CH-04: Interfacing of Instrumentation Systems

Baikuntha Acharya (baikunth2a@gmail.com)

Department of Electronics & Computer Engineering

Tribhuvan University, Institute of Engineering, Pulchowk Campus

Baikuntha Acharya

Senior Lecturer / Deputy Head,

Research Coordinator

Department of Electronics & Computer Engineering,

Sagarmatha Engineering College, Lalitpur

www.linkedin.com/in/baikunth2a

www.baikunthaacharya.com.np

CH-04: Interfacing of Instrumentation Systems

4.1 Microprocessor and microcontroller and their selection criteria, and applications.

4.2 The **PPI 8255** and interfacing of peripherals (LED, 7-segment display, dip switch, 8-bit ADC, 8/10-bit DAC using mode 0 and mode 1) with 8085 microprocessor.

4.3 Microcontrollers (**Atmega328, STM32**): Architecture, pin configuration, and their application.

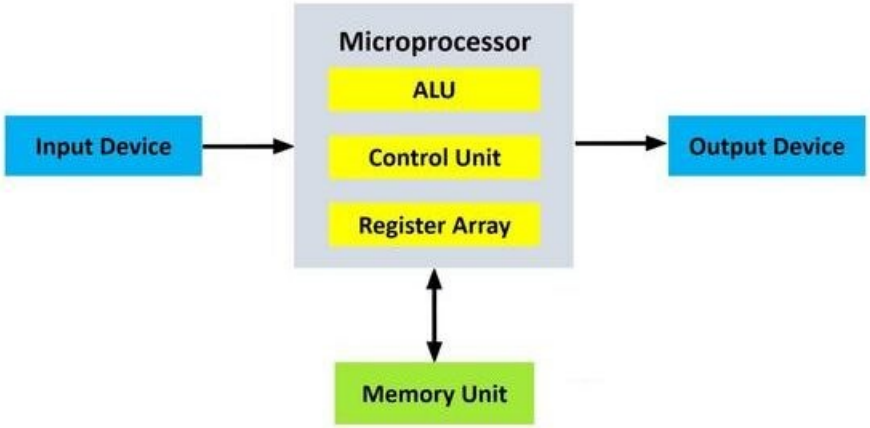
4.4 Sensor/Actuator interfacing with **Atmega328P** (Arduino): Analog and digital sensors, implementation of communication protocols, interrupt-based interfacing.

Modern Instrumentation – Emerging Fields

- ✓ IoT & LPWAN Technologies
- ✓ Wireless Sensor Network (WSN)
- ✓ Digital Twin
- ✓ Cyber Physical Systems
- ✓ Industry 4.0 & 5.0
- ✓ Machine-to-Machine Communication (M2M)
- ✓ SCADA (Supervisory Control and Data Acquisition)
- ✓ SCPI (Standard Commands for Programmable Instruments)
- ✓ PLC & DCS (Programmable Logic Controllers & Distributed Control Systems)
- ✓ Edge and Fog Computing
- ✓ Energy Harvesting in Sensor Networks

Microprocessor and Microcontroller

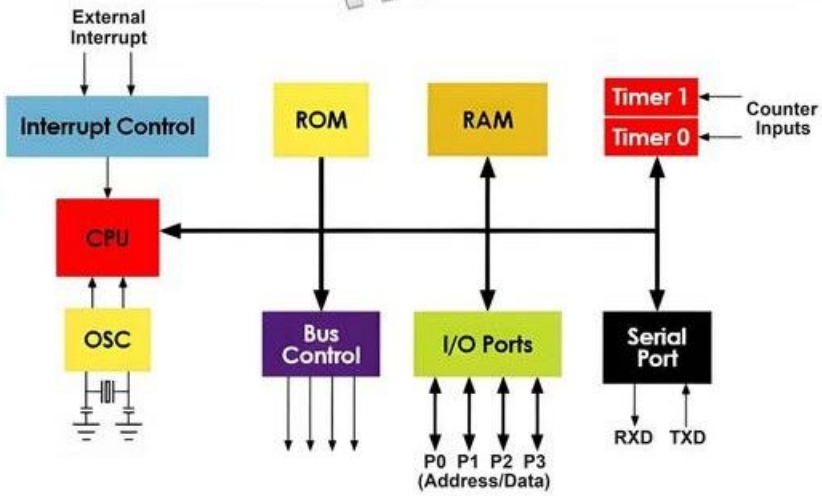
Microprocessor



General Block Diagram of Microprocessor

vs

Microcontroller



General Block Diagram of Microcontroller

Microprocessor and Microcontroller (Cont..)

Microprocessor	Microcontroller
General Purpose Processors	Special Purpose Processors
It contains complete functional CPU only	In addition to CPU, it has timers, I/O ports, ADC/DACs, internal RAM and ROM and other features
Designer can select the size of memory, number of I/O ports, timers etc to be used in the system	Size of memory, number of I/O ports, timers etc will be fixed for a particular microcontroller
Clock speed is very high in GHz range	Clock speed is low in MHz range
Microprocessor based systems are expensive and consumes more power	Microcontroller based systems are cheap and consumes less power
Powerful addressing modes and many instructions to move data between memory and CPU	It includes many **bit handling instructions** along with byte processing instructions.
Access time for external memory and I/O devices is more, resulting in slower system	Access time for on-chip memory and I/O devices is less, resulting in a faster system

Microprocessor and Microcontroller (Cont..)

Common Microcontroller Families

Architecture	Introduced	Technical Highlights	Popular MCUs / Vendors
ARM Cortex-M	~2004	32-bit RISC, low-power, scalable (M0/M3/M4/M7/M33), real-time capable, excellent peripheral support	STM32 (ST), XMC4000 (Infineon), NXP LPC, TI Tiva
AVR	1996	8-bit RISC, high code efficiency, popular with hobbyists and education (Arduino), some industrial use	ATmega328, ATtiny – Microchip (Atmel)
8051	1980	CISC, Harvard architecture, simple instruction set, widely cloned and extended, long lifecycle	Intel (original), Silicon Labs, Nuvoton, Atmel, STC
PIC	1975 (8-bit), 2000s (32-bit)	RISC architecture, extensive peripheral variety, available in 8/16/32-bit families.	PIC16, PIC18 (8-bit), PIC32 (32-bit) – Microchip

Microprocessor and Microcontroller (Cont..)

Features of Microcontrollers

- ✓ General Purpose Input/Output (GPIO).
- ✓ Analog-to-Digital Converters (ADCs).
- ✓ Timers and Counters.
- ✓ Communication Interfaces (e.g., UART, SPI, I2C, CAN).
- ✓ Memory (e.g., Flash for program storage, SRAM for variables/stack, EEPROM for persistent storage).
- ✓ Interrupts

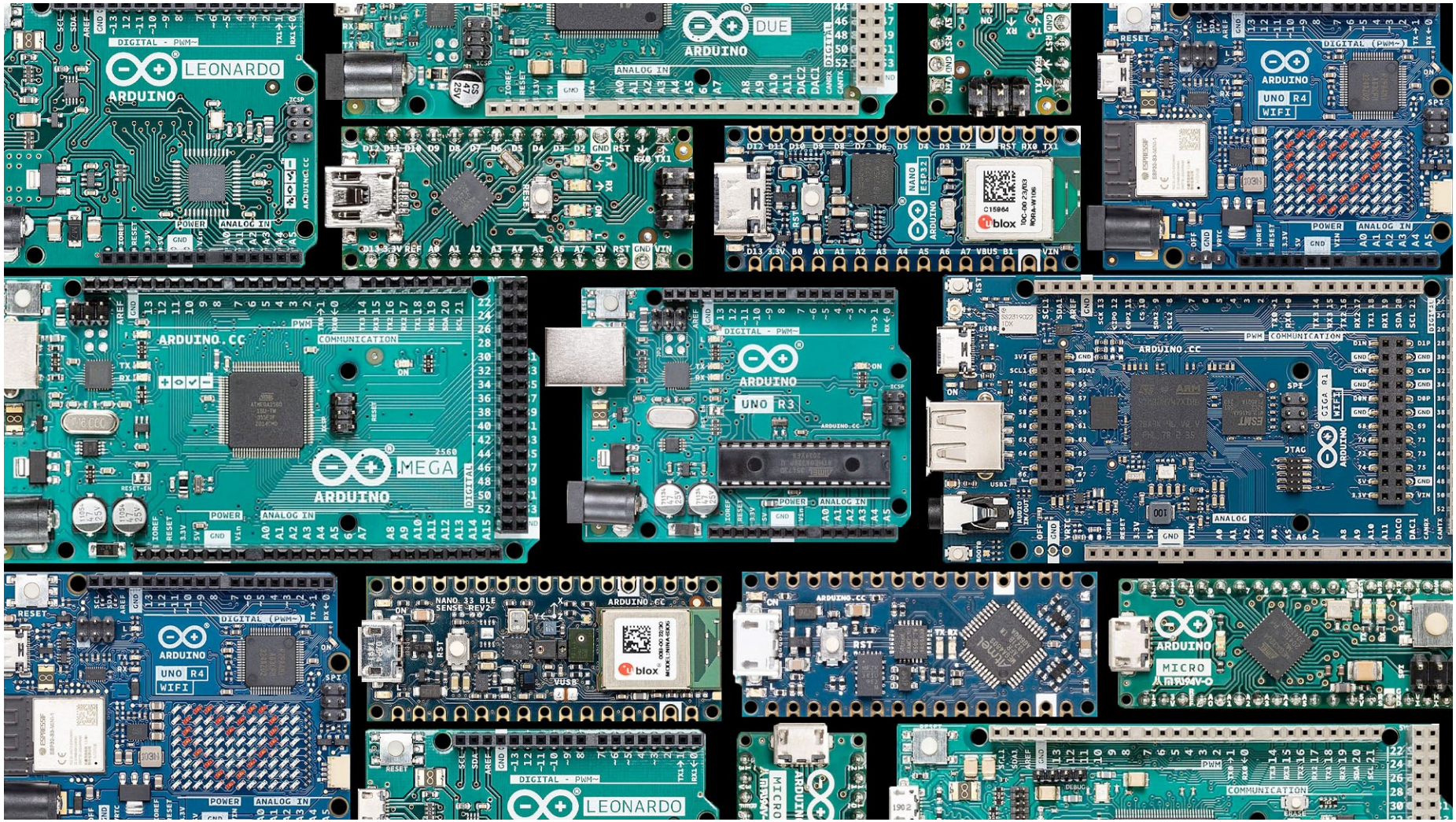
Microprocessor and Microcontroller (Cont..)

Commonly Used Microcontrollers

Platform / MCU	Core	Notable Boards	Features & Typical Use Cases
STM32 (ST)	Cortex-M0/M4/M7	Nucleo, Discovery	High performance, low power, rich I/O – used in data loggers, industrial control
Arduino (ATmega328, etc.)	AVR / Cortex-M0+	Uno R3, Uno R4, Nano, Portenta	Easy to use, large ecosystem – ideal for education, rapid prototyping
ESP32 / ESP32-S3/C6	Xtensa LX6 / RISC-V	ESP32 DevKit, ESP32-S3	Built-in Wi-Fi + BLE – great for IoT, wireless instrumentation
Raspberry Pi Pico / Pico W	Cortex-M0+	Pico, Pico W	Affordable, supports C/C++ & MicroPython – GPIO control
Microchip SAM D21/D51	Cortex-M0+/M4	Arduino Zero, custom boards	Rich peripherals, good ADC
Renesas RA / RX	Cortex-M / RX	RA6M5 Eval Kit	Industrial-grade, secure – used in automation, medical electronics

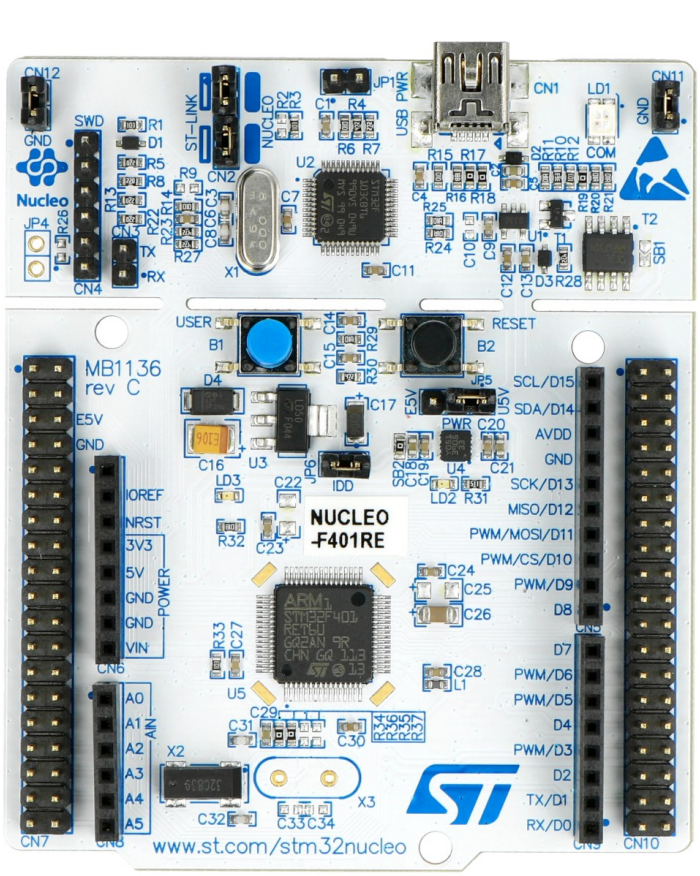
Microprocessor and Microcontroller (Cont..)

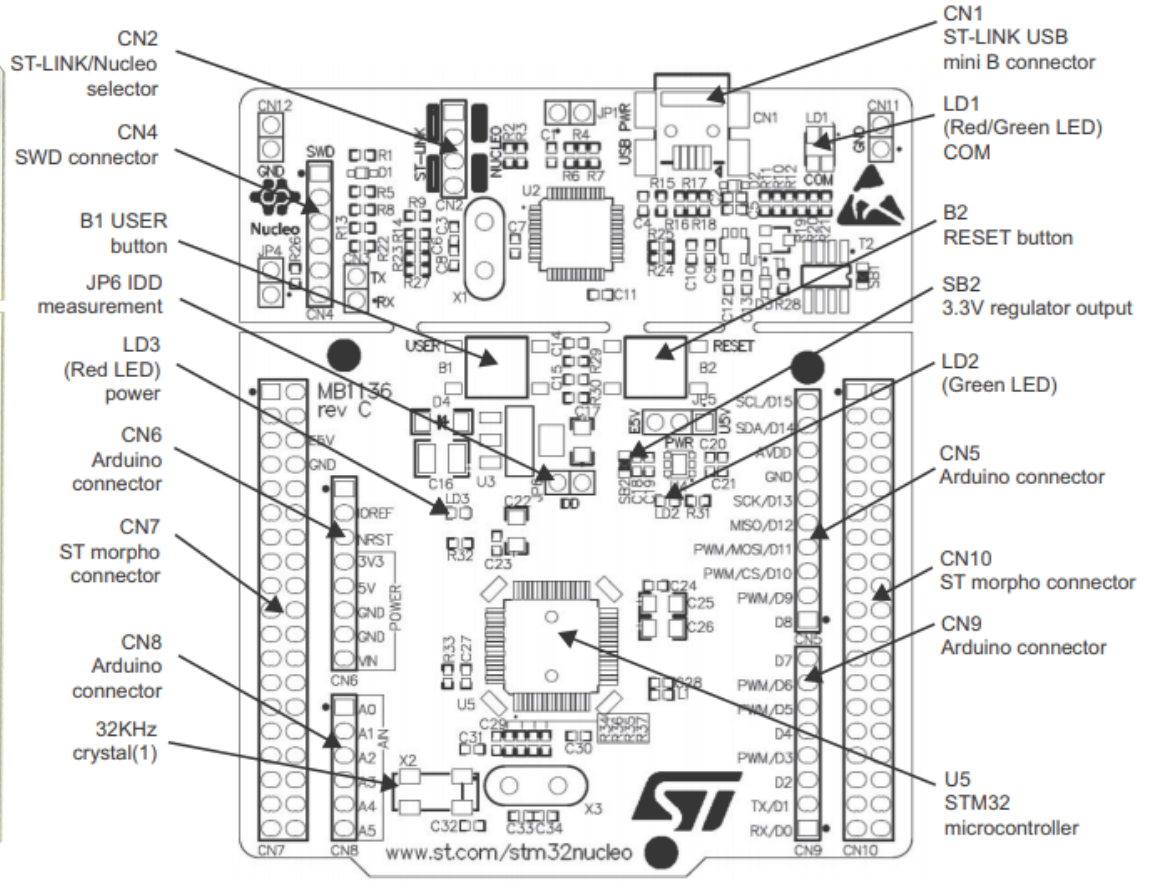
Arduino



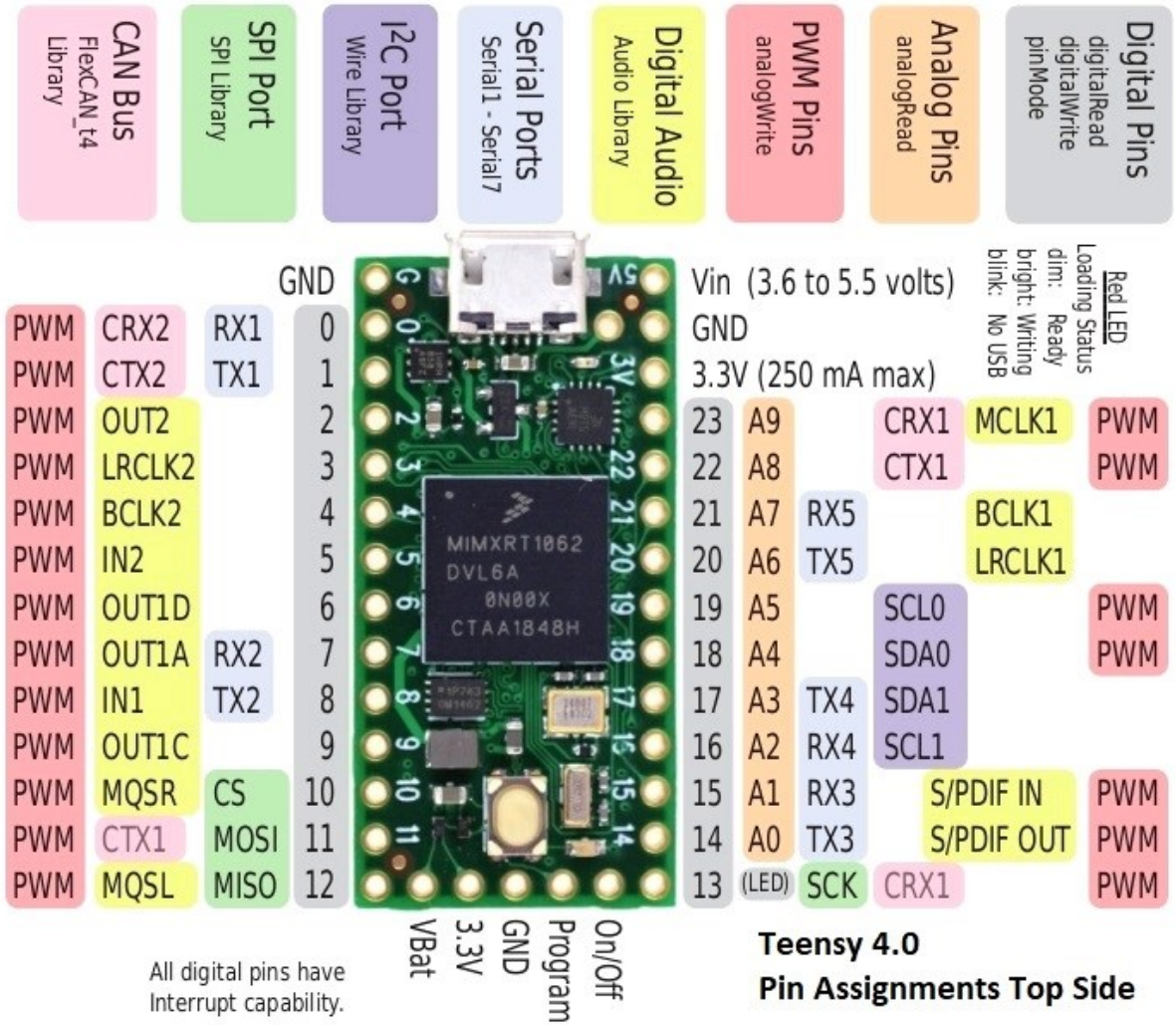
Microprocessor and Microcontroller (Cont..)

STM32 Nucleo-F401RE





Example MCU: Teensy 4.0



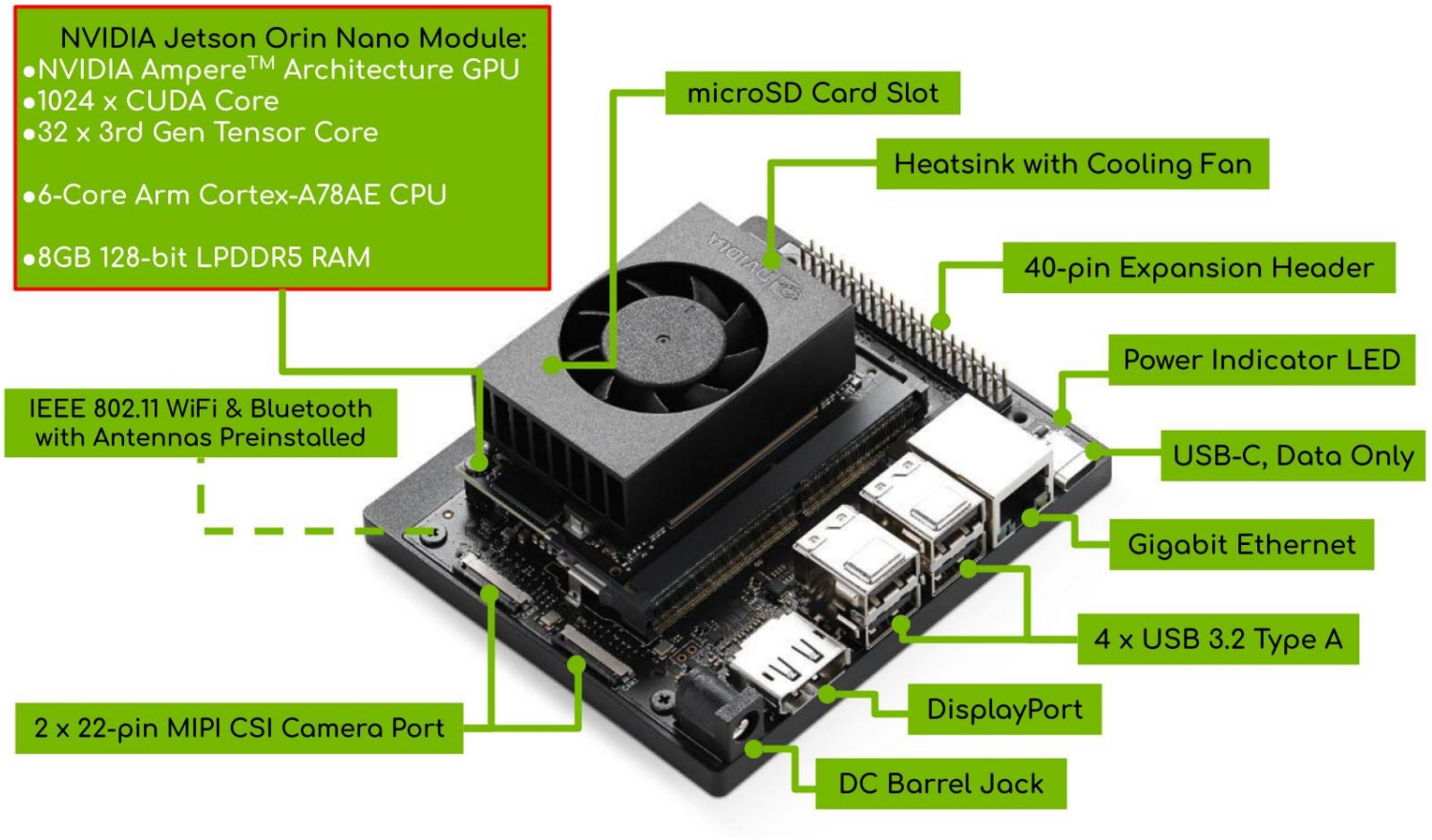
Microprocessor and Microcontroller (Cont..)

Commonly Used Microprocessors (SBCs)

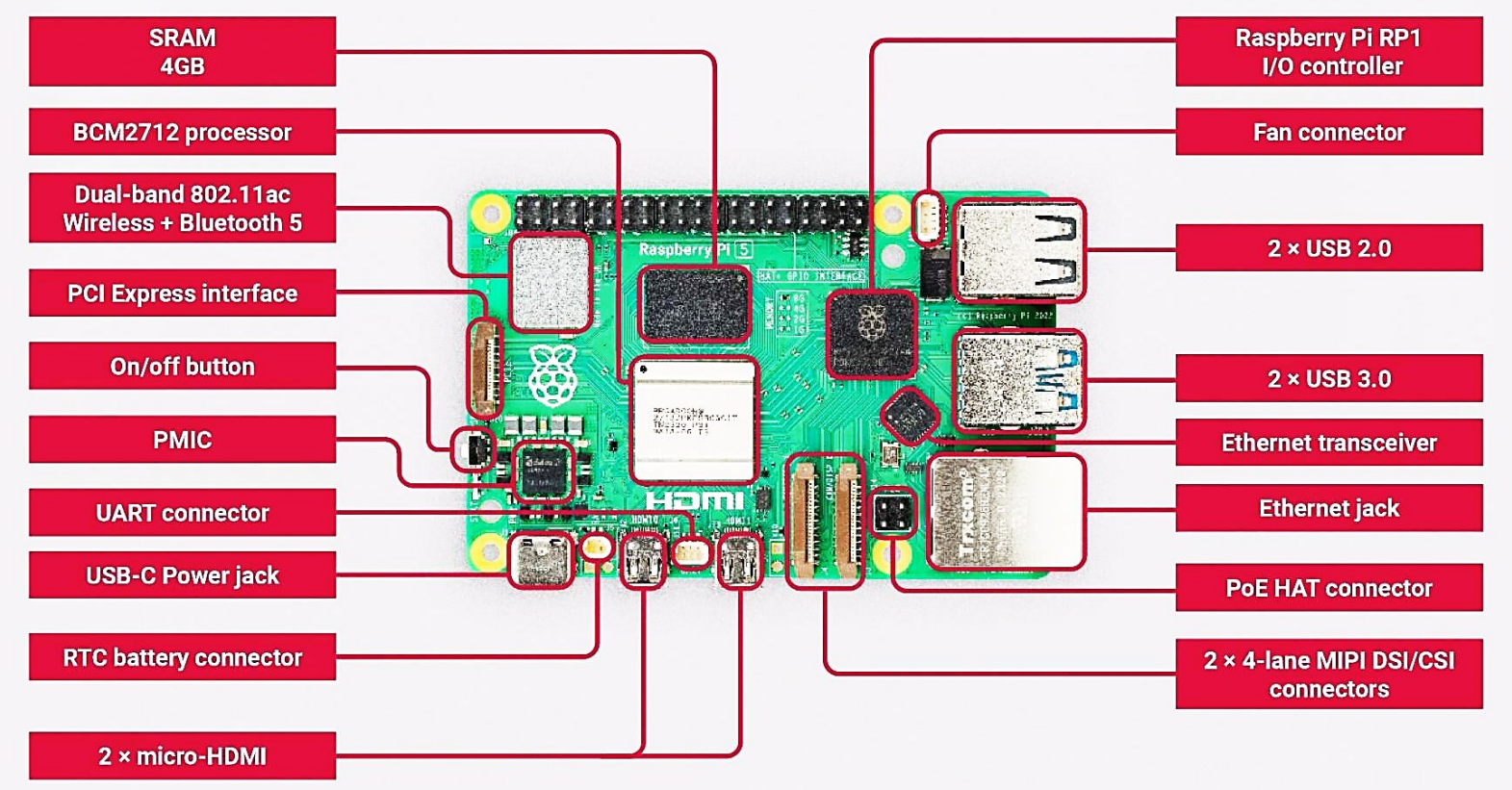
Platform / MPU	Core	Notable Boards	Features & Typical Use Cases
NVIDIA Jetson Series	Cortex-A + CUDA GPU	Jetson Orin Nano, Xavier NX	AI/ML acceleration, vision – robotics, smart cameras, edge AI
Raspberry Pi 4 / 5	Cortex-A72 / A76	Pi 5, Compute Module 4	General-purpose Linux platform – HMI, gateways, edge computing
TI Sitara (AM64x/62x)	Cortex-A53 + R5	BeagleBone AI-64	Real-time + Linux – motor control, industrial Ethernet
Qualcomm QRB / Snapdragon	Cortex-A + Adreno GPU	RB5, DragonBoard	High-end multimedia & AI – autonomous systems, robotics
Rockchip RK3566 / RK3588	Cortex-A76/A55 + NPU	Radxa Rock 5	High-performance AI + video – edge inferencing, multimedia
Allwinner D1s / A40i	RISC-V / Cortex-A7	LicheePi, Banana Pi	Low-cost Linux-ready – touch panels, entry HMI
Intel Atom / Celeron	x86_64	UP Squared, LattePanda	Full OS, legacy support – industrial PCs, gateways

Microprocessor and Microcontroller (Cont..)

Nvidia Jetson Orin Nano



Example SBC (Raspberry Pi 5)



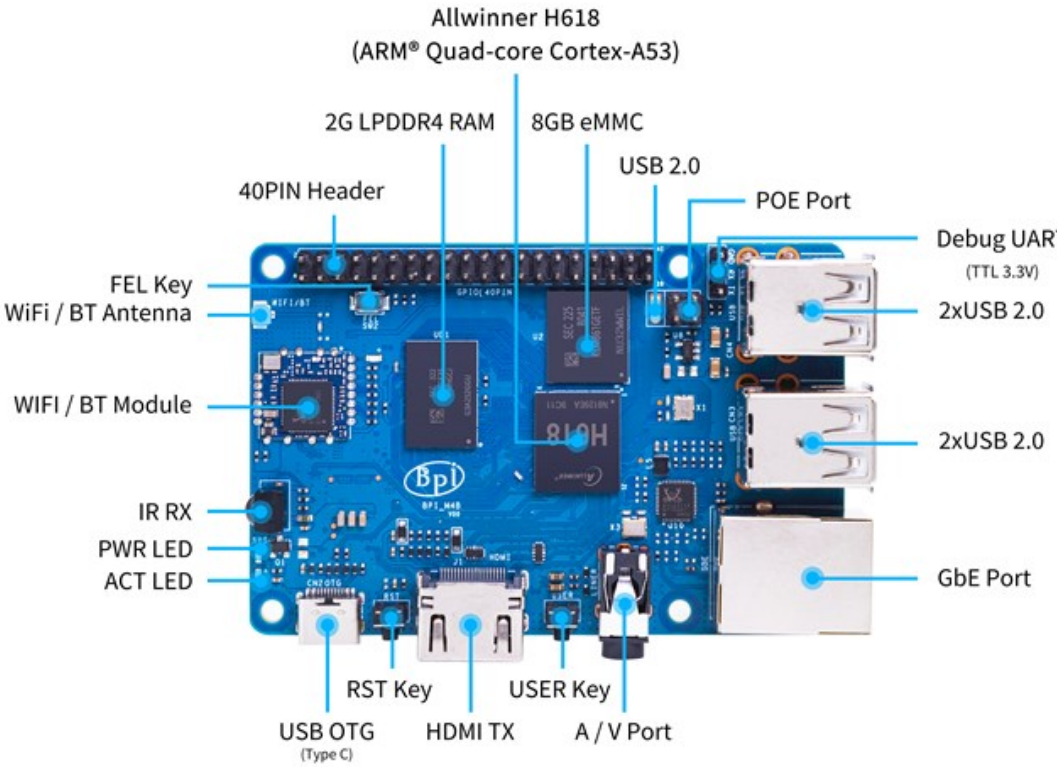
Microprocessor and Microcontroller (Cont..)

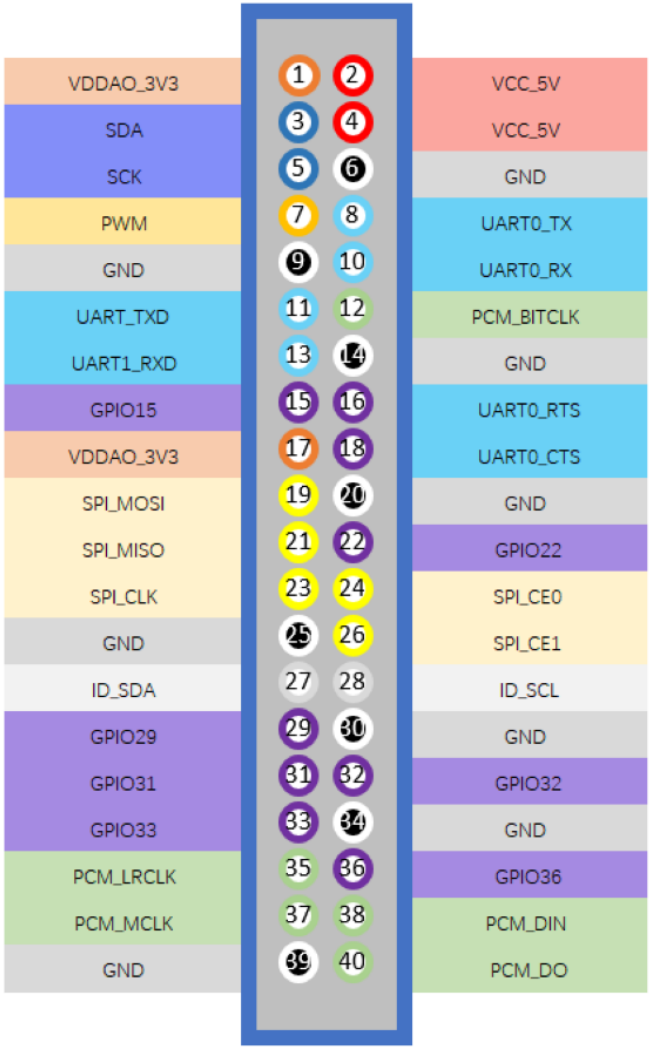
Example - Raspberry Pi 5 - Pinouts

		Physical Pins			
Function	BCM	pin#	pin#	BCM	Function
3.3 Volts		1	2		5 Volts
GPIO/SDA1 (I2C)	2	3	4		5 Volts
GPIO/SCL1 (I2C)	3	5	6		GND
GPIO/GCLK	4	7	8	14	TX UART/GPIO
GND		9	10	15	RX UART/GPIO
GPIO	17	11	12	18	GPIO
GPIO	27	13	14		GND
GPIO	22	15	16	23	GPIO
3.3 Volts		17	18	24	GPIO
MOSI (SPI)	10	19	20		GND
MISO(SPI)	9	21	22	25	GPIO
SCLK(SPI)	11	23	24	8	CEO_N (SPI)
GND		25	26	7	CE1_N (SPI)
RESERVED		27	28		RESERVED
GPIO	5	29	30		GND
GPIO	6	31	32	12	GPIO
GPIO	13	33	34		GND
GPIO	19	35	36	16	GPIO
GPIO	26	37	38	20	GPIO
GND		39	40	21	GPIO

Microprocessor and Microcontroller (Cont..)

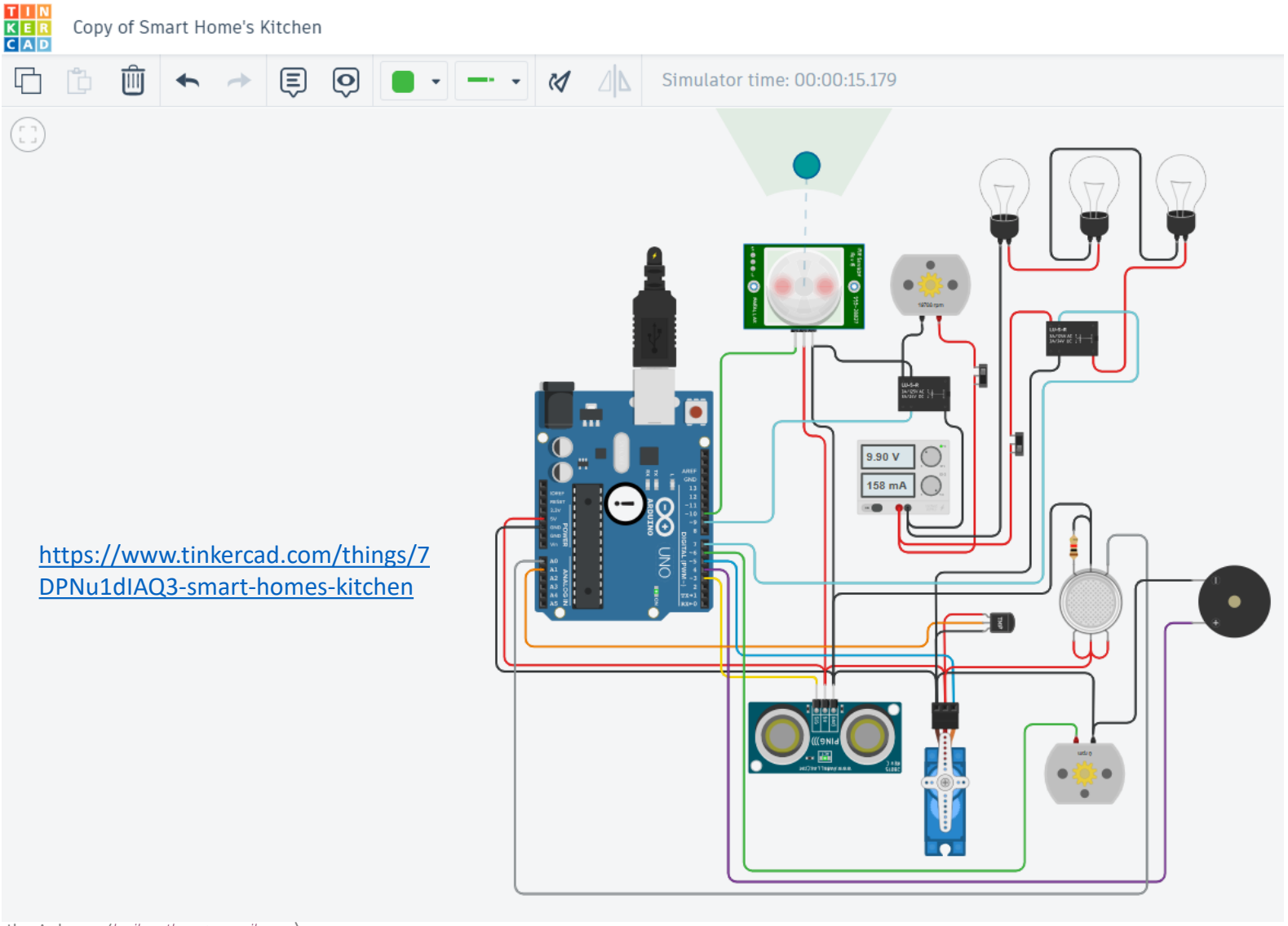
Example SBC (Banana Pi BPI-M4 Berry)





Microprocessor and Microcontroller (Cont..)

Example: Simple Simulation Tool



https://www.tinkercad.com/things/7DPNu1dIAQ3-smart-homes-kitchen

Microprocessor and Microcontroller (Cont..)

Selection Criteria

- ✓ Performance Requirements
  - Processing Speed (Clock frequency).
  - Instruction Set Architecture (ISA), such as Cortex-M0, M3, M4.
  - Need for a Floating Point Unit (FPU) for accelerating calculations.
- ✓ Peripheral Requirements
  - Number and type (Analog, Digital) of GPIOs.
  - ADC resolution and number of channels.
  - Available Timers and PWM channels.
  - Required Communication Interfaces (e.g., UART, SPI, I2C, CAN, USB, Ethernet).
  - Presence of specific hardware (e.g., USB controller).

Microprocessor and Microcontroller (Cont..)

Selection Criteria (Cont..)

- ✓ Memory Requirements
  - Flash memory size (for program storage).
  - SRAM size (for variables and stack).
  - EEPROM size (for persistent storage).
- ✓ Power Consumption and Voltage.
- ✓ Development Environment and Support
  - Availability of IDEs (e.g., Mbed Studio, STM32CubeIDE, Arduino IDE).
  - Availability of software libraries.
  - Community and documentation support.
- ✓ Cost and Availability.
- ✓ Packaging and Physical Constraints.

Microprocessor and Microcontroller (Cont..)

