ECE3411 – Spring 2019

Lecture 5

Analog Digital Converter

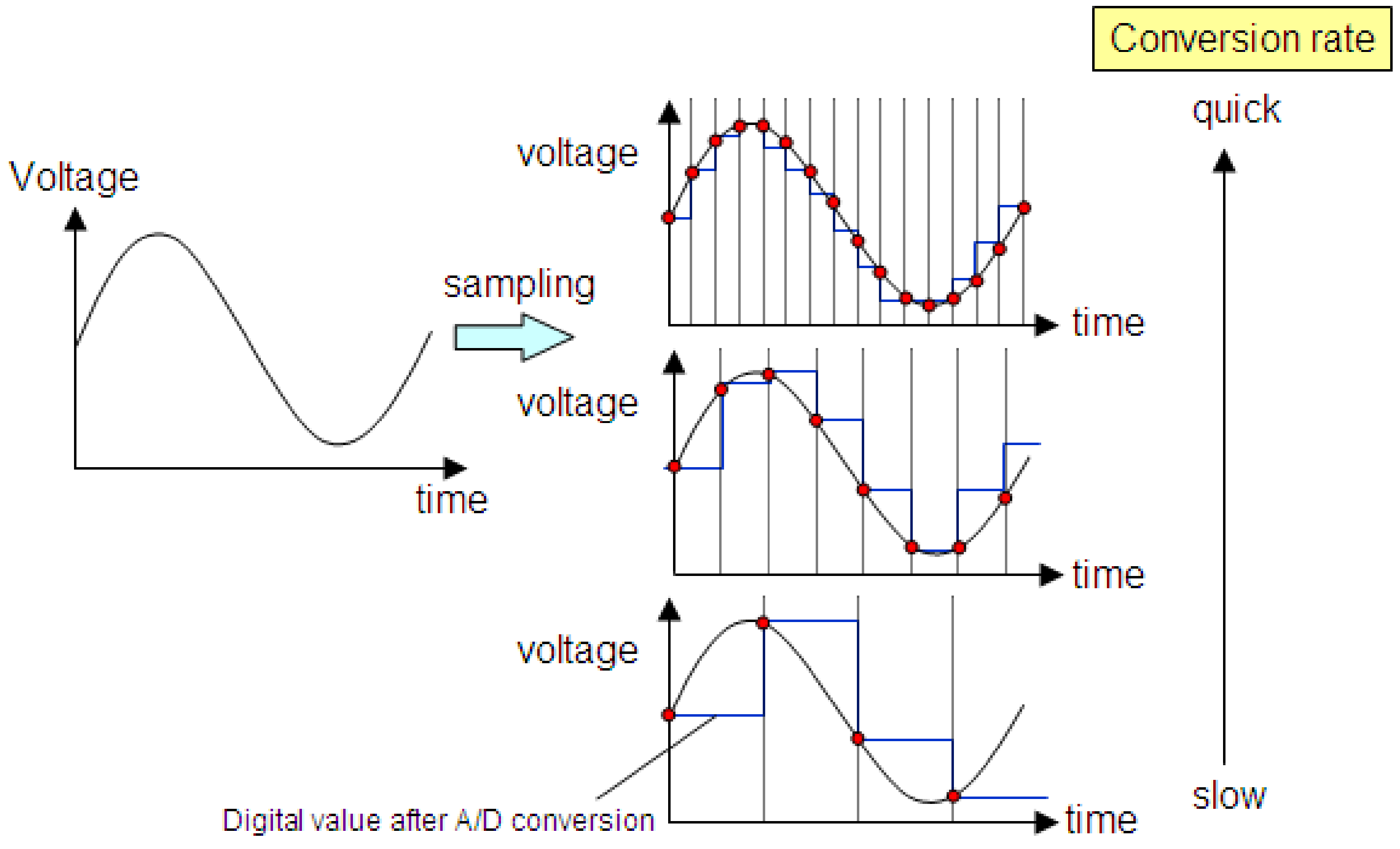
Sung Yeul Park
Department of Electrical & Computer Engineering
University of Connecticut
Email: sung_yeul.park@uconn.edu

Copied from Lecture 2a, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

Based on the Atmega328P datasheet and material
from Bruce Land's video lectures at Cornel

