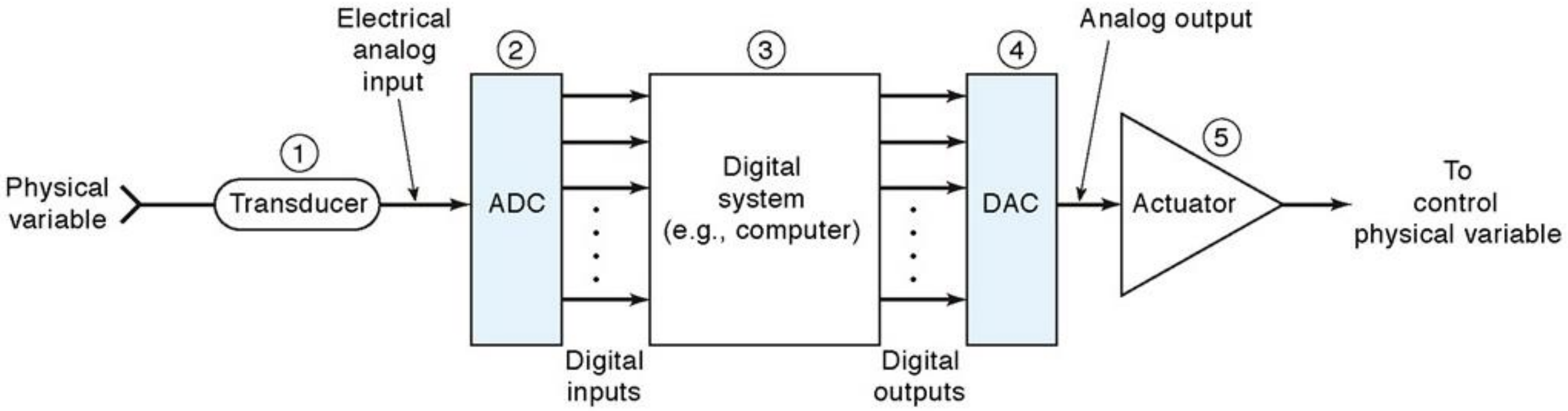
Module 5 - Analog Interfacing

Interfacing With the Analog World

- Transducer
- ADC
- Computer
- DAC
- Actuator



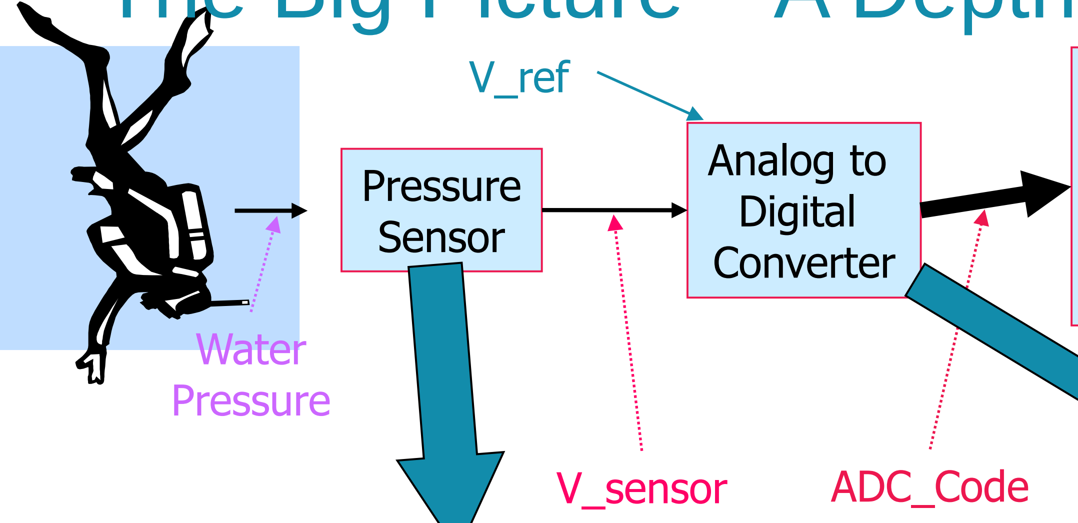
Why It's Needed

- Embedded systems often need to measure values of physical parameters
- These parameters are usually continuous (*analog*) and not in a digital form which computers (which operate on discrete data values) can process

- Temperature
  - Thermometer (do you have a fever?)
  - Thermostat for building, fridge, freezer
  - Car engine controller
  - Chemical reaction monitor
  - Safety (e.g. microprocessor processor thermal management)
- Light (or infrared or ultraviolet) intensity
  - Digital camera
  - IR remote control receiver
  - Tanning bed
  - UV monitor
- Rotary position
  - Wind gauge
  - Knobs
- Pressure
  - Blood pressure monitor
  - Altimeter
  - Car engine controller
  - Scuba dive computer
  - Tsunami detector
- Acceleration
  - Air bag controller
  - Vehicle stability
  - Video game remote
- Mechanical strain
- Other
  - Touch screen controller
  - EKG, EEG
  - Breathalyzer

CONVERTING BETWEEN ANALOG AND DIGITAL VALUES

The Big Picture – A Depth Gauge




```
// Your software
ADC_Code = ADC0->R[0];
V_sensor = ADC_code*V_ref/1023;
Pressure_kPa = 250 * (V_sensor/V_supply+0.04);
Depth_ft = 33 * (Pressure_kPa –  Atmos_Press_kPa)/101.3;
```

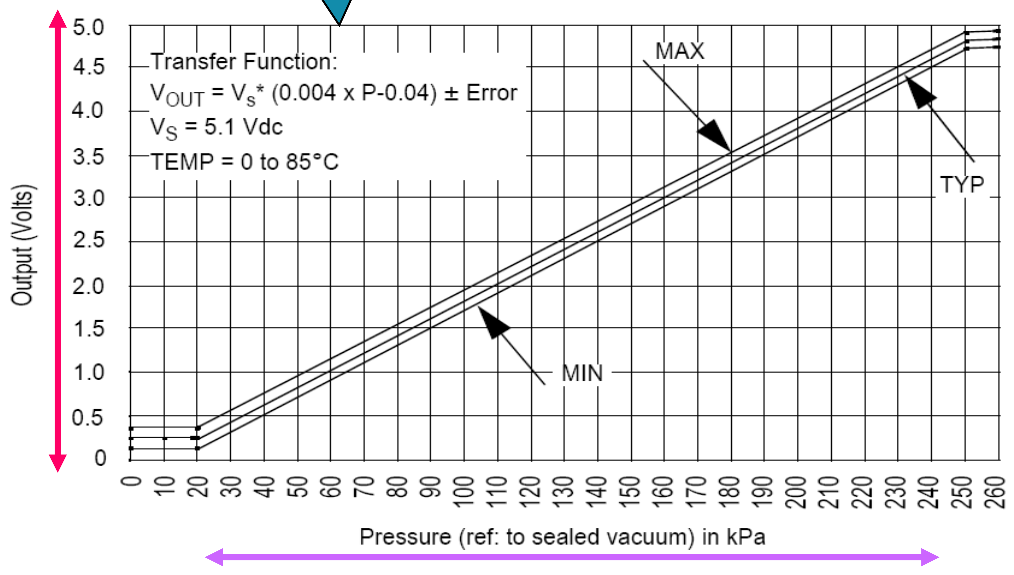
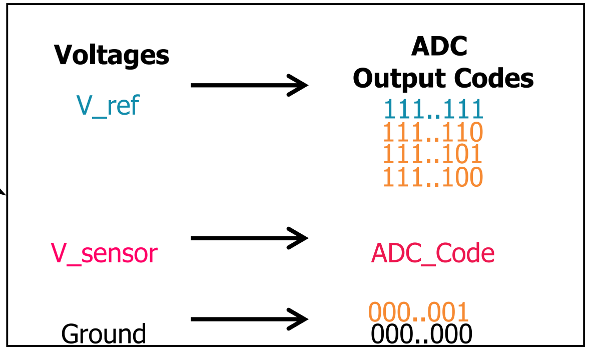


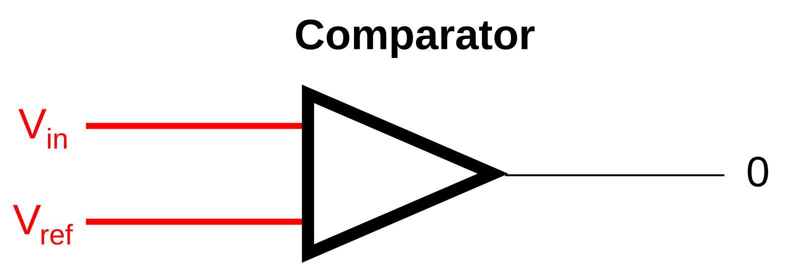
Figure 4. Output vs. Absolute Pressure



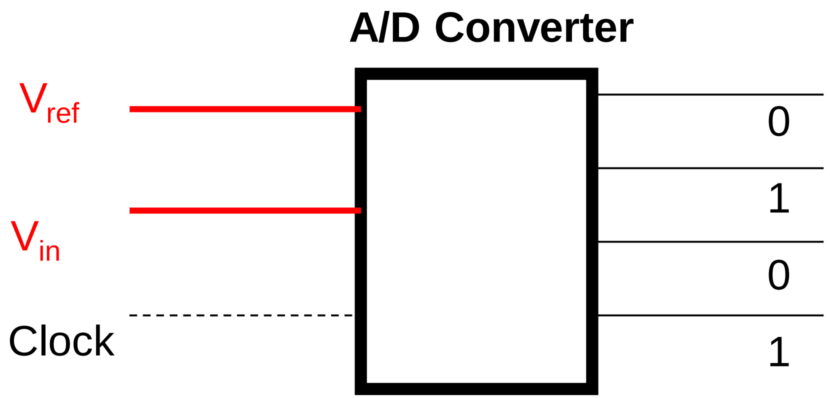
1. Sensor detects *water pressure* and generates a proportional *output voltage* V_sensor
2. ADC generates a proportional digital *integer* (code) based on V_sensor and V_ref
3. Code can convert that integer to a something more useful
    1. first a float representing the *voltage*,
    2. then another float representing *pressure*,
    3. finally another float representing *depth*

Getting From Analog to Digital

- A **Comparator** tells us "Is $V_{in} > V_{ref}$?"
  - Compares an **analog input voltage** with an **analog reference voltage** and determines which is larger, returning a 1-bit number
  - E.g. Indicate if depth > 100 ft
  - Set V_{ref} to voltage pressure sensor returns with 100 ft depth.

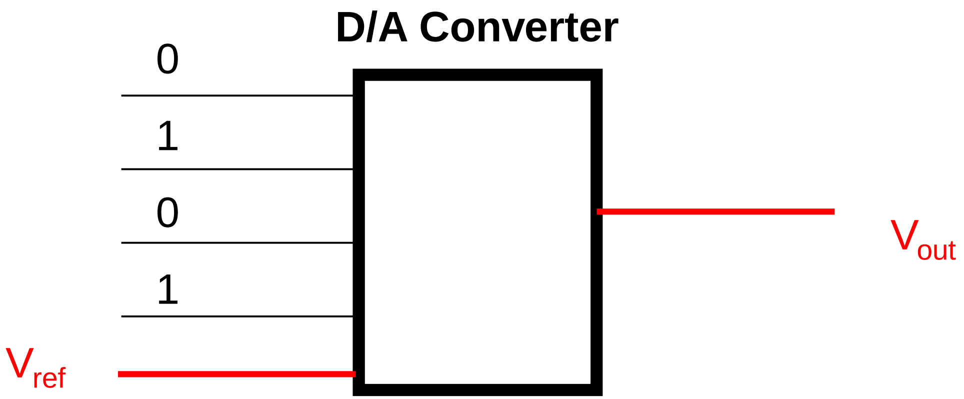


- An **Analog to Digital converter** [AD or ADC] tells us how large V_{in} is as a fraction of V_{ref}.
  - Reads an analog input signal (usually a voltage) and produces a corresponding multi-bit number at the output.
  - E.g. calculate the depth

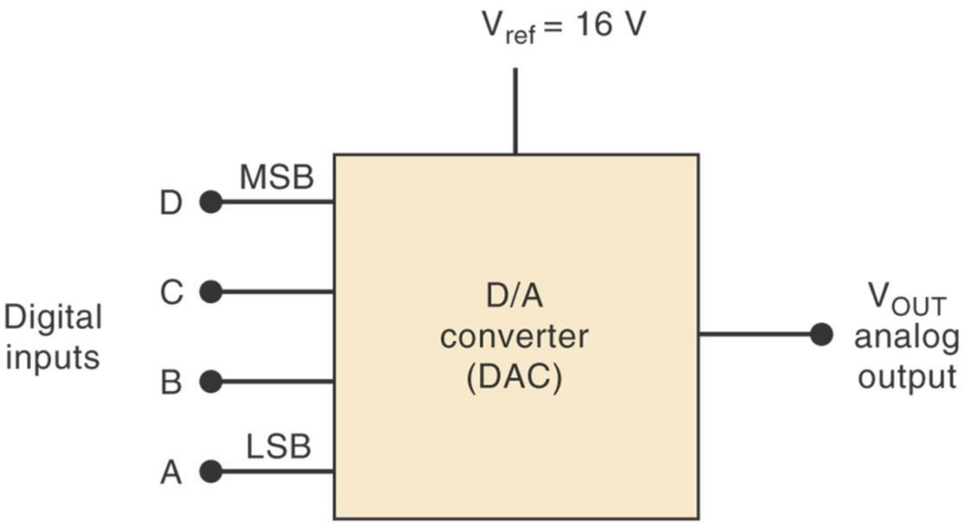


Digital to Analog Conversion

- May need to generate an analog voltage or current as an output signal
  - E.g. audio signal, video signal brightness.
- DAC: "Generate the analog voltage which is this fraction of V_{ref}"
- Digital to Analog Converter equation
  - n = input code
  - N = number of bits of resolution of converter
  - V_{ref} = reference voltage
  - V_{out} = output voltage. Either
    - $V_{out} = V_{ref} * n/(2^N)$ *or*
    - $V_{out} = V_{ref} * (n+1)/(2^N)$
    - *The offset +1 term depends on the internal tap configuration of the DAC – check the datasheet to be sure*



