AGH University of Science and Technology
IEiT
Department of Electronics

MICROPROCESSORS TECHNOLOGY II

Exceptions and Interrupts

Paweł Russek
http://www.fpga.agh.edu.pl/upt2
15 Nov 2016

1 Introduction

1.1 Goal

The main objective of this tutorial is to introduce exceptions mechanism in today's microcontrollers. Freescale's Kinetis L family which is ARM Cortex M0 based solution will be use for that purpose. The basics of exceptions will be recalled and Kinetis specific programming will be presented. Energy efficiency of interrupt-driven programming will be highlighted.

1.2 Prerequisites

- PC with pre-installed Keil's uVision software
- FRDM-KL46Z development kit
- Basic knowledge of software development for FRDM-KL46Z using Keil's uVision Keil's uVision

1.3 Literature

- Joseph Yiu, The Definitive Guide to the ARM Cortex-M0, Elsevier, 2011
- KL46 Sub-Family Reference Manual, Freescale Semiconductor
- Kinetis L Peripheral Module Quick Reference, Freescale Semiconductor
- Power Management for Kinetis and ColdFire+ MCUs. , Freescale Semiconductor

2 Theoretical background

2.1 Exceptions and Interrupts

Exceptions are events that cause change to program control: instead of continuing program execution, the processor suspends the current executing task and executes a part of the program code called the exception handler. After the exception handler is completed, it will then resume the normal program execution. The events could either be external or internal. When an event is from an external source, it is commonly known as an interrupt or interrupt request (IRQ). The exception handlers for interrupt events are commonly known as interrupt service routines (ISRs).

2.2 ARM Cortex exceptions support

Cortex-M0 and the Cortex-M1 processors include a Nested Vectored Interrupt Controller (NVIC). If an interrupt is accepted, it communicates with the processor so that the processor can execute the correct ISR.

The Cortex-M0 processor supports up to 32 external interrupts - commonly referred as IRQs. In addition, the Cortex-M0 processor also supports a number of internal exceptions.

The exception numbers, from 1 to 15, are for system exceptions generated inside the processor, and the exception numbers for the on-chip peripherals and external interrupt sources are

from 16 to 47. Interrupts are usually generated by on-chip peripherals, or by external input through I/O ports.
For example System Tick Timer (SysTick) exception is internally generated by NVIC. The exception number for SysTick is 15. The System Tick Timer provides a regular interrupt for the system.

Exception Type	Exception Number	Description
Reset	1	Power on reset or system reset.
NMI	2	Nonmaskable interrupt—highest priority exception that cannot be disabled. For safety critical events.
Hard fault	3	For fault handling—activated when a system error is detected.
SVCall	11	Supervisor call—activated when SVC instruction is executed. Primarily for OS applications.
PendSV	14	Pendable service (system) call—activated by writing to an interrupt control and status register. Primarily for OS applications.
SysTick	15	System Tick Timer exception—typically used by an OS for a regular system tick exception. The system tick timer (SysTick) is an optional timer unit inside the Cortex-M0 processor.
IRQ0 to IRQ31	16 - 47	Interrupts—can be from external sources or from on-chip peripherals.

Table 1: ARM Cortex M0 exception types

2.3 CMSIS and Tools Support for Exceptions

Cortex Microcontroller Software Interface Standard (CMSIS) provides a standardized software interface to the processor features. The CMSIS standardizes access functions for accessing NVIC, and System Tick timer (SysTick) exception such as interrupt control and SysTick initialization.
Interrupt service routines are able to be coded directly as C functions without the need for an assembler.

2.4 Vector Table

The beginning of the program image contains the vector table. It contains the starting addresses (vectors) of exceptions. Vector table and interrupt names are device specific, defined in device specific start-up code.

