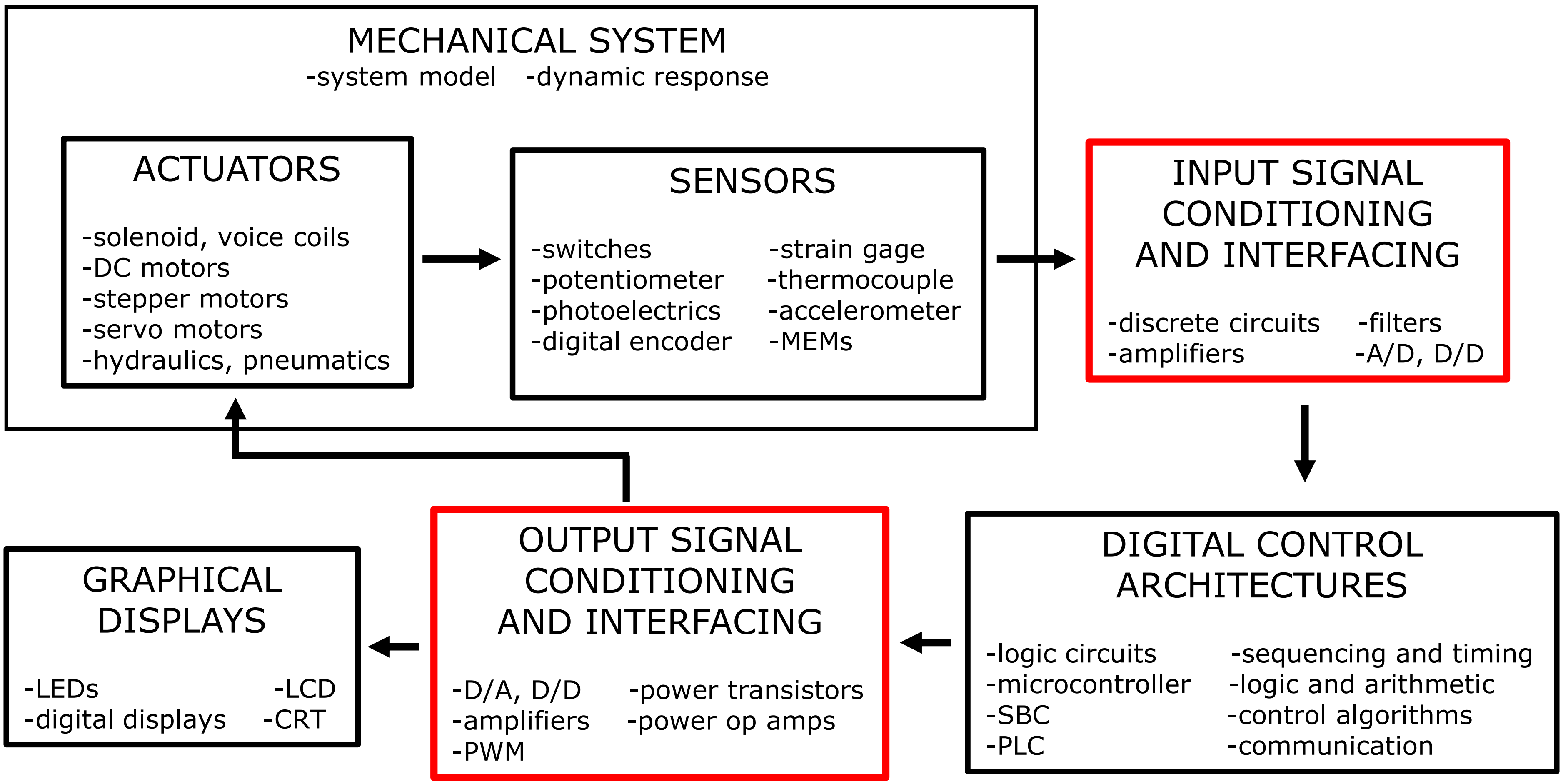
MA2012 INTRODUCTION TO MECHATRONICS SYSTEMS DESIGN

