LAB 4 - Milestone 4

Juan Aguilar
ID: 2331246

Introduction

This lab builds upon the foundation established in Lab 2, with several key modifications to the code and system functionality. A new typedef structure has been created to define a lookup table (LUT) that includes a character array and a function pointer. This allows user input (in three-letter command form) to be dynamically matched with specific functions. UART2 has been updated to include a mechanism for parsing and updating the flag for each three-character command. The while loop in main() contains the execute_command() function. This function uses the lookup table to identify and call the correct response function. To manage command flow, a flag and prev_command array have been introduced to ensure input is read properly, avoiding duplication and enabling smoother processing of new commands. These additions help maintain reliable communication between the HC-05 Bluetooth module (UART1) and the UART0 output (console terminal). Additionally, the inclusion of PWM is important, as it allows for variable processing for different signals. Its implementation in changing the red LED is a simple example that shows how varying the voltage of the signal can lead to the perception of the LED getting dimmer. Part of that is due to the frequency and duty cycle defining the period of the signal passing through. Further applications may include driving and changing the speed of a motor, for the upcoming robot project.

Specifications

- **Microcontroller:** Tiva-C Series TM4C123GXL
- **Bluetooth Module:** HC-05
- **Power Supply:** 3.3V from Tiva-C
- **UART0:** USB to PC terminal (for output display)
- **UART1:** Bluetooth communication via HC-05
- **GPIO Port F:** Controls on-board RGB LED

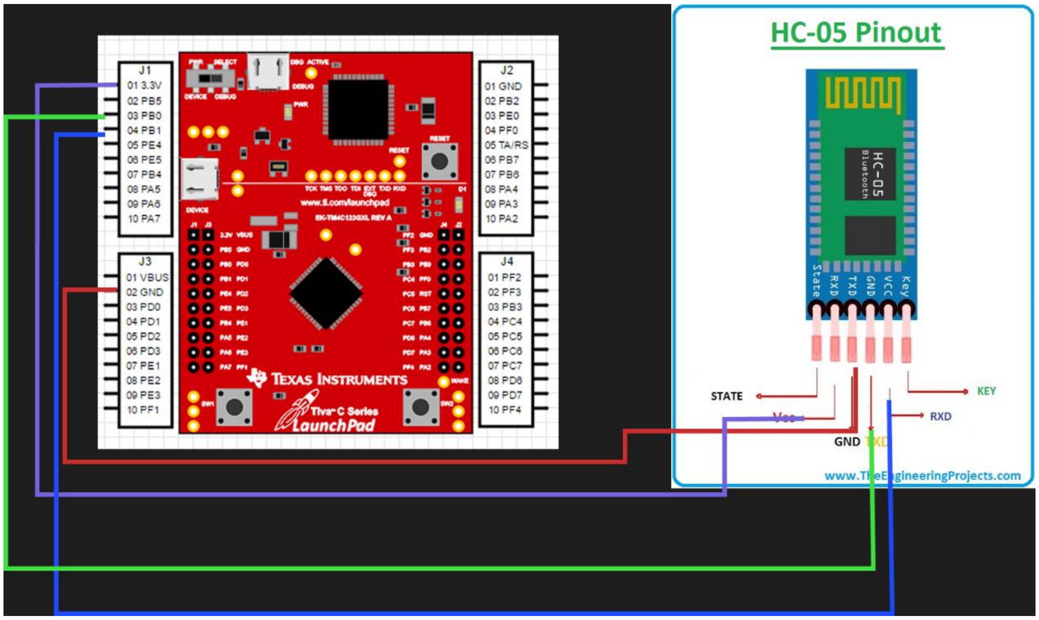

Figure 1: Pin Schematic

APIs Used

API	Purpose	Key Parameters
ROM_FPUEnable()	Enables Floating Point Unit	
ROM_SysCtlClockSet()	Sets system clock	Clock source, divider, oscillator type
ROM_SysCtlPeripheralEnable()	Enables peripheral clocks	e.g., SYSCTL_PERIPH_GPIOF
ROM_GPIOPinTypeGPIOOutput()	Sets GPIO pins as output	GPIO base, pin number
GPIOPinWrite()	Writes digital value to GPIO	Port base, pins, value
ROM_IntMasterEnable()	Enables global interrupts	
ROM_GPIOPinConfigure()	Assigns alternate function to pins	e.g., GPIO_PA0_U0RX
ROM_GPIOPinTypeUART()	Enables UART functionality on pins	Port, pins
ROM_UARTConfigSetExpClk()	Sets UART communication parameters	UART base, system clock, baud rate, config
ROM_IntEnable()	Enables interrupts for specific modules	e.g., INT_UART0, INT_UART1
ROM_UARTIntEnable()	Enables UART interrupts (RX, TX)	e.g., UART_INT_RX
SysCtlPWMClockSet()	Sets PWM module clock divider	SYSCTL_PWMDIV_64

