CAB202 Topic 11: Analog to Digital Conversion Pulse Width Modulation

Luis Mejias

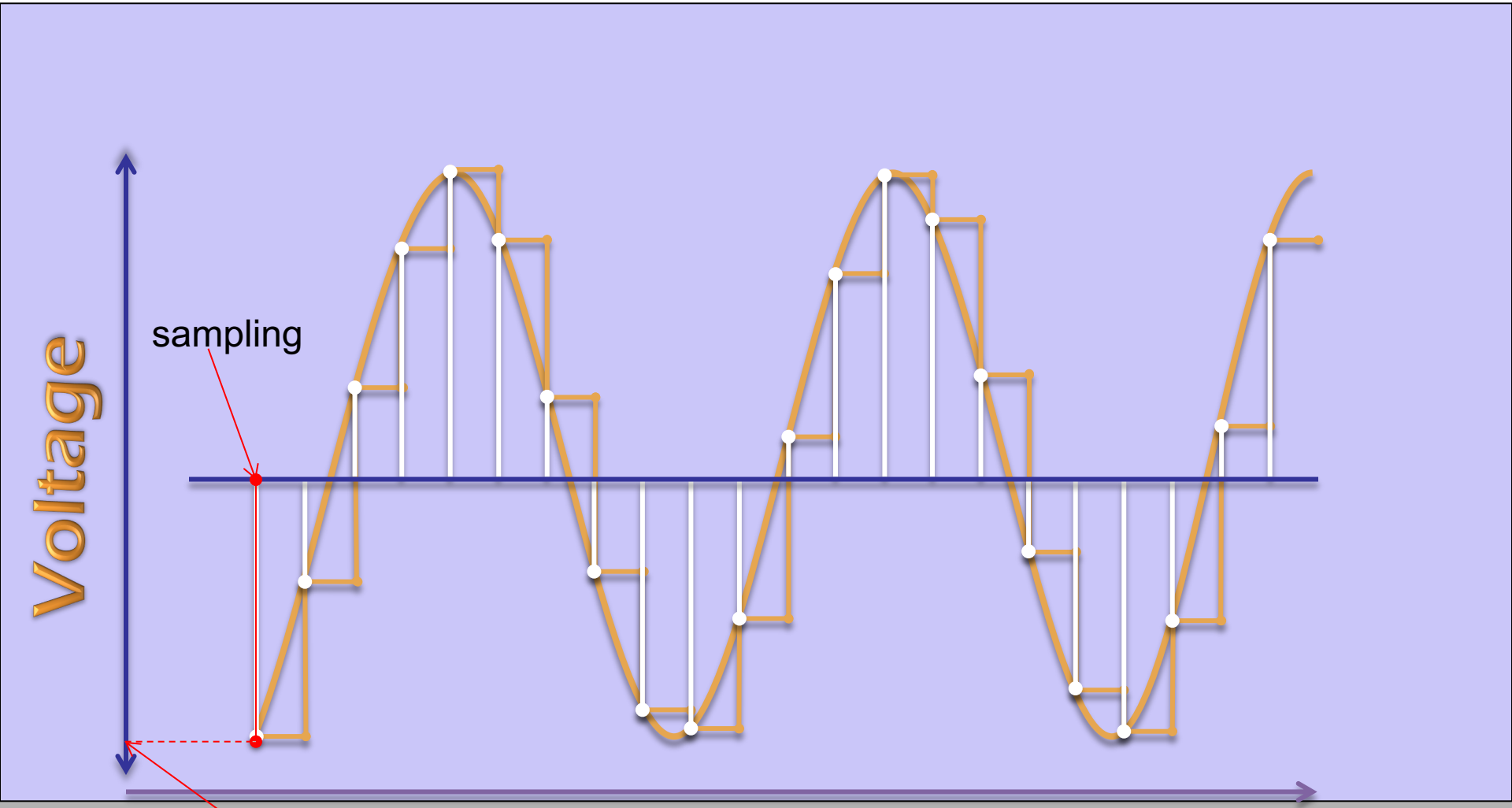
Outline

- Overview Topic 10
- ADC on the Atmega32U4
- ADC Operation & Examples
- PWM on the ATMega32U4
- PWM operation and examples

