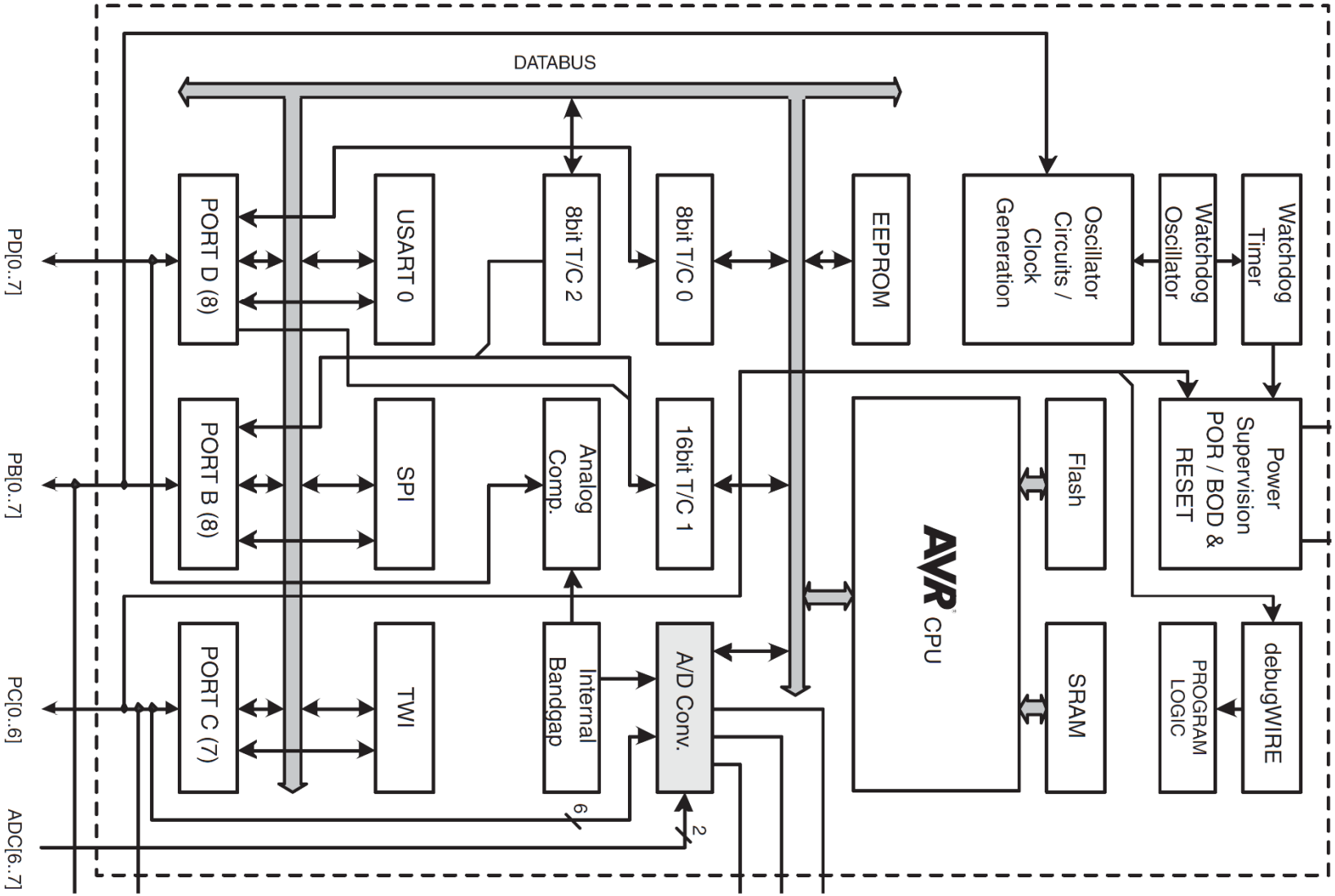
ENGR290: Introduction to microcontrollers and sensors

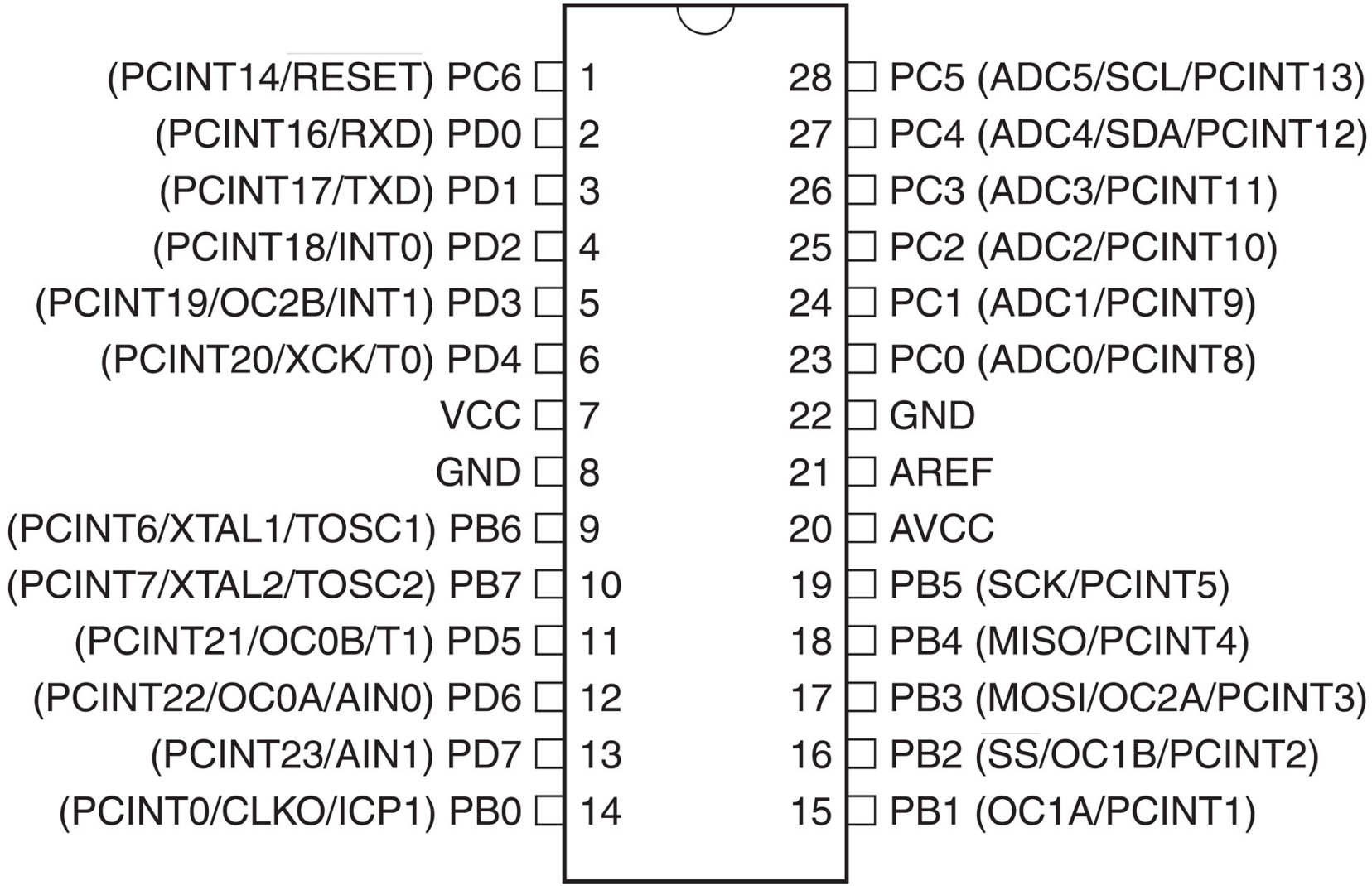
OVERVIEW

- What is a microcontroller?
- Peripherals
- General Purpose Input/Output (GPIO) ports
- Timers and Pulse Width Modulation (PWM)
  - 8-bit Timer0
  - 16-bit Timer1
- Interrupts
- Analog to Digital Converter (ADC)
- Communication Peripherals
  - Universal Synchronous/Asynchronous Receiver Transmitter (USART/UART)
  - Two Wire Interface (TWI or I2C)
- Sensors
  - Infrared rangefinder
  - Ultrasonic rangefinder
  - Inertial Measurement Unit (IMU)
- Programming Environment
  - Arduino
  - Atmel Studio
- References