4-bit DAC

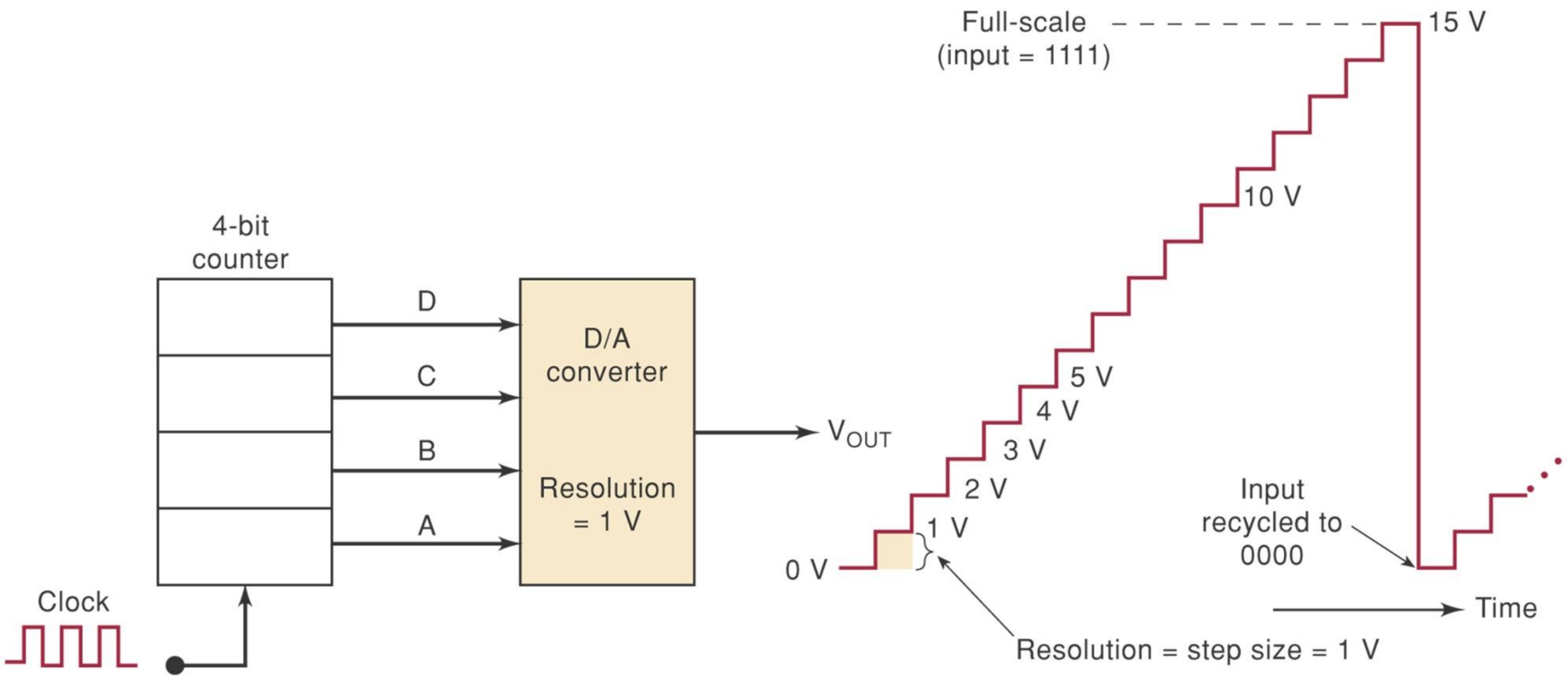


(a)

D	C	B	A	V_{OUT}	
0	0	0	0	0	Volts
0	0	0	1	1	
0	0	1	0	2	
0	0	1	1	3	
0	1	0	0	4	
0	1	0	1	5	
0	1	1	0	6	
0	1	1	1	7	
1	0	0	0	8	
1	0	0	1	9	
1	0	1	0	10	
1	0	1	1	11	
1	1	0	0	12	
1	1	0	1	13	
1	1	1	0	14	
1	1	1	1	15	Volts

(b)

More



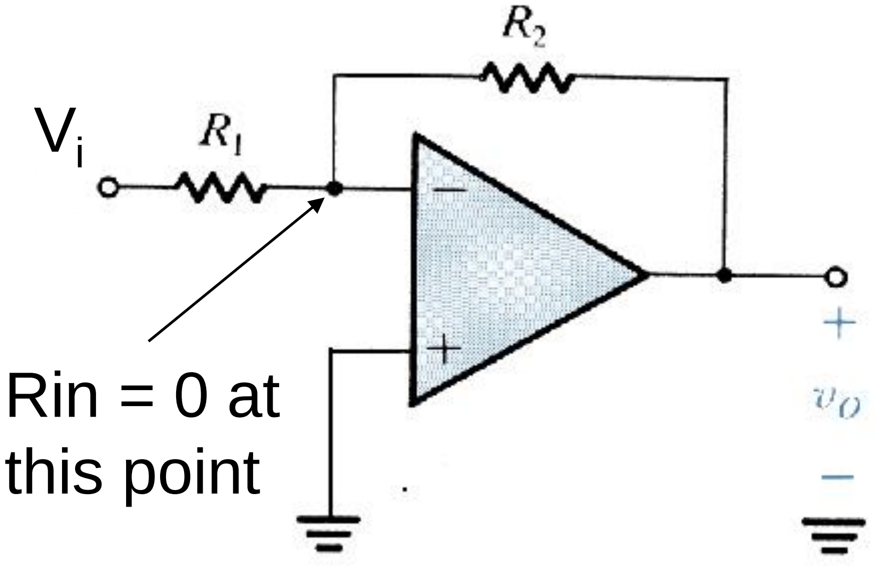
Nominal Full-scale value = 16 V

Resolution = Step Size = LSB = 16V / 2^4 = 1 V

Full scale output = Nominal Full-scale value – Step Size = 15 V

Summary of Op-amp Behavior

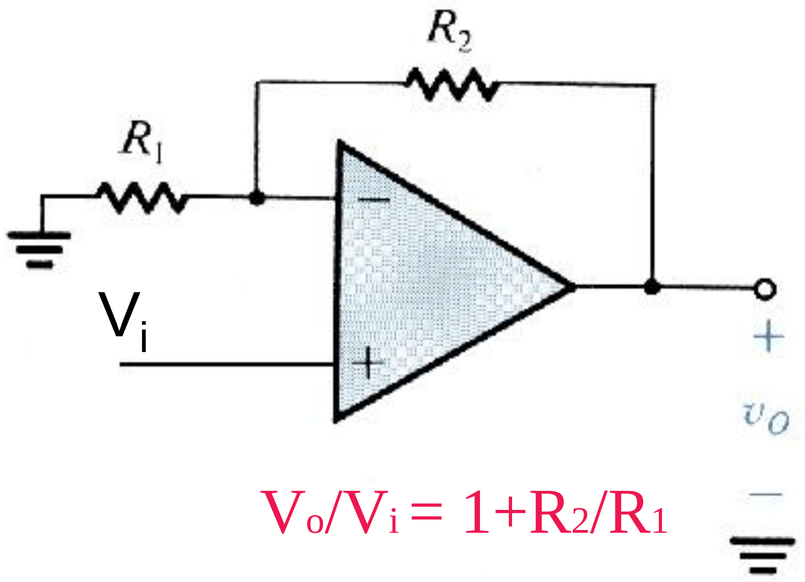
Inverting configuration



$V_o / V_i = - R_2/R_1$

$R_{in} = R_1$

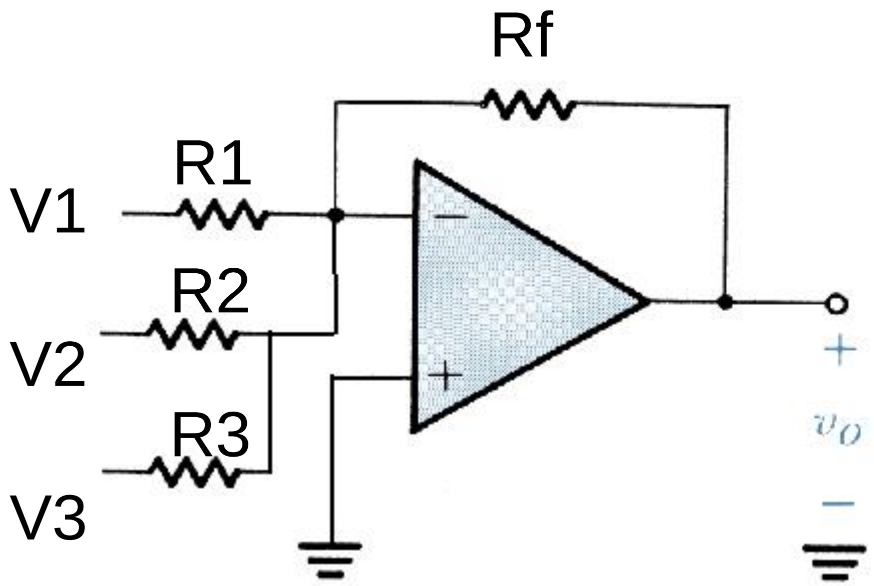
Noninverting configuration



$V_o/V_i = 1+R_2/R_1$

R_{in} = infinity

The Weighted Summer



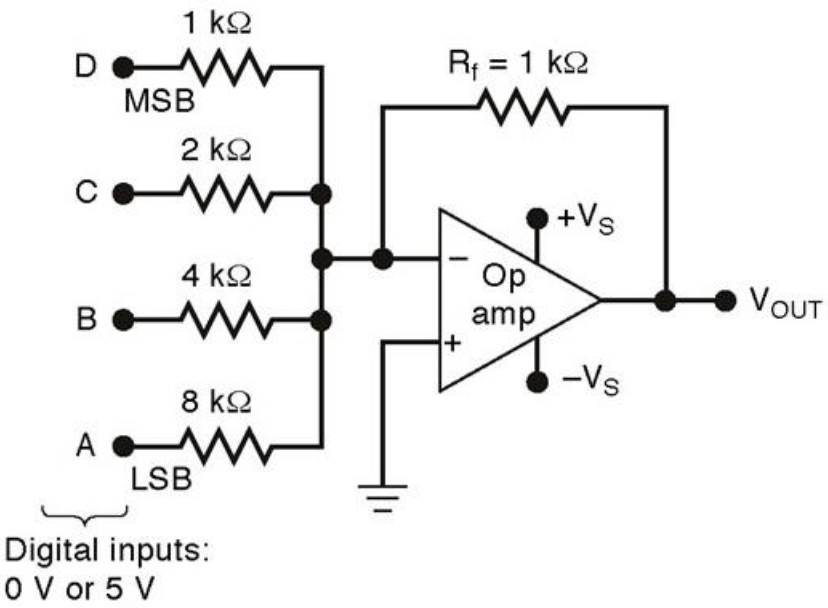
Current in R_1, R_2, and R_3 add to current in R_f

