3

Interfaces

Uwe R. Zimmer - The Australian National University

References

[Burns2007]
Burns, A & Wellings, A
Concurrent and Real-Time Programming in Ada, edition
Cambridge University Press 2007

[Motorola1996]
Motorola
Time Processing Unit
Reference Manual 1996 pp. 1-142

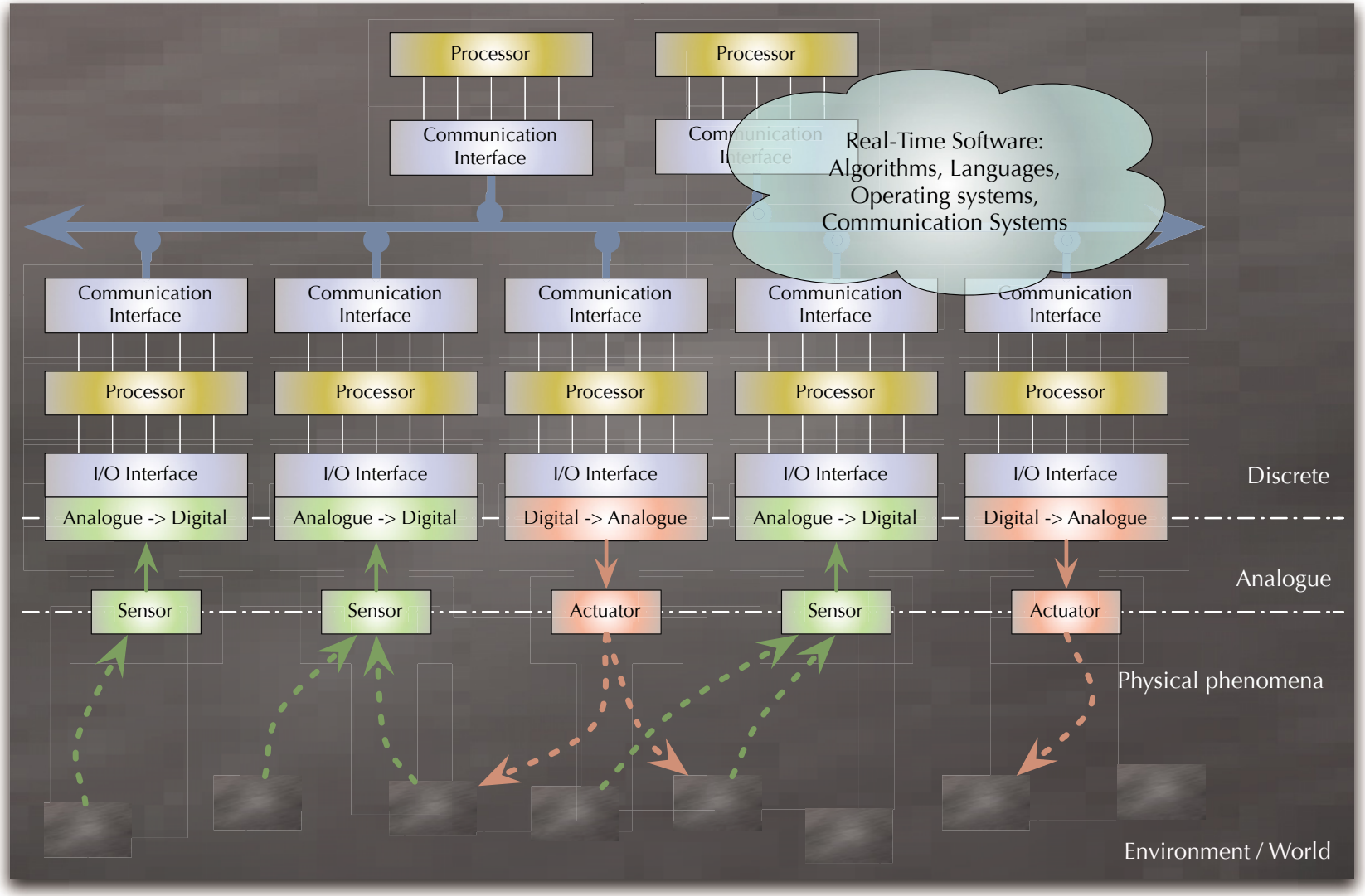
[Motorola2000]
Motorola
MPC565 & MPC566
2000 pp. 1-1312

[Peacock1997]
Peacock, GR
Standards for temperature sensors
http://www.temperatures.com/resources/standards/ , 1997

[Semiconductors**1999**]
National Semiconductors
LM 12L458 - 12 bit, Data Acquisition
1999 pp. 1-36

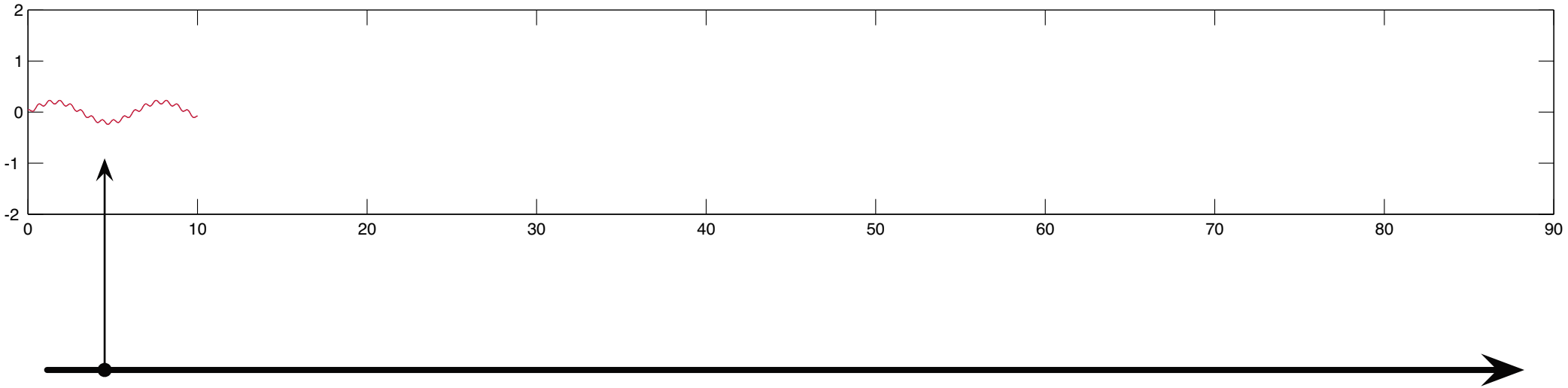
[Semiconductors**2002**]
National Semiconductors,
ADC 08200 - 8 bit, 20 MSPS to 200 MSPS
Datasheet 2002 pp. 1-19

Real-Time Systems Components: Interfaces



A/D, D/A & Interfaces

Signal chain

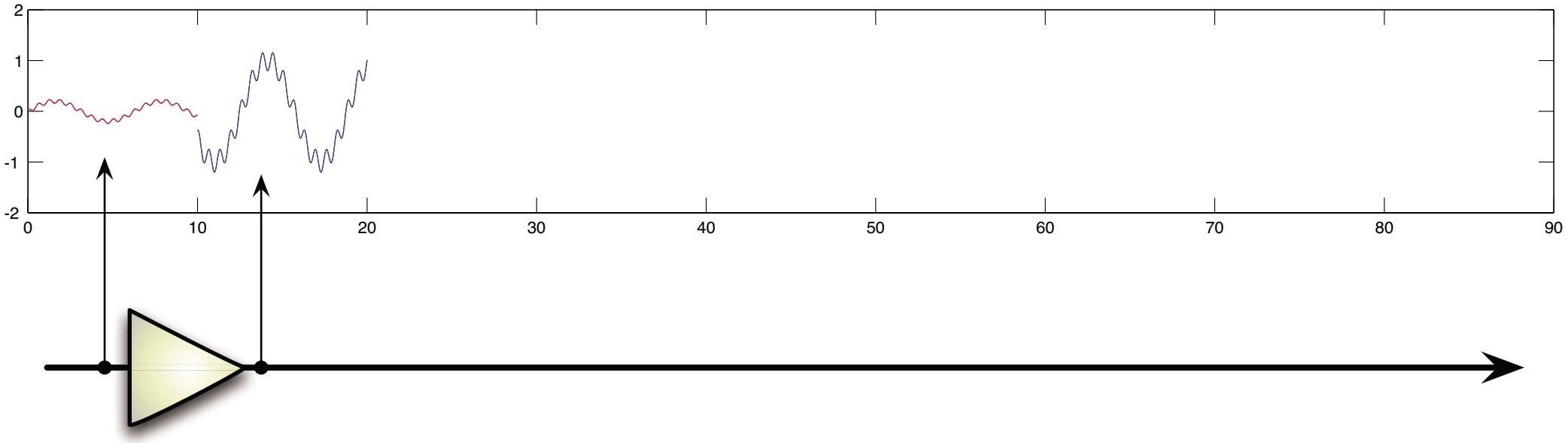


Original signal

(Weakish, noise- and interference-affected, carries additional higher frequency signals)

A/D, D/A & Interfaces

Signal chain

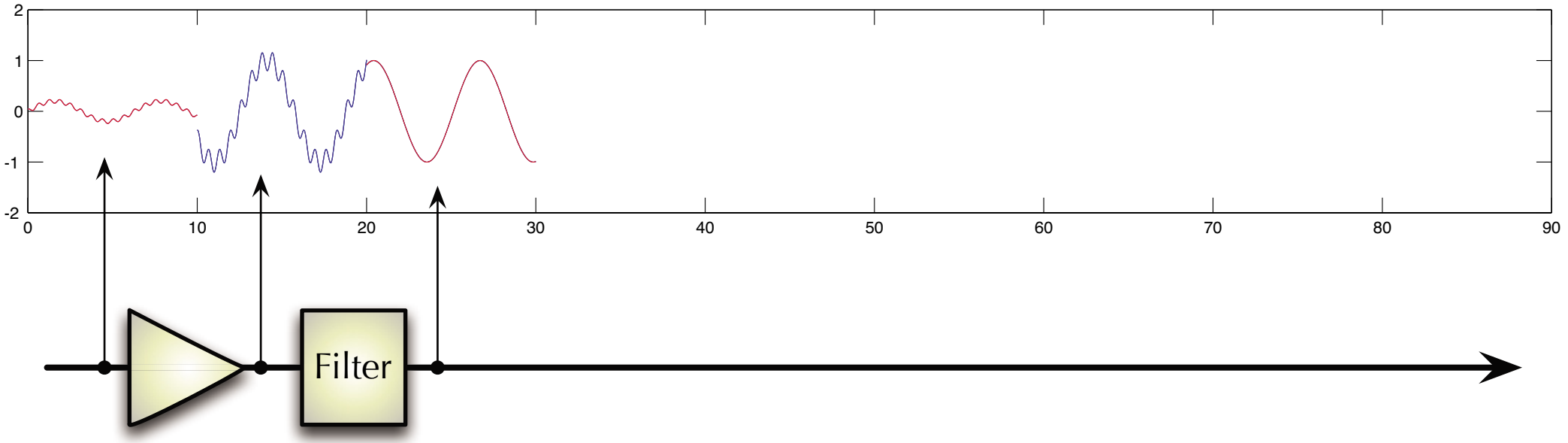


Amplified signal

(Stronger, noise- and interference-affected, carries additional higher frequency signals)

A/D, D/A & Interfaces

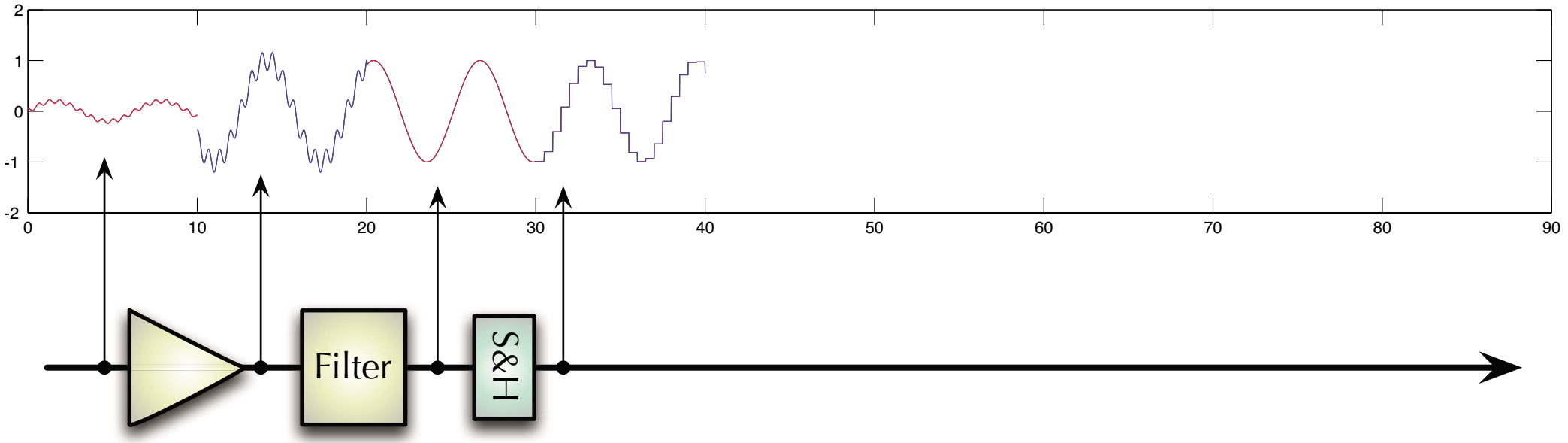
Signal chain



Filtered ("low passed") signal

(Higher frequency signals have been eliminated – essential precondition for next stage)

Signal chain

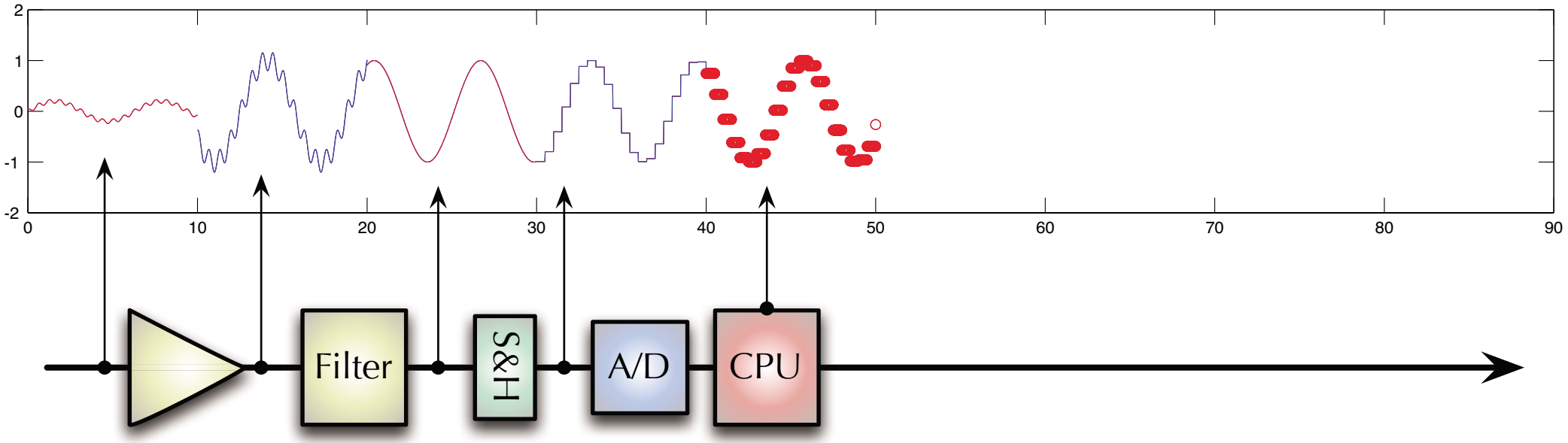


Signal after sample-and-hold circuit

(Samples are taken (over short time-span) and held until the next sample time)

A/D, D/A & Interfaces

Signal chain

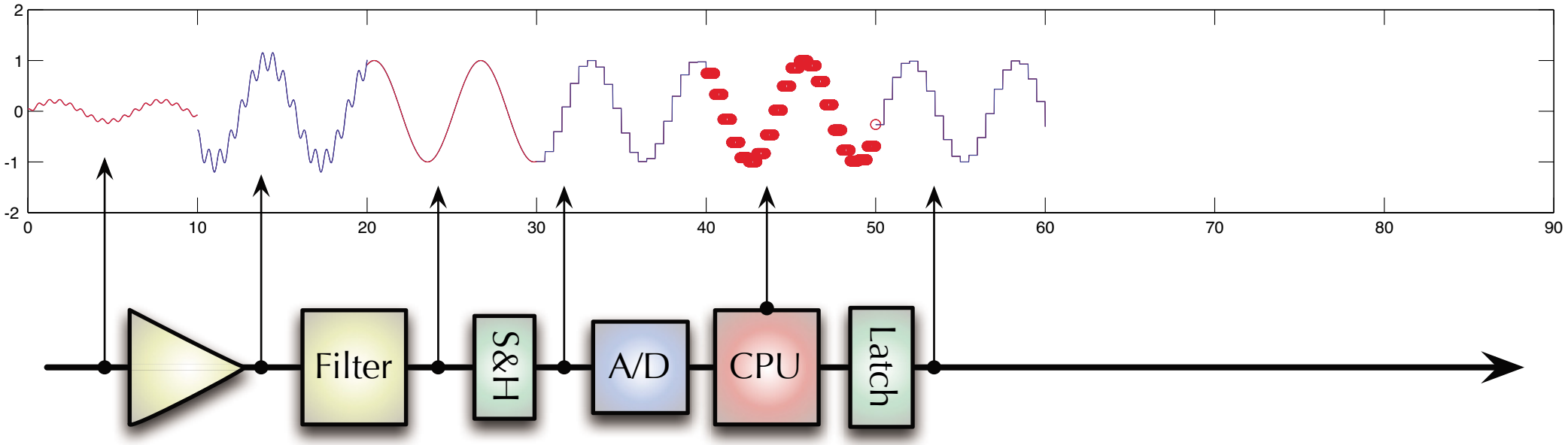


Discrete values inside CPU memory (after A/D conversion)

(Discrete, quantized representation)

A/D, D/A & Interfaces

Signal chain

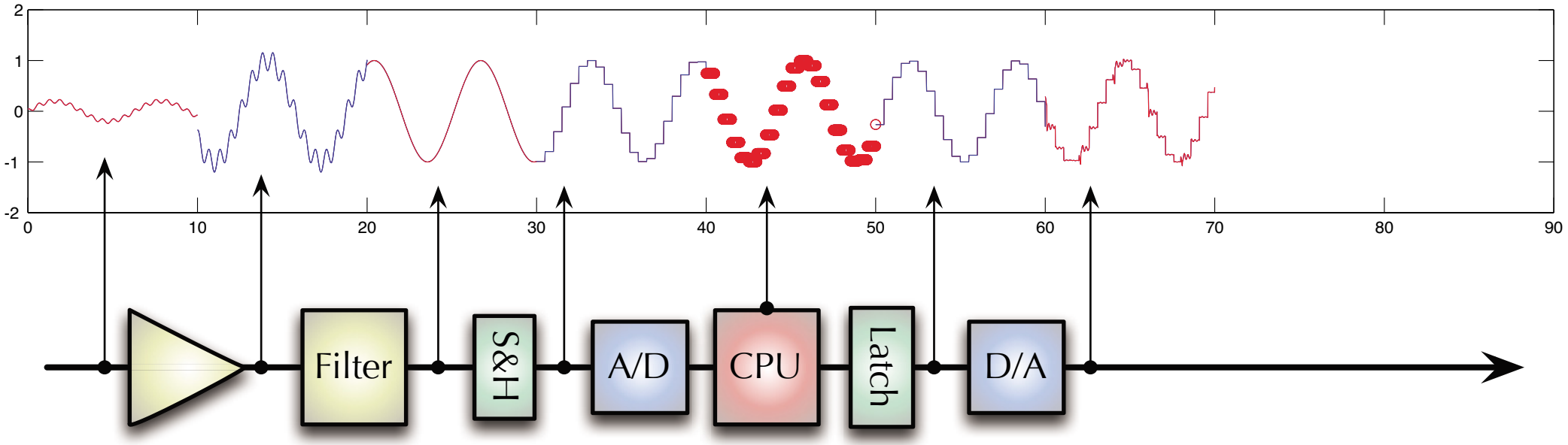


Discrete signal at CPU output gate

(Discrete, quantized representation presented on digital output interface)

A/D, D/A & Interfaces

Signal chain

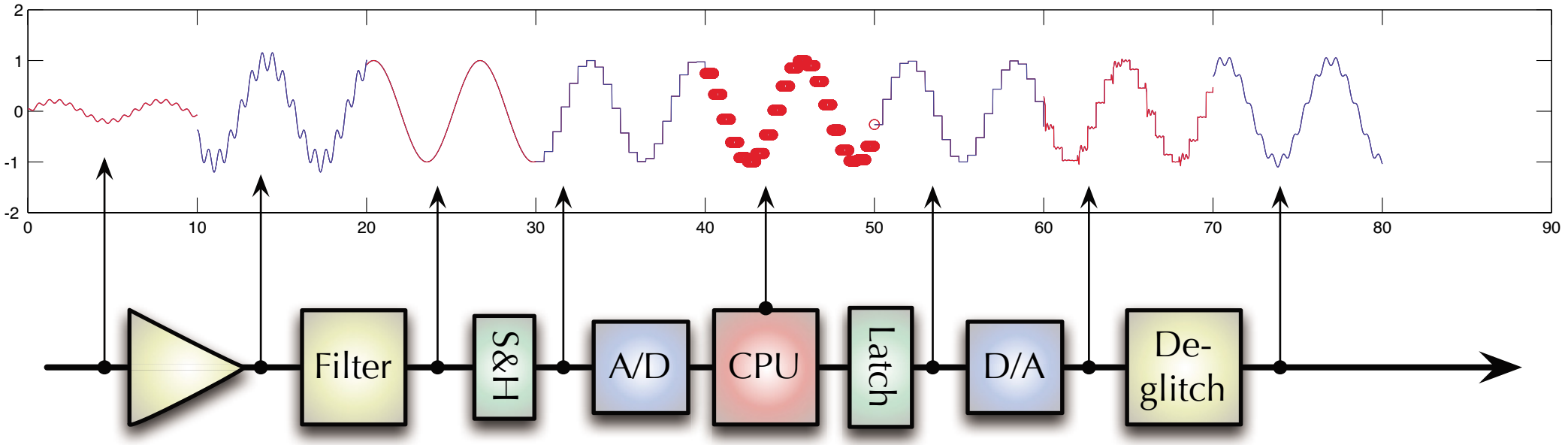


Analogue signal after D/A conversion

(Limited bandwidth leads to glitches in the analogue signal)

A/D, D/A & Interfaces

Signal chain

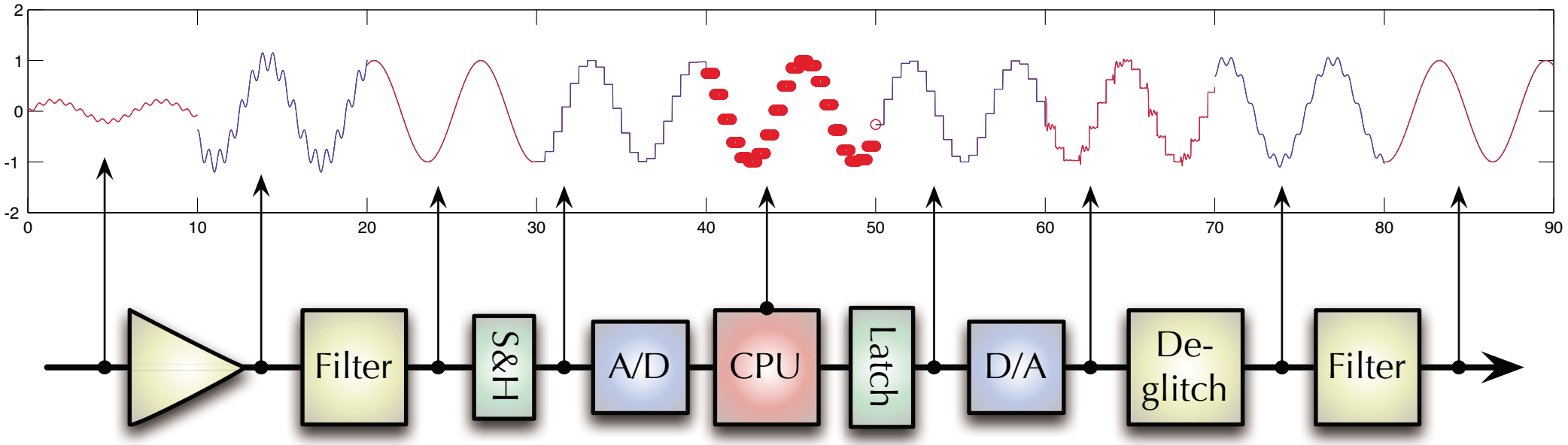


Smoothed ("deglitched") signal

(Synchronized filter leads to predictable, analogue transition steps)

A/D, D/A & Interfaces

Signal chain

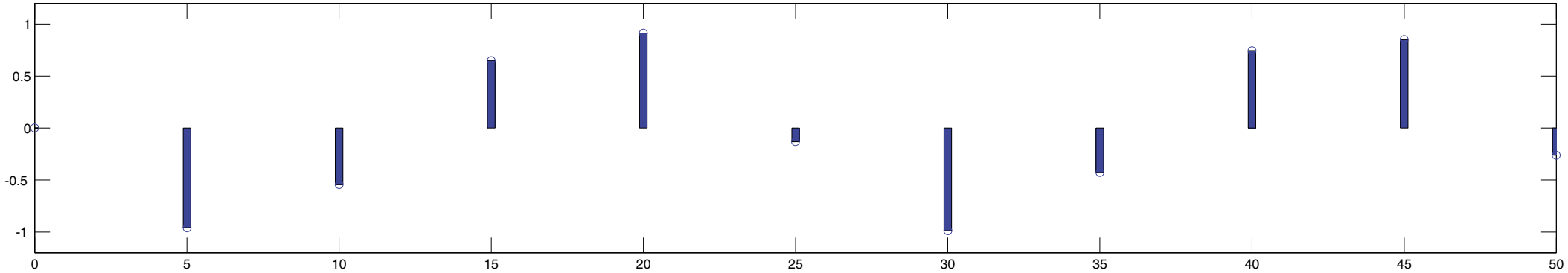


Filtered ("low passed") signal

(Signals introduced by the conversion process are eliminated)

A/D, D/A & Interfaces

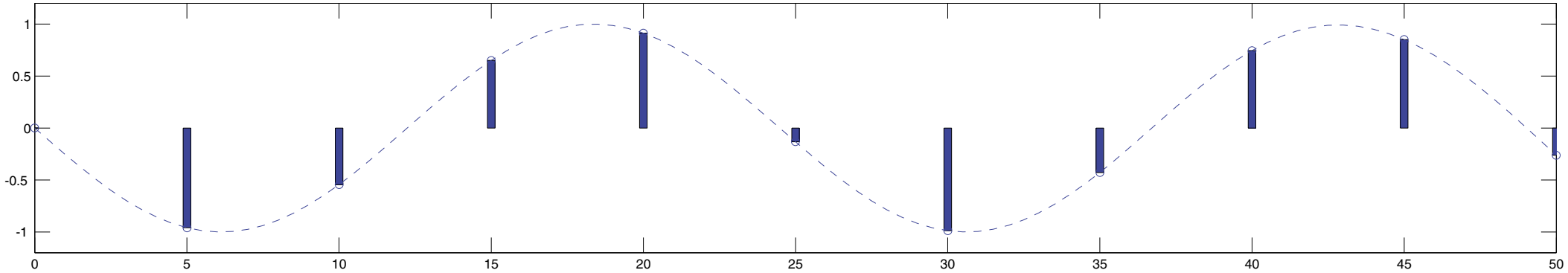
Sampling



Sample data with frequency f_s

A/D, D/A & Interfaces

Sampling

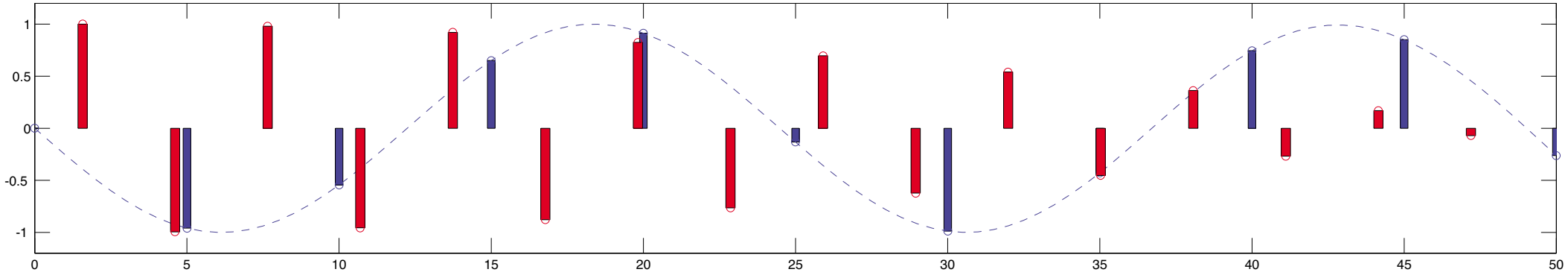


Sample data with frequency f_s

☞ Interpolation suggests a source signal ----------

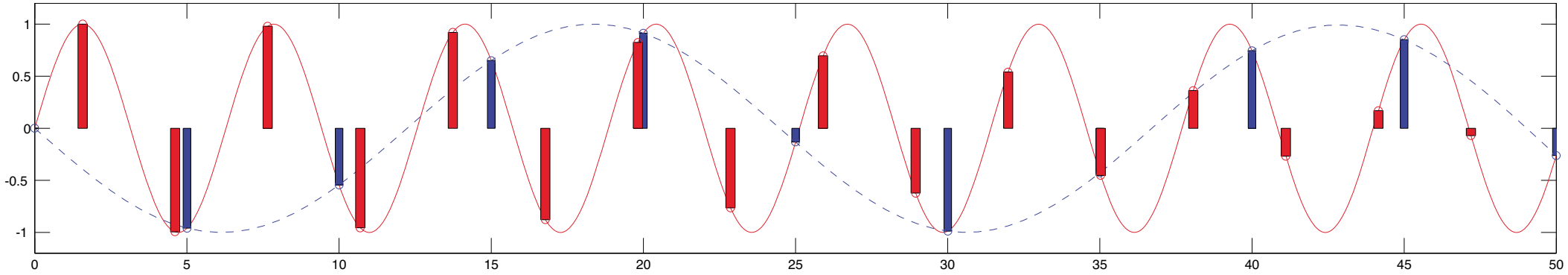
A/D, D/A & Interfaces

Sampling



Sample data with frequency f_s

Sampling

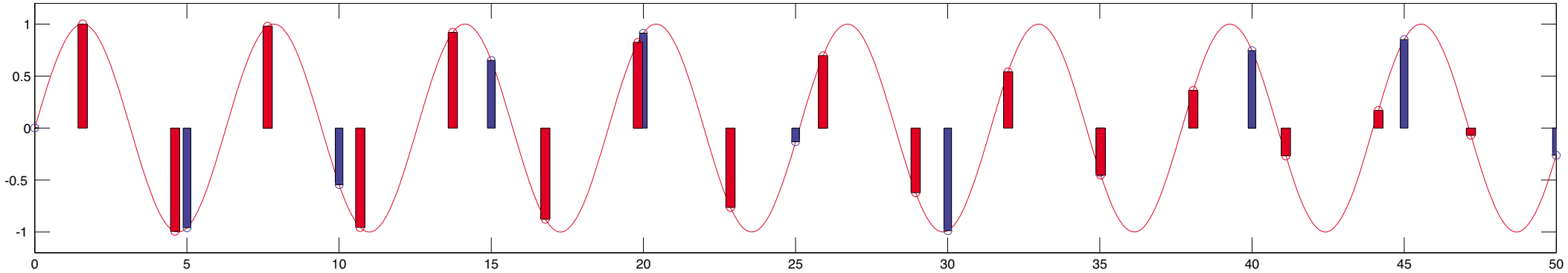


Sample data with frequency f_s

☞ Interpolation suggests a source signal ——————

☞ The phenomenon of the wrongly observed signal - - - - - - - - - - -
at a lower frequency f_s is called **aliasing**.