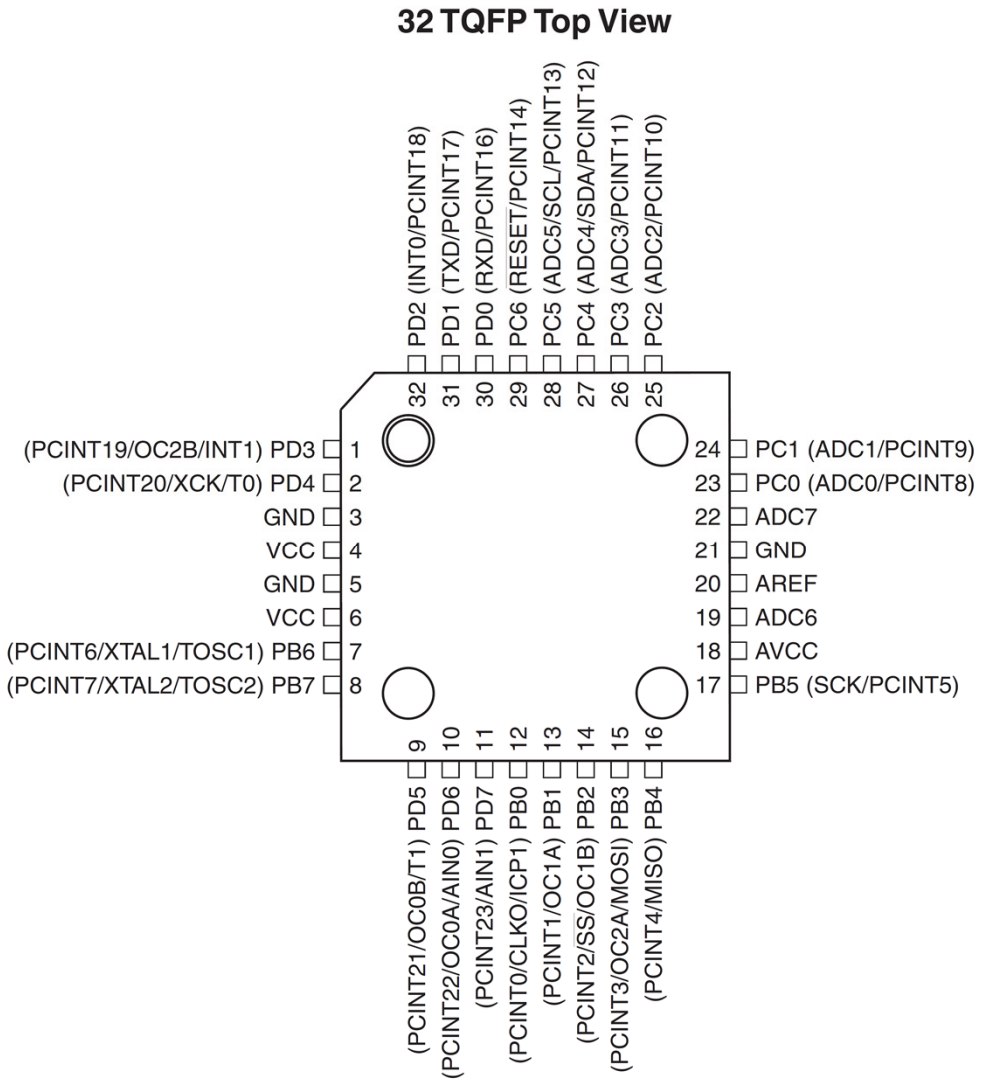
What is a microcontroller?



PIN LAYOUT, P-DIP



PIN LAYOUT, 32 TQFP

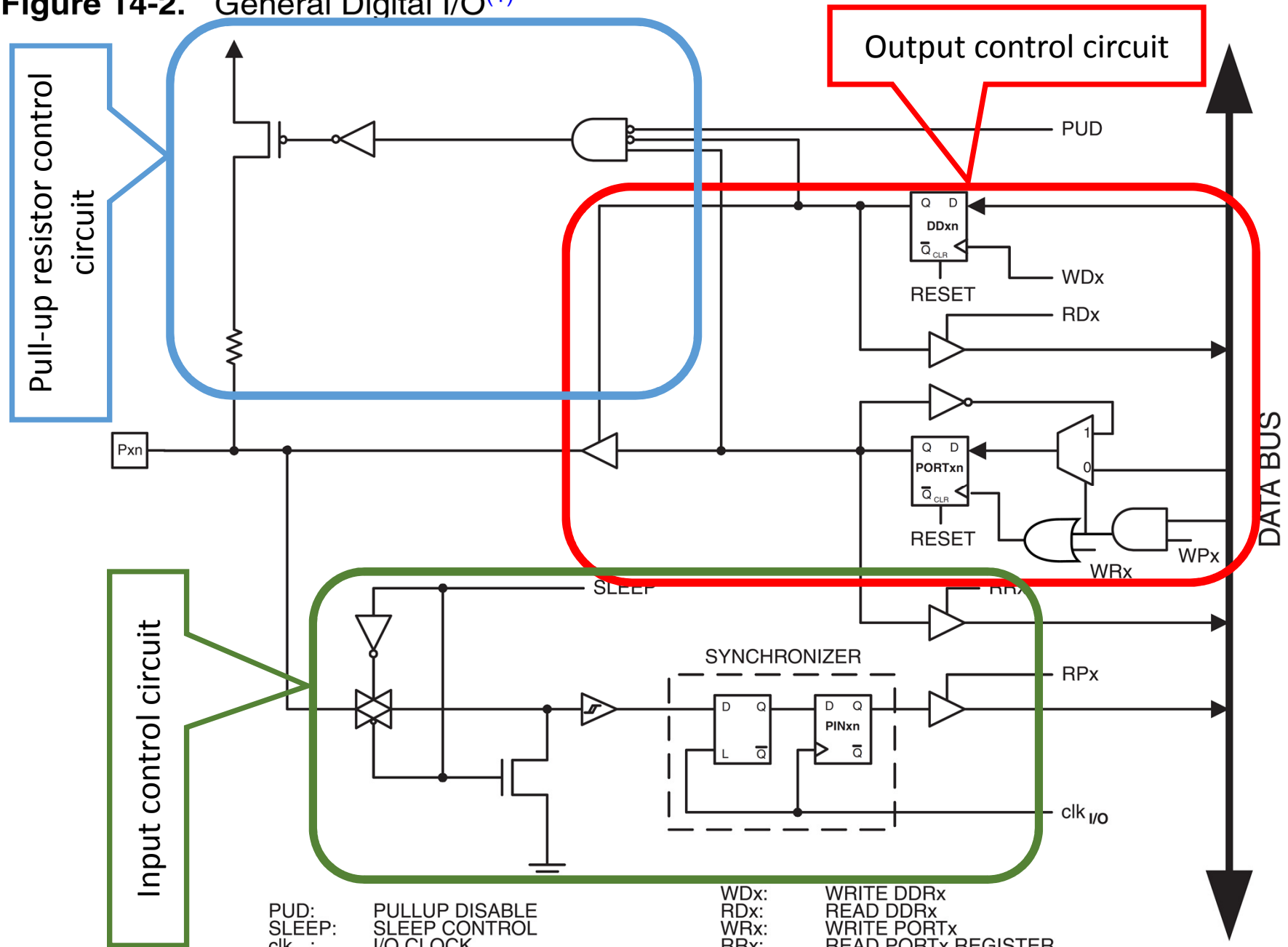


Peripherals

- **Timer/Counters**:
  - 8-bit with Separate Prescaler and Compare Mode
  - 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Pulse Width Modulation (PWM) Channels
- **Analogue to Digital Converter (ADC) (10-bit)**:
  - Analogue inputs
  - Temperature Measurement
- **Programmable Serial USART**
- Master/Slave SPI Serial Interface
- **Byte-oriented 2-wire Serial Interface (Philips I2C compatible)**
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

General Purpose Input/Output (GPIO) ports

Figure 14-2. General Digital I/O[(1)]



GPIO PORTs – overview

- Three I/O ports: B (8-bit), C (7-bit) and D (8-bit).
- Define I/O port's pins **direction** DDRx (x=B, C or D).
- Direction can be set for each pin **independently**.
- Read (direction=0, default) - Input, Write (direction=1) – Output.
- Output pins **can** be read!
- **Writing** to **input** pins enables (1) or disables (0) **internal pull-up resistors**.
- Write: PORTx=value.
- Read: variable=PINx.
- Unused pins must be defined as inputs and internally pulled-up.

