

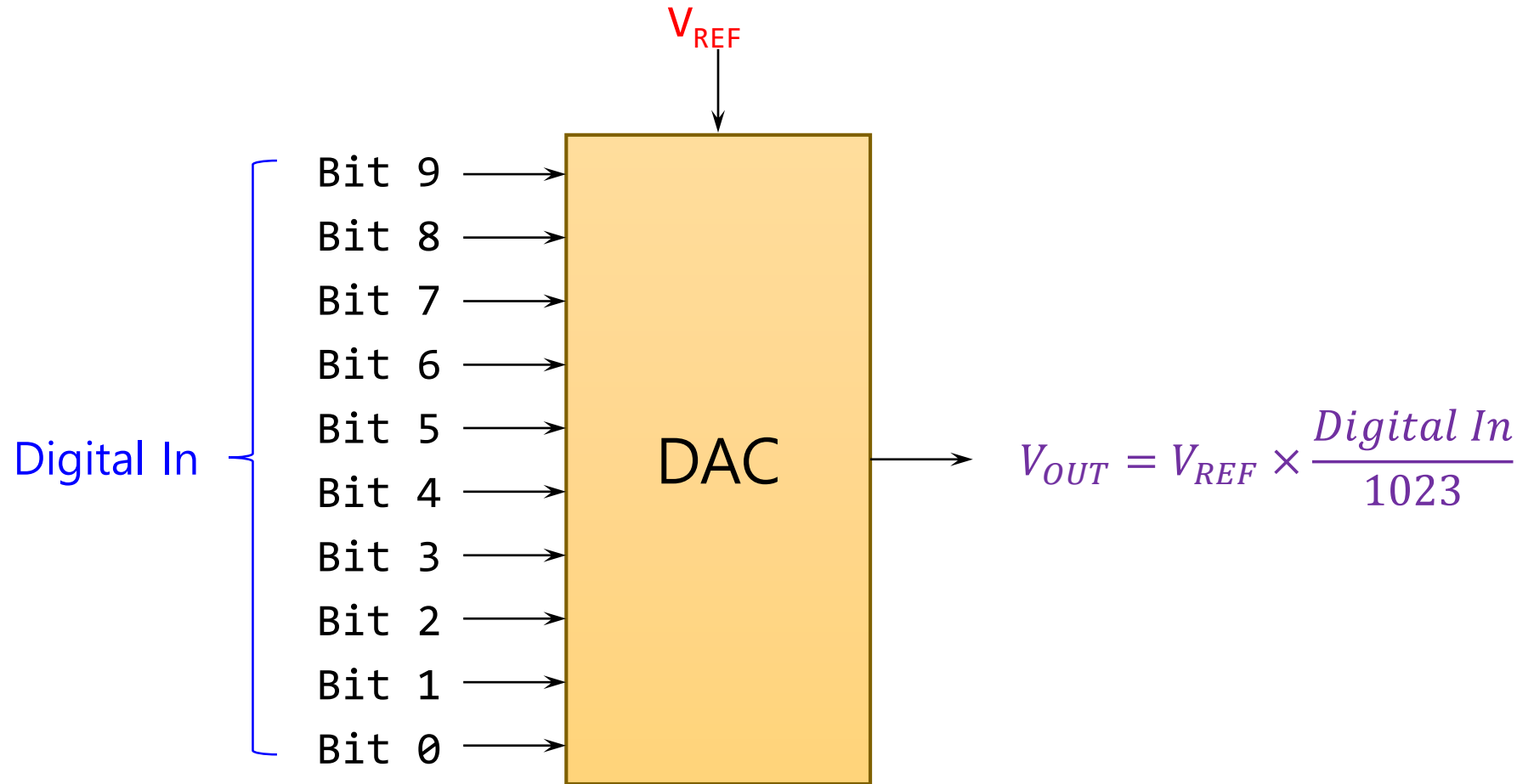
ADC

Analog-to-Digital Converter

ATmega328PB ADC Features

- 10-bit Resolution
- 13 – 260 μ s Conversion Time
- Up to 76.9 kSPS (Up to 15 kSPS at Maximum Resolution)
- 8 Multiplexed Single Ended Input Channels
- 2 Additional Multiplexed Single Ended Input Channels (TQFP and QFN/MLF Package only)
- Temperature Sensor Input Channel
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

