Introduction to Arduino

Prof. Dhaval Shah

Outline

- What is Microcontroller
- Microprocessor vs Microcontroller
- Classification
- Arduino Boards
- Features of Arduino UNO
- ATmega328
- Arduino Programming

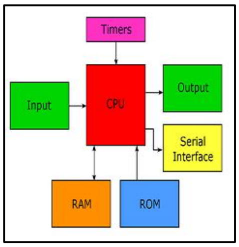
What is Microcontroller?

- Microcontrollers are small computers integrated into a single chip
- It contains :
  - Processing core
  - Flash Memory for program
  - I/O peripherals
  - RAM
  - Peripherals such as clocks, timers, PWM etc…

- Microprocessors are used for general purpose applications, while microcontrollers are self sufficient and are used for specific tasks.
- Microcontrollers are an example of embedded systems.

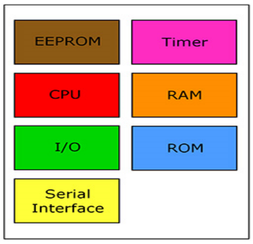
Microprocessor Vs Microcontroller

Microprocessor



CPU and several supporting chips

Microcontroller

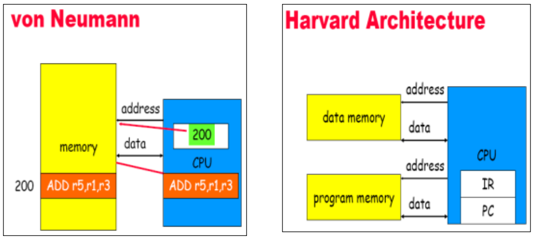


CPU on single chip

Classification

- Based on Hardware
  - Von Neuman
  - Hardvard
- Based on Instruction Set Architecture
  - RISC (Reduced Instruction Set Computing)
  - CISC (Complex Instruction Set Computing)

Classification: Based on Hardware



Classification Based on ISA

Complex Instruction Set Computer (CISC)

- Requires multiple cycles for a execution.
- Different instructions of different length and format
- Limited general purpose register
- A large Number of instructions

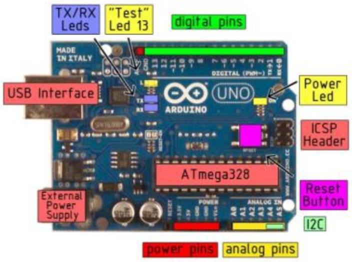
Reduced Instruction Set Computer (RISC)

- Instruction can be executed in a single cycle.
- Each instruction of fixed length and format
- Large general purpose register set
- Compaq instruction set

What is Arduino?

- The Arduino is a microcontroller development platform(not a microcontroller….) board with a USB plug.
- It is an open-source physical computing platform.
- It can be used to develop stand-alone interactive objects or can be connected to software on your computer.
- Easy-to-use hardware and software.
- It’s intended for students, artists, designers, hobbyists and anyone who tinker with technology.
- It is programmed in Arduino Programming language(APL) similar to C/C++.

Arduino UNO



- 16 MHz with auto-reset,
- 6 Analog In,
- 14 Digital I/O
- 6 PWM

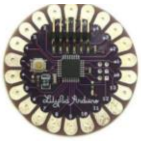
Courtesy: A000066-Arduino-datasheet-38879526.pdf (octopart.com)

Features of Arduino UNO

- An easy USB interface
- convenient power management and built-in voltage regulation
- An easy-to-find, and dirt cheap, microcontroller "brain“
  - timers, PWM pins, external and internal interrupts, and multiple sleep modes
- 16 MHz clock
- 32 KB of flash memory
- 13 digital pins and 6 analog pins
- An ICSP connector
- An on-board LED

Other Flavors

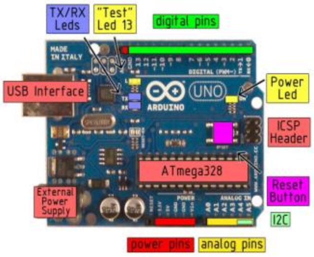
MEGA


LILYPAD

MINI

NANO 43mm x 18mm

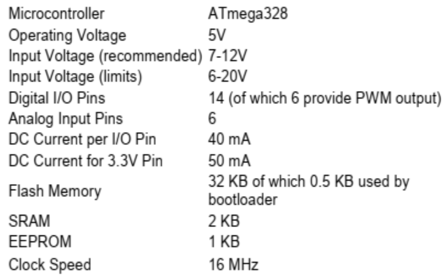
Arduino MEGA



- 16 MHz with auto-reset,
- 16 Analog In,
- 54 Digital I/O
- 6 PWM

Courtesy: A000066-Arduino-datasheet-38879526.pdf (octopart.com)

Arduino MEGA – Technical Specs



Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

Courtesy: A000066-Arduino-datasheet-38879526.pdf (octopart.com)