Bitwise Operators review

|	bit OR
&	bit AND
~	bit NOT
^	bit EXLUSIVE OR (XOR)
<<	bit LEFT SHIFT
>>	bit RIGHT SHIFT
=	overwrite ALL bits (1clk)
|=, &=, ^=	read-modify-write (3clk)

I/O: PORTs – Examples

- Direction:
  - DDRB=((1<<PB0)|(1<<PB3)|(1<<PB4)|(1<<PB6)); //Setting port B's pins 0, 3, 4 and 6 as outputs, the rest - as inputs
  - DDRD|=(1<<PD2); // changing the direction of pin2 to output
  - DDRC&=~(1<<PC0); // changing the direction of pin0 to input

- Writing to port B (outputting through the port):
  - PORTB=(1<<PB3); // sets bit 3, resets all other bits.
  - PORTB|=((1<<PB3)|(1<<PB6)); // sets bits 3 and 6, i.e. output high level.
  - PORTB&=~((1<<PB0)|(1<<PB4)); // resets bits 0 and 4, i.e. output low level.

- Reading from ports (inputting data through the port):
  - my_variable_D=PIND; //read all 8 bits, including outputs
  - pinB3_status=(PINB&(1<<PB3)); //check status of pin3

ENGR290 controller-specific initialisation

ENGR290 controller has a number of external peripherals (Bluetooth, motor drivers, RF receiver, etc.) connected to the microcontroller. It is important to initialise microcontroller's GPIOs properly and use them to configure, enable or disable the external peripherals. The full list of GPIO pins connections is available in a separate document. The code below must be included at the beginning of your main(). (Note when "=", and "|=" or "&=" is used.)

```
cli(); // ------- Ports init -------
  DDRB =((1<<PB4)|(1<<PB2)|(1<<PB1)|(1<<PB0));
  PORTB&=~((1<<PB2)|(1<<PB1)|(1<<PB0));  //optional as the default values are 0.
```

What would have been written to the port, if "=" was used?

```
  PORTB|=((1<<PB3)|(1<<PB4)|(1<<PB5));  //PB5 is an input. Why is it written 1 to?
  DDRC =((1<<PC2)|(1<<PC1));
  PORTC&=~((1<<PC2)|(1<<PC1)); //optional as the default values are 0
  PORTC|=(1<<PC0); //if you don't use PC3-PC5 what should you add to this line?
  DDRD =((1<<PD7)|(1<<PD6)|(1<<PD5)|(1<<PD4)|(1<<PD1));
  PORTD&=~((1<<PD7)|(1<<PD6)); //optional as the default values are 0
  PORTD|=((1<<PD1)|(1<<PD2)|(1<<PD3)|(1<<PD4)|(1<<PD5));
```

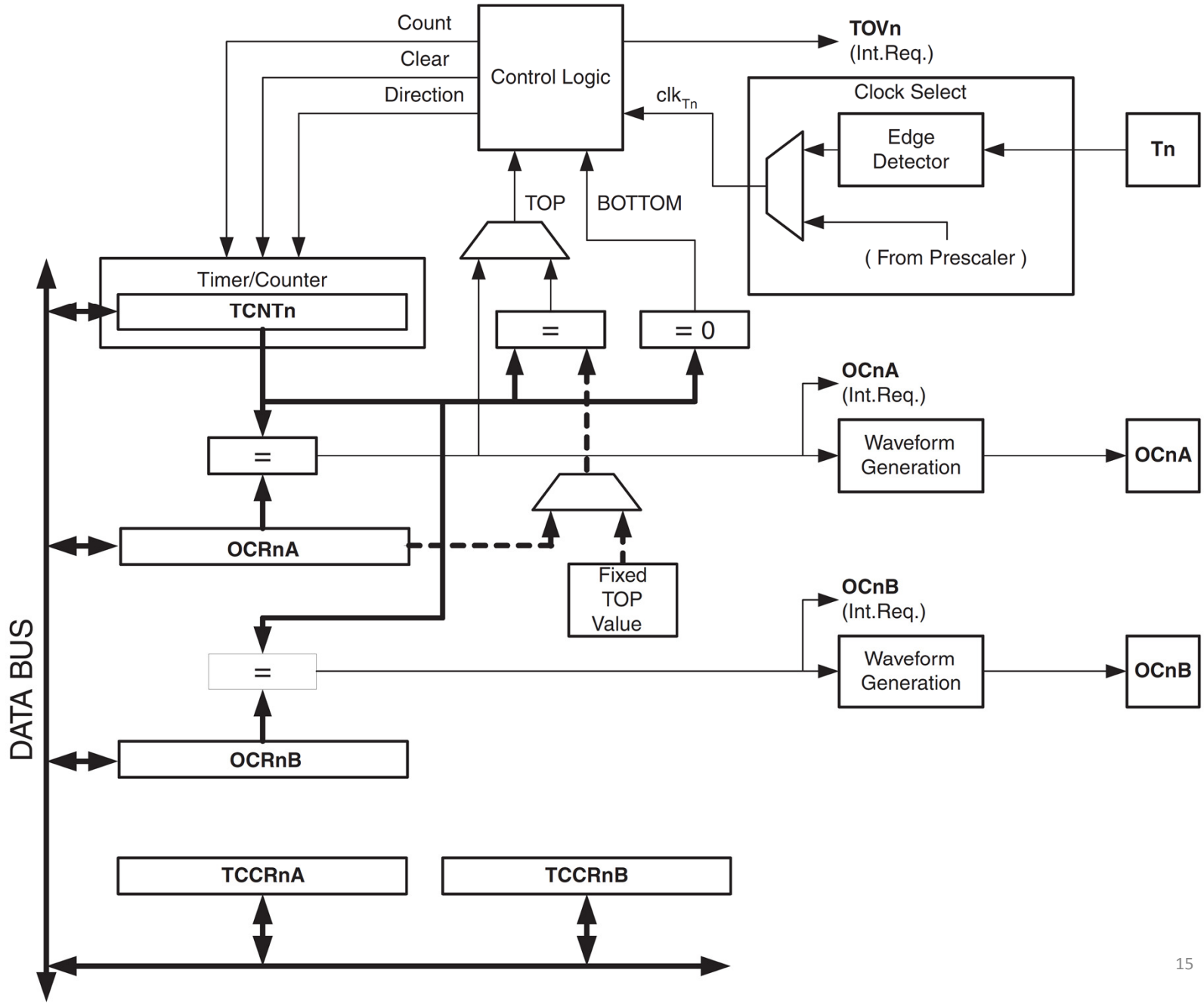
From now on, you have to use in your code only "|=" or "&=" when you manipulate the port's bits.

