**1)UART continuous receiving code with comments and explanation**

**#include "ti\_msp\_dl\_config.h" // Include the device-specific MSP (Microcontroller) DriverLib configuration header**

**#include "stdio.h" // Include standard input/output library for using printf**

**/\***

**\* Enable internal loopback mode.**

**\* When loopback mode is enabled, any data transmitted from the UART TX pin will**

**\* be internally routed back to the RX pin (received by the device itself).**

**\* Data from external devices connected to the RX pin will be ignored.**

**\*/**

**#define ENABLE\_LOOPBACK\_MODE true // This enables UART internal loopback mode (used for self-testing)**

**// Define the size of the UART packet (number of bytes to be transmitted/received in one loop iteration)**

**#define UART\_PACKET\_SIZE (4)**

**/\* Array to store the received data from UART.**

**\* This will hold a packet of `UART\_PACKET\_SIZE` bytes, which is 4 in this case.**

**\*/**

**uint8\_t rxPacket[UART\_PACKET\_SIZE];**

**int main(void)**

**{**

**// Initialize the system configuration. This sets up peripherals like clocks, GPIOs, and UART.**

**SYSCFG\_DL\_init();**

**// Infinite loop to continuously receive and process UART data**

**while (1)**

**{**

**// Loop through the `UART\_PACKET\_SIZE` to receive one packet of data**

**for (uint8\_t i = 0; i < UART\_PACKET\_SIZE; i++) {**

**// Receive one byte of data from UART (UART\_0\_INST is the instance of UART being used)**

**// DL\_UART\_receiveDataBlocking is a blocking function, so it waits until data is received**

**rxPacket[i] = DL\_UART\_receiveDataBlocking(UART\_0\_INST);**

**// Print the received character to the console. This displays the data received from UART.**

**// The %c format specifier prints it as a character.**

**printf("%c", rxPacket[i]);**

**}**

**// Print a newline character after receiving the packet to separate each packet in the console output.**

**printf("\n");**

**}**

**}**

Changes in the syscfg

1. In advanced config : change uart mode to **rs485** mode

EXT Driver Setup Val (Ticks) to 5

EXT Driver Setup Hold Val (Ticks) to 5

Enable fifo in

In pinmux peripheral pin config set the TX -> PA11/57

In pinmux peripheral pin config set the RX -> PA10/56

**2) Sending big data in chunks of 4 bytes**

#include "ti\_msp\_dl\_config.h"

/\*

\* Define the number of bytes for the UART packet size.

\* The maximum size of the UART FIFO buffer is 4 bytes, so we send data in chunks of 4 bytes.

\*/

#define UART\_PACKET\_SIZE (4)

/\*

\* Define the delay (in cycles) for 5ms to ensure the UART TX is idle before starting transmission.

\* This allows some time between transmissions to avoid any overlap or collision.

\*/

#define UART\_TX\_DELAY (160000)

/\*

\* Data buffer for UART transmission.

\* This array contains 14 characters that will be sent through UART.

\*/

uint8\_t txPacket[] = {'M', 'S', 'P', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'};

/\*

\* Offset variable to track the current position in the data buffer being transmitted.

\* `totalBytes` is the total number of bytes to transmit, calculated based on the size of `txPacket`.

\*/

uint8\_t offset = 0;

uint16\_t totalBytes = sizeof(txPacket); // Total bytes to send

int main(void)

{

/\* Initialize the system configuration, including peripherals like UART \*/

SYSCFG\_DL\_init();

/\* Main loop to send data continuously \*/

while(1)

{

/\* Reset offset to 0 before each transmission cycle to restart sending from the beginning \*/

offset = 0;

/\* Loop to send all the data in the `txPacket` buffer \*/

while(offset < totalBytes)

{

/\* Delay for 5ms to ensure UART TX is idle before transmission starts \*/

delay\_cycles(UART\_TX\_DELAY);

/\* Set DE (Data Enable) high before starting the transmission \*/

DL\_GPIO\_setPins(GPIO\_DE\_TX\_PORT, GPIO\_DE\_TX\_TRANSMITTER\_PIN);

/\*

\* Determine how many bytes to send in the current iteration.

