TABLE OF CONTENT

I. INTRODUCTION OF INFRARED RECEIVER THEORY - COMPONENT STRUCTURE

1. Introduction

In today's interconnected world, where convenience is paramount, the humble Infrared (IR) remote control stands as an unsung hero, quietly orchestrating our interactions with various electronic devices. From televisions to air conditioners, from gaming consoles to smart home appliances, IR remote controls have become ubiquitous in modern households, offering a seamless interface between users and their electronic gadgets.

As our topic is " IR Remote control" so at first, we want to define what is the IR remote. At its core, an IR remote control functions by emitting infrared light pulses, which are then received and interpreted by an IR receiver within the target device. These pulses are encoded with specific commands that trigger various functions, such as changing channels, adjusting volume, or powering the device on or off. The beauty of IR remote controls lies in their simplicity and effectiveness, offering users a convenient way to interact with their devices from a distance.

2. Concept Of Infrared Light

Infrared light (infrared rays) is light that cannot be seen with the naked eye, with a wavelength of about 0.86µm to 0.98µm. Infrared rays travel at the same speed as light.

Infrared rays are easily absorbed and have poor penetrating ability. In infrared remote control, the infrared beam emitted is narrow and directional, so when received, it must be in the right direction.

To be able to transmit infrared rays well, it is necessary to avoid interference pulses. It is necessary to use stable transmit and receive codes to determine whether it is a transmitted pulse or noise. The best working frequency is from 30 KHz to 60 KHz, but usually use around 36 KHz.

3. Block Diagram

Initialization -> Activative: This block represents the process of initializing the IR receiver and activating its functionality. It ensures that the receiver is ready to receive signals from the remote control.

Button Pressed: This block signifies the user interaction with the remote control. When a button on the remote control is pressed, it sends a corresponding signal.

Signal Transmission : Upon pressing a button on the remote control, a signal is generated indicating the specific command associated with the pressed button.

Receive signal: The signal generated by the remote control is transmitted through the air as infrared radiation. The IR receiver captures this transmitted signal.

Decode Signal: The received signal is then decoded by the IR receiver to extract the command embedded within it. This decoding process translates the infrared signal into a usable command.

Device Status Check: Once the command is decoded, the IR receiver checks the device's current status to determine the appropriate action to take. For example, if the command is to increase the volume, the receiver checks if the device is currently on and adjusts the volume accordingly.

4. Programming environment and load the program

4.1. Microchip Studio and PROGISP

Microchip Studio, formerly known as Atmel Studio, is an integrated development environment (IDE) designed for Microchip Technology's AVR and ARM microcontrollers. Built with a range of powerful tools and features, Microchip Studio provides a comprehensive platform for embedded application development, from writing source code to debugging and analysis.

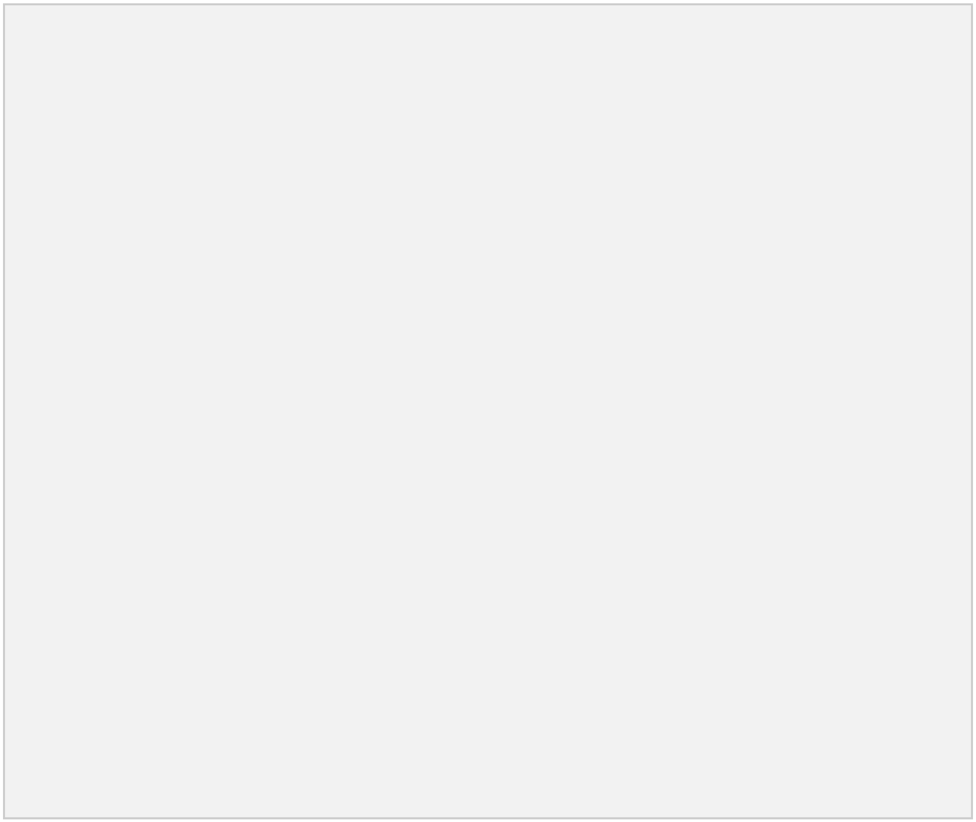
Step 1: Create Project with Microchip Studio

Step 2: Choose the device

Step 3: Write the code, input library, and build project

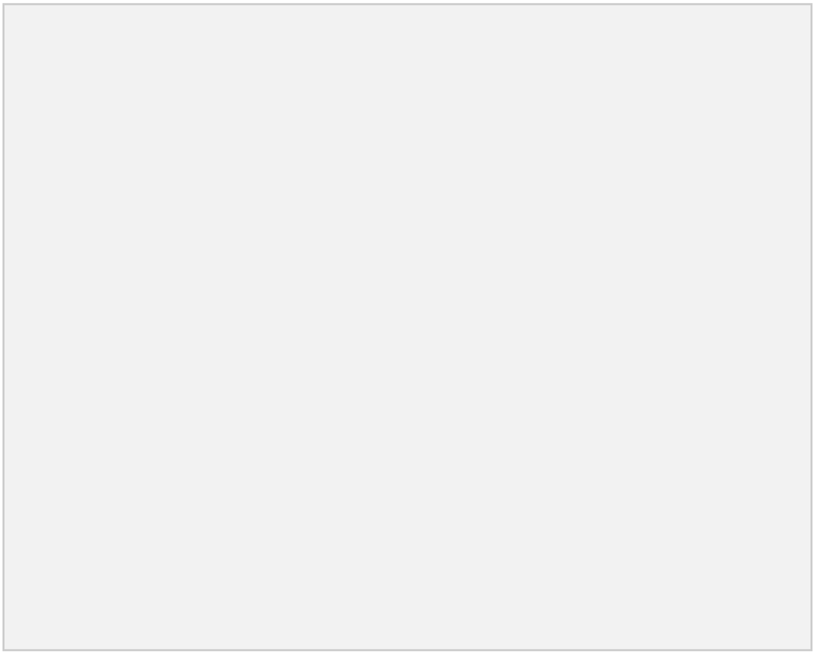
Step 4: Run the Progisp loader program

Detailed instructions here::
https://giaiphapchung.vn/huong-dan-dung-phan-mem-progisp-nap-cho-avr-va-isp



Microcontroller connection diagram

After that, program to receive USBISP charging circuit (Choose HEX file after build in Step 3) and select chip type (ATmega328P)



Step 5: Configure "Fuse And Lock Config" of ATmega328P (This part is very important for AVR, if not clear it will lead to Fuse Bits => Chip Error)

- Default: Return to the original configuration.
- Read: Read the Quartz frequency from the chip into the program.
- WTDON: WTDON is always set to 1 when configuring "Fuse And Lock Config" for ATmega328P (WTDON = 1: Prevent chip reset)
- Quartz crystal configuration
- Write: Configure the Quartz frequency from the program to the chip

Step 6: Load the FLASH, EEPROM program.

Step 7: Click Auto: to load the program you just selected.

Step 8: When the words: Erase,Write Flash,Verify Flash,Verify Eeprom,Successfully done appear, it means you have finished loading the program.

4.2. Arduino IDE

Another easier way to load programs is to use the Arduino IDE.
Arduino IDE is an integrated development environment (IDE) designed to easily develop and load code for Arduino boards and AVR-based microcontrollers. The ATmega328P is one of the most popular

microcontrollers used on Arduino boards such as the Arduino Uno. To load code for ATmega328P using Arduino IDE, you need to prepare the following steps:

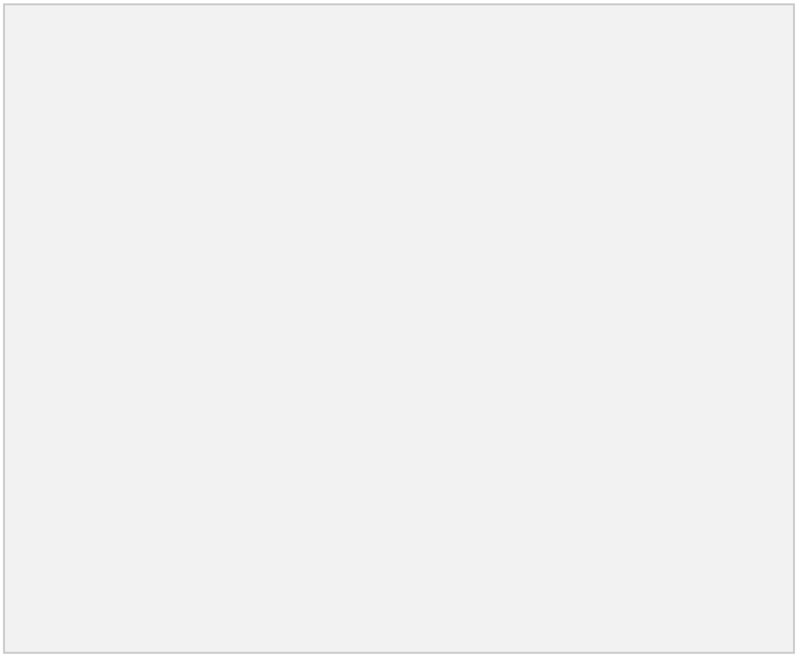
Load directly on Arduino UNO

Step 1: Prepare the Environment: First, you need to install Arduino IDE on your computer. Arduino IDE is available for many operating systems and can be downloaded for free from the official Arduino website.

Step 2: Connecting the Arduino Board: Connect the Arduino board containing the ATmega328P to the computer using the USB port.

