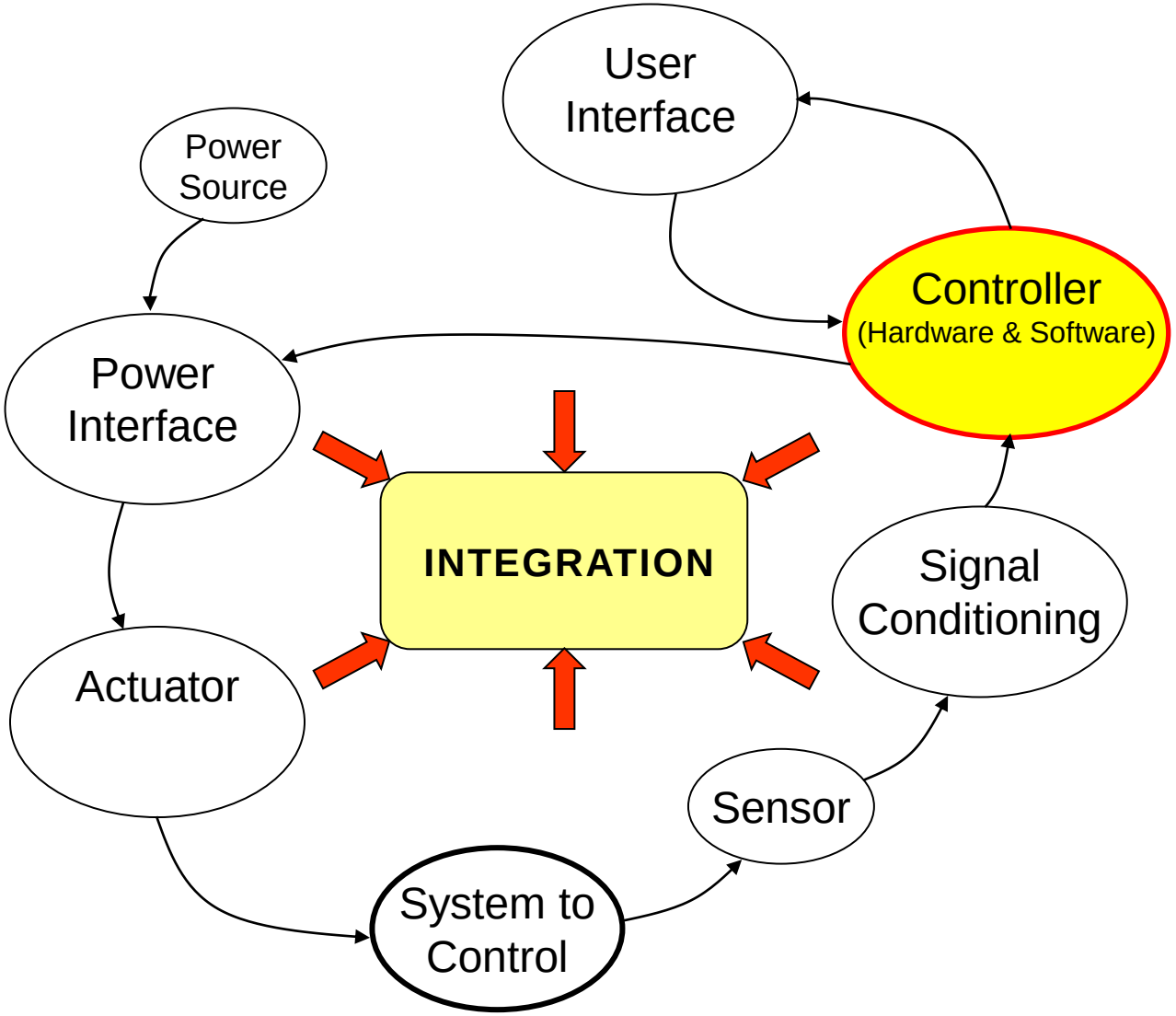
PC Interfacing

Microcontroller

Dr. Montassar Aidi Sharif

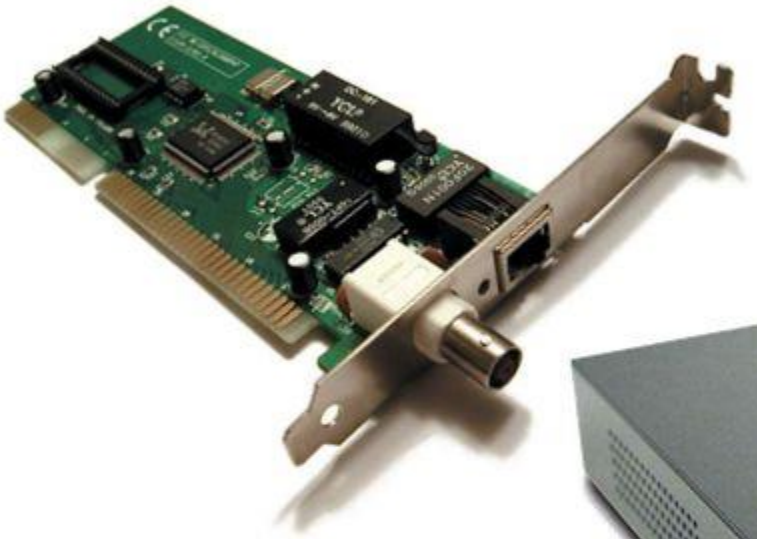
Mechatronics Concept Map

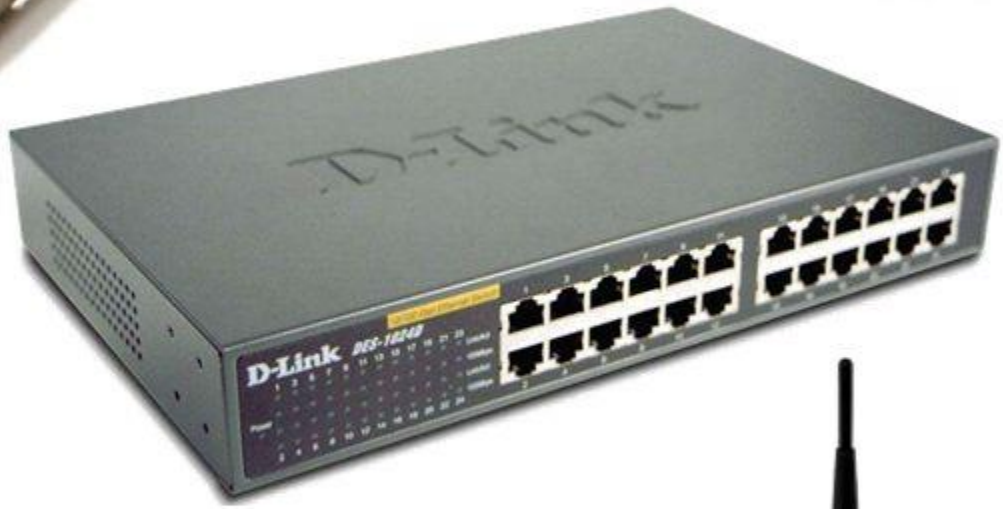


Why do we need to learn Microcontroller

cell phone

COMMUNICATION DEVICES



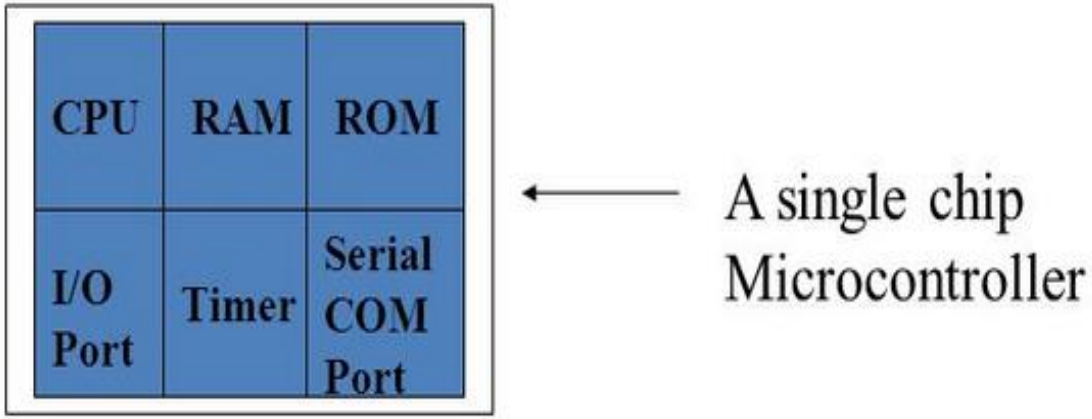


air conditioner, washer/dryer

Toys

Then What is a Microcontroller ?

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example：Motorola's 6811, Intel's 8051, PIC 16X

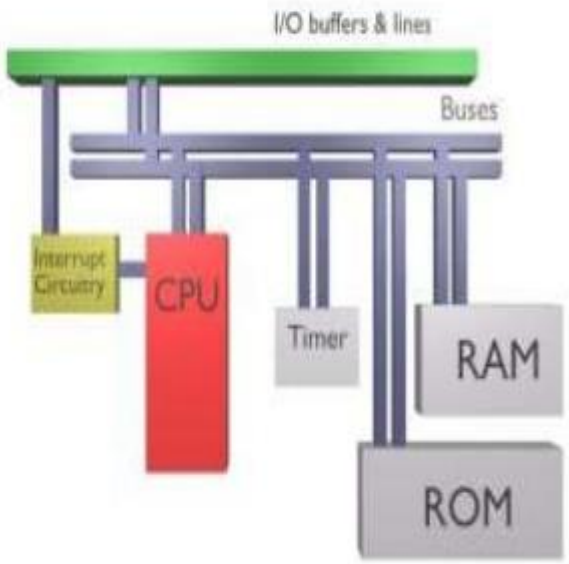


Component of microcontroller

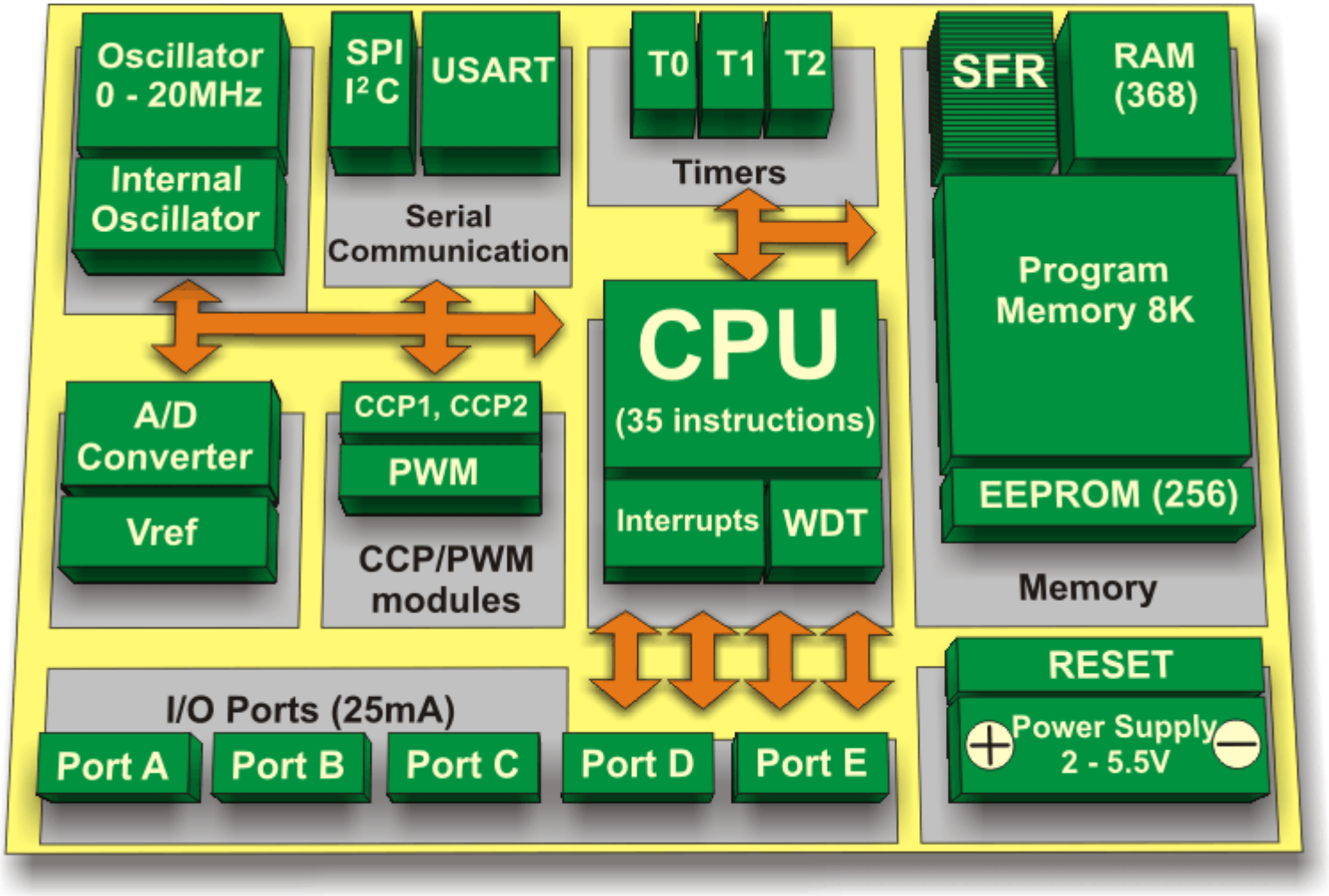
Components

- A microcontroller has seven main components:

- Central processing unit (CPU)
- ROM
- RAM
- Input and Output
- Timer
- Interrupt circuitry
- Buses



Component of microcontroller



Microprocessor

VS

Microcontroller

Microprocessor vs. Microcontroller

Types of Microcontrollers

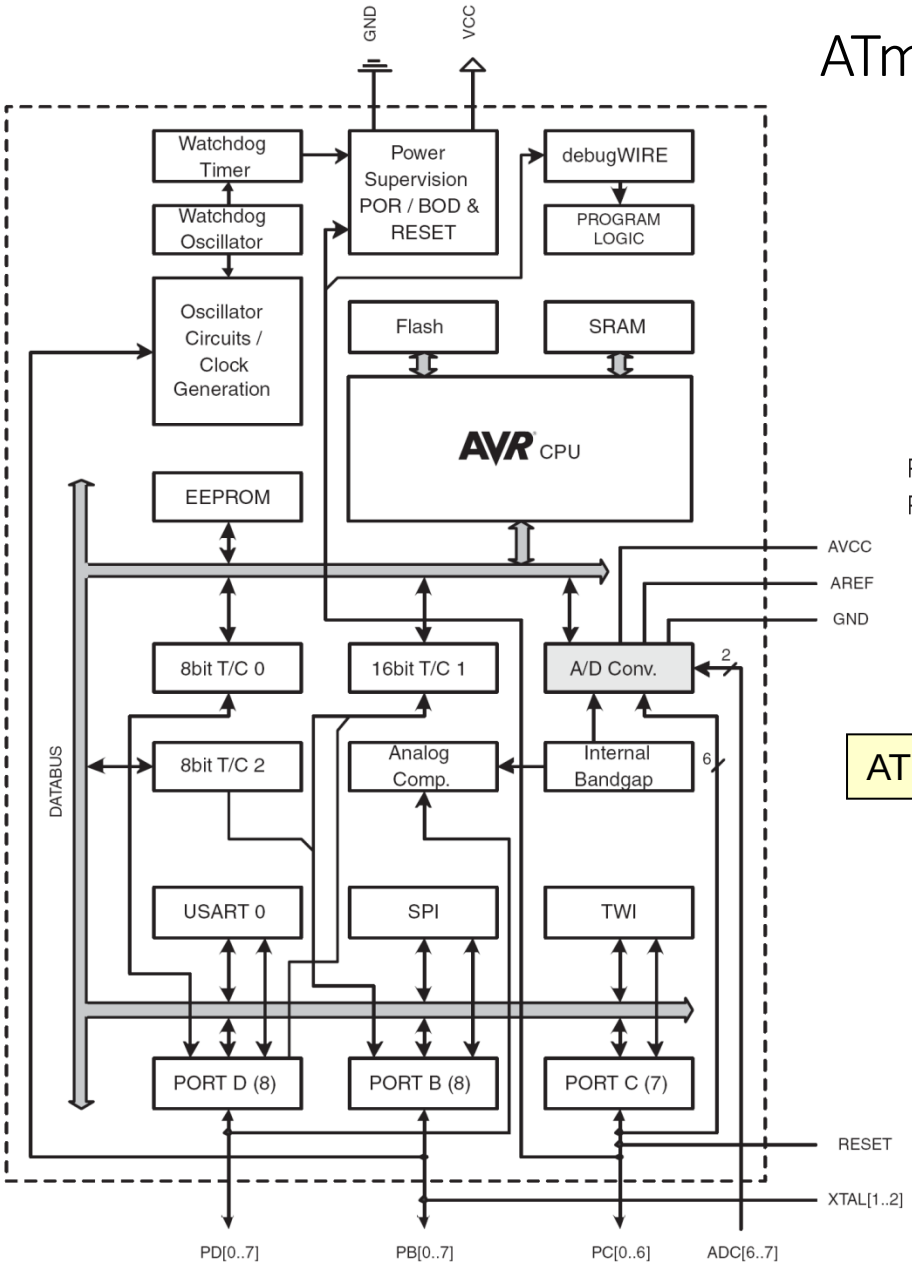
Advantage of **microcontroller**

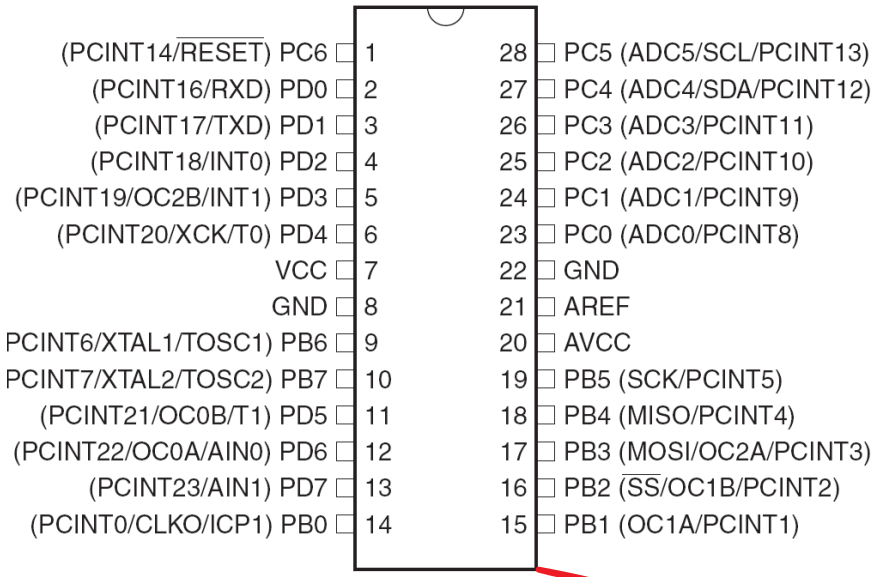
30 to 100 faster than other microcontroller

smaller size

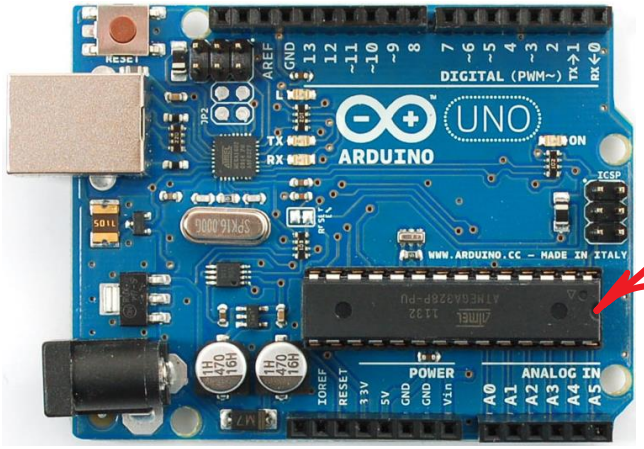
easy to programmed

ATmega328 Internal Architecture





ATmega328 data sheet pp. 2, 5



http://www.adafruit.com/index.php?main_page=popup_image&pID=50

ATmega328 Features

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1K Bytes EEPROM
  - 512/1K/1K/2K Bytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5.V, 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

