Install and Setup Arduino IDE

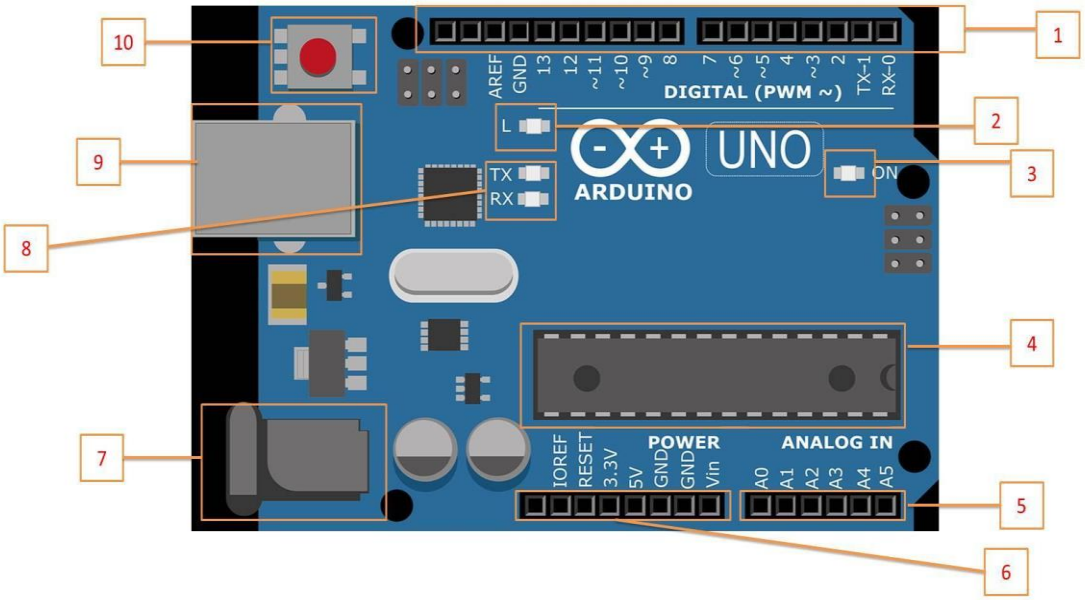
- Download Arduino IDE according to your OS, from the below link:

  https://www.arduino.cc/en/Main/Software

- For Setup, follow the instructions from the below link:

  https://www.arduino.cc/en/Guide/Windows

Arduino Uno Development Board:



1. **Digital pins** Use these pins with digitalRead(), digitalWrite(), and analogWrite(). analogWrite() works only on the pins with the PWM symbol.
2. **Pin 13 LED** The only actuator built-in to your board. Besides being a handy target for your first blink sketch, this LED is very useful for debugging.
3. **Power LED** Indicates that your Arduino is receiving power. Useful for debugging.
4. **ATmega328 microcontroller** The heart of your board.
5. **Analog in** Use these pins with analogRead().

6. **GND and 5V pins** Use these pins to provide +5V power and ground to your circuits.
7. **Power connector** This is how you power your Arduino when it's not plugged into a USB port for power. Can accept between 7-12V.
8. **TX and RX LEDs** These LEDs indicate communication between your Arduino and your computer. Expect them to flicker rapidly during sketch upload as well as during serial communication. Useful for debugging.
9. **USB port** Used for powering your Arduino Uno, uploading your sketches to your Arduino, and for communicating with your Arduino sketch (via Serial. println() etc.).
10. **Reset button** Resets the ATmega microcontroller.

CH-1: Basics

The Arduino program has 2 parts:

Setup(): It is executed once at the start of the program

Loop(): It is executed over and over again.

Goal: Send data on the terminal of Arduino.

```
void setup() {
  begin
  print (“HELLO WORLD from setup”)
}
void loop() {
  print (“HELLO WORLD from loop”)
}
```

This program will print “HELLO WORLD from setup” once, and then print “HELLO WORLD from loop” endlessly.

Goal: Print “HELLO WORLD” with a delay of 2 sec.

```
void setup() {
  begin
}
void loop() {
  print (“HELLO WORLD”)
  delay
}
```

Functions used:

Serial.begin():

Sets the data rate in bits per second (baud) for serial data transmission.

Syntax: Serial.begin(speed)

Serial.begin(speed, config)

Parameters:

speed: in bits per second (baud)

config: sets data, parity, and stop bits.

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/

Serial.print():

Prints data to the serial port as human-readable ASCII text.

Syntax: Serial.print(val)

Serial.print(val, format)

Parameters:

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

Returns: Will return the number of bytes written.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/serial/print/

Serial.println():

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').

This command takes the same forms as Serial.print().

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/serial/println/

CH-2: Read Input

Goal: Read a character from serial monitor and send it again on serial monitor.

```
void setup() {
  begin
}
void loop() {
  print (enter a character)
  delay
  read (data sent)
  print (the character received)
}
```

Functions used:

Serial.available():

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer.