Applications in Instrumentation

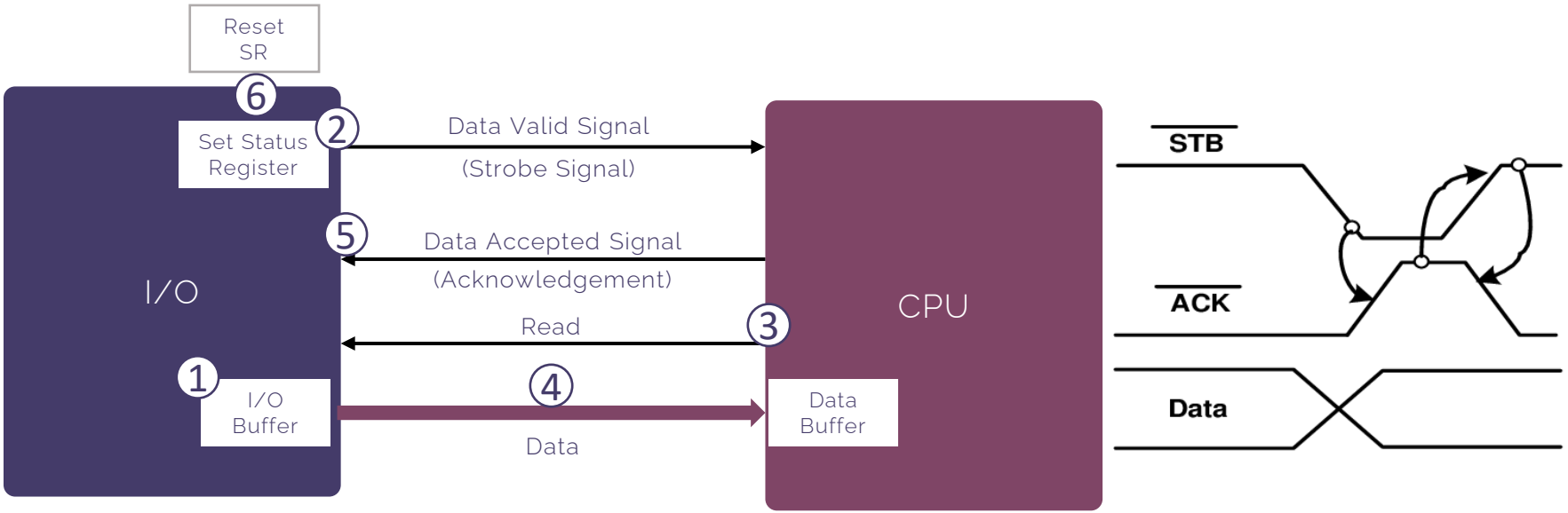
- ✓ Data Acquisition Systems (DAS), Data Logging.
- ✓ Embedded Control Systems (Open/Closed Loop)
  - Closed-loop control in process automation (PID control, event-based triggers).
- ✓ Remote Monitoring and IoT
  - Wireless communication (Wi-Fi, LoRa, BLE) using modules (ESP32, nRF52, etc.).
  - Remote sensor nodes with low-power MCUs.
  - Example: Environmental monitoring systems (air quality, water level sensors).
- ✓ Signal Conditioning and Processing
  - Digital filtering, calibration, linearization of sensor outputs.
  - Example: Filtering ECG signals in biomedical instruments.
- ✓ Industrial Automation.
- ✓ Smart Systems (e.g., Smart Home, Smart Street Lighting, Irrigation Systems).

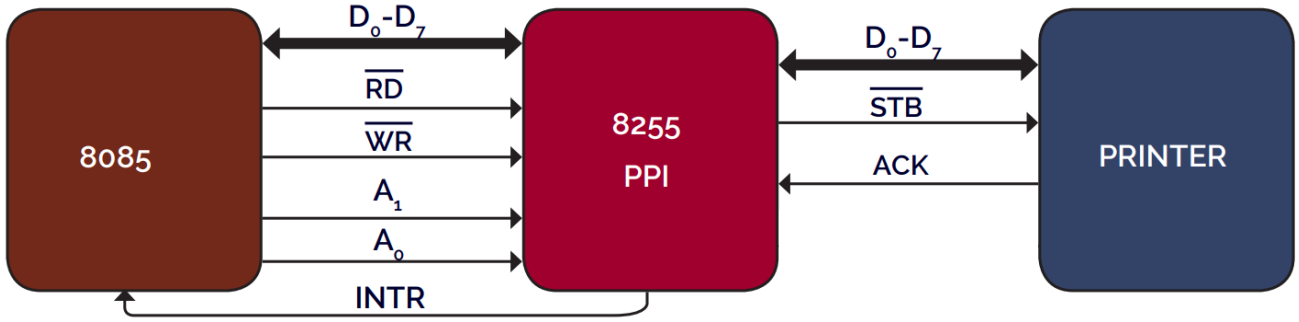
Brief History of I/O Interfacing

- ✓ Intel 8085 Microprocessor (*March 1976*)
  - **Features:** 8-bit, +5V supply, integrated clock, serial I/O, 64 KB memory, 3–5 MHz clock, five interrupts.

- ✓ Intel 8255 PPI (*Early 1970s (circa 1974–1975)*).
  - **History**: Designed for 8080/8085, enabled peripheral interfacing (e.g., keyboards, LEDs). Enhanced 8255A (1980s) improved drive. Obsolete by 2000s, replaced by microcontrollers, but 82C55 used in niches.
  - **Need**: Early microprocessors lacked sufficient I/O; 8255 provided programmable, flexible interfacing with ***handshaking mechanism***, simplifying hardware design.

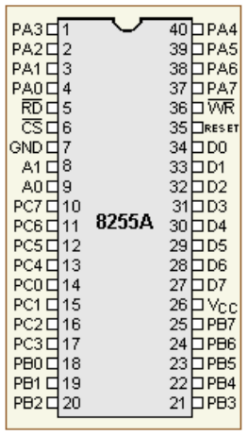
Concept of Handshake Transmission

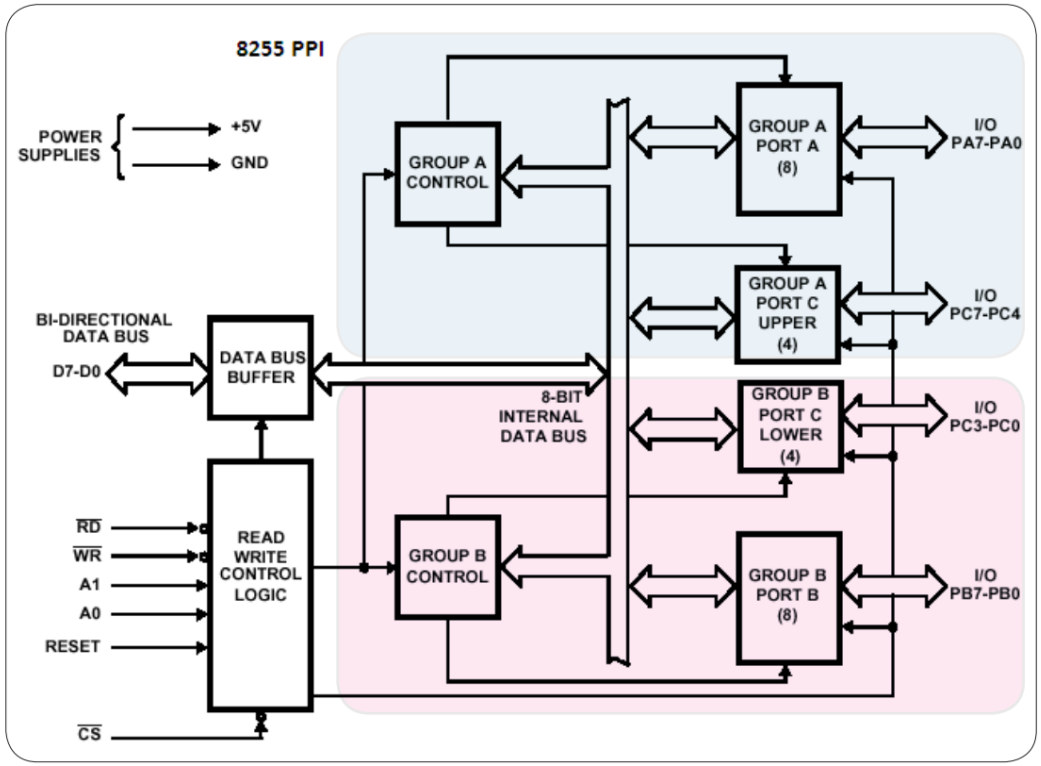




8255 as a General Purpose Programmable I/O

- ✓ PPI 8255 is a general purpose programmable I/O device:
  - designed to interface the CPU with its peripherals with configurable parallel port.
  - used for synchronized parallel data transfer between processor and slow peripheral such as ADC, DAC, keyboard etc.





8255 as a General Purpose Programmable I/O

- ✓ Has 3-configurable parallel I/O ports - can be used as either input or output.
  - Port-A (8-bit)
  - Port-B (8-bit)
  - Port-C (8-bit): - Port-C upper (PC4-PC7) - 4bit, Port-C lower (PC0-PC3) - 4bit.
- ✓ 8-bit data bus buffer:
  - Data, control words and status information are transferred through the data buffer.
- ✓ The function of the Read and Write Control Logic block is:
  - to manage all of the internal and external transfers of data, control or status words.
- ✓ A0 and A1 :
  - controls the selection of one of the three ports or the control word register.
  - 00 → Port A, 01 → Port B 10 → Port C, 11 → Control Register

8255 as a General Purpose Programmable I/O

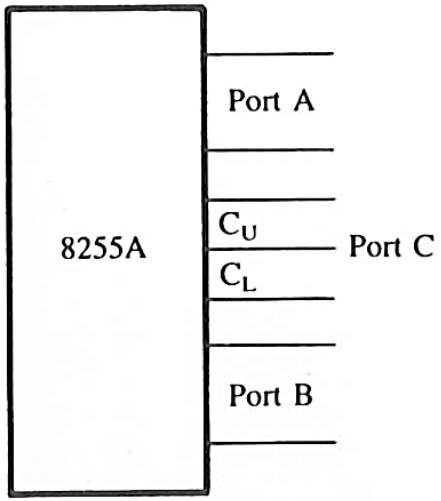
- ✓ The three ports are further grouped as follows:
  - Group A: consisting of port A and upper part of port C.
  - Group B: consisting of port B and lower part of port C.

A_0	A_1	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input-Output Operation
0	0	0	1	0	PORT A --> µP Data bus
0	1	0	1	0	PORT B --> µP Data bus
1	0	0	1	0	PORT C --> µP Data bus
0	0	1	0	0	µP Data bus --> PORT A
0	1	1	0	0	µP Data bus --> PORT B
1	0	1	0	0	µP Data bus --> PORT C
1	1	1	0	0	µP Data bus --> Control Word

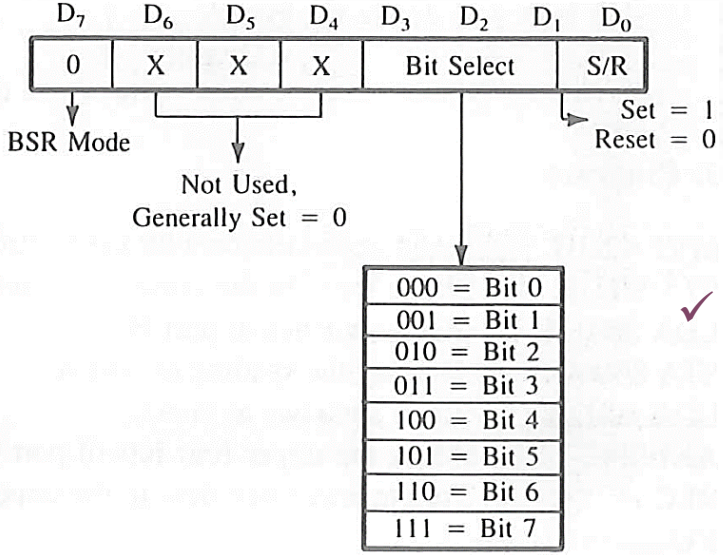
- ✓ Operational modes of 8255 :
  - Bit Set/Reset mode (BSR mode).
  - Input/Output mode (I/O mode).

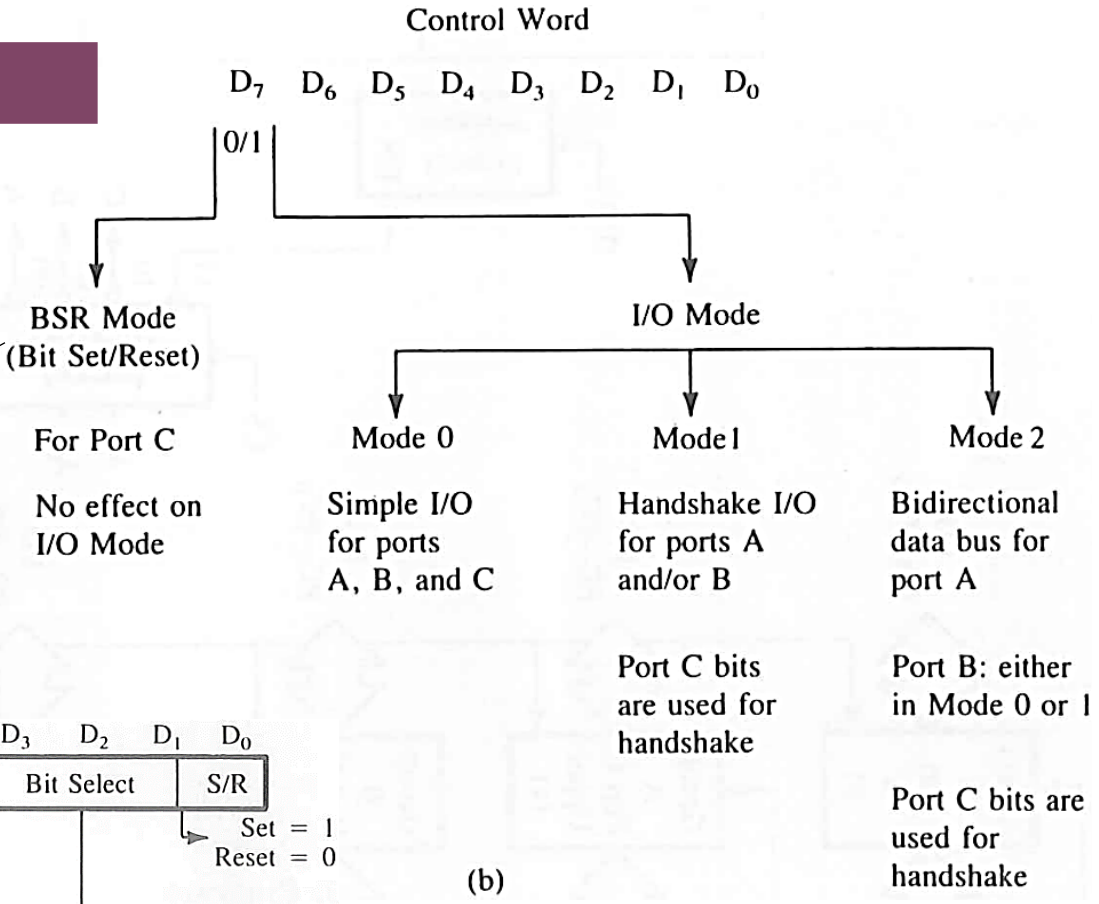
Operational Modes of 8255

Bit Set/Reset (BSR) Mode



(a)





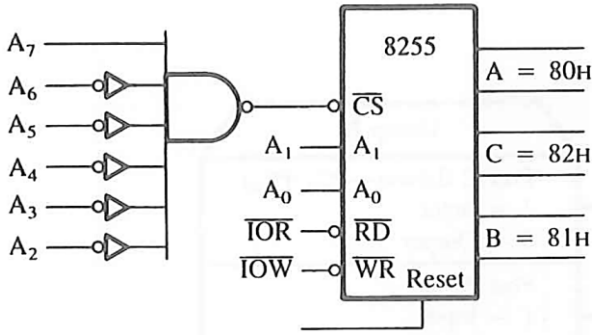
(b)

✓ Problem 1:

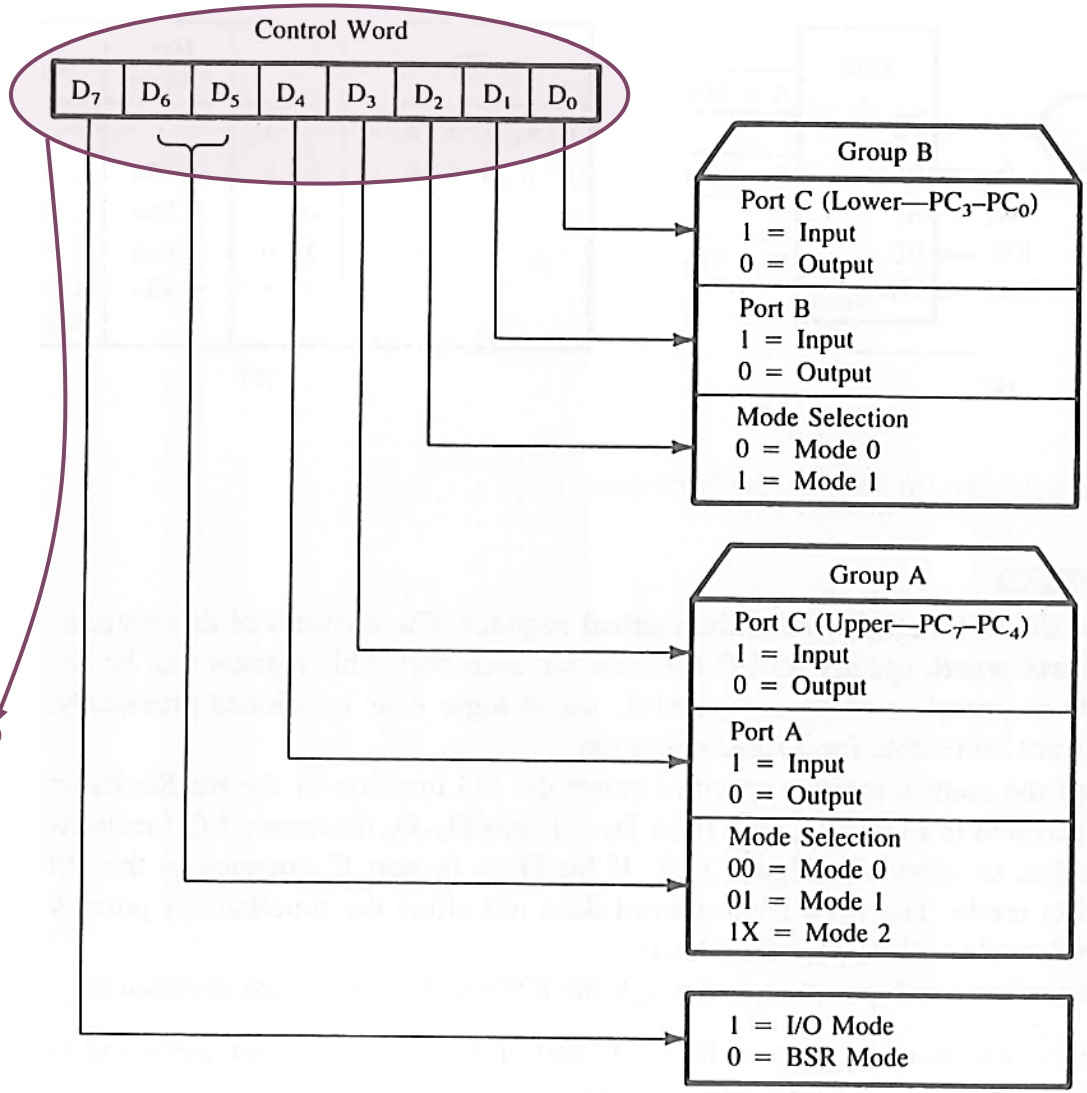
- Write Control word to set PC5 in BSR mode.

Operational Modes of 8255 (Cont..)

Input/Output (I/O) Mode



$\overline{CS}$			Hex Address	Port
A_7 A_6 A_5 A_4 A_3 A_2	A_1	A_0		
1 0 0 0 0 0	0	0	= 80H	A
	0	1	= 81H	B
	1	0	= 82H	C
	1	1	= 83H	Control Register

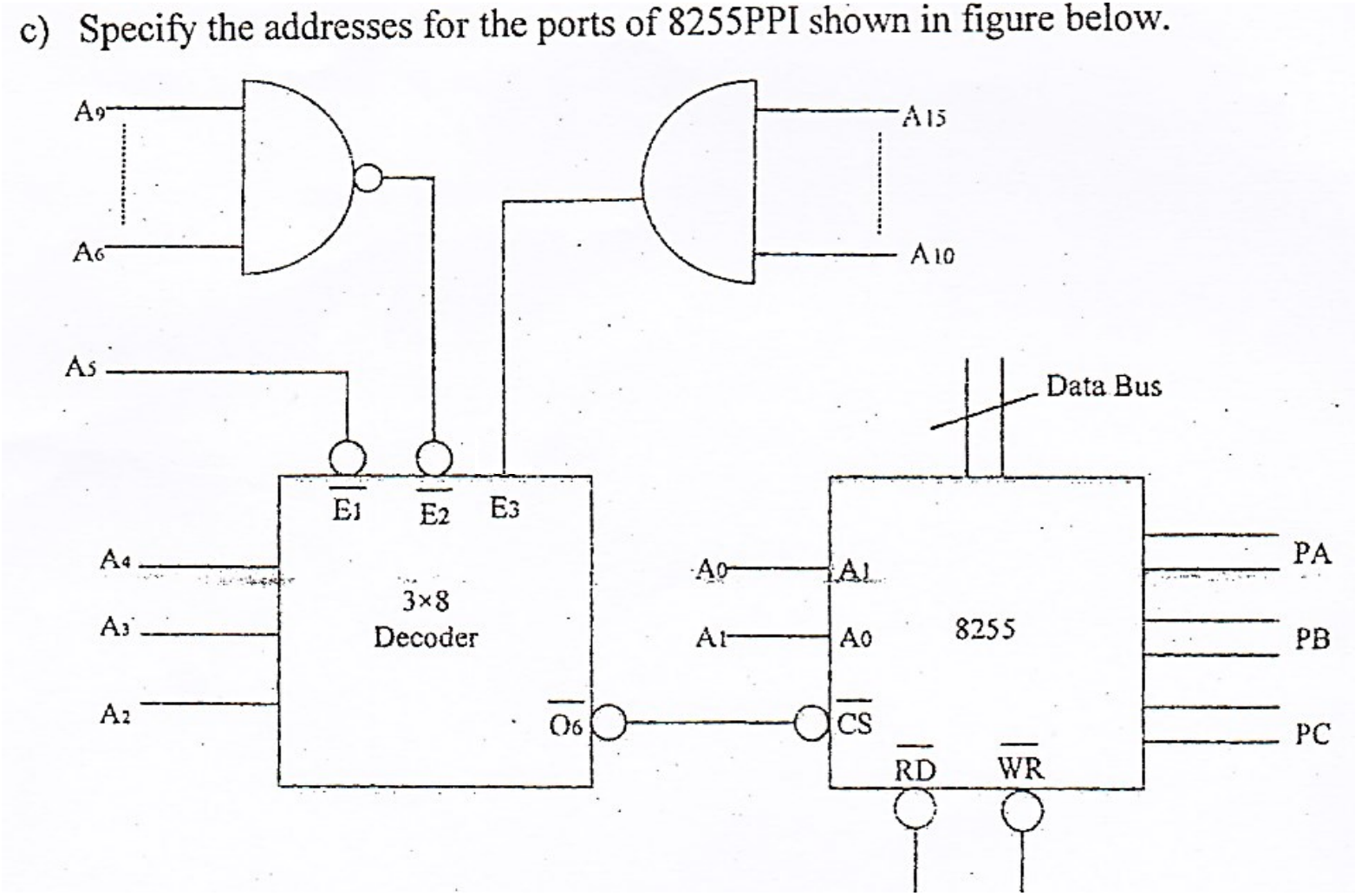


✓ Problem 2:

- Write control word for port-A as i/p, port-B as o/p in mode 1.

Port Address Calculation

c) Specify the addresses for the ports of 8255PPI shown in figure below.



✓ See class note for the solution

I/O Mode

Mode 0 - Simple I/O

- ✓ Allows the programming of each port as either input or output port.
- ✓ The two halves of port C are independent and can be:
  - either used together as an additional 8-bit port.
  - or they can be used as individual 4-bit ports.
- ✓ It does not support handshaking or interrupt capability.
- ✓ The input ports are buffered while outputs are latched.

Programming in 8085 with 8255

- ✓ WAP to enable alternate LED at port-B in mode 0.
  - MVI A,80H **1**0000**00**0 (Control word)
  - OUT 03H 0000000**11** (Address of control register)
  - MVI A, 55H 0**1**0**1**0**1**0**1** (Alternate LEDs)
  - OUT 01H 0000000**01** (Address of port-B)
  - HLT

Programming in 8085 with 8255

✓ *WAP to blink a LED connected in PC_0.*

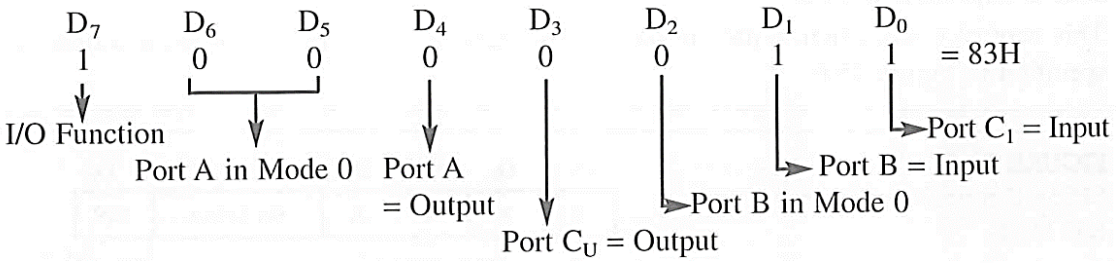
```
MVI A, 80H
OUT F3H
LOOP: MVI A, 01H
OUT F3H
CALL DELAY
MVI A, 00H
CALL DELAY
JMP LOOP
HLT
```

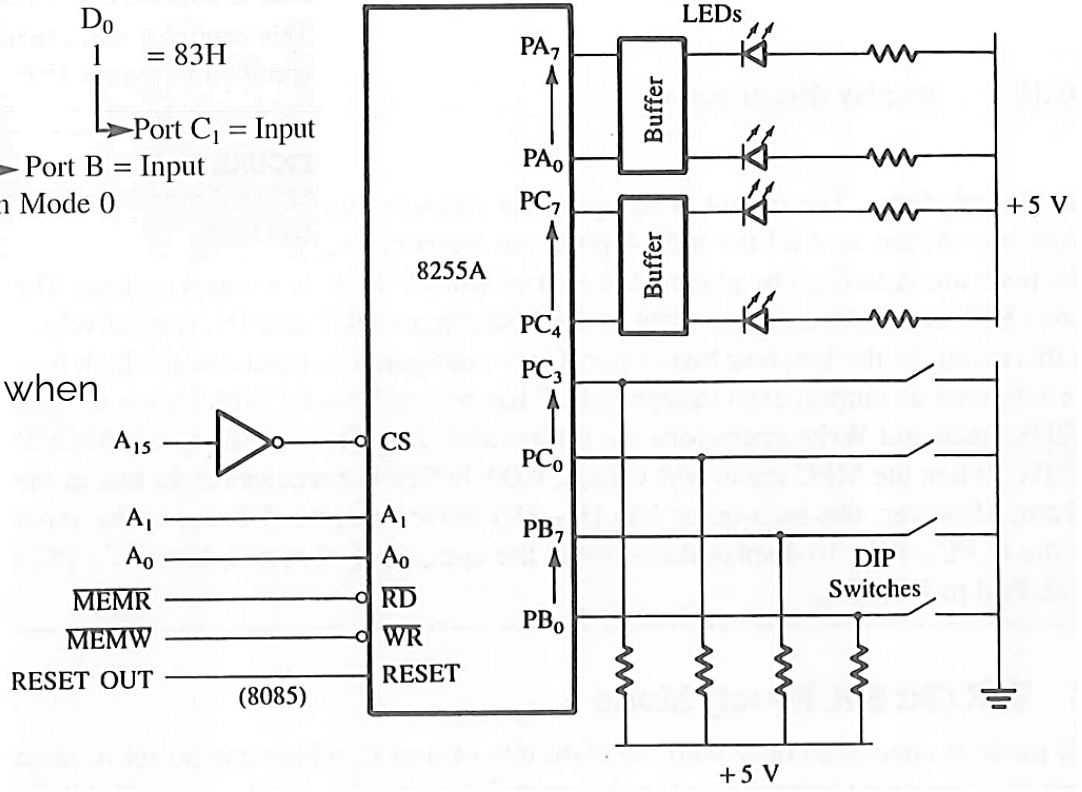
```
; Delay Subroutine
DELAY: MVI B, FFH
ILOOP2: MVI C, FFH
ILOOP1: DCR C
JNZ ILOOP1
DCR B
JNZ ILOOP2
RET
```

I/O Mode (Cont..)

Mode 0 - Simple I/O - Example

- ✓ Write a program to read the DIP switches and display the reading from port B at port A and from port C_L at port C_U.





Port Address: This is a memory-mapped I/O; when the address line A15 is high.

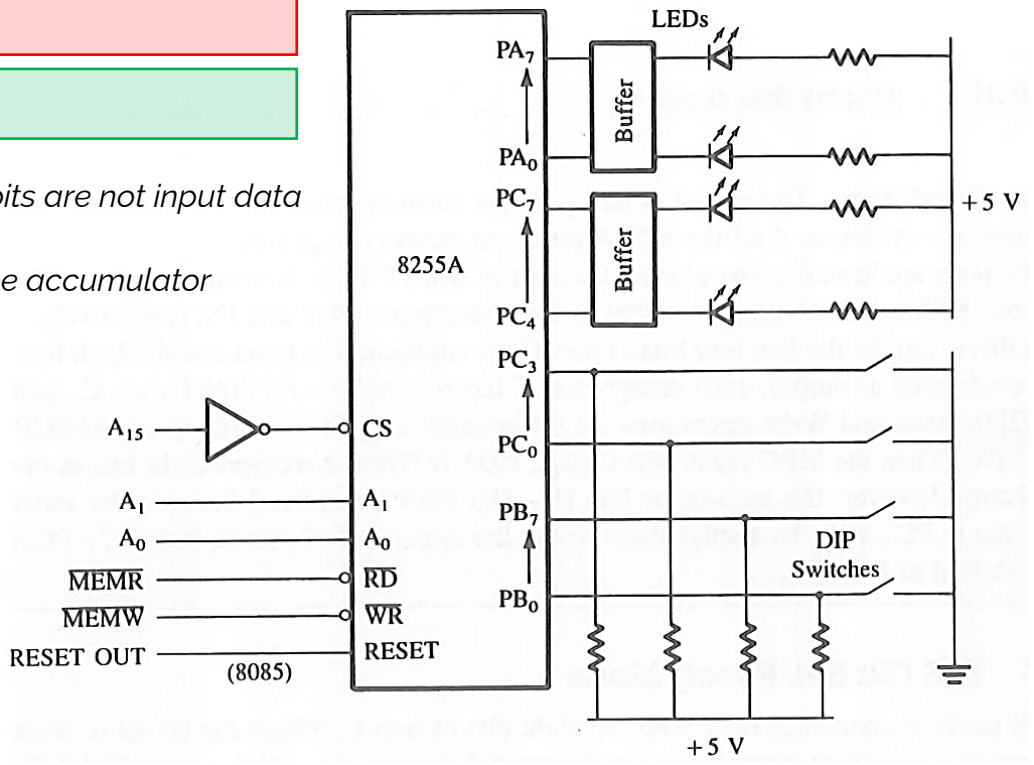
Port A	=	8000H ($A_1 = 0, A_0 = 0$)
Port B	=	8001H ($A_1 = 0, A_0 = 1$)
Port C	=	8002H ($A_1 = 1, A_0 = 0$)
Control Register	=	8003H ($A_1 = 1, A_0 = 1$)

I/O Mode (Cont..)

Mode 0 - Simple I/O - Example

```
MVI A, 83H  ;Load accumulator with the control word
STA 8003H   ;Write word in the control register to initialize the ports
LDA 8001H   ;Read switches at port B
STA 8000H   ;Display the reading at port A
LDA 8002H   ;Read switches at port C
ANI 0FH     ;Mask upper four bits of port C; these bits are not input data
RLC         ;Rotate and place data in the upper half of the accumulator
RLC
RLC
RLC
STA 8002H   ;Display data at port C_U
HLT
```

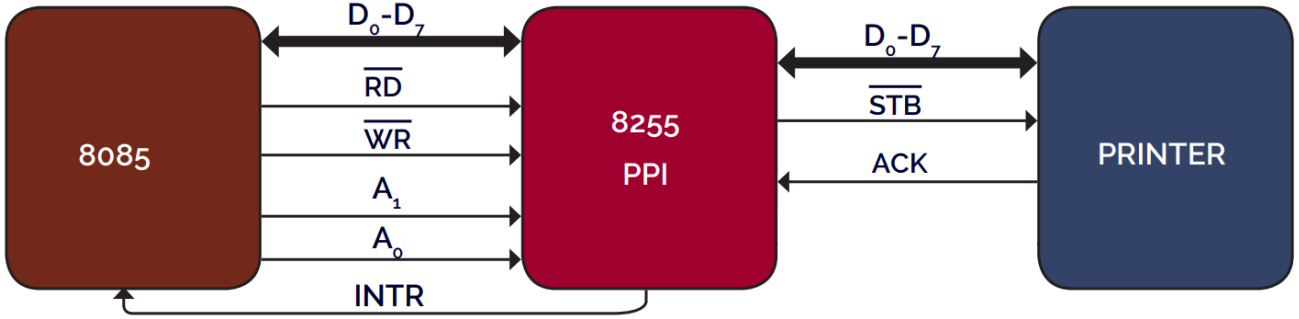
Minimum configuration circuit concept



I/O Mode (Cont..)

Mode 1 – Handshake/Strobed I/O Mode

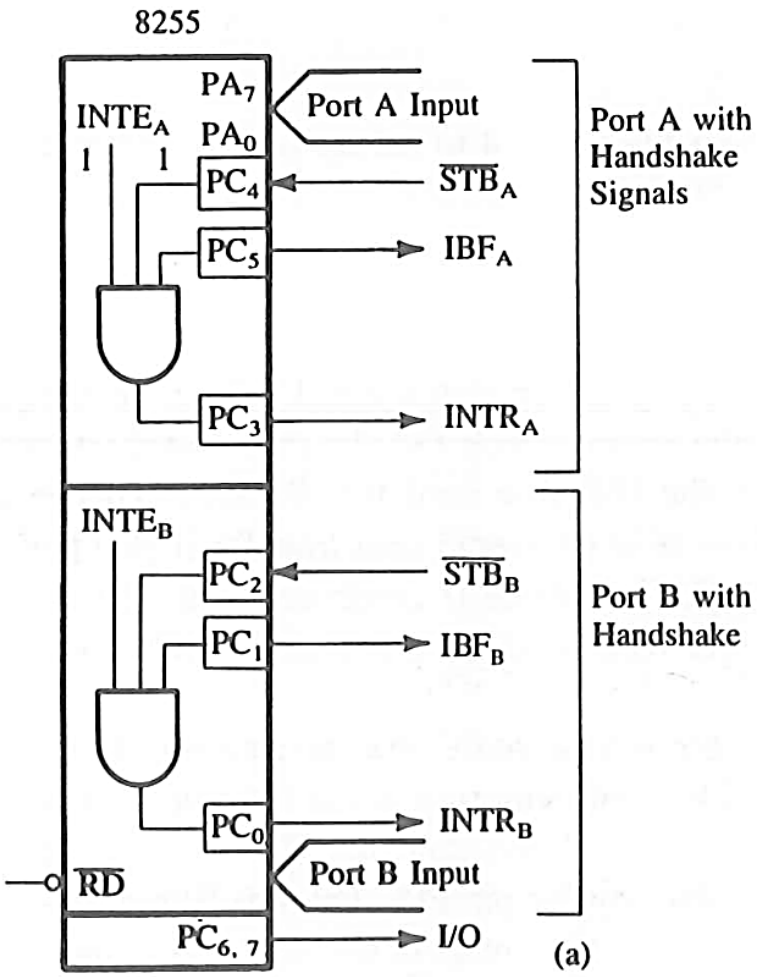
- ✓ Either port A or port B can work as input or output port.
- ✓ Port C bits are used for handshake signals.
- ✓ It has interrupt handling capacity and input and output are latched.
- ✓ Example: A CPU wants to transfer data to a printer
  - Speed of processor is very fast as compared to relatively slow printer
  - So, synchronization is required with either **strobed** or **interrupt I/O.**



I/O Mode (Cont..)

Mode 1 – Handshake/Strobed I/O Mode

Input Control Signals



Status Word—Mode 1 Input

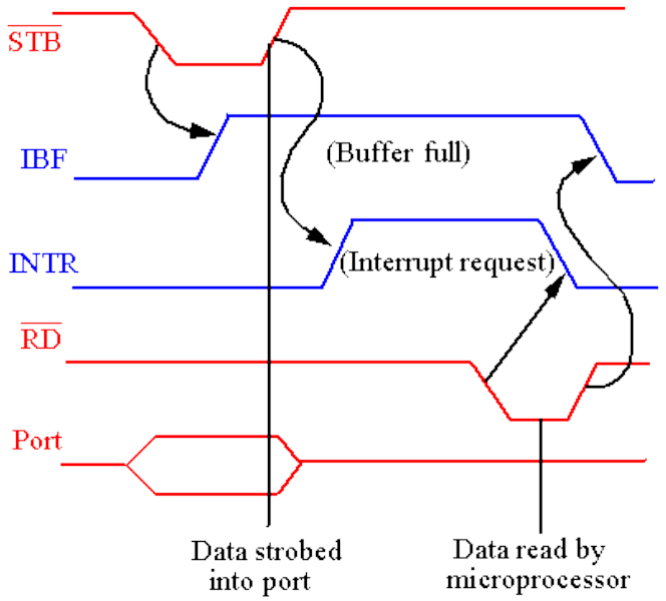
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$

Control Word—Mode 1 Input

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	1	1/0	1	1	X

I/O Mode
Port A Mode 1
Port A Input
Port B Input
Port B Mode 1
$PC_{6,7}$
1 = Input
0 = Output

Timing Diagram



I/O Mode (Cont..)

Mode 1 – Handshake/Strobed I/O Mode

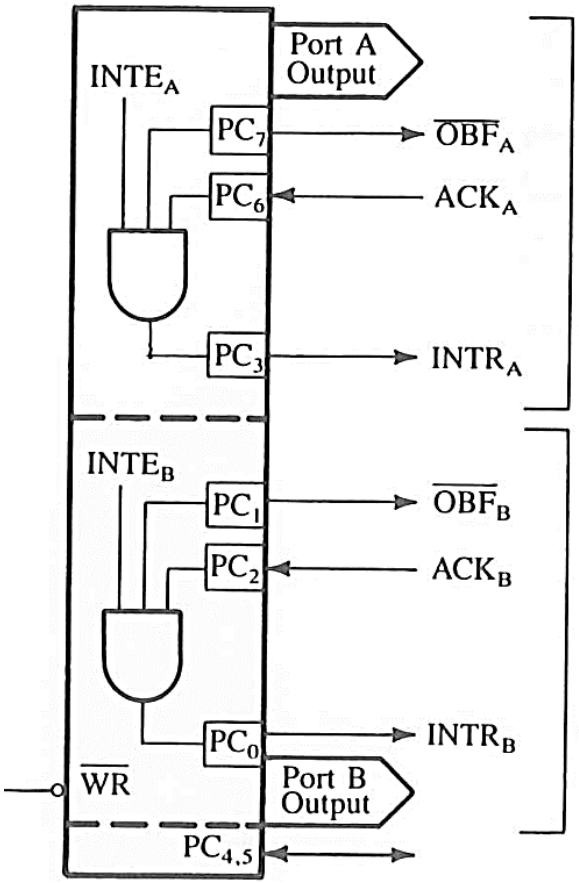
Input Control Signals (Cont..)

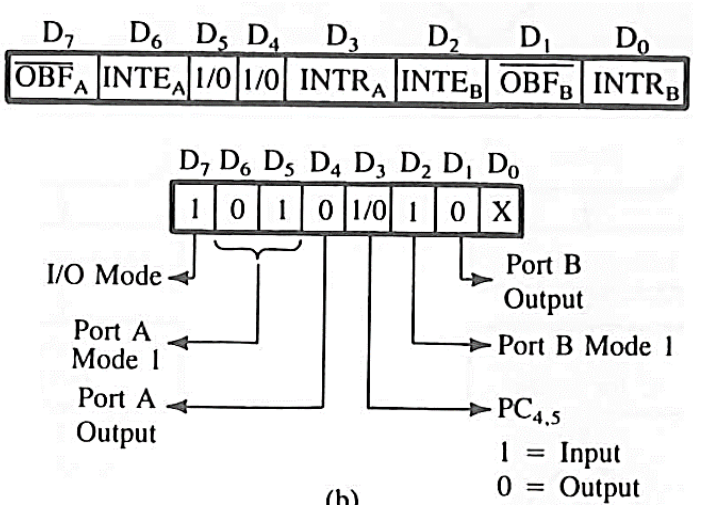
- ✓ STB' (Strobed Input)
  - Active low signal generated by peripherals to indicate that it has transmitted a data.
  - The 8255A, in response to STB' generates IBF and INTR.
- ✓ IBF (Input Buffer Full)
  - Acknowledgement by 8255A input latch has received the data bytes.
  - Resets when microprocessor reads the data
- ✓ INTR (Interrupt request)
  - Output signal for interrupting microprocessor
  - Generated if: STB', IBF, and INTE are all at logic 1, Resets on falling edge of RD' signal.
- ✓ INTE (Interrupt enable): programmed via port PC_4(port A) or PC_2(port B) bit position.

