Lab.10 UART and ADC

0516076 施威綸

1. Lab objectives 實驗目的

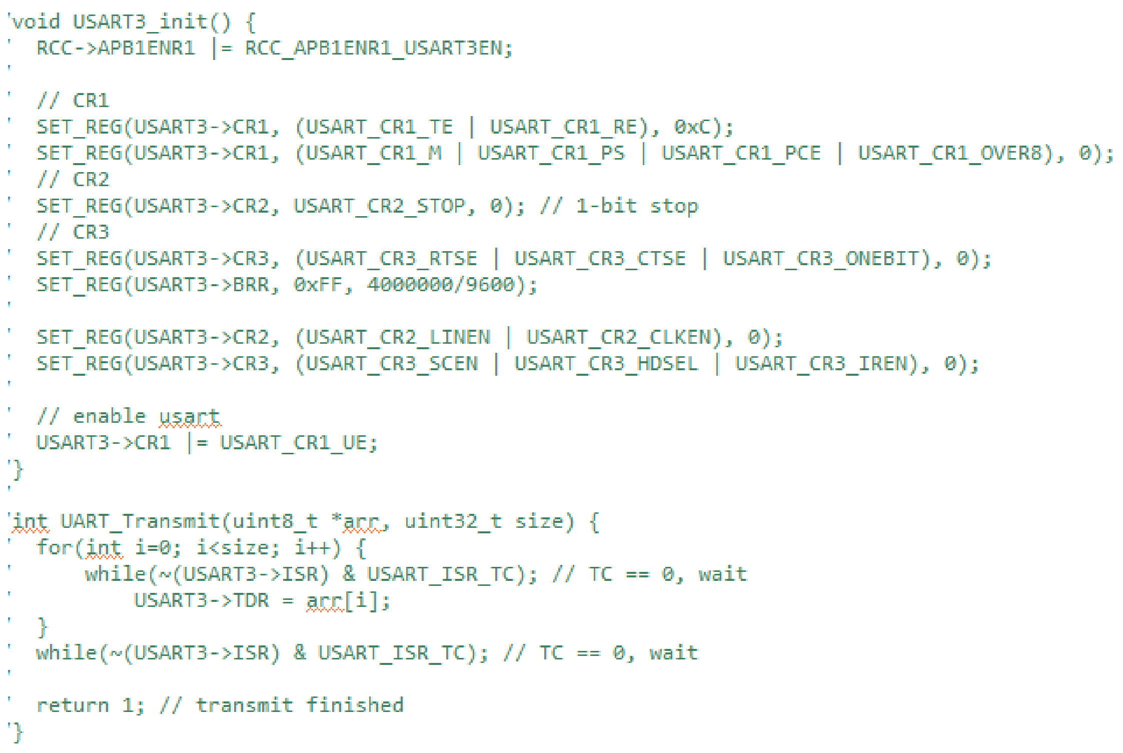
- Understand the use of UART.
- Understand the use of ADC.

2. Steps 實驗步驟

2.1. Hello World!

When the blue button on the board is pressed (PC13), use the UART to transfer the “Hello World!” string to the computer. And it can be displayed on the serial monitor of the computer side.

(a) Initial GPIO for Tx, Rx.
(b) Set up UART registers
(c) Pass each characters in the string to the transmit data register(TDR) for serial transmission.