Syntax: Serial.available()

Parameters: None

Returns: The number of bytes available to read.

For Arduino reference: https://www.arduino.cc/reference/en/language/functions/communication/serial/available/

Serial.read():

Reads incoming serial data.

Syntax: Serial.read()

Parameters: None

Returns: The first byte of incoming serial data available.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/serial/read/

CH-3: Write

Goal: Read the string from serial monitor and send it again on serial monitor.

```
void setup() {
  begin
}
void loop() {
  print (enter the string)
  delay
  read (data sent)
  print (the string received)
  delay
}
```

Functions used:

Serial.write():

Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the Serial.print() function instead.

Syntax: Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

Parameters:

val: a value to send as a single byte.

str: a string to send as a series of bytes.

buf: an array to send as a series of bytes.

len: the number of bytes to be sent from the array.

Returns: Will return the number of bytes written

For Arduino reference:

https://www.arduino.cc/reference/en/language/functions/communication/serial/write/

CH-4: Delay

delay():

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax: delay(ms)

Parameters:

ms: the number of milliseconds to pause

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/time/delay/

delayMicroseconds():

Pauses the program for the amount of time (in microseconds) specified as parameter. (There are million microseconds in a second.)

Syntax: delay(us)

Parameters:

us: the number of microseconds to pause

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/time/delaymicroseconds/

micros():

Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes.

Syntax: time = micros()

Parameters: None

Returns: Number of microseconds passed since the program started.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/time/micros/

millis():

Returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

Syntax: time = millis()

Parameters: None

Returns: Number of milliseconds passed since the program started

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/time/micros/

CH-5: Configuration of pins

pinMode():

Configures the specified pin to behave either as an input or an output

Syntax: pinMode(pin, mode)

Parameters:

pin: The Arduino pin number to set the mode of.

mode: INPUT, OUTPUT, or INPUT_PULLUP.

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/

CH-6: Digital Output

Goal: Toggle the internal and one external LED.

```
void setup() {
  begin
  Set the pins as input or output
  Initialize the pins
}
void toggleLED() {
  set pin high
  delay
  set pin low
  delay
  (setting high-low will toggle the led)
}
void loop() {
  toggleLED ()
  delay
}
```

Goal: Display 0 to 9 in a 7-Segment LED.

Define the Arduino pins as pinA, pinB and so on.

```
void setup() {
  begin
  Set and initialize the pins
}
void display0 {
  Set the necessary pins
}
void display1 {
  Set the necessary pins
}
void display2 {
Set the necessary pins
}
void display3 {
  Set the necessary pins
}
void display4 {
  Set the necessary pins
}
void display5 {
  Set the necessary pins
}
void display6 {
  Set the necessary pins
}
void display7 {
  Set the necessary pins
}
```

```
void display8 {
  Set the necessary pins
}
void display9 {
  Set the necessary pins
}
void displaydecimal {
  Set the necessary pins
}
void numericalDisplay (i, isdigital) {
  if i = 0 display0
  if i = 1 display1
  if i = 2 display2
  if i = 3 display3
  if i = 4 display4
  if i = 5 display5
  if i = 6 display6
  if i = 7 display7
  if i = 8 display8
  if i = 9 display9
  if i = isdigital displaydecimal
}
void loop() {
  for (i = 0; i < 10; i++) {
    numericalDisplay(i, 0)
    delay
  }
  numericalDisplay(100, 1)
```

```
  delay
}
```

Functions used:

digitalWrite():

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V for HIGH, 0V (ground) for LOW.

If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

Syntax: digitalWrite(pin, value)

Parameters:

pin: the Arduino pin number.

value: HIGH or LOW.

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

CH-7: Digital Input

Goal: Digital I/O.

```
void setup() {
  set and initialize the i/p - o/p pins
}
void loop() {
   read the status of the button.
   if (button pressed) turn on LED
   else the LED should be off
   delay
}
```

Goal: Digital I/O using Debounce.

```
variable prev2
variable curr2
void setup() {
   set and initialize the i/p and o/p pins
   prev2 = 0
}
debounce delay value = 50ms
```

```
bool checkDebounce(int prev) {
  int currReading
  lastChangeTime = millis()
  while(true) {
   currReading = status of i/p pin
   if(currReading != prev) {
     lastChangeTime = millis()
     prev = currReading
   }
  if((millis()-lastChangeTime) > debounceDelay) {
     return true
   }
  }
  delay
 }
void toggleLED() {
  set pin high
  delay
  set pin low
  delay
  (setting high-low will toggle the led)
}
void loop() {
  curr2 = status of i/p pin
  if(curr2 != prev2) {
  if(checkDebounce(curr2)) {
     prev2 = curr2
     toggleLED ()
```

```
    }
  }
  delay
}
```

Functions used:

digitalRead():

Reads the value from a specified digital pin, either HIGH or LOW.