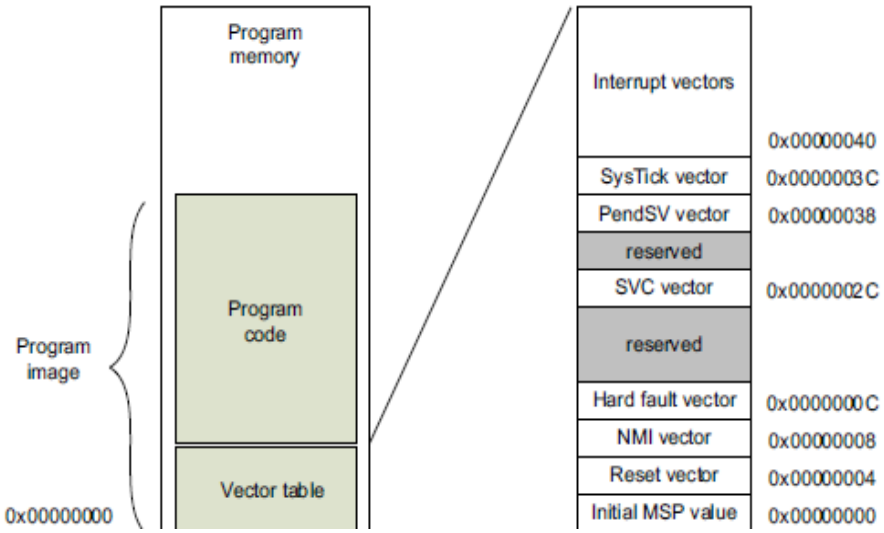


Figure 1: Vector Table in Program Image

2.5 Interrupt-Driven Programming

The Cortex-M0 processor can have much lower average power consumption in Interrupt-driven application. Whenever application requires lower power, processing can be carried out in interrupt service routines so that the processor can enter sleep mode when no processing is required. Interrupts are usually generated by external sources or on chip peripherals to wake up the processor.

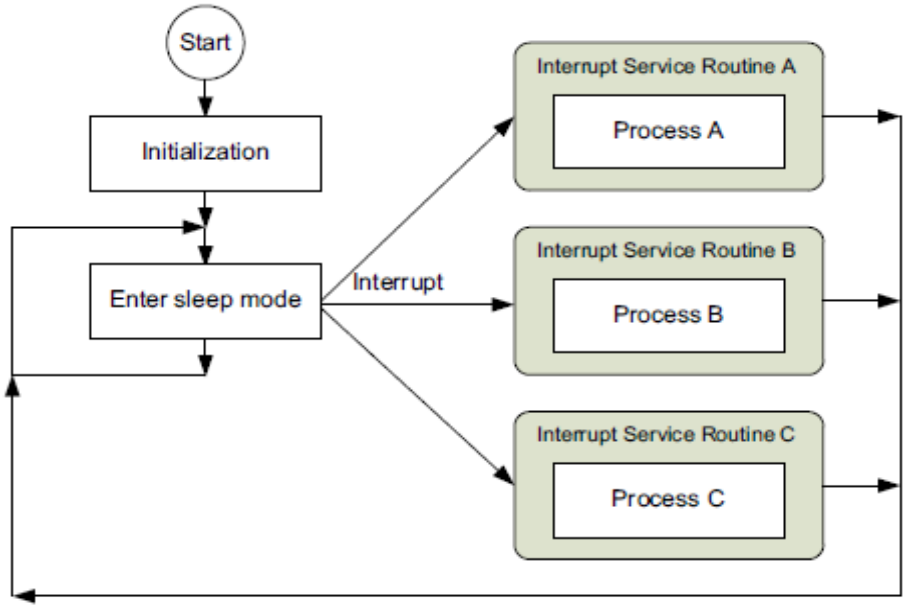


Figure 2: Interrupt driven application

2.6 ARM Cortex-M0 Power Modes

The ARM Cortex-M0+ power modes are Run, Sleep, and Deep Sleep. To enter Sleep state ARM Cortex-M0+ uses a wake from interrupt (WFI) instruction. Additionally when the DEEPSLEEP bit in the SCR is set to 1, ARM Cortex-M0 invokes Deep Sleep mode on WFI instruction. NVIC can wake-up processor from Sleep to Run mode. However, NVIC can be put in the state of retenion in Deep Sleep.

2.6.1 Wakeup Interrupt Controller

The Wakeup Interrupt Controller (WIC) is an additional unit. In low-power applications, the microcontroller can enter standby state with most of the processor units powered down. In this situation, the WIC can perform the function of interrupt servicing while the NVIC and the processor core are inactive. When an interrupt request is detected, the WIC informs the power management to power up the system so that the NVIC and the processor core can then handle the rest of the interrupt processing.

2.6.2 Kinetis Power Modes

Kinetis L family of microcontrollers features eleven different power modes. Run, Wait, and Stop modes of Kinetis are equivalents of ARM's Run, Sleep, and Deep Sleep. There are VLPR, VLPW, VLPS, LLS, and VLLS3/2/1/0 modes additionally offered by Kinetis (see table below).