GPIOPinTypePWM()	Configures GPIO pin as PWM output	Port base, GPIO pin
PWMGenPeriodSet()	Sets PWM period	PWM base, PWM generator, load value
PWMGenConfigure()	Configures PWM behavior	PWM base, PWM generator, behavior
PWMPulseWidthSet()	Sets PWM output duty cycle	WM base, PWM output pin
PWMOutputState()	Enables or disables PWM output	PWM base, PWM output pin, true/false value
PWMGenEnable()	Starts the PWM generator	PWM base, PWM generator

Procedures

Pins Used

1. PB0: UART1 RX (from HC-05 TX)
2. PB0: UART1 TX (from HC-05 RX)
3. PA0: UART0 RX (to PC)
4. PA1: UART0 TX (to PC)
5. PF1, PF2, PF3: Onboard RGB LEDs (R, B, G)
6. +3.3V: Power for HC-05
7. GND: Ground
8. A0, A1: Internal Use (UART0)

Order of Execution

1. **Initialize FPU:**
    a. Enables floating-point operations, especially for interrupts.
2. **System Clock Setup:**
    a. Uses 16 MHz crystal with ROM_SysCtlClockSet().
3. **Peripheral Enable:**
    a. Enables GPIO Port F (LEDs), GPIO A & B, UART0, and UART1.
4. **Configure GPIOF:**
    a. PF1, PF2, PF3 set as digital outputs to control onboard RGB LEDs.
5. **Configure UART Pins:**
    a. Use ROM_GPIOPinConfigure() and ROM_GPIOPinTypeUART() for:
        i. PA0/PA1 (UART0)
        ii. PB0/PB1 (UART1)
6. **UART Config:**
    a. UART0: Baud rate 115200 (PC terminal)
    b. UART1: Baud rate 9600 (Bluetooth)

7. **Interrupts:**
    a. Global interrupts enabled via ROM_IntMasterEnable()
    b. UART-specific interrupts enabled with ROM_IntEnable() and ROM_UARTIntEnable().
8. **PWM_init() Function Call:**
    a. Calls the function for enabling the PWM generator, setting width pulse, GPIO port F pin 1 for LED light pulsing, with a frequency of 1kHz. Clock frequency, load value, and count down mode is configured as well.
9. **Send Initial Message:**
    a. UART0 and UART1 both send startup strings using UARTSend() and UARTSend1().

UART ports must be fully initialized before any string is sent. Otherwise, behavior can be unpredictable, including communication failure or Tiva-C malfunction. PWM configuration is also necessary prior to initialization to avoid glitches and hardware damage.

Functions

- **execute_command()**
    - Accepts a 3-character string from input buffer.
    - Searches a lookup table of commands and function pointers.
    - If matched, calls the respective function.
- **Command Functions**
    - Receives instructions for matched commands.
    - For lab 2, a message is sent to UART0 to confirm that the instruction is valid.
    - Future iterations will include new APIs to modify robot behavior.
- **UART0 Interrupt Handler**
    - Buffers input characters.
    - Echos input back to itself, and UART1
- **UART1 Interrupt Handler**
    - Buffers input characters.
    - Echos input back to itself, and UART0
    - Changes the LED color based on each character input
    - Sets command_received flag to true when 3-character input is complete
- **PWM Initialization Function**
    - Enables PWM port base
    - Configures behavior and output pins for PWM
    - Defines period, load values, and pulse width
    - Initiaites the behavior and generator for PWM
- **BrightnessLevel() Function**
    - Varies duty cycle between 1 and 100