Timers

8-bit Timer/Counter0 with PWM

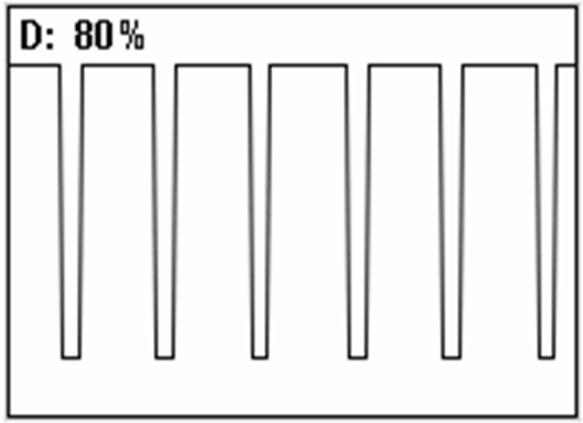
- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- **Variable PWM Period**
  - Fast PWM
  - Phase Correct PWM
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)

Figure 15-1. 8-bit Timer/Counter Block Diagram



What is Pulse-Width Modulation?

- Pulse-Width Modulation (PWM) is a modulation technique that generates a square wave with variable duty-cycle to represent the amplitude of an analog signal.
- Applications include:
  - Generation of analog voltages
  - Dimming LEDs
  - Controlling speed of motors
  - Generation of sound



http://en.wikipedia.org/wiki/Duty_cycle

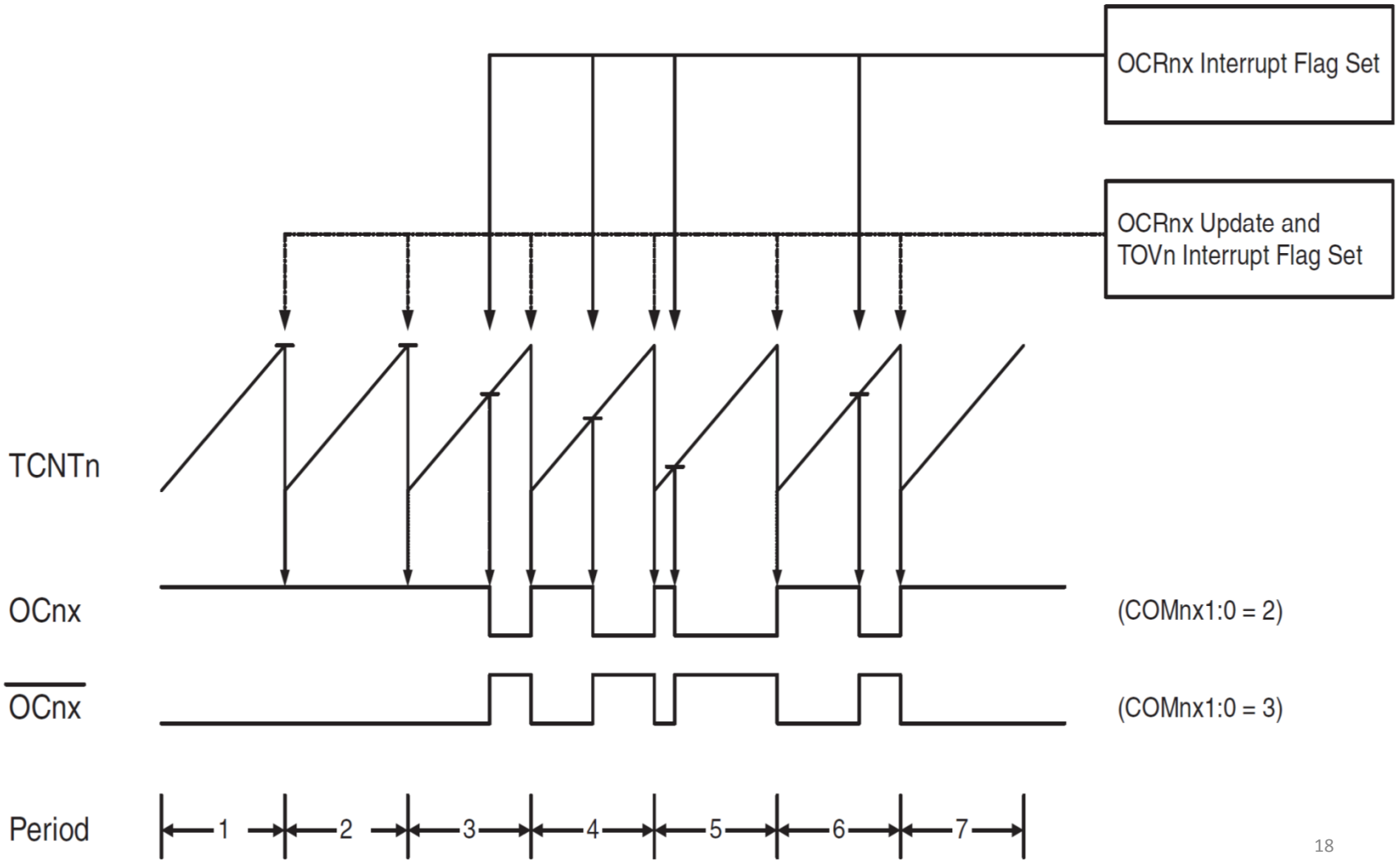
- Duty Cycle $D = \frac{t_{ON}}{t_{ON}+t_{OFF}} = \frac{t_{ON}}{T}$

PWM output pin modes

- **Inverted Mode –** In this mode, if the waveform value is greater than the compare level, then the output is set high, or else the output is low, i.e. Output Compare Register (OCR) contains “inverted” duty cycle value. This is the mode to use when you work with “**active-LOW**” loads, e.g. an LED with its anode connected to Vcc.
- **Non-Inverted Mode –** In this mode, the output is high whenever the compare level is greater than the waveform level and low otherwise, i.e. OCR contains real duty cycle value. This is the mode to use when you work with “**active-HIGH**” loads, e.g. an LED with its cathode connected to GND.
- **Toggle Mode –** In this mode, the output toggles whenever there is a compare match. If the output is high, it becomes low, and vice-versa. Note that the output frequency will be divided by 2.

