ECE3810 Microprocessor System Design Laboratory

Laboratory Report #2

Name: 胡瑞

Student ID: 122050014

Date: 2024.9.29

The Chinese University of Hong Kong, Shenzhen

During the lab, four experiments are involved for different purposes. Each experiment will be shortly introduced in the following section. Details of each experiment will be elaborated after the short introduction.

- Experiment 1: Study the initiation of clock tree and USART2 port
- Experiment 2: Adding initiation of USART1 port and its clock, setup and transmit data by USART
- Experiment 3: Develop a more general method to transmit data and display by USART
- Experiment 4: Develop its own library including clock tree, USART functions

1. Experiment 1

1.1 Design

The design of experiment 1 is provided by the handout. Specific coding is shown below.

```
void EIE3810_clock_tree_init(void)
{
  u8 PLL=7;
  u8 temp=0;
  RCC->CR |= 0x00010000; //Sets the HSEON bit in the RCC_CR register, enabling the High-Speed External (HSE) clock.
  while(!((RCC->CR>>17)&0x1));//Waits until the HSERDY bit in RCC_CR is set, indicating that the HSE clock is stable.
  RCC->CFGR &= 0xFFFDFFFF; //Clears the PLLXTPRE bit in RCC_CFGR, setting the HSE clock divider for PLL entry to "not divided".
  RCC->CFGR |= 1<<16; //Sets the PLLSRC bit in RCC_CFGR, selecting HSE as the PLL input clock source.
  RCC->CFGR |= PLL<<18; //Sets the PLLMUL bits in RCC_CFGR, configuring the PLL multiplication factor to 9x.
  RCC->CR |=0x01000000;//Sets the PLLON bit in RCC_CR, enabling the PLL.
  while(!(RCC->CR>>25));//Waits until the PLLRDY bit in RCC_CR is set, indicating that the PLL is locked and ready.
  RCC->CFGR &=0xFFFFFFFE;//Clears the SW bits in RCC_CFGR.
  RCC->CFGR |=0x00000002;//Sets the SW bits in RCC_CFGR to select PLL as the system clock source.
  while(temp !=0x02) //Checks the SWS bits in RCC_CFGR to confirm that PLL is indeed being used as the system clock.
  {
    temp=RCC->CFGR>>2;
    temp &= 0x03; //If check condition fails, not allow the clock and continue to check.
  }
  RCC->CFGR &= 0xFFFFFC0F;//Clears the PPRE1 bits in RCC_CFGR.
  RCC->CFGR |= 0x00000400;//Sets the PPRE1 bits in RCC_CFGR to divide HCLK by 2 for APB1 peripheral clocks.
  FLASH->ACR = 0x32;//Set FLASH with 2 wait states
  RCC->APB1ENR |= 1<<17; //Sets the USART2EN bit in RCC_APB1ENR, enabling the clock for USART2.
}
```

Figure 1 Clock tree init function

```
void EIE3810_USART2_init(u32 pclk1, u32 baudrate)
{
  //USART2
  float temp;
  u16 mantissa;
  u16 fraction;
  temp=(float) (pclk1*1000000)/(baudrate*16);
  mantissa=temp;
  fraction=(temp-mantissa)*16;
  mantissa<<=4;
  mantissa+=fraction;
  RCC->APB2ENR |= 1<<2; //Enables the clock for GPIOA (USART2 uses GPIOA pins).
  GPIOA->CRL &= 0xFFFF00FF; //Clears the configuration bits for PA2 and PA3 in GPIOA_CRL.
  GPIOA->CRL |= 0x00008B00; //Configures PA2 (USART2_TX) as alternate function output push-pull, and PA3 (USART2_RX) as input floating.
  RCC->APB1RSTR |= 1<<17; //Setting the USART2 reset bit.
  RCC->APB1RSTR &= ~(1<<17); //Resetting the USART2 reset bit.
  USART2->BRR=mantissa;//Sets the baud rate register for USART2.
  USART2->CR1=0x2008; //Configures the USART2, enable the USART port and also its transmitter.
}
```

Figure 2 USART2 init function

Step 9 from handout: The procedure of clock tree init function is enabling the HSE clock; wait until the HSERDY bit is set, indicating the HSE clock is instead ready and stable; setting the HSE clock divider to be “not divided”; selecting HSE to be the PLL input clock source; configuring the PLL multiplication factor of 9x; enabling the PLL; waits until the PLL is ready and stable; clearing the SW bit and then setting this bit to select PLL to be the system clock source; checking PLL is indeed the clock source; clearing the PPRE1 bit and then set this bit to divide HCLK

by 2 for APB1; setting FLASH with 2 wait states; finally enabling the clock for USART2.

