Embedded Systems
Postlab 5

By Rajendra Singh (111601017), final year, CSE, IIT PALAKKAD

6th September 2019

Objective

To familiarize the basics of serial communication by configuring and using UART of FRDM-KL25Z microcontroller in C++ language.

Abstract

Communication can be either parallel or serial. In serial communication data is transferred sequentially bit-by-bit along a channel but in parallel communication multiple bits are sent simultaneously over multiple channels. Serial communication can be synchronous or asynchronous. In synchronous communication the transmitter and receiver are synchronised by a common clock in contrast to asynchronous communication where both the transmitter and receiver have clock generators that must be configured to run at the same speed. Asynchronous communication uses a single transmitting channel to send one byte of data at a time at a specified rate, known as the baud rate which is measured in bits per second. In embedded systems and industrial controls, asynchronous serial communication is still very common and useful. Most microcontrollers feature UART (Universal Asynchronous Receiver/ Transmitter) peripherals internal to the microcontroller.

In this lab, we will try to setup Serial communication between the FRDM-KL25Z and Tera Term. Through UART0 of the microcontroller, we will transmit and receive data through Tera Term.

Introduction

- The PLL/FLL clock select (PLLFLLSEL) must be set to 1(MCGPLLCLK clock with fixed divide by two) in the SIM_SOPT2 register. The maximum possible frequency of MCGFLLCLK in FRDM KL25Z is 48MHz. This means the UART0 clock frequency will therefore be 48MHz/2=24MHz.

- The UART0 clock source select bits (UART0SRC) must be set in the SIM_SOPT2 register. These bits are set to ‘01’ =MCGFLLCLK clock or MCGPLLCLK/2 clock. This means either the MCGFLLCLK (48MHz on the KL25Z) or MCGPLLCLK/2 (24MHz) will be the clock that drives UART0.

- The UART0 Clock Gate Control (UART0) in the SIM_SCGC4 register must be enabled.

- The Pin Mux Control (MUX) bits of PORTA_PCR1 and PORTA_PCR2 must be set to ‘010’ = Alternative 2. Alternative 2 on these pins is UART0_RX for PTA1 and UART0_TX for PTA2.

- The Oversampling Ratio (OSR) size must be set in UART0_C4.

- The Baud Rate Modulo Divisor (SBR) must be set in the UART0_BDH and UART0_BDL registers based on the desired baud rate and the clock settings. The SBR is a 13 bit long field split between UART0_BDH and UART0_BDL. The lower 8 bits (SBR[7:0]) are in UART0_BDL and the upper 5 bits (SBR[12:9]) are the lowest bits in UART0_BDH.

*SBR = clock_rate/(OSR * baud_rate) clock_rate = 24MHz (based on the settings for UART0SRC and PLLFLLSEL) OSR = 16*

- Eg: For baud_rate = 115200 bps,

*SBR = 24000000/(16 * 115200) ~ 13 = 0x0D UART0_BDH = 0x00 UART0_BDL = 0x0D*

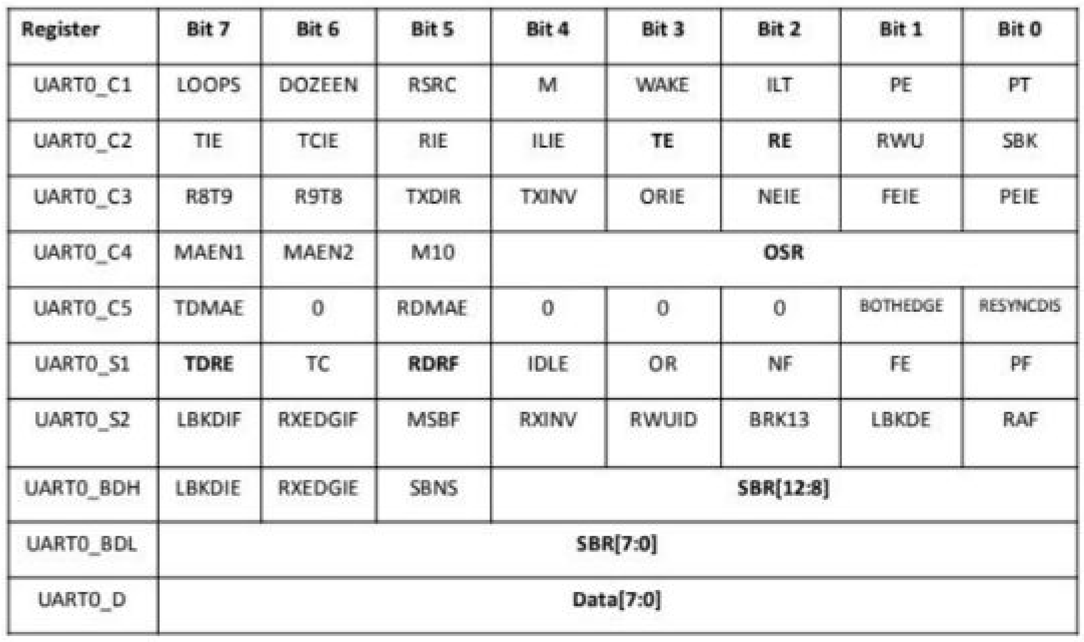
- The Transmitter Enable (TE) and Receiver Enable (RE) bits in UART0_C2 must be set to 1 to enable the transmitter and receiver respectively.

