Analog to Digital Converters

ADCs – AVR implementation

Digital Representation

- How do you represent a real number in a given number of bits
  - Quantization – mapping of codes to physical values
- Choosing quantization levels
  - Assume you want to represent 0V-5V given a 10-bit ADC
  - 0V = 0, 5V = 1023
  - LSB ~= 0.0488V
  - MSB ~= 2.5V

Analog to Digital Converters

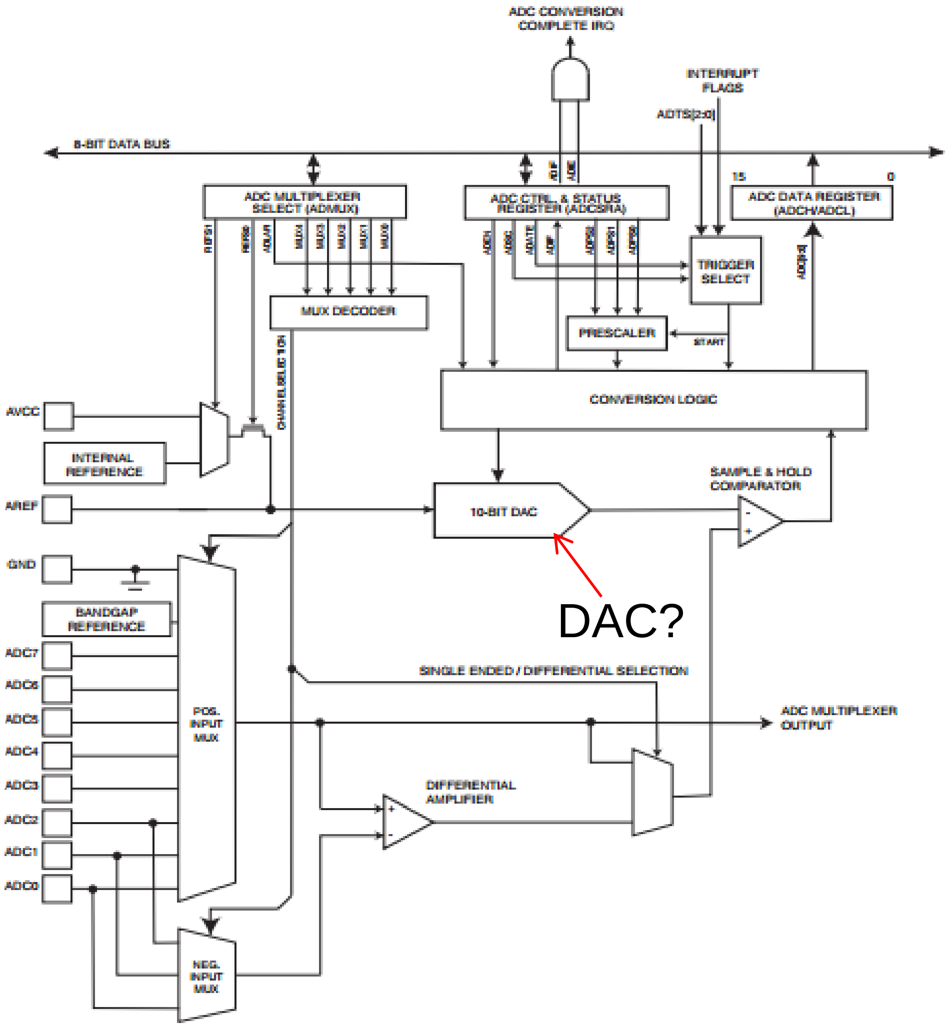
- Devices which convert a physical quantity (usually voltage) to a digital number
  - Abbreviated ADC, A/D, A to D
- Multiple kinds of architectures
  - Parallel/Serial stages
  - Single/Multiple conversion steps
  - One or multiple clock cycles
- Each architecture has tradeoffs
  - Power/size/speed/accuracy

AtMega169P ADC

22.1 Features

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ±2 LSB Absolute Accuracy
- 13 µs - 260 µs Conversion Time (50 kHz to 1 MHz ADC clock)
- Up to 15 ksps at Maximum Resolution (200 kHz ADC clock)
- Eight Multiplexed Single Ended Input Channels
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

