EIE3810 Microprocessor System Design Laboratory

Lab 5. Timer

School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen

2023-2024 Term 1

1. Objectives

In Lab 5, we will spend the 2-week session to study the following:

- To study how to use the General Purpose Timer of Cortex-M3.
- To study how to use the SYSTICK Timer of Cortex-M3.
- To study the generation of a Pulse Width Modulation (PWM) signal.
- To learn to use timer to acquire input from JOYPAD.

2. Basics

"Timer" is often equivalent to "counter" in microprocessor systems. Counter counts the number of pulses/edges from a signal source. If the signal source provides periodic pulses/edges (i.e., clock), the counter can provide timing information to the microprocessor.

To increase the precision of the timer, a high frequency clock signal will be employed as the timer's source. As the frequency is high, a counter with a large number of bits is needed to get a longer cycle time. A pre-scalar can be employed to reduce the number of bits required for the counter. When the counter overflows, a specific register's bit will be set to indicate overflowing, or an interrupt will be pended if it is enabled.

A modern microprocessor usually has a few counters to deal with different rates of signals. For some sub-programs that need a lower rate time base, sharing a counter to save resource is also possible. For a multi-task system, 10ms is a suitable time unit, and it functions like a heartbeat of the system.

Switching mode is one of the modern techniques for power control. It is commonly used for lighting (dimming), motor speed controlling (electric cars), and power supply (electronic appliances' DC power supplies). Using a microprocessor to generate and control Pulse Width Modulation (PWM) is easy and low-cost. Fig. 1 shows a typical PWM signal. It contains a signal with a fixed period. But there is an adjustable duty cycle, which is the percentage of pulse width in the period.

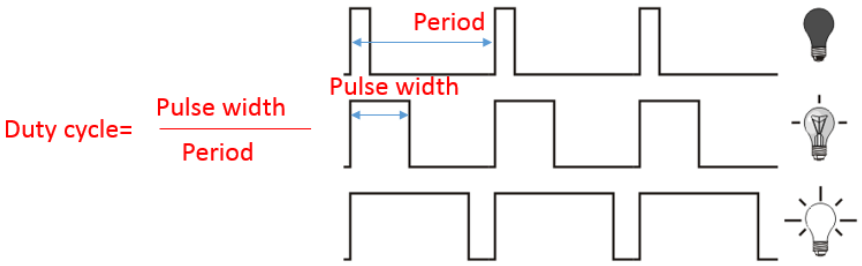


Fig. 1 PWM signal and the light behavior

In general, when the duty cycle is larger, the light will be brighter. (In our development board, as the LED is lit when a low voltage is applied, it is the opposite.)

Cortex-M3 has four kinds of timers: Basic Timers, SYSTICK Timer, General Purpose Timers, and Advanced Timers. In this lab, we will learn to use General Purpose Timers and SYSTICK.

- Basic Timers: Interrupt TIM6-7
- **SYSTICK Timer: Interrupt SysTick**
- **General Purpose Timers: Interrupt TIM2-5**
- Advanced Timers: Interrupt TIM1 and TIM8

When the preset time is reached, there will be a related interrupt. In the interrupt handler, you can include the process coding. But the handler should be as short as possible, preventing it from occupying the processor too long and affecting other events that need a real-time response. All time-consuming processes should be placed out of the handler.

For interrupt position, you can refer to Fig. 2.

Position	Priority	Type of priority	Acronym	Description	Address
...					
-	6	settable	SysTick	System tick timer	0x0000_003C
...					
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4
26	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation interrupts	0x0000_00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
...					
43	50	settable	TIM8_BRK	TIM8 Break interrupt	0x0000_00EC
44	51	settable	TIM8_UP	TIM8 Update interrupt	0x0000_00F0
45	52	settable	TIM8_TRG_COM	TIM8 Trigger and Commutation interrupts	0x0000_00F4
46	53	settable	TIM8_CC	TIM8 Capture Compare interrupt	0x0000_00F8
...					
50	57	settable	TIM5	TIM5 global interrupt	0x0000_0108
...					
54	61	settable	TIM6	TIM6 global interrupt	0x0000_0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000_011C

Fig. 2 Vector table for interrupt

In this lab, you will use TIM3. It is similar to set up the interrupt and handler subroutine.

