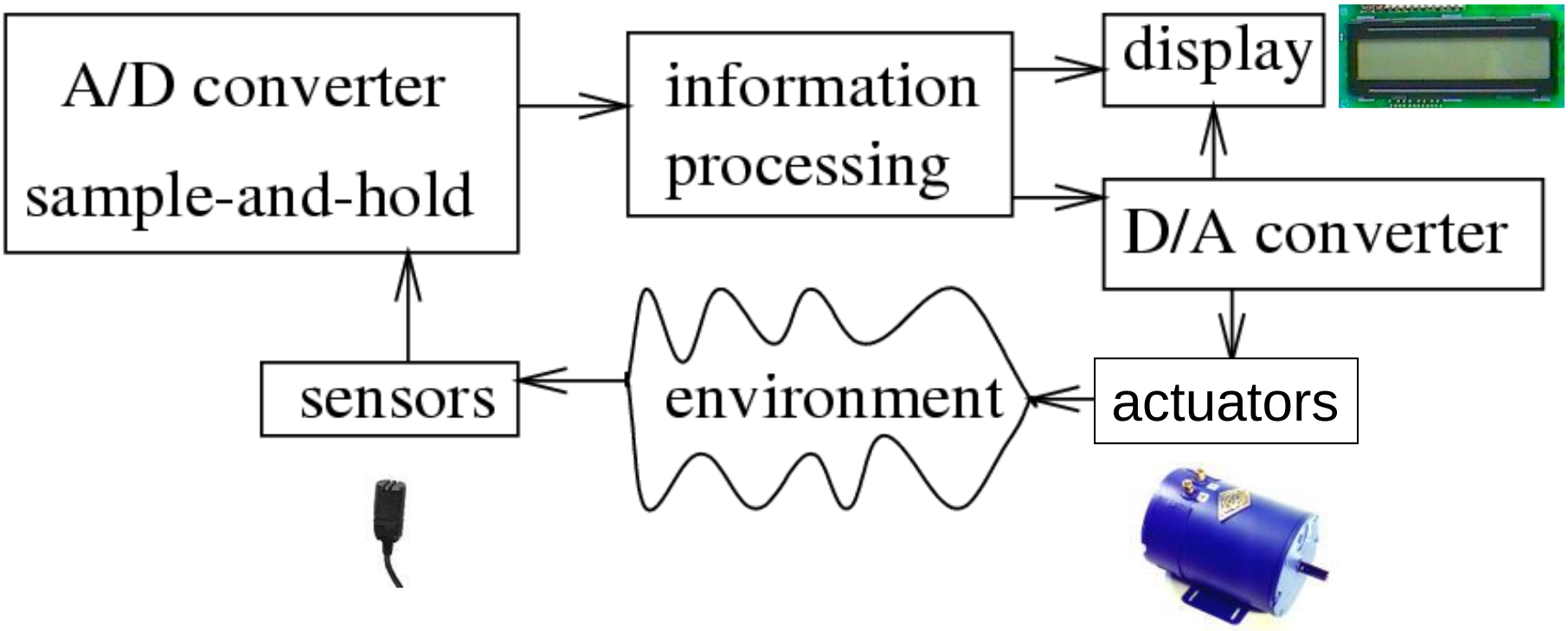
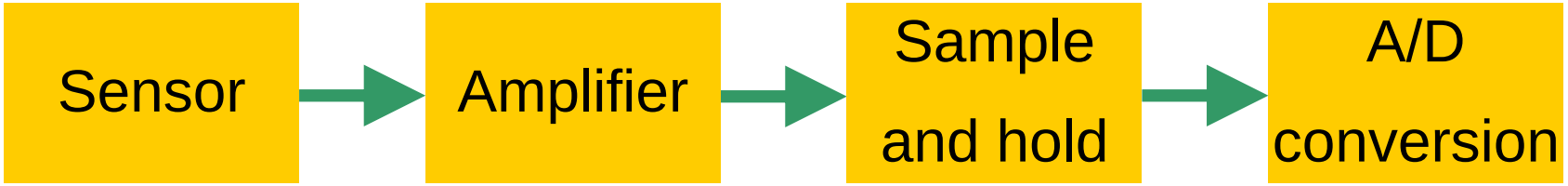
Embedded Systems 12

REVIEW: Embedded System Hardware

Embedded system hardware is frequently used in a loop (***„hardware in a loop“***):



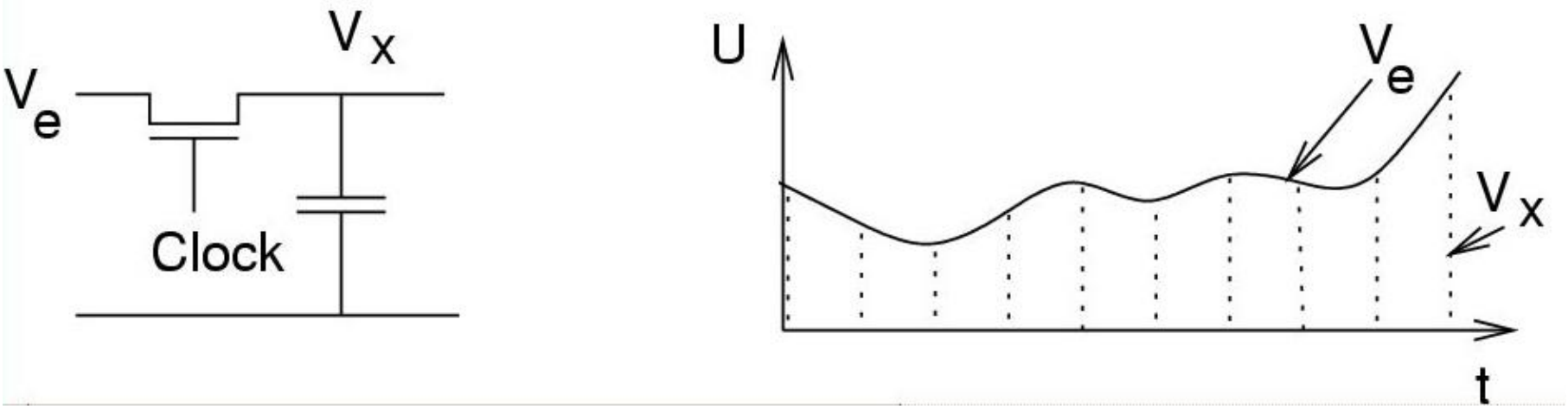
REVIEW: Standard layout of sensor systems



- Sensor: detects/measures entity and converts it to electrical domain
  - May entail ES-controllable actuation: e.g. charge transfer in CCD
- Amplifier: adjusts signal to the dynamic range of the A/D conversion
  - Often dynamically adjustable gain: e.g. ISO settings at digital cameras, input gain for microphones (sound or ultrasound), extremely wide dynamic ranges in seismic data logging
- Sample + hold: samples signal at discrete time instants
- A/D conversion: converts samples to digital domain