DAC (Digital to Analog Converter)



Successive Approximation ADC

DAC = Digital-to-Analog
converter

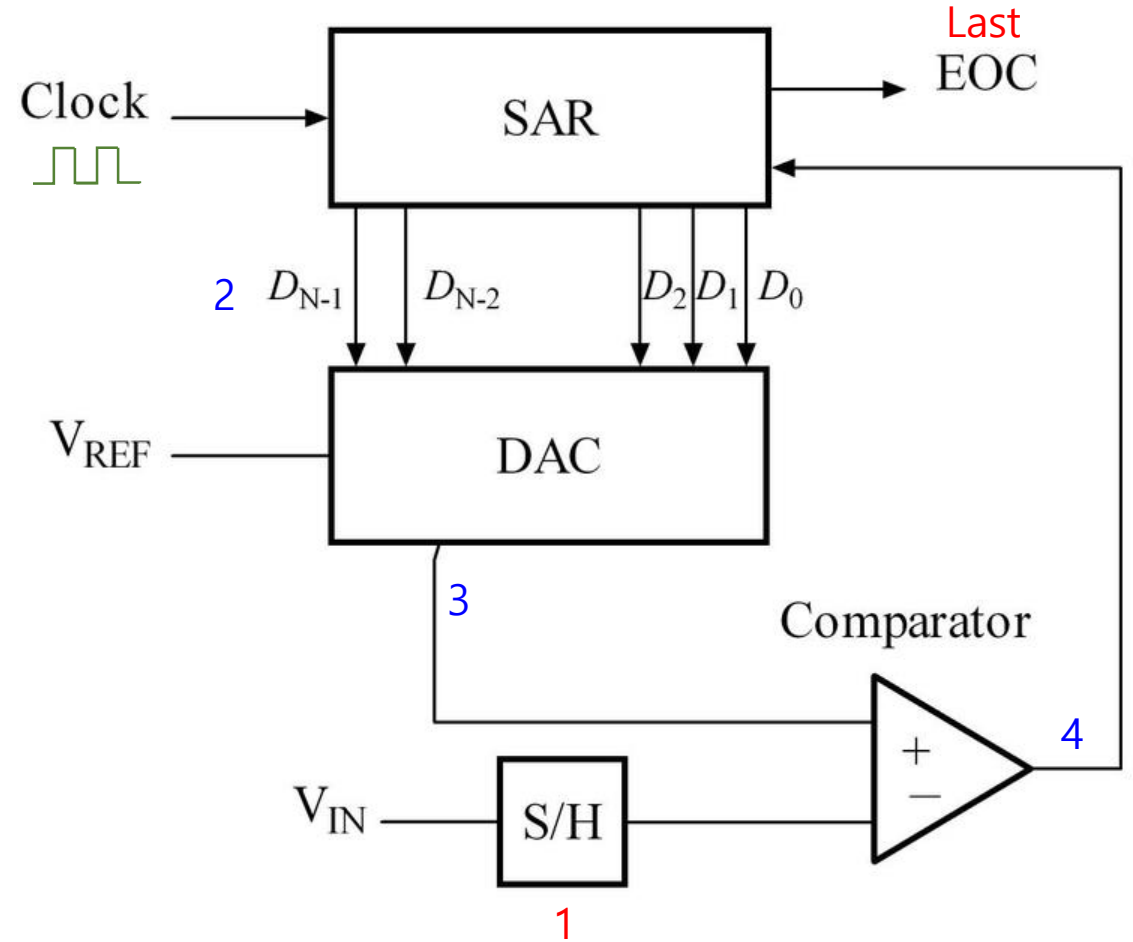
EOC = end of conversion

SAR = successive approximation
register

S/H = sample and hold circuit

V_{IN} = input voltage

V_{REF} = reference voltage



ATmega328PB ADC Conversion Results