$V_o = -R_f(V_1/R_1 + V_2/R_2 + V_3/R_3)$

This circuit is called a weighted summer

D/A Converters

- A summing operational amplifier with a resolution of 0.625 V

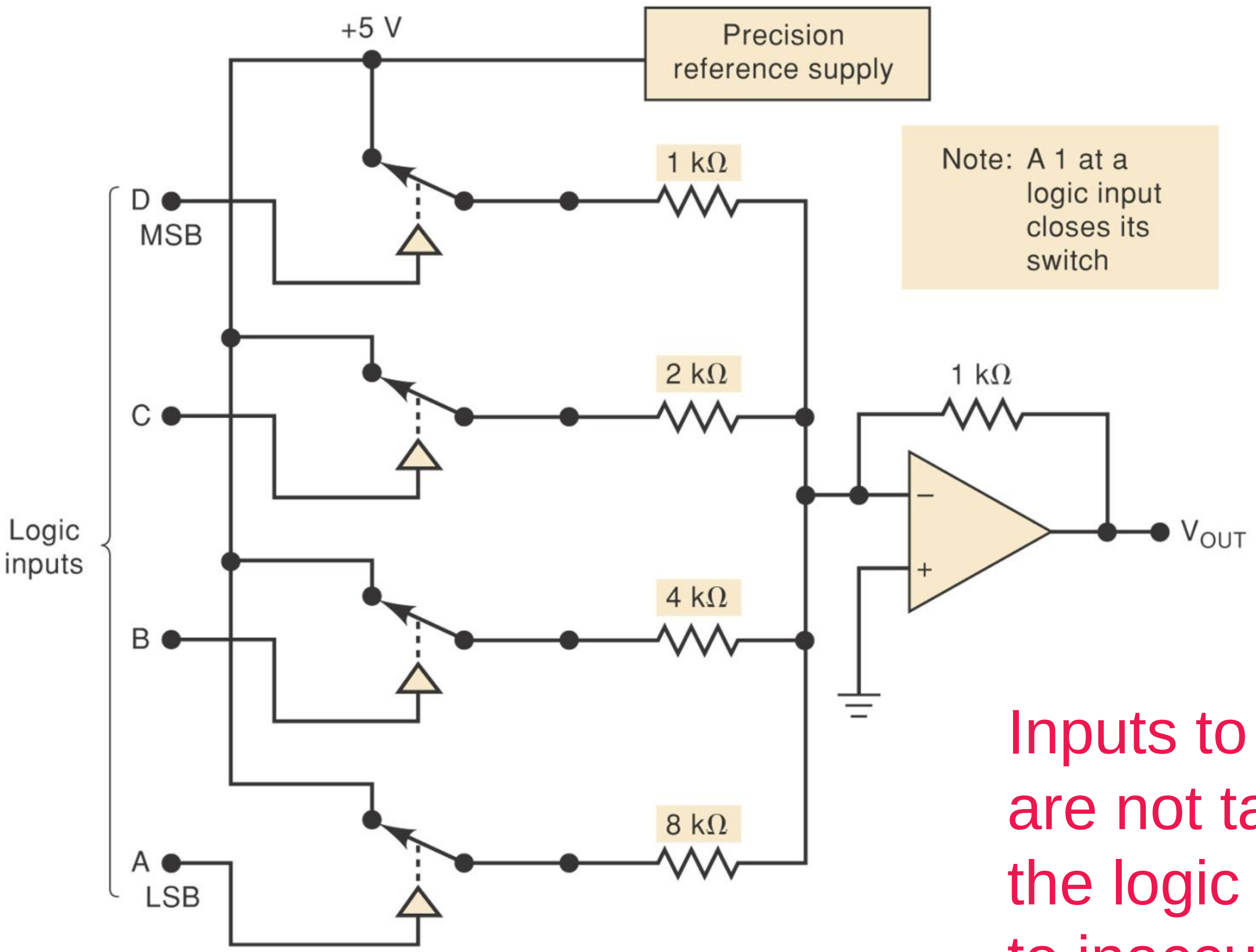


Input code D	C	B	A	V_{OUT} (volts)	
0	0	0	0	0	
0	0	0	1	–0.625	← LSB
0	0	1	0	–1.250	
0	0	1	1	–1.875	
0	1	0	0	–2.500	
0	1	0	1	–3.125	
0	1	1	0	–3.750	
0	1	1	1	–4.375	
1	0	0	0	–5.000	
1	0	0	1	–5.625	
1	0	1	0	–6.250	
1	0	1	1	–6.875	
1	1	0	0	–7.500	
1	1	0	1	–8.125	
1	1	1	0	–8.750	
1	1	1	1	–9.375	← Full-scale

Resolution = |5V(1K/8K)| = .625V

Max out = 5V(1K/8K + 1K/4K + 1K/2K + 1K/1K) = -9.375V

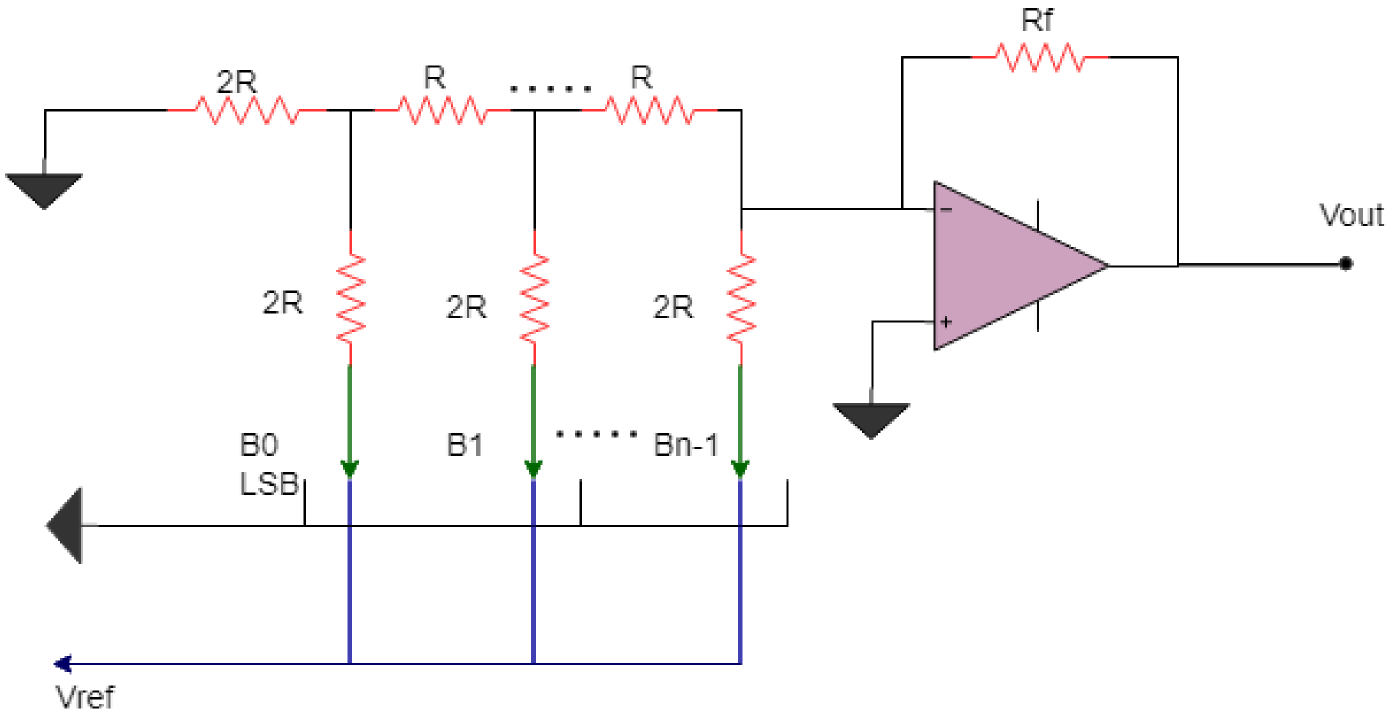
Complete Four-bit DAC Including a ***Precision*** Reference Supply.



Inputs to the DAC are not taken from the logic inputs due to inaccuracies.

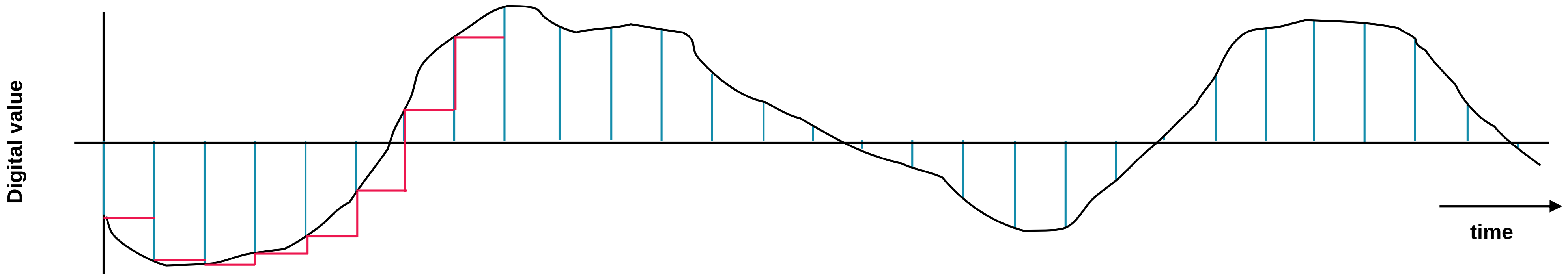
Basic *R*/2*R* ladder DAC

B is the value of the binary input



https://microcontrollerslab.com/r-2r-ladder-dac-digital-to-analog-converter-working-examples-circuits/

Waveform Sampling and Quantization



- A waveform is **sampled** at a constant rate – every Δt
  - Each such sample represents the instantaneous amplitude at the instant of sampling
  - *“At 37 ms, the input is 1.91341914513451451234311… V”*
  - Sampling converts a **continuous time** signal to a **discrete time** signal

- The sample can now be **quantized** (converted) into a digital value
  - Quantization represents a **continuous** (analog) value with the closest **discrete** (digital) value
  - *“The sampled input voltage of 1.91341914513451451234311… V is best represented by the code 0x018, since it is in the range of 1.901 to 1.9980 V which corresponds to code 0x018.”*

Forward Transfer Function Equations

What code n will the ADC use to represent voltage V_{in}?

General Equation

n = converted code

V_{in} = sampled input voltage

V_{+ref} = upper voltage reference

V_{-ref} = lower voltage reference

N = number of bits of resolution in ADC

$$n = \left\lfloor \frac{(V_{in} - V_{-ref})\, 2^N}{V_{+ref} - V_{-ref}} + 1/2 \right\rfloor$$

Simplification with V_{-ref} = 0 V

$$n = \left\lfloor \frac{(V_{in})\, 2^N}{V_{+ref}} + 1/2 \right\rfloor$$

$$n = \left\lfloor \frac{3.30\text{V} \;\; 2^{10}}{5\text{V}} + 1/2 \right\rfloor \quad = 676$$

$\lfloor X \rfloor = I$ *floor function: nearest integer I such that I <= X*

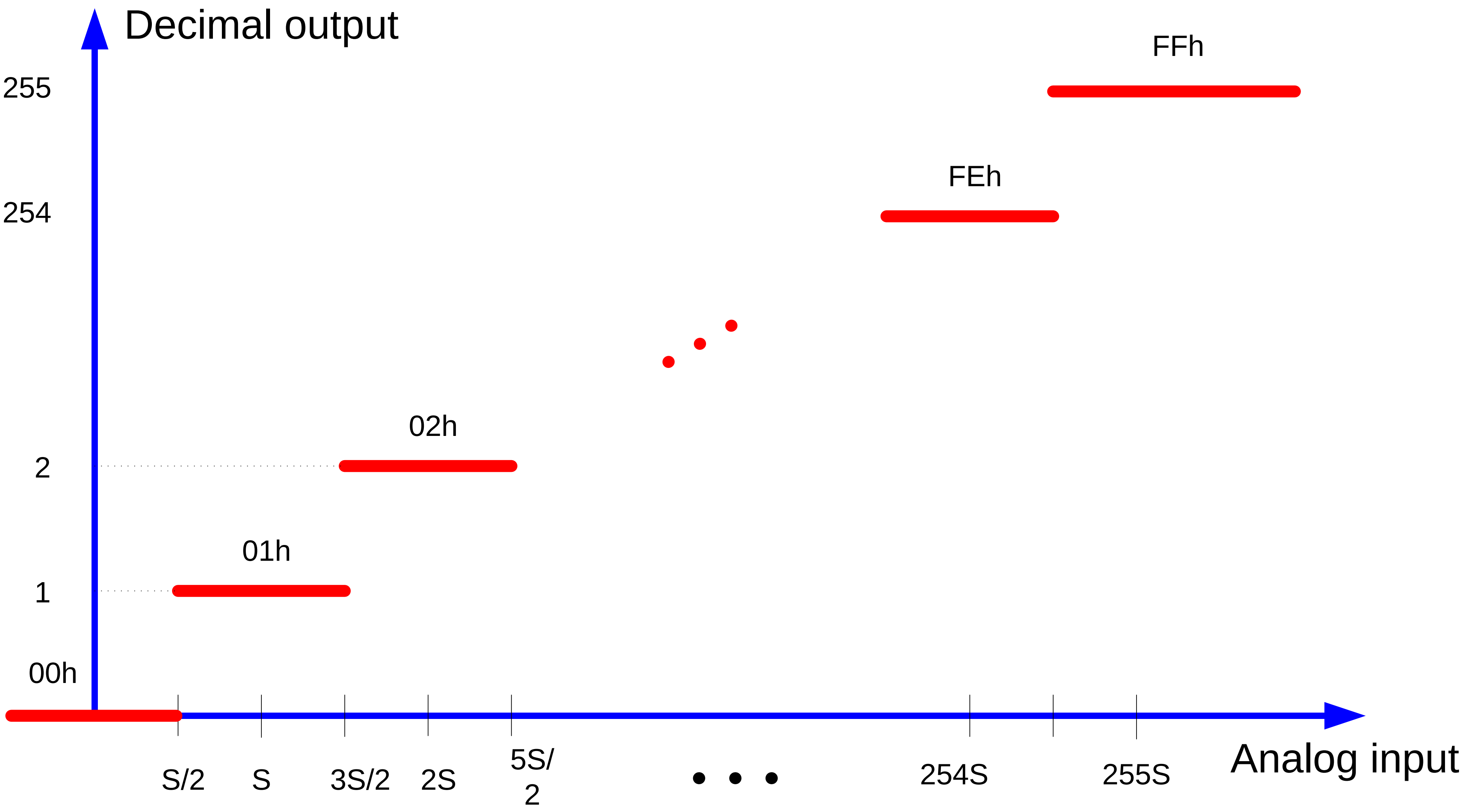
floor(x+0.5) rounds x to the nearest integer

ARM

Ideal ADC Example N=8

$$V_{-ref} = 0V$$

$$\text{Step Size (SS)} = \frac{V_{+ref}}{2^N}$$



Inverse Transfer Function

What range of voltages V_{in_min} to V_{in_max} does code n represent?

General Equation

n = converted code

V_{in_min} = minimum input voltage for code n

V_{in_max} = maximum input voltage for code n

V_{+ref} = upper voltage reference

V_{-ref} = lower voltage reference

N = number of bits of resolution in ADC

$$V_{in\_min} = \frac{n - \frac{1}{2}}{2^N}\left(V_{+ref} - V_{-ref}\right) + V_{-ref}$$

$$V_{in\_max} = \frac{n + \frac{1}{2}}{2^N}\left(V_{+ref} - V_{-ref}\right) + V_{-ref}$$

Simplification with V_{-ref} = 0 V

$$V_{in\_min} = \frac{n - \frac{1}{2}}{2^N}\left(V_{+ref}\right)$$

$$V_{in\_max} = \frac{n + \frac{1}{2}}{2^N}\left(V_{+ref}\right)$$

What if the Reference Voltage is not known?