Discretization of time

V_e is a mapping $\mathbb{R} \rightarrow \mathbb{R}$

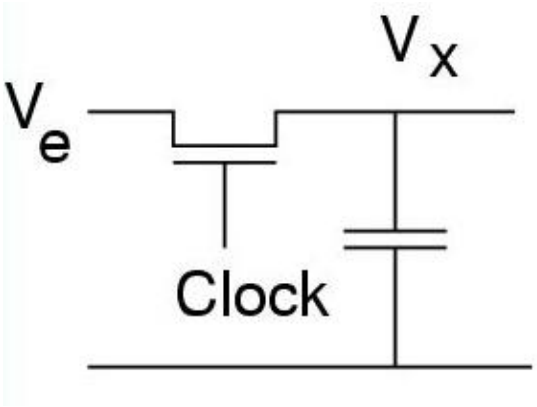


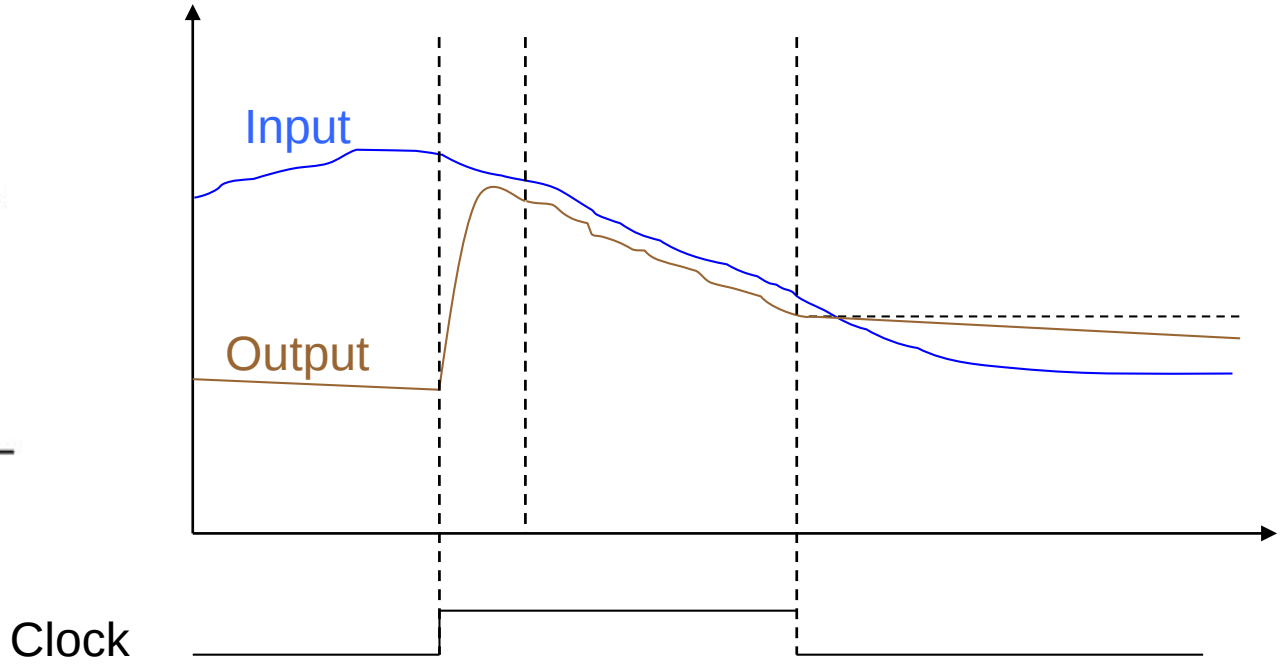
V_x is a **sequence** of values or a mapping $\mathbb{Z} \rightarrow \mathbb{R}$

Discrete time: sample and hold-devices.

Ideally: width of clock pulse -> 0

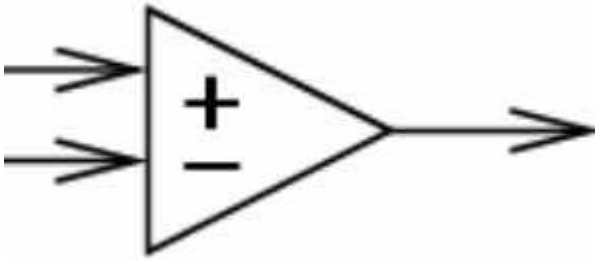
Sample and Hold





Discretization of values: A/D-converters
1. Flash A/D converter (1)

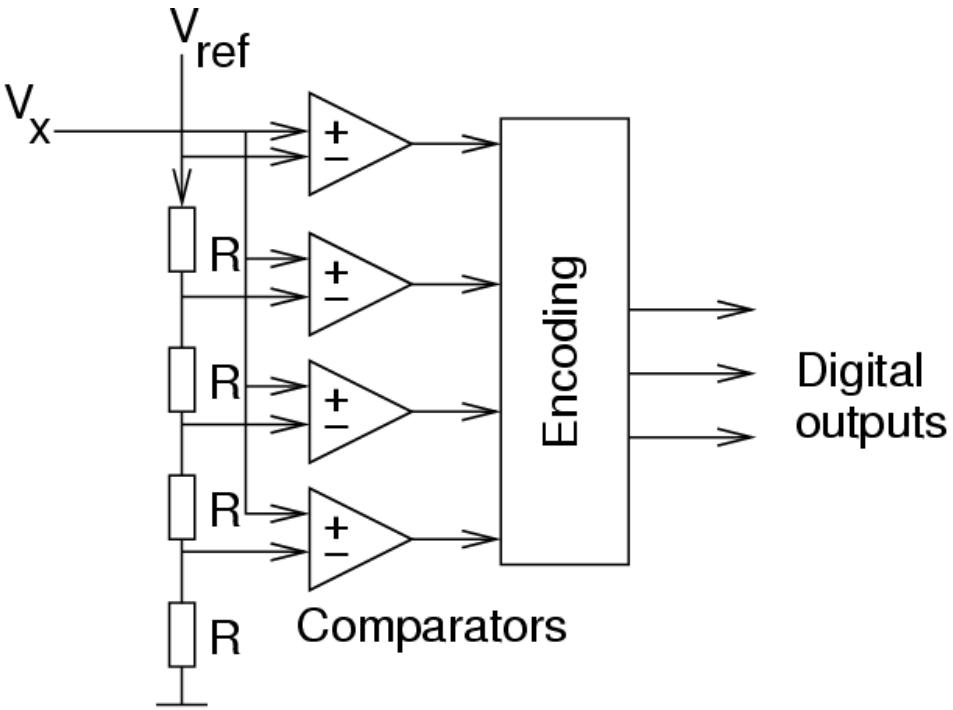
- Basic element: analog comparator



  - Output = ´1´ if voltage at input + exceeds that at input -.
  - Output = ´0´ if voltage at input - exceeds that at input +.

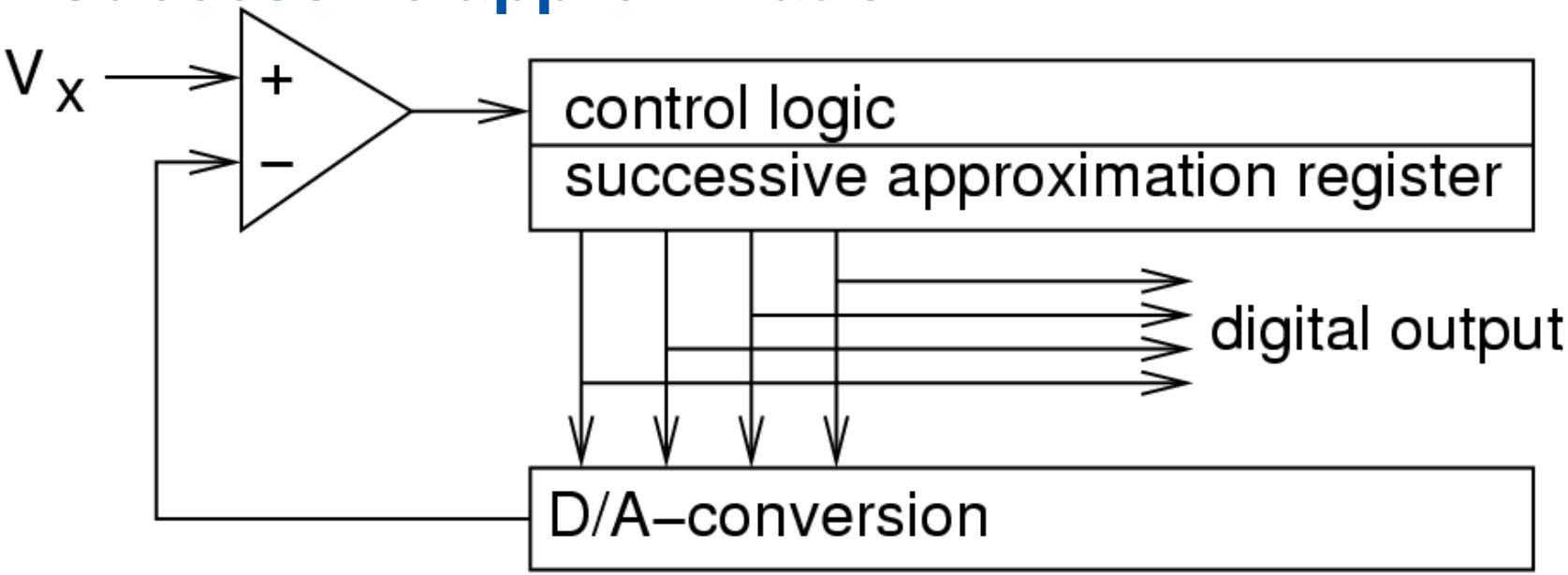
- Idea:
  - Generate *n* different voltages by voltage divider (resistors), e.g. V_{ref}, ¾ V_{ref}, ½ V_{ref}, ¼ V_{ref}.
  - Use *n* comparators for parallel comparison of input voltage V_x to these voltages.
  - Encoder to compute digital output.