- For single ended conversion, the result is

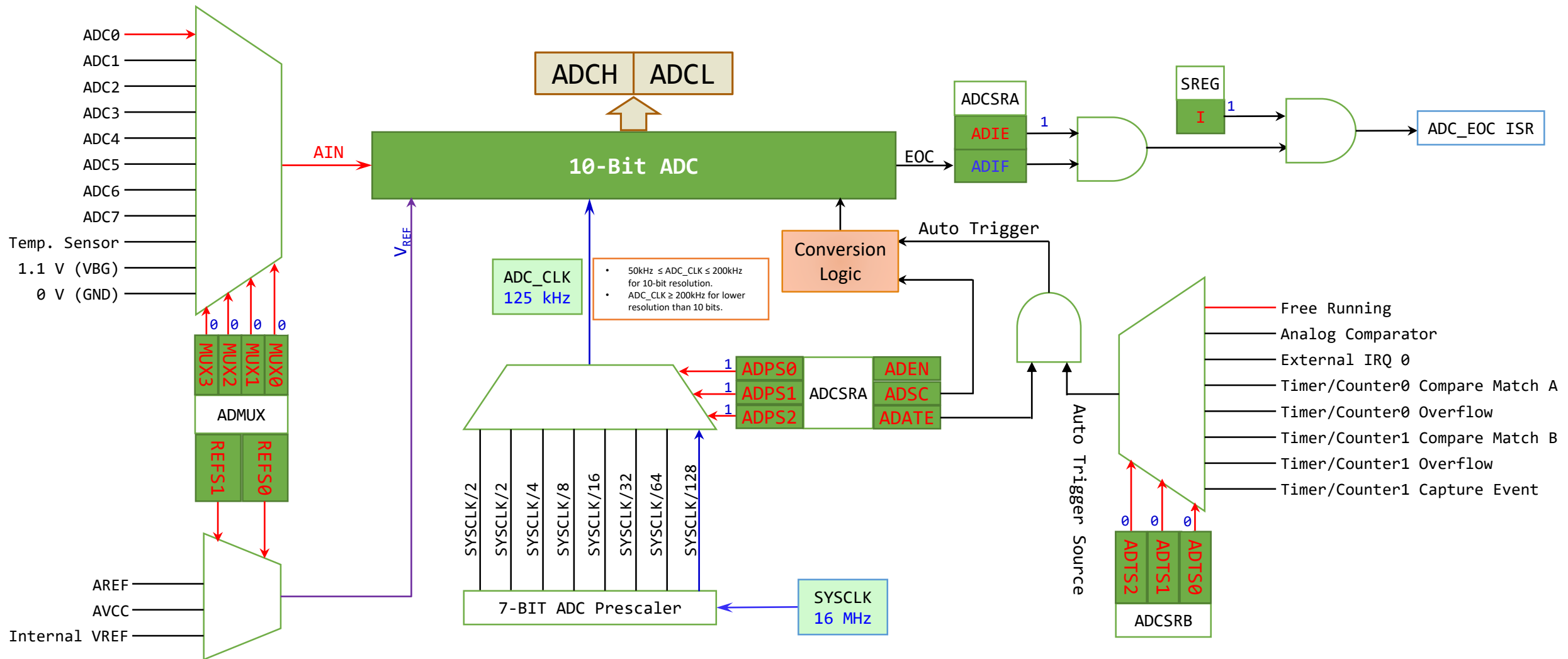
$$ADC_{out} = \frac{V_{IN}}{V_{REF}} \times 1023$$

where V_{IN} is the voltage on the selected input pin, and V_{REF} the selected voltage reference.

- 0x000 (0b0000000000): represents analog ground.
- 0x3FF (0b1111111111): represents the selected reference voltage minus one LSB.