- Example - running off an unregulated battery (to save power)
- Measure a known voltage and an unknown voltage

$$V_{unknown} = V_{known} \frac{n_{unknown}}{n_{known}}$$

- Many MCUs include an internal fixed voltage source which ADC can measure for this purpose
- Can also solve for Vref

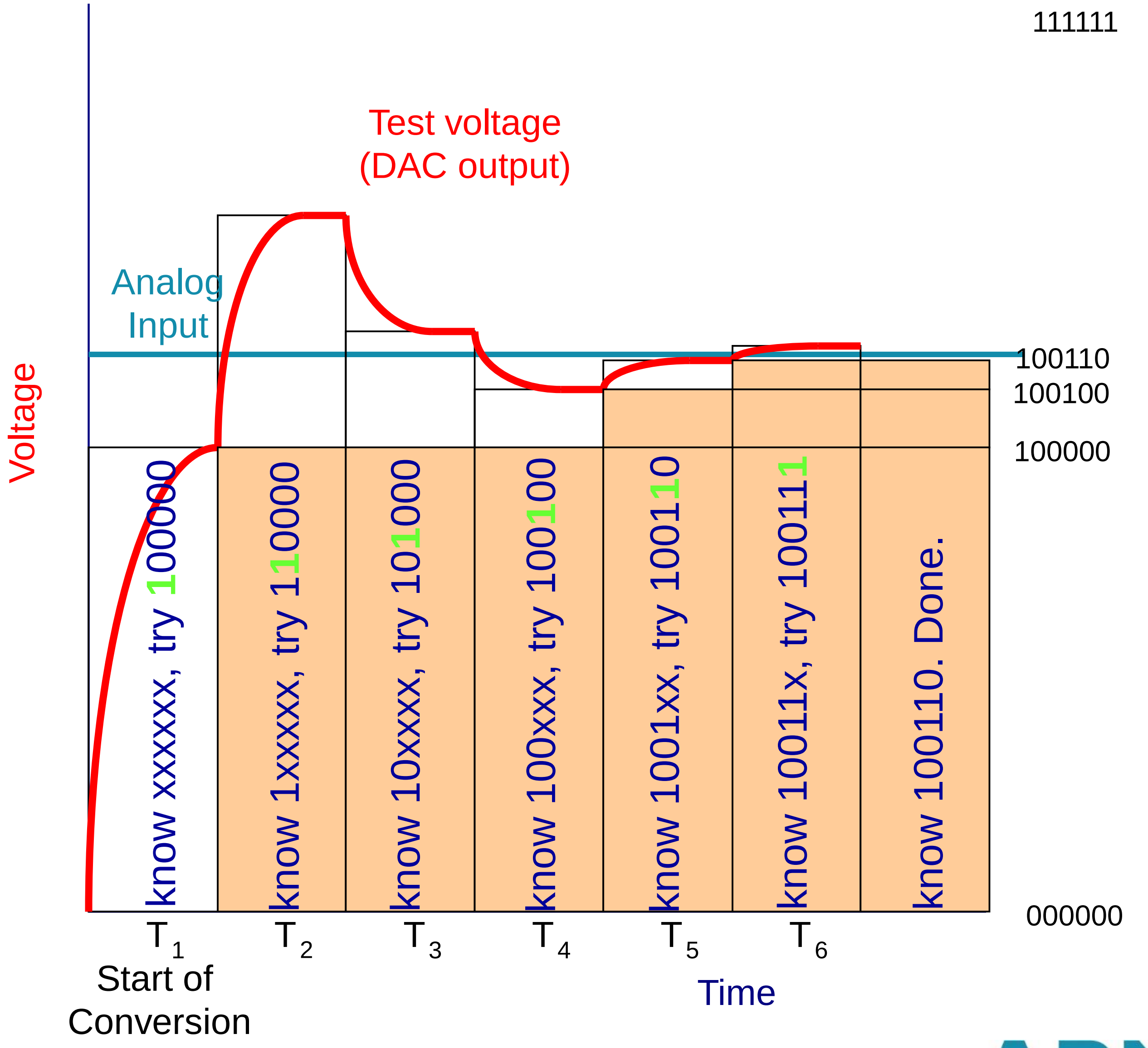
$$V_{ref} = V_{known} \frac{2^N}{n}$$

"My ADC tells me that channel 27 returns a code of 0x6543,
so I can calculate that V_{REFSH} = 1.0V * 2^{16}/0x6543 = …

ANALOG TO DIGITAL CONVERSION CONCEPTS

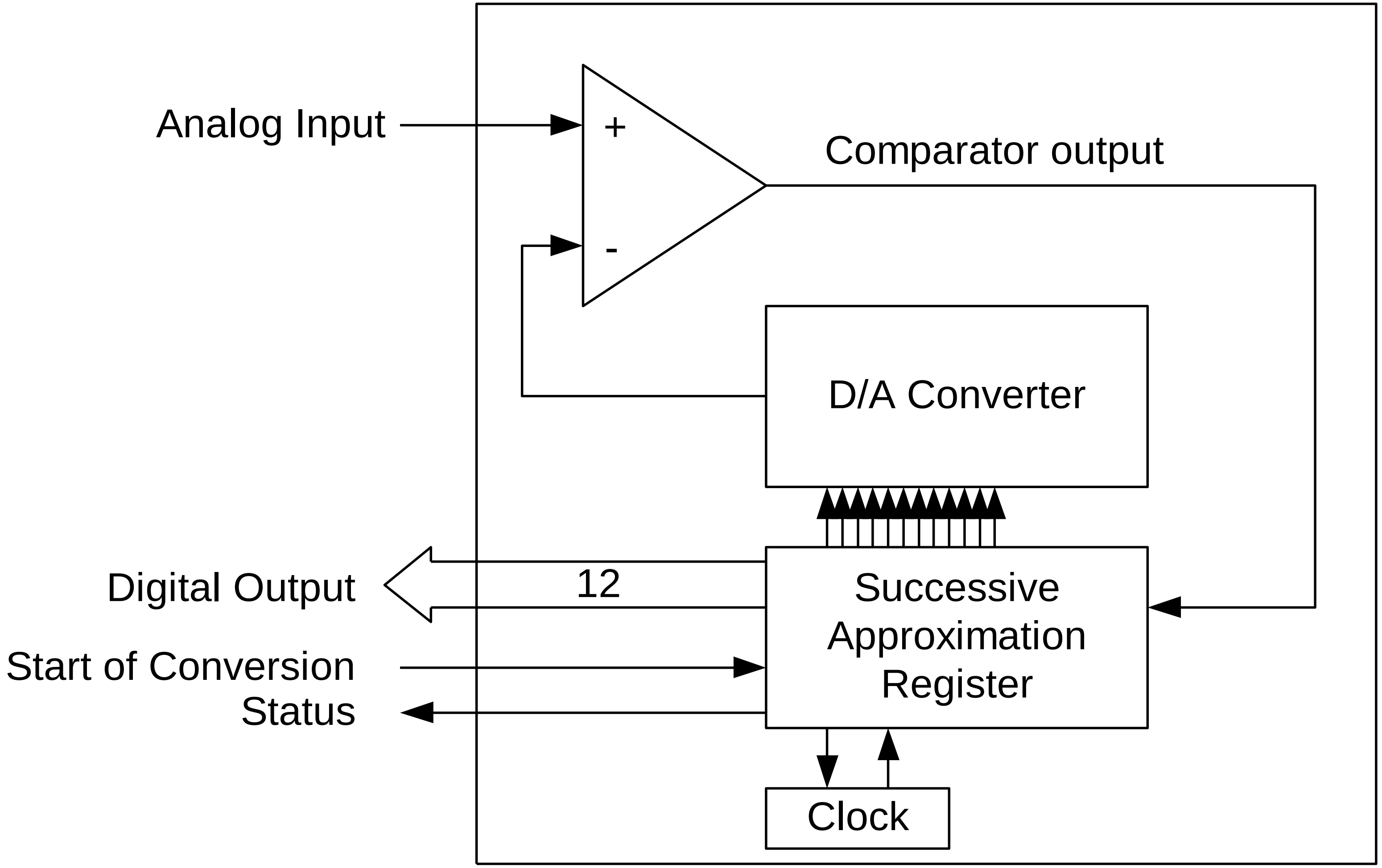
ADC - Successive Approximation Conversion

- Successively approximate input voltage by using a binary search and a DAC
- SA Register holds current approximation of result
- Set all DAC input bits to 0
- Start with DAC's most significant bit
- Repeat
  - Set next input bit for DAC to 1
  - Wait for DAC and comparator to stabilize
  - If the DAC output (test voltage) is **smaller** than the input then set the current bit to 1, else clear the current bit to 0



A/D - Successive Approximation

Converter Schematic



ADC Performance Metrics

- Linearity measures how well the transition voltages lie on a straight line.
- Differential linearity measure the equality of the step size.
- Conversion time: between start of conversion and generation of result
- Conversion *rate* = inverse of conversion *time*

Sampling Problems

- Nyquist criterion
  - $F_{sample} >= 2 * F_{max\ frequency\ component}$
  - Frequency components above ½ F_{sample} are aliased, distort measured signal

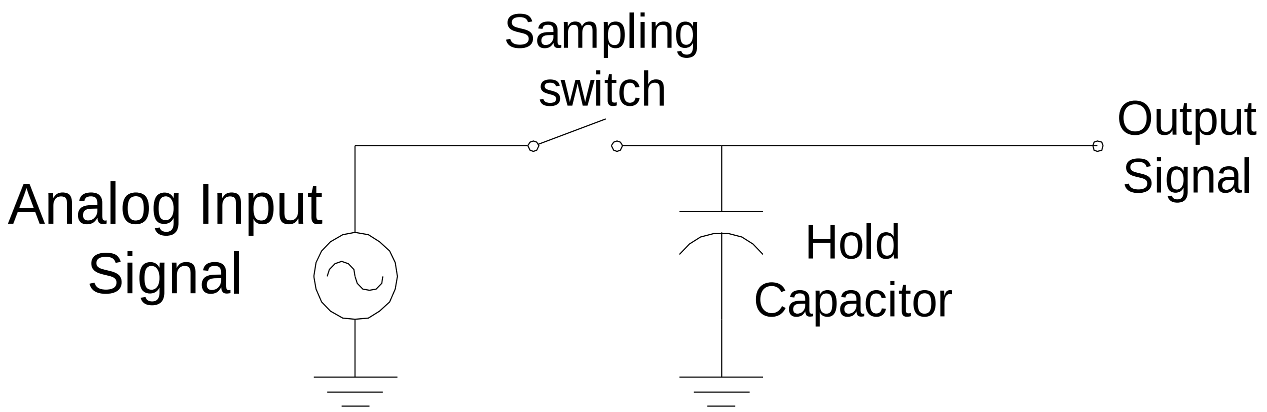
- Nyquist and the real world
  - This theorem assumes we have a perfect filter with “brick wall” roll-off
  - Real world filters have more gentle roll-off
  - Inexpensive filters are even worse
  - So we have to choose a sampling frequency high enough that our filter attenuates aliasing components adequately

Inputs

- Differential
  - Use two channels, and compute difference between them
  - Very good noise immunity
  - Some sensors offer differential outputs (e.g. Wheatstone Bridge)
- Multiplexing
  - Typically share a single ADC among multiple inputs
  - Need to select an input, allow time to settle before sampling
- Signal Conditioning
  - Amplify and filter input signal
  - Protect against out-of-range inputs with clamping diodes

ARM

Sample and Hold Devices



- Some A/D converters require the input analog signal to be held constant during conversion (e.g. successive approximation devices)
- In other cases, peak capture or sampling at a specific point in time requires a sampling device.
- A “sample and hold” circuit performs this operation
- Many A/D converters include a sample and hold circuit