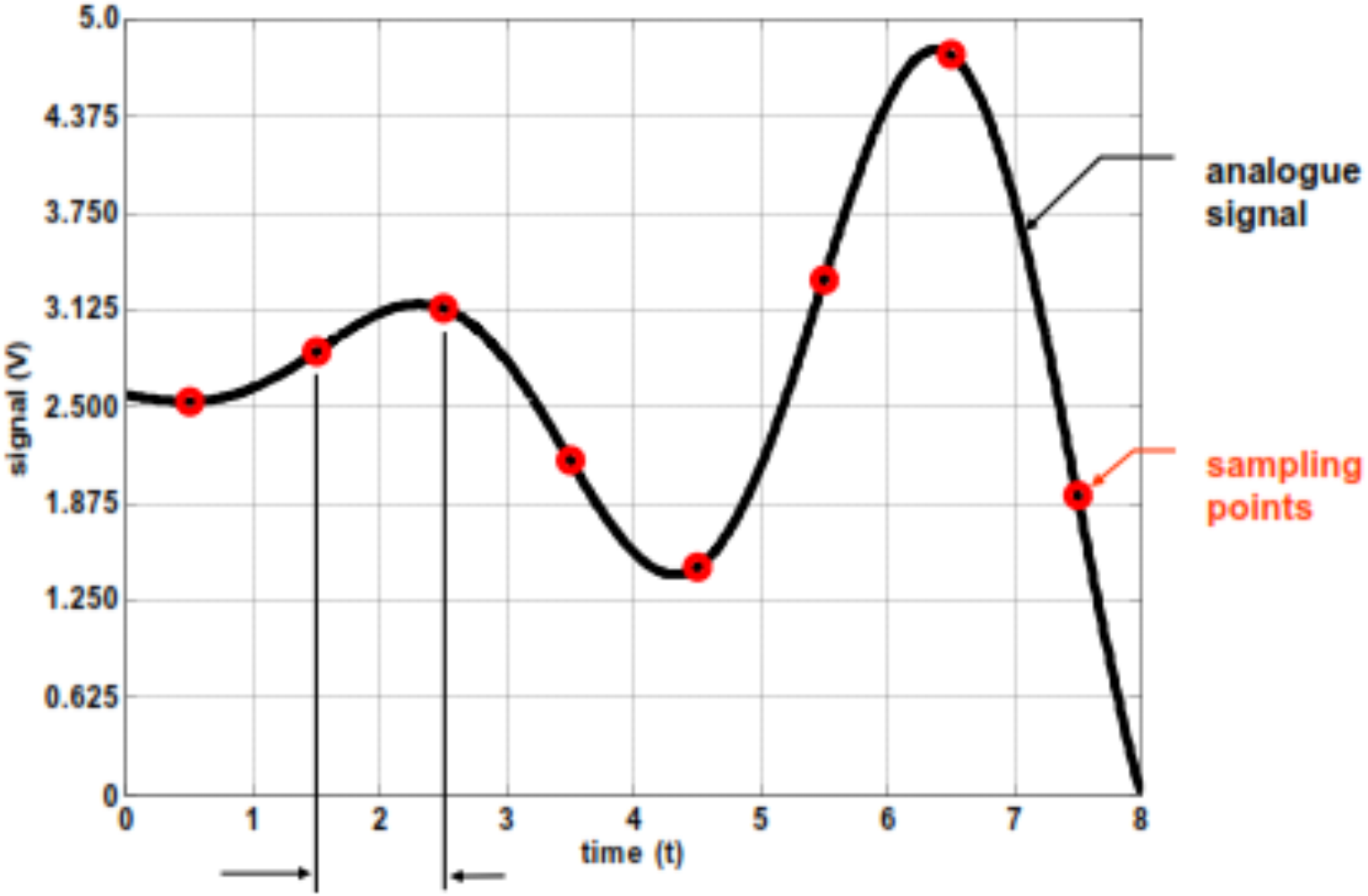
Analog Signals

- Most of the physical quantities around us are continuous.
- By continuous we mean that the quantity can take any value between two extremes.
- For example the atmospheric temperature can take any value (within certain range).
- If an electrical quantity is made to vary directly in proportion to this value (temperature, etc.) then what we have is an analog signal which in most cases is a voltage.
- We have to convert this into digital form if we want to manipulate it with a digital microcontroller.
- For this an ADC or analog to digital converter is needed.