Nyquist's Criterion

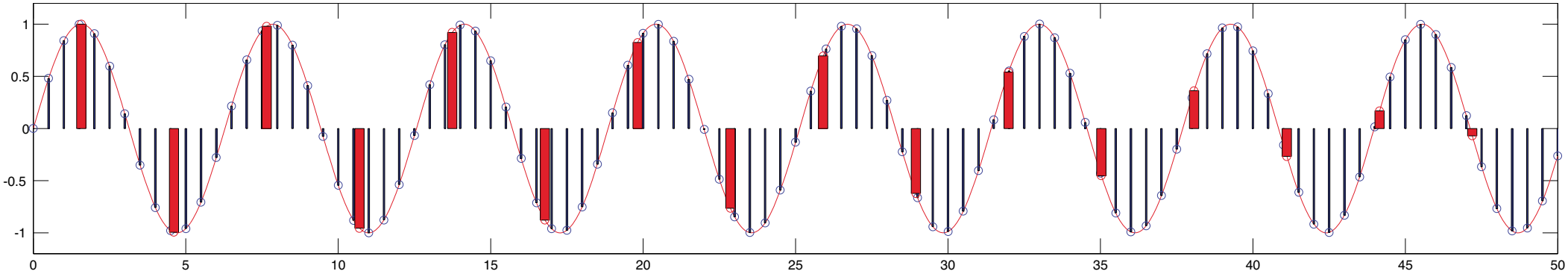


☞ An analog signal with bandwidth f_a must be sampled at:

$$f_s > 2f_a$$

☞ *Perfect* measurements taken at $f_s > 2f_a$ result in *no* information loss due to sampling.

Nyquist's Criterion



☞ An analog signal with bandwidth f_a must be sampled at:

$$f_s > 2f_a$$

☞ *Perfect* measurements taken at $f_s > 2f_a$ result in *no* information loss due to sampling.

☞ Due to actual (quantized) measurements: oversampling is required.

Quantization

A resolution of N bits provides 2^N possible discrete output levels:

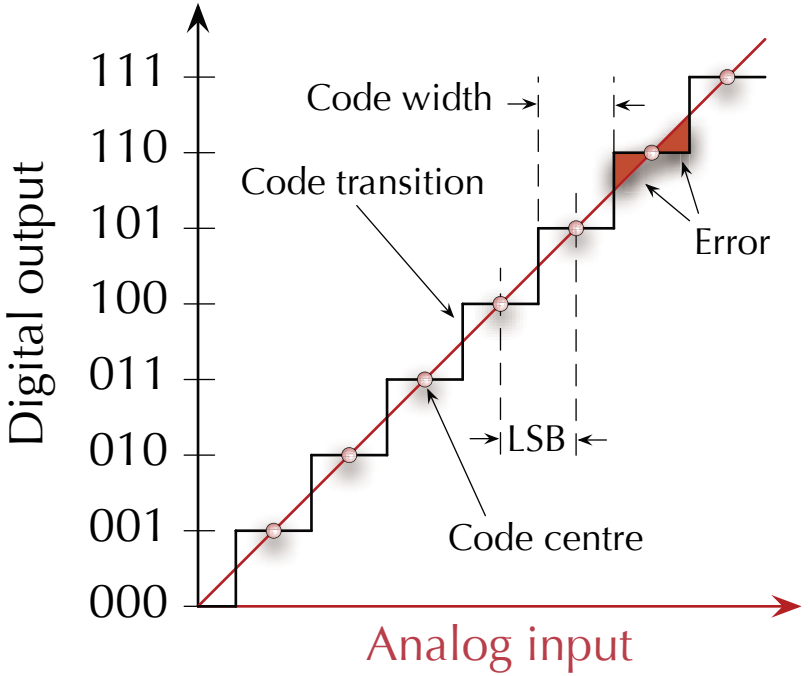
- Smallest distinguishable value q (**Least Significant Bit** or **LSB**):

$$q = \frac{1}{2^N}$$

- Ratio $1/q$ expressed in decibel (dB):

$$10 \log 2^{2N} = N \cdot 20 \log 2 \approx N \cdot 6.02 dB$$

(Decibel (dB) is a ration of powers defined as: $10 \log \frac{P_1}{P_0}$ or by signal amplitudes: $10 \log \frac{A_1^2}{A_0^2}$)



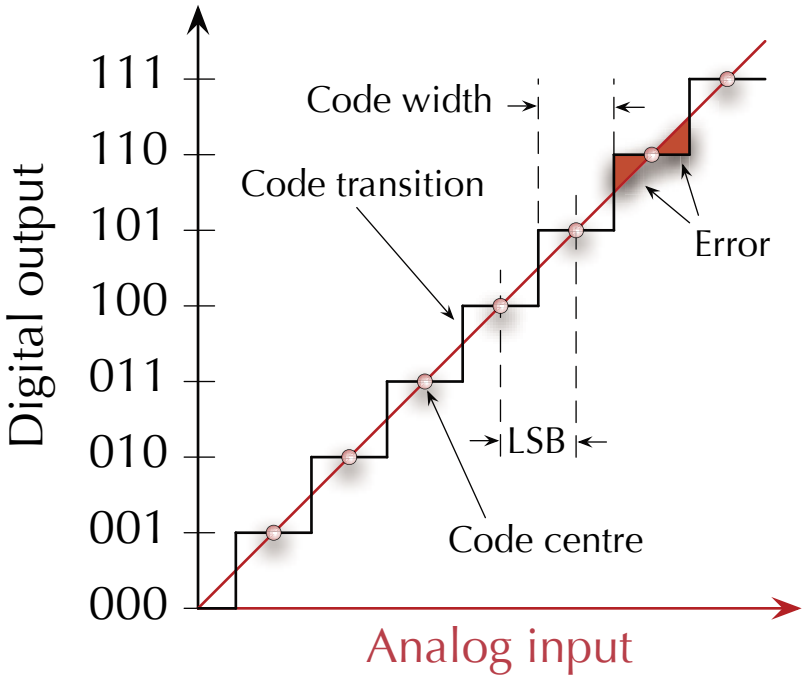
Quantization

The **mean square error** over one step:

$$\overline{E^2} = \frac{1}{q}\int_{-\frac{q}{2}}^{\frac{q}{2}} E^2 dE = \frac{q^2}{12}$$

Root mean square (rms) noise voltage:

$$\frac{q}{\sqrt{12}}$$



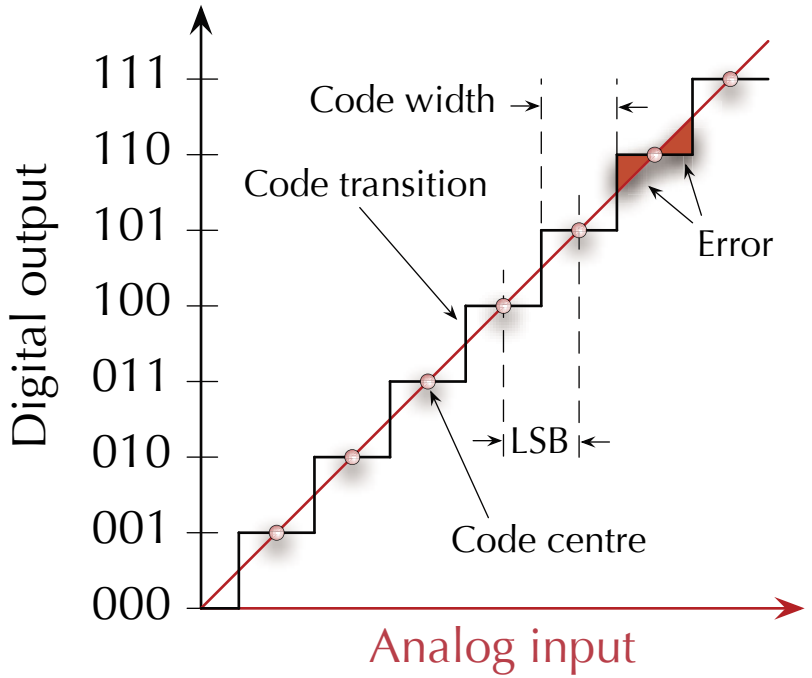
Quantization

The signal S with respect to the rms noise:

$$\frac{S}{\frac{q}{\sqrt{12}}} = S \cdot 2^N \sqrt{12}$$

or as the **signal to noise ratio** in decibel:

$$SNR[\text{dB}] = 10 \log\left(\frac{\overline{S^2}}{\frac{q^2}{12}}\right)$$



Quantization

Assuming an ideal input signal:

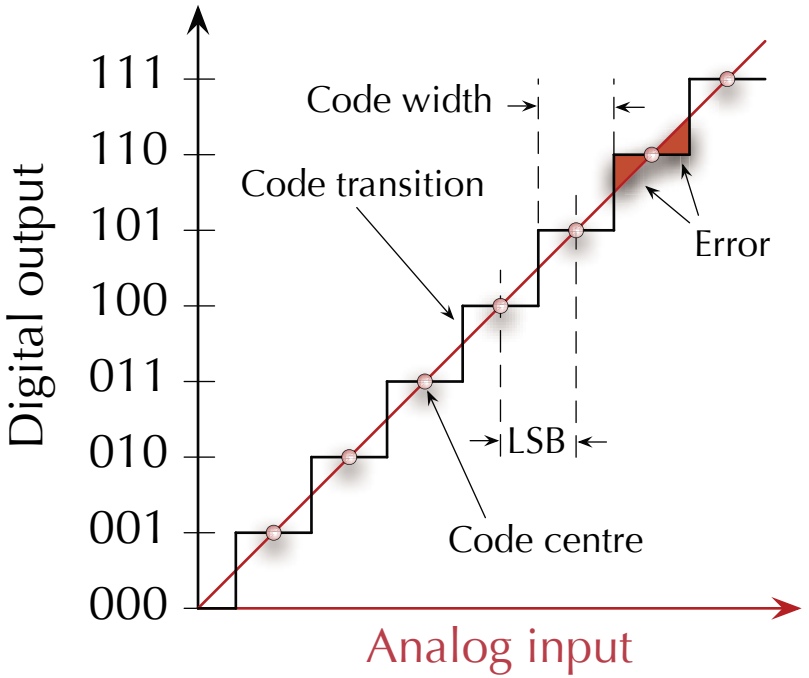
$$F(t) = A\sin\omega t \text{ with } q = \frac{2A}{2^N} \text{ and } \overline{S^2} = \frac{A^2}{2}$$

then the signal to noise ratio is:

$$SNR[\text{dB}] = 10\log\left(\frac{\overline{S^2}}{\frac{q^2}{12}}\right) = 10\log\left(\frac{3\cdot 2^{2N}}{2}\right)$$

$$SNR[\text{dB}] = 20N\log 2 + 10\log {}^3\!/_2$$

$$SNR[\text{dB}] \approx N\cdot 6.02 + 1.76$$



Quantization

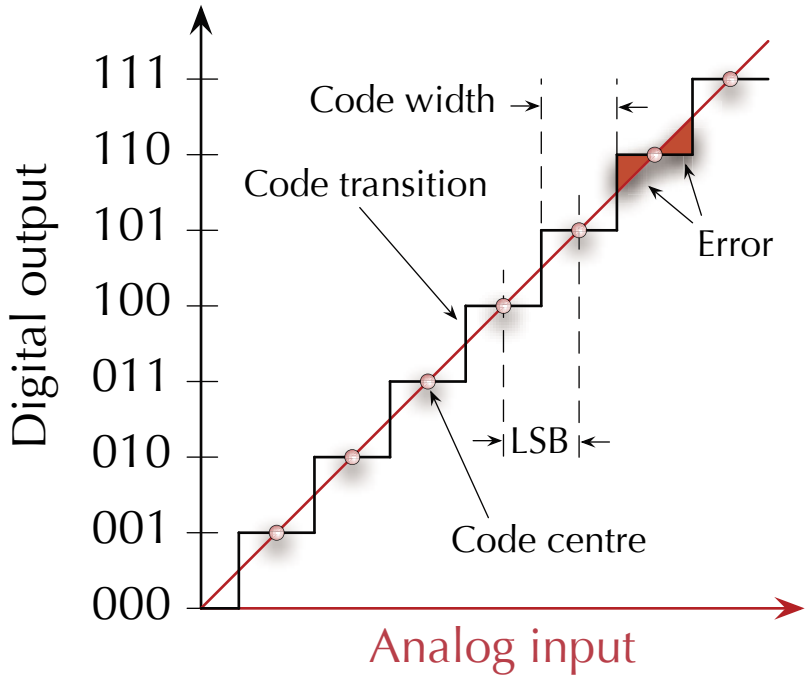
Determining the **effective number of bits** (**ENOB**):

$$SNR_{ideal}[\text{dB}] = 20N\log 2 + 10\log {}^{3}\!/_{2}$$

☞ $$ENOB = \frac{SNR_{actual} - 10\log {}^{3}\!/_{2}}{20\log 2}$$

☞ $$ENOB = \frac{SNR_{actual} - 1.76}{6.02}$$

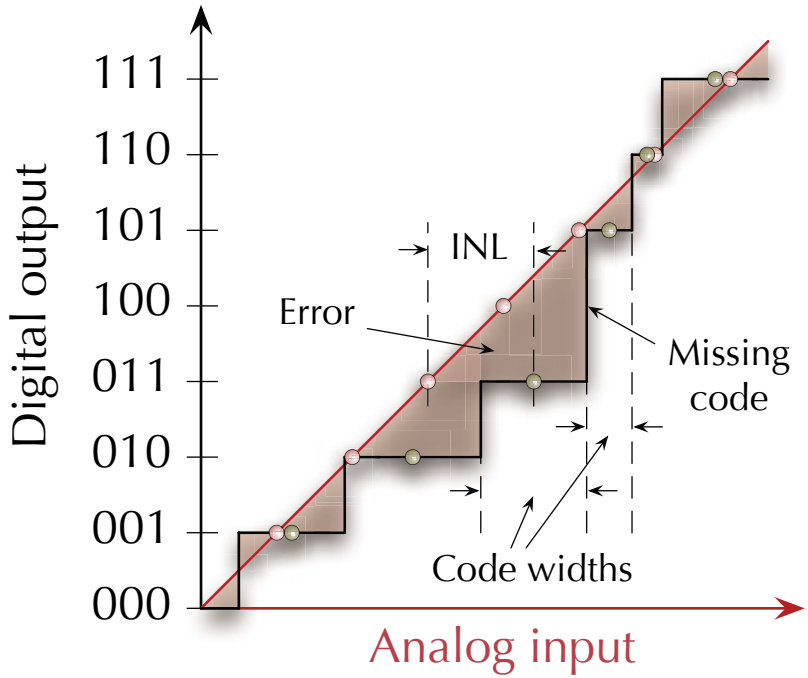
☞ $ENOB = N$ for $SNR_{actual} = SNR_{ideal}$



Quantization

Actual A/D converters are also characterized by:

- **Integral Non-Linearity** (**INL**): Maximal difference between the actual and ideal code centres.
- **Differential Non-Linearity** (**DNL**): Differences between successive code widths.
- **Missing codes**: reduce SNR by $20 \log 2$ or 6.02 dB for each missing code.
- **Response time / Latency**, **Throughput / Maximal sampling rate**



A/D converters

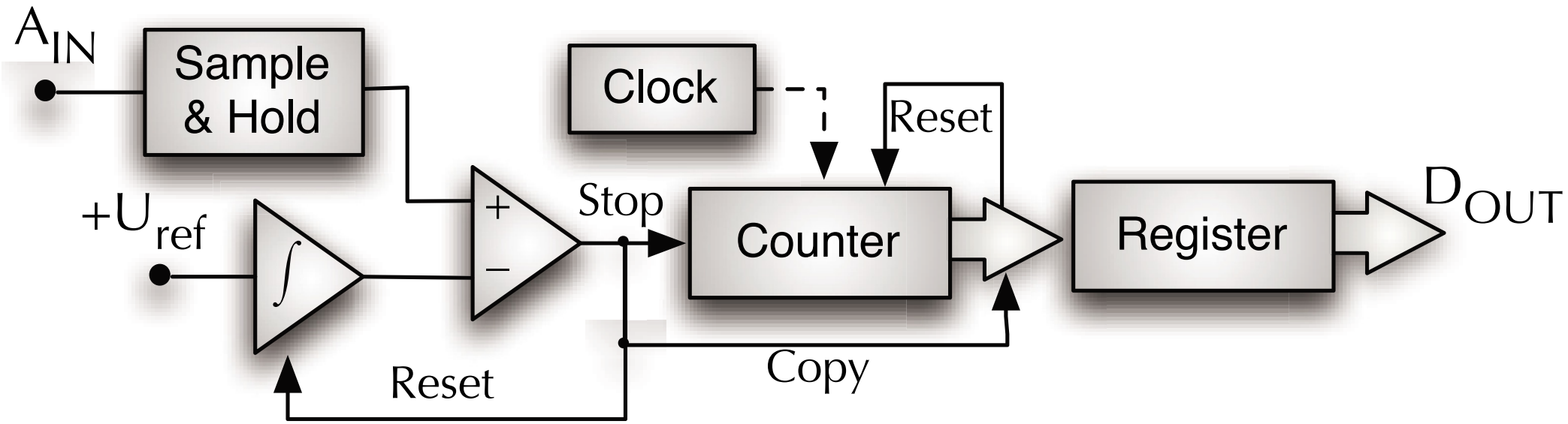
Some criteria to select the fitting A/D converter for an application:

- Throughput (maximal sampling frequency).
- Accuracy (ENOB, SNR).
- Latency (time from sensing to delivery).
- Power consumption.
- Complexity (also affects: price).

☞ Trade-offs are to be expected:

- Maximizing throughput will reduce accuracy and increase power consumption.
- Maximizing accuracy will reduce throughput and increase latency. (… other trade-offs)

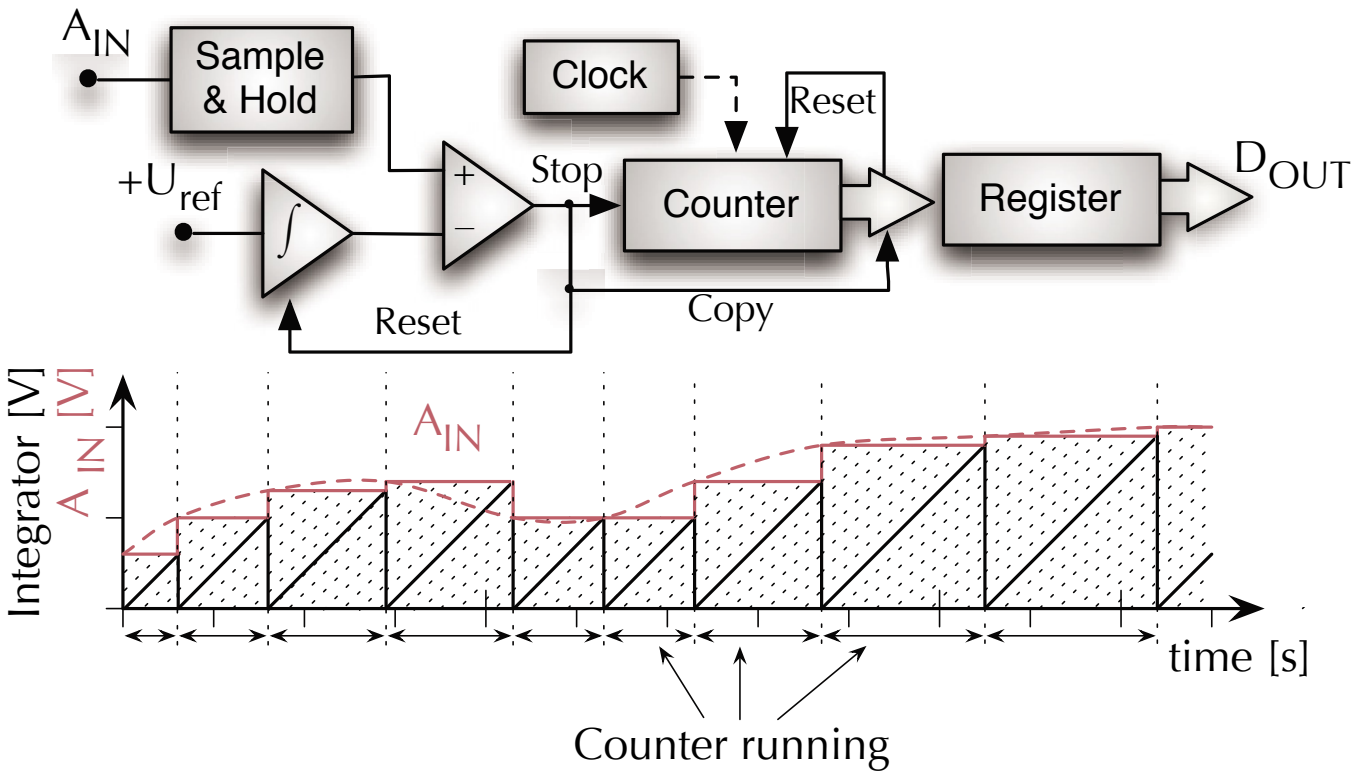
Integrating A/D converters (Single Slope)



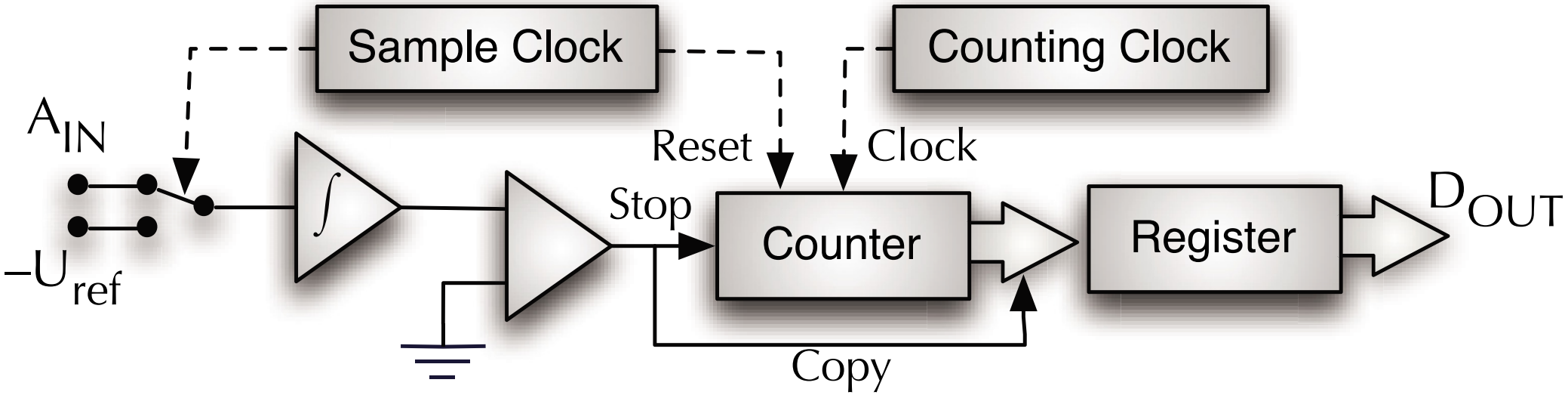
Integrating A/D converters (Single Slope)

Integrate a reference voltage U_{ref} until it matches the input voltage A_{IN}.

- **Sampling frequency** depends on input signal.
- **Accuracy** depends on U_{ref}, integrator and clock.
- **Simple** components.
- **Slow** (typically ~100 Hz)



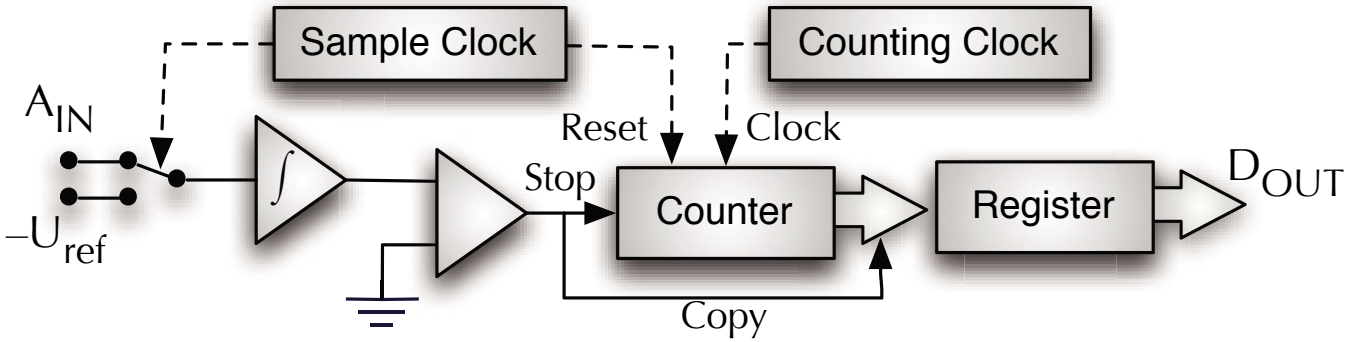
Integrating A/D converters (Dual Slope)

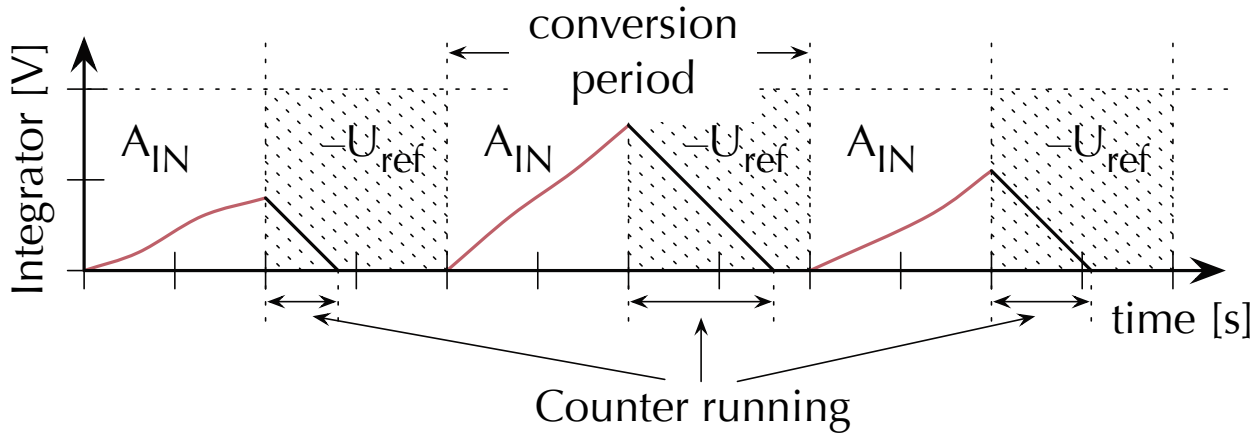


Integrating A/D converters (Dual Slope)

Input voltage A_{IN} is integrated for a constant time. The integrator is then discharged by a constant reference voltage U_{ref}. The discharge time is measured and is proportional to A_{IN}.

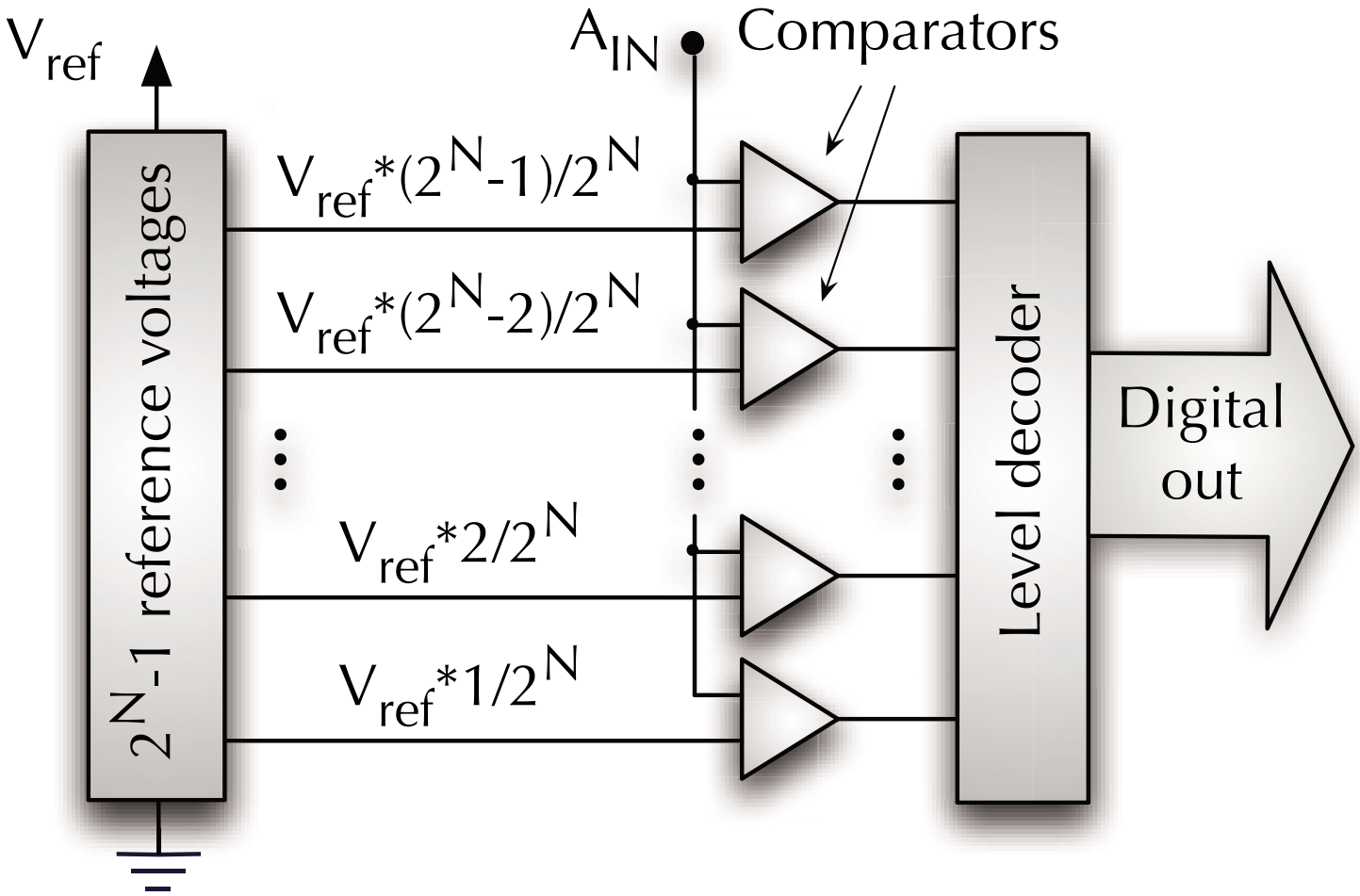
- Can **smooth** the input signal, and **suppress** specific frequencies.





