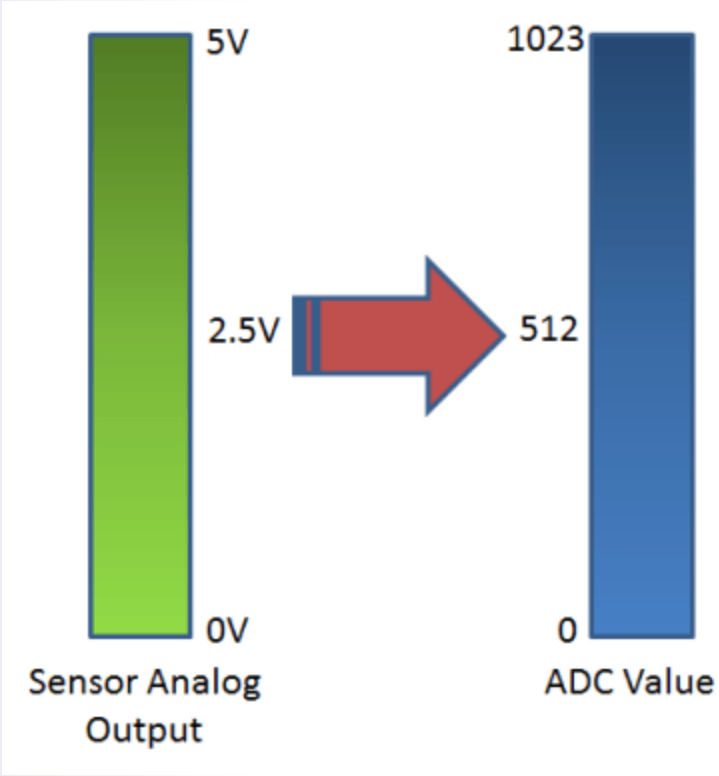
ADC in AVR Micro-Controller

- Required for Homework-4
- Also Refer Chapter 20 and 21 from Data Sheet



ATMEGA16/32

- 8 channels » 8 pins
- 10 bit resolution
- $2^{10} = 1024$ steps

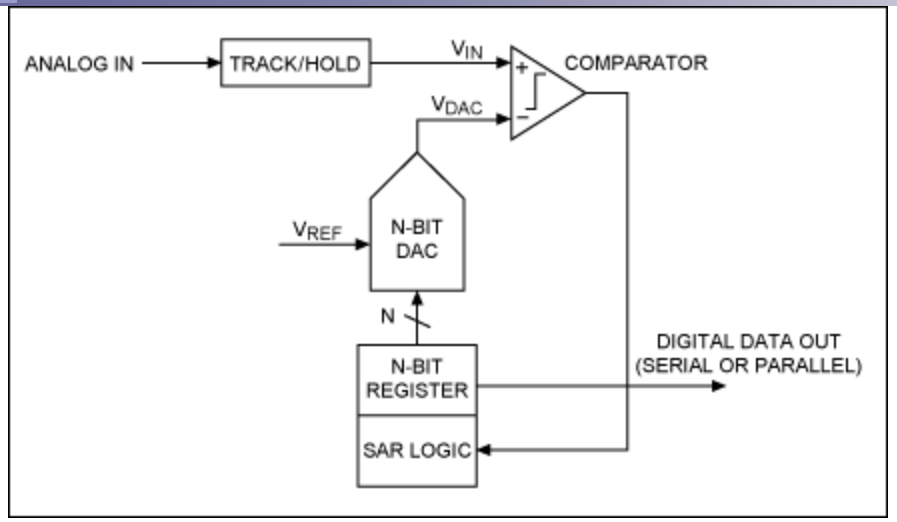
ADC in AVR Micro-Controller

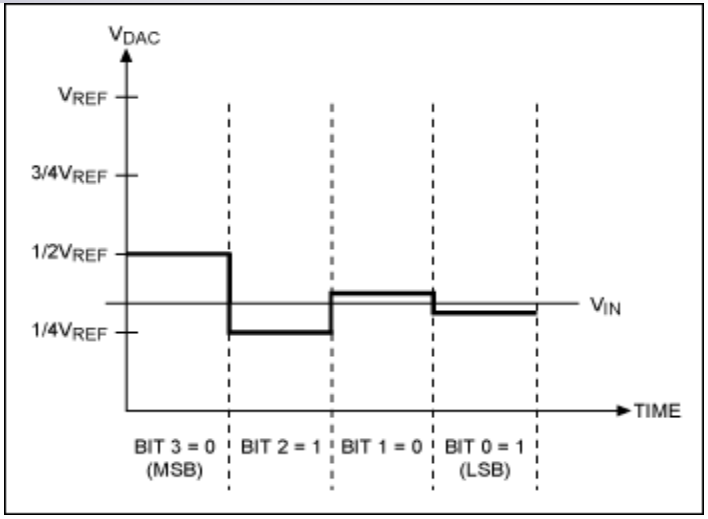
- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ±2 LSB Absolute Accuracy
- 13 µs - 260 µs Conversion Time (50 kHz to 1 MHz ADC clock)
- Up to 15 ksps at Maximum Resolution (200 kHz ADC clock)
- Eight Multiplexed Single Ended Input Channels

ADC in AVR Micro-Controller

- Optional Left Adjustment for ADC Result Readout
- 0 - VCC 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