Step 10 from handout: For USART2 init function, first creating two u16 variable to store mantissa and fraction; then perform the calculation to get mantissa and fraction using desired pclk and baud rate. Enabling the clock for GPIOA, clearing the bits for PA2 an PA3; setting the output to be alternate function output push pull and input to be pull-up or pull-down; Set and reset the USART2, and then set the baud rate for USART2; finally enabling USART2 and its transmitter.

1.2 Result

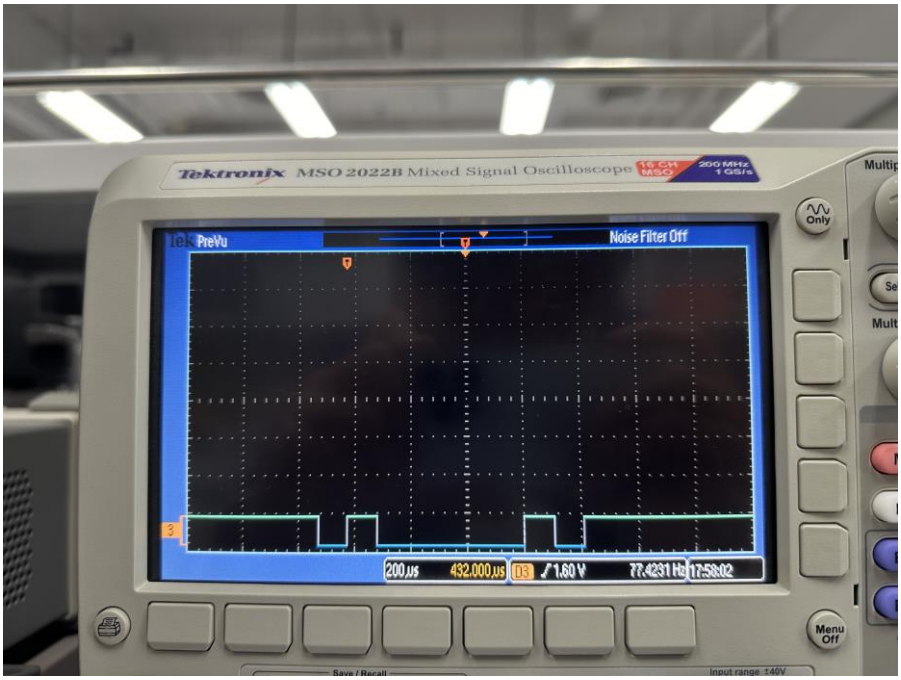

Figure 3 Logic analyzer waveform

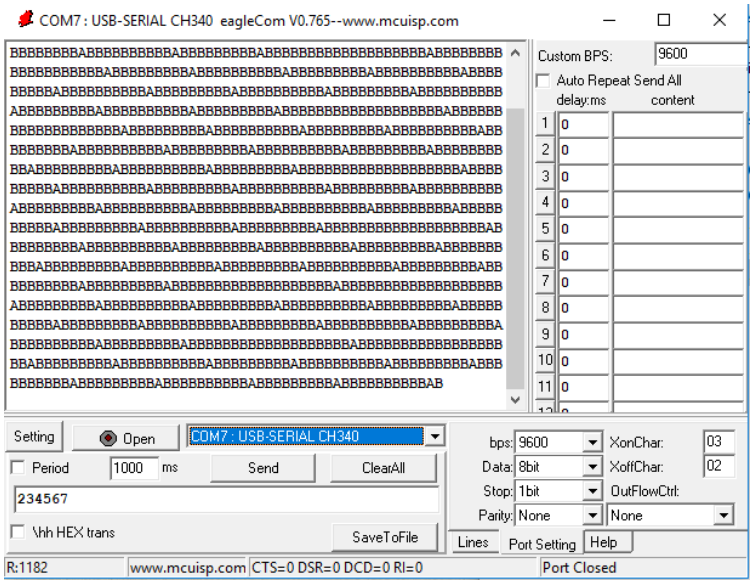

Figure 4 eagleCom result of exp 1

Step 7 and 8 from handout: Using figure 3 to compare with the serial transmission from the handout, the data transmitted is different, which is the middle section of the waveform. However, waveform from figure 3 also contains a Start bit as LOW and a Stop bit as HIGH. The Par bit for figure 3 is also HIGH. After modifying parameter of Delay() from 50000 to 100, letter transmitted and shown in eagleCom becomes faster.