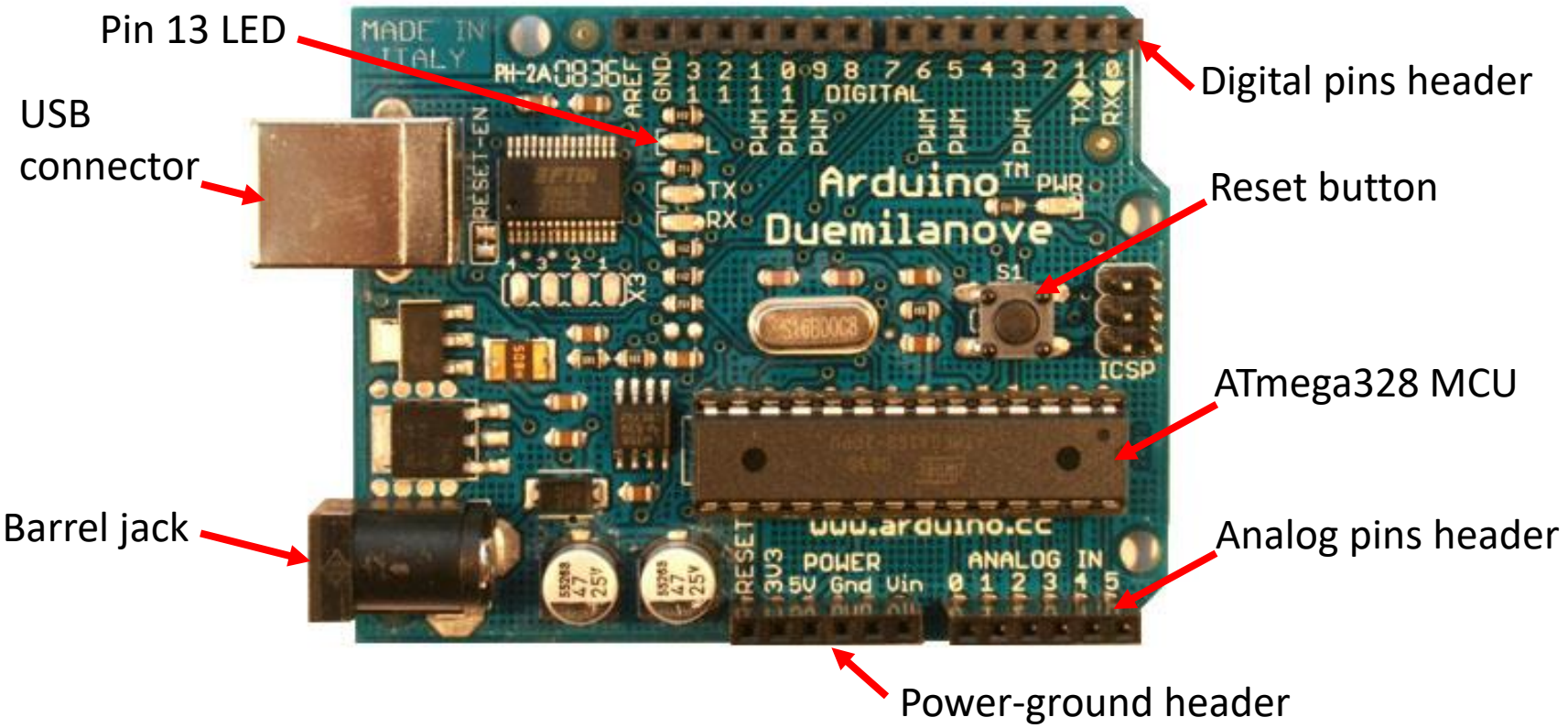
ATmega328 data sheet p. 1

http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf

Arduino Duemilanove

http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove

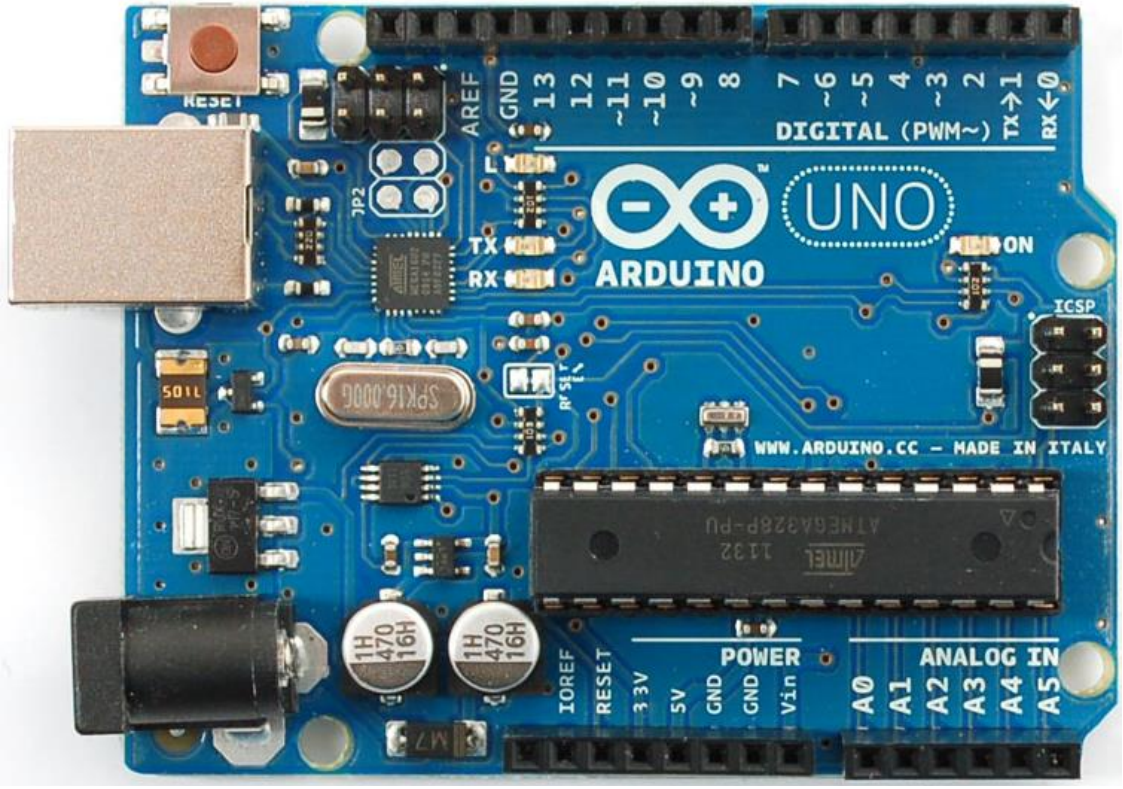
See the handout: **Arduino_ATmega328_pin_mapping_and_schematic**



http://arduino.cc/en/uploads/Main/ArduinoDuemilanove.jpg

Arduino Uno R3

ATmega16u2 replaces FT232RL for USB-serial communication



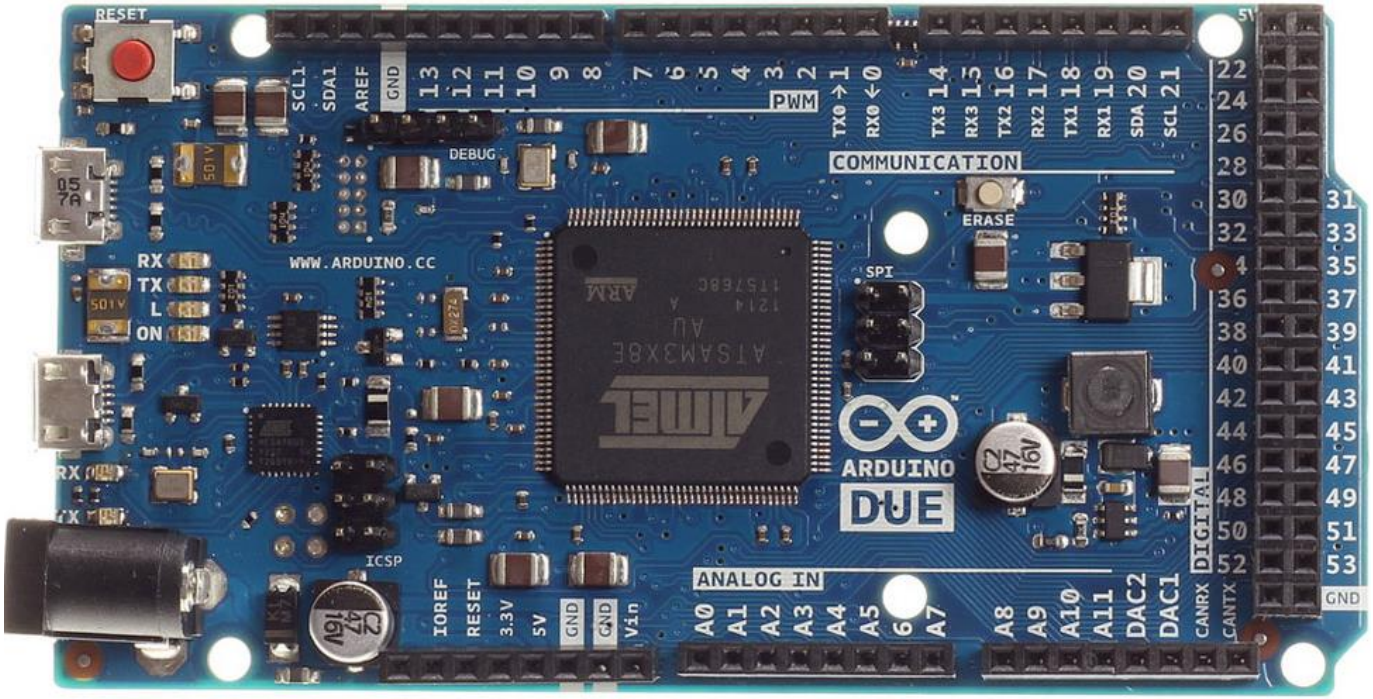
http://www.adafruit.com/index.php?main_page=popup_image&pID=50

See: http://learn.adafruit.com/arduino-tips-tricks-and-techniques/arduino-uno-faq

Arduino Due

Note: **3.3 V** !!

Atmel SAM3X8E processor (32 bit ARM Cortex M3 architecture, 84MHz)



http://www.adafruit.com/index.php?main_page=popup_image&pID=1076

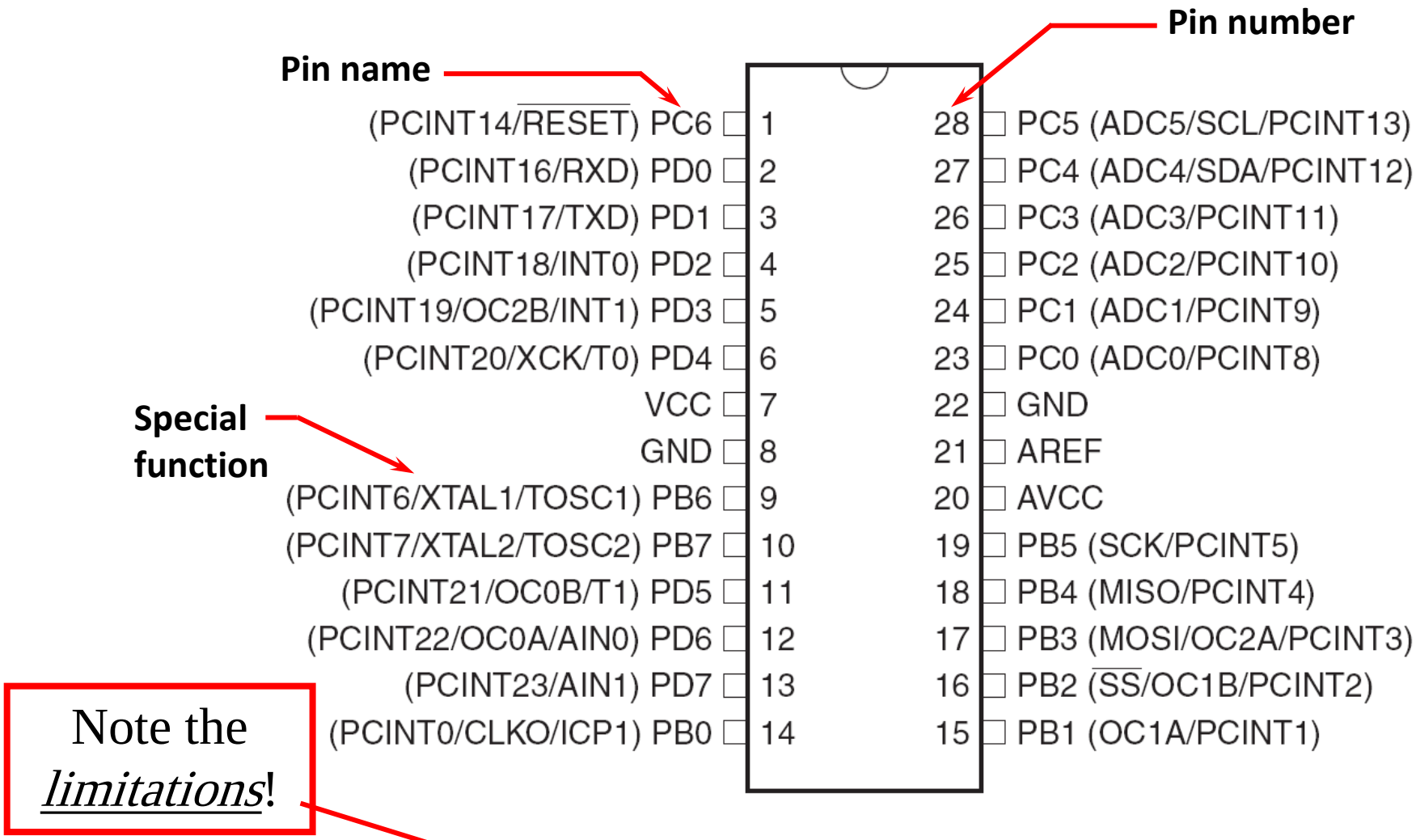
See: http://arduino.cc/en/Main/ArduinoBoardDue

Arduino Duemilanove/Uno Features

Microcontroller	ATmega168/328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz

http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove

ATmega328 Microcontroller



Note the *limitations*!

p. 316

Source:http://www.atmel.com/dyn/products/product_card.asp?PN=ATmega328P

Port Pin Data Directionality

- Input
  - When you want to take information from the external world (sensors) ***into*** the MCU
- Output
  - When you want to change the state of something ***out*****side** the MCU (turn a motor on or off, etc.)
- Pins default to input direction on power-up or reset
- Your program can set or change the directionality of a pin at any time

Setting the Pin Data Direction

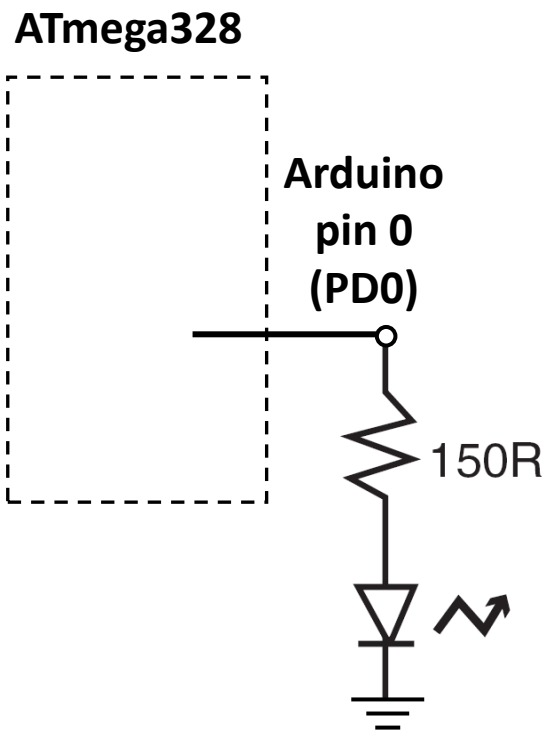
- Arduino
  - pinMode(*pin_no.*, *dir*)
    - Ex. Make Arduino pin 3 (PD3) an *output*
      - `pinMode(3, OUTPUT);`
      - `pinMode(PIN_D3, OUTPUT); // with me106.h`
  - Note: one pin at a time
    - Suppose you wanted Arduino pins 3, 5, and 7 (PD3, PD5, and PD7) to be outputs?
    - Is there a way to make them all outputs at the same time?
      - Yes! Answer coming later…

Pin Voltages

- Microcontrollers are fundamentally ***digital*** devices. For digital IO pins:
  - Information is ‘coded’ in two discrete states:
    - HIGH or LOW (logic: 1 or 0)
    - Voltages
      - TTL
        - 5 V (for HIGH)
        - 0 V (for LOW)
      - 3.3 V CMOS
        - 3.3 V (for HIGH)
        - 0 V (for LOW)

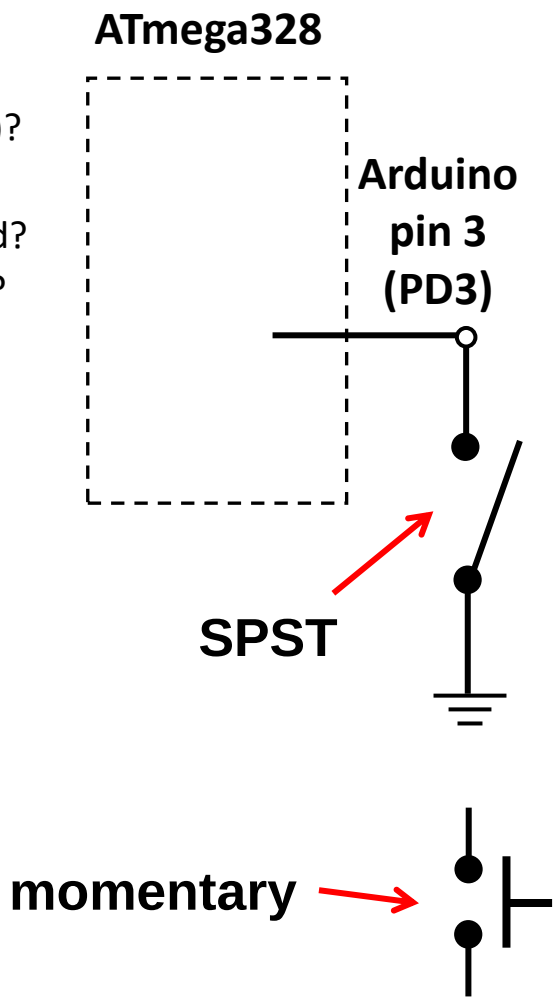
Pin Used as an Output

- Turn on an LED, which is connected to pin Arduino pin 0 (PD0) (note the resistor!)
  - What should the data direction be for pin 0 (PD0)?
    - `pinMode(____, ____);`
  - Turn on the LED
    - **`digitalWrite(PIN_LED,HIGH);`**
  - Turn off the LED
    - **`digitalWrite(PIN_LED,LOW);`**



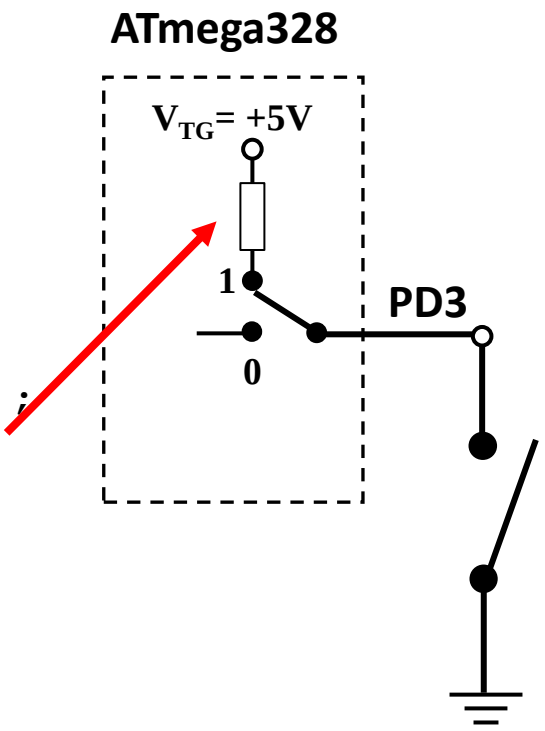
Pins as Inputs and Pull-up Resistors - 1

- Using a switch as a sensor
  - Ex. Seat belt sensor
  - Detect the switch ***state***
    - What should the data direction be for Arduino pin 3 (PD3)?
    - `pinMode(____, ____);`
    - What will the voltage be on PD3 when the switch is closed?
    - What will the voltage be on PD3 when the switch is open?
      - Indeterminate!



Pins as Inputs and Pull-up Resistors - 2

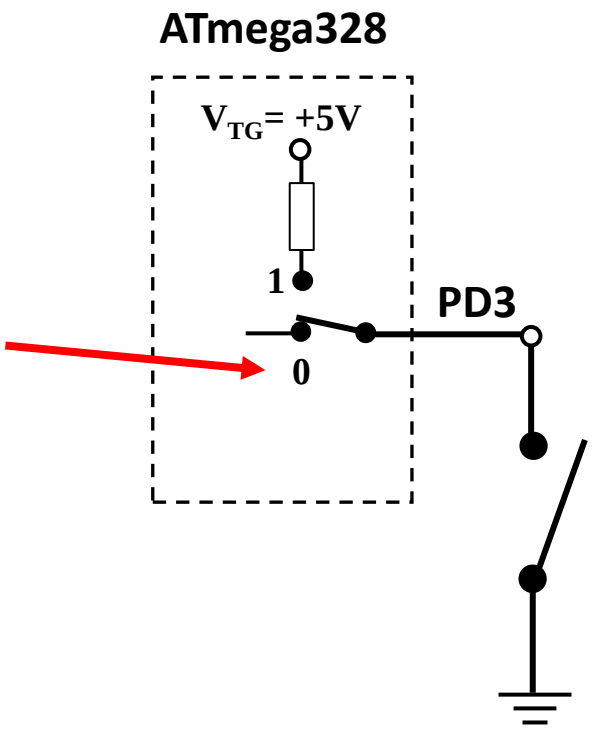
- Switch as a sensor, cont.
  - Make the voltage on the pin *determinate* by turning on the pull-up resistor for PD3
    - Assuming PD3 is an input:
      - `digitalWrite(PIN_SWITCH,HIGH);` turns on the "pull-up" resistor
      - `pinMode(PIN_SWITCH,INPUT_PULLUP);`
    - What will the voltage on PD3 be when the switch is open?
      - V_{TG}
    - What will the voltage on PD3 be when the switch is closed?