- If-statements ensure that the duty cycle increases or decreases based on current values
- Delay is implemented to allow the value to smoothly update

```c
// global variables
uint8_t pins[] = {GPIO_PIN_3, GPIO_PIN_2, GPIO_PIN_1}; // FP1 = RED, PF2 = BLUE, PF3 = GREEN
int led_count = 1;

void
UART0IntHandler(void)
{
    uint32_t ui32Status;

    // get and clear interrupt status
    ui32Status = ROM_UARTIntStatus(UART0_BASE, true);
    ROM_UARTIntClear(UART0_BASE, ui32Status);

    while (ROM_UARTCharsAvail(UART0_BASE))
    {
        char c = ROM_UARTCharGetNonBlocking(UART0_BASE);

        // send to UART1
        ROM_UARTCharPutNonBlocking(UART1_BASE, c);

        // echo to UART0
        ROM_UARTCharPutNonBlocking(UART0_BASE, c);
    }
}
```

```c
void
UART1IntHandler(void)
{
    uint32_t ui32Status;

    // get and clear interrupt status
    ui32Status = ROM_UARTIntStatus(UART1_BASE, true);
    ROM_UARTIntClear(UART1_BASE, ui32Status);

    while(ROM_UARTCharsAvail(UART1_BASE))
    {

        char c = ROM_UARTCharGetNonBlocking(UART1_BASE);

        // send to UART0
        ROM_UARTCharPutNonBlocking(UART0_BASE, c);

        // echo to UART1
        ROM_UARTCharPutNonBlocking(UART1_BASE, c);

        //turn on current LED, turn off the others
        GPIOPinWrite(GPIO_PORTF_BASE, pins[led_count], pins[led_count]);
        GPIOPinWrite(GPIO_PORTF_BASE, pins[(led_count+1)%3], 0);
        GPIOPinWrite(GPIO_PORTF_BASE, pins[(led_count+2)%3], 0);

        // Advance the counter
        led_count = (led_count + 1) % 3;


        input_cmd[command_char_count] = c;

        command_char_count++;

        if (command_char_count == 3){
            input_cmd[3] = '\0';
            command_received = true;
        }
    }
}
```

Figure 2: Global Variables & UART Interrupt Handlers

```c
void forward(void);
void scan(void);
void stop(void);
void left(void);
void right(void);
void backward(void);
void clear(void);

typedef void (*Fn)(void);
typedef struct {
    char cmd[4]; // command + '/0'
    Fn cmd_funct; // pointer to command function
}lookup;

lookup lookup_table[] = {
    {"FWD", forward},
    {"SCN", scan},
    {"STP", stop},
    {"LFT", left},
    {"RGT", right},
    {"BAC", backward},
    {"CLC", clear}
};

int table_entries = sizeof(lookup_table) / sizeof(lookup_table[0]);
```

```c
// function calls based on lookup_table
void forward() {
    char terminal_string[] = "\n\rMoving Forward!\r\n";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
}

void scan() {
    char terminal_string[] = "\n\rScanning!\r\n";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
}

void stop() {
    char terminal_string[] = "\n\rStopping!\r\n";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
}

void left() {
    char terminal_string[] = "\n\rMoving Left!\r\n";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
}

void right() {
    char terminal_string[] = "\n\rMoving Right!\r\n";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
}

void backward() {
    char terminal_string[] = "\n\rMoving Backwards!\r\n";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
}

void clear(){
    char terminal_string[] = "\r\033[2J";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
}
```

Figure 3: Lookup Table Definition & Command Function Logic

```c
int
main(void)
{
    //
    // Enable lazy stacking for interrupt handlers.  This allows floating-point
    // instructions to be used within interrupt handlers, but at the expense of
    // extra stack usage.
    //
    ROM_FPUEnable();
    ROM_FPULazyStackingEnable();
    //
    // Set the clocking to run directly from the crystal.
    //
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                       SYSCTL_XTAL_16MHZ);
    //
    // Enable the GPIO port that is used for the on-board LED.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    //
    // Enable the GPIO pins for the LED.
    //
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

    // default green LED enable
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
    //
    // Enable UART0 and UART1
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    //
    // Enable processor interrupts.
    //
    ROM_IntMasterEnable();
    //
    // Set GPIO A0, A1, B0, B1
    //
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    GPIOPinConfigure(GPIO PB0 U1RX);
```

```c
    GPIOPinConfigure(GPIO_PB0_U1RX);
    GPIOPinConfigure(GPIO_PB1_U1TX);
    ROM_GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    //
    // UART0, 115200, 8-N-1 and UART1, 9600, 8-N-1
    //
    ROM_UARTConfigSetExpClk(UART0_BASE, ROM_SysCtlClockGet(), 115200,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));

    ROM_UARTConfigSetExpClk(UART1_BASE, ROM_SysCtlClockGet(), 9600,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));
    //
    // Enable the UART interrupt.
    //
    ROM_IntEnable(INT_UART0);
    ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    ROM_IntEnable(INT_UART1);
    ROM_UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);
    //
    // Prompt for text to be entered.
    //
    char terminal_string[] = "\033[2JPlease enter characters in the remote Bluetooth Terminal and see the color change: ";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
    UARTSend1((uint8_t *) terminal_string, strlen(terminal_string));
    //
    // Loop forever echoing data through the UART.
    //
    while(1)
    {
        if (command_received){
            execute_command((char *)input_cmd);
            command_received = false;
            command_char_count = 0;
        }
    }
}
```