2.1 TIM3 (General Purpose Timer)

In TIM3, there are two important registers: TIM3_ARR (auto-reload register), and TIM3_PSC (Prescaler). Refer to pages 417-418 in *RM0008*.

The counter clock frequency CK_CNT is equal to f_{CK} / (PSC[15:0] + 1). For f_{CK}, check Fig. 3.

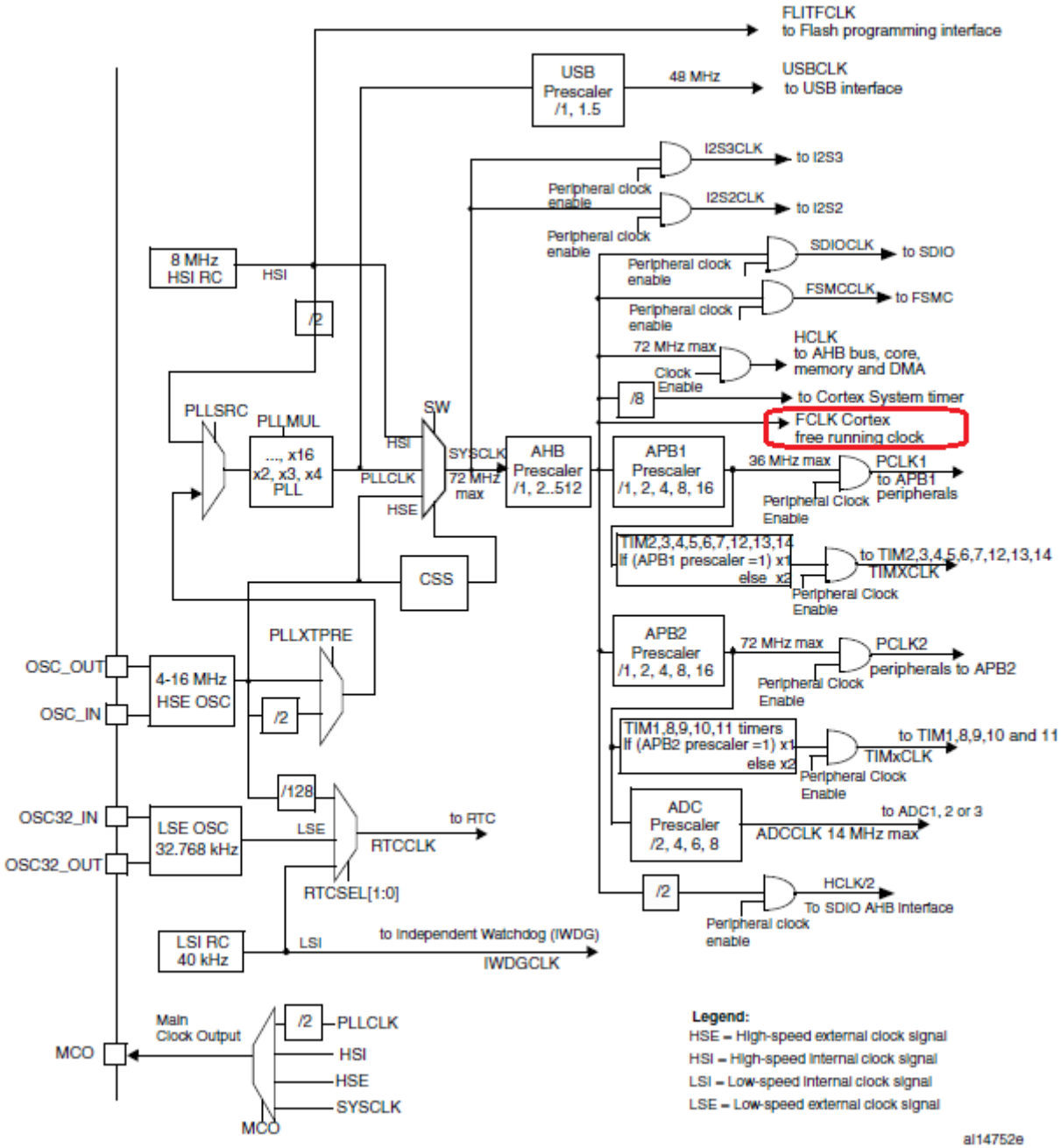


Fig. 3. Clock tree

In up-counting mode, the TIM3 counter will start from 0, and count at the frequency of CK_CNT, and when its value goes beyond the value in TIM3_ARR, there will be an overflow. Thereafter, TIM3 interrupt will be pending (if you enable this interrupt).