Figure 22-1. Analog to Digital Converter Block Schematic

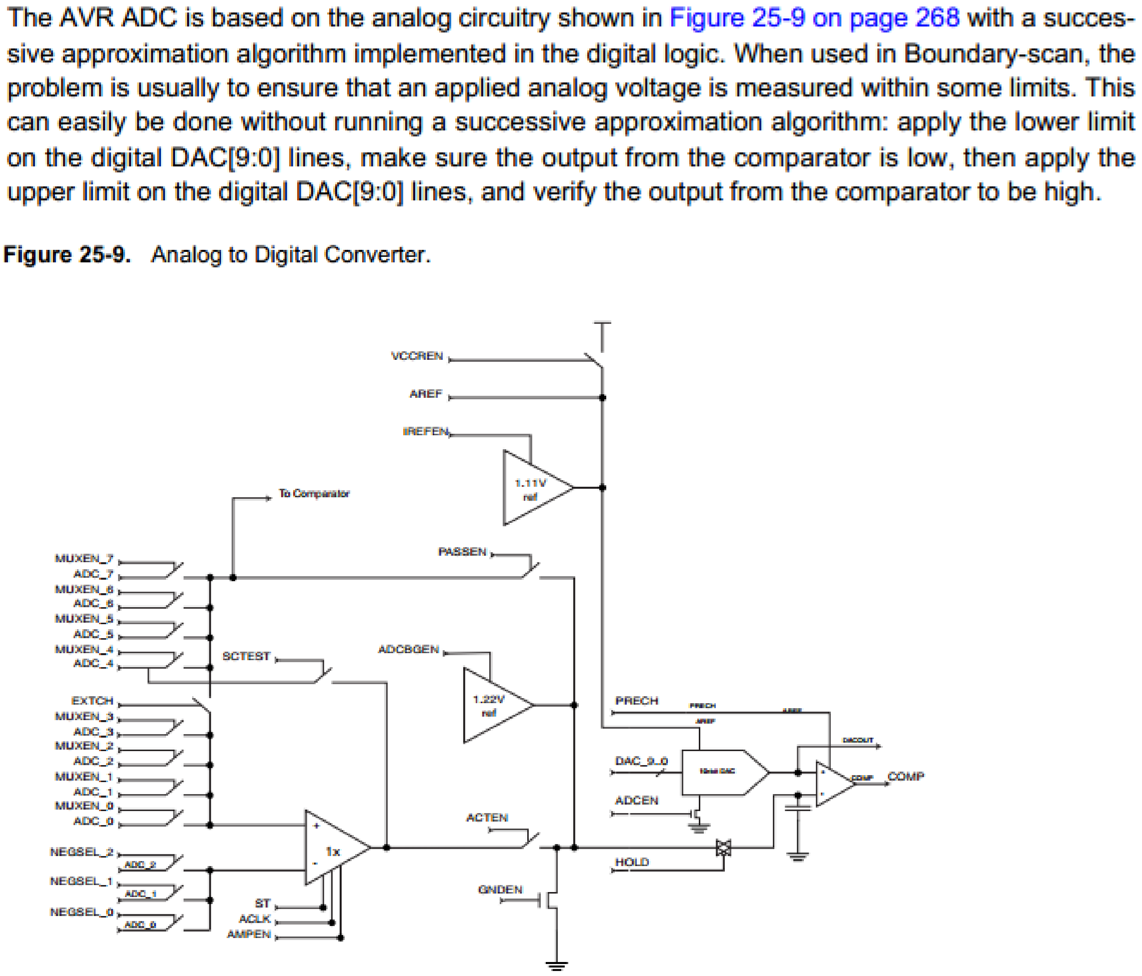


DAC?

Port F

AtMega169P ADC

- From the datasheet



The AVR ADC is based on the analog circuitry shown in Figure 25-9 on page 268 with a successive approximation algorithm implemented in the digital logic. When used in Boundary-scan, the problem is usually to ensure that an applied analog voltage is measured within some limits. This can easily be done without running a successive approximation algorithm: apply the lower limit on the digital DAC[9:0] lines, make sure the output from the comparator is low, then apply the upper limit on the digital DAC[9:0] lines, and verify the output from the comparator to be high.

Figure 25-9. Analog to Digital Converter.

Successive Approximation (Hardware)