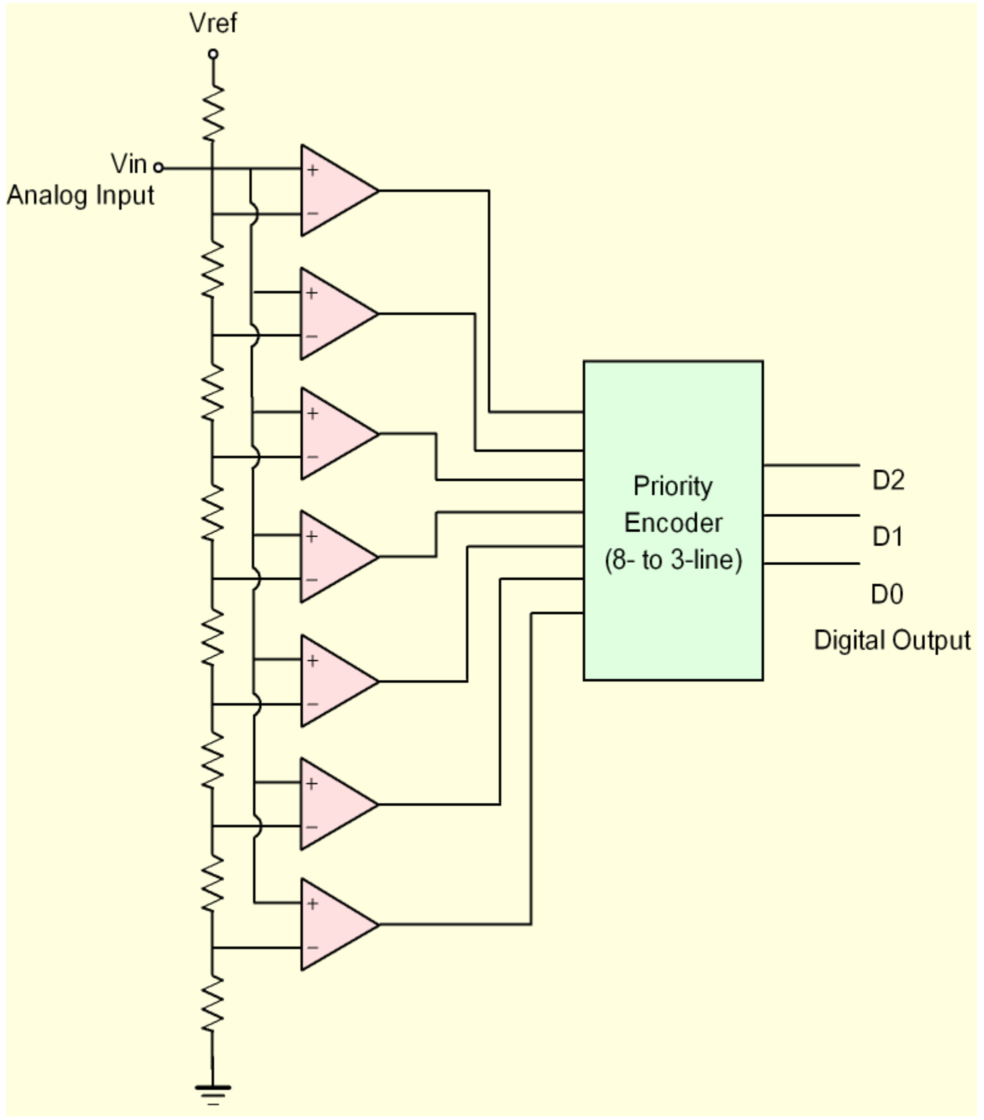
Introduction

- Why do we need Analog-Digital Conversion?
  - Real world is Analog
  - Digital computers process Digital signals
  - ADC/DAC serve as interface between Computers and Real world!
- Analog Signals are "Continuous"
  - A "Discrete" version of the analog signal is created by "Sampling" the analog signal
  - ADC then maps each sample onto a quantized range of voltages which can be represented by binary values.



ADC Types: Flash ADC

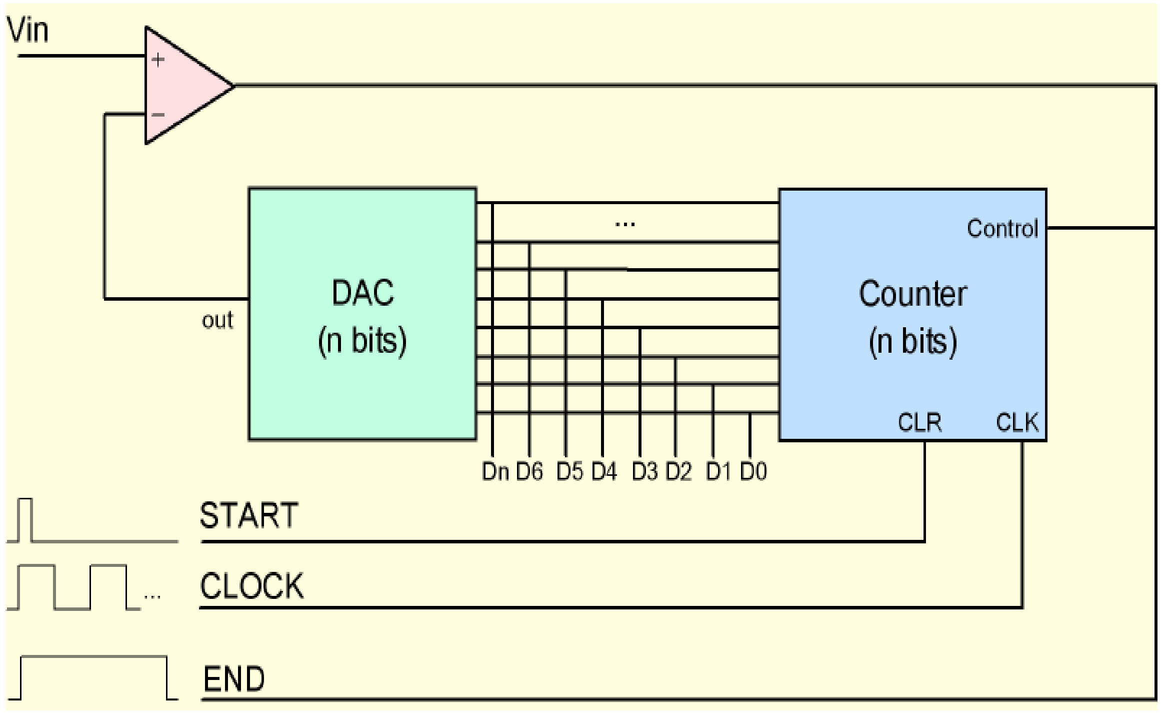
- Parallel Design
  - A resistor divider network generates discrete voltage levels
  - Input voltage is compared against all the voltage levels at once
  - Priority Encoder considers the first "HIGH" input from the top as valid, and converts it to binary form.
- Advantage: Fast
  - Conversion takes just one cycle
- Disadvantage: A lot of components needed.
  - $2^n - 1$ comparators needed for n bit ADC