KL25 ANALOG INTERFACING PERIPHERALS

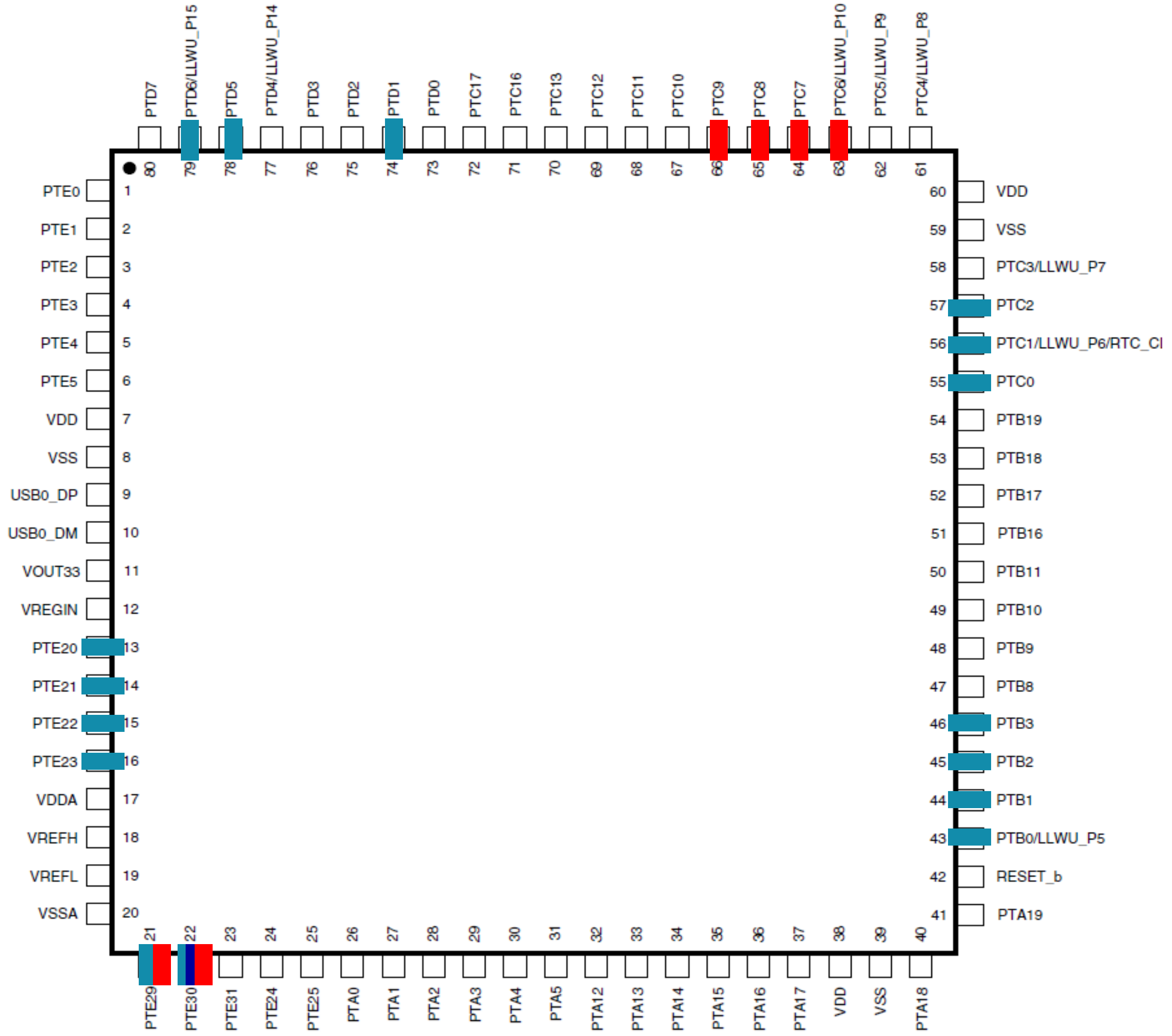
Sources of Information

- KL25 Subfamily Reference Manual (Rev. 1, June 2012)
  - Describes architecture of peripherals and their control registers
  - Digital to Analog Converter
    - Chapter 30 of KL25 Subfamily Reference Manual
  - Analog Comparator
    - Chapter 29 of KL25 Subfamily Reference Manual
  - Analog to Digital Converter
    - Chapter 28 of KL25 Subfamily Reference Manual

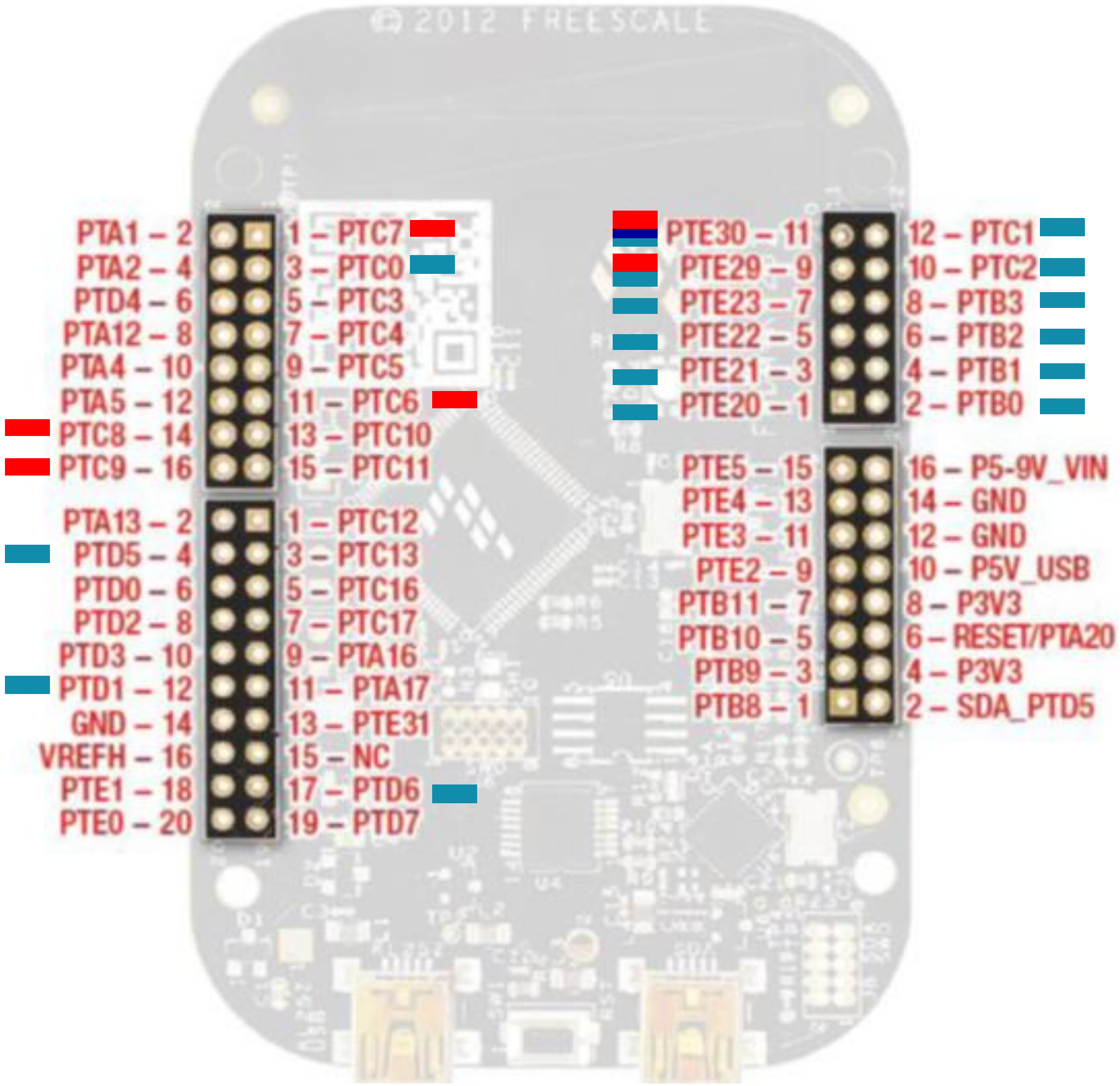
- KL25 Sub-family Data Sheet (Rev. 3, 9/19/2012)
  - Describes circuit-specific performance parameters: operating voltages, min/max speeds, cycle times, delays, power and energy use

KL25Z Analog Interface Pins

- 80-pin QFP
- Inputs
  - 1 16-bit ADC with 14 input channels
  - 1 comparator with 6 external inputs, one 6-bit DAC
- Output
  - 1 12-bit DAC



Freedom KL25Z Analog I/O



Inputs

- 14 external ADC channels
- 6 external comparator channels

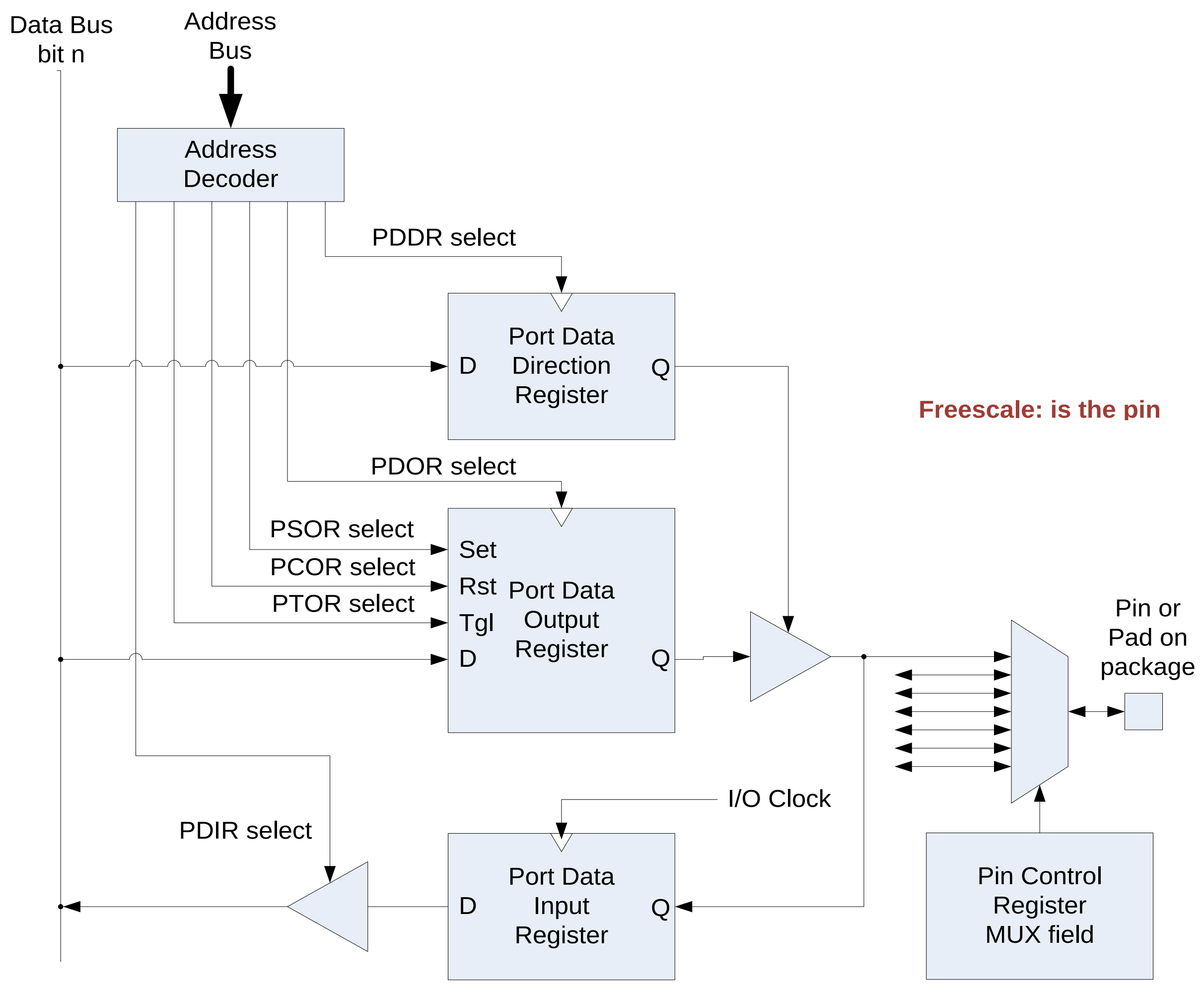
Output

- 1 12-bit DAC

Using a Pin for Analog Input or Output

- Configuration
  - Direction
  - MUX

- Data
  - Output (different ways to access it)
  - Input



Freescale: is the pin

Pin Control Register to Select MUX Channel

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
64	52	40	28	PTC7	CMP0_IN1	CMP0_IN1	PTC7	SPI0_MISO			SPI0_MOSI		
65	53	—	—	PTC8	CMP0_IN2	CMP0_IN2	PTC8	I2C0_SCL	TPM0_CH4				

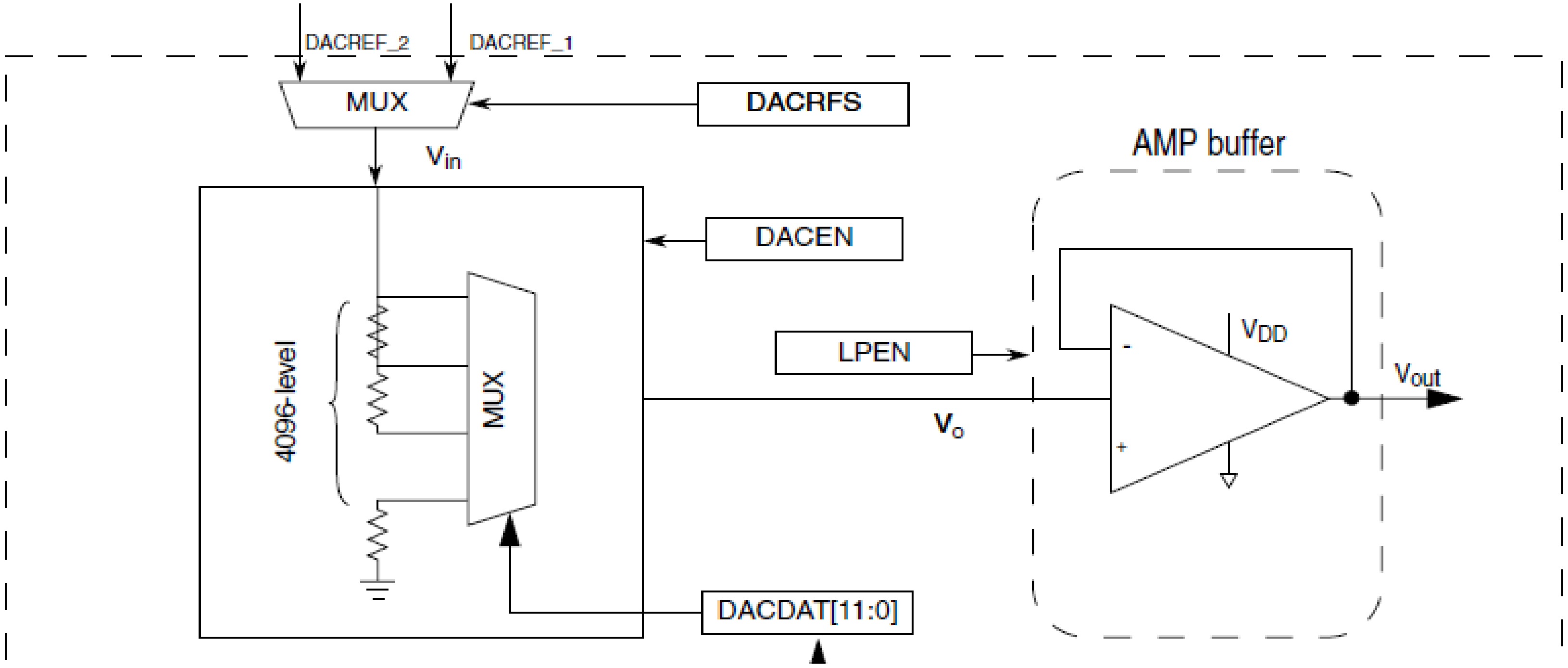
- MUX field of PCR defines connections

MUX (bits 10-8)	Configuration
000	Digital circuits disabled, analog enabled
001	Alternative 1 – GPIO
010	Alternative 2
011	Alternative 3
100	Alternative 4
101	Alternative 5
110	Alternative 6
111	Alternative 7

```
PORTC->PCR[7] &= ~PORT_PCR_MUX_MASK;
PORTC->PCR[7] |= PORT_PCR_MUX(0);
```