- SHA
    - Sample and Hold
- DAC
    - Digital to Analog Converter

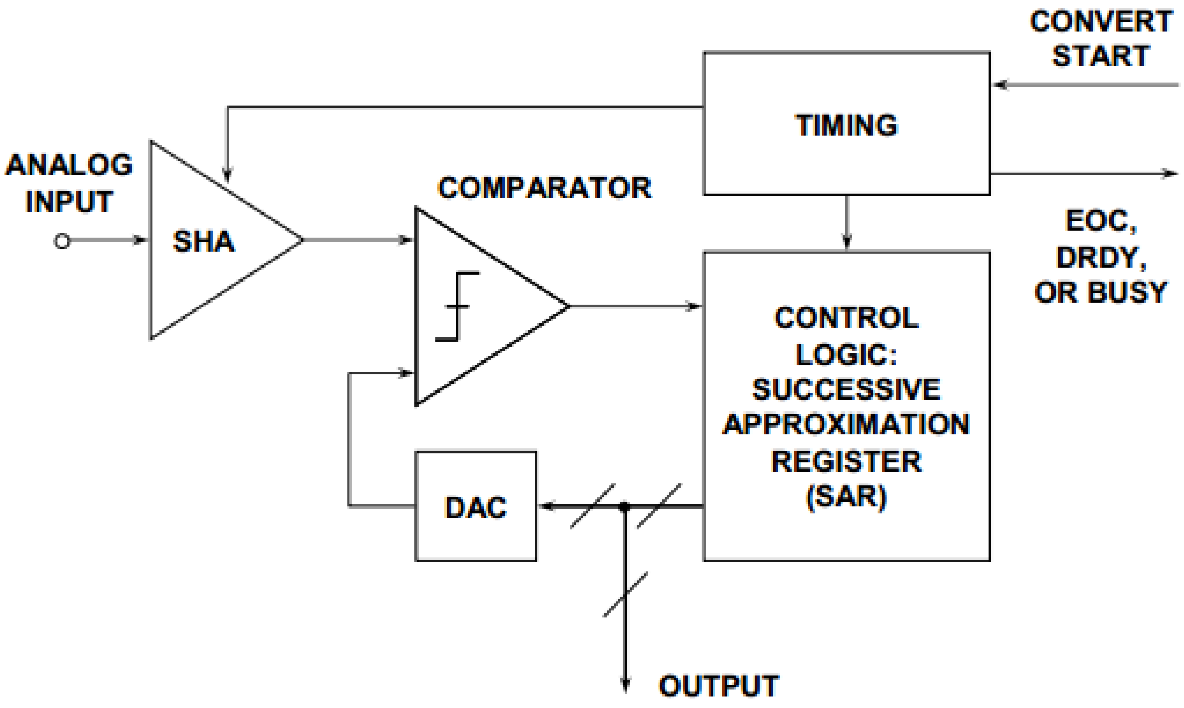
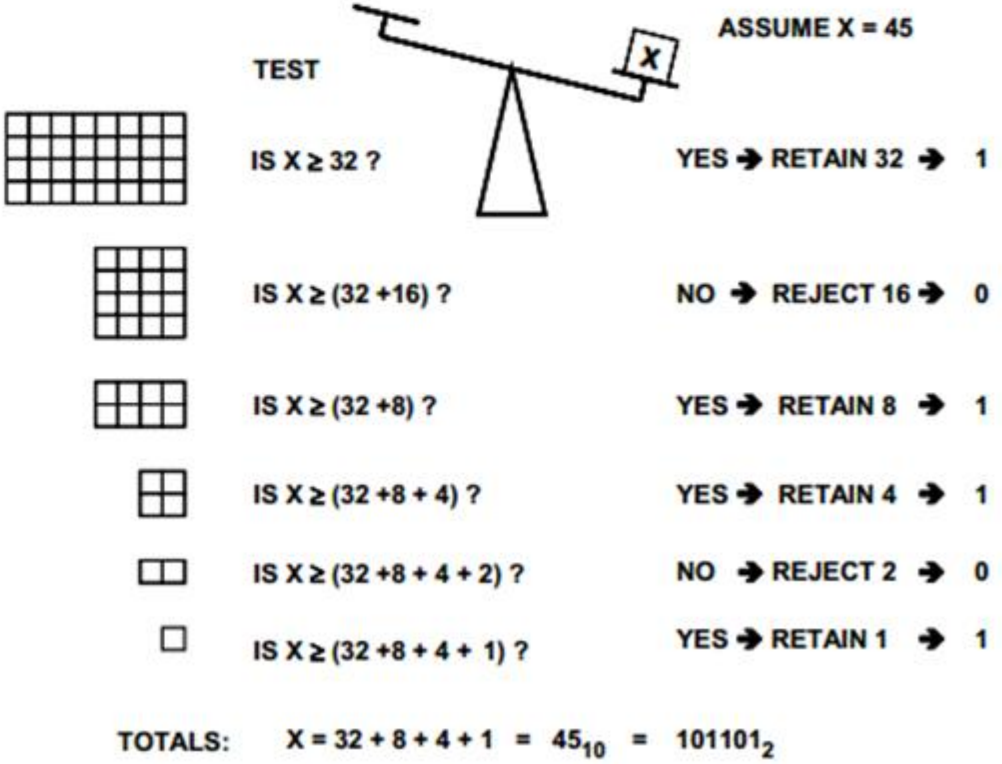


Figure 1: Basic Successive Approximation ADC (Feedback Subtraction ADC)

From Analog Devices ADC Architectures II document by Walt Kester

Successive Approximation (Implementation)

- Algorithm
  - Compare against half of range at each point
  - Continue narrowing until limited to LSB
  - Error upper bounded by 1 LSB (assuming value is just under next LSB, accurate comparator and DAC)



TEST		
IS X ≥ 32 ?	YES → RETAIN 32 →	1
IS X ≥ (32 +16) ?	NO → REJECT 16 →	0
IS X ≥ (32 +8) ?	YES → RETAIN 8 →	1
IS X ≥ (32 +8 + 4) ?	YES → RETAIN 4 →	1
IS X ≥ (32 +8 + 4 + 2) ?	NO → REJECT 2 →	0
IS X ≥ (32 +8 + 4 + 1) ?	YES → RETAIN 1 →	1