Discretization of values: A/D-converters
1. Flash A/D converter (2)



- Parallel comparison with reference voltage
- **Applications**: e.g. in video processing

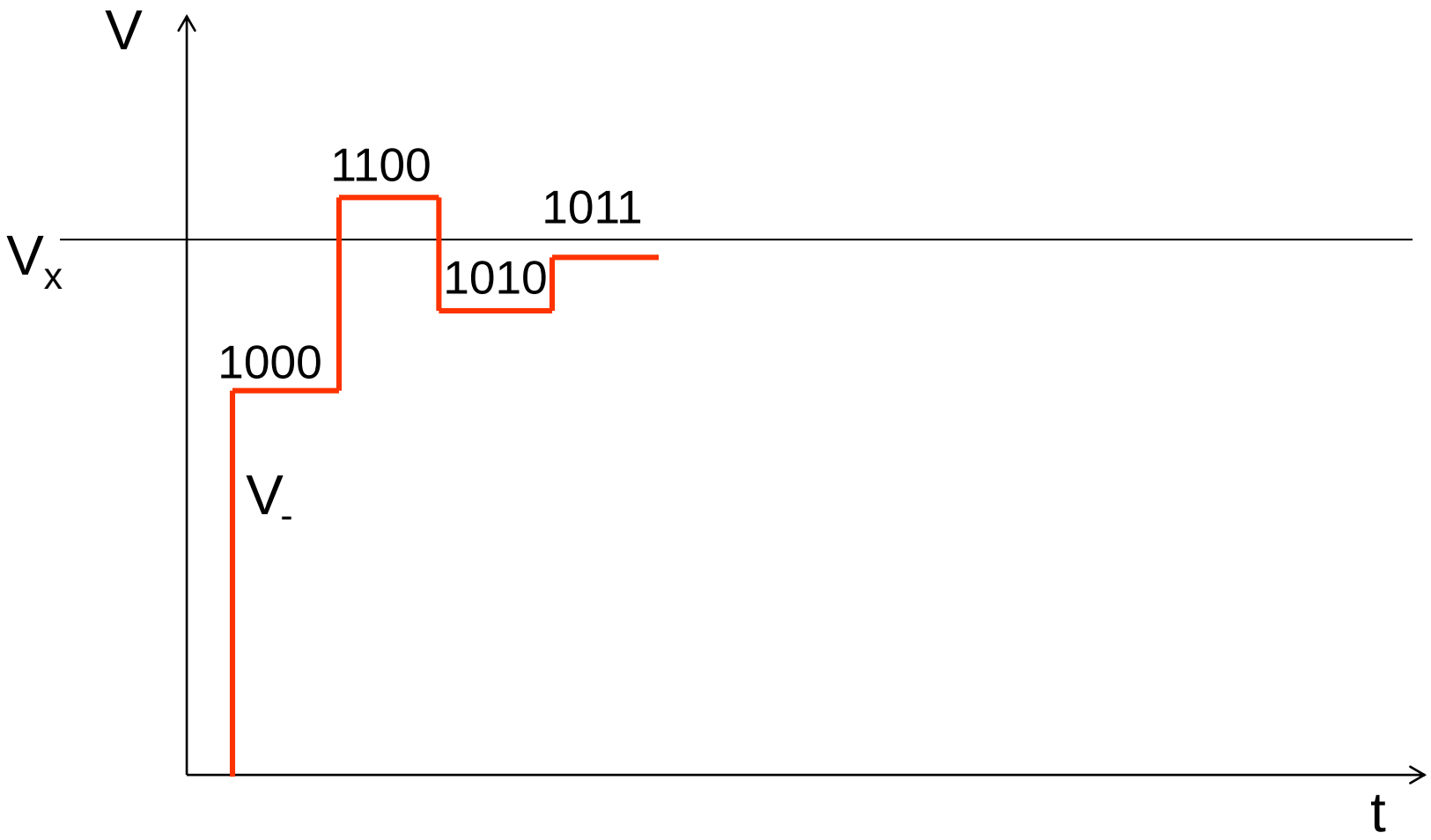
Discretization of values
2. Successive approximation



Key idea: binary search:

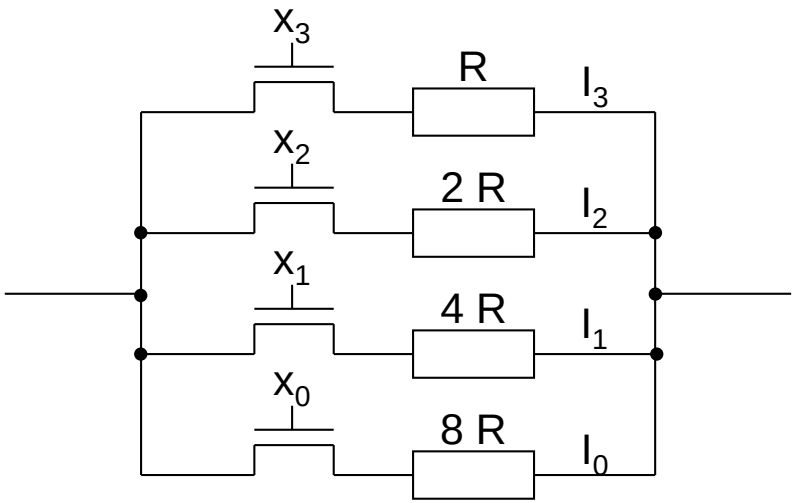
- Set MSB='1'
- if too large: reset MSB
- Set MSB-1='1'
- if too large: reset MSB-1

Successive approximation (2)



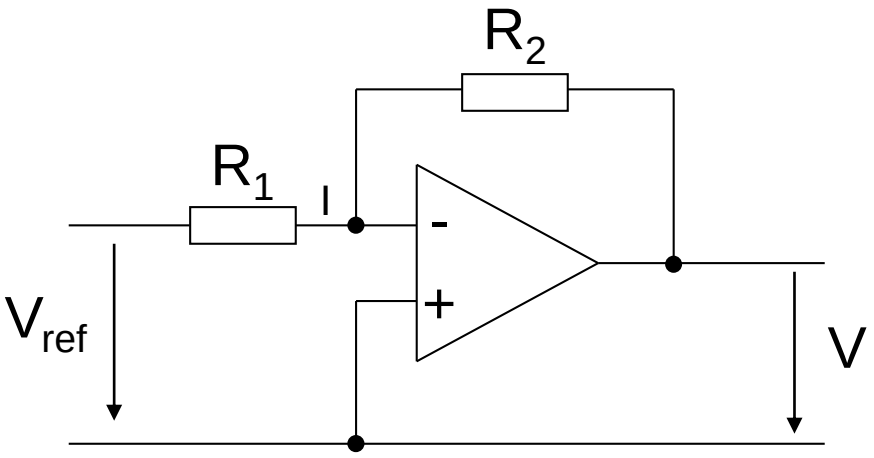
Digital-to-Analog (D/A) Converters

- Convert digital value to conductivity proportional to the digital value

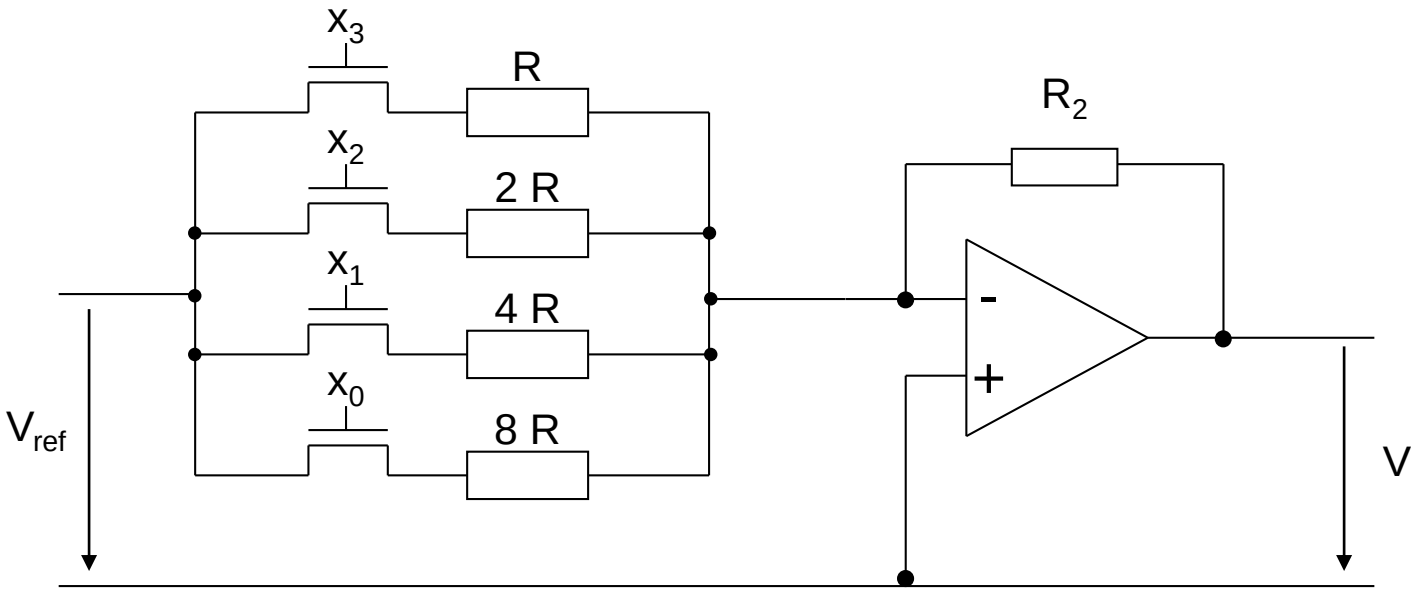


Operational amplifier

- Use operational amplifier to convert conductivity to voltage: $V = - V_{ref} R_2 / R_1$