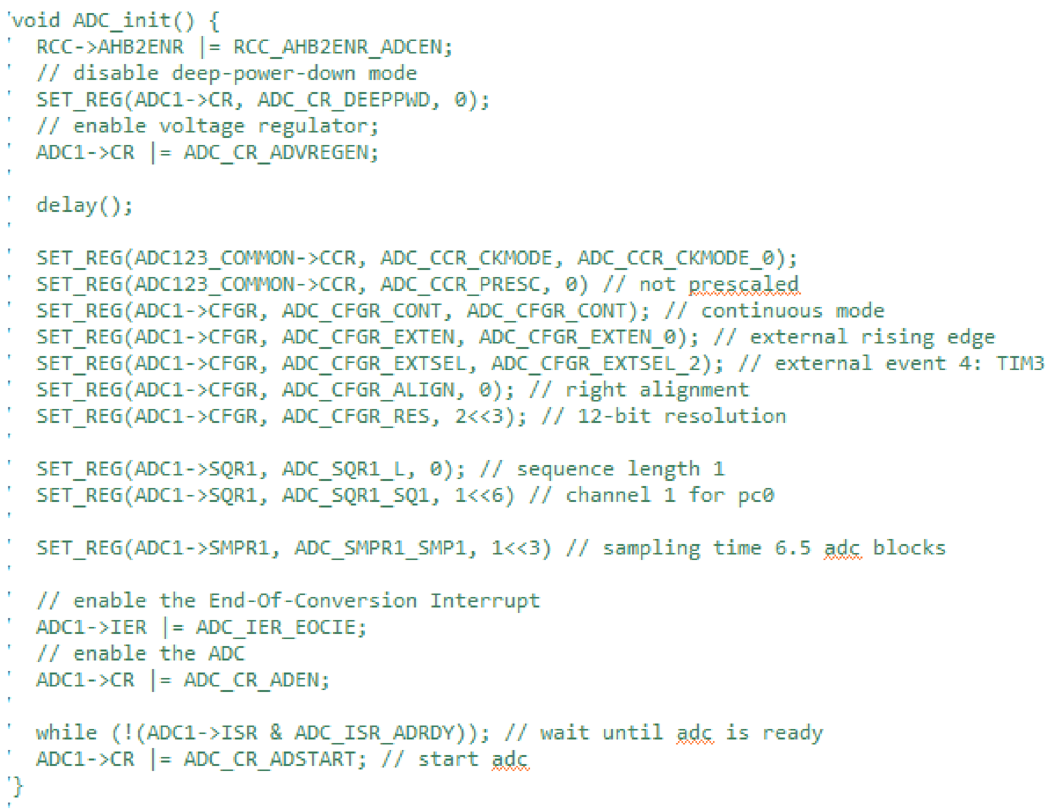
```
void USART3_init() {
  RCC->APB1ENR1 |= RCC_APB1ENR1_USART3EN;

  // CR1
  SET_REG(USART3->CR1, (USART_CR1_TE | USART_CR1_RE), 0xC);
  SET_REG(USART3->CR1, (USART_CR1_M | USART_CR1_PS | USART_CR1_PCE | USART_CR1_OVER8), 0);
  // CR2
  SET_REG(USART3->CR2, USART_CR2_STOP, 0); // 1-bit stop
  // CR3
  SET_REG(USART3->CR3, (USART_CR3_RTSE | USART_CR3_CTSE | USART_CR3_ONEBIT), 0);
  SET_REG(USART3->BRR, 0xFF, 4000000/9600);

  SET_REG(USART3->CR2, (USART_CR2_LINEN | USART_CR2_CLKEN), 0);
  SET_REG(USART3->CR3, (USART_CR3_SCEN | USART_CR3_HDSEL | USART_CR3_IREN), 0);

  // enable usart
  USART3->CR1 |= USART_CR1_UE;
}

int UART_Transmit(uint8_t *arr, uint32_t size) {
  for(int i=0; i<size; i++) {
      while(~(USART3->ISR) & USART_ISR_TC); // TC == 0, wait
          USART3->TDR = arr[i];
  }
  while(~(USART3->ISR) & USART_ISR_TC); // TC == 0, wait

  return 1; // transmit finished
}
```