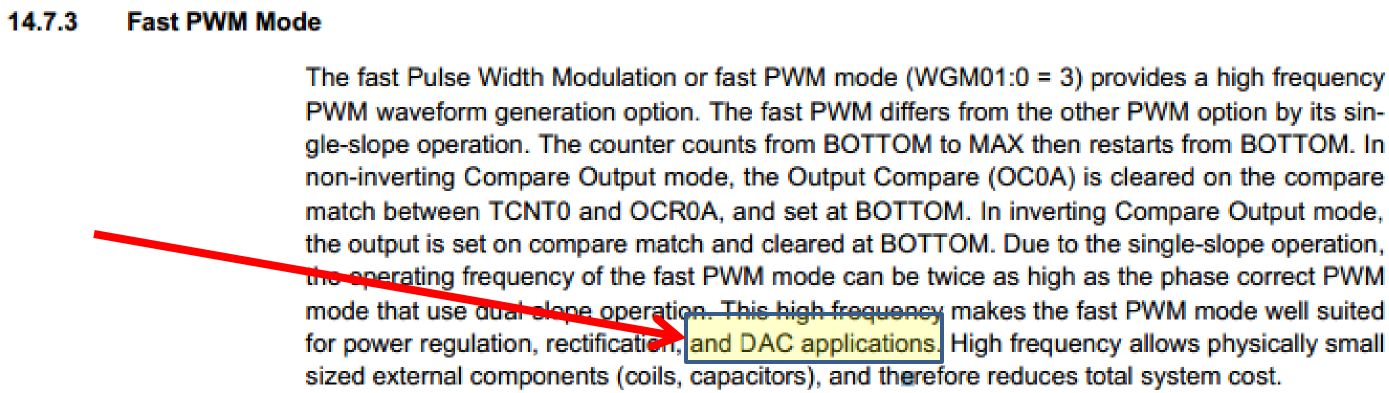
TOTALS: $X = 32 + 8 + 4 + 1 = 45_{10} = 101101_2$

Figure 4: Successive Approximation ADC Algorithm

From Analog Devices ADC Architectures II document by Walt Kester

Digital to Analog Converter

- Devices which convert a digital number (usually voltage) to a physical quantity
  - Abbreviated DAC, D/A, D to A
- Several Implementations
  - We will look at the Pulse Width Modulation (PWM) method



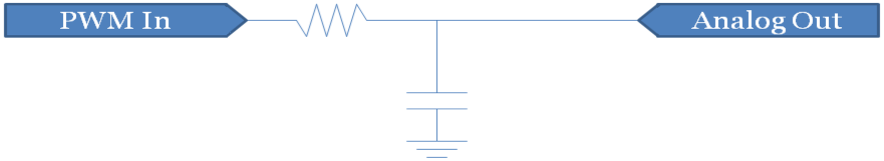
14.7.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM01:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0A) is cleared on the compare match between TCNT0 and OCR0A, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

PWM DAC Implementation

- Recall the discussion of the LED brightness using PWM
  - Higher duty cycle = bright
  - Lower duty cycle = dim
- Similar idea
  - Higher duty cycle = larger voltage
  - Lower duty cycle = smaller voltage
  - Apply PWM digital voltage across RC circuit to smooth waveform