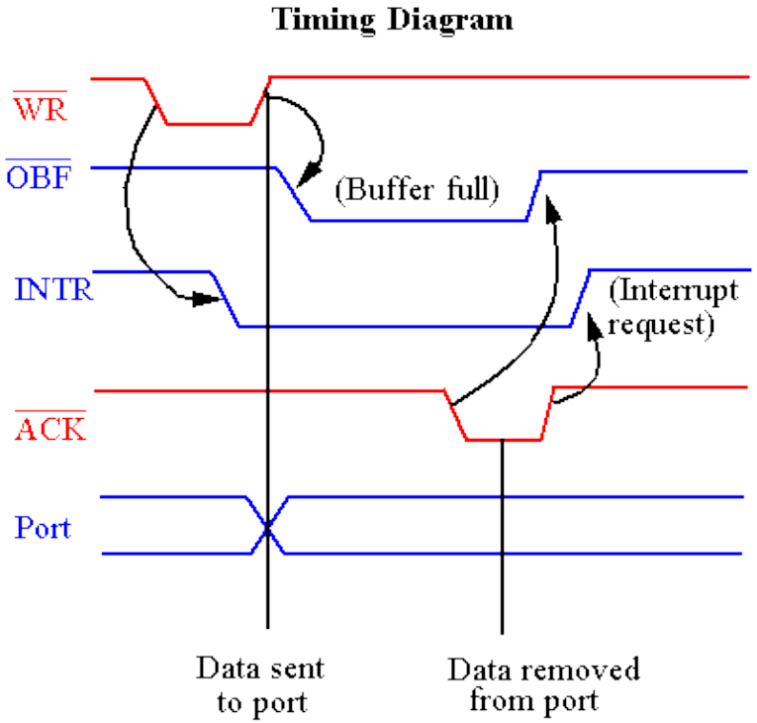
I/O Mode (Cont..)

Mode 1 – Handshake/Strobed I/O Mode

Output Control Signals







Requesting next data

I/O Mode (Cont..)

Mode 1 – Handshake/Strobed I/O Mode

Output Control Signals (Cont..)

- ✓ OBF' (Output Buffer Full)
  - Low: whenever data are output (OUT) to the port A or port B latch.
  - High: whenever the ACK pulse returns from the external device.
- ✓ ACK (Acknowledge)
  - Response from an external device, indicating that it has received the data.

I/O Mode (Cont..)

Mode-1 Example

✓ Concept: Interrupt I/O Vs Status Check I/O

Q. In the given figure below, port A is designed as the input port for a keyboard with interrupt I/O, and port B is designed as the output port for a printer with status check I/O.

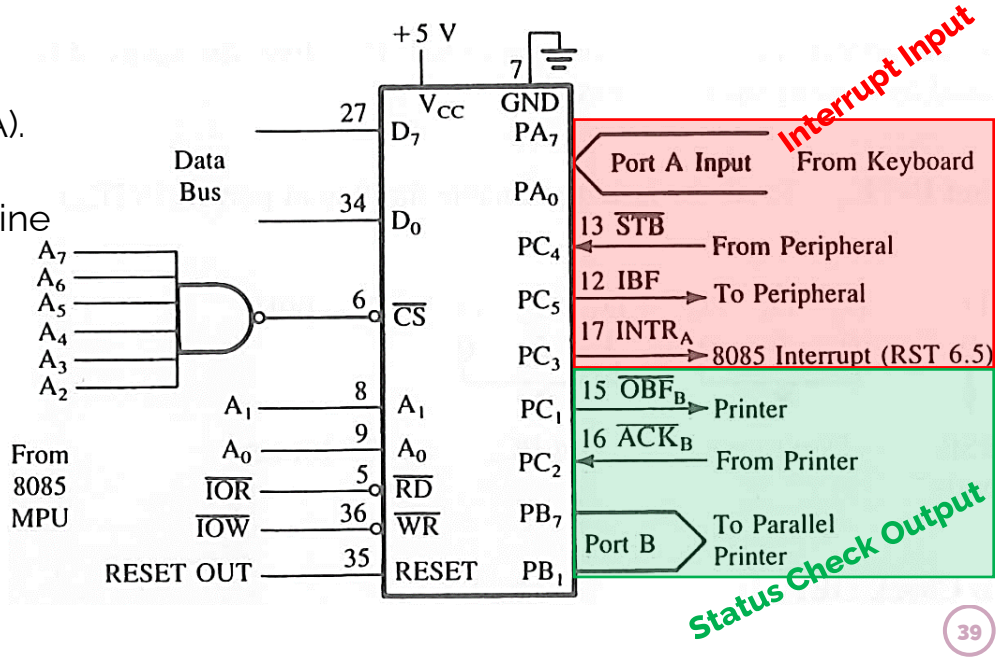
i. Find port addresses by analyzing the decode logic.

ii. Determine the control word to set up port A as input and port B as output in Mode 1.

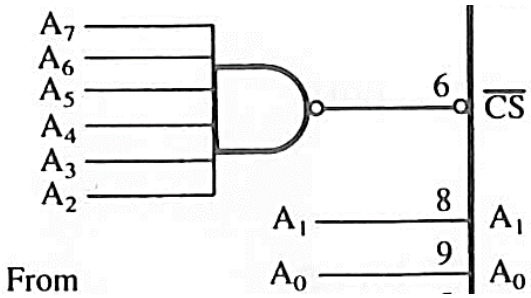
iii. Determine the BSR word to enable $INTE_A$ (port A).

iv. Determine the masking byte to verify the OBF_B line in the status check I/O (port B).

v. Write initialization instructions and a printer subroutine to output characters that are stored in memory.



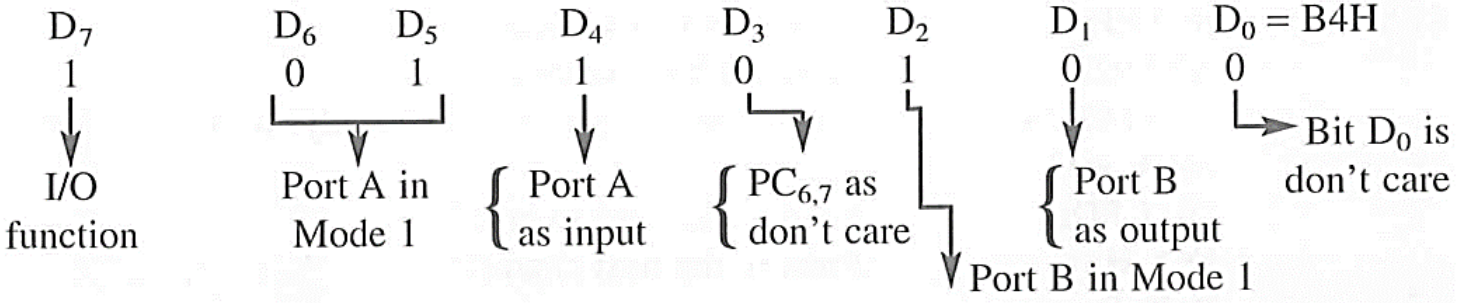
Mode-1 Example (Cont..)



1. Port Addresses The 8255A is connected as peripheral I/O. When the address lines A_7–A_2 are all 1, the output of the NAND gate goes low and selects the 8255A. The individual ports are selected as follows:

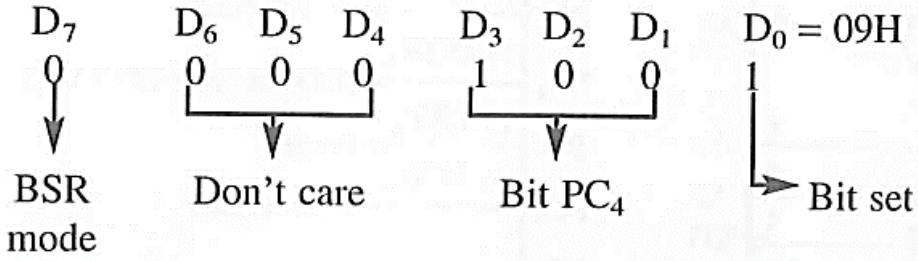
Port A	= FCH ($A_1 = 0$, $A_0 = 0$)
Port B	= FDH ($A_1 = 0$, $A_0 = 1$)
Port C	= FEH ($A_1 = 1$, $A_0 = 0$)
Control Register	= FFH ($A_1 = 1$, $A_0 = 1$)

2. Control Word to Set Up Port A as Input and Port B as Output in Mode 1



Mode-1 Example (Cont..)

3. BSR Word to Set $INTE_A$ To set the Interrupt Enable flip-flop of port A ($INTE_A$), bit PC_4 should be 1.



4. Status Word to Check $\overline{OBF_B}$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
X	X	X	X	X	X	$\overline{OBF_B}$	X

Masking byte: 02H

Mode-1 Example (Cont..)

5. Initialization Program

```
                MVI A,B4H       ;Word to initialize port A as input,
                                ;   port B as output in Mode 1
                OUT FFH
                MVI A,09H       ;Set INTE_A (PC_4)
                OUT FFH         ;Using BSR Mode
                EI              ;Enable interrupts
                CALL PRINT      ;Continue other tasks

Print Subroutine
PRINT:  LXI H,MEM       ;Point index to location of stored characters
        MVI B,COUNT     ;Number of characters to be printed
NEXT:   MOV A,M         ;Get character from memory
        MOV C,A         ;Save character
STATUS: IN FEH          ;Read port C for status of OBF
        ANI 02H         ;Mask all bits except D1
        JZ STATUS       ;If it is low, the printer is not ready; wait in the loop
        MOV A,C
        OUT FDH         ;Send a character to port B
        INX H           ;Point to the next character
        DCR B
        JNZ NEXT
        RET
```

Interrupt Input Setup

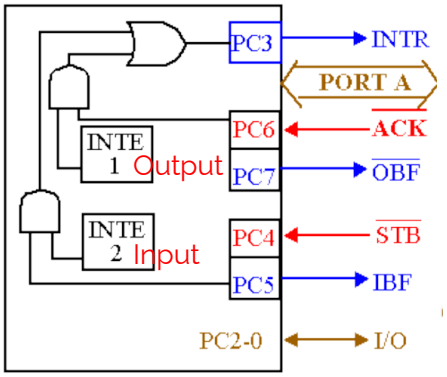
Status Check Output

Assumed: Interrupt ISR stores the input in MEM Location.

```
    LXI D, MEM
ISR:
    PUSH PSW
    IN FCH
    STAX D
    INX D
    POP PSW
    RET
```

I/O Mode (Cont..)

Mode 2: Bidirectional Data Transfer



Mode 2 and Mode 0 (Input)

Control Word

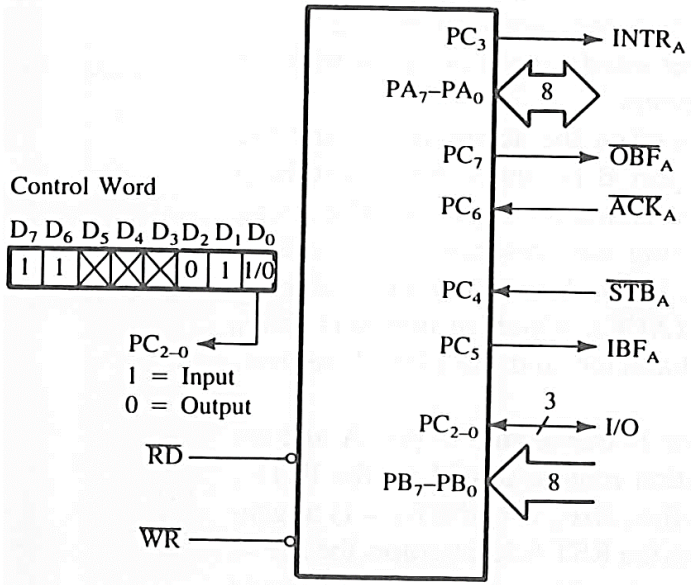
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

1 | 1 | X | X | X | 0 | 1 | I/O

PC_{2-0}

1 = Input

0 = Output

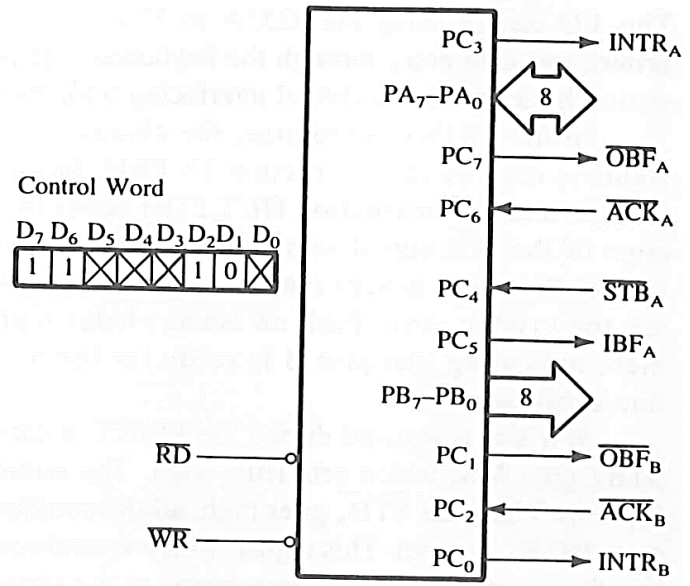


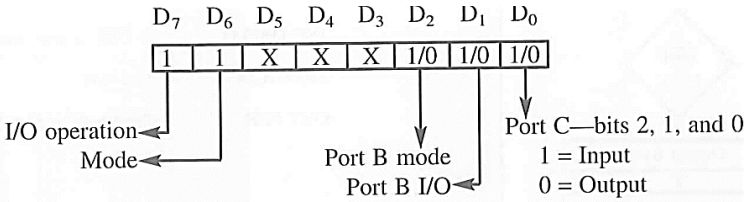
Mode 2 and Mode 1 (Output)

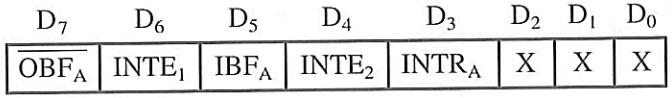
Control Word

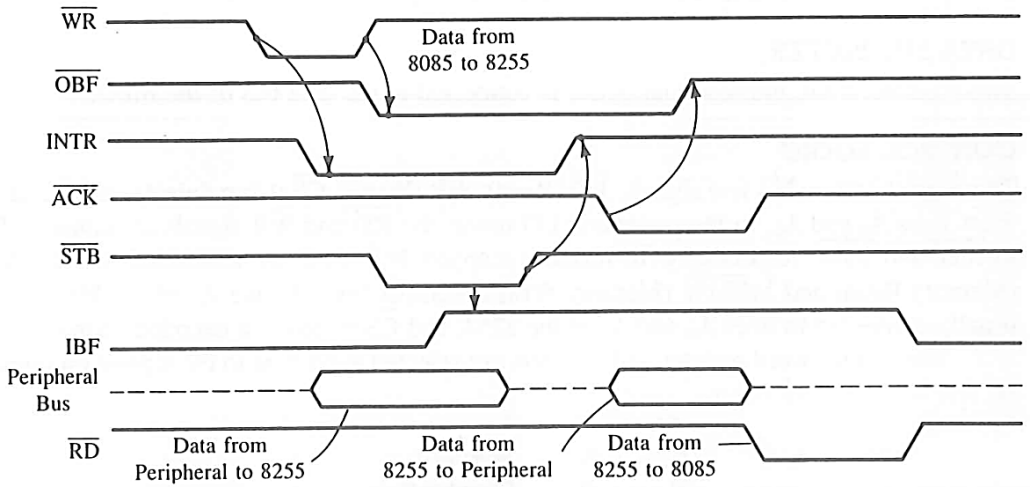
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

1 | 1 | X | X | X | 1 | 0 | X





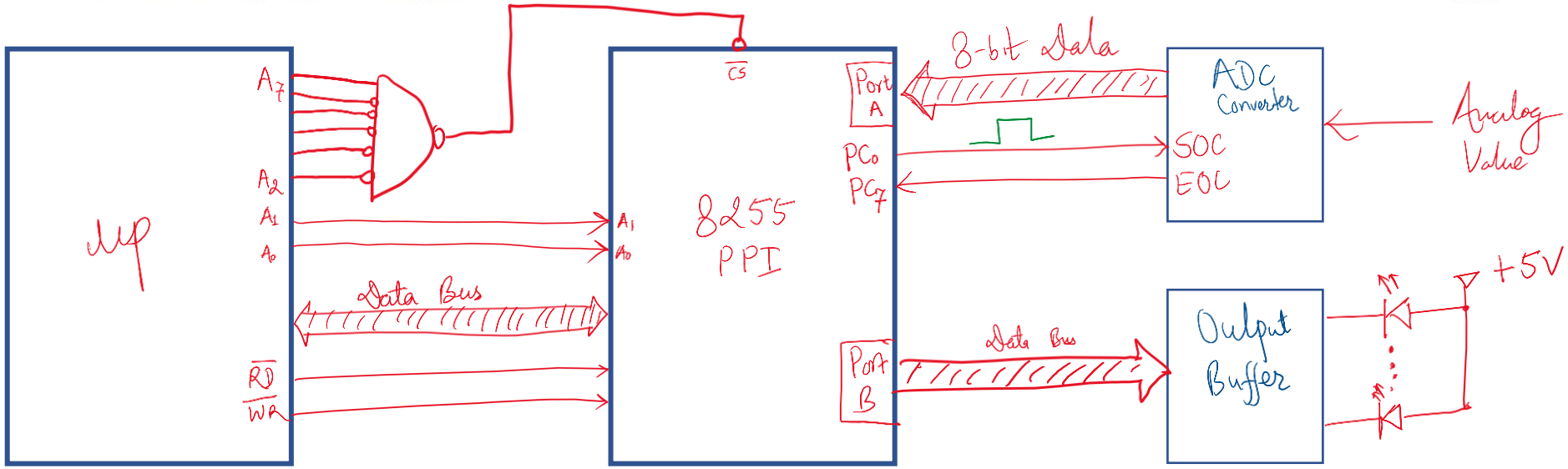




Mode-0 Example (8-bit ADC Interfacing)

2. A/D converter requires signal to start the conversion and indicates with the end of conversion signal. 8255A PPI is interfaced with 8085 microprocessors at 80H. Microprocessor reads 8-bits O/P data of the ADC at port A and display the same data to eight LED's connected at port B of 8255A. State any assumptions made.

a) Identify the address captured by the card [1]
b) Determine the necessary control words [2]
c) Draw the schematic interfacing circuit [2]
d) Write a program to perform the operation [3]



Port Addresses:

Port-A : 1000 0000 ⇒ 80H
Port-B : 10000001 ⇒ 81H
Port-C ⇒ 82H
Control Register ⇒ 83H

Control Word

1 00 1 1 0 0 0 ⇒ 98H

1 — I/O
00 — Mode 0
1 — Port-A Input
1 — PC upper Input
0 — Port-B Output
0 — PC lower Output
Mode 0

Mode-0 Example (Cont..)

```
        MVI A, 98H
        OUT 83H
LOOP:   MVI A, 01H
        OUT 83H
        CALL DELAY
        MVI A, 00H
        OUT 83H
CHECK:  IN 82H
        RAL
        JNC CHECK
        IN 80H
        OUT 81H
        JMP LOOP
        HLT
```

01H → 0000 0001 (BSR Mode); PC0; set

00H → 0000 0000; PC0; Reset

Generates a Pulse for SOC

Read PC for EOC

Check EOC bit = 1 or not.

If EOC, display the reading

do Again

Additional Question

Assume that your group has decided to make a PC based control system for a wine company. After studying the system, your group found out that the following to be implemented for controlling purpose:

- Pressure measurement (6 points)
- Temperature measurement (5 points)
- Weight measurement (1 point)
- Volume measurement for filling (5 points)
- Your group also decided to use 8255A PPI card at base address 0550H.

a) List out collected documents and components.

b) List out different signals you need to derive and or can be directly connected to your interfacing circuit.

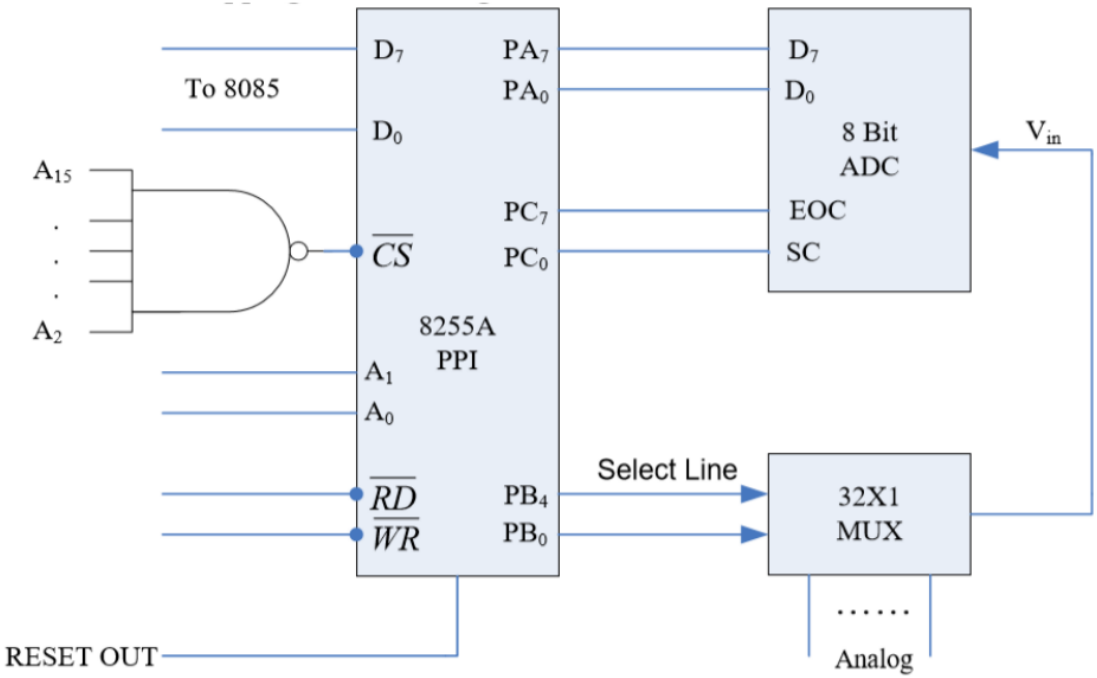
c) Draw minimum mapping circuit for above system.

d) What are the address captured by card.

e) Generate necessary control word.

f) Write a program module for measuring the pressure of all the points and control if the pressure is not in a range. (*Assume suitable data if necessary*)

Additional Question - Solution



Control word to set up port ports in above configuration:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	1	1	0	0	0	=98H

BSR word to set PC_0:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	0	0	0	0	0	0	1	= 01H

BSR word to reset PC_0:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	0	0	0	0	0	0	0	= 00H

a) Components:

- 8255A card, ADC, MUX, Memory, Processor, NAND logic gates.
- Power supplies (+5V, GND), connecting wires PCB Soldering set etc.

Documents:

- Data sheets and technical documentation of above components.
- CIrcuit schematic diagram.

b) Signals needed to be derived on directly connected to circuit:

- A_1, A_2, Chip Select (CS) for Port selection of of 8255A,
- RESET signal ⍰ Read ($\overline{RD}$) and Write ($\overline{WR}$) signals.
- Start Conversion (SC) and End of Conversion (EOC)

Additional Question – Solution (Cont..)

e) Control word and BSR words:

Control word to set up port ports in above configuration:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	0

=98H

BSR word to set PC_0:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	1

= 01H

BSR word to reset PC_0:

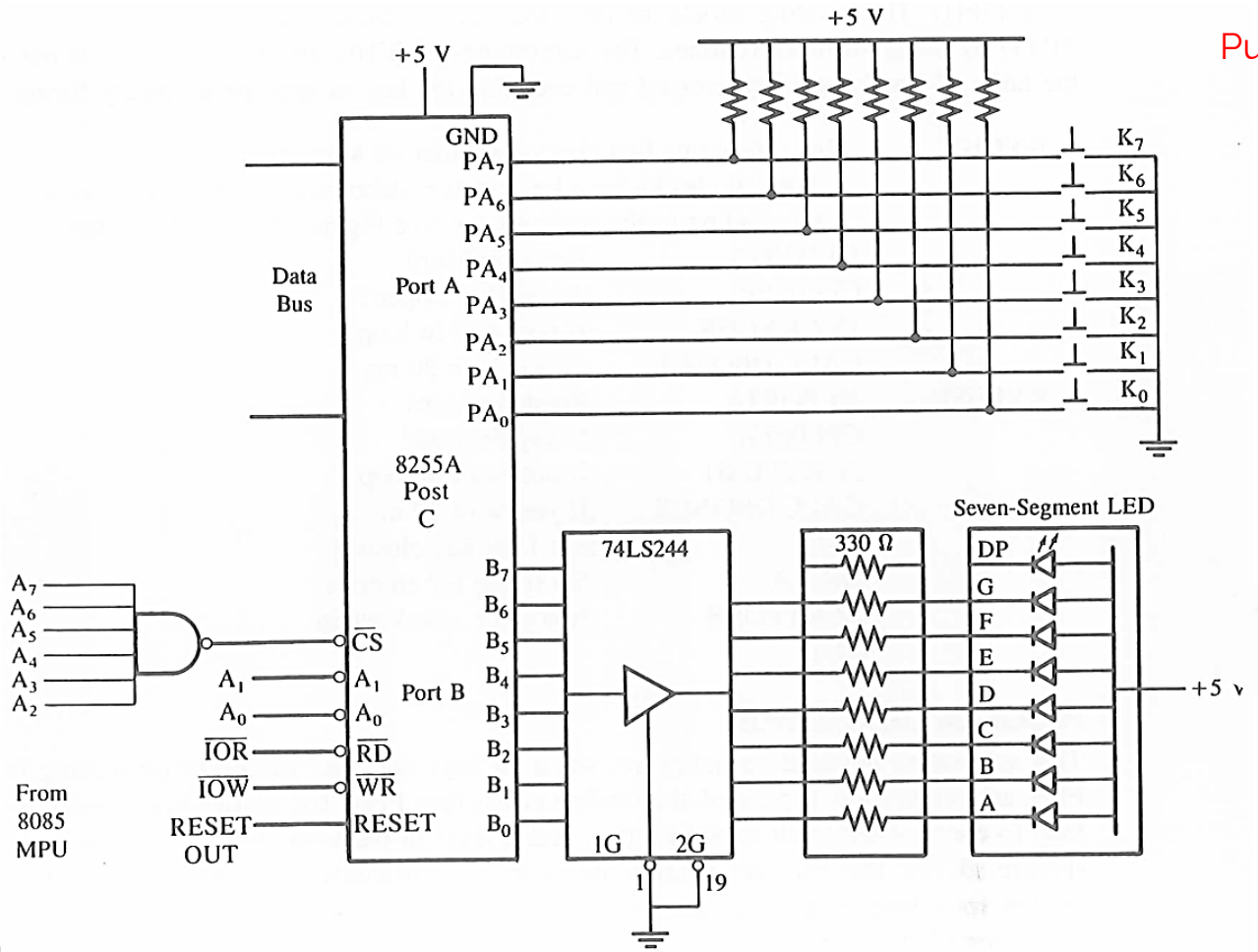
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0

= 00H

```
LXI H, MEMORY
MVI A, 98H
STA 0553H        ;write control word in CR
MVI C, 06H       ;set counter to read total 6 pressure points
MVI B, 00H       ;selection of points for MUX
NEXT: MOV A, B
STA 0551H        ;select first pressure point
MVI A, 01H       ;load A with BSR word to set PC0
STA 0553H        ;set SC line
CALL DELAY
MVI A, 00H       ;load A with BSR word to reset PC0
STA 0553H        ;reset SC line
READ: LDA 0552H  ;read port C
RAL
JNC READ         ;check for PC7
LDA 0550H        ;read data from port A
MOV M, A         ;store value in memory
CPI MAX_VALUE    ;compare with maximum value
JNC CONTROL      ;control value
CPI MIN_VALUE    ;compare with minimum value
JC CONTROL       ;control value
INR B
INX H
DCR C
JNZ NEXT
HLT
```

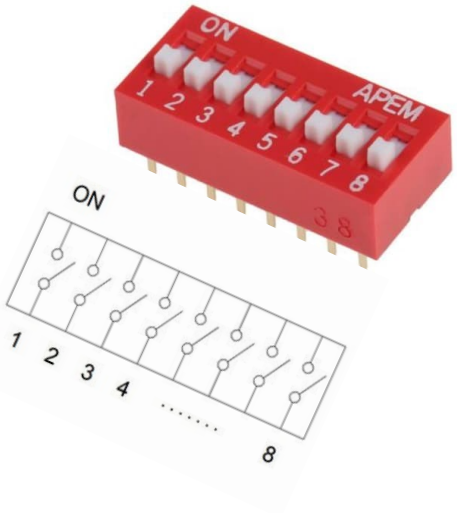
Interfacing of Switches & 7SEGMENT Display

- ✓ Write a program using 8255A to read a keypress from a pushbutton keyboard on port A and display the corresponding digit on a seven-segment LED connected to port B, using simple I/O Mode 0.



Push Button:

DIP Switches



Interfacing of Switches & 7SEGMENT Display

Main Program

```
KYBORD: ;This program initializes the 8255A ports; port A and port B in Mode 0
        ;   and then calls the subroutine modules discussed
        ;   previously to monitor the keyboard
PORTA   EQU FCH          ;Port A address
PORTB   EQU FDH          :Port B address
CNTRL   EQU FFH          ;Control register
CNWORD  EQU 90H          ;Mode 0 control word, port A input and port B output
STACK   EQU 20AFH        ;Beginning stack address
        LXI SP,STACK
PPI:    MVI A,CNWORD
        OUT CNTRL        ;Set up port A in Mode 1
NEXTKY: CALL KYCHK       ;Check if a key is pressed
        CALL KYCODE      ;Encode the key
        CALL DSPLAY      ;Display key pressed
        JMP NEXTKY       ;Check the next key pressed
```

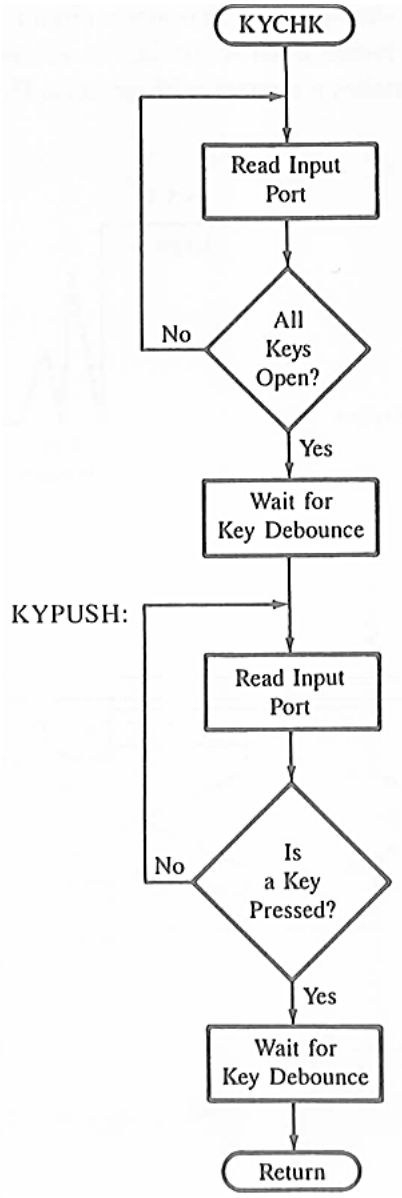
Interfacing of Switches & 7SEGMENT Display

```
KYCHK:   ;This subroutine first checks whether all keys are open.
         ;   Then, it checks for a key closure, debounces the key, and places
         ;   the reading in the accumulator. See Figure 15.16 for flowchart.
         IN PORTA        ;Read keyboard
         CPI 0FFH        ;Are all keys open?
         JNZ KYCHK       ;If not, wait in loop
         CALL DBONCE     ;If yes, wait 20 ms
KYPUSH:  IN PORTA        ;Read keyboard
         CPI 0FFH        ;Is key pressed?
         JZ KYPUSH       ;If not, wait in loop
         CALL DBONCE     ;If yes, wait 20 ms
         CMA             ;Set 1 for key closure
         ORA A           ;Set 0 flag for an error
         JZ KYPUSH       ;It is error, check again
         RET
```

Waiting for previous key to release!

Waiting for any new key pressed!

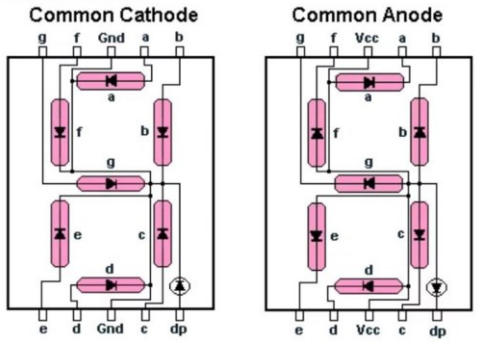
Verify the actual input




```
KYCODE:  ;This routine converts (encodes) the binary hardware reading of the key
         ;   pressed into appropriate binary format according to the number of the
         ;   key.
         MVI C,08H       ;Set code encounter
NEXT:    DCR C           ;Adjust key code
         RAL             ;Place MSB in CY
         JNC NEXT        ;If bit = 0, go back to check next bit
         MOV A,C         ;Place key code in the accumulator
         RET
```

Interfacing of Switches & 7SEGMENT Display

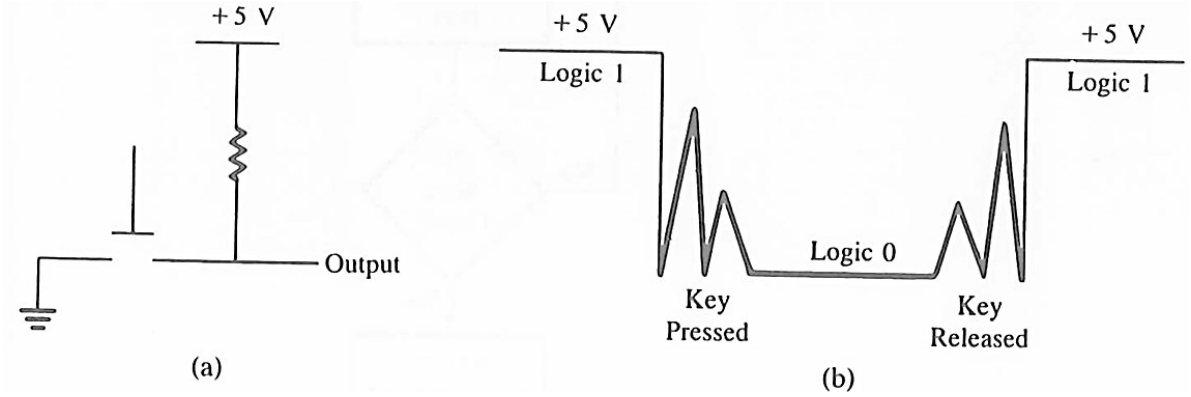
```
DSPLAY:  ;This routine takes the binary number and converts into its common-
         ;   anode seven-segment LED code. The codes are stored in memory
         ;   sequentially, starting from the address CACODE
         ;Input: Binary number in accumulator
         ;Output: None
         ;Modifies contents of HL and A
         LXI H,CACODE      ;Load starting address of code table in HL
         ADD L             ;Add digit to low-order address in L
         MOV L,A           ;Place code address in L
         MOV A,M           ;Get code from memory
         OUT PORTB         ;Send code to port B
         RET
CACODE:  ;Common-anode seven-segment codes are stored sequentially in memory
         DB 40H,79H,24H,30H,19H,12H  ;Codes for digits from 0 to 5
         DB 02H,78H,00H,18H,08H,03H  ;Codes for digits from 6 to B
         DB 46H,21H,06H,0EH          ;Codes from digits from C to F
```



SSD Configuration

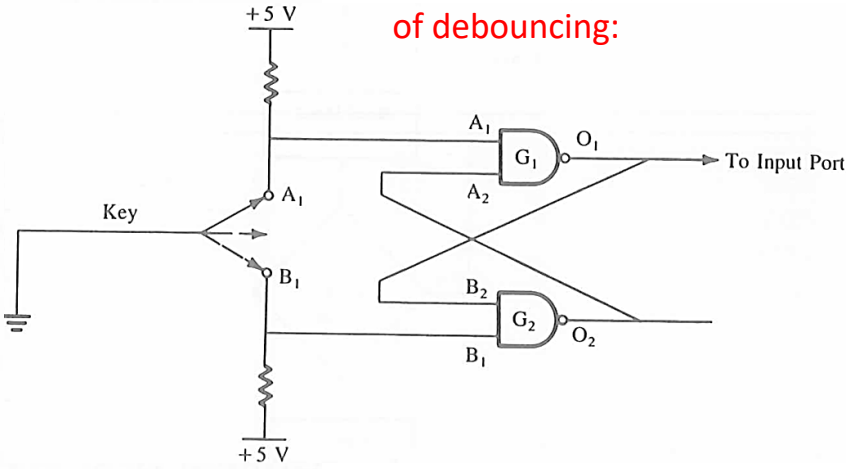
Interfacing of Switches & 7SEGMENT Display

Debouncing




```
DBONCE: ;This is a 20 ms delay routine
        ;The delay COUNT should be calculated based on system frequency
        ;This does not destroy any register contents
        ;Input and Output = None
        PUSH B          ;Save register contents
        PUSH PSW
        LXI B,COUNT     ;Load delay count
LOOP:   DCX B           ;Next count
        MOV A,C
        ORA B           ;Set Z flag if (BC) = 0
        JNZ LOOP
        POP PSW         ;Restore register contents
        POP BC
        RET
```

Hardware approach of debouncing:

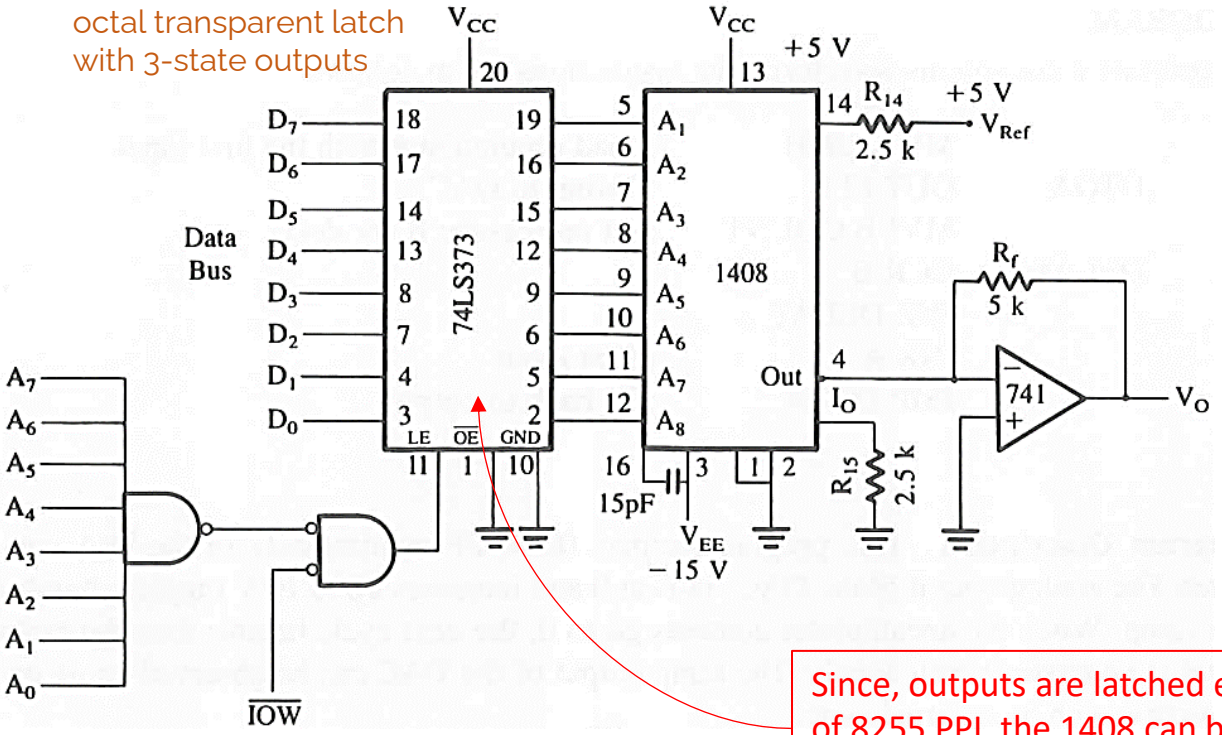


Interfacing of DAC

8-bit DAC Interfacing (1408)

✓ Design an output port with the address FFH to interface the 1408 D/A converter that is calibrated for a 0 to 10 V range.

- Write a program to generate a continuous **ramp waveform**.




```
        MVI A,00H
DTOA:   OUT FFH
        MVI B,COUNT
DELAY:  DCR B
        JNZ DELAY
        INR A
        JMP DTOA
```

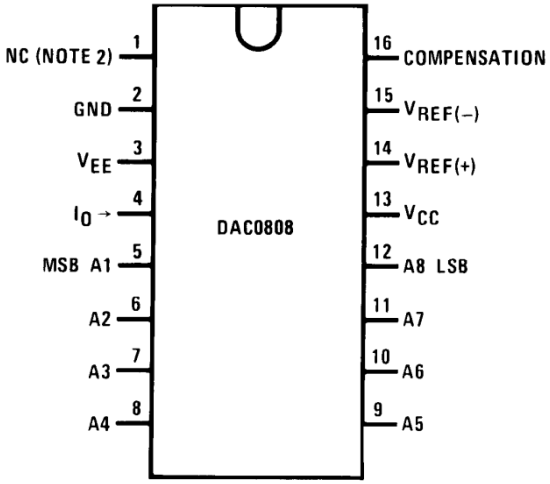
Since, outputs are latched even in Mode 0 of 8255 PPI, the 1408 can be connected directly to one of the ports of 8255 PPI.