Picture Source: www.hardwaresecrets.com

ADC Types: Ramp ADC

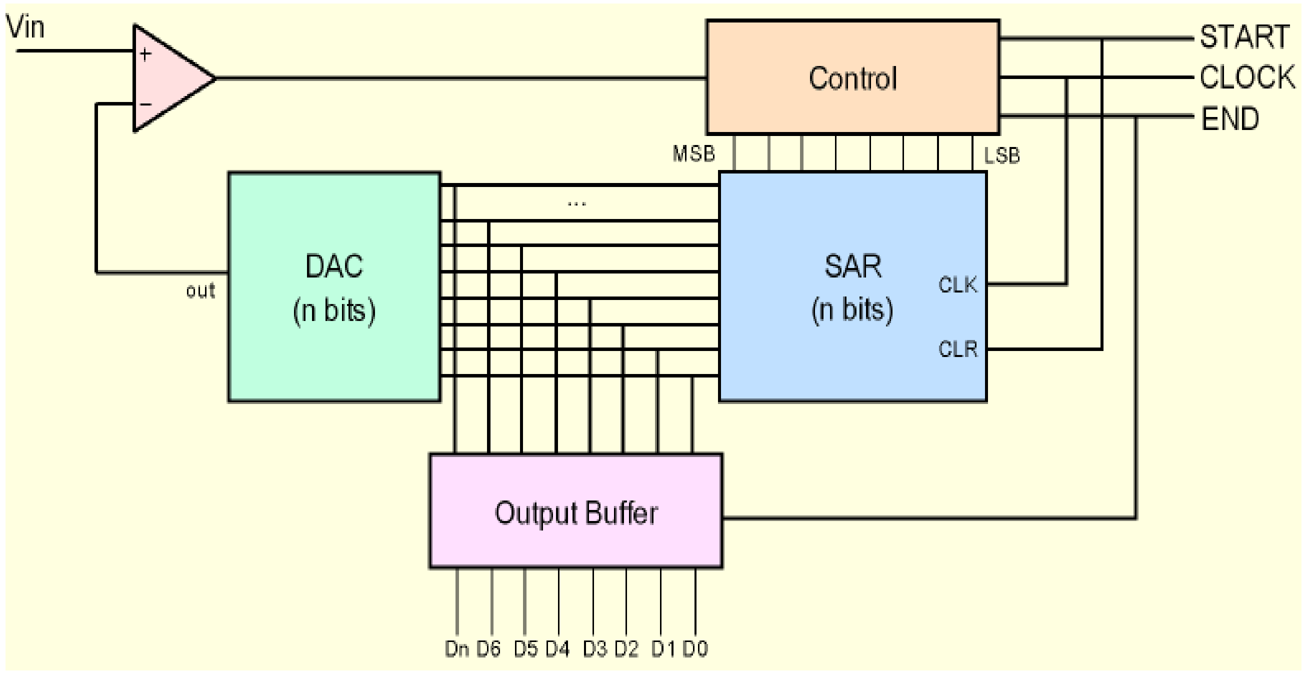
- Sequential Design
  - A Counter counts from $0 \cdots 2^n$
  - A DAC generates discrete voltage levels corresponding to the digital values $0 \cdots 2^n$ (i.e. a voltage Ramp)
  - In each cycle, input voltage is compared against the current voltage level generated by DAC
  - The comparator generates a "HIGH" value as soon as the ramp crosses the input value. The corresponding counter value becomes the output.
- Advantage: Only a few components needed.
- Disadvantage: Very slow.
  - $2^n - 1$ cycles (in worst case) for n bit ADC conversion



