Serial Communication (UART) example in Kinetis Development Software (KDS) with FDRM-K64F

By:
Paul Garate
Augusto Panecatl

Description

In this document you will find a detailed step by step guide of how to configure the UART module on Kinetis K devices in Kinetis Design Studio, this simple code shows how to establish communication between the computer using a serial terminal and your freedom board.

1. Clock Gating

First of all, we need to enable the clock gate corresponding to the ports we will use. In this case we will use the UART0(SIM_SCGC4) and PORTB(SIM_SCGC5)

System Clock Gating Control Register 5 (SIM_SCGC5)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0													1	0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0		PORTE	PORTD	PORTC	PORTB	PORTA	1		0	0	0	0		1	LPTMR
W																
Reset	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0

10 PORTB	Port B Clock Gate Control This bit controls the clock gate to the Port B module. 0 Clock disabled 1 Clock enabled
9 PORTA	Port A Clock Gate Control This bit controls the clock gate to the Port A module. 0 Clock disabled 1 Clock enabled

SIM_SCGC5_PORTn_MASK are defined as mask to enable the module's clock, where "n" corresponds to the specific GPIO PORT we want to activate, i.e:

SIM_SCGC5 = SIM_SCGC5_PORTB_MASK;

SIM_SCGC4_UARTn_MASK are defined as mask to enable the module's clock, where "n" corresponds to the specific UART we want to activate, i.e:

SIM_SCGC4 = SIM_SCGC4_UART0_MASK;

By declaring the mask we are writing 0x400 to the SIM_SCGC5 register, setting up the 10th bit of the System Clock Gating Control Register 5 which enables Port B; since the UART0_RX and UART0_TX in the FRDM-K64 board are assigned to Alternative 3 pins in the B ports we need to enable the clock gating to the port and enable the UART0 clock.

```
SIM_SCGC4 |= SIM_SCGC4_UART0_MASK;      /*Enable the UART clock*/
SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;     /*Enable the PORTB clock*/
```

2. Pin Control Register configuration

Once the clock gating has been setup we need to configure the pin function using the multiplexor, according to the FRDM-K64's schematic the UART0_TX and UART0_RX is assigned to pins:

144 LQFP	144 MAP BGA	121 XFBGA	100 LQFP	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	EzPort
95	E10	B10	62	PTB16	DISABLED		PTB16	SPI1_SOUT	UART0_RX	FTM_ CLKIN0	FB_AD17	EWM_IN		
96	E9	E9	63	PTB17	DISABLED		PTB17	SPI1_SIN	UART0_TX	FTM_ CLKIN1	FB_AD16	EWM_OUT_ b		

The next step is to configure the Pin Control Register to define the pin function:

Pin Control Register n (PORTx_PCRn)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LK	0				MUX			0	DSE	ODE	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	*	*	*	0	*	0	*	0	*	*	*

The bits we need to configure are those assigned to the MUX field, the pins need to be set as Alternative 3.

Field	Description
10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).

According to the board's schematic we need to configure the following pins as Alternative 3: ***UART0_TX= PortB pin 17, UART_TX= PortB pin 16.***

In configuration register ***PORTx_PCRn*** "***x***" corresponds to the port whilst "***n***" corresponds to the pin

```
PORTB_PCR16 |= PORT_PCR_MUX(3);
PORTB_PCR17 |= PORT_PCR_MUX(3);
```

*(PORT_PCR_MUX(3) = 0x300)

3. Configuring the UART module

First we need to disable the Transmisor and Recive byr writing in Control Register 2 and set the Control Register 1 with its default value.

UART Control Register 2 (UART*x*_C2)

Bit	7	6	5	4	3	2	1	0
Read Write	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

Field	Description
3 TE	Transmitter Enable Enables the UART transmitter. TE can be used to queue an idle preamble by clearing and then setting TE. When C7816[ISO_7816E] is set/enabled and C7816[TTYPE] = 1, this field is automatically cleared after the requested block has been transmitted. This condition is detected when TL7816[TLEN] = 0 and four additional characters are transmitted. 0 Transmitter off. 1 Transmitter on.
2 RE	Receiver Enable Enables the UART receiver. 0 Receiver off. 1 Receiver on.