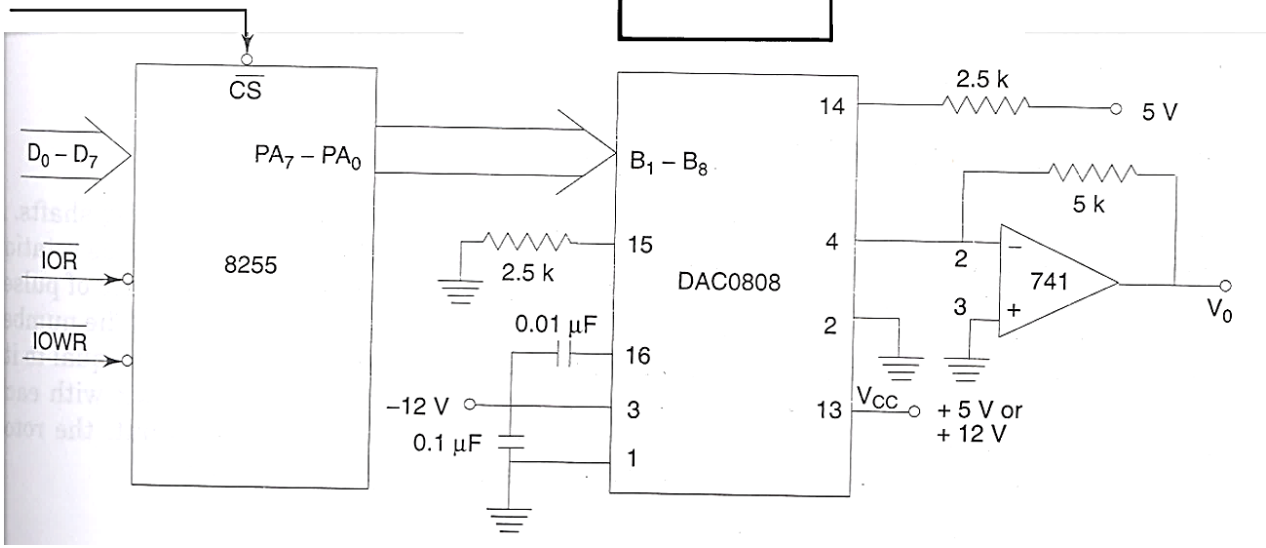
Interfacing of DAC with PPI (Cont..)

DAC0808 – 8-bit DAC Interfacing

- ✓ Write an ALP to generate a triangular wave of range 0-5V.

The DAC0808 will interface directly with popular TTL, DTL or CMOS logic levels, and is a direct replacement for the MC1508/MC1408.



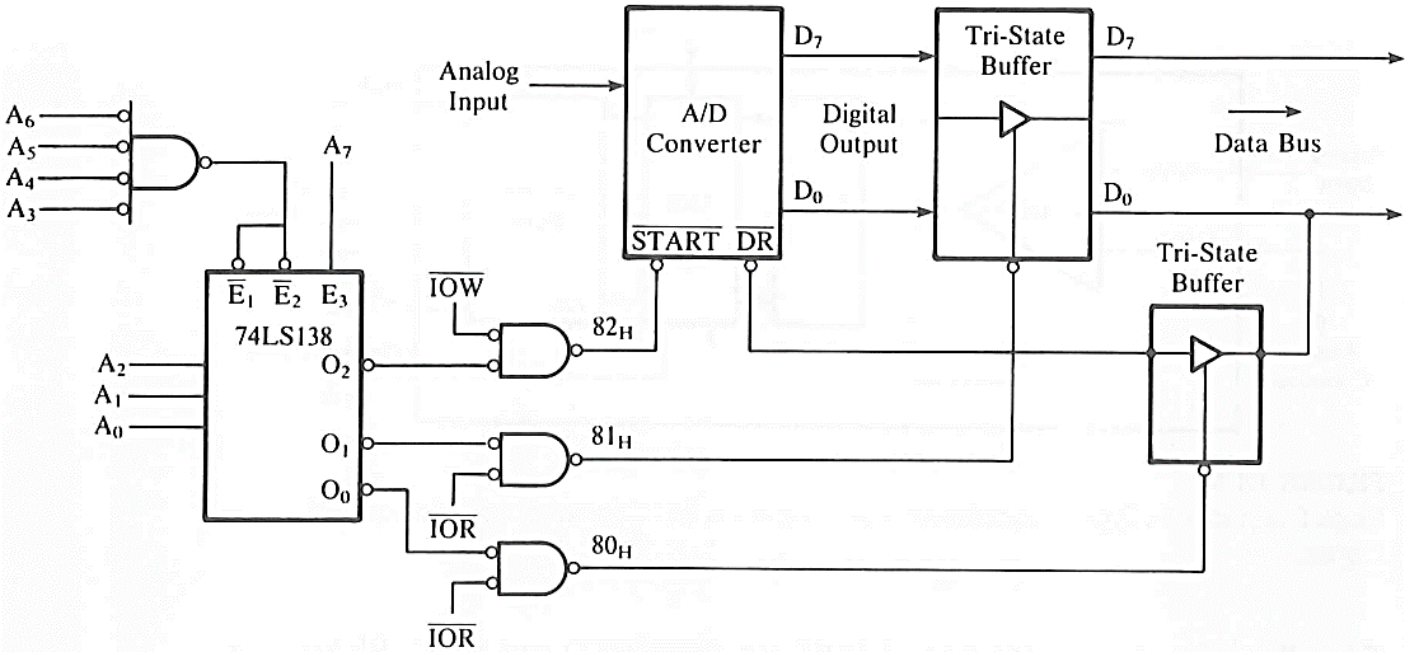


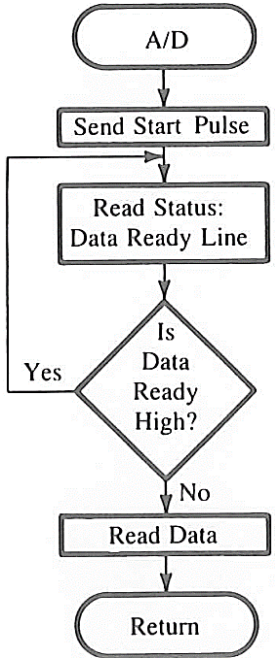

```
MVI A, 80H
OUT CONTROL_R
MVI A, 00H

L1: OUT PORT A
INR A
CPI FFH
JNZ L1

L2: OUT PORT A
DCR A
JNZ L2

JMP L1
HLT
```

Interfacing ADC (Cont..)

1. Interfacing an 8-Bit ADC using Status Check

- ✓ Typically, the analog signal can range from 0 to 10 V, or ±5 V.






```
        OUT 82H
TEST:   IN 80H
        RAR
        JC TEST
        IN 81 H
        RET
```

Interfacing ADC (Cont..)

Interfacing of ADC0808 using PPI

- ✓ 8-bit successive approximation converter.
- ✓ Takes eight different analog inputs
  - *Using ABC selection line*

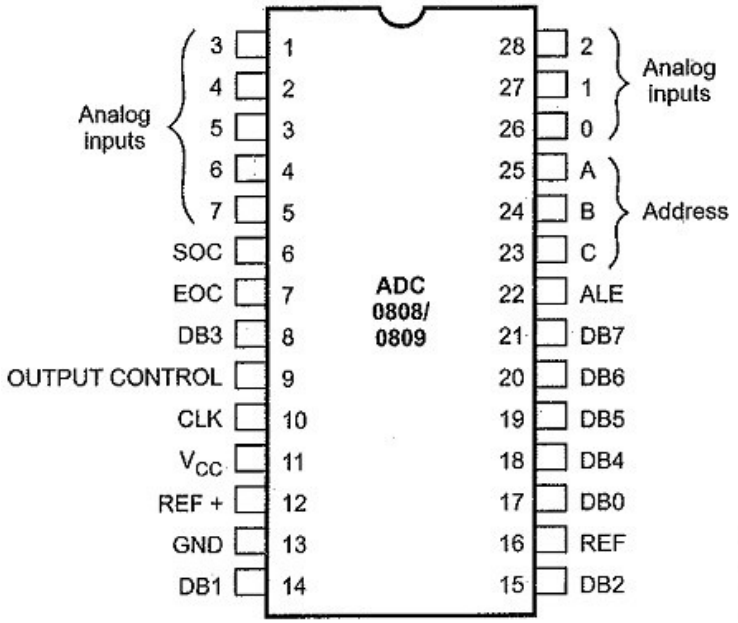
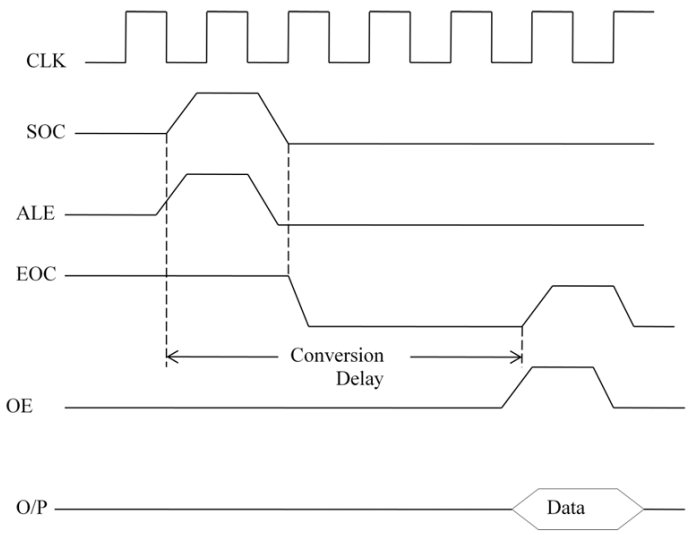
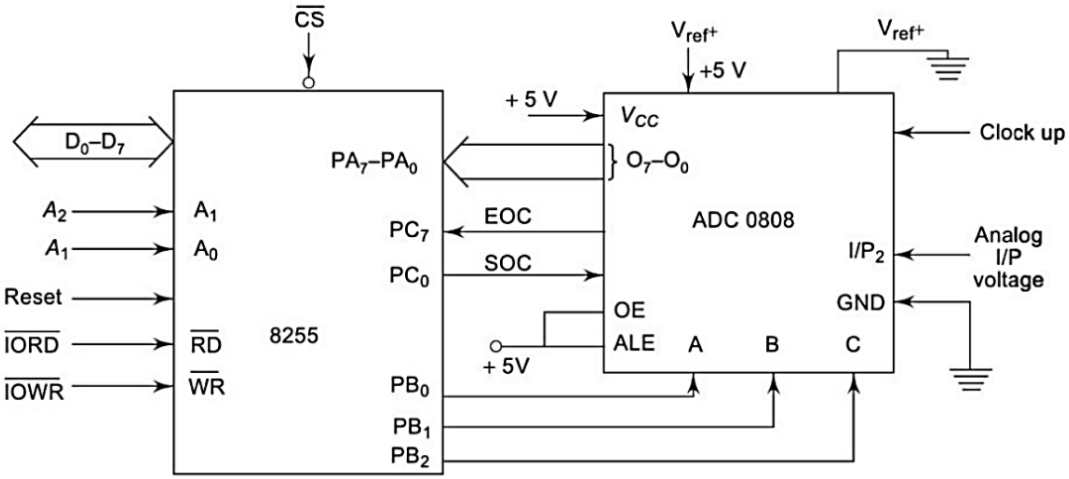


Fig. 14.124 Pin diagram of 0808/0809

Timing Diagram of ADC 0808:





Interfacing ADC (Cont..)

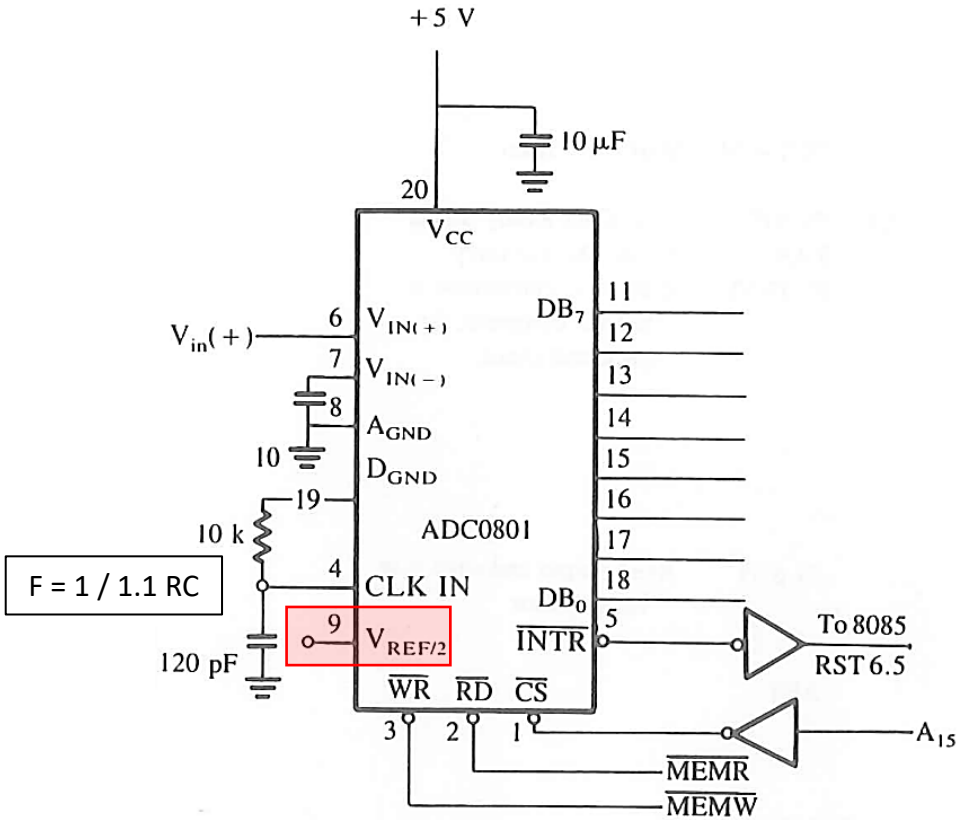
Interfacing of 0808 ADC using PPI

```
        MVI A, 98H      ; Load control word for 8255
        OUT CWR         ; Send control word to control register
        MVI A, 02H      ; Select analog input channel (e.g., channel 2)
        OUT 02H         ; Write to Port B (assumed address 02H)
        MVI A, 01H      ; Start of conversion – send HIGH to PC0
        OUT 03H         ; Write to Port C (assumed address 03H)
        CALL DELAY
        MVI A, 00H      ; Set PC0 LOW to complete SOC pulse
        OUT 03H
WAIT:   IN 03H          ; Read Port C
        RAL             ; Rotate left to check bit 7 (EOC)
        JNC WAIT        ; If carry = 0 (EOC not set), keep polling
        IN 00H          ; Read digital result from Port A (assumed address 00H)
        HLT             ; Stop
```

Interfacing ADC (Cont..)

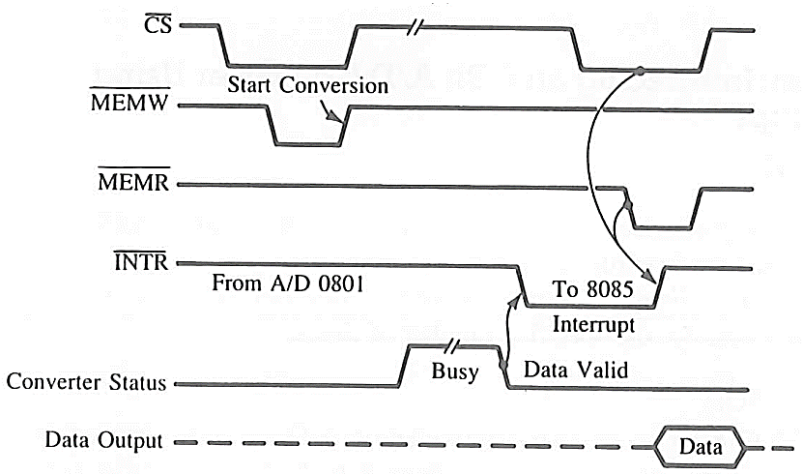
Interfacing an 8-Bit ADC using Interrupt

- ✓ Internal clock → Connecting a register and a capacitor at pins 19 & 4 respectively.
- ✓ When WR' goes low, the internal **SAR** is reset, output lines go into the high impedance state.
- ✓ When V_{cc} is +5V, the input voltage can range from 0V to 5V → output will be from 00H to FFH.



ISR

```
LDA 8000H       ;Read data
MOV M, A        ;Store data in memory
INX H           ;Next memory location
STA 8000H       ;Start next conversion.
EI              ;Enable interrupt again
RET
```



Microcontrollers (Atmega328, STM32)

✓ ATmega328 Microcontroller

- The ATmega328P is an 8-bit AVR microcontroller by **Microchip (formerly Atmel)**, featuring 32 KB Flash, 2 KB SRAM, 1 KB EEPROM, 23 I/O lines, 10-bit ADC, timers, USART, SPI, and I2C.
- It is suitable for small-scale prototyping embedded systems, Arduino-based prototyping, and educational use.

ARM: Advanced RISC Machine

ARM processors are used in smartphones, tablets, laptops, servers, routers, cameras, consoles, and many other devices.

✓ STM32 Microcontroller

- STM32 is a 32-bit ARM Cortex-M microcontroller series by STMicroelectronics, featuring cores from **M0 to M7** for scalable performance.
- It integrates rich peripherals including multi-channel ADCs, DACs, timers, USART, SPI, I2C, USB, CAN, and Ethernet.
- With low-power modes, high-speed clocks, and real-time capabilities, STM32 is ideal for IoT, industrial control, and commercial embedded systems.

Microcontrollers (Atmega328, STM32)

AVR Vs ARM Cortex-M Architecture

Dhrystone MIPS (Million Instructions per Second), or DMIPS, is a measure of computer performance relative to the performance of the DEC VAX 11/780 minicomputer of the 1970s.

Feature	AVR Architecture (Old)	ARM Cortex-M Architecture
Origin	Atmel (now Microchip Technology)	ARM Holdings (licensed by many vendors)
Core Type	8-bit RISC	32-bit RISC (ARMv6-M, ARMv7-M, ARMv8-M)
Instruction Set	AVR (proprietary, compact)	Thumb-2 (highly efficient 16/32-bit mixed)
Clock Speed	Up to ~20 MHz (typical)	Up to 600+ MHz (e.g., Cortex-M7, M85)
Performance	~1 MIPS/MHz	~1.25–1.5 DMIPS/MHz (Cortex-M0+ to M7)
Memory Address Space	Limited (up to 64 KB Flash, 4 KB SRAM common)	Up to MBs of Flash/RAM (depending on SoC)
Power Efficiency	Good for simple low-power apps	Excellent (deep sleep modes, dynamic scaling)
Market Adoption	Niche in hobbyist & legacy systems	Widely adopted in industrial, consumer & IoT sectors
Typical Applications	Hobby projects, small embedded devices	IoT, wearables, industrial control, medical devices

AVR Architecture Introduction

- ✓ AVR is a family of microcontrollers developed since **1996 by Atmel**, acquired by Microchip Technology in 2016.
- ✓ They are 8-bit RISC single-chip microcontrollers based on a **modified Harvard architecture**.
- ✓ AVR was one of the first microcontroller families to use **on-chip flash memory** for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.
- ✓ In 2006, Atmel released microcontrollers based on the **32-bit AVR32** architecture. This was a completely different architecture unrelated to the 8-bit AVR, intended to compete with the ARM-based processors.
- ✓ The ATmega328 is a single-chip microcontroller of **megaAVR family**. It has a modified Harvard architecture **8-bit RISC processor core**.

Atmega328 (AVR) Microcontroller (Cont..)

Internal Architecture - Atmega328

resets the microcontroller if the software fails to periodically reset it, ensuring recovery from system hangs or malfunctions.

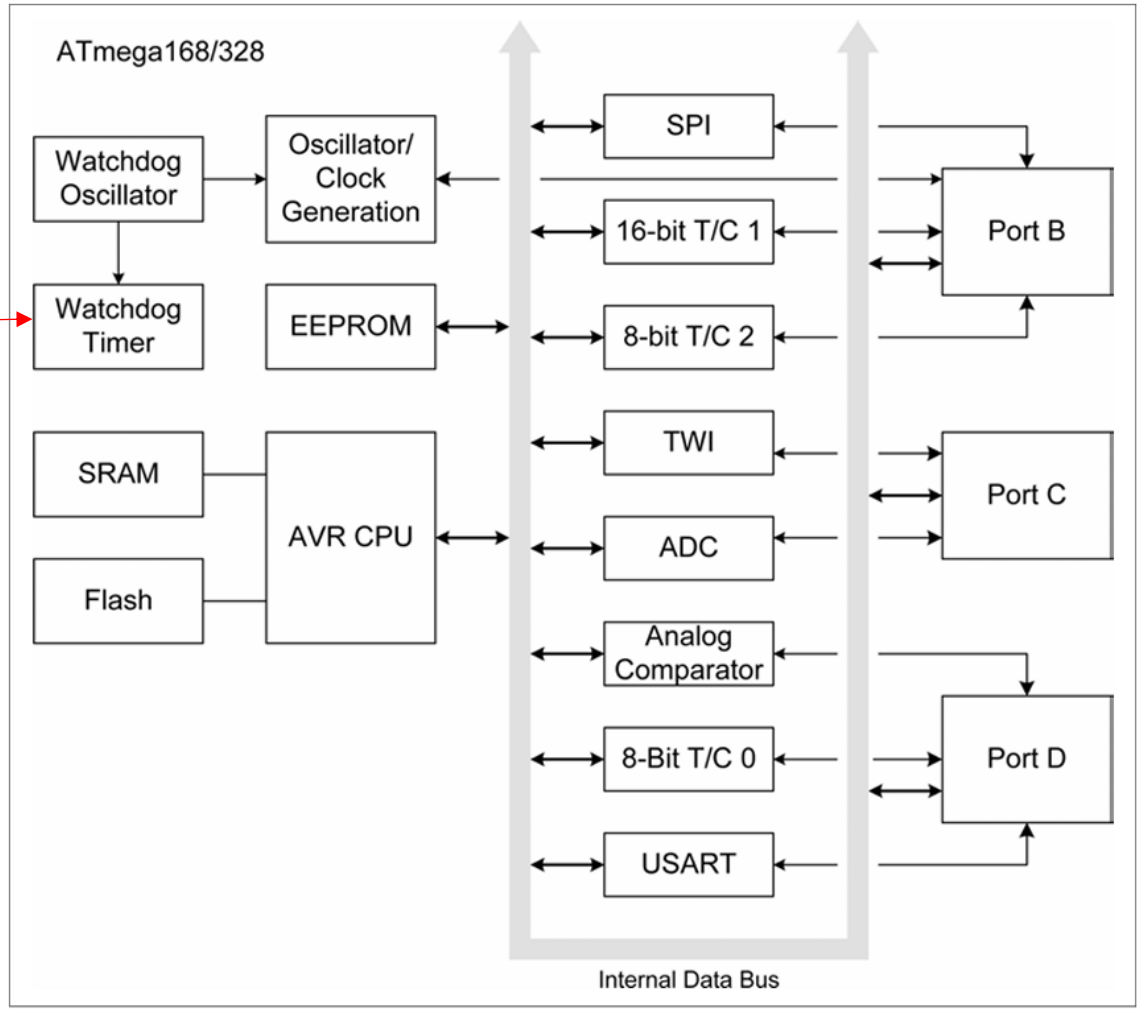
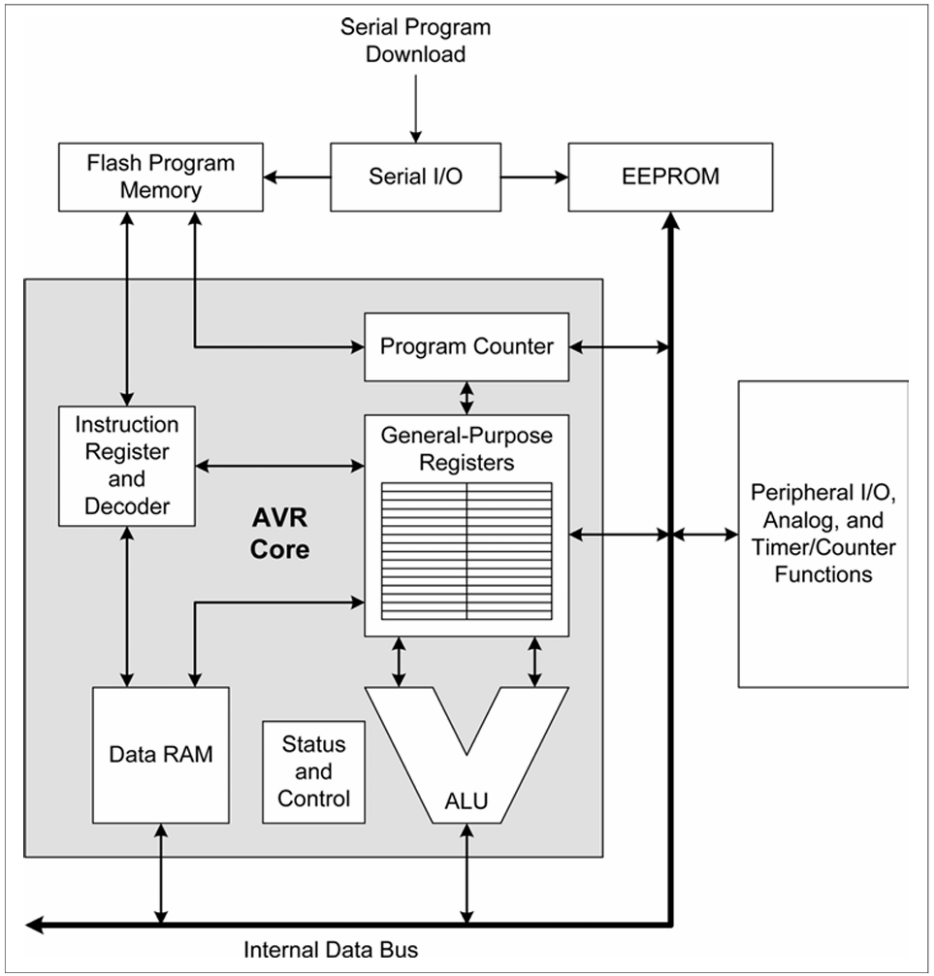


Figure 3-1. ATmega168/328 microcontroller internal block diagram

AVR CPU General Block Diagram



Atmega328 (AVR) Microcontroller (Cont..)

Features

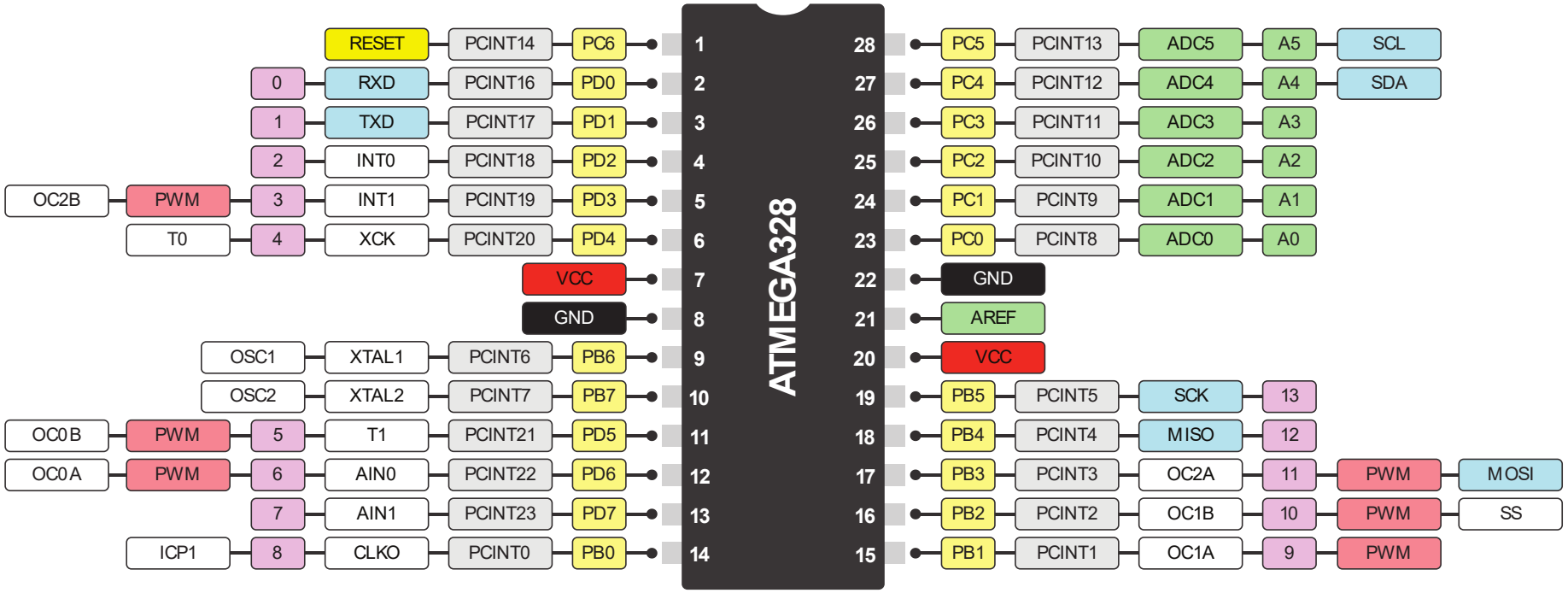
✓ Features

- Memory (Flash – 32KB, SRAM – 2KB, EEPROM – 1KB).
- Clock Frequency – Up to 20MHz
- In-system programming by on-chip boot program
- Two 8-bit timer/counters with separate prescaler and compare mode
- One 16-bit timer/counter with separate prescaler, compare mode, capture mode
- Real-time counter with separate oscillator (*Timer2 using a 32.768 kHz crystal on TOSC1/TOSC2, independent of the main clock.*)
- **Six-channel 10-bit ADC** (depends on package type)
- Programmable watchdog timer
- Analog comparator
- 23 programmable I/O lines and Six PWM channels
- Serial Interface Logic (USART, SPI, TWI/I2C).

Atmega328 (AVR) Microcontroller (Cont..)

Pin Configuration

- ✓ ATmega328 has three ports designated as B, C, and D. Ports B and D are 8-bit ports.
- ✓ Port C has six pins available that can be used as ADC inputs. PC4 and PC5 are also connected to the TWI logic (I2C).
  - Please note that there is no PC7 pin and Port C in the ATmega168/328 part.



Atmega328 (AVR) Microcontroller (Cont..)

Pin Functions

Label	Full Form	Description
PBx	**Port B, Bit x**	General Purpose I/O pin from Port B (e.g., PB0 is bit 0 of Port B)
PCx	**Port C, Bit x**	General Purpose I/O pin from Port C
PDx	**Port D, Bit x**	General Purpose I/O pin from Port D

VCC	**Voltage Common Collector**	Supply voltage (typically +5V)
GND	**Ground**	Electrical ground
AREF	**Analog Reference**	Reference voltage for ADC
AVCC	**Analog VCC**	Supply voltage for analog circuitry (connected to V_{CC})

ADCx	**Analog-to-Digital Converter channel x**	Channel for analog signal input (e.g., ADC0 to ADC5)

OC0A/B	**Output Compare for Timer 0 A/B**	PWM output for Timer 0
OC1A/B	**Output Compare for Timer 1 A/B**	PWM output for Timer 1
OC2A/B	**Output Compare for Timer 2 A/B**	PWM output for Timer 2
PWM	**Pulse Width Modulation**	Pins that support analogWrite using timers

Atmega328 (AVR) Microcontroller (Cont..)

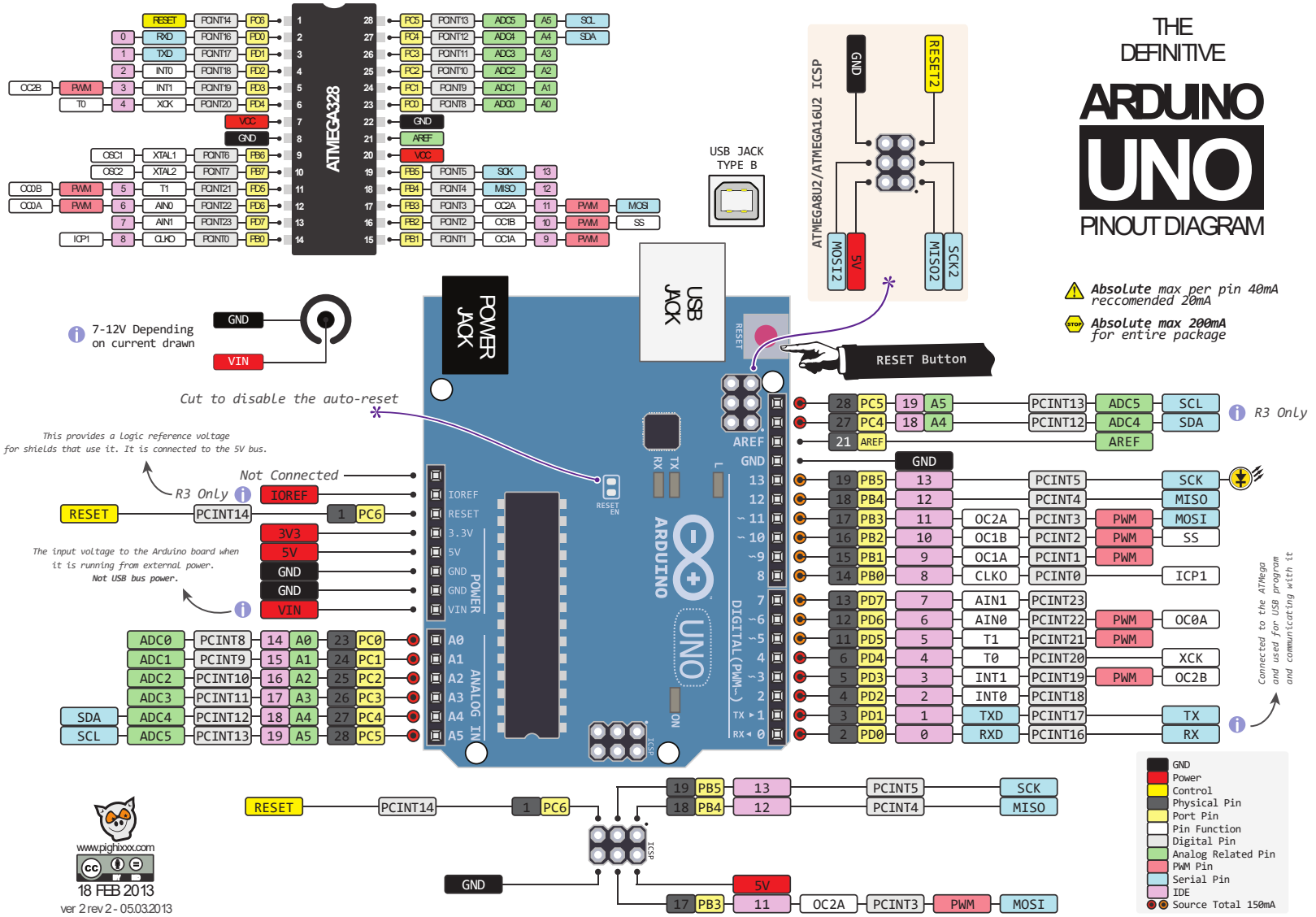
Pin Functions (Cont..)

RXD	**Receive Data**	UART receive pin (PD0)
TXD	**Transmit Data**	UART transmit pin (PD1)
SDA	**Serial Data Line**	I^2C data line (PC4)
SCL	**Serial Clock Line**	I^2C clock line (PC5)
MOSI	**Master Out Slave In**	SPI data from master to slave (PB3)
MISO	**Master In Slave Out**	SPI data from slave to master (PB4)
SCK	**Serial Clock**	SPI clock line (PB5)
SS	**Slave Select**	SPI slave select (PB2)

INT0/INT1	**External Interrupt 0/1**	Used for edge-triggered interrupts (PD2, PD3)
PCINTx	**Pin Change Interrupt x**	Allows interrupt on pin state change
T0, T1	**Timer 0, Timer 1 Clock Input**	Input for timer clocking
ICP1	**Input Capture Pin for Timer 1**	Captures external events for Timer 1
CLKO	**Clock Output**	Outputs system clock to external device
XCK	**External Clock Input for USART**	Used in synchronous UART communication

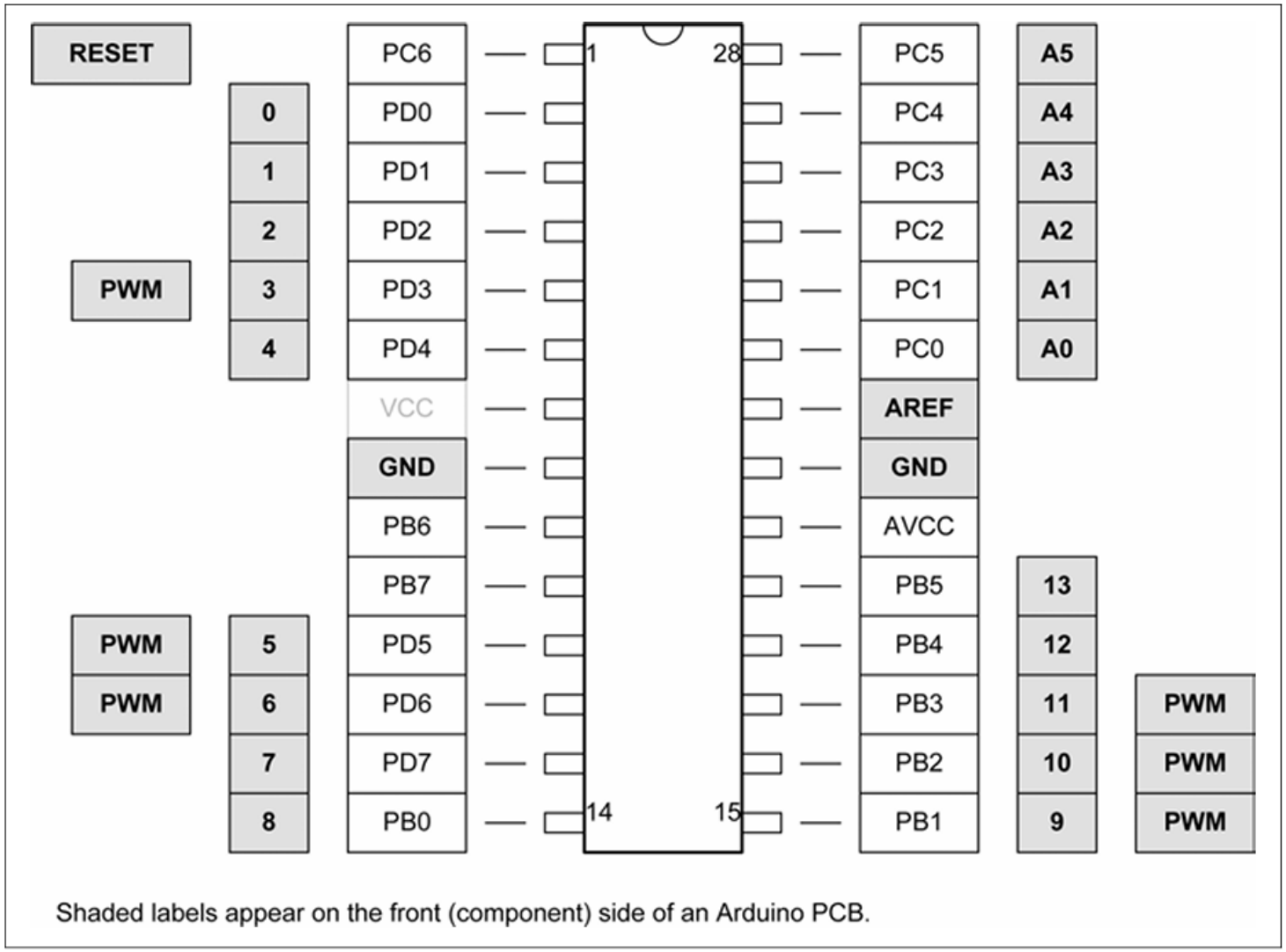
Atmega328 (AVR) Microcontroller (Cont..)

Arduino Pin Assignments



Atmega328 (AVR) Microcontroller (Cont..)

Arduino Pin Assignments - Simplified



Pin Configuration (Cont..)

Ports

✓ **Three Ports:**

- **Port B (PB0–PB7):** 8-bit, used for digital I/O (e.g., LEDs, switches) and special functions like **SPI** or **PWM**.
- **Port C (PC0–PC5):** 6-bit, primarily for analog-to-digital conversion (**ADC**); PC4 (SDA) and PC5 (SCL) support **I2C**; PC6 is typically the **RESET** pin (not general I/O).
- **Port D (PD0–PD7):** 8-bit, handles digital I/O, **UART**, and **external interrupts**.

✓ **Features:**

- Bidirectional pins with programmable pull-up resistors to stabilize inputs.
- **Symmetrical drive:** Each pin can source or sink up to 20 mA (200 mA total limit).
- Pins enter high-impedance (tri-state) mode during reset to prevent unwanted currents.

Pin Configuration (Cont..)

Pin Functions

- ✓ Analog Inputs
  - **6 ADC Channels:** PC0–PC5 (pins 23–28) offer 10-bit resolution (1024 levels) for converting analog signals (e.g., from sensors) to digital.
  - **Dual Use:** These pins can also function as digital I/O, except PC4/PC5 when used for I2C.
- ✓ PWM Outputs
  - OCxn pins (*OC0A, OC0B, OC1A, OC1B, OC2A, and OC2B*) are available as PWM outputs.
- ✓ External Interrupts
  - The port D pins **PD2** and **PD3** are specifically intended for use as external interrupt inputs. However, any of the PCINT0 to PCINT23 pins may also be used as external interrupt inputs.