Syntax: digitalRead(pin)

Parameters:

pin: The Arduino pin number you want to read

Returns: HIGH or LOW

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/

CH-8: Analog Read

Goal: Analog Read.

```
variable prevValue;
variable currValue;
void setup() {
  begin
  set and initialize the i/p and o/p pins
  prevValue = currValue = 0
}
void loop() {
  currValue = read the analog value
  if (currValue != prevValue) {
    print the value in decimal
    prevValue = currValue
 }
  delay
}
```

Funtions used:

analogRead():

Reads the value from the specified analog pin (A0 – A5). It will map input voltages between 0 and the operating voltage (5V or 3.3V) into integer values between 0 and 1023.

Syntax: analogRead(pin)

Parameters:

pin: the name of the analog input pin to read from

Returns: The analog reading on the pin.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

CH-9: Analog Output (PWM)

Goal: Generate a PWM signal with increasing duty cycle and then back to 0.

```
void setup(){
  begin
  initialize the pin as i/p or o/p
}
void loop() {
  for(i = 0; i < 256; i++) {
    print (“Duty cycle of PWM: “)
    print ((i*100)/255)
    write(pin, i)
    delay
  }
  for (i = 255; i >= 0; i--) {
    print (“Duty cycle of PWM: “)
    print ((i*100)/255)
    write(pin, i)
    delay
  }
}
```

Check the output waveform using waveforms application.

Funtions used:

analogWrite():

Writes an analog value (PWM wave) to a pin (Pin no.: 3, 5, 6, 9, 10, 11).

After a call to analogWrite(), the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

Syntax: analogWrite(pin, value)

Parameters:

pin: The Arduino pin to write to.

value: The duty cycle: between 0 (always off) and 255 (always on).

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/

CH-10: Interrupt

Goal: Toggle the LED using Interrupt.

```
void blink13 (int j) {
  for (i = 0; i < j; i++) {
      set pin high
      delay
      set pin low
      delay
    }
}
void setup() {
  set the pins as i/p or o/p
  Interrupt
}
void loop() {
}
```

Functions used:

attachInterrupt():

External Interrupt

Syntax: attachInterrupt(digitalPinToInterrupt(pin), ISR, mode) (recommended)

attachInterrupt(interrupt, ISR, mode) (not recommended)

Normally you should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number.

Parameters:

interrupt: the number of the interrupt.

pin: The Arduino pin number.

ISR: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

LOW to trigger the interrupt whenever the pin is low.

CHANGE to trigger the interrupt whenever the pin changes value.

RISING to trigger when the pin goes from low to high.

FALLING for when the pin goes from high to low.

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

interrupts():

Re-enables interrupts (after they've been disabled by noInterrupts()). Interrupts allow certain important tasks to happen in the background and are enabled by default.

Syntax: interrupts()

Parameters: None

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/interrupts/interrupts/

noInterrupts():

Disables interrupts.

Syntax: noInterrupts()

Parameters: None

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/interrupts/nointerrupts/

CH-11: Timer based Interrupt

Goal: Toggle the LED using Timer based Interrupt.

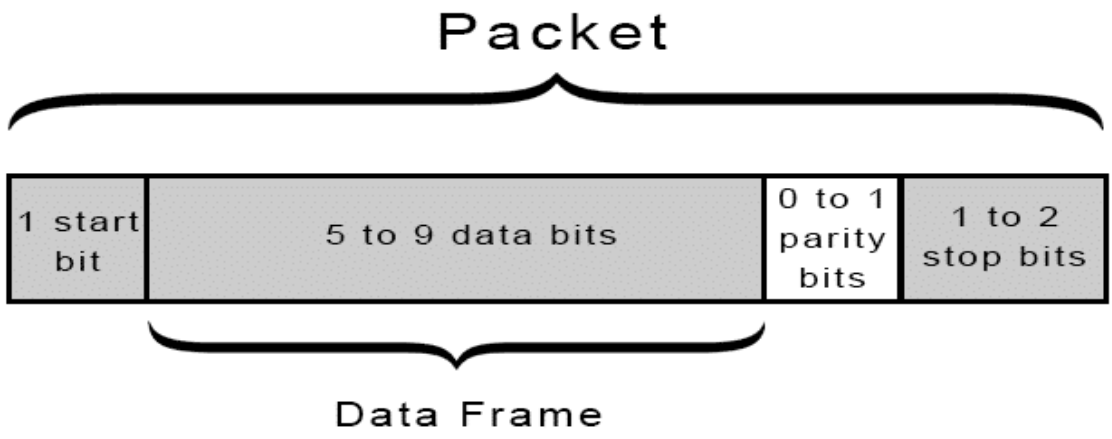
```
void setTimer1InitValue (initValue) {
 set timer counter register value to initValue
 TCNT1 = initValue
}
void setTimer1CompareValue(matchValue) {
set the compare register value to matchValue
OCR1A = matchValue
}
void setupTimer1(mode, preScaler, initValue, matchValue) {
disable all interrupts
initialize both the TCC Registers
Set initial value from which the timer will start counting
setTimer1InitValue (initValue);
set the prescaler
TCCR1B |= (1 << CS12)
set the mode of operation
if (mode == 0) {
   enable timer overflow interrupt
   TIMSK1 |= (1 << TOIE1)
 }
```

```
  else {
    enable timer compare
    TIMSK1 |= (1 << OCIE1A)
  }
Enable the interrupts
}
void toggleLED {
   set pin high
   delay
   set pin low
   delay
}
interrupt service routine
ISR(TIMER1_OVF_vect) {
  setTimer1InitValue(0)
  toggleLED()
}
void setup() {
  begin
  set and initialize the pins
  setupTimer1(0, 0, 0, 0);
}
void loop() {
  delay
}
```