The Analog to Digital Converter (ADC)



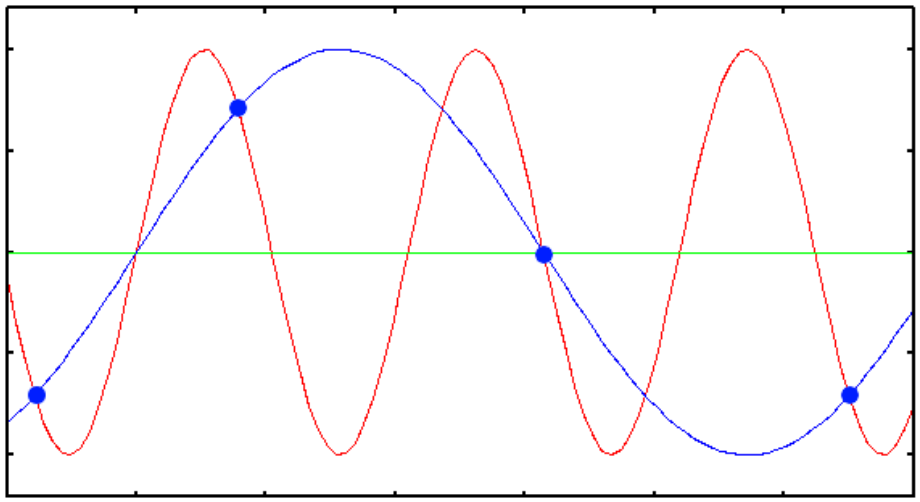
Sampling


5.0
4.375
3.750
3.125
2.500
1.875
1.250
0.625
0
signal (V)
0
1
2
3
4
5
6
7
8
time (t)
analogue signal
sampling points
sampling period Ts
What is a suitable sampling period for a signal

Nyquist Sampling Rate

- The Nyquist rate is the minimum sampling rate required to avoid aliasing, equal to twice the highest frequency contained within the signal.

