

# BLINKY WITH CONSOLE

## ASSIGNMET – 03

BASATI SIVAKRISHNA

22976

DESE EPD

### Code explanation:

GPIO ( push buttons ) has configured for the interrupt actions. The handler has implemented in startup file for interrupts. Upon receiving the interrupt the handlers will clear the received interrupt flag and set the variables values accordingly. The UART reception is happening inside the delay loop. The count need to be upgraded by calibration method.

**UART RECEPTION IN DELAY LOOP:** (note: the j<3180 is not yet modified according to delay required)

```
268
269 void delayMs(int n) {
270     int i, j;
271     for(i = 0; i < n; i++)
272         for(j = 0; j < 3180; j++) {
273             if(UARTCharsAvail(UART0_BASE))
274             {
275                 rx_val[command_index] = UARTCharGet(UART0_BASE);
276                 if(rx_val[command_index]== '\n')
277                     command_rxd = 1;
278                 (command_index>30)? (command_index = 0): (command_index++);
279             }
280         } // Do nothing for 1 ms
281
282 }
283
284
285
```

The delaysms() function written in such a way that only 30 characters will be received and character after 30<sup>th</sup> will be kept at position 1 of rx\_val[] array.

Once new line(\n) is encountered the entire data received via UART will be passed to command parser () function.

Command parser will get modified accordingly (removing extra characters and capital letters). The modified command will get divided into two parts. 1. Parsed\_cmnd and 2. Parsed data.

And it will check the validity command and data using ascii values.

Whenever there is an invalid command, the `command_parser()` function will set the valid flag to 0 and it prints the command help message.

Logic to reject the unnecessary characters in received buffer.

```
01 while((rx_val[a] != '\n'))
02 {
03     if(!(((rx_val[a]>47) && (rx_val[a]<58)) || ((rx_val[a]>64) && (rx_val[a]<91)) || ((rx_val[a]>96) && (rx_val[a]<123))))
04     {
05         a++;
06         continue;
07     }
08     else
09     {
10         if((rx_val[a]>64) && (rx_val[a]<91))
11             rx_val[a] = rx_val[a] + 32;
12         modify_cmd[b] = rx_val[a];
13         a++;
14         b++;
15     }
16 }
```

If the command given is valid then according to the given command either respective led blinks or blink rate changes.

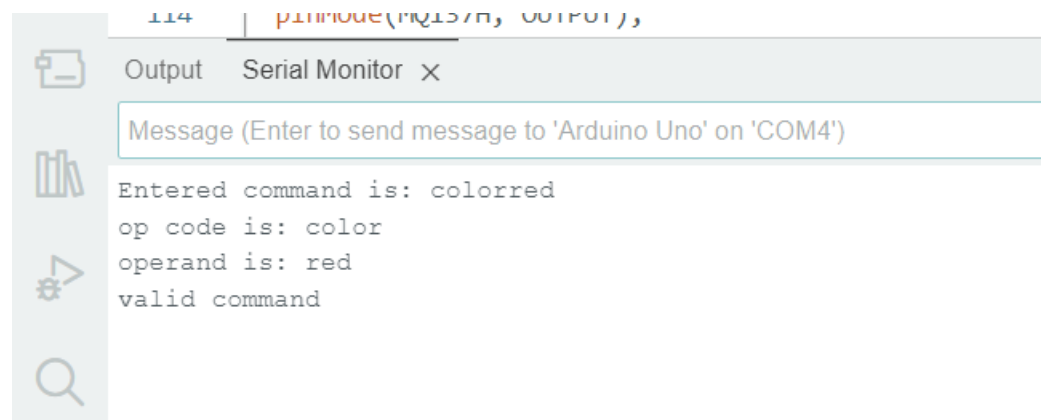
A switch case structure is used to determine the type of led to glow. and small mathematical formula is used to determine blink rate based on the command.

blink blink\_rate: Delay calculation.

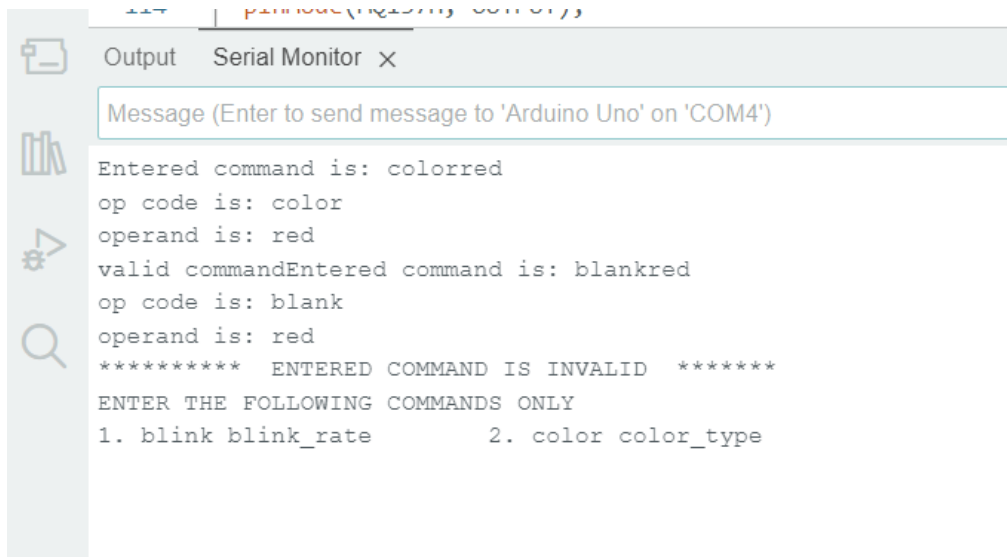
```
161
162     }
163     else if((strcmp(parsed_cmd, "blink")==0))
164     {
165         counter = atoi(parsed_data);
166         counter = ((60000)/(2*counter));
167         cmd_mode = 1;
168     }
169     }
```

**OUTPUT: Modified command will be echoed without spaces**

**Using valid command:**



### Using invalid command:



The screenshot shows the Arduino IDE interface with the Serial Monitor open. The input field contains the text "Message (Enter to send message to 'Arduino Uno' on 'COM4')". The output area displays the following text:

```
Entered command is: colorred
op code is: color
operand is: red
valid commandEntered command is: blankred
op code is: blank
operand is: red
***** ENTERED COMMAND IS INVALID *****
ENTER THE FOLLOWING COMMANDS ONLY
1. blink blink_rate      2. color color_type
```

### Entering the extra characters like special characters and spaces.



The first screenshot shows the Arduino IDE interface with the Serial Monitor open. The input field contains the text "bl & i nk 1@20". The output area displays the following text:

```
Entered command is: blink120
op code is: blink
operand is: 120
valid command
```

The second screenshot shows the Arduino IDE interface with the Serial Monitor open. The input field contains the text "Message (Enter to send message to 'Arduino Uno' on 'COM4')". The output area displays the following text:

```
Entered command is: blink120
op code is: blink
operand is: 120
valid command
```