ATmega328

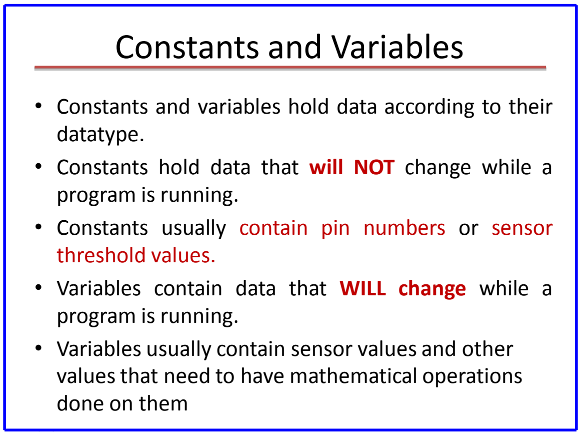
- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller
- **Advanced RISC Architecture**
  – Most Single Clock Cycle Execution
  – 32 x 8 General Purpose Working Registers
  – High Endurance Non-volatile Memory Segments
  – 32KBytes of In-System Self-Programmable Flash program **Memory**
  – 1KBytes EEPROM
  – 2KBytes Internal SRAM
- Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
- In-System Programming by On-chip Boot Program

Arduino Programming

- Arduino programming language is based on C/C++
- Simpler and easier to learn (Arduino programming is like building with LEGO blocks)
- Certain rules must be followed and different building blocks can be used to build bigger parts.
  - Every line must either end with a semicolon ';' unless it's a conditional, loop, or function
  - Comments start with a //
  - Comments are text that the program ignores

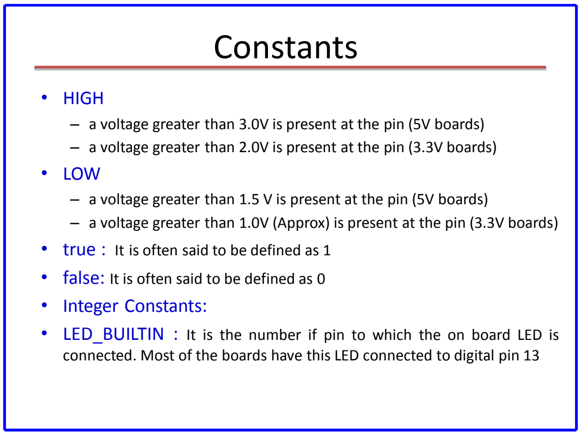
Arduino Programming

- Arduino programming language can be divided in three main parts:
  - functions
    - Digital I/O, Analog I/O, Interrupts, Communication, Time, etc..
  - Values
    - Constants, Variables, datatypes, conversions
  - Structures
    - Sktech (loop, setup), operators (arithmetic, comparison, boolean, bitwise, etc…), control structure (break, if, else, for, while, etc…..)



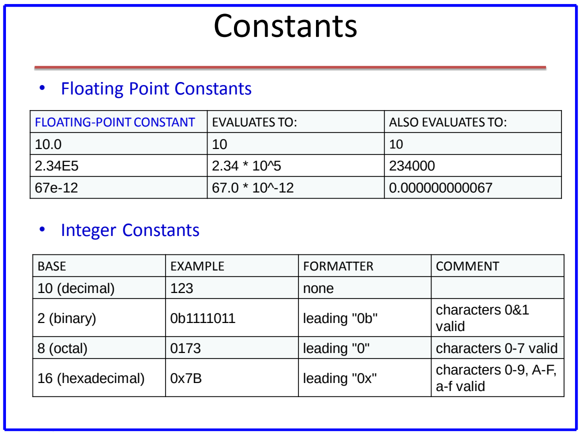
Constants and Variables

- Constants and variables hold data according to their datatype.
- Constants hold data that **will NOT** change while a program is running.
- Constants usually contain pin numbers or sensor threshold values.
- Variables contain data that **WILL change** while a program is running.
- Variables usually contain sensor values and other values that need to have mathematical operations done on them



Constants

- HIGH
  - a voltage greater than 3.0V is present at the pin (5V boards)
  - a voltage greater than 2.0V is present at the pin (3.3V boards)
- LOW
  - a voltage greater than 1.5 V is present at the pin (5V boards)
  - a voltage greater than 1.0V (Approx) is present at the pin (3.3V boards)
- true : It is often said to be defined as 1
- false: It is often said to be defined as 0
- Integer Constants:
- LED_BUILTIN : It is the number if pin to which the on board LED is connected. Most of the boards have this LED connected to digital pin 13



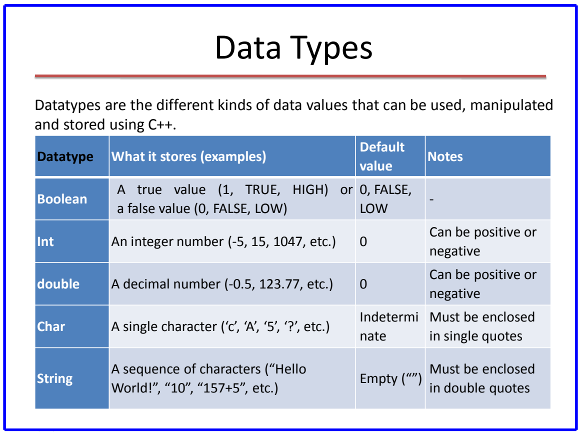
Constants

- Floating Point Constants

FLOATING-POINT CONSTANT	EVALUATES TO:	ALSO EVALUATES TO:
10.0	10	10
2.34E5	2.34 * 10^5	234000
67e-12	67.0 * 10^-12	0.000000000067

- Integer Constants

BASE	EXAMPLE	FORMATTER	COMMENT
10 (decimal)	123	none	
2 (binary)	0b1111011	leading "0b"	characters 0&1 valid
8 (octal)	0173	leading "0"	characters 0-7 valid
16 (hexadecimal)	0x7B	leading "0x"	characters 0-9, A-F, a-f valid



Data Types

Datatypes are the different kinds of data values that can be used, manipulated and stored using C++.