$$\text{Nyquist Rate} = 2 \times f_{max}$$

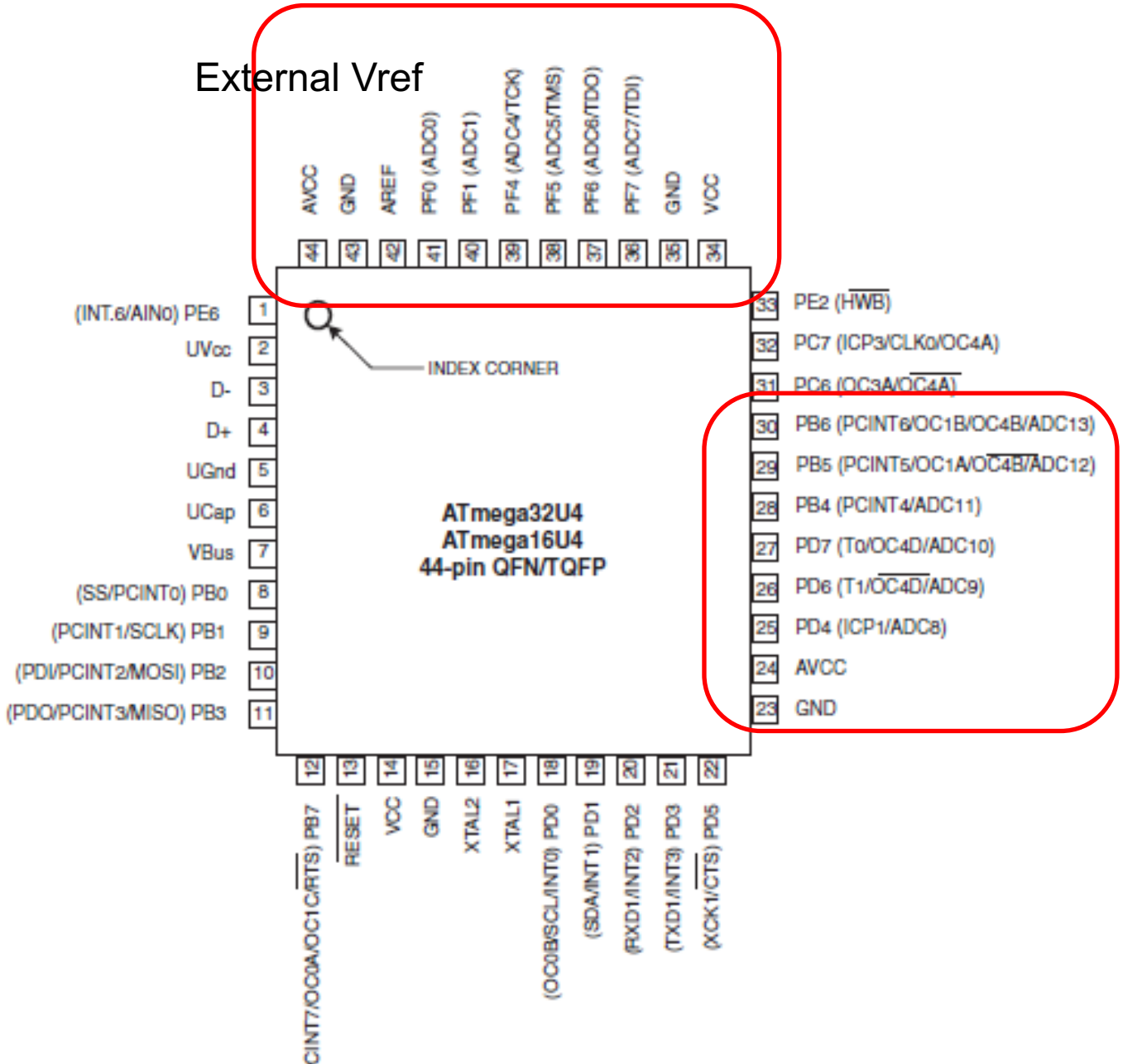


The ADC on the Atmega32u4

- **10/8-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **± 2 LSB Absolute Accuracy**
- **65 - 260 µs Conversion Time**
- **Up to 15 kSPS at Maximum Resolution**
- **Twelve Multiplexed Single-Ended Input Channels**
- **One Differential amplifier providing gain of 1x - 10x - 40x - 200x**
- **Temperature sensor**
- **Optional Left Adjustment for ADC Result Readout**
- **0 - V_{CC} ADC Input Voltage Range**
- **Selectable 2.56 V 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**

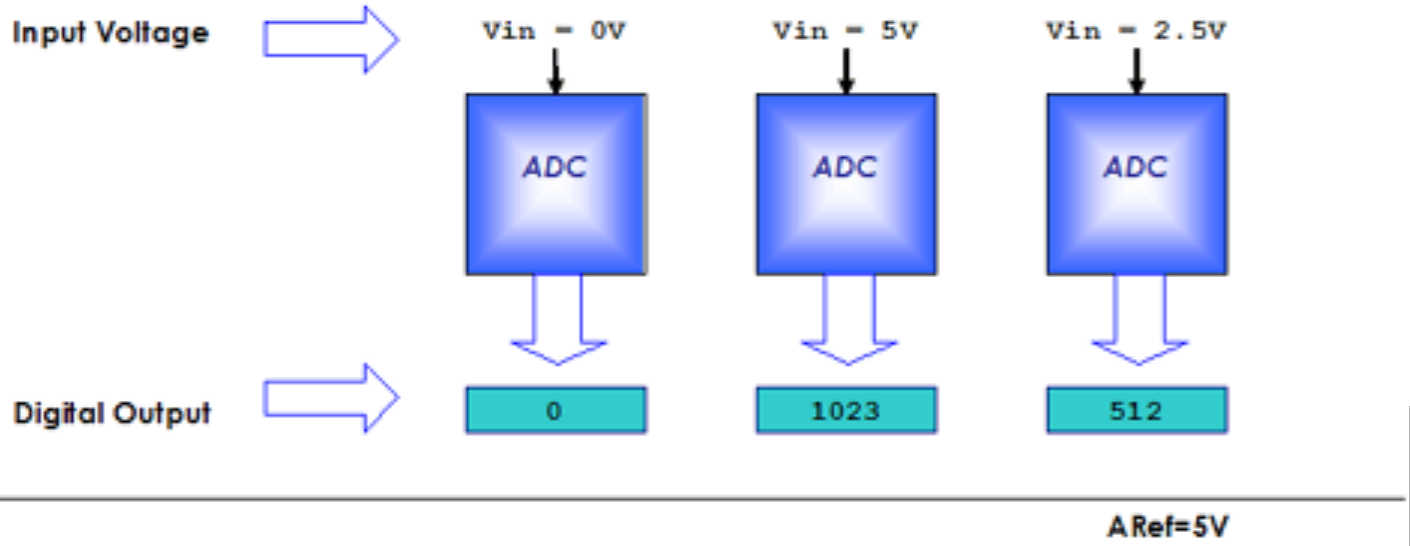
The ATmega16U4/ATmega32U4 features a 10-bit successive approximation ADC. The ADC is connected to an 12-channel Analog Multiplexer which allows six single-ended voltage inputs constructed from several pins of Port B, D and F. The single-ended voltage inputs refer to 0V (GND).