Pins as Inputs and Pull-up Resistors - 3

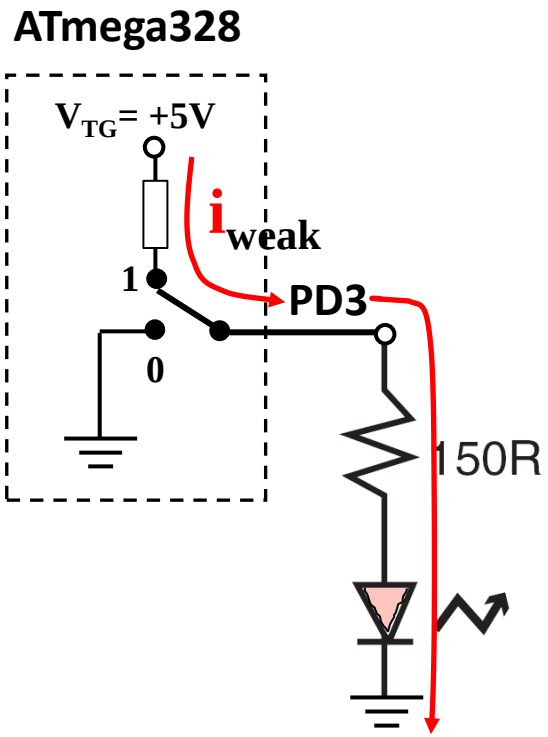
- Switch as a sensor, cont.
  - To turn *off* the pull-up resistor
    - Assuming PD3 is an input:

    `digitalWrite(PIN_SWITCH,LOW);`
    turns the “pull-up” resistor off



Pins as Inputs and Pull-up Resistors - 4

- Possibility of ‘weak drive’ when pull-up resistor is turned on
  - Pin set as an *input* with a pull-up resistor turned on can source a small current
    - Remember this!



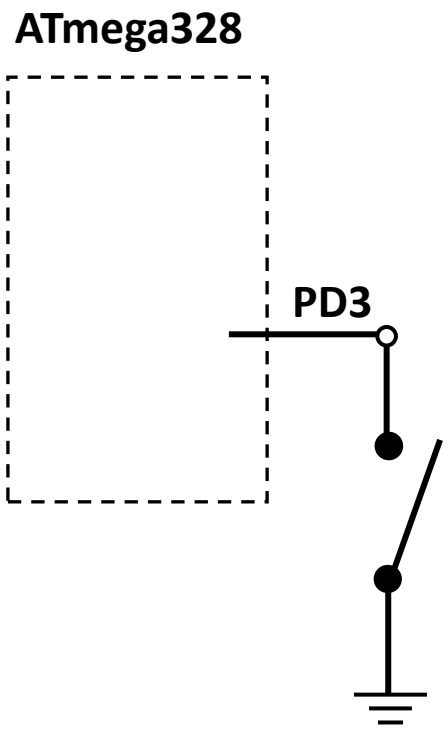
Structure of an Arduino Program

- An arduino program == '**sketch**'
  - Must have:
    - setup()
    - loop()
  - setup()
    - configures pin modes and registers
  - loop()
    - runs the main body of the program forever
      - like while(1) {…}
  - Where is main() ?
    - Arduino simplifies things
    - Does things for you

```
/* Blink  -  turns on an LED for DELAY_ON msec,
then off for DELAY_OFF msec, and repeats
BJ Furman rev. 1.1   Last rev: 22JAN2011
*/
#define LED_PIN  13    // LED on digital pin 13
#define DELAY_ON  1000
#define DELAY_OFF 1000

void setup()
{
  // initialize the digital pin as an output:
  pinMode(LED_PIN, OUTPUT);
}

// loop() method runs forever,
// as long as the Arduino has power

void loop()
{
  digitalWrite(LED_PIN, HIGH);    // set the LED on
  delay(DELAY_ON);  // wait for DELAY_ON msec
  digitalWrite(LED_PIN, LOW);   // set the LED off
  delay(DELAY_OFF);  // wait for DELAY_OFF msec
}
```

Digital IO – Practice 1

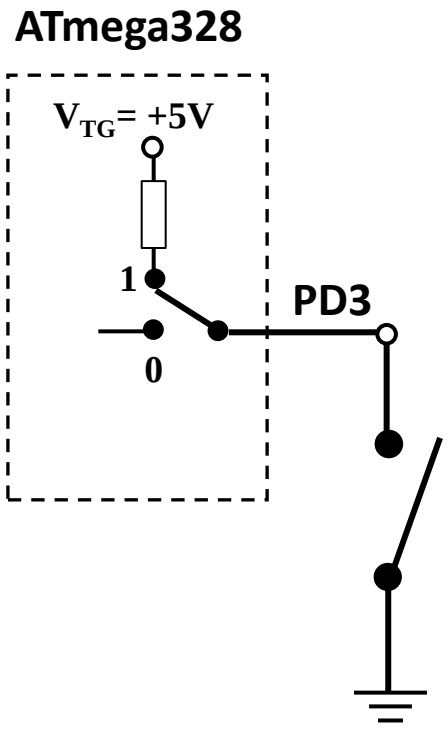
- ‘Reading a pin’
  - Write some lines of C code for the Arduino to determine a course of action if the seat belt has been latched (switch closed).
    - If latched, the ignition should be enabled through a call to a function ig_enable().
    - If not latched, the ignition should be disabled through a call to a function ig_disable()
  - Write pseudocode first



Digital IO – Practice 1 Pseudocode

- ‘Reading a pin’
  - Pseudocode:

```
Set up PD3 as an input
Turn on PD3 pull-up resistor
Read voltage on Arduino pin 3 (PIN_D3)
IF PIN_D3 voltage is LOW (latched), THEN
        call function ig_enable()
ELSE
        call function ig_disable()
```



Digital IO – Practice 1 Code

- ‘Reading a pin’
  - Pseudocode:

    Set up PD3 as an input

    Turn on PD3 pull-up resistor

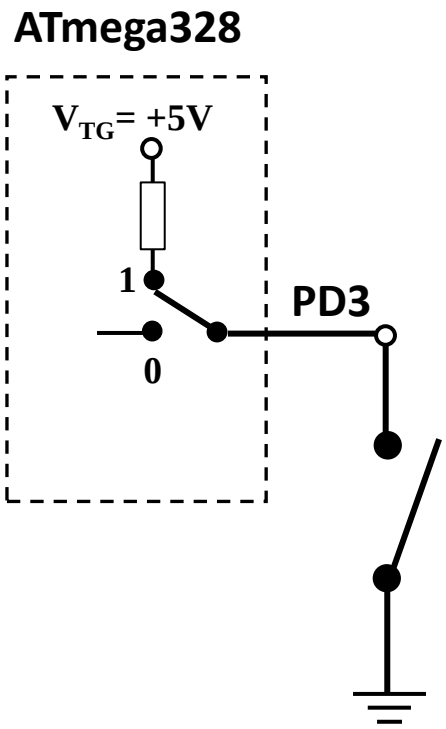
    Read voltage on Arduino pin 3 (PIN_D3)

    IF PIN_D3 voltage is LOW (latched), THEN

        call function ig_enable()

    ELSE

        call function ig_disable()

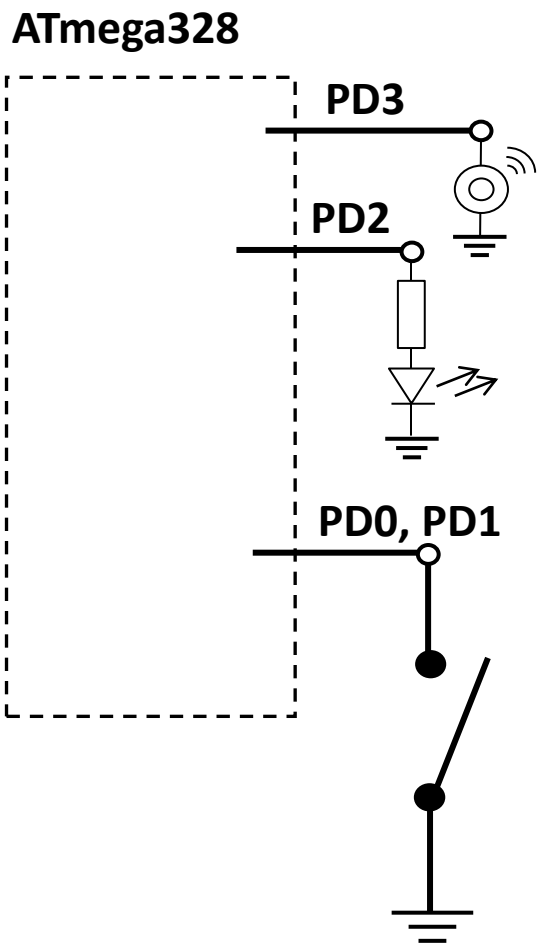


One way →
(snippet, not full program)

```
#define PIN_SWITCH 3
#define LATCHED LOW
pinMode(PIN_SWITCH,INPUT_PULLUP);
belt_state = digitalRead(PIN_SWITCH);
if (belt_state == LATCHED)
{ ig_enable(); }
else
{ ig_disabled(); }
```

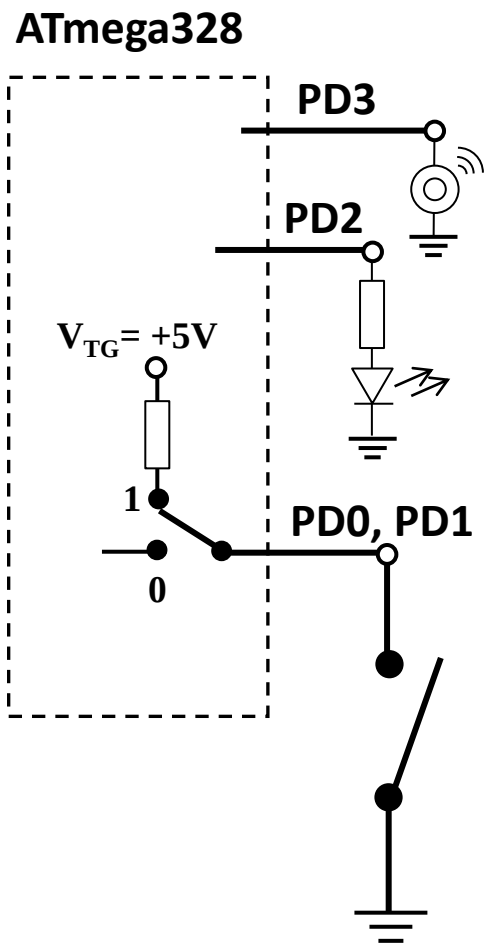
Digital IO – Practice 2

- ‘Reading from and writing to a pin’
  - Write some lines of C code for the Arduino to turn on a lamp (PD2) and buzzer (PD3) if the key is in the ignition (PD0 closed), but seat belt is not latched (PD1 open)
    - (diagram shows only one of the two switches, but both are similar)
  - Pseudocode first



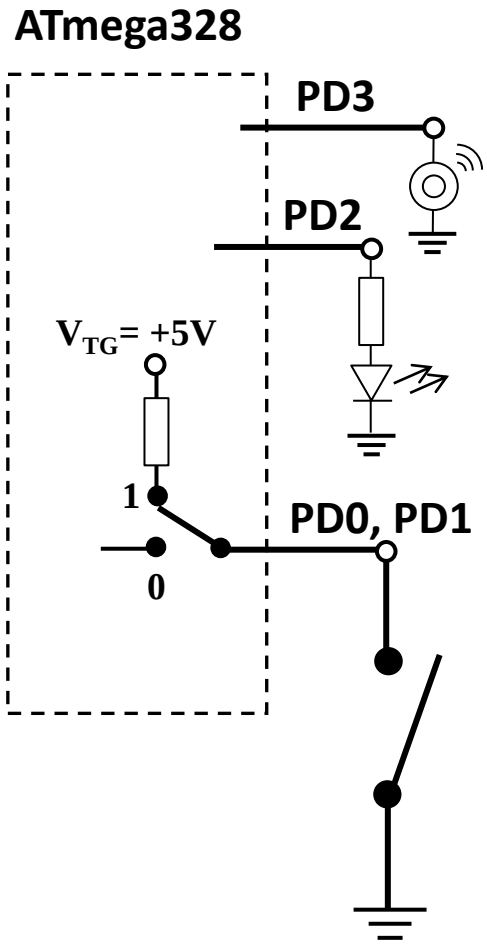
Digital IO – Practice 2 Pseudocode

- Pseudocode:
  - Set up data direction of pins
    - Make PD0 and PD1 inputs
    - Turn on pull up resistors for PD0 and PD1
    - Make PD2 and PD3 outputs
  - Loop forever
    - IF key is in ignition THEN
      - IF belt is latched, THEN
        - Turn off buzzer
        - Turn off lamp
      - ELSE
        - Turn on lamp
        - Turn on buzzer
    - ELSE
      - Turn off buzzer
      - Turn off lamp



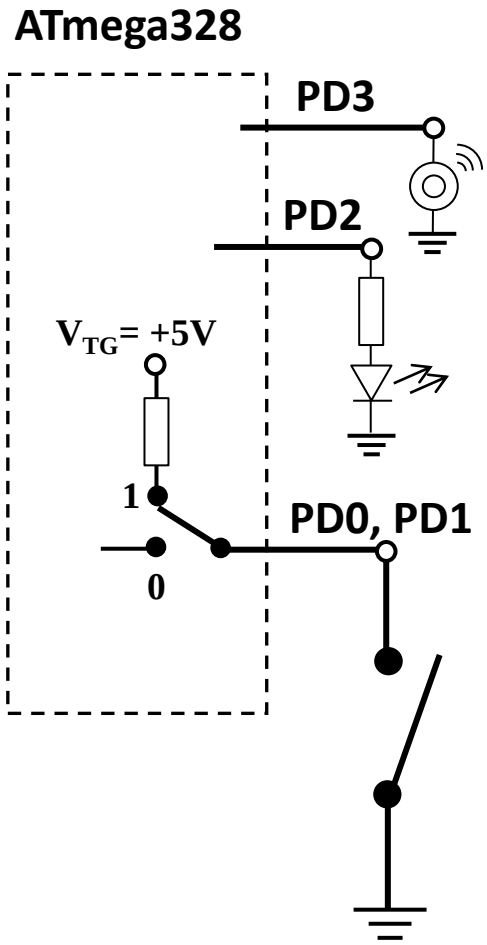
Digital IO – Practice 2 (Arduino style code)

```
#define PIN_IGNITION  0
#define PIN_SEATBELT  1
#define PIN_LED  2
#define PIN_BUZZER  3
#define SEATBELT_LATCHED  LOW
#define KEY_IN_IGNITION LOW
#define LED_ON  HIGH
#define LED_OFF  LOW
#define BUZZER_ON  HIGH
#define BUZZER_OFF  LOW
void setup()
{
  pinMode(PIN_IGNITION, INPUT_PULLUP);  // key switch
  pinMode(PIN_SEATBELT, INPUT_PULLUP); // belt latch switch
  pinMode(PIN_LED, OUTPUT);  // lamp
  pinMode(PIN_BUZZER, OUTPUT);  // buzzer
}
void loop()
{ /* see next page for code */}
```



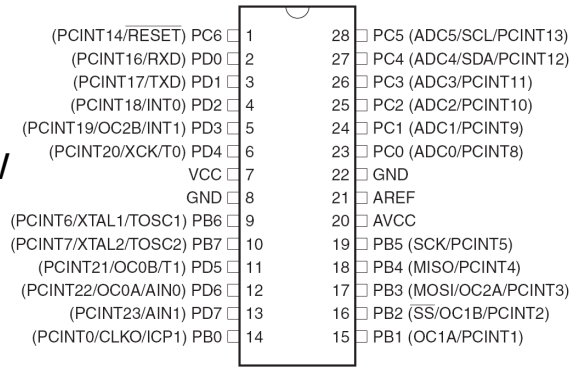
Digital IO – Practice 2 (Arduino style code)

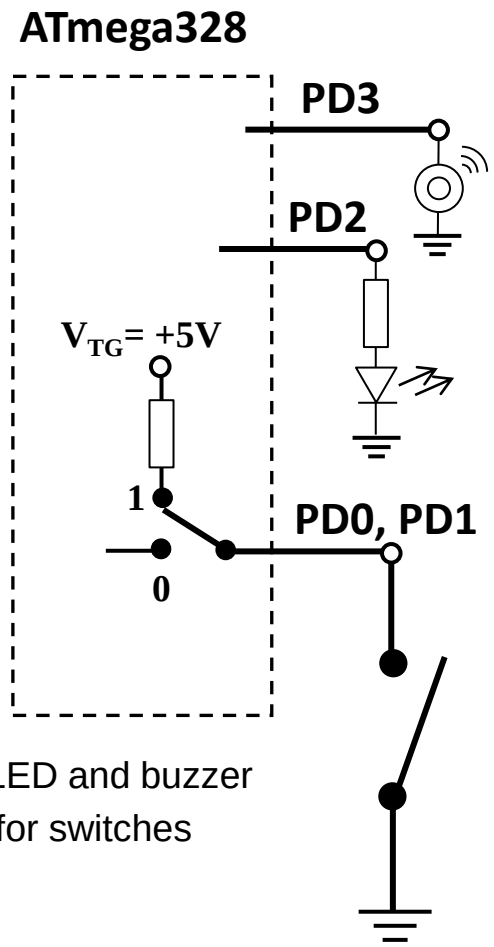
```
/* see previous page for code before loop() */
void loop()
{
  int key_state = digitalRead(PIN_IGNITION);
  int belt_state = digitalRead(PIN_SEATBELT);
  if (key_state == KEY_IN_IGNITION)
   {
     if (belt_state == SEATBELT_LATCHED)
      {
        digitalWrite(PIN_BUZZER, BUZZER_OFF);
        digitalWrite(PIN_LED, LED_OFF);
      }
        else  // key is in ignition, but seatbelt NOT latched
         {
           digitalWrite(PIN_BUZZER, BUZZER_ON);
           digitalWrite(PIN_LED, LED_ON);
         }
    else  // key is NOT in ignition
      {
        digitalWrite(PIN_BUZZER, BUZZER_OFF);
        digitalWrite(PIN_LED, LED_OFF);
      }
   }
}
```



Digital IO – Practice 3 (Register style code)

```
/* NOTE: #defines use predefined PORT pin numbers  for ATmega328 */
#define PIN_IGNITION  PD0
#define PIN_SEATBELT  PD1
#define PIN_LED  PD2
#define PIN_BUZZER  PD3
#define SEATBELT_LATCHED  LOW
#define KEY_IN_IGNITION LOW
#define LED_ON  HIGH
#define LED_OFF  LOW
#define BUZZER_ON  HIGH
#define BUZZER_OFF  LOW
#define  _BIT_MASK( bit )  ( 1 << (bit) )  // same as _BV( bit)
void setup()
{
    PORTD = 0;  // all  PORTD pullups off
    DDRD  = _BIT_MASK(PIN_LED)  | _BIT_MASK(PIN_BUZZER);  // LED and buzzer
    PORTD | = _BV(PIN_IGNITION) | _BV(PIN_SEATBELT);  // pullups for switches
}
```

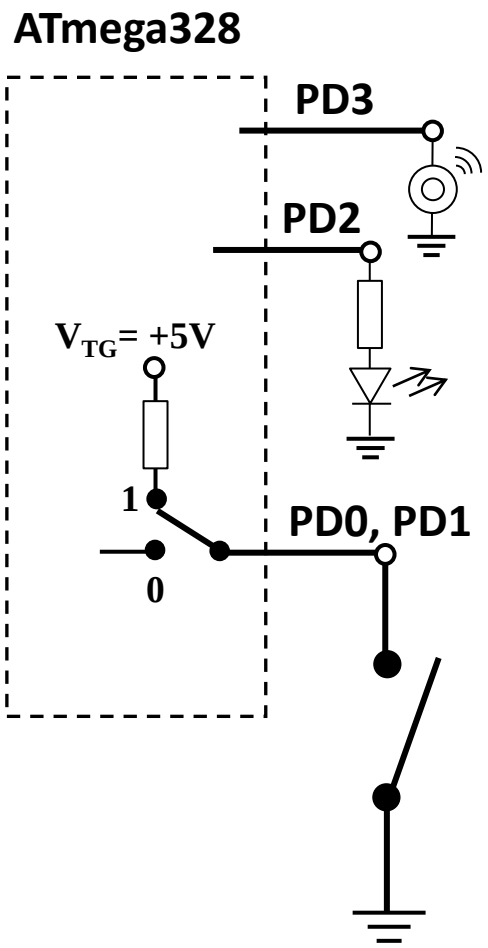




/* See next page for loop() code */

Digital IO – Practice 3 (Register style code)

```
/* see previous page for setup() code */
void loop()
{
    uint8_t current_PORTD_state, key_state, belt_state;
    current_PORTD_state = PIND;  // snapshot of PORTD pins
    key_state = current_PORTD_state  &  _BV(PIN_IGNITION);
    belt_state = current_PORTD_state  &  _BV(PIN_SEATBELT);
    if (key_state  == KEY_IN_IGNITION)
    {
        if (belt_state == SEATBELT_LATCHED)
        {
          PORTD &=  ~( _BV(PIN_LED)  |  _BV(PIN_BUZZER) );
        }
        else
        {
          PORTD |=  ( _BV(PIN_LED)  |  _BV(PIN_BUZZER) );
        }
    }
    else
    {
      PORTD  & = ~( _BV(PIN_LED)  |  _BV(PIN_BUZZER) );
    }
}
```



Summary

- Data direction
  - Input is default, but okay to set explictly
  - Output
    - Arduino style: pinMode(*pin_no, mode*)
    - Alternate: Set bits in DDRx
- Pull-up resistors
  - Pin must be an input
    - Arduino style: digitalWrite(*pin_no, state*)
    - Alternate style: Set bits in PORTx

Summary, cont.