2.2. Read photo resistor value

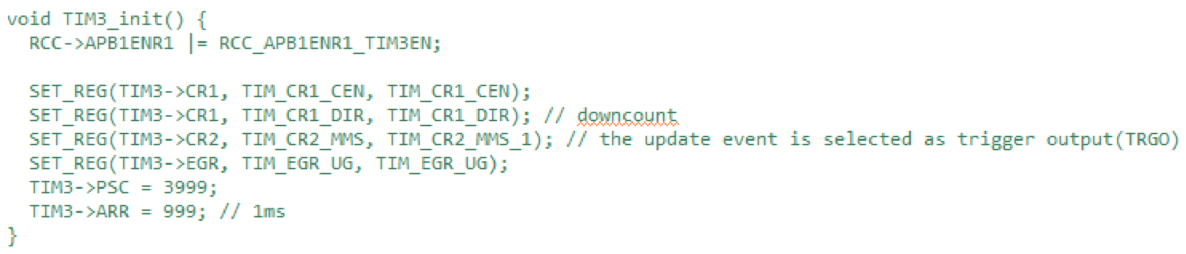
Use the ADC (Analog-to-Digital-Converter) provided on the board to read the value of the photo resistor in 12-bit resolution using

Interrupt, and use the UART output value each time the button is pressed.

(a) Set up ADC
(b) Set up timer for ADC sampling
(c) Set up interrupt for button (EXTI, NVIC)
(d) Set ADC interrupt handler
(e) Transmit value from the protocol in Lab10.1



```c
void ADC_init() {
  RCC->AHB2ENR |= RCC_AHB2ENR_ADCEN;
  // disable deep-power-down mode
  SET_REG(ADC1->CR, ADC_CR_DEEPPWD, 0);
  // enable voltage regulator;
  ADC1->CR |= ADC_CR_ADVREGEN;

  delay();

  SET_REG(ADC123_COMMON->CCR, ADC_CCR_CKMODE, ADC_CCR_CKMODE_0);
  SET_REG(ADC123_COMMON->CCR, ADC_CCR_PRESC, 0) // not prescaled
  SET_REG(ADC1->CFGR, ADC_CFGR_CONT, ADC_CFGR_CONT); // continuous mode
  SET_REG(ADC1->CFGR, ADC_CFGR_EXTEN, ADC_CFGR_EXTEN_0); // external rising edge
  SET_REG(ADC1->CFGR, ADC_CFGR_EXTSEL, ADC_CFGR_EXTSEL_2); // external event 4: TIM3
  SET_REG(ADC1->CFGR, ADC_CFGR_ALIGN, 0); // right alignment
  SET_REG(ADC1->CFGR, ADC_CFGR_RES, 2<<3); // 12-bit resolution

  SET_REG(ADC1->SQR1, ADC_SQR1_L, 0); // sequence length 1
  SET_REG(ADC1->SQR1, ADC_SQR1_SQ1, 1<<6) // channel 1 for pc0

  SET_REG(ADC1->SMPR1, ADC_SMPR1_SMP1, 1<<3) // sampling time 6.5 adc blocks

  // enable the End-Of-Conversion Interrupt
  ADC1->IER |= ADC_IER_EOCIE;
  // enable the ADC
  ADC1->CR |= ADC_CR_ADEN;

  while (!(ADC1->ISR & ADC_ISR_ADRDY)); // wait until adc is ready
  ADC1->CR |= ADC_CR_ADSTART; // start adc
}
```



```c
void TIM3_init() {
  RCC->APB1ENR1 |= RCC_APB1ENR1_TIM3EN;

  SET_REG(TIM3->CR1, TIM_CR1_CEN, TIM_CR1_CEN);
  SET_REG(TIM3->CR1, TIM_CR1_DIR, TIM_CR1_DIR); // downcount
  SET_REG(TIM3->CR2, TIM_CR2_MMS, TIM_CR2_MMS_1); // the update event is selected as trigger output(TRGO)
  SET_REG(TIM3->EGR, TIM_EGR_UG, TIM_EGR_UG);
  TIM3->PSC = 3999;
  TIM3->ARR = 999; // 1ms
}
```

```
void EXTI_config() {
  // clock enable
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

  // manage the external interrupt to GPIOs
  SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI13_PC;

  // set up the interrupt mask register EXTI_IMR1
  EXTI->IMR1 |= EXTI_IMR1_IM13;

  // set the rising-triggered interrupt
  EXTI->RTSR1 |= EXTI_RTSR1_RT13;
}

void NVIC_config() {
  NVIC_SetPriority(EXTI15_10_IRQn, 0); // high priority
  NVIC_SetPriority(ADC1_2_IRQn, 1);
  NVIC_EnableIRQ(EXTI15_10_IRQn);
  NVIC_EnableIRQ(ADC1_2_IRQn);
}
void ADC1_2_IRQHandler() {
  value = ADC1->DR;
//    int debug = value;

  // End of conversion flag, set by hardware, cleared by software
  ADC1->ISR |= ADC_ISR_EOC;
}
```