Step 3: Select Board and Port: In the Arduino IDE, select the board that corresponds to the one you are using via the "Tools > Board" menu. Then, select the COM port to which the board is connected via the "Tools > Port" menu. After that, select the Programmer. Select AVR ISP to do with AVR code.

Step 4: Write and import library:
Write your code in the Arduino IDE editor and import the library.

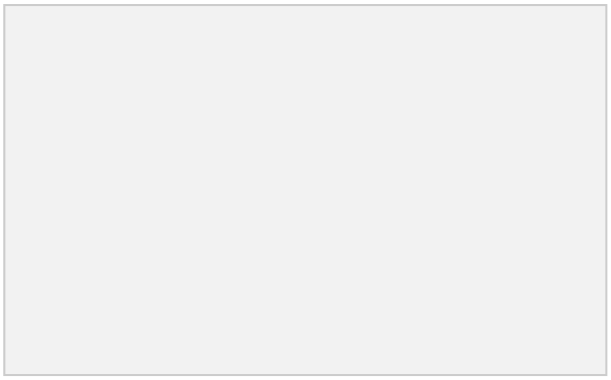
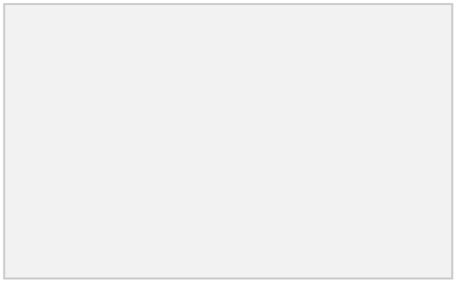


Step 5: After finishing writing, press the "Upload" button to compile and load the code into the ATmega328P microcontroller.

Step 6: Remove the ATmega328P chip on the UNO and install it on the breadboard after loading is complete

Loaded via USB UART Switch Circuit CP2102:

Step 1: Suppose the ATmega328P, after loading the bootloader and plugging in other components, is an Arduino. Plug in the pins below according to the model to load the code.

Step 2: Perform the same steps as Load directly on Arduino UNO

II. REQUIREMENT

1. Overview

Internal Name	IR Remote control
External Name	IRC
MSRP (Solo)	$20 to $100, VND 400,000-VND 2,000,000
Channel	Online Only
Launch Target	2024

An infrared (IR) remote control project typically involves designing and building a device capable of sending and receiving IR signals to control various electronic
appliances remotely. IR remote controls simplify the operation of electronic devices from a distance, such as TVs, home entertainment systems, appliances, lighting, and even car audio systems. They offer convenience and accessibility, enhancing user
experience in various settings.

Priority Description

P0: Must Have Requirement i.e. Product will not launch without it
P1: Not a requirement for launch but is needed 3-6 months post-launch

Electronic Components	Product Requirement	Specifications	Quantity	Price
ATmega328P	microcontroller	Core processor: AVR Core size: 8bit Operating voltage: 1,8-5,5V	2	120.000 đ
9V Battery Base Wire	Get electricity from 9V battery		2	6.000 đ
Battery	Source	9V	2	16.000 đ
IC L7805	Voltage regulator IC L7805CV	Minimum input voltage: 7VDC Output voltage: 5VDC	2	5.000 đ
Diode 1N4007	Use rectifier current		2	3.500 đ

Infrared LED	Used in infrared remote control.	Size: 5x5x3.6mm Operating voltage: 1.2 -1.6 VDC.	1	4.500 đ
Infrared Receiver Module	receive infrared through the infrared receiver	Size: 6.4 x 7.4 x 5.1mm Operating voltage: 2,7 ~ 5,5 VDC Frequency: 37,9 kHz	1	10.000 đ
16 MHz Crystal	Control the buses and processing clock		2	4.000 đ
Capacitor		22pF	-	-
		330nF	-	-
Resistor		10k	-	-
		1k	-	-
		200	-	-
keypad			1	12.000 đ
LCD			1	50.000 đ
Module i2c LCD			1	18.000 đ
Wire			-	-
Breadboard			2	50.000 đ
PCB			2	300.000 đ

2. Product Requirements

Industrial Design

#	Feature/ Characteristic	Product Requirements	Priority	Technical/ Engineering Specifications	Comments
I.1	Compatibility	Ensure compatibility with a wide range of electronic devices	P0		

I.2	Button Layout	Design an intuitive button layout for easy navigation and operation.	P0		
I.3	Ergonomics	Comfortable handling and ease of use over prolonged periods.	P0		
I.4	Button	Allow to change the status ò device	P0		
I.5	Programming Capability	Offer the ability to program custom functions or macros for advanced users	P1		
I.6	Branding	Logos should be tastefully integrated into the design without detracting from the overall aesthetic appeal	P0		
I.7	Placement	Small and convenient, can be carried easily	P0		
I.8	User Interface	LCD displays to enhance user experience and provide feedback	P0		

Display Screen

#	Feature/ Characteristic	Product Requirements	Priority	Technical/ Engineering Specifications	Comments
II.1	LCD or OLED Display	Utilize a low-power LCD or OLED display to provide visual feedback to the user	P0		

II.2	Menu Navigation	Design a menu system that is displayed on the screen, allowing users to navigate through different functions and settings easily. This could include options for controlling various devices, setting up macros, or adjusting settings.	P0		
II.3	Feedback and Confirmation	Use the display to provide feedback and confirmation when buttons are pressed, ensuring that the user's commands are registered correctly.	P0		
II.4	Status Indicators	Display status indicators on the screen to alert users to important information, such as low battery levels or connectivity issues.	P1		
II.5	Integration with Device Information	Integrate the display with device information databases to provide additional context or details about the devices being controlled, such as channel listings for TVs or track information for audio systems.	P1		

Connectivity

#	Feature/ Characteristic	Product Requirements	Priority	Technical/ Engineering Specifications	Comments

III.1	Infrared (IR) Communication	The primary method of connectivity for an IR remote control is through infrared signals. The remote control emits IR signals that are received by IR receivers on the target devices, allowing for wireless control.	P0		
III.2	Range	The range is sufficient to cover the desired distance between the user and the target devices	P0		
III.3	Interference	Minimize potential sources of interference that could affect IR communication, such as other IR devices or sunlight.	P1		
III.4	Signal Strength	The strength of the IR signals emitted by the remote control can affect connectivity. Ensure that the signals are strong enough to reliably control the target devices without interference or dropout.	P0		

Power

#	Feature/ Characteristic	Product Requirements	Priority	Technical/ Engineering Specifications	Comments
IV.1	Power	Using disposable batteries (e.g., AAA, AA) or rechargeable batteries	P0		
IV.2	Battery Life	Optimize power consumption to maximize battery life and minimize the frequency of battery replacements or	P1		

		recharging…)			
IV.3	Backup Power	Should have secondary batteries or capacitors, to retain settings or memory during battery replacement or power interruptions	P1		

Durability

#	Feature/Characteristic	Product Requirements	Priority	Technical/Engineering Specifications	Comments
V.1	Material	High-quality, durable materials for the construction of the remote control, such as ABS plastic or aluminum alloy	P0		
V.2	Weather Resistant	The device is designed to function reliably in wireless mode with an IPX6 rating. Operation under higher ratings has not been tested yet.	P0		
V.2	Drop Test	Our remote control boats buttons and switches designed for a minimum of 10,000 press cycles, promising longevity and reliability under frequent use.	P0		
V.3	Impact Resistance	Our remote control can endure impact from accidental drops of up to 1 meter onto hard surfaces, maintaining its functionality and integrity.	P0		
V.4	Temperature Range	Ranging from - 10°C to 50°C, our remote control consistently delivers reliable performance, regardless of environmental conditions	P0		
V.5	Humidity Tolerance	Designed to withstand humidity levels of up to 90% RH	P1		

Packaging

#	Feature / Characteristic	Product Requirements	Priority	Technical/ Engineering Specifications	Comments
VI.1	Material	Crafted from durable and eco-friendly materials, such as recyclable cardboard or biodegradable plastics	P0		
VI.2	Branding	Design incorporates the brand identity and product messaging cohesively, utilizing vibrant colors, clear imagery, and concise text to communicate the product's features and benefits effectively.	P0		
VI.3	Sustainability Commitment	Sustainability goals, our packaging is fully recyclable or biodegradable	P0		
VI.4	Multi-Lingual Support	Including Vietnamese language and imported countries's language	P0		

Out of Box Experience (OOBE)

#	Feature/Characteristic	Product Requirements	Priority	Technical/Engineering Specifications	Comments
VII.1	First Impressions	The packaging is designed to make a strong first impression, with eye- catching branding, vibrant colors	P0		
VII.2	Easy Access:	Easy-to-access arrangement that allows them to retrieve the remote control effortlessly, minimizing frustration	P0		

III. HARDWARE DESIGN AND THE OPERATIONS OF THE COMPONENT

1. ATmega328P

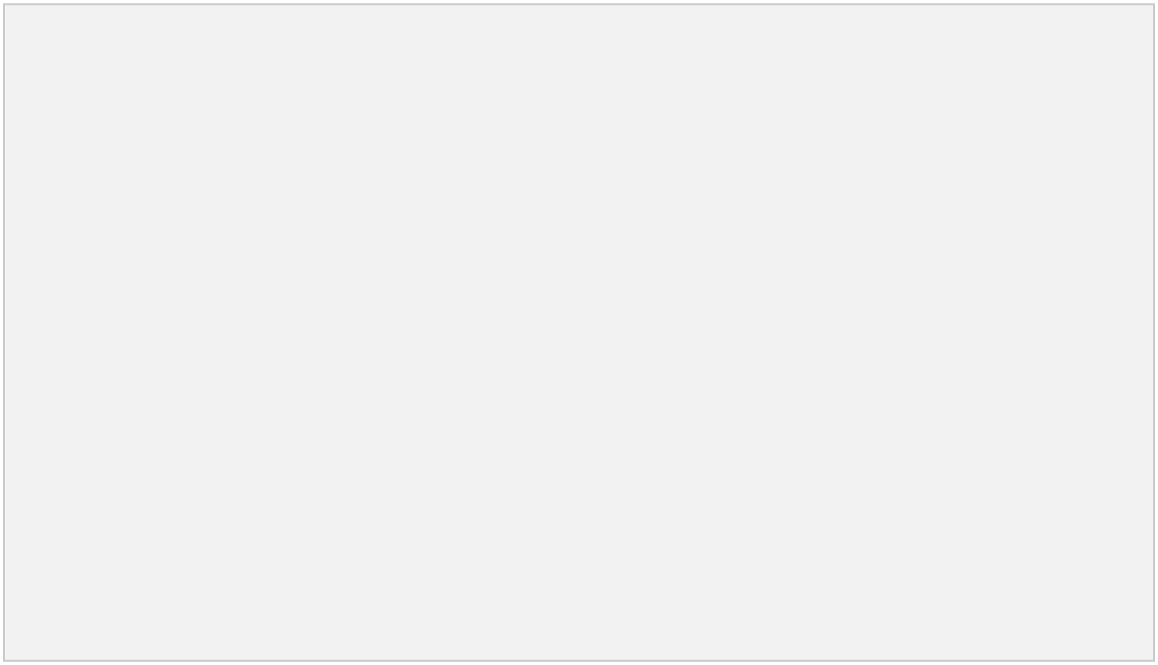


Figure: ATmega328P

Data sheet:
https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf

- Manufacturer: Microchip
- Manufacturer Part Number: ATMEGA328P-PU
- Voltage - Supply (Vcc/Vdd): 1.8~ 5.5VDC
- Description: IC MCU 8BIT 32KB FLASH 28DIP
- Lead Free Status / RoHS Status: Lead free / RoHS Compliant
- Packaging: Tube
- Core Processor: AVR
- Core Size: 8-Bit
- Speed: 20MHz
- Connectivity: I²C, SPI, UART/USART
- Peripherals: Brown-out Detect/Reset, POR, PWM, WDT
- Number of I /O: 23
- Program Memory Size: 32KB (16K x 16)
- Program Memory Type: FLASH
- EEPROM Size: 1K x 8
- RAM Size: 2K x 8
- Data Converters: A/D 6x10b

