Exercise 3

In this exercise we study resource efficient design methodologies by focusing on power management. We write a simple program using different approaches and study their effect on power usage. The most important thing to realize when designing embedded software as well as hardware is that the resources are far more limited than in a conventional PC environment. For example by reducing a system's power usage, it might win its competitors on the market and thus justify the extra design effort.

Hardware setup

For measuring the current drawn by the processor we need to modify our nRF52 board. **This will be done only for couple of boards** which will be used to current measurement. In the schematic sheet of the boards power supply, we see that SB9 needs to be cut in order to measure the current using a current meter. Some of the boards might already have been prepared. Current can also be measured soldering 10 ohm shunt resistor to R6 and measuring the voltage drop using a two-channel oscilloscope.

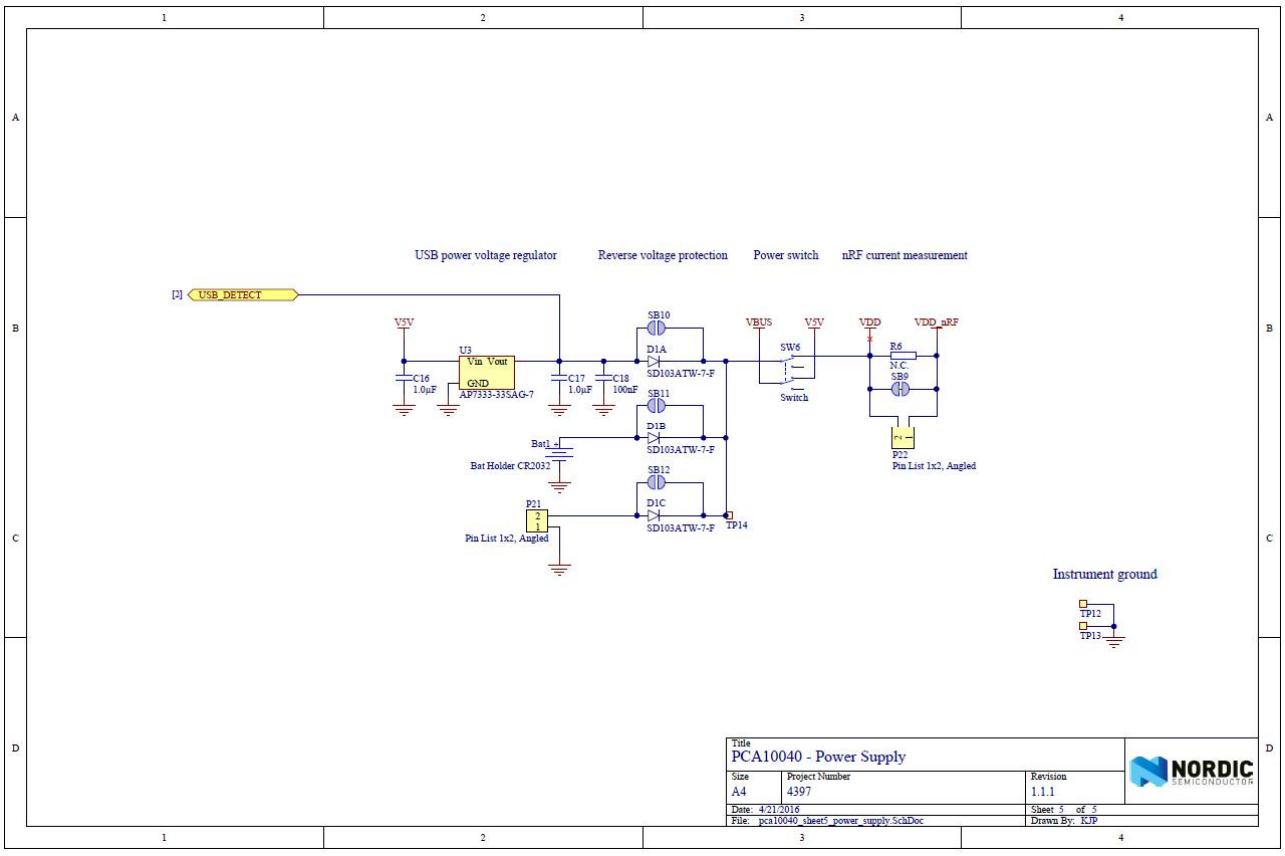

Figure 1: Schematic diagram of the DK's power supply

Simple led blink

Here we write a 'dumb' led blink program that uses *nrf_delay_ms* function to wait between the GPIO accesses. We use nRF's *Board Support Package (BSP)* to access the leds which is much simpler than using the GPIO module by itself.

```
/**
 * A simple function for blinking the led
 */
void blink()
{
    bsp_board_led_invert(0); // blink led1
    nrf_delay_ms(500); // do a busy wait
}
```

Now try to call this in your main function and run the code. Notice the led blink at 1 Hz. The measurement should now show rather a large current draw in the system.

Led blink with timer

The next thing to do is to integrate the timer we created in the previous exercise to the project. So now the blink function should act as a timeout handler for our timer. Adjust the timer to go off every 500 ms. Leave the while loop in your main function empty and let the interrupt context do the work.

Now measure the current. Has there been any significant drop between this and the previous code?

Power saving

In the previous example we used a timer to decide when to light up the led, but the processor is still busy 100% of the time. Next we will let it go to sleep mode when no processing is needed. To accomplish this we use ARM's sleep instruction __WFE() which stands for Wait For Event. This command puts the processor into a sleep mode where it shuts down some of its systems. When an event or an interrupt is detected, it will wake up and run the interrupt/event handler. Insert the following to your main function and observe how the current draw changes. We call __SEV() in case there has already been an event flag set and we missed it.

```
while(1)
{
    __WFE(); // sleep until an event wakes us up

    __SEV(); // set the event register

    __WFE(); // clear it
}
```

Program size

By following good programming practices like avoiding global variables and using pointers to pass function parameters you can optimize the code by yourself. The C compiler also does a lot of optimization for the code before turning it into machine code, e.g. loop unrolling and deleting unused variables. As we learned in exercise 1, we can tell the compiler how much should be optimized. –O0 optimization level does the least optimization but still some. –O3 is the maximum optimization level and reduces the size significantly. You can use the *Options for Target* tab to control the optimization or by using preprocessor commands e.g. `#pragma O3` and use different levels for different parts of the program.

Try to change the optimization level and see how it affects the program size. Open the .map file from the _build directory which can be found from the same directory as the project file. Find the *Read-only* and *Read/Write* sizes of our code. Also try to locate the program's stack address and size. It's important to know how large your stack is, so that there will be no overflows.