To setup this register, there are more steps. You can refer to Experiment 1.

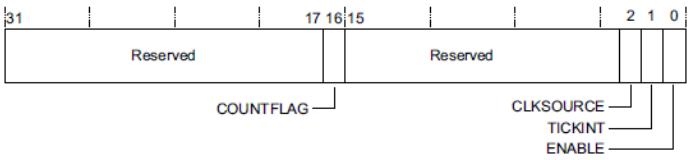
2.2 SYSTICK

SYSTICK is simpler than other timers. Just set

- Reload value
- Clock source
- Counter enable
- Interrupt enable
- Handler for each overflow

There are a number of registers to be used in this lab. For more register information, refer to page 8-8~8-12 in *Cortex-M3 Technical Reference Manual*. Here below we provide a number of them.

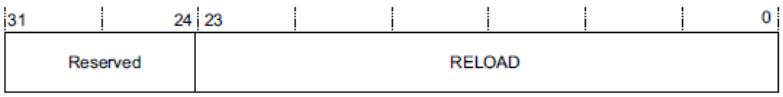
◦ SysTick Control and Status Register



Bits	Field	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read. Clears on read by application of any part of the SysTick Control and Status Register. If read by the debugger using the DAP, this bit is cleared on read-only if the MasterType bit in the AHB-AP Control Register is set to 0. Otherwise, the COUNTFLAG bit is not changed by the debugger read.
[2]	CLKSOURCE	0 = external reference clock. 1 = core clock. If no reference clock is provided, it is held at 1 and so gives the same time as the core clock. The core clock must be at least 2.5 times faster than the reference clock. If it is not, the count values are Unpredictable.
[1]	TICKINT	1 = counting down to 0 pends the SysTick handler. 0 = counting down to 0 does not pend the SysTick handler. Software can use the COUNTFLAG to determine if ever counted to 0.
[0]	ENABLE	1 = counter operates in a multi-shot way. That is, counter loads with the Reload value and then begins counting down. On reaching 0, it sets the COUNTFLAG to 1 and optionally pends the SysTick handler, based on TICKINT. It then loads the Reload value again, and begins counting. 0 = counter disabled.

Fig. 4 SysTick Control and Status Register

◦ SysTick Reload Value Register



Bits	Field	Function
[31:24]	-	Reserved
[23:0]	RELOAD	Value to load into the SysTick Current Value Register when the counter reaches 0.

Fig. 5 SysTick Reload Value Register

2.3 PWM

One General Purpose Timer can also generate four channels of Pulse Width Modulation (PWM) signals.

Take TIM3 as an example. Aside of TIM3_ARR (auto-reload register) and TIM3_PSC (Prescaler), we have to know other registers, TIM3_CCRx (x=1, 2, 3, 4).

Pulse width modulation mode allows generating a signal with a period determined by the value of the TIM3_ARR register and TIM3_PSC, similarly to Part 2.1 aforementioned. When TIM3 counter reaches the value in TIM3_ARR (in up-counting mode), the counter will be returned to 0. This indicates the length of the period of the PWM signal.

In TIM3_CCRx (x=1, 2, 3, 4), x indicates one of the 4 channels of PWM signals, which TIM3 can generate. The duty cycle is determined by the value of the TIM3_CCRx (x=1, 2, 3, 4). Channel x should be configured as an output. TIM3_CCRx is the value to be loaded in the actual capture/compare x register (preload value). In the up-counting situation, the PWM voltage output will be kept (high/low) when the counter register starts from 0, and change the logic level when the counter reaches the value in CCRx. When the counter exceeds the value in TIM3_ARR, it restarts the counter and the PWM voltage is renewed for another period.