\* Send up to `UART\_PACKET\_SIZE` bytes or whatever is remaining (if fewer than 4 bytes).

\*/

uint8\_t bytesToSendNow = ((totalBytes - offset) > UART\_PACKET\_SIZE) ? UART\_PACKET\_SIZE : (totalBytes - offset);

/\* Fill the UART FIFO with the calculated number of bytes from the `txPacket` starting at `offset` \*/

DL\_UART\_Main\_fillTXFIFO(UART\_0\_INST, &txPacket[offset], bytesToSendNow);

/\*

\* Wait for the UART to finish sending all bytes.

\* This checks if the UART transmission is still in progress.

\*/

while (DL\_UART\_isBusy(UART\_0\_INST))

;

/\* Set DE (Data Enable) low after completing the transmission \*/

DL\_GPIO\_clearPins(GPIO\_DE\_TX\_PORT, GPIO\_DE\_TX\_TRANSMITTER\_PIN);

/\* Move the offset forward by the number of bytes just sent \*/

offset += bytesToSendNow;

}

/\* Large delay between transmission cycles (approximately 2 seconds) \*/

delay\_cycles(32000000);

}

}

**3)** **combining the code i.e. sending and receiving part**

#include "ti\_msp\_dl\_config.h"

#include "stdio.h"

#define UART\_PACKET\_SIZE (4) // Define the size of the UART packet (4 bytes)

#define UART\_TX\_DELAY (16000000)

#define BUFFER\_SIZE (100)

// Buffer to store the received UART data

uint8\_t rxPacket[UART\_PACKET\_SIZE];

uint8\_t txPacket[UART\_PACKET\_SIZE];

uint8\_t buffer[BUFFER\_SIZE];

//uint8\_t sampleSendingArray[]= {'M','A','H','E','S','H','-',}

int main(void) {

SYSCFG\_DL\_init(); // Initialize system configuration (including peripherals)

while(1) {

// Receive UART packet of size UART\_PACKET\_SIZE

for(uint8\_t i = 0; i < UART\_PACKET\_SIZE; i++) {

rxPacket[i] = DL\_UART\_receiveDataBlocking(UART\_0\_INST); // Receive one byte of data

printf(" %c", rxPacket[i]); // Print the received byte as a character

txPacket[i] = rxPacket[i];

}

printf("\n"); // Print a newline after the packet is printed

// now the code from here to send that data that has beed received

/\* Optional delay to ensure UART TX is idle before starting transmission \*/

delay\_cycles(UART\_TX\_DELAY);

/\* Transmit data, set DE high before \*/

DL\_GPIO\_setPins(GPIO\_UART\_0\_TX\_PORT,GPIO\_UART\_0\_TX\_PIN);

DL\_UART\_Main\_fillTXFIFO(UART\_0\_INST,&txPacket[0],UART\_PACKET\_SIZE);

/\*

\* Wait until all bytes have been transmitted and the TX FIFO

\* is empty

\*/

while(DL\_UART\_isBusy(UART\_0\_INST))

;

DL\_GPIO\_clearPins(GPIO\_UART\_0\_TX\_PORT,GPIO\_UART\_0\_TX\_PIN);

}

}

**4) ADC for getting digital value for analog voltage**

1. Things to do in sysconfig
   1. In basic configuration change the sample clock divider to **divide by 8**
   2. In interrupt configuration set enable interrupt to **MEM0 result loded interrupt**
2. **Code for ADC**

