Mircoprocessor Lab9 Report

Group1 0416324胡安鳳 0410137 劉家麟

2.Problem2 The ADC and UART communication

(1) What is UART?

A universal asynchronous receiver-transmitter (UART /ˈjuːɑːrt/) is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. The electric signaling levels and methods are handled by a driver circuit external to the UART. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers.

(2) How to set up the UART ?

**void** USART1\_Init(**void**)

{

//f CK can be f LSE , f HSI , f PCLK , f SYS .,we can just use the clock from STM32, which is 4MHz

RCC->APB2ENR |= RCC\_APB2ENR\_USART1EN;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UART CR1 setting\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CR1 may reference to p1346 of the manual

CR1 clear the bits of M(Data length/8bits is default) PS() PCE(Parity check) TE RE, then set the bits of TE RE where TE enables the function of transmittion and

RE enables the function of reception

In short, M bits=0-->Data frame to be 8 bits (this is suitable for ASCII Character transmittion)

PS bit=0-->Evan parity (this is the deault status for the UART transmittion in the picocom terminal ,or maybe in the most serial terminal?)

PCE bit=0-->Parity checking disable (this is the deault status for the UART transmittion in the picocom terminal ,or maybe in the most serial terminal?)

TE bit=1-->Transmitting enable

RE but=1--> Receiving enable

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MODIFY\_REG(USART1->CR1, USART\_CR1\_M | USART\_CR1\_PS | USART\_CR1\_PCE | USART\_CR1\_TE | USART\_CR1\_RE |

USART\_CR1\_OVER8, USART\_CR1\_TE | USART\_CR1\_RE);

// CR2 for how much bit indicating the stop, now 1 bit

MODIFY\_REG(USART1->CR2, USART\_CR2\_STOP, 0x0); //0x0 for 1-bit stop

// CR3 clear the bits of RTSE , CTSE and ONEBIT, these are used for RS232, different from our LAB9

MODIFY\_REG(USART1->CR3, (USART\_CR3\_RTSE | USART\_CR3\_CTSE | USART\_CR3\_ONEBIT), 0x0);

/\*uint16\_t brr15\_4 = USART1->BRR & 0b1111111111110000;

brr15\_4 >>= 4;

uint16\_t brr2\_0 = USART1->BRR & 0b111;

brr2\_0 <<= 1;

uint16\_t baud\_x = brr15\_4 | brr2\_0;\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Baud rate setting,\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

oversampling by16 (since over8 is cleared)

USARTDIV is how fast the communication port in computer wants to transmit and receive (they should be the same value)

default terminal setting is 9600, then we set the baud rate = fCK/USARTDIV , for default fCK = fSYS --> 4M / 9.6K

or check manual p 1319 for 72MHz--> BRR with OVER16 USARTDIV = 9600-->1D4C so for 4MHz is 1D4C/18 about 416(DEC) which = 4MHz/9.6K

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MODIFY\_REG(USART1->BRR, 0xFFFF /\*clear all and reset\*/, 4000000/9600);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*asynchronous mode setting\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In asynchronous mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART\_CR2 register,

- SCEN, HDSEL and IREN bits in the USART\_CR3 register.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

USART1->CR2 &= ~(USART\_CR2\_LINEN | USART\_CR2\_CLKEN);

USART1->CR3 &= ~(USART\_CR3\_SCEN | USART\_CR3\_HDSEL | USART\_CR3\_IREN);

// Enable UART

USART1->CR1 |= (USART\_CR1\_UE);

}

(3) How to send data through the UART ?

Note: the data is sent byte by byte to the receiver, which is suitable for ASCII-encoded data transmission

**void** USART1\_Transmit(uint8\_t \*arr, uint32\_t size)

{

/\*

Character transmission procedure

1. Program the M bits in USART\_CR1 to define the word length.

2. Select the desired baud rate using the USART\_BRR register.

3. Program the number of stop bits in USART\_CR2.

4. Enable the USART by writing the UE bit in USART\_CR1 register to 1.

5. Select DMA enable (DMAT) in USART\_CR3 if multibuffer communication is to take

place. Configure the DMA register as explained in multibuffer communication.

6. Set the TE bit in USART\_CR1 to send an idle frame as first transmission.

7. Write the data to send in the USART\_TDR register (this clears the TXE bit). Repeat this

for each data to be transmitted in case of single buffer.