CH-12: UART

UART (Universal Asynchronous Transmitter Receiver): UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end. UART will convert the incoming and outgoing data into the serial binary stream. It is a one-to-one communication.

Bit Frame:



The functions Serial.print and Serial.write does sends and prints the data to the serial monitor but it also transmits the data through Tx pin of the Arduino. Thus the same functions are used for UART communication.

SoftwareSerial library helps us to use any other pins as the Tx and Rx pins.

SoftwareSerial Library:

The Arduino hardware has built-in support for serial communication on pins 0 (Rx) and 1 (Tx).

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality.

To include this library: #include <SoftwareSerial.h>

SoftwareSerial():

Create an instance of a SoftwareSerial object. Multiple SoftwareSerial objects may be created, however only one can be active at a given moment.

Syntax: SoftwareSerial name (rxPin, txPin, inverse_logic)

(Example: SoftwareSerial mySerial (rxPin, txPin))

Name is the SoftwareSerial object name of your choice.

Parameters:

rxPin: the pin on which to receive serial data.

txPin: the pin on which to transmit serial data.

inverse_logic: used to invert the sense of incoming bits (the default is normal logic). If set, SoftwareSerial treats a LOW (0v on the pin, normally) on the RX pin as a 1-bit (the idle state) and a HIGH (5V on the pin, normally) as a 0-bit. It also affects the way that it writes to the TX pin. Default value is false.

Returns: Nothing

For Arduino reference: https://docs.arduino.cc/learn/built-in-libraries/software-serial

Every command takes the same as in normal mode just the syntax will be different.

Syntax: mySerial.function()

Goal: UART Send and Receive.

UART Send:

```
void setup() {
  begin
```

```
  counter = 0
}
void loop()
  if (counter%3 == 0) {
    write("00")
    delay
  }
  else if (counter%3 == 1) {
    write("1111")
    delay
  }
  else {
    write("Some junk character")
    delay
  }
  counter++;
  delay
}
```

UART Receive:

```
void setup() {
  begin
}
void loop() {
  str[256];
  strLen = 0;
  while (if serial available) {
```

```
   read the data and store into the string
  }
  Terminate the string's last character as 0
  str[strLen] = 0
  print
  delay
}
```

Goal: UART Send and Receive. (See the output when printBuffer function is called)

UART Send:

```
txData[]
txDataLen = 200
void setup()
{
   begin
   while(serial not avilable) {
      delay
  }
}
(if print buffer is enabled, then you will get weird result at the destination)
void printBuffer (len, *buf) {
  print (sending data: it's length)
  for(i = 0; i < len; i++) {
    print(data)
  }
```

```
  print("\n")
}
void writeUART(len, *buf) {
  write(data)
  write('\n')
//  printBuffer(len, buf)
}
void loop()
{
  for(int i = 0; i < 10; i++) {
    writeUART(20*(i+1), txData)
    delay
  }
}
```

UART Receive:

```
rxData
rxDataLen = 100
void setup() {
   begin
}
void printBuffer(len, *buf) {
    print (received data: it's length and the data)
    print("\n")
}
void readUART(printRecvd) {
  count = 0;
```

```
  while(true) {
    if(Serial is available) {
      read the data
      if('\n') break
        count++
      if (count >= rxDataLen) break
    }
  }
  Terminate the string
  if(printRecvd) {
    printBuffer(count, rxData)
  }
}

void writeUART(len, *buf){
  write(data)
  write('\n')
}
void loop()
{
  count = 0;
  if (Serial is available) {
    readUART(true)
  }
  delay
}
```

Goal: UART Send and Receive on both sides. (See the output when printBuffer function is called)