The wake up from LLS and VLLS mode is enabled by LLWU module. LLWU is a Kinetis specific module. This on-chip peripheral module is called a Low-Leakage Wakeup Unit (LLWU) because

it allows internal peripherals and external input pins as a source of wakeup from low-leakage modes. It is operational only in LLS and VLLSx modes.

Power Mode	CPU	NIVC	Peripherals	Wake-up
Run	On	On	On	-
VLPR (Very Low Power Run)	2-4 MHz	On	On	CPU/through bits in SCGSx
Wait	Off	On	On	NVIC
VLPW (Very Low Power Wait)	Off	On	On	NVIC
Stop	Off	Off	Off	WIC
VLPS (Very Low Power Stop)	Off	Off	Off	WIC
LLS (Low Leakage Stop)	Off	Off	some are On	LLWU
VLLS3 (Very Low Leakage Stop3)	Off	Off	some are On	LLWU/through reset flow
VLLS2 (Very Low Leakage Stop2)	Off	Off	some are On	LLWU/through reset flow
VLLS1 (Very Low Leakage Stop1)	Off	Off	some are On	LLWU/through reset flow
VLLS0 (Very Low Leakage Stop0)	Off	Off	some are On	LLWU/through reset flow

3 LABORATORY PREPARATIONS

3.1 Download the interrupt laboratory zip file from the laboratory web site.

Link: http://www.fpga.agh.edu.pl/upt2/?download=2014_10_26_int_tutorial.zip

3.2 Unpack the file to the selected directory on your laboratory PC.

Directory: c:/MDK_ARM/<lab_day>_<lab_time>_<students_names>/IntLabProj

3.3 Locate and open the uVision's project file 'int_tutorial.uvprojx'

The contains several files. In the project's folder 'src' you have:

- main.c – contains main() function,
- leds.c – contains leds and FSM related functions,
- buttons.c – contains SW1 button procedures,
- lptimer.c – contains Low Power Timer (LPTMR) services,
- lowpower.c – contains functions to put microcontroller into Low Leakage Stop mode.

3.5 Build the project and download binaries to your FRDM board. After reset, the board should blink twice with the green and red leds if everything is done correctly.

3.4 Open main.c file.

Each of leds, buttons, lptimer, and lowpower related files has an initialize function which is called by main() before it enters an infinite loop (refer to figure 2).

3.5 Open leds.c file and locate ledsInitialize() function.

The steps in IO device initialization procedure are:

- Programming the clock control circuitry to enable the clock signal to the peripheral,

- programming of I/O configurations,
- welcome procedure that ensures device and configuration correctness.

3.6 Locate above steps for leds and customize your welcome procedure according your own taste.

4 System Tick Exception programming

4.1 The vector table

The vector table for your IntApp application is defined in a program start-up code. The code is located in startup_MKL46Z4.s file.

4.1.1 Open startup_MKL46Z4.s

The vector table can be easily found as it is marked by a label '__Vectors'. Reset is a special type of exception. It is located at address 0x000_0004.

Note: The address 0x0000_0000 of ARM Cortex-M0 contains an initial stack pointer value.

Before your program enters main() function, it starts execution at address labelled 'Reset_Handler'. Necessary system initialization is performed before the CPU executes your code.

4.1.2 Determine the name and locate a code of the system initialization function which is performed in the reset handler before it jumps to main() function.

4.2 Enabling System Tick (SysTick) exception

System tick (SysTick) exception handler name is already defined in the vector table.

4.2.1 Find the name of SysTick exception handler in startup_MKL46Z4.s file

The SysTick handler definition has a [WEAK] modifier and it can be overwritten by a user code.

Note: The modifications of the program code that should be done by you in this tutorial are denoted by a 'ToDo_<nbr>' mark. The mark is given in a comment and a <nbr> refers to the modification number.

4.2.2 Replace the name of the function 'executeEvery1ms()'in 'main.c' file with a proper name of SysTick handler which is defined in startup_MKL46Z4.s file. [ToDo 4.1]

There is SysTick_Config() function in CMSIS library. It is the System Tick Configuration function that initializes the System Timer and its interrupt. It starts the System Tick Timer counter is in free running mode to generate periodic interrupts. The function takes number of ticks between two interrupts as a parameter.

