8.6– Analog to Digital Converter Peripheral (ADC) & SSD1306 OLED Display

Prof. David McLaughlin

ECE Dept

UMass Amherst

Spring 2024

Lab Assignment #2 Design, build, test, demonstrate a digital thermometer. Labs will meet next to work in this assignment. See github for the assignment statement.

What's next?
Analog to Digital Converter (ADC)

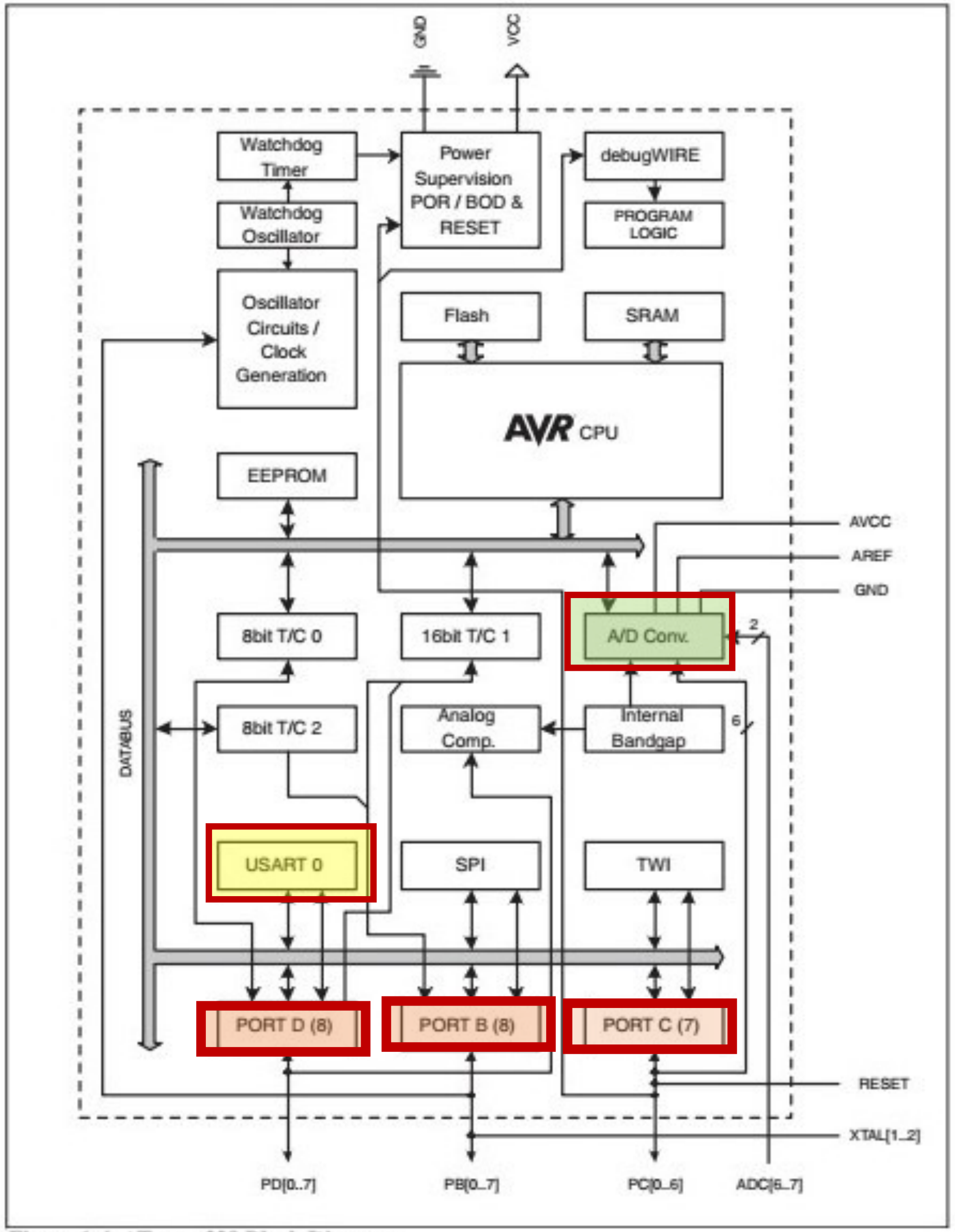


Figure 1-4. ATmega328 Block Diagram

ADC

```
int main(void) {
    unsigned int digitalValue;
    DDRC = 0x00;                                  //set PD0 = ADC0 as input
    ADMUX = 0xC0;                                 //select ADC0; Vref=1.1V
    ADCSRA = 0x87;                                //enable ADC; speed 125 KHz
    while (1) {
        …
        ADCSRA |= (1 << ADSC);                    //start ADC conversion
        while ((ADCSRA & (1 << ADIF)) == 0);      //wait till finished converting
        digitalValue = ADCL | (ADCH << 8);        //read the ADC digital value
        …
    }
}
```

What's next?
Analog to Digital Converter (ADC)

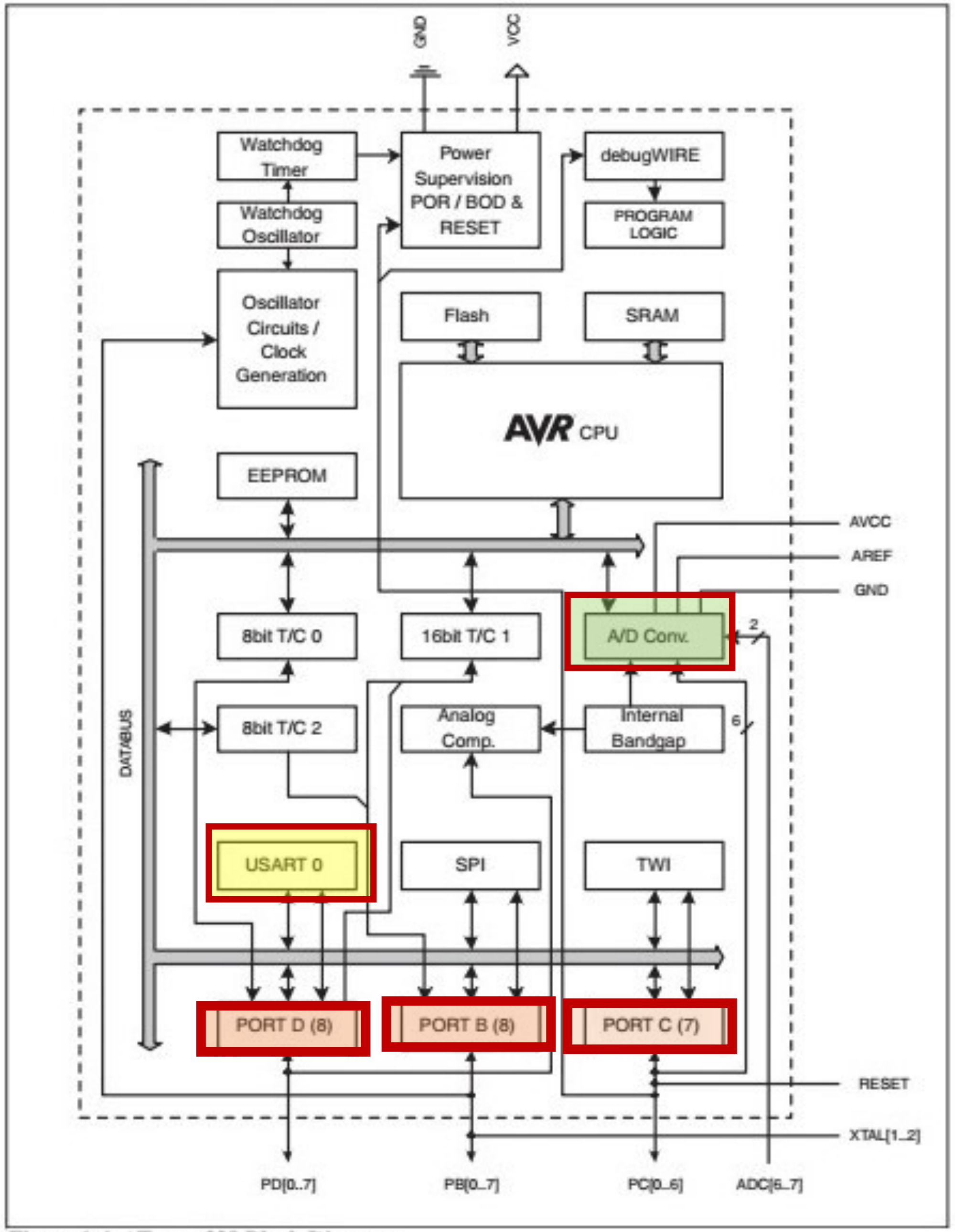


Figure 1-4. ATmega328 Block Diagram

ADC

```
#include ”my_adc_lib.h”

int main(void) {
	unsigned int digitalValue;
	adc_init()
	while (1) {
		…
		digitalValue = get_adc();
		…
	}
}
```

Temperature Sensor - TMP36

SEN-10988 ROHS

17

$1.50

Volume sales pricing

- 1 +

ADD TO CART

Quantity discounts available

DESCRIPTION FEATURES DOCUMENTS

This is the same temperature sensor that is included in our SparkFun Inventor's Kit. The TMP36 is a low voltage, precision centigrade temperature sensor. It provides a voltage output that is linearly proportional to the Celsius temperature. It also doesn't require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the –40°C to +125°C temperature range. We like it because it's so easy to use: Just give the device a ground and 2.7 to 5.5 VDC and read the voltage on the Vout pin. The output voltage can be converted to temperature easily using the scale factor of 10 mV/°C.

Tags

SENSOR TEMPERATURE TMP36

The TMP36 is specified from −40°C to +125°C, provides a 750 mV output at 25°C, and operates to 125°C from a single 2.7-5.5 V supply. The TMP36 has an output scale factor of 10 mV/°C.