- Oscillator Type: Internal
- Operating Temperature: -40°C ~ 85°C

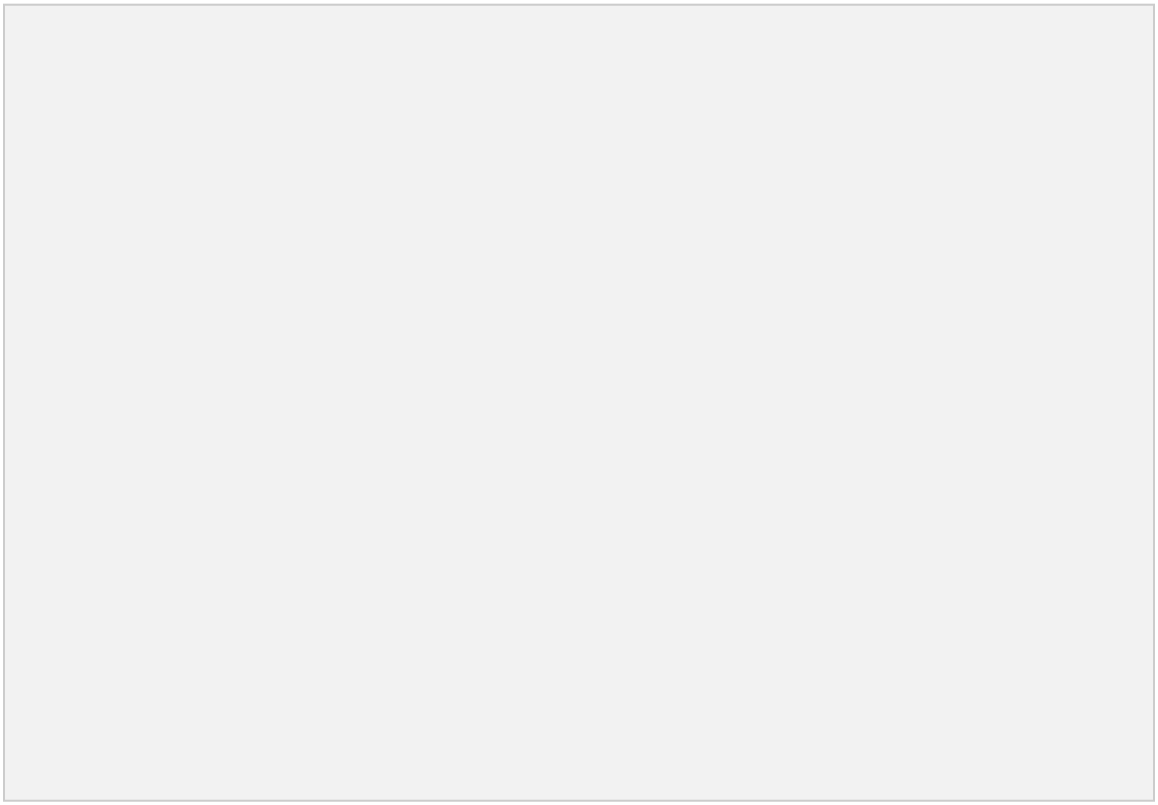


Figure: Arduino UNO with ATmega328P chip and equivalent I/O pin

Port Pin	Arduino Pin	Alternate Functions
PB7		XTAL2 (Chip Clock Oscillator pin 2) Used as clock pin for Crystal Oscillator or Low-frequency Crystal Oscillator. When used as a clock pin, the pin can not be used as an I/O pin. TOSC2 (Timer Oscillator pin 2) In this mode, a crystal Oscillator is connected to this pin and used for asynchronous clocking of Timer/Counter2 and the pin cannot be used as an I/O pin. PCINT7 (Pin Change Interrupt 7) The PB7 pin can serve as an external interrupt source.

PB6		XTAL1 (Chip Clock Oscillator pin 1 or External clock input) Used for all chip clock sources except internal calibrated RC Oscillator. When used as a clock pin, the pin can not be used as an I/O pin. TOSC1 (Timer Oscillator pin 1) In this mode, a crystal Oscillator is connected to this pin and used for asynchronous clocking of Timer/Counter1 and the pin cannot be used as an I/O pin. PCINT6 (Pin Change Interrupt 6) The PB6 pin can serve as an external interrupt source.
PB5	**D13**	SCK (SPI Bus Master clock Input) When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB5. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB5. PCINT5 (Pin Change Interrupt 5) The PB5 pin can serve as an external interrupt source.
PB4	**D12**	MISO (SPI Bus Master Input/Slave Output) When the SPI is enabled as a Master, this pin is configured as an input regardless of the setting of DDB4. When the SPI is enabled as a Slave, the data direction of this pin is controlled by DDB4. PCINT4 (Pin Change Interrupt 4) The PB4 pin can serve as an external interrupt source.
PB3	**D11**	MOSI (SPI Bus Master Output/Slave Input) When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB3. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB3. OC2A (Timer/Counter2 Output Compare Match A Output) This pin can serve as an external output for the Timer/Counter2 in Compare Match mode or PWM mode. PCINT3 (Pin Change Interrupt 3) The PB3 pin can serve as an external interrupt source.

PB2	**D10**	SS (SPI Bus Master Slave select) When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB2. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB2. OC1B (Timer/Counter1 Output Compare Match B Output) This pin can serve as an external output for the Timer/Counter1 in Compare Match mode or PWM mode. PCINT2 (Pin Change Interrupt 2) The PB2 pin can serve as an external interrupt source.
PB1	**D9**	OC1A (Timer/Counter1 Output Compare Match A Output) This pin can serve as an external output for the Timer/Counter1 in Compare Match mode or PWM mode. PCINT1 (Pin Change Interrupt 1) The PB1 pin can serve as an external interrupt source.
PB0	**D8**	ICP1 (Timer/Counter1 Input Capture Input) The PB0 pin can act as an Input Capture Pin for Timer/Counter1. CLKO (Divided System Clock Output) The divided system clock can be output on the PB0 pin. It will also be output during reset. PCINT0 (Pin Change Interrupt 0) The PB0 pin can serve as an external interrupt source.
PC6	**RESET**	RESET (Reset pin) When the RSTDISBL Fuse is programmed, this pin functions as a normal I/O pin, and ATmega328P will have to rely on Power-on Reset and Brown-out Reset as its reset sources. When the RSTDISBL Fuse is unprogrammed, the reset circuitry is connected to the pin, and the pin can not be used as an I/O pin. PCINT14 (Pin Change Interrupt 14) The PC6 pin can serve as an external interrupt source.

PC5	**A5 / D19**	ADC5 (ADC Input Channel 5) PC5 can also be used as ADC input Channel 5. Note that ADC input channel 5 uses digital power (VCC). SCL (2-wire Serial Bus Clock Line) When the 2-wire Serial Interface (I2C) is enabled the PC5 pin is disconnected from the port and becomes the Serial Clock I/O pin for the 2-wire Serial Interface. PCINT13 (Pin Change Interrupt 13) The PC5 pin can serve as an external interrupt source.
PC4	**A4 / D18**	ADC4 (ADC Input Channel 4) PC5 can also be used as ADC input Channel 4. Note that ADC input channel 5 uses digital power (VCC). SDA (2-wire Serial Bus Data Input/Output Line) When the 2-wire Serial Interface (I2C) is enabled the PC5 pin is disconnected from the port and becomes the Serial Data I/O pin for the 2-wire Serial Interface. PCINT12 (Pin Change Interrupt 12) The PC4 pin can serve as an external interrupt source.
PC3	**A3 / D17**	ADC3 (ADC Input Channel 3) PC3 can also be used as ADC input Channel 3. Note that ADC input channel 3 uses analog power (AVCC). PCINT11 (Pin Change Interrupt 11) The PC3 pin can serve as an external interrupt source.
PC2	**A2 / D16**	ADC2 (ADC Input Channel 2) PC2 can also be used as ADC input Channel 2. Note that ADC input channel 3 uses analog power (AVCC). PCINT10 (Pin Change Interrupt 10) The PC2 pin can serve as an external interrupt source.
PC1	**A1 / D15**	ADC1 (ADC Input Channel 1) PC1 can also be used as ADC input Channel 1. Note that ADC input channel 3 uses analog power (AVCC). PCINT9 (Pin Change Interrupt 9) The PC1 pin can serve as an external interrupt source.
PC0	**A0 / D14**	ADC0 (ADC Input Channel 0) PC0 can also be used as ADC input Channel 0. Note that ADC input channel 3 uses analog power (AVCC). PCINT8 (Pin Change Interrupt 8) The PC0 pin can serve as an external interrupt source.