- Read digital state of a pin
  - Arduino style: digitalRead(*pin_no*)
  - 'Register-style': need to form a bit mask and use it to 'single-out' the bit of interest
- Write to a pin (assuming it is an output)
  - Arduino style: digitalWrite(*pin_no, state*)
  - 'Register-style': use a bit mask and bit manipulation techniques to set or clear only the bits of interest

Analog to Digital Converters

What is Analog to Digital Converters (ADC)?

(ADC) are an electronic integrated circuit (IC) which transforms a signal from analog to digital form.

Why ADC is needed?

All microprocessors works with digital signal so ADC Provides a link between the analog and the digital worlds of signal processing.

What is ADC classification ?

1) The first group includes SAR, and flash-type converters.

SAR (Successive Approximation Register)

2) The second group includes integrator and voltage to frequency converters.

SAR ADC: The SAR ADC was the first converter to go mainstream. Over time, this converter topology appeared across a variety of applications, including process control, medical, and early digital audio systems. These applications benefit from the SAR ADC's output conversion ranges of 8 bits to 20 bits. However, the SAR ADC's claim to fame is that it captures a snapshot of the analog input signal, using multiple signal snapshots to paint a picture over time. Successive Approximation Register (SAR) Analog-to-Digital Converters (ADCs) are a great choice when you need low power consumption and superior AC and DC performance in your analog-to-digital conversion application.

VFC ADC: Since an analog to digital converter ends up producing a number, wouldn't it be convenient if we could simply count something and have that something proportional to the potential being measured? A voltage to frequency converter ADC does just that. A voltage is integrated until it reaches some pre-set threshold, at which point a comparator trips, doing 2 things:

1) short circuit the capacitor, resetting the integrator to zero.
2) produce a pulse that is counted.

V to F converters integrate noise, and so are useful under circumstances similar to dual slope units. V to F ADCs are precise, accurate, simple, and inexpensive. The precision is directly proportional to the time over which counting occurs, and inversely proportional to the time required to integrate a single count.

What is the tradeoff between ADC groups?

Accuracy

Speed

What is the uses of ADC (successive-approximation and the Integrator)?

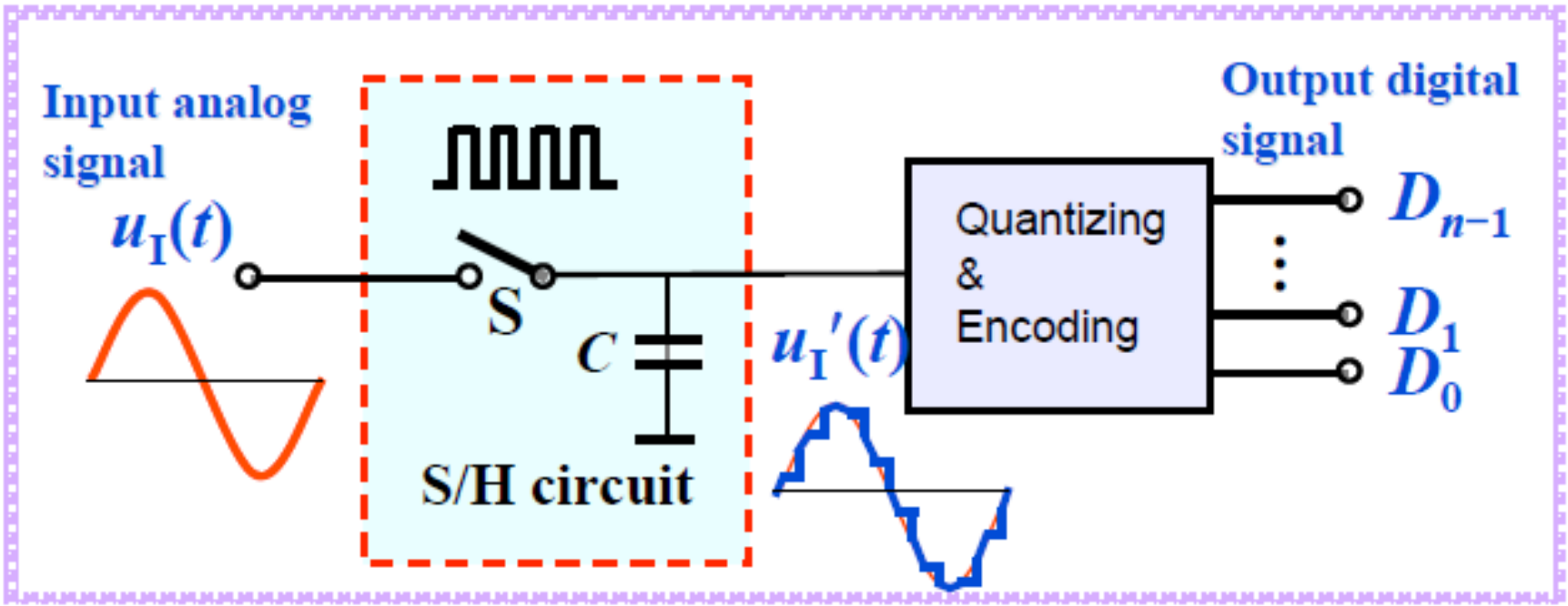
successive-approximation is used as data loggers

The integrator types are used as digital meter

What is ADC process?

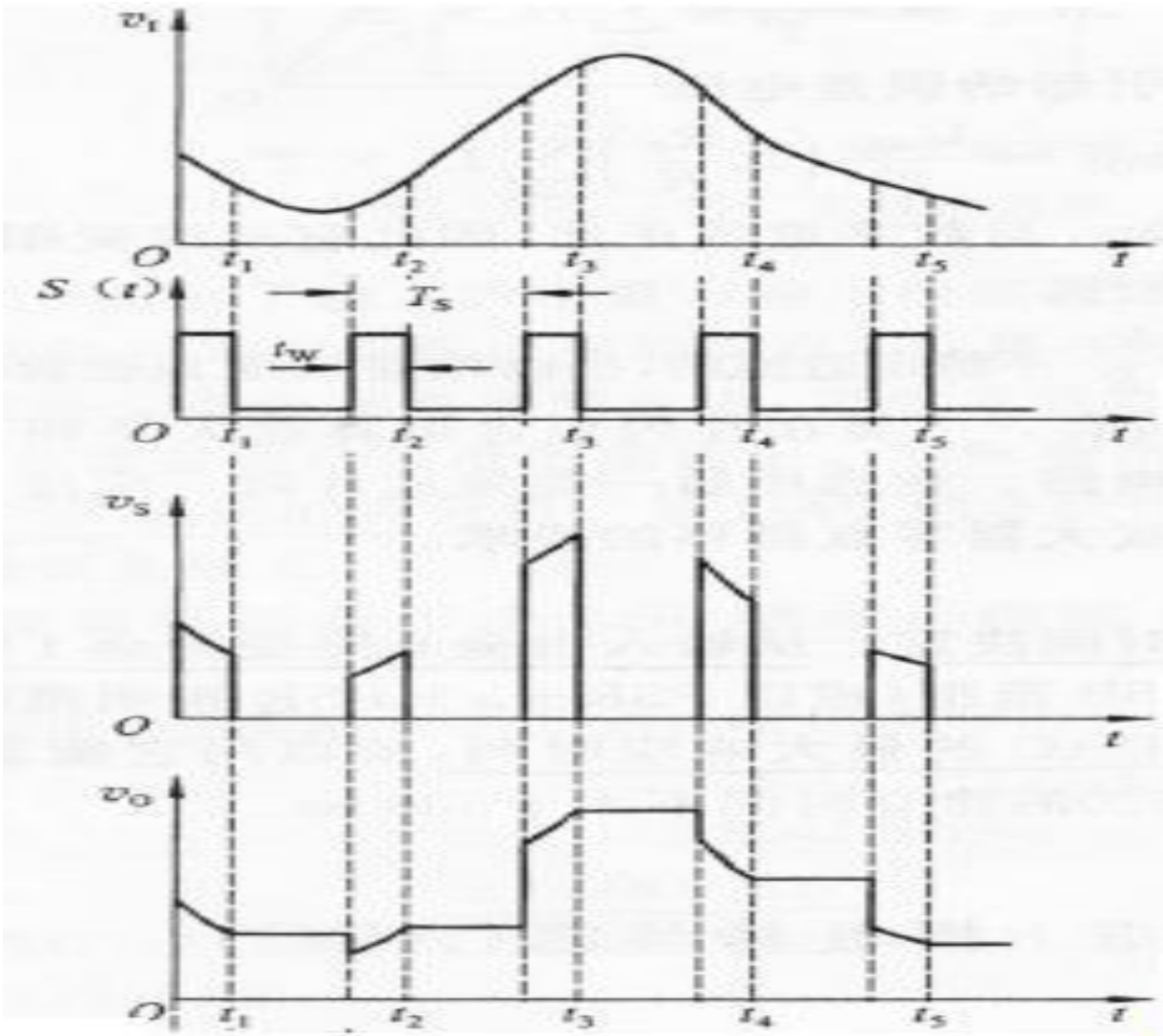
Sampling and Holding

Quantizing and Encoding


Input analog signal
$u_I(t)$
S
C
S/H circuit
$u_I'(t)$
Quantizing & Encoding
Output digital signal
D_{n-1}
D_1
D_0

Sampling and Holding:-

It is a process of taking a sufficient number of discrete values at point on a waveform that will define the shape of waveform.



Continuous Signal

Sampling pulse

Sampled signal

Sampled and held signal

For a 3-bit A/D, with Vmin=0 V, V max=10 V, what is the code-word for Vin=5.7 Volts?

Solution:-

$N=2^n$

For a 3 bit A/D converter, $N=2^3=8$.

Analog quantization size:

$Q = (Vmax - Vmin) / N = (10V - 0V) / 8 = 1.25V$

Output States	Discrete Voltage Ranges (V)	Output Binary Equivalent
0	0.00-1.25	000
1	1.25-2.50	001
2	2.50-3.75	010
3	3.75-5.00	011
4	5.00-6.25	100
5	6.25-7.50	101
6	7.50-8.75	110
7	8.75-10.0	111

Resolution:-

Resolution is the number of bits used for conversion (8 bits, 12 bits, …)

How to increase Accuracy of A/D Conversion?

Increasing the resolution

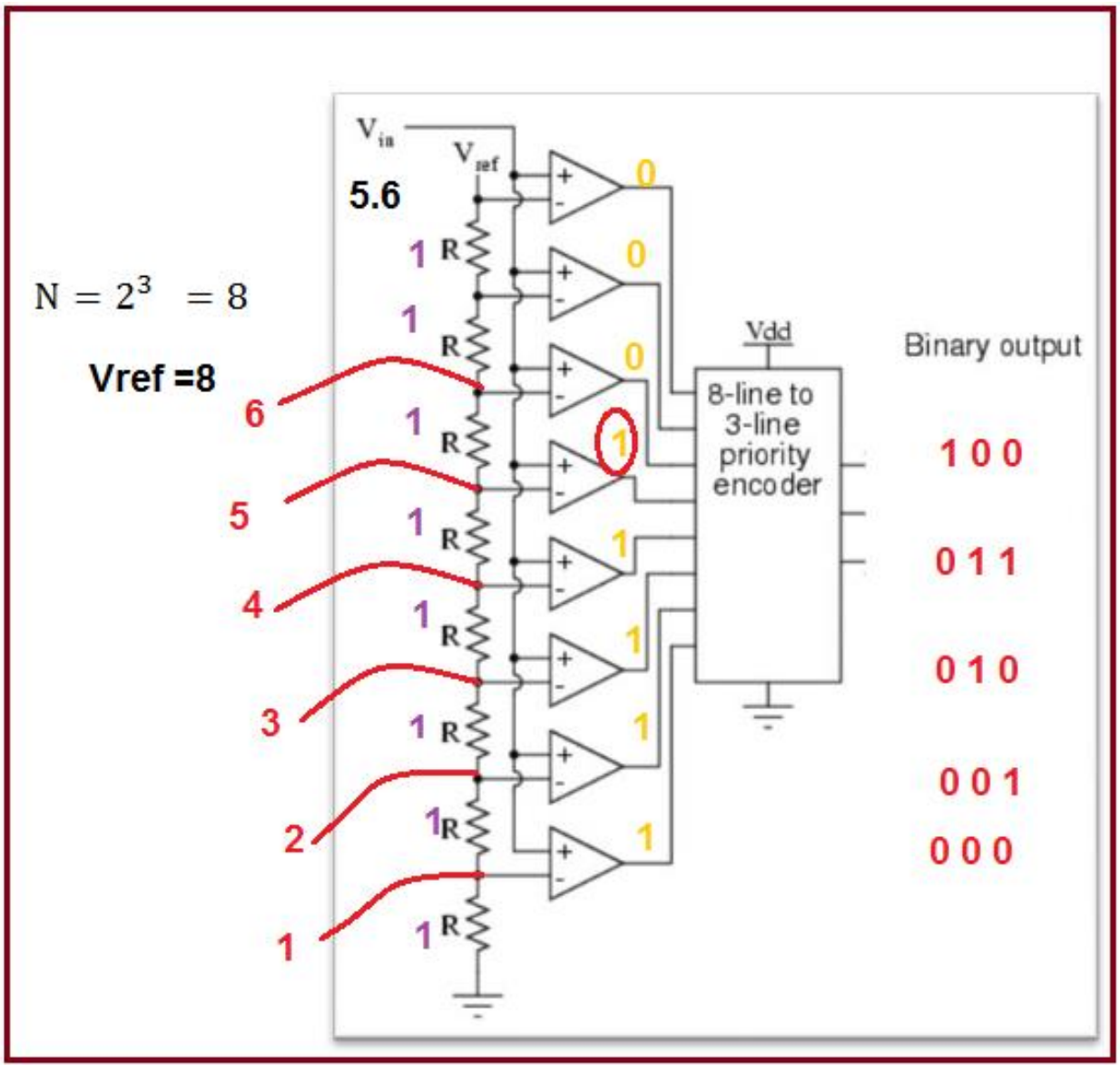
Increasing the sampling

Flash A/D Converter

What is flash A/D Converter Components

1) Resistors use the resistors to form a ladder voltage divider
2) Comparators comparing input signal to reference voltage.
3) Priority encoder produces a binary output.

Find the code-word for a 3 bit, flash A/D with Vref=8V with Vin=5.6V.



How does Flash A/D work?

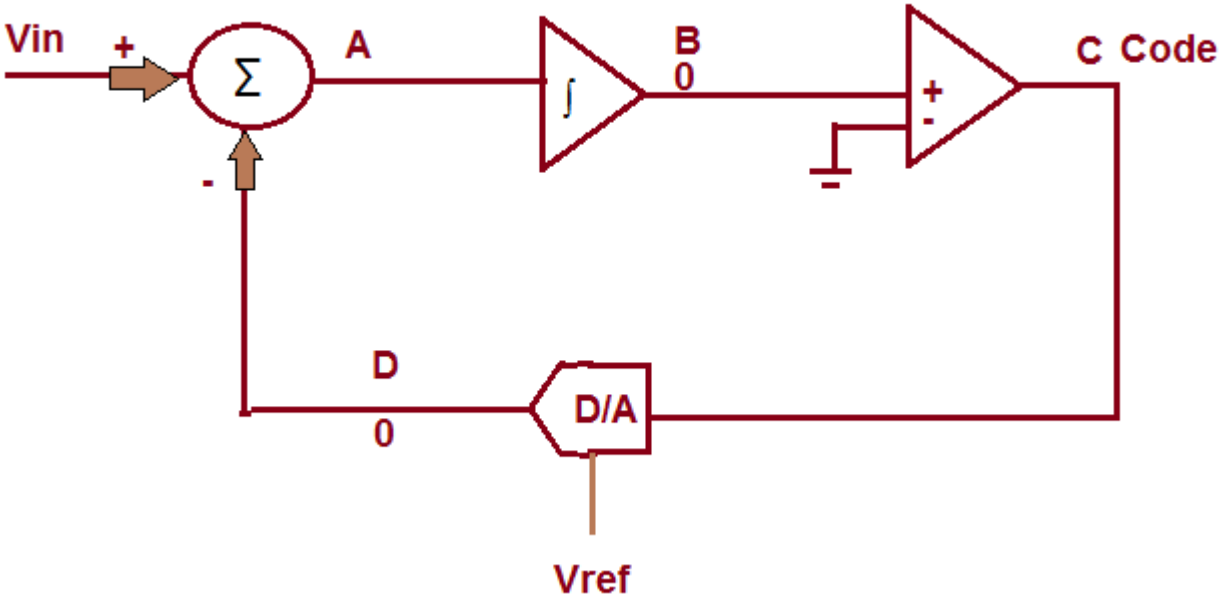
1- Vref is divided over the resistors

2- The voltage on the resistors is compared with Vin

3- The output of the comparator is fed to the priority encoder.

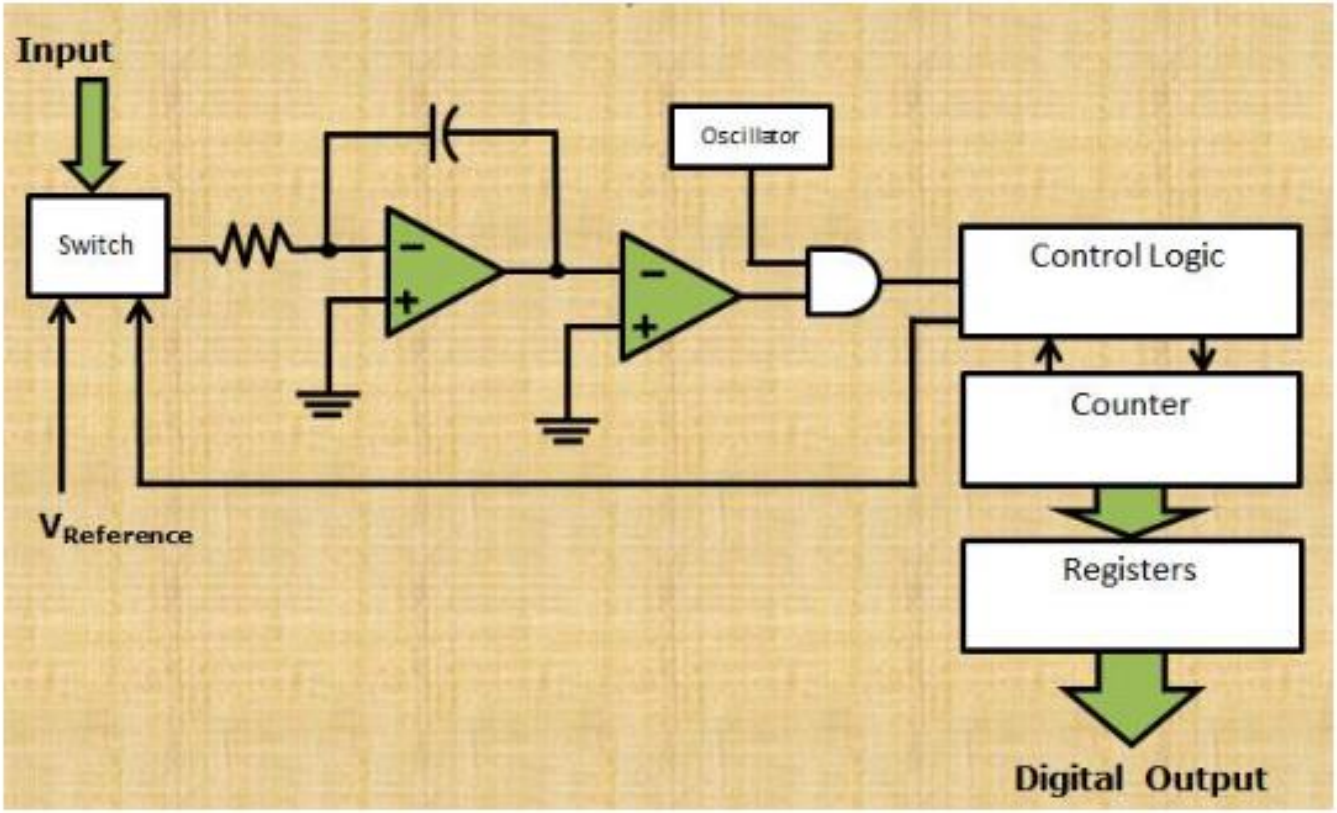
What is sigma delta A/D Converter Components draw the CCT

1- Adder
2- Integrator
3- Comparator
4- D/A



What is dual slope A/D Converter Components draw the CCT

Integrator
Electronically
Cont
Counter
Clock
Control Logic
Comparator



Digital to Analog Converters

What is the D/A? why is it needed? Do we always need it?