Lecture 5

Prof Ang Wei Tech

College of Engineering
School of Mechanical and Aerospace Engineering

MECHATRONIC SYSTEM COMPONENTS



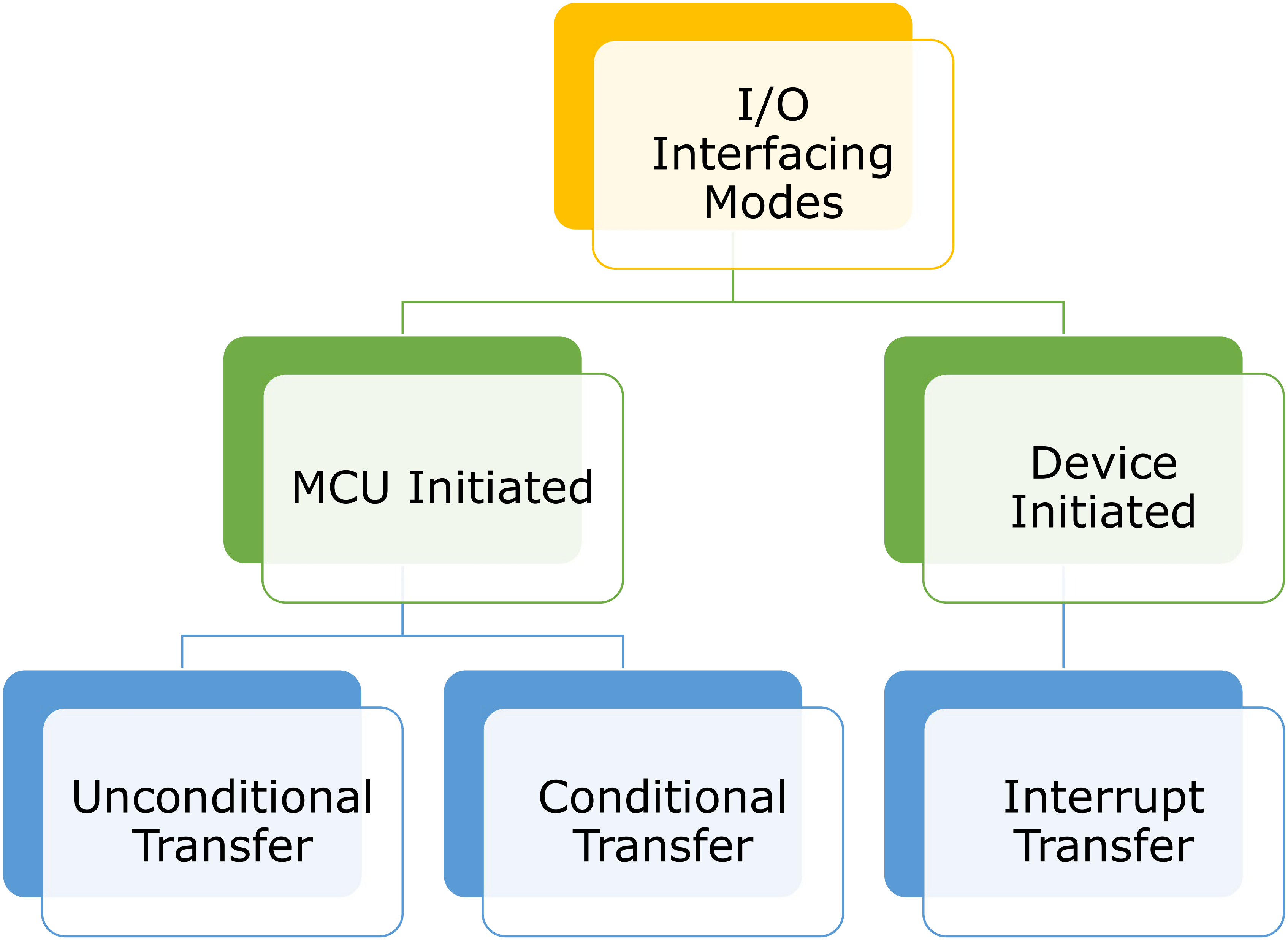
PROCESS/ INSTRUMENT CONTROL I/O

- MCU operates automatically and continuously under the control of the program stored in ROM, no human intervention
  - Operation is changed only by changing the content of the ROM
- During execution, MCU receives data from devices monitoring some physical states (temperature, speed, etc.), operates on the data and sends data or control signals to the process/ instrument via output devices
- E.g. Traffic red-light camera, car air-bag system, fire/ burglar alarm system, etc.

KEYBOARD ENTRY/DISPLAY I/O

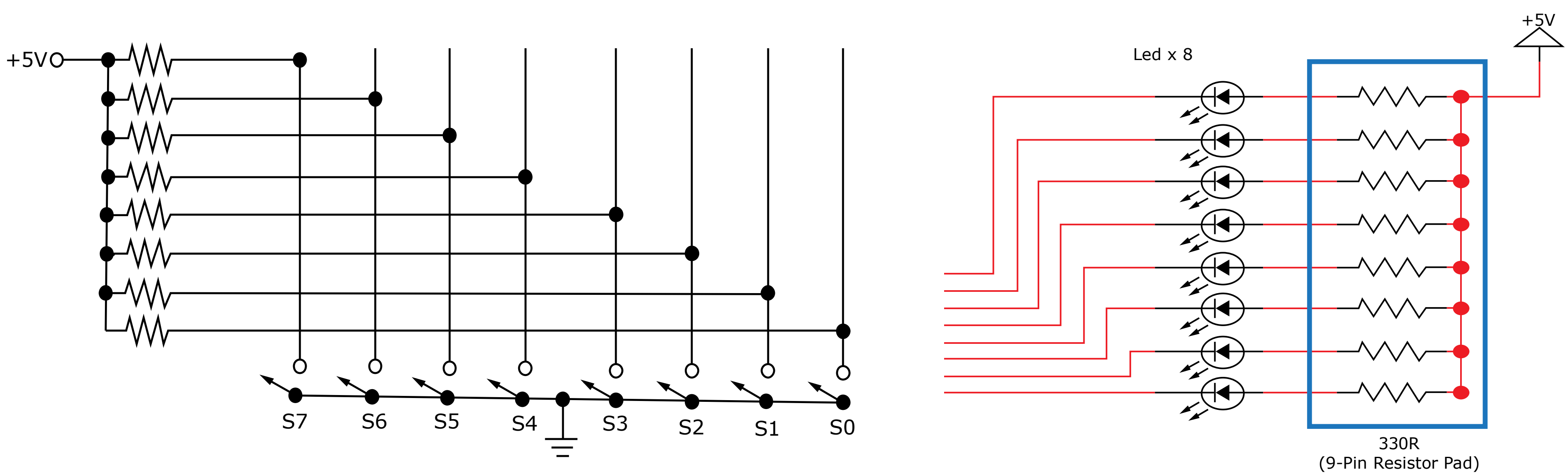
- Communications with human operators
- MCU executes a keyboard monitoring program stored in ROM
  - Reads the keyboard continuously until a key is actuated, determines the actuated key, and executes the appropriate instructions
- Once the instructions are executed, it gets back to keyboard monitoring program
- E.g. Point-of-sale machine, lift, television, etc.

INPUT / OUTPUT INTERFACING



MCU-INITIATED TRANSFER

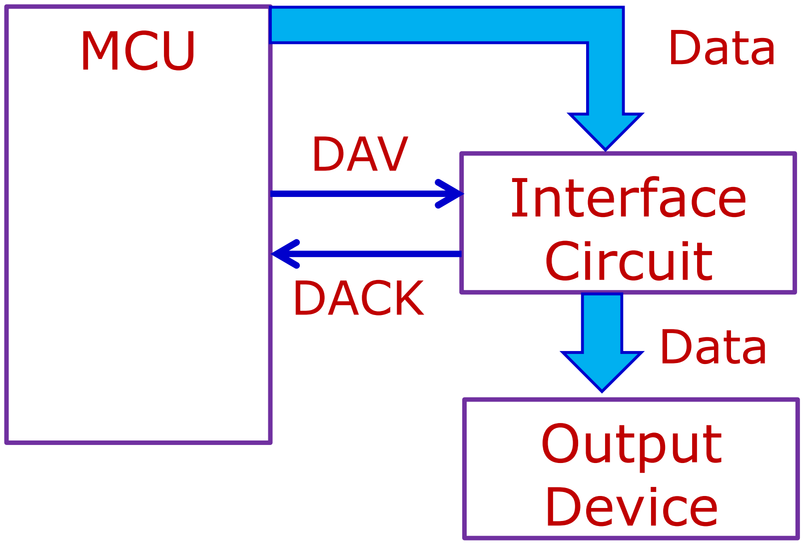
- Unconditional transfer
  - I/O device must always be ready for communication
  - Examples
    - To input a 8-bit data word from a set of 8 switches
    - Output data to LEDs



MCU-initiated transfer

MCU-INITIATED TRANSFER

- Conditional transfer
  - Communication takes place only when the I/O device is ready
  - Handshaking
    - MCU send a data available (DAV) control signal to an output device