V_{out} = 750 mV @ T_c = 25^0 C
scale factor 10 mV/ ^{0} C

$V_{out} = 750 + 10(T_c\text{-}25)$ mV

Check the pinout for yourself. Don't rely on this figure…

V_s V_{out} Gnd

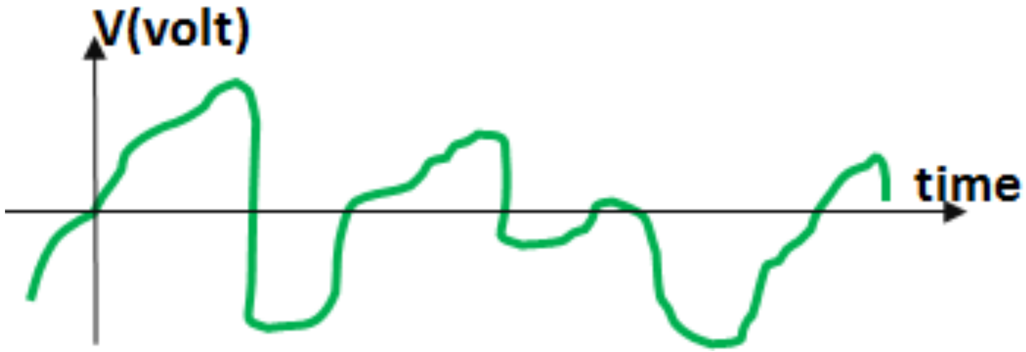
TMP36 Temp Sensor

Tc	Vout (mV)	Tf
-40	100	-40
-30	200	-22
-20	300	-4
-10	400	14
0	500	32
10	600	50
20	700	68
30	800	86
40	900	104
50	1000	122
60	1100	140
70	1200	158
80	1300	176
90	1400	194
100	1500	212
110	1600	230
120	1700	248

10 mV/deg C

~ 5 mV/deg F

Analog Signal



Check the pinout for yourself. Don't rely on this figure…

TMP36 Temp Sensor

Digital Signal

…

1101111

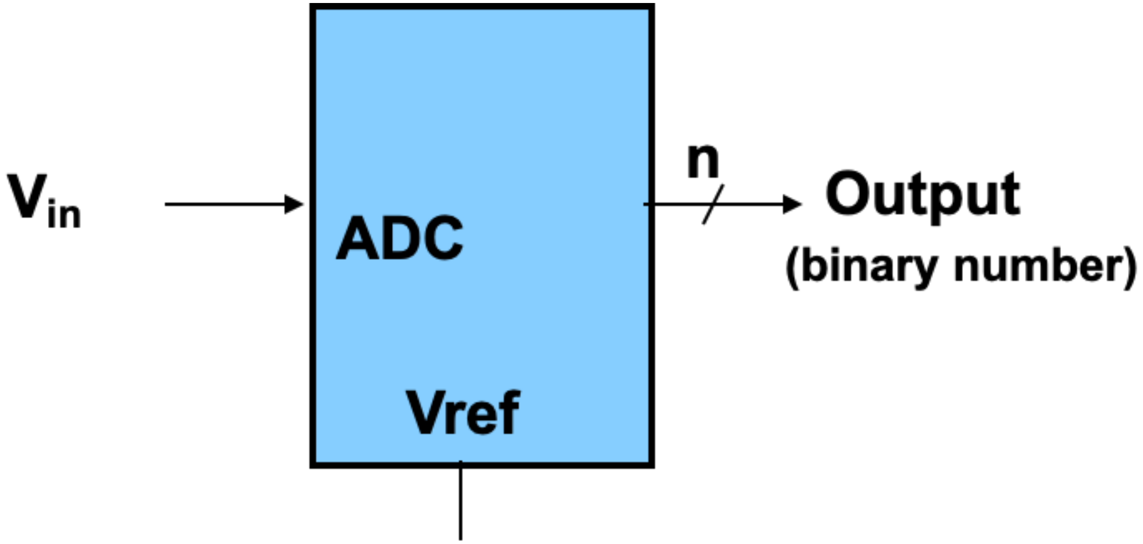
1010111

0101000

1010111

…

Analog to Digital Converter (ADC)

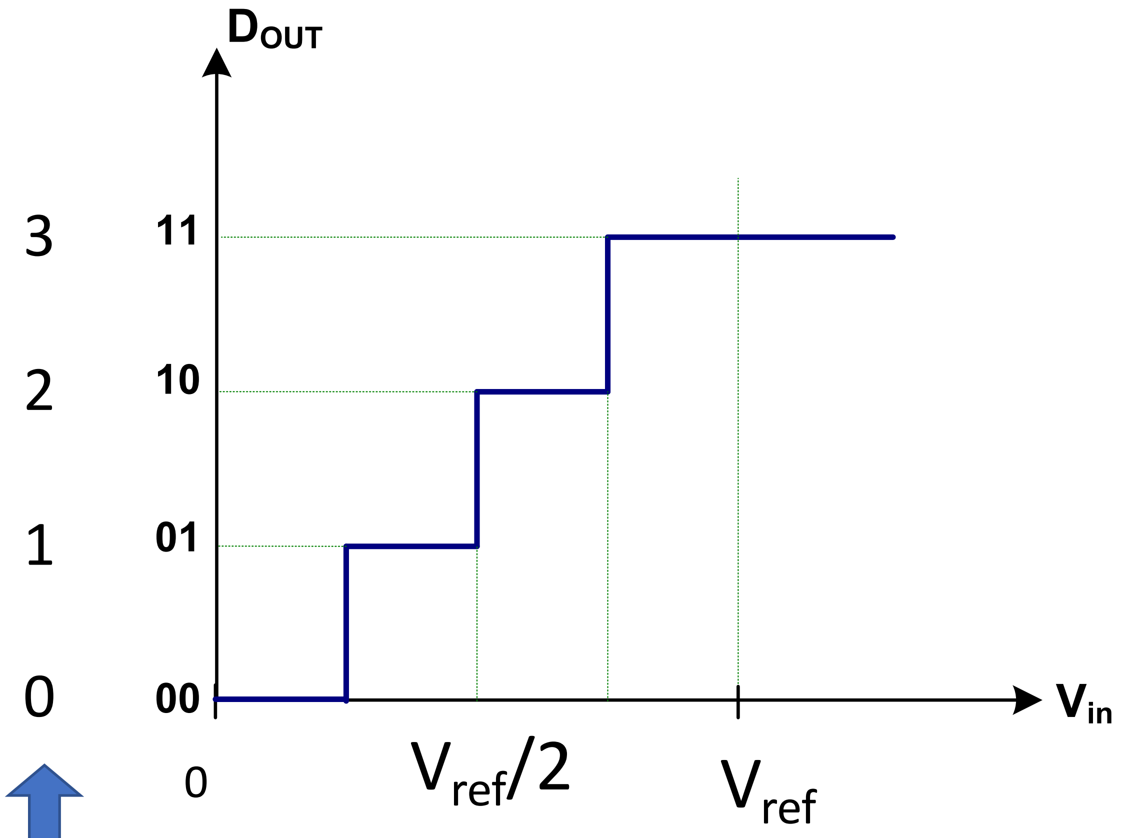


$$V_{step} = V_{ref}/2^n$$

$$D_{out} = V_{in}/V_{step}$$

$$V_{in} = D_{out}V_{step}$$

2 bit ADC (n=2)