Atmega328 (AVR) Microcontroller (Cont..)

Timers & Interrupts

✓ Timer/Clock I/O

- **Three Timers:**
  - Timer0, Timer1 (8-bit), and Timer2 (16-bit) for timing or PWM generation.
  - PWM outputs: OC0A/B (PD6/PD5), OC1A/B (PB1/PB2), OC2A/B (PB3/PD3).
- **Shared Pin:** PD5 doubles as T1 (external clock input) and OC0B.

✓ External Interrupts

- **Dedicated Interrupts:** INT0 (PD2, pin 4) and INT1 (PD3, pin 5) trigger on rising/falling edges or low levels.
- **Pin Change Interrupts:** PCINT0–PCINT23 (most pins) detect voltage changes, offering flexible interrupt options.

*Explore yourself the available instruction sets and registers inside Atmega328P.

Few Registers inside Atmega328P

✓ General Purpose I/O (GPIO)

Register	Full Name	Function
DDRx	Data Direction Register (x = B, C, D)	Configures pin as **input (0)** or **output (1)**
PORTx	Port Data Register	Writes logic **HIGH (1)** or **LOW (0)** to output pins
PINx	Port Input Pins Register	Reads the current **logic level** on input pins

Category	Important Registers
I/O	DDRx, PORTx, PINx
Timers	TCCRnA/B, TCNTn, OCRnA/B, TIMSKn, TIFRn
UART	UBRR0H/L, UCSR0A/B/C, UDR0
Interrupts	SREG, EIMSK, EIFR, PCICR, PCMSKx
ADC	ADMUX, ADCSRA, ADCL/H
SPI	SPCR, SPSR, SPDR
TWI (I^2C)	TWBR, TWSR, TWAR, TWCR, TWDR
Power/Reset	MCUCR, MCUSR, PRR, WDTCSR

*Explore yourself the available instruction sets and more registers and their functions.

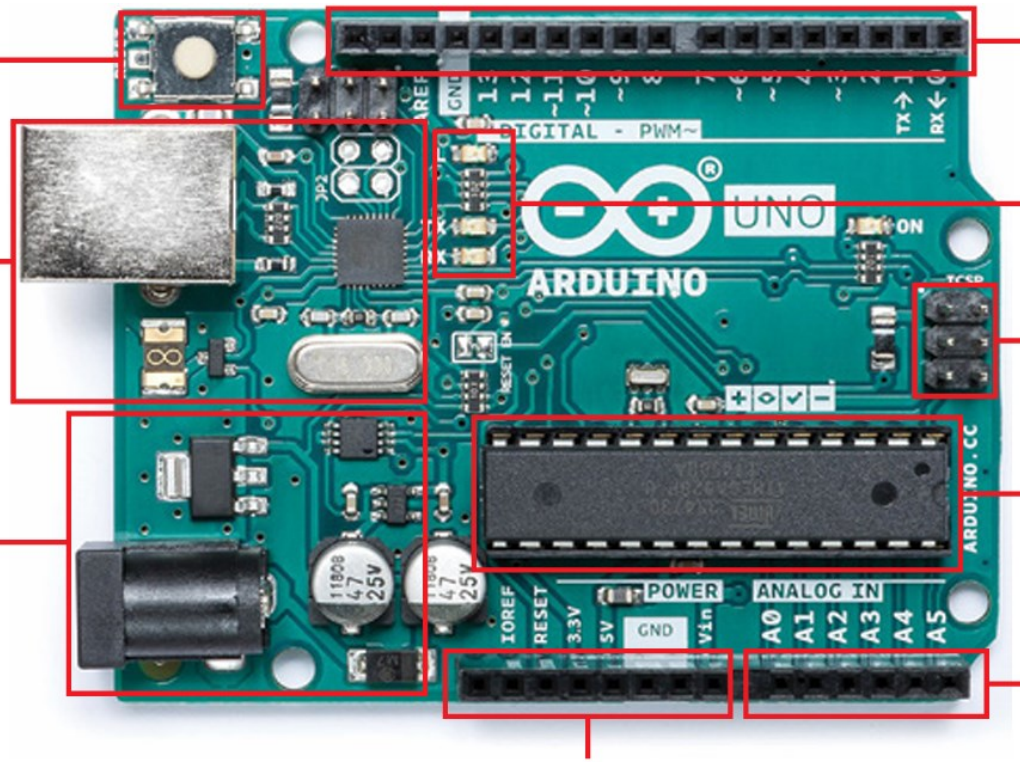
Atmega328 (AVR) Microcontroller (Cont..)

Electrical Characteristics

- ✓ Frequency: 8MHz (*Default internal oscillator*)
  - External crystals (e.g., Arduino Uno) usually run at **16 MHz**.
- ✓ Supply Voltage: 1.8V–5.5V (5V typical for Arduino).
- ✓ Input Voltage Levels (V_{CC} = 2.4–5.5V)
  - **Digital LOW Voltage Level:** –0.5V to 0.3×V_{CC} (*e.g., 0.5–1.5V at 5V*).
  - **Digital HIGH Voltage Level**: 0.6×V_{CC} to V_{CC} + 0.5V (e.g., 3–5.5V at 5V).
- ✓ Power Consumption:
  - Active mode (~5.2 mA at 8 MHz)
  - Idle (~1.2 mA).
  - Power down mode with 3V V_{CC} (4.2 µA – WDT enabled, 0.1 WDT disabled)
- ✓ Current Limits: 20 mA per pin, 200 mA total across all pins.

Atmega328 (AVR) Microcontroller (Cont..)

ATmega328P as Arduino Board μCU



Programming & I/O: Reset Button

Programming & Power: USB-to-Serial Interface

Power: 7-12VDC Input and Voltage Regulator

I/O: General Purpose, PWM, and Communication Buses

I/O: Debug LED and USB LEDs

Programming: In-Circuit Serial Programming (ICSP) Header

Microcontroller: ATmega 328P

I/O: Analog-to-Digital (ADC) Inputs

Power: Voltage and GND pins

STM32 (Cortex-M) Microcontroller

Overview

- ✓ ARM Cortex-M microprocessors are based on a **32-bit RISC architecture** optimized for low-power, real-time embedded systems.
  - Example: (32-bit Arm Cortex-M4 processor) on a NUCLEO board.
- ✓ Features include faster clock speeds, a larger instruction set, Memory Protection Unit (MPU), and Floating Point Unit (FPU).
- ✓ Key Internal Components
  - Cortex-M4 Processor Core.
  - Memory (Flash, SRAM).
  - Extensive Peripherals (communication cores, timers, GPIO, ADCs, DACs).
  - Power Management.
  - Reset & Clock Control.

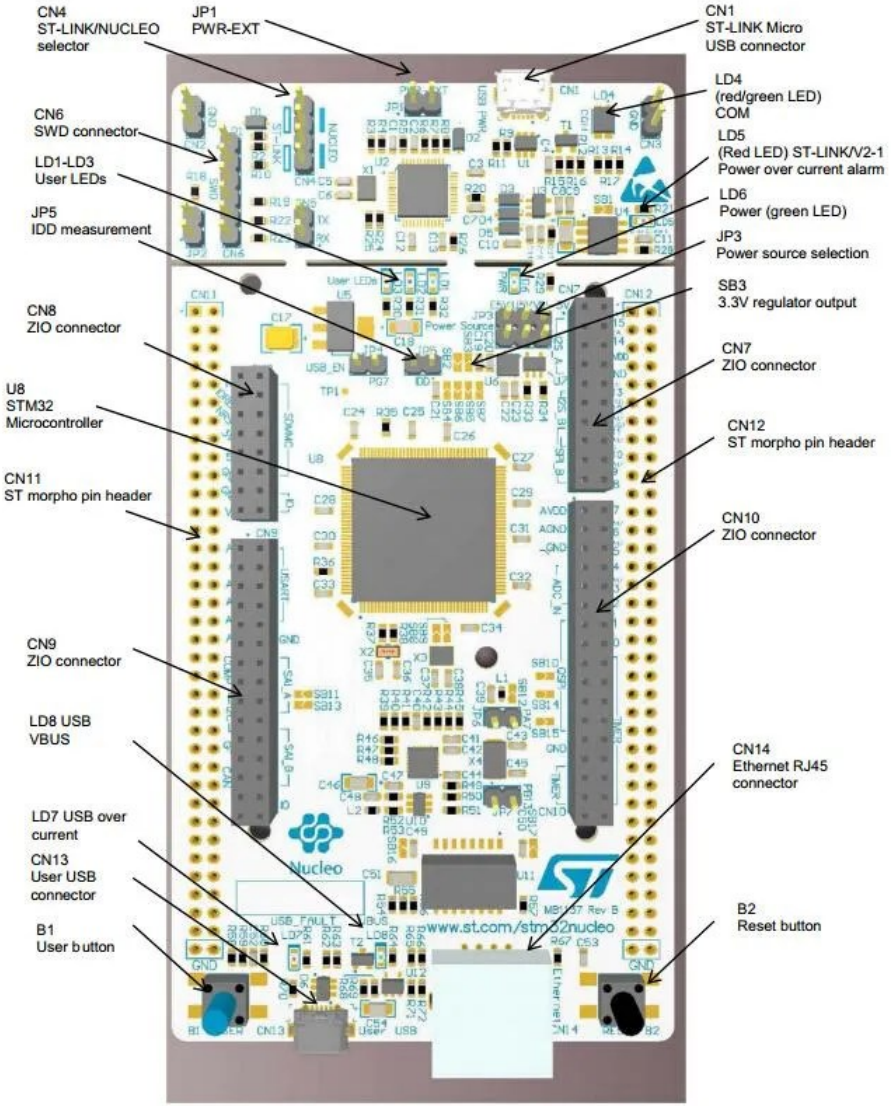
MPU (Memory Protection Unit): Hardware that controls access permissions to memory regions

Compare: ARM AVR x86 MIPS RISC-V

STM32 (Cortex-M) Microcontroller

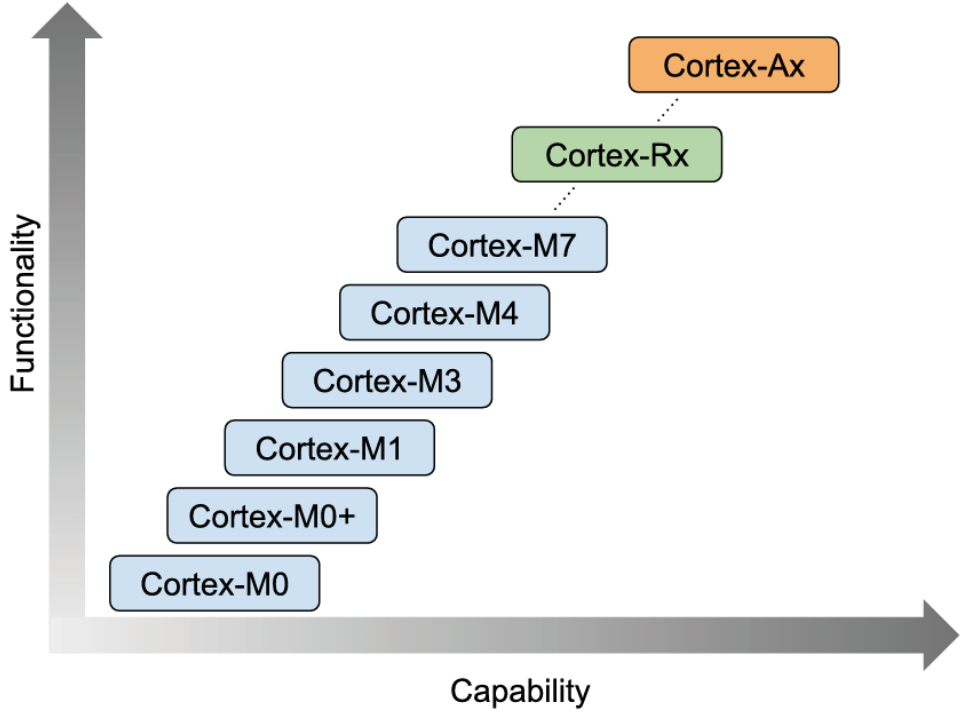
Example (STM32F29ZI in Nucleo Board)

- ✓ **An ARM (Advanced RISC Machines) processor**
  - is a 32/64-bit RISC-based CPU architecture by Arm Ltd.
  - It features a **Load/Store design**, Thumb/Thumb-2 instruction sets, and supports **pipelining**, MPU, FPU, DSP, **SIMD** capabilities, and low-power modes.
  - **Thumb ((16-bit)** and **Thumb-2 (mixed 16/32-bit)** are compressed instruction sets used in ARM processors to improve **code density** and **power efficiency**, especially in embedded systems.



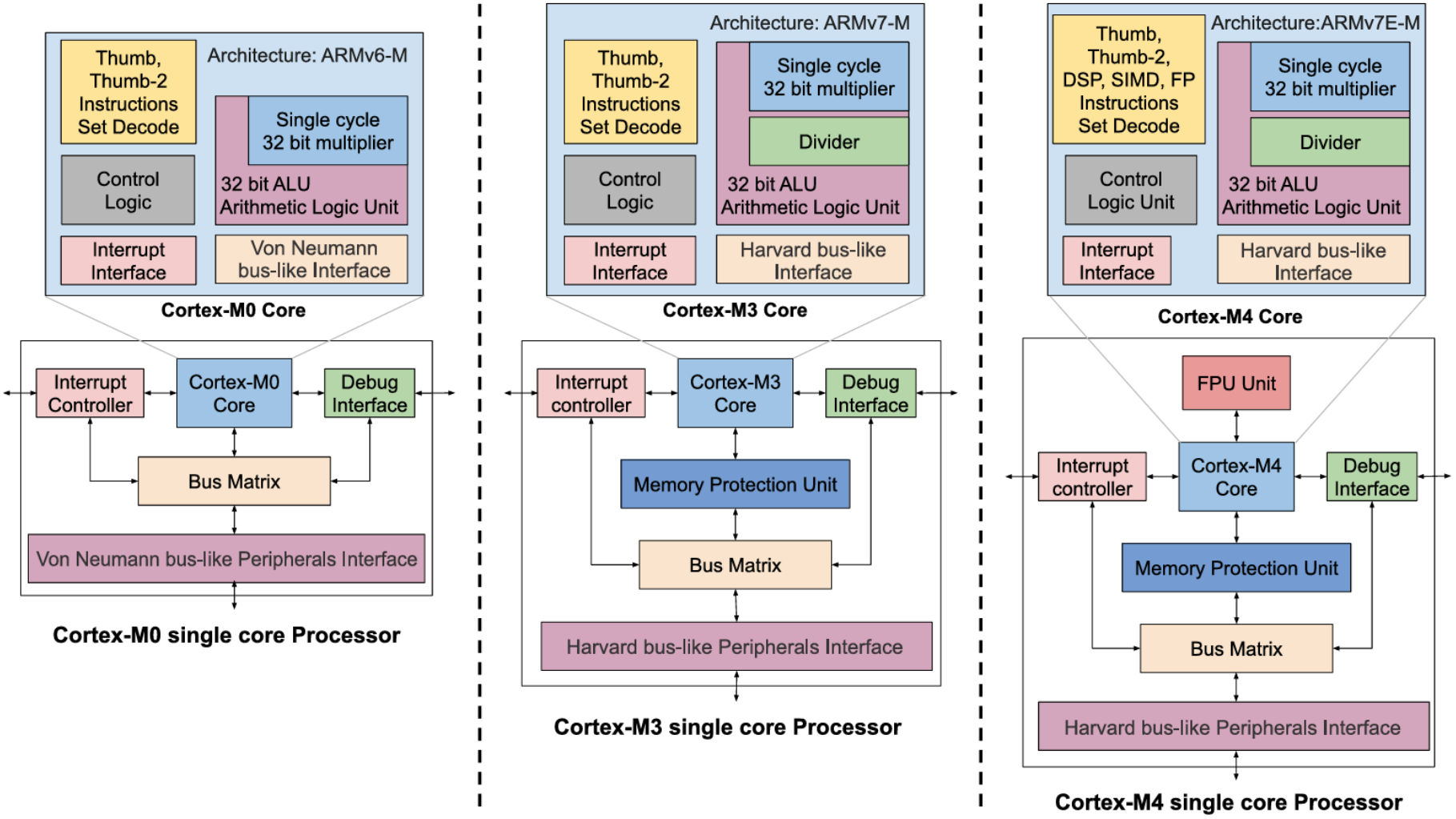
ARM Cortex–Processor Family

- ✓ The STM32 (Specifically STM32F429ZIT6U) microcontroller is manufactured by **STMicroelectronics** using a **Cortex-M4** processor designed by Arm Ltd.
- ✓ The Cortex-M processors are the **most energy-efficient** embedded devices of the family and have the lowest cost.



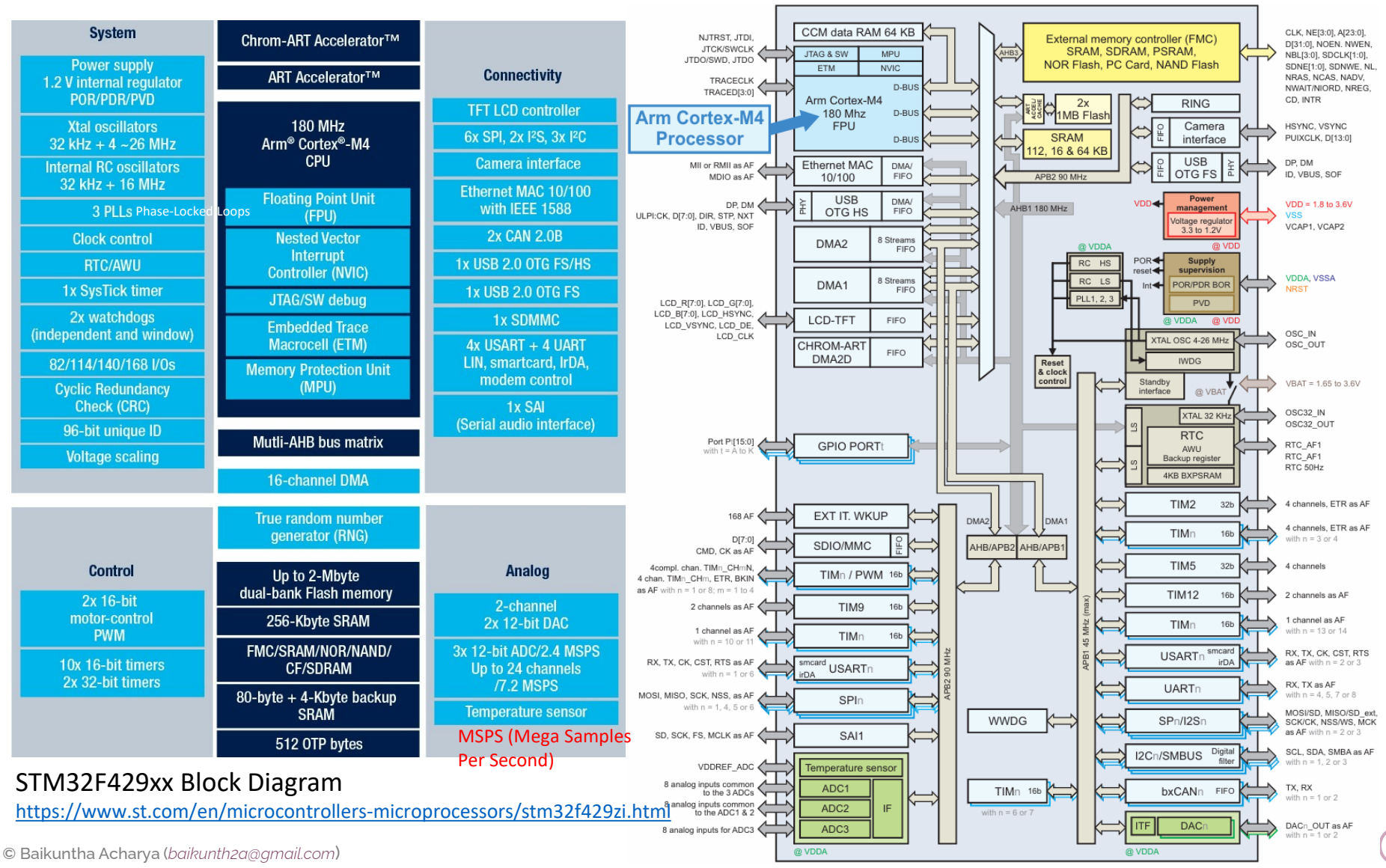
ARM Cortex–M General Architecture

Review the concepts of RISC, CISC, ARM Architecture, Von Neuman, Harvard, Floating point operations, SIMD, etc.



STM32 (Cortex-M) Microcontroller

E.g.: STM32F429ZI Block Diagram



STM32F429xx Block Diagram

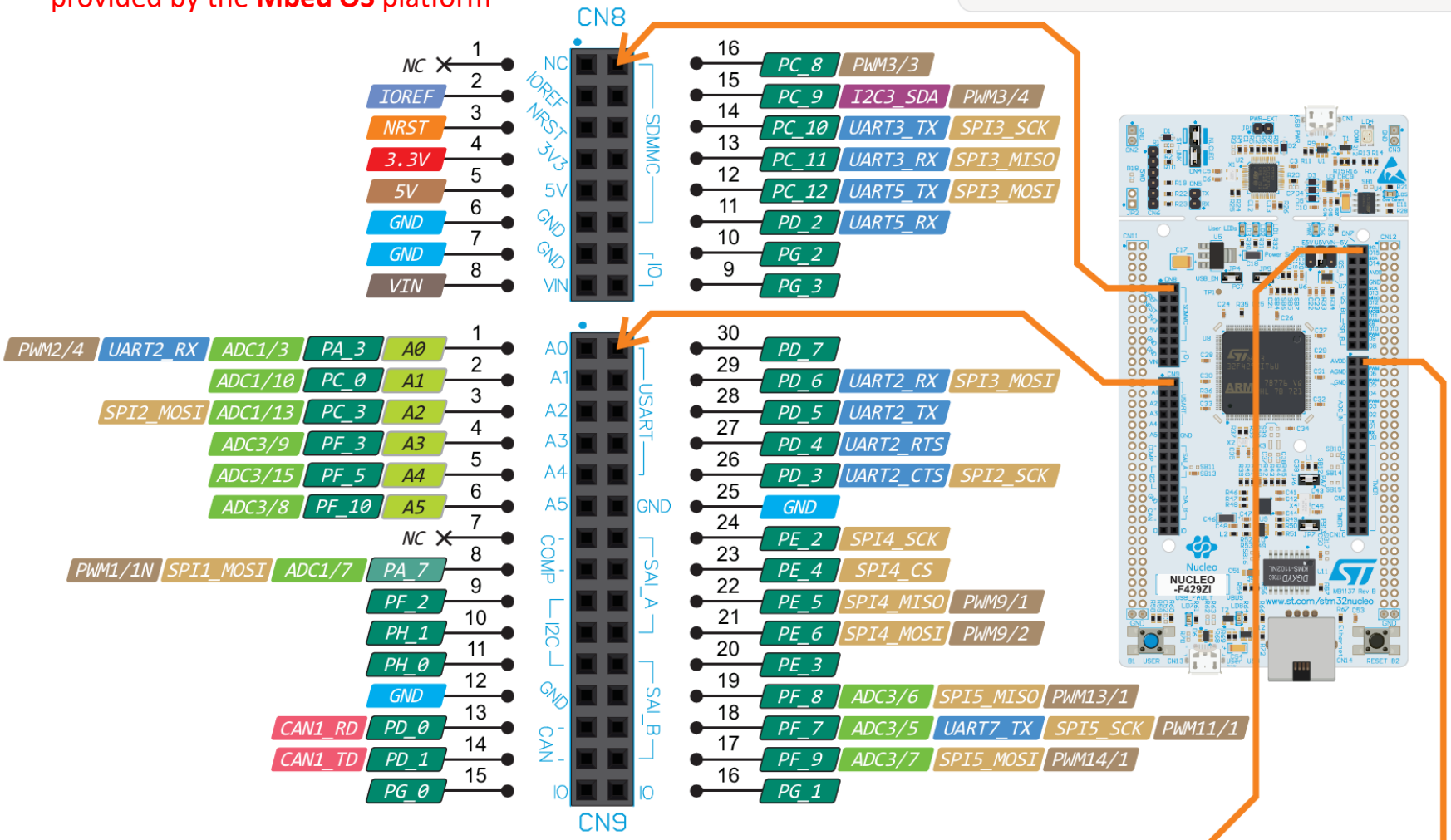
https://www.st.com/en/microcontrollers-microprocessors/stm32f429zi.html

Pin Configuration - NUCLEO-F429ZI board

Mbed Pin refers to a **general-purpose input/output (GPIO) pin abstraction** provided by the **Mbed OS** platform

Labels usable in code

Dn	*Arduino Digital Input/Output (GPIO). n=number.*
An	*Arduino Analog Input (ADC). n=number.*
Pl_n	*Mbed pin without conflict. l=letter, n=number.*
Pl_n	*Mbed pin connected to other components. l=letter, n=number.*
LEDn	*Mbed pin (DigitalOut) connected to LEDs. n=number.*

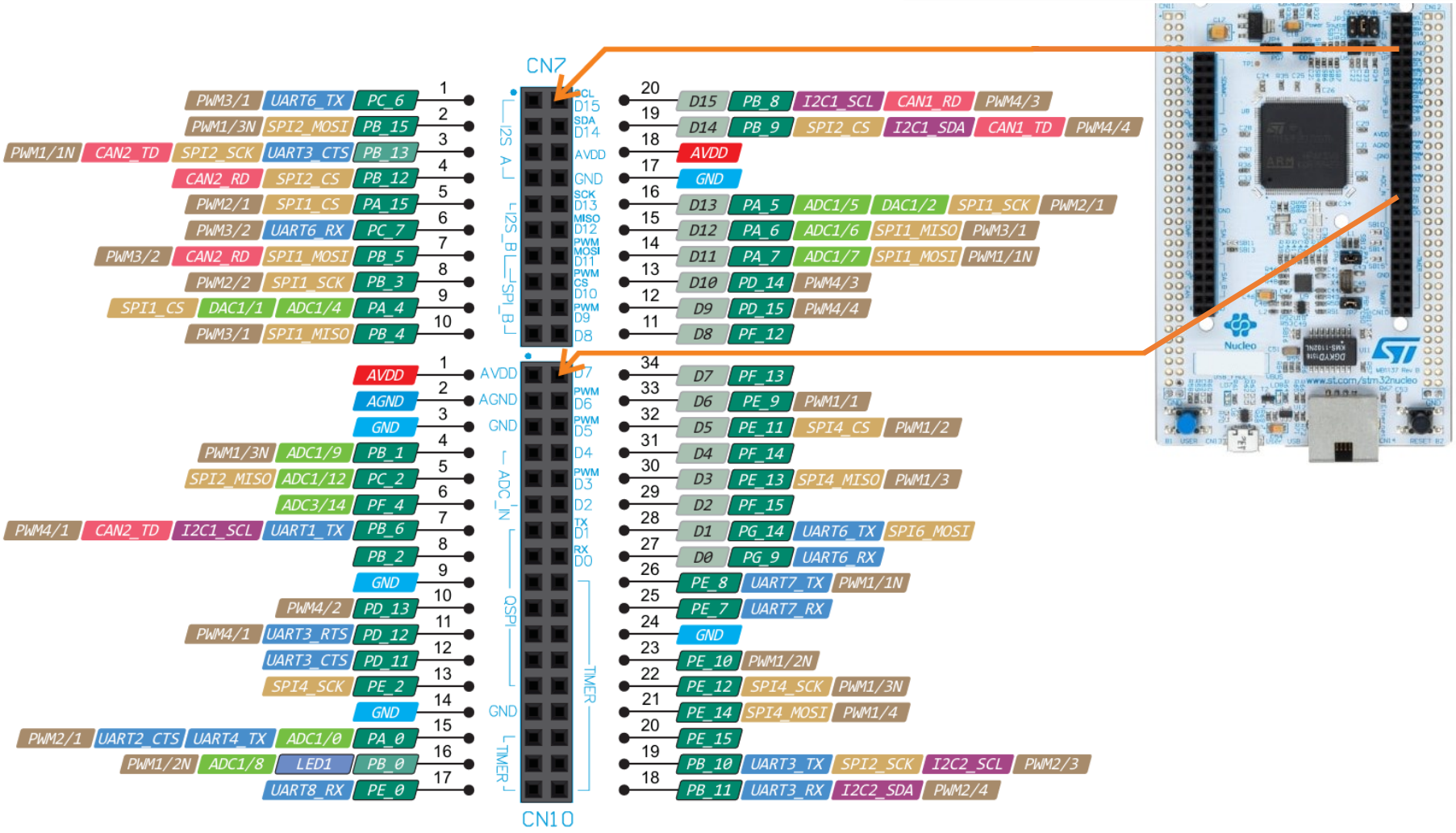


STM32 (Cortex-M) Microcontroller

Pin Configuration - NUCLEO-F429ZI board

Labels usable in code

Dn	Arduino Digital Input/Output (GPIO). n=number.
An	Arduino Analog Input (ADC). n=number.
Pl_n	Mbed pin without conflict. l=letter, n=number.
Pl_n	Mbed pin connected to other components. l=letter, n=number.
LEDn	Mbed pin (DigitalOut) connected to LEDs. n=number.



Programming in ARM Cortex-based μP

✓ Programming in ARM Cortex-based Microprocessors

- Programming these μPs involves **C/C++ development**, using hardware abstraction libraries and toolchains specific to the target device.

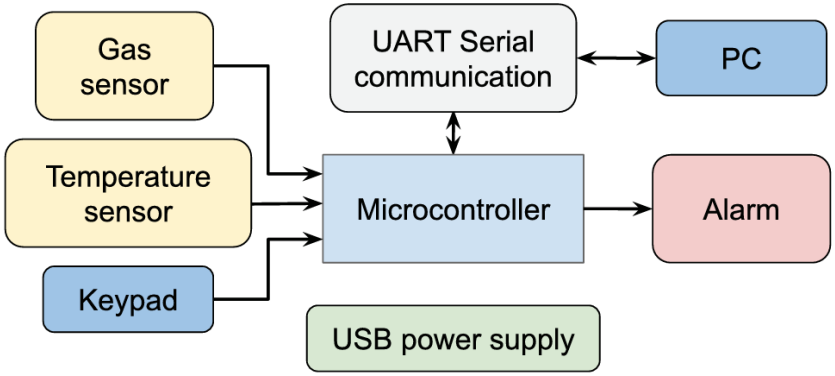
✓ Development Workflow

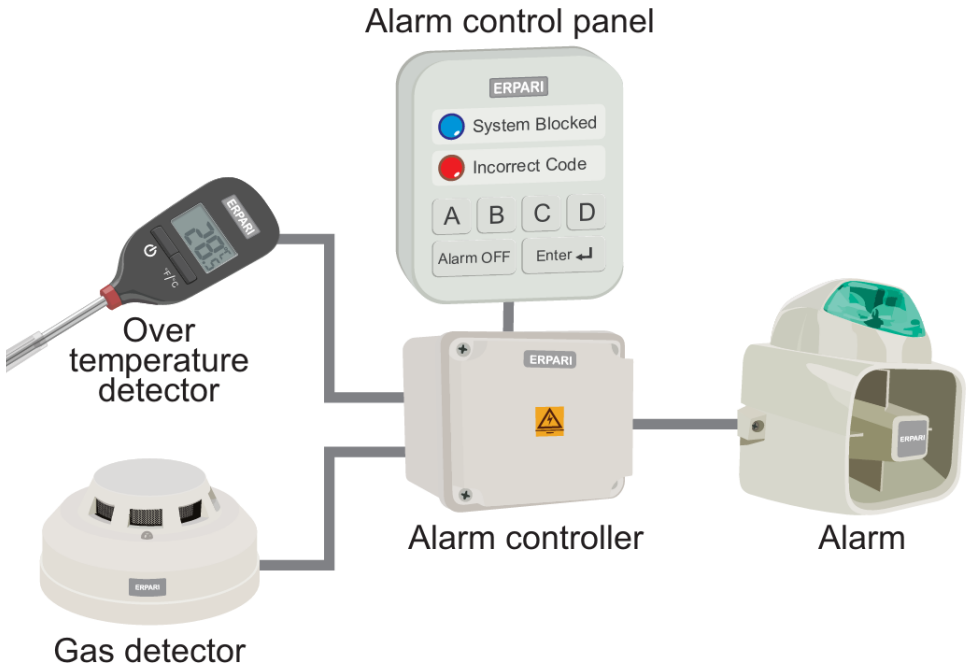
- **Toolchain**: GCC (arm-none-eabi), Keil MDK, IAR, or STM32CubeIDE.
- **Languages**: Primarily **C**, with support for **assembly** and **C++**.
- **Firmware Libraries**: CMSIS (ARM's low-level API), HAL/LL drivers (vendor-specific, e.g., STM32 HAL).
- **Build Process**: Compile → Link → Flash → Debug.

**Certain ARM Cortex-based μP also supports Mbed OS: Free open source IoT OS and development tools.*

Example System using STM32 Board

- ✓ The smart home system may include a gas sensor, a temperature sensor, an alarm, a keypad, and a communication link with a PC.
  - This first implementation will detect fire using an over temperature detector and a gas detector and, if it detects fire, it will activate the alarm until a given code is entered.

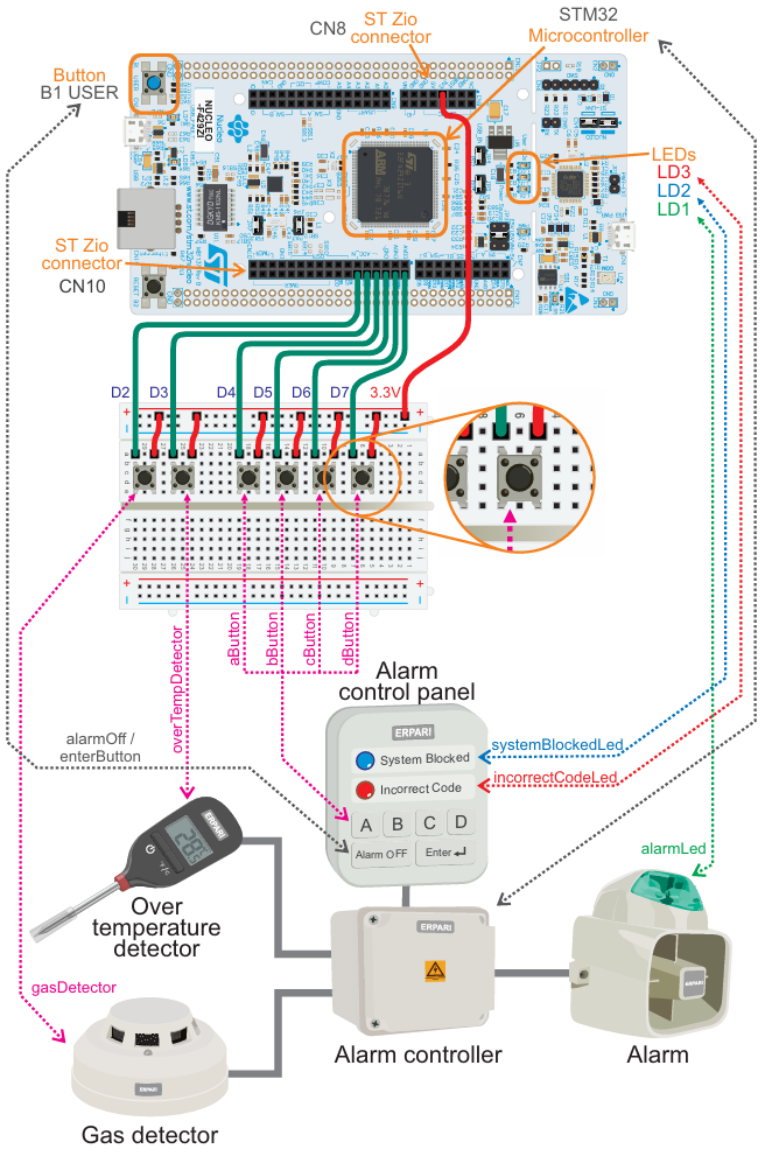




Example – Smart Home System

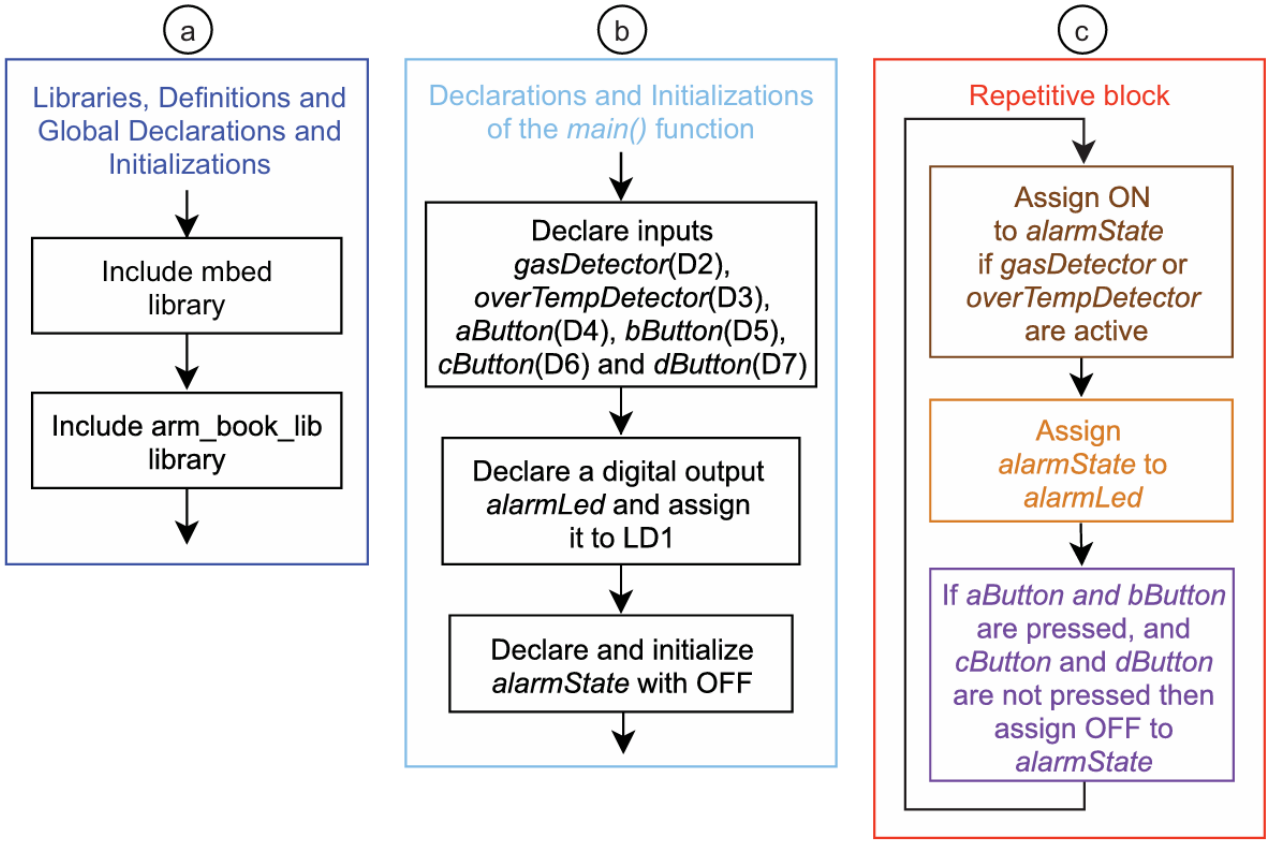
Circuit Connection

Smart home system	Representation used
Alarm	LD1 LED (green)
System blocked LED	LD2 LED (blue)
Incorrect code LED	LD3 LED (red)
Alarm Off / Enter button	B1 USER button
Gas detector	Button connected to D2 pin
Over temperature detector	Button connected to D3 pin
A button	Button connected to D4 pin
B button	Button connected to D5 pin
C button	Button connected to D6 pin
D button	Button connected to D7 pin



Example – Smart Home System

- ✓ **Problem:** Keep the alarm active after gas or over temperature were detected and secure the alarm deactivation by means of a code.
  - **Main parts of program**:



Example – Smart Home System (Coding)

```
#include "mbed.h"
#include "arm_book_lib.h"

int main()
{
    DigitalIn gasDetector(D2);
    DigitalIn overTempDetector(D3);
    DigitalIn aButton(D4);
    DigitalIn bButton(D5);
    DigitalIn cButton(D6);
    DigitalIn dButton(D7);

    DigitalOut alarmLed(LED1);

    gasDetector.mode(PullDown);
    overTempDetector.mode(PullDown);
    aButton.mode(PullDown);
    bButton.mode(PullDown);
    cButton.mode(PullDown);
    dButton.mode(PullDown);

    alarmLed = OFF;

    bool alarmState = OFF;

    while (true) {

        if ( gasDetector || overTempDetector ) {
            alarmState = ON;
        }

        alarmLed = alarmState;

        if ( aButton && bButton && !cButton && !dButton) {
            alarmState = OFF;
        }
    }
}
```

Include the libraries "mbed.h" and "arm_book_lib.h"

- Digital input objects named *gasDetector*, *overTempDetector* and *aButton* to *dButton*, are declared and assigned to D2 to D7, respectively.
- Digital output object named *alarmLed* is declared and assigned to LED1.
- *gasDetector*, *overTempDetector* and *aButton* to *dButton* are configured with internal pull-down resistors.
- *alarmLed* is assigned OFF.
- *alarmState* is declared and assigned OFF.