A/D, D/A & Interfaces

Flash A/D converters

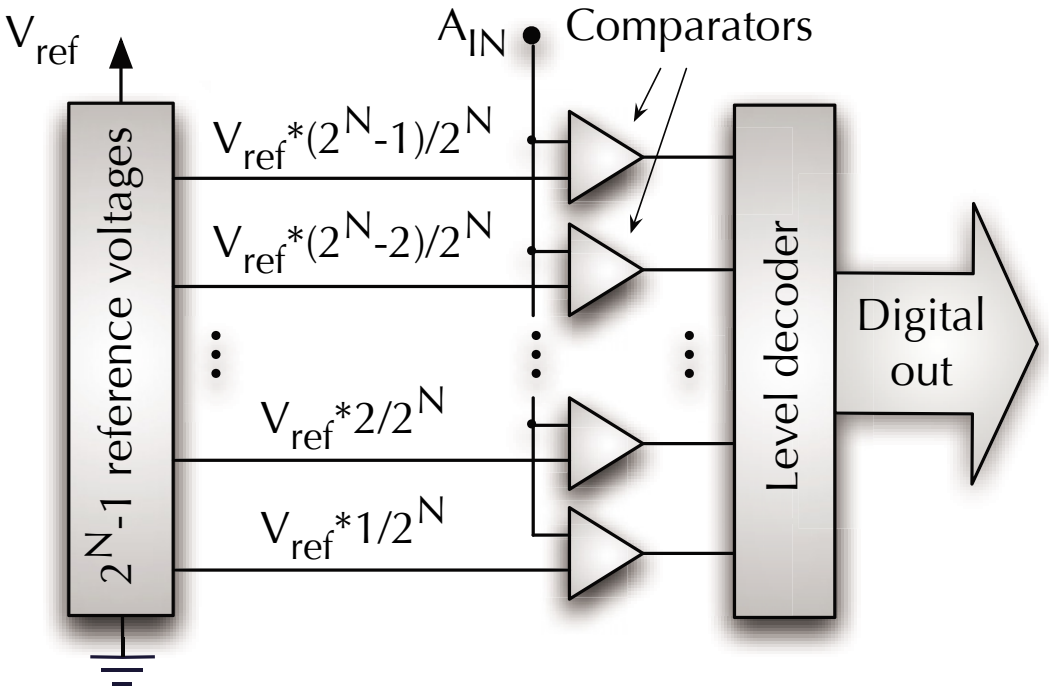


Flash A/D converters

$2^N - 1$ concurrent comparators identify the signal in one step.

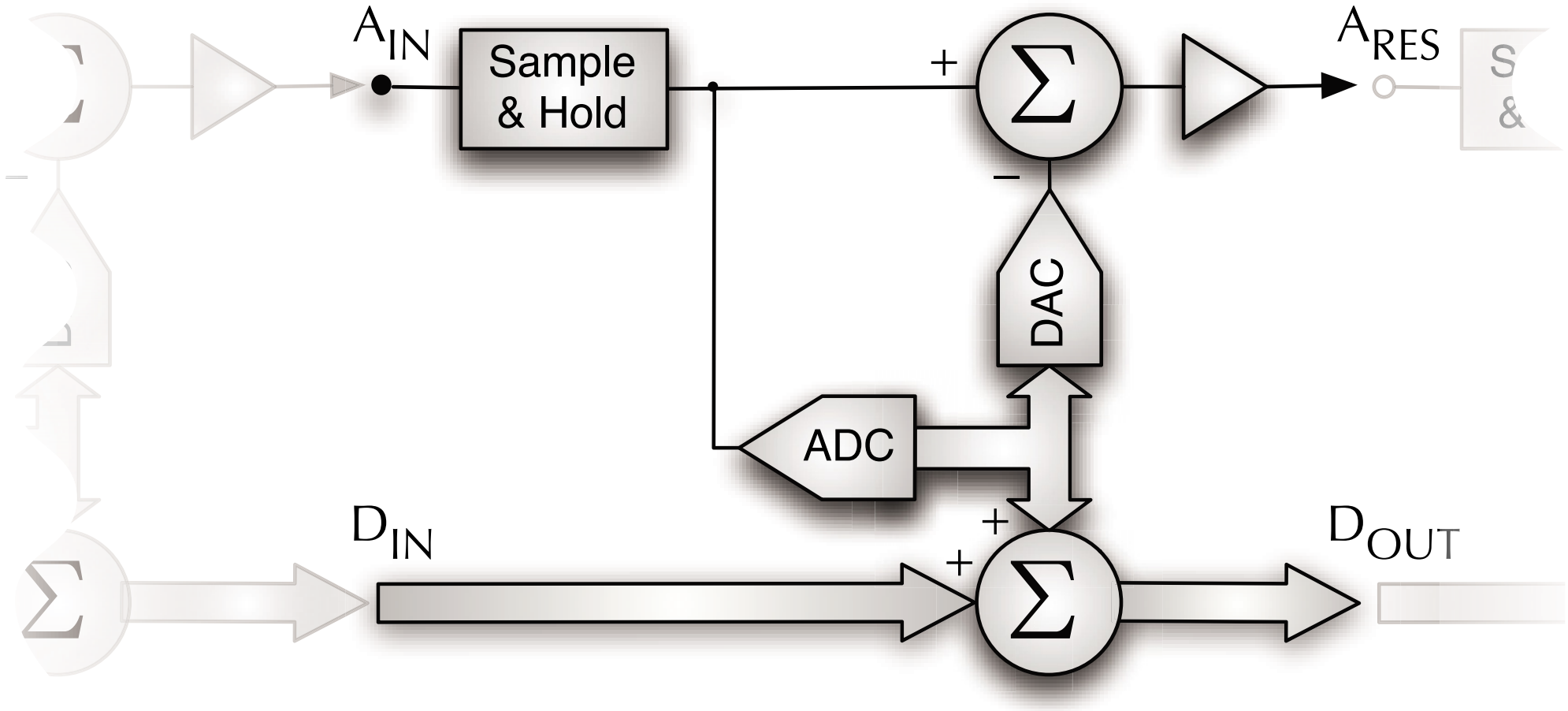
- **Fastest converter technology**: Single step conversion, minimal latency.
- **Complex for higher resolution**: Required circuitry scales with $2^N - 1$.
- **Accuracy** depends on the accuracy of the reference voltages.

☞ Typical applications: high speed, low resolution (e.g. video and radar converters).



A/D, D/A & Interfaces

Pipelined A/D converters

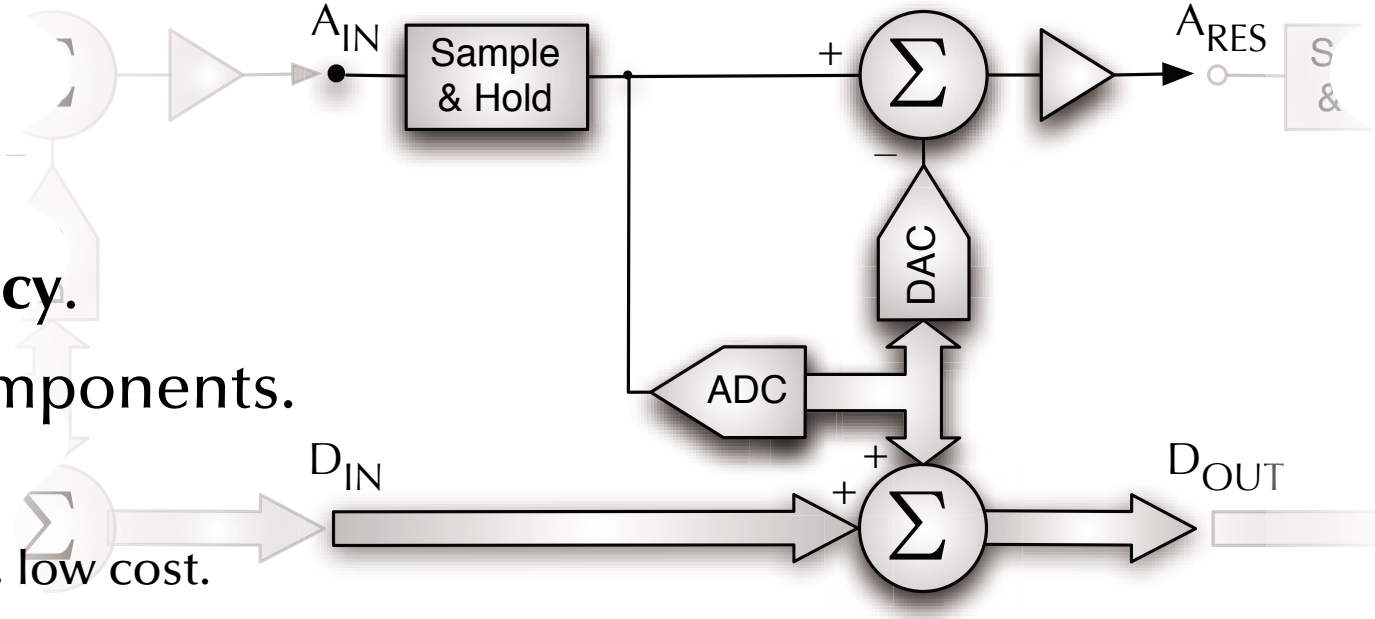


Pipelined A/D converters

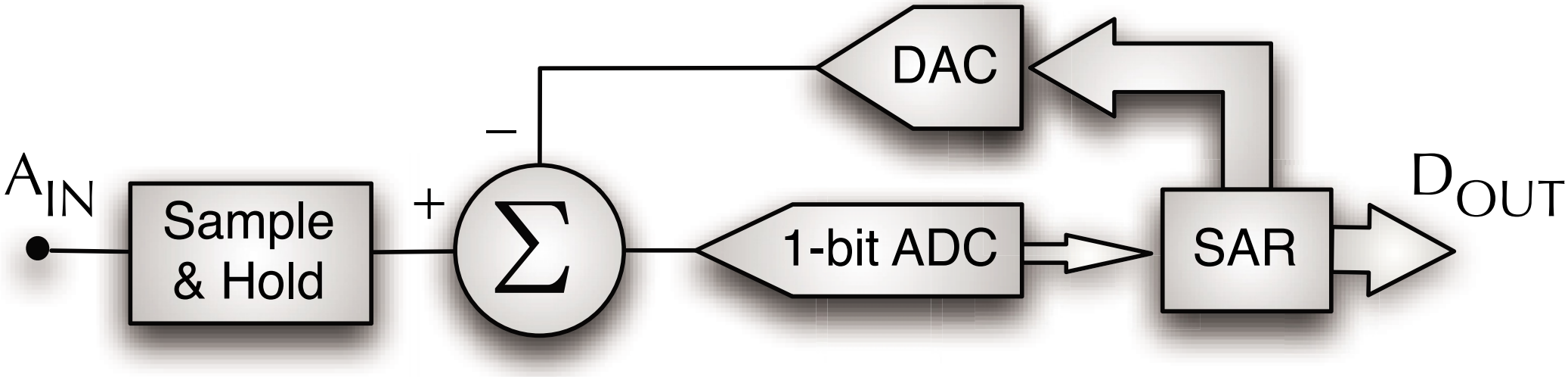
p pipeline stages with m bits each provide a pm bits converter. Each stage subtracts the analog value which has been converted (provides the rest as a residue value A_{RES}) and accumulates the digital output.

- **Keeps the throughput** (almost): All pipeline stages operate concurrently.
- **Trades resolution for latency**.
- **Accuracy** depends on components.

☞ Typical applications:
high resolution, high throughput, low cost.



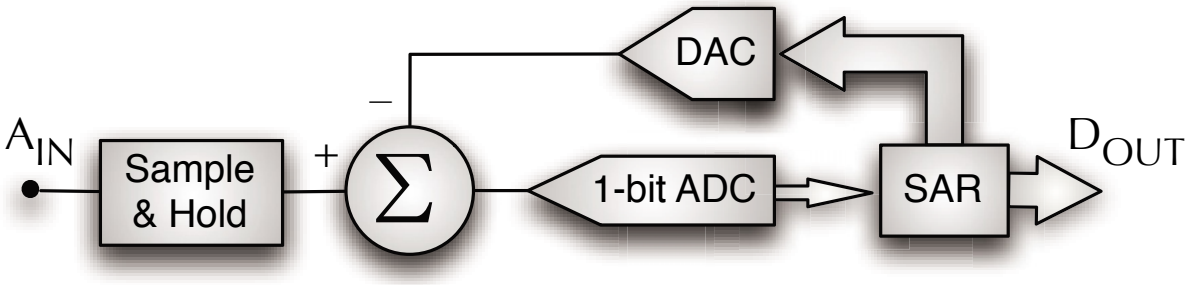
Successive Approximation Register (SAR) converters



Successive Approximation Register (SAR) converters

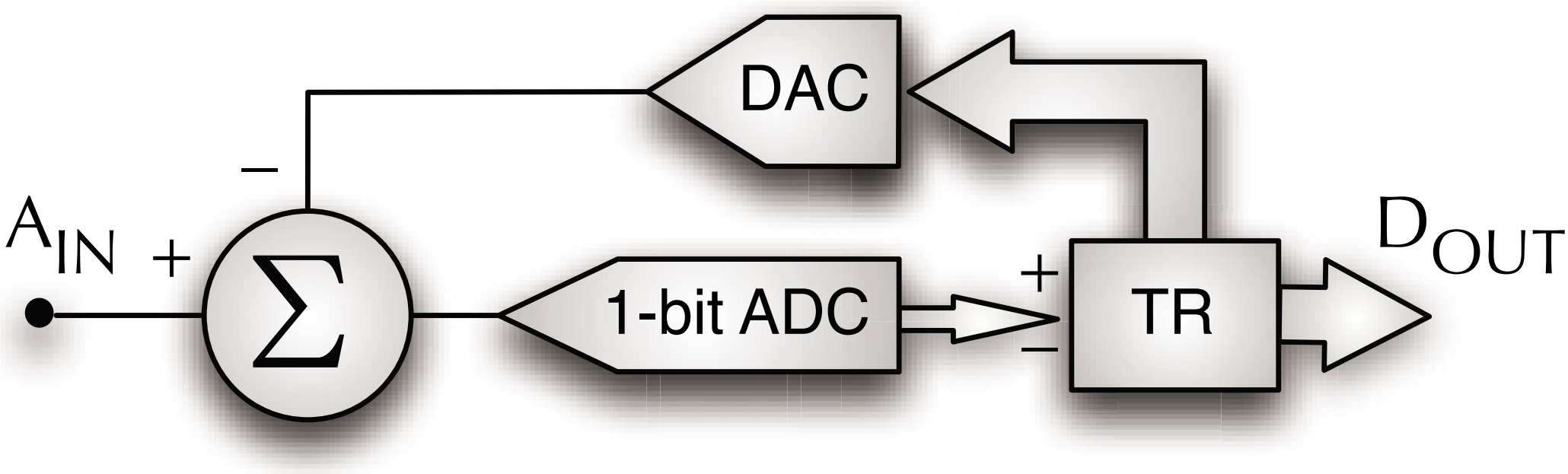
Single bit A/D converter converts one bit at a time, starting with the most significant bit, e.g. comparing to $1/2, 1/4, 1/8, \ldots, 1/2^N$ of full scale.

- **Minimal circuitry** – (almost) independent of resolution.
- Typically **slow**.
- **Accuracy** depends on the accuracy of the single bit ADC and the DAC.



☞ Typical applications: typically slow, low budget applications – yet can also be used in high accuracy applications.

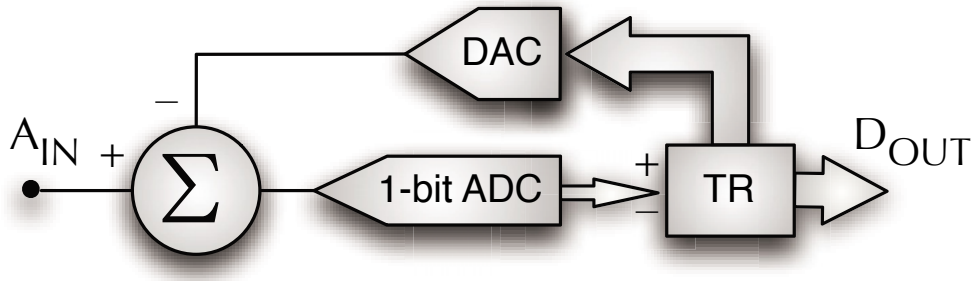
Tracking Register (TR) A/D converters



Tracking Register (TR) A/D converters

Continuous single bit conversion compares the current digital output with the analog input and counts a register up/down accordingly.

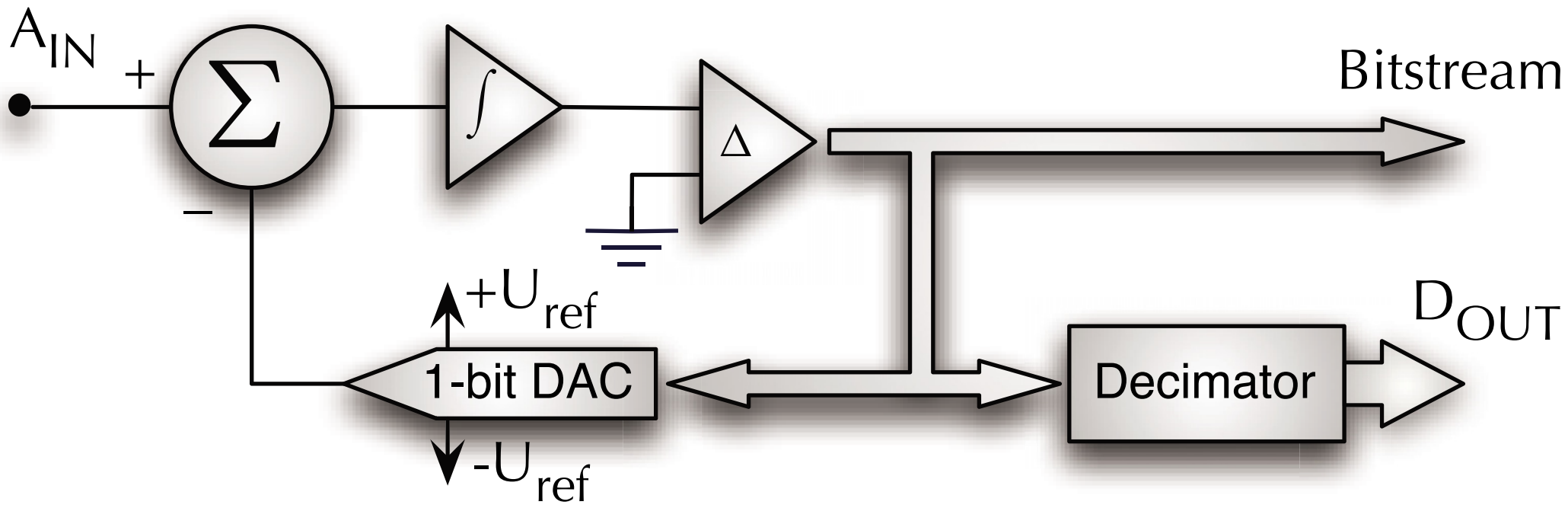
- **Minimal circuitry** (no S&H) – (almost) independent of resolution.
- **Speed** depends on amplitude changes in the input signal ☞ no constant sampling frequency.
- **Accuracy** depends on the accuracy of the single bit ADC and the DAC.



☞ Typical applications: Tracking slowly changing signals at high frequency. Rarely used today.

A/D, D/A & Interfaces

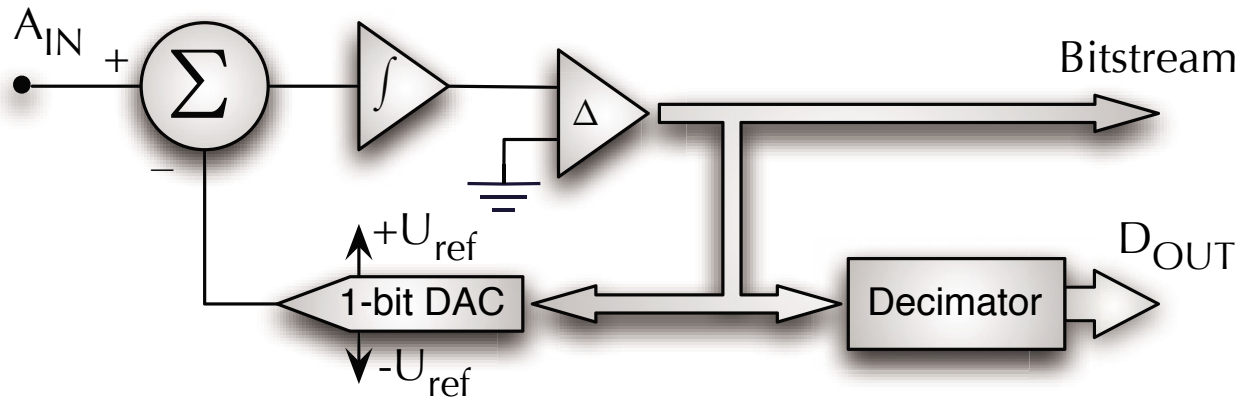
Σ-Δ A/D converters



Σ-Δ A/D converters

$+U_{ref}$ or $-U_{ref}$ is subtracted from the input signal and integrated.

The high frequent comparison of the integrator against ground results in a bitstream signal (which is also fed back).



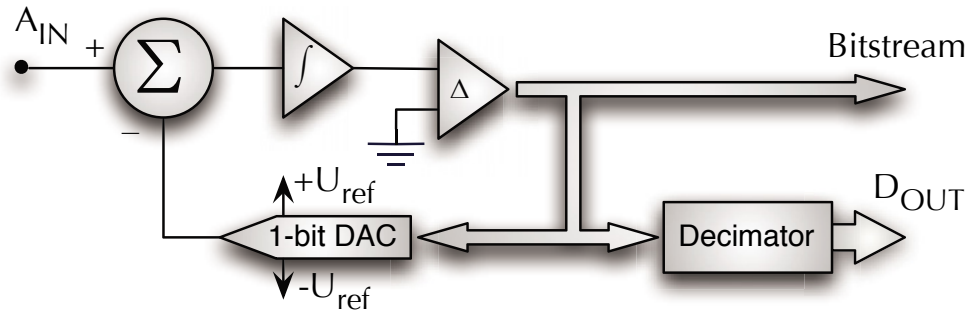
The density of '1's in the bitstream represents the input signal.

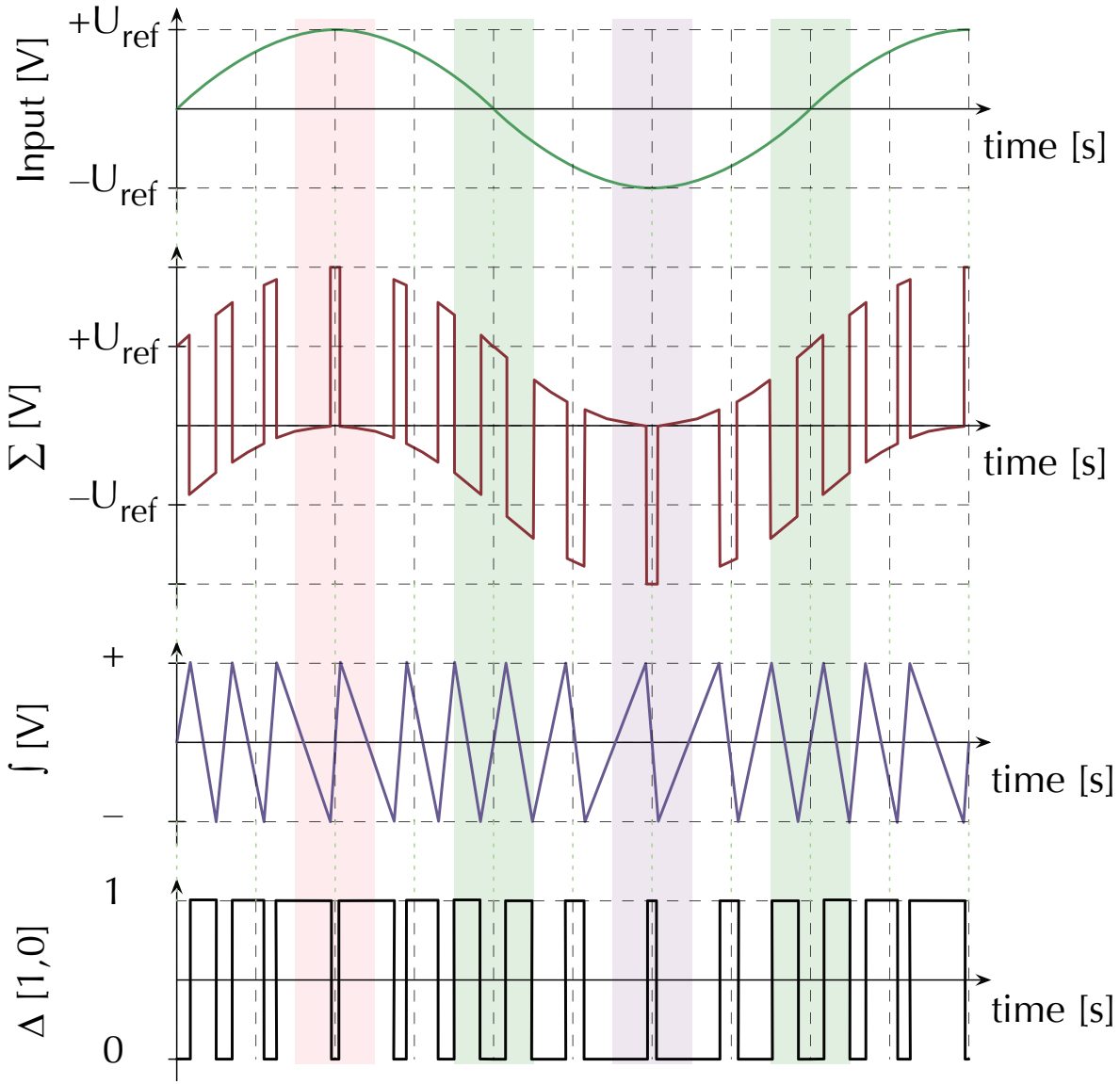
The bitstream can be deployed as such or be translated into digital words of varying lengths.

A/D, D/A & Interfaces

Σ-Δ A/D converters

(Voltages)

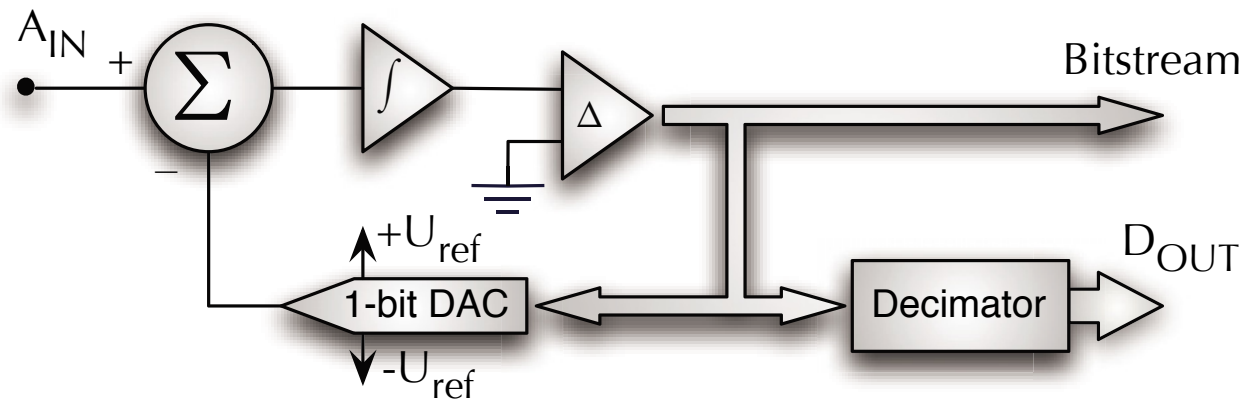




Σ-Δ A/D converters

The sampling frequency of the bitstream with respect to the digital output frequency (oversampling) determines accuracy.

- **Latency** depends on bitstream frequency.
- **Accuracy** depends on number of bits decimated.
- Method is inherently **linear**.