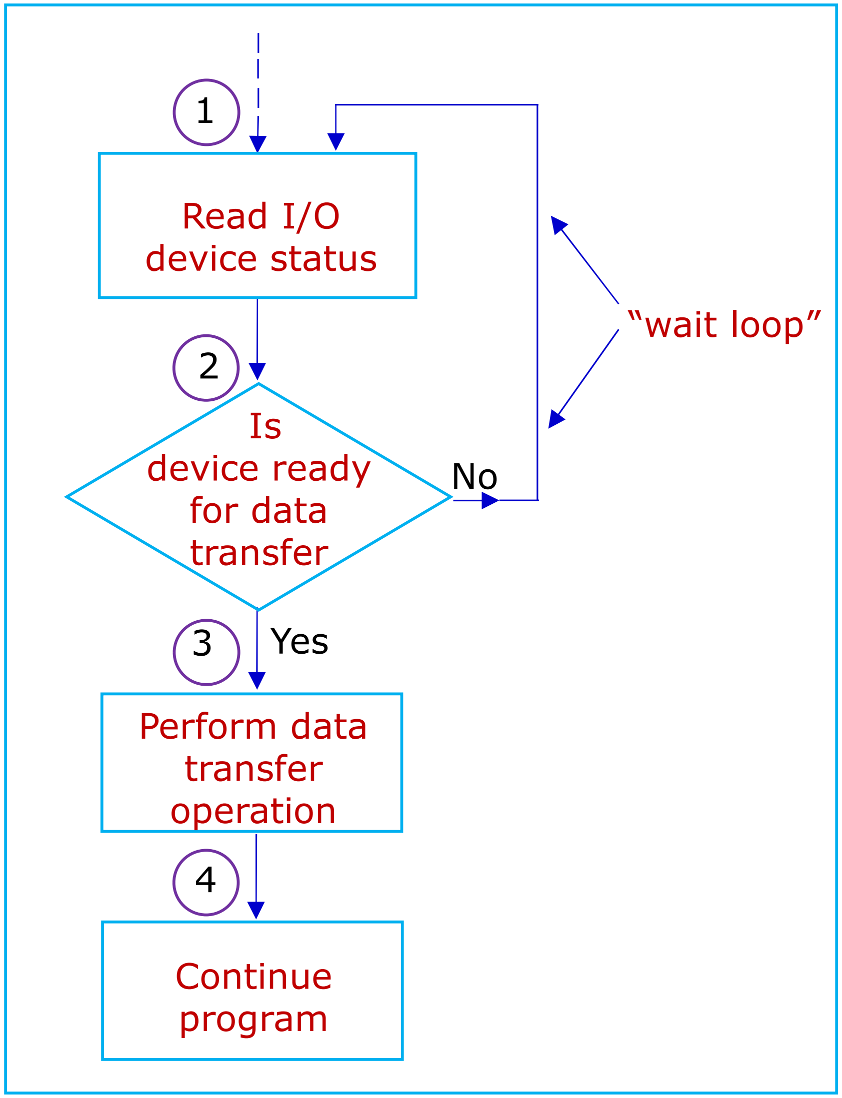


MCU-initiated transfer - handshaking

    - Upon receiving the DAV signal, the output device accepts the data, then send a data accepted (DACK) control signal back to the MCU

CONDITIONAL (POLLED) I/O TRANSFER

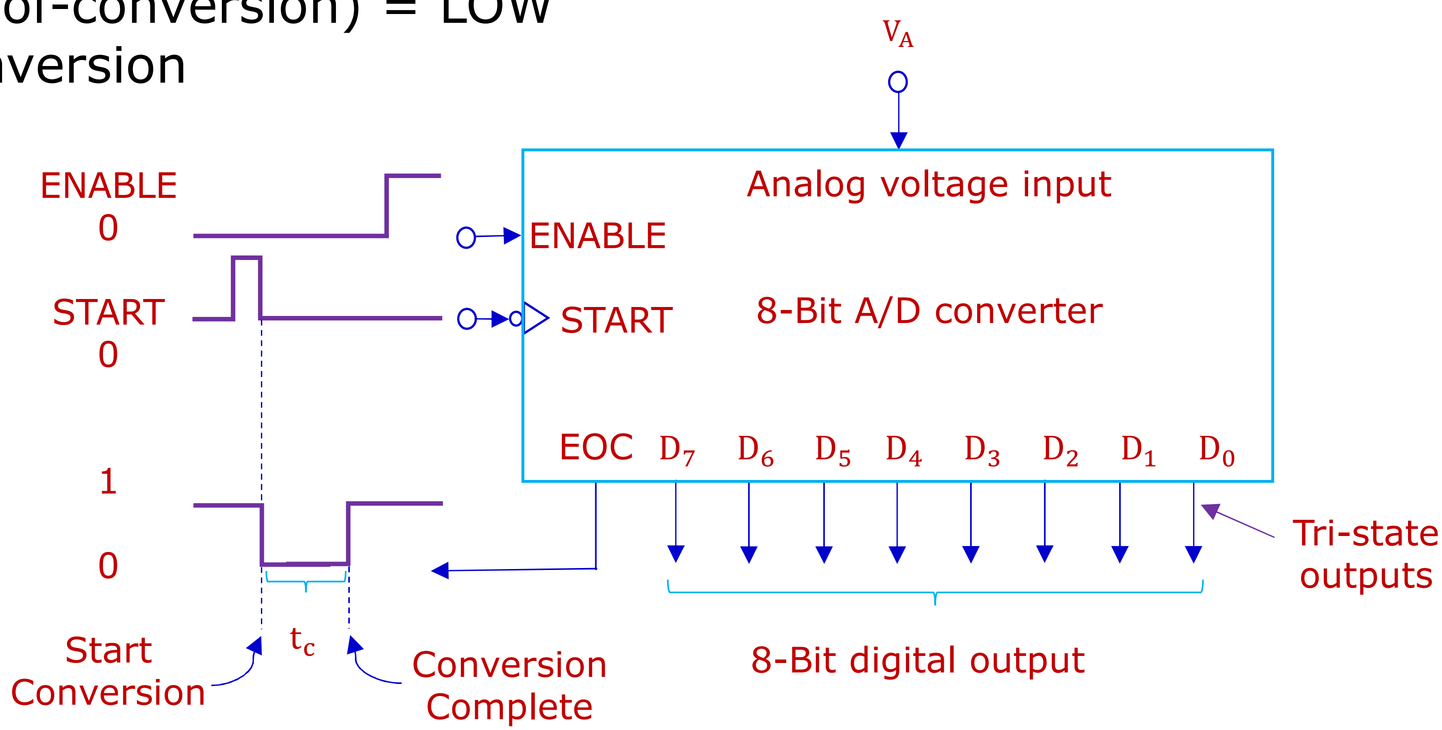
- MCU must read status information from the I/O device (1)
- Test this status to see if the device is ready for data transfer (2)
  - Remain in the wait loop until device is ready
- Perform the data transfer (3)
- Handshaking is needed



Conditional (polled) I/O transfer

CONDITIONAL (POLLED) I/O TRANSFER – E.G.