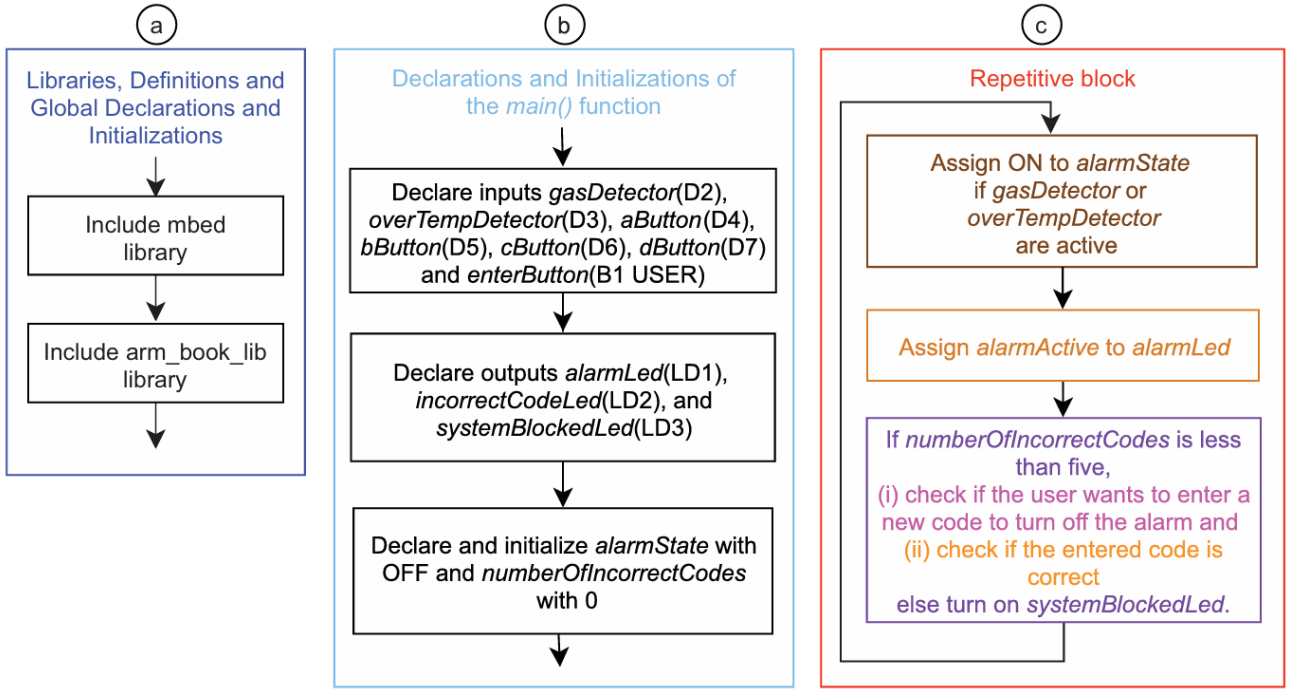
Do forever loop:

- If *gasDetector* or *overTempDetector* is pressed, assign *alarmState* with ON.
- Assign *alarmLed* with *alarmState*.
- If *aButton* and *bButton* are pressed, and neither *cButton* nor *dButton* are pressed, assign *alarmState* with OFF.

Example – Smart Home System

✓ **Challenge:** Block the System when Five Incorrect Codes are Entered.

Functionality	Buttons that should be pressed	Corresponding DigitalIn
To test if a given code turns off the Alarm LED	Any of A, B, C, and/or D + Enter	(D4, D5, D6, D7) + BUTTON1
The correct code that turns off the Alarm LED	A + B + Enter	D4 + D5 + BUTTON1
To turn off the Incorrect code LED and enable a new attempt to turn off the Alarm LED	A + B + C + D	D4 + D5 + D6 + D7
To turn off the System Blocked LED (that is turned on after entering five incorrect passwords)	No buttons available (power should be removed or B2 RESET button pressed)	



Example – Smart Home System

✓ **Challenge:** Block the System when Five Incorrect Codes are Entered.

```
1 #include "mbed.h"
2 #include "arm_book_lib.h"
3
4 int main()
5 {
6     DigitalIn enterButton(BUTTON1);
7     DigitalIn gasDetector(D2);
8     DigitalIn overTempDetector(D3);
9     DigitalIn aButton(D4);
10    DigitalIn bButton(D5);
11    DigitalIn cButton(D6);
12    DigitalIn dButton(D7);
13
14    DigitalOut alarmLed(LED1);
15    DigitalOut incorrectCodeLed(LED3);
16    DigitalOut systemBlockedLed(LED2);
17
18    gasDetector.mode(PullDown);
19    overTempDetector.mode(PullDown);
20    aButton.mode(PullDown);
21    bButton.mode(PullDown);
22    cButton.mode(PullDown);
23    dButton.mode(PullDown);
24
25    alarmLed = OFF;
26    incorrectCodeLed = OFF;
27    systemBlockedLed = OFF;
28
29    bool alarmState = OFF;
30    int numberOfIncorrectCodes = 0;
```

Include the libraries "mbed.h" and "arm_book_lib.h"

- Digital input objects named *gasDetector*, *overTempDetector* and *aButton* to *dButton*, are declared and assigned to D2 to D7, respectively.
- Digital output objects named *alarmLed* *incorrectCodeLed*, and *systemBlockedLed* are declared and assigned to LED1, LED2 and LED3, respectively.
- *gasDetector*, *overTempDetector* and *aButton* to *dButton* are configured with internal pull-down resistors.
- *alarmLed*, *incorrectCodeLed*, and *systemBlockedLed* are assigned OFF.
- *alarmState* is declared and assigned OFF.
- *numberOfIncorrectCodes* is declared and assigned 0 (zero).

Example – Smart Home System

✓ **Challenge:** Block the System when Five Incorrect Codes are Entered.

```
31
32      while (true) {
33
34          if ( gasDetector || overTempDetector ) {
35              alarmState = ON;
36          }
37
38          alarmLed = alarmState;
39
40          if ( numberOfIncorrectCodes < 5 ) {
41
42              if ( aButton && bButton && cButton && dButton && !enterButton ) {
43                  incorrectCodeLed = OFF;
44              }
45
46              if ( enterButton && !incorrectCodeLed && alarmState) {
47                  if ( aButton && bButton && !cButton && !dButton ) {
48                      alarmState = OFF;
49                      numberOfIncorrectCodes = 0;
50                  } else {
51                      incorrectCodeLed = ON;
52                      numberOfIncorrectCodes = numberOfIncorrectCodes + 1;
53                  }
54              }
55          } else {
56              systemBlockedLed = ON;
57          }
58      }
59  }
```

Do forever loop:

- If *gasDetector* or *overTempDetector* are active, assign *alarmState* with ON.
- Assign *alarmLed* with *alarmState*.
- If *numberOfIncorrectCodes* is less than five, implement the logic (discussed in the text) in order to determine the state of *incorrectCodeLed* and if *alarmState* must be assigned with OFF, and if *numberOfIncorrectCodes* must be set to 0 (zero) or increased by 1.
- If *numberOfIncorrectCodes* is greater than or equal to five, assign systemBlockedLed with ON.

Example – Smart Home System

- ✓ The smart home system is now connected to the character LCD display using the I2C bus

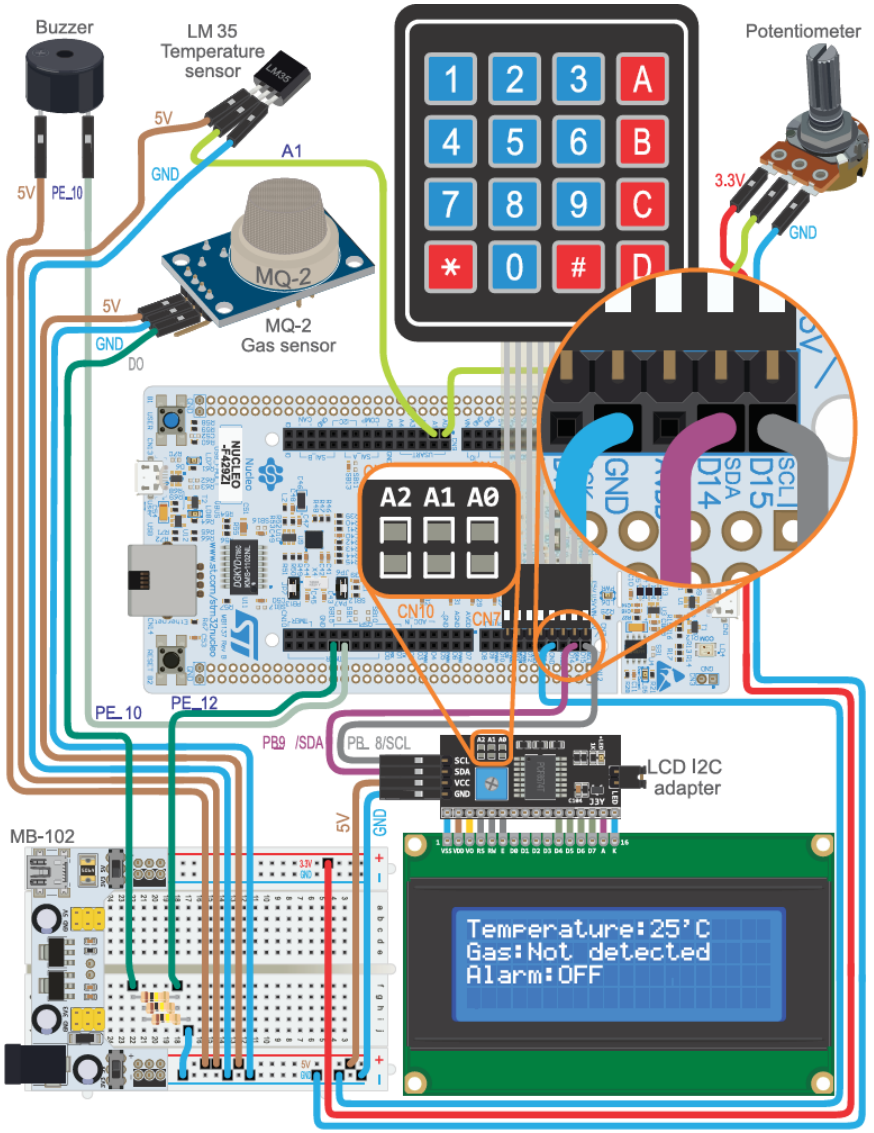


Figure 6.16 The smart home system is now connected to the character LCD display using the I2C bus.

Why ARM Cortex-M-based μP?

✓ Many legacy architectures (like 8051 or basic PICs) are being phased out in advanced product lines. The main reasons for the adoption of ARM Cortex-M-based μP are:

1. **Performance & Efficiency**:
   - 32-bit RISC, higher clock speeds (up to 200 MHz+), better IPC than 8/16-bit AVR, PIC, 8051.
   - Ultra-low-power modes (e.g., Cortex-M0+ at microamps) vs. power-hungry legacy MCUs.
   - Cortex-M4/M7 include FPU and DSP, ideal for signal processing, absent in legacy architectures.
2. **Scalability**:
   - Wide range (M0 to M7) for simple to complex tasks, with software compatibility.
   - 32-bit addressing (up to 4GB) vs. 64KB/128KB limits of 8051/PIC/AVR.
   - Standardized ARM architecture reduces vendor lock-in.

Why ARM Cortex-M-based μP?

4. **Memory & Peripherals**:
    - Larger flash (MBs) and SRAM (100s KB) vs. limited 8051/PIC/AVR memory.
    - Modern peripherals (USB, CAN, Ethernet) vs. basic legacy options.
5. **Future-Proofing**:
    - Industry standard with widespread adoption in IoT, automotive, industrial.
    - Vendors (e.g., Microchip) shifting to Cortex-M (e.g., SAM series).
6. **IoT & Security**:
    - Built-in Bluetooth, Wi-Fi, TrustZone (M23/M33), secure boot for IoT.
    - Legacy MCUs lack modern connectivity and security features.
7. **Commercial Use & Reliability**
    - Offers industrial-grade reliability and industrial certifications.

*** While ARM Cortex-M microcontrollers provide superior performance and modern features, AVR, PIC, and 8051-based microcontrollers are well-suited for simpler projects with no strict processing or low-power constraints, or for hobbyist and learning purposes due to their simple architectures, legacy system compatibility, and abundant educational resources.*

Applications

Choosing Right μC

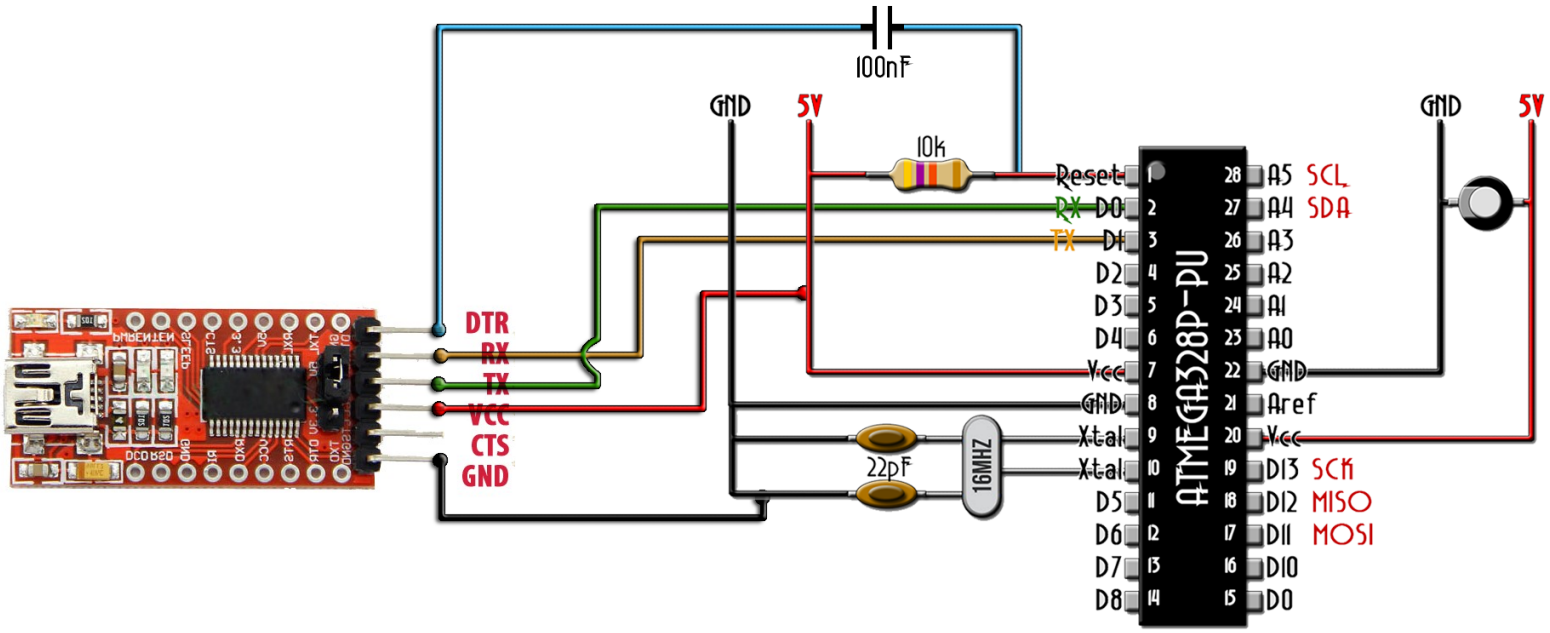
Project Priority	Scenario	Recommended	Why Choose?
Ease of Use	Beginner/Educational Projects	ATmega328	Arduino IDE and vast community simplify learning and prototyping.
	Rapid Prototyping	ATmega328	Quick setup with Arduino libraries speeds up initial development.
Cost Efficiency	Simple Hobby Projects	ATmega328	Low cost ($1–$2/unit) and sufficient for basic tasks like sensors or LEDs.
	Cost-Sensitive Projects	ATmega328	Cheaper for simple designs, though STM32 may save BOM cost in complex systems.
Performance	High-Performance Applications	STM32-based	32-bit ARM Cortex-M (up to 168 MHz) handles complex algorithms and real-time tasks.
	Advanced Peripheral Needs	STM32-based	Built-in USB, CAN, Ethernet reduce external components for seamless integration.
Power Efficiency	Battery-Powered Devices	STM32-based	Advanced low-power modes (microamps) optimize battery life for IoT or wearables.
Reliability & Scalability	Commercial Products	STM32-based	ECC memory, certifications, and 10+ year availability ensure long-term reliability.
	Long-Term Support	STM32-based	Vendor support and supply guarantees suit extended product lifecycles.
	Connectivity (Wi-Fi/Ethernet)	STM32-based	Native connectivity options simplify design vs. ATmega328's external modules.

Applications

- ✓ Student Case Study!

Minimum Interfacing Circuit

Review the concept of bootloader in μP



Prototyping Breadboard Connection

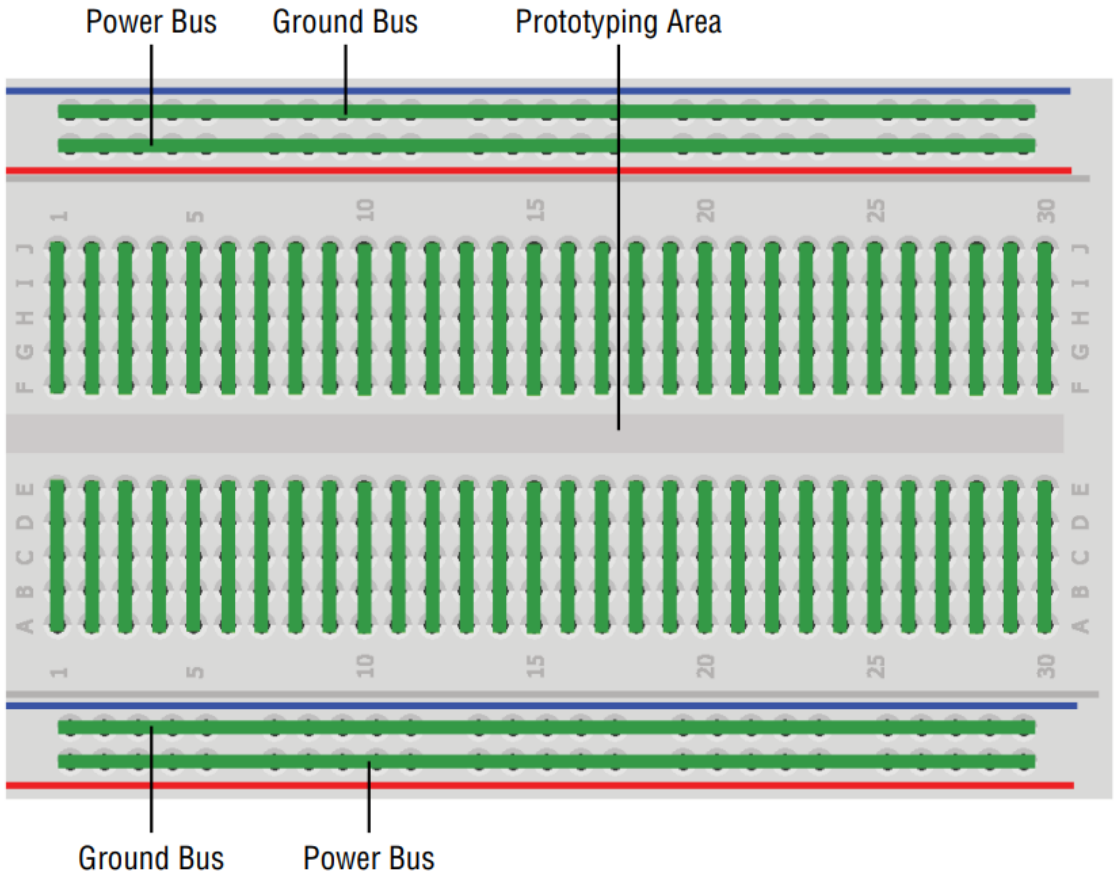
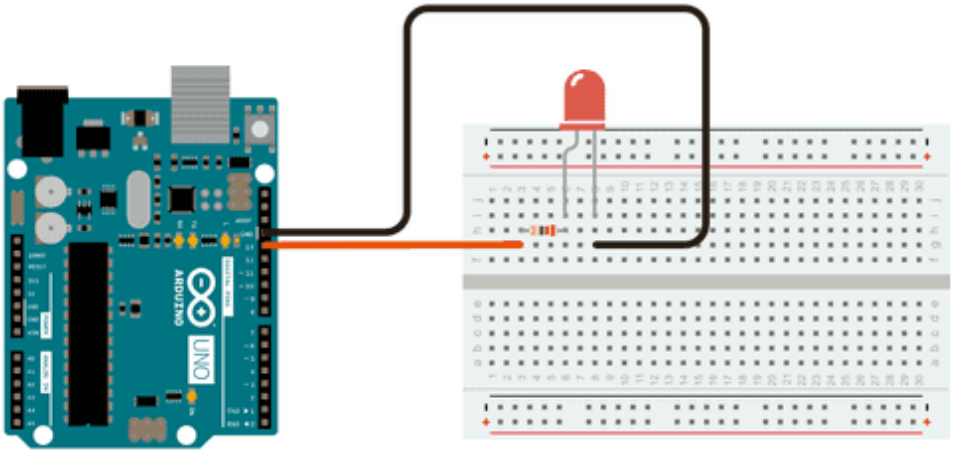
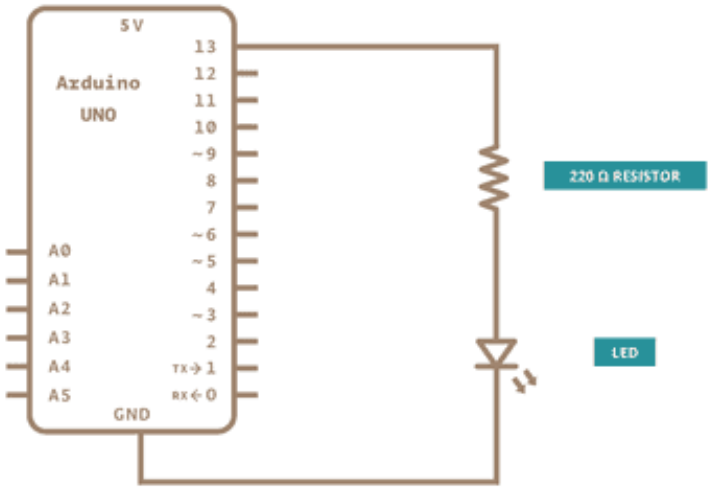


Figure 2-1: Breadboard electrical connections

Hello World - Blinking LEDs






```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

Programming using AVR Assembly

- ✓ AVR assembly is low-level and hardware-specific, while Arduino abstracts hardware control using C++.
- ✓ **Basic AVR Assembly Program:**
  - Toggles PORTB pins repeatedly

```
; set power-up/reset vector
.ORG 0X0000
RJMP MAIN

; code entry point
.ORG 0X0100
MAIN:
LDI             R16, 0XFF       ; load R16 with 0b11111111
OUT             DDRB, R16       ; set port B data direction register

; endless loop - toggles port B pins on and off
LOOP:
LDI             R16, 0X00       ; load R16 with zero
OUT             PORTB, R16      ; write to port B
LDI             R16, 0XFF       ; now load it with 0xFF
OUT             PORTB, R16      ; and write it to port B
RJMP  LOOP        ; jump back and do it again
```

Uploading AVR Assembly Program (CMD)

- Write `.s` file with AVR instructions
- Assemble with `avr-as` → generates `.o` file
- Link with `avr-gcc` → produces .elf
- Convert to `.hex` using `avr-objcopy`
- Upload with `avrdude` using **USBtinyISP** or **ICSP programmer**

```
avrdude -c usbtiny -p m328p -U flash:w:program.hex
```

Equivalent Arduino Program

```
void setup() {
  DDRB = 0xFF;        // Set PORTB as output
}

void loop() {
  PORTB = 0x00;       // Turn OFF PORTB
  PORTB = 0xFF;       // Turn ON PORTB
}
```

Programming using Microchip Studio

- ✓ Programming the Atmega328P involves writing code, compiling it, and uploading the firmware to the microcontroller using hardware tools.
  - **Write code** in C/C++ using AVR-GCC within Microchip Studio's IDE
  - Build the project to **generate a HEX file** for the target microcontroller
  - **Connect Atmega328P to a programmer** (e.g., Atmel-ICE or USBasp) via the ISP (In-System Programming) interface (MOSI, MISO, SCK, RESET, VCC, GND)
  - **Select the correct device** and tool in Microchip Studio, then use the "Start Without Debugging" or "Device Programming" option to flash the HEX file
  - Ensure power supply and clock source are correctly configured on the hardware for successful programming.

Programming using Arduino IDE

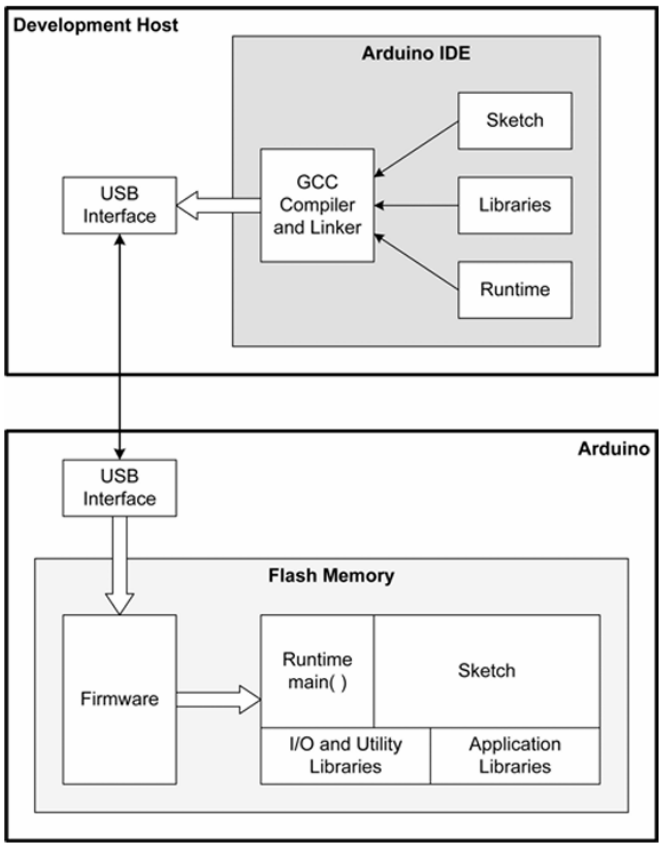
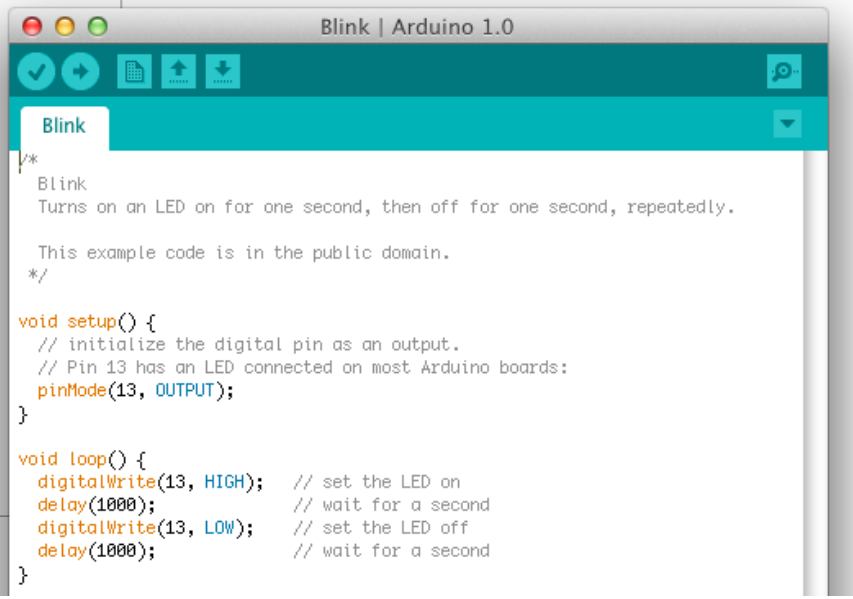


Figure 5-4. Arduino software organization

Uploading Guidance

Uploading Arduino Program (via Arduino IDE):

- Write program in Arduino IDE
- Connect board via USB
- Select board and port from **Tools** menu
- Click **Upload** – bootloader handles flashing using serial (D0/D1)



Commonly Used Libraries in Arduino

- ✓ Sketch → Import Library from the IDE toolbar. This will also show any libraries that you have added to the environment.
  - We can add libraries manually.

EEPROM
Supports reading and writing to "permanent" storage using an AVR's built-in EEPROM

Ethernet
Used with the Arduino Ethernet shield for Ethernet connectivity

Firmata
Provides communications with applications on the computer using a standard serial protocol

GSM
Used with the GSM shield to connect to a GSM/GPRS network

LiquidCrystal
Contains functions for controlling liquid crystal displays (LCDs)

SD
Provides support for reading and writing SD flash memory cards

Servo
A collection of functions for controlling servo motors

SPI
Supports the use of the Serial Peripheral Interface (SPI) bus

SoftwareSerial
Implements serial communication on any of the digital pins

Stepper
A collection of functions for controlling stepper motors

TFT
Provides functions for drawing text, images, and shapes on the Arduino TFT screen

WiFi
Supports the Arduino WiFi shield for wireless networking

Wire
Supports the two-wire interface (TWI/I2C) for sending and receiving data over a network of devices or sensors

Esplora
Provides functions to access the various actuators and sensors mounted on the Esplora board (used with Esplora only)

USB
Used with the Leonardo, Micro, Due, and Esplora boards for serial I/O over the USB connection

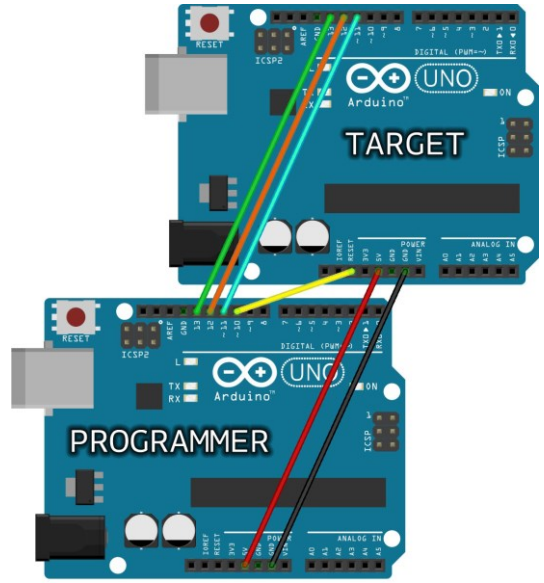
Keyboard
Sends keystrokes to an attached computer

Mouse
Controls cursor movement on a connected computer

Commonly Used Sensors in Arduino

- ✓ Temperature Humidity Sensors
- ✓ Vibration Sensors
- ✓ Pressure Sensors
- ✓ Hall Effect Sensor
- ✓ Infrared Sensors
- ✓ Light Sensors
- ✓ Audio Sensors
- ✓ Contact/Touch Sensors
- ✓ Load Cell Sensors
- ✓ Ultrasonic Sensors
- ✓ Water/Moisture Sensors
- ✓ Accelerometer Sensors
- ✓ Limit Switch
- ✓ Rotary Encoder

Uploading Without the Bootloader



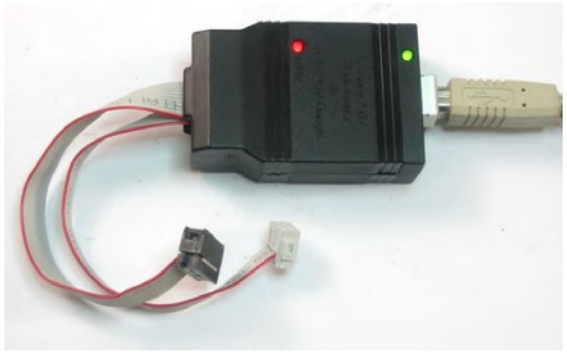


Figure 6-10. The USBtinyISP from Adafruit (assembled)

Figure 6-11. The Pocket AVR Programmer from SparkFun

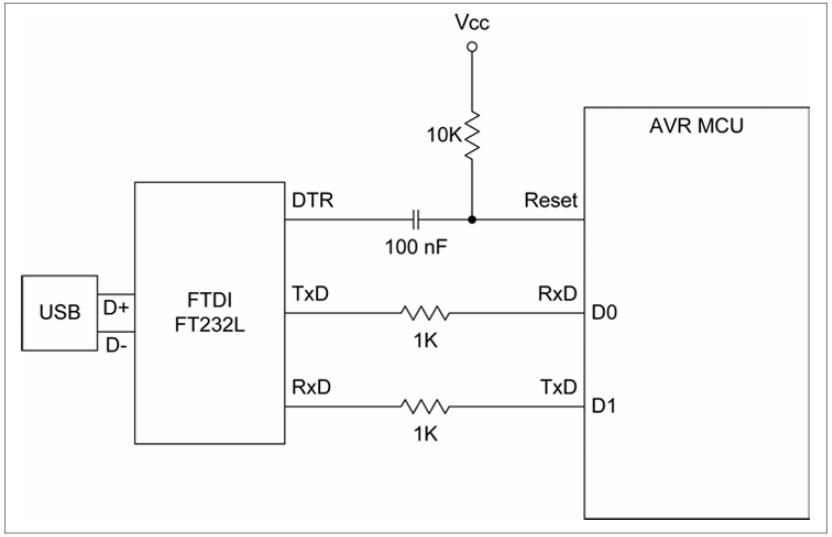


Figure 6-9. Arduino USB interface using an FTDI converter IC

Digital Read

Concept - Pull-up & Pull-down

✓ Pull-up:

**Review the concept of floating state and leakage current.*

- A pull-up resistor connects an input pin to a high voltage (usually Vcc) to ensure it reads as HIGH when not actively driven.

✓ Pull-down:

- A pull-down resistor connects an input pin to ground to ensure it reads as LOW when not actively driven.

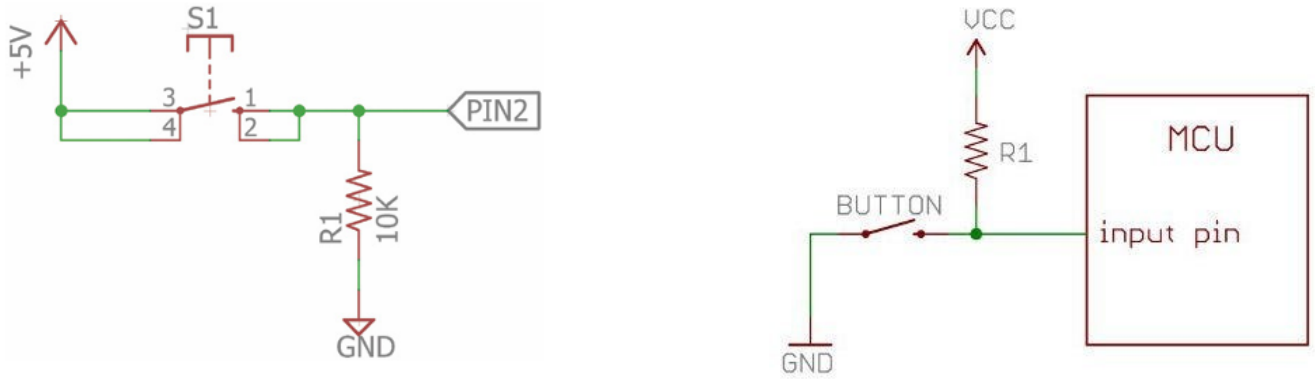
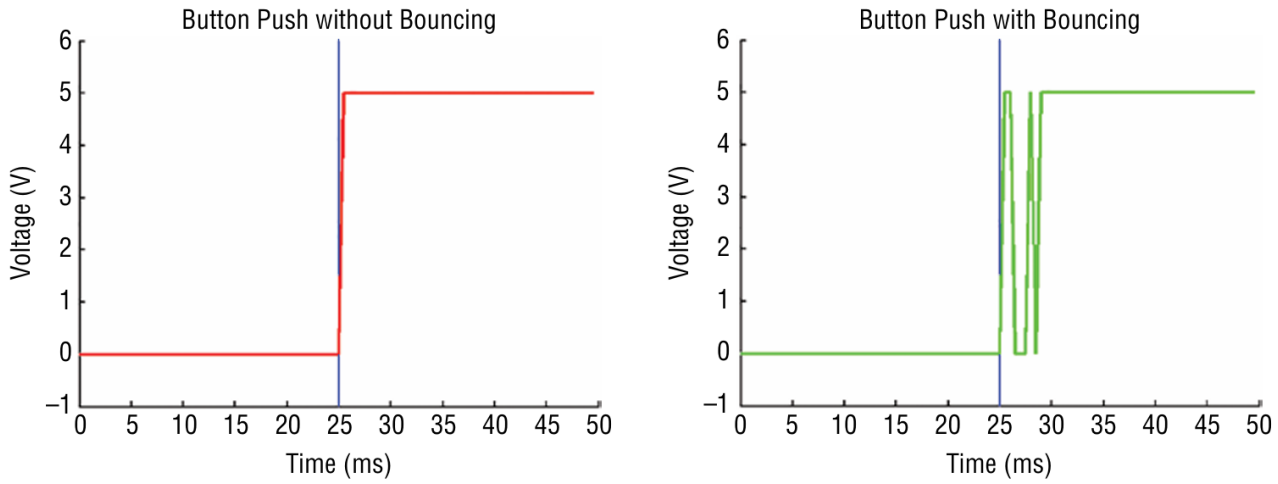


Figure 2-5: Pushbutton input with pull-down resistor schematic

Digital Read (Cont..)

Concept – Switch Debouncing

- ✓ When you push a button down, the signal you read does not just go from low to high; it bounces up and down between those two states for a few milliseconds before it settles.
  - This creates problem in toggle-sensitive logics.



**Review hardware and software approaches to handle bouncy buttons.*

Simple Digital Input/Output

Switch & LED Interfacing

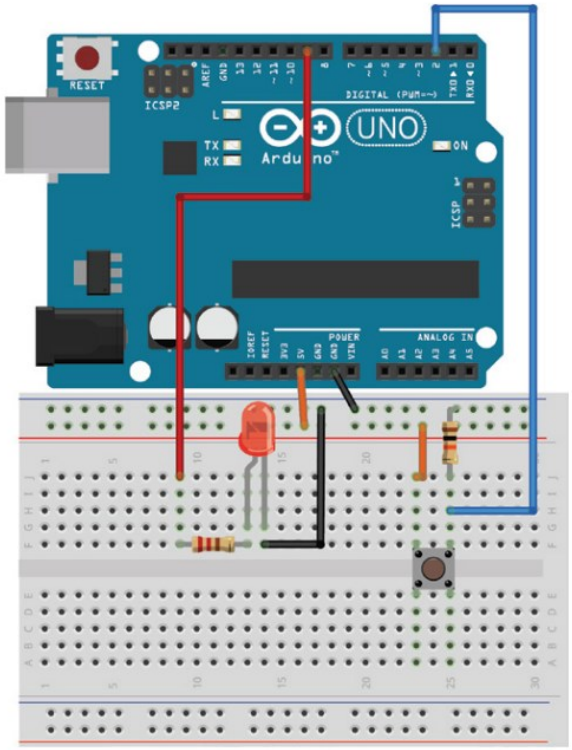


Figure 2-6: Wiring an Arduino to a button and an LED

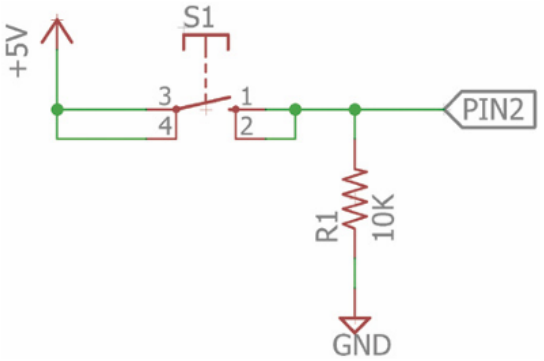


Figure 2-5: Pushbutton input with pull-down resistor schematic

Simple LED control with a button–led_button.ino