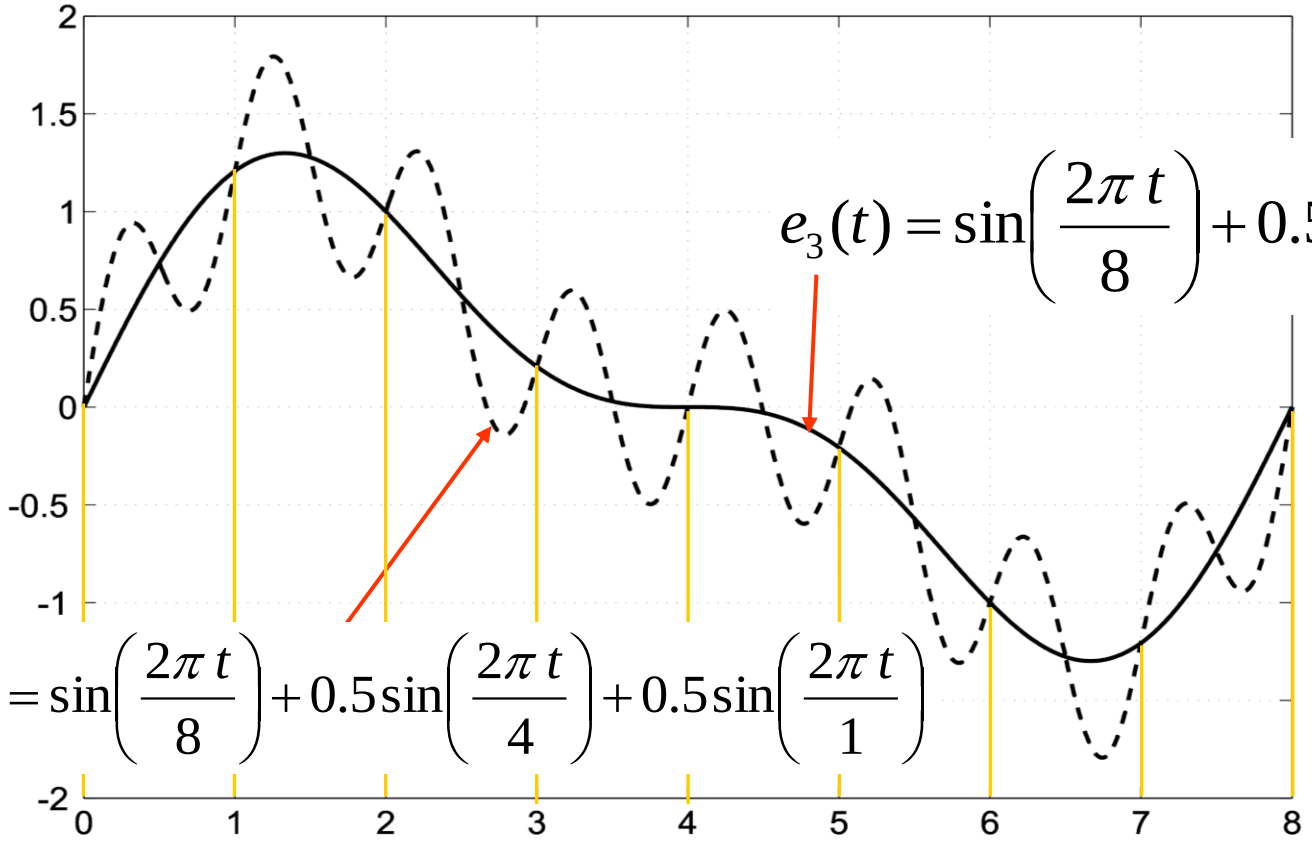
Digital-to-Analog (D/A) Converters (3)



Design Issues with Sensors

- Calibration
  - Relating measurements to the physical phenomenon
  - Can dramatically increase manufacturing costs
- Nonlinearity
  - Measurements may not be proportional to physical phenomenon
  - Correction may be required
  - Feedback can be used to keep operating point in the linear region
- Sampling
  - Aliasing
  - Missed events
- Noise
  - Analog signal conditioning
  - Digital filtering
  - Introduces latency

Aliasing



$$e_3(t) = \sin\left(\frac{2\pi t}{8}\right) + 0.5\sin\left(\frac{2\pi t}{4}\right)$$

$$e_4(t) = \sin\left(\frac{2\pi t}{8}\right) + 0.5\sin\left(\frac{2\pi t}{4}\right) + 0.5\sin\left(\frac{2\pi t}{1}\right)$$

- Periods of p=8,4,1
- Indistinguishable if sampled at integer times, p_s=1

Aliasing

Nyquist criterion (sampling theory):

Aliasing can be avoided if we restrict the frequencies of the incoming signal to less than half of the sampling rate.

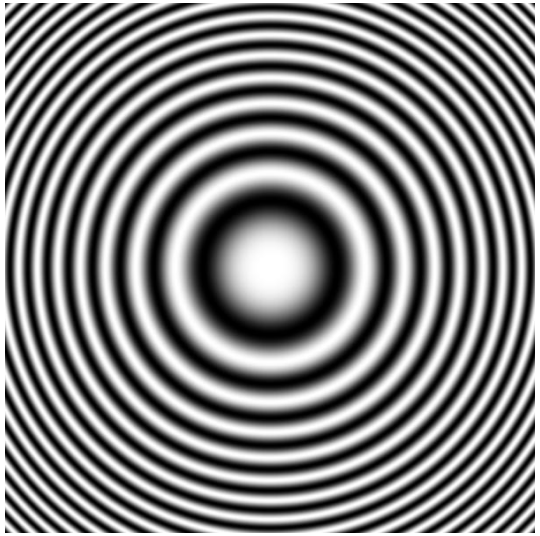
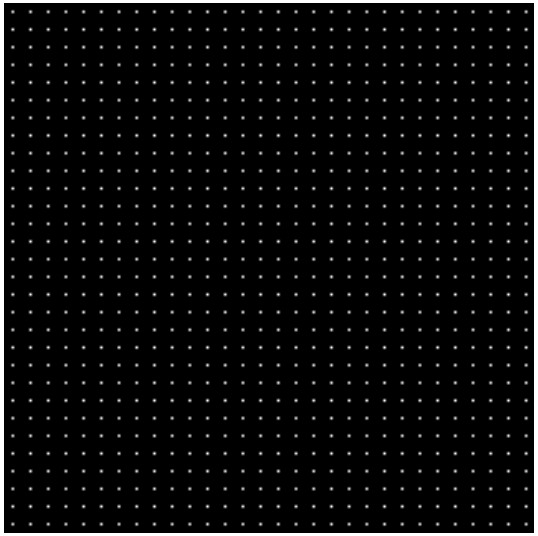
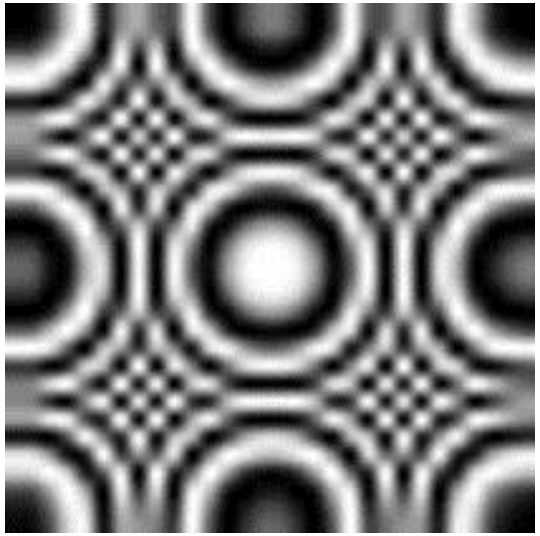
$p_s < ½\ p_N$ where p_N is the period of the "fastest" sine wave

or $f_s > 2\ f_N$ where f_N is the frequency of the "fastest" sine wave

f_N is called the **Nyquist frequency**, f_s is the **sampling rate**.