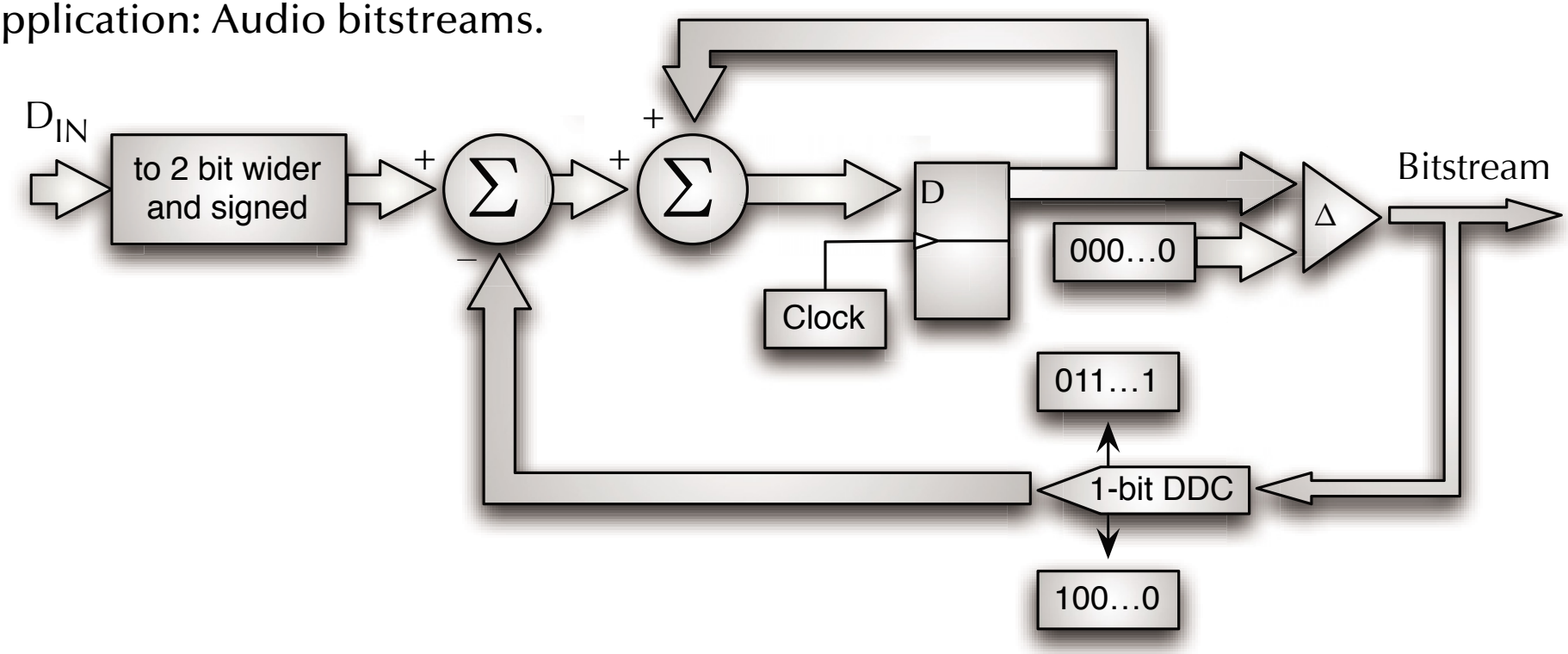


☞ Typical applications: High accuracy (typically 16-24 bits), moderate throughput (24 bit high quality audio converters are usually of this type).

Σ-Δ D/D converters

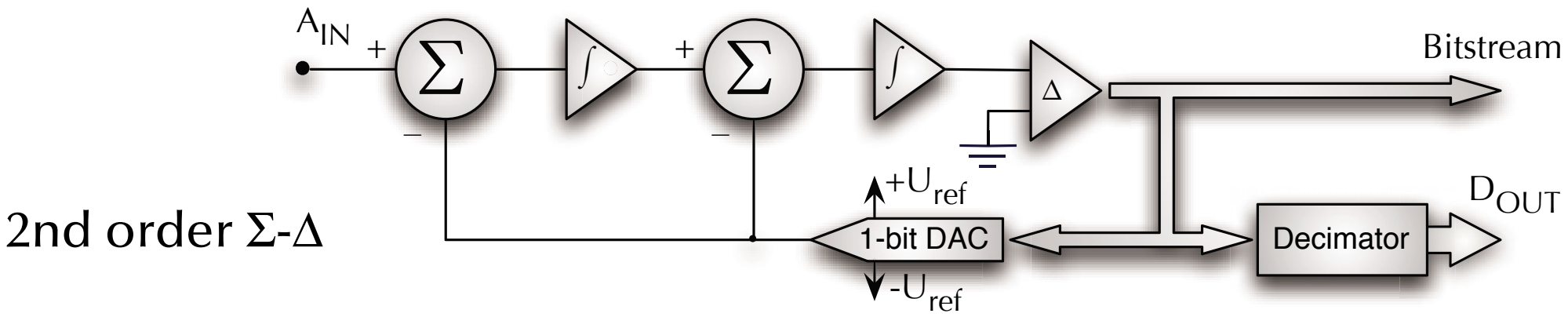
Digital form converts to bitstream or to analog.

☞ Typical application: Audio bitstreams.



Higher order Σ-Δ A/D converters

Effective Number Of Bits (Signal to Noise Ratio) can be improved by adding further integrator stages.

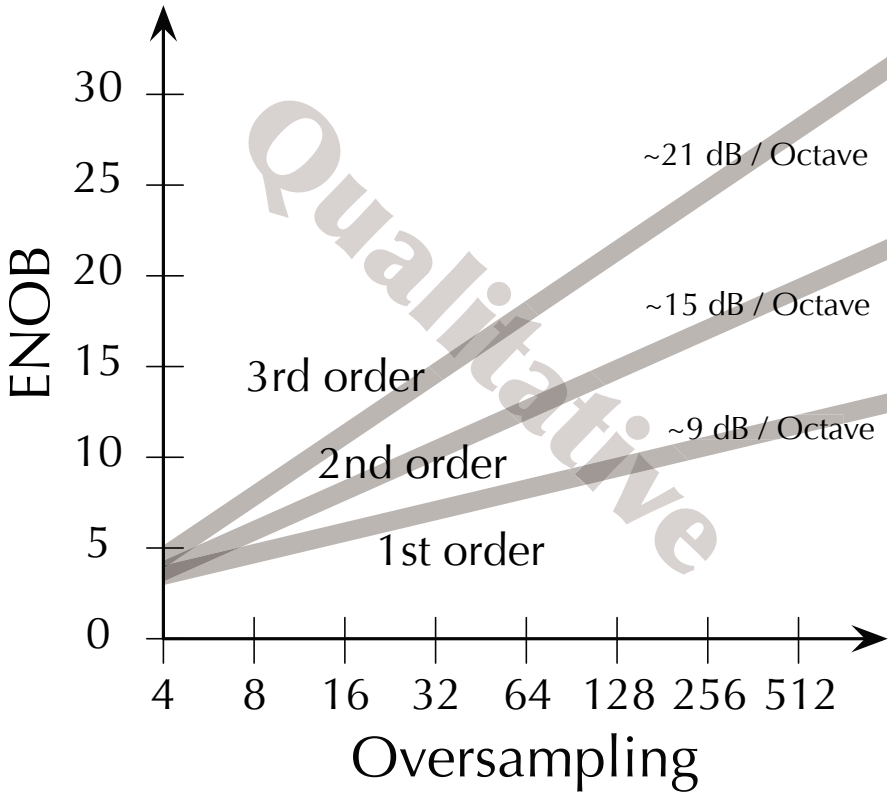


- Σ-Δ converters are not subject to aliasing, as they implicitly implement a low pass filer via the integrators.

Higher order Σ-Δ A/D converters

Dependency of the accuracy on the sampling frequency of the bitstream with respect to the output frequency (oversamping).

- ☞ Theoretical/qualitative result only – achievable accuracy depends on more factors ☞ datasheets.



A/D, D/A & Interfaces

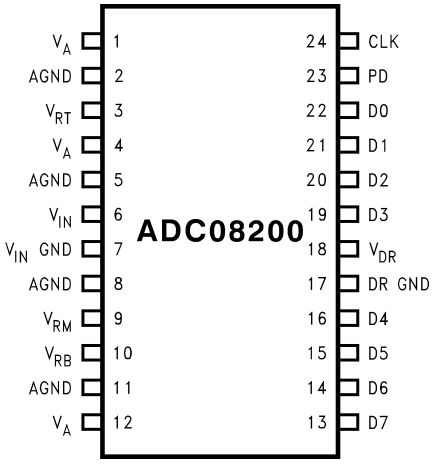
A/D converters matrix

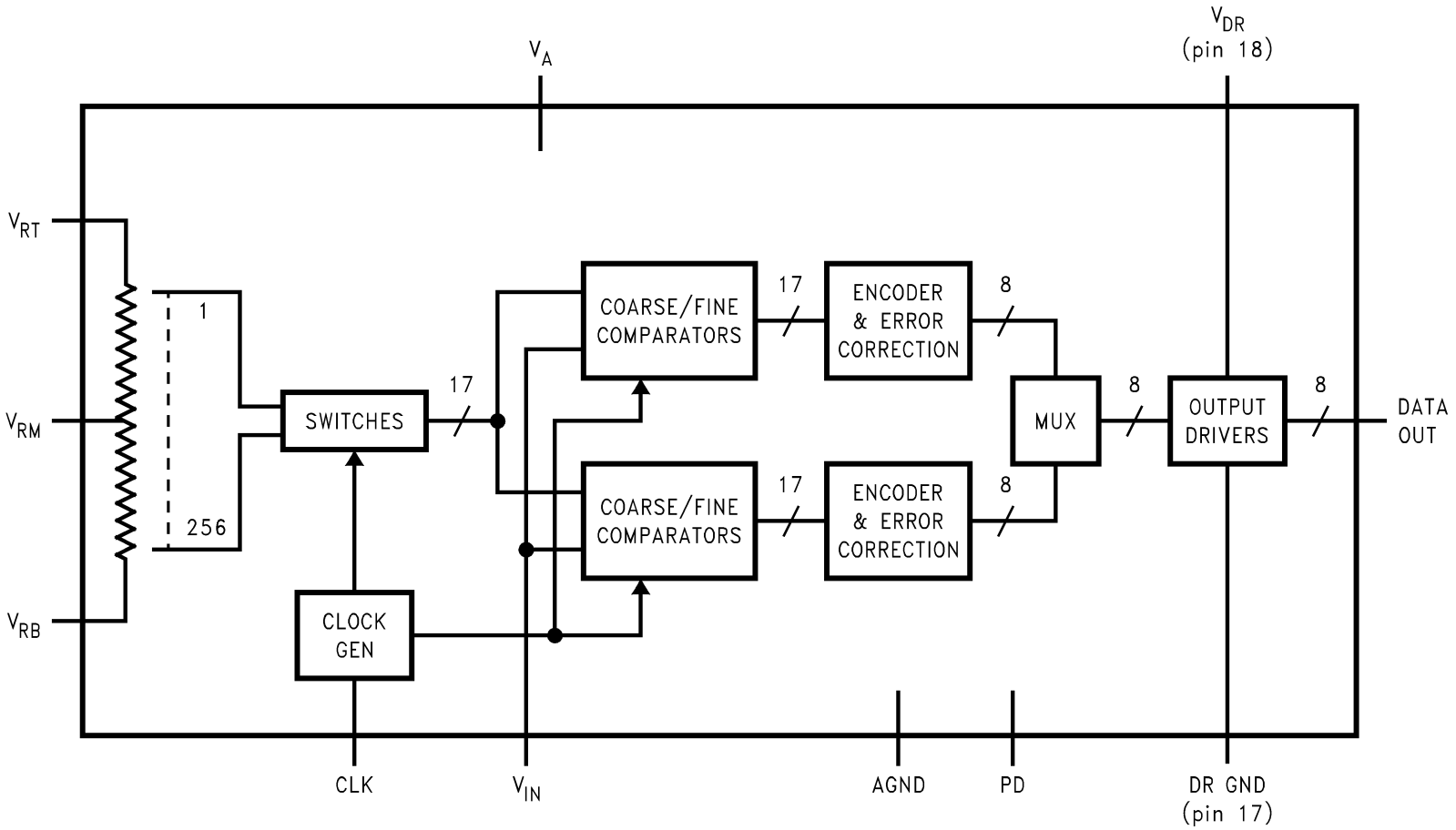
	Integrating	SAR	Σ-Δ	Pipeline Flash	Flash
Throughput	$O(1/2^N)$	$O(1/N)$	throughput/ latency vs. accuracy – typ. high accuracy	$O(1)$	$O(1)$
Latency	$O(2^N)$	$O(N)$		$O(p)$	$O(1)$
Accuracy	can be high	medium-high		typ. low-medium	typ. low
Resolution	typ. 8-16 bit	typ. 8-24 bit	typ. 16-24 bit	typ. 8-16 bit	typ. 4-8 bit
Size / Components	$O(1)$	$O(1)$	$O(1)$	$O(p \cdot 2^m)$	$O(2^N)$
Notes	Can suppress some frequencies	Cost efficient and can be very accurate	Flexible architecture, typ. very accurate	Compromise between speed and cost	Fastest converter

A/D, D/A & Interfaces: Examples

ADC 08200

(National Semiconductors)



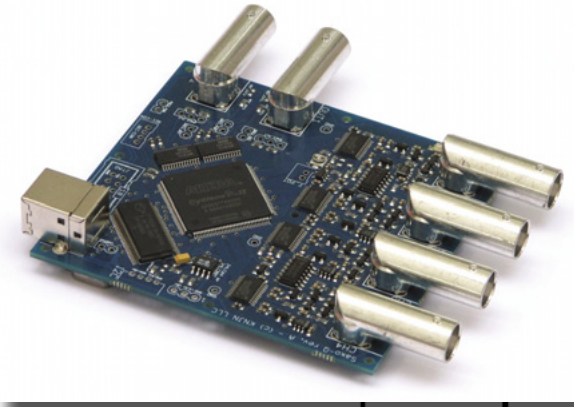


A/D, D/A & Interfaces: Examples

typ. application: video processing
– here: board by KNJN LLC, U.S.A.
featuring four ADC 08200

ADC 08200

(National Semiconductors)



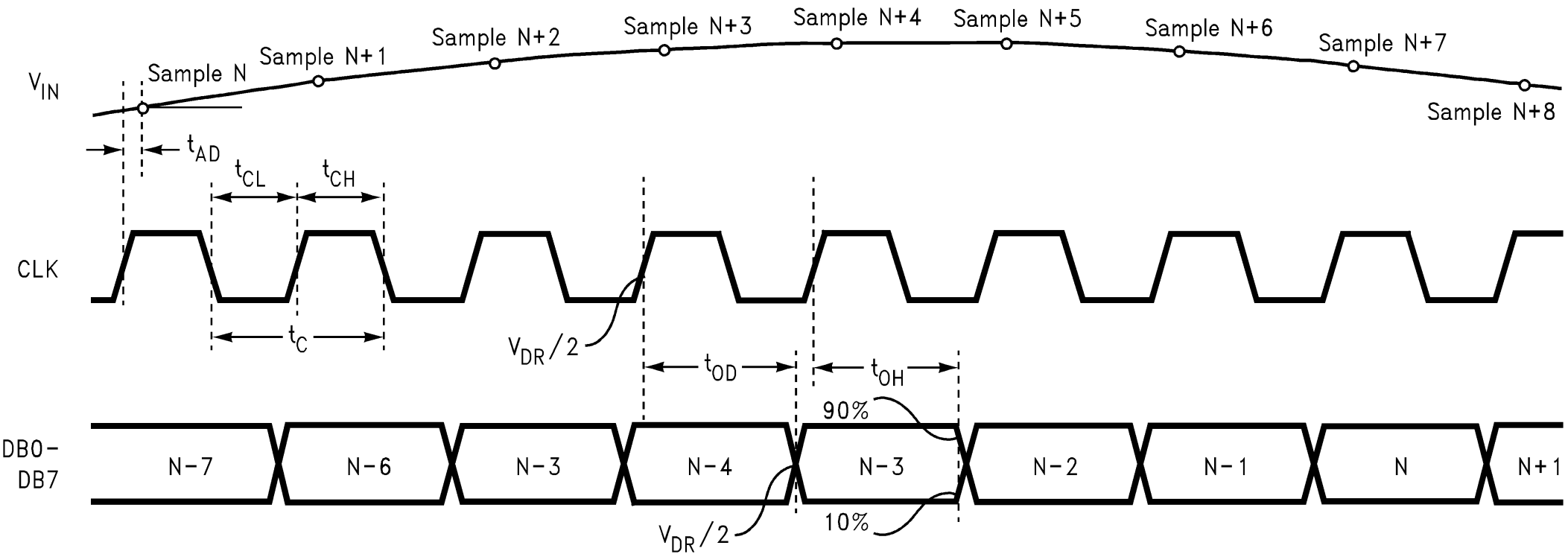
High speed
8 bit
converter with
2 flash stages
and
2 pipelines

ADC 08200 – Basic specifications

- **Resolution**: 8 bit
- **Sampling frequency**: 10 - 200 (230) MHz
- **Differential Non-Linearity** (DNL): ±0.4 LSB (typical), ±0.95 LSB (max.)
- **ENOB**: 7.5 (at 4 MHz), 7.3 (at 50 MHz), 7.0 (at 100 MHz)
- No **missing codes**
- **Power** consumption 1.05 mW/MSPS, 1 mW (power down)
- **Latency**: 6 cycles (pipeline delay)
- **Aperture** (sampling delay): 2.6 ns, with 2 ps rms jitter

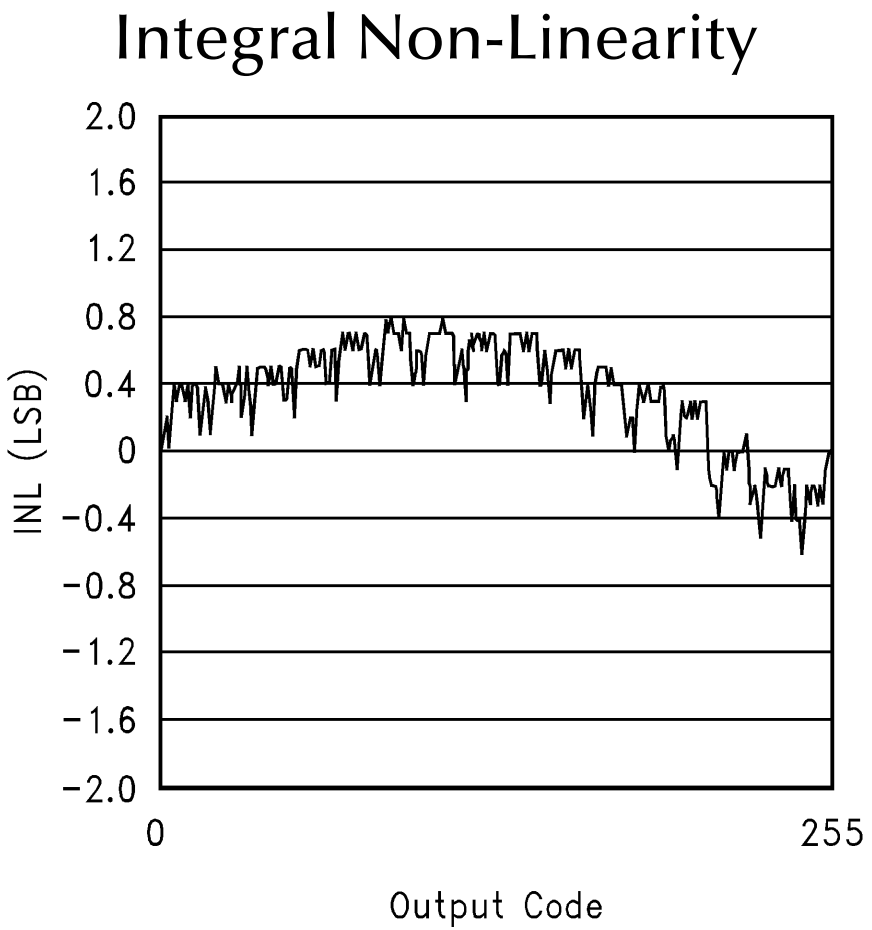
A/D, D/A & Interfaces: Examples

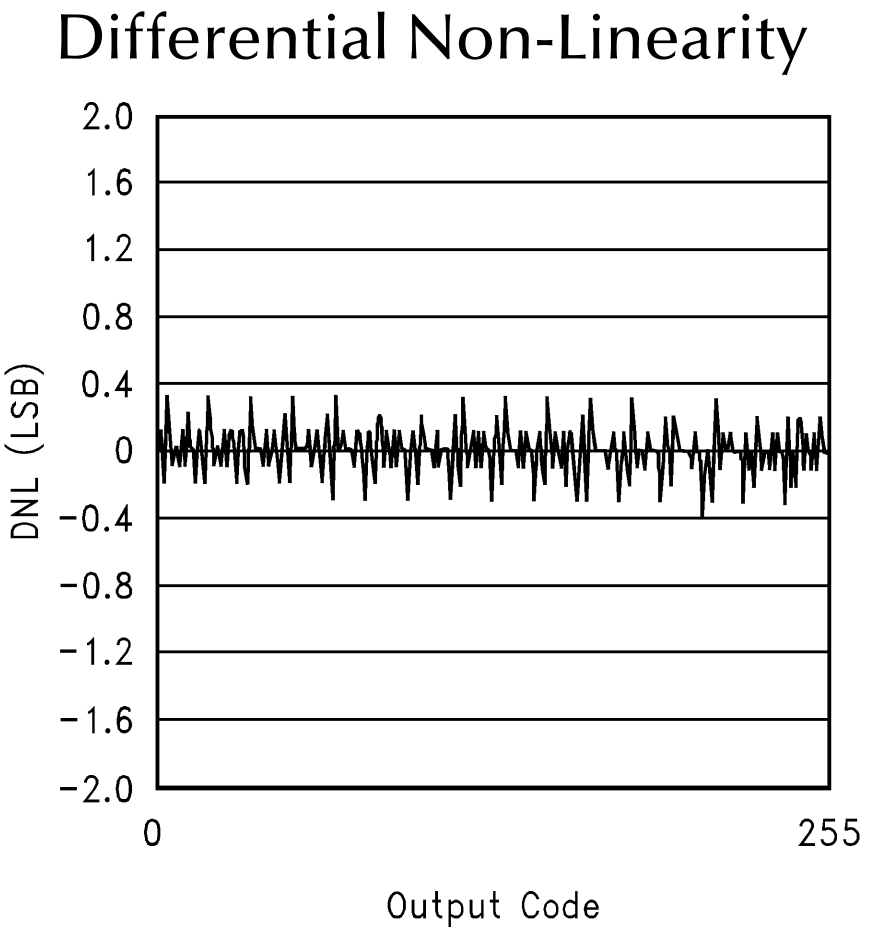
ADC 08200 – Timing



A/D, D/A & Interfaces: Examples

ADC 08200 – Non-Linearites

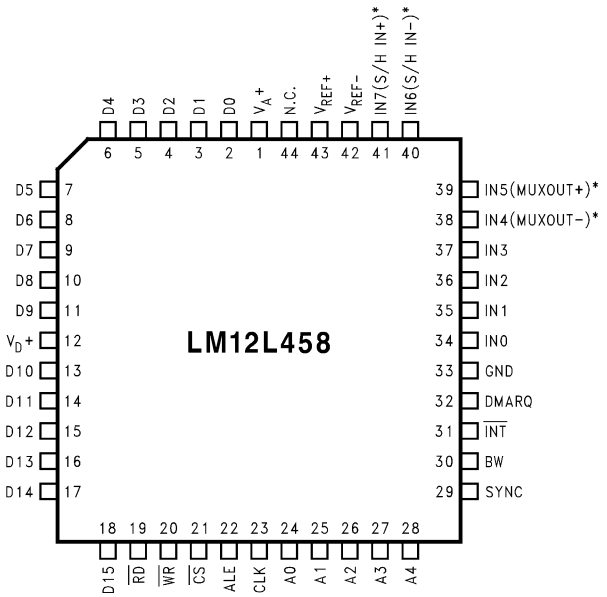


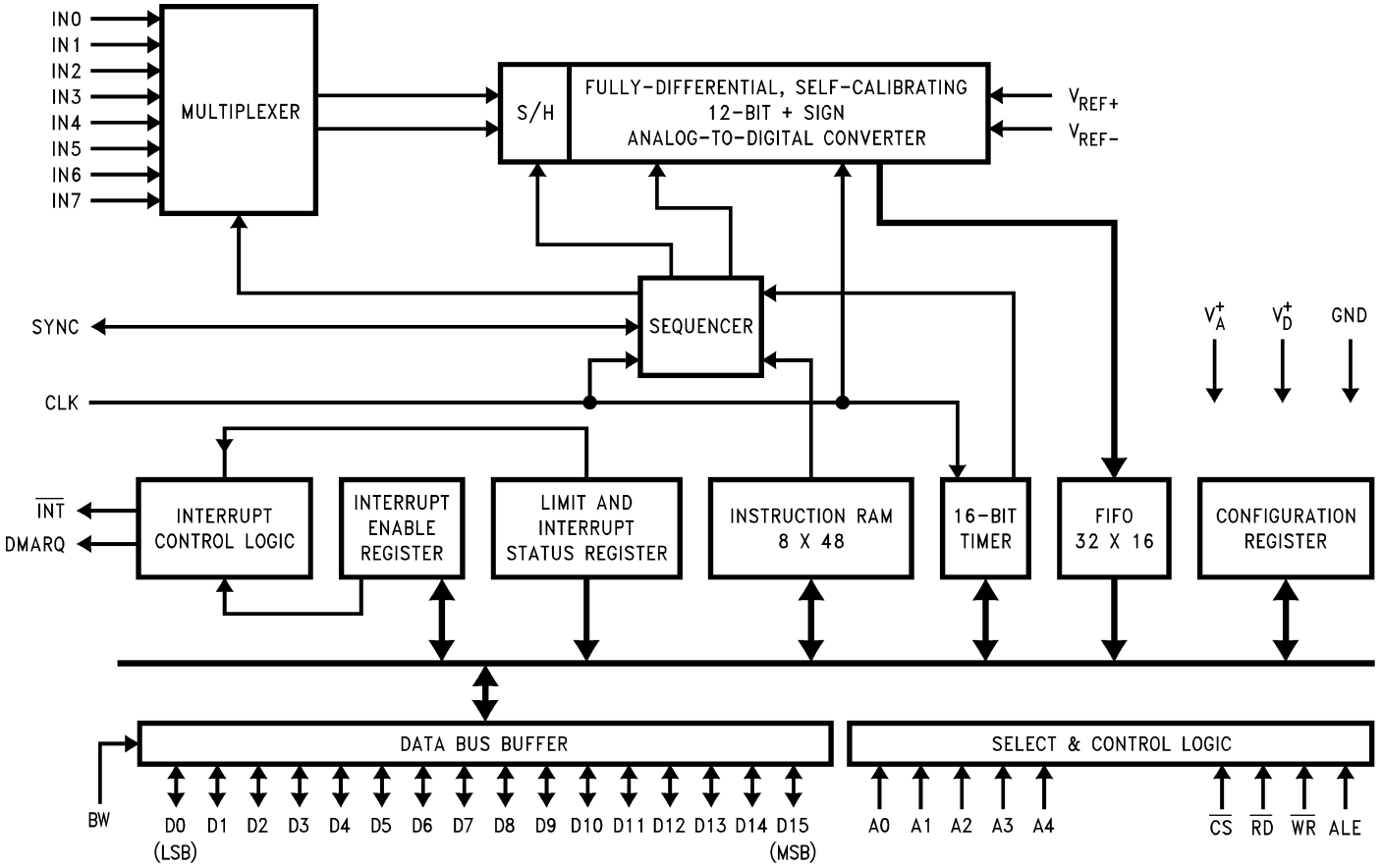


A/D, D/A & Interfaces: Examples

LM12L458

(National Semiconductors)





A/D, D/A & Interfaces: Examples

LM12L458 – Basic specifications

- **Channels:** 8 (multiplexed).
- **Resolution**: 8 bit + sign or 12 bit + sign (SAR converter).
- **Sampling frequency**: max. 106 kHz.
- **Power** consumption 15 mW; 6 µW (power down, clocked stopped).
- **Programmable** acquisition times, sequences and conversion rates.
- 32 word conversion **FIFO buffer**.
- **Self-calibration** and **diagnostic** mode.
- 8 or 16 bit wide **data bus**.

☞ Typical applications: Data logging, process control, low power devices.

LM12L458 – Register bank

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 00)	R/W	Acquisition Time				Watch-dog	8/$\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 01)	R/W	Don't Care						>/$\overline{<}$	Sign	Limit #1							
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 10)	R/W	Don't Care						>/$\overline{<}$	Sign	Limit #2							
1	0	0	0	Configuration Register	R/W	Don't Care				DIAG	Test = 0	RAM Pointer		I/O Sel	Auto $Zero_{ec}$	Chan Mask	Stand-by	Full CAL	Auto-Zero	Reset	Start
1	0	0	1	Interrupt Enable Register	R/W	Number of Conversions in Conversion FIFO to Generate INT2					Sequencer Address to Generate INT1			INT7	Don't Care	INT5	INT4	INT3	INT2	INT1	INT0
1	0	1	0	Interrupt Status Register	R	Actual Number of Conversion Results in Conversion FIFO					Address of Sequencer Instruction being Executed			INST7	"0"	INST5	INST4	INST3	INST2	INST1	INST0
1	0	1	1	Timer Register	R/W	Timer Preset High Byte								Timer Preset Low Byte							
1	1	0	0	Conversion FIFO	R	Address or Sign			Sign	Conversion Data: MSBs				Conversion Data: LSBs							
1	1	0	1	Limit Status Register	R	Limit #2: Status								Limit #1: Status							

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 00)	R/W	Acquisition Time				Watch-dog	$8/\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 01)	R/W	Don't Care						$>/\overline{<}$	Sign	Limit #1							
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 10)	R/W	Don't Care						$>/\overline{<}$	Sign	Limit #2							

Instruction RAM entries consist of:

- **Loop** (1 bit): indicates the last instruction and branches to the first one.
- **Pause** (1 bit): halts the sequencer before this instruction.
- V_{IN+}, V_{IN-} ($2 \cdot 3$ bits): select the input channels ('000' selects ground in V_{IN-}).
- **Sync** (1 bit): wait for an external sync. signal before this instruction.
- **Timer** (1 bit): wait for a preset 16-bit counter delay before this instruction.

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 00)	R/W	Acquisition Time				Watch-dog	$8/\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 01)	R/W	Don't Care						>/$\overline{<}$	Sign	Limit #1							
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 10)	R/W	Don't Care						>/$\overline{<}$	Sign	Limit #2							

Instruction RAM entries consist of (cont.):

- $8/\overline{12}$ (1 bit): selects the resolution (8 bit + sign or 12 bit + sign).
- **Watchdog** (1 bit): activates comparisons with two programmed limits.
- **Acquisition time** D (4 bits): the converter takes $9 + 2D$ cycles (12 bit mode) or $2 + 2D$ cycles (8 bit mode) to sample the input. Reasonable times depend on the input resistance and clock frequency: $D \approx 0.45 \cdot R_S[\text{k}\Omega] \cdot f_{CLK}[\text{MHz}]$ for 12 bit conversions.
- Limits ($2 \cdot 9$ bits, including sign and comparator): trigger levels for watchdog operation.