2. Experiment 2

2.1 Design

```
//USART2 clock (APB1)
RCC->CFGR &= 0xFFFFFC0F;//Add comments
RCC->CFGR |= 0x00000400;//Add comments

//USART1 clock (APB2)
RCC->CFGR &= 0xFFFFC7FF; // Clear PPRE2 bits (for APB2)
  RCC->CFGR |= 0x00000000; // Set PPRE2 bits to not divide HCLK for APB2 (72MHz)

FLASH->ACR = 0x32;//Set FLASH with 2 wait states
RCC->APB1ENR |= 1<<17; //Add comments

// Add this line to enable USART1 clock
RCC->APB2ENR |= 1<<14; // Enable USART1 clock
RCC->APB2ENR |= 1<<2; // Enable GPIOA clock
```

Figure 5 Clock tree setup

```
91  void EIE3810_USART1_init(u32 pclk2, u32 baudrate)
92  {
93      float temp;
94      u16 mantissa;
95      u16 fraction;
96
97      // Calculate BRR value
98      temp = (float) (pclk2*1000000)/(baudrate*16);
99      mantissa = temp;
100     fraction = (temp - mantissa) * 16;
101     mantissa <<= 4;
102     mantissa += fraction;
103
104     // Enable clock for GPIOA
105     RCC->APB2ENR |= 1 << 2;
106
107     // Configure PA9 (TX) and PA10 (RX)
108     GPIOA->CRH &= 0xFFFFF00F; // Clear bits for PA9 and PA10
109     GPIOA->CRH |= 0x000008B0; // Set PA9 as alternate function output push-pull, PA10 as input floating
110
111     // Enable clock for USART1
112     RCC->APB2ENR |= 1 << 14;
113
114     // Reset USART1
115     RCC->APB2RSTR |= 1 << 14;
116     RCC->APB2RSTR &= ~(1 << 14);
117
118     // Set baud rate
119     USART1->BRR = mantissa;
120
121     // Enable USART, enable transmitter, enable receiver
122     USART1->CR1 = 0x2008;
123 }
```

Figure 6 USART1 initialization

As shown in the figure 5, the clock setup of USART1 is similar to the setup of USART2 except the bit representing USART1 is different from USART2. We need to use F and 8 to clear PPRE2 bit for APB2 firstly, and then set PPRE2 bit to 0 so that HCLK will not be divided because what we want is the full frequency 72MHz for this port. After the setup, we also set the APB2ENR bit to enable the USART1 clock and GPIOA clock for future use.

As shown in the figure 6, the USART1 setup is also very similar to the USART2 setup in experiment 1. Several differences include that pclk1 for USART2 is 36MHz, pclk2 for USART1 is 72MHz; in line 108 and 109 of figure 6, PA9 and PA10 in CRH of GPIOA are configured instead of PA2 and PA3 in CRL. This is because PA9 and PA10 are responsible for RX and TX functions, located at CRH register. In addition, line 115 and 116 of figure 6 reset the USART1. Because USART1 uses APB2, which has different bits arrangement from APB1, the bit is 14 instead of 17.

```
1  #include "stm32f10x.h"
2  void EIE3810_clock_tree_init(void);
3  void EIE3810_USART2_init(u32, u32);
4  void EIE3810_USART1_init(u32, u32);
5  void Delay(u32);
6
7
8  int main(void)
9  {
10   EIE3810_clock_tree_init();
11   EIE3810_USART1_init(72, 9600);
12
13   //Wait for approximately 1 sec after RESET
14   Delay(10000000);
15
16   while(1){
17     // ASCII codes for "122050014"
18         uint8_t digits[] = {49, 50, 50, 48, 53, 48, 48, 49, 52};
19
20         // Send each ASCII code with a delay
21         for(int i = 0; i < 9; i++)
22       {
23         USART1->DR = digits[i];
24         Delay(10000000);
25       }
26       Delay(900000000);
27   }
28 }
29
30 void Delay(u32 count)
31 {
32   u32 i;
33   for (i=0;i<count;i++);
34 }
35
```

Figure 7 main function

The main function will transmit ASCII version of my student ID number to the USART1 port, displaying these number in the eagleCom. It also uses two Delay function to control the interval between transmission of adjacent two number.