UART Send & Receive:

```
txData[]
txDataLen = 200
rxData[241]
rxDataLen = 240
void setup()
{
  begin
  while (Serial not available) {
    delay
 }
}
(if print buffer is enabled, then you will get wierd result at the destination)
void printBuffer (*str, len, *buf) {
 print("\n")
 print the string and its length
 for (i = 0; i < len; i++) {
   print the data
 }
 print("\n")
}
```

```
void writeUART(printBuf, len, *buf) {
  wirte the data
  write('\n');
  if (printBuf) printBuffer("SendRecv:Sending", len, buf)
}
void readUART(printRecvd) {
  count = 0
  loopCount = 0
  while(true) {
    if (Serial is available) {
      read the data
      if('\n') break
        count++
      if (count >= rxDataLen) break;
    }
    else {
      loopCount++
    }
    If (loopCount == 16*1024) break;
  }
Terminate the string
  If (printRecvd) {
    printBuffer("SendRecv::Receiving", count, rxData)
  }
}

int loopCount = 0
void loop()
```

```
{
  loopCount = (loopCount+1) % 20
  writeUART(false, 20*(loopCount+1), txData)
  delay
  readUART(true)
  delay
}
```

UART Receive & Send:

```
include SoftwareSerial.h library
rxData[101]
rxDataLen = 100



void setup()
{
  begin
}

void printBuffer(reverse, *str, len, *buf) {
    print("\n")
    print the string and its length
    variable offset
    for (i = 0; i < len; i++) {
      if (reverse) offset = len-i-1
      else offset = i
      print(buf[offset])
```

```
    }
    print(buf)
    print("\n")
}
void writeUART(reverse, printBuf, len, *buf) {
  variable offset
  for (i = 0; i < len; i++) {
    if (reverse) offset = len-i-1
    else offset = i
    write(buf[offset])
  }
  write('\n')
  delay
  if (printBuf) printBuffer(reverse, "RecvSend::Sending", len, buf)
}
int readUART (printRecvd) {
  count = 0
  if (serial not available) return count
  while (true) {
    if (Serial is available) {
      read the data
      if('\n') break
        count++
      if (count >= rxDataLen) break
    }
  }
  Terminate the string
  If (printRecvd) {
```

```
    printBuffer (false, "RecvSend::Received", count, rxData)
  }
  return count

void loop()
{
  count = 0
  if (Serial is available) {
    count = readUART(false)
  }
  If (count>0) {
    writeUART(true, false, count, rxData)
  }
  else {
    print("Nothing")
  }
  delay
}
```

Goal: UART Send and Receive on both sides using SoftwareSerial library.

UART Send & Receive:

```
include SoftwareSerial.h library
txData[] =
"...0001...0123456789...0002...0123456789...0003...0123456789...0004...0123456789.
..0005...0123456789...0006...0123456789...0007...0123456789...0008...0123456789...
0009...0123456789...0010...0123456789"
txDataLen = 200
```

```
rxData[256]
rxDataLen = 255
superloop
SoftwareSerial newUART(2, 3); // (RX, TX)
void setup()
{
  begin
  while (Serial not available) {
  delay
 }
 superloop = 0
}
(if print buffer is enabled, then you will get wierd result at the destination)
void printBuffer (*str, len, *buf) {
 print("\n")
 print(superloop, DEC)
 print(str)
 print(len, DEC)
 write(buf, len)
}
void writeUART (printBuf, len, *buf) {
 if (printBuf) printBuffer("SendRecv::Send", len, buf)
 send (buf, len)
 send ('\n')
}
void readUART (bool printRecvd) {
 count = 0
 loopCount = 0
```

```
  while (true) {
    if (data available) {
      read the data
      if (rxData[count] == '\n') break
        count++
      if (count >= rxDataLen) break
    }
    else {
      loopCount++
    }
    If (loopCount == 16*1024) break
  }
  rxData[count] = 0
  if (printRecvd) {
    printBuffer ("SendRecv::Recv", count, rxData)
  }
}
void loop()
{
  txDataSendLen = 20*((superloop%10)+1)
  writeUART(true, txDataSendLen, txData)
  delay
 while (data not available)                    (endless loop till char is available)

  (Loop and clear all available chars in the UART
  BASICALLY handle if the UART sends answer in small parts
  till all the receive is done)
  do {
```

```
   readUART(true)
   delay
  }
while (data available)
delay
superloop++
}
```

UART Receive & Send:

```
include SoftwareSerial.h library
rxData[101]   (change it to 256 and see the difference)
rxDataLen = 100  (change it to 255 and see the difference)
txData[256]
txDataLen = 255
superLoop
SoftwareSerial newUART(2, 3)  (RX, TX)

bool setTxDATA(reverse, bufLen, *buf) {
  offset = 0
  if (bufLen >= txDataLen) bufLen = txDataLen
  if (bufLen <= 0) return false;
  for (i = 0; i < bufLen; i++) {
    if (reverse) offset = bufLen-i-1
    else offset = i
    txData[offset] = buf[i]
  }
  txData[bufLen] = 0
```