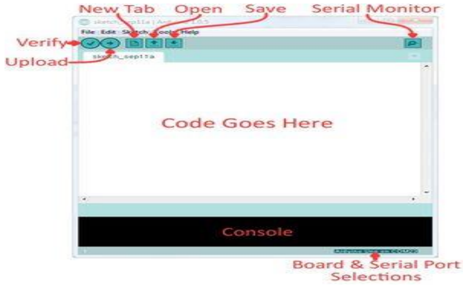
Datatype	What it stores (examples)	Default value	Notes
Boolean	A true value (1, TRUE, HIGH) or a false value (0, FALSE, LOW)	0, FALSE, LOW	-
Int	An integer number (-5, 15, 1047, etc.)	0	Can be positive or negative
double	A decimal number (-0.5, 123.77, etc.)	0	Can be positive or negative
Char	A single character (‘c’, ‘A’, ‘5’, ‘?’, etc.)	Indeterminate	Must be enclosed in single quotes
String	A sequence of characters (“Hello World!”, “10”, “157+5”, etc.)	Empty (“”)	Must be enclosed in double quotes

Operators

- The results of these operations are usually stored in a variable.

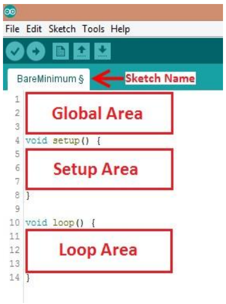
Operator	What it does
=	Assigns a value to a variable
+	Adds two or more values
-	Subtracts two or more values
*	Multiplies two or more values
/	Divides two or more values
++	Increment by 1
--	Decrement by 1
==	Checks if two value are equal
!=	Checks if two value are not equal
> or <	Greater than/ Less than comparison
<= or >=	Less than/greater than or equal to comparison
&& or ||	Boolean AND or Boolean OR Used to cascade multiple Boolean operations

Arduino IDE



- Get the latest version of Arduino IDE from https://www.arduino.cc/en/Main/Software

How to write a code in Arduino?



```
void setup() {

}

void loop() {

}
```

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
    int led = 13;

    // the setup routine runs once when you press reset:
    void setup() {
    // initialize the digital pin as an output.
    pinMode(led, OUTPUT);
    }

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

I/O Programming and Interfacing

Prof. Dhaval Shah

Outline

- I/O interface
- Digital IO functions
- LED interfacing
- Push button interfacing
- Button Debounce
- Hex Keypad interfacing
- Seven segment interfacing

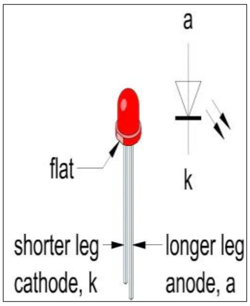
I/O interfacing

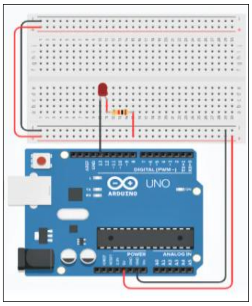
- An **input/output** pin, or I/O pin, is the interface between a microcontroller and another circuit.
- Used to exchange the information between two devices
- The digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands.

Digital IO functions

1. pinMode() : Configures the specified pin to behave either as an input or an output
   - Syntax : pinMode(pin, mode)
     - Pin: the Arduino pin number to set the mode of
     - Mode: INPUT, OUTPUT or INPUT_PULLUP
2. digitalRead() : Read the value from a specified digital pin, either HIGH or LOW
   - Syntax : digitalRead(pin, value)
3. digitalWrite () : Write a HIGH or LOW to a digital pin
   - Syntax : digitalRead(pin, value)

LED interfacing





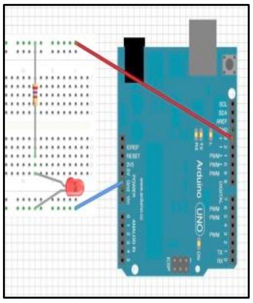
How to write a code in Arduino?

- Define led pin--→ use constant int type

```
int led = 13;
```

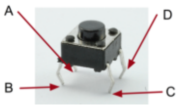
- Setup the pin modes for led under void setup
  - pinMode(ledPin, OUTPUT);
  - pinMode(buttonpin, INPUT);
- Define the on/off conditions under void loop

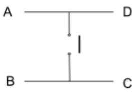
```
void loop() {
digitalWrite(led, HIGH); // turn the LED on

delay(1000); // wait for a second
digitalWrite(led,  LOW);  //  turn  the  LED  off

delay(1000); // wait for a second
}
```



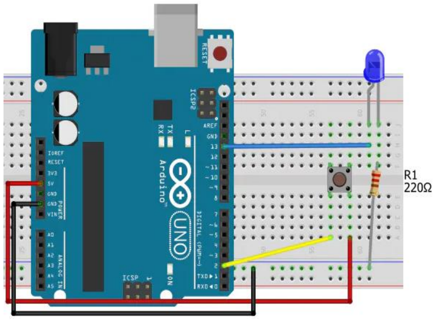
Push Button Switch

- When you press a button or flip a lever, they connect two contacts together so that current flow through them.
- There are only really two electrical connections, as inside the switch package pins B and C are connected together, as are A and D.