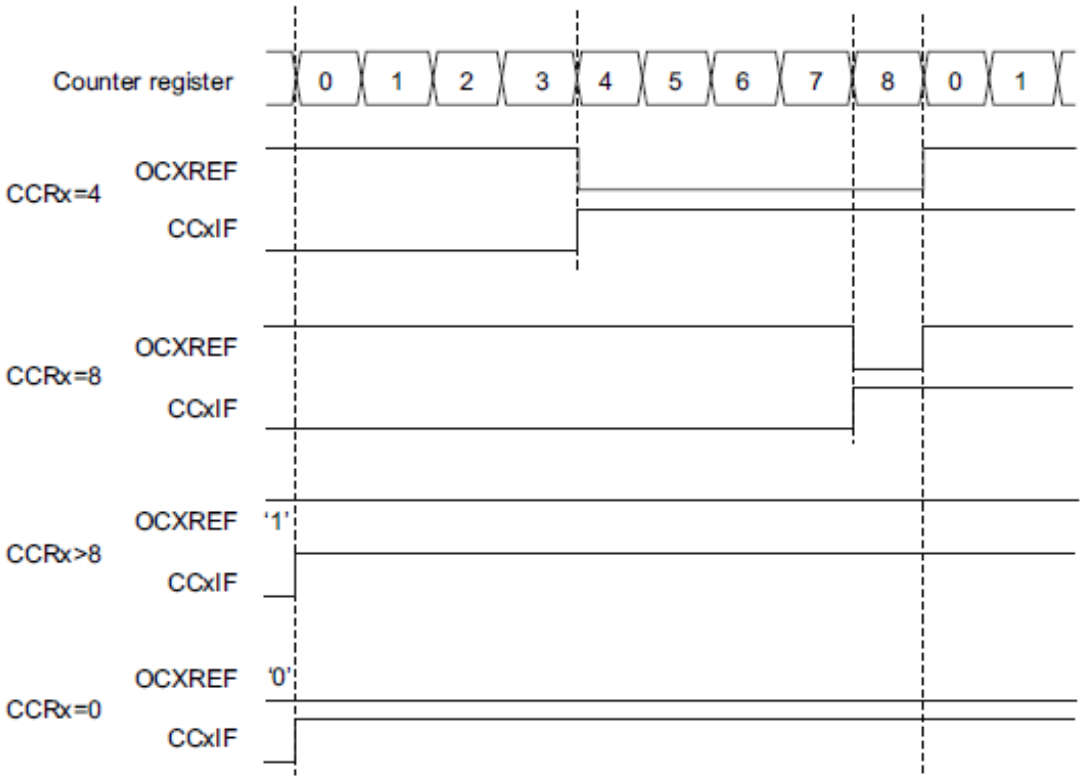


Fig. 6 Waveform for PWM

3. Experiments

3.1 Experiment 1: Using TIM3 and interrupt handler

In this experiment, the interrupt handler runs two times a second and toggles LED DS0. The system clock has been set to 72 MHz with EIE3810_clock_tree_init(). The clock signal is firstly divided by the pre-scalar 7200 (7199+1) and then by the counter 5000 (4999+1). Therefore the LED flashing period is 72,000,000/7,200/5,000/2 = 1 Hz.

Procedures:

1) Type in the codes as in Fig. 7. **There are two bugs in the code.** Try to find and correct them.
2) Compile and run the program with the project board.
3) Measure the DS0 flashing period with an oscilloscope.
4) Change to any other flashing frequency.

```
22  int main(void)
23  {
24    EIE3810_clock_tree_init();
25    EIE3810_LED_Init();
26    EIE3810_NVIC_SetPriorityGroup(5);//Set PRIGROUP
27    EIE3810_TIM3_Init(4999, 7199); //Add comments
28    while(1)
29    {
30      ;
31    }
32  }
33
34  void EIE3810_TIM3_Init(u16 arr, u16 psc)
35  {
36    //TIM3
37    RCC->APB1ENR|=1<<1;//Add comments
38    TIM3->ARR=arr;//Add comments
39    TIM3->PSC=psc;//Add comments
40    TIM3->DIER|=1<<0;//Add comments
41    TIM3->CR1|=0x01;//Add comments
42    NVIC->IP[30]=0x45;//Add comments
43    NVIC->ISER[0]=(1<<30);//Add comments
44  }
45
46  void TIM3_IRQHandler(void)
47  {
48    if (TIM3->SR & 1<<0)//Add comments
49    {
50      GPIOB->ODR ^=1<<5;//Add comments
51    }
52    TIM3->SR &= ~(1<<0);//Add comments
53  }
54
```

Fig. 7 Sample coding for Experiment 1

[Demonstration] When you have completed 3), demonstrate to instructor, TA or technician.

