Scheduler Tutorial

This tutorial demonstrates how the scheduler can be used to transfer execution from the interrupt context to the main context.

app_timer scheduler

2016-04-08: Updated tutorial to cover SDK version 11.

Introduction

Scope

The following topics will be included in this tutorial:

- Configuration of the Scheduler library
- Using the scheduler with any interrupts or events
- Using the scheduler with the application timer

Necessary prior knowledge

It is expected that you have basic knowledge of how to use Keil in order to build and download your application to your kit. Refer to Getting Started in the SDK documentation if this is unfamiliar territory. It is also expected that you are familiar of the concept of interrupts and event handlers. Please refer to the Pin Change Interrupt Example if you need to refresh this topic.

The section on using the scheduler with the application timer require some familiarity with the application timer. Refer to the Application Timer tutorial if required.

Necessary equipment and software

Hardware:

- nRF51 DK or nRF52 DK.

Software:

- Keil 5 or alternatively ARM GCC (instructions will assume that Keil is used).
- nRFgo Studio or nrfjprog.

Other files:

- nRF5 SDK version 11
- Example project: "scheduler_tutorial.zip"

Within the tutorial project folder there are sub-folders for different boards:

- pca10028 project is to be used with the nRF51 DK.
- pca10040 project is to be used with the nRF52 DK.

Preparations

To get started, download the example code. It configures General purpose input/output (GPIO) pins for the buttons and LED's that will be used in the tutorial. To build it, extract the compressed project to <SDK>\examples\peripheral. The example is intended for the zipped version of the SDK. It will not work with the Pack installer. If you need help with this please have a look at this thread on DevZone: Compiling github projects. Open the project file (.uvprojx) and hit *Build*. The application should compile without any errors or warnings.

Erase the flash of the DK using nRFgo Studio or nrfjprog in order to ensure that no SoftDevice is installed. You only have to do this once. The application can be downloaded to the DK from Keil by pressing *Download*. LED 1 should start to toggle when you click button 1 and stop toggling when you click button 2. The other LED's should remain dark.

The Scheduler

The scheduler is used for transferring execution from the interrupt context to the main context (your main loop). This ensures that all interrupt handlers are short, and that all interrupts are processed as quickly as possible. This can be very beneficial for some applications.

Conceptually the scheduler works by using a event queue. Events are scheduled by being put in the queue using `app_sched_event_put()`. This will typically be done from a event handler running in a interrupt context. By calling `app_sched_execute()`, events will be executed and popped from the queue one by one until the queue is empty. This will typically be done in the main loop. There is no concept of priorities, so events are always executed in the order they were put in the queue.

The Schedule handling library documentation has a good overview of how the scheduler works, and you should spend a minute to compare the sequence diagrams for applications using the Scheduler and applications not using the scheduler. The Scheduler API is simple and you are recommended to use a few minutes to review it.

Some of the libraries in the SDK, such as the Application Timer, include support for the Scheduler. Others, such as the GPIO tasks and events (GPIOTE) driver or your own interrupt routines does not, and you will have to do a bit more manual work. In this tutorial you will initialize the Scheduler, and use it both manually with GPIOTE and with the application timer.

Visualizing the function of the scheduler

In this tutorial you will not change the functionality of the example, which is to control toggling of LED 1 using button 1 and 2. You will only change how the event handlers are executed, and we will use LED 2 and 3 to visualize this.

Both `timer_handler()` and `button_handler()` in the example project contain a section of code that use LED 2 and 3 respectively to show if the code is running in main or interrupt context. The LED is turned on when running in main and turned off when running in a interrupt context. This code has no practical use in a real application, but it is useful here in the tutorial as a visual indication that the scheduler is doing it's job.

Note that LED 2 and 3 always stays turned off as you press button 1 and 2 at this point, indicating that both the `timer_handler()` and `button_handler()` functions are executed in a interrupt context.

Add required files and includes

In order to use the scheduler you will have to add a few files and includes to your project. Right click on the nRF_Drivers group in the project window. Click *Add existing files to group 'nRF_Drivers'* and add:

- ..\..\..\..\..\components\libraries\scheduler\app_scheduler.c
- ..\..\..\..\..\components\libraries\util\app_util_platform.c

Then go to Project -> Options for target ... -> C/C++. From there, add the following directory to the Include Paths:

- ..\..\..\..\..\components\libraries\scheduler

Then include the required header files by adding the following lines below the existing include statements:

```
#include "app_scheduler.h"
#include "nordic_common.h"
```

Initialization

The scheduler is initialized using the `APP_SCHED_INIT()` macro, which takes two parameters:

- `EVENT_SIZE`: the maximum size of events to be passed through the scheduler.