Figure 4: Main Function Logic

```c
void execute_command(char *cmd){
    if(strcmp(cmd, prev_input_cmd) == 0){
        char terminal_string[] = "\n\rCommand cannot be used again!\r\n";
        UARTSend((uint8_t *) terminal_string, strlen(terminal_string));
        return;
    }

    int i; // like, whyy?
    for(i = 0; i < table_entries; i++){
        if (strcmp(cmd, lookup_table[i].cmd) == 0) {
            lookup_table[i].cmd_funct(); // run the matched function

            for(i = 0; i < 4; i++){
                    prev_input_cmd[i] = cmd[i];
            }
            return;
        }
    }
    char terminal_string[] = "\n\rCommand does not exist!\r\n";
    UARTSend((uint8_t *) terminal_string, strlen(terminal_string));

}
```

Figure 5: Execute Command Function

```
void PWM_init(void){

    // PWM1 & GPIO portF enable
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_PWM1));
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));

    // PWM clock divider (systemclk / 64)
    SysCtlPWMClockSet(SYSCTL_PWMDIV_64);

    // Pin 1 to PWM output 5
    GPIOPinConfigure(GPIO_PF1_M1PWM5);

    // output pin
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);

    // clk frequency & load value
    ui32PWMClock = SysCtlClockGet() / 64;
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;

    // PWM gen 2, period set, count down mode, pulse width
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, ui32Load);
    PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, (ui32Load * dutyCycle) / 100);

    // PWM output signal enabled, gen 2 started
    PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
    PWMGenEnable(PWM1_BASE, PWM_GEN_2);

}
```

Figure 6: PWM_init() Function

```
void BrightnessLevel(){

    int brightness = 1;
    dutyCycle = 50;

    while(1){

        dutyCycle += brightness;

        if(dutyCycle >= 100){
            dutyCycle = 100;
            brightness = -1;
        }
        else if(dutyCycle <= 0){
            dutyCycle = 1;
            brightness = 1;
        }
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, (ui32Load * dutyCycle) / 100);
        SysCtlDelay(SysCtlClockGet() / 200);
    }
}
```

Figure 7: BrightnessLevel() Function

Results

For the following cases, the top terminal represents UART0, the connection to the computer, and the bottom terminal represents UART1, the connection to the HC-05 module. There are three cases to consider. The first case covers an input that does not match the look up table, which leads to a message output to UART0, letting the user know the command does not exist. The second case covers an input that matches the lookup table, which leads to a message output to UART0, letting the user know the command is being executed. The final case covers an input that matches the lookup table, but was called in the previous input, displaying the message that the command cannot be used twice in a row. Furthermore, the implementation of PWM leads to the red LED pulsing from a low brightness to a high brightness, and back down to a low brightness, repeating indefinitely, as shown in the figures below.