**#include "ti\_msp\_dl\_config.h"**

**#include "stdio.h"**

**volatile uint8\_t flag = 0;**

**uint16\_t result = 0 ;**

**int main(void){**

**SYSCFG\_DL\_init();**

**NVIC\_EnableIRQ(ADC12\_0\_INST\_INT\_IRQN);**

**while(1){**

**DL\_ADC12\_startConversion(ADC12\_0\_INST);**

**if(flag == 0){**

**\_\_WFI();**

**}**

**result = DL\_ADC12\_getMemResult(ADC12\_0\_INST, DL\_ADC12\_MEM\_IDX\_0) ;**

**printf("%d \n",result);**

**DL\_ADC12\_disableConversions(ADC12\_0\_INST);**

**flag = 0;**

**DL\_ADC12\_enableConversions(ADC12\_0\_INST);**

**}**

**}**

**void ADC12\_0\_INST\_IRQHandler(void){**

**switch(DL\_ADC12\_getPendingInterrupt(ADC12\_0\_INST)){**

**case DL\_ADC12\_IIDX\_MEM0\_RESULT\_LOADED :**

**flag = 1 ;**

**break;**

**default:**

**break ;**

**}**

**}**

1. **DAC code for getting equivalent voltage for corresponding digital value**

**Changes need to do in syscfg file for DAC**

1. **Go to DAC and enable DAC**
2. **Change the positive reference voltage to VREF+**
3. **Change the negative reference voltage to VREF-**
4. **In DMA configuration enable DAC output**
5. **In advance configuration change the DAC and output amplifier to -> output and input amplifier on**
6. **In other dependencies -> MFCLK(middle frequency precision clock ) -> enable MFCLK**
7. **Go to vref from the side bar not in DAC and change the internal voltage to 2.5 volt**

**Code for DAC**

**#include "ti\_msp\_dl\_config.h"**

**#include "stdio.h"**

**#define vrefVoltage (2500)**

**#define digitalValue (1234)**

**#define maxDigitalValue (4095)**

**uint16\_t result = 0;**

**int main(void){**

**SYSCFG\_DL\_init();**

**result = (vrefVoltage \* digitalValue) / maxDigitalValue;**

**printf("%d \n",result);**

**DL\_DAC12\_output12(DAC0,digitalValue ) ; // remember to write the DAC0**

**DL\_DAC12\_enable(DAC0); // and here as well**

**while(1){**

**\_\_WFI();**

**}**

**}**

**6) SPI send packet**

Things that needs to change in the config file in spi

1) set target bit rate to 500000

In spi controller select the default chip select to cs1

2) in spi poci select chip select 1 (cs1) and enable its pin config

2) in pinmux config select

1) spi peripheral to SPI1

2) SPI(sclk) PB9/61

3) SPI (PICO) PB8/60

4) SPI(POCI ) PB7/59

5) SPI (CS1) to PB17/ 14

// optional 6) in pin config (cs1) -> in digital iomux feature set the internal resistor to -> **pull up resistor**