(DAC) converts a digital signal or values to an analog voltage or current output.

Derive the equation for Vout for a binary weighted resister with Rf=R/2.

$$V_{out} = -I\,R_f$$

$$= -R_f\left(\frac{V_1}{R} + \frac{V_2}{2R} + \frac{V_3}{4R} + \cdots \frac{V_n}{2^{n-1}R}\right)$$

IF $R_f = R/2$

$$V_{out} = -\left(\frac{V_1}{2} + \frac{V_2}{4} + \frac{V_3}{8} + \cdots \frac{V_n}{2^n}\right)$$

For example, a 4-Bit converter yield

$$V_{out} = -V_{ref}\left(b_3\frac{1}{2} + b_2\frac{1}{4} + b_1\frac{1}{8} + b_0\frac{1}{16}\right)$$

Where b corresponds to Bit-3, b to Bit-2, etc.

Derive the equation for Vout for a binary weighted resister with Rf=R/3. Draw the block-diagram of a binary-weighted resistor cct.

$$V_{out} = -I\,R_f$$
$$= -R_f\left(\frac{V_1}{R} + \frac{V_2}{2R} + \frac{V_3}{4R} + \cdots \frac{V_n}{2^{n-1}R}\right)$$

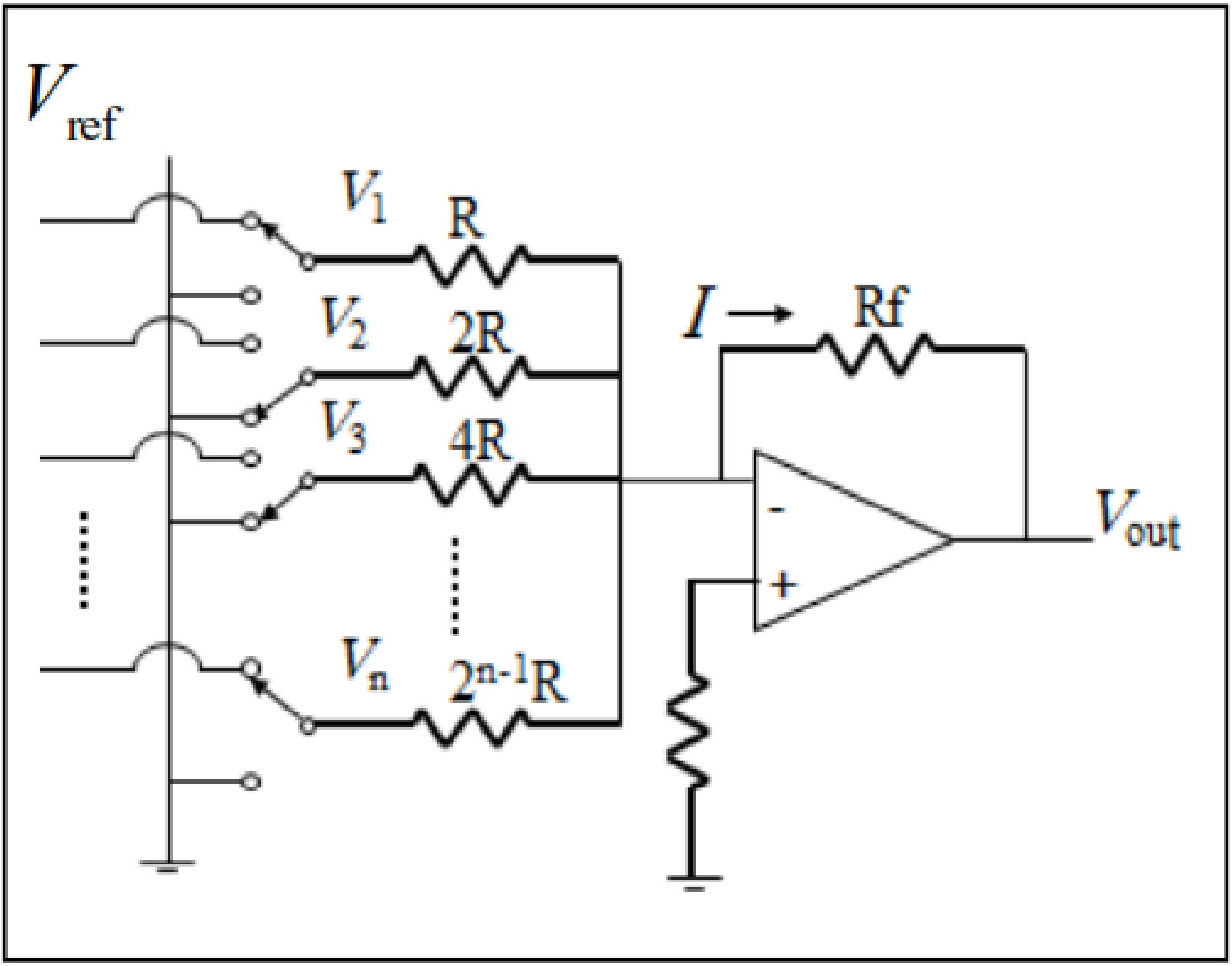
IF $R_f = \frac{R}{3}$

$$V_{out} = -\left(\frac{V_1}{3} + \frac{V_2}{6} + \frac{V_3}{12} + \cdots \frac{V_n}{2^n}\right)$$

For example, a 4-Bit converter yield

$$V_{out} = -V_{ref}\left(b_3\frac{1}{3} + b_2\frac{1}{6} + b_1\frac{1}{12} + b_0\frac{1}{24}\right)$$

Where b corresponds to Bit-3, b to Bit-2, etc.


V_{ref}
V_1
R
V_2
2R
V_3
4R
V_n
$2^{n-1}R$
I
Rf
-
+
V_{out}

Derive the equation for Vout for a binary weighted resister with Rf=R/6.

$$V_{out} = -I\,R_f$$
$$= -R_f\left(\frac{V_1}{R} + \frac{V_2}{2R} + \frac{V_3}{4R} + \cdots \frac{V_n}{2^{n-1}R}\right)$$

IF $R_f = \frac{R}{6}$

$$V_{out} = -\left(\frac{V_1}{6} + \frac{V_2}{12} + \frac{V_3}{24} + \cdots \frac{V_n}{2^n}\right)$$

For example, a 4-Bit converter yield

$$V_{out} = -V_{ref}\left(b_3\frac{1}{6} + b_2\frac{1}{12} + b_1\frac{1}{24} + b_0\frac{1}{42}\right)$$

Where b corresponds to Bit-3, b to Bit-2, etc.

Find Vout for a binary weighted resistor D/A with Vref=5 Volts, and codeword = 101111

For a 6-Bit converter yield

$$V_{out} = -V_{ref}\left(b_5\frac{1}{2} + b_4\frac{1}{4} + b_3\frac{1}{8} + b_2\frac{1}{16} + b_1\frac{1}{32} + b_0\frac{1}{64}\right)$$

$$V_{out} = -5\left(1 * \frac{1}{2} + 0 * \frac{1}{4} + 1_* \frac{1}{8} + 1 * \frac{1}{16} + 1 * \frac{1}{32} + 1 * \frac{1}{64}\right) = -3.6718$$

Find Vout for a binary weighted resistor D/A with Vref=3.3 Volts, and codeword = 101101

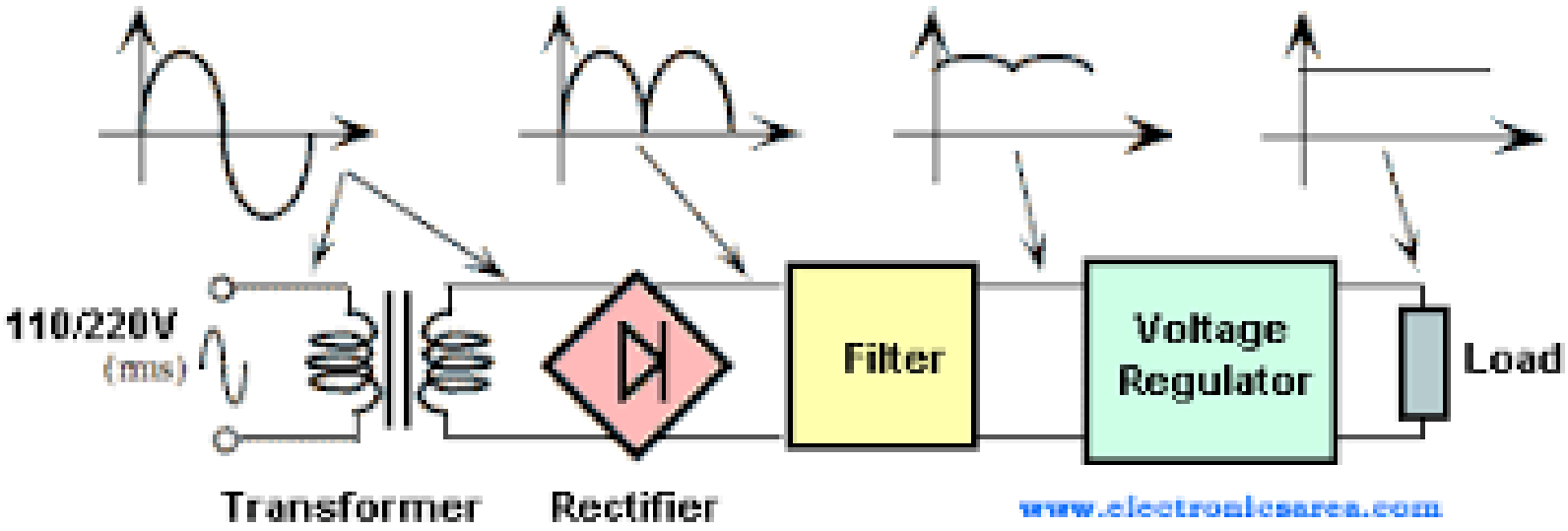
For a 6-Bit converter yield

$$V_{out} = -V_{ref}\left(b_5\frac{1}{2} + b_4\frac{1}{4} + b_3\frac{1}{8} + b_2\frac{1}{16} + b_1\frac{1}{32} + b_0\frac{1}{64}\right)$$

$$V_{out} = -3.3\left(1 * \frac{1}{2} + 0 * \frac{1}{4} + 1_* \frac{1}{8} + 1 * \frac{1}{16} + 0 * \frac{1}{32} + 1 * \frac{1}{64}\right) = -2{,}3203$$

DC Power Supply

The aim of a DC power supply is to provide the required level of DC power to the load using an AC supply at the input. DC power supplies usually have the following parts: Transformer, Rectifier, Smoothing, and Regulation.



Picture 1. Regulated power source Block Diagram

AC/DC POWER SUPPLY

Circuit diagram :

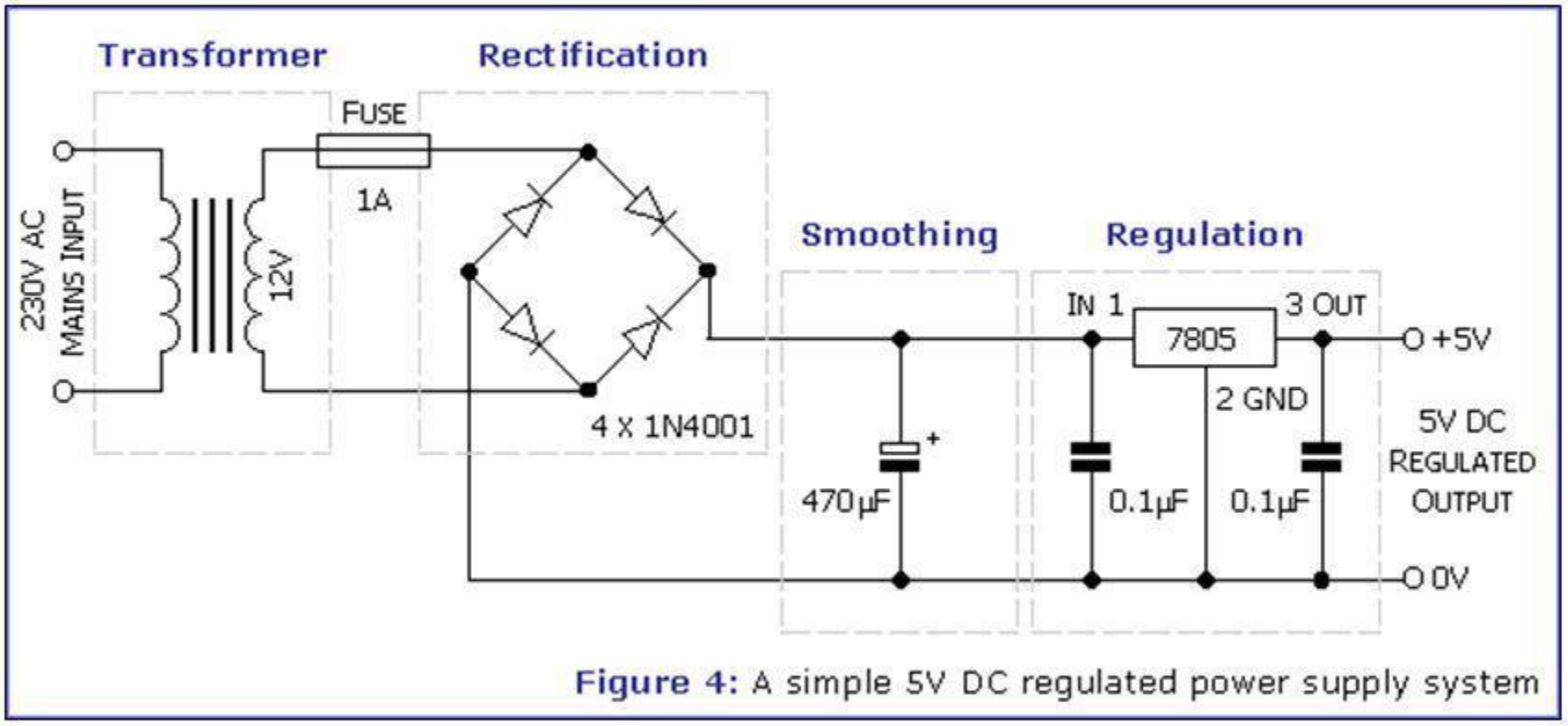
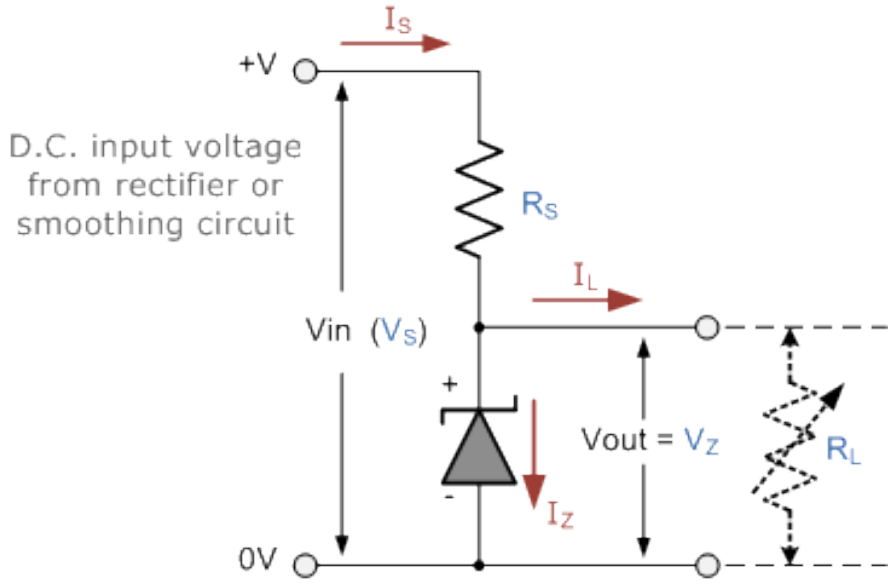


Figure 4: A simple 5V DC regulated power supply system

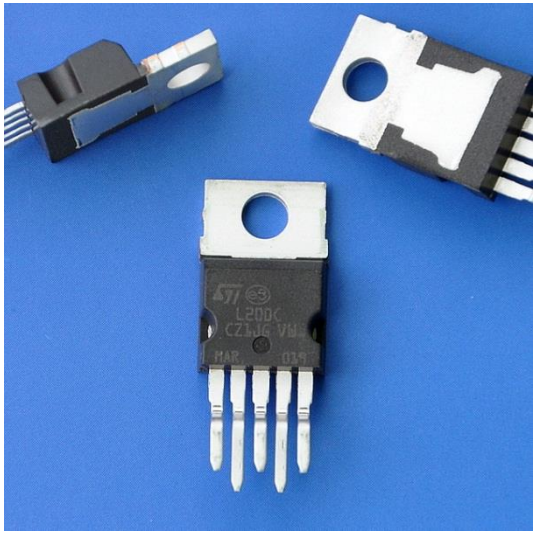
Note: This configuration applies for 230V (Europe).

power supply using Zener diode

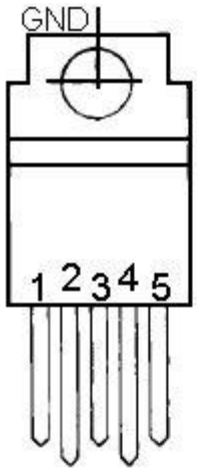
The simplest way of generating a fixed voltage is to use Zener diodes. The regulated voltage can vary from 2.4V to 75V using the BZX79 series diodes. The diodes in this series are rated at 500 mW and the tolerance of the stabilizing voltage is
5%.

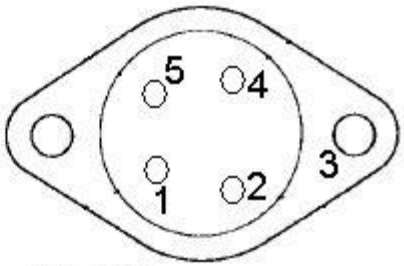


Voltage Regulator



a L200C adjustable voltage regulator. It can supply a regulated voltage from 2.85 to 36V with an output current up to 2 A. It features current limiting, thermal shutdown and input over voltage protection up to 60V. The quiescent current is typically 4.2 mA.





1 : INPUT
2 : LIMITING
3 : GND
4 : REFERENCE
5 : OUTPUT

www.electronicecirc

Voltage Converters

a voltage inverter which converts +5V voltage to-5V using an SI7660CJ voltage converter (Siliconix). The chip is able to generate a negative voltage output which is equal to the positive voltage input in the range 1.5V to 10V.

Voltage Converters

MAX680CPA converts a +5V voltage to +10V and-10V it is doubler and inverter (Maxim). The input voltage ranges from 2V to 6V. The internal resistances for the positive and negative output are 150 ~and 90Ω respectively. If a 10 mA current is drawn from both outputs, the positive voltage falls to 7V and the negative voltage becomes -6.1V. The quiescent current of the device is typically 1 mA for a 5V power supply.

Isolated voltage supply circuits

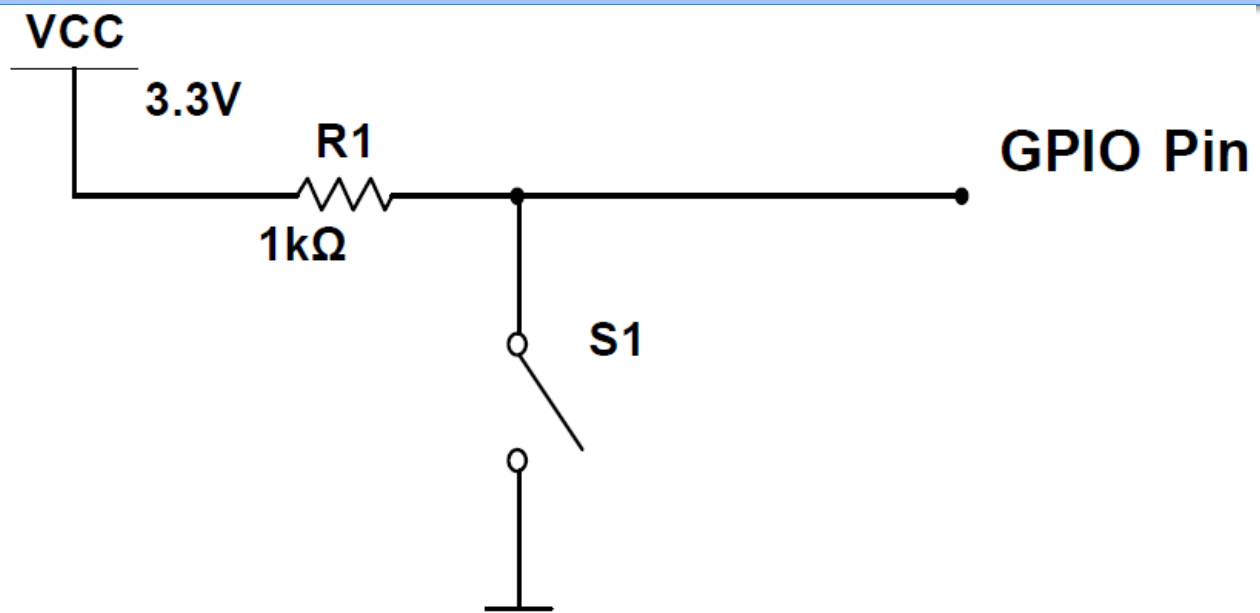
This circuit is used when a complete isolation between two circuits is required. NME and NMA series DC-to-DC converters are high efficiency voltage converters.