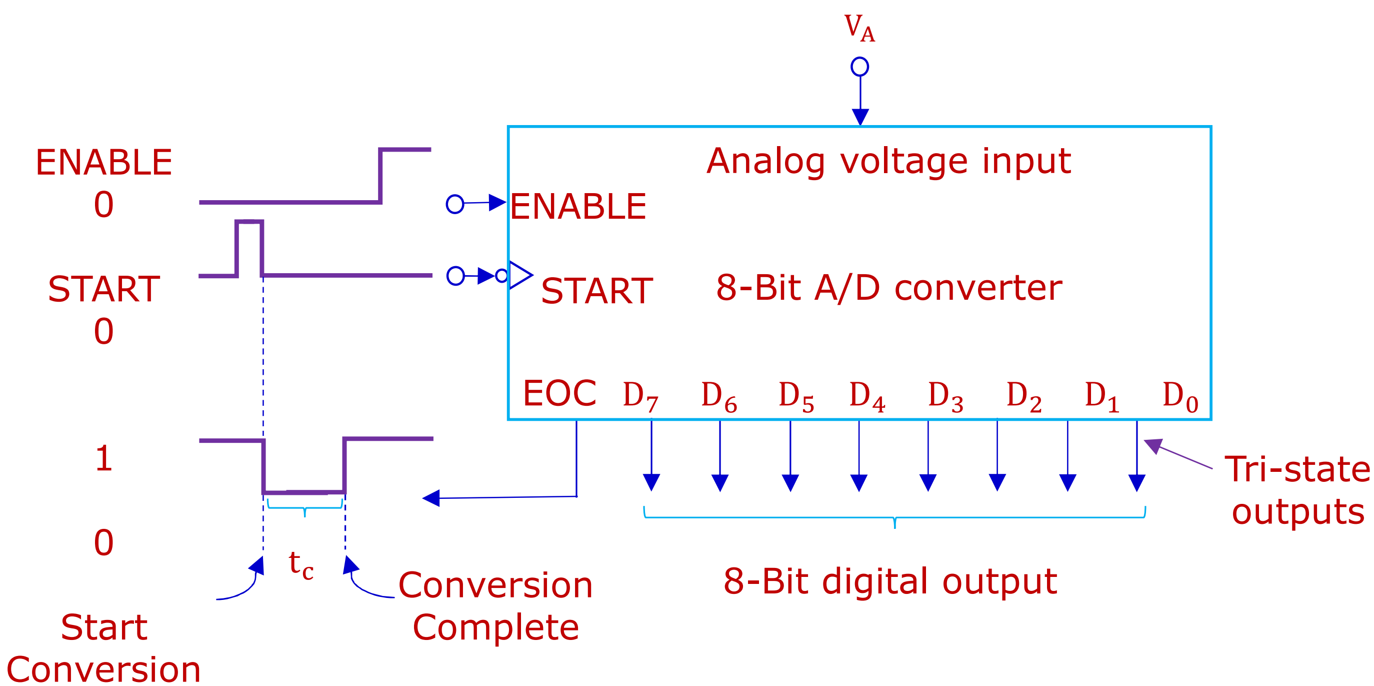
- Type equation here. Converts analog voltage input V_A to an 8-bit output (D_7-D_0)
- Conversion process is initiated by a pulse to START
- Conversion time, t_c can be up to 100μs
- EOC (end-of-conversion) = LOW during conversion
- EOC = HIGH when conversion is completed
- ENABLE = HIGH, make the latched binary output available
- ENABLE = LOW, output at High-Z state (disconnected)



Conditional (polled) I/O transfer

CONDITIONAL (POLLED) I/O TRANSFER – E.G.

- Communications between MCU and ADC
  - MCU issues a START pulse to the ADC to convert V_A to its digital equivalent
  - MCU polls the status of EOC output until conversion is completed
  - MCU reads the ADC output into one of its internal registers



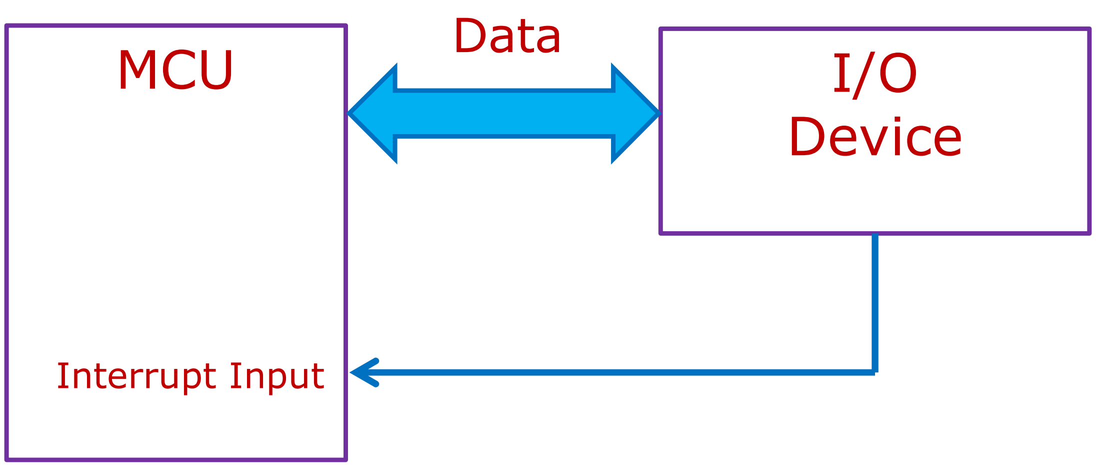
Conditional (polled) I/O transfer

MCU INITIATED – CONDITIONAL (POLLED) I/O TRANSFER

- Data acquisition subroutine
  - May be accessed at any point in the user's program
- Disadvantages
  - Need to wait for I/O devices to be ready
  - MPU can do other things while waiting, especially when I/O devices are slow