Push Button Switch - Interfacing



Push Button Switch - Interfacing

- Define button pin --→ use a constant int type

  const int buttonPin = 2;

- Define led pin--→ use constant int type

  const int ledPin = 10;

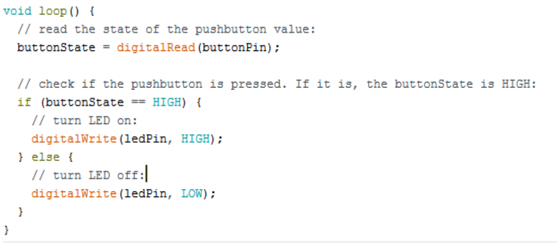
- Initialize button state -→

  int buttonState = 0;

- Setup the pin modes for both the pins under void setup
  - pinMode(ledPin, OUTPUT);
  - pinMode(buttonpin, INPUT);

Push Button Switch - Interfacing

- Set the conditions for LED on or off under void loop after reading button state



```
void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Push Button Switch - Debounce

- System that counts the **number of times a button** is pressed, One may count individual presses as multiple hits. The solution to this problem is called denouncing
- pushbutton or **any switch's position is changed noise is generated**.
  - noise (contact) occurs because the switch contact is metal and it has elasticity
- contact bounce: switch is moved to a new position it strikes a metal contact and physically bounces a few times.
- Bouncing happens in a matter of milliseconds – but microcontroller is moving so fast that it will detect a transition between two states every time the button bounces

Push Button Switch - Debounce

- // Define the pins (constants won't change)

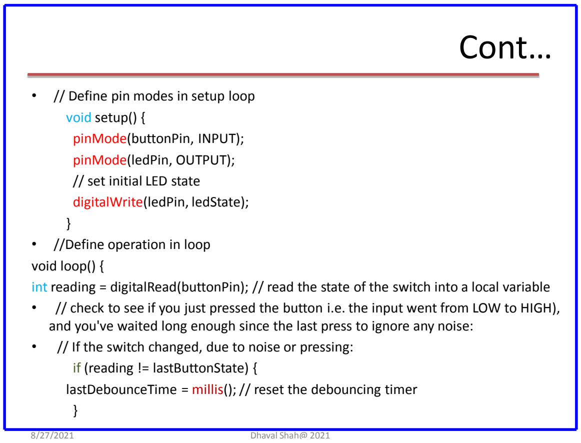
  const int buttonPin = 2; // the number of the pushbutton pin
  const int ledPin = 13; // the number of the LED pin

- // Variables will change:

  int ledState = HIGH; // the current state of the output pin
  int buttonState; // the current reading from the input pin
  int lastButtonState = LOW; // the previous reading from the input pin

- // New variables are unsigned longs because the time, measured in milliseconds, will quickly become a bigger number than can be stored in an int.

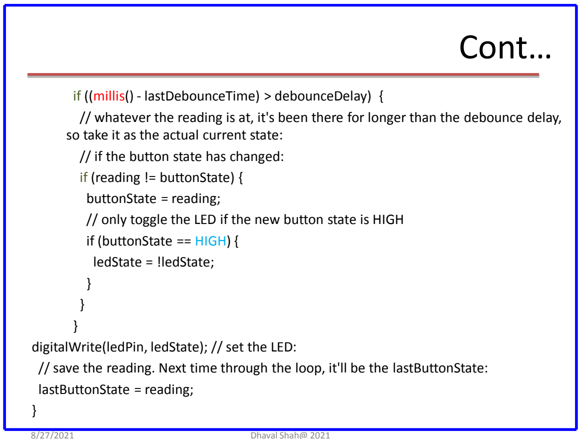
  unsigned long lastDebounceTime = 0; // the last time the output pin was toggled
  unsigned long debounceDelay = 50; // the debounce time; increase if the output flickers



Cont…

```
•   // Define pin modes in setup loop
      void setup() {
        pinMode(buttonPin, INPUT);
        pinMode(ledPin, OUTPUT);
        // set initial LED state
        digitalWrite(ledPin, ledState);
      }
•   //Define operation in loop
void loop() {
int reading = digitalRead(buttonPin); // read the state of the switch into a local variable
•   // check to see if you just pressed the button i.e. the input went from LOW to HIGH),
    and you've waited long enough since the last press to ignore any noise:
•   // If the switch changed, due to noise or pressing:
        if (reading != lastButtonState) {
      lastDebounceTime = millis(); // reset the debouncing timer
        }
```

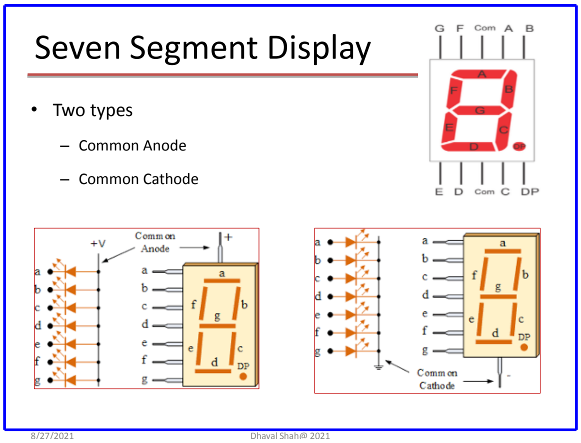
8/27/2021 Dhaval Shah@ 2021



Cont…