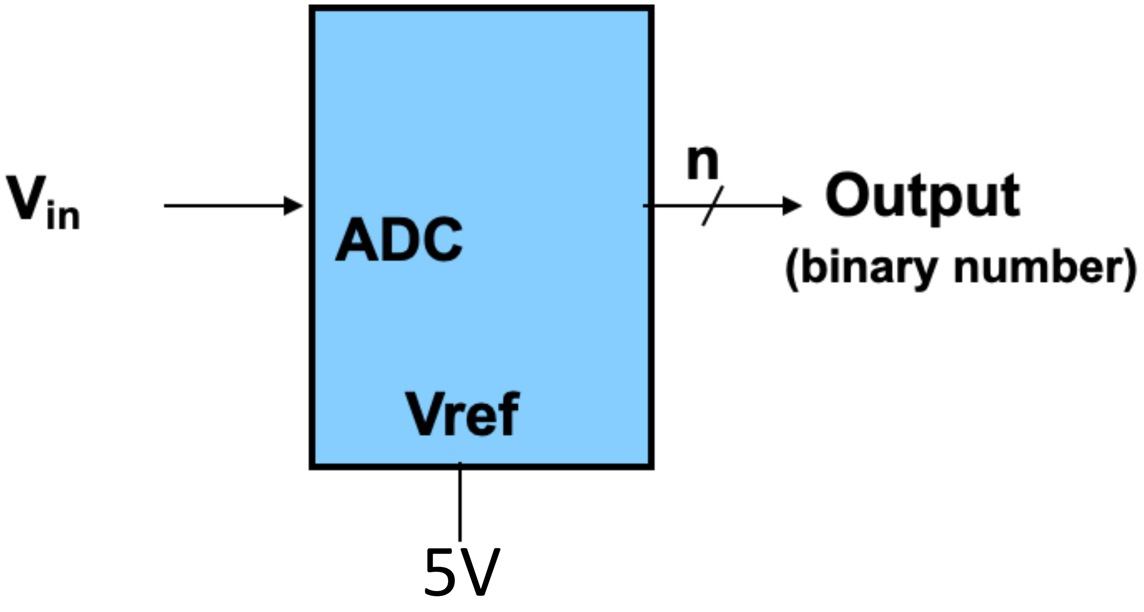
$$V_{step} = V_{ref}/4$$

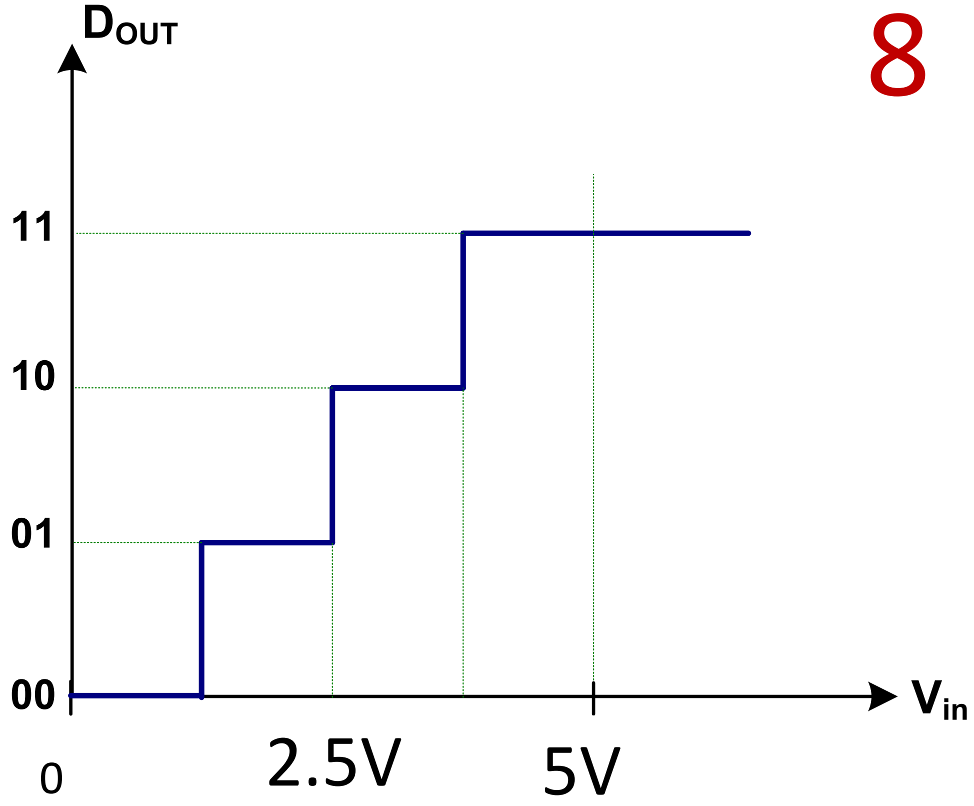
$$D_{out} = V_{in}/V_{step} = V_{in}/(V_{ref}/4)$$

n=2 bit ADC, V_{ref} = 5V

2^2 = 4 different values

$V_{step} = V_{ref}/4 = 5/4 = 1.25$ V = 1250 mV





Vin	Dout (binary)	Dout (decimal)
0 - 1.25V	00	0
1.25 - 2.5	01	1
2.5 - 3.75	10	2
3.75 - 5	11	3

Conversion from Dout back to input voltage:
Vin = Dout * Vstep

ex:

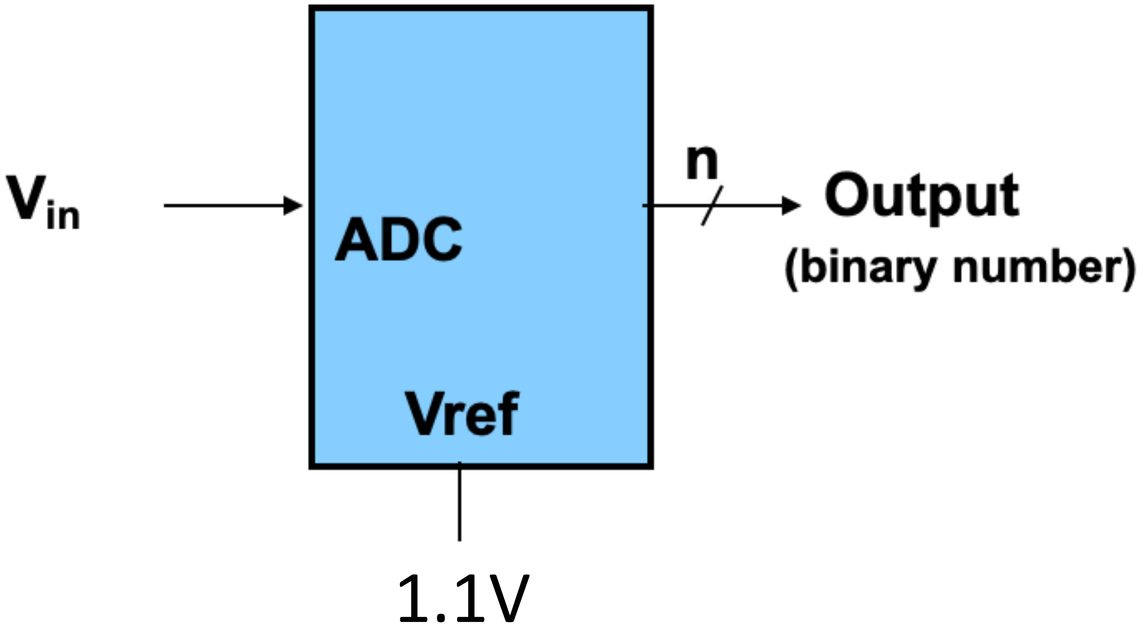
Vin = 2.6V; Dout = 2.6/1.250 = 2 or 0b10
Convert back: Vin = 2 *1.250 = 2.5V (quantization error)

Vin = 4.9; Dout = 4.9/1.250 = 3 or 0b11
Convert back: Vin = 3 * 1.250 = 3.75V (quantization error)

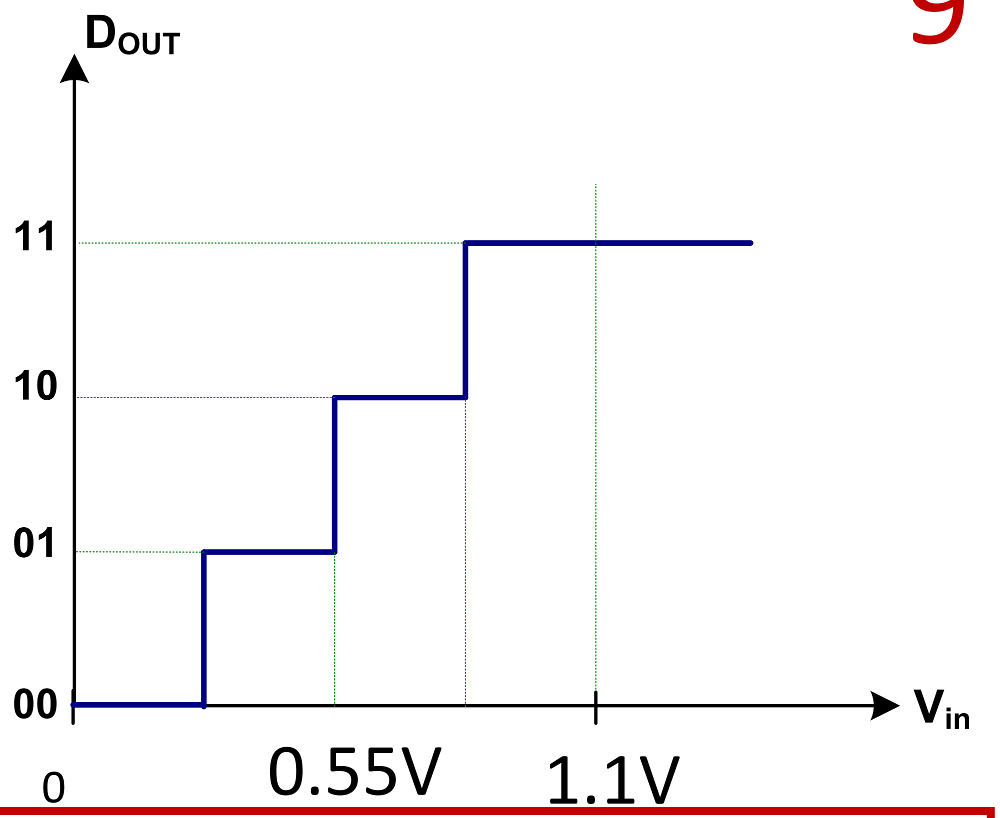
n=2 bit ADC, V_{ref} = 1.1V

2^2 = 4 different values

Vstep = $V_{ref}/4$= 1.1/4 = 0.275 V = 275 mV



Vin	Dout (binary)	Dout (decimal)
0 - 0.275V	00	0
0.275 - 0.55	01	1
0.55 - 0.825	10	2
0.825 - 1.1	11	3



Conversion from Dout back to input voltage:
Vin = Dout * Vstep

ex:

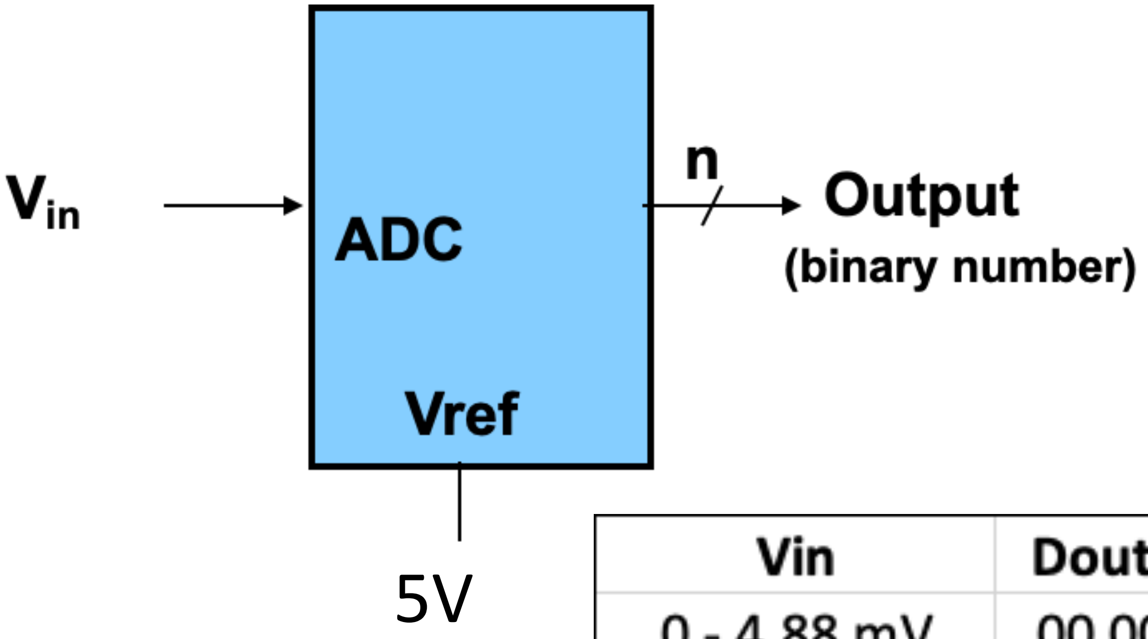
Vin = 0.5V; Dout = 0.5/0.275 = 1 or 0b01
Convert back: Vin = 1 *0.275 = 0.275V (quantization error)

Vin =0.9V; Dout = 0.9/0.275 = 3 = 0b11
Convert back: Vin = 3 * 0.375 = 0.825 (quantization error)

n=10 bit ADC, V_{ref} = 5V

2^{10} = 1024 different values

Vstep = V_{ref}/1024= 5/1024 = 0.00488 V = 4.88 mV



Convert from Dout back to Vin:
Vin = Dout * Vstep = Dout * 0.004883

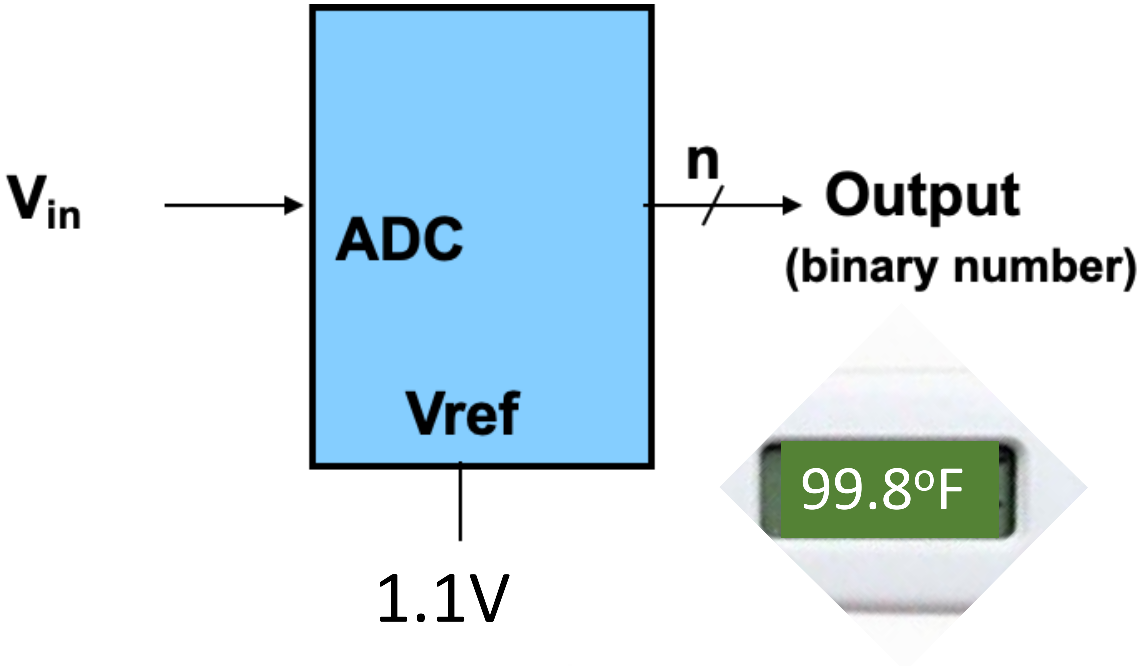
examples
Dout = 0 → Vin = 0 * 0.004883 = 0 V
Dout = 1 → Vin = 1 * 0.004883 = 0.004883=4.88 mV
Dout = 1023 → Vin = 1023 * 0.004883 = 4.995 V

Vin	Dout (binary)	Dout (hex)	Dout (decimal)
0 - 4.88 mV	00 0000 0000	0x000	0
4.88 - 9.76 mV	00 0000 0001	0x001	1
...	...	...	...
4990 - 4995 mV	11 1111 1110	0x3FE	1022
4995 - 5000 mV	11 1111 1111	0x3FF	1023

n=10 bit ADC, V_{ref} = 1.1V

2^{10} = 1024 different values

Vstep = V_{ref}/1024= 1.1/1024 = 0.00107 = 1 mV



Recall from slide #3 TMP36 sensitivity is:
10 mV/ °C   or ~ 5mV/ °F.

This ADC has a quantization of 1 mV/bit

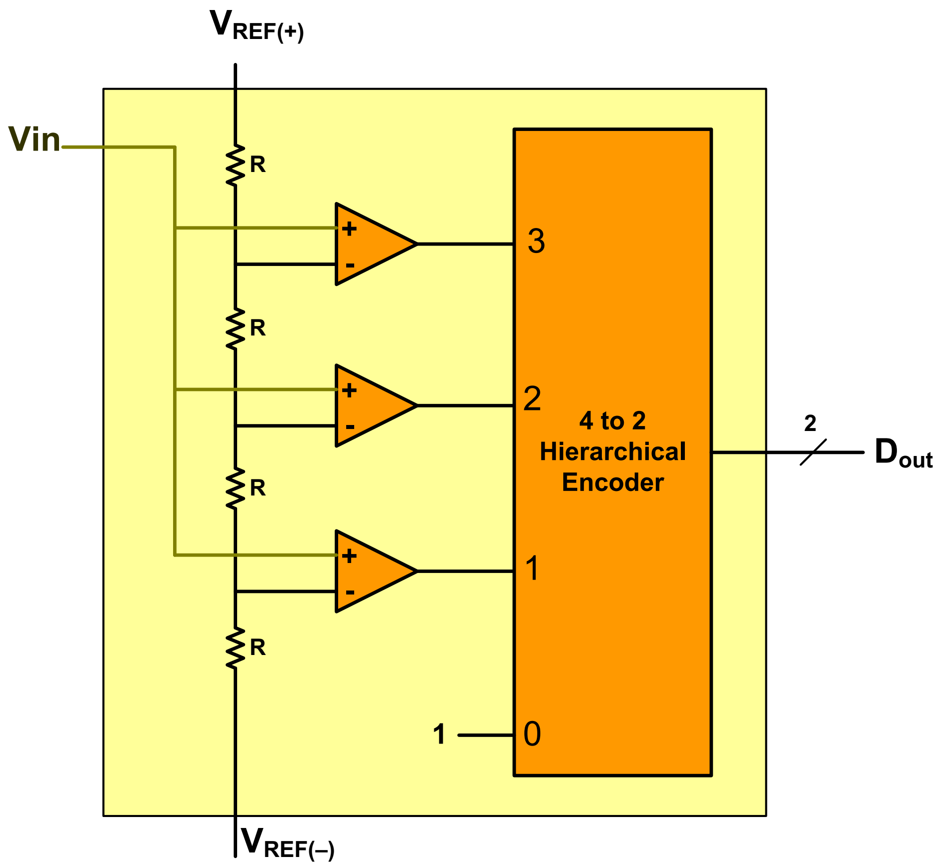
Temperature Measurement Resolution
(1 mV/bit) / (10mV/°C) = 0.1 °C /bit
(1 mV/bit) / (5mV/°F) = 0.2 °F /bit

Compare with previous slide (n=10 bit; V_{ref} = 5V) where ADC quantization is 4.88mV/bit ≅ 5mV/bit

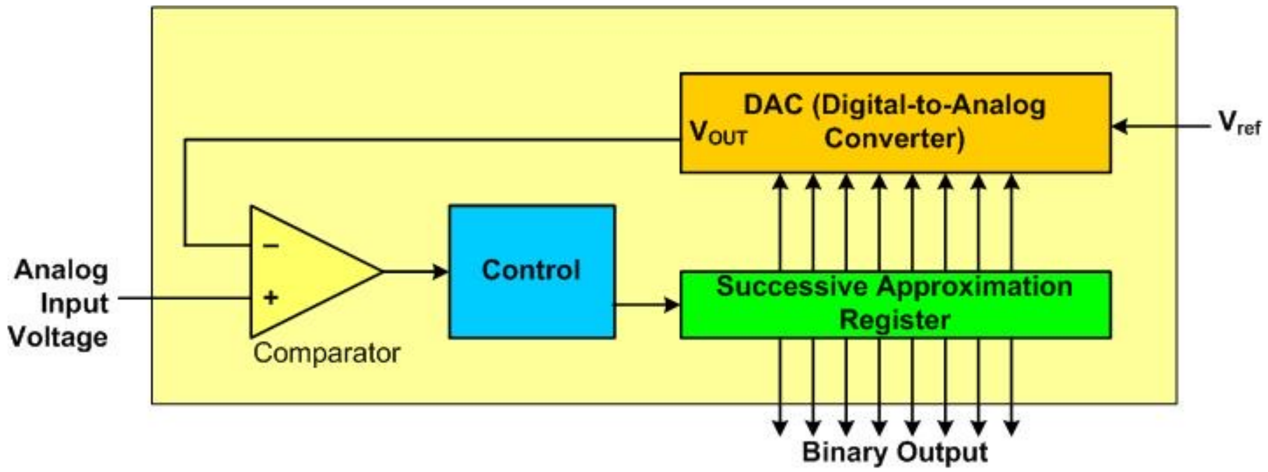
Temperature Measurement Resolution
(5 mV/bit) / (10mV/°C) = 0.5 °C /bit
(5 mV/bit) / (5mV/°F) = 1 °F /bit