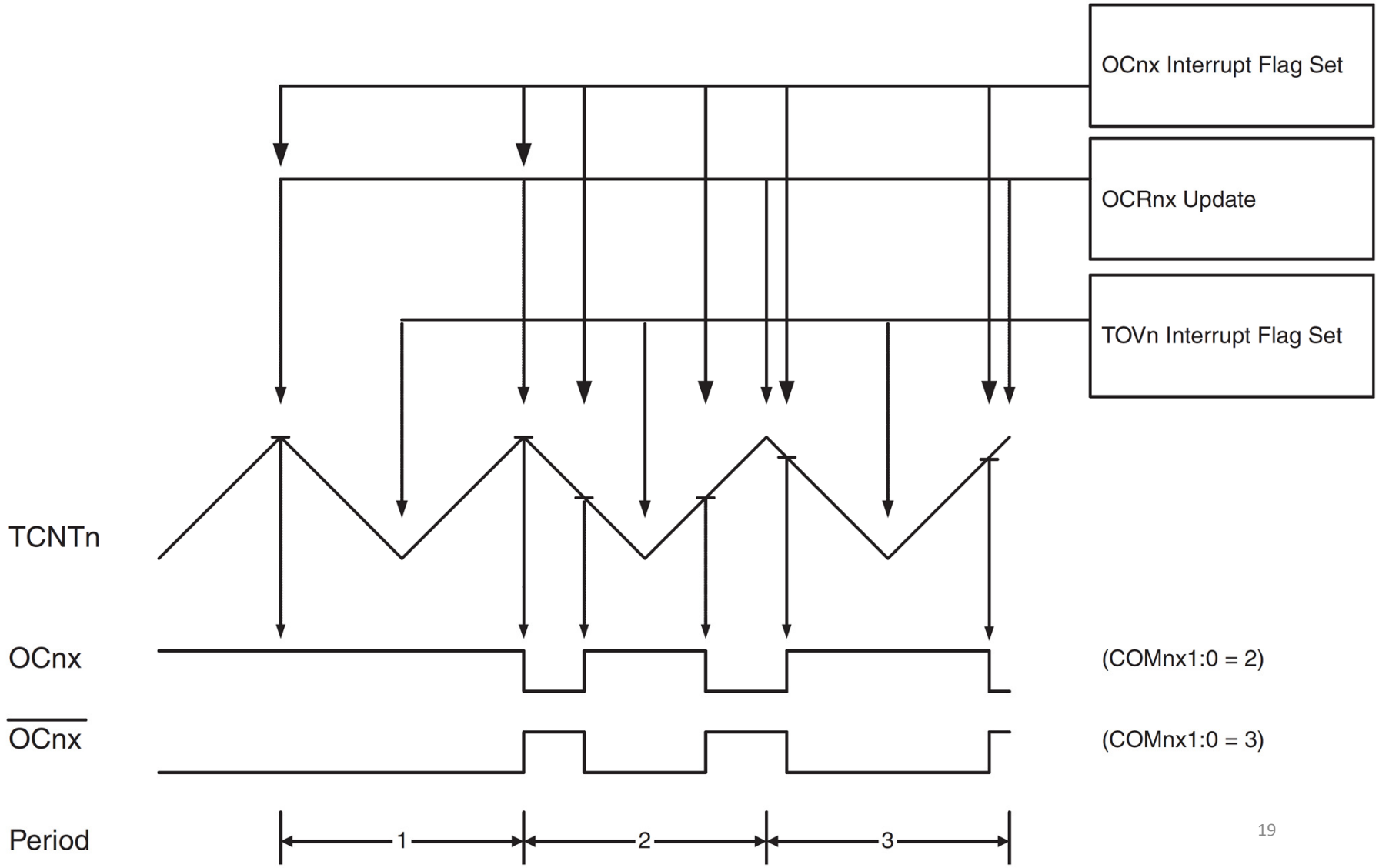
PWM: Fast PWM

Figure 15-6. Fast PWM Mode, Timing Diagram



Phase Correct PWM

Figure 15-7. Phase Correct PWM Mode, Timing Diagram

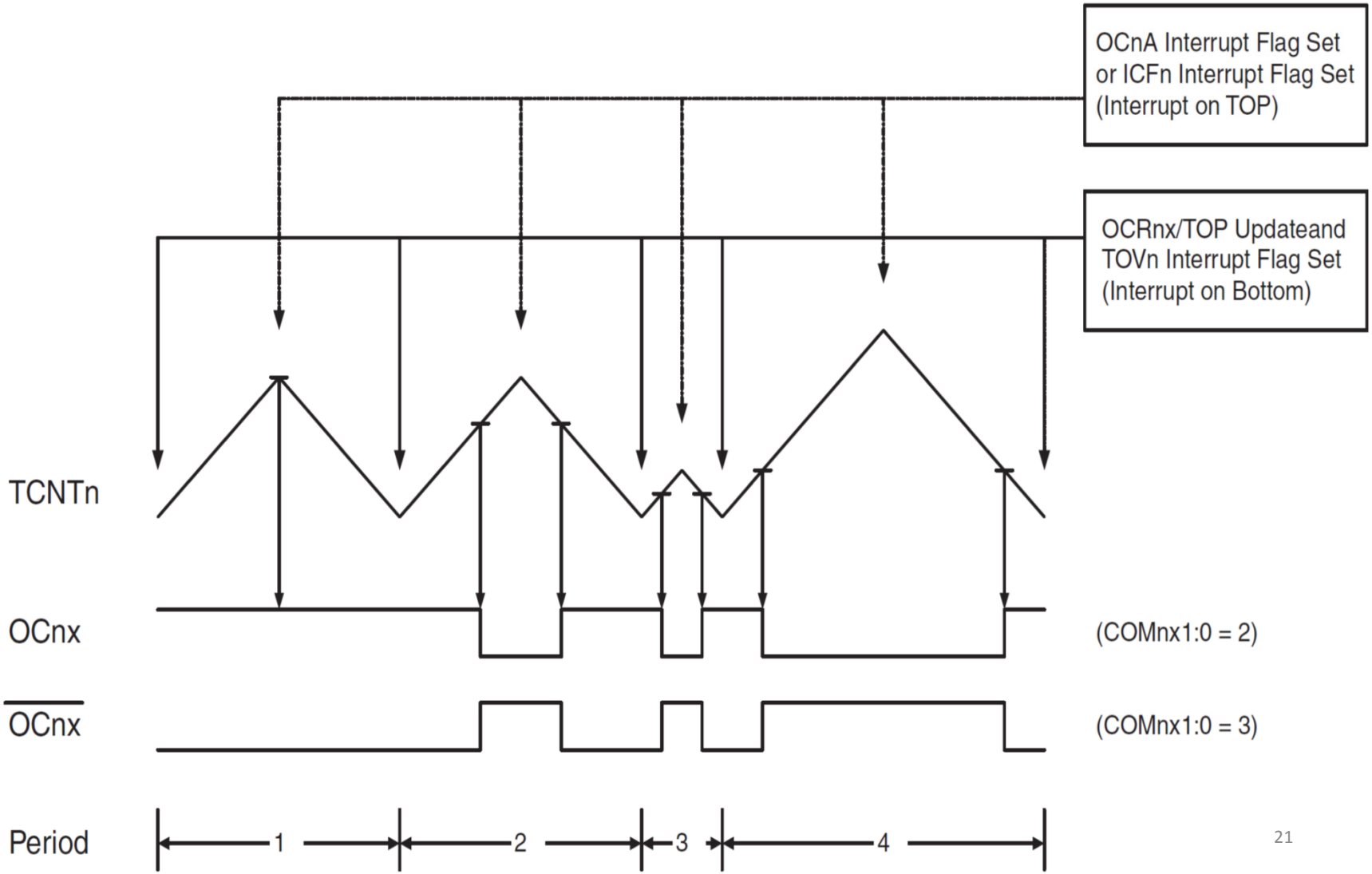


16-bit Timer/Counter1 with PWM

- True 16-bit Design (i.e. allows 16-bit PWM)
- Two independent Output Compare Units
- Double Buffered Output Compare Registers
- **One Input Capture Unit**
- Input Capture Noise Canceler
- Clear Timer on Compare Match (Auto Reload)
- Variable PWM Period
  - Fast PWM
  - Phase Correct PWM
  - **Frequency and Phase Correct PWM**
- Frequency Generator
- External Event Counter
- Four independent interrupt Sources (TOV1, OCF1A, OCF1B, and **ICF1**)

Phase and Frequency Correct PWM Mode

Figure 16-9. Phase and Frequency Correct PWM Mode, Timing Diagram



Register description

- **TCCR1A – Timer/Counter1 Control Register A**

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **TCCR1B – Timer/Counter1 Control Register B**

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **TIMSK1 – Timer/Counter1 Interrupt Mask Register**

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

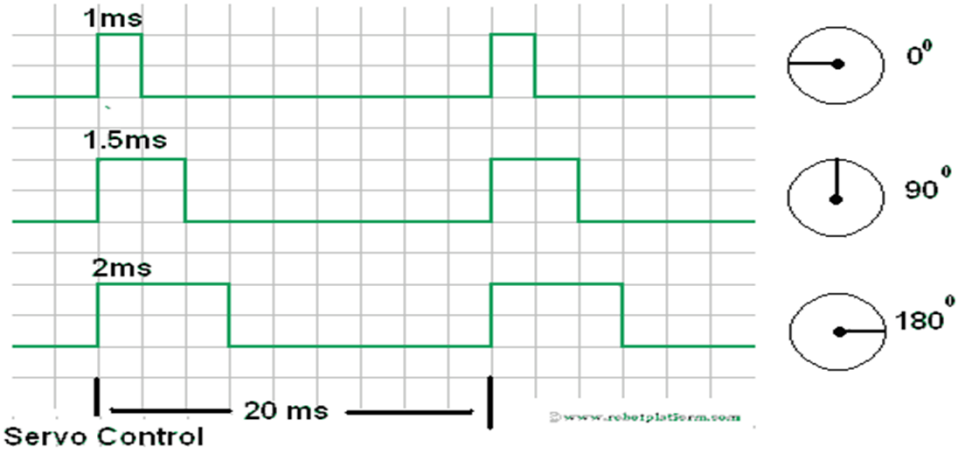
- **TIFR1 – Timer/Counter1 Interrupt Flag Register**

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Example: Timer1 for servo-motor control

To control a servo motor, the controller should produce a square wave pulse:

- t_{on}(min)=1ms
- t_{on}(max)=2ms
- T=20ms or f=50Hz



http://www.robotplatform.com/knowledge/servo/servo_control_tutorial.html

```
TCCR1A|=(1<<COM1A1); // Define PWM pin (PB1) operation.
TCCR1B|=(1<<WGM13);  //PWM, Phase and Frequency Correct, TOP=ICR1
ICR1=2500; //50Hz PWM: F_CPU=16MHz
OCR1A=187; // 1.5ms pulse; servo in the middle (90º) position
TIMSK1|=(1<<ICIE1); // Enable interrupt
TCCR1B|=((1<<CS11)|(1<<CS10)); // Prescaler=64. Start the timer.
```

Example: Timer0 for a DC motor control

Output of DC motors, including the fans* used for the project, can be controlled by varying the power supply voltage applied to them.