Writing the next line, you will disable the TE and RE with the following masks

```
UART0_C2 &= ~(UART_C2_TE_MASK | UART_C2_RE_MASK );  /*Disable Tx and Rx*/
```

UART_C2_TE_MASK = 0x8 = 1000, If you deny the mask you will write 0 instead 1.
UART_C2_RE_MASK = 0x4 = 0100, If you deny the mask you will write 0 instead 1.

Writing 0 in the Control Register 1 the register will be set as Default mode.

```
UART0_C1 = 0;              /*Default settings of the register*/
```

The next thing to do is calculate the baud rate with the next formula:

UART baud rate = UART module clock/(Baud Rate * 16)

```
ubd = (uint16_t)((21000*1000)/(9600 * 16));  /* Calculate baud settings */
```

Once the baud rate has been calculated we need to configure the ***UARTx_BDH*** and ***UARTx_BDL*** registers with the necessary alues to obtain such baud rate.

UART Baud Rate Registers: High (UARTx_BDH)

Address: Base address + 0h offset

Bit	7	6	5	4	3	2	1	0
Read Write	LBKDIE	RXEDGIE	SBNS	SBR				
Reset	0	0	0	0	0	0	0	0

UART Baud Rate Registers: Low (UARTx_BDL)

ddress: Base address + 1h offset

Bit	7	6	5	4	3	2	1	0
Read Write	SBR							
Reset	0	0	0	0	0	1	0	0

This registers control the prescale divisor for UART baud rate generation and the only field to write is SBR in both register.

We will be setting the SBR with the next value = 0x88.

```
temp = UART0_BDH & ~(UART_BDH_SBR(0x1F));/*Save the value of UART0_BDH except SBR*/
UART0_BDH = temp | (((ubd & 0x1F00) >> 8));
UART0_BDL = (uint8_t)(ubd & UART_BDL_SBR_MASK);
```

The UART0_BDH is taking 0 as value, ubd = 0x88 & 0x10FF = 0x88. Shifting 0x88 >> 8 = 0.
The UART0_BDL is taking 0x88 as value, ubd = 0x88 & 0xFF = 0x88.

*UART0_BDL_SBR_MASK = 0xFF

The last thing to do is enabling the Transmitter and Receiver, just as we disable the Tx and Rx in the first step of the configuration:

```
UART0_C2 |=(UART_C2_TE_MASK |UART_C2_RE_MASK); /* Enable receiver and transmitter*/
```

UART0_C2 | = (0x8 | 0x4).

4. Code

The code includes 3 different functions:

```
void put(char *ptr_str) to send text strings:

  put("\r\nSerial code example\r\n");    /*Print the text*/
```

This function will call the next function (***uart_putchar***) sending the content of a pointer which points to the string.

```
void put(char *ptr_str)
      {
      while(*ptr_str)
      uart_putchar(*ptr_str++);
      }
```

The `void uart_putchar (char ch)` will receive the content of the pointer and transfer that value in UARTx_D which returns the contents of the read-only receive data registered and writes go to the write-only transmit data register.

```
void uart_putchar (char ch)
{
      /* Wait until space is available in the FIFO */
      while(!(UART0_S1 & UART_S1_TDRE_MASK));
      /* Send the character */
      UART0_D = (uint8_t)ch;
}
```

The last function waits for a character to be written in the serial terminal, when done the value is saved in the char variable "ch", and right next the function will call `uart_putchar(ch),` sending the variable ch with the value that you wrote.

```
uint8_t uart_getchar ()
{
      /* Wait until character has been received */
      while (!(UART0_S1 & UART_S1_RDRF_MASK));
      /* Return the 8-bit data from the receiver */
      return UART0_D;
}
```

The main code is sent to the serial terminal "Serial code example" using the put function, then it runs an infinite while cycle which calls **uart_getchar**() and then waits for a character to be written, it saves the character in the "ch" variable, calls the **uart_putchar**(ch) by sending the character to the serial terminal making the echo.

```
put("\r\nSerial code example\r\n");     /*Print the text*/

while(1)
      {
      ch = uart_getchar();
      uart_putchar(ch);
      //BLUE_TOGGLE;
      }
return 0;
```