[In Report] Include your test result.

[Source code] Understand the code and add the comments.

[Question] What is the relation between the LED flashing period, pre-scalar value, and clock counter value?

3.2 Experiment 2: Using TIM4 and TIM3 together

STM32F103 has four General Purpose Timers. Based on Experiment 1, for DS0, set up DS1 with 4 Hz flashing rate and make it controlled by TIM4. In the end, the two LEDs are controlled by different timers.

Procedures:

1) Refer to Experiment 1. Set up TIM4 to control DS1 to have a flashing rate of 4 Hz. TIM3 will continue to work as in Experiment 1.
2) Compile and run the program with the project board.
3) Keep DS0 flashing at 1 Hz. Change DS1 to flash oppositely with DS0, i.e. when DS0 is on, DS1 is off, and vice versa.

[Demonstration] When you have completed 2) and 3), demonstrate to instructor, TA or technician, that your program works.

[In Report] Use an evidential way to show in your report that you have completed this experiment.

[Source code] Provide the source code with adequate comments.

3.3 Experiment 3: Setup SYSTICK with a 10ms periodic tick

In Experiment 2, we used two timers for two individual processes. Actually, one timer is sufficient to control a few processes, i.e., multi-tasking. Cortex-M3 has a special and simple 24-bit timer SYSTICK that can auto-reload and generate a periodic event to serve as the heartbeat of a system.

Procedures:

1) Type and complete the function "EIE3810_SYSTICK_Init()" as in Fig. 8. Include it into main.c. The SYSTICK will generate an interrupt every 10ms.

```
 99  void EIE3810_SYSTICK_Init()
100 {
101   //SYSTICK
102   SysTick->CTRL = 0; //Clear SysTick->CTRL setting
103   SysTick->LOAD = ???; //What should be filled?
104                        //Refer to Cortec-M3 Technical Reference Manual Page8-10.
105   SysTick->CTRL = ???;  //What should be filled?
106                        //Refer to Cortec-M3 Technical Reference Manual Page8-8.
107   //CLKSOURCE=0: FCLK/8
108   //CLKSOURCE=1: FCLK
109   //CLKSOURCE=0 is synchronized and better than CLKSOURCE=1
110   //Refer to clock tree on page 92 of RM0008
111 }
```

Fig. 8 EIE3810_SYSTICK_Init()

2) Modify SysTick_Handler() as shown in Fig. 9. This subroutine is in stm32f10x_it.c. Try to locate this file in your folder first. After modification, do not forget to include stm32f10x_it.c in the "Manage Project Items" dialog box, so that you can compile it.

```
135  void SysTick_Handler(void)
136 {
137   task1HeartBeat++;
138 }
139
```

Fig. 9 SysTick_Handler()

3) task1HeartBeat is a global variable. Define it in main.c. Moreover, you need to add the declaration in stm32f10x_it.h. Locate stm32f10x_it.h and add the declaration as below.

```
extern u8 task1HeartBeat;
```

Fig. 10 Declaration of task1HearBeat in stm32f10x_it.h

4) Type and modify the coding in Fig. 11 in main.c, and set DS0 and DS1 to flash at 5 Hz and 2 Hz, respectively.

```
while(1)
{
  if (task1HeartBeat>=11)
  {
    task1HeartBeat=0;
    //Write task1 here.
  }
}
```

Fig. 11 The while loop in main()

[Demonstration] When you have completed 4), demonstrate to instructor, TA or technician, that your program works.

[In Report] Use an evidential way to show in your report that you have successfully completed this experiment. Elaborate in details how you realize step 4).

[Question] How to change the SysTick frequency to 200 Hz?

[Source code] Provide the source code with adequate comments.