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0 to 1	0 to 1	0 to 1	Instruction RAM (RAM Pointer = 00)	R/W	Acquisition Time				Watch-dog	8/$\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop

```
type ChannelPlus  is (Ch0, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7);
type ChannelMinus is (Gnd, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7);
type Resolutions  is (TwelveBit, EightBit);
type Aquisition_D is new Natural range 0..15; -- 9+2D (12bit), 2+2D (8bit)

for ChannelPlus  use (Ch0 => 0, Ch1 => 1, Ch2 => 2, Ch3 => 3,
                      Ch4 => 4, Ch5 => 5, Ch6 => 6, Ch7 => 7);

for ChannelMinus use (Gnd => 0, Ch1 => 1, Ch2 => 2, Ch3 => 3,
                      Ch4 => 4, Ch5 => 5, Ch6 => 6, Ch7 => 7);

for Resolutions  use (TwelveBit => 0, EightBit => 1);

type Instruction is record
        EndOfLoop, Pause, Sync, Timer, Watchdog : Boolean;
        Vplus                                   : ChannelPlus;
        Vminus                                  : ChannelMinus;
        Resolution                              : Resolutions;
        AquisitionTime                          : Aquisition_D;
     end record;
```

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-				V_{IN-}			V_{IN+}				
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync							Pause	Loop
	1	1	1																		

```
Units_Per_Word : constant Integer := Word_Size / Storage_Unit;
for Instruction use record
      EndOfLoop      at 0*Units_Per_Word range  0.. 0;
      Pause          at 0*Units_Per_Word range  1.. 1;
      Vplus          at 0*Units_Per_Word range  2.. 4;
      Vminus         at 0*Units_Per_Word range  5.. 7;
      Sync           at 0*Units_Per_Word range  8.. 8;
      Timer          at 0*Units_Per_Word range  9.. 9;
      Resolution     at 0*Units_Per_Word range 10..10;
      Watchdog       at 0*Units_Per_Word range 11..11;
      AquisitionTime at 0*Units_Per_Word range 12..15;
   end record;
```

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-											
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop
	1	1	1																		

```
for Instruction'Size      use 16; -- Bits
for Instruction'Alignment use 2;  -- Storage_Units (Bytes)
for Instruction'Bit_Order use High_Order_First;

type Instructions is array (0..7) of Instruction;
   pragma Pack (Instructions);

ADC_Instructions : Instructions;

for ADC_Instructions'Address use To_Address (16#0000132D#);
```

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-											
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop

```
ADC_Instructions (0) := (EndOfLoop      => False,
                         Pause          => False,
                         Vplus          => Ch0,
                         Vminus         => Gnd,
                         Sync           => True,
                         Timer          => False,
                         Resolution     => EightBit,
                         Watchdog       => False,
                         AquisitionTime => 10);
ADC_Instructions (1) := (EndOfLoop      => True,  -- last instruction
                         Pause          => False,
                         Vplus          => Ch1,
                         Vminus         => Ch2,
                         Sync           => False,
                         Timer          => False,
                         Resolution     => TwelveBit,
                         Watchdog       => False,
                         AquisitionTime => 0);
```

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-											
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop

```
ADC_Instructions (0) := (EndOfLoop       => False,
                         Pause           => False,
                         Vplus           => Ch0,
                         Vminus          => Gnd,
                         Sync            => False,
                         Timer           => False,
                         Resolution      => EightBit,
                         Watchdog        => False,
                         AquisitionTime  => 10);
ADC_Instructions (1) := (EndOfLoop       => True,  -- last instruction
                         Pause           => False,
                         Vplus           => Ch1,
                         Vminus          => Ch2,
                         Sync            => False,
                         Timer           => False,
                         Resolution      => TwelveBit,
                         Watchdog        => False,
                         AquisitionTime  => 0);
```

Those assignments to the actual controller's registers will program the attached external controller (LM12L458) immediately.

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-				V_{IN-}			V_{IN+}				
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync							Pause	Loop
	1	1	1																		

Data structures in 'C':

```
enum ChannelPlus  {Ch0=0, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7};
enum ChannelMinus {Gnd=0, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7};
enum Resolutions  {TwelveBit=0, EightBit};

struct {
   unsigned int EndOfLoop      : 1;
   unsigned int Pause          : 1;
   ChannelPlus  Vplus          : 3;
   ChannelMinus Vminus         : 3;
   unsigned int Sync           : 1;
   unsigned int Timer          : 1;
   Resolutions  Resolution     : 1;
   unsigned int Watchdog       : 1;
   unsigned int AquisitionTime : 4;
} Instruction;
```

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-				V_{IN-}			V_{IN+}				
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync							Pause	Loop
	1	1	1																		

Data structures in 'C':

```
Instruction    InstructionsA[8];
InstructionsA *Instructions;
Instructions = 0x0000132D;

*Instructions (0).EndOfLoop      = 0;
*Instructions (0).Pause          = 0;
*Instructions (0).Vplus          = Ch0;
*Instructions (0).Vminus         = Gnd;
*Instructions (0).Sync           = 1;
*Instructions (0).Timer          = 0;
*Instructions (0).Resolution     = EightBit;
*Instructions (0).Watchdog       = 0;
*Instructions (0).AquisitionTime = 10;
```

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-				V_{IN-}			V_{IN+}				
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync							Pause	Loop
	1	1	1																		

Data structures in 'C':

```
Instruction   InstructionsA[8];
InstructionsA *Instructions;
Instructions = 0x0000132D;

*Instructions (0).EndOfLoop          = 0;
*Instructions (0).Pause              = 0;
*Instructions (0).Vplus              = Ch0;
*Instructions (0).Vminus             = Gnd;
*Instructions (0).Sync               = 1;
*Instructions (0).Timer              = 0;
*Instructions (0).Resolution         = EightBit;
*Instructions (0).Watchdog           = 0;
*Instructions (0).AquisitionTime     = 10;
```

Not portable as record- and bit-ordering are undefined in C

LM12L458 – Instructions

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	Instruction RAM	R/W	Acquisition				Watch-											
0		to		(RAM Pointer = 00)		Time				dog	8/$\overline{12}$	Timer	Sync	V_{IN-}			V_{IN+}			Pause	Loop
	1	1	1																		

☞ In C: use macro-assembler style programming instead:

Read up on the local bit and byte ordering and set bits inside an `int`:

```
unsigned int setbits (unsigned int *r,
                      unsigned int n,          /* set n bits     */
                      unsigned int p,          /* at position p  */
                      unsigned int x)          /* to bitstring x */
{
   unsigned int mask;
   mask  = ~(~0 << n);
   *r   &= ~(mask << p);
   *r   |= (x & mask) << p;
   return (*r);
}
```

LM12L458 – Configuration register

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	Configuration Register	R/W	Don't Care				DIAG	Test = 0	RAM Pointer		I/O Sel	Auto $Zero_{ec}$	Chan Mask	Stand-by	Full CAL	Auto-Zero	Reset	Start

Configuration register entries consist of:

- **Start** (1 bit): starts the sequencer.
- **Reset** (1 bit): sets the instruction pointer to '000'.
- **Auto-Zero** (1 bit): triggers a 'short' calibration (76 cycles – 1 offset sample).
- **Full-Cal** (1 bit): initiates a full calibration (4944 cycles, 8 samples) ☞ interrupt.
- **Stand-by** (1 bit): disconnects the external clock and preserves the registers.
  After powering up again (~ 10 ms): a specific interrupt is issued.
- **Chan-Mask** (1 bit): format selection for the FIFO output registers.

LM12L458 – Configuration register

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	Configuration Register	R/W	Don't Care				DIAG	Test = 0	RAM Pointer		I/O Sel	Auto $Zero_{ec}$	Chan Mask	Stand-by	Full CAL	Auto-Zero	Reset	Start

Configuration register entries consist of (cont.):

- **Auto-Zero**$_{ec}$ (1 bit): auto-zeros the ADC automatically in every conversion.
- **I/O Sel** (1 bit): sets the Sync pin to input or output mode.
- **RAM Pointer** (2 bits): selects the current (16-bit) part in each 48-bit instruction.
- **Test=0** (1 bit): production testing mode: leave this bit at '0'.
- **DIAG** (1 bit): connects $V_{IN}+$ and $V_{IN}-$ to $V_{REF}+$ and $V_{REF}-$ for testing purposes.

LM12L458 – Sequencer

```
IP := 0;
loop
   repeat
      if Auto_Zero or Full_Cal then Calibrate;
   until Start_bit;
   if Instr (IP).Timer then Run_Timer;
   Current_Signal := Aquisition (Instr (IP).Aquisition_Time);
   if Instr (IP).Watchdog then begin
      if Instr (IP).Sync then Wait_for_external_sync;
      Compare (Current, Instr (IP).Limit_1);
      if Instr (IP).Sync then Wait_for_external_sync;
      Compare (Current, Instr (IP).Limit_2);
   else
      if Instr (IP).Sync then Wait_for_external_sync;
      Convert_and_store_in_FIFO (Current_Signal);
   end;
   if Instr (IP).Loop then IP := 0;
                      else IP := IP + 1;
end loop;
```

LM12L458 – Conversion FIFO

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	0	Conversion FIFO	R	Address or Sign			Sign	Conversion Data: MSBs				Conversion Data: LSBs							

Conversion FIFO buffer is a *read-only* register:

- Every *read* on this address will *delete* this result from the internal memory and will shift the next result into the visible conversion FIFO register.
- The FIFO holds up to 32 conversion results.
- Data will be lost, if the results are not read fast enough to prevent a buffer overrun.

☞ The controller can issue specific interrupts or initiate a DMA transfer, when a given number of results are accumulated or a certain instruction is completed.

LM12L458 – Data transfer options

A4	A3	A2	A1	Purpose	Type	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	Interrupt Enable Register	R/W	Number of Conversions in Conversion FIFO to Generate INT2					Sequencer Address to Generate INT1			INT7	Don't Care	INT5	INT4	INT3	INT2	INT1	INT0
1	0	1	0	Interrupt Status Register	R	Actual Number of Conversion Results in Conversion FIFO					Address of Sequencer Instruction being Executed			INST7	"0"	INST5	INST4	INST3	INST2	INST1	INST0

Interrupts:

- Can be associated by an existing instruction and/or triggered by a specific number of conversion results.

☞ The interrupt can be handled by another controller, DMA mechanism, or the actual CPU.

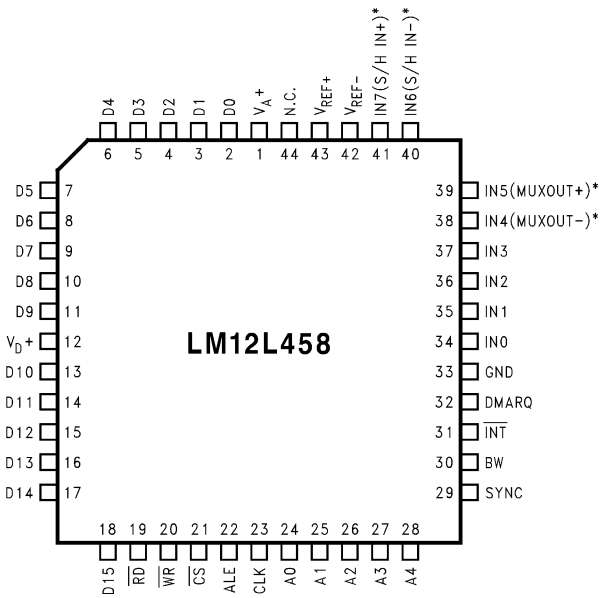
Polling:

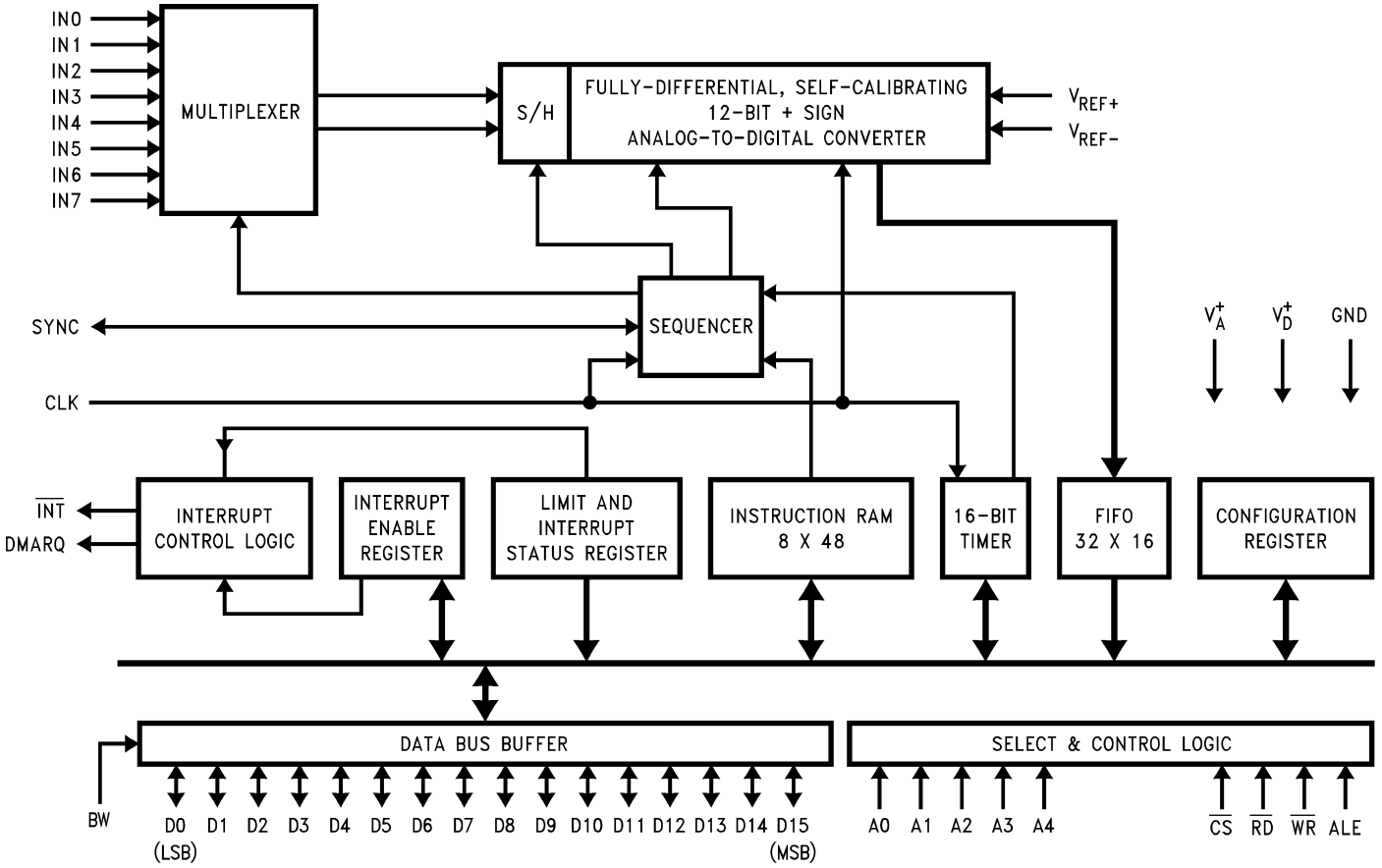
☞ In dedicated, high integrity, or low latency controller setups the client controller/CPU will poll rather than wait for interrupts.

A/D, D/A & Interfaces: Examples

LM12L458

(National Semiconductors)





Control Register 2

```
type ADC_CR2 is record
   ADON       : Enabler;          -- A/D Converter ON / OFF (read & write)
   CONT       : CONT_Options;     -- Continuous conversion (read & write)
   Reserved_1 : Bits_6;
   DMA        : Enabler;          -- Direct memory access mode (for single ADC mode) (read & write)
   DDS        : Enabler;          -- DMA disable selection (for single ADC mode) (read & write)
   EOCS       : Enabler;          -- End of conversion selectionat (read & write)
   ALIGN      : ALIGN_Options;    -- Data alignment (read & write)
   Reserved_2 : Bits_4;
   JEXTSEL    : JEXTSEL_Options; -- External event select for injected group (read & write)
   JEXTEN     : EXTEN_Options;    -- External trigger enable for injected channels (read & write)
   JSWSTART   : Enabler;          -- Start conversion of injected channels (read & write)
   Reserved_3 : Bit;
   EXTSEL     : EXTSEL_Options;   -- External event select for regular group (read & write)
   EXTEN      : EXTEN_Options;    -- External trigger enable for regular channels (read & write)
   SWSTART    : Enabler;          -- Start conversion of regular channels (read & write)
   Reserved_4 : Bit;
end record with Volatile, Size => Word'Size;
```

Control Register 2

```
for ADC_CR2 use record
   ADON       at 0 range  0 ..  0; -- A/D Converter ON / OFF (read & write)
   CONT       at 0 range  1 ..  1; -- Continuous conversion (read & write)
   Reserved_1 at 0 range  2 ..  7;
   DMA        at 0 range  8 ..  8; -- Direct memory access mode (for single ADC mode) (read & write)
   DDS        at 0 range  9 ..  9; -- DMA disable selection (for single ADC mode) (read & write)
   EOCS       at 0 range 10 .. 10; -- End of conversion selection (read & write)
   ALIGN      at 0 range 11 .. 11; -- Data alignment (read & write)
   Reserved_2 at 0 range 12 .. 15;
   JEXTSEL    at 0 range 16 .. 19; -- External event select for injected group (read & write)
   JEXTEN     at 0 range 20 .. 21; -- External trigger enable for injected channels (read & write)
   JSWSTART   at 0 range 22 .. 22; -- Start conversion of injected channels (read & write)
   Reserved_3 at 0 range 23 .. 23;
   EXTSEL     at 0 range 24 .. 27; -- External event select for regular group (read & write)
   EXTEN      at 0 range 28 .. 29; -- External trigger enable for regular channels (read & write)
   SWSTART    at 0 range 30 .. 30; -- Start conversion of regular channels (read & write)
   Reserved_4 at 0 range 31 .. 31;
end record;
```

Control Register 2

```
ADC_CR2_Reset : constant ADC_CR2 :=
  (ADON       => Disable,           -- A/D Converter ON / OFF (read & write)
   CONT       => Single,            -- Continuous conversion (read & write)
   Reserved_1 => 0,
   DMA        => Disable,           -- Direct memory access mode (for single ADC mode) (read & write)
   DDS        => Disable,           -- DMA disable selection (for single ADC mode) (read & write)
   EOCS       => Disable,           -- End of conversion selectionat (read & write)
   ALIGN      => Right,             -- Data alignment (read & write)
   Reserved_2 => 0,
   JEXTSEL    => Timer_1_CC4_event, -- External event select for injected group (read & write)
   JEXTEN     => Disabled,          -- External trigger enable for injected channels (read & write)
   JSWSTART   => Disable,           -- Start conversion of injected channels (read & write)
   Reserved_3 => 0,
   EXTSEL     => Timer_1_CC1_event, -- External event select for regular group (read & write)
   EXTEN      => Disabled,          -- External trigger enable for regular channels (read & write)
   SWSTART    => Disable,           -- Start conversion of regular channels (read & write)
   Reserved_4 => 0);
```

Micro-controller definition

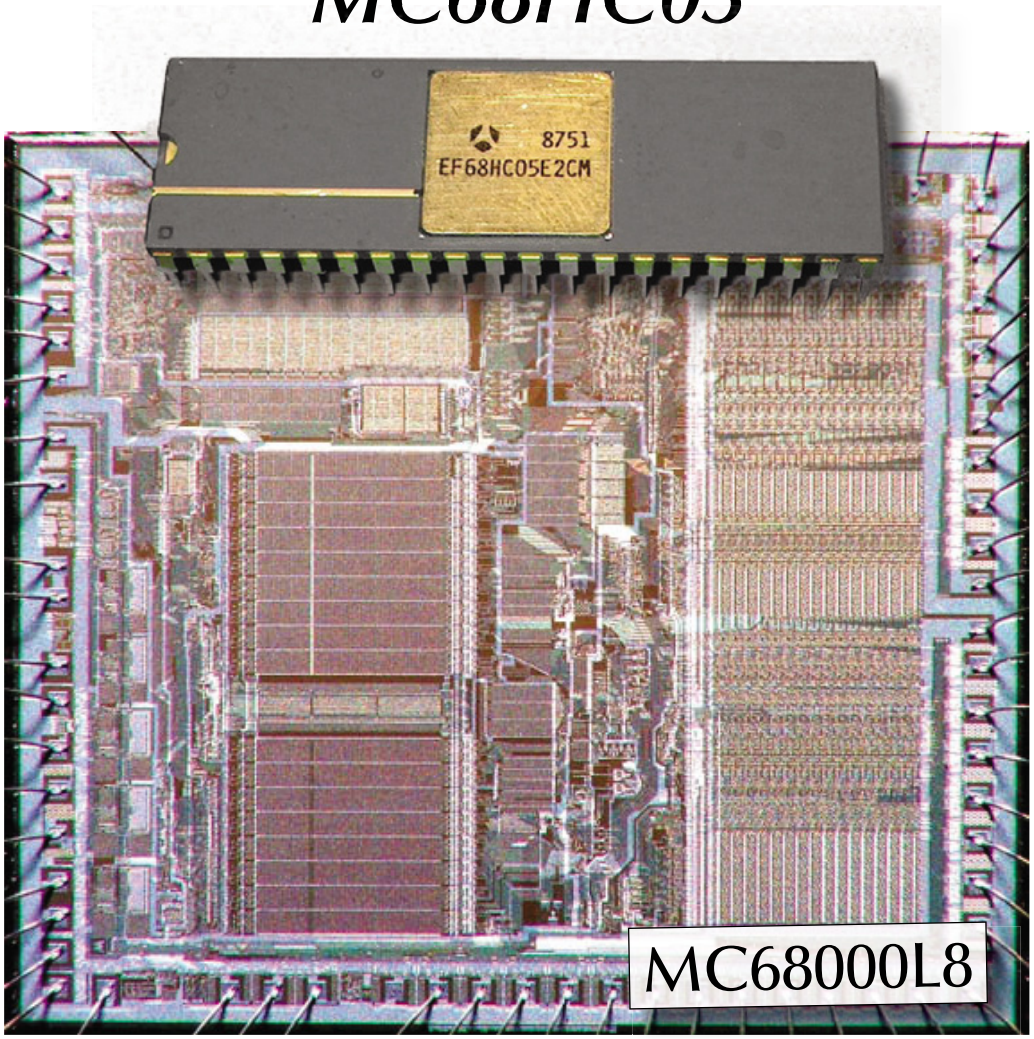
☞ Short: "Computer system on a chip"

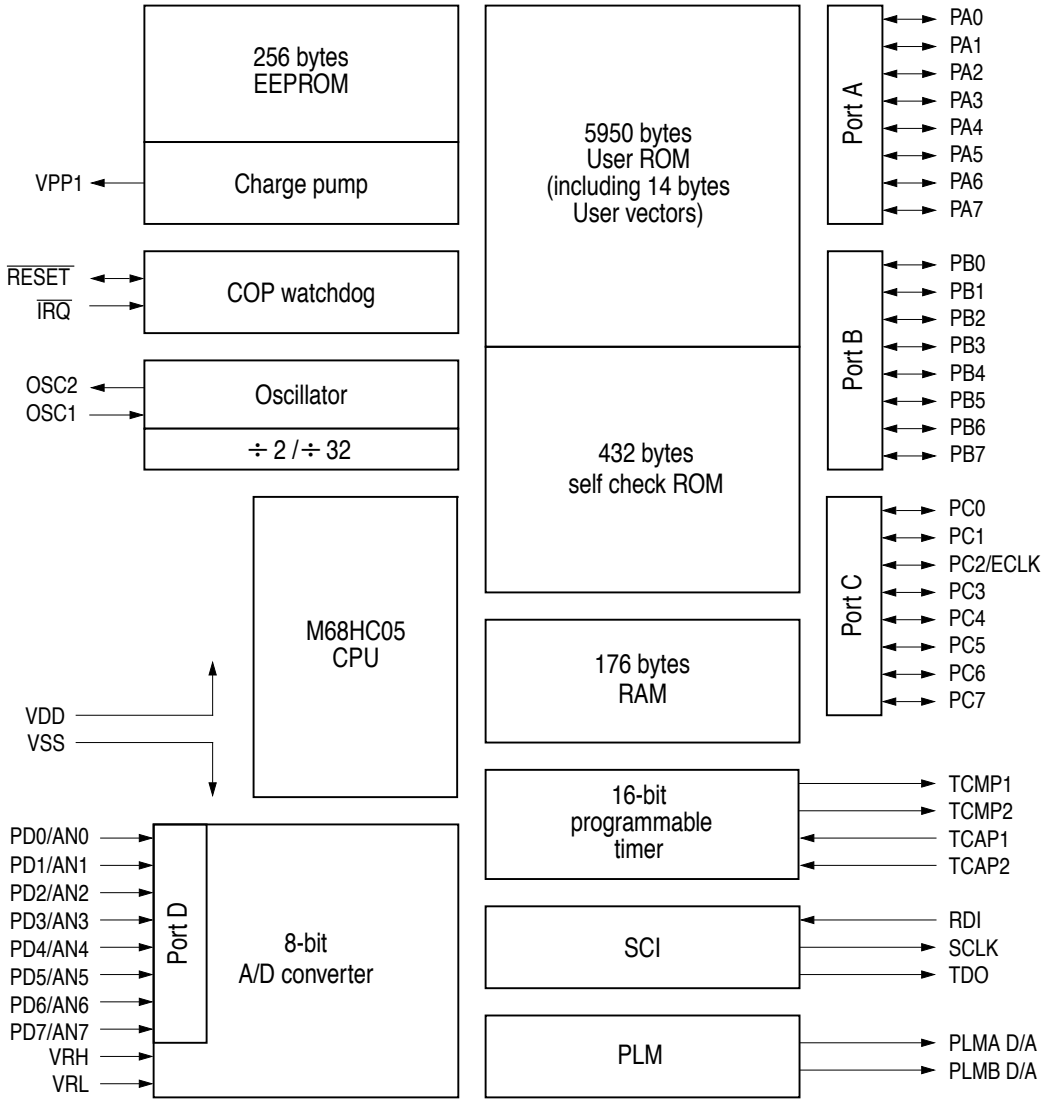
Typical elements found in a micro-controller include:

- CPU (typ. 4-64 bit word size) – also as multi-cores.
- Memory (typ. a few hundred Bytes to many MBytes) as RAM, ROM, EPROM and/or Flash.
- Clock generator.
- Timers and general interrupt logic.
- Basic I/O (often as multiple, partly autonomous I/O units):
  - General purpose digital I/O lines – often combined with PWM generators, signal width detectors, counters, watchdog- or timer-triggers.
  - A/D and D/A converters (typ. 6-12 bit).
- Higher level I/O channels: Ethernet, UART, I^2C, Fast serial, Can-bus, …

Micro-controller examples

MC68HC05

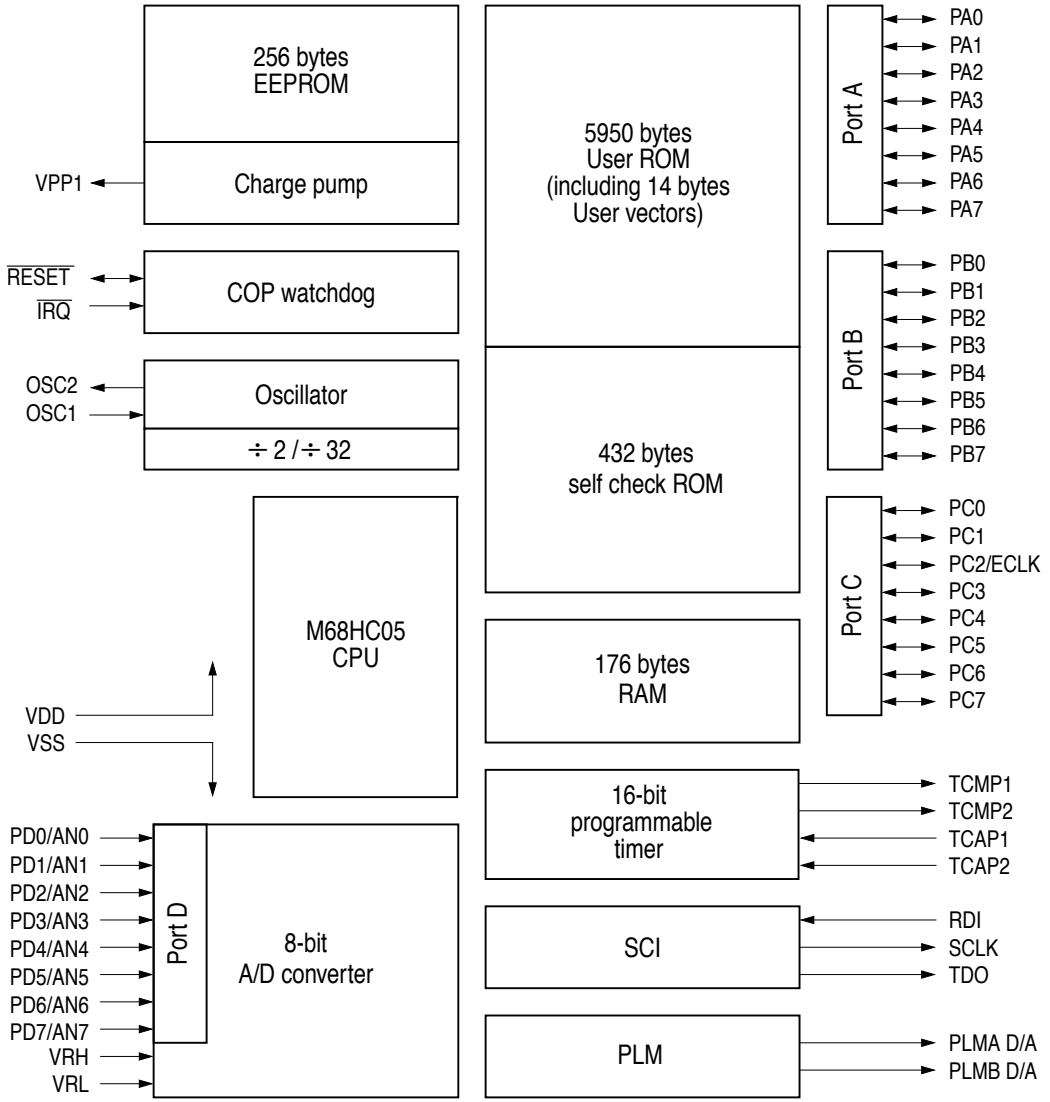




Micro-controller examples

MC68HC05

- **Clock**: max. 2.1 MHz internal (4.2 MHz external).
- **RAM**: 176 bytes.
- **ROM**: 5936 bytes.
- **EEPROM**: 256 bytes.
- **Power** saving modes (stop, wait, slow).
- **Serial**: 46-76800 baud (at 2.4576 MHz) 79-131072 baud (at 4.194394 MHz).
- **Parallel** I/O: 3 · 8 bit; Parallel in: 1 · 8 bit.
- **Timers**: 1 · 16 bit.
- **A/D**: 8 channels, 8 bit.
- **PWM**: 2 generators.