ATmega328PB ADC Block Diagram



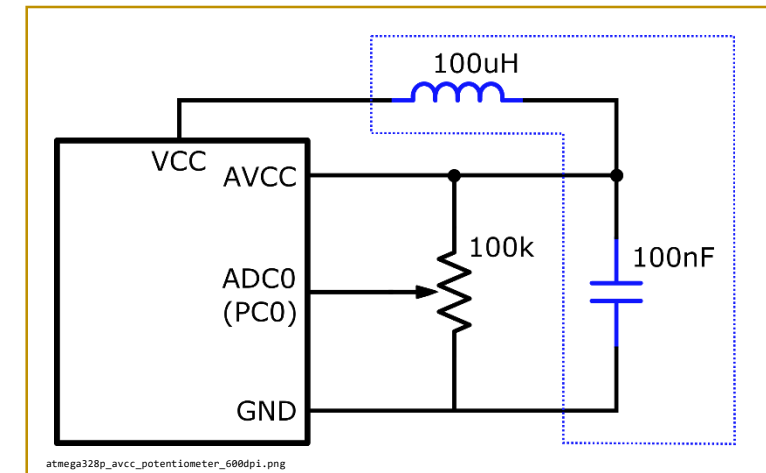
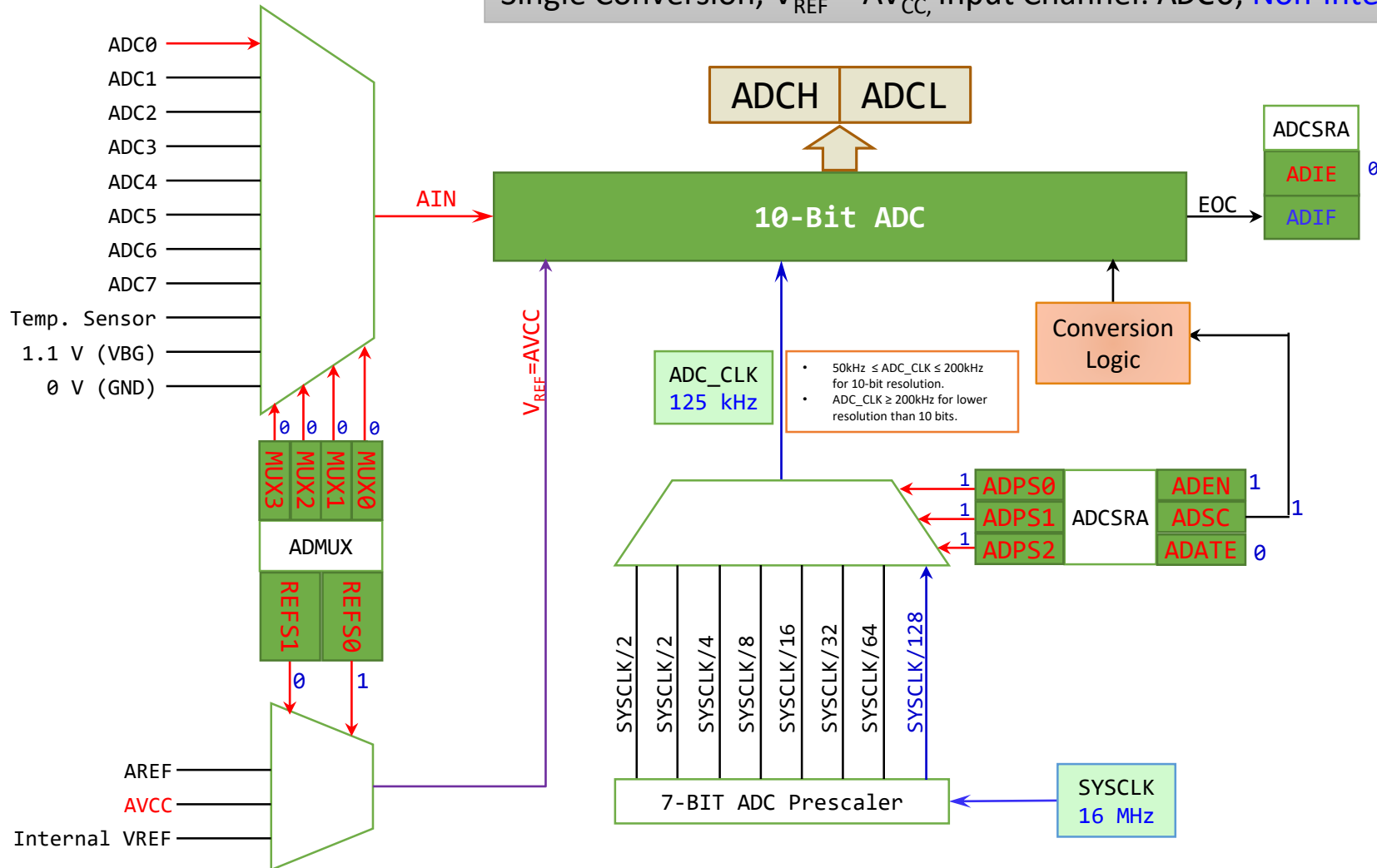
ATmega328P ADC Clock