PWM lets us change the RMS value of the voltage applied to the fans.

TCCR0A|=(1<<COM0A1); // non-inverted pin operation

TCCR0A|=(1<<WGM00); // PWM, Phase Correct

OCR0A=0; // D=0, i.e. the fan is OFF. OCRA=255 will turn the fan ON at 100% of its power.

TCCR0B|=((1<<CS01)|(1<<CS00)); // Prescaler=64. Start the timer.

Note that in both examples only one channel (A) was used. Think how you would modify TCCRxA to enable channel B PWM output.

* The fans use brushless motors and brushless motors controllers. This combination responds to the change of power supply voltage similarly to a brushed DC motor.

Interrupts

INTERRUPTS: What is an Interrupt?

- **Interrupts:**
  - Interrupt the main process.
  - Execute a pre-defined function (Interrupt Service Routine (ISR)).
  - Asynchronous to the main process.
  - Cannot receive or return any variable/value.
  - Operate with global variables only.
  - Variables used by interrupts MUST be declared “volatile”.

- **Types of interrupts:**
  - Internal
  - External

INTERRUPTS: Getting started

1. Identify the source of interrupt:
   - Internal – configure the source of interrupt
   - External – make sure the source of interrupt is connected to the corresponding pin
2. Define the ISR
3. Configure and enable the interrupt
4. Enable global interrupts

- Status Register – SREG
  - I-bit: Global interrupt enable – this bit must be set in order for any interrupt to function
  - sei(); // used to set I-bit in SREG
  - cei(); // used to clear I-bit in SREG

Table 12-4. Reset and Interrupt Vectors in ATmega168A and ATmega168PA

VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

External interrupts

INTERRUPTS: examples (1/2)

External interrupt on INT0 pin connected to a push-button

1. The pin is connected to a pull-up resistor to Vcc and a push-button (normally open) to GND.
2. Interrupt Vector INT0:

```
ISR(INT0_vect)  {
    button_press++; // ISR code. Keep it short!
}
```

3. Configure and enable the interrupt:

```
EICRA|=(1<<ISC01); // The falling edge of INT0 generates an interrupt request, i.e. “button-down”.
EIMSK|=(1<<INT0); // External Interrupt Request 0 Enable
```

4. **sei(); // Enable global interrupts**

INTERRUPTS: examples (2/2)

Interrupt on Timer0 overflow (e.g. real time clock)

1. Identify the source of interrupt – internal peripheral Timer0
2. Define the Interrupt Vector TIMER0_OVF

```
ISR(TIMER0_OVF_vect) {
    current_time++; // ISR code. Keep it short!
}
```

3. Configure and enable the interrupt:

```
TCCR0A=0; // Normal Mode. Optional step as the default value is 0.
TCCR0B=((1<<CS01)|(1<<CS00)) ; // Prescaler=64.
TIMSK0=(1<<TOIE0); // Enable overflow interrupt.
```

4. **sei(); // Enable global interrupts**

INTERRUPTS: Exercise/homework

1. Simple interrupt:
   - Write a program that blinks LED D5 with frequency of 1Hz
   - The control must be done by the means of an interrupt:
     - The main() must contain only port and peripherals initialisation.
     - while(1) loop must be empty.

2. PWM:
   - Write a program that gradually lights up the LED D4 from 0 to max in 3 seconds, then dims it from max to 0 in 5 seconds and repeats the cycle.
   - The control must be done by the means of PWM on Timer0 (channel B) and Timer1 interrupt (you can adapt part of your code from #1)
     - The main() must contain only port and peripherals initialisation.
     - while(1) loop must be empty.

Analog-to-Digital Converter

What is Analog-to-Digital Converter? (1/2)

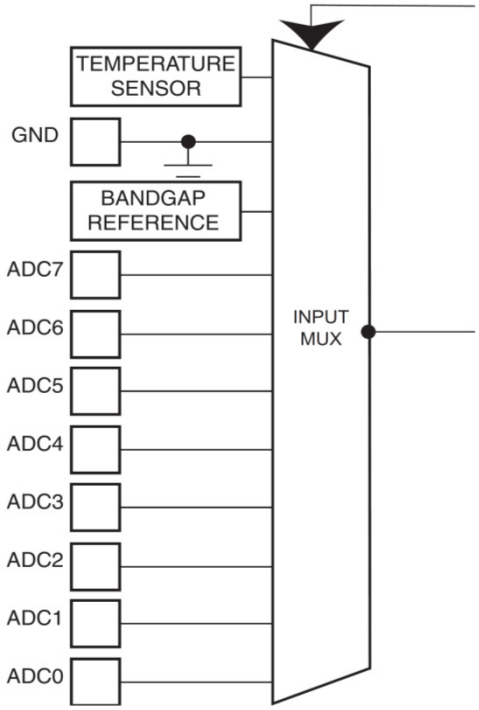
- Needed when interfacing a microcontroller with various sensors that output analog voltage, e.g. GP2D12
- ADC translates an analog signal to an 8- or 10-bit number that the microcontroller can process:

$$ADC = \frac{V_{in} * 1024 \text{ (or 256 for 8-bit)}}{V_{ref}}$$

- ADC voltage reference:
  - AREF (external)
  - AVcc
  - Internal voltage reference (1.1V for Atmega328)
- Optimal $V_{ref}=V_{in}(max)$:
  - $V_{ref}>V_{in}(max)$ – loss of performance (resolution) – no reason to use.
  - $V_{ref}<V_{in}(max)$ – loss of data (saturation) – sometimes might be useful.

What is Analog-to-Digital Converter? (2/2)

- Multiple ADC inputs:
  - Physical ADCs (rare!)
  - Multiplexed channels (common)
- Conversion Modes
  - Single conversion
  - Free-running
- ADC Interrupt
- ADC data register
  - 8- or 10-bit (V_{ref}=5V):
    - 5/1024=4.9mV
    - 5/256=19.5mV
  - Result alignment (left or right)
- Prescaler: sampling frequency



ADC registers (1/3)

- ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0
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

Table 24-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Table 24-4. Input Channel Selections

MUX3...0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8(1)
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

ADC registers (2/3)

ADCL and ADCH – The ADC Data Register

ADLAR = 0

Bit	15	14	13	12	11	10	9	8
(0x79)	–	–	–	–	–	–	**ADC9**	**ADC8**
(0x78)	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADC1**	**ADC0**
	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

Bit	15	14	13	12	11	10	9	8
(0x79)	**ADC9**	**ADC8**	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**
(0x78)	**ADC1**	**ADC0**	–	–	–	–	–	–
	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

ADC registers (3/3)

- ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
(0x7A)	**ADEN**	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**
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

- DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0
(0x7E)	–	–	**ADC5D**	**ADC4D**	**ADC3D**	**ADC2D**	**ADC1D**	**ADC0D**
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

How to work with ADC

- Initialise the ADC:
  - Select Voltage Reference
  - Select ADC Channel
  - Select prescaler
  - Enable ADC
  - Enable ADC Interrupt (if desired)
  - Select Conversion Mode
- Getting the data:
  - Start the ADC
  - Wait until conversion is complete
  - Read the ADC Register(s)

ADC example

The example below initialises the ADC as follows:

- Resolution – 8bit (the code should read ADCH).
- Internal reference voltage.
- Channel #6.
- Free-running mode.
- Prescaler = 128. Slow-running mode.
- Interrupt enabled – the code should be placed in the ISR (ISR (ADC_vect){}).

```
ADMUX=((1<<ADLAR)|(1<<REFS0)|(1<<MUX2)|(1<<MUX1));
ADCSRA=((1<<ADEN)|(1<<ADIE));
ADCSRA|=(1<<ADPS0)|(1<<ADPS1)|(1<<ADPS2);
ADCSRA|=(1<<ADATE);
ADCSRA|=(1<<ADSC); // Start conversion
```

Notes: ADCSRA can be written in one command. The result of the first conversion after ADC was enabled or channel changed **must be scrapped**.