ADC Pins in ATMega32U4

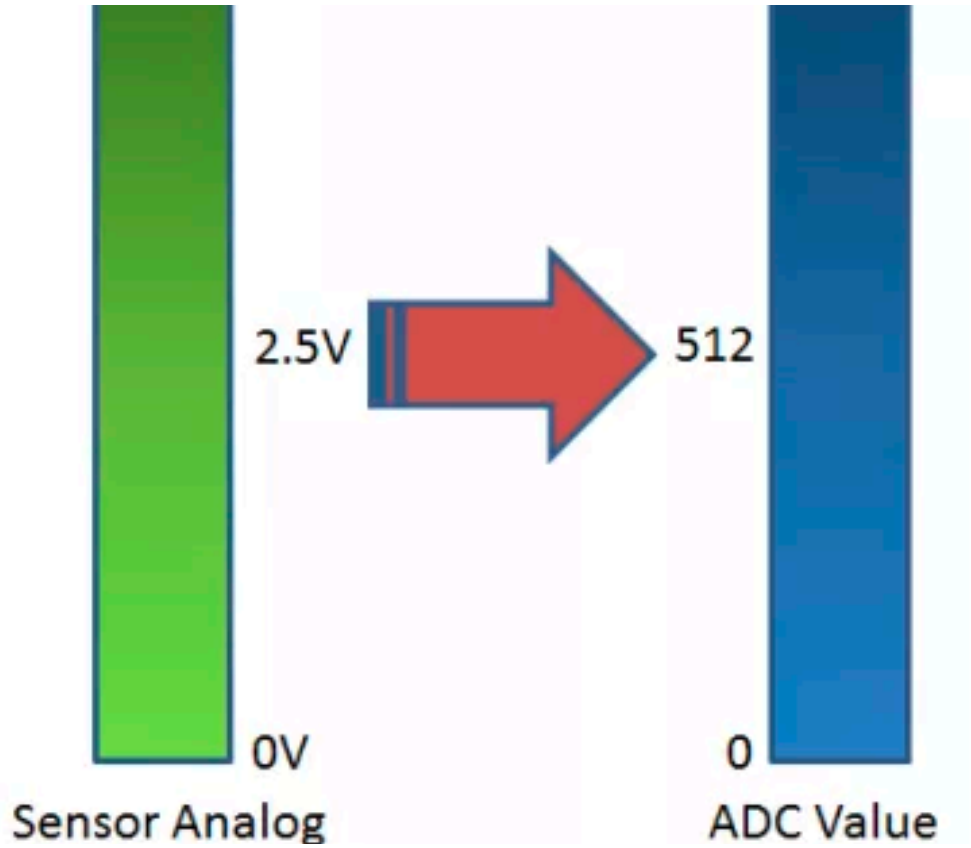


Resolution

- An ADC converts an input voltage into an integer number and therefore has a limited resolution.
- A 10 Bit ADC has a range from 0-1023. (2^10=1024)
- A reference voltage determines the max voltage which can be digitally converted.



Resolution



ATMEGA16/32

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

ADC operation

- The ADC converts an analog input voltage to a 10-bit digital value through successive approximation.
- The ADC is enabled by setting the ADC Enable bit, in a register.
- Voltage reference and input channel selections will not go into effect until the correct bit is set.
- The ADC generates a 10-bit result which is presented in the ADC Data Registers (ADCH and ADCL).
- Conversion can be single or continuous.

ADC operation

- ADC prescaler
  - Analog signal is converted into digital signal at some regular interval.
  - This interval is determined by the clock frequency.
  - ADC operates within a frequency range of 50kHz to 200kHz.
  - CPU clock frequency is much higher (in the order of MHz).
  - So to achieve it, frequency division must take place. The prescaler acts as this division factor.