- Data can be transmitted by writing to the UART Data Register (UART0_D). Data should only be written to the UART if the transmitter is not busy. The status of the transmitter can be monitored with the Transmit Data Register Empty Flag (TDRE) in UART Status Register 1 (UART0_S1). When TDRE is 1, the transmitter can be written to.

- Received data can be read from the UART Data Register (UART0_D). Data is available when the Receive Data Register Full Flag (RDRF) in UART0_S1 is 1. Data should only be read from UART0_D after verifying if RDRF bit is 1. Data will be transmitted and received in the ASCII format.

- The UART can experience errors which will lock up the receiver until they are cleared. They are indicated by the OR (Receiver Overrun Flag), NF (Noise Flag), FE (Framing Error Flag) and PE (Parity Error Flag) in the UART0_S1 register. If these bits are set, they must be cleared by writing a 1 to the corresponding bit field before a character can be successfully read from the UART.

Summary of UART registers is shown below:



Register	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
UART0_C1	LOOPS	DOZEEN	RSRC	M	WAKE	ILT	PE	PT
UART0_C2	TIE	TCIE	RIE	ILIE	**TE**	**RE**	RWU	SBK
UART0_C3	R8T9	R9T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
UART0_C4	MAEN1	MAEN2	M10	**OSR**				
UART0_C5	TDMAE	0	RDMAE	0	0	0	BOTHEDGE	RESYNCDIS
UART0_S1	**TDRE**	TC	**RDRF**	IDLE	OR	NF	FE	PF
UART0_S2	LBKDIF	RXEDGIF	MSBF	RXINV	RWUID	BRK13	LBKDE	RAF
UART0_BDH	LBKDIE	RXEDGIE	SBNS	**SBR[12:8]**				
UART0_BDL	**SBR[7:0]**							
UART0_D	**Data[7:0]**							

EXERCISE 1

This aim of this exercise is to transmit data using UART0

CODE:

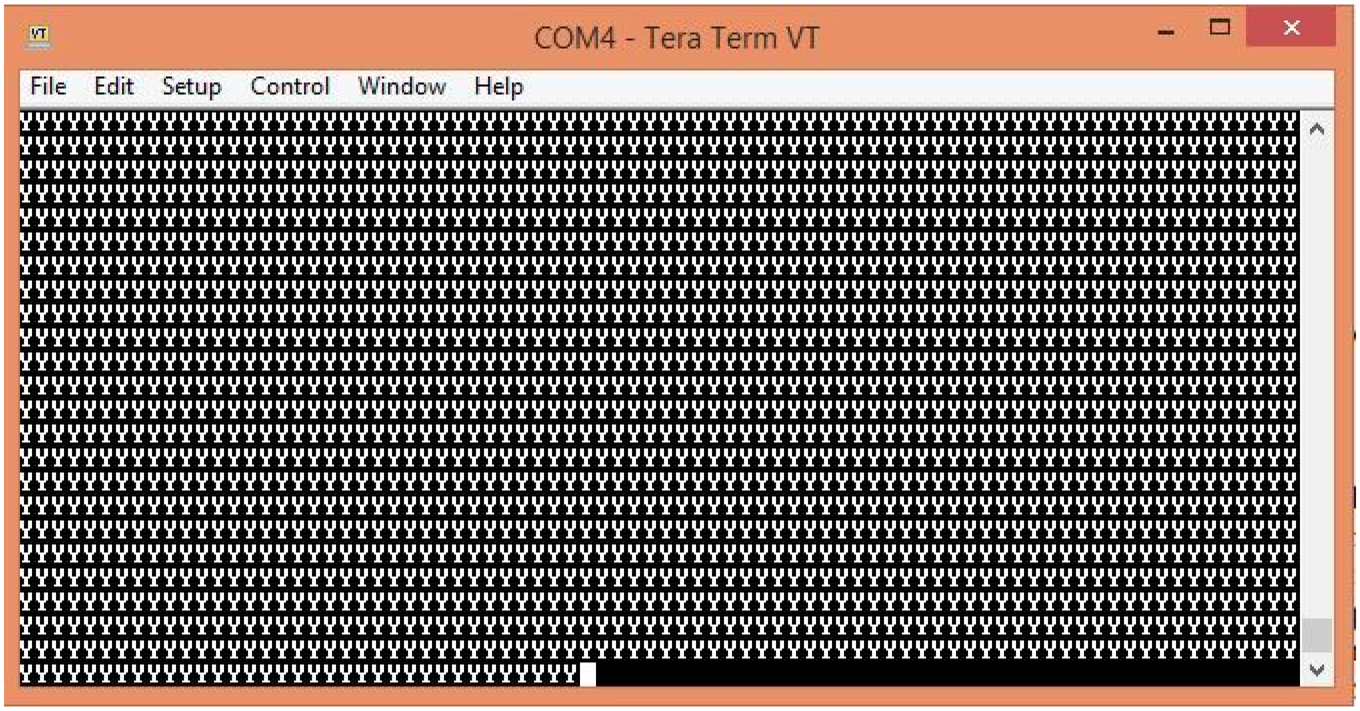
```
//Transmitting Data 'Y' using UART0
#include <MKL25Z4.H>

void UART0_init(void);

int main (void)
{
        UART0_init();                                          //Initiating UART0 as transmitter
        while (1)                                              //Infinite Loop
                {
                        while(!(UART0->S1 & 0x80)) {}
                        //In a loop Till transmitting data buffer is not empty
                        UART0->D = 'Y';                        //Transmitting 'Y' via UART0
                }
}
void UART0_init(void)
{
        SIM->SCGC4 |= 0x0400;          // enable clock for UART0
        SIM->SOPT2 |= 0x04000000;          // Selecting MCGFLLCLK clock or MCGPLLCLK/2 as clock sou

        // Transmitter, Receiver disabled
        UART0->C2 = 0;
        // Baudrate updated
        UART0->BDH = 0x00;
        UART0->BDL = 0x0D;
        // Setting OverSampling Ratio 01111
        UART0->C4 = 0x0F;
        // Baudrate divisor
        // Baudrate = Baud Clock/ ((OSR+1)xBR)
        // UART _RX and UART _TX use separate pins
        UART0->C1 = 0x00;
        //Transmitter enabled & Receiver disabled
        UART0->C2 = 0x08;
        // Clock for PORT A Enabled
        SIM->SCGC5 |= 0x0200;
        // MUXing PORT A to use as UART
        PORTA->PCR[2] = 0x0200;
}
```

Analysis:

In this example we transmit data ‘Y’ at a baud rate of 11520 bps. The data is transmitted only when the data buffer is empty i.e. **TDRE** bit of S1 register of UART0 has the logic ‘0’

Firstly, UART0 is initialized by enabling the clock, disabling Transmitter & Receiver before setting the baud rate. Then transmitter is enabled. Further, clock for PortA is enabled and muxed to use it as UART

Output:



```
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

EXERCISE 2

This aim of this exercise is to flashes the on-board LED red on receiving data ‘g’ and turn on green LED for other ASCII value.

CODE:

```
//Running LED based on the character received via UART0

#include <MKL25Z4.H>


void UART0_init(void);

void LED_init(void);

void LED_set(char value);


int main (void)