See e.g. [Oppenheim/Schafer, 2009]

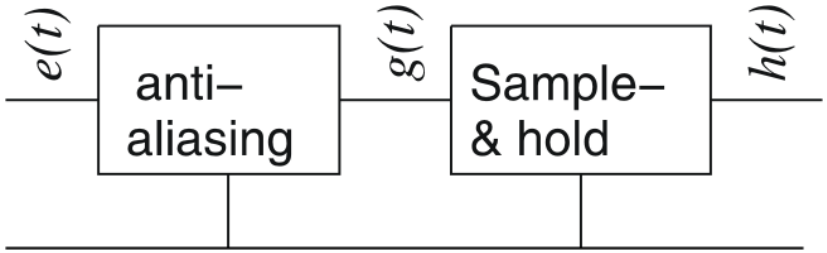
Graphics

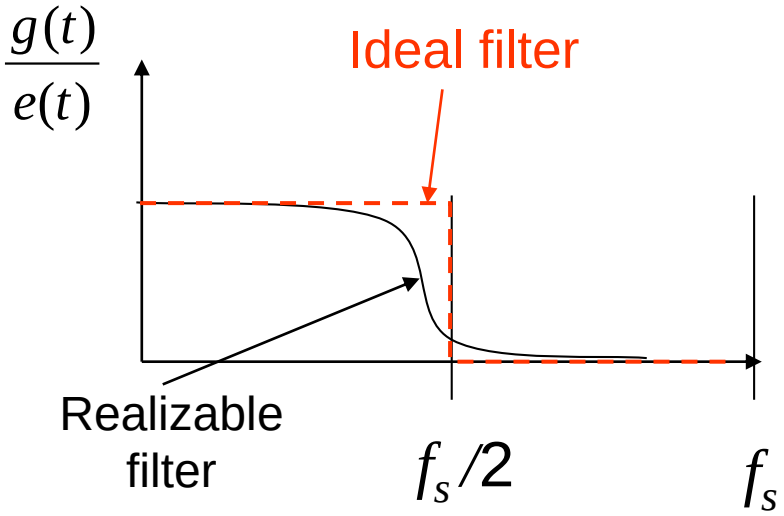




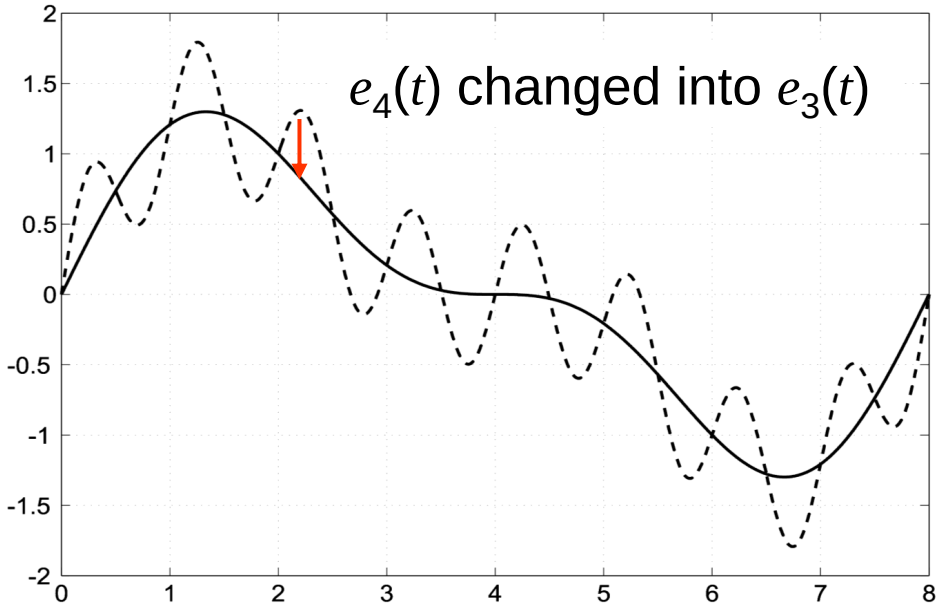
(Wikimedia Commons)

Anti-aliasing filter

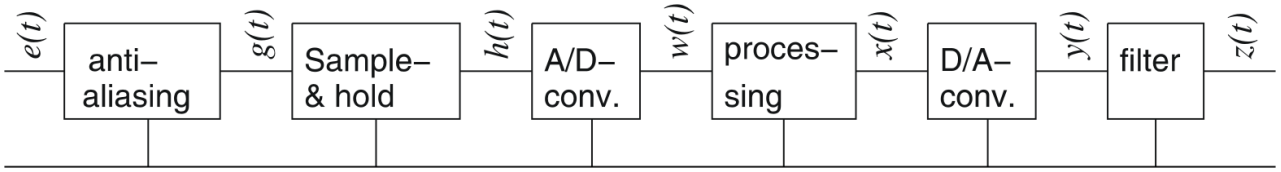
A filter is needed to remove high frequencies

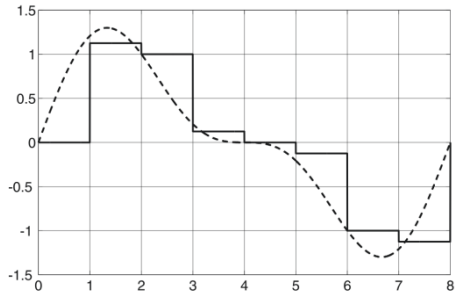






Possible to reconstruct input signal?





- Assuming Nyquist criterion met
- Let $\{t_s\}$, $s = ...,-1,0,1,2, ...$ be times at which we sample $g(t)$
- Assume a constant sampling rate of $1/p_s$ ($\forall s$: $p_s = t_{s+1} - t_s$).
- According to sampling theory, we can approximate the input signal using the **Shannon-Whittaker interpolation:**

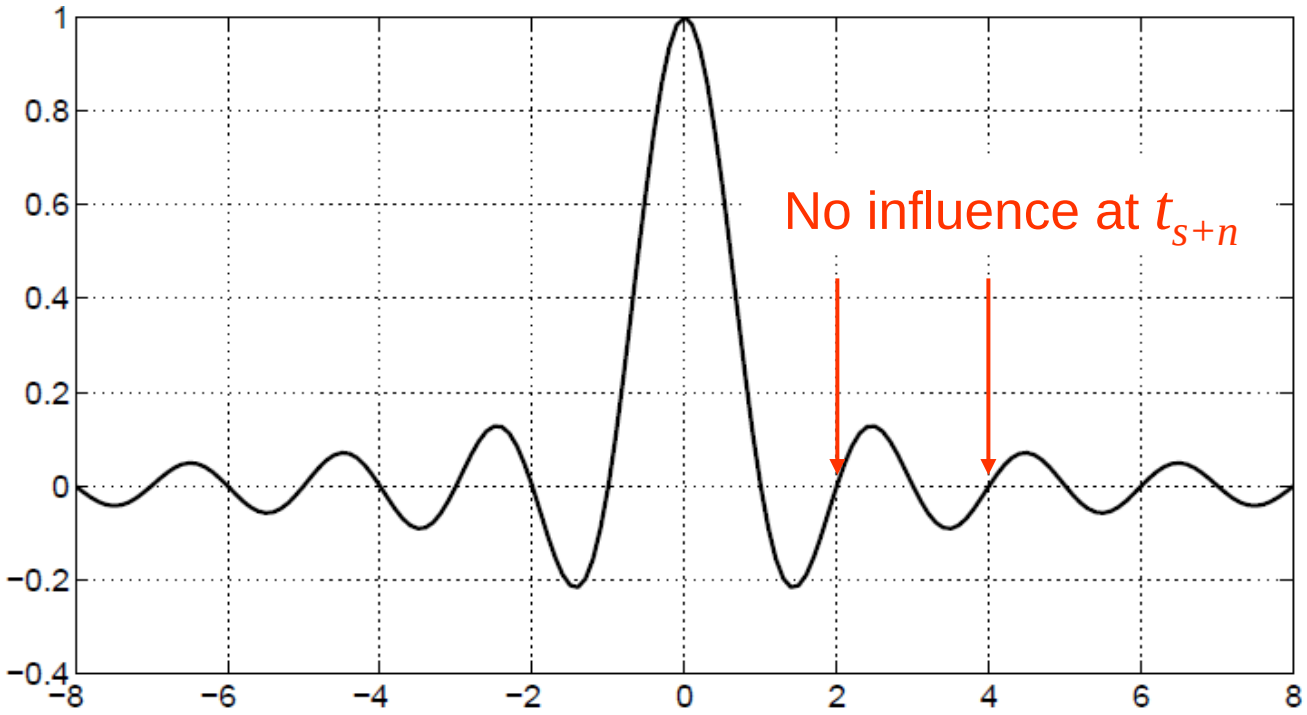
$$z(t) = \sum_{s=-\infty}^{\infty} \frac{y(t_s)\sin\frac{\pi}{p_s}(t - t_s)}{\frac{\pi}{p_s}(t - t_s)}$$

