

ITU Control & Automation Eng. Dept.

KON309E Microcontroller Systems

Experiment 3: Interrupts



Aim: Using interrupts to cycle through LEDs and control their brightness using PWM.

For this experiment, you will need to consult the usual reference documents:

- LPC824 user manual
- LPC82x datasheet
- Alakart schematic diagram

Write the code using the peripheral support libraries (Xpresso SDK).

In this experiment, we will keep the three LED configuration, but only one external switch. The three LEDs will be turned ON sequentially using a timer INT and their brightness will be cycled by pressing the external button. The external button will be connected through a pin interrupt (PINT).

The main loop of the program must not do anything. It must be completely empty.

PART I

1. Construct a circuit consisting of three LEDs of any color you like and one external button.
 2. Make sure that the processor system clock is configured to 30MHz (you may copy the clock configuration code provided in the class example projects)
 3. Configure the **MRT timer** such that it generates an INT every 10 ms.
 4. Write an ISR for the MRT. The ISR must have a counter that counts to 50, turns off the current LED and turns on the next LED.
 5. At this stage, you should get a sequencer that changes to the next LED every 500ms.
 6. **Use a chronometer to check that you really get the correct timing.** (Time 10 complete sequences and make sure it is close to 15s= 10 x 3 x 0.5s)
- Note: The clock input of the MRT peripheral is the "system clock" which is set to 30MHz. See Appendix 1 and also "Fig. 49 MRT clocking" of the user manual.

Write in your report:

What values must be written to which registers so that the desired INT frequency of 10ms can be accomplished.

Store the code for this part in a folder named "EX3_PART1".

PART II

In PART I, we built a LED light sequencer. Now we will add LED brightness functionality to the light sequencer by using PWM. Keep the code in PART I and add the following to it:

1. Connect the LEDs to the outputs of the SCT **using the SWM peripheral**.
2. Configure the SCT to generate a PWM signal such that the brightness of the LEDs can be changed.
3. Configure the PWM frequency to be 10kHz.
4. The MRT INT should still be active, and still served by the ISR as in PART I.
5. Every 500ms, change the **PWM duty ratio** of the currently ON LED to 0%, and the next LED to 50%. (You can use more or less PWM duty ratio to get a visually pleasing brightness.)

As a result, we should be getting the same effect as in PART I, but the LEDs should be dimmer.

Write in your report:

1. How the SWM is configured so that the LEDs are now connected as PWM outputs of SCT. What values must be written to which registers?
2. How the SCT is configured to produce 3 PWM outputs. What values must be written to which registers?

Store the code for this part in a folder named "EX3_PART2".

PART III

In PART III, we will change the LED brightness by pressing the button. Keep the code of PART II and add the following functionality to it: Set a pin interrupt (PINT) for the external button. When we press that button, the default brightness of the LEDs should cycle through 5%, 10%, 25% and 99%:

1. Configure the external button as a PINT such that **when it is pressed**, an INT is triggered (i.e. correctly configure falling edge or rising edge).
2. In the ISR of the PINT, change the value of a **volatile** global variable: It must default to 5 and at each execution of the PINT ISR, it should cycle through the following values: 10, 25, 99, 5, 10, 25...

As a result, when the button is pressed, the brightness of the LEDs in the sequencer will change from dim to brighter and back to dim again. The sequencer speed will remain the same.

We use seemingly strange values for the PWM duty ratio (5%, 10%, 25% and 99% rather than evenly spaced 25%, 50% 75% and 100%) because of the nonlinear way the human eye sees brightness. You can use a different set of 4 duty ratios that is pleasing to you. If you do so, explain to the lab assistant what values you have used.

Write in your report:

1. How the SWM and Pin Interrupt peripherals are configured so that the external button is configured as a PINT. What values must be written to which registers?

2. Describe how you link the two ISRs so that the value set in the PINT ISR is used in the MRT ISR to set the PWM duty ratios.

Store the code for this part in a folder named "EX3_PART3".

PART IV: Bonus

This is the fun part. Try to obtain different visual effects. You for example you may implement a simple pseudorandom number generator to set the timing, sequence and brightness of the LEDs, or do other visual effects. How about doing an informal competition among lab groups?

If you implement this part, describe in your report what you intended to do and how you implemented the idea.

You do not need to provide a code for this part; it will be evaluated by the class assistant and fellow students.

A bonus of +5 points will be given.

Note: Please pay attention to the following:

- **Insert Alakart into the breadboard also** when preparing your circuit.
- Use a $10k\Omega$ pull up resistor for button and 220Ω series resistors for LEDs.
- The long pins of the LEDs are anodes.
- Connect V_{DD} (3.3V) and **Ground** pins of the microcontroller to the breadboard's (+) and (-) sockets via jumpers.
- The processor pin where the button is connected are set as a digital input, and those where the LEDs are connected are set as digital outputs.