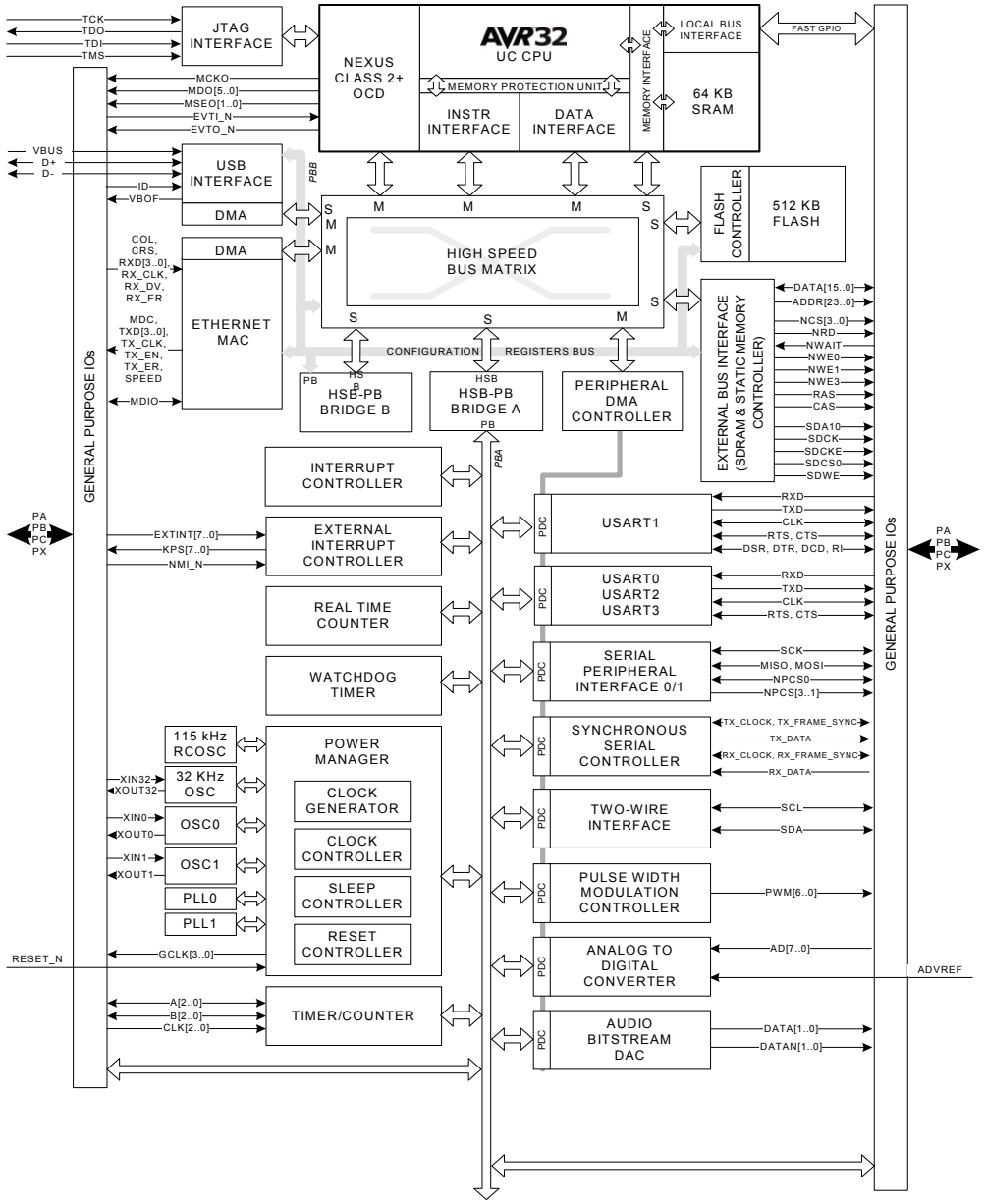


Interfaces

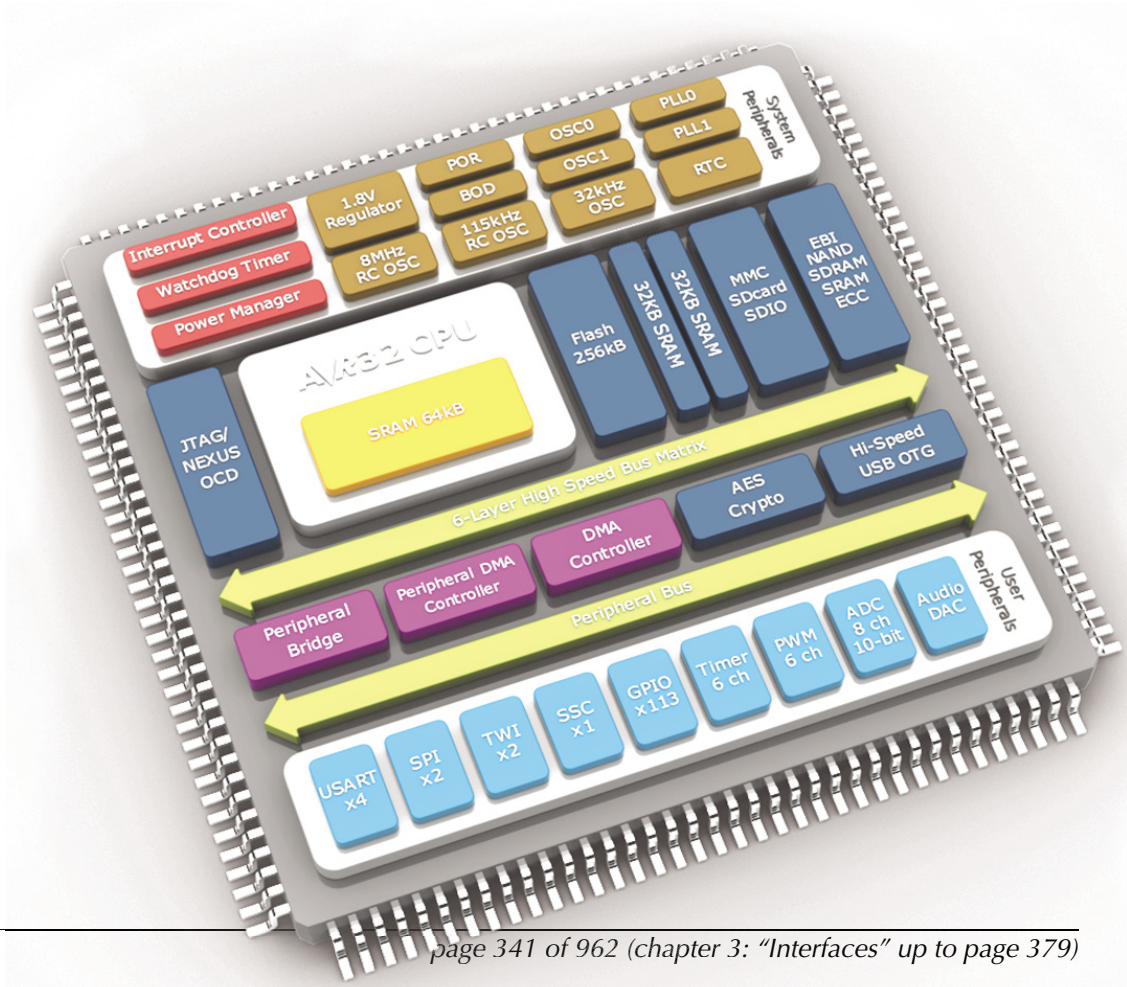
```
MAIN   BRCLR    6,TSR,MAIN   ;Loop here till Output Compare flag set
       LDA      OCMP+1       ;Low byte of Output Compare register
       ADD      #$D4         ;Add
       STA      TEMPA        ;Save till high half calculated
       LDA      OCMP         ;High byte of Output Compare register
       ADC      #$30         ;Add  (+carry)
       STA      OCMP         ;Update high byte of Output Compare register
       LDA      TEMPA        ;Get low half of updated value
       STA      OCMP+1       ;Update low half and reset Output Compare flag
       LDA      TIC          ;Get current TIC value
       INCA                  ;TIC := TIC + 1
       STA      TIC          ;Update TIC
       CMP      #20          ;20th TIC?, 1 second passed?
       BLO      NOSEC        ;If not, skip next clear
       CLR      TIC          ;Clear TIC on 20th
NOSEC  EQU      *
       JSR      TIME         ;Update time-of-day & day-of-week
       JSR      KYPAD        ;Check/service keypad
       JSR      A2D          ;Check Temp Sensors
       JSR      HVAC         ;Update Heat/Air Cond Outputs
       JSR      LCD          ;Update LCD display
       BRA      MAIN         ;End of main loop
```

Interfaces

Micro-controller examples



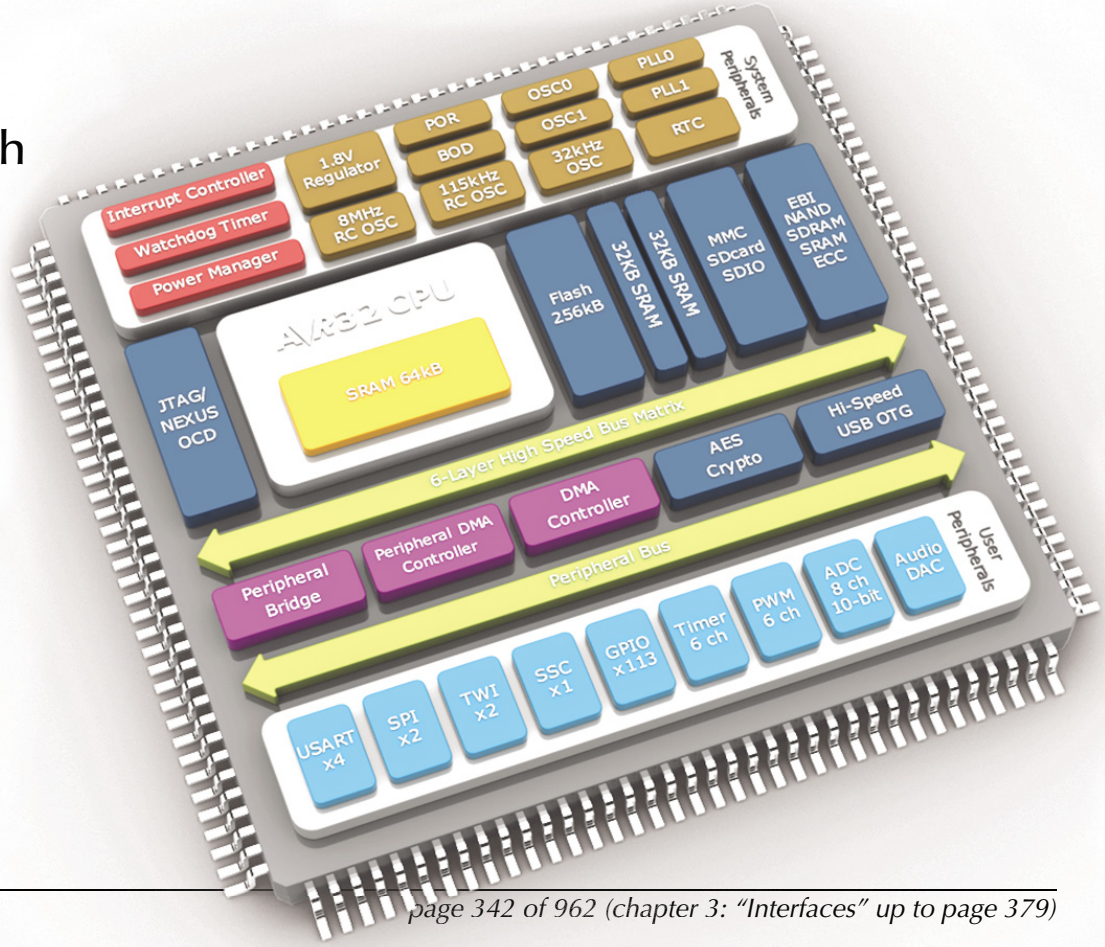
AVR32



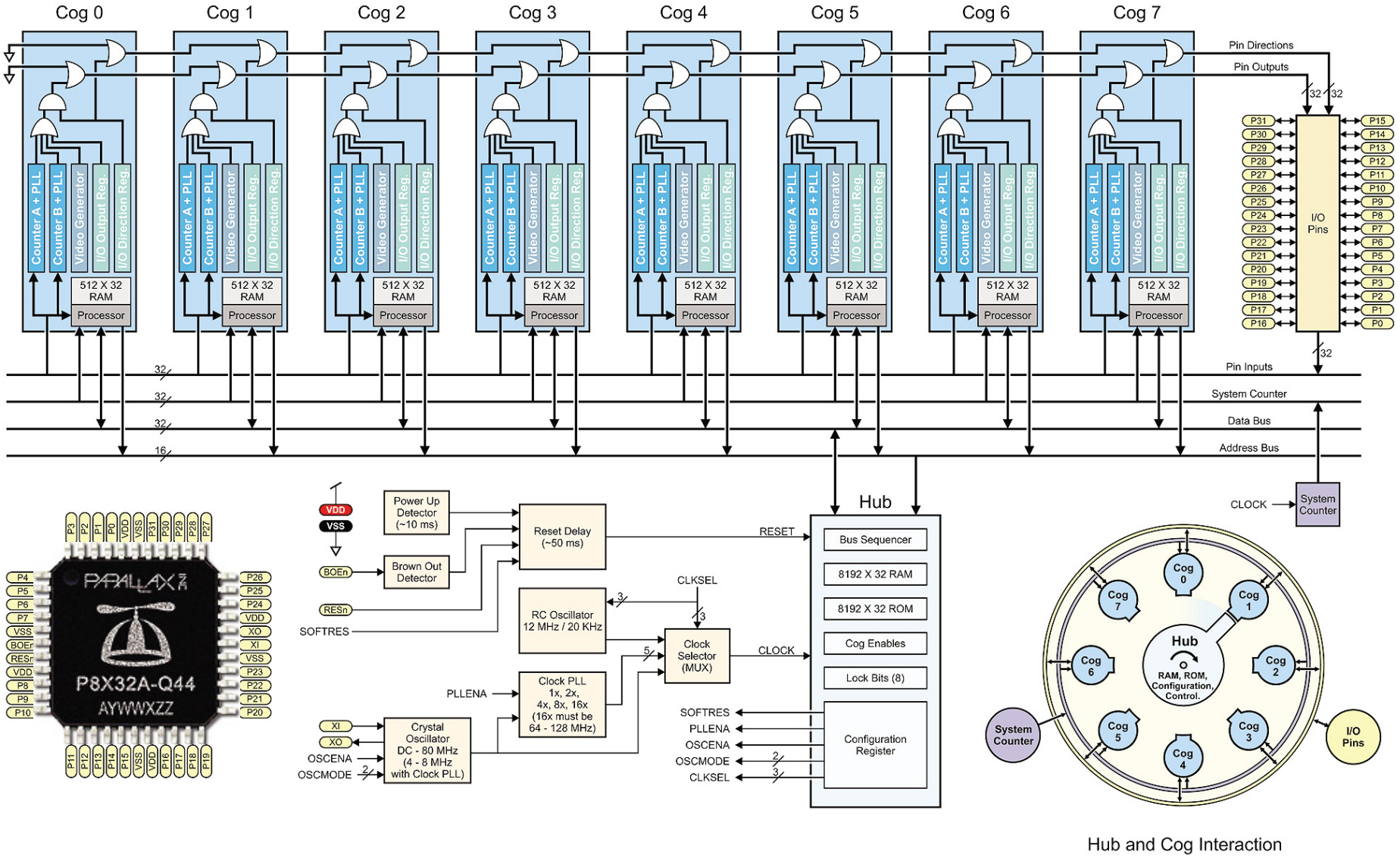
Micro-controller examples

AVR32

- CPU: 32 bit RISC with DSP extensions.
- Clock: 66 MHz.
- Memory: up to 32 kB SRAM, up to 512 kB Flash
- Separate DMA bus for peripherals.
- Power: up to 120 mW (132 µW in sleep)
- Nexus debug port.
- up to 113 GPIO with up to 7 PWM channels.
- 1 · 32 bit real time counter.
- 6 · 16 bit timers
- 1 · Ethernet, 1 · SSC, 4 · UART (incl. SPI)
- 8 · 10 bit ADC (SAR).
- 2 · DDC bitstream output (Σ-Δ).

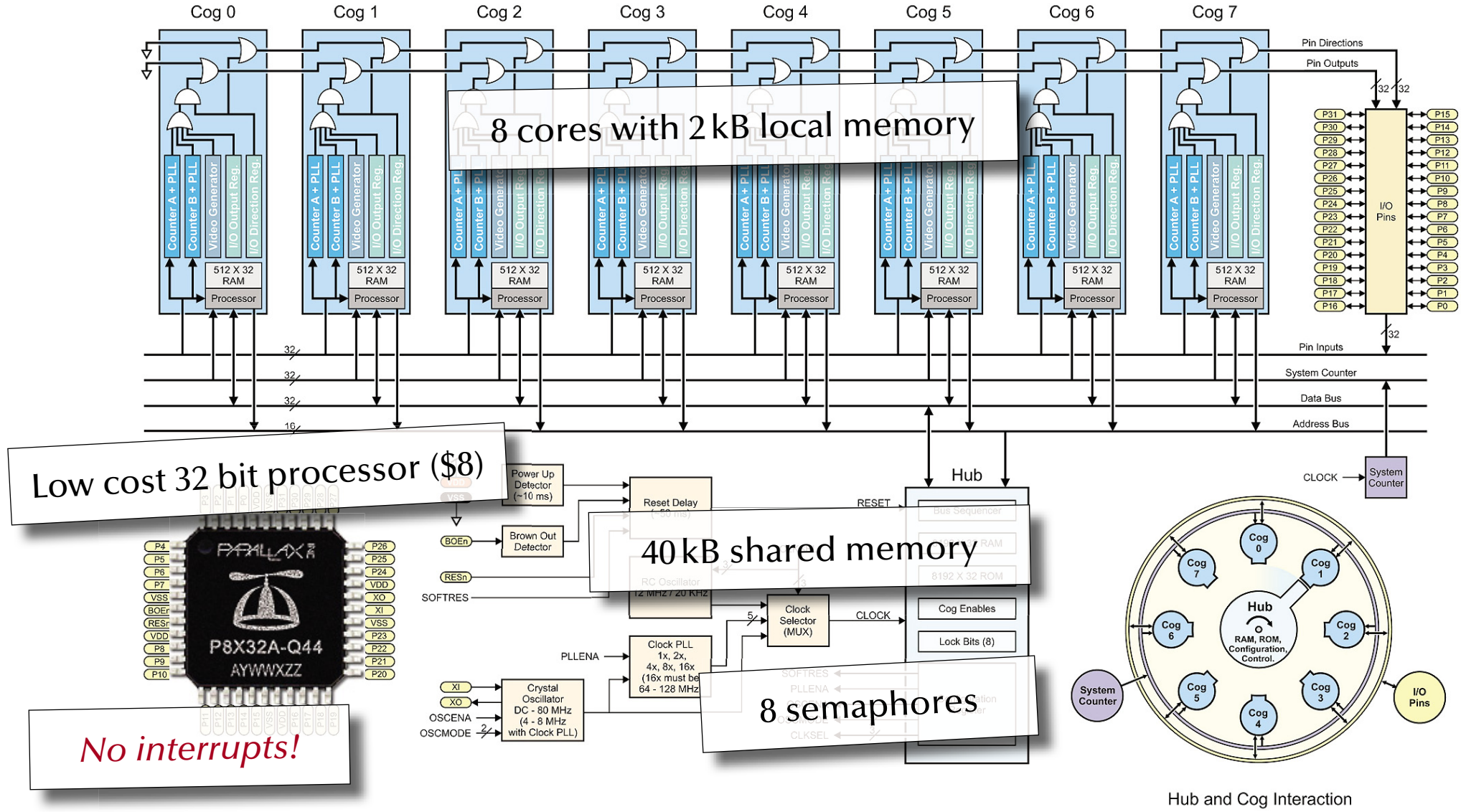


Alternative Processor Architectures: Parallax Propeller



Hub and Cog Interaction

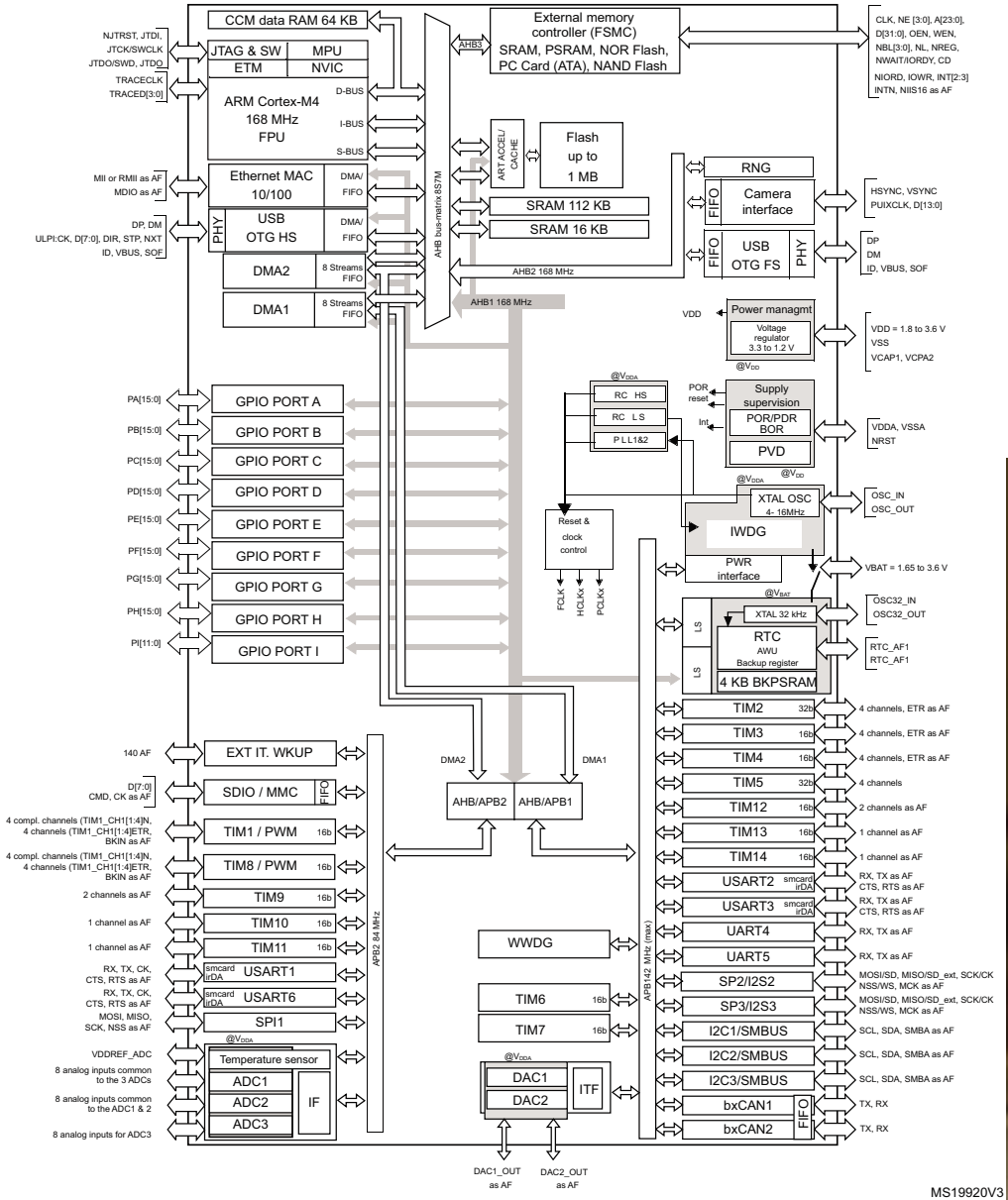
Alternative Processor Architectures: Parallax Propeller (2006)



Hub and Cog Interaction

Micro-controller examples

STM32F40xxx



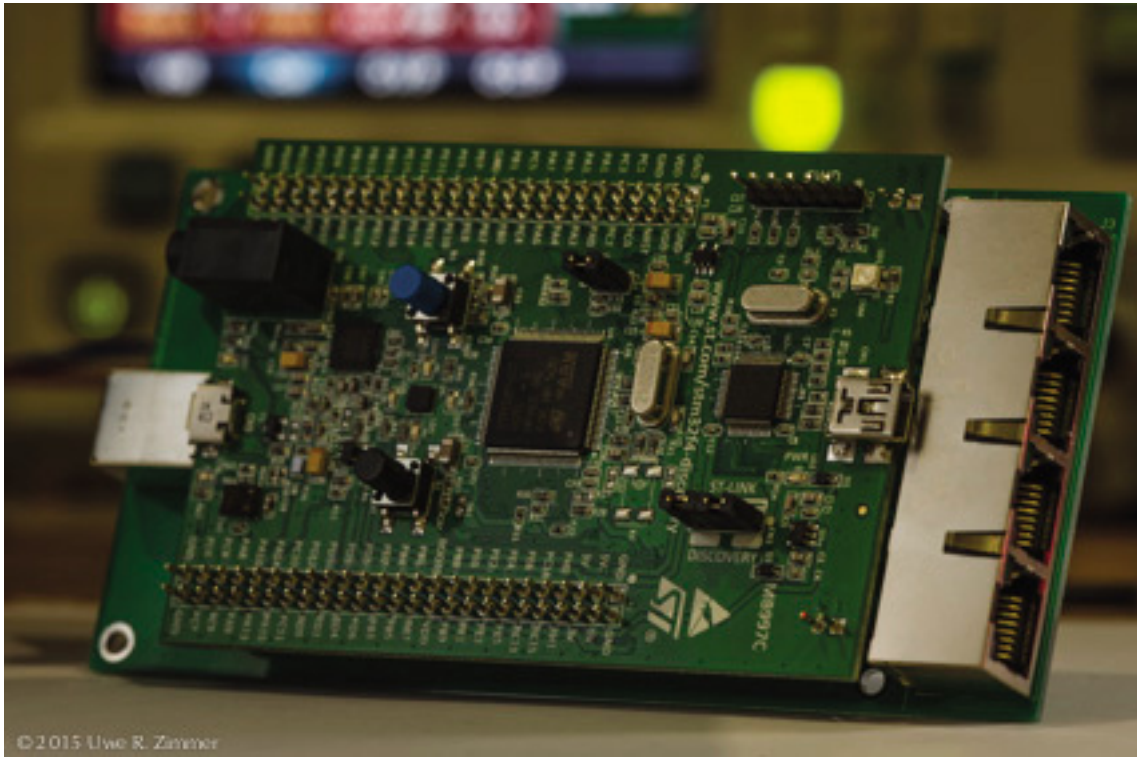
Interfaces

- Power dissipation: ≈ 1-420 mW
- 2-168 MHz, 32-bit ARM Cortex-M4F Core
- 1 MB Flash, 192 KB RAM, MPU
- 3-axis accelerometer
- Audio interface with class D speaker driver
- USB OTG FS
- 12x 16-bit timers, 2x 32-bit timer
- Watchdog timer
- 16x DMA channels
- 2x CAN, 3x I2C, 3x SPI, 6x UART's,
- IEEE1588 (Precision Time Protocol)
- RTC, PLL
- 2x 12-bit DAC
- 3x 12-bit ADC (16 channels),
- Nested vectored interrupt controller
- Random number generator