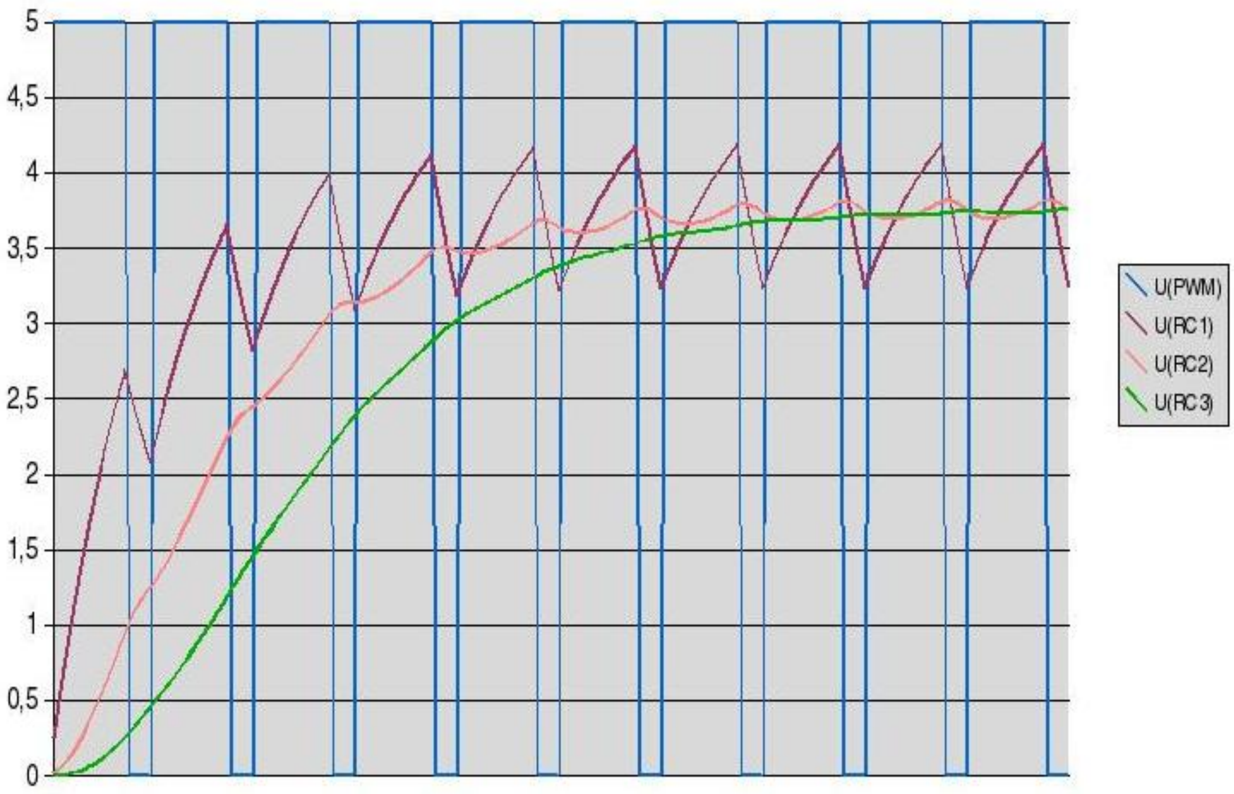
PWM DAC



PWM voltage is smoothed by RC circuit

Strength of RC circuit determines accuracy of mean voltage and time to mean voltage

Note that RC1 achieves mean voltage in about 3 PWM cycles where RC3 achieves it in 8



From: http://www.avr-asm-tutorial.net/avr_en/AVR_ADC500.html

Successive Approximation Timing

- How many clock cycles would the AVR 10-bit hardware take to convert with Successive Approximation?
    - Assume that the DAC can achieve accurate results within 1 clock cycle

- Binary Search Tree
    - Therefore, an N-bit conversion takes N-steps
    - 10 clock cycles for our example

