Meenakshi Shankar
Sr No. 22400
M.Tech EPD

EMBEDDED SYSTEM DESIGN (E3-257)

LAB ASSIGNMENT 3

Explanation of Code

This C code is designed for the TM4C123G Tiva C Series LaunchPad, a development board based on the ARM Cortex-M4 microcontroller. Let's break down the code into key sections:

Header Files and Definitions:

- The code includes necessary header files such as `stdint.h`, `stdbool.h`, and specific Tiva C Series header files.

- It defines constants for different LED colors and their combinations.

Function Prototypes:

- `**void separate_funct(void)**`: A function that processes incoming commands from UART and extracts relevant information.

Break down of the `**separate_funct**` function:

This function processes the incoming UART command stored in `cmnd_val

1. Command Processing Loop:

- Iterates through the characters in `**cmnd_val**` until a carriage return (`'\r'`) is encountered.

- Converts alphabetic characters to lowercase and removes non-alphanumeric characters.

- Stores the cleaned command in the `**full_cmnd**` array.

```
   }
while((cmnd_val[a] != '\r'))
{
      if(!(((cmnd_val[a]>='A')&&(cmnd_val[a]<='Z')) || ((cmnd_val[a]>='a')&&(cmnd_val[a]<='z')) || ((cmnd_val[a]>='0')&&(cmnd_val[a]<='9'))))
         {
         a++;
         continue;
         }
      else
      {
         if((cmnd_val[a]>='A') && (cmnd_val[a]<='Z'))
            cmnd_val[a] = cmnd_val[a] + 32;
         full_cmnd[b] = cmnd_val[a];
         a++;
         b++;
      }
}
```

Meenakshi Shankar
Sr No. 22400
M.Tech EPD

2. Prints Extracted Command:

- Prints the extracted command via UART for debugging purposes.

```
printstring("Request: ");
for(int i = 0; i<b; i++)
        UART0_Transmitter(full_cmnd[i]);
UART0_Transmitter('\n');
UART0_Transmitter('\r');
for(int i = 0; i<30; i++)
cmnd_val[i] = '\0';
for(int i = 0; i<5; i++)
cmnd.type[i] = full_cmnd[i];
}
printstring("Option: ");
for(int i = 0; i<5; i++)
    UART0_Transmitter(cmnd.type[i]);
UART0_Transmitter('\n');
UART0_Transmitter('\r');
printstring("Value: ");
for(int i = 0; i<(b-5); i++)
{
   cmnd.data[i] = full_cmnd[i+5];
   UART0_Transmitter(cmnd.data[i]);
}
UART0_Transmitter('\n');
UART0_Transmitter('\r');
```

3. Prints Extracted Option:

- Prints the extracted command type (`cmnd.type`) via UART.

4. Validation:

- Checks if the command type is either `color` or `blink`.

- For `color`, checks if the data is a valid color.

- For `blink`, checks if the data consists of numeric characters.

- Sets the `check` flag accordingly.

5. Error Handling:

- If the command or data is invalid, prints an error message and sets the `check` flag to 0.

Meenakshi Shankar
Sr No. 22400
M.Tech EPD

- The function returns if there's an error.

This function essentially processes incoming UART commands, cleans and separates the command type and data, and performs basic validation on the input. It serves as a key part of the program's command handling logic.

- `*void printstring(char *str)*` : A function to transmit a string through UART.

- `***void UART0_Transmitter(unsigned char data)*`:** A function to transmit a byte through UART.\

- `***void delayMs(int n)*`:** A delay function, where it also checks for incoming UART characters.

The block of code you provided is part of a loop that checks for available characters in the UART0 receiver buffer using `UARTCharsAvail(UART0_BASE)`. Let's break down this code snippet:

```
if(UARTCharsAvail(UART0_BASE))
            {
                cmnd_val[cmnd_ind] = UARTCharGet(UART0_BASE);
                if(cmnd_val[cmnd_ind] == '\b')
                {
                    cmnd_val[cmnd_ind] = '\0';
                    cmnd_ind = cmnd_ind-1;
                    cmnd_val[cmnd_ind] = '\0';
                    UART0_Transmitter('\b');
                    UART0_Transmitter(' ');
                    UART0_Transmitter('\b');
                    cmnd_ind = cmnd_ind-1;
                }
                else
                {
                UART0_Transmitter(cmnd_val[cmnd_ind]);
                }
                if(cmnd_val[cmnd_ind]== '\r')
                    cmnd_cntrl = 1;
                (cmnd_ind>30)? (cmnd_ind = 0): (cmnd_ind++);
            }
```

Meenakshi Shankar
Sr No. 22400
M.Tech EPD

***Backspace Handling: ***

- Checks if the read character is a backspace (`'\b'`).

- If true:

  - Sets the current character in `cmnd_val` to `'\0'`.

  - Decrements `cmnd_ind` to move back in the array.

  - Sets the previous character to `'\0'`.

  - Sends backspace, space, and backspace characters via UART to simulate the effect of erasing the last character on the terminal.

***Normal Character Handling: ***

- If the read character is not a backspace:

  - Sends the character back to the terminal via `UART0_Transmitter`.

  - If the read character is a carriage return (`'\r'`), sets `cmnd_cntrl` to 1, indicating the end of the command.

This code segment is part of the UART input handling mechanism. It reads characters from the UART0 receiver buffer, processes backspaces, sends characters back to the terminal, and tracks the end of a command marked by a carriage return (`'\r'`). The circular buffer indexing ensures that the array is treated as a circular buffer.

Global Variables and Structures:

- `*volatile char **cmnd_val**[30]*`: An array to store incoming UART characters.

- `*volatile int **cmnd_ind***`: An index variable for the `cmnd_val` array.

- `*volatile int **cmnd_cntrl***`: A control variable indicating that a complete command has been received.

- `*volatile int **blink_mode***`: A flag to toggle between continuous LED operation and blinking.

- `*int **blink_rate***`: Variable to control the rate of LED blinking.

- `*struct **cmnd**` : A structure to store the parsed command, consisting of a type and data*.

Meenakshi Shankar
Sr No. 22400
M.Tech EPD

Main Function:

1. GPIO and Interrupt Setup:

- Configures GPIO pins for the onboard LEDs and switches (SW1 and SW2).

- Sets up interrupts for SW1 and SW2 presses.

2. UART Setup:

- Initializes UART0 for communication with a baud rate of 115200.

3. Infinite Loop:

- Waits for interrupts and continuously updates the LED color based on the `colour_mode` variable.

- Implements a delay between LED changes.

4. Interrupt Handling (GPIOF_Handler):

- Handles interrupts generated by SW1 and SW2.

- SW1 press cycles through LED colors, while SW2 press changes the blinking factor.

5.UART Input Processing (delayMs Function):

- Checks for incoming UART characters during the delay loop.

- Stores characters in `cmnd_val` and triggers processing when a carriage return ('\r') is received.

6.Command Processing (separate_funct Function):

- Parses the received command and separates it into type and data.

- Handles commands related to LED color change (`color`) and LED blinking rate (`blink`).

Meenakshi Shankar
Sr No. 22400
M.Tech EPD

Overall:

The code combines GPIO configuration, interrupt handling, UART communication, and LED control. It interprets commands received via UART to change LED colors or modify the blinking rate. The program runs in an infinite loop, reacting to interrupts and UART inputs. It demonstrates the integration of peripherals on the TM4C123G microcontroller for LED control and communication.