DEVICE-INITIATED TRANSFER

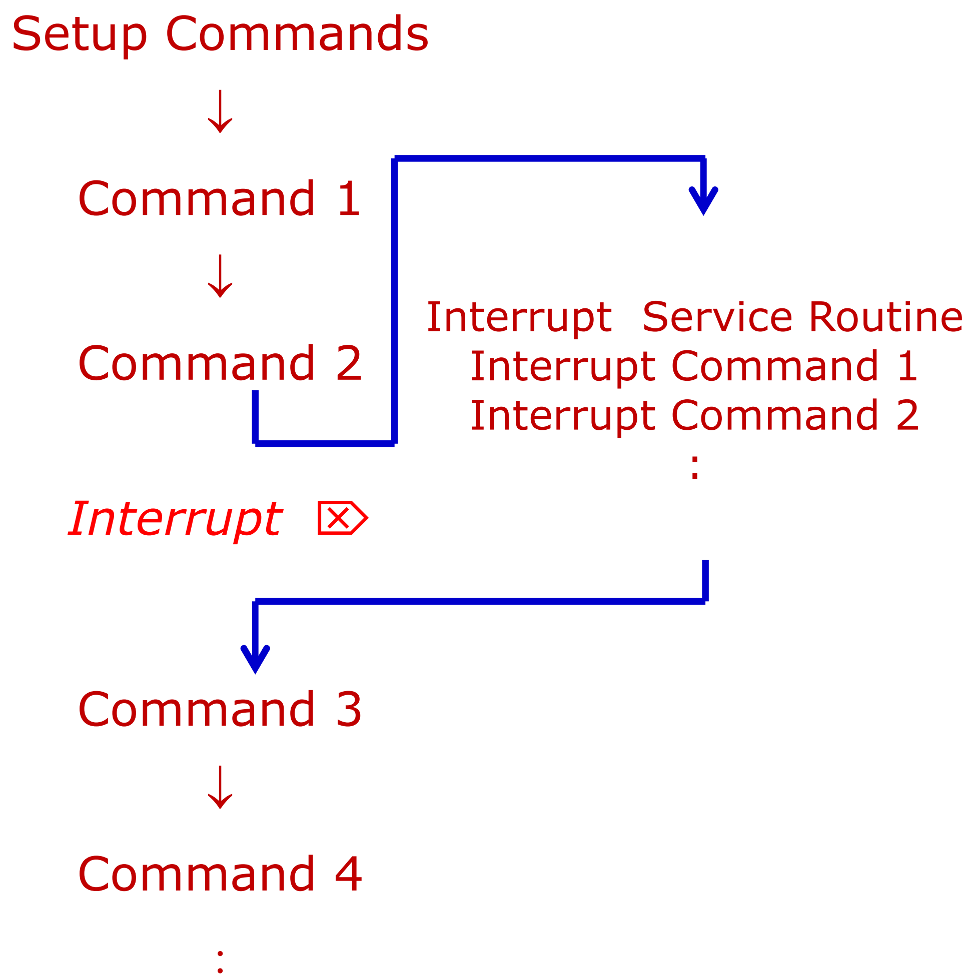
- Interrupt transfer
  - Handshaking is required
  - I/O device sends a signal to an interrupt input to inform the MCU it is ready for data transfer
  - Hardware interrupts are triggered by a state (HIGH/ LOW) or a change in state (HIGH $\rightarrow$ LOW or LOW $\rightarrow$ HIGH)



Interrupt transfer

INTERRUPT TRANSFER

- When an interrupt occurs, all the important registers content which define the current state of the MCU are immediately stored away in a dedicated memory location, before going to the ISR
- Interrupt Service Routine (ISR)
  - Contains commands for transferring to and from the interrupting I/O devices
- Upon returning from ISR, MCU returns to the previous state by restoring the contents of the important registers



Interrupt transfer

ARDUINO'S HARDWARE INTERRUPTS

- UNO – INT 0 (Pin 2) & INT 1 (Pin 3)
- attachInterrupt(interrupt, ISR, mode)
  - interrupt = 0 (Pin 2) or 1 (Pin 3)
  - ISR = interrupt service routine must take no parameters and return nothing
  - Mode:
    - LOW = triggers interrupt when pin is LOW
    - CHANGE = triggers interrupt when pin changes state
    - RISING = triggers interrupt when pin goes LOW→HIGH
    - FALLING = triggers interrupt when pin goes HIGH→LOW

INTERRUPT SERVICE ROUTINE IN ARDUINO

- Cannot have parameters and returns nothing
- Only 1 ISR can run at any one time, other ISRs (if any) will be turned off until the current one is executed
- Functions which rely on timer interrupts will not work while ISR is running, e.g. delay(), millis()
- Global variables are used to pass parameters between main program and ISR, i.e. declare them as *volatile*

- Example:

```
int pin = 1;
volatile int state = LOW;

void setup(){
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE
);
}

void loop(){
  digitalWrite(pin, state);
}

void blink(){
  state = !state;
}
```

POLLING VS INTERRUPTING

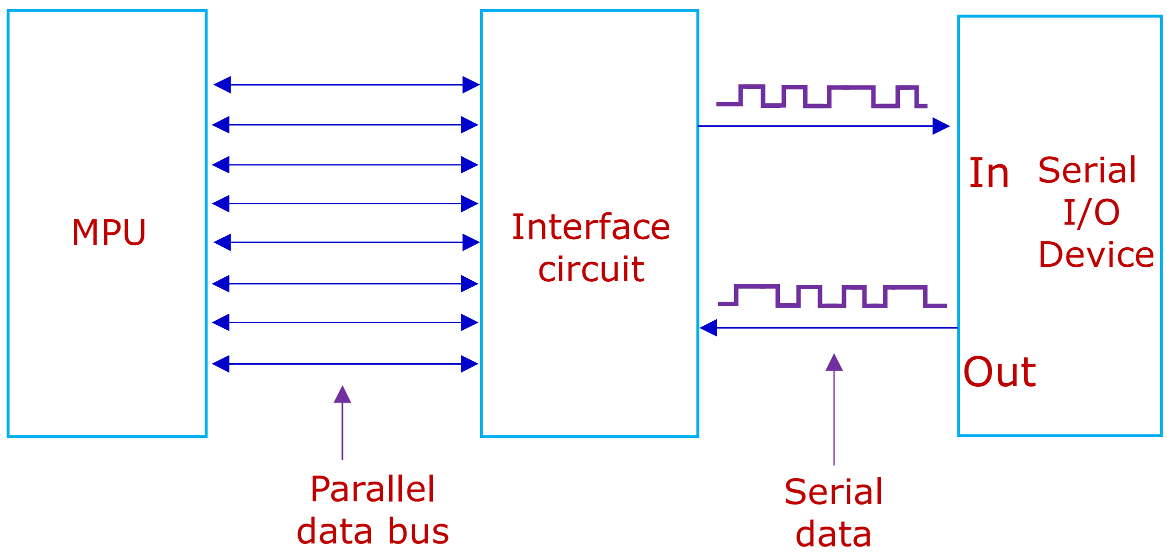
- Advantages of Polling
  - Ease of software implementation
- Advantages of Interrupting
  - Multi-tasking – the MCU can process other commands while waiting for an I/O device to be ready
  - Acquisition accuracy – for fast acquisition tasks
    - E.g. Reading from an encoder on a fast rotating motor shaft. These pulses are too short for polling method to capture, resulting in missing pulses

COMMUNICATIONS

- Parallel data communications
  - Multiple bits of data are transmitted all at one time
  - One data line/ pin per bit is needed
  - Advantage
    - Faster data transfer rate
- Serial data communications
  - Data is transmitted one bit after another
  - Only one data line/ pin is needed
  - Advantages
    - Cheaper to implement, because physical pins/lines are costly
    - Easier to integrate into IC & PCB design, because fewer physical pins/lines result in smaller footprint