- ADC Clock
 - By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution.
 - If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.
- ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5

ATmega328PB ADC Example 1 (Polling, Single Channel) (1)

Single Conversion, $V_{REF} = AV_{CC}$, Input Channel: ADC0, Non-interrupt, System Clock Frequency= 16MHz



ATmega328PB ADC Example 1 (Polling, Single Channel) (2)

Single Conversion, $V_{REF} = AV_{CC}$, Input Channel: ADC0, Non-interrupt, System Clock Frequency= 16MHz

```
/* adc_single_channel_manual_trig.c
 * Convert single channel (ADC0) manually */

#include <stdio.h>
#include <avr/io.h>

void uart_init(void);

int main(void)
{
    unsigned int adc_data;

    uart_init();                // 9,600 bps, 8-data, 1 stop

    ADMUX = 0b01000000;        // Vref=AVCC, convert ADC0.
    ADCSRA = 0b10010111;        // Enable ADC. ADC_CLK=16MHz/128=125kHz.

    while (1)
    {
        ADCSRA |= (1 << ADSC);    // Start ADC
        while ((ADCSRA & (1 << ADIF)) == 0) // Wait for End of Conversion
            ;
        ADCSRA |= (1 << ADIF);    // Clear ADIF

        adc_data = ADCW;          // Read 10-bit ADC result
        printf("ADC result=%u\n", adc_data); // display the result
    }
}
```

```
/* usart.c */

#define F_CPU 16000000UL
#define UART_BAUD_RATE 9600UL
#define DIVISOR (((F_CPU / (UART_BAUD_RATE * 16UL))) - 1)

#include <stdio.h>
#include <avr/io.h>

int uart_putchar(char ch, FILE *stream);

FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_RW);

void uart_init(void)
{
    UCSRB |= (1 << RXEN0) | (1 << TXEN0); // enable TX and RX
    UCSRC |= (1 << UCSZ00) | (1 << UCSZ01); // 8-bit word
    UBRR0 = DIVISOR;

    stdout = &uart_str;
}

int uart_putchar(char ch, FILE *stream)
{
    while ((UCSR0A & (1 << UDRE0)) == 0);

    UDR0 = ch;
    return 0;
}
```