The system variable SystemCoreClock gives the system clock frequency value. It can be used to trim the SysTick interrupts intervals.

4.2.3 Enable and configure SysTick exception using SysTick_Config() function. [ToDo 4.2]

4.2.4. Build the application and run on the FRDM board. After initialization, leds should change every 600 ms in the sequence: RED, GREEN, and OFF now.

4.2.5. Modify the program to change the leds sequence every 1s.

5 PIN INTERRUPTS

Pin interrupts are provided by mechanism that is implemented in Kinetis L PORT module. The PORT module has the following features:

- Pin interrupt on selected pins,
- interrupt flag and enable registers for each pin
- support for edge sensitive (rising, falling, both) or level sensitive (low, high) configured per pin,
- pull-up and pull-down functionality.

All control bits that are responsible for pin functions are located in Port Control Register (PCR). Each pin has its own 32-bit PCR register to select its properties.

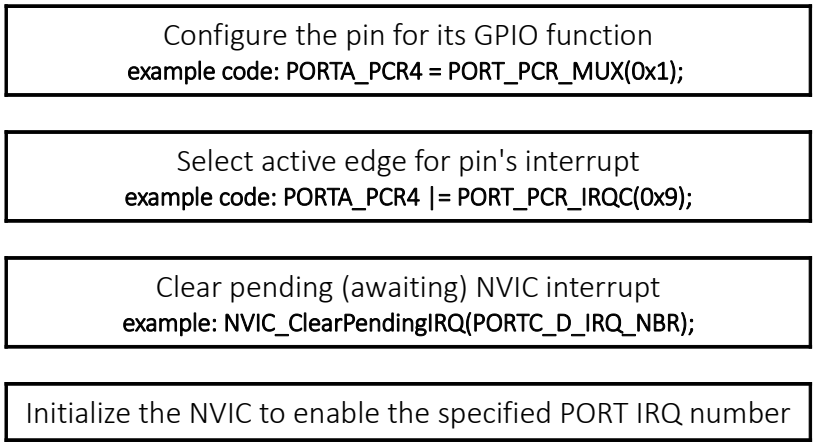
In FRDM-KL46Z, SW1 button are connected to pin3 of port C (PORTC[3]). You will use SW1 to generate PORTC_PORTD interrupt in your application. For function of individual bits of PCR see '*PORTx_PCRn field descriptions*' table in '*KL46 Sub-Family Reference Manual*'

5.0.1. In the table below, write what is the selected PCR's bit field functionality.

No.	Bit field name	Description
1	MUX	
2	PE	
3	PS	
4	IRQC	
5	ISF	

5.1 ENABLE PIN INTERRUPT

The following procedure must be done to enable a pin interrupt.



example code: NVIC_EnableIRQ(PORTC_D_IRQ_NBR);

5.1.1 Define a correct interrupt (IRQ) number for PORTC_PORTD. [ToDo 5.1] in buttons.c file

5.1.2 Put a proper name of PORTC_PORTD Interrupt service routine. [ToDo 5.2]

5.1.3 Make sure that interrupt service flag (ISF) in Port Control Register is cleared during ISR execution. This prevents CPU to be interrupted immediately after ISR is completed. [ToDo 5.3]

5.1.4 Set bit in PCR register to enable pull resistor. [ToDo 5.4]

5.1.5 Set bit in PCR register to select pull up. [ToDo 5.5]

5.1.6 Set proper value for IRQC bit field in PCR register to select a falling edge interrupts. [ToDo 5.6]

5.1.7 Call button initialization function in main.c. [ToDo 5.7]

5.1.7 Build and run the application. You should be able to control leds' sequence speed by pushing SW1 button.

6 Low Power Timer Interrupt

6.1 Low Power Timer

Now, it will be demonstrated how to use the low-power timer (LPTMR) to schedule leds' FSM changes. In this tutorial, LPO is used as LPTRM clock source. It will be used to implement a one second period timer.

6.1.1 Refer to '*KL46 Sub-Family Reference Manual*' to elaborate the 'LPO' acronym.

The low-power timer (LPTMR) can be configured to operate as a 16-bit time counter with prescaler. It provides a compare function. Additionally, LPTRM can generate interrupt on compare event when the value in the counter equals the value in special compare register. Furthermore, it can operate in low-leakage power mode. This feature allows to use LPTRM to wake up CPU periodically from low-power state to perform a scheduled task. The low-power wake-up cannot be performed using SysTick.