Weighting factor for influence of $y(t_s)$ at time t

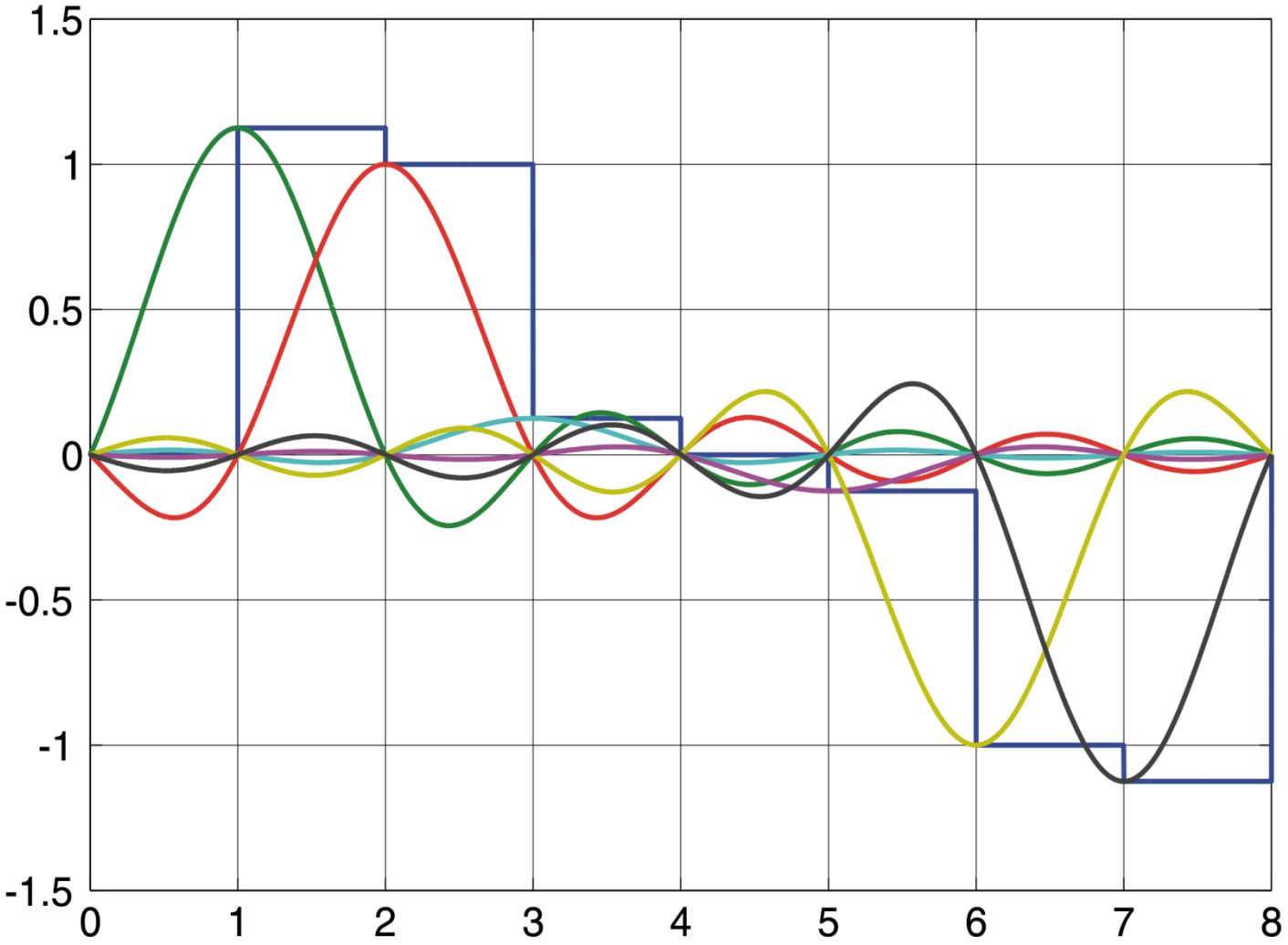
[Oppenheim, Schafer, 2009]

Weighting factor for influence of $y(t_s)$ at time t

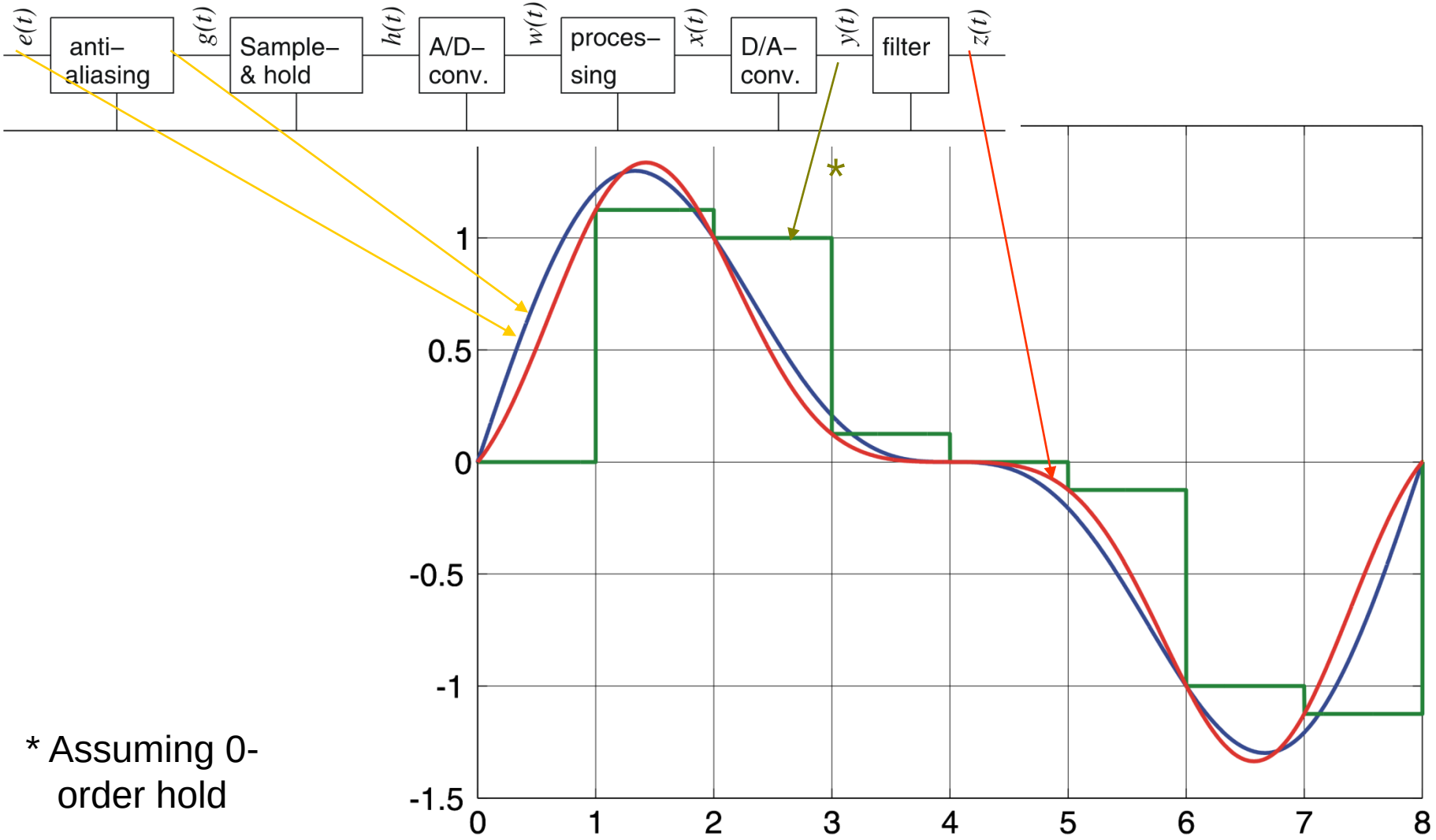
$$sinc(t - t_s) = \frac{sin\left(\frac{\pi}{p_s}(t - t_s)\right)}{\frac{\pi}{p_s}(t - t_s)}$$



Contributions from the various sampling instances



(Attempted) reconstruction of input signal

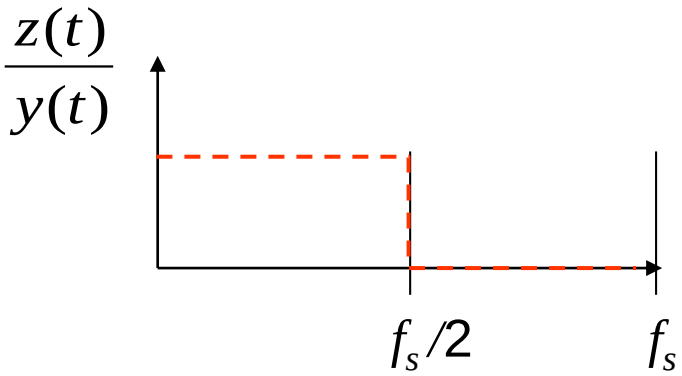


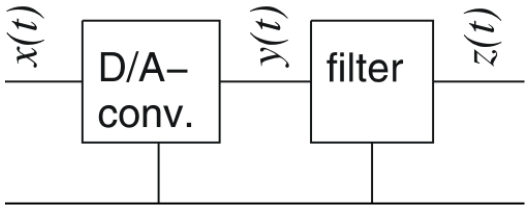
* Assuming 0-order hold

How to compute the *sinc*( ) function?

$$z(t) = \sum_{s=-\infty}^{\infty} \frac{y(t_s)\sin\frac{\pi}{T_s}(t-t_s)}{\frac{\pi}{T_s}(t-t_s)}$$

- **Filter theory:** The required interpolation is performed by an ideal low-pass filter (*sinc* is the Fourier transform of the low-pass filter transfer function)