```
        if ((millis() - lastDebounceTime) > debounceDelay)  {
         // whatever the reading is at, it's been there for longer than the debounce delay,
       so take it as the actual current state:
         // if the button state has changed:
         if (reading != buttonState) {
          buttonState = reading;
          // only toggle the LED if the new button state is HIGH
          if (buttonState == HIGH) {
           ledState = !ledState;
          }
         }
        }
digitalWrite(ledPin, ledState); // set the LED:
 // save the reading. Next time through the loop, it'll be the lastButtonState:
 lastButtonState = reading;
}
```

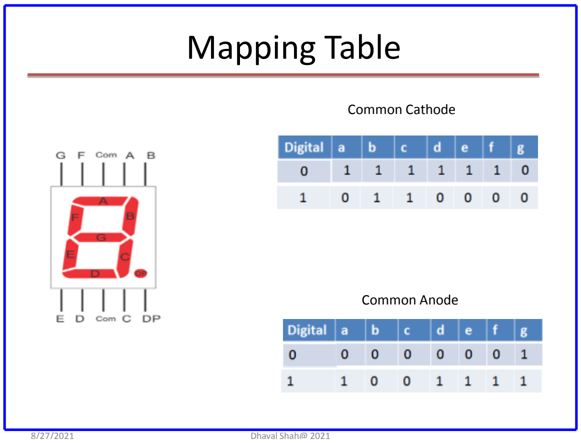
8/27/2021 Dhaval Shah@ 2021



Seven Segment Display

- Two types
  - Common Anode
  - Common Cathode

8/27/2021 Dhaval Shah@ 2021



Mapping Table

G F Com A B
A B C D E F G DP
E D Com C DP

Common Cathode

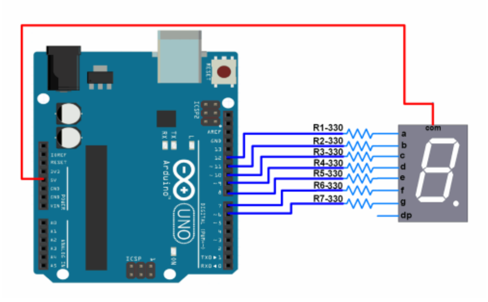
Digital	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0

Common Anode

Digital	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1

8/27/2021 Dhaval Shah@ 2021

Interfacing with Arduino



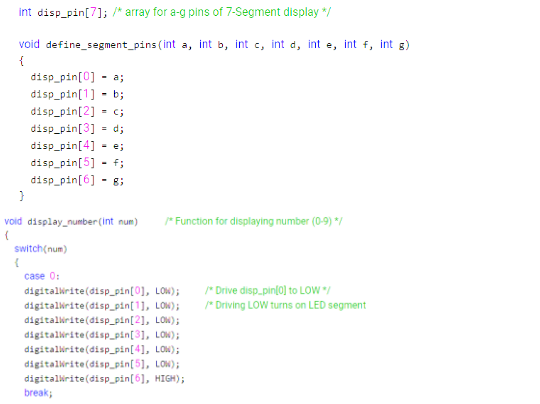
Interfacing with Arduino

- Write a code to display number in order or
- turn LEDs on and off in order to become familiar with how a seven-segment display functions.
  - Declare pins
  - Setup mode of each pin

```
pinMode(2,OUTPUT);
pinMode(3,OUTPUT);
pinMode(4,OUTPUT);
pinMode(5,OUTPUT);
pinMode(6,OUTPUT);
pinMode(7,OUTPUT);
 pinMode (8,OUTPUT);
```

Cont…

```
void loop()
{
  // loop to turn leds od seven seg ON

  for(int i=2;i<9;i  )
  {
    digitalWrite(i,HIGH);
    delay(600);
  }

  // loop to turn leds od seven seg OFF
  for(int i=2;i<9;i  )
  {
    digitalWrite(i,LOW);
    delay(600);
  }


  delay(1000);

}
```



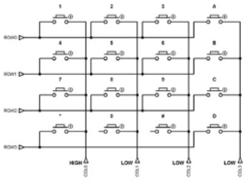
```
  int disp_pin[7]; /* array for a-g pins of 7-Segment display */

  void define_segment_pins(int a, int b, int c, int d, int e, int f, int g)
  {
    disp_pin[0] = a;
    disp_pin[1] = b;
    disp_pin[2] = c;
    disp_pin[3] = d;
    disp_pin[4] = e;
    disp_pin[5] = f;
    disp_pin[6] = g;
  }