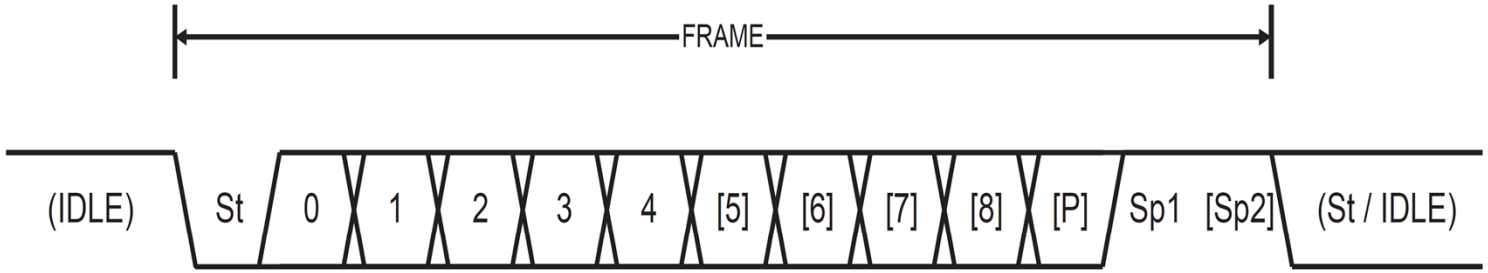
Communication peripherals

Types of communication peripherals

- Universal Asynchronous Receiver Transmitter (UART)
  - Point-to-point.
  - Signals:
    - Receive (RX).
    - Transmit (TX).
- Serial Peripheral Interface (SPI)
  - Bus with Master device.
  - Signals:
    - Clock (CLK) – from Master to all Slaves.
    - Master-Out Slave-In (MOSI) – Data from Master to Slaves.
    - Master-In Slave-Out (MISO) – Data from Slaves to Master.
    - Slave Select (/SS) – From Master to Slave. Each Slave gets its own /SS.
- Two-Wire Interface (TWI or I2C)
  - Bus with multiple Masters.
  - Signals:
    - Serial Data (SDA) – Bi-directional data line between Master and Slave.
    - Serial Clock (SCL) – Clock signal from Master to Slave. Slave can pull it down to indicate BUSY state.

UART: How it works

Figure 20-4. Frame Formats



St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit. Can be odd or even.

Sp Stop bit, always high.

IDLE No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

UART: What needs to be set

- Baud Rate. Standard rates: 2400, … 9600, …. 57600, … 115200 bits per second (UBRR).
- Frame Format:
  - Start bit (1).
  - Data bits (7, 8 or 9).
  - Parity (None, Odd, Even).
  - Stop bits (1 or 2).
- Select operation mode (TX/RX): TXEN and/or RXEN
- Enable UART specific interrupt (if desired):
  - UART Transmitter has 2 interrupts:
    - Data Register Empty (UDRE): indicates transmit buffer ready to receive new data.
    - Transmit Complete (TXC): triggers when entire frame shifted out and no new data present in buffer.
  - UART Receiver
    - Receive Complete (RXC): triggers when unread data present in receive buffer.
- Read/Write UART data (UDR)

UART: Registers

- UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
Read/Write	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	1	0	0	0	0	0

- UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Initial Value	0	0	0	0	0	0	0	0

- UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
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	1	1	0

UART: Example (1/2)

- Definitions:

```
#define F_CPU 16000000UL
#define BAUD 2400UL
#define UBRR ((F_CPU)/((BAUD)*(16UL))-1)
```

- In the main() initialisation section:

```
UBRR0H = (uint8_t)((UBRR)>>8);
UBRR0L = (uint8_t)UBRR;
UCSR0B|=(1<<RXCIE0)|(1<<RXEN0); // enable receiver and its interrupt
UCSR0B|=(1<<TXCIE0)|(1<<TXEN0); // enable transmitter and its interrupt
UCSR0C=(3<<UCSZ00); // 8-bit, 1 stop, no parity
```

Note: PC's and microcontroller's UARTs **must have the same settings**!!!

UART: Example (2/2)

- RX ISR (checking for frame, parity or overrun error):

```
ISR (USART_RX_vect) {
        RX_buff=UDR0; //MUST read UDR in order to reset the RX interrupt flag
        if (!((UCSR0A&(1<<FE0))|(UCSR0A&(1<<DOR0))|(UCSR0A&(1<<UPE0))))
flags.RX_error=0;
        else flags.RX_error=1; //main() should check this flag
}
```

- TX ISR:

```
ISR (USART_TX_vect) {
  msg++;
  if (*msg) UDR0=*msg;
}
```

Sensors

Available sensors

- **Sharp GP2Y0A02YK0F Analog Distance Sensor**
  - Maximum range: 150 cm
  - Minimum range: 20 cm
  - Sampling rate: 21 Hz
  - Minimum operating voltage: 4.5 V
  - Maximum operating voltage: 5.5 V
  - Supply current: 33 mA
  - Output type: analog voltage
  - Output voltage differential: 2.05 V
  - https://www.pololu.com/product/1137

- **Ultrasonic Range Finder - LV-MaxSonar-EZ1**
  - 42kHz Ultrasonic sensor
  - Operates from 2.5-5.5V
  - Low 2mA supply current
  - 20Hz reading rate
  - RS232 Serial Output - 9600bps
  - Analog Output - 10mV/inch
  - PWM Output - 147uS/inch
  - https://www.sparkfun.com/products/639

Expansion connector pinout

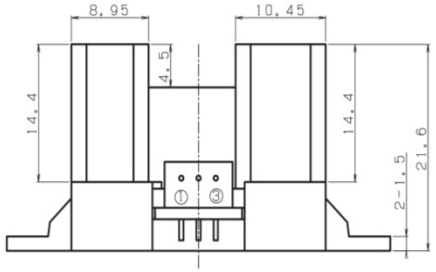
PC4	**PD4**	**PD3**	**+5V**
PC3	**PC5**	**PD2**	**GND**

Front view

PC3 – ADC3 – available for analog sensor
PC4 – SDA used for IMU TWI
PC5 – SCL used for IMU TWI
PD2 – INT0/GPIO – available for digital sensor (PWM) or s/w UART
PD3 – INT1/GPIO – available for digital sensor (PWM) or s/w UART
PD4 – GPIO – available for digital sensor (PWM) or s/w UART