PARALLEL-SERIAL INTERFACE

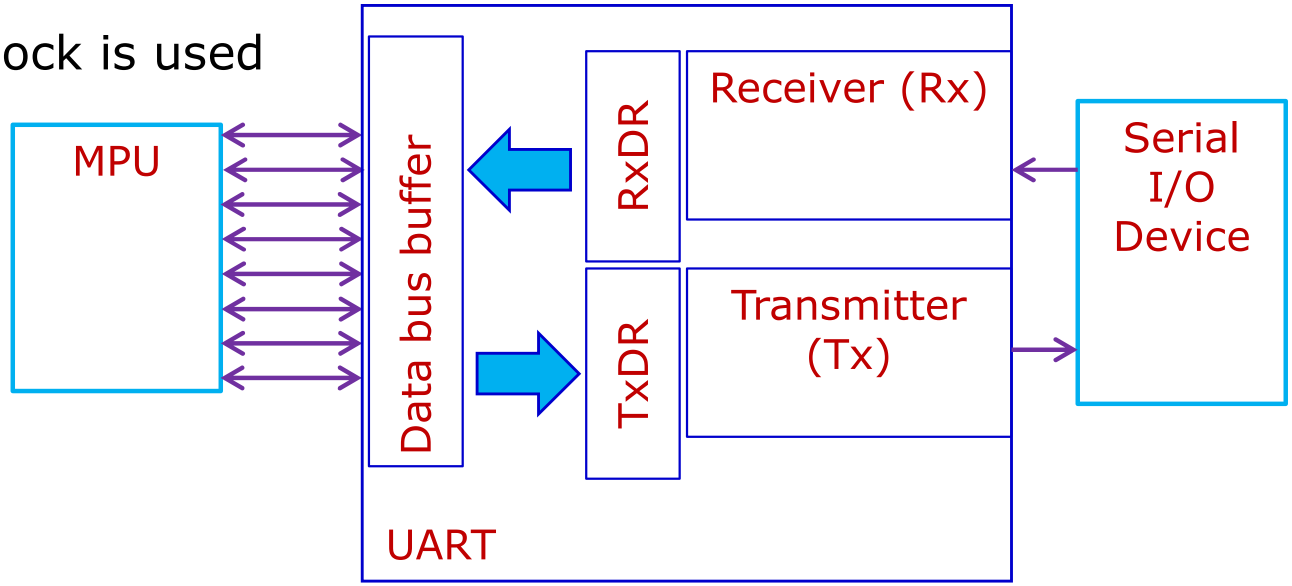
- For serial/parallel conversions for communications between MPU and serial I/O device
  - Converts a *N* bit parallel word from the MPU data bus to a serial data word
  - Converts a serial data signal from a serial device to an *N* bit parallel data word



Parallel-serial interface

UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER (UART)

- A serial receiver (Rx)
  - To convert a serial input to a parallel format, and store it in the receiver data register (RxDR) for eventual transmission to MPU
- A serial transmitter (Tx)
  - To take a parallel word from transmitter data register (TxDR) and convert it to a serial format for transmission
- A bidirectional data bus buffer
  - To pass data from MPU to TxDR, or from RxDR to the MPU over system data bus
- Baud rate generator
  - Sometimes external clock is used



Universal Asynchronous Receiver Transmitter (UART)

DATA TRANSMISSION RATE

- Bit time interval, T_B = period
- Baud rate = Data rate = rate of data transmission = $1/T_B$ (bits/s or Mbps)
- Common baud rates: 19200, 14400, 9600, etc.
- Example: If Data rate = 9600 baud, what is the time duration of 1 bit?
  - Data rate = 9600 bits/s
  - Bit time = 1/9600 = 104.17 μs

SERIAL COMMUNICATION PROTOCOLS

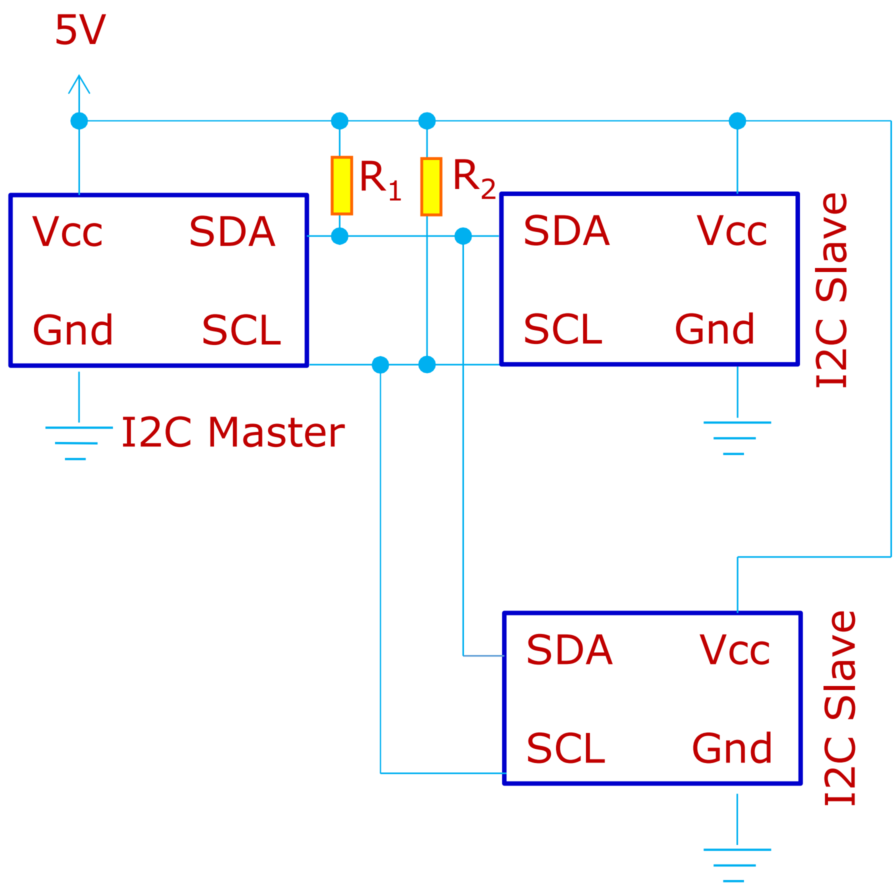
- Arduino supports two serial communication protocols
  - I2C Bus (Asynchronous)
  - SPI Bus (Synchronous)
- Bus
  - Groups of wires used as a common path connecting all the inputs and outputs of several registers/ devices so that data can be easily transferred from any one register/ device to any other using various control signals

SERIAL DATA COMMUNICATION

- Asynchronous Communication
  - Transmitter can send data to receiver at any time
  - Time delay between transmission of two words may be indeterminate
  - Transmitter clock need not synchronise with receiver clock
- Synchronous Communication
  - Transmitting and receiving are synchronised by common clock pulses
  - Transmitter sends data to receiver continually
  - Transmitter sends meaningless data (e.g. sync characters 16_{16} ASCII) continually when there is no data to send