PD7	**D7**	AIN1 (Analog Comparator Negative Input) Analog Comparator Negative Input Pin. Configure internal pull-up switched off and port pin as an input. PCINT23 (Pin Change Interrupt 23) The PD7 pin can serve as an external interrupt source.
PD6	**D6**	AIN0 (Analog Comparator Positive Input) Analog Comparator Positive Input Pin. Configure internal pull-up switched off and port pin as an input. OC0A (Timer/Counter0 Output Compare Match A Output) This pin can serve as an external output for the Timer/Counter0 in Compare Match mode or PWM mode. PCINT22 (Pin Change Interrupt 22) The PD6 pin can serve as an external interrupt source.
PD5	**D5**	T1 (Timer/Counter 1 External Counter Input) This pin can serve as an external output for the Timer/Counter1 in Compare Match mode or PWM mode. OC0B (Timer/Counter0 Output Compare Match B Output) This pin can serve as an external output for the Timer/Counter0 in Compare Match mode or PWM mode. PCINT21 (Pin Change Interrupt 21) The PD5 pin can serve as an external interrupt source.
PD4	**D4**	XCK (USART External Clock Input/Output) USART external clock. T0 (Timer/Counter 0 External Counter Input) This pin can serve as an external output for the Timer/Counter0 in Compare Match mode or PWM mode. PCINT20 (Pin Change Interrupt 20) The PD4 pin can serve as an external interrupt source.
PD3	**D3**	INT1 (External Interrupt 1 Input) The PD3 pin can serve as an external interrupt source OC2B (Timer/Counter2 Output Compare Match B Output) This pin can serve as an external output for the Timer/Counter2 in Compare Match mode or PWM mode. PCINT19 (Pin Change Interrupt 19) The PD3 pin can serve as an external interrupt source.
PD2	**D2**	INT0 (External Interrupt 0 Input) The PD2 pin can serve as an external interrupt source. PCINT18 (Pin Change Interrupt 18)

		The PD2 pin can serve as an external interrupt source.
PD1	**D1**	TXD (USART Output Pin) When the USART Transmitter is enabled, this pin is configured as an output regardless of the value of DDD1. PCINT17 (Pin Change Interrupt 17) The PD1 pin can serve as an external interrupt source.
PD0	**D0**	RXD (USART Input Pin) When the USART Receiver is enabled this pin is configured as an input regardless of the value of DDD0. PCINT16 (Pin Change Interrupt 16) The PD0 pin can serve as an external interrupt source.

2. 9V - 5V Regulator

In the simulation circuit, I supply 5V power using a 9v to 5v regulator circuit. Based on the operating principle of IC 7805, it is possible to convert the input voltage: 7V - 18V DC to a fixed voltage at 5V.

The 1N4007 diode is used to prevent reverse current flow in an electrical circuit, by only allowing current to flow in a specific direction. When the current flows in the opposite direction, this diode cuts the current and stops it.
In circuits, capacitors can be used to maintain a stable voltage for short periods of time when current suddenly changes, helping to prevent unwanted fluctuations.

Figure: Internal structure of IC 7805

At the heart of IC 7805 is a transistor (Q16) that can control the current between input and output. They can also control the output voltage. The bandgap reference (yellow) keeps the voltage stable. They take the scaled output voltage as input (Q1 and Q6) and also provide error signals (to Q7) to indicate if too high or too low a voltage level is detected. The main task of the bandgap is to provide a stable and accurate reference, even in cases where the chip's temperature changes.

The error signal from the bandgap reference will be amplified by the error amplifier (orange). This amplified signal will drive the output transistor through Q15. They will proceed to close the negative feedback loop that controls the output voltage.

The starting circuit (green) helps provide initial current to the bandgap circuit. The purple circuit provides protection against overheating (Q13), excessive input voltage (Q19), excessive output current (Q14). These circuits will reduce the output current or turn off the regulator, protecting them from damage in the event of a fault. The voltage divider (blue) reduces the voltage on the output pin for use thanks to the bandgap reference.

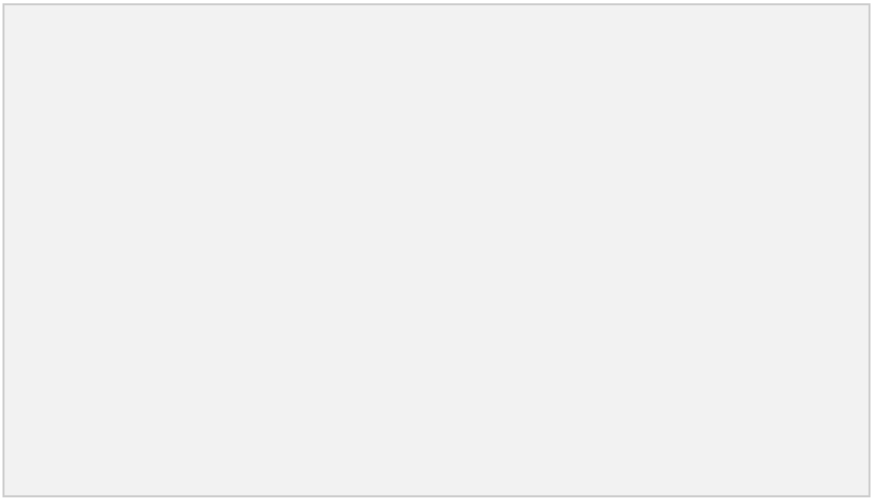


Figure: 9V - 5V Regulator

In fact, we has implemented this circuit and got approximate results

3. PCF8574 + LCD1602

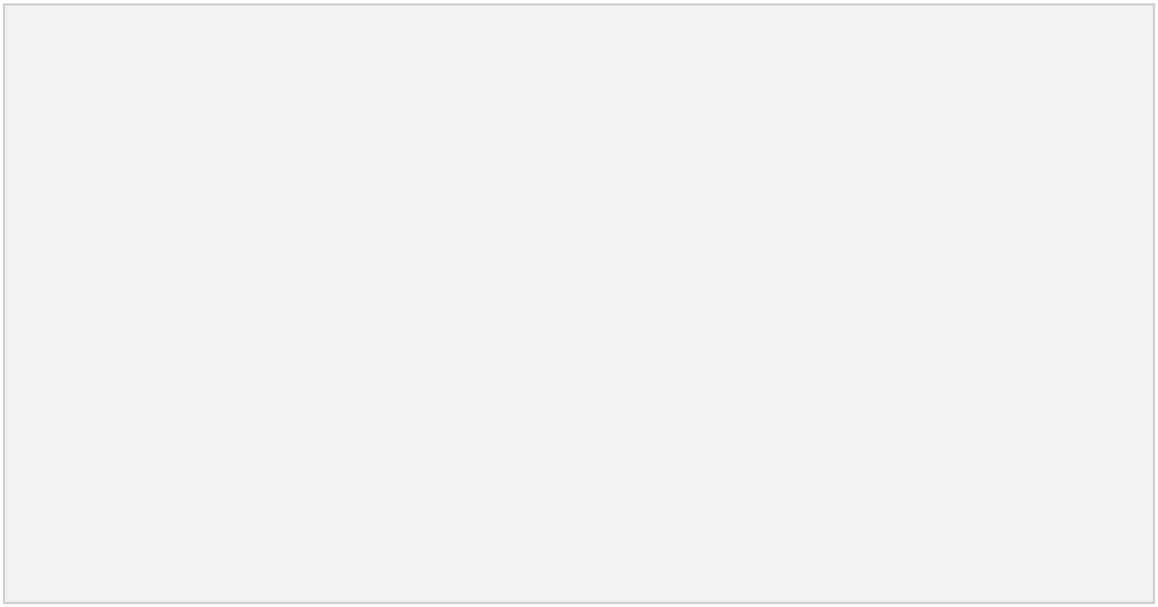
PCF8574	**LCD1602**
Operating voltage: 2.5~6VDC Communication: I2C, address setting by DIP Switch. Number of expansion interface pins: 8 I/O. Frequency: 100kHz maximum	Operating voltage is 5 V. Dimensions: 80 x 36 x 12.5 mm There is a backlight LED, which can use a variable resistor or PWM to adjust the brightne to use less power. Can be controlled with 6 signal wires

No	Symbol	Describe	Value
1	VSS	GND	0V
2	VCC		5V
3	V0	Contrast	
4	RS	Select register	RS=0 (low) selects the command register RS=1 (high) selects data register
5	R/W	Select read/write data register	R/W=0 write register R/W=1 read register
6	E	Enable	
7	DB0	Data transmission pin	8 bit: DB0-DB7
8	DB1		
9	DB2		
10	DB3		
11	DB4		

12	DB5		
13	DB6		
14	DB7		
15	A	Background led anode	0V to 5V
16	K	Background led cathode	0V

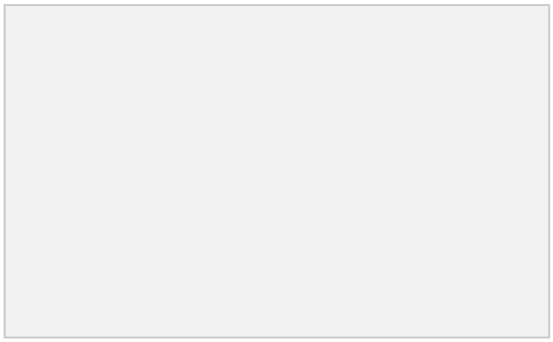
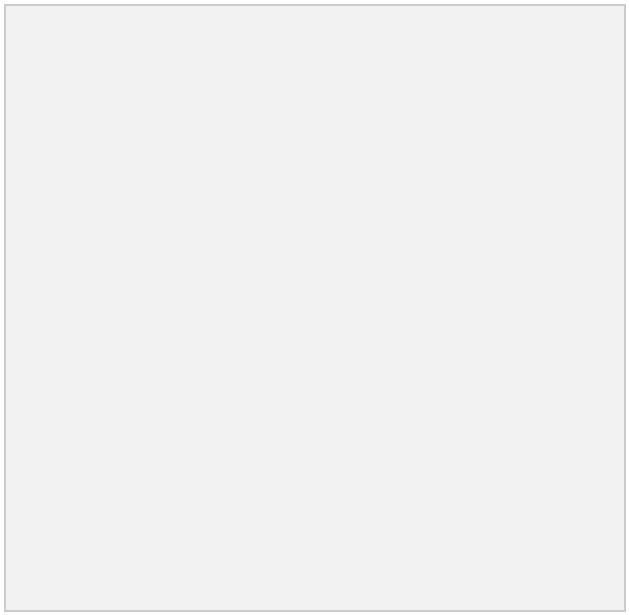
I/O expansion circuit Expander PCF8574 is used to expand the I/O communication pin of the Microcontroller via I2C interface, the circuit has the ability to expand 8 I/O to help you communicate with many devices. Through a few simple setup steps, the DIP Switch integrated circuit makes it easy to change the I2C address. Using PCF8574 brings many benefits such as expanding the number of GPIOs, saving GPIO pins, easy communication via I2C interface, reducing development costs and flexible integration in embedded applications.