ADC operation

- ADC Registers – ADMUX, ADCSRA, ADCH, ADCL and SFIOR
  - Register are used to
    - Enable the ADC
    - Start the conversion
    - Select the channel and gain
    - Set Reference voltage
    - Trigger mode (single conversion or continuous)
    - Enable interrupt
    - Set prescaler
    - Etc..
  - So, by setting values in the registers we make possible the ADC process.

ADC operation: Setting the registers

- ADMUX – ADC Multiplexer Selection Register
  - Used to choose the reference voltage, left adjust the result and select channel

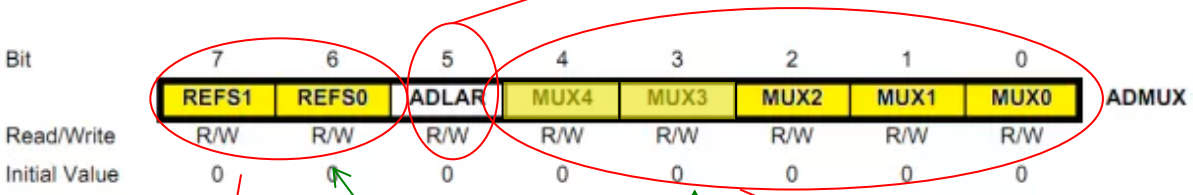
Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bits of interest are in yellow

ADC operation: Setting the registers

- ADMUX

Left adjust: see slide 18




```
ADMUX = (1<<REFS0);
or
ADMUX = 0b 0 1 0 0 0 0 0 0;
```

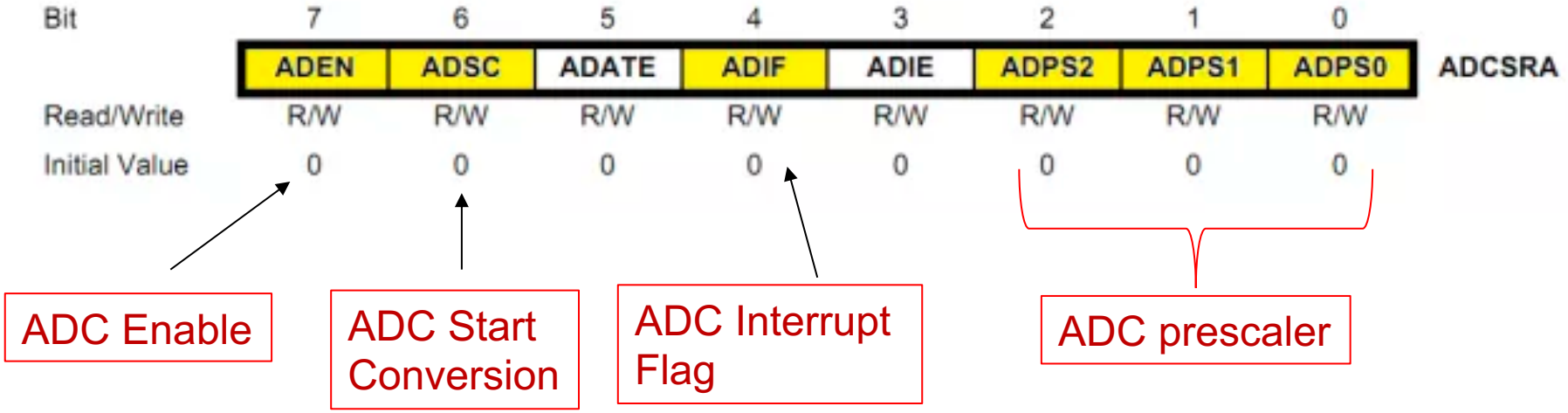
Will select Vref=Vcc and channel 0 (ADC0)

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

MUX4..0	Single Ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

ADC operation: Setting the registers

- ADCSRA – ADC Control and Status Register A
  - Used to enable, start the conversion, select trigger mode, prescaler, etc.