I2C BUS

- I2C bus is controlled by a master device (MCU)
- One or more slave (I/O) devices receive control signal from the master device
- All devices share the same clock signal (SCL) and a bidirectional data line (SDA)
- Only master device can initiate communications between master and slaves to avoid bus contention



I2C Bus

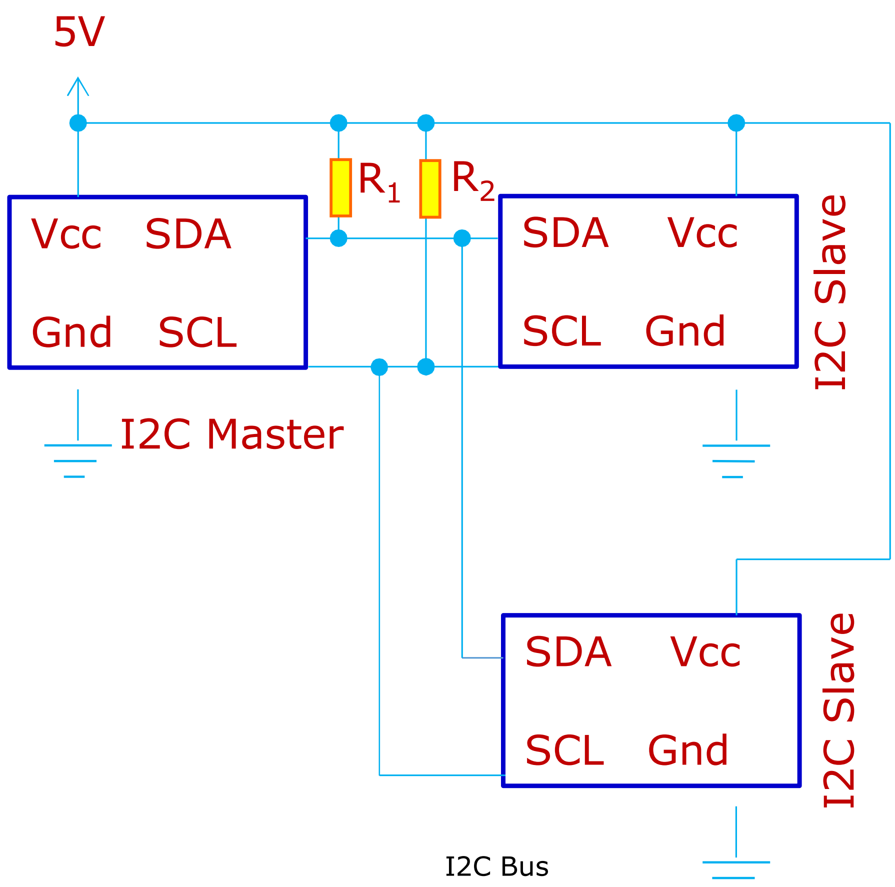
Note: pull-up resistors are needed to maintain HIGH state when all devices are disconnected

I2C BUS

- Each slave device has it own unique 7-bit address or ID number
  - Address may be fixed or selectable (manufacturer dependent)

- When master initiates a communication, a device address is transmitted

- Only the slave device with the correct address shall respond to the master



I2C BUS – COMMUNICATION PROTOCOL

- Steps to communicate with different I2C slave devices need to follow protocol defined by manufacturer in datasheets
- Basic steps:
  1. Master sends a Start bit
  2. Master sends a 7-bit slave address of intended device
  3. Master sends a Read (1) or Write (0) bit depending on application
  4. Slave responds with an “acknowledge”, i.e. ACK bit (0)
  5. In Write mode, master sends 1 byte of information (command or data) at a time, and slave respond with ACKs. In Read mode, master receives 1 byte of information at a time and sends an ACK to the slave after each byte
  6. When communication has been completed, master sends a Stop bit

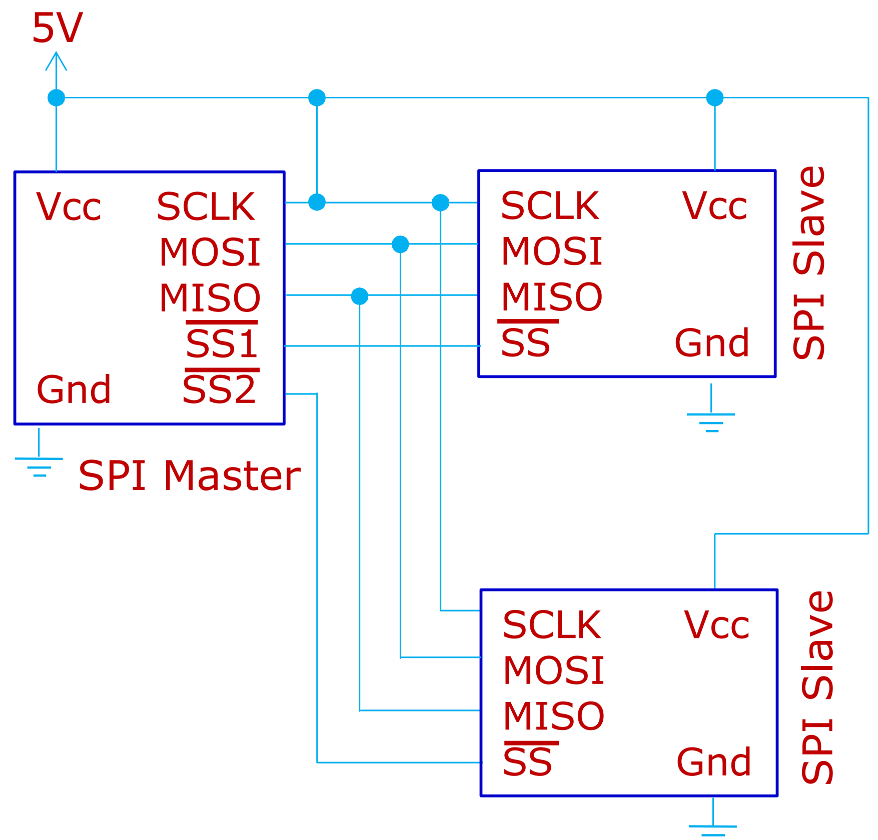
SPI BUS

- 3 pins for communications between master all slaves
  - Shared/Serial Clock (SCLK)
  - Master Out Slave In (MOSI)
  - Master In Slave Out (MISO)