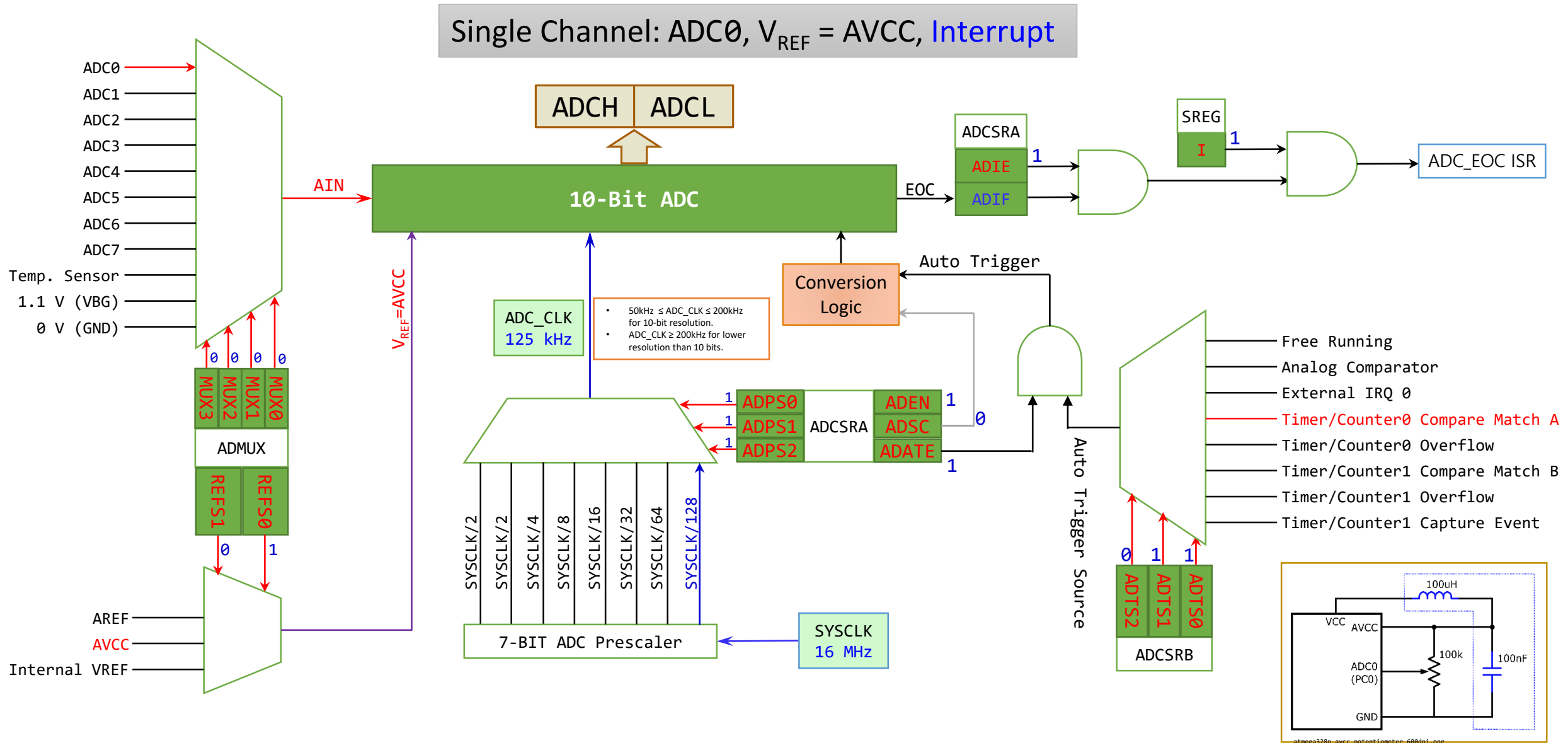


ATmega328PB ADC Example 2 (Interrupt, Single Channel) (1)

Make an application which converts ADC0 analog input signal to 10-bit digital signal.

- System Clock Frequency: 16MHz
- V_{REF} : AVCC
- ADC Clock: 125 kHz
- Sampling rate: about 100 samples per second (SPS)
- ADC auto trigger signal: Timer/Counter0 Compare Match A

ATmega328PB ADC Example 2 (Interrupt, Single Channel) (2)



ATmega328PB ADC Example 2 (Interrupt, Single Channel) (3)