There are several registers devoted to manipulate LPTMR functions. Here, in this tutorial we will refer to Prescale Register (PSR), Control Status Register (CSR), and Compare Register (CMR).

6.1.1 Write the selected registers bit field name and its functionality in the table below.

LPTMR register	Bit field	Bit name and functionality
Prescale Register (LPTMR0_PSR)	PCS	
	PBYP	
Control Status Register (LPTMR0_CSR)	TFC	
	TMS	
	TCF	

	TIE	
	TEN	
LCompare Register (LPTMR0_CMR)		
Counter Register (LPTMR0_CNT)		

6.2 Configuration of LPTMR

The following procedure must be done to configure LPTRM.

Configure the clock source and the prescaler

example code:

```
LPTMR0_PSR = (LPTMR_PSR_PRESCALE(0) // 0000 is div 2
            | LPTMR_PSR_PBYP_MASK
            | LPTMR_PSR_PCS(clock_source)) ;
```

Configure LPTRM timer/counter functions

example code:

```
LPTMR0_CSR = (LPTMR_CSR_TCF_MASK // Clear any pending interrupt
            | LPTMR_CSR_TIE_MASK; // LPT interrupt enabled
```

Set LPTRM Compare Register compare field

example code:

```
LPTMR0_CMR = LPTMR_CMR_COMPARE(count); //Set compare value
```

Clear pending (awaiting) NVIC interrupt and initialize the NVIC to enable the specified LPTRM IRQ number

example code:

```
NVIC_ClearPendingIRQ(LPTRM_IRQ_NBR);
NVIC_EnableIRQ (LPTRM_IRQ_NBR);
```

Enable LPTRM counter

example code:

```
LPTMR0_CSR |= LPTMR_CSR_TEN_MASK;
```

6.1.1 Determine the NVIC's IRQ number for LPTRM timer. And put is as definition od symbol LPTMR_ALARM_INT_NBR [ToDo 6.1]

6.1.2 Set PBYP bit of PSR register to bypass prescaler and fed counter input directly with LPO clock [ToDo 6.2]

6.1.3 Set the compare field in CMR register to rise an interrupt every 400ms [ToDo 6.3]

6.1.4 Disable LPTMR to Timer Free-Running Counter mode by clearing bit TFC in CSR register. Thus CNT will be cleared when it is equal CMR [ToDo 6.4]

6.1.5 Clear Timer Mode Select bit in CSR register to select timer mode [ToDo 6.5]

6.1.6 Set TCF bit in CSR to clear pending interrupt [ToDo 6.6]

6.1.7 Set TIE bit in CSR to enable LPTMR0 interrupt [ToDo 6.7]

6.1.8 Enable TEN bit in CSR to start counting [ToDo 6.8]

6.1.9 Determine a proper ISR name for LPTRM0 and rename the routine in lptmr.c [ToDo 6.9]

6.1.10 Disable SysTick interrupt, and initialize LPTMR in main.c file. [ToDo 6.10]

6.1.11 Build the application. You should experience the same functionality as in exercise before except LPTMR is now in use (which is obviously not distinguishable).

7 INTERRUPT PRIORITIES

In the Cortex-M0 processor, each exception has a priority level. The priority level affects whether the exception will be carried out or if it will wait until later (stay in a pending state). The Cortex-M0 processor supports four programmable priority levels for peripheral devices i.e.: 0, 1, 2, and 3 priority level. The lower the priority number is the higher the interrupt priority (0 priority is the highest), Additionally, Reset, Non-Maskable Interrupt (NMI) and Hard-Fault exceptions have the extra priority levels (-3, -2, -1) respectively. These are non-programmable priorities and they have the highest privilege in the system.

7.1 INTERUPTS SEQUENCE

If an exception event occurs while no other exception handler is running, then the processor will accept it and the exception handler will be executed. The process of switching from a current running task to an exception handler is called preemption.

If the processor is already running another exception handler but the new exception has higher priority level than the current level, then preemption will also take place. The running exception handler will be suspended, and the new exception handler will be executed. This is commonly known as nested interrupt or nested exception. After the new exception handler is completed, the previous exception handler can resume execution and return to thread when it is completed.