Picture Source: www.hardwaresecrets.com

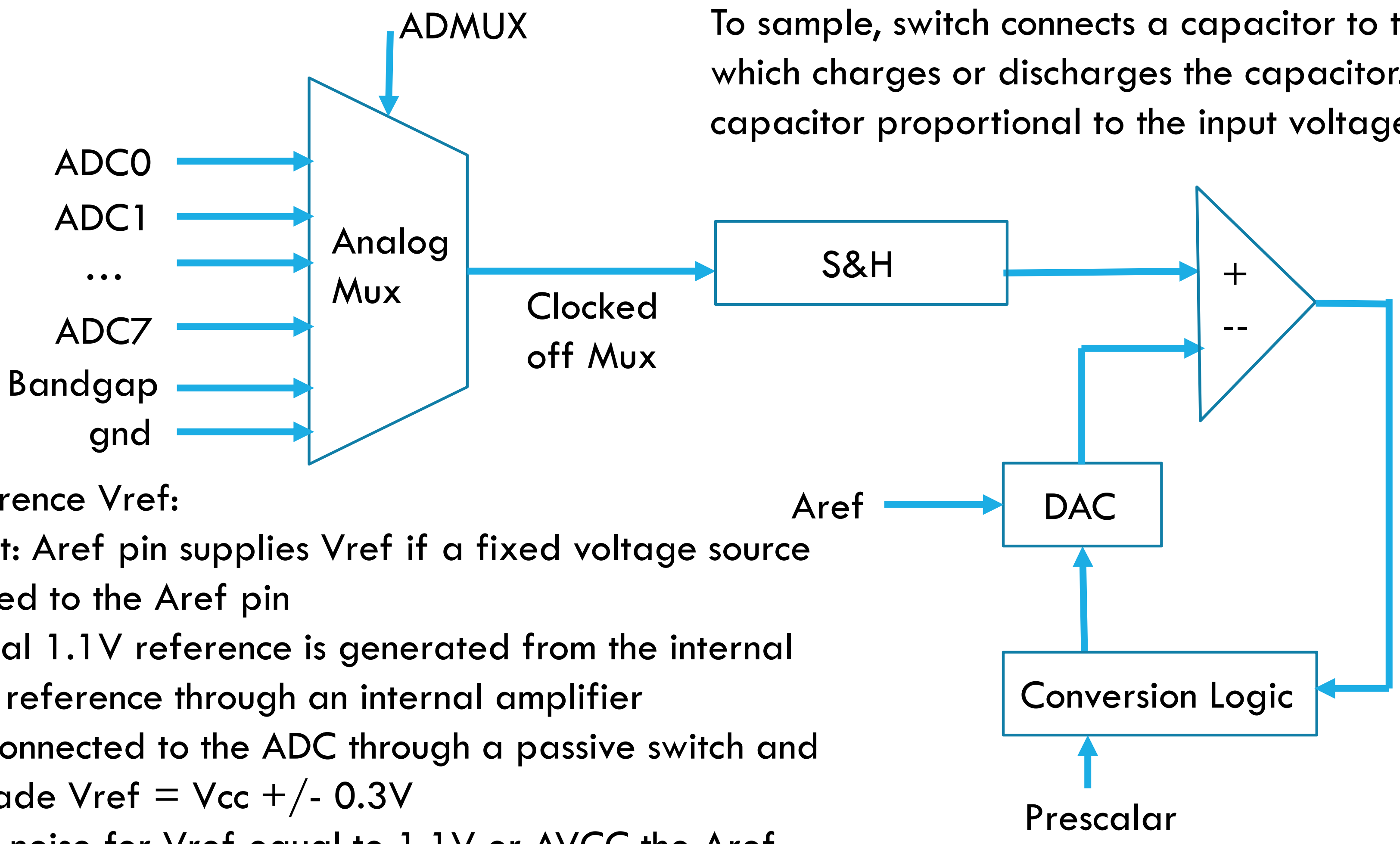
ADC Types: Successive Approximation ADC

- Sequential Design
  - Closest digital value is approximated by “Binary Search”
  - First, the MSB of SAR is set to 1, and the comparator decides whether the input voltage is higher or lower than DAC voltage. The bit value is adjusted accordingly.
  - The process is repeated for each bit from MSB down to LSB
  - The final SAR value becomes the output.
- Most widely used ADC type.
- Advantages:
  - Only a few components needed.
  - Conversion takes just n cycles.



Picture Source: www.hardwaresecrets.com

ATMega328P ADC Diagram



To sample, switch connects a capacitor to the output of a buffer amplifier, which charges or discharges the capacitor. This makes voltage across the capacitor proportional to the input voltage. To hold, the switch disconnects.

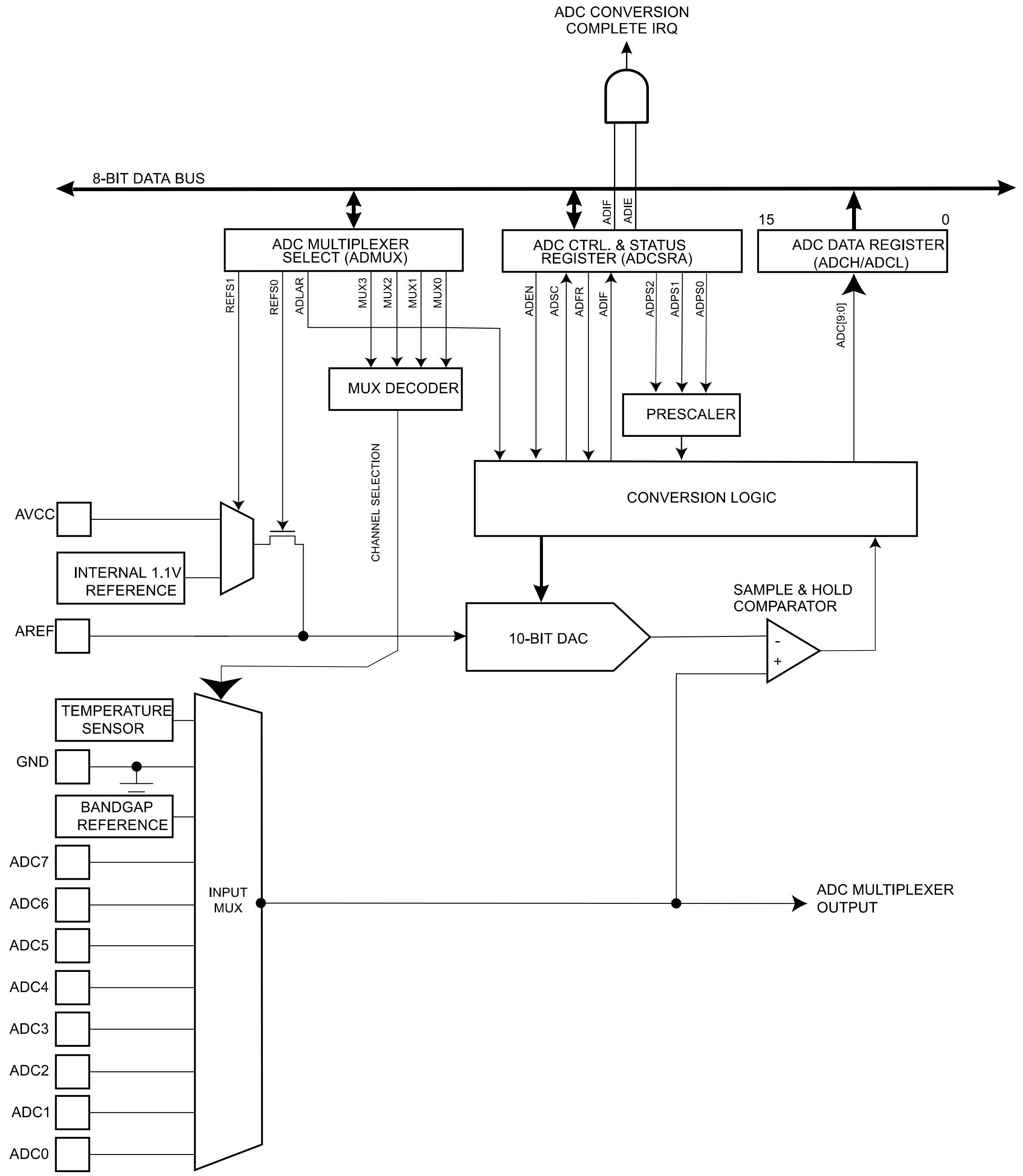
Conversion logic implements a successive approximation algorithm (a binary search; one bit per search):

- DAC takes as input the output of the conversion logic and converts it to an analog voltage where Aref sets the full range
- Analog comparator decides whether the DAC output or input voltage is the largest

Voltage reference Vref:

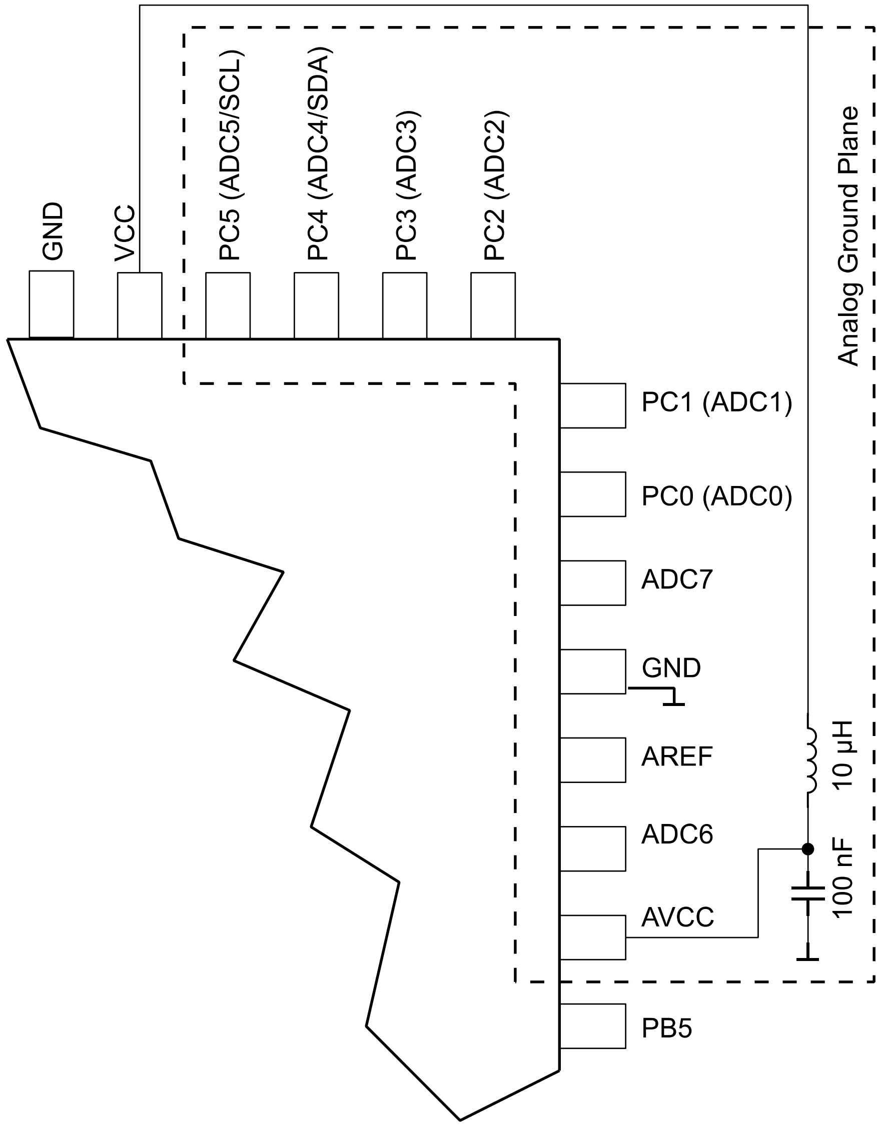
- By default: Aref pin supplies Vref if a fixed voltage source is connected to the Aref pin
- The internal 1.1V reference is generated from the internal bandgap reference through an internal amplifier
- AVCC is connected to the ADC through a passive switch and can be made Vref = Vcc +/- 0.3V
- To reduce noise for Vref equal to 1.1V or AVCC the Aref pin can be externally decoupled by a capacitor to ground