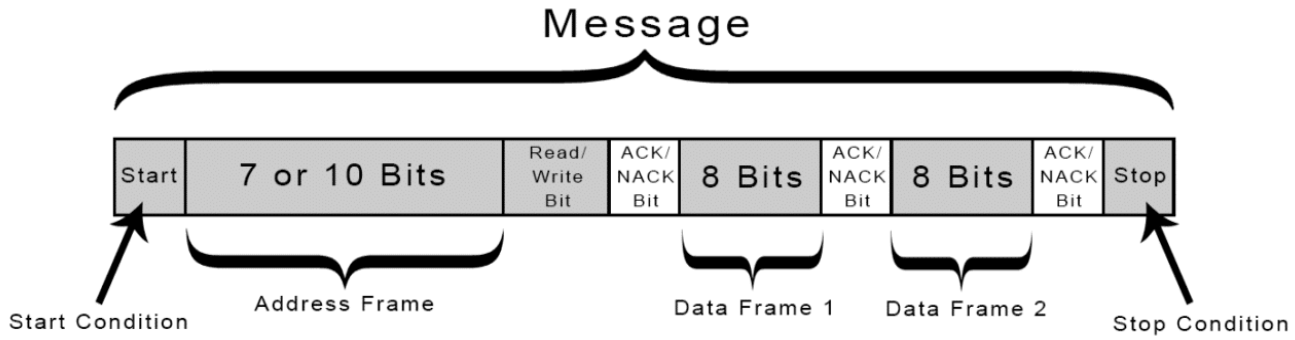
```
  return true
}
void setup()
{
  begin
  superLoop = 0
}
void printBuffer(*str, len, *buf) {
    print ("\n")
    print (superLoop, DEC)
    print(str)
    print(len, DEC)
    print(buf)
}
void writeUART(printBuf, len, *buf) {
  write(data)
  write('\n')
  delay
  if (printBuf) printBuffer("RecvSend::Send", len, txData)
}

int readUART (printRecvd) {
  count = 0
  if (serial not available) return count
  while(true) {
    if (serial is available) {
      read the data
      if ('\n') break
```

```
      count++
     if (count >= rxDataLen) break
    }
  }
  Terminate the string
  If (printRecvd) {
    printBuffer ("RecvSend::Recv", count, rxData)
  }
  return count
}
void loop()
{
  count = 0
  while (serial is available) {
    count = readUART(true)
    if (count>0) {
      if (setTxDATA(true, count, rxData)) {
        writeUART(true, count, txData)
      }
    }
  }
  else {
    print("Nothing")
  }
  delay
  superLoop++
}
```

CH-13: I2C

I2C: I2C uses only two bidirectional open-collector or open-drain lines: serial data line (SDA) and serial clock line (SCL), pulled up with resistors. the data line can't change when the clock line is high, it can change only when the clock line is low. It is a half-duplex communication.

Bit Frame:



Wire Library:

This library allows you to communicate with I2C/TWI devices.

I2C pins in UNO: A4 (SDA), A5 (SCL)

To include this library: #include <Wire.h>

Goal: I2C Mater-Slave (Master send and Slave Receive)

I2C Master:

include Wire.h library

```
tx_data[] = "HELLO"
void setup()
{
  begin
}
void loop()
{
 begin transmission(slave address)
 for(i=0; i<6; i++)
 {
    Send (tx_data[i])
    Delay
 }
  end transmission
  delay
}
```

I2C Slave:

```
include Wire.h library
rx_data[10]
void setup()
{
  begin(address)
  Wire.onReceive(receiveEvent)
  begin
}
void loop()
```

```
{
  delay
}
void receiveEvent (howmany)
{
  while (data available)
  {
    for (i=0; i<6; i++)
   {
     Read the data
     delay
     print the data
   }
  }
 delay
}
```

Functions used:

Wire.begin():

This function initializes the Wire library and join the I2C bus as a controller or a peripheral. This function should normally be called only once.

Syntax: Wire.begin()

Wire.begin(address)

Parameters:

address: the 7-bit slave address (optional); if not specified, join the bus as a controller device.

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/wire/begin/

beginTransmission():

This function begins a transmission to the I2C peripheral device with the given address.

Syntax: Wire.beginTransmission(address)

Parameters:

address: The 7-bit address of the device to transmit to.

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/wire/begintransmission/

Wire.write():

This function writes data from a peripheral device in response to a request from a controller device, or queues bytes for transmission from a controller to peripheral device

Syntax: Wire.write(value)

Wire.write(string)

Wire.write(data, length)

Parameters:

value: a value to send as a single byte.

string: a string to send as a series of bytes.

data: an array of data to send as bytes.

length: the number of bytes to transmit.

Returns: The number of bytes written.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/wire/write/

Wire.available():

This function returns the number of bytes available for retrieval with read().

Syntax: Wire.available()

Parameters: None

Returns: The number of bytes available for reading.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/wire/available/

Wire.read():

This function reads a byte that was transmitted from a peripheral device to a controller device after a call to requestFrom() or was transmitted from a controller device to a peripheral device.