DIGITAL TO ANALOG CONVERTER

DAC Overview



- Load DACDAT with 12-bit data N
- MUX selects a node from resistor divider network to create $V_o = (N+1)*V_{in}/2^{12}$
- V_o is buffered by output amplifier to create V_{out}
  - $V_o = V_{out}$ but V_o is high impedance - can't drive much of a load, so need to buffer it

DAC Operating Modes

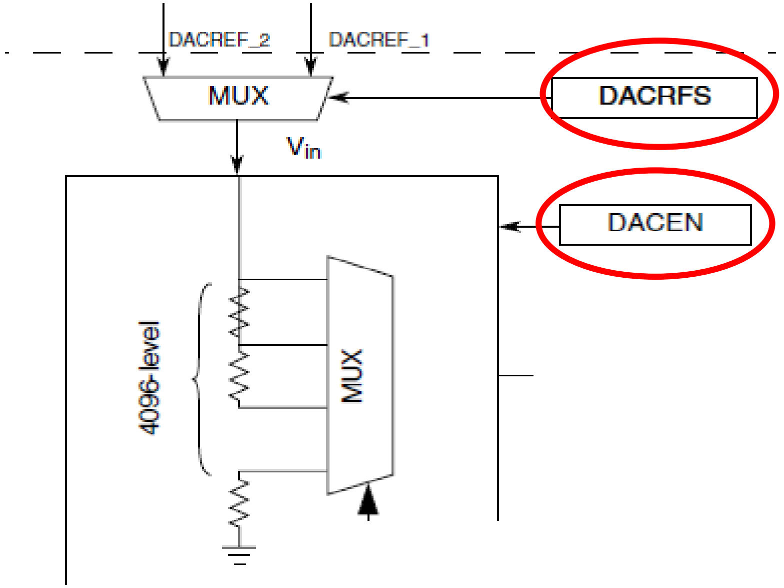
- Normal
  - DAT0 is converted to voltage immediately

- Buffered
  - Data to output is stored in 16-word buffer
  - Next data item is sent to DAC when a selectable trigger event occurs
    - Software Trigger - write to DACSWTRG field in DACx_C0
    - Hardware Trigger - from PIT timer peripheral
  - Normal Mode
    - Circular buffer
  - One-time Scan Mode
    - Pointer advances until reaching upper limit of buffer, then stops
  - Status flags in DACx_SR

DAC Control Register 0: DACx_C0

Bit	7	6	5	4	3	2	1	0
Read	DACEN	DACRFS	DACTRGSEL	0	LPEN	0	DACBTIEN	DACBBIEN
Write	DACEN	DACRFS	DACTRGSEL	DACSWTRG	LPEN		DACBTIEN	DACBBIEN
Reset	0	0	0	0	0	0	0	0

- DACEN - DAC Enabled when 1
- DACRFS - DAC reference voltage select
  - 0: DACREF_1. Connected to VREFH
  - 1: DACREF_2. Connected to VDDA
- LPEN - low-power mode
  - 0: High-speed mode. Fast (15 us settling time) but uses more power (up to 900 uA supply current)
  - 1: Low-power mode. Slow (100 us settling time) but more power-efficient (up to 250 uA supply current)
- Additional control registers used for buffered mode



DAC Control Register 1: DACx_C1

Bit	7	6	5	4	3	2	1	0
Read	DMAEN	0				DACBFMD	0	DACBFEN
Write								
Reset	0	0	0	0	0	0	0	0

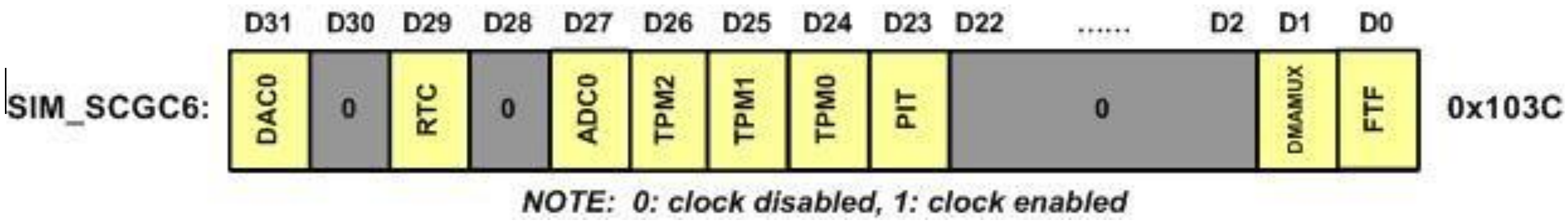
- DACBFEN
  - 0: Disable buffer mode
  - 1: Enable buffer mode
- DACBFMD - Buffer mode select
  - 0: Normal mode (circular buffer)
  - 1: One-time scan mode

DAC Data Registers

- These registers are only eight bits long
- DATA[11:0] stored in two registers
  - DATA0: Low byte [7:0] in DACx_DATnL
  - DATA1: High nibble [11:8] in DACx_DATnH

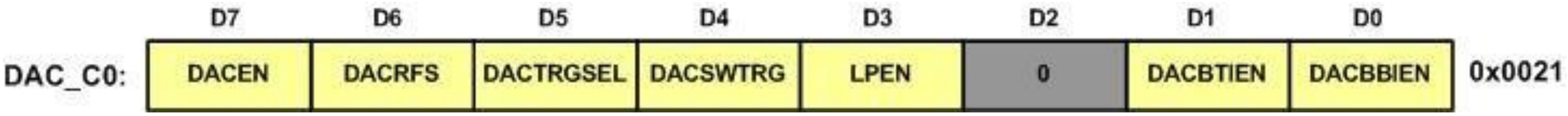
Example: Waveform Generator

- Supply clock to DAC0 modul
  - Bit 31 of SIM SCGC6



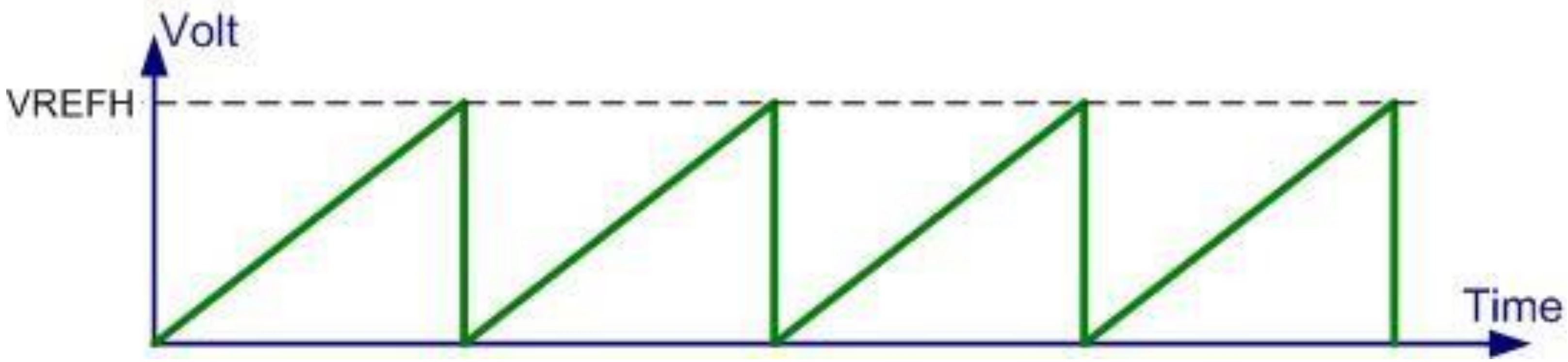
- Set Pin Mux to Analog (0)

- Enable DAC
- Configure DAC
  - Reference voltage
  - Low power mode?
  - Normal mode (not buffered)
- Write to DAC data register



Bit	Field	Descriptions
7	DACEN	DAC Enable (0: DAC is disabled, 1: DAC is enabled)
6	DACRFS	DAC Reference Select (0: DACREF_1, 1: DACREF_2)
5	DACTRGSEL	DAC Trigger Select (0: hardware trigger, 1: software trigger)
4	DACSWTRG	DAC Software Trigger
3	LPEN	DAC Low Power Control (0: High-Power mode, 1: Low-Power mode)
1	DACBTIEN	DAC Buffer read pointer Top flag Interrupt Enable
0	DACBBIEN	DAC Buffer read pointer Bottom flag Interrupt Enable

Program 7.4 from Mazidi etal - Saw Tooth Generation




```
void DAC0_init(void);
void delayMs(int n);
int main (void) {
int i;
DAC0_init(); /* Configure DAC0 */
while (1) {
for (i = 0; i < 0x1000; i += 0x0010) {
/* write value of i to DAC0 */
DAC0->DAT[0].DATL = i & 0xff; /* write low byte */
DAC0->DAT[0].DATH = (i >> 8) & 0x0f;/* write high byte */
delayMs(1); /* delay 1ms */
}
}
}
```

Code

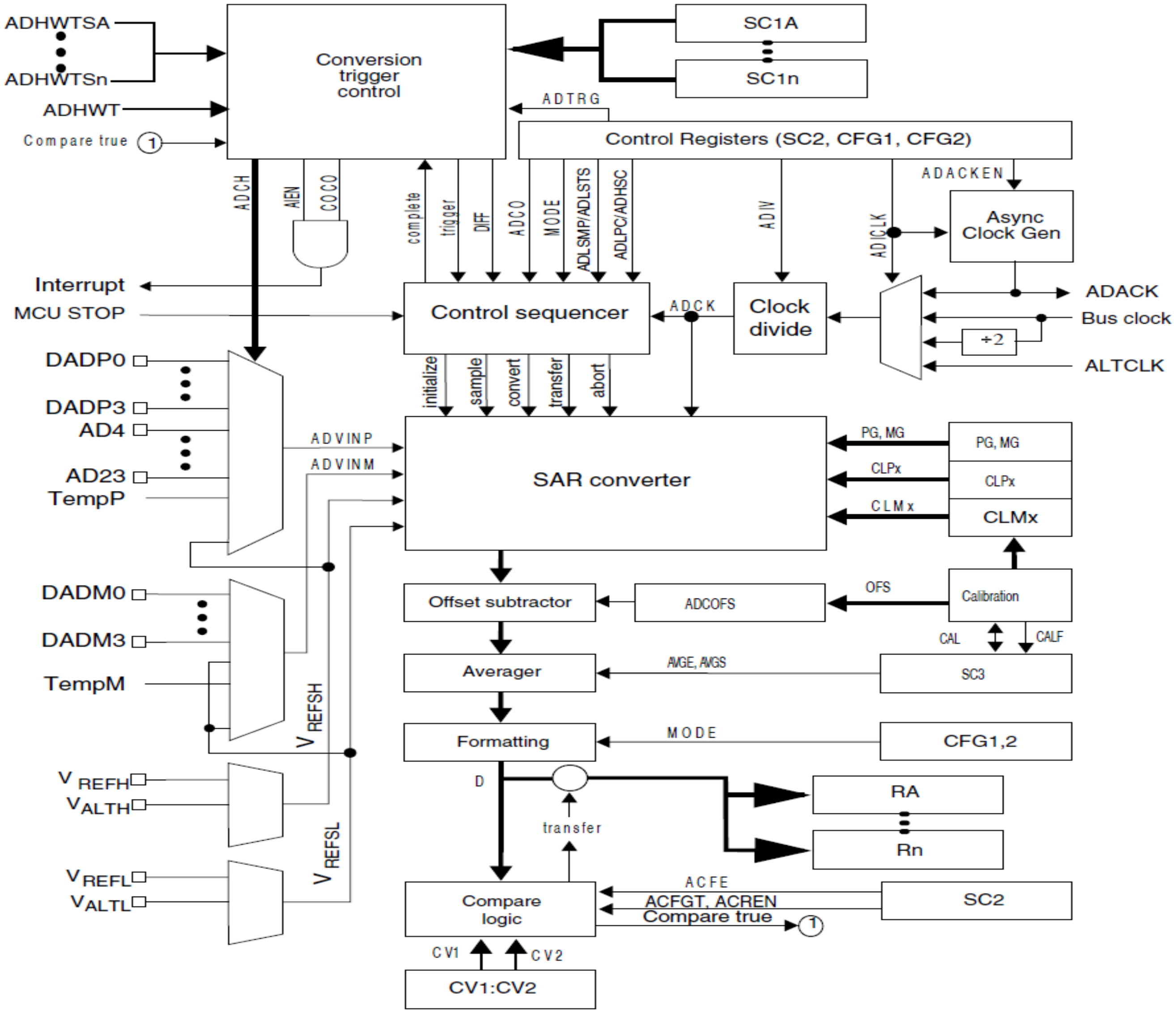
```
void DAC0_init(void)
{
PORTE->PCR[30] &= ~(0x700); /* alt function 0 */
SIM->SCGC6 |= 0x80000000; /* clock to DAC module */
DAC0->C1 = 0; /* disable the use of buffer */
DAC0->C0 = 0x80 | 0x20; /* enable DAC and use software trigger */
}
/* Delay n milliseconds
The CPU core clock is set to MCGFLLCLK at 41.94 MHz in SystemInit(). */
void delayMs(int n) {
int i;
int j;
for(i = 0 ; i < n; i++)
for (j = 0; j < 7000; j++) {}
}
```