**6)** if we are using interrupt for receiving then enable the **receive** interrupt from interrupts

**Function for sending the Data DL\_SPI\_fillTXFIFO8(inst,&txPacket[0],size);**

**Function for receiving data through SPI DL\_SPI\_receiveDataBlocking8(inst);**

**\*\*\* Remember to write 8 at both sending and receiving function**

### **SPI simple code for sending and receiving**

**/\* code for sending 4 bytes of data to the spi peripheral \*/**

**#include "ti\_msp\_dl\_config.h"**

**#include "stdio.h"**

**#define SPI\_PACKET\_SIZE (4)**

**// uart transmission array to send data to peripheral**

**uint8\_t txPacket[SPI\_PACKET\_SIZE] = {'M','S','P','!'} ;**

**// empty decleration of rx array for storing the received data**

**uint8\_t rxPacket[SPI\_PACKET\_SIZE] ;**

**int main(void){**

**SYSCFG\_DL\_init();**

**/\* sending the data to the peripheral \*/**

**DL\_SPI\_fillTXFIFO8(SPI\_0\_INST,&txPacket[0],SPI\_PACKET\_SIZE);**

**while(DL\_SPI\_isBusy(SPI\_0\_INST))**

**;**

**// receiving the data from peripheral**

**for (uint8\_t i = 0; i < SPI\_PACKET\_SIZE ; i++ ){**

**rxPacket[i] = DL\_SPI\_receiveDataBlocking8(SPI\_0\_INST);**

**}**

**}**

**SPI simple code for sending and receiving using interrupt**

**#include "ti\_msp\_dl\_config.h"**

**#include "stdio.h"**

**#define SPI\_PACKET\_SIZE (4)**

**volatile uint8\_t flag = 0;**

**uint8\_t txPacket[SPI\_PACKET\_SIZE] = { 1, 2, 3, 4 };**

**uint8\_t rxPacket[SPI\_PACKET\_SIZE];**

**int main(void)**

**{**

**SYSCFG\_DL\_init();**

**NVIC\_EnableIRQ(GPIO\_GRP\_0\_INT\_IRQN);**

**NVIC\_EnableIRQ(SPI\_0\_INST\_INT\_IRQN);**

**while (1)**

**{**

**\_\_WFI();**

**if (flag == 1)**

**{**

**for (int i = 0; i < SPI\_PACKET\_SIZE; i++)**

**{**

**printf("%c", rxPacket[i]);**

**}**

**printf("\n");**

**flag = 0;**

**}**

**}**

**}**

**void GROUP1\_IRQHandler(void)**

**{**

**switch (DL\_GPIO\_getPendingInterrupt(GPIO\_GRP\_0\_PORT))**

**{**

**case DL\_GPIO\_IIDX\_DIO18:**

**DL\_SPI\_fillTXFIFO8(SPI\_0\_INST, &txPacket[0], SPI\_PACKET\_SIZE);**

**while (DL\_SPI\_isBusy(SPI\_0\_INST))**

**{**

**;**

**}**

**break;**

**default:**

**break;**

**}**

**}**

**void SPI1\_IRQHandler(void)**

**{**

**switch (DL\_SPI\_getPendingInterrupt(SPI\_0\_INST))**

**{**

**case DL\_SPI\_IIDX\_RX:**

**for (uint8\_t i = 0; i < SPI\_PACKET\_SIZE; i++)**

**{**

**rxPacket[i] = DL\_SPI\_receiveDataBlocking8(SPI\_0\_INST);**

**}**

**flag = 1 ;**

**break;**

**default :**

**break ;**

**}**

**}**

**7) PWM changing the intensity of the led continuously**

**Changes need to do in the configuration**

1. **Change the PWM period count to 200000**
2. **In pinmux change timer peripheral to TIMG12**
3. **In pinmux -> compare pin 0 -> pB13/1**
4. **In pinmux -> compare pin 1 -> pB31/39**