Figure 8: Initialization of the Program

Figure 9: Case Where Command Does Not Exist

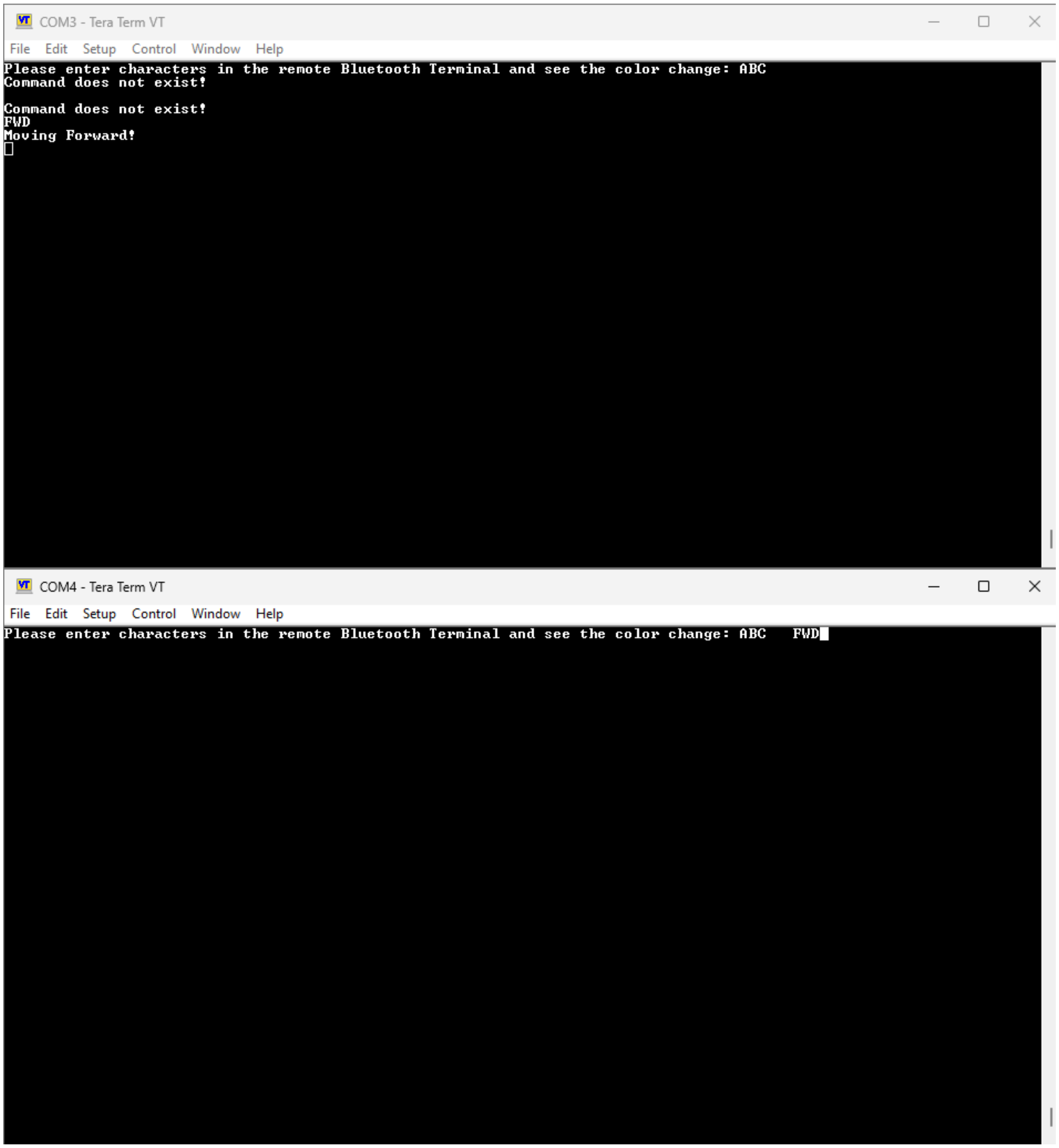