void display_number(int num)       /* Function for displaying number (0-9) */
{
  switch(num)
  {
    case 0:
    digitalWrite(disp_pin[0], LOW);     /* Drive disp_pin[0] to LOW */
    digitalWrite(disp_pin[1], LOW);     /* Driving LOW turns on LED segment
    digitalWrite(disp_pin[2], LOW);
    digitalWrite(disp_pin[3], LOW);
    digitalWrite(disp_pin[4], LOW);
    digitalWrite(disp_pin[5], LOW);
    digitalWrite(disp_pin[6], HIGH);
    break;
```

HEX Keypad

- It is simply an arrangement of 16 push button switches in a 4X4 matrix form.
- Typical applications are : code locks, calculators, automation systems or simply any thing that requires a character or numeric input

HEX Keypad – Column Scanning

- The most common way to implement this by making a column pin high while making the rest of the columns low and do that in sequence
- column "0" is high and the rest of the columns are low. If you press the "1" button, row "0" will be high because by then it will be connected to column "0".
- scan row "0" using *digitalWrite()* while column "0" is high, this means the user pressed button "1".

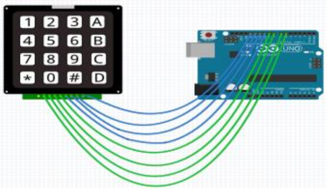


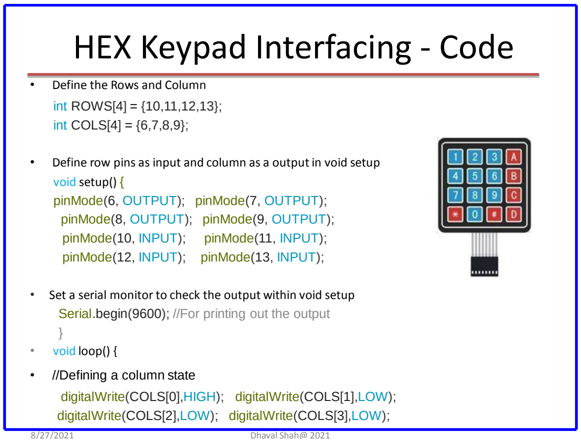
Cont…

When this button is pressed...	If...	Then...
1	Col0=high	Row0=high
2	Col1=high	Row0=high
3	Col2=high	Row0=high
4	Col0=high	Row1=high
5	Col1=high	Row1=high
6	Col2=high	Row1=high
7	Col0=high	Row2=high
8	Col1=high	Row2=high
9	Col2=high	Row2=high
0	Col1=high	Row3=high
*	Col0=high	Row3=high
#	Col2=high	Row3=high
A	Col3=high	Row0=high
B	Col3=high	Row1=high
C	Col3=high	Row2=high
D	Col3=high	Row3=high

HEX Keypad Interfacing

- The hex keypad will have 8 connection wires namely R1, R2, R3, R4 and C1, C2, C3, C4 representing the rows and columns respectively.





HEX Keypad Interfacing - Code

- Define the Rows and Column

```
int ROWS[4] = {10,11,12,13};
int COLS[4] = {6,7,8,9};
```

- Define row pins as input and column as a output in void setup

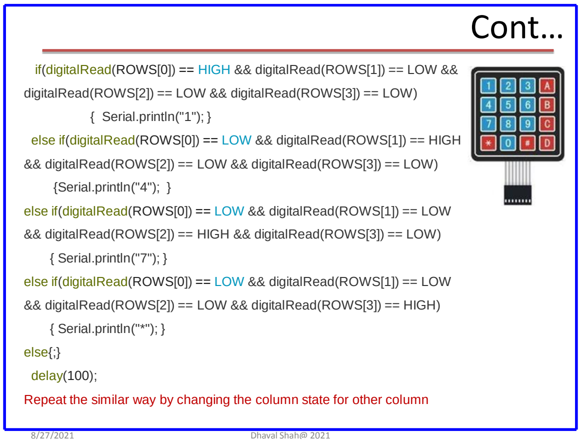
```
void setup() {
pinMode(6, OUTPUT);   pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);   pinMode(9, OUTPUT);
  pinMode(10, INPUT);     pinMode(11, INPUT);
  pinMode(12, INPUT);    pinMode(13, INPUT);
```

- Set a serial monitor to check the output within void setup

```
  Serial.begin(9600); //For printing out the output
  }
```

- void loop() {
- //Defining a column state

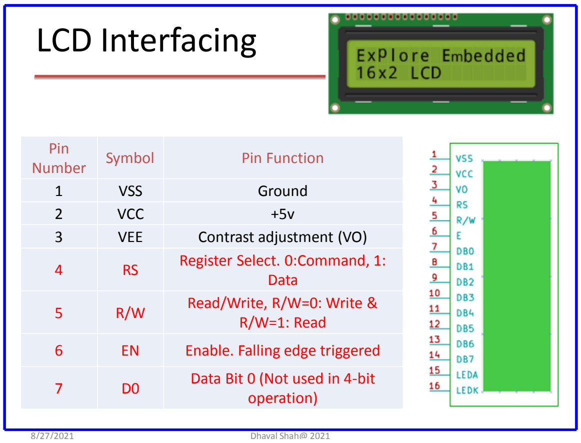
```
  digitalWrite(COLS[0],HIGH);   digitalWrite(COLS[1],LOW);
 digitalWrite(COLS[2],LOW);   digitalWrite(COLS[3],LOW);
```



Cont…