```
const int LED=9;          // The LED is connected to pin 9
const int BUTTON=2;       // The Button is connected to pin 2

void setup()
{
  pinMode (LED, OUTPUT);      // Set the LED pin as an output
  pinMode (BUTTON, INPUT);    // Set button as input (not required)
}

void loop()
{
  if (digitalRead(BUTTON) == LOW)
  {
    digitalWrite(LED, LOW);
  }
  else
  {
    digitalWrite(LED, HIGH);
  }
}
```

Digital Sensors

- ✓ Digital sensors produce a discrete voltage signal that has only two states: HIGH or LOW.
  - The advantage of digital sensors is that they are more immune to noise and interference, and they can communicate directly with Arduino using protocols, such as I2C, SPI, or UART.

- ✓ **Example:** PIR Motion Sensor
  - Detects movement by sensing infrared radiation changes and outputs HIGH when motion is detected

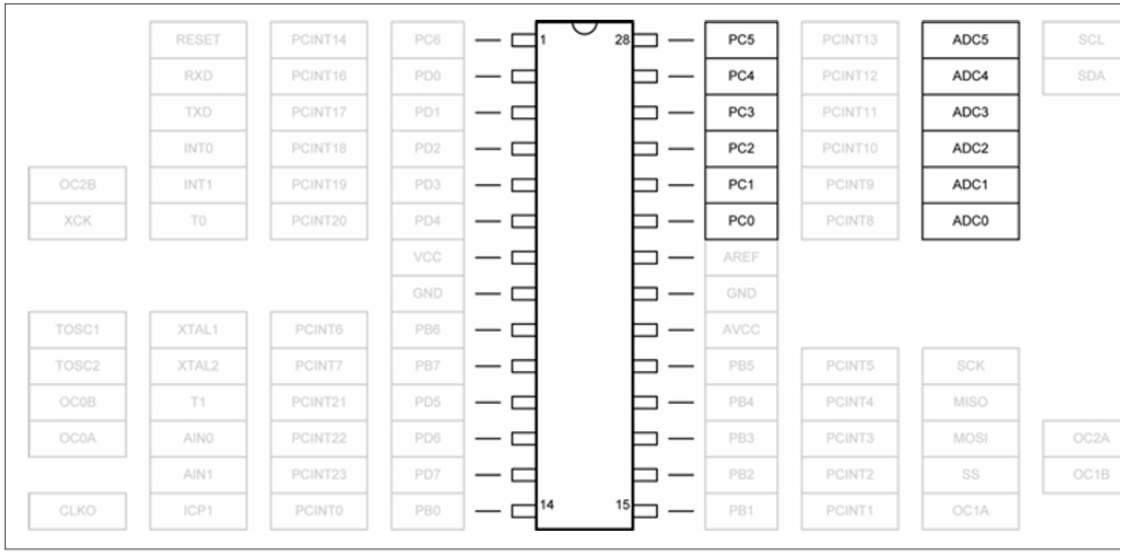
- ✓ **Example:** Touch Sensor
  - Detects physical touch or proximity and outputs HIGH when touched or triggered.

Digital Sensors

- ✓ Student's Work - Review the various digital sensors interfacing examples!

ADC in Atmega328P

- ✓ The ATmega328P has a built-in 10-bit Analog-to-Digital Converter (ADC).
- ✓ This ADC allows the microcontroller to read analog voltages and convert them into a 10-bit digital value (0-1023).
  - The ADC can only measure a single analog input at a time.
  - The ADC uses a reference voltage (via AREF pin).
  - Common reference voltage range is 1.0 to VCC.



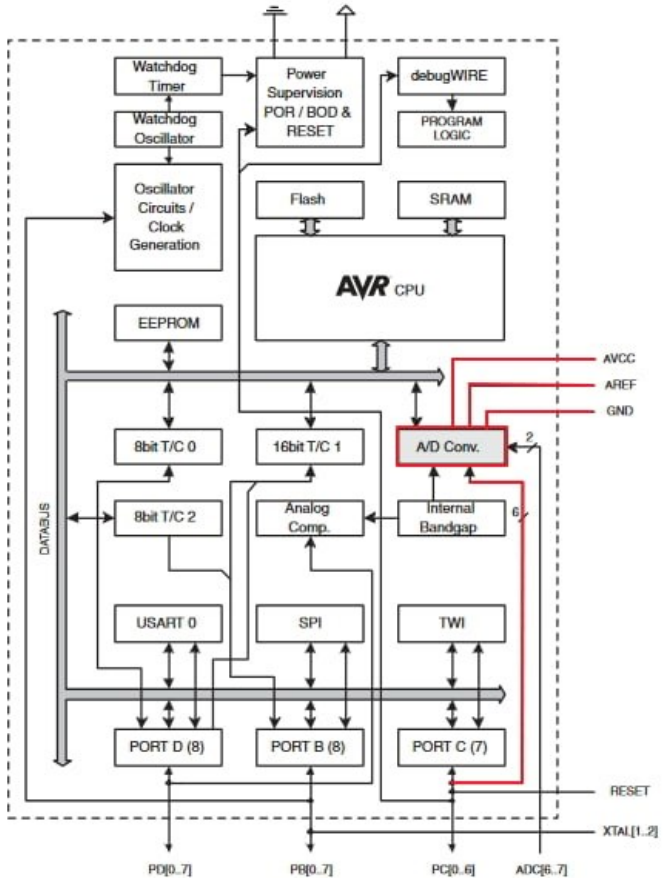
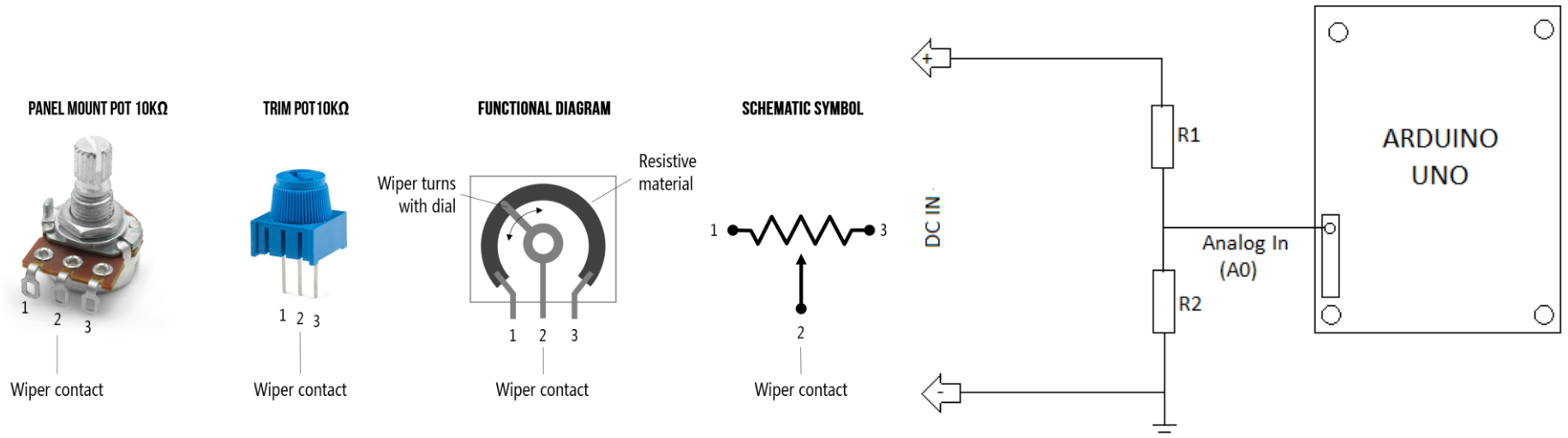


Figure 3-4. ATmega168/328 microcontroller ADC input pins

Analog Input

- ✓ ADC Functionality:
  - ATmega328's 10-bit ADC converts analog signals from six pins (ADC0–ADC5) into digital values (0–1023) using a reference voltage, typically 5V.
- ✓ Configuration:
  - Set via ADMUX for channel and reference, and ADCSRA for enabling ADC and prescaler, supporting single or continuous conversion modes.



Analog Sensors

E.g.: TMP36 Temperature Sensor

- ✓ Temperature alert system: Cold → Blue, Hot → Red, Normal → Green.
  - Output voltage: 0V at −50°C and 1.75V at 125°C
  - Temperature (in °C) = (100 × voltage) − 50

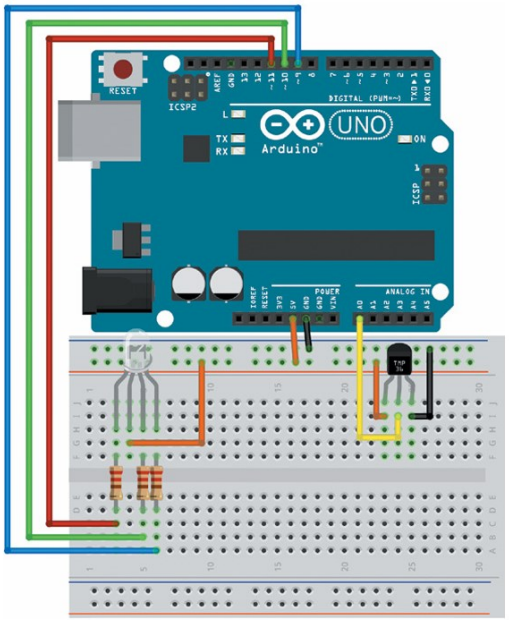


Figure 3-8: Temperature sensor circuit

Temperature alert sketch–tempalert.ino

```
// Temperature Alert!
const int BLED=9;           // Blue LED Cathode on Pin 9
const int GLED=10;          // Green LED Cathode on Pin 10
const int RLED=11;          // Red LED Cathode on Pin 11
const int TEMP=0;           // Temp Sensor is on pin A0

const int LOWER_BOUND=139; // Lower Threshold
const int UPPER_BOUND=147; // Upper Threshold

int val = 0;                // Variable to hold analog reading


void setup()
{
  pinMode (BLED, OUTPUT); // Set Blue LED as Output
  pinMode (GLED, OUTPUT); // Set Green LED as Output
  pinMode (RLED, OUTPUT); // Set Red LED as Output
}
```

```
void loop()
{
  val = analogRead(TEMP);

  // LED is Blue
  if (val < LOWER_BOUND)
  {
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
  }
  // LED is Red
  else if (val > UPPER_BOUND)
  {
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, HIGH);
  }
  // LED is Green
  else
  {
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, HIGH);
  }
}
```

Analog Sensors

E.g.: Light Dependent Resistor (LDR)

- ✓ Automatic Night Light - Project

Automatic nightlight sketch—nightlight.ino

```
// Automatic Night Light

const int WLED=9;         // White LED Anode on pin 9 (PWM)
const int LIGHT=0;        // Light Sensor on Analog Pin 0
const int MIN_LIGHT=200; // Minimum Expected light value
const int MAX_LIGHT=900; // Maximum Expected Light value
int val = 0;              // Variable to hold the analog reading

void setup()
{
  pinMode(WLED, OUTPUT); // Set White LED pin as output
}

void loop()
{
  val = analogRead(LIGHT);                      // Read the light sensor
  val = map(val, MIN_LIGHT, MAX_LIGHT, 255, 0); // Map the light reading
  val = constrain(val, 0, 255);                 // Constrain light value
  analogWrite(WLED, val);                       // Control the White LED
}
```

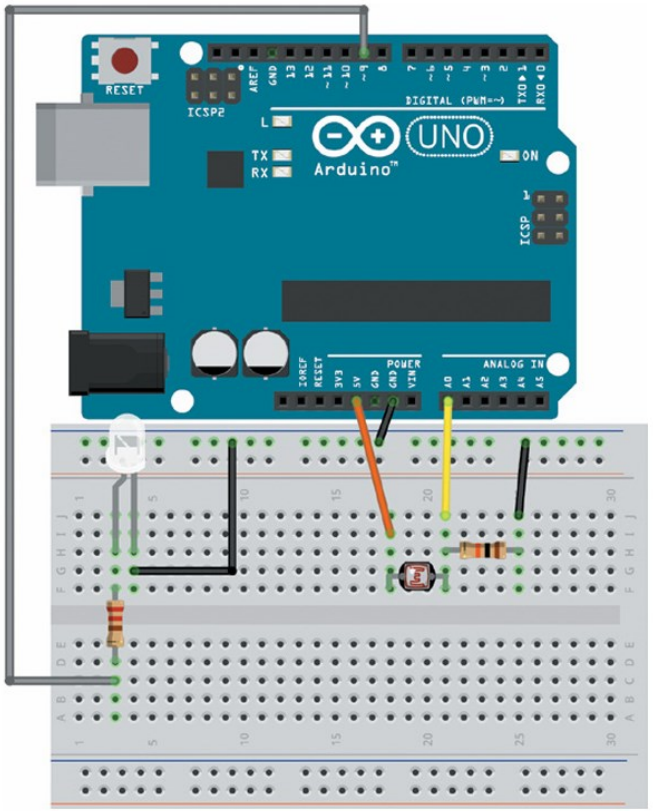


Figure 3-11: Photoresistor circuit

Concept – Analog Output using PWM

- ✓ Arduino can use the `analogWrite()` command to generate PWM signals that can emulate a pure analog signal.
  - Generated using timers within the microcontroller
  - UNO uses ~ marked pins (3, 5, 6, 9, 10, and 11) for PWM signal.

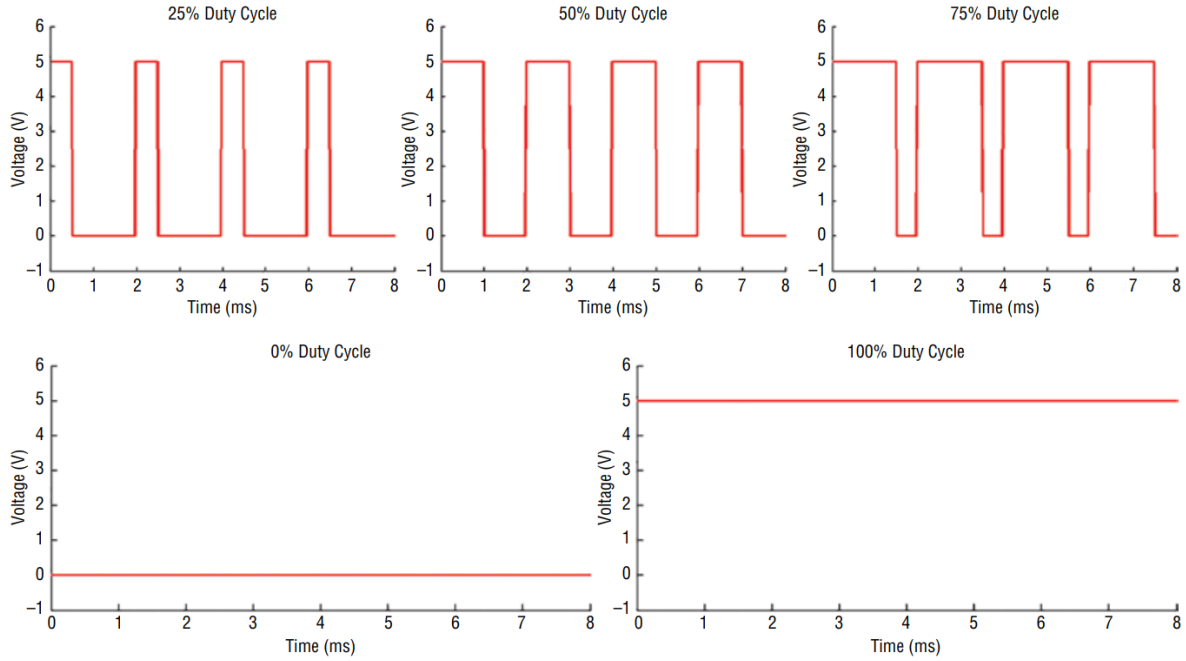
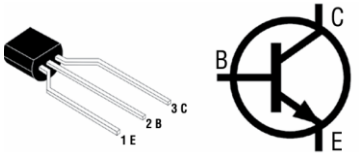


Figure 2-4: PWM signals with varying duty cycles

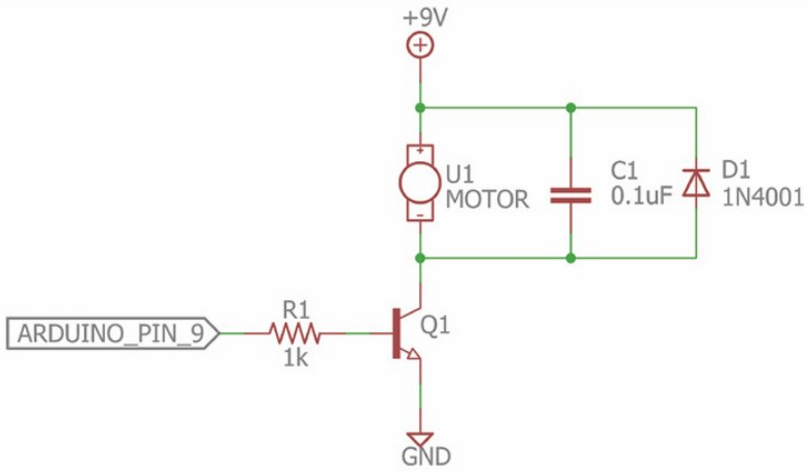
Exercise – Projects with PWM Concept

- ✓ Controlling LED brightness and color transition in RGB LED
- ✓ Controlling DC Motor speed
- ✓ Controlling Servo Motor position
- ✓ Generating audio signals
- ✓ Etc.

DC Motor using Transistor



- ✓ Microcontrollers can't directly drive high-current devices like motors. A transistor amplifies the low-current signal or acts as a switch to control higher current from an external source.
  - GPIO pins output limited current (<50 mA)
  - High-power devices require external supply
  - Transistor operates in saturation/cut-off to switch
    - or amplify the signal



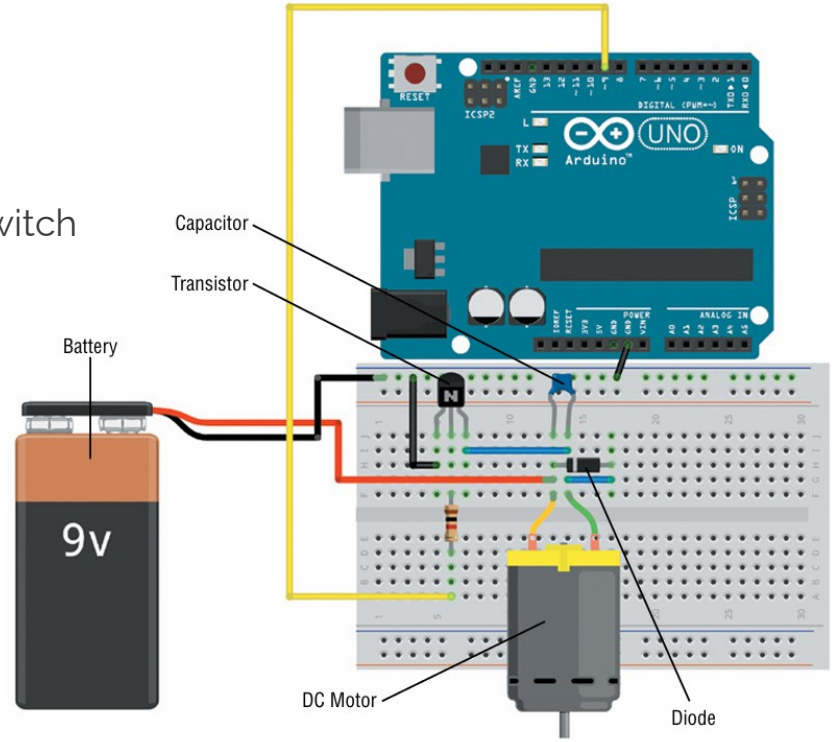


Figure 4-3: DC motor wiring

DC Motor using Transistor - Coding

Automatic speed control–motor.ino

```
//Simple Motor Speed Control Program

const int MOTOR=9;    //Motor on Digital Pin 9

void setup()
{
    pinMode (MOTOR, OUTPUT);
}

void loop()
{
    for (int i=0; i<256; i++)
    {
        analogWrite(MOTOR, i);
        delay(10);
    }
    delay(2000);
    for (int i=255; i>=0; i--)
    {
        analogWrite(MOTOR, i);
        delay(10);
    }
    delay(2000);
}
```

Actuator interfacing with Atmega328P

DC Motor using H-Bridge (L293D)

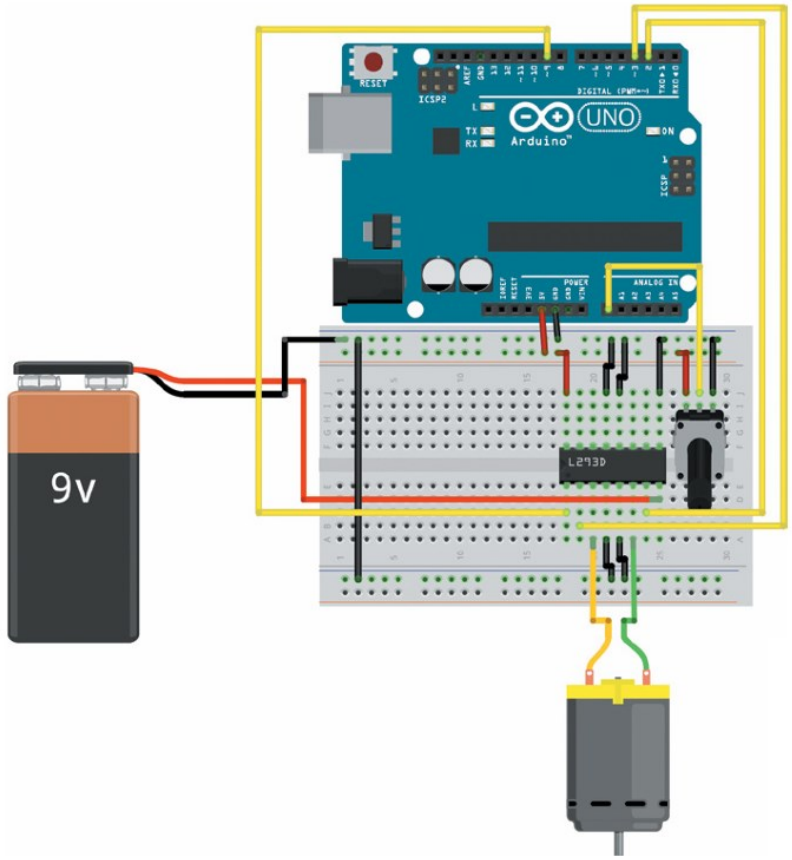
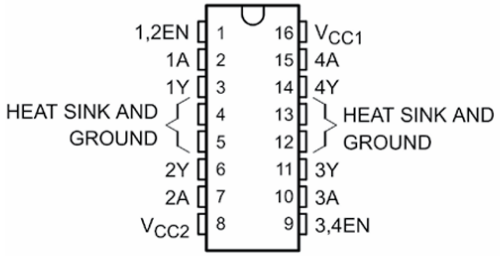


Figure 4-7: H-bridge wiring diagram

```
//Motor goes forward at given rate (from 0-255)
void forward (int rate)
{
    digitalWrite(EN, LOW);
    digitalWrite(MC1, HIGH);
    digitalWrite(MC2, LOW);
    analogWrite(EN, rate);
}

//Motor goes backward at given rate (from 0-255)
void reverse (int rate)
{
    digitalWrite(EN, LOW);
    digitalWrite(MC1, LOW);
    digitalWrite(MC2, HIGH);
    analogWrite(EN, rate);
}

//Stops motor
void brake ()
{
    digitalWrite(EN, LOW);
    digitalWrite(MC1, LOW);
    digitalWrite(MC2, LOW);
    digitalWrite(EN, HIGH);
}
```



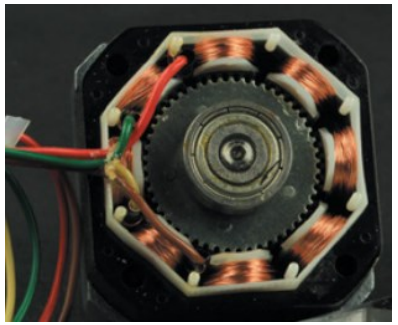
FUNCTION TABLE (each driver)

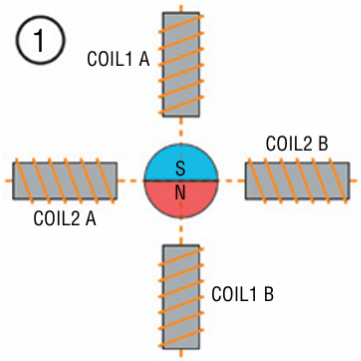
INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

H = high-level, L = low-level
X = irrelevant
Z = high-impedance (off)
† In the thermal shutdown mode, the output is in a high-impedance state regardless of the input levels.

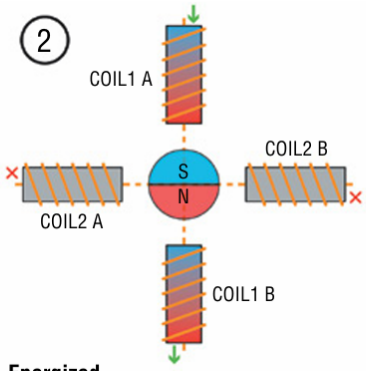
Figure 4-6: H-bridge pin-out and logic table

Stepper Motor Interfacing

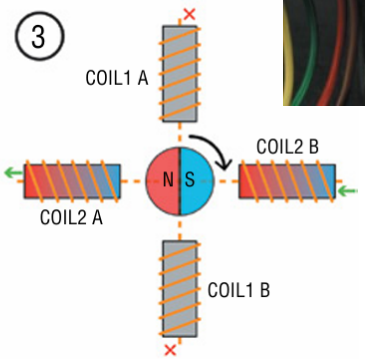




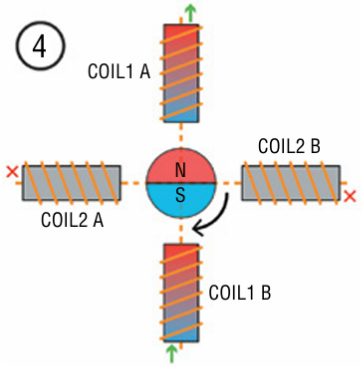
Not Energized
No current flowing through coils. Rotor can spin freely.



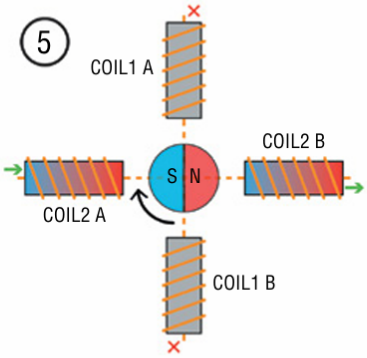
Energized
Current flows through coil 1, inducing a magnetic field. Rotor locks in position.



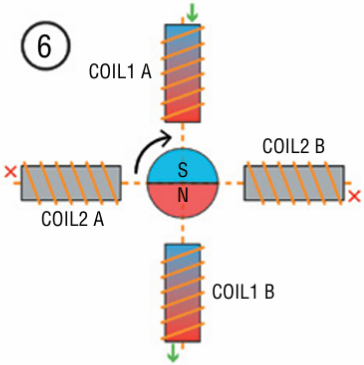
Energized, Rotating
Current flows through coil 2, inducing a new magnetic field. Permanent magnet (shaft) rotates to align to it.



Energized, Rotating
Current flows through coil 1 again, in opposite direction from step 2. Magnetic field direction now opposite of step 2. Shaft rotates to align to it.



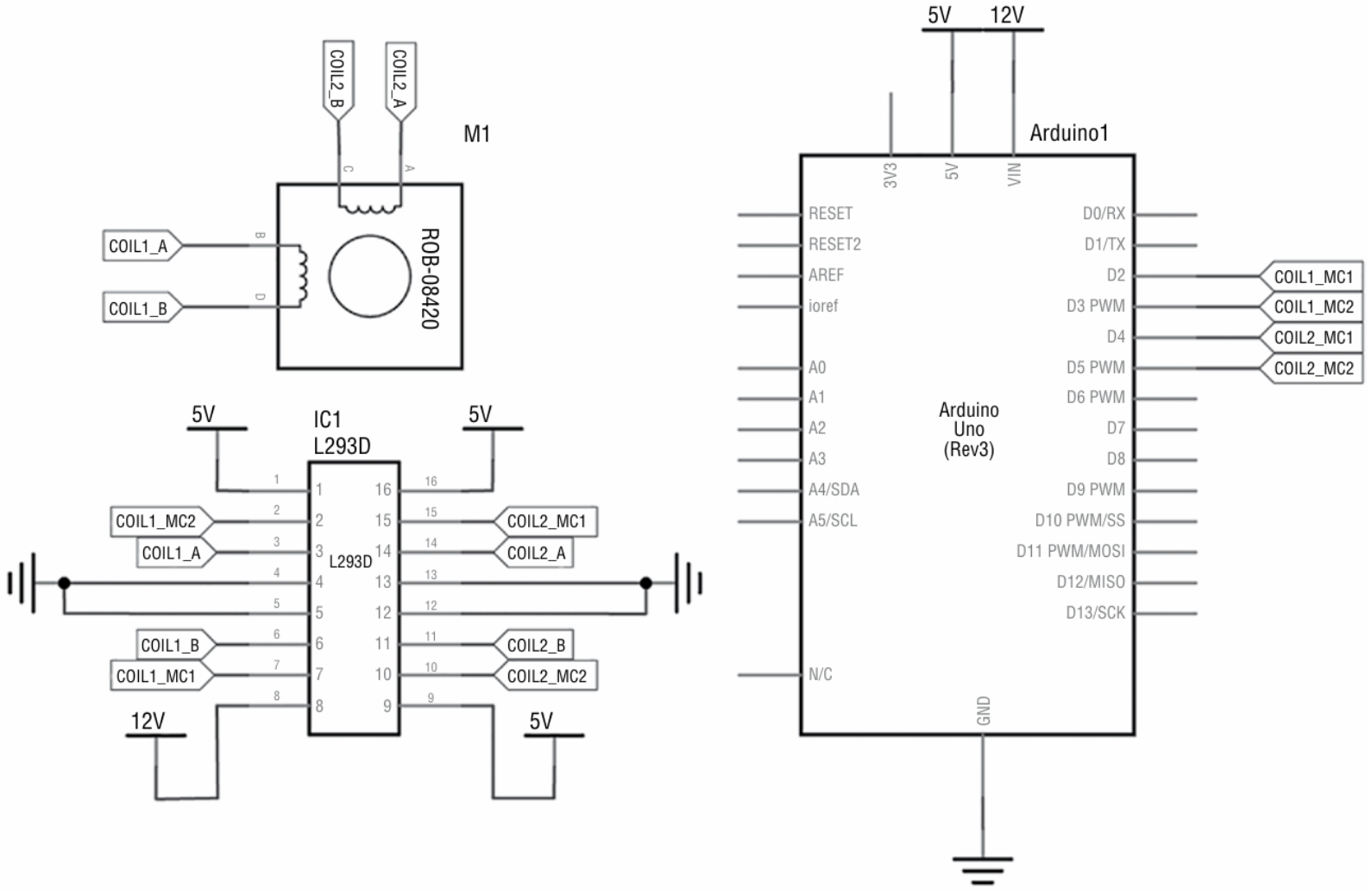
Energized, Rotating
Current flows through coil 2, again, in opposite direction from step 3. Magnetic field direction now opposite of step 3. Shaft rotates to align to it.



Energized, Rotating
Step 2 repeats. Shaft rotates again, returning to starting position.

Actuator interfacing with Atmega328P

Stepper Motor Interfacing (Cont..)



Actuator interfacing with Atmega328P

Stepper Motor Interfacing (Cont..)

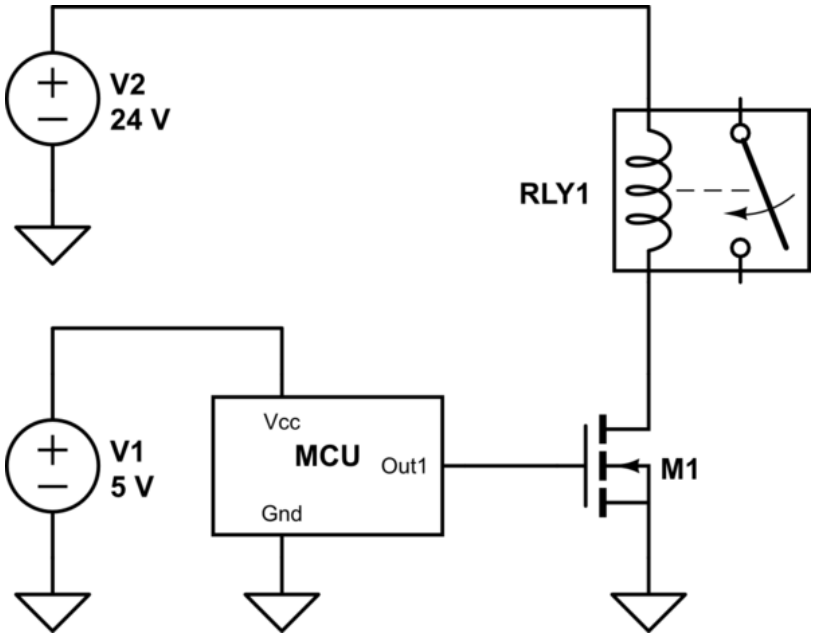
Simple stepper control—stepper.ino

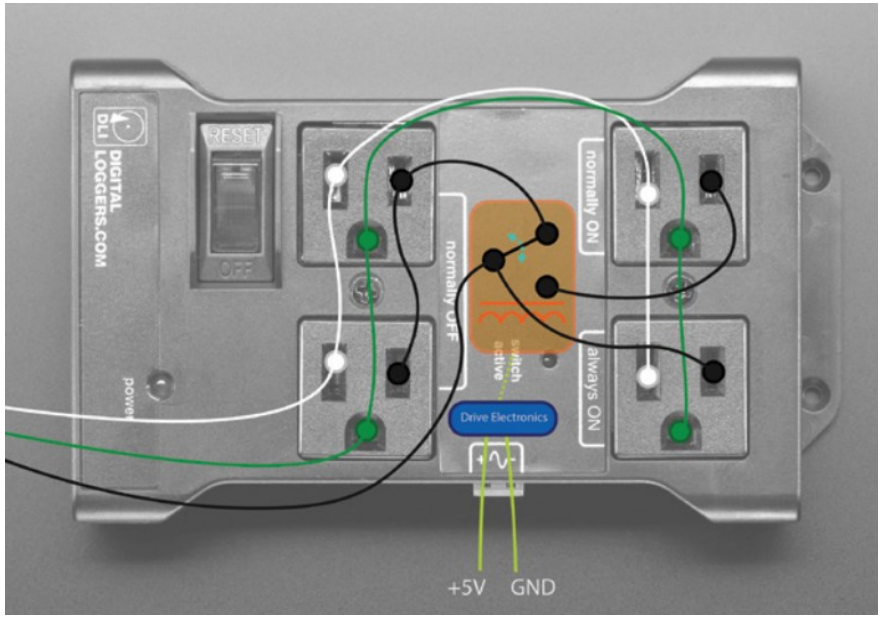
```
//Simple Stepper Control with an H-Bridge

#include <Stepper.h>

//Motor Constants
//Most NEMA-17 Motors have 200 steps/revolution
const int STEPS_PER_REV = 200; //200 steps/rev

//H-Bridge Pins
const int COIL1_MC1  = 2;  //COIL 1 Switch 1 Control
const int COIL1_MC2  = 3;  //COIL 1 Switch 2 Control
const int COIL2_MC1  = 4;  //COIL 2 Switch 1 Control
const int COIL2_MC2  = 5;  //COIL 2 Switch 2 Control

// Initialize the stepper library - pass it the Switch control pins
Stepper myStepper(STEPS_PER_REV, COIL1_MC1, COIL1_MC2, COIL2_MC1, COIL2_MC2);

void setup()
{
  //Set the stepper speed
  myStepper.setSpeed(60); // 60 RPM
}

void loop()
{
  // step one revolution in one direction:
  myStepper.step(STEPS_PER_REV);
  delay(500);