**#include "ti\_msp\_dl\_config.h"**

**#include "stdio.h"**

**int main(void)**

**{**

**// Initialize system configuration and peripherals**

**SYSCFG\_DL\_init();**

**// Start the PWM timer**

**DL\_TimerG\_startCounter(PWM\_0\_INST);**

**while (1) // Infinite loop to continuously modify PWM signal**

**{**

**// First loop: Gradually decrease the PWM duty cycle**

**for (int i = 200000; i >= 0; i -= 20000)**

**{**

**// Set the PWM duty cycle value by adjusting the compare value**

**DL\_TimerG\_setCaptureCompareValue(PWM\_0\_INST, i, DL\_TIMER\_CC\_0\_INDEX);**

**// Print "decrement" to indicate the decrementing phase**

**printf("decrement \n");**

**// Introduce a delay to slow down the transition of PWM values**

**delay\_cycles(16000000); // Delay for approximately 1 second (based on clock cycles)**

**}**

**// Second loop: Gradually increase the PWM duty cycle**

**for (int i = 0; i <= 200000; i += 20000)**

**{**

**// Set the PWM duty cycle value by adjusting the compare value**

**DL\_TimerG\_setCaptureCompareValue(PWM\_0\_INST, i, DL\_TIMER\_CC\_0\_INDEX);**

**// Print "increment" to indicate the incrementing phase**

**printf("increment \n");**

**// Introduce a delay to slow down the transition of PWM values**

**delay\_cycles(16000000); // Delay for approximately 1 second (based on clock cycles)**

**}**

**}**

**}**

**8) Timer one shot up and down counting**

**#include "ti\_msp\_dl\_config.h"**

**#include "stdio.h"**

**int main(void){**

**SYSCFG\_DL\_init();**

**NVIC\_EnableIRQ(TIMER\_0\_INST\_INT\_IRQN);**

**DL\_TimerA\_startCounter(TIMER\_0\_INST);**

**}**

**void TIMER\_0\_INST\_IRQHandler (void){**

**switch(DL\_TimerA\_getPendingInterrupt(TIMER\_0\_INST)){**

**case DL\_TIMERA\_IIDX\_ZERO :**

**DL\_GPIO\_togglePins(GPIOB,GPIO\_GRP\_0\_LED1\_PIN);**

**DL\_TimerA\_stopCounter(TIMER\_0\_INST);**

**// start the counter again to toggle the led**

**DL\_TimerA\_startCounter(TIMER\_0\_INST);**

**}**

**}**

**Changes need to be made in configuration**

**/\* read me**

**\* in configurations change the timer to periodic up or down counting**

**\* use the clock LFCLK**

**\* set the timer period and if more than 1 second then use divide by three or 4**

**\* change interrupt to load event or zero event**

**\***

**\*/**

**9) I2C controller code and config**

**Changes need to be made in the syscfg for controller**

1. **In basic configuration enable the controller mode**
2. **Set the custom bus speed to 400000**
3. **In advance config -> enable-> enable i2c after aftr initialization**
4. **In advance config -> disable analog glitch filter**
5. **In pinmux -> set i2c serial data line (SDA) -> PB3**
6. **In pinmux -> set i2c serial clockline (SDL) -> PB2**
7. **In case of slave or target device**

**1 ) in basic configuration enable target mode**

**2) in I2C target config -> enable target’s own address**

**3) in I2C target config -> set target address to 0x48**

**4) In advance config -> enable-> enable i2c after aftr initialization**

**5)**

**Algorithm for I2C controller   
The logic flow between the two codes is almost identical:**

* **Fill the TX FIFO**
* **Start the controller transfer (TX)**
* **Delay between transmission and reception**
* **Start the controller transfer (RX)**
* **Receive data from RX FIFO**
* **Toggle GPIO pin for LED**
* **Add delay between iterations**

**Code for I2C controller**

**#include "ti\_msp\_dl\_config.h"**

**#include "stdio.h"**

**#define I2C\_TX\_PACKET\_SIZE (8)**

**#define I2C\_RX\_PACKET\_SIZE (5)**

**#define I2C\_TARGET\_ADDRESS (0x48)**

**/\* Data to send to the Target \*/**

**uint8\_t gTxPacket[I2C\_TX\_PACKET\_SIZE] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,**

**0x07, 0x08 };**

**/\* Data received from Target \*/**

**volatile uint8\_t gRxPacket[I2C\_RX\_PACKET\_SIZE];**

**void delay\_cycles(uint32\_t cycles)**

**{**

**while (cycles-- > 0)**

**{**

**\_\_asm("NOP");**

**}**

**}**

**int main(void)**

**{**

**SYSCFG\_DL\_init();**

**while (1)**

**{**

**/\* Fill FIFO and send the packet to the target \*/**

**DL\_I2C\_fillControllerTXFIFO(I2C\_INST, gTxPacket, I2C\_TX\_PACKET\_SIZE);**

**DL\_I2C\_startControllerTransfer(I2C\_INST, I2C\_TARGET\_ADDRESS,**

**DL\_I2C\_CONTROLLER\_DIRECTION\_TX,**

**I2C\_TX\_PACKET\_SIZE);**

**/\* Wait for the transmission to complete \*/**

**delay\_cycles(100000); // Small delay between transmission and reception**

**/\* Receive data from target \*/**

**DL\_I2C\_startControllerTransfer(I2C\_INST, I2C\_TARGET\_ADDRESS,**

**DL\_I2C\_CONTROLLER\_DIRECTION\_RX,**

**I2C\_RX\_PACKET\_SIZE);**

**for (uint8\_t i = 0; i < I2C\_RX\_PACKET\_SIZE; i++)**

**{**

**while (DL\_I2C\_isControllerRXFIFOEmpty(I2C\_INST))**

**;**

**gRxPacket[i] = DL\_I2C\_receiveControllerData(I2C\_INST);**

**printf("%c ", gRxPacket[i]);**

**}**

**printf("\n");**

**/\* Toggle LED after successful transfer \*/**

**DL\_GPIO\_togglePins(GPIO\_LEDS\_PORT, GPIO\_LEDS\_USER\_LED\_1\_PIN);**

**/\* Add delay between iterations to avoid overwhelming the I2C bus \*/**

**delay\_cycles(16000000);**

**}**

**}**