Flash (left) vs Successive Approximation (right) ADC

Flash (direct conversion) ADC – voltage ladder



Successive Approximation ADC



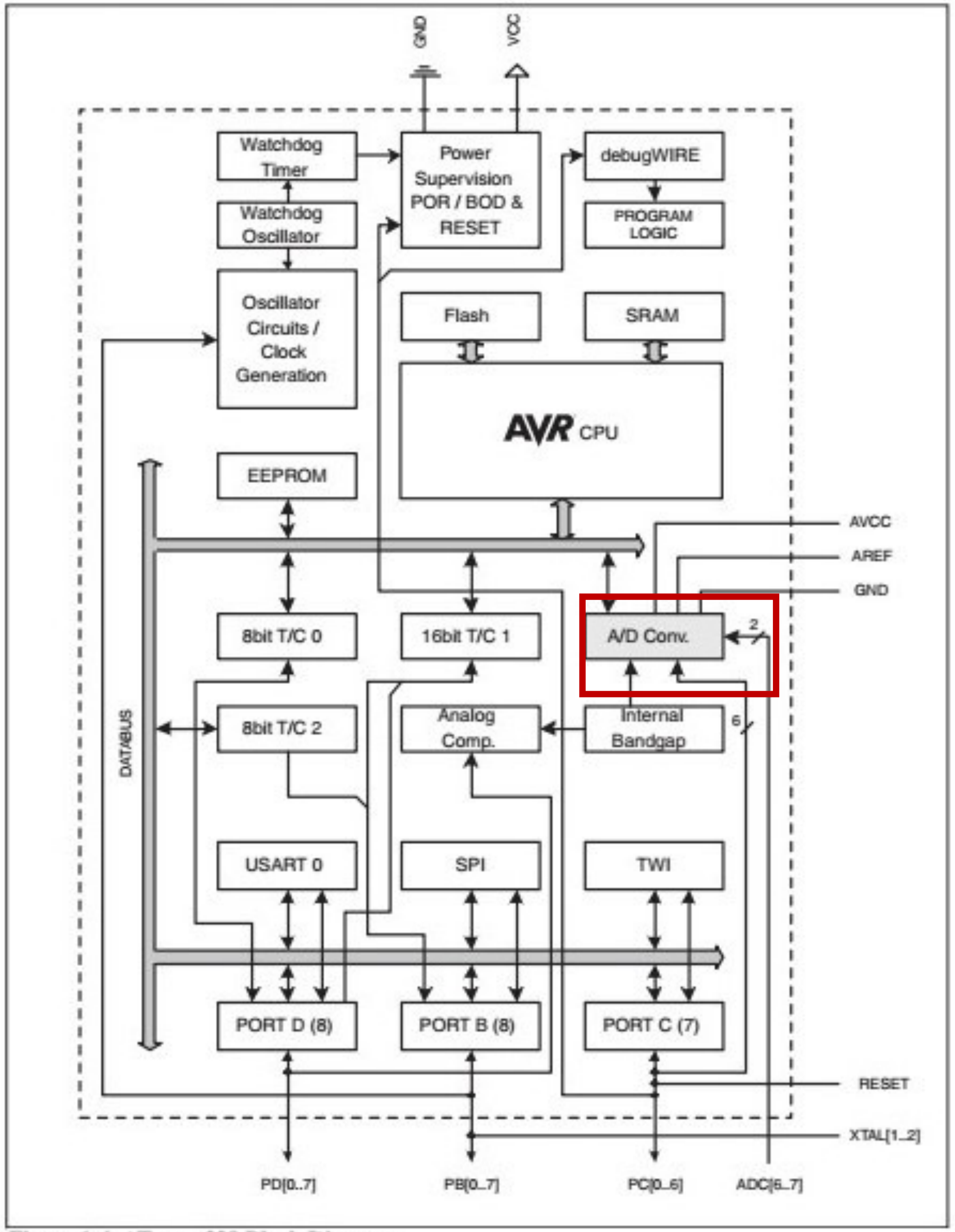
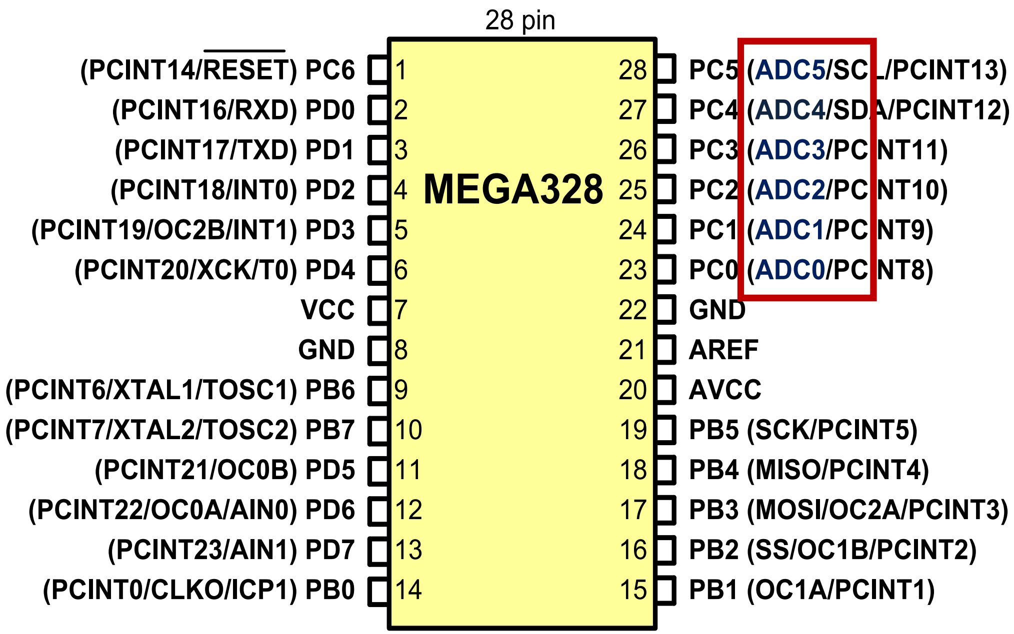