```
  if(digitalRead(ROWS[0]) == HIGH && digitalRead(ROWS[1]) == LOW &&
digitalRead(ROWS[2]) == LOW && digitalRead(ROWS[3]) == LOW)
              {  Serial.println("1"); }
 else if(digitalRead(ROWS[0]) == LOW && digitalRead(ROWS[1]) == HIGH
&& digitalRead(ROWS[2]) == LOW && digitalRead(ROWS[3]) == LOW)
       {Serial.println("4");  }
else if(digitalRead(ROWS[0]) == LOW && digitalRead(ROWS[1]) == LOW
&& digitalRead(ROWS[2]) == HIGH && digitalRead(ROWS[3]) == LOW)
      { Serial.println("7"); }
else if(digitalRead(ROWS[0]) == LOW && digitalRead(ROWS[1]) == LOW
&& digitalRead(ROWS[2]) == LOW && digitalRead(ROWS[3]) == HIGH)
      { Serial.println("*"); }
else{;}
 delay(100);
```

Repeat the similar way by changing the column state for other column

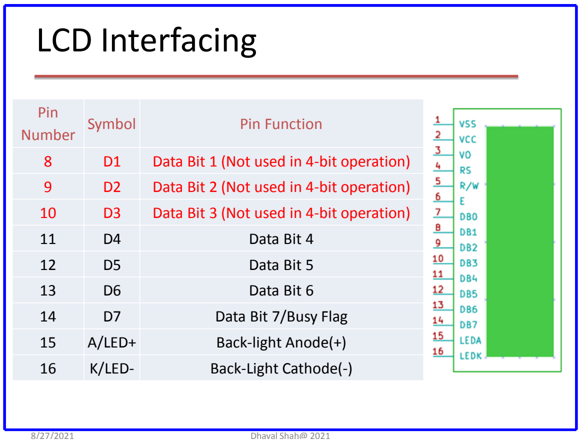


LCD Interfacing

ExPlore Embedded
16x2 LCD

Pin Number	Symbol	Pin Function
1	VSS	Ground
2	VCC	+5v
3	VEE	Contrast adjustment (VO)
4	RS	Register Select. 0:Command, 1: Data
5	R/W	Read/Write, R/W=0: Write & R/W=1: Read
6	EN	Enable. Falling edge triggered
7	D0	Data Bit 0 (Not used in 4-bit operation)

1 VSS
2 VCC
3 VO
4 RS
5 R/W
6 E
7 DB0
8 DB1
9 DB2
10 DB3
11 DB4
12 DB5
13 DB6
14 DB7
15 LEDA
16 LEDK

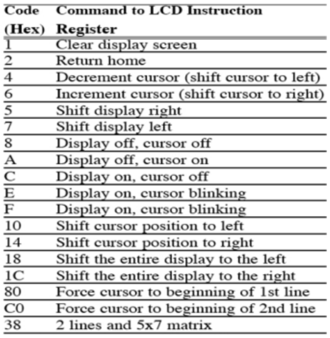


LCD Interfacing

Pin Number	Symbol	Pin Function
8	D1	Data Bit 1 (Not used in 4-bit operation)
9	D2	Data Bit 2 (Not used in 4-bit operation)
10	D3	Data Bit 3 (Not used in 4-bit operation)
11	D4	Data Bit 4
12	D5	Data Bit 5
13	D6	Data Bit 6
14	D7	Data Bit 7/Busy Flag
15	A/LED+	Back-light Anode(+)
16	K/LED-	Back-Light Cathode(-)

1 VSS
2 VCC
3 VO
4 RS
5 R/W
6 E
7 DB0
8 DB1
9 DB2
10 DB3
11 DB4
12 DB5
13 DB6
14 DB7
15 LEDA
16 LEDK

LCD Commands



Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

LCD functions in Arduino

- LiquidCrystal()
  - Creates a variable of type LiquidCrystal.
  - The display can be controlled using 4 or 8 data lines.
  - Syntax
    - LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
    - LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
  - Example
    - LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);

Cont…

- begin()
  - Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display
  - Syntax
    - lcd.begin(cols, rows)
  - Example
    - lcd.begin(16, 2);
- clear()
  - Clears the LCD screen and positions the cursor in the upper-left corner
  - Syntax
    - lcd.clear();

Cont…

- home()
  - Positions the cursor in the upper-left of the LCD.
  - Can use lcd.clrear(), as it does same function
- setCursor()
  - Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed.
  - Syntax
    - lcd.setCursor(col, row)
  - Example
    - lcd.setCursor(0, 1)

Cont…

- write()
  - Write a character to the LCD.
- print()
  - Prints text to the LCD.
- cursor()
  - Display the LCD cursor: an underscore (line) at the position to which the next character will be written.
- noCursor()
  - Hides the LCD cursor.
- blink()
  - Display the blinking LCD cursor.

Cont…

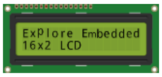
- display()
  - Turns on the LCD display, after it's been turned off with noDisplay(). This will restore the text (and cursor) that was on the display.
- scrollDisplayLeft()/scrollDisplayRight()
  - Scrolls the contents of the display (text and cursor) one space to the left/right.
- autoscroll()
  - Turns on automatic scrolling of the LCD. This causes each character output to the display to push previous characters over by one space.
- leftToRight()/rightToLeft()
  - Set the direction for text written to the LCD to left-to-right/ right-to-left

Steps to interface LCD with Arduino

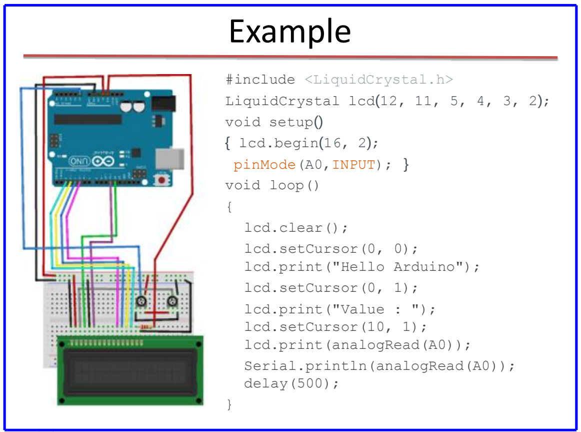
- Void Setup part
- Step: 1
  - Include the library LiquidCrystal.h
  - Syntax: #include <LiquidCrystal.h>
- Step: 2
  - Initialize the library with the number of the interface pins. With the function "LiquidCrystal lcd()".
  - Syntax: LiquidCrystal lcd(RS, E, D4, D5, D6, D7);
  - LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
- Step: 3
  - Set the number of columns and rows by the function "lcd.begin(16, 2)“
  - Define other pins, if any

Cont…

- Void loop part
- Step: 4
  - clear the display using the function "lcd.clear()"
- Step: 5
  - set the cursor to that particular point (starting point) by the function "lcd.setCursor()
  - Syntax: lcd.setCursor(0, 0);



- Step: 6
  - Print the character/data by the function "lcd.print(“Hello Arduino”);
  - This will print on first row as per the current setting



Example

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup()
{ lcd.begin(16, 2);
 pinMode(A0,INPUT); }
void loop()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Hello Arduino");
  lcd.setCursor(0, 1);
  lcd.print("Value : ");
  lcd.setCursor(10, 1);
  lcd.print(analogRead(A0));
  Serial.println(analogRead(A0));
  delay(500);
}
```