Bits of interest are in yellow

ADC operation: Setting the registers

- ADCSRA

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: it enables the ADC feature.
Bit 6 – ADSC – ADC Start Conversion – Write this to '1' before starting any conversion.
Bit 5 – ADATE – ADC Auto Trigger Enable – Setting it to '1' enables auto-triggering of ADC
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 is determined by selecting the proper combination from the following.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
// prescaler = 128
or
ADCSRA = 0b 1 0 0 0 0 1 1 1;
// prescaler = 128
```

So f_ADC=8 mhz / 128 = 62.5 khz (sampling frequency)

ADC operation: Setting the registers

- ADCL and ADCH – ADC Data Registers
  - The result of the ADC conversion is stored here. Since the ADC has a resolution of 10 bits, it requires 10 bits to store the result. Hence one single 8 bit register is not sufficient. We need two registers – ADCL and ADCH (ADC Low byte and ADC High byte) as follows. The two can be called together as ADC.

```
uint16_t adc_read(uint8_t ch)
{
   // select the corresponding channel 0~7

   // start single conversion
   // write '1' to ADSC

   // wait for conversion to complete
   // ADSC becomes '0' again


    //return conversion
   return (ADC);
}
```

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

Programming ADC

- Library support:
  - cab202_adc.c, cab202_adc.h
- Two functions:
  - adc_init()
    - Register setup to access ADC0 and ADC1 on pins F0 and F1 respectively.
    - These are connected to potentiometers 0 and 1 on the TeensyPewPew
  - adc_read(uint8_t channel)
    - Initiates ADC comparison

Dive into ADC in C: DEMOS

- adc_led.c
  - Reads values of ADC0 and ADC1
  - Displays values on LCD
  - Turns LED on when ADC above threshold

- adc_lcd.c
  - Reads ADC values
  - Move a small shape on the LCD screen

The Digital to Analog Converter (DAC)

- You guessed it! Microcontrollers have accompanying DACs.

- It does exactly the opposite function of an ADC. It takes a digital value and converts it into an pseudo-analog voltage.

- It can be used to do an enormous amount of things. One example is to synthesize a waveform. We can create an audio signal from a microcontroller. Imagine that!

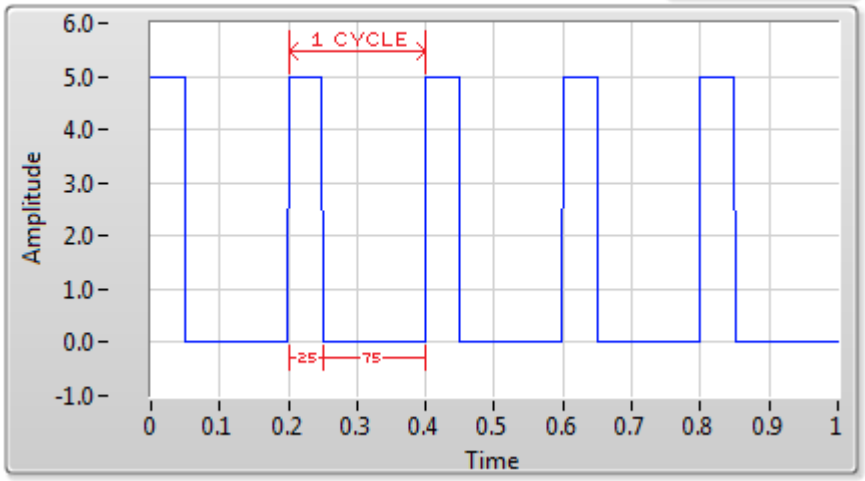
Pulse-Width Modulation (PWM)

- Modulation technique used to encode information into a signal, although its main use is for regulating power supplied to a load.

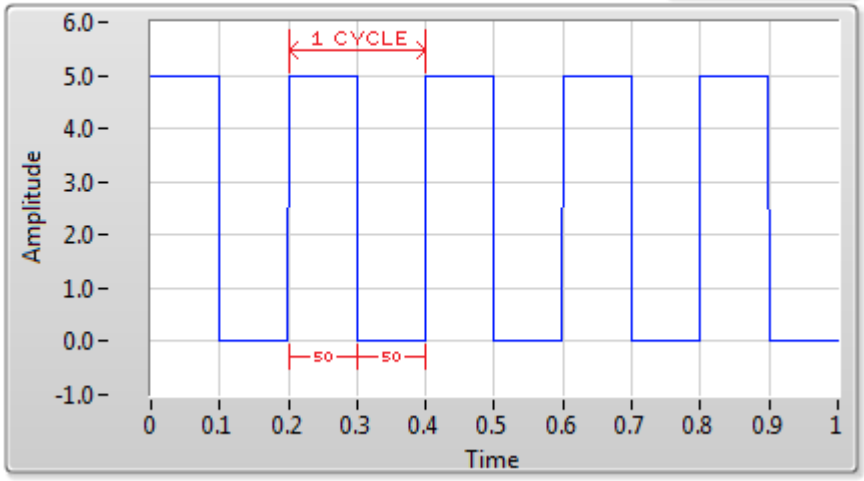
- An Analog signal can be generated using a digital source.