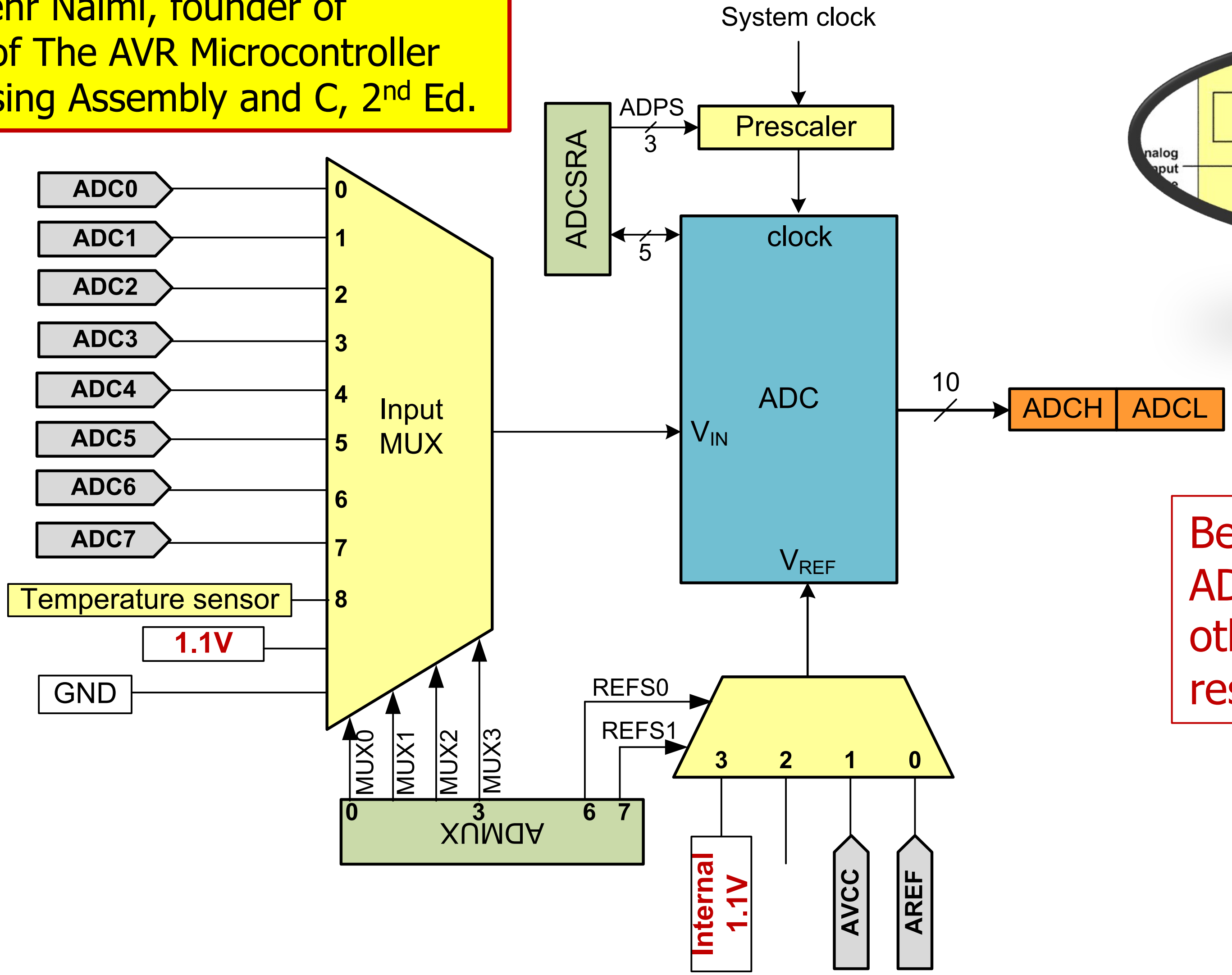
Figure 1-4. ATmega328 Block Diagram

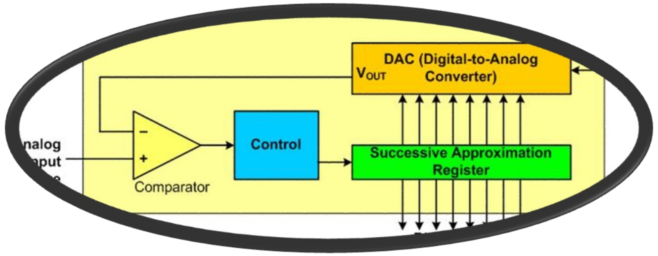


ATmega328P
10 bit successive approximation ADC
6 ADC channels on 28 pin DPIP (ADC0-ADC5)
2 additional channels on 32 pin QFP version (ADC6-ADC7)

ADC in AVR

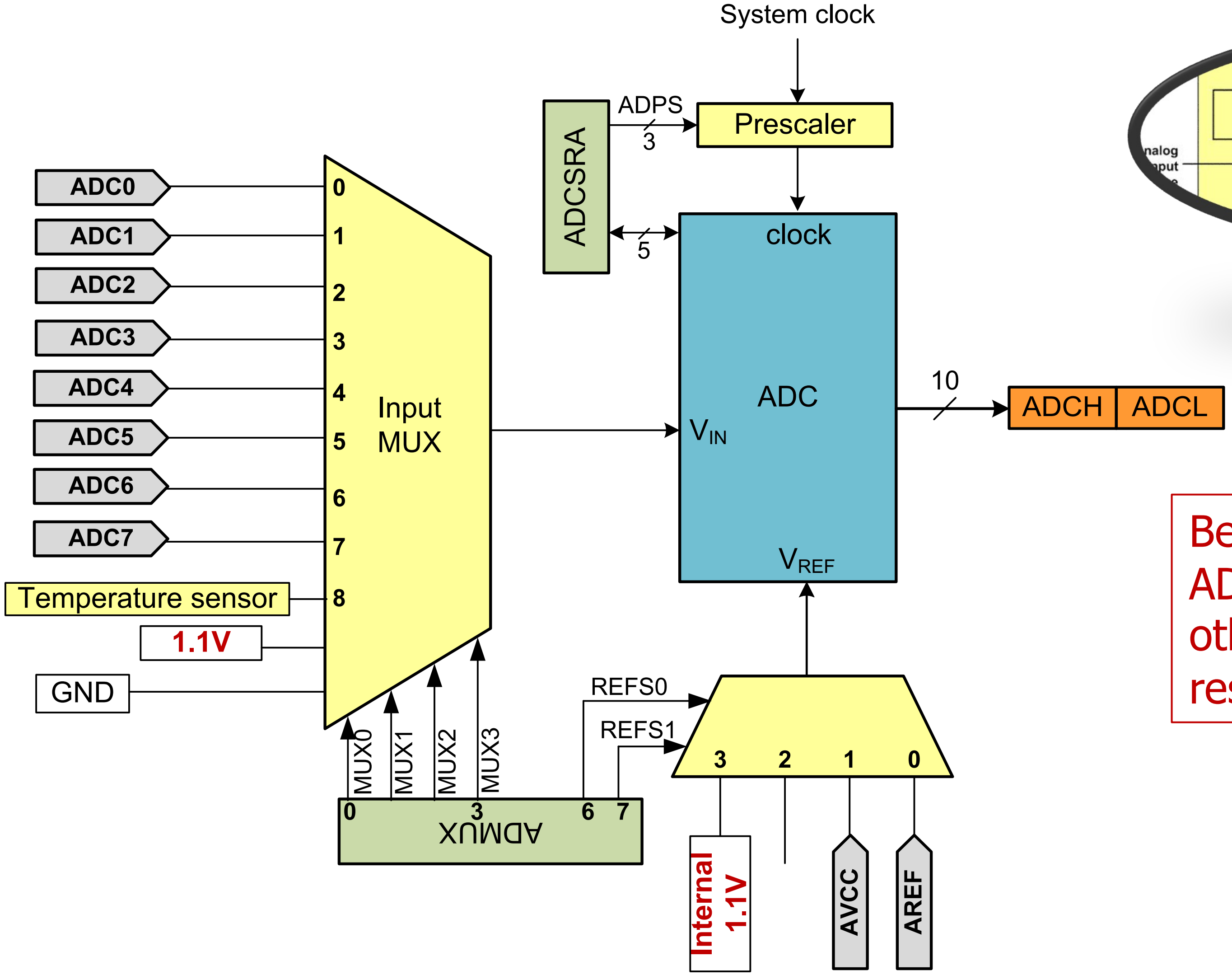
Slides courtesy of Mr. Sepehr Naimi, founder of Nicerland.com, co-author of The AVR Microcontroller and Embedded Systems using Assembly and C, 2nd Ed.

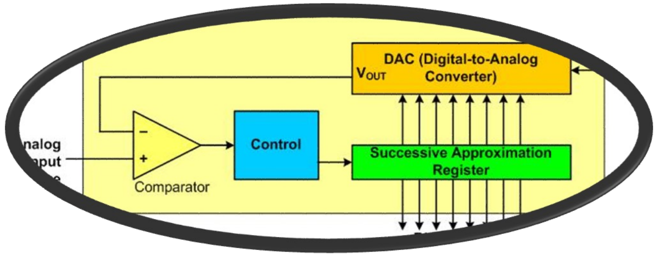




Be sure to read ADCL before ADCH otherwise incorrect result!

ADC in AVR



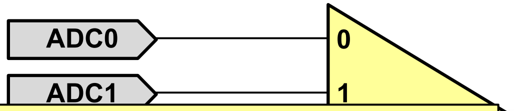


Be sure to read ADCL before ADCH otherwise incorrect result!

ADMUX

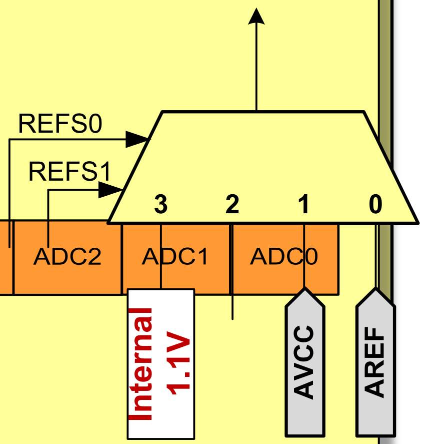
D7							D0
REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0

- MUX0-MUX3: input select



ADLAR = 0

ADCH								ADCL							
-	-	-	-	-	-	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0



ADLAR =1

ADCH								ADCL							
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	-	-	-	-	-	-

ADCSRA

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN- Bit7 ADC Enable

This bit enables or disables the ADC. Writing this bit to one will enable and writing this bit to zero will disable the ADC even while a conversion is in progress.

ADSC- Bit6 ADC Start Conversion

To start each coversion you have to write this bit to one.

ADATE- Bit5 ADC Auto Trigger Enable

Auto Triggering of the ADC is enabled when you write this bit to one.

ADIF- Bit4 ADC Interrupt Flag

This bit is set when an ADC conversion completes and the Data Registers are updated