3.4 Experiment 4: Drive an LED with PWM

In this experiment, we will use a PWM signal that is generated by TIM3 to drive LED DS0 with various brightness.

TIM3 has four channels of PWM signals. Each channel corresponds to one physical I/O pin. The pin numbers are pre-defined and can be remapped to others as in Fig. 12. Because LED DS0 connects to PB5, we need to remap TIM3_CH2 output by setting TIM3_REMAP[1:0]='10'. The register related remapping bits are in the alternate function register AFIO_MAPR.

Alternate function	TIM3_REMAP[1:0] = "00" (no remap)	TIM3_REMAP[1:0] = "10" (partial remap)	TIM3_REMAP[1:0] = "11" (full remap) (1)
TIM3_CH1	PA6	PB4	PC6
TIM3_CH2	PA7	PB5	PC7
TIM3_CH3	PB0		PC8
TIM3_CH4	PB1		PC9

1. Remap available only for 64-pin, 100-pin and 144-pin packages.

Fig. 12 TIM3 alternate function remapping

Procedures:

1) Type EIE3810_TIM3_PWMInit() as in Fig. 13 into your main.c.

```
36  void EIE3810_TIM3_PWMInit(u16 arr, u16 psc)
37  {
38    RCC->APB2ENR |=1<<3; //Add comment
39    GPIOB->CRL &= 0xFF0FFFFF; //Add comment
40    GPIOB->CRL |= 0x00B00000; //Add comment
41    RCC->APB2ENR |= 1<<0;  //Add comment
42    AFIO->MAPR &= 0xFFFFF3FF; //Add comment
43    AFIO->MAPR |= 1<<11; //Add comment
44    RCC->APB1ENR |= 1<<1; //Add comment
45    TIM3->ARR = arr; //Add comment
46    TIM3->PSC = psc; //Add comment
47    TIM3->CCMR1 |= 7<<12; //Add comment
48    TIM3->CCMR1 |= 1<<11; //Add comment
49    TIM3->CCER |= 1<<4; //Add comment
50    TIM3->CR1 = 0x0080; //Add comment
51    TIM3->CR1 |= 1<<0; //Add comment
52  }
```

Fig. 13 EIE3810_TIM3_PWMInit()

2) Type the coding in Fig. 14 into main.c.

```
15  #define LED0_PWM_VAL TIM3->CCR2
16
17  int main(void)
18  {
19    u16 LED0PWMVal=0;
20    u8 dir=1;
21    EIE3810_clock_tree_init();
22    EIE3810_LED_Init();
23    EIE3810_TIM3_PWMInit(9999, 71); //Add comment
24    while(1)
25    {
26      Delay(1500);
27      if (dir) LED0PWMVal++; //Add comment
28      else LED0PWMVal--; //Add comment
29      if (LED0PWMVal >5000) dir=0; //Add comment
30      if (LED0PWMVal ==0) dir=1; //Add comment
31      LED0_PWM_VAL = LED0PWMVal; //Add comment
32    }
33  }
34
```

Fig. 14 main.c

3) Compile and download your programme to the project board. Measure the output of LED DS0 with an oscilloscope.

[Demonstration] When you have completed 3), demonstrate to instructor, TA or technician, that your program works.

[In Report] Use an evidential way to show in your report that you have completed this experiment.

[Question] If LED0_PWM_VAL is 0, 9999, or 10000, what happens to the DS0 signal in an oscilloscope, and what happens to LED0?

[Source code] Provide the source code with adequate comments.

3.5 Experiment 5: Read JOYPAD key input based on TIM3

The development board has a JOYPAD (Fig. 15) containing 10 keys, 8 of which are effective, including A, B, SELECT, START, UP, DOWN, LEFT, and RIGHT.