Figure 28-1. Analog-to-Digital Converter Block Schematic Operation



Pin Assignment

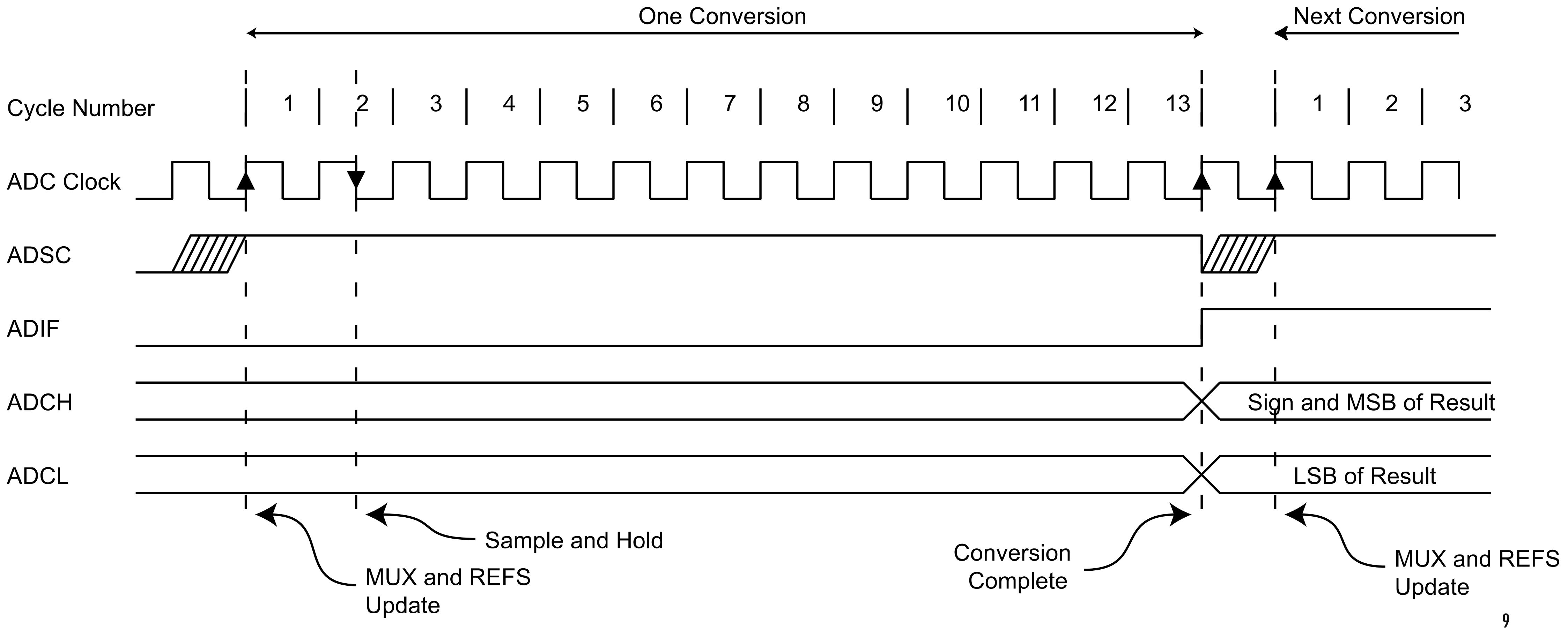
Figure 28-9. ADC Power Connections



Normal Conversion

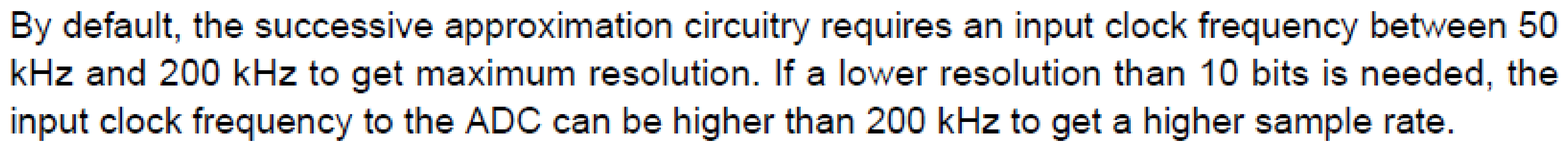
- Takes 13 cycles

Figure 28-5. ADC Timing Diagram, Single Conversion



Accuracy

- Capacitor in S&H leaks and can therefore not hold a value for too long
  - There exists a minimum sample speed/frequency
- Conversion logic takes time, so we cannot sample too fast
  - There exists a maximum sample speed/frequency
  - The faster you sample, you get a smaller number of accurate output bits (since the binary search cannot completely finish)


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.

- Noise: MCU produces up to 150mV line noise, there are other sources such as electrical field, etc.
  - Use capacitances close to the CPU to eliminate most of the inductance

Prescaler

Table 28-5. Input Channel Selection

ADPS[2:0]	Division Factor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

- E.g., a prescaler of 128 gives 16MHz/128 = 125000 (between 50 and 200 kHz)
- To complete the binary search takes 13 cycles = 13/125000 = 104 micro seconds
- Gives 10 bits uncalibrated accuracy at a linear scale to Vref

- CPU clock is at least twice as fast as the ADC's acceptable frequency; therefore the smallest prescaler must be >=2

ADMUX Register

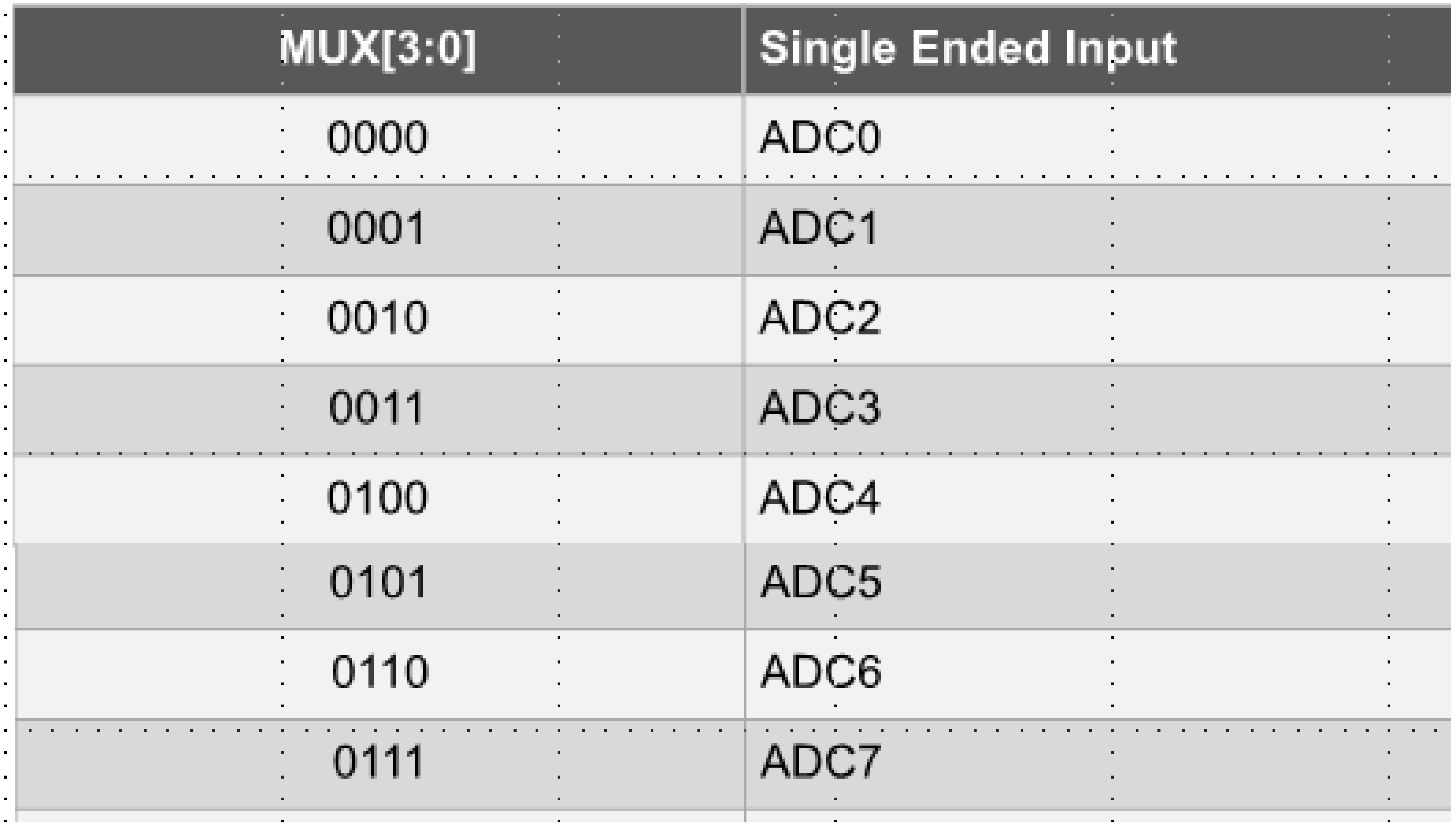
ADC Multiplexer Selection Register

Name:	ADMUX
Offset:	0x7C
Reset:	0x00
Property:	-

Bit	7	6	5	4	3	2	1	0
	REFS [1:0]		ADLAR		MUX [3:0]			
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned OFF
01	AV_{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V voltage reference with external capacitor at AREF pin