Syntax: Wire.read()

Parameters: None

Returns: The next byte received.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/wire/read/

Goal: I2C Mater-Slave (Send by master and Receive by slave, followed by Request from master and transmit by slave)

I2C Master:

include wire.h library

Define slave address

Set the value and length of buffer

txBuffer = "0123456789"

txBufferLen = 10

```
void setup() {
   begin
}
void loop() {
  delay
  print ("Sending Data to the I2C Slave")
  begin transmission
  write (txBuffer)
  end transmission
  delay
  print ("Receiving Data to the I2C Slave")
  request from slave
  while(not available)) {
```

```
   delay
  }
  while(available) {
   read the data
  }
  Print the received data
}
```

I2C Slave:

```
include wire.h library

Define slave address
Set the value and length of buffer
rxBuffer
txBufferLen = 10
void setup() {
  begin
  I2C begin
  Receive(rxEvent)
  Request(rqEvent)
}
void rxEvent(howMany)
{
  rxBuffer = ""
  while( Wire isavailable) {
   read the data
  }
```

```
  print(rxBuffer)
}
void rqEvent()
{
  rxBuffer += "000"
  write(rxBuffer)
}
void loop() {
;
}
```

Functions used:

requestFrom():

This function is used by the controller device to request bytes from a peripheral device.

Syntax: Wire.requestFrom(address, quantity)

Wire.requestFrom(address, quantity, stop)

Parameters:

address: the 7-bit slave address of the device to request bytes from.

quantity: the number of bytes to request.

stop: true or false. true will send a stop message after the request, releasing the bus. False will continually send a restart after the request, keeping the connection active.

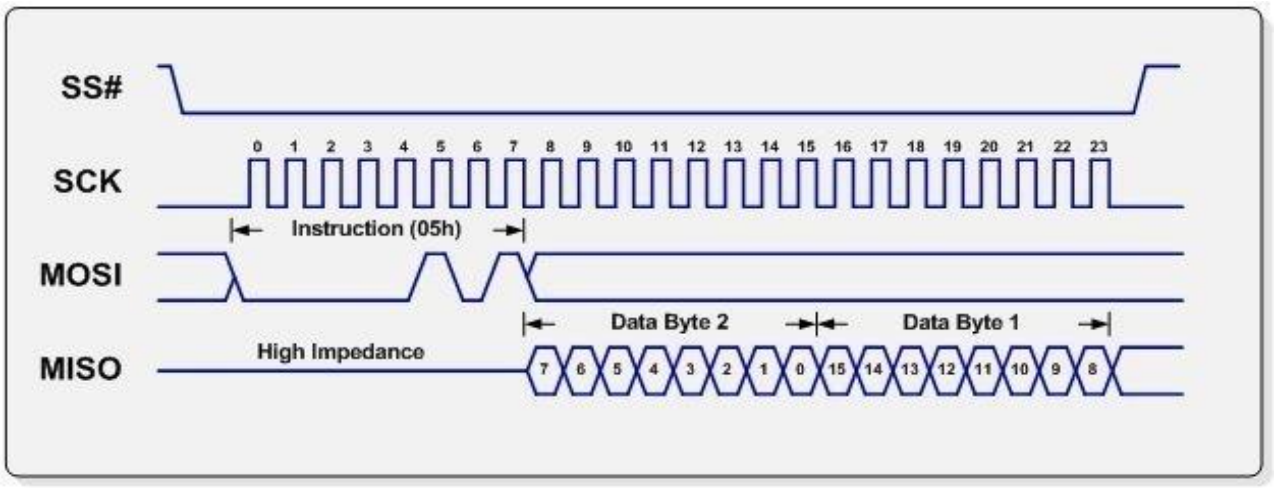
Returns: The number of bytes returned from the peripheral device.

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/wire/requestfrom/

CH-14: SPI

SPI: SPI is a four-wire synchronous serial communication protocol. SPI is one master and multi slave communication. It is a full duplex communication. Multiple slave-devices may be supported through selection with individual chip select (CS).

Bit Frame:



SPI Library:

This library allows you to communicate with SPI devices, with the Arduino as the master device.

SPI pins in UNO: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)

To include this library: #include < SPI.h>

Goal: SPI Mater-Slave

SPI Master:

```
include SPI.h library
void setup (void)
{
  begin
  Set the slave select pin high
  SPI begin
  Set the clock to 1 MHz
}
void loop(void)
{
  x[6]="HELLO"
  Mastersend;
  for (i=0; i<6; i++)
  {
     Set the slave select pin low
     Send the data
  }
  Set the slave select pin high
  delay
  }
```

SPI Slave:

```
include SPI.h library
received
Slavereceived
void setup()
{
  Set the clock frequency, mode of transmission and MSB or LSB first
  begin
  set the i/p-o/p pins
  SPCR |= _BV(SPE)
  received = false
  SPI Interrupt
}
ISR (SPI_STC_vect)
{
  Slavereceived = SPDR
  received = true
}
void loop()
{
  If (received)
   {
    Print (Slavereceived)
```

```
   delay
   }
}
```

