ADC → Analog to Digital Convert

↳ ① volt

3.3V ⟶ 10111100

↳ ② Common use sensor

ADC

↳ No of Bits (Resolution)

↳ Reference → volt / Max volt

↳ circuit → to convert

Analog → Circuit → Digital

↳ Resolution → 2

↳ volt Reference → 4 volt

↳ ditial volt = 11 →

ADC => Max Volt – step

$$step = \frac{Max\ volt}{2^{Res}} = \frac{V_{Ref}}{2^{Res}} = \frac{4}{2^2} = 1V$$

00 ⟶ 0V

01 ⟶ step x 0 + step x 1 = 1V

10 ⟶ step x 2 + step x 0 = 2V

11 ⟶ step x 2 + step x 1 = 3V

DAC → Digital to Analog.

Digital → ▭ → Analog

↳ No of Bits

↳ V Ref

↳ step

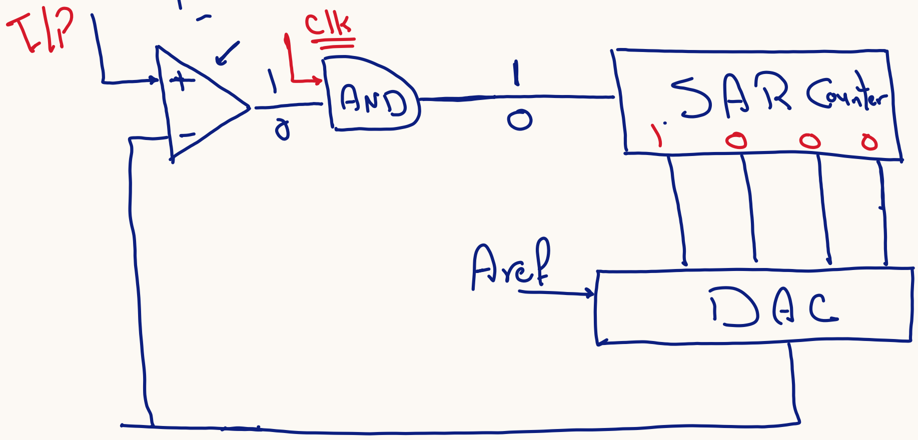
Analog volt = Digital x step

$$= 0110\ 0110 * \frac{V_{ref}}{2^{Res}}$$

* Atmega32 → not support DAC

ADC . Circuit

↳ [successive Approximation register] [SAR ADC]

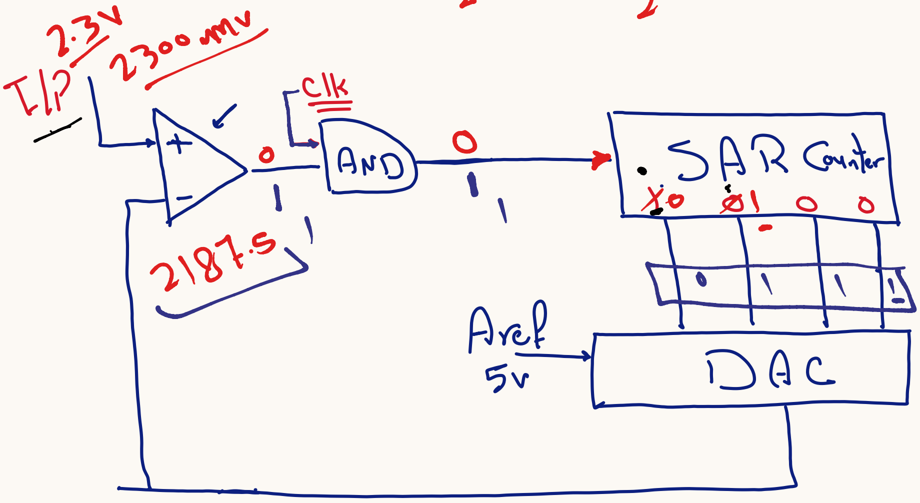


→ Conversion

1. Clock for And gate.
2. default value
   ↳ 1 at MSB
   ↳ and the other bit 0
3. Dac Convert to Analog
4. Compare

[1] Aref = 5v   [2] Resolution = 4

[3] step = $\frac{5v}{2^4} = \frac{5000}{2^4} = 312.5 mv$



[1] SAR Counter = 1000

[2] DAC →
Analog = Digital × step
= 8 × 312.5 mv
= 2500 mv

[3] OP Amp
I/P < Feed Back
2300 mv < 2500 mv
x → [0]

[4] And gate
clock → 1
OP Amp → 0 → [0]

[5] SAR → Input for SAR Counter = [0]

when Input SAR = 0
↳ last Bit will be Zero
↳ Next Bit will be one

[0100] → New value for SAR

Created by Notein

[6] DAC → $\underbrace{0100}_{4}$ * 312.5 ⇒ 1250 mv

[7] OPAMP → I/P > Feed Back
2300 > 1250 → [1]

[8] And → OPAmP → 1 } → [1]
clk → 1

[9] SAR → Input [1]
if Input = 1
↳ the last Bit still the same value
↳ the Next Bit will be one

[0110] →

[10] DAC → 6 x 312.5 ⇒ 1875 mv

[11] OPAMP → 2300 > 1875 → [1]

[12] AND → OP → 1 } → 1
clk → 1

[13] SAR → [1]
$\underbrace{0111}$ → 7 →

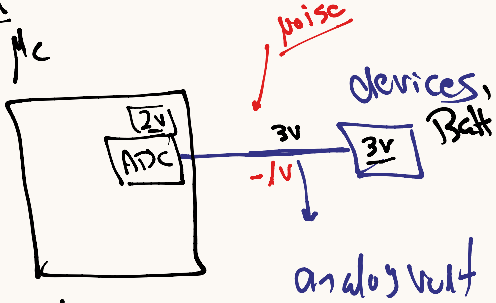
[13] DAC → 7 X 312.6 → $\underbrace{2187.5 \text{ mv}}$ → [0111]

ADC in Atmega 32

* Conversion time → depend on ADC clock
* 15 kilo sample Per Second → 15 kilo Conversion in one Second.