Single Channel: ADC0, $V_{REF} = AVCC$, Interrupt

```
/* Single channel(ADC0), Timer/Counter0 Compare Match trigger,
   End of Conversion Interrupt. */

#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>

void uart_init(void);

unsigned int adc_data_raw;
volatile unsigned char eoc_flag;    // must be volatile

int main(void)
{
    unsigned int adc_data;

    uart_init();                    // for printf()

    ADMUX = 0b01000000; // Vref=AVCC, convert ADC0.
    ADCSRA = 0b10111111; // Enable ADC.
                          // ADC Auto trigger.
                          // Clear ADIF
                          // Enable ADC Interrupt.
                          // ADC_CLK=125kHz.

    ADCSRB = 0b011; // ADC Auto Trig Src: TC0 Compare Match A
```

```
TCCR0A = 0b10; // Set TC0 to CTC mode: TOP=OCR0A
TCCR0B = 0b101; // TC0 Prescale Ratio=1024 (15,625Hz)
OCR0A = 156 - 1; // ADC auto trig freq = 15,625Hz/156 ≈ 100Hz

sei();

while (1)
{
    while (eoc_flag == 0); // Wait for End of Conversion

    cli(); // Disable ADC EOC interrupt
    adc_data = adc_data_raw; // Read 10-bit ADC result
    sei(); // Enable ADC EOC interrupt

    eoc_flag = 0;

    printf("ADC result=%u\n", adc_data); // display the result
}

ISR(ADC_vect)
{
    adc_data_raw = ADCW; // Read 10-bit ADC result
    eoc_flag = 1; // indicate that new ADC data available

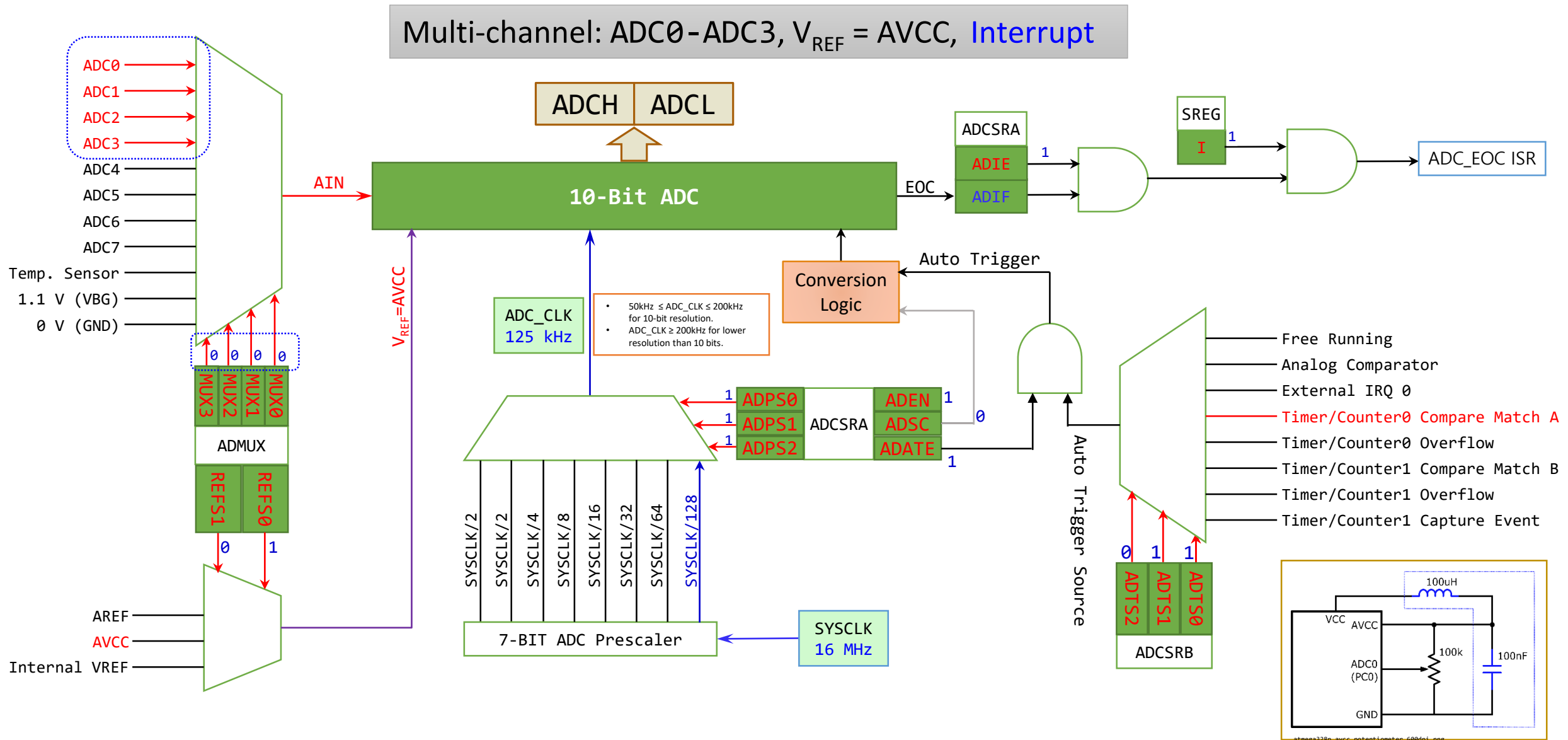
    TIFR0 |= (1 << OCF0A); // Clear OCF0A flag.
}
```