The NME series operate from a 5V or 12V DC input and provide an isolated +5V, 12V or 15V output, depending on types. Up to 200 mA supply current is available from the 5V type, 84 mA from the 12V type and 67 mA from the 15V type. The NMA series provide dual +-5V, +-12V and +-15V DC supplies from a single 5V or 12V DC input. Up to 100 mA is available from the 5V type and 42 mA from the 15V type

Digital signal generators

an eight-channel logic status generator circuit. It consists of eight single pole double throw (SPDT) switches and eight l k metal film resistors.

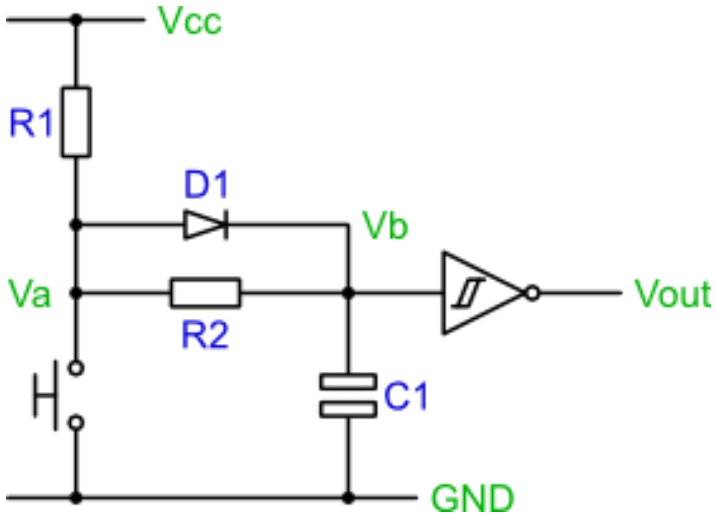
When a switch is off, the status of the corresponding channel is high. When it is switched on, a logic low is generated. This logic generator suffers that the output signal is not 'clean' when it changes the status. When the switch changes position, the output signal does not change from one state to the other instantly. It consists of a number of oscillations within a very short period of time.

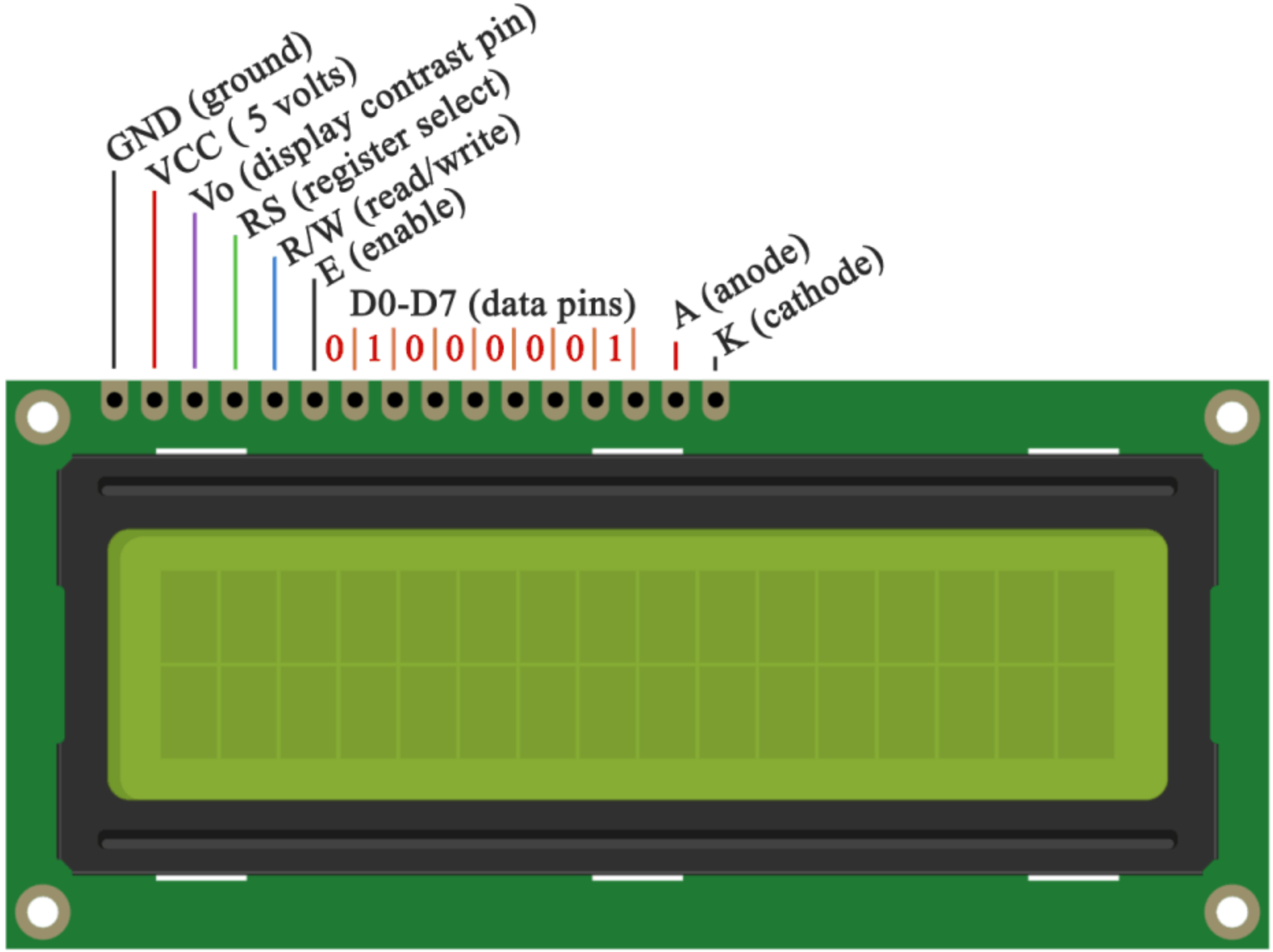


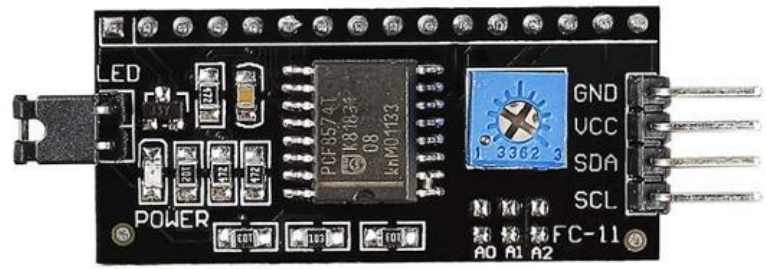
Digital signal generators

To solve this problem, a de-bouncing circuit is used. (circuit using a Schmitt trigger inverter, 74LS14). When the switch is closed, the output gives logic 1. When the switch is open, the output gives logic 0.

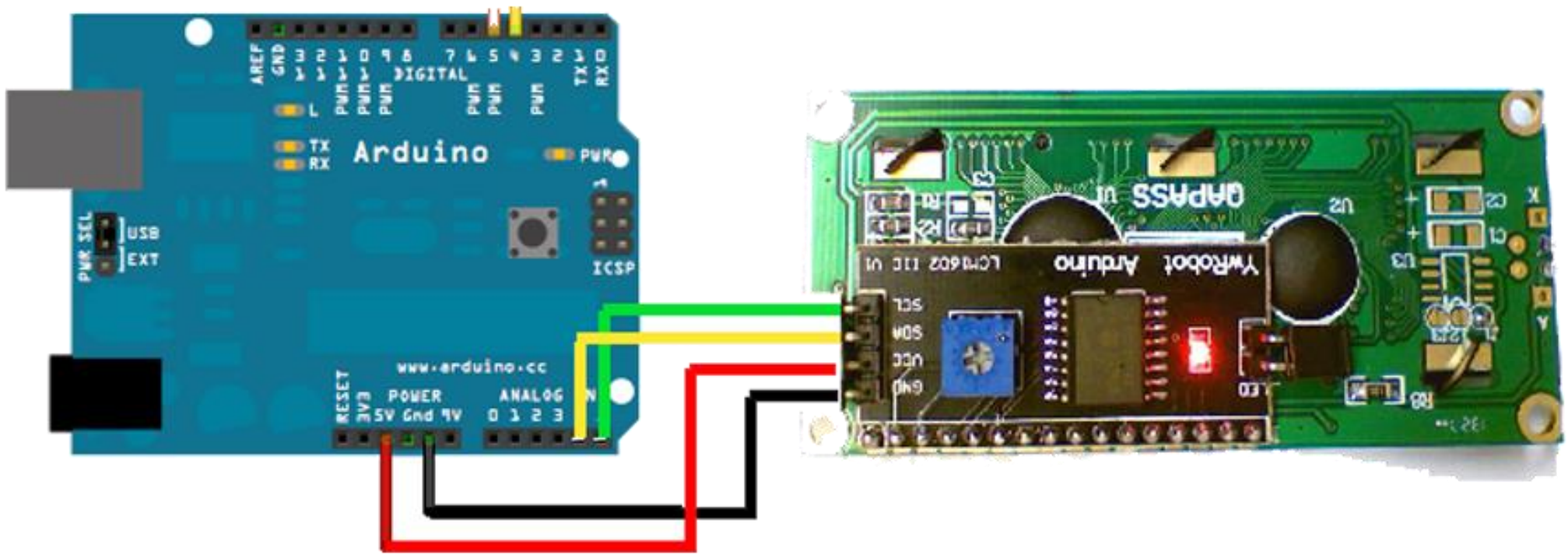
generator circuit with (SPDT)

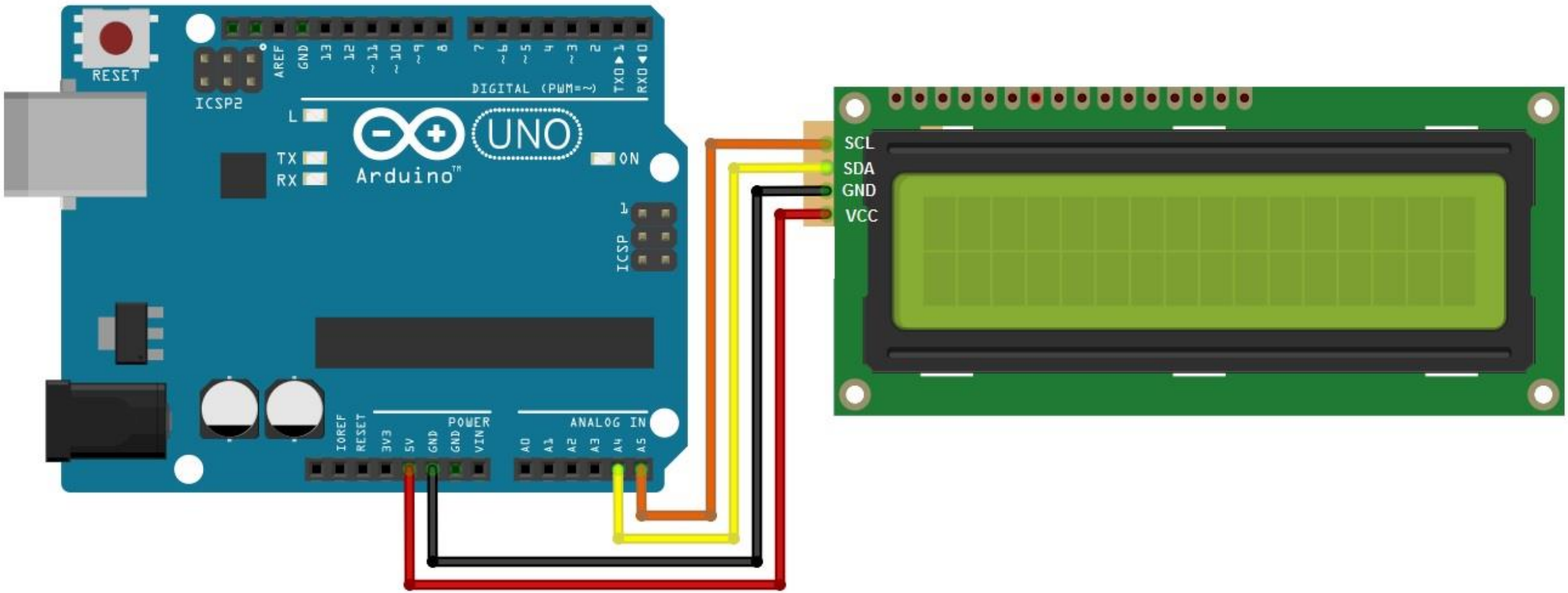



GND (ground)
VCC (5 volts)
Vo (display contrast pin)
RS (register select)
R/W (read/write)
E (enable)
D0-D7 (data pins)
0 1 0 0 0 0 0 1
A (anode)
K (cathode)


LED
POWER
GND
VCC
SDA
SCL
A0 A1 A2
FC-11


1
16
VSS VDD VO RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A K
hello,world!
SunFounder


AREF
GND
DIGITAL
TX
RX
L
Arduino
PWR
USB
EXT
PWR SEL
ICSP
www.arduino.cc
POWER
ANALOG
RESET
3V3
5V
Gnd
Vin
SCL
SDA
VCC
GND
YwRobot Arduino
LCM1602 IIC V1


RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
Arduino™
UNO
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
SCL
SDA
GND
VCC

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,20,4);  // set the LCD address to
0x27 for a 16 chars and 2 line display

void setup()
{
  lcd.init();                    // initialize the lcd
  lcd.init();
  // Print a message to the LCD.
  lcd.backlight();
  lcd.setCursor(1,0);
  lcd.print("hello everyone");
  lcd.setCursor(1,1);
  lcd.print("konichiwaa");
}


void loop()
{
}
```

```
#include<UltraDistSensor.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,20,4);

UltraDistSensor mysensor;
float reading;

void setup() {
   lcd.init();                  // initialize the lcd
   lcd.backlight();
   Serial.begin(9600);
   mysensor.attach(12,13);//Trigger pin , Echo pin
}

void loop() {
   reading=mysensor.distanceInCm();
    lcd.setCursor(0,0);
    lcd.print("Distance :- ");
     lcd.print(reading);

   delay(1000);
}
```

Introduction to

PC Interfacing

Software Engineering Dept.

Technical College of Kirkuk

4th Class

References

- PC Interfacing Using Centronic, RS323, and game ports, 1988
  By: Pei Ann
  ISBN: 0240514483

- The 8085 microprocessor architecture and programming
  By: R. Gaunkar

Introduction to

Parallel Interfacing

Lecture No.1
Parallel Port Interfacing

Introduction

- **Port:** is a physical interface through which data or signal is transferred to or received from peripherals.
- The Centronic, RS232 and game ports are the most common I/O ports that a modern computer has.
- Some notebook computers may not have a game port, but centronic and the RS232 ports are the universal feature of all types of computers.
- Originally, these ports were designed for specific application, centronic ports are used for connecting computers to printers; RS232 ports for connecting printers, modems, and mice; and game ports for connecting joysticks. They can also be used for other application.
- Therefore it is very useful to understand how these ports works and how to make the best use of them.

The Centronic (Parallel) port

- It is also known as LPT (Line Print Port) or printer port.
- It is an industrial standard interface designed for connecting printers to a computer.
- A traditional computer at least has one LPT port installed.
- In total, four Centronic ports may be installed on a computer and they have logic names LPT1 to LPT4.

Port Connectors

- The port connectors on a computer and on a printer are different in design.
- The one on the computer is a 25-pin D-type female connector.

- The one on the printer is a 36-pin Centronic-type female connector

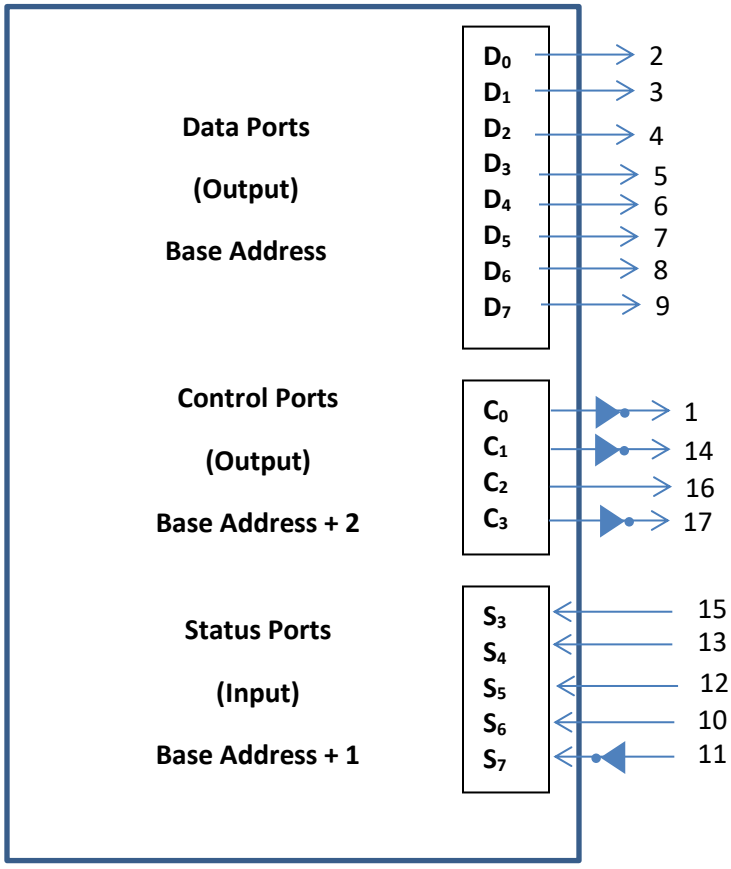
- The pin functions of the two connectors are shown in the table below.

	Pin number		
	25-way D-type	*36-way Amphenol*	*Signal*
C_0 ←	1	1	$\overline{\text{STROBE}}$
	2	2	D0
	3	3	D1
	4	4	D2
D_0 To D_7	5	5	D3
	6	6	D4
	7	7	D5
	8	8	D6
	9	9	D7
S_6 →	10	10	$\overline{\text{ACK}}$
S_7 →	11	11	BUSY
S_5 →	12	12	PE
S_4 →	13	13	SELECT
C_1 ←	14	14	$\overline{\text{AUTOFEED}}$
S_3 →	15	32	$\overline{\text{ERROR}}$
C_2 ←	16	31	$\overline{\text{INIT}}$
C_3 ←	17	36	$\overline{\text{SELECT-IN}}$
	18–25	19–30, 33	Signal ground
	–	15	Not connected
	–	16	0 V (logic ground)
	–	17	Chassis ground
	–	18	Not connected
	–	34	Not connected
	–	35	Logic 1

- A printer cable is used to connect the PC and the printer, the length of the cable must not exceed 5 meters.

Internal Hardware Organization

- The I/O lines in the port are organized into three groups, namely, the data group, the control group and the status group.
- Figure (1.b) shows the logic structure of the LPT port.



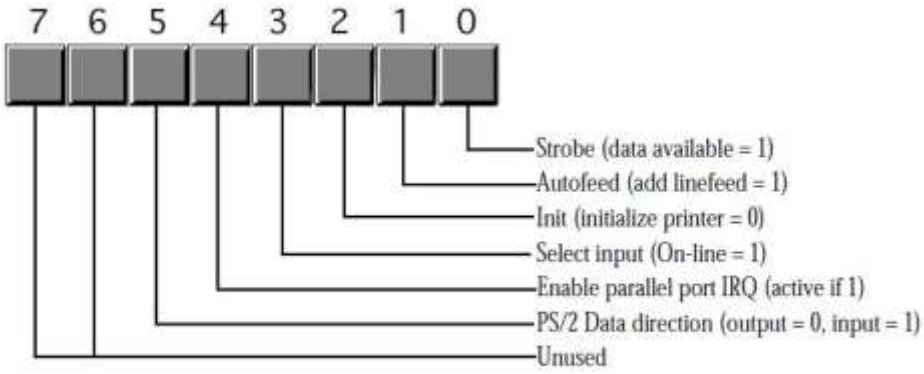
Data Group

This group sends data from PCs to external devices. It has 8 latched output lines and the group is associated with an 8-bit CPU port. The address is : base address.

Control Group

This group controls the operation of external devices. It contains 4 latched output lines. The group is controlled by a CPU port having an address : base address + 2.

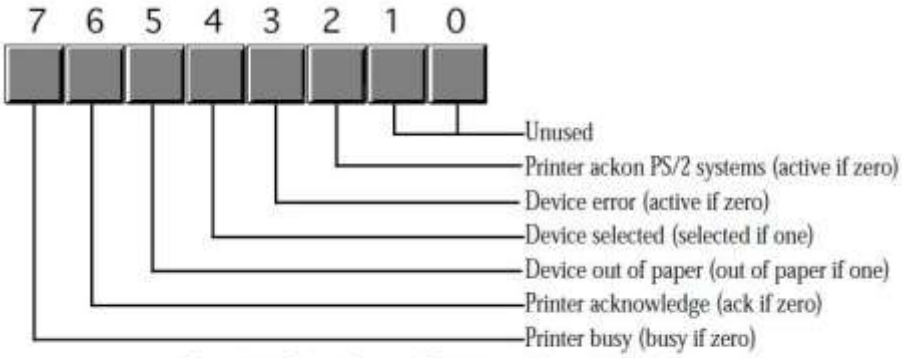
Note that bit 0,1 and 3 are inverted, and bit 2 is not.



Parallel Port Control Register

Status Group

The group is used by the computer to obtain the current status of external devices. It contains 5 lines. It is fed into a CPU port, the address of which base address is : base address + 1. Note that bit-7 is inverted.



Parallel Port Status Register (read only)

LPT Port Base Addresses

- The base address of LPT1 and LPT2 are shown below:
  LPT1 : 956 (**$03BC_H$**) or 888 (**0378_H**)
  LPT2 : 632 (**278_H**)

- The base address of LPT1 varies. This depends on the hardware configuration of the computer.
- The base address can be found directly from the user's program by using the facilities provided by the computer Basic Input Output System (BIOS).
- When a computer is powered on or reset, the BIOS checks all possible Centronic ports. If it finds one, it writes the addresses (a 2 byte or Word) of that port to two specific memory locations as the size of each memory location is 8 bit.
- The port address is stored in a reserved memory location specifically designed for this task.
- By reading the contents of these memory locations, the base address of LPT1 or others can be obtained.
- The memory locations for where the base address of LPT1 to LPT4 are listed below:

  LPT1: 0408_H – 0409_H
  LPT2: $040A_H$ – $040B_H$
  LPT3: $040C_H$ – $040D_H$
  LPT4: $040E_H$ – $040F_H$

- There is another useful 8 bit memory location (0411_H). it stores the total number of Centronic ports installed. The information is contained in bits 6 and 7, as follows:

7	6	5	4	3	2	1	0
*	*	-	-	-	-	-	-

↓ ↓

Bit 7	Bit 6	Meaning
0	0	No Centronic ports installed
0	1	One Centronic ports installed
1	0	Two Centronic ports installed
1	1	Three Centronic ports installed

- The following is written in QBasic. It displays the total number of installed Centronic ports and the base address of LPT1 to LPT4.