ADMUX Register



MUX[3:0]	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7

ADCH/ADCL: ADC Data Registers

ADC Data Register Low and High Byte (ADLAR=0)

Name:	ADCL and ADCH
Offset:	0x78
Reset:	0x00
Property:	ADLAR = 0

If ADLAR is set to 0,

- read ADCL for low order bits, and
- until ADCH is read the ADC is locked out

For 8-bit conversion, set ADLAR to 1 and read ADCH

Bit	15	14	13	12	11	10	9	8
							ADC[9:8]	
Access							R	R
Reset							0	0

Bit	7	6	5	4	3	2	1	0
	ADC[7:0]							
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

ADCSRA: ADC Status Register A

ADC Control and Status Register A

Name:	ADCSRA
Offset:	0x7A
Reset:	0x00
Property:	-

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS [2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

- Bit 7: ADEN – analog converter enable bit; set this bit to 1 if you want to do a conversion
- Bit 6 ADSC – AD start conversion; if it is set to 1, then a conversion is started for you and it is auto set back to 0 when done
  - You can poll this bit and as soon as it is 0, you know the conversion is done
  - Or you can poll the interrupt flag (or use the corresponding ISR if enabled):
- Bit 4: ADIF – AD interrupt flag; will be set when a conversion is done and will trigger an interrupt if ADIE is set
  - Warning: do not mess with this flag, e.g., use ADCSRA |= (1<<ADSC);

ADCSRA: ADC Status Register A

ADC Control and Status Register A

Name: ADCSRA
Offset: 0x7A
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS [2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

- Bit 3: ADIE – AD interrupt enable; if turned on, write the ISR to handle what happens when conversion finishes
- Bit 5: ADATE – allows one out of 8 selected events to trigger the ADC converter when coupled with the ADCSRB register
- Bits 0,1,2: prescaler (see earlier slide)

ADCSRB

ADC Control and Status Register B

Name:	ADCSRB
Offset:	0x7B
Reset:	0x00
Property:	-

Bit	7	6	5	4	3	2	1	0
		ACME				ADTS [2:0]		
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

ADTS[2:0]	Trigger Source
000	Free Running mode
001	Analog Comparator
010	External Interrupt Request 0
011	Timer/Counter0 Compare Match A
100	Timer/Counter0 Overflow
101	Timer/Counter1 Compare Match B
110	Timer/Counter1 Overflow
111	Timer/Counter1 Capture Event

Example code ADC, no interrupt

```c
// Borrowed from Bruce Land - Cornell University

// Performs single, left adjusted conversions and prints to UART

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
#include <math.h>
#include "uart.h"

volatile int Ain, AinLow;
volatile float Voltage;
char VoltageBuffer[6];

FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);
```

Example code ADC, no interrupt

```
void main(void)
{
        DDRC &= 0x00;      // PC1 = ADC1 is set as input

        uart_init();
        stdout = stdin = stderr = &uart_str;

        // ADLAR set to 1 → left adjusted result in ADCH
        // MUX3:0 set to 0001 → input voltage at ADC1
        ADMUX = (1<<MUX0) | (1<<ADLAR);

        // ADEN set to 1 → enables the ADC circuitry
        // ADPS2:0 set to 111 → prescalar set to 128 (104us per conversion)
        ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);

        // Start A to D conversion
        ADCSRA |= (1<<ADSC);
        fprintf(stdout,"\n\rStarting ADC demo...\n\r");
```

Takes more than 1ms, hence conversion will finish which takes 104us

Example code ADC, no interrupt

```
	while (1)
	{
		// Read from ADCH to get the 8 MSBs of the 10 bit conversion
		Ain = ADCH;

		// Typecast the volatile integer into floating type
		// data, divide by maximum 8-bit value, and
		// multiply by 5V for normalization
		Voltage = (float)Ain/256.00 * 5.00;

		// ADSC is cleared to 0 when a conversion completes.
		// Set ADSC to 1 to begin a conversion.
		ADCSRA |= (1<<ADSC);

		// Write Voltage to string format and print
		// (3 char string + "." + 2 decimal places)
		dtostrf(Voltage, 3, 2, VoltageBuffer);
		fprintf(stdout,"%s\n\r",VoltageBuffer);
	}
	return 0;
}
```

Takes more than 1ms, hence conversion will finish which takes 104us

Conversion needs to finish

- Conversion needs to finish before the next conversion is called

- Use a print statement
- Delay functionality (of at least 104us)
- `while (!(ADCSRA & (1<<ADSC) == 0)) { }`
  - The most efficient solution

Connections

- Connect the board as in Fig.1.
- Use a potentiometer to get the sensing value to the ADC peripheral Pins
- Connect AREF pin to 5V to use the external voltage reference properly