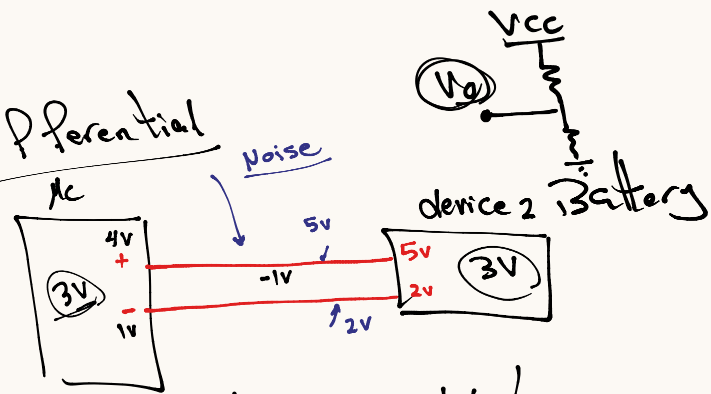
→ Single Ended & Differential

Single Ended



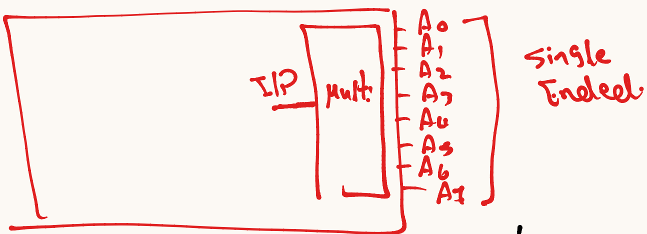
sent the analog voltage b/ one wire

Differential

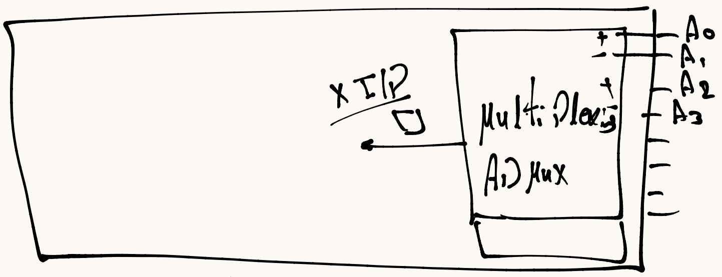


sent the 2 value b/ two wire → and μc will differenital this two

↳ 8 Multiplexed Single Ended Input Channels.



↳ 7 Differential Input channel



↳ 2 Differential I/P Channel with gain → 10 / 200.

$$\frac{1000}{2^{10}} = \frac{1000}{1024} = 0.976\ mV$$

$$\frac{10}{200} = 0.5\ mV$$

ADC Register

① ADMUX → | 7 | 6 |

* Bit 7 : 6 :: Reference selection
  ① Connect on Aref Pin External volt (0,0)
  ② Connect Aref with AVcc (0,1)
  ③ Reserved (1,0) Not used.
  ④ Internal voltage 2.56 V →

Aref

Bit 5 → left Adjust Result. → 0 → Right
↳ 1 → left

SAR

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ADC Data

left Adjust

| 9 | 8 | 7 | 6 | 5 | 4 | 3 | D2 |
| D1 | D0 |

Righ Adjust

| D9 | D8 |
| D7 ← D0 |

Bit 4 : Bit 0 → 5 Bit to select
↳ single End / Differentian
↳ which channel if you use single Ended
↳ Differential → No gain, 10, 200

00000] → single Ended
00111

00000 → channel 0
00001 → " 1
00010 → " 2
00011 → " 3

Differential → 1 gain
10000
11101

② ADCSRA → | ADEN | ADSC | ADATE |

* Bit 7 → ADC_Enable
↳ 1 → Enable
↳ 0 → turnoff
Not have Power

Bit 6 : ADC_start Conversion
↳ have start Conversion operation from Analog to digital.

Single Conversion Mode
→ [1] → b/s w → start Conv
→ [0] → b/ H.w → Conv. Finished

Free Running Mode (Auto trigger)
↳ start Conversion
↳ must be one at First time

Notes: ① Must Enable ADC First
② must set this bit each time

ADCSRA

* Bit 5 → (Auto trigger) Enable / Disable
0 → Single Mode
1 → Auto trigger
* Free Running
* EXTI
* timer

Bit 4 : ADC Interrupt Flag

* this flag will be set when Conversion Finish & Data store in Register

Polling ←
while (Flag == 0);
to clear this Bit
setBit (ADCSRA, ADIF);

Interrupt
ADIE → 1
GIE → 1
Clear Automatic

Bit 3 → ADC Interrupt Enable
↳ SIE
↳ 0 → not allow ADC Interrupt Proc
↳ 1 → allow ADC Interrupt Pro
but must GIE = 1

ADC Interrupt No → 16

void __vector_16( ) __attribute__(( ));

Bit 2:0, MF clock → Prescaler to select ADC_clock

→ system clock / Division factor
8000 000 / 2 = 4 MHz
" / 4 = MHz

③ Data Register

ADC H] → store the Digital volt.
ADC L

#define ADCH *((volatile uint8_t*) 0x25)
#define ADCL *((volatile uint8_t*) 0x24)
#define ADC_D *((volatile uint16*) 0x24)

ADC_D → [0x24, 0x25]

uint16
→ 0x24
0x25