```
10    DEF SEG = 0
20    PRINT "The number total number of Centronic ports is:",(PEEK(&H0411) AND (128 + 64)) / 64
30    PRINT "Address of LPT1:",PEEK(&0408) + 256 * PEEK(&H0409)
40    PRINT "Address of LPT2:",PEEK(&040A) + 256 * PEEK(&H040B)
50    PRINT "Address of LPT3:",PEEK(&040C) + 256 * PEEK(&H040D)
60    PRINT "Address of LPT4:",PEEK(&040E) + 256 * PEEK(&H040F)
70    INPUT X
```

Line 20 reads the byte stored in memory location 0411_H using the "PEEK()" command. Bits 6 and 7 of this byte are masked by "AND (128 + 64)". Then, the result is shifted 6 bits towards the LSB using a division command "/64". Line 30 reads two bytes from two memory locations holding the LSB and MSB part of the base address for LPT1.

Lines 40,50 and 60 perform the same action for LPT2, LPT3 and LPT4.

How to Output and Input Data Via Centronic Port

The Centronic port is treated as three components as three separate I/O ports: Two of which is output and one is input.

Let us take an example of controlling the LPT1. Assuming that the address of the data, control and status ports is 888,890 and 889 respectively. In order to send data to data port and control port, the following QBasic commands are used:

OUT 888 , X
OUT 890, X

Where X is the output value in decimal.

Some lines of the control and status ports are inverted. This has to be taken into consideration when outputting data.

To read data from the status port, the following command can be used:

Y = INP(889)

Where Y is the decimal value of the input data

The input data bits correspond to bits 3 to 7 of the status port, and one line is inverted. This has to be taken into account.

Bit Manipulation

Here are some useful basic bit manipulation techniques:

- **Bit Weight:**

  The relationship between a bit and its weight is given below:

Bit 0	1	⟶ decimal value
Bit 1	2	
Bit 2	4	
Bit 3	8	
Bit 4	16	
Bit 5	32	
Bit 6	64	
Bit 7	128	

- **To Make a bit high:**

  The following example shows how to make bit 3 (bit weight = 8) of the data port to go high while keeping the status of the other unchanged. If the original status of bit 3 is high, it will still be high. If the original status is low, it becomes high.

```
10    X = original_Data    OR    8
20    OUT 888, X
```

- **To make a bit low:**

  The following Qbasic example shows how to make bit 4 (weight = 16) of the data port go low

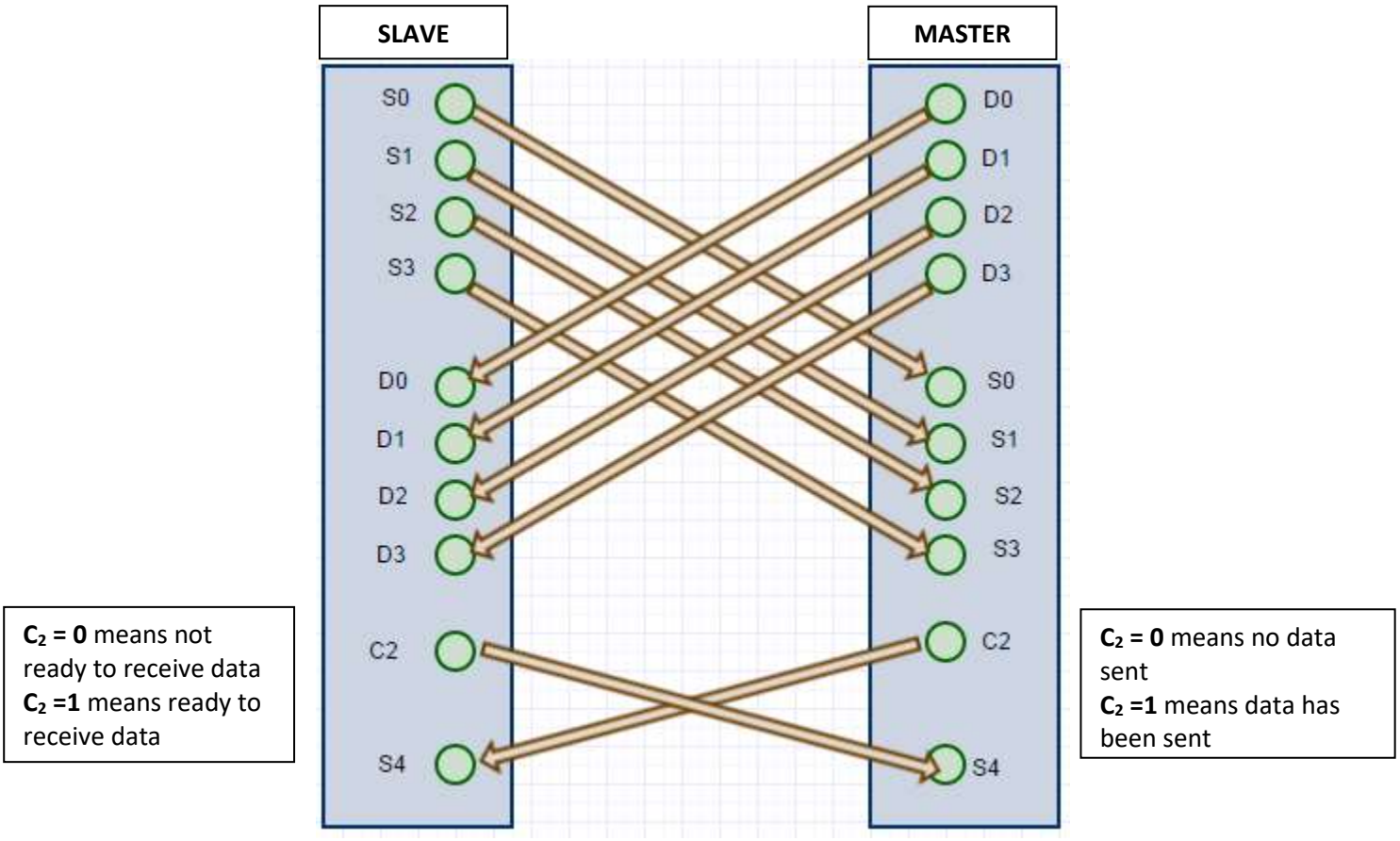
```
10    X = Original_Data AND (255 – 16)
20    OUT 888 , X
```

Parallel port Interfacing – Examples

EX1: Design a 4 bit data link to connect two PCs through the printer port. And write a QBasic algorithm to operate this link and send 10 blocks of data each has 4 bits , state any assumptions needed.

Assumptions:

- One of the two computers is the master and the other should be the slave.
- The master sends the data (4 – bit) and the slave receives it.
- The selection of a computer to be master or a slave is performed manually.



Hardware Design of the LPT Port

DIM a(10) : b(10)
Baseaddress = 888
D_Port = Baseaddress

```
            C_Port = Baseaddress + 2
            S_Port = Baseaddress + 1
            PRINT "This Computer is :"
            PRINT "[1] Master"
            PRINT "[2] Slave"
            INPUT S
            IF S = 2 THEN GOTO 500
                                        REM --- MASTER PROGRAM ---
            i = 0
100         OUT C_Port, 0000 0000B
200         X = INP (S_Port)
            If ( X AND 128 ) / 128 = 0 THEN GOTO 300
            FOR J = 0 TO 100
            NEXT J
            GOTO 200
300         if i + 1 > 10 THEN GOTO 400
            i = i + 1
            OUT D_Port , a(i)
            OUT C_Port, 0000 0100B
            FOR J =1 TO 100
            NEXT J
            GOTO 100
400         PRINT "End Of Transmission"
            GOTO 900
500                                     REM ---SLAVE PROGRAM---
            i=1 : k=1
600         OUT C_Port, 0000 0100
700         X = INP ( S_Port )
            IF ( X AND 128) / 128 = 0 THEN 800
            FOR J=0 TO 1000
            NEXT J
            k=k+1
            IF K > 3 THEN GOTO 850
            GOTO 700
800         OUT C_Port, 0000 0000
            b(i) = INP (S_Port)
            i = i + 1
            GOTO 600
850         PRINT "End of Transmission"
900         END
```

HOMEWORK (1):

Design a PC interface to control the operation of a water heater through computer software. Suggest a suitable hardware, and then write a QBasic code to:

1. **Set the value of the required temperature.**
2. **Read the actual temperature (5 Bits)**
3. **Compare the actual temperature with the set temperature such that:**

   If $t_{set} > t_{act}$ then turn the heater ON
   If $t_{set} <= t_{act}$ then turn the heater OFF

4. **GOTO the first step after a suitable delay**

<u>Homework (1) Solution:</u>

```
Baseaddress = 888
D_PORT = Baseaddress
S_PORT = Baseaddress + 1
C_PORT = Baseaddress + 2
TSET = 20D
OUT D_PORT , 0
200         X = (INP(S_PORT)   XOR  1000 0000₂)
            IF (X  AND  248) / 8 < TSET GOTO 300