4. Keypad

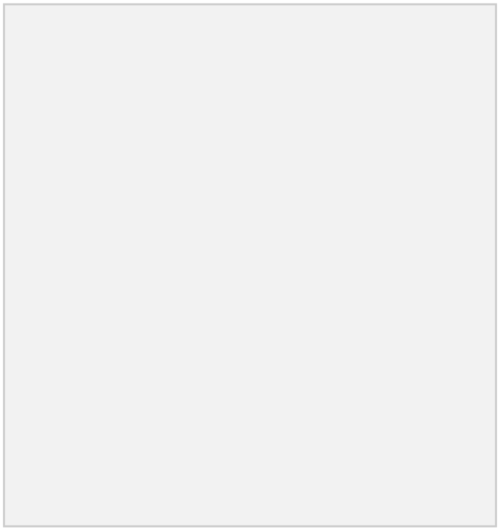


A keypad is a common input device used to enter data from users into electronic systems.

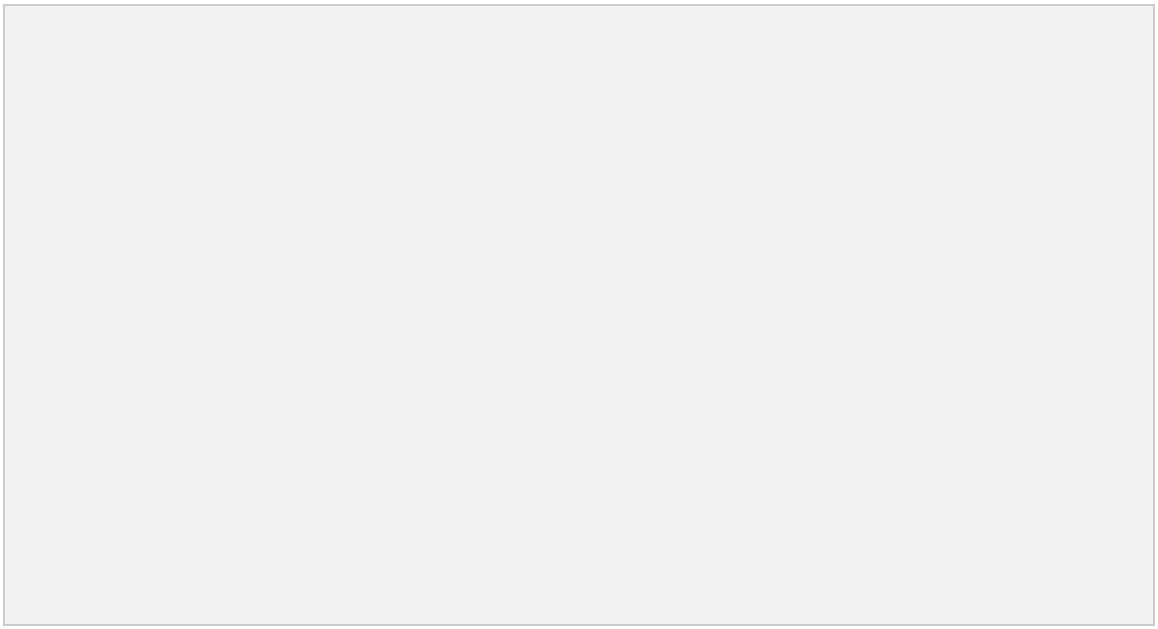
5. Emitting LED - Infrared LED

5MM Infrared Transmitter LED	**5MM Infrared Receiver Led**
Voltage 1.4-1.6V Wavelength 940nm Current - Forward DC 100mA	Dimensions 5x5x3.6mm Voltage 1.2 -1.6 VDC. Wavelength 940nm
	

In photoresistors, photodiodes and phototransistors, the energy of light shines on the semiconductor and energizes the electrons beyond the band gap.
Conversely, when an electron from the conduction band falls to the valence band, it will emit an energy E=h.f
Conduction band Valence band Forbidden band hf. When forward-biasing a P-N connection, free electrons from the N region pass through the P region and recombine with holes (in terms of energy, we say the electrons in the conduction band - with high energy - fall to the chemical band). value - has low energy - and combines with holes), when recombined, it produces energy.



6. Relay



Relay Circuit 1 KY-019 5VDC has a disconnection contact consisting of 3 points NC (normally closed), NO (normally open) and COM (common pin) completely isolated from the main board, in normal state NC is not activated will connect to COM, when the COM state is activated, it will switch to connect to NO and lose connection to NC.

Voltage used: 5VDC.

Trigger signal: TTL 3.3~5VDC, high level High Relay closes, low level Low Relay turns off.

The relay consumes about 80mA current.

Maximum switching voltage: AC250V ~ 10A or DC30V ~ 10A (For safety, use for loads with capacity <100W).

IV. SOFTWARE DESIGN AND THE OPERATIONS OF THE COMPONENT

1. IR transmitter

Function PWM_Init(uint16_t period):

Figure: ATmega328P pinout

- This function is used to initialize the configuration of the PWM controller.
- PORTB's pin setting is output (DDRB = 0xFF) to control the PWM signal.
- Set the Timer/Counter 1 controller modes (TCCR1A and TCCR1B) to the appropriate PWM mode and enable the PWM compare function (COM1A1 and COM1B1).
- Set the cyclic counting mode (Fast PWM) for Timer/Counter 1 (WGM11, WGM12, WGM13). (WGM13:WGM10 = 1110 is Fast PWM mode)
- Set the maximum count value for Timer/Counter 1 (ICR1) with the period value given from the function parameter.

Function PWM_Set_Duty(uint16_t duty):

- This function is used to set the duty cycle of the PWM pulse.
- Set the comparison value (OCR1A) with the duty value given from the function parameter.

main() function:

● In the main function, the program calls the PWM_Init() function with a pre-calculated period value to configure the PWM controller with a transmit frequency of 38 MHz.

● The program then calls the PWM_Set_Duty() function with the desired duty cycle value to set the duty rate of the PWM pulse.

Calculate the period value to create a frequency of 38 MHz:

● The PWM transmission frequency is calculated based on the period value.

● In this case, the period value is calculated based on the formula

$$period = \frac{1}{38k} * \frac{1}{0.125} * 10^6 = 210.5$$

● With F_CPU = 8 MHz and desired transmit frequency of 38 MHz, the calculated period value is 210.5.

● However, because period must be an integer, the final value used is 211.

```
void PWM_Init (uint16_t period){
    DDRB = 0xFF;
    TCCR1A |= (1 << WGM11) | (1<< COM1A1) | (1 << COM1B1);
    TCCR1B |= (1 << WGM12) | (1 << WGM13);
    ICR1 = period;
    //F_CPU = 8MHz => T_CPU =1/8M =0.125us
    //T = period * 0.125 us
}


void PWM_Set_Duty(uint16_t){
    OCR1A = duty;
}


int main(){
    PWM_Init(211);      //38Mhz
    PWM_set_duty(105);  //duty cycle
}
```

keyfind() function:

● This function is used to find and return the value of the pressed key.

● Use an infinite loop (while(1)) to iteratively read data from the key matrix.

● First, configure the pins of the key matrix as output pins (KEY_DDR = 0xF0) and high size (KEY_PRT = 0xFF).

- Perform an inspection of the columns of the key matrix to determine if any key was pressed (colloc = (KEY_PIN & 0x0F)).
- Next, perform the key row determination by testing the rows sequentially (KEY_PRT = 0xEF, KEY_PRT = 0xDF, KEY_PRT = 0xBF, KEY_PRT = 0x7F).
- When the pressed key is found, the function will return the corresponding value from the keypad array.

```c
#define KEY_PRT   PORTD
#define KEY_DDR   DDRD
#define KEY_PIN   PIND

#define COL_PRT PORTB
#define COL_DDR DDRB
#define COL_PIN PINB

unsigned char keypad[4][4] = {
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};

unsigned char colloc, rowloc;

char keyfind(){
    while(1) {
        KEY_DDR = 0x0F;  // PD0-PD3 as input and PD4-PD7 as output
        KEY_PRT = 0xF0;  // PD0-PD3 pull-up enabled, PD4-PD7 pull-up disabled

        do {
            KEY_PRT &= 0x0F;
            asm("NOP");
            colloc = (KEY_PIN & 0x0F);
        } while(colloc != 0x0F);

        do {
            do {
                _delay_ms(20); // 20ms key debounce time
                colloc = (KEY_PIN & 0x0F);
            } while(colloc == 0x0F);

            _delay_ms (40); // 40 ms key debounce time
            colloc = (KEY_PIN & 0x0F);
        } while(colloc == 0x0F);

        COL_DDR = 0x00; // PB2, PB3 as input
        COL_PRT = 0xFF; // PB2, PB3 pull-up enabled
        if(COL_PIN & (1 << PB2)) {
            rowloc = 0;
            break;
        }

        if(COL_PIN & (1 << PB3)) {
            rowloc = 1;
            break;
        }

        KEY_PRT = 0xFB; // PD2-PD7 as output except PD2
        asm("NOP");
        colloc = (KEY_PIN & 0x0F);
        if(colloc != 0x0F) {
            rowloc = 2;
            break;
        }

        KEY_PRT = 0xF7; // PD2-PD7 as output except PD3
        asm("NOP");
        colloc = (KEY_PIN & 0x0F);
        if(colloc != 0x0F) {
            rowloc = 3;
            break;
        }
    }

    if(colloc == 0x0E)
        return(keypad[rowloc][0]);
    else if(colloc == 0x0D)
        return(keypad[rowloc][1]);
    else if(colloc == 0x0B)
        return(keypad[rowloc][2]);
    else
        return(keypad[rowloc][3]);
}
```

Main program

To encode signals for infrared (IR) transmission systems, manufacturers often apply encoding standards such as NEC, Sony, and many others. Each of these encoding standards uses a special encoding method to avoid confusion between signals. In our project, we decided to use the character '$' to encode the IR signal.

During design and implementation, the application of the '$' character as part of the signal encoding will need to be accounted for and ensure its independence from other encoding standards, and ensure the efficiency and reliability of data transmission through the IR system.

To ensure the accuracy of signal transmission and avoid erroneous signal reception, we recommend including an inverting signal during transmission. This inverted signal will be created by performing bit inversion on the original signal before transmission.