Serial Communication

Prof. Dhaval Shah

Outline

- Introduction
- Asynchronous Serial Communication
- Data Framing
- Serial Port Programming
- SPI Protocol
- I2C Protocol
- LCD interfacing using I2C

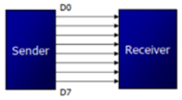
Acknowledgement

– Muhammad Mazidi, The 8051 Microcontroller and Embedded Systems using Assembly and C, Pearson Edu..

Introduction

- Two ways to transmit the data
  - Parallel
    - Often 8 or more lines (wire conductors) are used to transfer data from one device to a device
    - Preferable for the devices that is only a few feet away
  - Serial
    - One bit at a time is transmitted
    - For longer distance (many meters away)

Parallel Transfer



Serial Transfer

Sender
Receiver

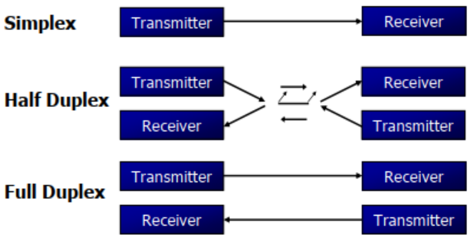
Serial Communication

- At the transmitting end, the byte of data must be converted to serial bits using parallel-in-serial-out shift register
- At the receiving end, there is a serial-in-parallel-out shift register to receive the serial data and pack them into byte.
- Digital signal can be transmitted without modulation for a short distance.
- If data is to be transferred on the telephone line, it must be converted from 0s and 1s to audio tones
  - This conversion is performed by a device called a modem, "Modulator/demodulator"

Types of Serial Communication

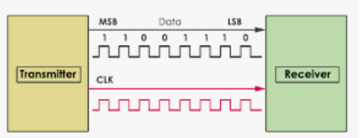
- Two types
  - Synchronous Serial Communication
    - Transfer a block of data at a time
  - Asynchronous Serial Communication
    - a single byte at a time
- Either of these method can be used by developing a software/code but it is tedious and long
- Special IC chips made by many manufacturers for serial communications
  - UART (Universal Asynchronous Receiver Transmitter)
  - USART (Universal Synchronous Asynchronous Receiver Transmitter)

Data Transmission Mode



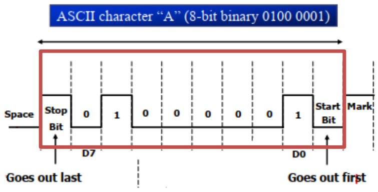
Synchronous Serial Communication

- Data is sent in a continuous stream at constant rate
- Requires that the clock for the synchronization between transmitter and receiver
- No additional bits require for communication setup
- Permits more information to be passed over a circuit per unit time



Asynchronous Serial Communication

- Preferred for character-oriented transmission
- Each character is placed in between start and stop bits, it is called as a framing
- The start bit is always a 0 (low) and the stop bit(s) is 1 (high)



ASCII character "A" (8-bit binary 0100 0001)

Data Transfer Rate

- The rate of data transfer in serial data communication is stated in bps (bits per second)
- Another widely used terminology for bps is baud rate
- Baud Rate
  - It is modem terminology and is defined as the number of signal changes per second
- In modems, there are occasions when a single change of signal transfers several bits of data
- As far as the conductor wire is concerned, the baud rate and bps are the same, and we use the terms interchangeably

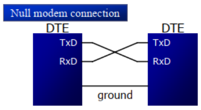
Cont…

- The data transfer rate of given computer system depends on communication ports incorporated into that system

- Example
  - IBM PC/XT could transfer data at the rate of 100 to 9600 bps
  - Pentium-based PCs transfer data at rates as high as 56K bps
  - In asynchronous serial data communication, the baud rate is limited to 100K bps

Data Communication Classification

- Current terminology classifies data communication equipment as
  - DTE (Data Terminal Equipment)
    - refers to terminal and computers that send and receive data
  - DCE (Data Communication Equipment)
    - refers to communication equipment, such as modems
- The simplest connection between a PC and controller requires a minimum of three pins, TxD, RxD, and GND.



Null modem connection