Operating systems for embedded systems

Assignment

Prof. Massimo Violante

Matteo Pezzatta 261948
Luca Romanello 258279

A.A. 2019/20

Chapter 1

Objective

In this assignment we are expected to develop a tunable lamp which is capable of varying the intensity of colour and to change the colour of lamp. To fulfill such an objective, a *S32K144* evaluation board is required to be employed for implementing the lamp. Its potentiometer is exploited for varying the lamp intensity and the colour changing is addressed to the switches *SW2* and *SW3*.
As far as the operating system is concerned, the Micrium μc/OS3 is the one to be utilised.
The operations of the lamp are explained as follows:
At the start-up the LED has to be switch on with the RED colour and it is activated by a *10 KHz* PWM signal with the corresponding duty cycle proportional to the potentiometer level. Therefore, when the potentiometer level is on the lowest value ($0\ V$), the corresponding duty cycle will be 0%, while when the potentiometer output is $5V$ (i.e. the maximum) the corresponding duty cycle will be 100%. When the output is between these values, the LED is activated with a duty cycle proportional to the potentiometer status.
Then, the colour must be changed according to what switch is pressed:
The sequence for the SW2 is *RED - GREEN - BLUE - RED.*
The sequence for the SW3 is *RED - BLUE - GREEN - RED.*
If the switch remains pressed, the LED must not change colour.

Chapter 2

Implementation

General idea

In order to implement the blinking of the three LED (blue, red, green) we can exploit the fact that some channels of FTM (i.e. flex timer module) are present on the board and they are physically connected to the LED of the board. Therefore, when the specific FTM channel has an output on the corresponding pin, the LED will be blinking according to the output of that channel: when 1, the led will be off, while when it is 0 the led will be on.
By looking at the drawing '*S32K144EVB-Q100*' on the page '*S32K144 MCU*' of the S32K144 board, it is possible to evaluate which are the FTM channels that activate directly the *RGB_LED* of the board and to choose the corresponding one (*FTM0*). The following pins are the ones of PortD which is connected to the led as well as FTM channels within many other purposes. In the table above are reported the mapping of the pins we are interested in.

Therefore, it is possible to deal with the problem of blinking of LED by working directly on that specific **FTM0** channels. In the next chapter this problem is addressed.
The operation of changing the intensity of the led and the switching of the colour has been decoupled in order to concurrently manage both of the issues. To implement the color change, the board will be instructed on how to notify the CPU that a colour change request has arrived from the switches. Then, when the request arrives, the CPU will handle the colour change according to the current status of the **FTM0**. By looking at the board schematic, it is possible to get the pins to which the switches are connected. We will work on those pins.
J2-10/PTC12 - SW2
J2-12/PTC13 - SW3

ADC

PIN	**PORT**	**FUNCTION**	**LED**
J2-06	PTD0	FTM0_CH2	RGB_BLUE
J2-02	PTD15	FTM0_CH0	RGB_RED
J2-04	PTD16	FTM0_CH1	RGB_GREEN

An implementation of the analog to digital converter is needed for the reading of the potentiometer position. The potentiometer is connected to the **J4-14** pin via **ADC0_SE12** channel.
The support of ADC operations is chosen to be via interrupt. This choice allows to free the CPU from the process if it has to wait for a new value, avoiding busy time consuming. This operation is employed by means of a semaphore initialized to 0, which it is able to synchronize the application code with the interrupt handler.
After the initialization, the **ADC0** is ready to start conversion. In the **StartupTask** the software triggers the **ADC0** so that it can start the conversion of the read value according to the potentiometer position. Therefore, the result is expressed with a resolution of 12 bits, so its value can vary from *0* up to $2^{12} - 1 = 4095$ and it is saved into a 16 bit unsigned integer variable (*ADC0_adc_chx*).
When the conversion is done, the interrupt handler posts on the semaphore so that other processes that were waiting for the result can now use it.

PWM

The intensity of the light of an LED is proportional to the duty cycle of the pwm signal that is feeding the LED. By construction, the channels of a FTM of the board are activated by the same clock, but each of the channels can be individually treated for what concern its period and its duty cycle. The *modulo* of the specific channel will determine the period, while the *compare* is determining its duty cycle.
The system clock at normal run mode is of *80MHz* . Knowing the the application must ensure a led blinking frequency $f_{pwm} = 10kHz$, we have to choose the couple *modulo*divider* for the FTM0 as:

$$modulo * divider = \frac{f_{clock}}{f_{pwm}} = 8000$$

A prescaler equal to 1 has been choosen so that *modulo = 8000-1*. That value will fix the frequency of the PWM signals that are blinking the LEDs. The modulo value is channel-dependent but in that application it is specified that all the LEDs should be blinking with the same frequency. Also the compare is a channel-dependent value, and it is of high interest in that application, as it will be used to modify the intensity of the LEDs as stated before.
The value of the pwm duty-cycle is contained in the **CnV** memory mapped register according to the read value of the potentiometer position from the ADC0. When the **CLKS** bit in *FTM_SC* register is 1, which means that the FTM0 should be running, the compare value is updated with immediate load. This methodology requires a specific procedure that is well explained in one of the available manuals of the S32K144 board.
In addition to the standard initialization required to activate the FTM clock, it is needed to modify some additional registers:

- The **PWMSYNC** bit of the *FTM_MODE* register is set to 0, to let both software and hardware to update the *FTM_CnV* register;
- The **SYNCMODE** bit in *FTM_SYNCONF* register is set to 1 to enable enhanced synchronization of the PWM;
- **CNTINC** bit of *FTM_SYNCONF* is set, in order to enable the update of the *CNTIN* register;

- Then the register *SYNCENx* (where x is the channel n) is used to enable synchronization of the registers *C(n)V* and *C(n+1)V*. To enable channels *0,1,2* it is needed to set bits **SYNCEN0** and **SYNCEN2** of the *COMBINE* register.

So the piece of code regarding these specific modifications has been implemented as follows:

```
FTM0->MODE |= FTM_MODE_FTMEN_MASK; /* 0x1u DO WE HAVE TO MAKE A BOOLEAN OPERATION OR NOT? */

FTM0->MODE &= 0xF7; /* to clear the PWMSYNC bit of the MODE register */

FTM0->SYNCONF |= 0x00000080; /* to set the bit SYNCMODE of FTM_SYNCONF register */

FTM0->SYNCONF |= 0x00000005; /* to set the CNTINC bit of FTM_SYNCONF register */

FTM0->COMBINE |= 0x00002020; /*to set SYNCEN0 bit of the COMBINE register */
```

Then after initialization, inside the endless loop we have:

- A new function **BSP_FTM0_CH0_PWM_CnV_Update** is called in order to update the compare value of the channels of the timer. The update of the *CnV* registers is done concurrently for each of the three channels at every call of the function. The **LDCK** bit of the *FTM_PWMLOAD* register is set to let the registers that have been modified to update with immediate load.

It is important to notice that during the initialization of the timer, a *first reading* of the value of the potentiometer position through the **ADC0** is employed. This operation is not strictly necessary since it would be done in a mere amount of time when entering into the endless loop of the StartupTask, but for correctness it has been decided that at the initialization of the device the correct value of the position of the potentiometer is notified to the CPU and the duty-cycle of the pwm is set accordingly.
In addiction, at the initialization of the *FTM* only the **PWMEN0** bit (corresponding to CH0) of the SC register is set so that *RED LED* is activated at the start-up.
It is implemented in the system as follows:

```
void BSP_FTM0_CH0_PWM_CnV_Update(void)
{
  FTM0->CONTROLS[0].CnV = FTM_CnV_VAL((8000-1)*(4095-ADC0_adc_chx)/0xFFF);
  FTM0->CONTROLS[1].CnV = FTM_CnV_VAL((8000-1)*(4095-ADC0_adc_chx)/0xFFF);
  FTM0->CONTROLS[2].CnV = FTM_CnV_VAL((8000-1)*(4095-ADC0_adc_chx)/0xFFF);
  FTM0->PWMLOAD |= 0x200; /* to set the LDOK bit of the PWMLOAD register */
}
```

In this piece of code, it is possible to notice that the value which has to be set as compare is the **complementary value** of *ADC0_adc_chx* with respect to the maximum allowed 4095. That is because the *LEDs* are *ON* when the value on the channel of the *FTM0* is 0. Therefore, when we think of the *LEDs* in *ON* state, we should think of the complementary duty-cycle of the *FTM0*.

Switches

In the initialization of the switches, for managing both the **SW2** and the **SW3** it is decided to operate as a *digital level-triggered input* via **interrupt**. Therefore, it is necessary to set both *PTC12* and *PTC13* pins as inputs and GPIO functions. The peripheral

has been configured to generate an interrupt when a rising-edge event happens (the button is pressed and the state changes from 0 to 1) and an interrupt handler function is executed when such an event happens. Only one Interrupt Handler function is created for addressing the interrupts from the two switches. The interrupt handler is as follows:

```
static void SW_int_hdlr(void)
{
   uint32_t ifsr2;
   uint32_t ifsr3;

   //CPU_CRITICAL_ENTER();
   OSIntEnter();

   ifsr2 = (PORTC->PCR[12]) & 0x01000000;
   ifsr3 = (PORTC->PCR[13]) & 0x01000000;

   if((ifsr2))
   {
      SW2_status = BSP_Switch_Read( SW2 );
      PORTC->PCR[12] |= 0x01000000;
   }

   if((ifsr3))
   {
      SW3_status = BSP_Switch_Read( SW3 );
      PORTC->PCR[13] |= 0x01000000;
   }

   //CPU_CRITICAL_EXIT();
   OSIntExit();
}
```