- `QUEUE_SIZE`: the maximum number of entries in scheduler queue.

Add the following defined close to the top of the file:

```
// Scheduler settings
#define SCHED_MAX_EVENT_DATA_SIZE       sizeof(nrf_drv_gpiote_pin_t)
#define SCHED_QUEUE_SIZE                10
```

Then write the following line above your main loop in order to initialize the Scheduler using the above parameters:

```
    APP_SCHED_INIT(SCHED_MAX_EVENT_DATA_SIZE, SCHED_QUEUE_SIZE);
```

The last thing you have to do is to add a call to `app_sched_execute()` in your main loop. The `app_sched_execute()` function will pull events and call its handler in the main context. It will execute all events scheduled since the last time it was called. Then the CPU would normally go to sleep again, as is the case in this tutorial with the call to `__WFI()` (or `sd_app_evt_wait()` if a SoftDevice was used). Add the following line to your main loop:

```
        app_sched_execute();
```

Using the scheduler with any interrupts or events

The scheduler is supported by some components in the SDK, including the application timer and the SoftDevice. It can also easily be used with your own interrupt routines or event handlers. The GPIOTE driver does not use the scheduler out of the box, so any GPIOTE event handlers will run in a interrupt context.

In order to use the scheduler you will have to create a new button scheduler event handler that will run from the main context (called from `app_sched_execute()`). Let's start by creating that handler, the `button_scheduler_event_handler()`. Add the following function to your file somewhere after `button_handler()` and before `gpiote_event_handler()`:

```
// Button handler function to be called by the scheduler.
void button_scheduler_event_handler(void *p_event_data, uint16_t event_size)
{
    // In this case, p_event_data is a pointer to a nrf_drv_gpiote_pin_t that represents
    // the pin number of the button pressed. The size is constant, so it is ignored.
    button_handler(*((nrf_drv_gpiote_pin_t*)p_event_data));
}
```

In the existing handler, `gpiote_event_handler()`, that runs in the interrupt context you will do as little as possible, and just schedule an event by putting it in to the schedulers event queue using `app_sched_event_put()`. In order to achieve this, modify `gpiote_event_handler()` by removing this line:

```
    button_handler(pin);
```

...replacing it with this:

```
    app_sched_event_put(&pin, sizeof(pin), button_scheduler_event_handler);
```

With changes you have made, the `gpiote_event_handler()` still runs in an interrupt context, but it is short (thus takes very little time) and only schedules the `button_scheduler_event_handler()` by putting an event into the schedulers event queue. The main work is done by the `button_scheduler_event_handler()` which is called by `app_sched_execute()` from the main loop.

Build and download the application to your target. When you now press button 1 or 2 you will see LED 2 being lit, which indicate that the code is now running in main context.

Using the scheduler with the Application Timer

Many applications use the Application timer. The following section will demonstrate how to use the scheduler with the application timer so that the timeout handlers are run in main context.

You will have to add a extra file to your project. This file contains two small functions that make the application use the Scheduler to run the timeout handlers. Right click on the nRF_Drivers group in the project window. Click *Add existing files to group 'nRF_Drivers'* and add:

- ..\..\..\..\..\components\libraries\timer\app_timer_appsh.c

Then include the corresponding header file by putting this include statement together with the other includes close to the beginning of the file:

```
#include "app_timer_appsh.h"
```

The application timer must be configured to use the scheduler. This is done by using the `APP_TIMER_APPSH_INIT()` macro to initialize the application timer. In order to do this, replace the the following line from your main function:

```
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_OP_QUEUE_SIZE, false);
```

...with this line:

```
    APP_TIMER_APPSH_INIT(APP_TIMER_PRESCALER, APP_TIMER_OP_QUEUE_SIZE, true);
```

The scheduler event queue must be made to hold the maximum size of the data types that can be used as event data. Therefore, we will update it in the proper way. You should remove the following line:

```
#define SCHED_MAX_EVENT_DATA_SIZE       sizeof(nrf_drv_gpiote_pin_t)
```

...and replace it with this line:

```
#define SCHED_MAX_EVENT_DATA_SIZE       MAX(APP_TIMER_SCHED_EVT_SIZE, sizeof(nrf_drv_gpiote_pin_t))
```

Again, build and download the application to your target. When you press button 1 LED 1 starts toggling as before, but LED 3 is lit, indicating that the `button_handler()` is called from the main context.

Further Reading

- See the Application timer tutorial for details on how to use the application timer.
- See the Schedule handling library section in the SDK documentation.
- See the Scheduler API Reference for a complete overview of the available API.