Basic Successive Approximation



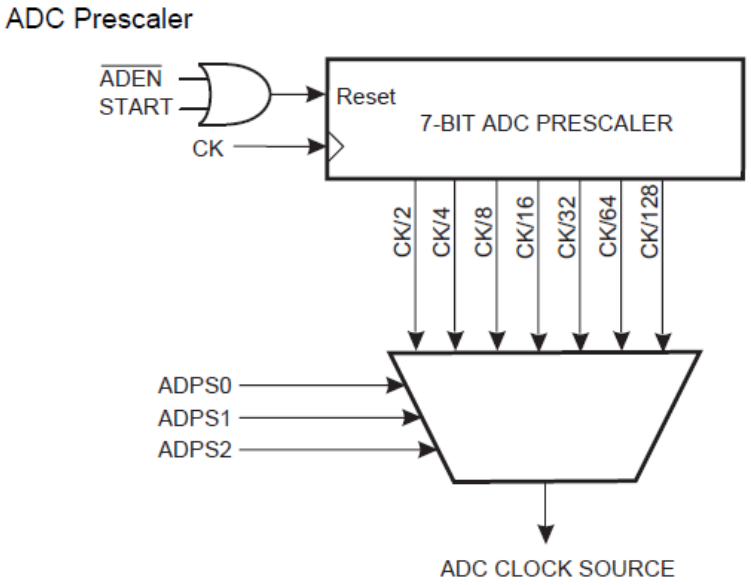


- The ATmega169P features a 10-bit successive approximation ADC
- The minimum value represents GND and the maximum value represents the voltage on the AREF pin (Pin #62) minus 1 LSB.
- Analog inputs are evaluated as a portion or ratio of a (doubled) reference single and converted to an integer scale ADC value from 0 to 2^{10}-1

ADC Operation in ATMega169p

- The analog input channel is selected by writing to the MUX bits in ADMUX.
- The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA
  - Voltage reference and input channel selections will not go into effect until ADEN is set
- The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL
  - ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion

ADC Prescaling



- ADC requires an input clock frequency between 50kHz and 200kHz 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 Registers

- To Access and Control ADC on ATMega Micro-Controller you need to focus on following Registers:
  1. ADMUX – ADC Multiplexer Selection Register
  2. ADCSRA – ADC Control and Status Register A
  3. ADCL and ADCH – ADC Data Registers
  4. SFIOR – Special Function I/O Register

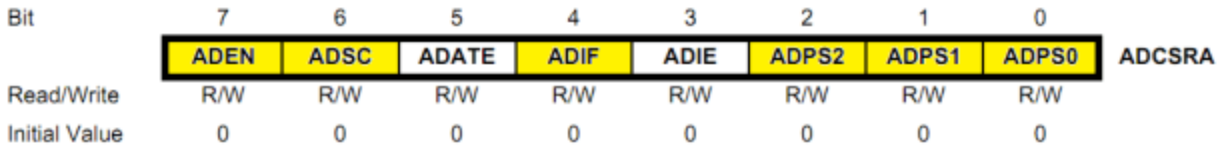
ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bits 7:6 – REFS1:0 – Reference Selection Bits
  - These bits are used to choose the reference voltage. The following combinations are used.
- Bit 5 – ADLAR – ADC Left Adjust Result
  - Make it ‘1’ to Left Adjust the ADC Result.
- Bits 4:0 – MUX4:0 – Analog Channel and Gain Selection Bits
- To initialize ADMUX, we write

```
ADMUX = (1<<REFS0);
```

ADCSRA – ADC Control and Status Register A



Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN – ADC Enable**
  - Enables the ADC feature
- **Bit 6 – ADSC – ADC Start Conversion**
  - Write this to ‘1’ before starting any conversion. This 1 is written as long as the conversion is in progress, after which it returns to zero.
- **Bit 5 – ADATE – ADC Auto Trigger Enable**
  - Setting it to ‘1’ enables auto-triggering of ADC. ADC is triggered automatically at every rising edge of clock pulse.

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – ADIF – ADC Interrupt Flag**
  - Whenever a conversion is finished and the registers are updated, this bit is set to ‘1’ automatically.
- **Bit 3 – ADIE – ADC Interrupt Enable**
  - When this bit is set to ‘1’, the ADC interrupt is enabled. This is used in the case of interrupt-driven ADC.
- **Bits 2:0 – ADPS2:0 – ADC Prescaler Select Bits**
  - The prescaler (division factor between XTAL frequency and the ADC clock frequency) is determined
- To initialize ADCSRA as follows:

```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); // prescaler = 128
```

ADCL and ADCH – ADC Data Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

- Since the ADC has a resolution of 10 bits, it requires 10 bits to store the result. Thus, ADCL and ADCH (ADC Low byte and ADC High byte) as follows.
- Upon setting ADLAR = 1, the conversion result is left adjusted.

SFIOR – Special Function I/O Register

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- In normal operation, we do not use this register. This register comes into play whenever ADATE (in ADCSRA) is set to ‘1’.
- **Bits 7:5 – ADC Auto Trigger Source**
  - Whenever ADATE is set to ‘1’, these bits determine the trigger source for ADC conversion. There are 8 possible trigger sources.

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADC Initialization and Reading Values

ADC Initialization

```
void adc_init()
{
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

ADC Reading Values

```
uint16_t adc_read(uint8_t ch)
{
  // select the corresponding channel 0~7
  // ANDing with '7' will always keep the value
  // of 'ch' between 0 and 7
  ch &= 0b00000111;  // AND operation with 7
  ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

  // start single convertion
  // write '1' to ADSC
  ADCSRA |= (1<<ADSC);

  // wait for conversion to complete
  // ADSC becomes '0' again
  // till then, run loop continuously
  while(ADCSRA & (1<<ADSC));
  return (ADC);
}
```