During the reception process, the inverted signal will be transmitted in parallel with the original signal. This enhances the ability to detect and correct errors during data reception. This way, users can ensure that the data received is accurate and not affected by noise or interference from other signals.

```
int main(){
  DDRB |= 0xFF;
  char j,d=0;

  USART_Init(9600);
  PWM_Init(211);
  PWM_Set_Duty(105);
  while(1){
    j = keyfind();
    switch (j)
    {
    case ('1'):
      PWM_Start();
      USART_TxChar('$');
      USART_TxChar('1');
      d = ~j;
      USART_TxChar(d);
      _delay_ms(10);
      break;

    case ('2'):
      PWM_Start();
      USART_TxChar('$');
      USART_TxChar('2');
      d = ~j;
      USART_TxChar(d);
      _delay_ms(10);
      break;

    case ('3'):
      PWM_Start();
      USART_TxChar('$');
      USART_TxChar('3');
      d = ~j;
      USART_TxChar(d);
      _delay_ms(10);
      break;
    default:
      PWM_Start();
      USART_TxChar('$');
      USART_TxChar('=');
      d = ~j;
      USART_TxChar(d);
      _delay_ms(10);
      break;
    }
    Timer_OFF();
  }
}
```

2. Receiver Infrared

i2c operation:

```
void i2c_init(){
    TWBR = 0x62;        //  Baud rate is set by calculating
    TWCR = (1 << TWEN);   //Enable I2C
    TWSR = 0x00;        //Prescaler set to 1

}
        //Start condition
void i2c_start(){
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTA); //start condition
    while (!(TWCR & (1<<TWINT)));                //check for start condition

}
        //I2C stop condition
void i2c_write(char x){                    //Cpn esta funcion se escribe en el bus TWDR
    TWDR = x;                              //Move value to I2C
    TWCR = (1<<TWINT) | (1<<TWEN);         //Enable I2C and clear interrupt
    while  (!(TWCR &(1<<TWINT)));
}

char i2c_read(){
    TWCR  = (1<<TWEN) | (1<<TWINT);  //Enable I2C and clear interrupt
    while (!(TWCR & (1<<TWINT)));    //Read successful with all data received TWDR
    return TWDR;
}
```

lcd operation:

lcd_init() is used to initialize the LCD screen, set necessary parameters and prepare the screen for data display
lcd_clear() to clear all content on the screen, clean the screen and prepare it for displaying new data.
lcd_print() will transfer the specified text data to the LCD screen, from the cursor's current position to the last position specified. (character)
lcd_print_int() will transfer the specified text data to the LCD screen, from the cursor's current position to the last position specified. (integer)
lcd_i2c_gotoxy() is used in LCD control applications that connect via the I2C protocol to move the display cursor to a specific location on the LCD screen.

```
#include "i2c.h"
#include <util/delay.h>
#define  F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#define LCD_DPRT PORTB
#define LCD_DDDR DDRB
#define LCD_RS 0
#define LCD_EN 1

void toggle()
{
  TWDR |= 0x02;          //---PIN En de la LCD en = 1;  -----Latching data in to LC
data register using High to Low signal
  TWCR = (1<<TWINT) | (1<<TWEN);  //---Enable I2C and clear interrupt-
  while  (!(TWCR &(1<<TWINT))); //---Simepre poner despues de la linea anterior al
mandar datos por TWDR
  delay(1);
  TWDR &= ~0x02;          //---PIN del Enable de la LCD en = 0;
  TWCR = (1<<TWINT) | (1<<TWEN);  //---Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
}

void lcd_cmd_hf(char v1)
{
  TWDR &=~0x01;          //PIN RS de la LCD rs = 0; ----Selecting register as Comma
register
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR &= 0x0F;          //----clearing the Higher 4 bits
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR |= (v1 & 0xF0);      //----Masking higher 4 bits and sending to LCD
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  toggle();
```

```
}

void lcd_cmd(char v2)
{
  TWDR&=~0x01;           //rs = 0; ----Selecting register as command register
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR &= 0x0F;                   //----clearing the Higher 4 bits
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR |= (v2 & 0xF0);      //----Masking higher 4 bits and sending to LCD
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  toggle();

  TWDR &= 0x0F;                    //----clearing the Higher 4 bits
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR |= ((v2 & 0x0F)<<4);   //----Masking lower 4 bits and sending to LCD
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  toggle();
}

void lcd_dwr(char v3)
{
  TWDR|=0x01;            //rs = 1; ----Selecting register as command register
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR &= 0x0F;             //----clearing the Higher 4 bits
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR |= (v3 & 0xF0);      //----Masking higher 4 bits and sending to LCD
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  toggle();

  TWDR &= 0x0F;          //----clearing the Higher 4 bits
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  TWDR |= ((v3 & 0x0F)<<4);   //----Masking lower 4 bits and sending to LCD
  TWCR = (1<<TWINT) | (1<<TWEN);  //Enable I2C and clear interrupt
  while  (!(TWCR &(1<<TWINT)));
  toggle();
}

void lcd_init()
{
  lcd_cmd_hf(0x30);       //-----Sequence for initializing LCD
  lcd_cmd_hf(0x30);       //-----     "            "              "
  lcd_cmd_hf(0x20);       //-----     "            "              "
  lcd_cmd(0x28);          //-----Selecting 16 x 2 LCD in 4Bit mode
  lcd_cmd(0x0C);          //-----Display ON Cursor OFF
  lcd_cmd(0x01);          //-----Clear display
  lcd_cmd(0x06);          //-----Cursor Auto Increment
  lcd_cmd(0x80);          //-----1st line 1st location of LCD
}


void lcd_msg(char *c)
{
  while(*c != 0)      //----Wait till all String are passed to LCD
  lcd_dwr(*c++);    //----Send the String to LCD
}

void lcd_rig_sh()
{
  lcd_cmd(0x1C);      //----Command for right Shift
  _delay_ms(200);
}

void lcd_lef_sh()
{
  lcd_cmd(0x18);      //----Command for Left Shift
  _delay_ms(200);
}

void lcd_i2c_gotoxy(uint8_t row, uint8_t col) {
    uint8_t position = 0x80; // Starting address of the LCD DDRAM

    // Calculate the position based on row and column
    if (row == 1) {
        position += col; // For the first row, just add the column number
    } else if (row == 2) {
        position += (0x40 + col); // For the second row, add 0x40 to switch to the
second line, then add the column number
    }

    // Send the command to set the cursor position
    TWDR &= ~0x01; // rs = 0; Selecting register as command register
    TWCR = (1 << TWINT) | (1 << TWEN); // Enable I2C and clear interrupt
    while (!(TWCR & (1 << TWINT)));

    TWDR &= 0x0F; // Clearing the higher 4 bits
    TWCR = (1 << TWINT) | (1 << TWEN); // Enable I2C and clear interrupt
    while (!(TWCR & (1 << TWINT)));

    TWDR |= (position & 0xF0); // Masking higher 4 bits and sending to LCD
    TWCR = (1 << TWINT) | (1 << TWEN); // Enable I2C and clear interrupt
    while (!(TWCR & (1 << TWINT)));
    toggle();

    TWDR &= 0x0F; // Clearing the higher 4 bits
    TWCR = (1 << TWINT) | (1 << TWEN); // Enable I2C and clear interrupt
    while (!(TWCR & (1 << TWINT)));

    TWDR |= ((position & 0x0F) << 4); // Masking lower 4 bits and sending to LCD
    TWCR = (1 << TWINT) | (1 << TWEN); // Enable I2C and clear interrupt
    while (!(TWCR & (1 << TWINT)));
    toggle();
}

void lcd_print(const char *data) {
    lcd_msg(data);
}

void lcd_print_int(int value) {
    char buffer[20];
    snprintf(buffer, sizeof(buffer), "%d", value);
    lcd_msg(buffer);
}

void lcd_clear() {
    lcd_cmd(0x01); // Send clear display command
    _delay_ms(50);      // Delay to allow LCD to clear the display
}
```

main function

```
char check,data,invdata,count=0;
int hello_user = 0;

ISR(USART_RXC_vect)
{
     if (hello_user == 0){
    helloUser();
    }

    if(count == 0)
    {
        if (USART_RxChar() == '$'){
        hello_user = 1;
        count++;
        }

        else
        count=0;
    }

    if (count == 1)
    {
        data = USART_RxChar();
        count++;
    }

    if (count == 2)
    {
        invdata = USART_RxChar();
        if ((data | invdata)== '1')
        {
            press_key_1();
        }
        if ((data | invdata)== '2')
        {
            press_key_2();
        }
        if ((data | invdata)== '3')
        {
            press_key_3();
        }
    count=0;
    }

    if (flashingLed1) {
        PORTC |= 0x02;
        _delay_ms(100);
        PORTC |= 0x00;
        _delay_ms(100);
    }
    if (flashingLed2) {
        PORTC |= 0x03;
        _delay_ms(100);
        PORTC |= 0x00;
        _delay_ms(100);
    }
    if (flashingLed3) {
        PORTC |= 0x04;
        _delay_ms(100);
        PORTC |= 0x00;
        _delay_ms(100);
    }
    if (flashingLed_all) {
        PORTC |= 0x0E;
        _delay_ms(100);
        PORTC |= 0x00;
        _delay_ms(100);
    }

    if (chasingLed_all == 1){
      chasingLEDs(400);
    }else if(chasingLed_all ==2){
      chasingLEDs(100);
    }else if(chasingLed_all ==3){
      chasingLEDs(50);
    }else if(chasingLed_all ==4){
      //turnOffAllLEDs();
    }

}

int main(void)
{
    lcdinit();
    USART_Init(1200);
    lcd_clear();
    sei();
    while(1);

}
```

MODE 1 operating

- Normal mode: turn on/off each light or all lights

```
void helloUser(){
    lcd_init();
    lcd_i2c_gotoxy(3, 0);
    lcd_print("HELLO USER");
    lcd_i2c_gotoxy(0, 1);
    lcd_print("PLEASE PRESS KEY!");
}


void turnOffAllLEDs() {
    PORTC |= 0x00;
}

void updateLEDLCD(int ledNumber, int led_status) {
  lcd_clear();
  lcd_i2c_gotoxy(3, 0);
  lcd_print("NORMAL MODE");
  lcd_i2c_gotoxy(0, 1);
  lcd_print("LED ");
  lcd_print_int(ledNumber);
  lcd_print(": ");
  if(led_status == 0){
    lcd_print("OFF");
  }else lcd_print("ON ");
}

void press_key_1(){
  buttonPressCount++;
  if (buttonPressCount > 5) {
    buttonPressCount = 1;
  }
  switch(buttonPressCount) {
    case 1:
    chasingLed_all =4 ;
    buttonPressCount_3 =4;
          flashingLed_all = false;
          flashingLed1 = false;
          flashingLed2 = false;
          flashingLed3 = false;
      chasing_off();
      PORTC |= 0x02;
      updateLEDLCD(1,1);
      break;
    case 2:
      turnOffAllLEDs();
      DDRC |= 0x03;
      updateLEDLCD(2,1);
      break;
    case 3:
      turnOffAllLEDs();
      PORTC |= 0x04;
      updateLEDLCD(3,1);
      break;
    case 4:
      turnOffAllLEDs();
      PORTC |= 0x0E;
      updateLEDLCD(123,1);
      break;
    case 5:
      turnOffAllLEDs();
      updateLEDLCD(123,0);
      break;
  }
}
```