Filter removes high frequencies present in $y(t)$

How precisely are we reconstructing the input?

$$z(t) = \sum_{s=-\infty}^{\infty} \frac{y(t_s) sin \frac{\pi}{T_s}(t - t_s)}{\frac{\pi}{T_s}(t - t_s)}$$

- **Sampling theory**:
  - **Reconstruction using *sinc* () is precise**
- However, it may be impossible to really compute $z(t)$

Limitations

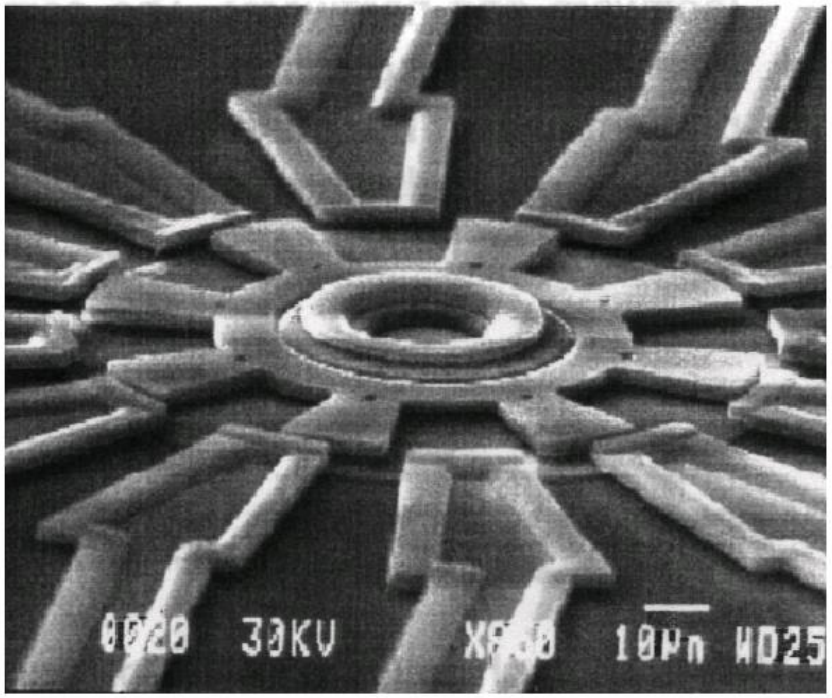
$$z(t) = \sum_{s=-\infty}^{\infty} \frac{y(t_s) sin \frac{\pi}{T_s}(t - t_s)}{\frac{\pi}{T_s}(t - t_s)}$$

- Actual filters do not compute $sinc(\ )$
  In practice, filters are used as an approximation.
  Computing good filters is an art itself!
- All samples must be known to reconstruct $e(t)$ or $g(t)$.
  ☞ Waiting indefinitely before we can generate output!
  In practice, only a finite set of samples is available.
- Actual signals are never perfectly bandwidth limited.
- Quantization noise cannot be removed.

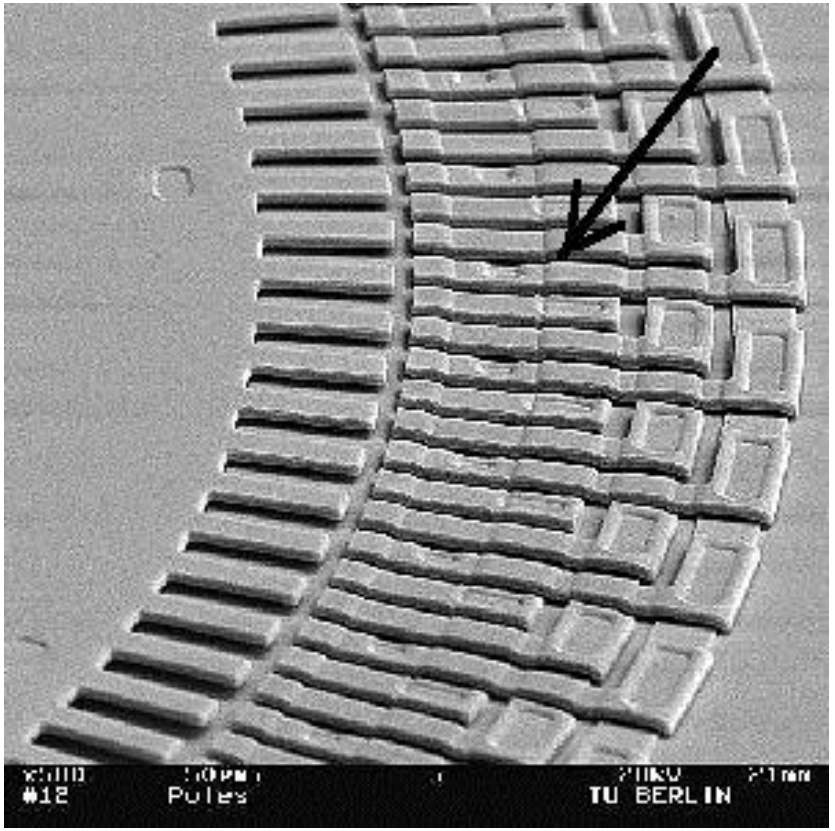
Actuators and output

- Huge variety of actuators and outputs
- Two base types:
  - analogue drive
    (requires D/A conversion, unless on/off sufficient)
    - CRTs, speakers, electrical motors with collector
    - electromagnetic (e.g., coils) or electrostatic drives
    - piezo drives
  - digital drive (requires amplification only)
    - LEDs
    - stepper motors
    - relais, electromagnetic valve (if actuation slope irrelevant)

Micromotors

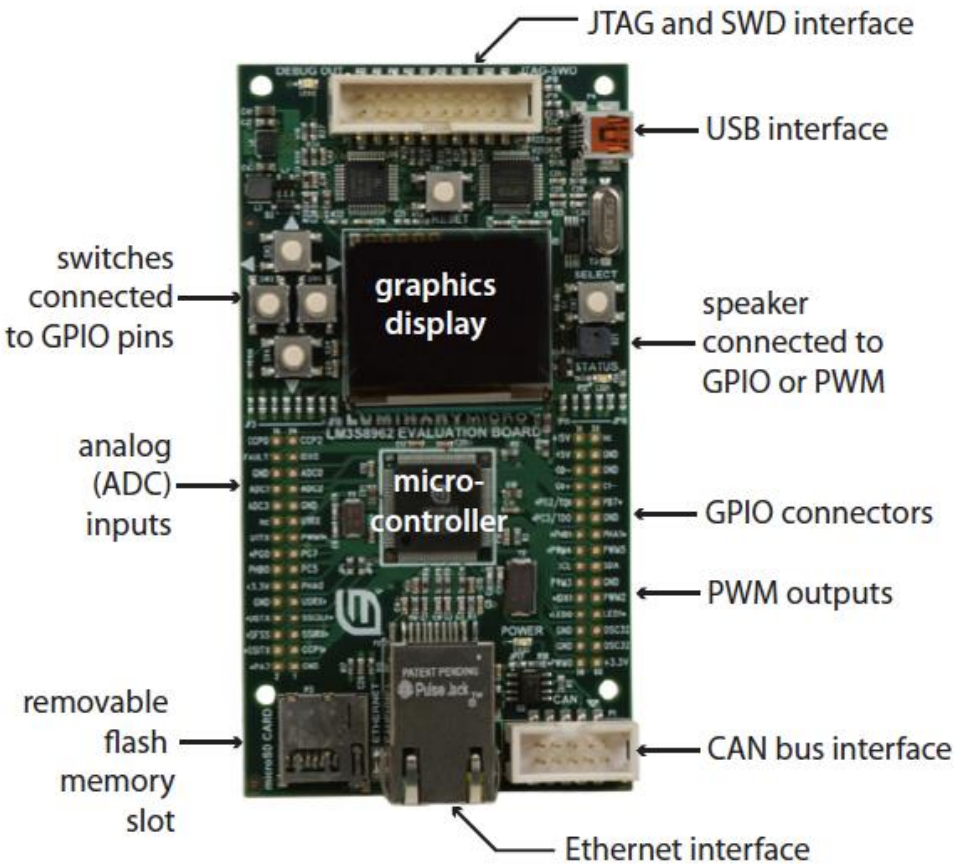


(© MCNC)



(TU Berlin)

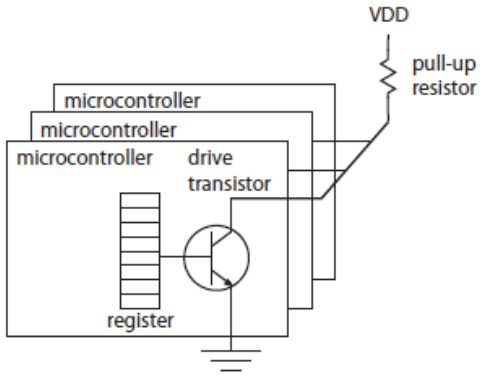
Interfaces



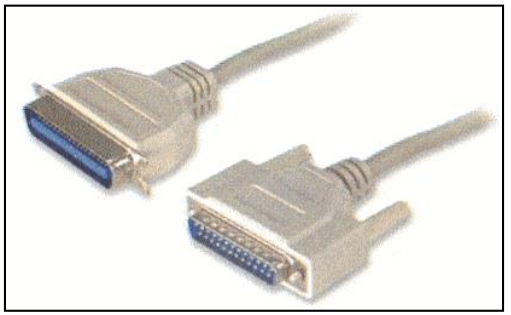
Stellaris®LM3S8962 evaluation board

Interfaces

- Pulse width modulation (PWM)
- General-Purpose Digital I/O (GPIO)



- Parallel
  - Multiple data lines transmitting data
  - Ex: PCI, ATA, CF cards, Bus



- Serial
  - Single data line transmitting data
  - Ex: USB, SATA, SD cards,

Example Using a Serial Interface

In an Atmel AVR 8-bit microcontroller, to send a byte over a serial port, the following C code will do:

```
while(!(UCSR0A & 0x20));
UDR0 = x;
```

- x is a variable of type uint8.
- UCSR0A and UDR0 are variables defined in header.
- They refer to memory-mapped registers.