{

        char c;

        UART0_init();                          // Initiating UART0 as receiver

        LED_init();                            // Initiating PTB18, PTB19 for turning on LED

        while (1) {

                while(!(UART0->S1 & 0x20)) {}  //Infinite loop till receiving data buffer is not empty

                c = UART0->D;                          //Receiving Values from Tera Terminal

                LED_set(c);                    //Running LED based on the the data received

        }

}


void UART0_init(void)

{

        SIM->SCGC4 |= 0x0400;                  // enable clock for UART0

        SIM->SOPT2 |= 0x04000000;           // Selecting MCGFLLCLK clock or MCGPLLCLK/2 as clock
source
```

```
        // Transmitter, Receiver disabled
        UART0->C2 = 0;
        // Baudrate updated
        UART0->BDH = 0x00;
        UART0->BDL = 0x18;
        // Setting OverSampling Ratio 01111
        UART0->C4 = 0x0F;
        // Baudrate divisor
        // Baudrate = Baud Clock/ ((OSR+1)xBR)
        //****************************************************//
        // UART _RX and UART _TX use separate pins
        UART0->C1 = 0x00;
        //Transmitter disabled & Receiver enabled
        UART0->C2 = 0x04;
        // Clock for PORT A Enabled
        SIM->SCGC5 |= 0x0200;
        // MUXing PORT A to use as (PCR1 - 010)
        PORTA->PCR[1] = 0x0200;
}

// Initiating PIN18 & PIN19 of PORTB to run red & Green Led respectively
void LED_init(void)
{
        SIM->SCGC5 |= 0x400;                    // enable clock to Port B
        //Initiating RED LED
        PORTB->PCR[18] = 0x100;                 // MUXing PORT B to use as (PCR18 - 001)
        PTB->PDDR |= 0x40000;                   //Setting Pin 18 as input and taking XOR
        PTB->PSOR = 0x40000;                    //Corresponding bit 18 in PDORB is set to logic 1.
        //Initiating GREEN LED
        PORTB->PCR[19] = 0x100;                 // MUXing PORT B to use as (PCR19 - 001)
        PTB->PDDR |= 0x80000;                   //Setting Pin 19 as input and taking XOR
        PTB->PSOR = 0x80000;                    //Corresponding bit 19 in PDORB is set to logic 1.
}

void LED_set(char value)
{
        if (value =='g')
        {
                //Red LED OFF
                PTB->PSOR = 0x40000;            //Corresponding bit 18 in PDORB is cleared to logic 1
                //Green LED ON
                PTB->PCOR = 0x80000;            //Corresponding bit 19 in PDORB is cleared to logic 0
        }
        else
        {
                //Green LED OFF
                PTB->PSOR = 0x80000;            //Corresponding bit 19 in PDORB is set to logic 1.
                //Red LED ON
                PTB->PCOR = 0x40000;            //Corresponding bit 18 in PDORB is cleared to logic 0
        }
 }
```

Analysis:

In this example, we are turning on LED based on the data we are receiving. In case of *'g'* , green LED turns on whereas for other ASCII values, red LED turns on. The data is transmitted only when the data buffer is empty i.e. **RDRE** bit of S1 register of UART0 has the logic '0'

Firstly, UART0 is initialized by enabling the clock, disabling Transmitter & Receiver before setting the baud rate. Then Receiver is enabled. Further, clock for PortA is enabled and muxed to use it as UART. Afterwards, LED pins are initialized i.e. Pin18, Pin19 of PortB

For other values, first Green LED is turned off. But, it was not so for data 'g'. Then LED became yellow which is a combination of red and green. To remove this difference, Red LED is turned off.

We are receiving data at a baud rate of 57600 bps.

Output:

Red Light

Yellow Light

Green Light

Conclusion

It was very interesting to learn the commands related to transmitting and receiving data. Understanding baudate and how the transmission and receiving actually occurs was very fascinating. Well, it so happens that it occurs only for ASCII value. So, hexadecimal was converted into ASCII number. For transmitting and receiving both, the PortA pins are different.

Reference

1) Google
2) **Cortex M0+ Generic User's Guide**
3) **Cortex M0+ Technical Reference Manual**