Micro-controller examples

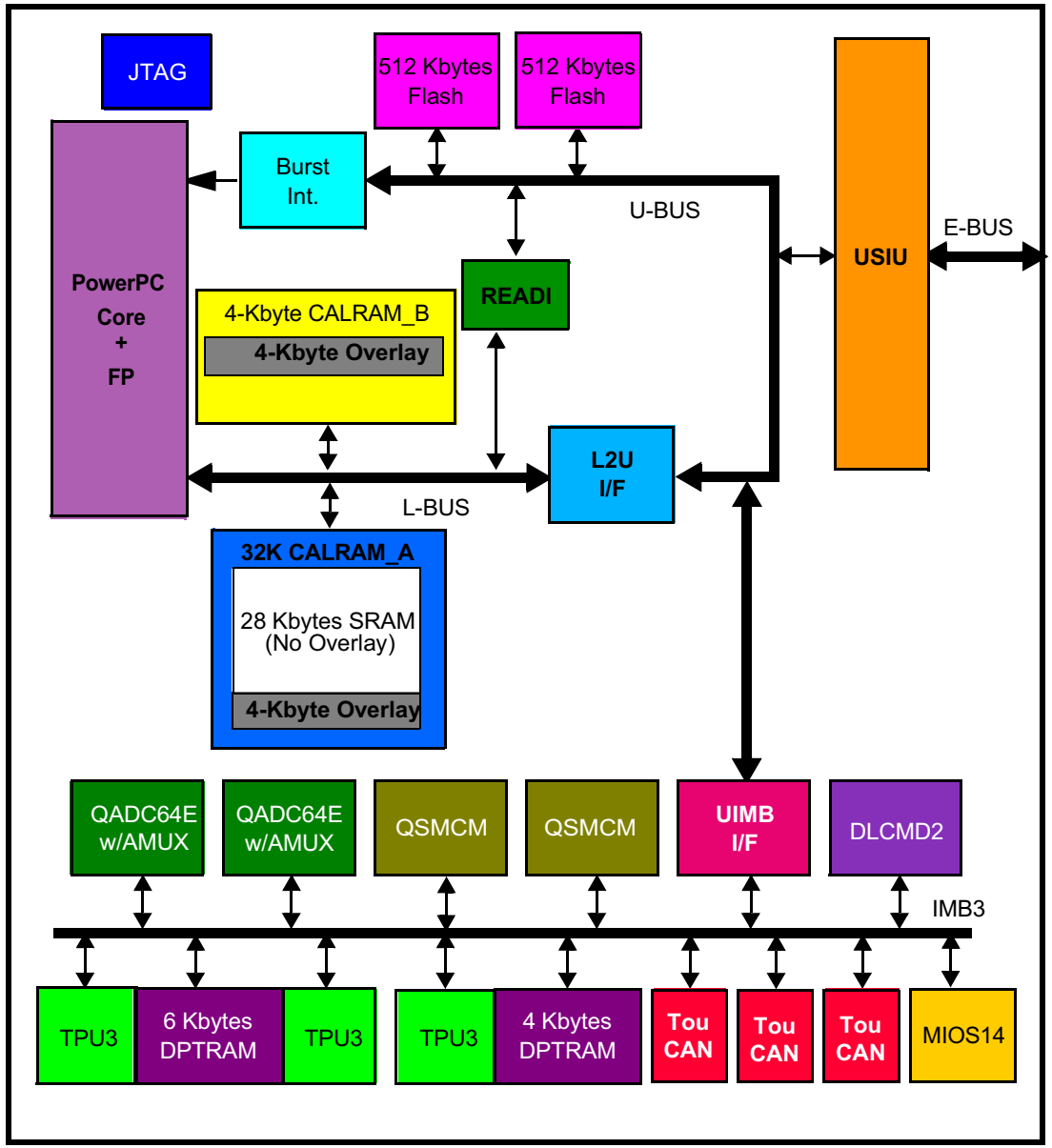
STM32F40xxx

Discovery board

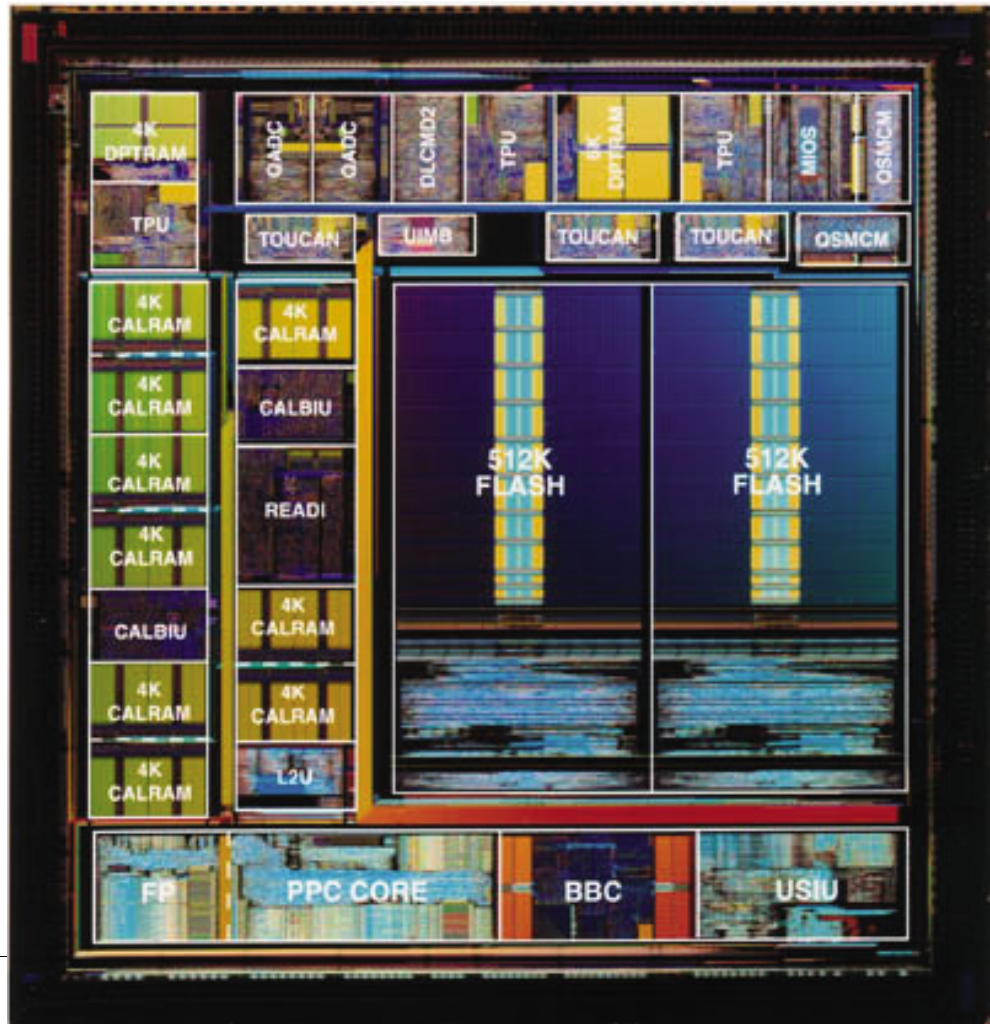


Interfaces

Micro-controller examples



MPC565

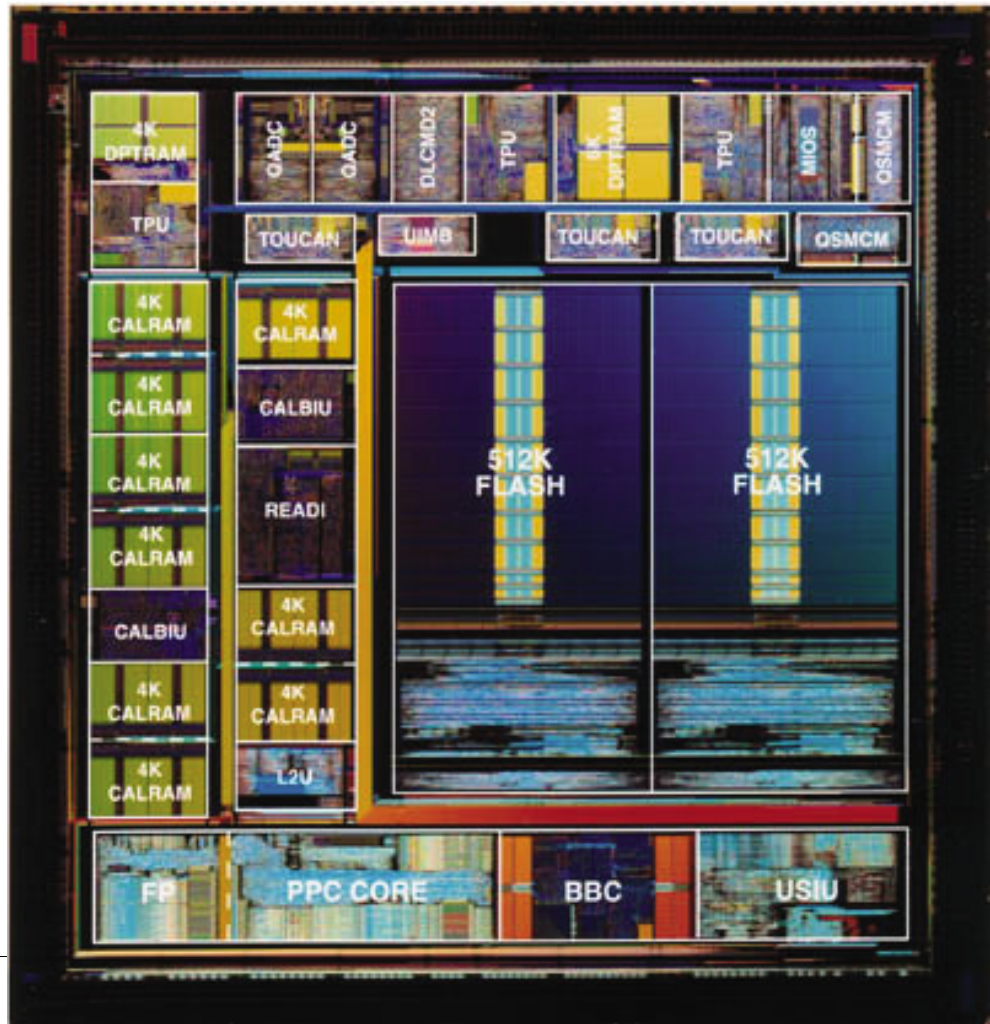


Interfaces

Micro-controller examples

MPC565

- **Power**: power dissipation: 0.8 - 1.12 W, -40° - +125°C
- **CPU**: PowerPC core (incl. FPU & BBC), 56 MHz
- **RAM**: flash: 1 MB, static: 36 kB
- **Time processing units**: 3 (via dual-ported RAM)
- **Timers**: 22 channels (PWM & RTC supported)
- **A/D** converters: 40 channels, 10 bit, 250 kHz
- **Can-bus**: 3 TOUCAN modules
- **Serial**: 2 interfaces
- **Data link controller**:
  SAE J1850 class B communications module
- **Real-time embedded application development interface**: NEXUS debug port (IEEE-ISTO 5001-1999)
- **Packing**: 352/388 ball PBGA

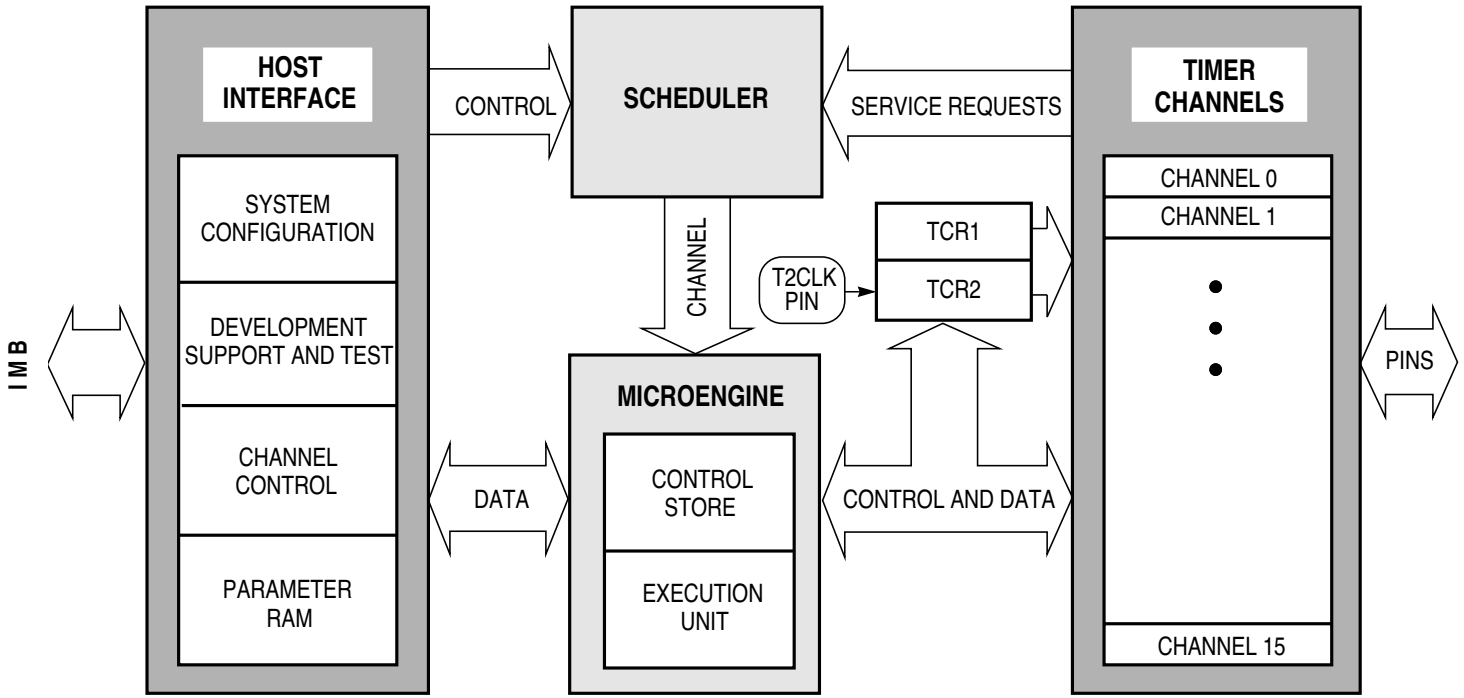


Micro-controller examples

Time Processing Unit

A special-purpose micro-controller:

- Independent µ-engine.
- 16 digital I/O channels with independent match and capture capabilities.
- Meant to operate these I/O channels for timing control purposes.
- Predefined µ-engine command set (ROM functions in control store).
- 2 · 16 bit time bases



Micro-controller examples

Time Processing Unit – Some predefined µ-engine functions

Period- / Pulse-width accumulator	The period/pulse-width accumulator (PPWA) algorithm accumulates a 16 bit or 24 bit sum of either the period or the pulse width of an input signal over a programmable number of periods or pulses (from 1 to 255).
Stepper motor	The stepper motor (SM) control algorithm provides for linear acceleration and deceleration control of a stepper motor with a programmable number of step rates of up to 14.
Position-synchronized pulse generator	The PSP function generates pulses of variable length at specified "angles." Angle clock period is measured (in TCR1 clocks) using the PMA/PMM function on another channel.
Period measurement	This function measures the period (in TCR1 clocks) between regularly occurring input transitions and makes this period available for use by other functions or by the CPU (optional detection of misses and additional transitions).
Pulse-width modulation	The TPU can generate a pulse-width modulation (PWM) waveform with any duty cycle from zero to 100% (within the resolution and latency capability of the TPU).
Synchronized pulse-width modulation	Three different operating modes allow the function to maintain complex timing relation-ships between channels without CPU intervention.
Quadruple decode	QDEC uses two channels to decode a pair of out-of-phase signals in order to present the CPU with directional information and a position value.

Micro-controller examples

Time Processing Unit – Emulation Mode

☞ Create your own µ-engine

Refer the control store of the µ-engine to the dual-ported RAM instead of the integrated ROM area and supply:

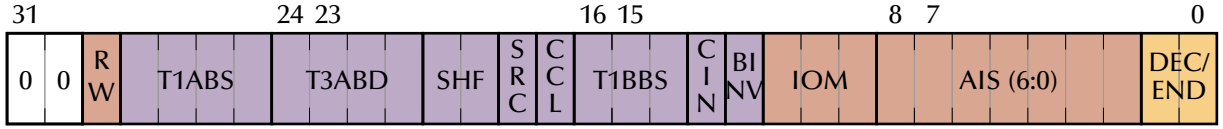
- Up to 16 µ-engine commands (functions).
- In 2-8 kB of long-word (32 bit) organized memory.
- Programmed in a 32 bit µ-instruction format (explained next).

☞ The dual-ported RAM is then cut off from the CPU (the TPU parameter RAM is not affected)

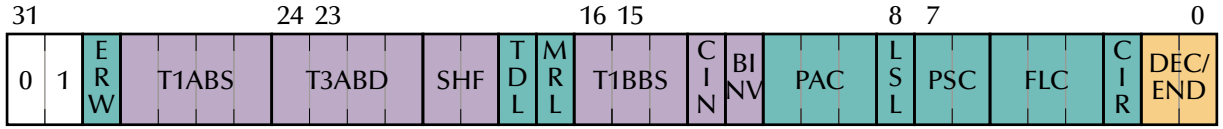
Micro-controller examples

Time Processing Unit – μinstructions formats:

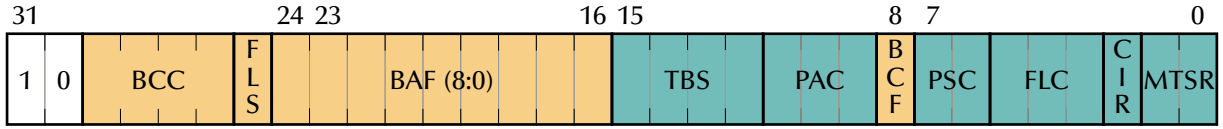
1: Execution unit and RAM:



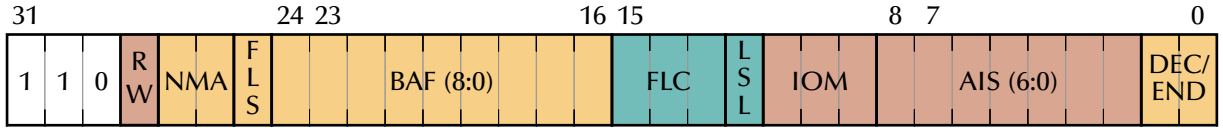
2: Execution unit, flag, and channel control:



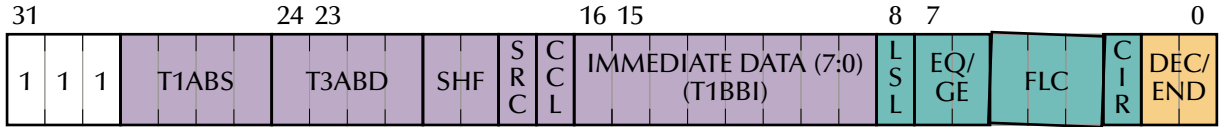
3: Conditional branch, flag, and channel control:



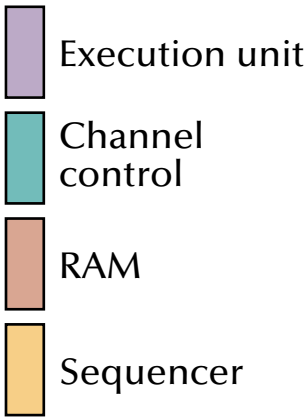
4: Jump, flag, and RAM:



5: Execution unit, immediate, and flag:



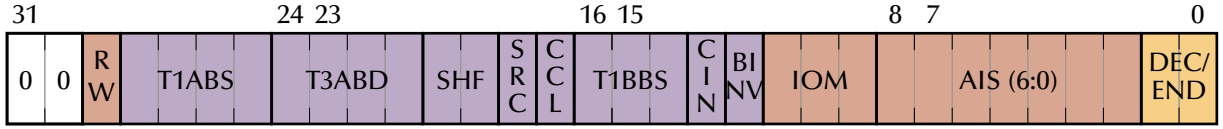
Operation groups:



Micro-controller examples

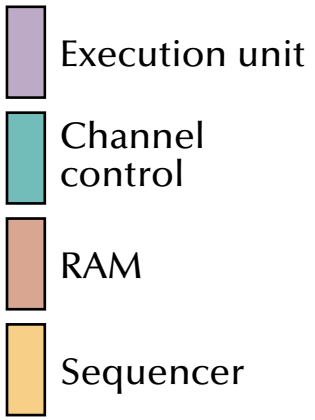
Time Processing Unit – μinstructions formats:

1: Execution unit and RAM:



RW	RAM	Read/Write Control
T1ABS	T1	A-Bus Source Control
T3ABD	T3	A-Bus Destination Control
SHF	AU	Shifter Control
SRC	Shift	Register Control
CCL	AU	Condition Code Latch Control
T1BBS	T1	B-Bus Source Control
CIN	AU	B-Bus Carry Control
BINV	AU	B-Bus Invert Control
IOM	RAM	Input/Output Mode Control
AIS	RAM	Address
DEC/END	SEQ	Decrementor / End Control

Operation groups:

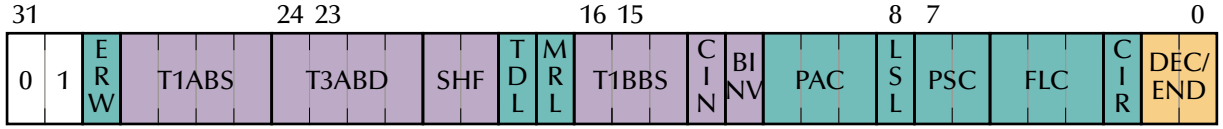


Interfaces

Micro-controller examples

Time Processing Unit – µinstructions formats:

2: Execution unit, flag, and channel control:



ERW	RAM	Event Register Write Control
T1ABS	T1	A-Bus Source Control
T3ABD	T3	A-Bus Destination Control
SHF	AU	Shifter Control
TDL	CC	Transition Detect Latch Negation Control
MRL	CC	Match Recognition Latch Negation Control
T1BBS	T1	B-Bus Source Control
CIN	AU	B-Bus Carry Control
BINV	AU	B-Bus Invert Control
PAC	CC	Pin Action Control
LLS	CC	Link Service Latch Negation Control
PSC	CC	Pin State Control
FLC	CC	Flag Control
CIR	CC	Channel Interrupt Request
DEC/END	SEQ	Decrementor / End Control

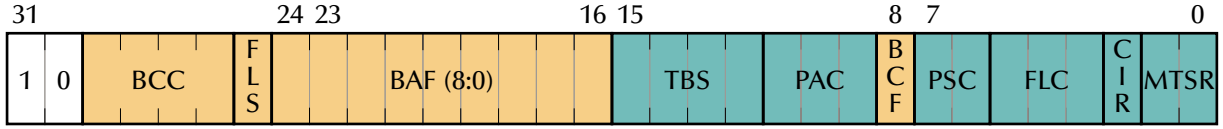
Operation groups:

- Execution unit
- Channel control
- RAM
- Sequencer

Micro-controller examples

Time Processing Unit – μinstructions formats:

3: Conditional branch, flag, and channel control:



BCC	SEQ	Branch Condition Code Field
FLS	SEQ	μPC Flush Control
BAF	SEQ	Branch Address Field
TBS	CC	Time Base Select Control
PAC	CC	Pin Action Control
BCF	SEQ	Branch Condition Control
PSC	CC	Pin State Control
FLC	CC	Flag Control
CIR	CC	Channel Interrupt Request
MTR	CC	Match/Transition Detect Service Request Inhibit Control

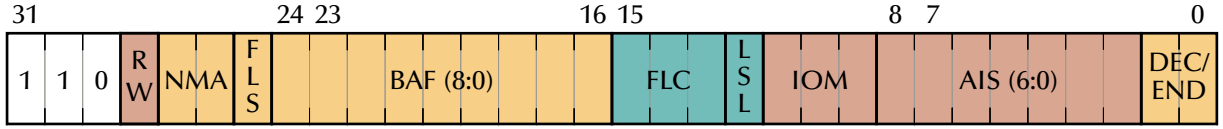
Operation groups:

- Execution unit
- Channel control
- RAM
- Sequencer

Micro-controller examples

Time Processing Unit – μinstructions formats:

4: Jump, flag, and RAM:



RW	RAM	Read/Write Control
NMA	SEQ	Next μPC Address Mode Control
FLS	SEQ	μPC Flush Control
BAF	SEQ	Branch Address Field
FLC	CC	Flag Control
LSL	CC	Link Service Latch Negation Control
IOM	RAM	Input/Output Mode Control
AIS	RAM	Address
DEC/END	SEQ	Decrementor / End Control

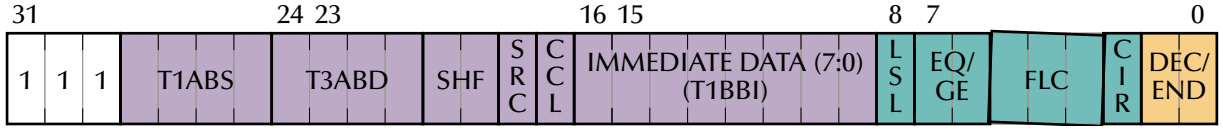
Operation groups:

- Execution unit
- Channel control
- RAM
- Sequencer

Micro-controller examples

Time Processing Unit – μinstructions formats:

5: Execution unit, immediate, and flag:



T1ABS	T1	A-Bus Source Control
T3ABD	T3	A-Bus Destination Control
SHF	AU	Shifter Control
SRC	Shift	Register Control
CCL	AU	Condition Code Latch Control
T1BBI	T1	B-Bus Immediate Data
LSL	CC	Link Service Latch Negation Control
EQ/GE	CC	Match Compare Register Control
FLC	CC	Flag Control
CIR	CC	Channel Interrupt Request
DEC/END	SEQ	Decrementor / End Control

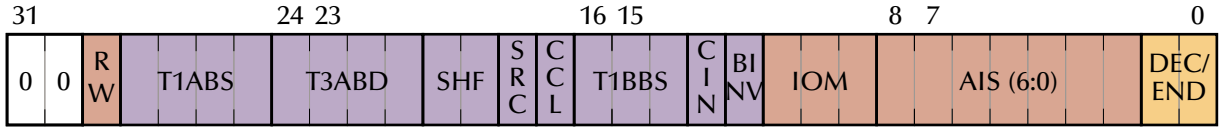
Operation groups:

- Execution unit
- Channel control
- RAM
- Sequencer

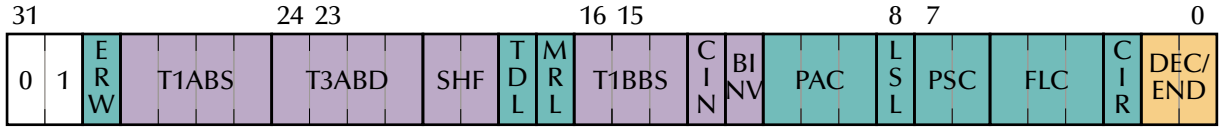
Micro-controller examples

Time Processing Unit – μinstructions formats:

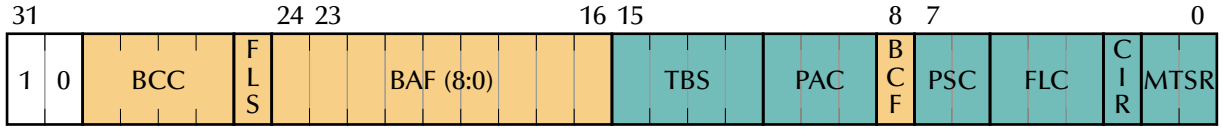
1: Execution unit and RAM:



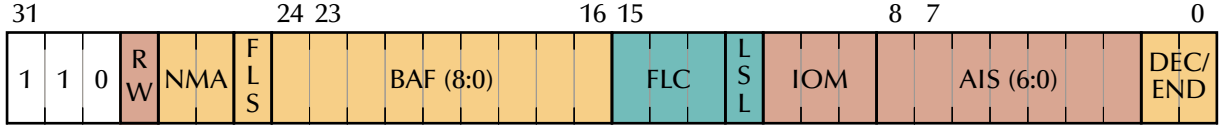
2: Execution unit, flag, and channel control:



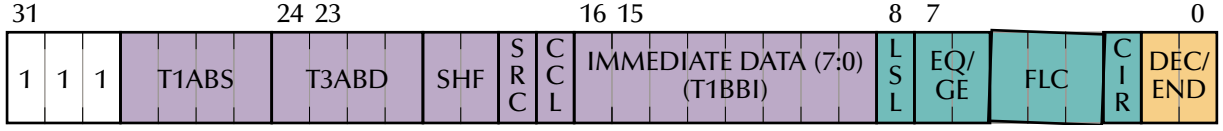
3: Conditional branch, flag, and channel control:



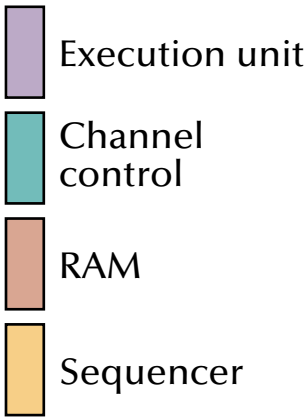
4: Jump, flag, and RAM:



5: Execution unit, immediate, and flag:



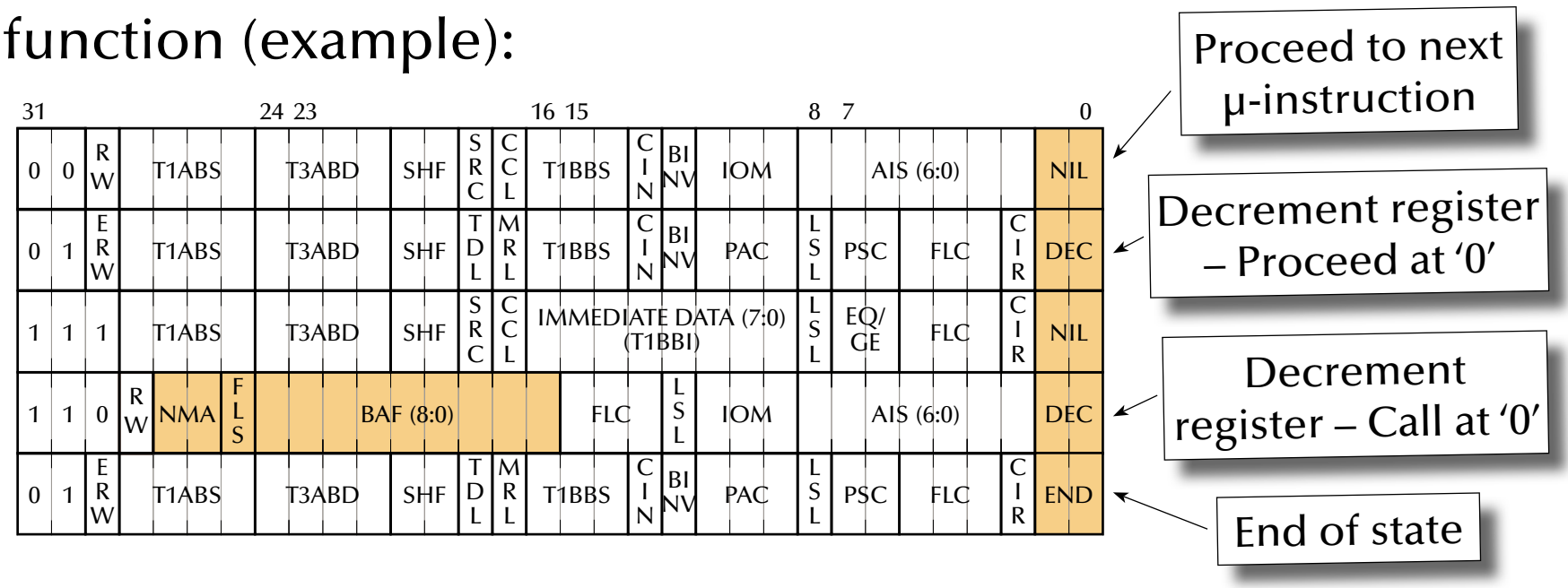
Operation groups:



Micro-controller examples

Time Processing Unit – Example Code

One state of a function (example):



Functions are composed of one or multiple states.

☞ States are *atomic* instruction blocks,
i.e. the scheduler will not interrupt.

