3-2 Milestone Two - Report

This program is designed to control the LED on the TI CC3220S LaunchPad using the UART and GPIO drivers. It reads input from the user via the UART interface and adjusts the LED based on the commands entered. Here's a breakdown of what each section of the code does:

#include <stdint.h>

#include <stddef.h>

#include <ti/drivers/GPIO.h>

#include <ti/drivers/UART2.h>

#include "ti\_drivers\_config.h"

These headers are necessary for accessing the GPIO and UART drivers. They allow the program to control the LED and communicate with the user via the UART interface.

**Main Function (mainThread)**

This function handles the core logic of the program that controls the LED via UART input.

1. **Variable Setup**:
   * char input;: This stores user input.
   * const char echoPrompt[] = "Echoing characters:\r\n";: This is the initial message displayed to the user when the program starts.
2. **Driver Initialization**:
   * GPIO\_init();: Initializes the GPIO module, enabling the program to control the LED.
3. **LED Configuration**:
   * GPIO\_setConfig(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_STD | GPIO\_CFG\_OUT\_LOW);: Configures the LED pin as an output. The LED is set to an "off" state initially.
4. **Setting Up UART**:
   * UART2\_Params\_init(&uartParams);: Initializes UART parameters.
   * uartParams.baudRate = 115200;: Sets the baud rate for UART communication to 115200, which determines the speed of data transmission.
   * uart = UART2\_open(CONFIG\_UART2\_0, &uartParams);: Opens the UART communication channel with the specified parameters.
5. **Error Handling for UART**:
   * If UART2\_open fails, the program enters an infinite loop to prevent further execution:

if (uart == NULL) {

while (1) {} // UART initialization failed

}

1. **Feedback and Initial LED Signal**:
   * The LED is turned on briefly using GPIO\_write(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_HIGH); to indicate successful program initialization.
   * UART2\_write(uart, echoPrompt, sizeof(echoPrompt), &bytesWritten);: Sends the echoPrompt message to inform the user that the program is ready for input.
2. **Main Loop**:
   * The program enters an infinite loop, waiting for user input. It reads a single character from UART using:

while (bytesRead == 0) {

status = UART2\_read(uart, &input, 1, &bytesRead);

}

1. **LED Control Based on User Input**:
   * The LED is controlled based on the input received from the user:
     + **Turn On LED**: If the user types 'O', the LED turns on using GPIO\_write(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_HIGH);.
     + **Turn Off LED**: If the user types 'F', the LED turns off using GPIO\_write(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_LOW);.
     + **Invalid Command**: Any other input results in an "Invalid Command" message (UART2\_write(uart, "\nInvalid Command.\r\n", 19, &bytesWritten);).

if (input == 'O') {

GPIO\_write(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_HIGH); // Turn ON LED

UART2\_write(uart, "\nLED is ON.\r\n", 13, &bytesWritten); // Provide feedback

} else if (input == 'F') {

GPIO\_write(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_LOW); // Turn OFF LED

UART2\_write(uart, "\nLED is OFF.\r\n", 14, &bytesWritten); // Provide feedback

} else {

UART2\_write(uart, "\nInvalid Command.\r\n", 19, &bytesWritten); // Handle invalid input

}

**Full code**

/\*

\* Copyright (c) 2020, Texas Instruments Incorporated

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* \* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* \* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\*

\* \* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

/\*

\* ======== uart2echo.c ========

\*/

**#include** <stdint.h>

**#include** <stddef.h>

/\* Driver Header files \*/

**#include** <ti/drivers/GPIO.h>

**#include** <ti/drivers/UART2.h>

/\* Driver configuration \*/

**#include** "ti\_drivers\_config.h"

/\*

\* ======== mainThread ========

\*/

**void** \***mainThread**(**void** \*arg0)

{

**char** input;

**const** **char** echoPrompt[] = "Echoing characters:\r\n";

UART2\_Handle uart;

UART2\_Params uartParams;

size\_t bytesRead;

size\_t bytesWritten = 0;

uint32\_t status = UART2\_STATUS\_SUCCESS;

/\* Call driver init functions \*/

**GPIO\_init**();

/\* Configure the LED pin \*/

**GPIO\_setConfig**(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_STD | GPIO\_CFG\_OUT\_LOW);

/\* Create a UART where the default read and write mode is BLOCKING \*/

**UART2\_Params\_init**(&uartParams);

uartParams.baudRate = 115200;

uart = **UART2\_open**(CONFIG\_UART2\_0, &uartParams);