  // step one revolution in the other direction:
  myStepper.step(-STEPS_PER_REV);
  delay(500);
}
```

**Explore the "Stepper.h" library file to understand its low-level implementation.*

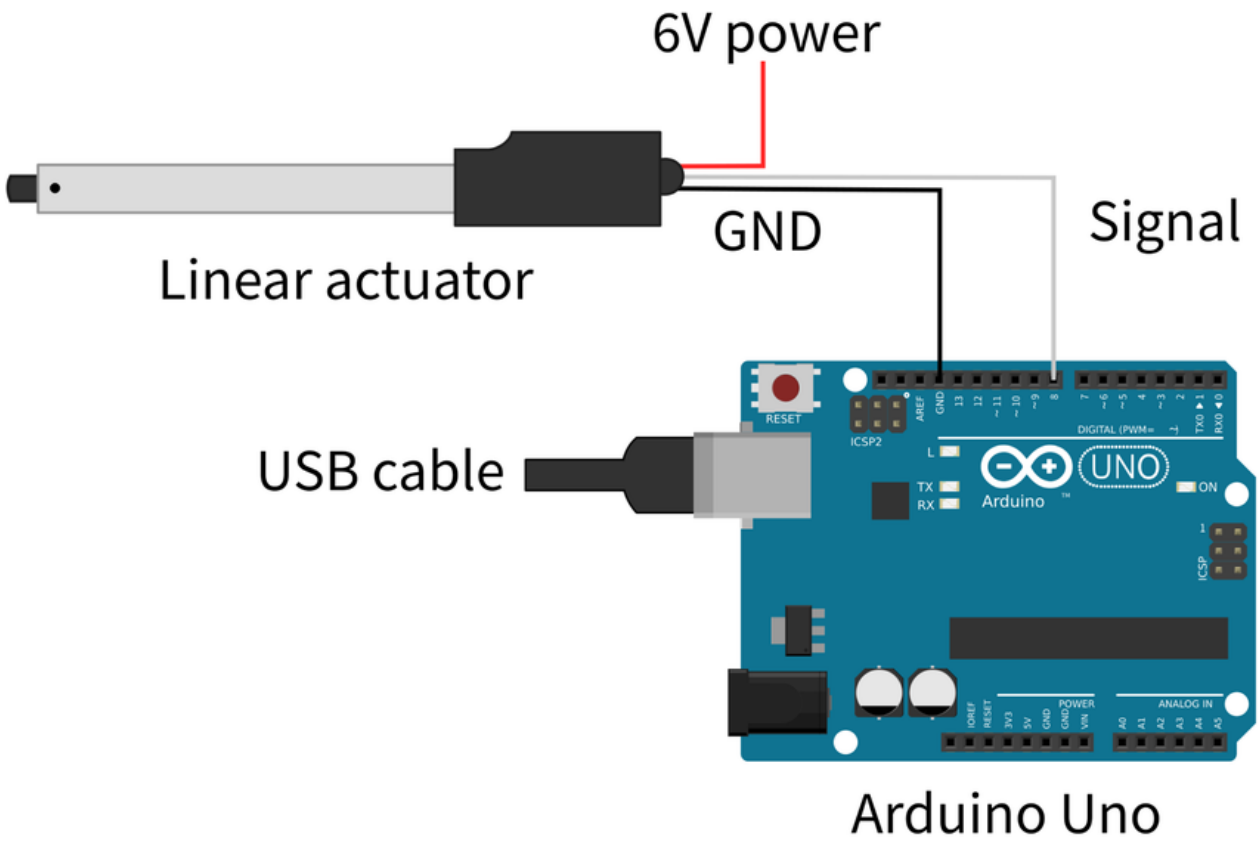
Working with Relays

- ✓ How to control high-power equipment using μP signals?
  - We use electromechanical relays or Power Transistors.





Example: Linear Actuator



Case Study!

Project: Self-driving roving robot

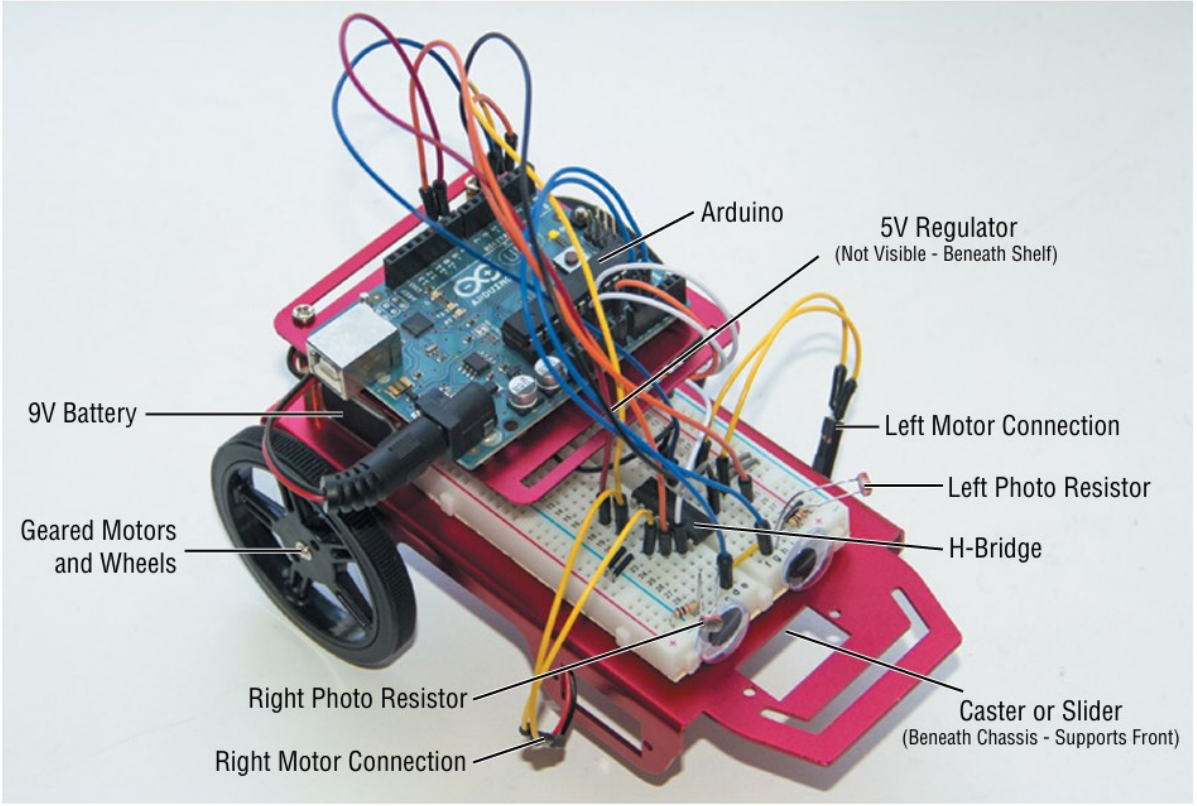
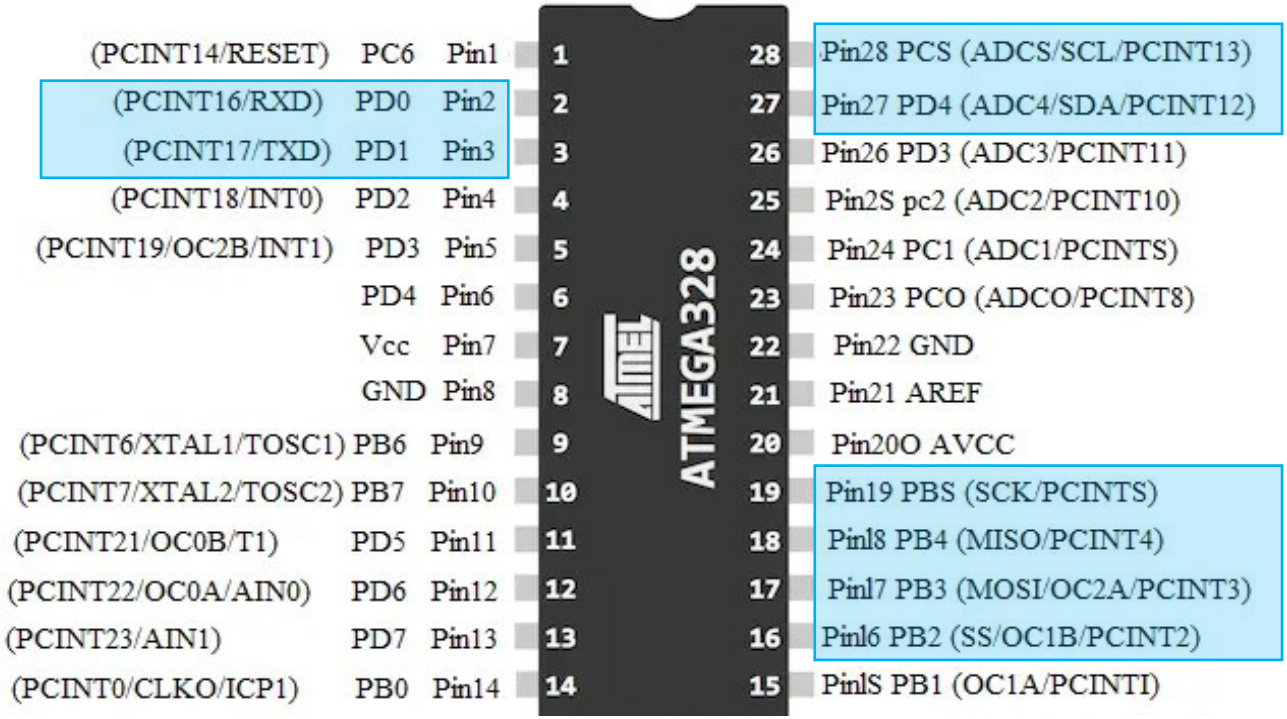


Figure 4-13: Fully built autonomous rover

Implementation of Communication Protocols

- ✓ Atmega328P supports communication protocols like UART, SPI, and I2C through its built-in hardware modules.
  - UART enables asynchronous serial communication with configurable baud rate
  - SPI supports full-duplex communication with master-slave architecture
  - I2C allows multi-master communication using address-based data transfer



UART

- ✓ Many AVR μP has a built-in USART (universal synchronous / asynchronous receiver-transmitter), also referred to as a UART.
- ✓ Used for point-to-point serial communication between two devices.
  - Uses separate RX (Receive) and TX (Transmit) pins.
  - Requires matching configuration parameters: Baud rate, data bits, parity, stop bits.
  - Typical baud rates are:
    - 9600, 115200

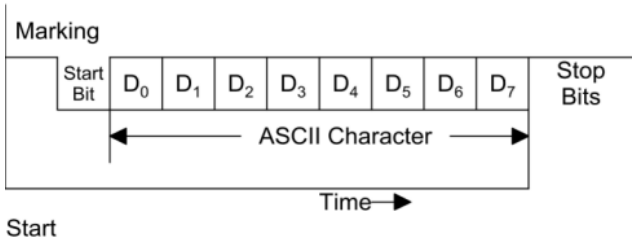


Fig: Asynchronous Serial Transmission Format

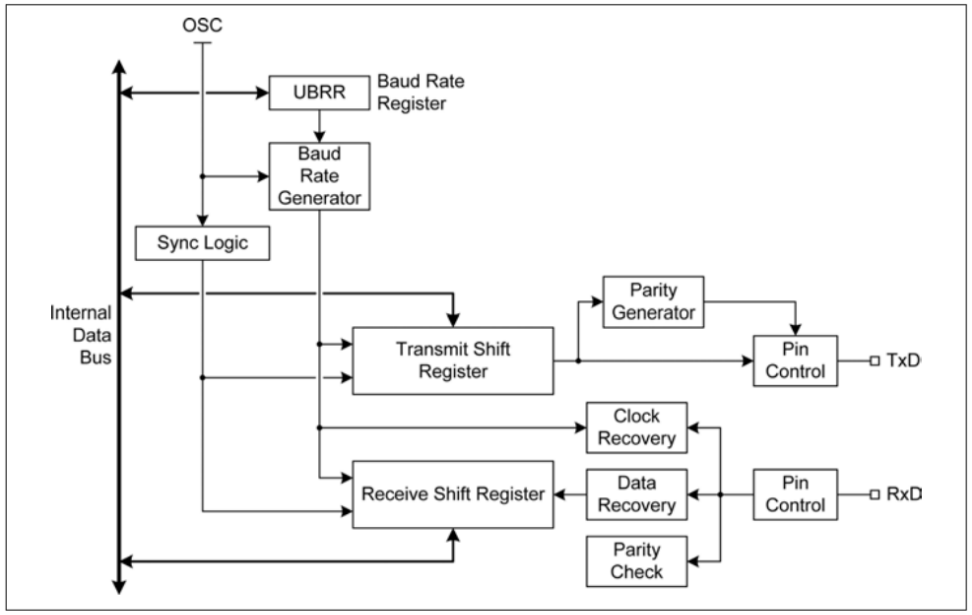


Figure 2-7. AVR USART block diagram

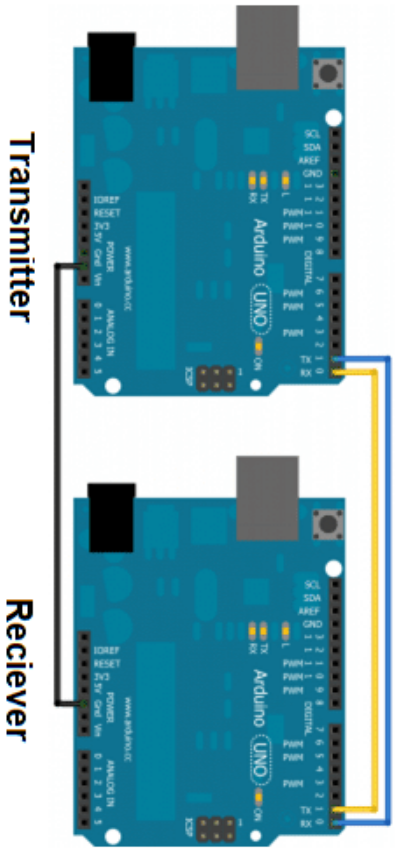
UART Implementation

- ✓ Atmega328P includes a hardware UART module for asynchronous serial communication via PD0 (RX) and PD1 (TX), used in Arduino as Serial.
  - UART is controlled by UDR0 (data), UCSR0A/B/C (control), and UBRR0 (baud rate) registers.
  - PD0 and PD1 are linked to a USB-to-serial converter (e.g., ATmega16U2 on Uno) for PC communication
  - Arduino's Serial.begin(), Serial.print(), and Serial.read() abstract UART register-level operations
  - UART buffer is managed with Serial.available() and Serial.read() for non-blocking input
  - Baud rate must match between microcontroller and peripheral or PC, set via Serial.begin(baud)
  - Other pins can be used as additional serial pins for UART using SoftwareSerial library.

*Explore **Processing Application** to have fun with Arduino.

UART Implementation - Example

- ✓ Connect a sensor/module that supports UART (e.g., GPS, Bluetooth, serial LCD) to:
  - TX of Arduino (Pin 1) → RX of sensor, RX of Arduino (Pin 0) ← TX of sensor Common GND




```
void setup() {
  Serial.begin(9600);              // Initialize UART at 9600 baud
  while (!Serial);                 // Wait for Serial ready (useful for native USB boards)
}

void loop() {
  Serial.write(0xA1);              // Send command/data byte
  Serial.write("OK", 2);           // Send multiple bytes

  if (Serial.available() > 0) {
    int b = Serial.read();         // Read incoming byte
    int next = Serial.peek();      // Look ahead at next byte
    // Use 'b' or 'next' as needed
  }

  Serial.flush();                  // Wait for all outgoing bytes to finish

  delay(1000);                     // Repeat periodically
}
```

Configuring UART

- ✓ Arduino's Serial.begin() uses default settings:
  - 8 data bits, no parity, 1 stop bit (8N1).
  - To customize it, you need to access the USART registers directly.

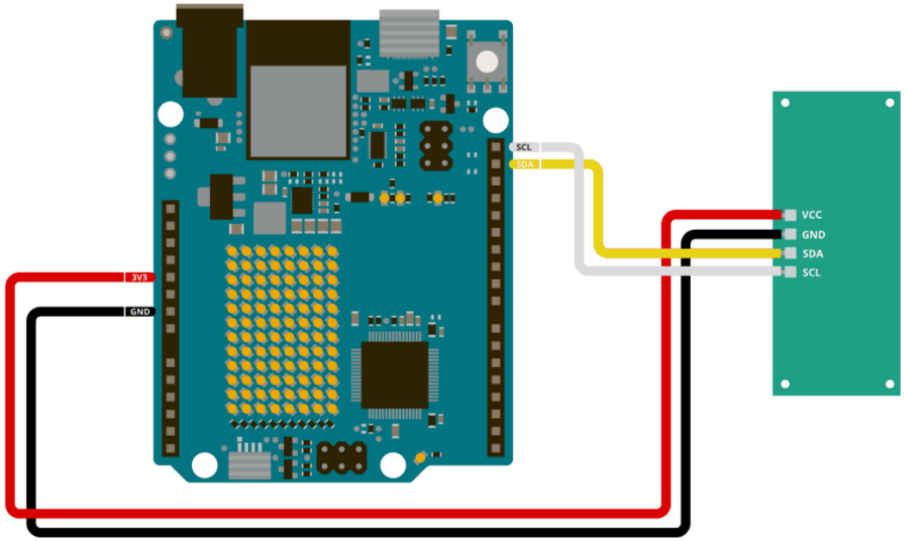
```
void setup() {
  // Set baud rate
  uint16_t ubrr = 103;              // 9600 baud for 16MHz clock
  UBRR0H = (ubrr >> 8);
  UBRR0L = ubrr;

  // Set frame format: 7 data bits, even parity, 2 stop bits
  UCSR0C = (1 << USBS0) |           // 2 stop bits
           (1 << UPM01) |           // Even parity
           (1 << UCSZ01);           // 7-bit (UCSZ01=1, UCSZ00=0)

  // Enable receiver and transmitter
  UCSR0B = (1 << RXEN0) | (1 << TXEN0);
}
```

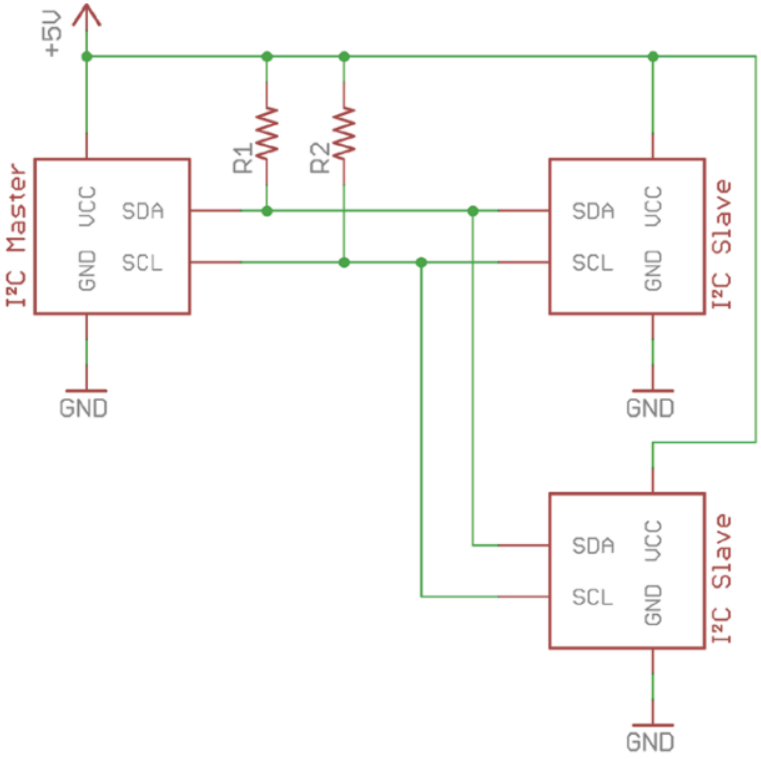
```
void loop() {
  // Transmit a byte
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = 0xA1;

  // Receive a byte
  if (UCSR0A & (1 << RXC0)) {
    uint8_t data = UDR0;
    // Process data
  }

  delay(1000);
}
```

Implementation of Communication Protocols

I2C / TWI

- ✓ Two-wire serial bus: SDA (Serial Data - Pin A4) and SCL (Serial Clock – Pin A5).
- ✓ Supports multi-master and multi-slave configurations on the same bus.
- ✓ Devices are identified by unique addresses.
- ✓ Requires pull-up resistors on the SDA and SCL lines.





Implementation of Communication Protocols

Commonly used functions in Wire Library

- `begin()` - Initialise the I2C bus
- `end()` - Close the I2C bus
- `requestFrom()` - Request bytes from a peripheral device
- `beginTransmission()` - Begins queueing up a transmission
- `endTransmission()` - Transmit the bytes that have been queued and end the transmission
- `write()` - Writes data from peripheral to controller or vice versa
- `available()` - returns the number of bytes available for retrieval
- `read()` - Reads a byte that was transmitted from a peripheral to a controller.
- `setClock()` - Modify the clock frequency
- `onReceive()` - Register a function to be called when a peripheral receives a transmission
- `onRequest()` - Register a function to be called when a controller requests data
- `setWireTimeout()` - Sets the timeout for transmissions in controller mode
- `clearWireTimeoutFlag()` - Clears the timeout flag
- `getWireTimeoutFlag()` - Checks whether a timeout has occurred since the last time the flag was cleared.

Interfacing TC74 Temp. Sensor using I2C

Write Byte Format

S	Address	WR	ACK	Command	ACK	Data	ACK	P
	7 Bits			8 Bits		8 Bits		

Slave Address

Command Byte: selects which register you are writing to.

Data Byte: data goes into the register set by the command byte.

Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address

Command Byte: selects which register you are reading from.

Slave Address: repeated due to change in data-flow direction.

Data Byte: reads from the register set by the command byte.

Receive Byte Format

S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits		

S = START Condition
P = STOP Condition
Shaded = Slave Transmission

Data Byte: reads data from the register commanded by the last Read Byte or Write Byte transmission.

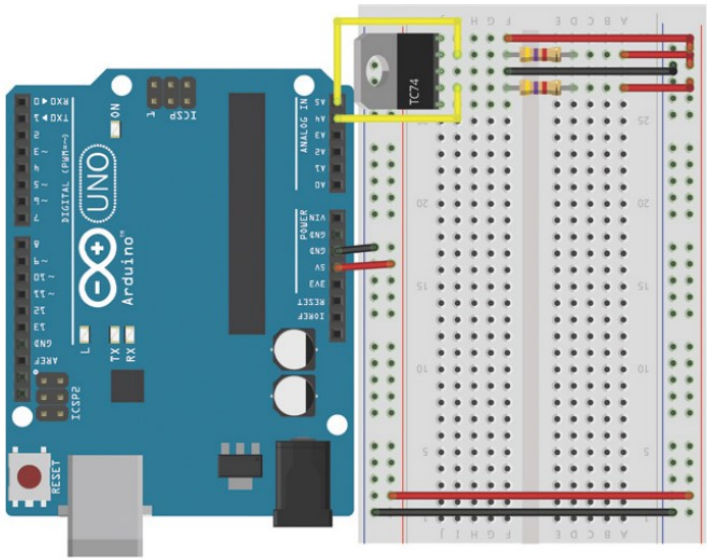


Figure 10-5: TC74 sensor communication scheme

Interfacing TC74 Temp. Sensor using I2C

✓ Steps:

- Send to the device's address in write mode and write a 0 to indicate that you want to read from the data register.
- Send to the device's address in read mode and request 8 bits (1 byte) of information from the device.
- Confirm that all 8 bits (1 byte) of temperature information were received.

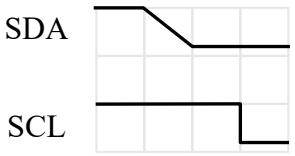
I²C temperature sensor printing code–read_temp.ino

```
//Reads Temp from I2C temperature sensor
//and prints it on the serial port

//Include Wire I2C library
#include <Wire.h>
int temp_address = 72; //1001000 written as decimal number

void setup()
{
  //Start serial communication at 9600 baud
  Serial.begin(9600);

  //Create a Wire object
  Wire.begin();
}
```

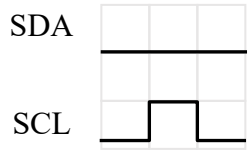
```
void loop()
{
  //Send a request
  //Start talking to the device at the specified address
  Wire.beginTransmission(temp_address);
  //Send a bit asking for register zero, the data register
  Wire.write(0);
  //Complete Transmission
  Wire.endTransmission();

  //Read the temperature from the device
  //Request 1 Byte from the specified address
  int returned_bytes = Wire.requestFrom(temp_address, 1);

  //If no data was returned, then something is wrong.
  if (returned_bytes == 0)
  {
    Serial.println("I2C Error"); //Print an error
    while(1);                   //Halt the program
  }

  // Get the temp and read it into a variable
  int c = Wire.read();

  //Do some math to convert the Celsius to Fahrenheit
  int f = round(c*9.0/5.0 +32.0);

  //Send the temperature in degrees C and F to the serial
  Serial.print(c);
  Serial.print("C ");
  Serial.print(f);
  Serial.println("F");

  delay(500);
}
```

Additional Slide

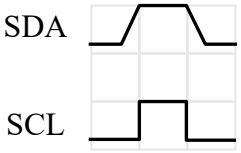
I2C Theory



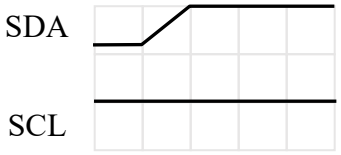
Start condition



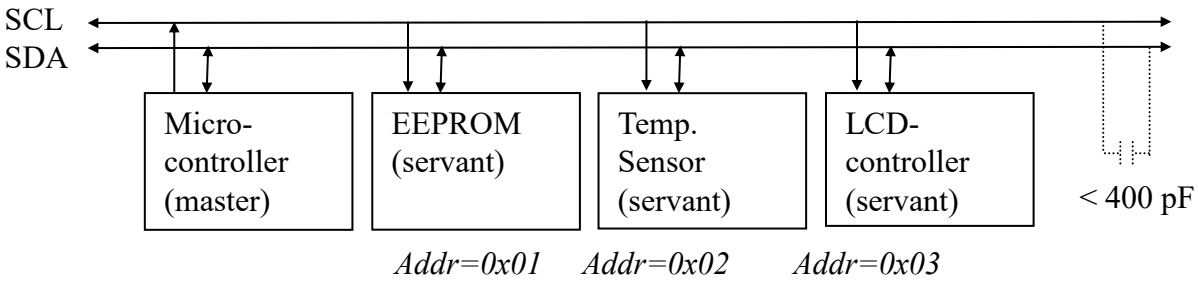
Sending 0

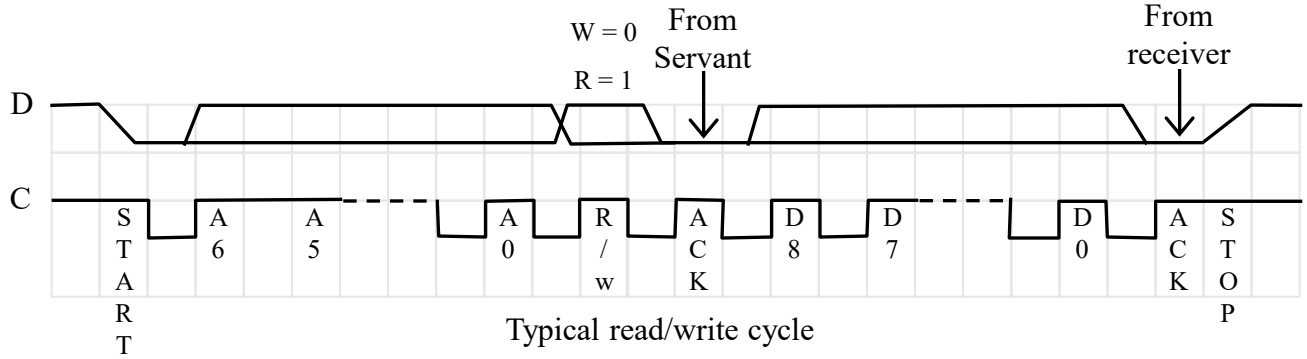


Sending 1

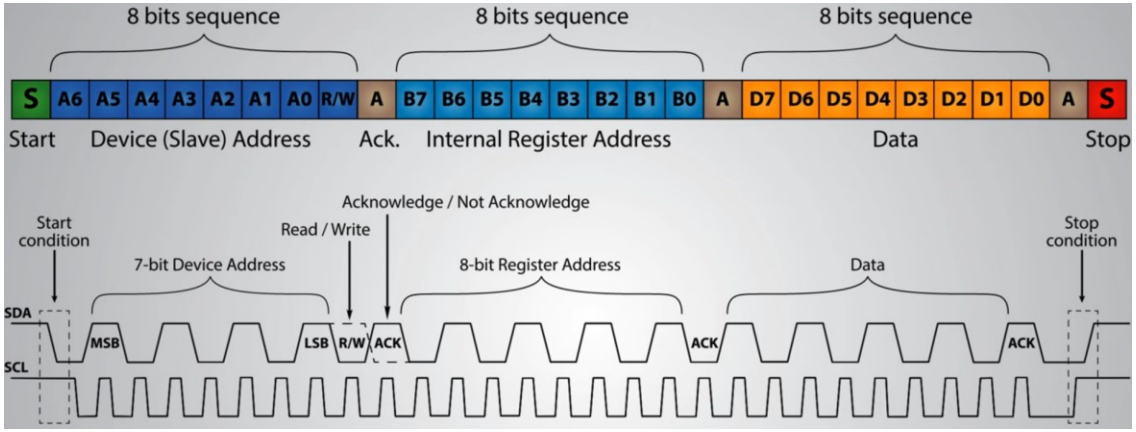


Stop condition





Typical read/write cycle

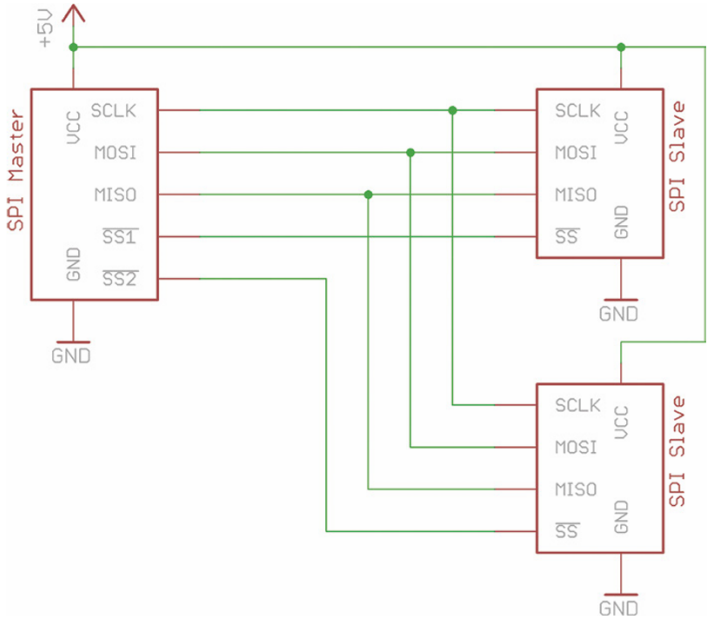


SPI (Serial Peripheral Interface) Bus

- ✓ A four-wire serial bus: MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SCLK (Serial Clock), and SS/CS (Slave Select/Chip Select).
- ✓ Master-slave protocol where the master initiates communication.
  - Synchronous communication controlled by the SCLK signal.

Table 11-2: SPI Communication Lines

SPI Communication Line	Description
MOSI	Used for sending serial data from the master device to a slave device.
MISO	Used for sending serial data from a slave device to the master device.
SCLK	The signal by which the serial data is synchronized with the receiving device, so it knows when to read the input.
SS	A line indicating slave device selection. Pulling it low means you are speaking with that slave device. Generally, only one SS line on a bus should be pulled low at a time.



SPI Library

- ✓ To interface an SPI device, match key parameters using SPISettings(speed, bitOrder, mode) based on the device datasheet.
  - **SPI.begin():** Initializes SPI hardware
  - **SPI.beginTransaction(SPISettings):** Applies device-specific settings
  - **SPI.transfer():** Sends and receives data
  - **SPI.endTransaction():** Ends safe access to SPI bus
  - **SPI.end():** Disables SPI
  - **setBitOrder(), setClockDivider(), setDataMode():** Legacy functions for manual configuration
  - **usingInterrupt():** Prevents conflict with interrupts during SPI transactions
  - **Flow:** begin() → beginTransaction() → pull CS LOW → transfer() → pull CS HIGH → endTransaction() → end() (if needed).

SPI Interface – Basic Steps

- ✓ The basic process for communicating with an SPI device is as follows:
  - Set the SS pin low for the device you want to communicate with.
  - Toggle the clock line up and down at a speed less than or equal to the transmission speed supported by the slave device.
  - For each clock cycle, send 1 bit on the MOSI line, and receive 1 bit on the MISO line.
  - Continue until transmitting or receiving is complete, and stop toggling the clock line.
  - Return the SS pin to high state.

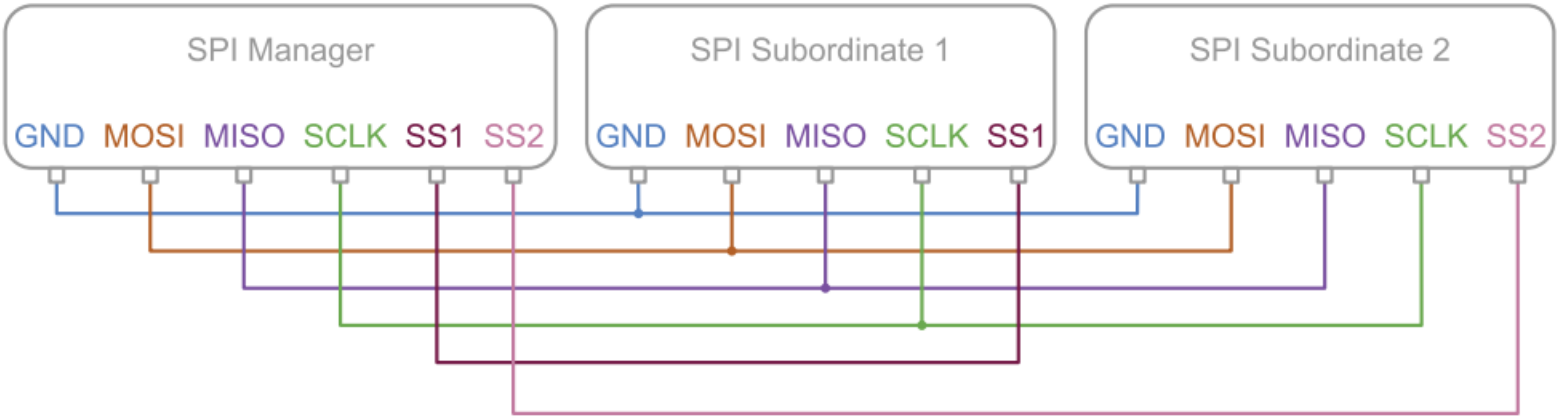


Figure 6.32 Example of a typical SPI bus connection between many devices.

Accelerometer interfacing using SPI

Table 11-4: Arduino Uno SPI pins

Arduino Uno pin	SPI Function
10	Chip Select (CS) / Slave Select (SS)
11	Master Out/Slave In (MOSI)
12	Master In/Slave Out (MISO)
13	Serial Clock (SCLK)

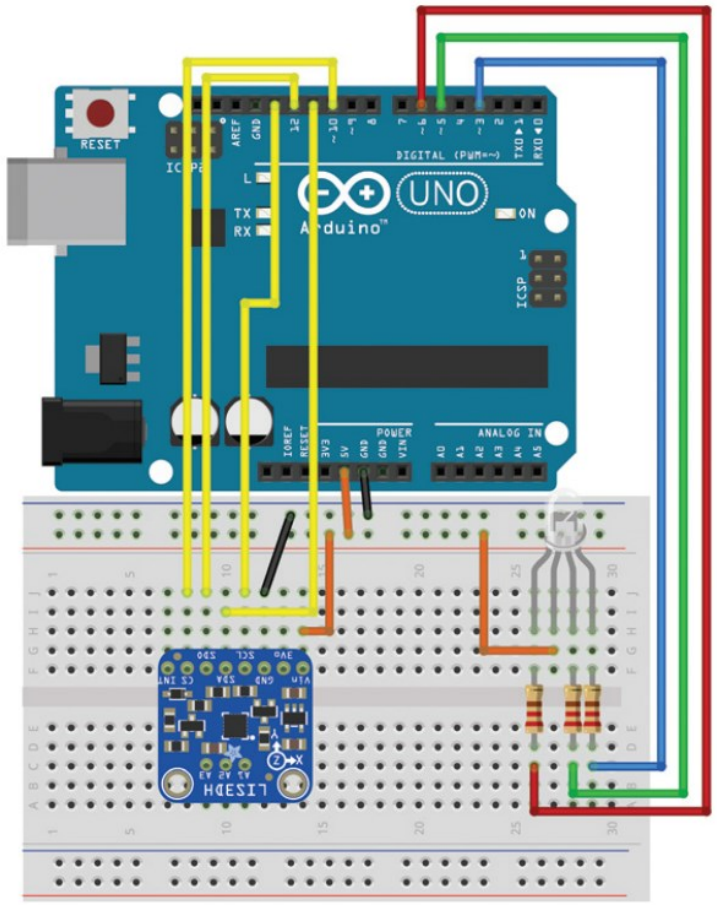


Figure 11-6: Accelerometer breakout and RGB LED wired to Arduino Uno

Accelerometer interfacing using SPI

Accelerometer-based orientation sensor—orientation.ino

```
// Uses the Z-Axis of an Accelerometer to detect orientation

// Include Libraries
// This library will, itself, include the SPI and Universal Sensor Libraries
#include <Adafruit_LIS3DH.h>

// Define the pins (the SPI hardware pins are used by default)
const int RED_PIN = 6;
const int GREEN_PIN = 5;
const int CS_PIN = 10;

// Set up the accelerometer using the hardware SPI interface
Adafruit_LIS3DH accel = Adafruit_LIS3DH(CS_PIN);

void setup()
{
  Serial.begin(9600); // Set up the serial port so we can see readings

  // Connect to the accelerometer
  if (!accel.begin())
  {
    Serial.println("Could not find accelerometer.");
    while (1); // Loop forever
  }

  // Set the sensitivity of the accelerometer to +/-2G
  accel.setRange(LIS3DH_RANGE_2_G);


  // Set the LED cathode pins as outputs and turn them off
  // HIGH is off because this is a common anode LED
  pinMode(RED_PIN, OUTPUT);
  digitalWrite(RED_PIN, HIGH);
  pinMode(GREEN_PIN, OUTPUT);
  digitalWrite(GREEN_PIN, HIGH);
}

void loop()
{
  // Get X, Y, and Z accelerations
  accel.read();

  // Print the Raw Z Reading
  Serial.print("Raw: ");
  Serial.print(accel.z);

  // Map the Raw Z Reading G's based on +/-2G Range
  Serial.print("\tActual: ");
  Serial.print((float(accel.z)/32768.0)*2.0);
  Serial.println("G");

  // Check if we are upside-down
  if (accel.z < 0)
  {
    digitalWrite(RED_PIN, LOW);
    digitalWrite(GREEN_PIN, HIGH);
  }
  else
  {
    digitalWrite(RED_PIN, HIGH);
    digitalWrite(GREEN_PIN, LOW);
  }

  // Get new data every 100ms
  delay(100);
}
```

Comparison

Table 11-3: SPI, I²C, and UART Comparison

SPI	I²C	UART
Can operate at the highest speeds.	Maximum speed is highly dependent upon physical characteristics of the bus (length, number of devices, pull-up strength, and so on).	Requires baud rate to be agreed upon by both devices in advance of starting communication.
Is generally easier to work with than I²C.	Requires only two communication lines.	Effectively uses zero protocol overhead—very simple to implement.
No pull-up resistors needed.	Can support communication between devices operating from different voltage rails.	No predefined master/slave—it is up to you to define the protocol.
Number of slave devices is limited only by number of available SS pins on master.	Number of slave devices is limited by availability of chips with particular slave addresses.	Cannot easily support multiple slave devices.
Has built-in Arduino hardware and software support.	Has built-in Arduino hardware and software support.	Has built-in Arduino hardware and software support.

Concept – Pooling vs Interrupt

- ✓ Review of the concept of:
  - Pooling vs Interrupt
    - *Multitasking*: Interrupts allow near-simultaneous task handling (e.g., blinking LEDs while responding to input); polling cannot.
  - Non-blocking code
  - Timer and watch-dog timers
  - RTOS firmware

External Hardware Interrupts

- ✓ Triggered by an electrical condition on a specific microcontroller pin (e.g., a rising edge, falling edge, or change in state).
- ✓ Execution of an Interrupt Service Routine (ISR) when the interrupt is triggered.
- ✓ On the ATmega328P-based Arduino Uno, digital pins 2 and 3 support external hardware interrupts via INT0 and INT1.
  - However, **pin change interrupts** are available on almost all digital and analog pins.
  - Still, for most Arduino functions like **attachInterrupt()**, only pins 2 and 3 are typically used. These interrupts are triggered by a change in signal level or edge (rising/falling) and used to respond immediately to events without polling.

External Hardware Interrupts (Cont..)

- ✓ attachInterrupt() links a digital pin to an interrupt service routine (ISR) that runs automatically on a specific signal change.
  - Syntax: attachInterrupt(digitalPinToInterrupt(pin), ISR_function, mode)
  - digitalPinToInterrupt(pin) converts a pin number to its interrupt number.
- ✓ ISR_function is the function to run when the interrupt triggers.
- ✓ mode specifies the trigger condition:
  - RISING: LOW to HIGH transition
  - FALLING: HIGH to LOW transition
  - CHANGE: any change in state
  - LOW or HIGH: constant level (used less often)

External Hardware Interrupts (Cont..)

- ✓ Example: Toggle LED on Button Press:

```
volatile bool ledState = false;

void setup() {
  pinMode(2, INPUT_PULLUP);      // Button input
  pinMode(13, OUTPUT);           // LED output
  attachInterrupt(digitalPinToInterrupt(2), toggleLED, FALLING);
}

void loop() {
  digitalWrite(13, ledState);
}

void toggleLED() {
  ledState = !ledState;          // Toggle LED state
}
```

Button presses often "bounce" and trigger multiple interrupts. We use hardware debouncing as software debouncing is challenging in this case. Analyze the scenario!

Timer Interrupts

- ✓ Triggered by internal microcontroller timers when a specified time interval has elapsed or a counter reaches a certain value.
- ✓ Timers are used for generating periodic events.
- ✓ The ATmega328P has three timers:
  - Timer0 – 8-bit (used by millis() and delay())
  - Timer1 – 16-bit (suitable for custom interrupts)
  - Timer2 – 8-bit
- ✓ Timers use registers like:
  - TCCR1A, TCCR1B: control
  - TCNT1: counter value
  - OCR1A: compare match value
  - TIMSK1: enable timer interrupt
- ✓ Use prescalers (e.g., 8, 64, 256, 1024) to slow down the counting rate.

Timer Interrupts

Simple timer interrupt blink test-timer1.ino

```
//Using Timer Interrupts with the Arduino
#include <TimerOne.h>
const int LED=13;

void setup()
{
  pinMode(LED, OUTPUT);
  Timer1.initialize(1000000); //Set a timer of length 1000000 microseconds
  Timer1.attachInterrupt(blinky); //Runs "blinky" on each timer interrupt
}

void loop()
{
  //Put any other code here.
}

//Timer interrupt function
void blinky()
{
  digitalWrite(LED, !digitalRead(LED)); //Toggle LED State
}
```

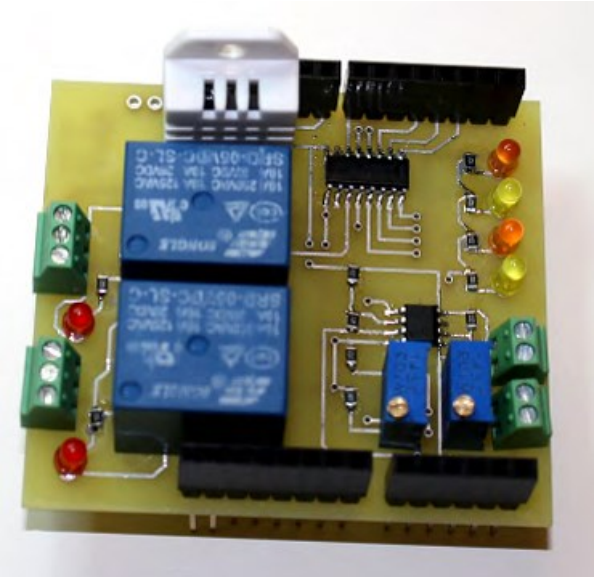
Explore millis() function in Arduino for non-blocking programming logic.

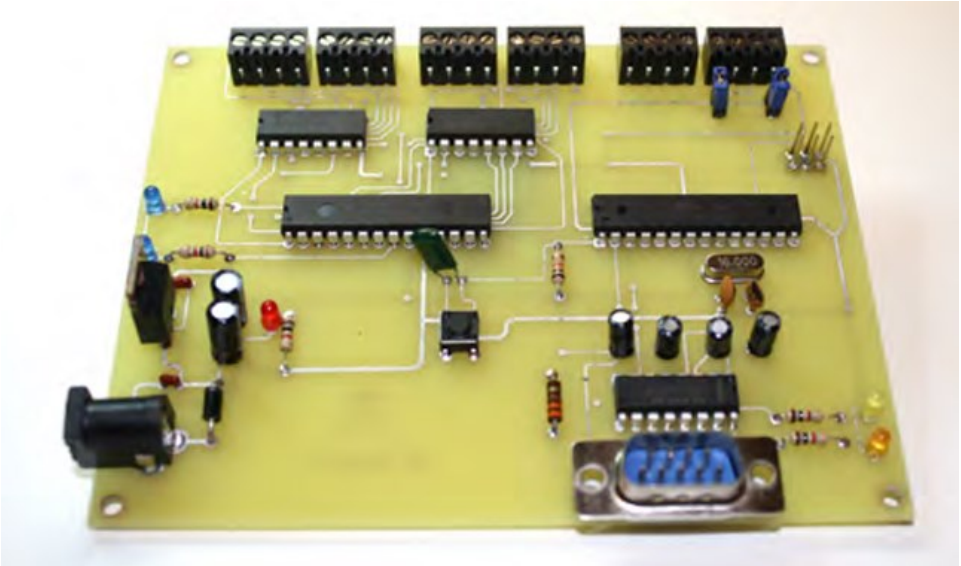
Design – Project Work

- ✓ Design a system containing at least a sensor and an actuator using ATmega328 or STM32 microcontroller.
- ✓ Explore the following boards/modules related to instrumentation system design:
  - ESP32 variants – for inbuilt Wi-Fi module for IoT projects
  - Bluetooth Modules
  - SD Card Module for Data Logging System
  - Other wireless and RF modules (Zigbee, LoRa, Sigfox, etc.)
  - LCD and OLED Display Modules
  - Joysticks and Touch Screens
  - Other instrumentation shields (Data loggers, data acquisition systems)

Creating Custom Components

- ✓ Define → Plan → Design → Prototype → Test → Fabricate → Acceptance test





References

- ✓ D. V. Hall, "Microprocessor and Interfacing Programming and Hardware' Tata McCraw Hill
- ✓ Ramesh S. Gaonkar, "Microprocessor Architecture, Programming and Application with 8085", Prentice Hall
- ✓ K. R. Fowler, Electronic Instrument Design- architecting for the life cycle, Oxford University Press, Inc., 2010.
- ✓ Hughes, J. M. (2016). Arduino: A technical reference: A handbook for technicians, engineers, and makers (1st ed.). O'Reilly Media.
- ✓ Blum, J. (2020). Exploring Arduino: Tools and techniques for engineering wizardry (2nd ed.). John Wiley & Sons.
- ✓ Lutenberg, A., Gomez, P., & Pernia, E. (2018). A beginner's guide to designing embedded system applications on Arm Cortex-M microcontrollers. ARM Education Media.