- Consist of two main components that define its behavior: a duty cycle and a frequency.
  - The duty cycle describes the amount of time the signal is in a high (on) stated as a percentage of the total time it takes to complete one cycle.
  - The frequency determines how fast the PWM completes a cycle (i.e. 1000 Hz would be 1000 cycles per second), and therefore how fast it switches between high and low states.
- By cycling a digital signal off and on at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices.

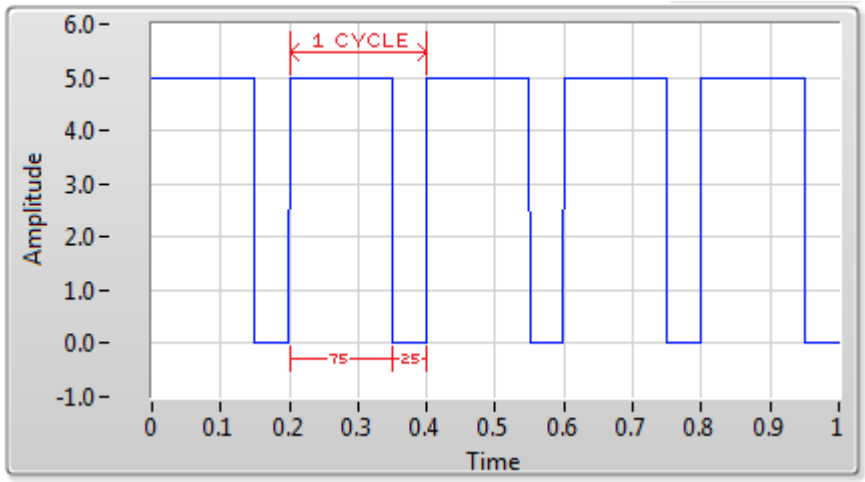
Pulse-Width Modulation (PWM)



25% duty cycle



50% duty cycle



75% duty cycle

Implementing PWM on ATMega32U4

- ATMega32U4 has 4 timers
- Each timer can be connected to 2 or more output pins
- Each timer has a counter, which cycles:
  - In one direction, from 0 to TOP, at which point the timer wraps back to 0, or from TOP downward to 0.
  - In two directions, from 0 to TOP, and then backward down to 0.
- A timer can be set to repeatedly compare its counter to a threshold value set in a compare register
  - When the counter reaches the threshold, or when it hits 0, it can toggle the value of a digital output pin.
- We can use this to implement PWM in hardware.

Implementing PWM: General Steps

- Setup:
  - Set timer overflow period by choosing prescaler and TOP value.
  - Set timer counter waveform:
    - Sawtooth: 0..TOP; TOP..0;
    - Triangular: 0..TOP..0
  - Enable PWM for designated output pin:
    - Set corresponding bit in DDR & Timer control register
- Run:
  - Set timer overflow compare register threshold
  - The pin will toggle on overflow and when the counter hits the compare value
  - The duty cycle emerges from this.

PWM Code Samples

- adc_pwm_backlight.c: Use Pot0 to control the brightness of the LCD backlight.
  - void setup(void) – Initialises PWM on OC4A channel (C7)
  - void set_duty_cycle(int cycle) – sets the output compare register for OC4A to a value between 0 and 1023 to adjust brightness.

- adc_pwm_led.c: Use Pot1 to control the brightness of the orange LED
  - void setup(void) – Initialise PWM on OC4D complement channel (D6)
  - *Unfortunately, OC4D is connected to LCD Slave Select, so the LCD display is unusable in this example*.

Summary

- ADC is a key operation when using microcontrollers and sensors
- Variables used to store ADC conversion values should be properly declared (10bits).
- PWM signal are quite common in robotic applications, e.g, drone servos, speed controllers, wheel motors, etc.