2.2 Result

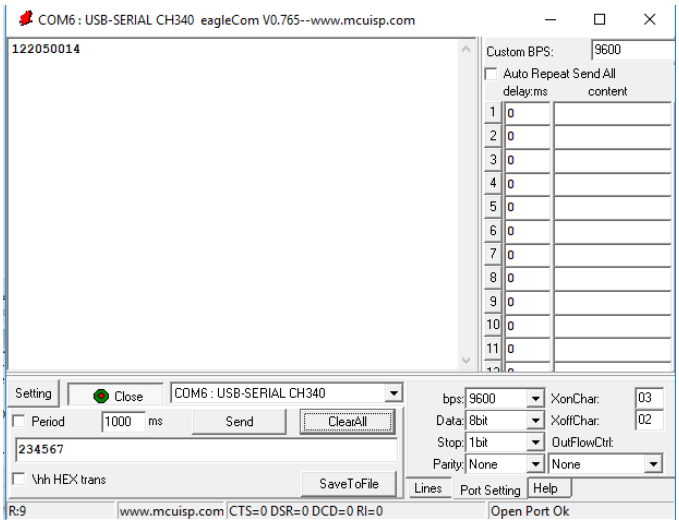

Figure 8 eagleCom result of exp 2

3. Experiment 3

3.1 Design

```
31  void USART_print(u8 USARTport, char *st)
32  {
33    u8 i=0;
34    while(st[i] != 0x00)
35    {
36      if (USARTport == 1) USART1 -> DR = st[i];
37      if (USARTport == 2) USART2-> DR = st[i];
38      while(!((USART1->SR>>7)&0x1)); //Modifed line
39      if (i == 255) break;
40      i++;
41    }
42  }
43
```

Figure 9 USART_print function with modified check

The modified line checks the TXE, which is the 7th bit of SR (Status Register). &0x1 representing the bitwise AND, which mask all bits except the least significant bit, which is now the original 7th bit of SR. The ! operator inverts this logic, so the loop continues while TXE is 0 (data register not empty) and exits when TXE becomes 1 (data register empty).