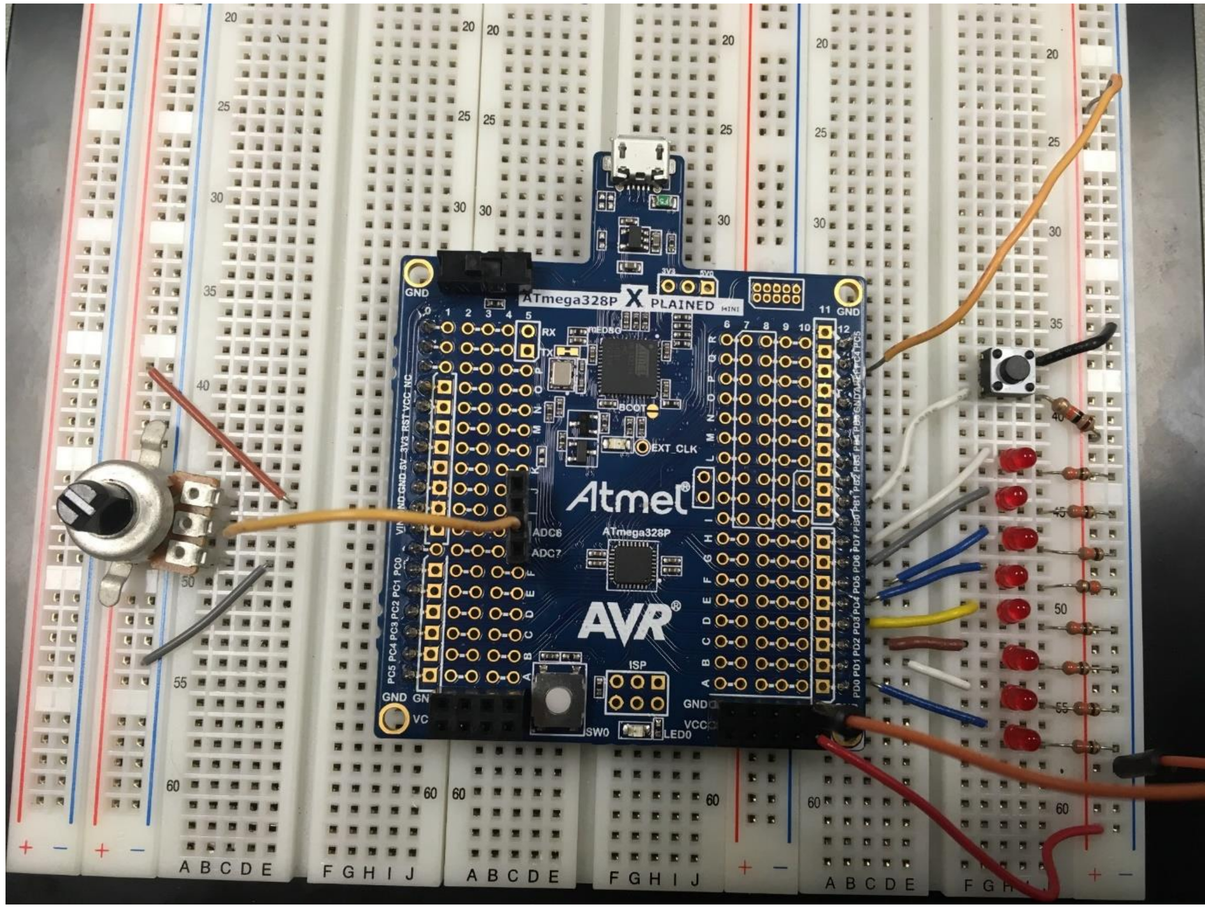
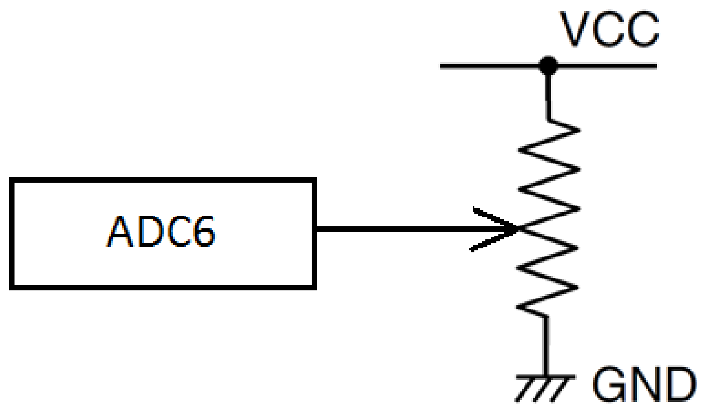


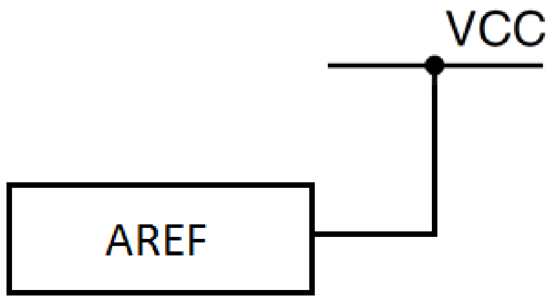
Fig1. Connections for ADC

Hardware Changes for Task1

In this lab, we'll be measuring analog voltages using ADC

- Connect the potentiometer to ADC6 pin as shown below.
- Connect AREF pin to VCC → Reference voltage becomes 5V.





Problem 1: Simple Voltmeter

We are going to design a simple voltmeter that measures voltages between 0-5V with ~4mV resolution.

- Connect a potentiometer to produce variable voltage at ADC6 pin.
- Connect AREF pin to VCC
- Read the analog input voltage using ADC every 1 s
- Convert the ADC reading to voltage measurement
- Print in the voltage to the UART console.

Note: Use the full 10-bit resolution of the ADC

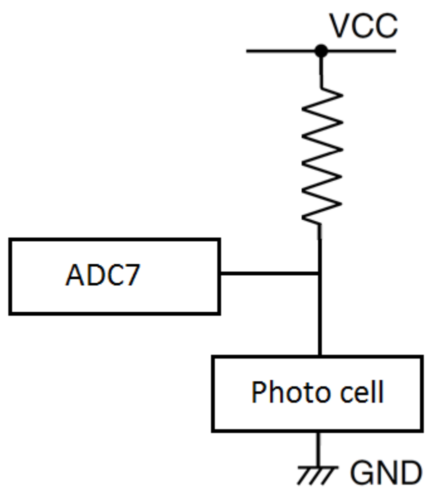
Now you should be able to observe different voltage readings as you twist the potentiometer knob.

Problem 2: Voltage detector

- Create a voltage divider with 2 10KΩ resistors from 5V to GND
- Connect AIN1 (D7) to midpoint of voltage divider
- Connect AIN0 (D6) to output of potentiometer
- When potentiometer value goes higher than 2.5V, turn on LED
  - Use ANALOG_COMP ISR to toggle the LED

Problem 3: Light Sensor

- Replace the potentiometer with a resistor divider network with a photo cell (light dependent resistor) and a 10kOhm resistor to produce variable voltage at ADC7 pin.
- Read the analog input voltage using ADC every 1s
- Print the photo sensor value


VCC
ADC7
Photo cell
GND