ANALOG TO DIGITAL CONVERTER

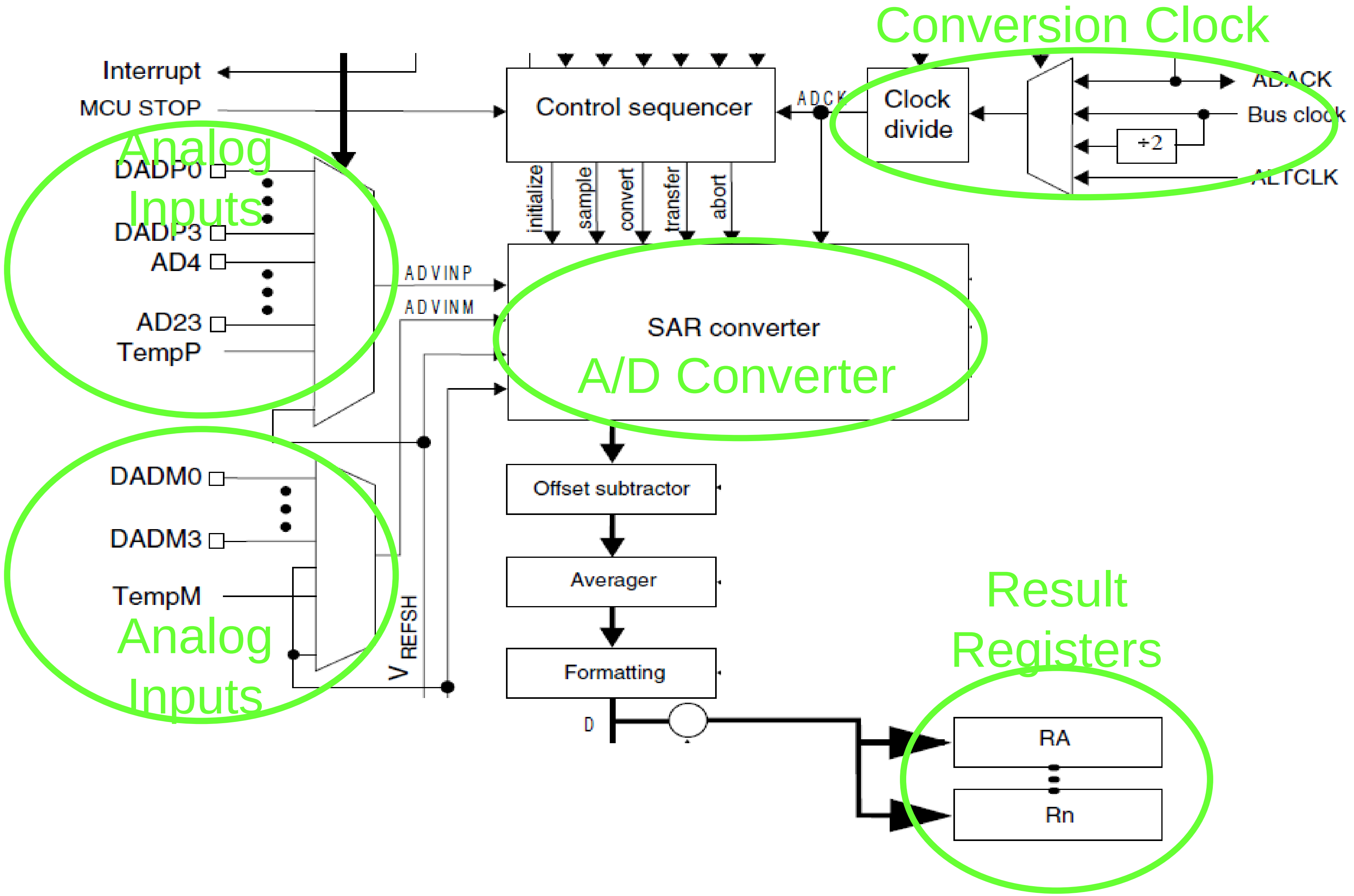
ADC Overview

- Uses successive approximation for conversion
- Supports multiple resolutions: 16, 13, 12, 11, 10, 9, and 8 bits
- Supports single-ended and differential conversions
- Signed or unsigned results available
- Up to 24 analog inputs supported (single-ended), 4 pairs of differential inputs
- Automatic compare and interrupt for level and range comparisons
- Hardware data averaging
- Temperature sensor

ADC System Overview



ADC System Fundamentals



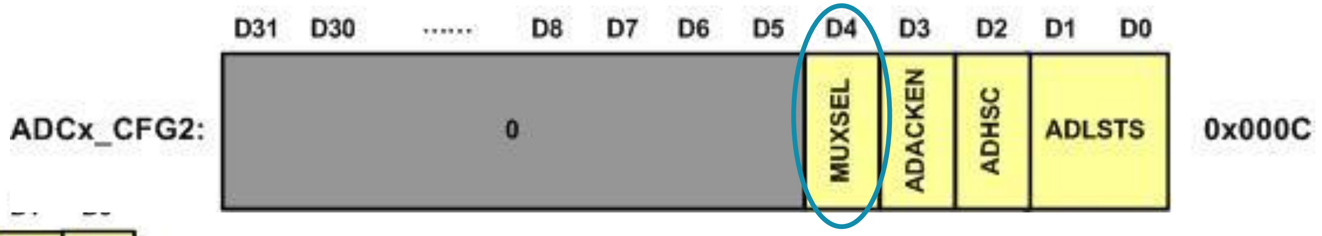
Using the ADC

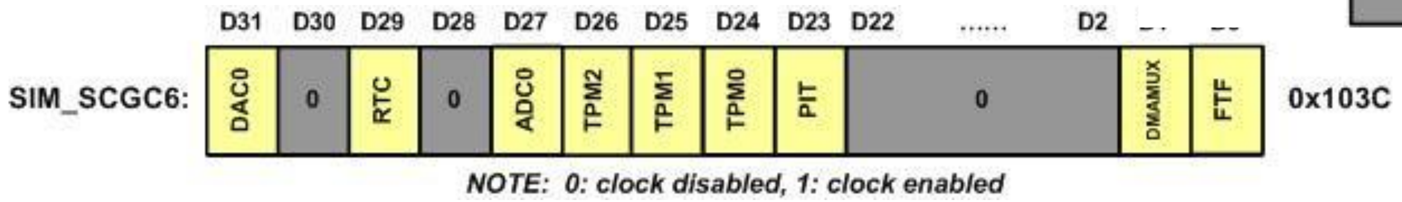
- ADC initialization
  - Configure clock
  - Select voltage reference
  - Select trigger source
  - Select input channel
  - Select other parameters

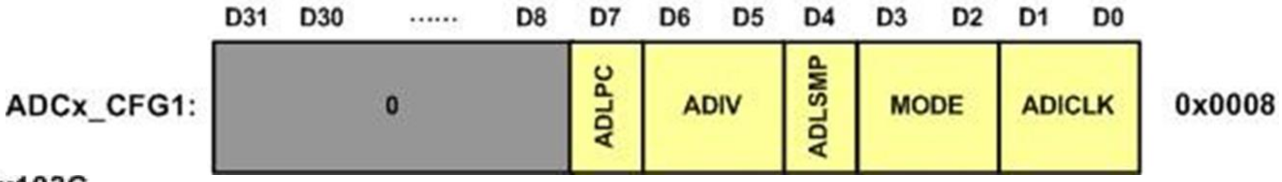
- Trigger conversion

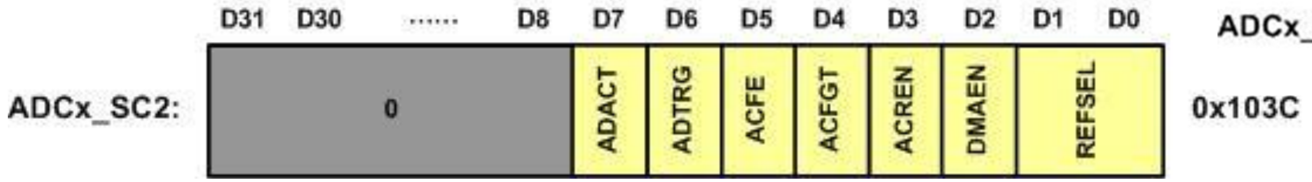
- Read results

ADC Registers









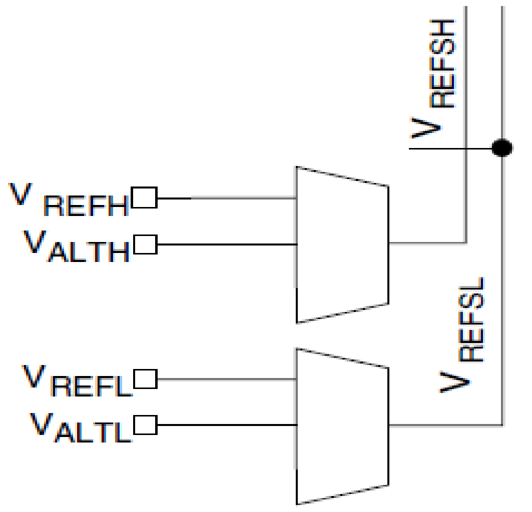
Bit	Field	Descriptions
7	ADACT	Conversion active: Indicates that the ADC is converting data (0: Conversion not in progress, 1: Conversion in progress)
6	ADTRG	ADC conversion trigger select (0: software trigger, 1: hardware trigger)
5	ACFE	Compare Function Enable (0: compare function disabled, 1: enabled)
4	ACFGT	Compare Function Greater Than Enable
3	ACREN	Compare range Enable
2	DMAEN	DMA Enable
1-0	REFSEL	Voltage Reference Select

Bit	Field	Descriptions
7	ADLPC	Low-Power Configuration
6-5	ADIV	Clock Divide Select: The clock is divided by 2^{ADIV} as shown in Figure 7-7.
4	ADLSMP	Sample time configuration (0: Short sample time, 1: Long sample time)
3-2	MODE	Conversion mode selection
1-0	ADICLK	Input Clock Select

ARM

Voltage Reference Selection

- Two voltage reference pairs available
  - V_{REFH}, V_{REFL}
  - V_{ALTH}, V_{ALTL}
- Select with SC2 register's REFSEL bits
  - 00: V_{REFH}, V_{REFL}
  - 01: V_{ALTH}, V_{ALTL}
  - 10, 11: Reserved
- KL25Z
  - V_{ALTH} connected to V_{DDA}



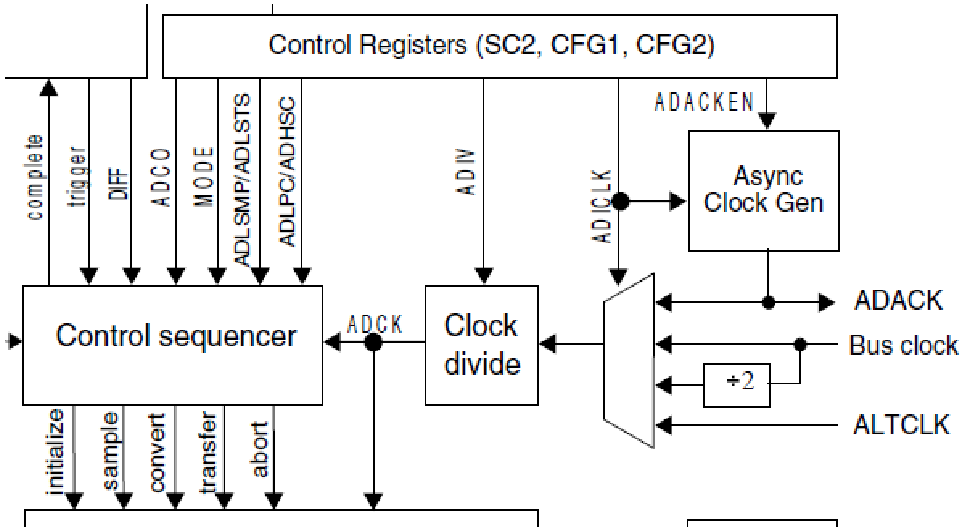
Conversion Options Selection

- Low power
  - Set ADLPC (in ADCx_CFG1) to 1
  - Slower max clock speed
- Long sample time select
  - Set ADLSMP (in ADCx_CFG1) to 1
  - Can select longer sample time with ADLSTS bits (in ADCx_CFG2) to add 20, 16, 10 or 6 ADCK cycles)
- Conversion mode
  - MODE (in ADCx_CFG1)
  - Sets result precision (8 through 16 bits)
- Continuous vs. single conversion
  - Set ADCO (in ADCx_SC3) to 1 for continuous conversions