However, if the processor is already running another exception handler that has the same or a higher priority level, the new exception will have to wait by entering a pending state. A pending exception can wait until the current exception level changes (Figure 3).

Figure 3: Pre-emption scenarios

7.1.1 Modify ISR to allow the CPU to stay in PORTC_PORTD interrupt routine as long as SW1 button is hold down [ToDo 7.1]

7.1.2 Set priorities of PORTC_PORTD interrupt and LPTMR interrupt to the lowest priority level 3 [ToDo 7.2a] [ToDo 7.2b]

7.2.1 Build and run the application. Explain the leds behaviour when SW1 is hold down.

Leds stop to change because .....

7.2.2 Increase LPTMR interrupt priority level to 2. [ToDo 7.3] Again, build and run the application. Now, explain the different leds behaviour when SW1 is hold down.

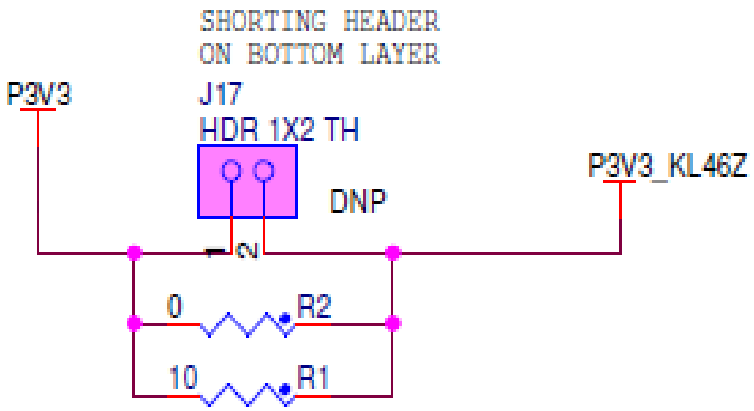
Leds continue to change because .....

8 Power conservation modes

8.1 Power measurements

The real-time current consumption is available if you measure the voltage drop across a resistor that is in series with the VDD of the microcontroller.

Power consumption of KL46Z can be performed using J17 header on FRDM-KL46Z.



Make sure that R2 resistor is removed and J17 shorting is cut on your FRDM-KL46Z kit.

8.1.1 Connect multimeter/oscilloscope to J17 header and measure current consumption of MCU.

Note: Make sure to perform a differential measurement when you use an oscilloscope.

8.1.2 Enable LLS mode in main.c. When build and download the new application. [ToDo 8.1]

Note: The MCU stops responding CMSIS-DAP Debugger when it is in LSS mode. To download a new version of the application when you run the program with LSS enabled, it is necessary to do it before MCU enters the while loop after a reset signal.
To download another application, press the reset on FRDM-KL46Z and click the download button in uVision very shortly afterwards.
Alternatively you can also download factory firmware to restore your FRDM-KL46Z functionality (See PRE-Lab tutorial how to do it)

8.1.3 Repeat current measurements and compare them to the previously get results. Fill the table below.

Table. Current consumption of MCU

	Current [mA]
Run mode	
LSS mode	

9 The Student Group Individual Task

Use interrupt-driven programming style and extend the tutorial application functionality by enabling the SW3 button of FRDM-KL46Z. Note that SW2 should still control the speed of the led sequence.

Hint: Test ISF bit in PCR registers to distinguish SW1 and SW3 buttons actions.

One task per group.

Group 1	Programme the SW3 button to change the running sequence of led blikning. Use two different sequences. For example: seq_1=(RED_ON, GREEN_ON, BOTH_OFF, BOTH_ON), seq_2=(RED_ON, BOTH_ON, GREEN_ON, BOTH_OFF)
Group 2	Programme the SW3 button to start/stop the running sequence of led blinking.
Gropu 3	Programme the SW3 button to reset the running sequence of led blinking. After the SW3 is pressed the sequence comes to the initial led state.
Group 4	Programme the SW3 button to freeze the running sequence of led blinking.
Group 5	Programme the SW3 button to reverse the running sequence of led blinking. After the SW3 is pressed the sequence starts to go in the opposite direction.
Group 6	Programme the SW3 button to implement the postponed stop of the running led sequence. After the SW3 is pressed the sequence continues to the initial state and stops. It starts again then the SW3 is pressed again.
Group 7	Programme the SW3 button to toggle the led sequence between the *normal, stop, and reverse* operation modes.