MODE 2 operating

- Flashing mode: flash each light or all lights

```
void updateLEDLCD_2(int status) {
  lcd_clear();
  lcd_i2c_gotoxy(3, 0);
  lcd_print("FLASH MODE");
  lcd_i2c_gotoxy(0, 1);
  if(status == 1){
    lcd_print("ON");
  }else lcd_print("OFF ");
```

```
}


void press_key_2(){
   buttonPressCount_2++;
      if (buttonPressCount_2 > 5) {
        buttonPressCount_2 = 1;
      }
      switch(buttonPressCount_2) {
        case 1:
          chasingLed_all =4 ;
          buttonPressCount_3 =4;
          turnOffAllLEDs();
          flashingLed2 = false;
          flashingLed3 = false;
          flashingLed1 = true;
          updateLEDLCD_2(1);
          break;
        case 2:
          flashingLed1 = false;
          flashingLed3 = false;
          flashingLed2 = true;
          updateLEDLCD_2(1);
          break;
        case 3:
          flashingLed1 = false;
          flashingLed2 = false;
          flashingLed3 = true;
          updateLEDLCD_2(1);
          break;
        case 4:
          flashingLed1 = false;
          flashingLed2 = false;
          flashingLed3 = false;
          flashingLed_all = true;
          updateLEDLCD_2(1);
          break;
        case 5:
          flashingLed_all = false;
          flashingLed1 = false;
          flashingLed2 = false;
          flashingLed3 = false;
          updateLEDLCD_2(0);
          break;
      }
}

//main function
if (flashingLed1) {
    PORTC |= 0x02;
    _delay_ms(100);
    PORTC |= 0x00;
    _delay_ms(100);
}
if (flashingLed2) {
    PORTC |= 0x03;
    _delay_ms(100);
    PORTC |= 0x00;
    _delay_ms(100);
}
if (flashingLed3) {
    PORTC |= 0x04;
    _delay_ms(100);
    PORTC |= 0x00;
    _delay_ms(100);
}
if (flashingLed_all) {
    PORTC |= 0x0E;
    _delay_ms(100);
    PORTC |= 0x00;
    _delay_ms(100);
}
```

MODE 3 operating

- Chasing mode: LED chasing mode

```
void press_key_3(){
        buttonPressCount_3++;
      if (buttonPressCount_3 > 4) {
        buttonPressCount_3 = 1;
      }
      switch(buttonPressCount_3) {
        case 1:
    flashingLed_all = false;
          flashingLed1 = false;
          flashingLed2 = false;
          flashingLed3 = false;
          chasing_off();
          turnOffAllLEDs();
          chasingLed_all = 1;
          updateLEDLCD_3(1);
          break;
        case 2:
          chasing_off();
          turnOffAllLEDs();
          chasingLed_all = 2;
          updateLEDLCD_3(2);
          break;
        case 3:
          chasing_off();
          turnOffAllLEDs();
          chasingLed_all = 3;
          updateLEDLCD_3(3);
          break;
        case 4:
          chasing_off();
          turnOffAllLEDs();
          chasingLed_all = 4;
          updateLEDLCD_3(4);
          break;

      }
}

void chasingLEDs(int speedDelay) {
    PORTC |= 0x02;
    _delay_ms(speedDelay);

    PORTC |= 0x03;
    _delay_ms(speedDelay);

    PORTC |= 0x04;
    _delay_ms(speedDelay);
    PORTC |= 0x00;
}






//main function
    if (chasingLed_all == 1){
      chasingLEDs(400);
    }else if(chasingLed_all ==2){
      chasingLEDs(100);
    }else if(chasingLed_all ==3){
      chasingLEDs(50);
    }else if(chasingLed_all ==4){
      //turnOffAllLEDs();
    }
```

V. TEST PROCEDURE AND RESULT

The purpose of this test report is to outline the testing procedures and results for the IR remote control designed for use with various electronic devices. The remote control was developed to provide users with a convenient and reliable means of controlling compatible devices via infrared signals.

Test Objectives:

- To verify the functionality of all buttons on the remote control.
- To ensure the remote operates within the specified range and angles.
- To test the compatibility and responsiveness of the remote with target devices.
- To evaluate the durability and user experience of the remote control.
- To confirm compliance with relevant standards and regulations.

1. Functionality Testing:

Tested each button on the remote control for responsiveness and accuracy. Confirmed that all buttons functioned correctly without any malfunctions.

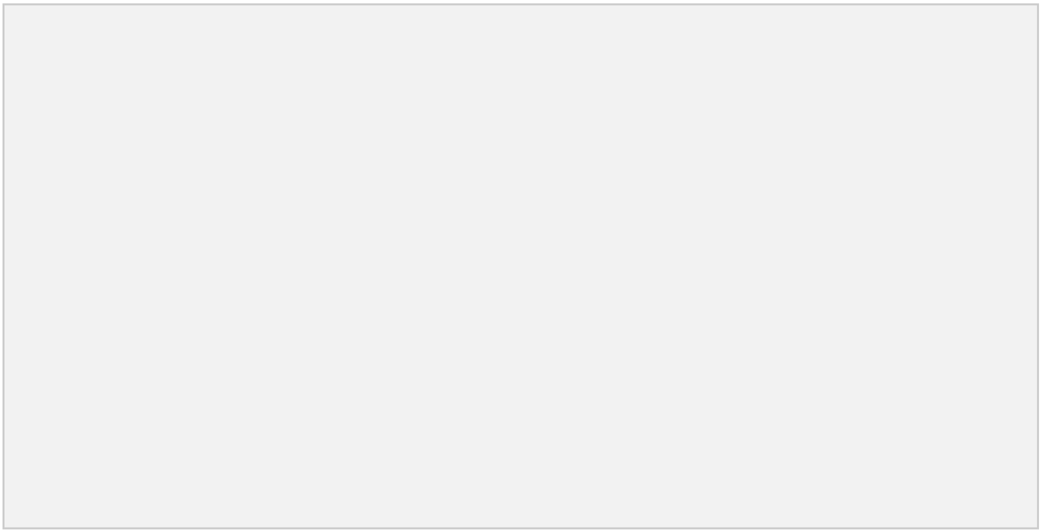


Figure: The system starts up when power is applied

Figure: Several operating modes of the system

2. Range and Angle Testing:

Tested the remote control at various distances and angles from the target device to ensure reliable operation. Confirmed that the remote worked within the specified range and angles.

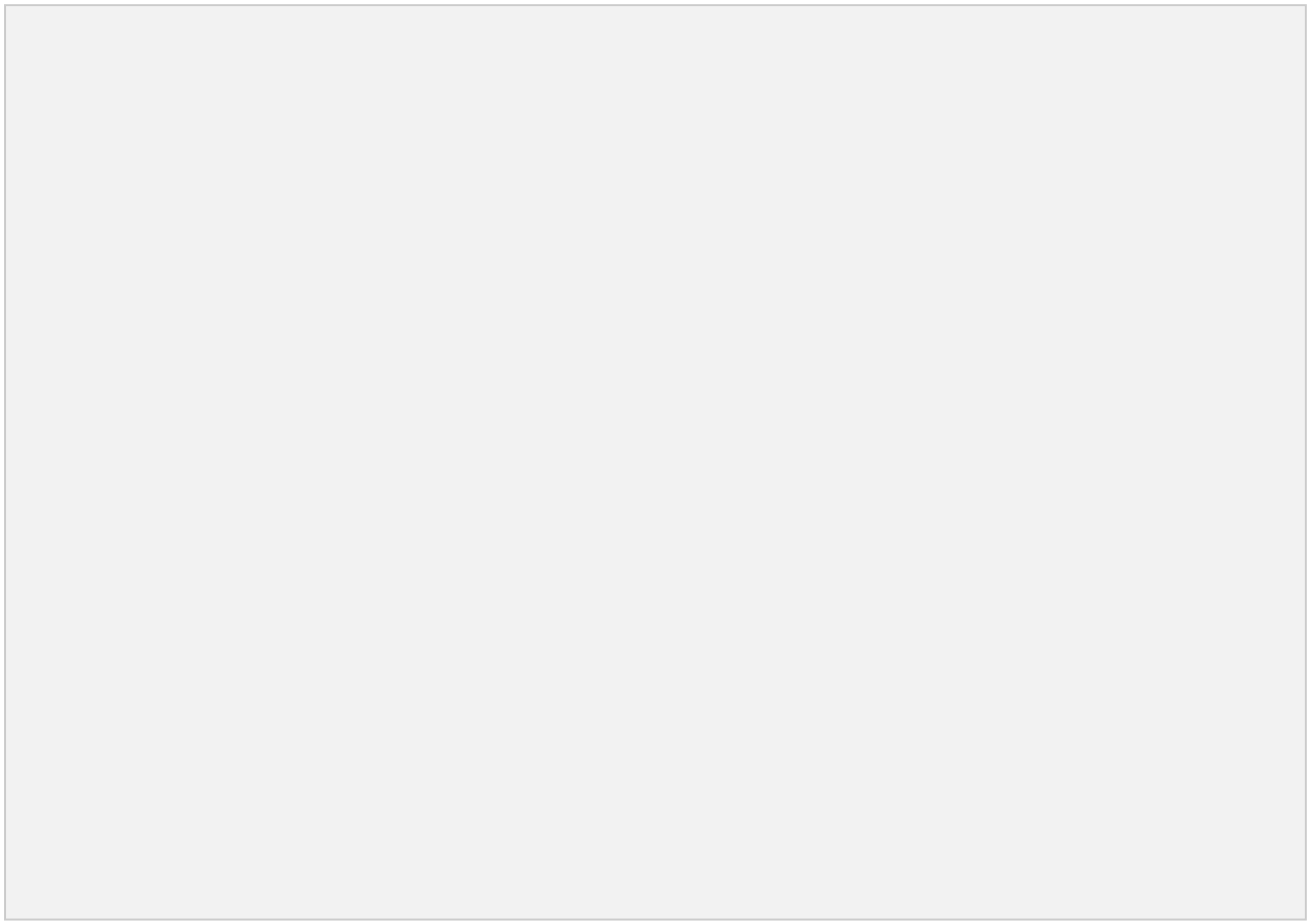


Figure: Example for range and angle testing

Because the system has not been processed for commercialization, infrared waves can be received 360 degrees and can be accurately received with a maximum range of 300cm.

3. Compatibility Testing:

Tested the remote control with multiple target devices to ensure compatibility. Verified that all functions supported by the devices could be controlled using the remote.