3.2 Result

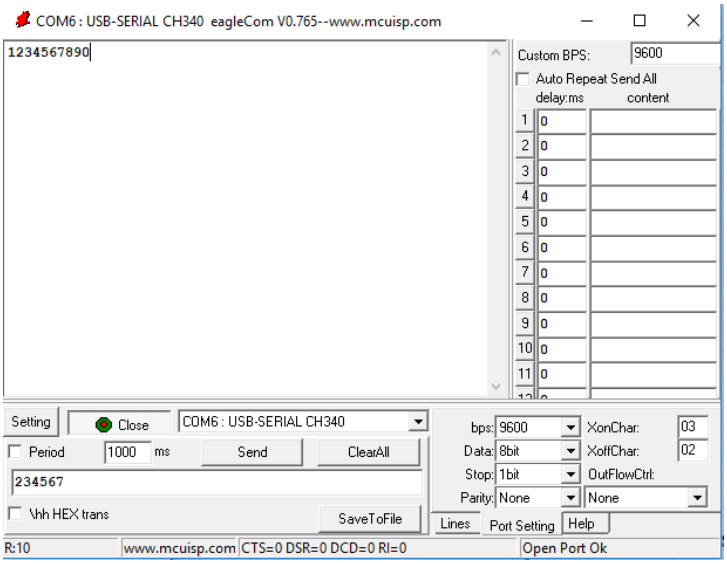

Figure 10 eagleCom result of exp 3

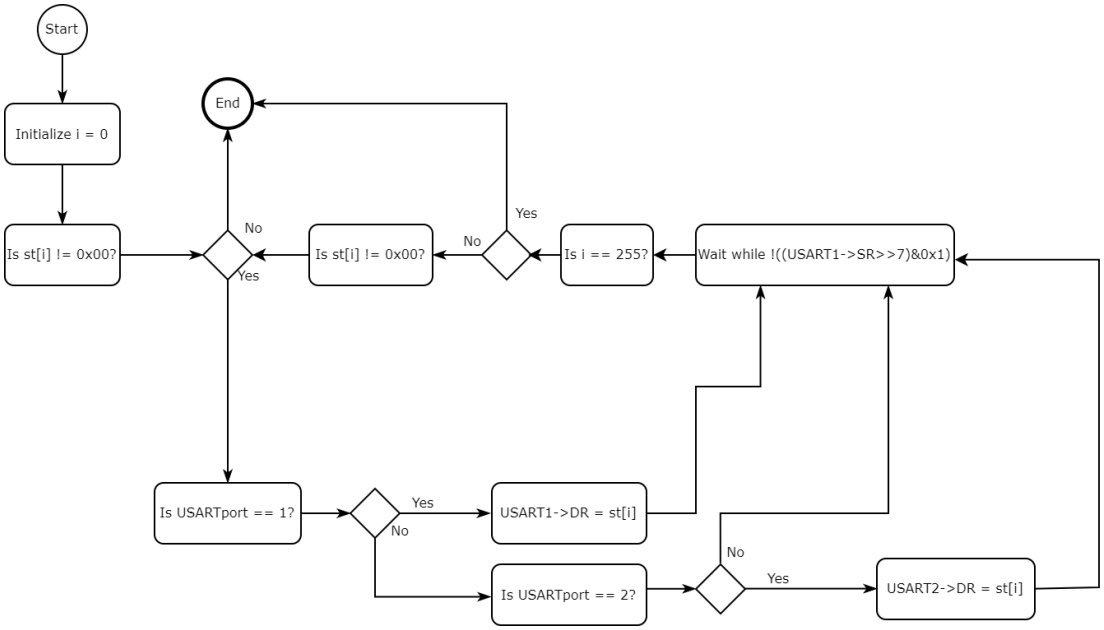

Figure 11 Flow chart of USART_print function

4. Experiment 4

4.1 Result

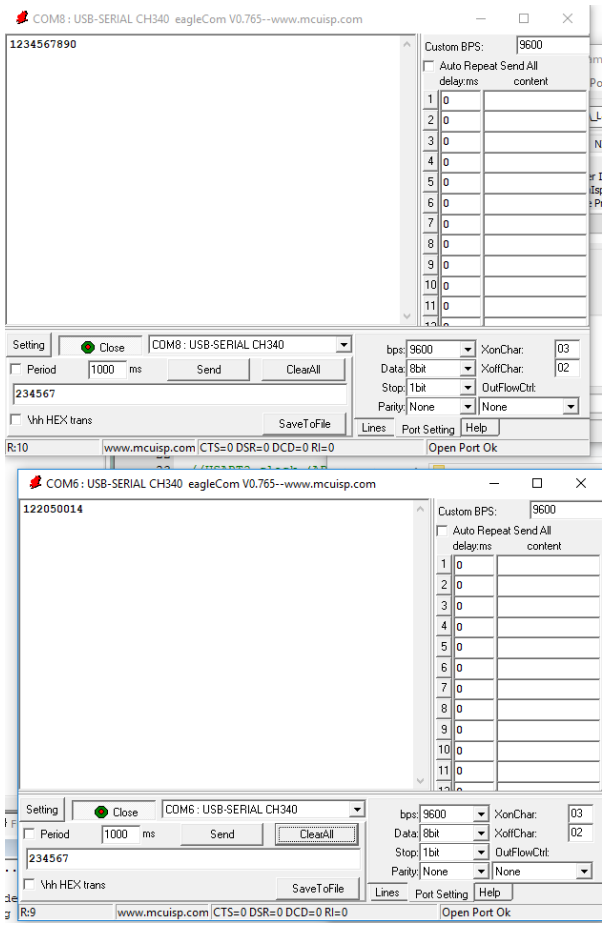

Figure 12 eagleCom result of exp 4

As shown in the figure 12, number “1234567890” is shown in COM8, which is the assigned COM number of USART1; number “122050014” is shown in COM6, which is the assigned COM number of USART2. These two number are shown simultaneously.

x. Conclusion

In this lab, we have conducted four experiments with different purposes. We have learnt the initiation of clock tree and USART2 port; adding initiation of USART1 port and its clock, setup and transmit data by USART; developing a more general method to transmit data and display by USART; developing its own library including clock tree, USART functions