Université Paul Sabatier TOULOUSE III

Lab Work

CSA – Architecture, System, Network

Lab Work 2 : Using the Timer

1 Timer by Polling

In this exercise, we will use the timer to make the yellow LED to blink at 2Hz, that is, twice par second. Rephrasing it, this means that the state of LED (from on to off and from off to one) has to be changed each 0.5 s. To count this duration, we will use the timer by polling a state bit to test if the duration has been elapsed.

To count 0.5 s, you have to keep in mind that:

- the maximal capacity of the timer is a register of 16-bits, i.e. it can count from 0 to 65535,
- it counts the number of pulses coming from MCK (Master Clock) which frequency is 4MHZ = 4,000,000 Hz,
- as this frequency is too fast to count 0.5 s (we should have to count 2,000,000 pulses while the timer registed is limited to 65,535), we can apply one of the divider on the MCK input frequency: clock1 (MCK / 2), clock2 (MCK / 8), clock3 (MCK / 32), clock 4 (MCK / 128), clock 5 (MCK / 1024).

The first step is to choose which divider to apply, that is, which divider enables to store the equivalent of 0.5 s in the 16-bit register of the timer (basically if 4,000,000 MCK pulses corresponds to 1 s, 2,000,000 pulses will correspond to 0.5 s) :

- with clock1, we have to count 2,000,000 / 2 = 1,000,000 in the timer – this is too much as maximal count is 65,535;
- with clock2, we have to count 2,000,000 / 8 = 250,000 – still too much
- with clock3, 2,000,000 / 32 = 62,500 that is less than 65,535 – good candidate!

Now, we have to configure the timer. This requires the following steps to be performed:

- stop the timer by writing to TC0_CCR,

```
TC0_CCR = TC_CLKDIS;
```

- configure the timer by setting TC0_CMR (clock 3 and reset on equality to RC – CPCTRG) and TC0_RC with the amount of time to count:

```
TC0_CMR = TC_TCCLKS_CLOCK3 | TC_CPCTRG;
TC0_RC = DELAY_IN_PULSES;
```

- restart and reset the timer by writing to TC0_CCR.

```
TC0_CCR = TC_SWTRG | TC_CLKEN;
```

Then, we have to configure the PIO to implement the blinking and, inside the main loop,

we have to test the bit TC_CPCS (set when CV = RC, equality of counter with RC) of the register TC0_SR to know when to change the state of the LED.

TO DO Implement the blink application inside `lab2/lab21`.

2 Timer with Interrupts

Another way, more flexible, to react to the events of the timer is to use the interrupt system. The timer can produce several types of interrupts but we are here only interested about causing an interrupt when the counter is equal to RC, bit TC_CPCS. To reset this interrupt bit in the timer, one has to read the register TC_SR to acknowledge the interrupt from the timer.

Briefly, to issue the interrupt from the timer, we have to:

- enable the interrupt from timer in AIC, line 12 defined as ID_TC0 (setting AIC_SMR, AIC_SVR and AIC_IECR);
- enable the interrupt in the timer (setting TC0_IER with bit TC_CPCS).

To acknowledge the interrupt, we have to:

- read register TC0_SR,
- read register AIC_EOICR.

TO DO Using the source file from `lab2/lab22`, implement the blinking LED application using interrupts.

3 Fast Click Game

In this exercise, we want to implement a game where the player has to click as fast as possible during a certain delay. The score is then displayed using a terminal connected to the USART.

To start the game, the player has to perform a click on PUSH1. Then, he has 5 s to click as fast possible on PUSH2. While the game is running (recording the number of clicks), the YELLOW LED is on and then off when the game is completed. The GREEN LED is on when PUSH1 is pressed and off when PUSH1 is released.

TO DO Move to directory `lab2/lab23` and implement the game using your preferred programming method (polling or interrupts).

NOTE To display the result to the user, you have use the USART (Universal Serial Asynchronous Receiver Transmitter) that is connected to the terminal of *BoradSim*. `Lab23.c` contains already the code to initialize the USART and functions to transmit characters:

- `putc`(*c*) – send character *c* to the terminal,
- `puts`(*s*) – send string *s* to the terminal.

As our is embedded and therefore has limited resources, there is no `printf`() function: you have t write yourself the conversion of the number of clicks into characters to display them to the user.