ADIE- Bit3 ADC Interrupt Enable

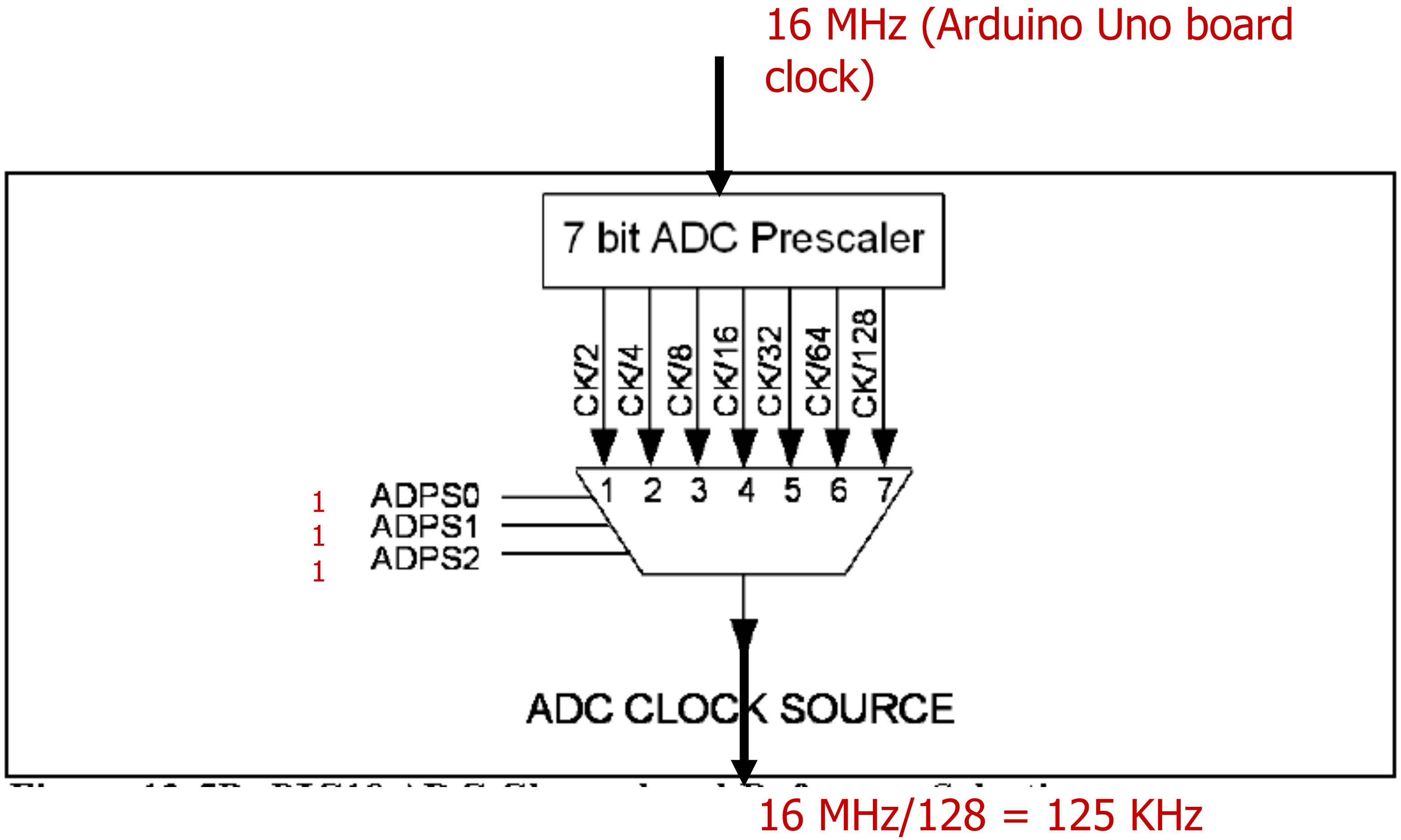
Writing this bit to one enables the ADC Conversion Complete Interrupt.

ADPS2:0- Bit2:0 ADC Prescaler Select Bits

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADC Prescaler

- PreScaler Bits let us change the clock frequency of ADC
- The frequency of ADC should not be more than 200 KHz

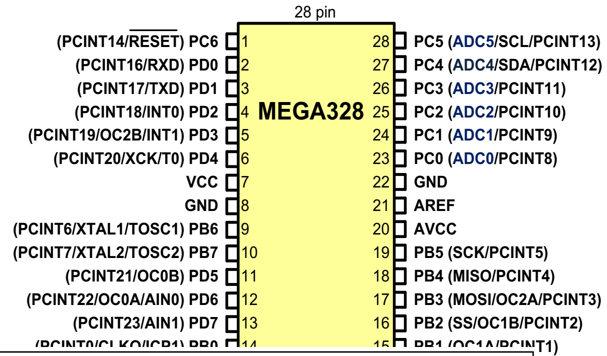


Steps in programming ADC

1) Make the pin for the selected ADC channel an input pin.
2) Enable ADC module
3) Select the conversion speed
4) Select voltage reference and ADC input channel.
5) Activate the start conversion bit by writing a one to the ADSC bit of ADCSRA.
6) Wait for the conversion to be completed by polling the ADIF bit in the ADCSRA register.
7) After the ADIF bit has gone HIGH, read the ADCL and ADCH registers to get the digital data output.
8) If you want to read the selected channel again, go back to step 5.
9) If you want to select another Vref source or input channel, go back to step 4.

Steps in programming ADC

1. Make the pin for the selected ADC channel an input pin.
2. Enable ADC module
3. Select the conversion speed
4. Select voltage reference and ADC input channels.
5. Activate the start conversion bit by writing a one to the ADSC bit of ADCSRA.
6. Wait for the conversion to be completed by polling the ADIF bit in the ADCSRA register.
7. After the ADIF bit has gone HIGH, read the ADCL and ADCH registers to get the digital data output.
8. If you want to read the selected channel again, go back to step 5.
9. If you want to select another Vref source or input channel, go back to step 4.



ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN- Bit7 ADC Enable
This bit enables or disables the ADC. Writing this bit to one will enable and writing this bit to zero will disable the ADC even while a conversion is in progress.

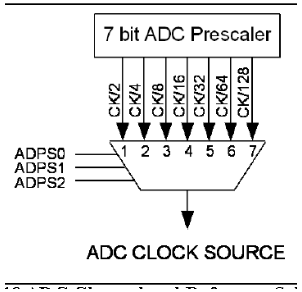
ADSC- Bit6 ADC Start Conversion
To start each coversion you have to write this bit to one.

ADATE- Bit5 ADC Auto Trigger Enable
Auto Triggering of the ADC is enabled when you write this bit to one.

ADIF- Bit4 ADC Interrupt Flag
This bit is set when an ADC conversion completes and the Data Registers are updated

ADIE- Bit3 ADC Interrupt Enable
Writing this bit to one enables the ADC Conversion Complete Interrupt.

ADPS2:0- Bit2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.



1. DDRC &= ~(1<<ADC0); //ADC0 as input

2. ADCSRA = (1<<ADEN);
3. ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
Note: ADCSRA = 1000 0111 = 0x87 combine 2&3 as:
ADCSRA = 0x87;

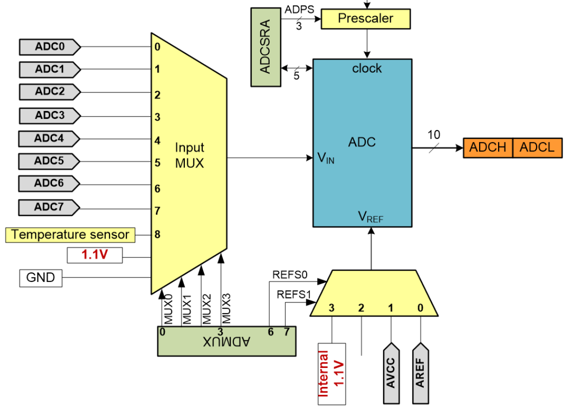
4. ADMUX =0x40 ; // 0100 0000 ADC0, Vref=AVCC=5V;

5. ADCSRA |=1<<ADSC

6.While (ADCSRA & (1<<ADIF) ==0); //wait till conversion is finished

7. digWord = ADCL | ADCH<<8; //digWord is int

D7							D0
REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0



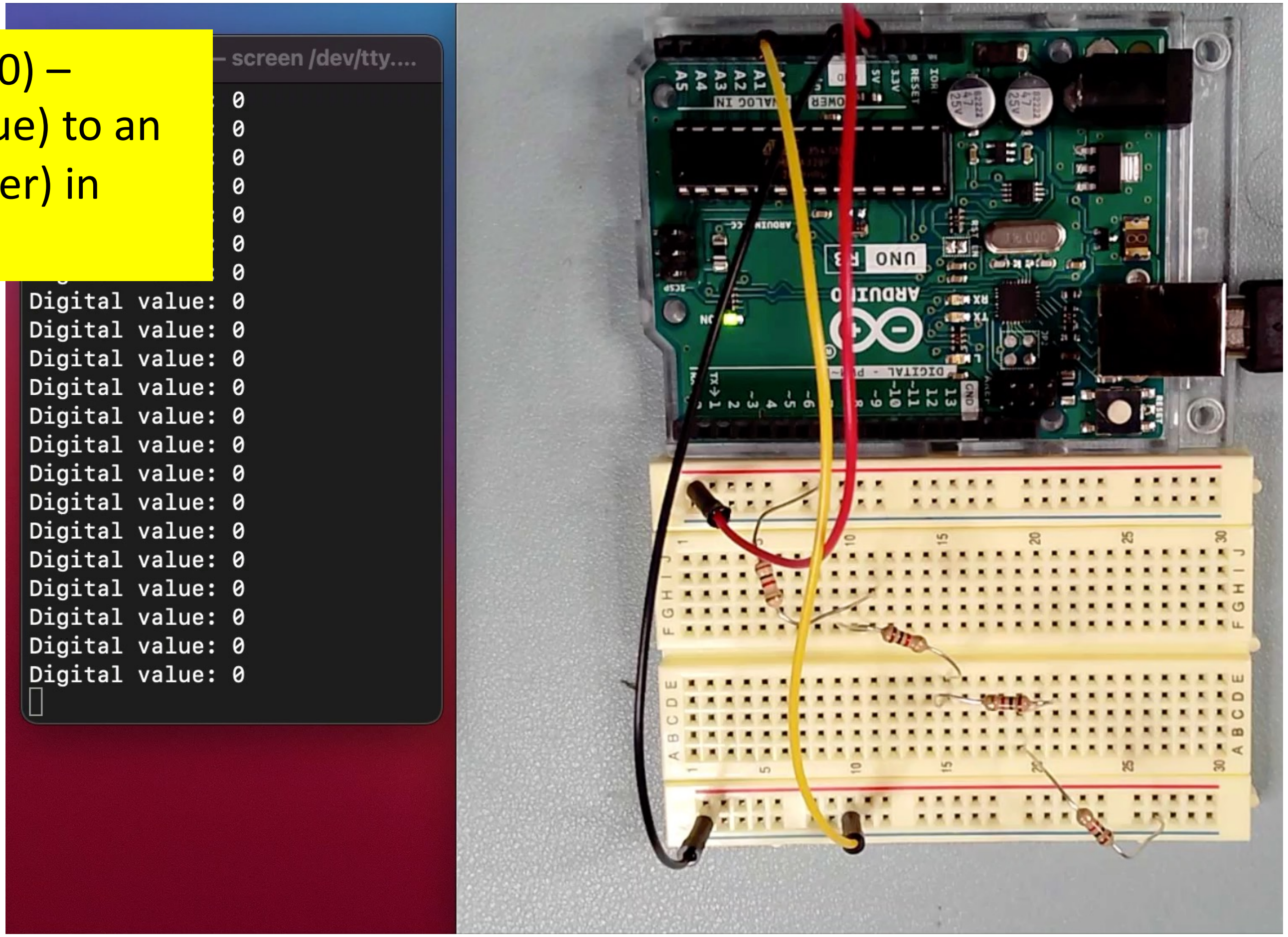
ADC Sampling with digital output word to serial monitor

```c
/*************************************************************
 * adcladderserialdemo.c
 * polls ADC pin 0 and writes 10 bit digital value to serial port
 * D. McLaughlin 2/28/21 written for ECE-231 Spring 2021
 *************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>     // Declares strlen() fu
#include <stdlib.h>     // Declares itoa() func
void uart_init(void);
void uart_send(char letter);
void send_string(char *stringAddress);

int main(void)
{
    unsigned int digitalValue;
    char buffer[6];
    ADMUX = 0x40;       // Select ADC0; Vref=AVcc=5V
    ADCSRA = 0x87;      // Enable ADC; set speed 125 KHz
    uart_init();
    while (1) {
        ADCSRA |= (1 << ADSC);                // Start ADC conversion
        while ((ADCSRA & (1 << ADIF)) == 0);// Wait till ADC finishes
        digitalValue = ADCL | (ADCH << 8);  // Read ADCL first !
        itoa(digitalValue, buffer, 10);       // Convert to character string
        send_string("Digital value: ");
        send_string(buffer);                  // Tx string
        uart_send(13);                        // Tx carriage return
        uart_send(10);                        // Tx line feed
        _delay_ms(200);
    }
    return 0;
}

> // initialize ATmega328P UART: enable TX, 8 bit, 1 stop bit,…
> void uart_init(void) {…
  // send a single ASCII character via UART
> void uart_send(char letter) {…
  // send string of ASCII characters
> void send_string(char *stringAddress) {…
```