Figure 10: Case Where Command Exists

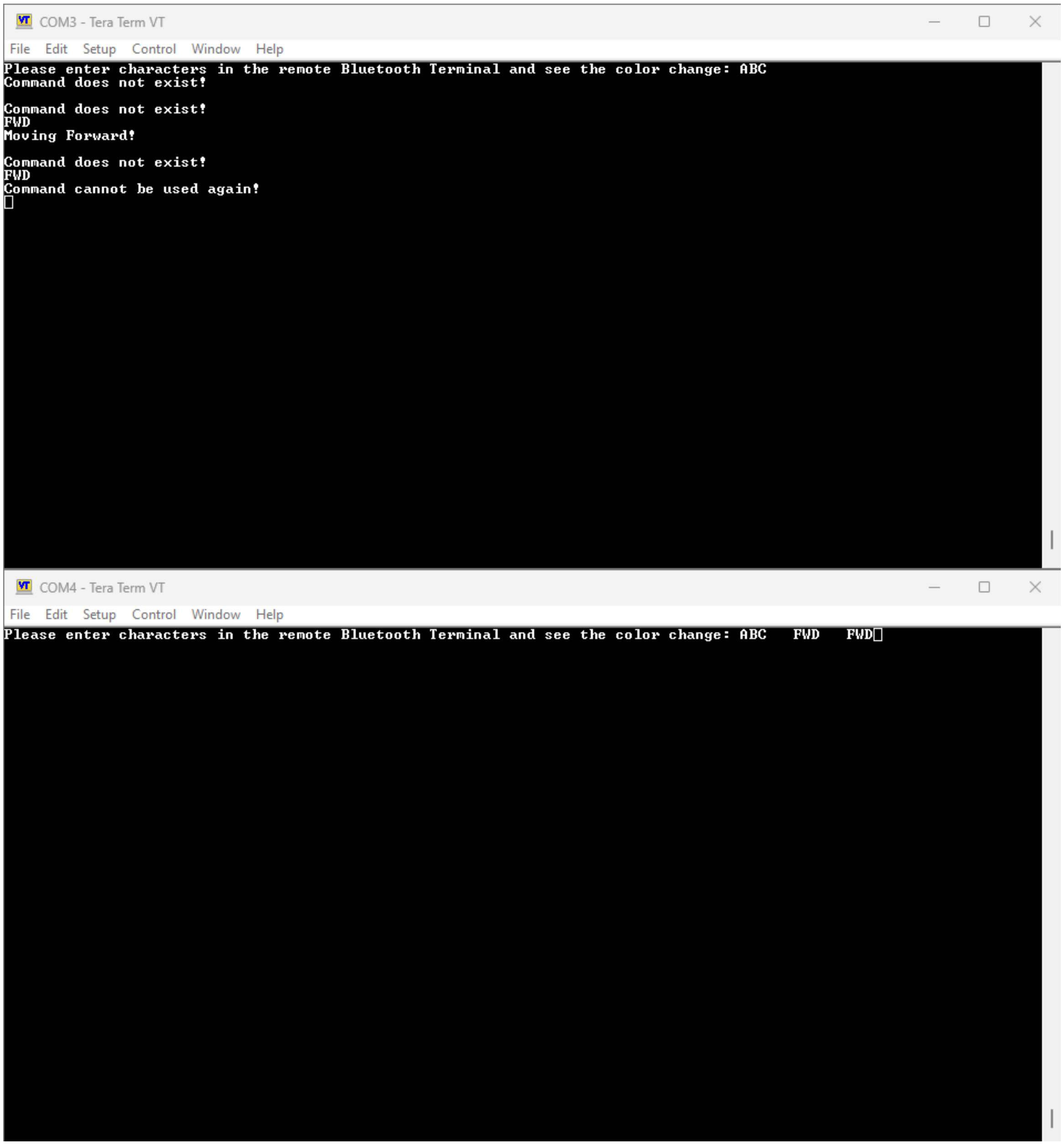

Figure 11: Case Where the Same Command Is Placed Twice in a Row