8. After writing the last data into the USART\_TDR register, wait until TC=1. This indicates

that the transmission of the last frame is complete. This is required for instance when

the USART is disabled or enters the Halt mode to avoid corrupting the last

transmission. \*/

**for**(**int** i=0;i<size;i++)

{

**while** (!READ\_BIT(USART1->ISR, USART\_ISR\_TXE)); //polling the USART is ready

USART1->TDR = arr[i];//transmitt data register, get the data and send to USART port

}

**while** (!READ\_BIT(USART1->ISR, USART\_ISR\_TC));

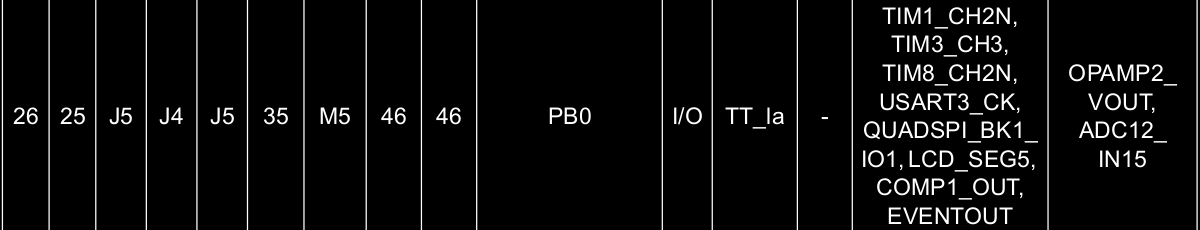
}

(4) What is ADC?

In electronics, an analog-to-digital converter (ADC, A/D, or A-to-D) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. An ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number representing the magnitude of the voltage or current. Typically the digital output is a two's complement binary number that is proportional to the input, but there are other possibilities

(5) How to set up and configure the ADC?

//Watch out the channel



**void** configureADC()

{

// TODO

RCC->AHB2ENR |= RCC\_AHB2ENR\_ADCEN; //Turn on the adc function

GPIOB->ASCR |= 1; //turn on the analog controller in PB0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADC clock config starts here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* ############################################################################

     \* Set the ADC clock first by using ADC common register Reset value: 0x0000 0000

     \* The ADC common register can be found at manual p608

     \* We set the clock source as the sysclk , which is default 4MHz

     \* From LSB to MSB

     \* Bits

     \* 4:0 Dual adc or not, we dont set dual adc, so keep reset value 00000 for independent mode

     \* 11:8 Delay b/w 2 sampling phases, setting for 5 clock cycle will be fine which is 0100

     \* 13 No DMA, reset value is fine

     \* 15:14 No DMA, reset value is fine

     \* 17:16 IMPORTANT!!!!!!!!!! Must config the Clock correctly!!!!!!!!!!! use HCLK hardware system clock/1 will be fine, set to 01

     \* 21:18 No division, reset value is fine

     \* 22 Not used, reset value is fine

     \* 23 Not used, reset value is fine

     \* 24 Not used, reset value is fine

     \* Else is the reserved value, should be kept in the reset state.

     \* ###########################################################################\*/

//10987654321098765432109876543210

ADC123\_COMMON->CCR |= ***0b00000000000000010000010000000000***;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADC clock and some other config ends here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADC main settings starts here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ADC1->CR &= ~ADC\_CR\_DEEPPWD; //Turn off the deep-power mode before the configuration

ADC1->CR |= ADC\_CR\_ADVREGEN; //Turn on the voltage regulator

**while**(!READ\_BIT(ADC1->CR, ADC\_CR\_ADVREGEN)); //Polling until the ADCVERGEN is pulled up ot 1

ADC1->CR |= 0x80000000; //Tell the ADC to do the calibration.

**while**((ADC1->CR & 0x80000000) >> 31); //Polling until the calibration is done

ADC1->CFGR &= ~ADC\_CFGR\_RES; // 12-bit resolution

ADC1->CFGR &= ~ADC\_CFGR\_CONT; // Disable continuous conversion

ADC1->CFGR &= ~ADC\_CFGR\_ALIGN; // Right align

//10987654321098765432109876543210

ADC1->SQR1 |= ***0b00000000000000000000001111000000***; //We use only one ADC channel but problem is, which channel--> rank1 for channel15(pb0)

//the higher sampling cycle can ensure the more detalied data but takes more process time, set12.5 will be fine, all takes 25 cycles