How to use IR sensor

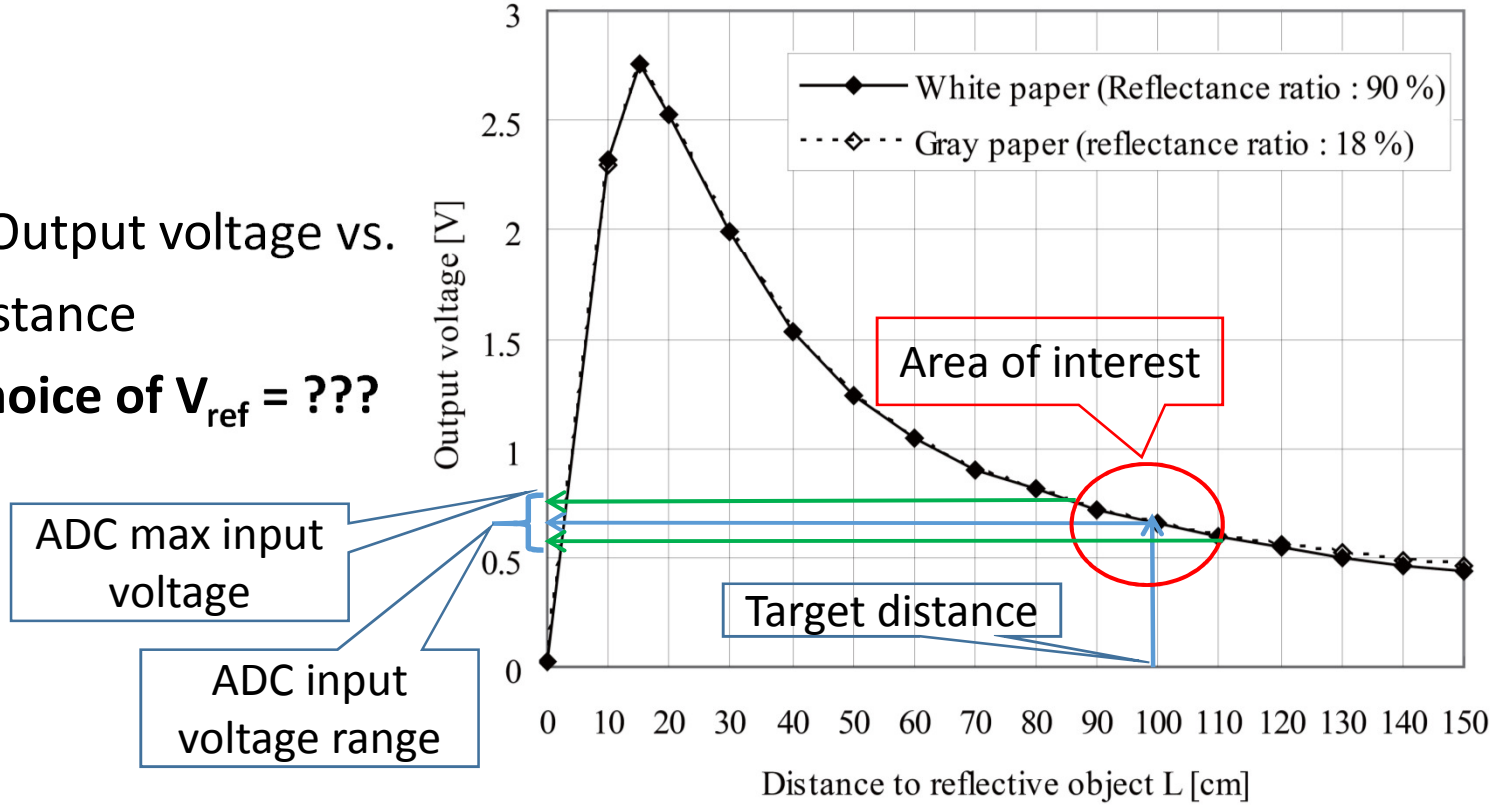
- Must use analog (ADC) input
- Pinout:



	Terminal	Symbol
①	Output terminal voltage	V_O
②	Ground	GND
③	Supply voltage	V_{CC}

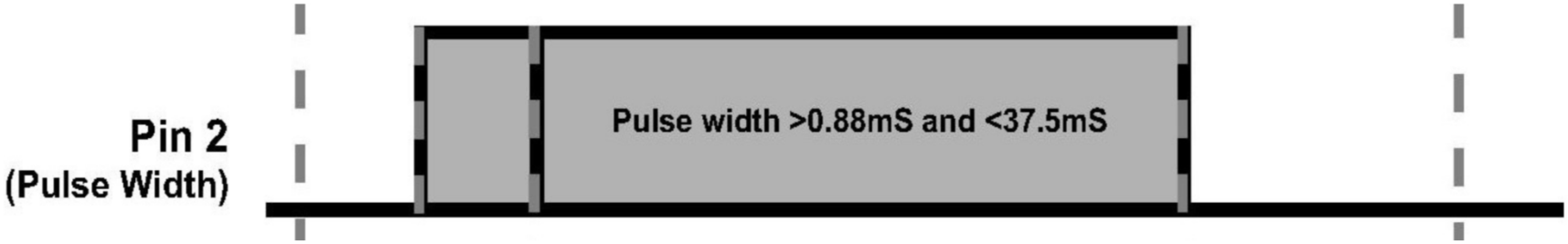
- Output voltage vs. distance

Choice of V_{ref} = ???



How to use Ultrasonic sensor

- Use Analog Output – connect to ADC **or**
- PWM Output – connect to INT0 or INT1 and use timer to measure the pulse length:



- Hints:
  - We already have Timer1, which overflows every 20ms making 2500 ticks, i.e. 8us per tick.
  - INT0 and INT1 can be set to trigger on both edges.
  - GPIO pin value can be read any time, no matter what function of the pin is.

How to use Inertial Measurement Unit (IMU)

- IMU=Accelerometer+Gyroscope+Magnetometer
- Hardware setup
  - IMU **must be powered off 3.3V power source,** e.g. https://www.digikey.ca/product-detail/en/diodes-incorporated/AP7381-33V-A/AP7381-33V-ADICT-ND/7914843
  - IMU connects to the uCU by TWI interface
  - uCU is powered off 5V power source, thus a **level shifter is a must,** e.g. https://www.sparkfun.com/products/15439

- Software
  - TWI initialisation:

    ```
    TWSR=0; // no prescaler
    TWBR=(uint8_t)(((F_CPU/SCL_CLOCK)-16)>>1);
    ```

  - Use the library available on Moodle (twi_290.c):
    - Start accelerometer, magnetometer and gyroscope.
    - Read accelerometer, magnetometer and gyroscope data (XYZ for each sensor).
    - For troubleshooting and calibration use UART to send data to a PC.

Programming Environment

Settings for Arduino & serial port

- **Arduino IDE is available in H849, H917, H929, H967**
- **Arduino IDE:**
  - Tools-> Board: "Arduino Duemilanove or Diecimila"
  - Tools->Processor: "Atmega328"
  - Tools->Port: select an appropriate port from the list
- **Serial port (must be logged in as administrator)**
  - Control Panel->Device manager->Ports (COM & LPT) (Note: the remote control MUST be connected at this point and the system should finish installing the drivers)
  - Double-click on "USB serial port (COMx)"
  - Select tab "Port settings":
    - Bits per second: **9600**
    - Data bits: 8
    - Parity: None
    - Stop bits: 1
    - Flow control: **HARDWARE**
  - Click on "Advanced" button. The following "Miscellaneous Options" must be selected:
    - Serial Enumerator
    - **Serial Printer**
    - **Set RTS on close**

Atmel Studio

- Why I'm switching over from the awesome Arduino IDE to Atmel Studio:

https://www.youtube.com/watch?v=648Tx5N9Zoc

  - Professional environment
  - Code size
  - Full control over what exactly the code does
  - Simulator, debugger, etc.
  - … and FREE

- **Atmel Studio 7**

https://www.microchip.com/mplab/avr-support/atmel-studio-7

- **Atmel Studio 7 and arduino bootloader**

https://www.avrfreaks.net/forum/atmel-studio-7-and-arduino-bootloader

REFERENCES

- Datasheet “8-bit Atmel Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash ATmega48A ATmega48PA ATmega88A ATmega88PA ATmega168A ATmega168PA ATmega328 ATmega328P”, Rev. 8271D–AVR–05/11
- Bit manipulation (AKA "Programming 101")

http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=37871

- AVR Timers – PWM Mode – Part I
- http://maxembedded.com/2011/08/avr-timers-pwm-mode-part-i/