7	6	5	4	3	2	1	0
ADLPC	ADIV		ADLSMP	MODE		ADICLK	
0	0	0	0	0	0	0	0

	DIFF	
MODE	0	1
0	Single ended 8-bit	Differential 9-bit 2's complement
1	Single ended 12-bit	Differential 13-bit 2's complement
2	Single ended 10-bit	Differential 11-bit 2's complement
3	Single ended 16-bit	Differential 16-bit 2's complement

Clock Configuration



- Select clock source with ADICLK
  - Bus Clock (default)
  - ADACK: Local clock, allows ADC operation while rest of CPU is in stop mode
  - ALTCLK: alternate clock (MCU-specific)
- Divide down selected clock by factor of ADIV, creating ADCK
- Resulting ADCK must be within valid range to ensure accuracy (See KL25 Subfamily datasheet)
  - 1 to 18 MHz (<= 13-bit mode)
  - 2 to 12 MHz (16-bit mode)

Clock Configuration Registers

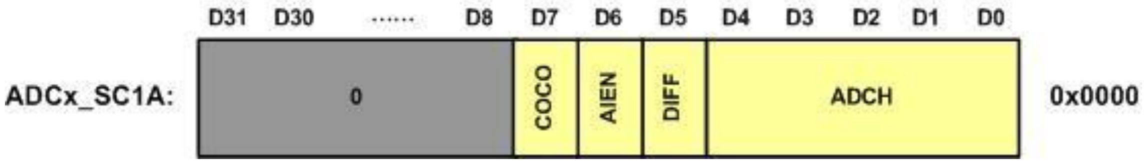
- ADCx_CFG1
  - ADIV: divide clock by 2^{ADIV}
    - 00: 1
    - 01: 2
    - 10: 4
    - 11: 8
  - ADICLK: Input clock select
    - 00: Bus clock
    - 01: Bus clock/2
    - 10: ALTCLK
    - 11: ADACK

7	6	5	4	3	2	1	0
ADLPC	ADIV		ADLSMP	MODE		ADICLK	
0	0	0	0	0	0	0	0

- ADCx_CFG2
  - ADACKEN: Enable asynchronous clock

4	3	2	1	0
MUXSEL	ADACKEN	ADHSC	ADLSTS	
0	0	0	0	0

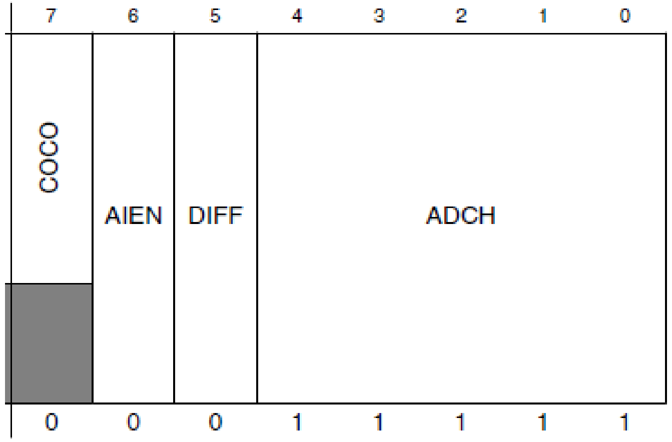
Registers

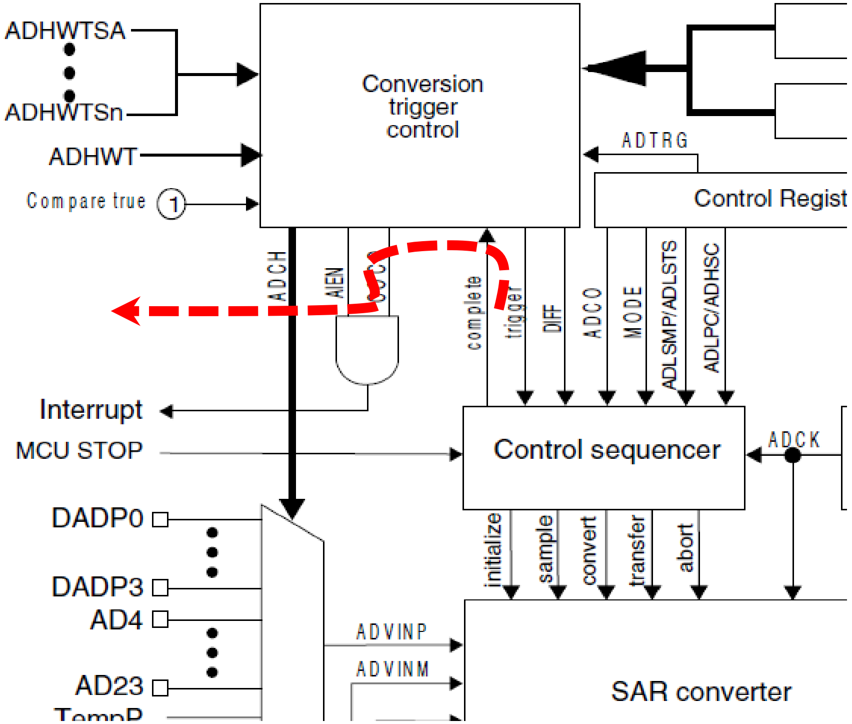


Bit	Field	Descriptions
7	COCO	Conversion Complete Flag: (0: Conversion is not completed, 1: Conversion is completed) The COCO is cleared when the ADCx_SC1n register is written or the ADCx_Rn register is read.
6	AIEN	Interrupt Enable: The ADC interrupt is enabled by setting the bit to HIGH. If the interrupt enable is set, an interrupt is triggered when the COCO flag is set.
5	DIFF	Differential mode (0: Single-ended mode, 1: Differential mode)
4-0	ADCH	ADC input channel: The field selects the input channel as shown in Figure 7-7. When DIFF = 0 (single-ended mode), values 0 to 23 choose between the 24 input channels (ADC_SE0 to ADC_SE23). When DIFF = 1 (Differential mode), values 0 to 3 select between the 4 differential channels. See the reference manual for more information. When ADCH = 11111, the module is disabled.

Pin Name	Description	Pin
ADC_SE0	ADC input 0	PTE20
ADC_SE3	ADC input 3	PTE22
ADC_SE4	ADC input 4	PTE21, PTE29
ADC_SE5	ADC input 5	PTD1
ADC_SE6	ADC input 6	PTD5
ADC_SE7	ADC input 7	PTD6, PTE23
ADC_SE8	ADC input 8	PTB0
ADC_SE9	ADC input 9	PTB1
ADC_SE11	ADC input 11	PTC2
ADC_SE12	ADC input 12	PTB2
ADC_SE13	ADC input 13	PTB3
ADC_SE14	ADC input 14	PTC0
ADC_SE15	ADC input 15	PTC1
ADC_SE23	ADC input 23, DAC0 output	PTE30
ADC_SE26	Temperature sensor	
ADC_SE27	Bandgap reference	
ADC_SE29	V_{REFH}	
ADC_SE30	V_{REFL}	
ADC_SE31	Module disabled	

Conversion Completion

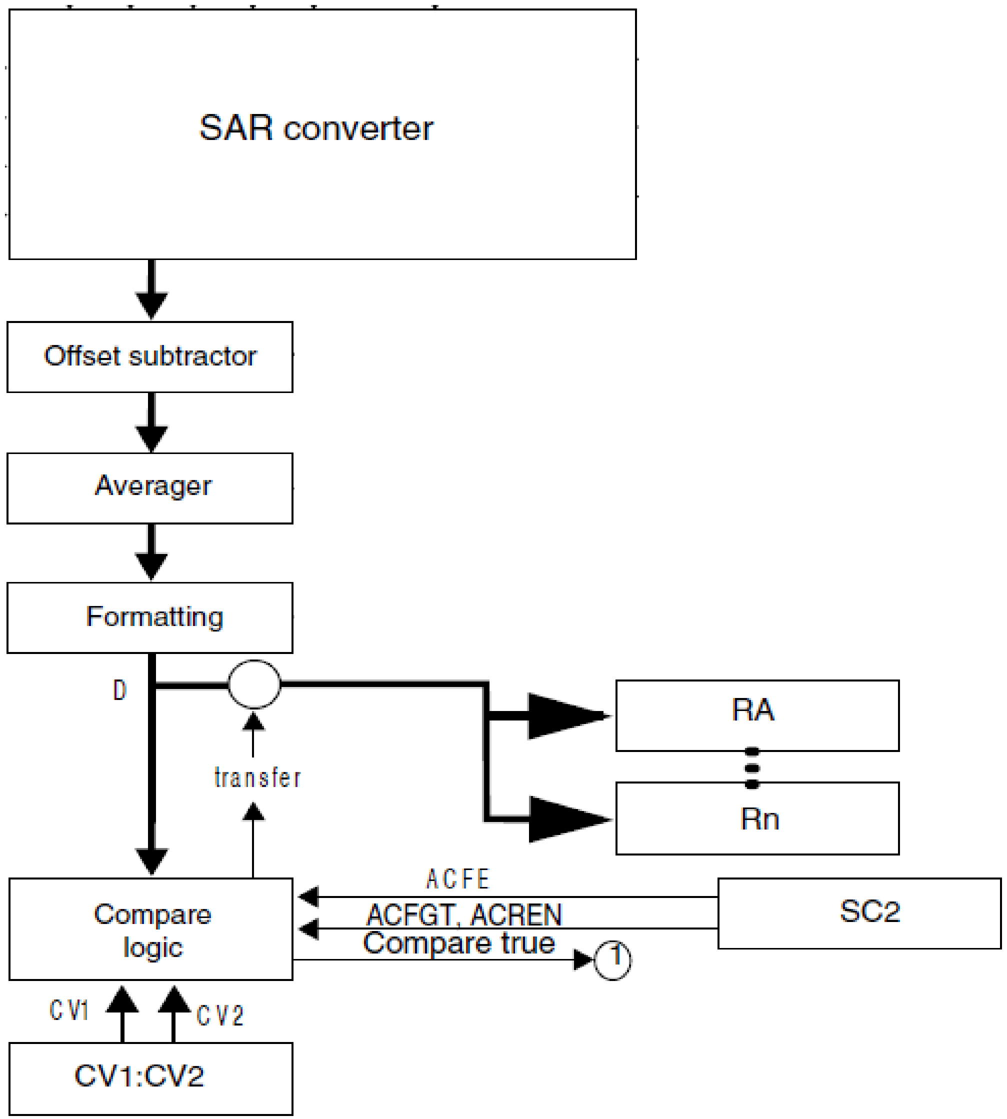




- Signaled by COCO bit in SC1n
- Can generate conversion complete interrupt if AIEN in SC1 is set
  - CMSIS-defined ISR name for ADC Interrupt is ADC0_IRQHandler

Result Registers

- Optional output processing before storage in result registers
  - Offset subtraction from calibration
  - Averaging: 1, 4, 8, 16 or 32 samples
  - Formatting: Right justification, sign- or zero-extension to 16 bits
  - Output comparison
- Two result registers RA and Rn
  - Conversion result goes into register corresponding to SC1 register used to start conversion (SC1A, SC1n)



Procedure – Polling Method

- Enable the clock to I/O pin used by the ADC channel. Table shows the I/O pins used by various ADC channels.
- Set the PORTX_PCRn MUX bit for ADC input pin to 0 to use the pin for analog input channel. This is actually the power-on default.
- Enable the clock to ADC0 modules using SIM_SCGC6 register.
- Choose the software trigger using the ADC0_SC2 register.
- Choose clock rate and resolution using ADC0_CFG1 register.
- Select the ADC input channel using the ADC0_SC1A register. Make sure interrupt is not enabled and single-ended option is used when you select the channel with this register.
- Keep monitoring the end-of-conversion COCO flag in ADC0_SC1A register.
- When the COCO flag goes HIGH, read the ADC result from the ADC0_RA and save it.
- Repeat steps 6 through 8 for the next conversion.

Example – Listing 6.5

```
#define ADC_POS (20)
void Init_ADC(void) {

        SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

        // Select analog for pin
        PORTE->PCR[ADC_POS] &= ~PORT_PCR_MUX_MASK;
        PORTE->PCR[ADC_POS] |= PORT_PCR_MUX(0);

        // Low power configuration, long sample time, 16 bit
single-ended conversion, bus clock input
        ADC0->CFG1 = ADC_CFG1_ADLPC_MASK | ADC_CFG1_ADLSMP_MASK
| ADC_CFG1_MODE(3) | ADC_CFG1_ADICLK(1);
        // Software trigger, compare function disabled, DMA
disabled, voltage references VREFH and VREFL
        ADC0->SC2 = ADC_SC2_REFSEL(0);
}
```

Listing 6.6

```
float Measure_Temperature(void){
        float n, temp;

        ADC0->SC1[0] = 0x00; // start conversion on channel 0

        // Wait for conversion to finish
        while (!(ADC0->SC1[0] & ADC_SC1_COCO_MASK))
                ;
        // Read result, convert to floating-point
        n = (float) ADC0->R[0];

        // Calculate temperature (Celsius) using polynomial equation
        // Assumes ADC is in 16-bit mode, has VRef = 3.3 V

        temp = -36.9861 + n*(0.0155762 + n*(-1.43216E-06 + n*(7.18641E-11
                + n*(-1.84630E-15 + n*(2.32656E-20 + n*(-1.13090E-25))))));
        return temp;
}
```