Report Content

Your report must be a formal account of what you did in the experiment. Make sure that it includes the following items:

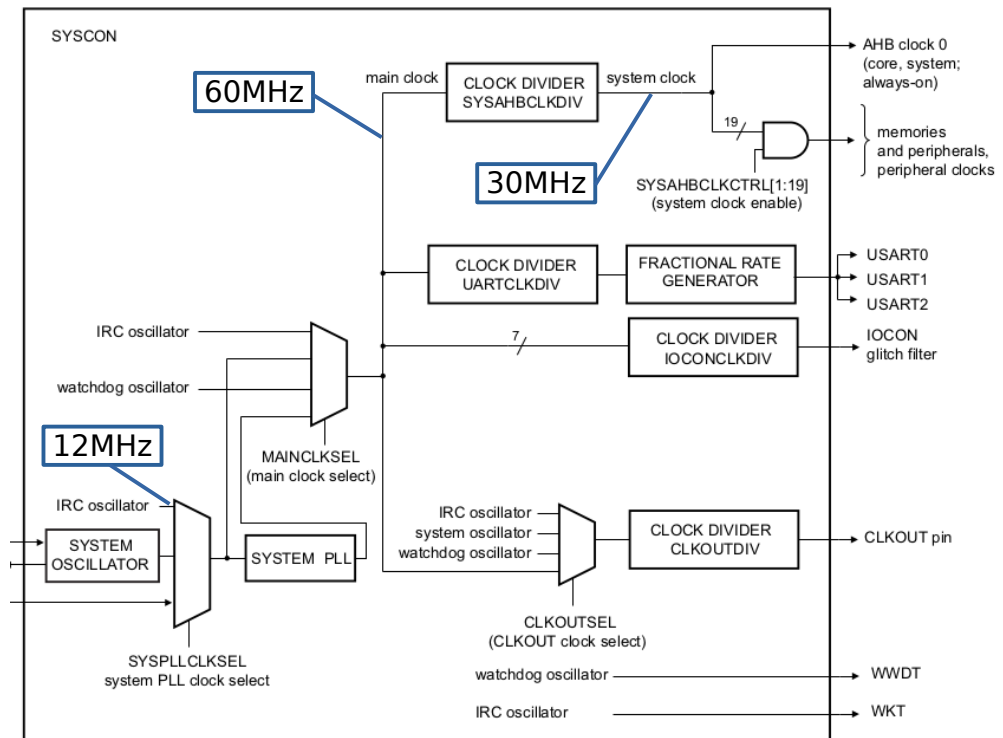
- How you have configured the processor i.e.:
 - What pins are inputs, what pins are outputs, etc.
 - Which peripheral devices were explicitly powered up,
 - How the GPIO and other peripherals were configured,
 - What values were written to which registers,
 - Which default settings of the peripherals were used.

These must appear in all of your reports.

- Propose a PWM frequency and **show how you calculate the prescaler and match register value for that frequency.**
- The finite state machine diagram that you have designed. Use **draw.io** to draw it. Make sure that you **clearly mark the state names, transition conditions and outputs.**
- **Any inclusion of code must be as formatted text.**
(Photographs or screenshots are not accepted!)
- What problems you have encountered and how you solved them.

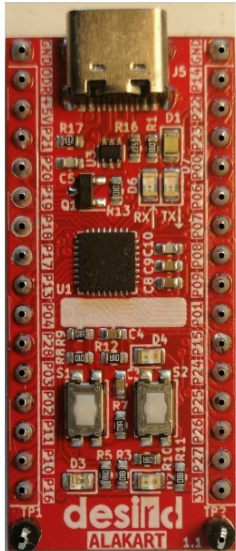
Appendix1: Clock configuration of the microcontroller

Please refer to Fig 5 of the User Manual. The internal RC clock (**IRC oscillator**) of the microcontroller is used at 12MHz. It is multiplied to 60MHz by the "**SYSTEM PLL**" to obtain the "**main clock**" and then divided by 2 in the "**CLOCK DIVIDER SYSAHBCLKDIV**" block to obtain the 30MHz "**system clock**". The system clock is used both in the microprocessor core and connected to the peripheral devices.



Appendix2: Pin connections of components on Alakart

- LEDs:
 - RED: D4 on GPIO PIN12
 - Blue: D3 on GPIO PIN16
 - Green: D2 on GPIO PIN27
 - White: D1 Power on
 - Green: D6 Transmit to PC
 - Red: D7 Transmit from PC
- Buttons:
 - S1: Reset
 - S2: ISP (enter boot mode)
Also: User button.
- Red Pins: Test
 - TP1: GPIO PIN16
 - TP2: GPIO PIN27
- Purple Pin: Open drain FET
 - ODR: GPIO PIN21

- GND
 - ODR
 - +5V
 - P21
 - P20
 - P19
 - P18
 - P17
 - P13
 - P04
 - P28
 - P03
 - P02
 - P11
 - P10
 - P16
 - TP1
- 
- GND
 - P14
 - P22
 - P23
 - P00
 - P06
 - P07
 - P08
 - P09
 - P01
 - P15
 - P24
 - P25
 - P26
 - P27
 - +3V3
 - TP2

Note:

- PIO0_10, PIO0_11 are open drain pins. Research what this means for the experiment.
- PIO0_2, and PIO0_3 are debugger pins by default. PinPIO0_5 is Reset pin by default. You can use them if you wish, but will **need to disable their default functionality in PINENABLE0 register** first.

If you wish to disable the Reset pin and make it into an ordinary pin, you will lose the ability to re-program the processor from your PC. Let's see how programming works. The processor is put into programming mode by the following sequence:

1. Processor is put into reset mode by pulling the Reset pin low.
2. The ISP (boot mode select) pin: PIO0_12 is pulled low.
3. Processor is taken out of reset by pulling the Reset pin high.

Then the code execution is not performed and the processor is in a boot mode where the flash ROM can be programmed.

By turning the Reset pin into an ordinary GPIO pin, you lose the ability to reset the processor from your PC. However, you can still re-program the processor by manually pressing the switches and applying power:

1. Remove power (unplug the USB connector)
2. Keep SW2 (ISP) pressed.
3. Apply power (USB). The processor is in a power-up reset state and the ISP pin has been pulled low. Therefore it goes into the boot mode and can now be programmed from the PC normally.