Successive Approximation Timing

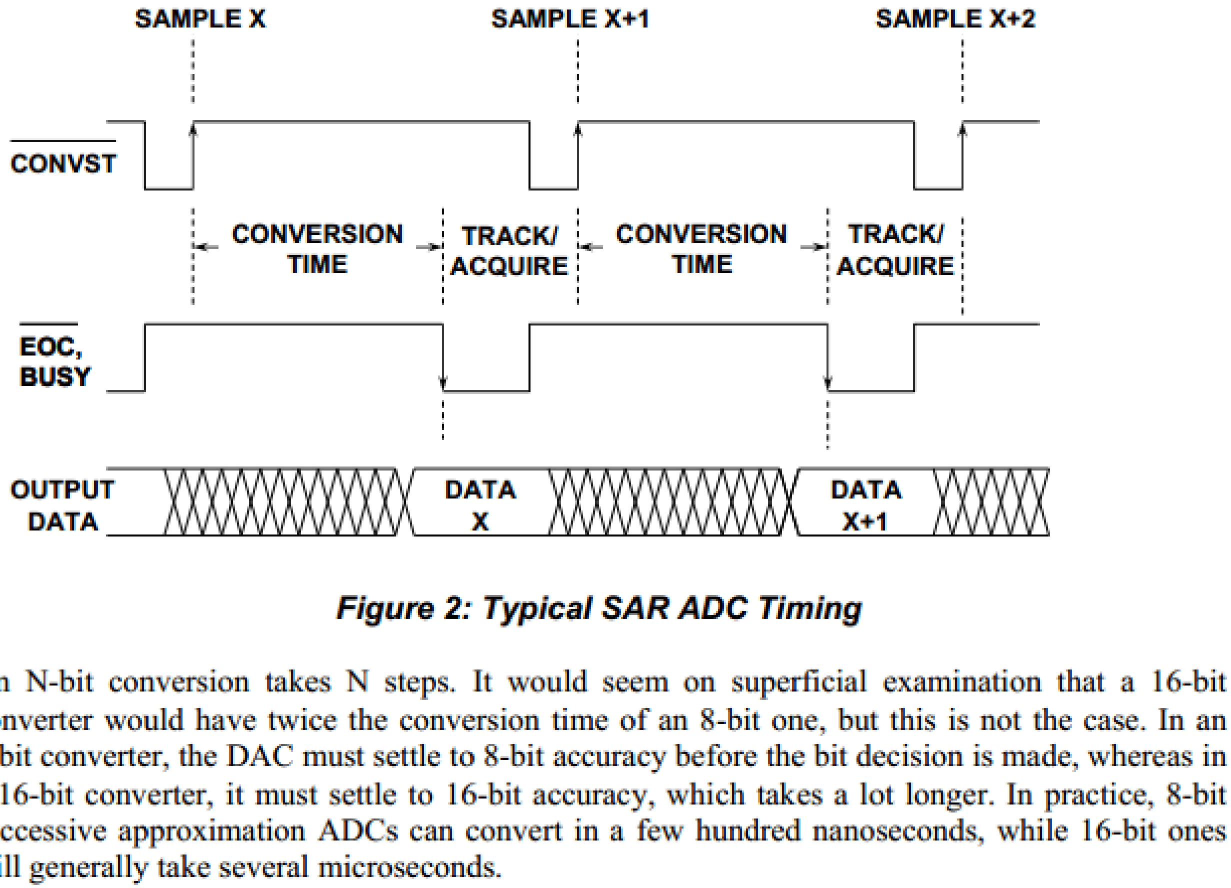


Figure 2: Typical SAR ADC Timing

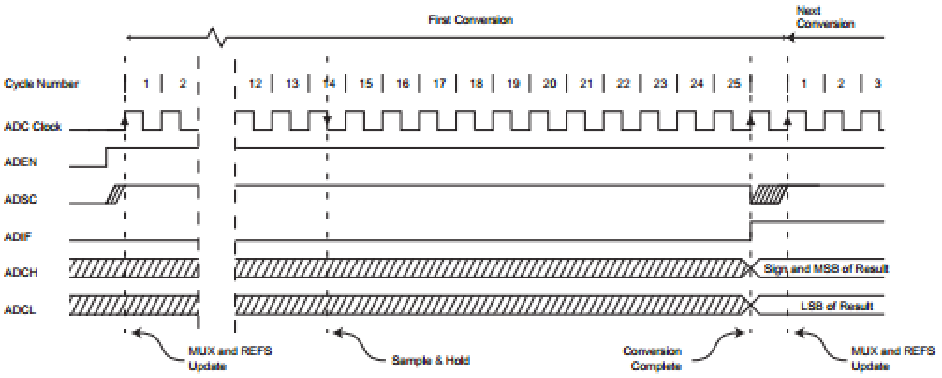
An N-bit conversion takes N steps. It would seem on superficial examination that a 16-bit converter would have twice the conversion time of an 8-bit one, but this is not the case. In an 8-bit converter, the DAC must settle to 8-bit accuracy before the bit decision is made, whereas in a 16-bit converter, it must settle to 16-bit accuracy, which takes a lot longer. In practice, 8-bit successive approximation ADCs can convert in a few hundred nanoseconds, while 16-bit ones will generally take several microseconds.

From Analog Devices ADC Architectures II document by Walt Kester

AVR Actual Timing

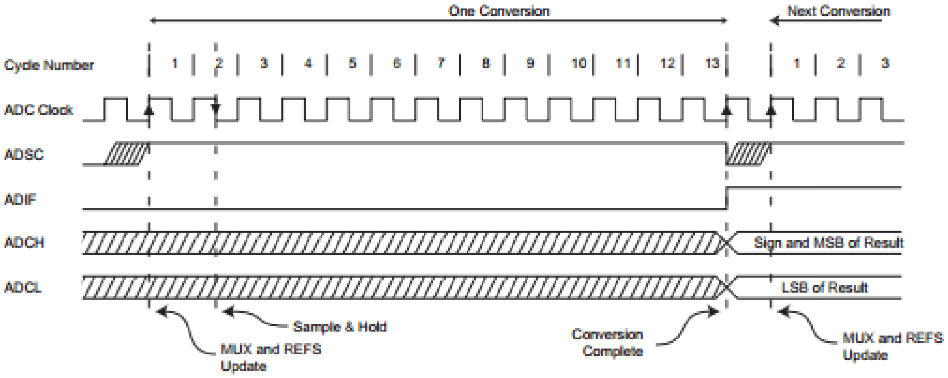
The first conversion (after enabling ADC) takes 25 ADC clock cycles in order to initialize the analog circuitry

Figure 22-4. ADC Timing Diagram, First Conversion (Single Conversion Mode)