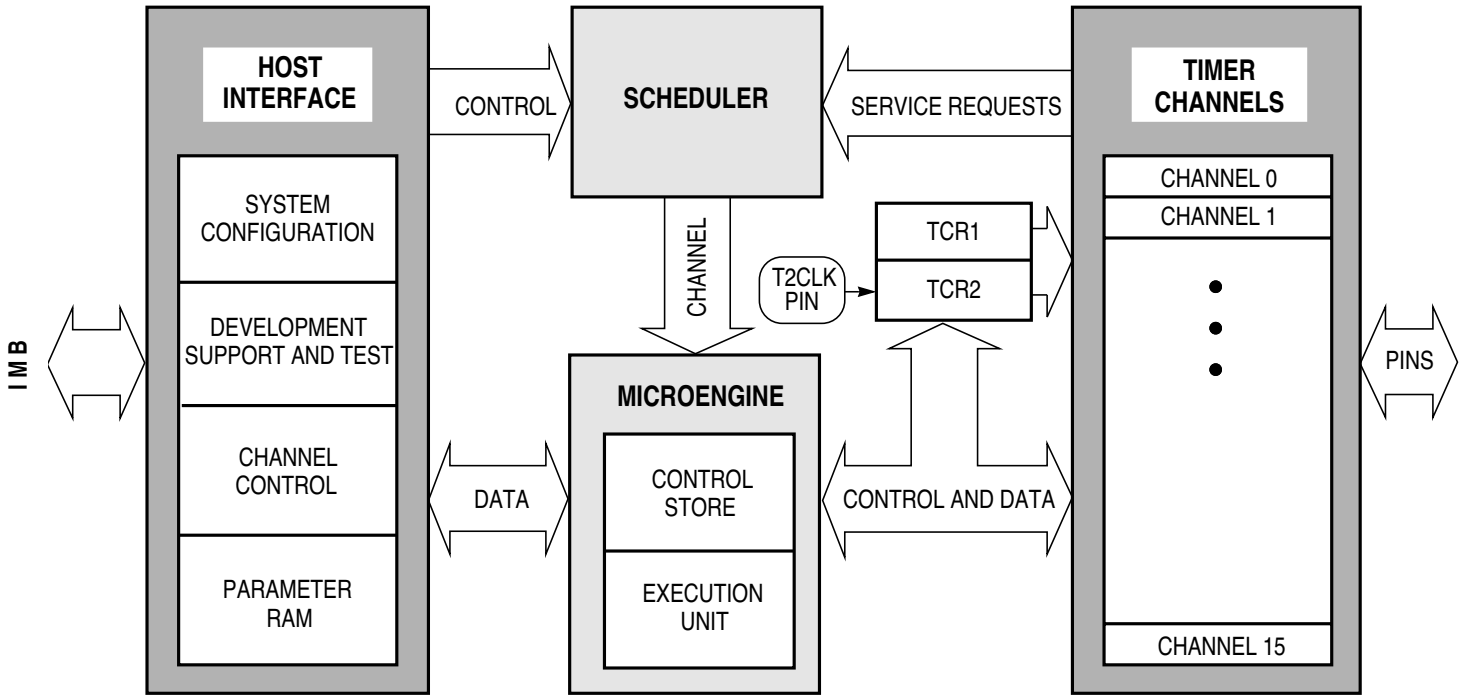
Micro-controller examples

Composing the μengine:

Entities to consider:

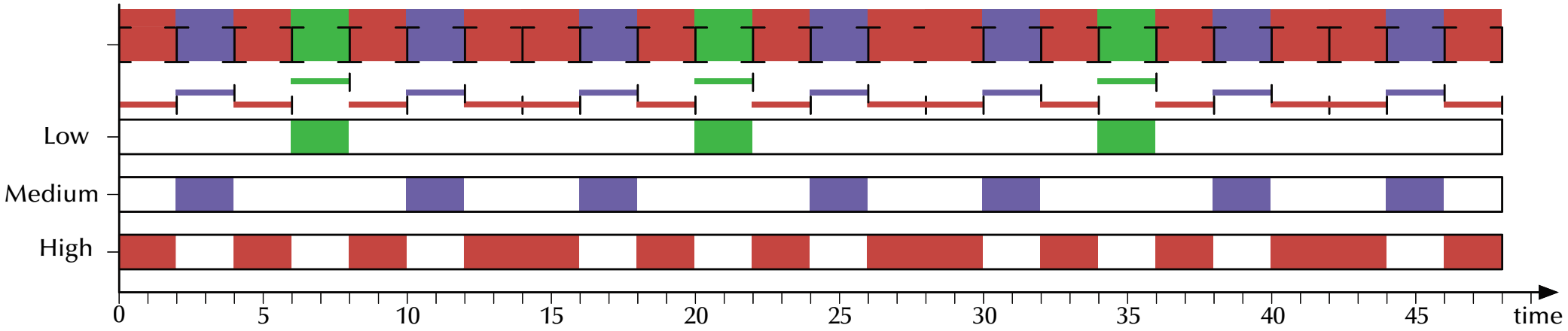
- **States** : non-interruptible μ-code-blocks.
- **Functions** : constructed of one or multiple states.
- **Channels** : 16 digital I/O lines with match and capture.
- **Priorities** of channels.
- **Timers** : 2 · 16 bit time-bases.

☞ Associate *functions*, *time-bases*, *channels* and *priorities* … and let it run!



Micro-controller examples

TPU Fixed scheduled, prioritized time slots

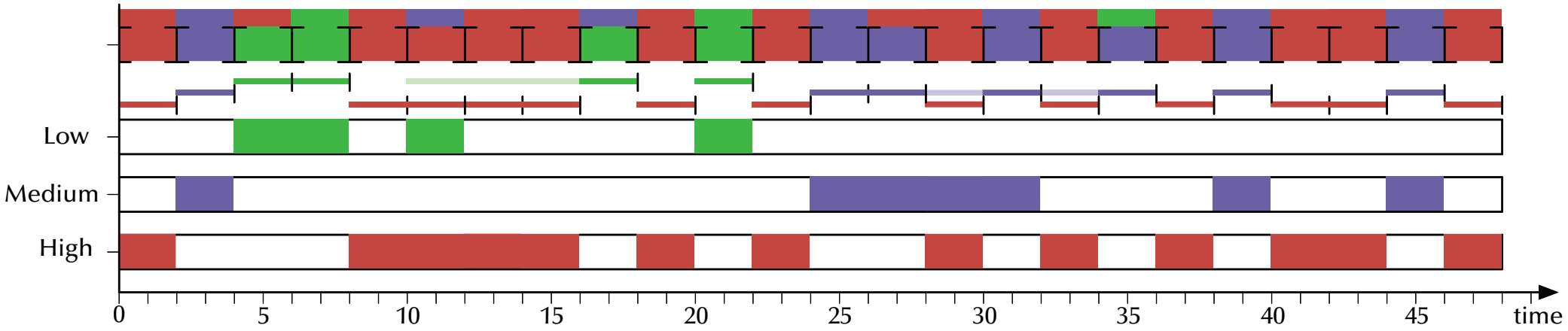


Round Robin schedule for all runnable states inside each priority.

☞ No state will be starved.

Micro-controller examples

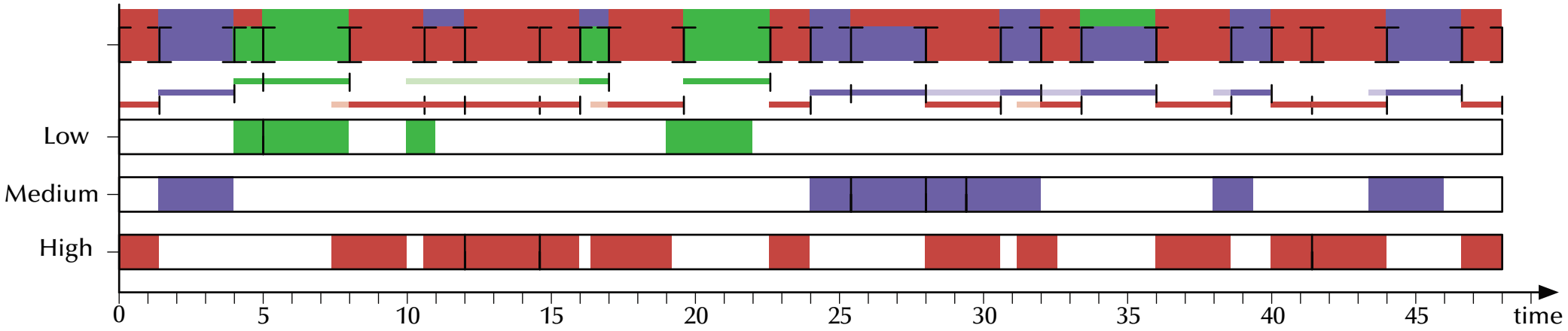
TPU Fixed scheduled, prioritized time slots



Unused slots will be re-assigned according to priorities and channel numbers.

Micro-controller examples

TPU Fixed scheduled, prioritized time slots



States have different and variable lengths.

☞ Calculating actual and maximal latencies requires full understanding of all states.

Latencies on TPUs

… for latencies of capture and match at each channel mind:

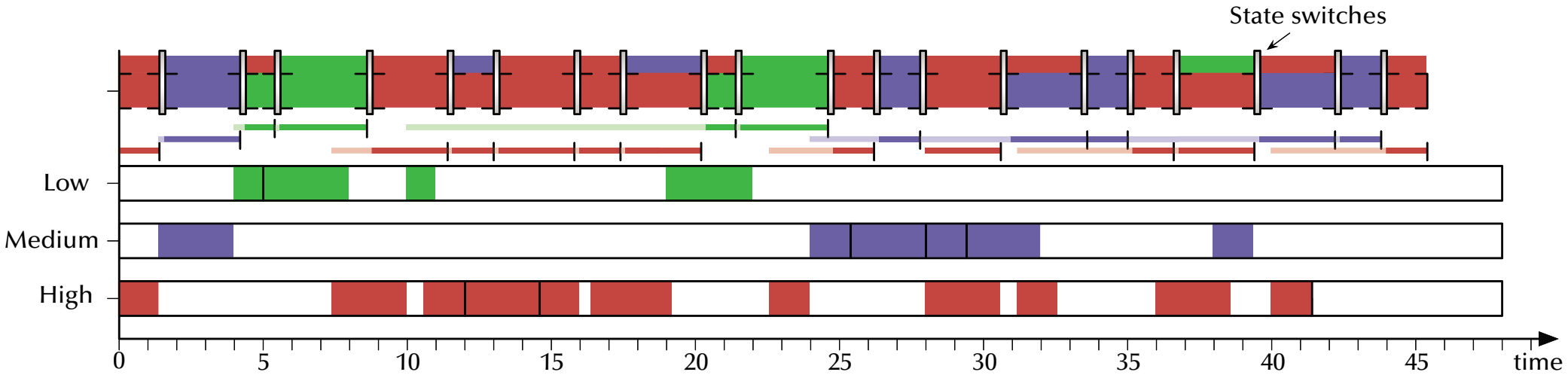
- only the time-base resolution (all channels are evaluated independently and in parallel).

… for the functions associated with individual channels mind the:

- number of active channels (max. 16).
- number of channels on each priority level (add max. 2 µ-cycles for each “state-switch”).
- number of available time slots on each priority level per full scheduler-cycle (4, 2, 1 slot{s}).
- number of µ-cycles to execute individual states of a function (2 µ-cycles per µ-instruction).
- number of RAM accesses during the execution of a state
  (each access may stall for 2 CPU-cycles).
- TPU clock cycle frequency.

Micro-controller examples

Determining actual TPU latencies

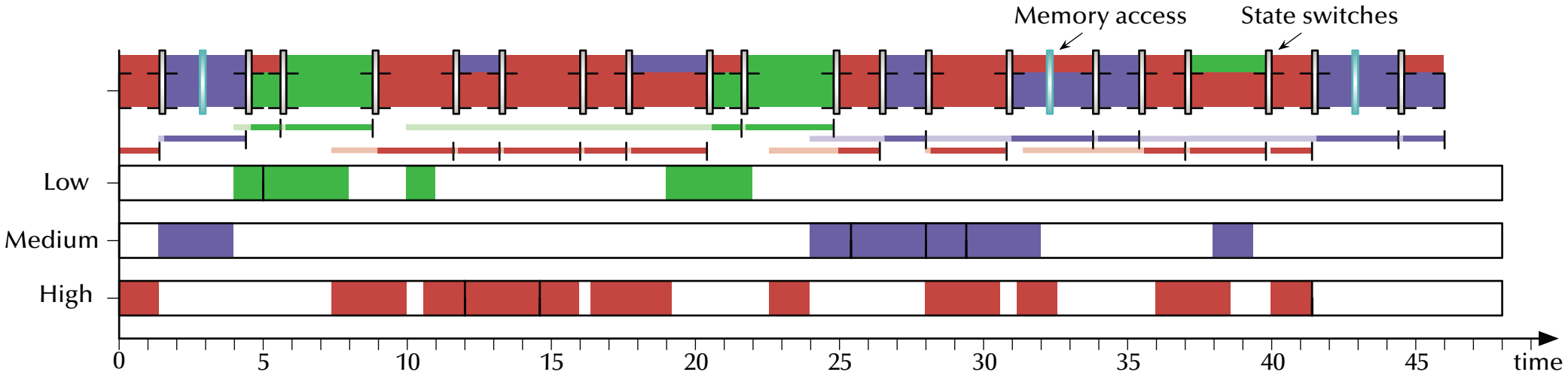


Emulate the known executing ready times for all states.

☞ Add two 2 µ-cycles for each state switch.

Micro-controller examples

Determining actual TPU latencies

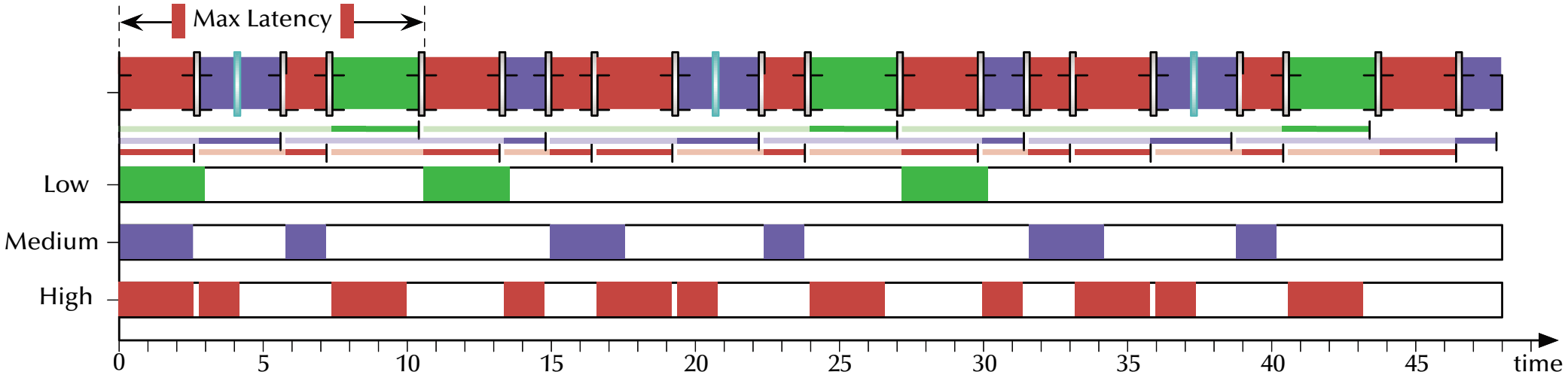


Emulate the known memory access patterns for all states.

☞ Add two 2 µ-cycles for each memory access (potential stall times).

Micro-controller examples

Determining maximal TPU latencies

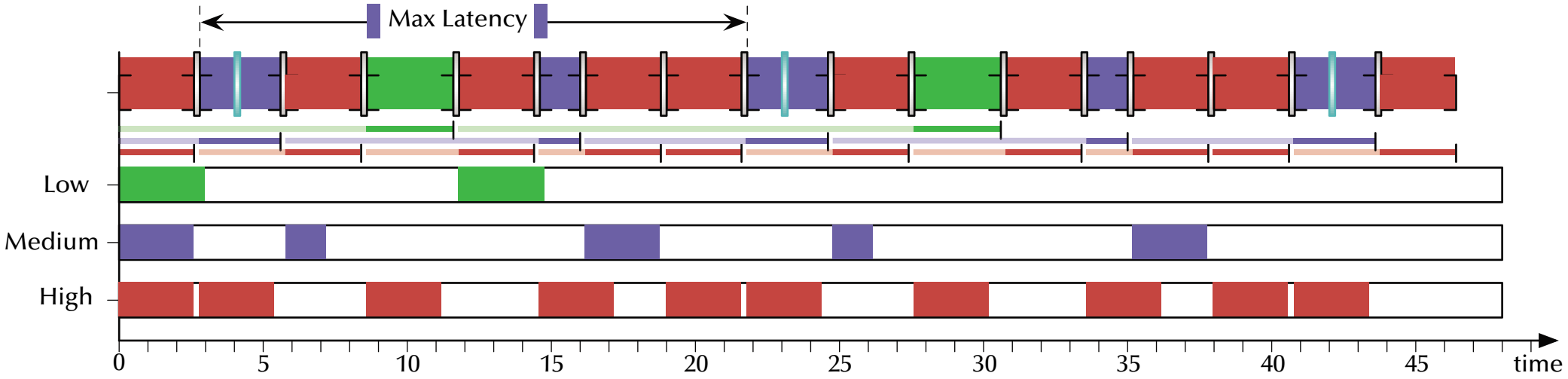


- ☞ Assume all states on the same priority runnable at all times and run their maximal lengths.
- ☞ Assume the longest state out of all higher priorities runnable at all times.
- ☞ Assume the longest states on each lower priority runnable at all times.
  Deploy a set out of those states which will cause the longest latency.

☞ Determine the longest latency inside a full hyper-cycle.

Micro-controller examples

Determining maximal TPU latencies

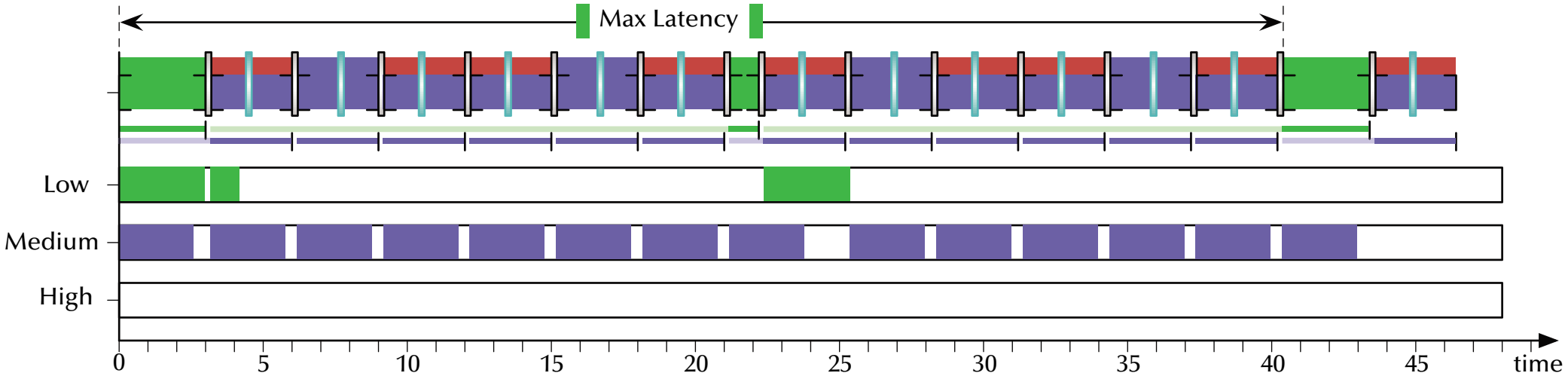


- ☞ Assume all states on the same priority runnable at all times and run their maximal lengths.
- ☞ Assume the longest state out of all higher priorities runnable at all times.
- ☞ Assume the longest states on each lower priority runnable at all times.
  Deploy a set out of those states which will cause the longest latency.

☞ Determine the longest latency inside a full hyper-cycle.

Micro-controller examples

Determining maximal TPU latencies



- ☞ Assume all states on the same priority runnable at all times and run their maximal lengths.
- ☞ Assume the longest state out of all higher priorities runnable at all times.
- ☞ Assume the longest states on each lower priority runnable at all times.
  Deploy a set out of those states which will cause the longest latency.

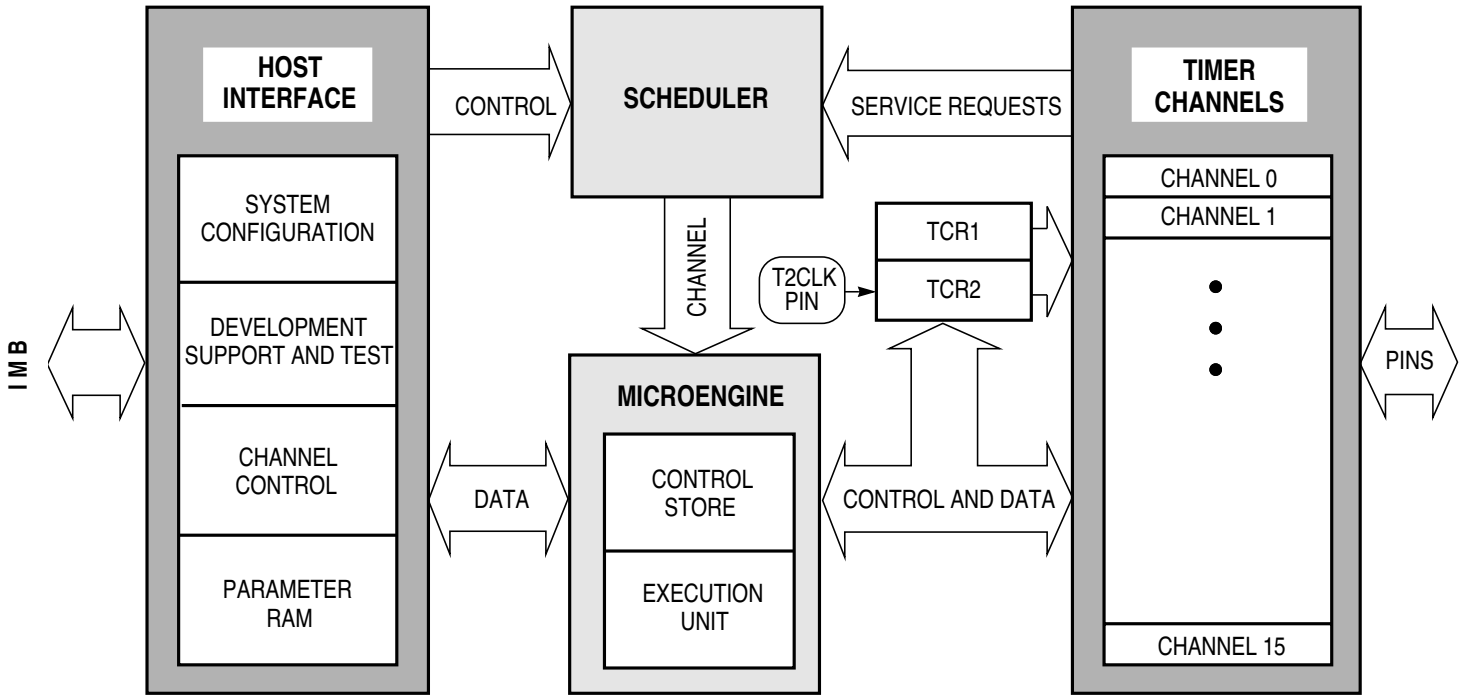
☞ Determine the longest latency inside a full hyper-cycle.

Micro-controller examples

Time Processing Unit

A special-purpose micro-controller:

- Independent µ-engine.
- 16 digital I/O channels with independent match and capture capabilities.
- Meant to operate these I/O channels for timing control purposes.
- Predefined µ-engine command set (ROM functions in control store).
- 2 · 16 bit time bases



Micro-controller examples

MPC565 : Time-base & Real-time clock

- **64 bit time base**
  (driven by an external clock: e.g. 20 MHz ☞ resolution: 50 ns; range: ~30,000 years).

- **Free running**
  (not influenced by any CPU action or resets).

- **2 reference registers**
  (used for compares and interrupt generation).

- **Real-Time clock**
  supplies full seconds (32 bit ☞ range: ~136 years)
  (not affected by CPU, resets, and operates in all low-power modes).

Micro-controller examples

MPC565 : Interrupt controller

- Handles up to 48 different sources
  (32 from internal modules, 8 from timers and clocks, and 8 external vectorized sources) and supply each of them with a unique interrupt-vector

- 8 interrupt levels are distinguished by the interrupt controller
  (32 interrupt levels are supplied by the internal modules, prioritized and vectorized interrupts are supplied by external sources)

- Latency: 20 clock cycles
  + bus collisions + CPU state saving + tasking system overhead

Micro-controller examples

MPC565 : Nexus debug port (IEEE-ISTO 5001-1999)

(Real-time embedded application development interface)

On-Line mode:

- **Program trace**: via branch trace messaging.
- **Data trace**: via data write messaging and data read messaging (can be reduced to selected areas).
- **Owner trace**: via ownership trace messaging (also indicates task creation and activation).
- **Run-time access** to memory map and special CPU registers.
- **Watchpoints**: CPU watchpoint status signals are snooped and transferred with high priority.

Off-line mode:

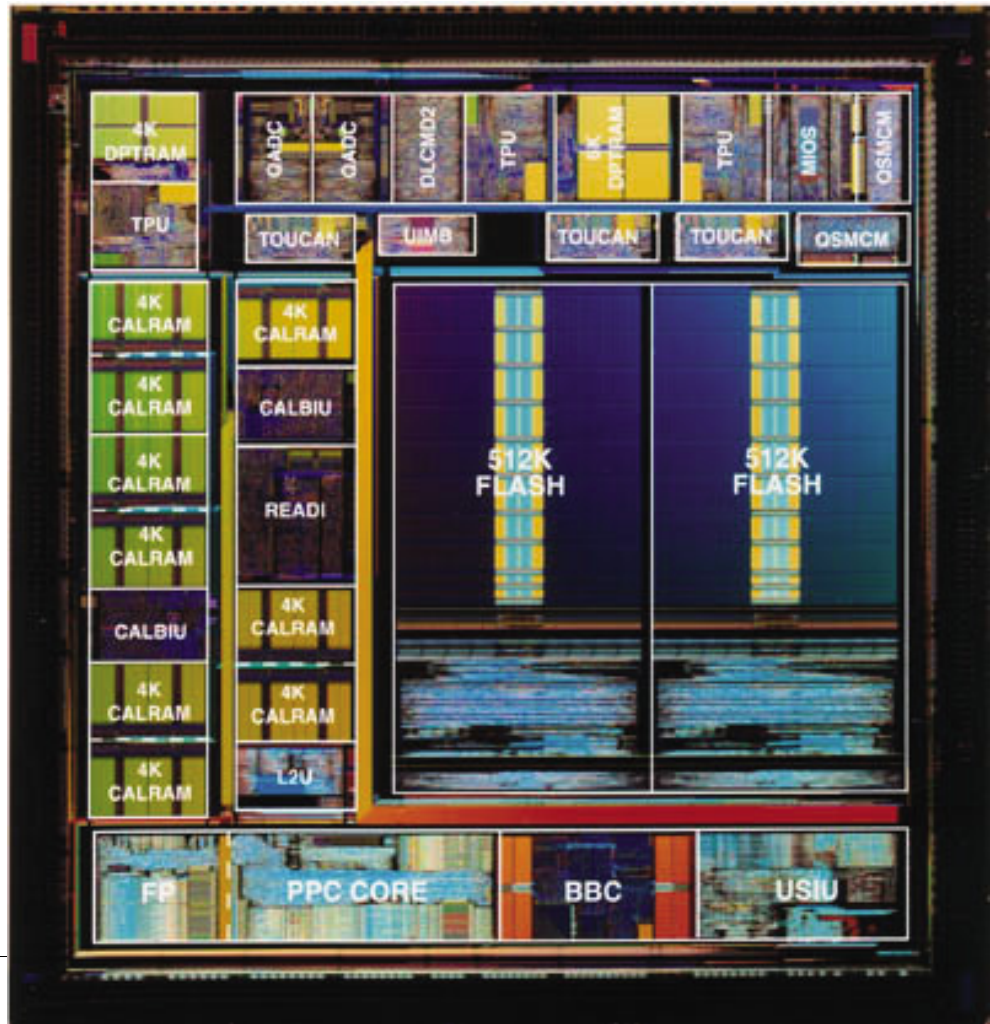
- **Read / Write access**: the READI module can take over the L-bus to manipulate data.
- **Access to all CPU registers** during halt.

Interfaces

Micro-controller examples

MPC565

- **Power**: power dissipation: 0.8 - 1.12 W, -40° - +125°C
- **CPU**: PowerPC core (incl. FPU & BBC), 56 MHz
- **RAM**: flash: 1 MB, static: 36 kB
- **Time processing units**: 3 (via dual-ported RAM)
- **Timers**: 22 channels (PWM & RTC supported)
- **A/D** convertors: 40 channels, 10 bit, 250 kHz
- **Can-bus**: 3 TOUCAN modules
- **Serial**: 2 interfaces
- **Data link controller**:
  SAE J1850 class B communications module
- **Real-time embedded application development interface**: NEXUS debug port (IEEE-ISTO 5001-1999)
- **Packing**: 352/388 ball PBGA



Interface architectures

Basic sampling control mechanisms

- **Status driven**: the computer polls for information (used in dedicated micro-controllers and pre-scheduled hard real-time environments).
- **Interrupt driven**: The data generating device may issue an interrupt when new data had been detected / converted or when internal buffers are full.
  - **Program controlled**: The interrupts are handled by the CPU directly (by changing tasks, calling a procedure, raising an exception, free tasks on a semaphore, sending a message to a task, …).
  - **Program initiated**: The interrupts are handled by a DMA-controller. No processing is performed. Depending on the DMA setup, cycle stealing can occur and needs to be considered for the worst case computing times.
  - **Channel program controlled**: The interrupts are handled by a dedicated channel device. The data is transferred and processed. Optional memory-based communication with the CPU. ☞ The channel controller is usually itself a dedicated µ-engine / -controller.

Interface architectures

Handling device responses

Responses from devices can be:

- Immediate.
- With a constant delay or within a defined time-frame.
- Unpredictable / sporadic.

Device handlers may thus:

- Perform a 'busy-wait' for the response.
- Reschedule the device-process by a constant delay.
- Schedule the device-process periodically and employ different time-slots for sending control / data and receiving status / data.
- React to triggers / calls / interrupts from the device.

- The device handler can be implemented as a *process* / *interrupt routine* / *dedicated µ-controller* / *DMA-controller* or a mixture of those.

Interface architectures

Handling device responses

☞ How to embed the unpredictable in predictable systems?

By providing the resources to cope with the assumed worst case
(and fall back to a lower, yet safe functionality beyond that).

Concrete:

- Either the unpredictable events need to be **synchronized** with the remaining real-time tasks without violating real-time constraints.
- Or **exclusive processing resources** (e.g. a dedicated micro-controller) for a specific device need to be provided.

Interface architectures

Language requirements for interfaces

- **Specify the device interface (protocol and formats) in all detail**
  (candidates: Ada, CHILL, ERLANG, Modula-2, C, …
  …or Macro-Assemblers level (if platform-independence or abstraction is not required)).
- **Handling asynchronous hardware messages (devices, timers, …)**
  Many different methods to implement a context-switch
  (candidates: all languages with some real-time orientation:
  PEARL, CHILL, ERLANG, Ada, RT-Java, POSIX, …)

- The term “high-level languages” in the real-time interface context:

Allow for
strong abstractions *while* being time and physics specific
down to the actual level of interface realities

Summary

Converters & Interfaces

- **Analogue signal chain in a digital system**
  - Sampling data, aliasing, Nyquist's criterion, oversampling
  - Quantization (LSB, rms noise voltage, SNR, ENOB), Missing codes, DNL, INL
- **A/D converters:**
  - Integrating (Single- / Dual-slope), Flash, Pipelined, SAR, Tracking,
    Σ-Δ , Σ-Δ DDA, n-th order Σ-Δ**.**
- **Examples:**
  - Fast and simple A/D converter example: National Semiconductor ADC08200
  - Multi-channel A/D data logging interface example: National Semiconductor LM12L458
  - Simple 8-bit µ-controller example: Motorola MC68HC05, Propeller.
  - Complex 32-bit µ-controller examples: AVR32 and Motorola MPC565 (including TPUs).
- **General device handling / sampling control / language requirements**