The system has 3 operating modes such as:

- Normal mode: turn on/off each light or all lights
- Flash mode: flash each light or all lights
- Chasing mode: LED chasing mode

4. Durability Testing:

Subjected the remote control to drop and impact tests to ensure durability. Since the circuit board is implemented on a breadboard, we do not evaluate the durability of the system.

5. User Experience Testing:

Evaluated the overall user experience, including ease of use and button layout. Ensured that the remote control provided an intuitive interface for users.

6. Power Testing:

To assess the power consumption of the circuit, we conducted measurements of voltage (U) and current (I) at the terminals of the power supply under various operational modes.

$$P = UI$$

where:

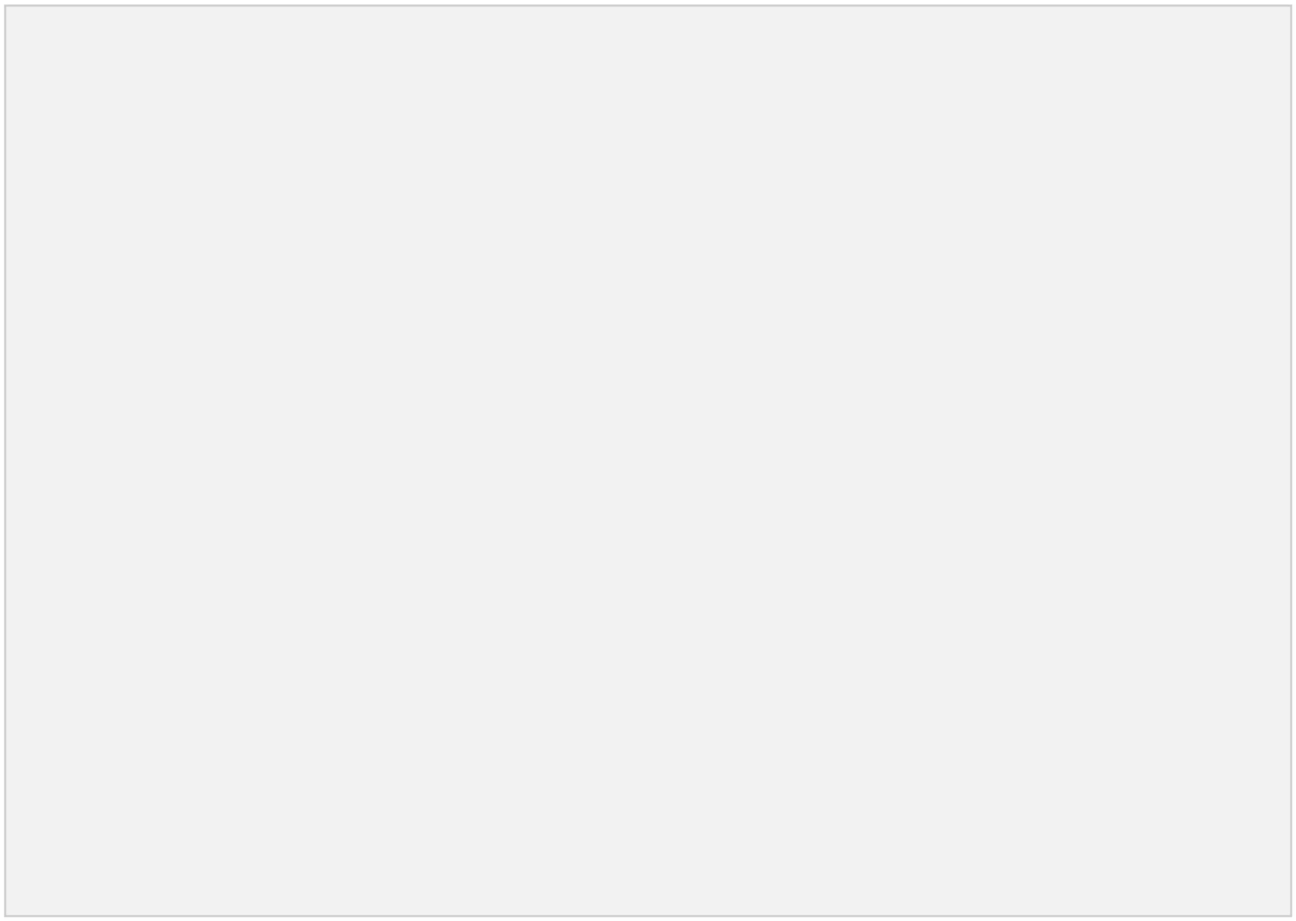
P is the power consumption (unit: watt, W).
U is the voltage (unit: volt, V) at the circuit's power supply.
I is the current (unit: ampere, A) at the circuit's power source.

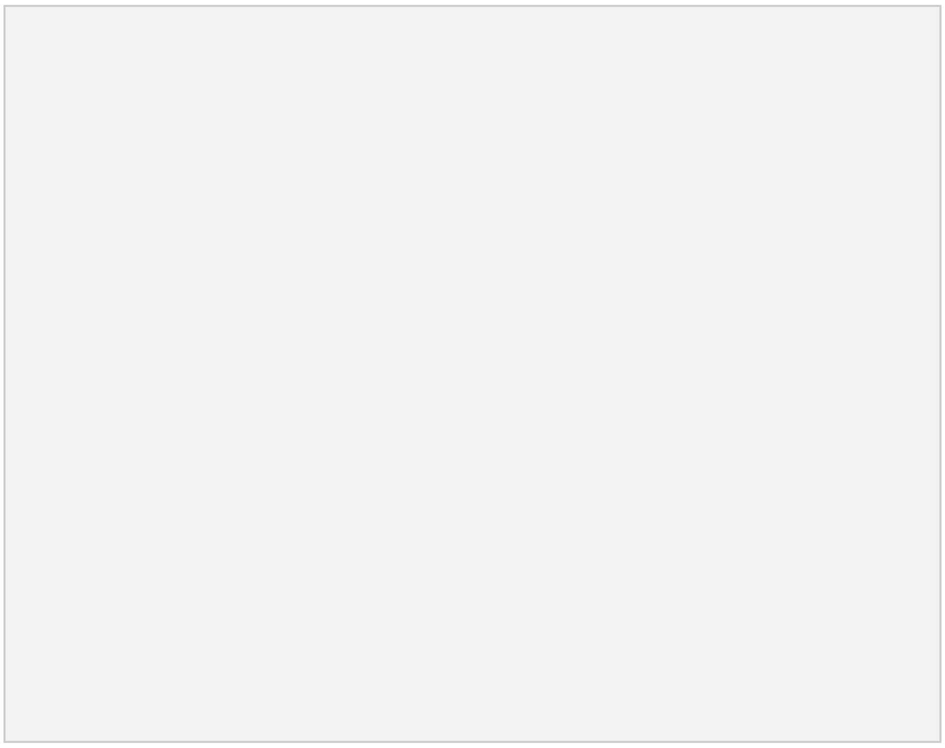
This measurement is often used to measure the total power consumption of a circuit or device. However, for more accurate measurements, one needs to measure each individual component for a more detailed analysis of power consumption and performance.

Our research team quantified the power consumption of the IR transmitter circuit to be approximately 0.05 watts in its standby state.

Because the output signal of the transmitter IC has a small current: - 0.1mA ÷ 1.0mA
so we have to amplify them. Therefore, I use two paired transistors Darlington to amplify the current signal for the infrared LED to emit stronger.

VI. SCHEMATIC AND PCB LAYOUT





VII. FUTURE WORK

The IR remote control has undergone comprehensive testing, demonstrating its reliability, compatibility, and user-friendliness. However, there are several areas where further improvements and future work could enhance its performance and usability.
The future work of the IR remote control involves a combination of technological advancements, user-centric design improvements, and integration with emerging technologies such as IoT and voice recognition. By addressing these areas, the remote control can remain competitive, adaptable to evolving user needs, and contribute to a more seamless and convenient user experience in the realm of home entertainment and automation.

Integration of Advanced Features:
Incorporating advanced features such as voice control or gesture recognition to enhance the user experience and cater to modern technological trends. These features could be integrated into the existing remote design to offer additional control options for users.
With the increasing popularity of smart home systems, future iterations of the IR remote control could be designed to integrate seamlessly with these systems. This would allow users to control not only traditional appliances but also smart home devices such as lights, thermostats, and security cameras.

Data Security and Privacy:
Given the increasing concerns around data security and privacy, future iterations of the remote control should prioritize encryption and secure communication protocols to protect user data from potential threats. Currently, numerous devices emit ambient IR waves that may interfere with the transmission signal of the remote control. To mitigate this interference, companies have implemented separate encoding schemes for each type of product signal during transmission.

Environmental Sustainability:
Consider the environmental impact of the remote control and explore ways to reduce waste, increase recyclability, and minimize carbon footprint in both production and usage phases.

- **Next-Generation Sustainable Remote** – battery-free remote-control prototype equipped with High-Density PV panel for ambient light harvesting, to ensure a more energy-efficiency platform that re-charges itself using ambient light from its surroundings. There are no batteries to be replaced providing maximum user convenience, with no battery disposal creating a more sustainable future.
- **Next-Generation Sustainable Materials** – battery-free remotes that are silicone-free – bringing greater sustainable value to the product solution. Instead of silicone rubber, which is difficult to recycle, the remote-control keypads use TPE – Thermoplastic Elastomer material which is designed for efficient recycling and reuse.

VIII. CONCLUSION

After a period of research and construction, the project was basically completed. With the efforts of each individual and the reasonable, tight and rhythmic division and coordination of work between each member of the group, during the process of implementing the project, we have obtained important results. The results are certain as follows:

The electrical circuit with small modules on the circuit is designed, completely constructed and has been tested many times and operates stably in practice. In this project, the group has presented quite fully the functions, The structure of each small module block on the integrated circuit board. Thus, helping readers grasp and understand the functions of each module easily. Besides, the content of the topic is presented in quite clear detail using common words and accompanying images to help readers easily understand and be able to do the same effectively. for a short time. Infrared transceiver system with the purpose of simulating infrared control signals on electronic devices such as televisions, air conditioners... and how to encode and decode the signals. Below are the highlights of the system:

- Solve the problems raised in the topic.
- The system consumes little power.
- Easy to observe, easy to use.
- There is a backup power supply for the circuit
- Low cost
- Displayed and updated continuously for users to easily grasp

IX. REFERENCE

1. AVR Microcontroller and Embedded Systems
https://electrovolt.ir/wp-content/uploads/2017/02/AVR_Microcontroller_and_Embedded_Electrovolt.ir_.pdf
2. IR Remote Control – Basics, Operation & Application
.https://www.elprocus.com/ir-remote-control-basics-operation-application/