2.3. Simple shell

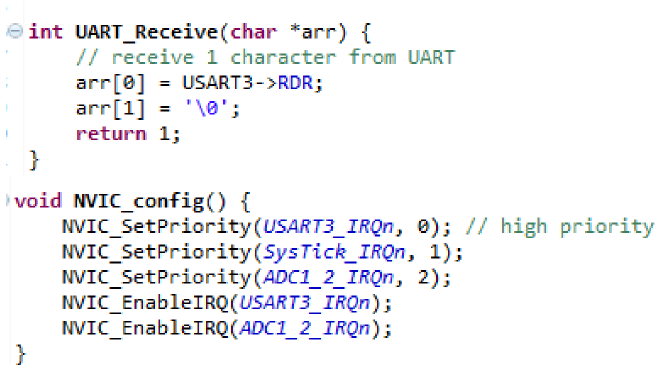
Implement Simple Shell on the board using UART and have three instructions

1) showid
    a) show your student ID
2) light
    a) Update and display the value of the photoresistor every 0.5 seconds (lab10.2), press 'q' to return to the original shell mode
3) led {on|off}
    a) "led on" will turn the PA5 LED on, "led off" will turn off

After the USB to UART is connected to the computer, any characters entered by the computer must be received through the board's UART RX and transmitted back to the computer using TX to achieve echo. The computer runs serial monitor (e.g. putty, pietty, MobaXterm ...) to use the shell.

(a) Enable RXNE interrupt
(b) Set UART receive
(c) Set up SysTick for 0.5 second
(d) Set interrupt priority among UART, SysTick, and ADC
(e) Implement the simple shell on UART IRQ handler

```
    // CR1
    SET_REG(USART3->CR1, (USART_CR1_TE | USART_CR1_RE), 0xC); // enable tx rx
    SET_REG(USART3->CR1, (USART_CR1_M | USART_CR1_PS | USART_CR1_PCE | USART_CR1_OVER8), 0);
    SET_REG(USART3->CR1, USART_CR1_RXNEIE, USART_CR1_RXNEIE); // enable RXNE interrupt
```



```
int UART_Receive(char *arr) {
    // receive 1 character from UART
    arr[0] = USART3->RDR;
    arr[1] = '\0';
    return 1;
}

void NVIC_config() {
    NVIC_SetPriority(USART3_IRQn, 0); // high priority
    NVIC_SetPriority(SysTick_IRQn, 1);
    NVIC_SetPriority(ADC1_2_IRQn, 2);
    NVIC_EnableIRQ(USART3_IRQn);
    NVIC_EnableIRQ(ADC1_2_IRQn);
}
```

3. Results and analysis 實驗結果與分析

ADC converts waveforms to digital data through different sampling rate and resolutions, and it’s a balance between the data size and the accuracy.

4. Conclusions and ideas 心得討論與應用聯想

Writing a simple shell is really useful for monitoring the machine status and input commands. This will be useful for projects more complicated in the future.