Send a Sequence of Bytes

```
for(i = 0; i < 8; i++) {
      while(!(UCSR0A & 0x20));
      UDR0 = x[i];
}
```

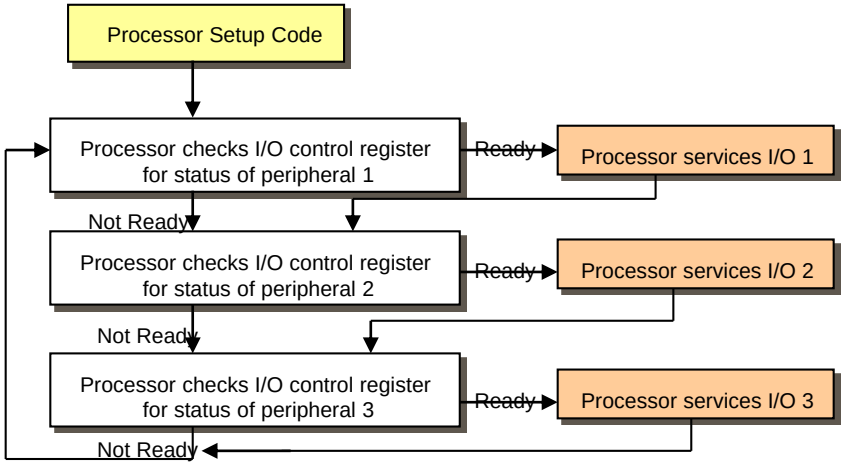
How long will this take to execute? Assume:

- 57600 baud serial speed.
- 8/57600 =139 microseconds.
- Processor operates at 18 MHz.

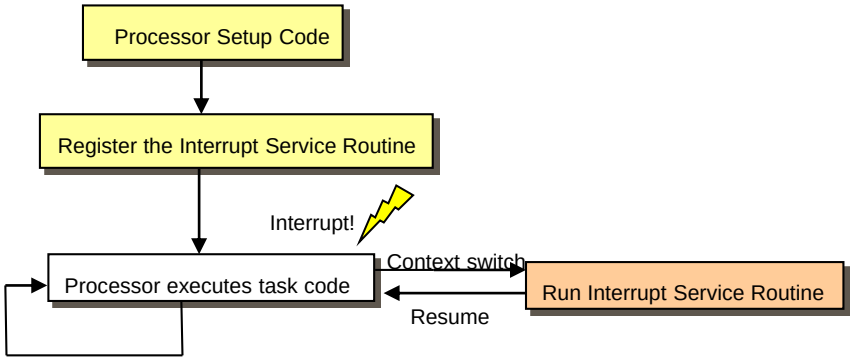
Each while loop will consume 2500 cycles.

Input Mechanisms in Software

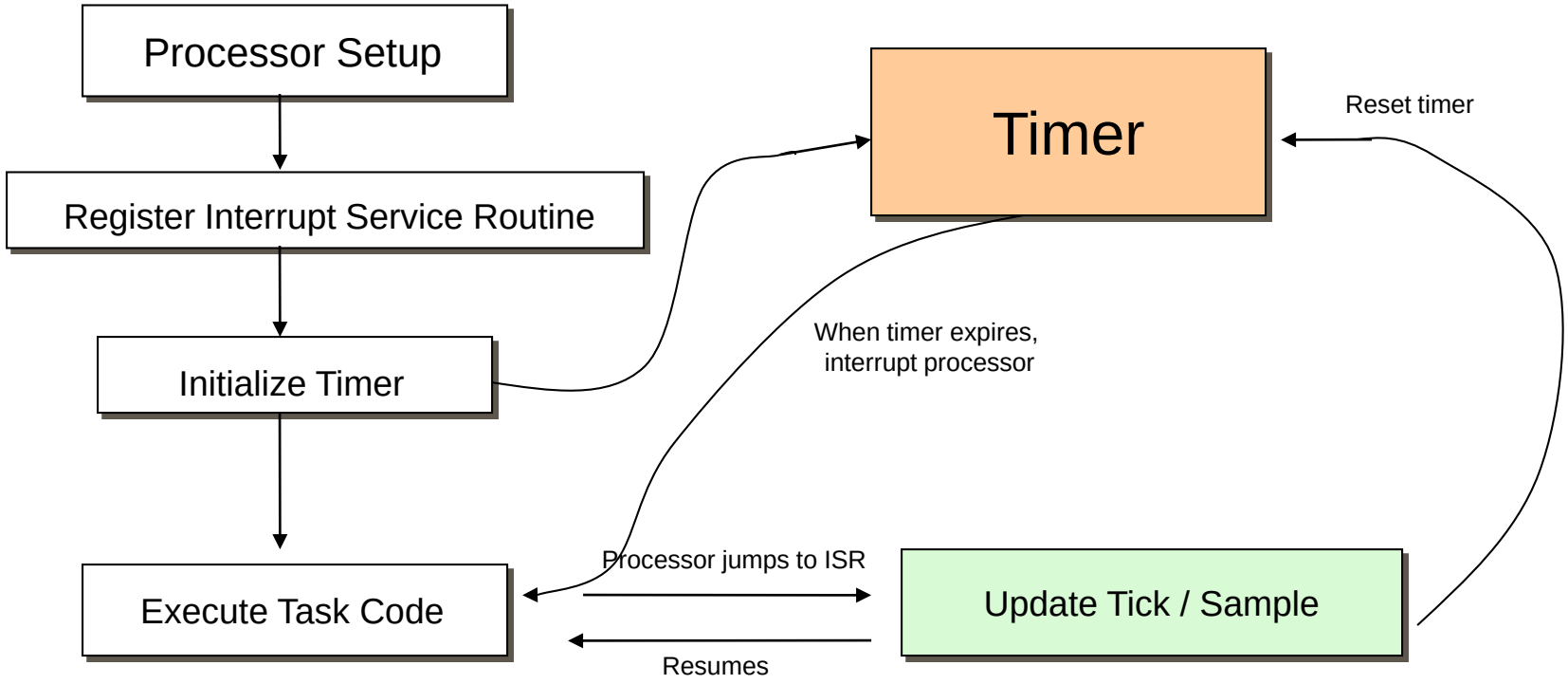
- Polling
  - Main loop checks each I/O device periodically.
  - If input is ready, processor initiates communication.



- Interrupts
  - External hardware alerts the processor that input is ready.
  - Processor suspends what it is doing, invokes an interrupt service routine (ISR).



Timed Interrupt



Example:
Do something for 2 seconds then stop

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

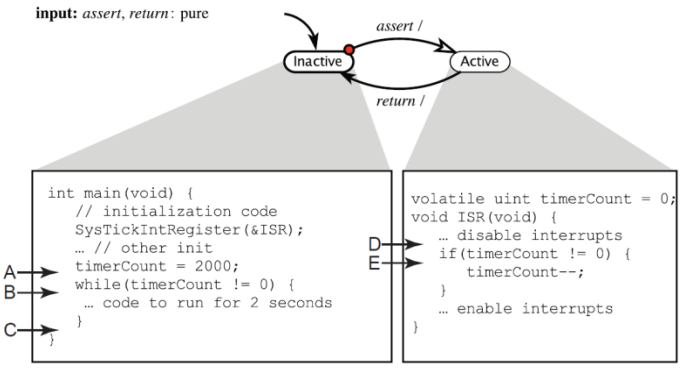
static variable: declared outside main() puts them in statically allocated memory (not on the stack)

volatile: C keyword to tell the compiler that this variable may change at any time, not (entirely) under the control of this program.

Interrupt service routine

Registering the ISR to be invoked on every SysTick interrupt

Example



variables: *timerCount*: `uint`
input: *assert*: pure