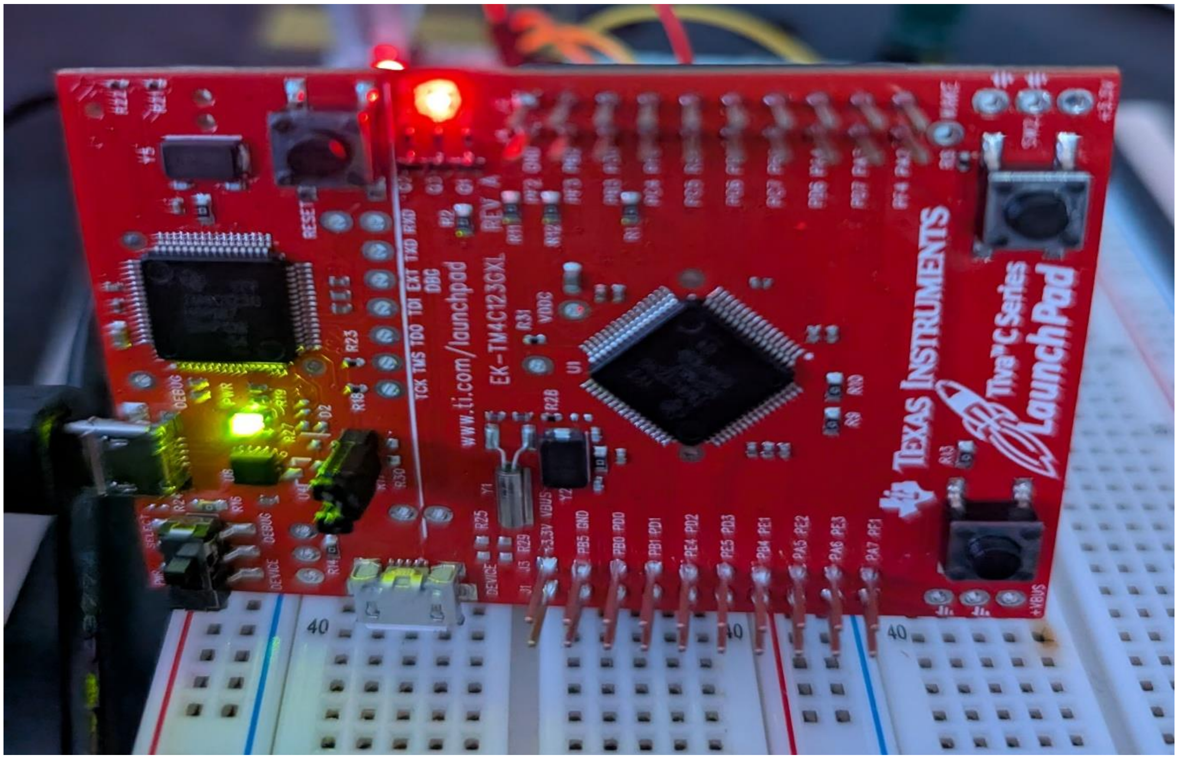

Figure 12: PWM Output at Duty Cycle of 1

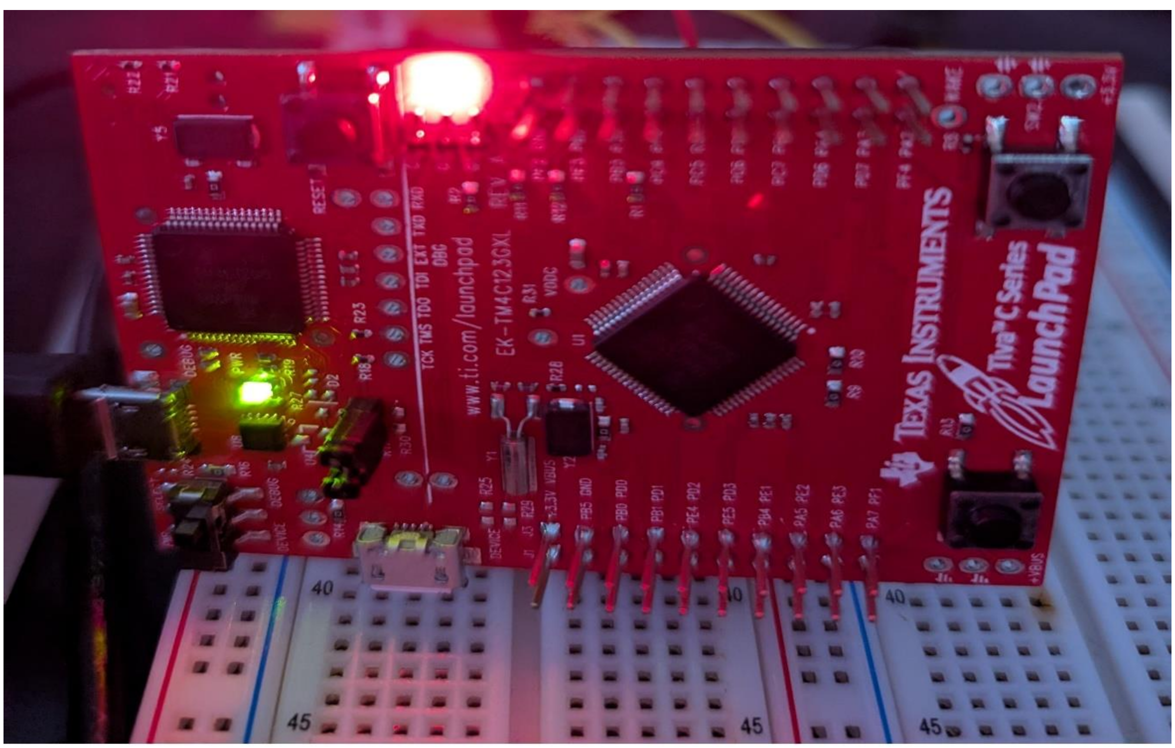

Figure 13: PWM Output at Duty Cycle of 100