            OUT D_PORT, 0000 0000₂
            FOR J=0 TO J = 4000
            NEXT J
            GOTO 200

300         OUT D_PORT, 0000 0001₂
            FOR J=0 TO J = 4000
            NEXT J
            GOTO 200
```

<u>HOMEWORK 2:</u>

(A) Suggest a suitable hardware for a computer controlled lighting system. The lights must be turned ON and OFF according to a light sensor connected to a PC through LPT1 port. Note that the output of the light sensor is binary as follows: 1 for daylight and 0 for darkness.

(B) Write the required software algorithm in QB

HOMEWORK 3:

(A) Suggest a suitable hardware for a computer to control a LED bulletin (نشرة ضوئية) consisted of 8 LED, the LED bulletin should only be on at night and automatically shut off at daylight. The bulletin has two patterns:

The first pattern of the LED bulletin should be as follows and repeated for 5 times

(LED1 – ON) (LED2 – OFF) (LED3 – ON) (LED4 – OFF) (LED5 – ON) (LED6 – OFF) (LED7 – ON) (LED8 – OFF)

The second pattern is also should be repeated for 5 times as follows:

(LED1 – ON) (LED2 – ON) (LED3 – OFF) (LED4 – OFF) (LED5 – ON) (LED6 – ON) (LED7 – OFF) (LED8 – OFF)

(B) Write a software algorithm to operate this LED bulletin.

Note: Assume that each LED will emit light for 1 sec during operation.

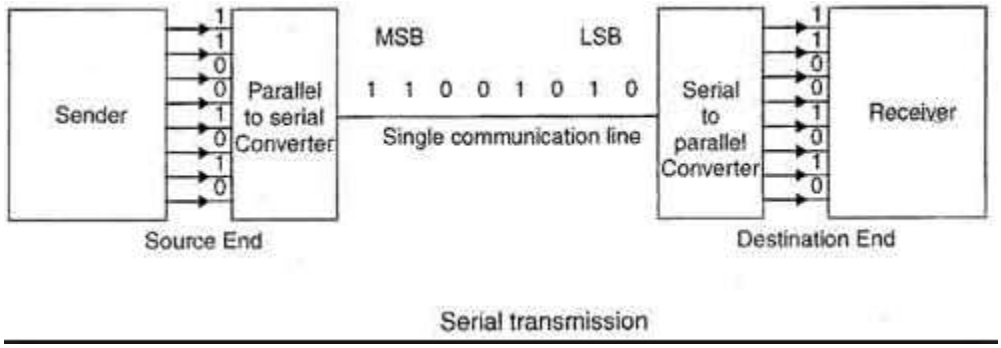
Introduction to

Serial

Interfacing

Introduction to Serial communication

Data transfer within a system is generally in parallel. All the data bits are transferred in parallel at the same instant. In some cases particularly in transferring in long distance, it is preferred to transfer the data in serial form. The transmitted bits are converted to a serial stream of bits by parallel to serial converter and one bit at a time is transferred on a single line to a receiving system. At the receiving end the data bits are reconstructed by serial to parallel conversion.



Serial Data Transfer Types:

There are three general types of serial data transfer

- Simplex (data transferred in one Direction Only)
- Half-Duplex (data transferred in Either direction
- Full Duplex (data transferred in both direction)

Serial Communication Formats:

In serial communication, data are transferred in either

- Synchronous or
- Asynchronous

The data transfer speed in these two formats is different.

Synchronous Communication:

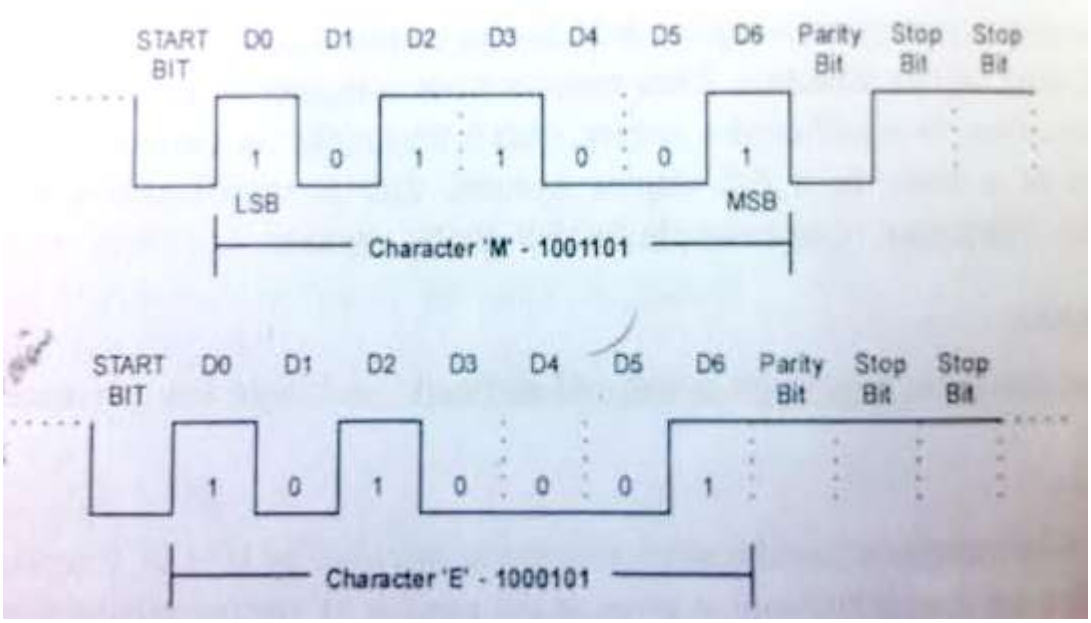
In this format, the transmitter and the receiver use a common clock signal for synchronizing the data transfer. The error in synchronous mode is detected by checksum technique in which the sum of all the bytes in the block is found and it's 2's complement is transmitted as the last byte in the block. The receiver adds all the bytes (including the last) and checks it is result is zero in which it indicates no error. This format is used in high speed communication.

Data transmitted	0010 0010 +
2's Complement of the data	1101 1110
checksum detect	----------
	0000 0000

Asynchronous Communication:

Asynchronous data transfer is used for low speed communication, typically at standard rates 110, 300, 600, 1200, 2400, 4800, 9600 and 112500 baud rate. The baud rate is the speed of data transfer in serial communication, it represents the rate at which the bits are transmitted and it is given as the number of bits per second.

The asynchronous communication format does not use any synchronizing clock or timing signal, instead it adds a framing bits with each character being transmitted, when no character is sent, the transmitter output logic high. Transmission of a character starts with a start bit (logic 0) followed by the character LSB bits first, a parity bit and end up with one or two stop bits (logic 1), this is called as one frame. The line remains in logic 1 till the transmission of next character begins with another start bit. The figure below shows the transmission of two 7-bit ASCII characters, 'M' (4DH) and 'E' (45H). It transmits a start bit, character bits (data bits), parity bit (for even parity) and two stop bits.



The parity bit is included in the frame for the receiver to check errors that may occur during transmission. The bit is made 0 or 1, so that the number of 1s in the character plus the parity bit is always is an odd in odd parity systems or even in even parity systems.

Asynchronous Serial Data Format Explained

The serial data stream contains the information of synchronounzation and the actual data to be transferred.

A serial data format includes four parts:

- A start bit (1 bits).
- Serial data bits (5,6,7 or 8 bits).
- parity check bit (1 bit).
- stop bit (1, 1.5 or 2 bits).

The electronic device that is responsible for generating and receives asynchronous serial data format is called ***Universal Asynchronous Receiver / Transmitter (UART).*** The serial data transmission format is generated by the transmitting UART.

The receiver detects the leading edge of the start bit, it then waits for one and a half bit times before reading the data bit. The reading should come exactly in the middle of the

first data bit. Then it waits for one bit time and reads the second bit. After reading all the data bits, the receiver detects the parity of the received data for error checking and reset itself during the stop bit. It is then ready for receiving the next data transmission.

Serial I/O Standards

There are several standards that specify the protocol for data transfer between devices such as RS-232, RS-422A, RS-423A and RS485. The RS-232 is the most commonly used standard.

RS-232 Serial Port Interfacing

The RS232 serial interface is an industrial bi-directional asynchronous serial data communication interface. For computers, it is used to connect printers, modems, mice and … etc. the maximum communication distance is 20 meters.

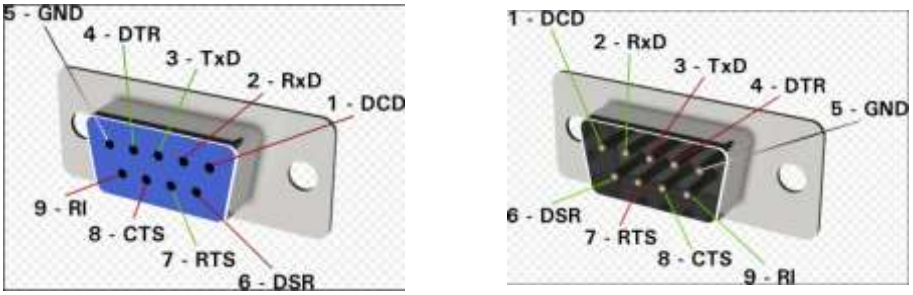
Unlike parallel I/O ports, which consist of a number of data lines and each time transmits a byte; the serial data transmission requires only one line. A byte is transmitted bit-by-bit. This reduces data lines between devices. It reduces the rate of the data transferred too.

RS-232 Port Connector and Connections:

A standard RS232 interface is a 25 pin interfaced housed in a 25 pin or 9 pin D-Type male connector. The figure below shows the functions of the connectors.

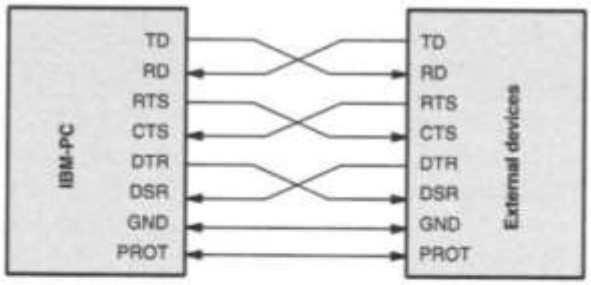
25 pin	9 pin	Name	Direction	Description
1		Prot	--	Protective Ground
2	3	TD	**Output**	Transmit data
3	2	RD	Input	Receive data
4	7	RTS	**Output**	Request to send
5	8	CTS	Input	Clear to send
6	6	DSR	Input	Data set ready
7	5	GND	--	Ground
8	1	DCD	Input	Data carrier detect
20	4	DTR	**Output**	Data terminal ready
22	9	RI	Input	Ring indicator
23		DSRD	I/O	Data signal rate detector

The arrows show the direction of data flow.

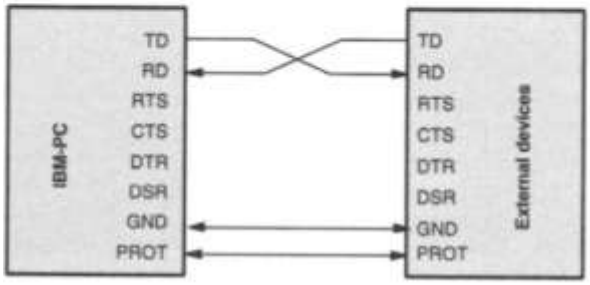


RS-232 Connectors and Cables

There are two types of RS232 link between a computer and an external device, as shown in the figure below (Null Modem)



(a) Null modem for connecting a computer to an external device



RS232 connections between a computer and an external device via 3 lines

Internal Hardware Organization:

An IBM-PC computer can have up to 4 RS232 interfaces. They are labeled COM1 to COM4. Each COM port is associated with a 16450 UART inside a computer.

The 8250/16450 UART:

There are eight 8 bits internal registers within the UART. The I/O addresses of these internal registers are calculated by adding the offset of the register to the base address of the COM port.

The offset and functions of the UART registers are summarized below:

00H: Transmitter hold register / receiver buffer register
Holds the data to be transmitted and stores received data

01H: Interrupt enable register
Sets the mode of interrupt request

02H: Interrupt Identification register
Check s the mode of interrupt request

03H: Data Format Register.
Sets the format of the serial transmission

04H: Modem control register
Sets the modem controls (RTS, DTR,…etc)

05H: Serialization status register
Contains information on status of the receiver and transmitter section.

06H: Modem status register.
Contains current status of the DCD,RI,DSR and CTS.

07H: Scratch pad register.
Acts as a general purpose memory byte.

OFFSET 03H: is the data format register which defines the serial data format such as the baud rate, number of data bits, number of stop bits and parity check.

The bit functions of the **data format register** are given below:

7	6	5	4	3	2	1	0
DLAB	**BRK**	**PAR2**	**PAR1**	**PAR0**	**STOP**	**DAB1**	**DAB0**

DLAB (Divisor Latch Bit):

1 = access to the divisor latch in which it is used to load the divisor to find the baud rate.
0 = access to the receiver buffer / transmitter hold register (offset 00H) and the interrupt enable register (offset 01H)

BRK (Break):
1 = Break on , 0 = break off

PAR2,1,0 (Parity):
000 = none (no parity)
001 = odd parity
011 = even parity
101 = mark
111 = space

STOP:
1 = 2 stop bits
0 = 1 stop bit

DAB1,0 (Data Bit):
00 = 5 data bit
01 = 6 data bit
10 = 7 data bit
11 = 8 data bit

OFFSET 00H: is the receiver buffer register and the transmitter hold register. The transmitter hold register can be accessed if the DLAB bit in the data format register

(offset 03H) is zero. If a byte is written to this address (00H) it is transferred to the transmitter shift register and it is output serially.

After a serial data is successfully received and converted into parallel format the data is transferred in the receiver buffer register. After reading the data from the register, the receiver buffer is cleared and is ready for receiving the next data.

When DLAB bit is 1, the receiver buffer / transmitter hold register (00H) and the interrupt enable register (01H) are used for loading the divisor. The first one holds the LSB byte and the second one holds the MSB byte of the divisor, they form a 16 bit divisor, and the value is calculated using the following equation:

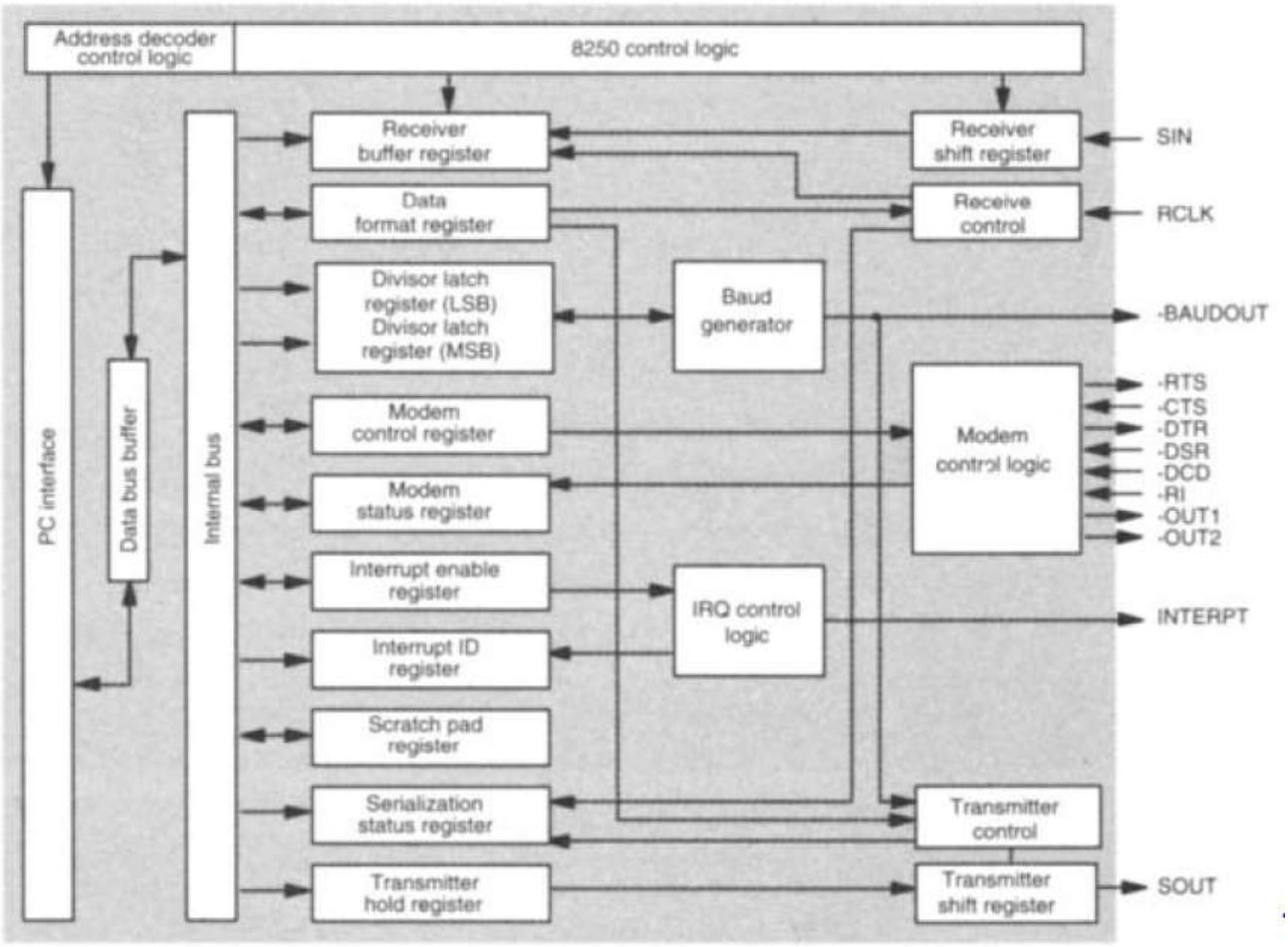
$$\textbf{Divisor} = \textbf{byte}_{\textbf{reg (00H)}} + 256 * \textbf{byte}_{\textbf{reg (01H)}}$$

In a computer the clock frequency to the UART is 1.8432 MHz. inside the UART, the reference frequency is the clock frequency divided by 16, giving 115200 HZ, the relationship between the divisor and the baud rate is :

$$\textbf{Baud Rate} = 115200 / \textbf{Divisor}$$

Baud rate of 9600 requires a divisor of 12. Therefore, when loading the divisor bytes into the registers, ‘12’ should be loaded into the receiver / transmitter buffer register at offset (00H) and ‘00’ loaded into the interrupt enable register at offset (01H). if 1 is loaded into the divisor register, it gives highest baud rate 115200.

OFFSET (07H): is a scratch pad memory which is a random access memory byte. Writing data into this register has no effect on the operation of the UART.



Internal Register design of the UART 16450

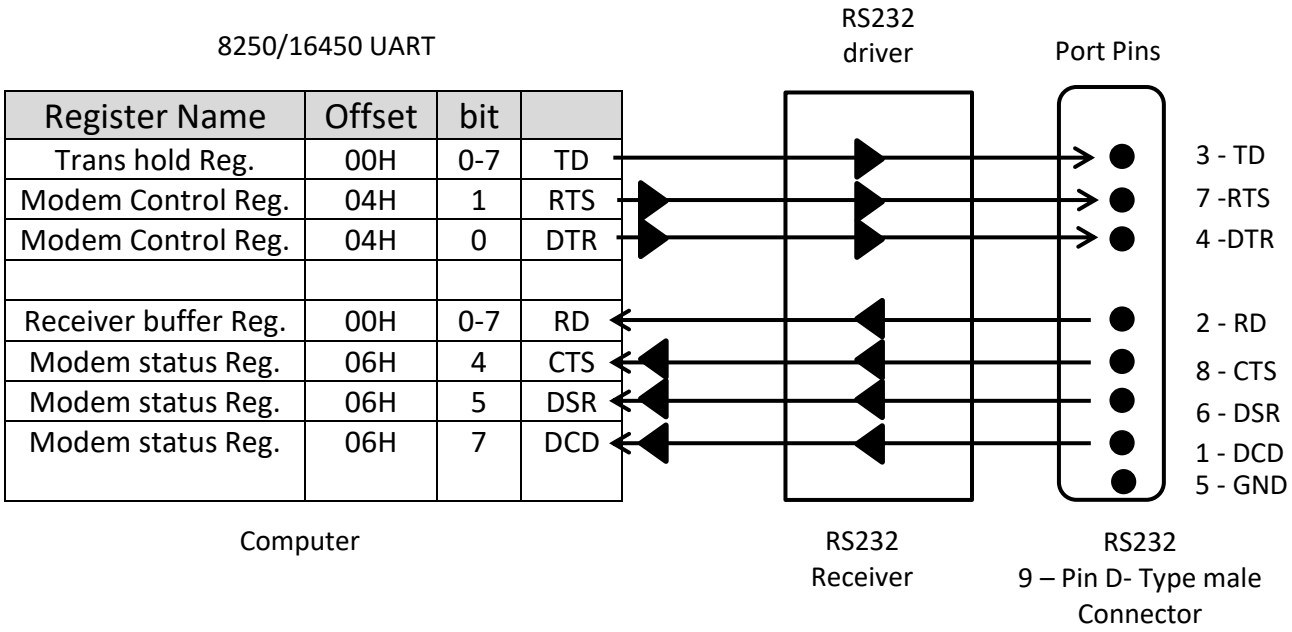
RS-232 Driver/Receiver:

Most AT computers uses the 16450 UARTs. XT computers use 8250 UART in which they are the same in specifications and internal architecture.

The UART uses the TTL voltage level. In order to achieve a longer distance communication, the TTL voltage level is converted to a higher voltage level (logic0=-12 to -3, logic 1 = +3 to +12). This is achieved by using dedicated RS232 drivers/Receiver's.

All drivers and receivers have an inverting actions. Al the UART produces the TTL or CMOS voltage levels only.

The RS 232 line driver/ receiver are connected between the UART and the RS-232 connector. The logic structure of the RS-232 port is shown in the below figure:



Base address of COM ports:

The base address of COM1 to COM4 are shown below

COM1 : $3F8_H$
COM2 : $2F8_H$
COM3 : $3E8_H$
COM4 : $2E8_H$

When a computer is switched or reset, the BIOS checks all possible RS232 addresses. If its fins an installed one, it writes the base address (a 2 byte address) into a specific memory locations.

For COM1 the location is 400H and 401H.
By reading these locations, the base address can be obtained.

The memory locations for COM1 to Com4 are listed below:

COM1 : $400_H – 401_H$
COM2 : $402_H – 403_H$
COM3 : $404_H – 405_H$
COM4 : $406_H – 407_H$

Another usefull 1 byte memory location is the 411_H memory location. It stores the total number of COMs installed. The information is obtained be checking the values of bits 0,1 and 2 as follows:

Bit 2	Bit 1	Bit 0	
0	0	0	No COM ports installed
0	0	1	One COM port Installed
0	1	0	Two COM port Installed
0	1	1	Three COM port Installed
1	0	0	Four COM port Installed

The following program is written in QBasic. It prints the number of COM ports installed and the base address of each port

```
10    SEG DEF = 0
20    PRINT “the Number of COM Ports installed is:”, (PEEK(&H411) AND (1+2+4))
30    PRINT “the base address for COM1 is:”, PEEK(&H400) +256 * PEEK(&H401)
40    PRINT “the base address for COM2 is:”, PEEK(&H402) +256 * PEEK(&H403)
50    PRINT “the base address for COM3 is:”, PEEK(&H404) +256 * PEEK(&H405)
60    PRINT “the base address for COM4 is:”, PEEK(&H406) +256 * PEEK(&H407)
70    INPUT X
```

How to Initialize a COM Port:

Before a COM port can be used, it must be configured to have a specific serial data format. The configuration includes the setting of the baud rate, number of data bits, parity and number of stop bits.

There are three methods of doing this:

1. DOS prompt
2. BIOS INT 14
3. Direct configuration to data format register.

The first method is to use the ‘MODE’ command under windows (DOS prompt). The syntax of the comman is:

```
                MODE COMm: baud = b,parity=p,data=d,stop=s
Or                          MODE COMm : b,p,d,s
```

MODE COM1:96,n,8,1 configures COM1 port to have a baud rate of 9600 bps, no parity check, 8-bit data length and 1 stop bit. The disadvantage of this method is that it does not allow users to change the serial data format within the user’s program.

The Second method uses the BIOS interrupt, INT 14H, which allows the configuration to be made within the user program.

The INT 14 that the BIOS use has 4 different service functions and it deals with the AH,AL and DX registers only.

The register AH is used to indicate the INT 14H service function that the BIOS has to execute (Configure, read, write or get port status) and can be selected as shown below:

AH = 00H - Configuration of the UART Service function AH = 01H –
AH = 01H – Write data bits to port
AH = 02H – Read data bits from port
AH = 03H – get status of the port

The DX register is used to set the port number to use:
DX = 00H using COM1
DX = 01H using COM2
DX = 02H using COM3
DX = 03H using COM4

The register AL is used as follows:

If AH = 00H, the AL register should contain the initialization bits of the COM port such as baud rate, data bits, stop bits and parity bits

7	6	5	4	3	2	1	0
BD2	**BD1**	**BD0**	**PAR1**	**PAR0**	**STOP**	**DA1**	**DA0**

Bit 7, 6 and 5 will define the baud rate as follows

110 baud – 000	1200 baud - 100
150 baud - 001	2400 baud - 101
300 baud - 010	4800 baud - 110
600 baud - 011	9600 baud - 111

Bit 4 and 3 will define the parity: 00 – for no parity, 01 for odd and 11 for even parity.

Bit 2 will define the stop bit: 0 – for 1 stop bit and 1 –for 2 stop bits.
Bit 1 and 0 is used to define the data length

00 -5 bits data
01 - 6 bits data
10 - 7 bits data
11 – 8 bits data

The table below will summarize the functions of the registers used in INT 14H as well as the AL register.

Function	AH	Other register	Returns
Initialize Serial Port	00H	Al = initialize parameters DX = port No.	AH = *port status AL = #modem status
Write data to serial port	01H	AL = data to be written DX = port No.	If success AH(7) = 0 If fail AH(7) = 1
Read data from serial port	02H	DX = port No.	AL = data read **If success AH(7) = 0** **If fail AH(7) = 1**
Get port status	03H	DX = port No.	AH = *Port Status AL = #Modem status ***NOTE: AH(0) = 0 means data not ready*** ***AH(0) = 1 means data ready***

*Bit Pattern of Port Status byte		# Bit Pattern of modem status byte	
D7	Time – out	D7	Receive line signal detected
D6	Transmitter shift register empty	D6	Ring indicator
D5	Transmitter hold register empty	D5	Data set ready
D4	Break detected	D4	Clear to send
D3	Framing error	D3	Change in receive line signal detect
D2	Parity error	D2	Ring indicator
D1	Overrun error	D1	Change in data set ready status
D0	**Data ready**	D0	Change in clear to send status

Bit details of the status and modem status port

Example: Use BIOS serial port service to

(i) Initialize COM1 port to 9600 baud, 8 bit data, 1 stop bit and no parity
(ii) Write the character 'A' to COM1 ('A' = 41H)
(iii) Read character form COM1
(iv) Read character by checking the status of the port

(i)
```
MOV AH,00H
MOV DX,00H
MOV AL,E3H
INT 14
```

(ii)
```
MOV AH,01H
MOV DX,00H
MOV AL,41H
INT 14
```

(iii)
```
MOV AH,02H
MOV DX,00H
INT 14H
```

(iv)
```
WAIT: MOV AH,03H
      MOV DX,00H
      INT 14
      AND AH,01H
JZ WAIT
      MOV AH,02H
      MOV DX,00H
      INT 14H
```

As a Summary, the limitation of this method is that we can not get higher buad rate as the maximum is 9600 bps while the 16450 UART can run at 115200 bps.

The Second method

In this method, the COM port is initialized and configured by directly writing the configuration data into the data format register (offset 03_H) of the UART. This is the most flexible way to configure the serial data format.
The following QBasic program configures the data format register as specified by the user for COM1:

```
Baseaddress = &H3F8
PRINT "enter the baudrate"
INPUT baudrate
PRINT "enter the number of data bits"
INPUT data
PRINT "Select a parity option [0] no parity,[1] odd parity,[3] even parity
[5]mark or [7] space"
Input parity
PRINT "enter the number of stop bits: 1 or 2 "
INPUT stopbits

Divisor = 115200 / baudrate
IF divisor < 256 THEN
divisor_byte1 = divisor
divisor_byte2 = 0
END IF

IF divisor >= 256 THEN
divisor_byte1 = divisor mod 256
divisor_byte2 = divisor / 256
END IF

port_format_byte = (data -5) + (stopbits - 1) * 4 + (parity * 8)
OUT baseaddress + 3, 128
OUT baseaddress , divisor_byte1
OUT baseaddress + 1, divisor_byte2
OUT baseaddress + 3, port_format_byte
```

Example:

Write software algorithm to configure COM1 port according to the following values, bit rate = 9600 bps, data bits = 8 bits, parity = even and stop bit = 1.

Solution 1

```
BASEADDRESS = &H3F8

DIVISOR = 115200 / 9600

IF DIVISOR < 256 THEN
DIVISOR_BYTE1 = DIVISOR
DIVISOR_BYTE2 = 0
END IF

IF DIVISOR >= 256 THEN
DIVISOR_BYTE1 = DIVISOR MOD 256
DIVISOR_BYTE2 = DIVISOR / 256
END IF

OUT (BASEADDRESS + &H03), 128
OUT BASEADDRESS, DIVISOR_BYTE1
OUT (BASEADDRESS + 01), DIVISOR_BYTE2
OUT (BASEADDRESS + 03), 1B_HEX
```

Or

Solution 2

```
BASEADDRESS = &H3F8
OUT (BASEADDRESS + &H03), 128
OUT BASEADDRESS, 12
OUT (BASEADDRESS + 01), 0
OUT (BASEADDRESS + 03), 1B_HEX
```

Example:

It is required to design a pc-based control system to control the operation of two fans (ON or OFF) using COM1 port according to the state of three separated digital sensors, please note that each sensor has one output only. The operation of the fans should be as follow

Sensor condition	Fans	
	Fan 1	Fan 2
All of the sensors are 0	OFF	OFF
Only one of the sensors (any one) is 1 and the other two is 0	ON	OFF
Only two of the sensors (any two) is 1 and the other one is 0	OFF	ON
All of the sensors are 1	ON	ON

```
        BASEADD =&H3F8
        OUT (BASEADD + 04H) , 00
10      SENSOR_STATUS = INP (BASEADD + 06H)
        S1 = (SENSOR_STATUS AND 128) / 128
        S2 = (SENSOR_STATUS AND 32) / 32
        S3 = (SENSOR_STATUS AND 16) / 16
        SUM = S1 + S2 + S3
        IF SUM = 0 THEN
        OUT (BASEADD + 04H), 00

        IF SUM = 1 THEN
        OUT (BASEADD + 04H), 02

        IF SUM = 2 THEN
        OUT (BASEADD + 04H), 01

        ELSE OUT (BASEADD + 04H), 03
        DELAY
        GOTO 10
        END
```