- Each slave device requires an additional slave select (SS) pin

- Total number of I/O pins required = 3 + *n*

  n = number of slave devices

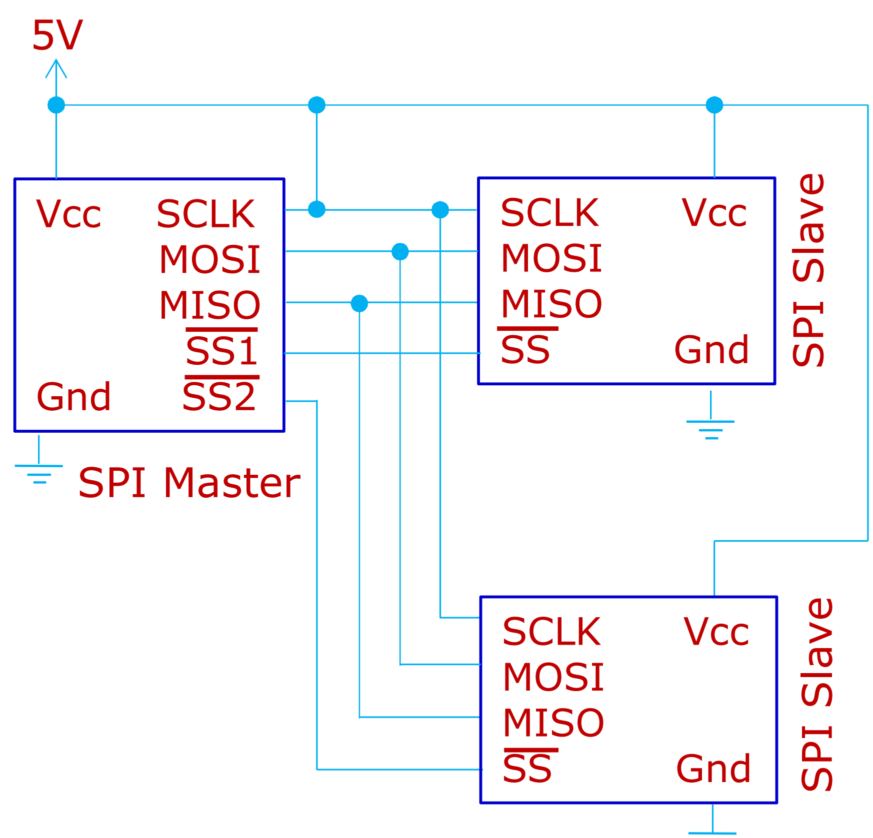


SPI Bus

SPI BUS

- SPI bus is a full-duplex serial communication protocol between master and one or more slaves
  - Full-duplex: Simultaneous bidirectional transmission of information

- All slave devices share MOSI, MISO & SCLK lines, hence all commands from master are sent to each slave device
- Only the slave device with SS pin set LOW shall respond to master

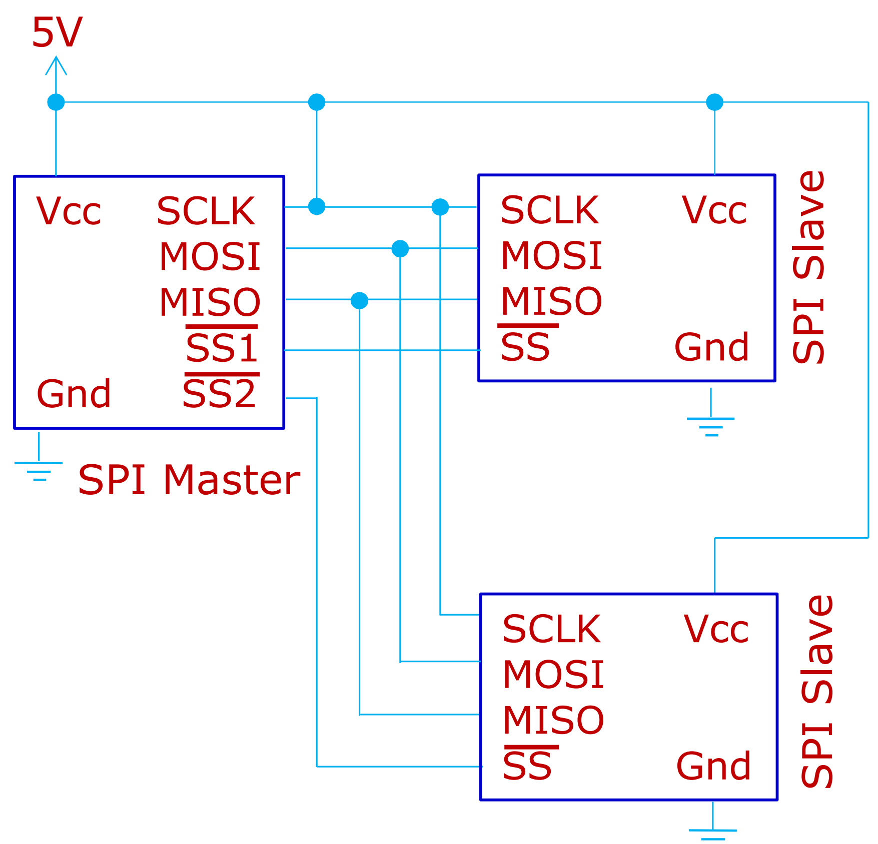


SPI Bus

SPI BUS

- SPI devices are synchronous, i.e. data is transmitted in sync with a SCLK
- 4 modes of communication

Mode	Clock Polarity	Clock Phase (Data capture on …)
0	Low at idle	Rising Edge
1	Low at idle	Falling Edge
2	High at idle	Falling Edge
3	High at idle	Rising Edge



SPI Bus

SPI BUS – COMMUNICATION PROTOCOL

- Basic process:
  1. Set the SS pin LOW for the targeted slave device
  2. Toggle SCLK (square wave) at a speed ≤ transmission speed supported by the slave device
  3. For each clock cycle, master sends 1 bit on MOSI and receives 1 bit on MISO
  4. Continue until data transmission is complete, and stop toggling the clock line
  5. Set SS pin to HIGH

SPI BUS

- Every clock cycle a bit must be sent and received (i.e. synchronous), but that bit may be meaningless
- Naming conventions (manufacturer dependent)
  - Slave Select (SS) ≡ Chip Select (CS)
  - Serial Clock (SCLK) ≡ Clock (CLK)
  - Master Out Slave In (MOSI) ≡ Serial Data In (SDI)
  - Master In Slave Out (MISO) ≡ Serial Data Out (SDO)

I2C VS SPI

I2C Advantages	SPI Advantages
Requires only 2 communication lines	Higher data transmission rate
	Easier to implement
	No pull-up resistors needed

SUMMARY

- Interfacing with I/O devices
  - MCU-Initiated
    - Unconditional transfer, no handshaking
    - Conditional transfer (Polling), handshaking is required
  - Device-Initiated
    - Interrupt transfer, handshaking is required
    - Characteristics of Interrupt Service Routine (ISR)
    - Hardware interrupt in Arduino
  - Polling vs Interrupting

SUMMARY

- Parallel vs Serial Communications
- Universal Asynchronous Receiver Transmitter (UART)
- Data transmission rate
- Synchronous vs Asynchronous communications
- I2C Bus and SPI Bus
  - Characteristics, Communication protocols
  - Advantages