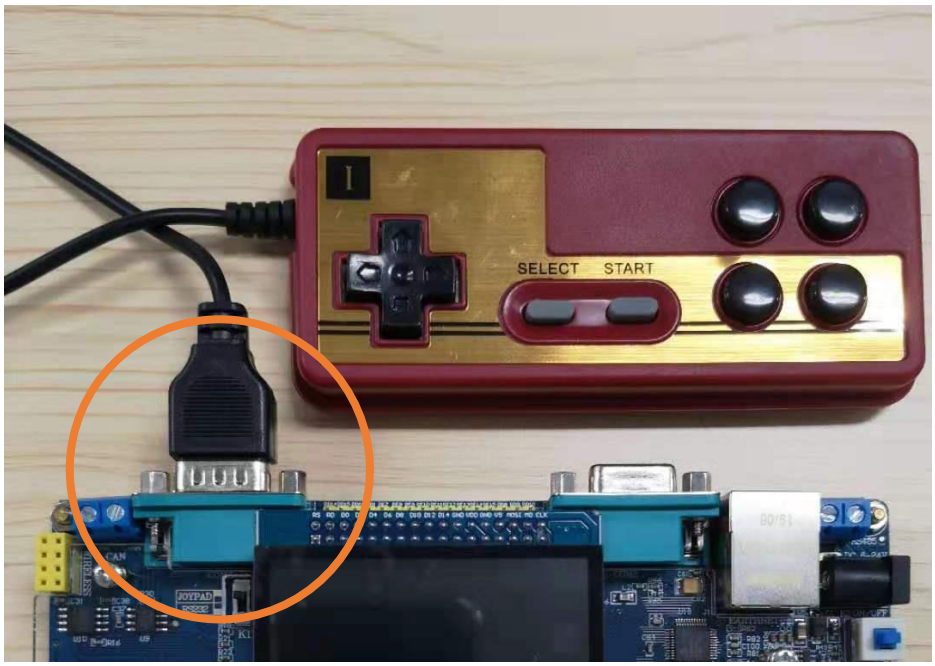


Fig. 15 JOYPAD connection

The JOYPAD is connected to the development board through the COM3 port on the upper left corner. The switch next to COM3 should be turned to “JOYPAD” rather than “RS232”, as shown in Fig. 16

Fig. 16 JOYPAD selection

Make sure the COM3 pins are connected to the microcontroller probably, as illustrated in Figure 17.

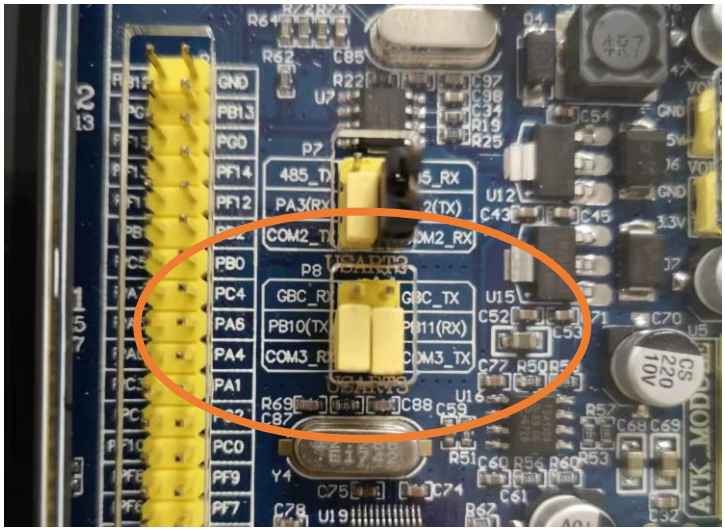


Figure 17. Locations of COM3 connection pins

Although JOYPAD utilizes the hardware COM3 port, it does not utilize USART for communication. Instead, there are 3 GPIO pins, i.e. PB11, PD3, and PB10.

- PB11: Output pin (JOYPAD_LAT). The microprocessor will first send out a pulse to the JOYPAD, and JOYPAD will then to put the key information to PB10.
- PD3: Output pin (JOYPAD_CLK). The microprocessor sends out 8 clock pulses (not very precise) to the JOYPAD. Before each rising edge, the microprocessor will read PB10. When JOYPAD receives the pulse, it will also determine whether to change the logic level on PB10 based on the key actions.
- PB10: Input pin (JOYPAD_DAT). The JOYPAD will send out the key information in this line. The voltage is high if there is no key input. The voltage will be changed to “0” in a time sequence based on Table 1. It can be observed that Fig. 18 is the signal for pressing “START”.