At the end, it is necessary to enable the interrupt for *PortC*.
When the event arrives, the **SW_int_hdlr** is run and it is able to recognize if and which one of the pins generated the interrupt. Then the **SW_int_hdlr** function saves the status of the switch on a global variable (*SWn_status*) in order to be read also by external functions. [9]
The update of the variable **SWn_status** is catched inside the endless loop of the StartupTask and it allows to call the **BSP_CH_switching** function, independently of the duty-cycle update. At the end, before the endless loop there will be the initialization of the ADC, PWM and Switches initialization functions.

```
BSP_ADC0_init_interrupt(); /* initialization of the ADC0 */

BSP_FTM0_CH0_PWM_Init(); /* initialization of the PWM */

BSP_Switch_Init(); /* initialization of the switch */ /*it switch on all led*/

while (DEF_TRUE)
{
    BSP_ADC0_convertAdcChan_interrupt(12); /* start conversion with ADC0 */

    /* change of colour */
      if(SW2_status)
      {
        BSP_CH_switching_2();
        SW2_status = 0;
      }
```

```
    if(SW3_status)
    {
      BSP_CH_switching_3();
      SW3_status = 0;
    }

  /* updating of the duty-cycle */

  OSSemPend(&ADC0sem, 0u, OS_OPT_PEND_BLOCKING, 0u, &os_err);

  BSP_FTM0_CH0_PWM_CnV_Update();
}
```

In fact, at each cycle, after starting the ADC conversion, the status of the two switches is checked with an *if statement* each, and when the condition about the switch status is *true*, it enters in a function that is intended to change the colour of the LED by updating the *SC* register of the *FTM0*, that is:

```
void BSP_CH_switching_2(void)
{
  uint32_t colour_mask;
  colour_mask = ((FTM0->SC) & (FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK | FTM_SC_PWMEN2_MASK));

  switch(colour_mask)
  {
     case FTM_SC_PWMEN0_MASK :
        FTM0->SC &= ~FTM_SC_PWMEN0_MASK;
        FTM0->SC |= FTM_SC_PWMEN1_MASK;
      break;
     case FTM_SC_PWMEN1_MASK :
        FTM0->SC &= ~FTM_SC_PWMEN1_MASK;
        FTM0->SC |= FTM_SC_PWMEN2_MASK;
     break;
     case FTM_SC_PWMEN2_MASK :
        FTM0->SC &= ~FTM_SC_PWMEN2_MASK;
        FTM0->SC |= FTM_SC_PWMEN0_MASK;
      break;
  }
}
```

That function is checking the *SC* register of the *FTM0* so that to identify which of the channels is active (**PWMEN0** bit - CH0, **PWMEN1** bit - CH1, **PWMEN2** bit - CH2) through a *switch statement*. Then, it clears the currently active channel and it activates the channel corresponding to the specific case, according to the given switch color sequence. Two of these functions have been built: one regarding SW2, the other related to SW3.

Results

The potentiometer voltage level, the frequency and the duty-cycle of the channels have been verifyied by means of *WaveForms sotware* and the use of the *Analog Discovery 2* board. The analog voltage of the potentiometer has been measured using one analog channel wire connected to the **J4-14** pin of the *S32144K* board corresponding to port *PTC14*. As regards to the measure of the digital signal coming form channel *CH0* of the

FTM0, the **DIO0** of the Analog Discovery 2 board has been connected to the **J2-02** pin corresponding to PTD15 port that corresponds to the red LED.
In fact, we have verified that when the voltage on the potentiometer is $5V$, the LEDs are at their maximum intensity level, while when we have $0V$ on the potentiometer, the LEDs are turned off. For intermediate voltage levels we have intermediate intensities of the LEDs.
As far as the frequency and duty-cycle are concerned, one period of the square waveform is measured by measuring the time of two consecutive rising waves, and starting from it, a value close to *10kHz* of frequency is founded out. The time values of the rising edges with respect to the zero of the time axis of WaveForms Logic mode were:

$$T_1 = \text{-}0.0000291 \text{ s} \; ; \; T_2 = 0.0000716 \text{ s} \; ; \; \text{T} = 0.0001007 \text{ s}$$

so the frequency will be approximately *9940 Hz*.
As we expected, being T_1 and T_2 measured with the PC coursor, the frequency is not exactly the perfect one, but it has to be considered only for an approximate verification.

Bibliography

[1] NXP *S32K1xx Serie Reference Manual*

[2] NXP *Feature of the FlexTimer Module - https://www.nxp.com/docs/en/application-note/AN5142.pdf*

[3] NXP *S32K144EVB-Q100*

[4] *bsp_adc_assignment.h*

[5] *bsp_adc_assignment.c*

[6] *bsp_pwm_assignment.h*

[7] *bsp_pwm_assignment.c*

[8] *bsp_switch_assignment.h*

[9] *bsp_switch_assignment.c*

[10] *main_assignment.c*

[11] Kopka, H. e Daly, P. W. (1995). *A Guide to LaTeX – Document Preparation for Beginners and Advanced Users*. Addison-Wesley.