Goal: SPI Mater-Slave (Send and Receive on both sides)

SPI Master:

```
include SPI.h library

masterTXBuf[] = "THIS IS THE MASTER SEND"
masterTXBufLen = 23
masterRXBuf[128]
masterBufOffset = 0
void setup (void)
{
  begin
  make SS pin High
  SPI begin
  Divide the clock by 16 to get 1MHz clock frequency
}
void spiTXRX() {
  masterBufOffset = 0
  i = 0
 make SS pin Low
 for (i = 0; i < masterTXBufLen; i++) {
   masterRXBuf[i] = SPI transfer(masterTXBuf[i])
   print("transmitted:")
   print(masterTXBuf[i])
```

```
    print("received:")
    println(masterRXBuf[i])
  }
  Make SS pin High
  Terminate the receive buffer string
}
void loop(void)
{
  spiTXRX()
  print("Sent:")
  print(masterTXBuf)
  print("Received:")
  print(masterRXBuf)
  delay
}
```

SPI Slave:

include SPI.h library

rxDone

slaveRX

slaveTX

slaveTXBuf[] =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

slaveTXBufLen = 62

slaveTXOffset

slaveRXBuf[256]

slaveRXBufLen = 256

slaveRXOffset = 0

```
char getslaveSend(void) {
  slaveTX = slaveTXBuf[slaveTXOffset]
  slaveTXOffset++
  if (slaveTXOffset == slaveTXBufLen) slaveTXOffset = 0
  return slaveTX
}
```

prevIfNum

loopNum

```
void setslaveReceived() {
```

```
  if (slaveRXOffset >= slaveRXBufLen-1) {
    slaveRXOffset = 0
    for (int i = 0; i < slaveRXBufLen; i++) {
      slaveRXBuf[i] = 0
    }
  }
  slaveRXBuf[slaveRXOffset] = slaveRX
  slaveRXOffset++
}
void setup()
{
  begin
  Set the clock frequency, mode of transmission and MSB or LSB first
  Set the i/p – o/p pins
  SPCR |= _BV(SPE)
  rxDone = false
  SPI interrupt

  slaveTXOffset = 0
  slaveRXOffset = 0
  SPDR = getslaveSend()

 prevIfNum = 0
 loopNum = 0
}
ISR (SPI_STC_vect)
{
  slaveRX = SPDR
```

```
  setslaveReceived()
  SPDR = getslaveSend()
  rxDone = true
}
void loop()
{
  If (rxDone) {
    rxDone = false
    print("RX:")
    print(slaveRX)
  }

  //print if a character has been received
  If (slaveRXOffset > 0 &&(slaveRXOffset != prevIfNum)) {
    prevIfNum = slaveRXOffset
    print("RXBUF:")
    print(slaveRXOffset, DEC)
    print(slaveRXBuf)
  }
}
```

Functions used:

SPI.begin():

Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.

Syntax: SPI.begin()

Parameters: None

Returns: Nothing

For Arduino reference:
https://www.arduino.cc/reference/en/language/functions/communication/spi/begin/

SPI.beginTransaction():

Initializes the SPI bus using the defined SPISettings.

Syntax: SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0))

SPISettings mySetting(speedMaximum, dataOrder, dataMode)

When all of your settings are constants, SPISettings should be used directly in SPI.beginTransaction().

If any of your settings are variables, you may create a SPISettings object to hold the 3 settings. Then you can give the object name to SPI.beginTransaction().

Parameters:

speedMaximum: The maximum speed of communication. For a SPI chip rated up to 20 MHz, use 20000000.

dataOrder: MSBFIRST or LSBFIRST

dataMode: SPI_MODE0, SPI_MODE1, SPI_MODE2, or SPI_MODE3Returns: None

For Arduino reference:

https://www.arduino.cc/reference/en/language/functions/communication/spi/begintransaction/

https://www.arduino.cc/reference/en/language/functions/communication/spi/spisettings/

SPI.setClockDivider():

Sets the SPI clock divider relative to the system clock.

Syntax: SPI.setClockDivider(divider)

Parameters:

divider: SPI_CLOCK_DIV2

SPI_CLOCK_DIV4

SPI_CLOCK_DIV8

SPI_CLOCK_DIV16

SPI_CLOCK_DIV32

SPI_CLOCK_DIV64

SPI_CLOCK_DIV128

Returns: Nothing

For Arduino reference: https://www.arduino.cc/reference/en/language/functions/communication/spi/setclockdivider/

SPI.transfer():

SPI transfer is based on a simultaneous send and receive.

Syntax: receivedVal = SPI.transfer(val)

SPI.transfer(buffer, size)

Parameters:

val: The byte to send out over the bus.

buffer: The array of data to be transferred.

Returns: The received data.

For Arduino reference: https://www.arduino.cc/reference/en/language/functions/communication/spi/transfer/