Table. 1 Keys and the corresponding time sequence

Keys	A	B	SELECT	START	UP	DOWN	LEFT	RIGHT
PB10 should be low, before the rising edge of pulse #	1	2	3	4	5	6	7	8

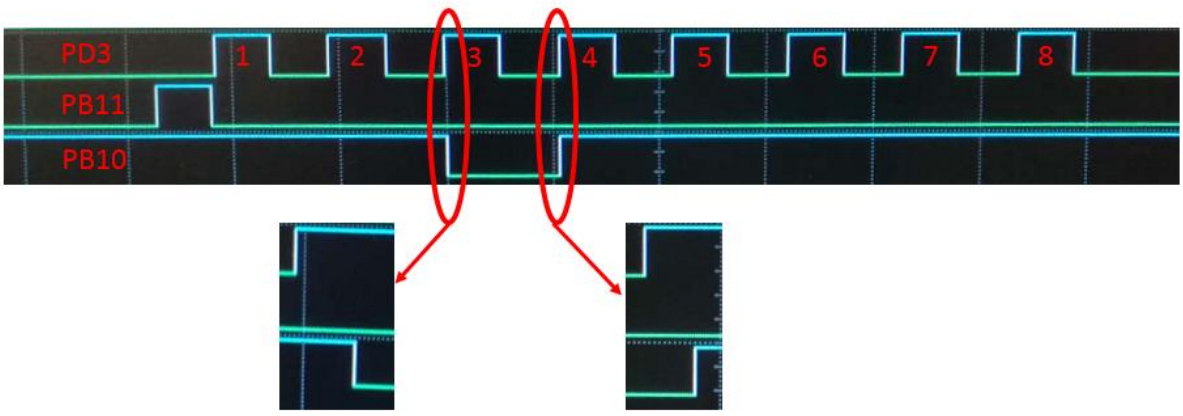


Fig. 18. Signals when START is pressed

Procedures:

1) Type the coding of Fig. 19 into main.c. Understand how it functions to read JOYPAD.

```
89  void JOYPAD_Init(void)
90  {
91    RCC->APB2ENR|=1<<3;//Add comments
92    RCC->APB2ENR|=1<<5;//Add comments
93    GPIOB->CRH&=0XFFFF00FF;//Add comments
94    GPIOB->CRH|=0X00003800;//Add comments
95    GPIOB->ODR|=3<<10; //Add comments
96    GPIOD->CRL&=0XFFFF0FFF; //Add comments
97    GPIOD->CRL|=0X00003000; //Add comments
98    GPIOD->ODR|=1<<3; //Add comments
99  }
100
101 void JOYPAD_Delay(u16 t)
102 {
103   while(t--);
104 }
105
106 u8 JOYPAD_Read(void)
107 {
108   vu8 temp=0;
109   u8 t;
110   GPIOB->BSRR |= 1<<11;//Add comments
111   Delay(80);//Add comments
112   GPIOB->BSRR |= 1<<27;//Add comments
113   for(t=0;t<8;t++)
114   {
115     temp>>=1;  //Add comments
116     if((((GPIOB->IDR)>>10)&0x01)==0) temp|=0x80; //Add comments
117     GPIOD->BSRR |= (1<<3);  //Add comments
118     Delay(80); //Add comments
119     GPIOD->BSRR |= (1<<19); //Add comments
120     Delay(80); //Add comments
121   }
122   return temp;
123 }
124
```

Fig. 19 JOYPAD related functions

2) It is not wise to have main() to repeatedly read JOYPAD in the while loop, as it will waste computational resource. You can read JOYPAD in the handler of TIM3 at a frequency of 100 Hz.
3) Complete the coding to realize: for each key pressed down, show the name of the key onto the LCD.

[Demonstration] When you have completed 3), demonstrate to instructor, TA or technician, that your program works.

[In Report] Include your test results.

[Source code] Provide the source code with adequate comments.

4. Lab Report and Source Code

Submit the report softcopy and your code (complete project folder of each experiment) in zip format to Blackboard by the deadline below:

- L01 (Friday Class): 9:00, Friday, December 1, 2023
- L02 (Tuesday Class): 15:00, Tuesday, November 28, 2023

Each day of late submission will result in 10% deduction in the report and source code raw marks.