**if** (uart == NULL)

{

/\* UART2\_open() failed \*/

**while** (1) {}

}

/\* Turn on user LED to indicate successful initialization \*/

**GPIO\_write**(CONFIG\_GPIO\_LED\_0, CONFIG\_GPIO\_LED\_ON);

**UART2\_write**(uart, echoPrompt, **sizeof**(echoPrompt), &bytesWritten);

/\* Loop forever echoing and handling commands \*/

**while** (1)

{

bytesRead = 0;

**while** (bytesRead == 0)

{

status = **UART2\_read**(uart, &input, 1, &bytesRead);

**if** (status != UART2\_STATUS\_SUCCESS)

{

/\* UART2\_read() failed \*/

**while** (1) {}

}

}

/\* Echo the character back to the terminal \*/

bytesWritten = 0;

**while** (bytesWritten == 0)

{

status = **UART2\_write**(uart, &input, 1, &bytesWritten);

**if** (status != UART2\_STATUS\_SUCCESS)

{

/\* UART2\_write() failed \*/

**while** (1) {}

}

}

/\* Control the LED based on user input \*/

**if** (input == 'O') {

**GPIO\_write**(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_HIGH); // Turn ON the LED

**UART2\_write**(uart, "\nLED is ON.\r\n", 13, &bytesWritten); // Feedback

} **else** **if** (input == 'F') {

**GPIO\_write**(CONFIG\_GPIO\_LED\_0, GPIO\_CFG\_OUT\_LOW); // Turn OFF the LED

**UART2\_write**(uart, "\nLED is OFF.\r\n", 14, &bytesWritten); // Feedback

} **else** {

**UART2\_write**(uart, "\nInvalid Command.\r\n", 19, &bytesWritten); // Invalid input feedback

}

}

}

**Summary of code**

This code reads characters from the UART interface one at a time. If the user types 'O', the LED turns on, and if they type 'F', it turns off. The code echoes each character back to the user and provides feedback about the LED state. The program runs in an infinite loop, continuously checking for new input.

**Lab Questions**

**How does the macro UART\_DATA\_BINARY impact the UART?**

* Answer: The UART\_DATA\_BINARY macro configures the UART driver to handle data as raw binary, rather than converting it to or from text formats. According to Texas Instruments (2020), this is useful for handling binary protocols or non-textual data formats because the data is transferred without any modification.

**How does the macro UART\_RETURN\_FULL impact the UART?**

* Answer: The UART\_RETURN\_FULL macro tells the UART driver to return only when the specified number of bytes is fully read or written. According to Texas Instruments (2020), it prevents partial data transfers, ensuring that all required bytes are processed in a single transaction.

**What driver call would you use to write 10 characters out of the UART?**

* Answer: The driver call to write 10 characters out of the UART would be:  
  UART2\_write(uart, data, 10, &bytesWritten);  
  This function writes exactly 10 characters from the data array and stores the number of bytes actually written in bytesWritten (Texas Instruments, 2020).

**What is the driver call to turn off LED 0?**

* Answer: The driver call to turn off LED 0 would be:  
  GPIO\_write(CONFIG\_GPIO\_LED\_0, CONFIG\_GPIO\_LED\_OFF);  
  This command writes a low signal to the LED GPIO pin, turning it off (Texas Instruments, 2020).

**What is the UART baud rate?**

* Answer: The UART baud rate is set to 115200 in this case. It can be adjusted when configuring the UART with UART2\_Params:  
  uartParams.baudRate = 115200; (Texas Instruments, 2020).

**References**

Texas Instruments. (2020). *SimpleLink™ CC32xx SDK: UART2 and GPIO Driver Guide*. Retrieved from https://www.ti.com/lit/ug/swru465c/swru465c.pdf

Texas Instruments. (2020). *CC3220 SimpleLink™ Wi-Fi® and Internet of Things Solution with MCU LaunchPad™ Development Kit*. Retrieved from https://www.ti.com/tool/CC3220SF-LAUNCHXL

Texas Instruments. (2020). *Code Composer Studio™ Integrated Development Environment (IDE)*. Retrieved from https://www.ti.com/tool/CCSTUDIO