Computer Systems 245 - Practical 8
Hints and clarifications

2014-10-02

Last revision: Thursday 1st October, 2015 at 16:55

1 Baud rate calculation

IMPORTANT:
The project has been set up to use a main cpu clock frequency $F_{CLK} = 12MHz$. This value does not appear in the code, but only in the settings of the project in e2 studio.
According to the text book (Dean, section 8.3), UART2 uses serial array unit 1, channel 1 and 0. We might need to know this when looking for the correct registers to program.

The baud rate at which the UART on the RL78 functions, is given by the formula on page 614 of the Hardware Manual.

(Baud rate) = Operation clock (F_{MCK}) frequency of target channel ÷ (SDRmn[15:9] + 1) ÷ 2 [bps]

As you'll see here, the baud rate is determined by two factors. The frequency at which the hardware is functioning (F_{MCK}), as well as the value programmed into the register SDR.

F_{MCK} is again determined by the frequency at which the CPU is operating (F_{CLK}), as well as some other dividers programmed into certain registers. The path taken to generate F_{MCK} is shown on page 188 of Dean. The most important register to look at is SPS, containing the PRS bits. It is described on page 615 of the Hardware Manual.

Depending on the setting of the CKS bit in the SMR register, different PRS bits are applicable. If we look at line 92 and 99 of r_cg_serial.c, we will note that CKS10 and CKS11 are both set to 0. Only the top half of table 12-4 on page 615 in the Hardware Manual is applicable.

More details on the CKS bits and the SMR register can be found on page 497 of the Hardware Manual.

```
SMR10 = _0020_SAU_SMRMN_INITIALVALUE | _0000_SAU_CLOCK_SELECT_CK00 |
        _0000_SAU_TRIGGER_SOFTWARE | _0002_SAU_MODE_UART |
        _0000_SAU_TRANSFER_END;

SMR11 = _0020_SAU_SMRMN_INITIALVALUE | _0000_SAU_CLOCK_SELECT_CK00 |
        _0100_SAU_TRIGGER_RXD | _0000_SAU_EDGE_FALL |
        _0002_SAU_MODE_UART | _0000_SAU_TRANSFER_END;
```

Coming back to the PRS bits, we can see on line 67 in r_cg_serial.c which PRS bits we set in the SPS register.

```
SPS1 = _0002_SAU_CK00_FCLK_2 | _0020_SAU_CK01_FCLK_2;
```

The meaning of these constants are defined in r_cg_serial.h.

```
#define _0002_SAU_CK00_FCLK_2            (0x0002U)    /* ck00 - fCLK/2^2 */
```

We see that we chose $F_{MCK} = F_{CLK}/2^2$.
$F_{MCK} = 12MHz/4 = 3MHz$

We now solved the first term of the baud rate equation.

Next we need to look at the second term, the divider programmed into the SDR register. SDR10 (TX) and SDR11 (RX) in our case for UART2.

Looking at line 96 and 103 in r_cg_serial.c, we can see the lines where these registers are programmed. Referencing the constants back to r_cg_serial.h we see we used:

```
SDR10 = _9A00_UART2_TRANSMIT_DIVISOR;
SDR11 = _9A00_UART2_RECEIVE_DIVISOR;
```

where

```
#define _9A00_UART2_RECEIVE_DIVISOR      (0x9A00U)
#define _9A00_UART2_TRANSMIT_DIVISOR     (0x9A00U)
```

0x9A00 (unsigned) in binary is 0b1001 1010 0000 0000

The formula specifies the divider as SDRmn[15:9]. Thus we only use the highest 7 bits of the SDR register as the divider.

The divider then becomes 0b1001101 = 77

Substituting these values back into our baud rate formula we get:

(Baud rate) = Operation clock (F_{MCK}) frequency of target channel ÷ (SDRmn[15:9] + 1) ÷ 2 [bps]
Baud rate = $3 \times 10^6 \div (77 + 1) \div 2$
= 19230.76923

We wanted to obtain a baud rate of 19200, but due to the SDR register providing only 7 bits of resolution we can work with, the closest we can get is 19230.77. There is thus an error between the computer's 19200 baud and the microcontroller's 19230.77 baud rate. It translates to an error of 0.16%.

When we measure the microcontroller's transmit UART line with the oscilloscope, we measure a single bit period of $52\mu s$. This is correct as this is the period for 19230.77 baud.

2 Notes on the assignment

1. The two constant names might be different, but will look similar. What we are actually refering to here is the values programmed into the SDR10 and SDR11 registers. The calculation that needs to be done here was already done earlier in this document.

2. The calculations that need to be done here are similar to that in question 1. The formula just needs to be implemented in the other direction to obtain the values for the SDR10 and SDR11 registers. Compare the wanted baud rate to the actual obtained baud rate to calculate the error. It might be that the divider is too large for the SDR register's 7 bits. In that case you will need to change the first term of the baud rate formula by selecting a different F_{CLK} to F_{MCK} divider (SPS register). Try to obtain a baud rate error as small as possible.

3. This question is unclear, but what we want you to do is to look at the formulas and figure on page 617 of the Hardware Manual. Say we set the microcontroller to use 19200 baud (actually 19230.77). What will the maximum and minimum baud rate be that it can still receive?

   We can do the same for transmitting too. Say the computer is listening on 19200 baud. If we make an error in the baud rate on the microcontroller, what is the minimum and maximum values we can program into the SDR register so that communication with the computer still functions?

4. This questions has two parts, and is similar to what we did in practical 1. Firstly the pin on which the LED is connected needs to be set as an output. This only needs to be done once on startup. The correct location in the code for this is the `R_MAIN_UserInit(void)` function in the `r_main.c` file. The function is already called once at startup, so all you need to do is to add one line of code inside it to set the correct pin for the LED as an output.

The second part is to toggle the LED whenever the character "l" is received. In the main function you'll see `if(uart2RxFlag)`. This check will be true whenever a new character has been received on the UART. Two lines lower down you'll see the received character (`uart2RxBuf[0]`) is read, 1 is added and then it is copied to the transmit buffer. On the next line the transmit buffer is queued for transmission.

Around here we can inspect the value of `uart2RxBuf[0]`. If it is an "l", we can either immediately toggle the LED, or call a function that handles the toggling of the LED. Functionising actions that will be called at multiple places in your program is a better style. The place to add your function is at the bottom of the `r_main.c` file. Also remember to add the function prototype at the top of the file.

5. The existing code copies one character into the transmit buffer, and then queues it for transmission, specifying that the length is 1. We can do something similar, checking if "h" has been received, adding the letters `H e l l o` into the transmit buffer, and then queue it for transmission, specifying a length of 5.