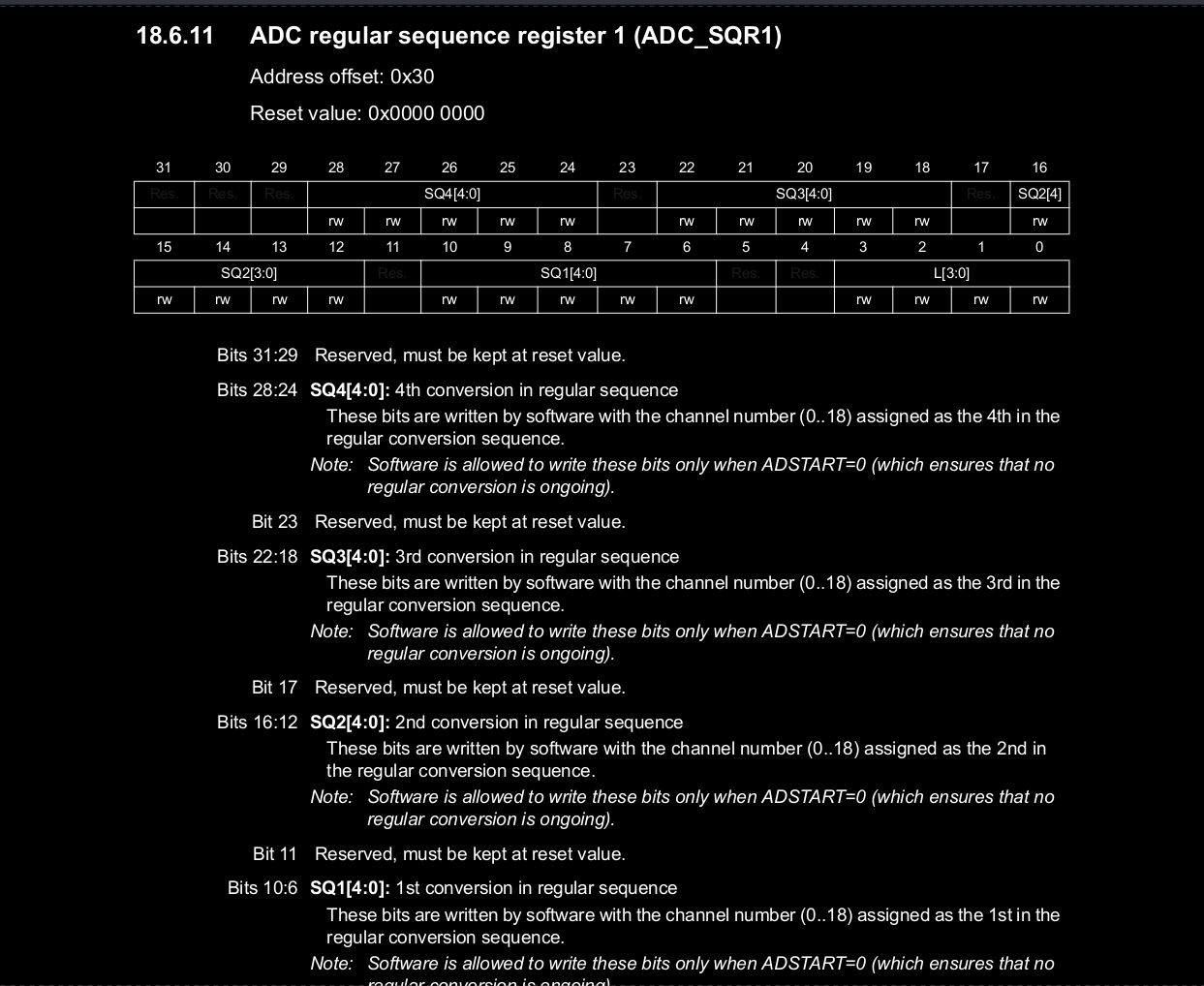
ADC1->SMPR1 |= ***0b00000000000000000000000000000010***;

//ADC conversion time may reference to manual p518

ADC1->IER |= ADC\_IER\_EOCIE; //When conversion ends, do the interrupt (the end of conversion interrupt)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADC main settings ends here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

(6) How to run the ADC to fetch the data?

Start the ADC only once, no until the button pressed(button event interruption has occurred)do we have to do the ADC conversion.

Suppose we do ADC conversion all the time, the result may be inaccurate.

startADC();

**while**(1)

{

**if**(check\_the\_fucking\_button())

{

get\_light\_resistor();

sprintf(buf,"Resistor value %f \r\n",resistor\_value);

USART1\_Transmit(buf,(uint32\_t)**sizeof**(buf));

}

//ADC Clock is 4MHz,

**void** startADC()

{

// TODO

ADC1->CR |= ADC\_CR\_ADEN; //Turn on the ADC, write to enable the adc

**while**(!READ\_BIT(ADC1->ISR, ADC\_ISR\_ADRDY)); //Wait until ADC is ready for conversion

}

**void** get\_light\_resistor()

{

ADC1->CR |= ADC\_CR\_ADSTART;

**while**(!READ\_BIT(ADC1->ISR, ADC\_ISR\_EOC)); //Polling until the ADC conversion of light resistor is done

resistor\_value = ADC1->DR;

}