itoa(digitalValue, buffer, 10) – converts an int (digitalValue) to an ascii character string (buffer) in base 10.



demo:
adcLadderSerialDemo

let’s look at an improved version that relies on user-contributed functions to handle all the ADC & UART tasks

https://github.com/ProfMcL/ECE231/blob/main/code/ADC_UART_OLED/adc_serial_test.c

0.96’ OLED – Organic LED Display

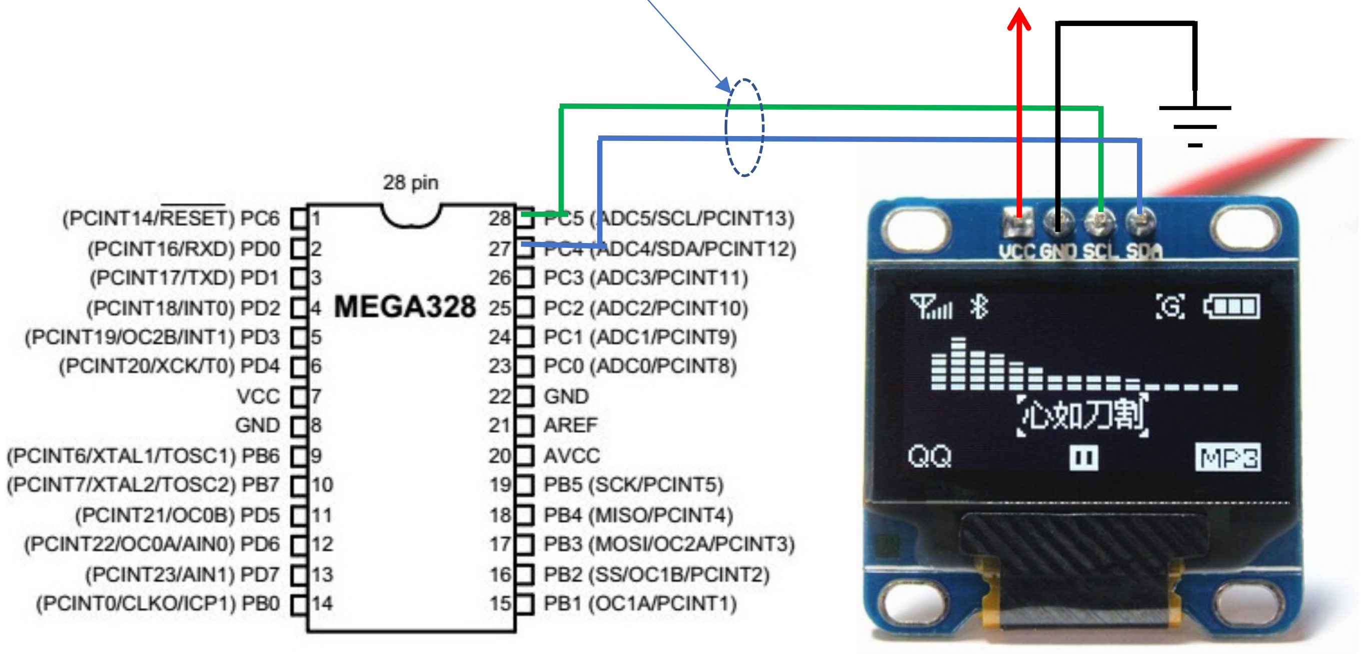
- 128 x 64 dot matrix display,
- SSD1306 display driver
- I2C interface

I²C Interface

SCL – serial clock

SDA – serial data

VCC = 5V



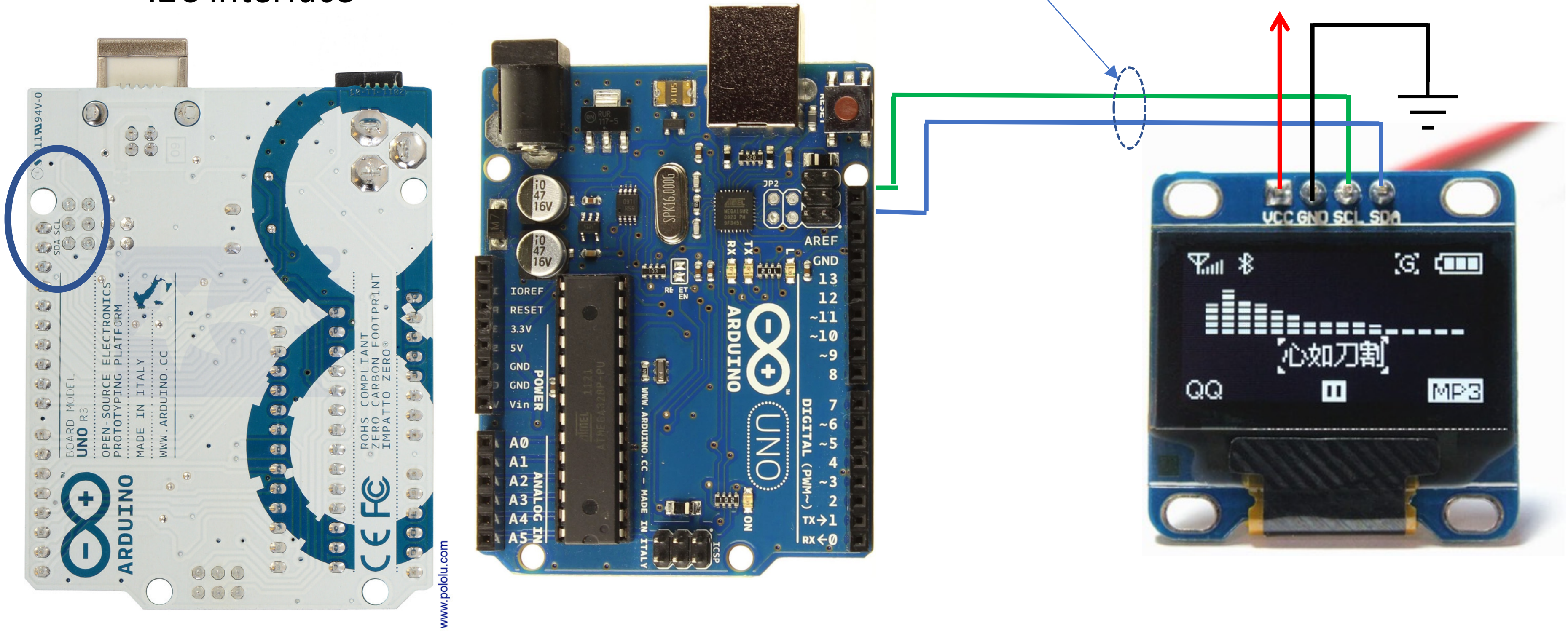
I²C = I2C = Inter-Integrated Circuit Interface

0.96’ OLED – Organic LED Display

- 128 x 64 dot matrix display,
- SSD1306 display driver
- I2C interface

I²C Interface
SCL – serial clock
SDA – serial data

VCC = 5V



I²C = I2C = Inter-Integrated Circuit Interface

Programming the OLED:

Write register-level code to control the dot matrix display and handle the I2C communication

or

We can use external libraries to handle the low-level coding details:

SSD1306.c
SSD1306.h
i2c.c
i2h.c

These libraries are not part of the C language or its standard libary. They are user-defined libraries, like my_ADC_lib.c and my_UART_lib.c that someone wrote and published on github for anyone to use (at their own risk).

let’s look at code that displays adc samples on the OLED display using the following user-contributed functions to handle all the ADC, UART, SSD1306, and I2C tasks:

my_adc_library.c

my_uart_library.c

SSD1306.c

i2c.c

https://github.com/ProfMcL/ECE231/blob/main/code/ADC_UART_OLED/adc_oled_test.c