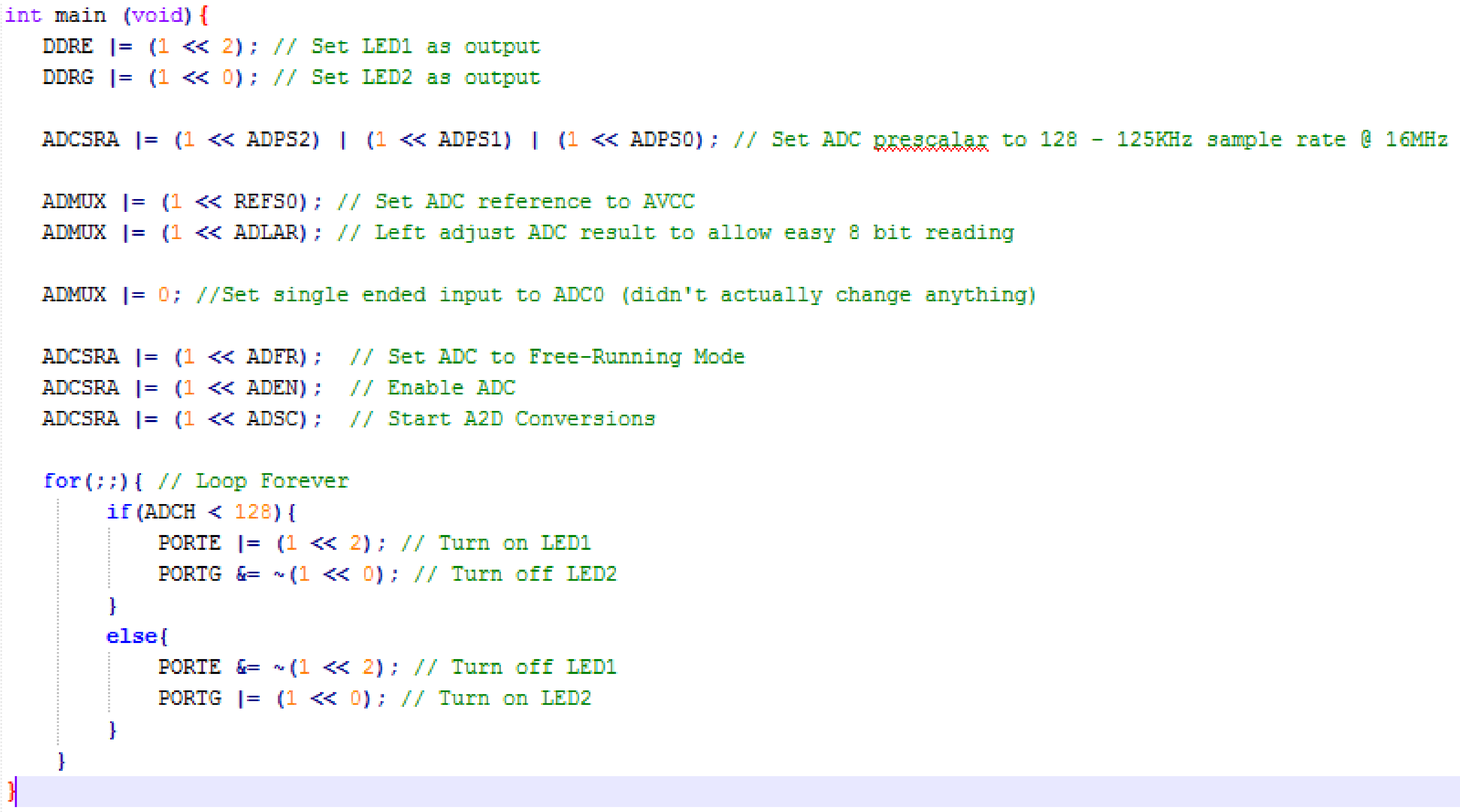


All subsequent conversions take 13 ADC clock cycles. Including sample and hold and conversion complete