ATmega328PB ADC Example 3 (Interrupt, Multi-Channel) (1)

Make an application which converts ADC0-ADC3 analog input signals to 10-bit digital signals.

- System Clock Frequency: 16MHz
- V_{REF} : AVCC
- ADC Clock: 125 kHz
- Sampling rate: about 25 samples per second (SPS) per channel
- ADC auto trigger signal: Timer/Counter0 Compare Match A

ATmega328PB ADC Example 3 (Interrupt, Multi-Channel) (2)



ATmega328PB ADC Example 3 (Interrupt, Multi-Channel) (3)

Multi-channel: ADC0-ADC3, $V_{REF} = AVCC$, Interrupt

```
/* Multi-channel(ADC0-ADC3), Timer/Counter0 Compare Match trigger,
   End of Conversion Interrupt. */

#define NUM_ADC_CH 4

#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>

void uart_init(void);

unsigned int adc_data_raw[NUM_ADC_CH];
volatile unsigned char eoc_flag; // must be volatile

int main(void)
{
    unsigned char i;
    unsigned int adc_data[NUM_ADC_CH];

    uart_init(); // 1Mbps

    ADMUX = (0b01 << REFS0); // Vref=AVCC, convert ADC0.
    ADCSRA = (1 << ADEN) // Enable ADC.
            | (1 << ADSC) // ADC Auto trigger.
            | (1 << ADIF) // Enable ADC Interrupt.
            | (0b111 << ADPS0); // ADC_CLK=125kHz.

    ADCSRB = (0b011 << ADTS0); // ADC Auto Trig Src: TC0 Cmp Mat A
    TCCR0A = (0b10 << WGM00); // Set TC0 to CTC mode: TOP=OCR0A
    TCCR0B = (0b101 << CS00); // TC0 Prescale Ratio=1024 (15,625Hz)
    OCR0A = 155; // ADC auto trig freq≈100Hz
```

```
sei(); // Enable global interrupt

while (1)
{
    while (eoc_flag == 0); // Wait for End of Conversion

    cli();
    for (i=0; i<NUM_ADC_CH; i++)
        adc_data[i] = adc_data_raw[i]; // Read ADC result
    sei();
    eoc_flag = 0;
    printf("ADC0=%u, ADC1=%u, ADC2=%u, ADC3=%u\n", adc_data[0], adc_data[1],
        adc_data[2], adc_data[3]); // display the result
}

ISR(ADC_vect)
{
    unsigned char ch_num;

    ch_num = ADMUX & 0x0F; // Extract current ADC channel number
    adc_data_raw[ch_num] = ADCW; // Save result
    ch_num++; // Next ADC channel
    if (ch_num == NUM_ADC_CH) // Last channel?
    {
        ch_num = 0; // Yes. Go to the 1st channel
        eoc_flag = 1; // Indicate that new ADC data available
    }
    ADMUX = (0b01 << REFS0) | ch_num; // Update next ADC channel number
    TIFR0 |= (1 << OCF0A); // Clear OCF0A flag.
}
```

Life Of Programmer's



10:30 A.M

Starts work

After 2 hrs

A Bug Occur



After 5 hrs

Java Broken



At Night

Oh Shit I
Just Forgot
Semicolon



ADC
END