Figure 22-5. ADC Timing Diagram, Single Conversion



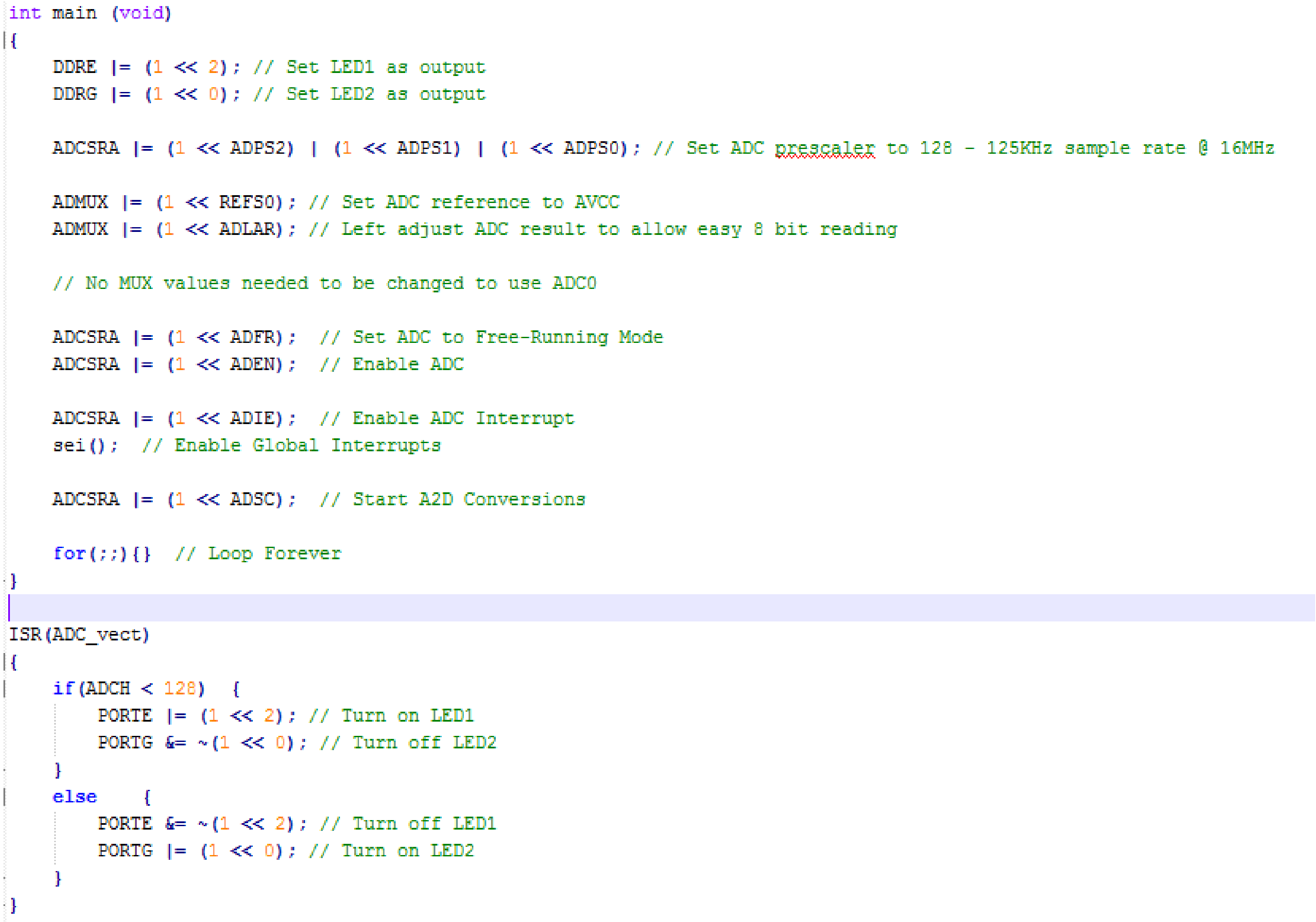
Using the AVR ADC - Polling



```
int main (void){
   DDRE |= (1 << 2); // Set LED1 as output
   DDRG |= (1 << 0); // Set LED2 as output

   ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Set ADC prescalar to 128 - 125KHz sample rate @ 16MHz

   ADMUX |= (1 << REFS0); // Set ADC reference to AVCC
   ADMUX |= (1 << ADLAR); // Left adjust ADC result to allow easy 8 bit reading

   ADMUX |= 0; //Set single ended input to ADC0 (didn't actually change anything)

   ADCSRA |= (1 << ADFR);  // Set ADC to Free-Running Mode
   ADCSRA |= (1 << ADEN);  // Enable ADC
   ADCSRA |= (1 << ADSC);  // Start A2D Conversions

   for(;;){ // Loop Forever
        if(ADCH < 128){
            PORTE |= (1 << 2); // Turn on LED1
            PORTG &= ~(1 << 0); // Turn off LED2
        }
        else{
            PORTE &= ~(1 << 2); // Turn off LED1
            PORTG |= (1 << 0); // Turn on LED2
        }
    }
}
```

From http://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-adc

Using the AVR ADC - Interrupt



```
int main (void)
{
    DDRE |= (1 << 2); // Set LED1 as output
    DDRG |= (1 << 0); // Set LED2 as output

    ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Set ADC prescaler to 128 - 125KHz sample rate @ 16MHz

    ADMUX |= (1 << REFS0); // Set ADC reference to AVCC
    ADMUX |= (1 << ADLAR); // Left adjust ADC result to allow easy 8 bit reading

    // No MUX values needed to be changed to use ADC0

    ADCSRA |= (1 << ADFR);  // Set ADC to Free-Running Mode
    ADCSRA |= (1 << ADEN);  // Enable ADC

    ADCSRA |= (1 << ADIE);  // Enable ADC Interrupt
    sei();  // Enable Global Interrupts

    ADCSRA |= (1 << ADSC);  // Start A2D Conversions

    for(;;){}  // Loop Forever
}

ISR(ADC_vect)
{
    if(ADCH < 128)  {
        PORTE |= (1 << 2); // Turn on LED1
        PORTG &= ~(1 << 0); // Turn off LED2
    }
    else    {
        PORTE &= ~(1 << 2); // Turn off LED1
        PORTG |= (1 << 0); // Turn on LED2
    }
}
```

From http://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-adc

Datasheet Reading Example

- Find what pins connect to the ADC
- Find what control registers need to be modified
- Learn about operation of ADC
- Learn what ADC values mean