ITM printing and Timing jitter

In this exercise you will use ITM (Instrumentation Trace Macrocell) to print debug messages to your PC.

MCUXpresso supports virtual consoles using semihosting. Unfortunately semihosting printing requires CPU to be stopped during printing, takes a lot of time and if a debugger is not connected the program hangs. Therefore semihosting can't be used in an application that runs without a debugger connected.

ITM on the contrary is integrated into the CPU. ITM is controlled by the software that runs on the device but it does not require a debugger to be present. ITM is available on ARM Cortex-M3 and M4 processor families. LPCXpresso supports displaying ITM trace information.

First some notes about enabling ITM and then further information about the assignment and implementation requirements.

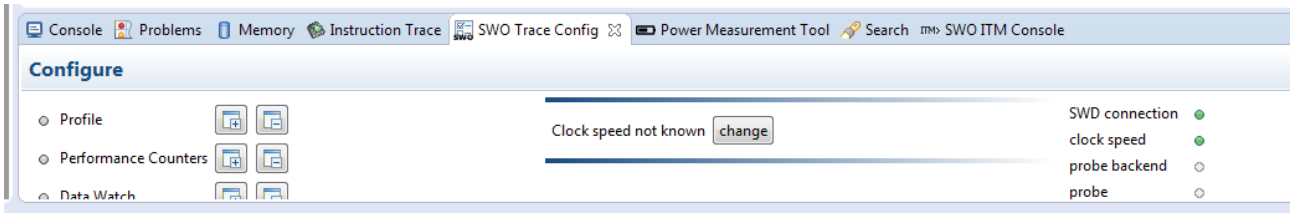
To enable ITM printouts you need to do the following things:

1. Add `ITM_write.c` and `ITM_write.h` to your project (see course workspace Documents/Sources)
2. Include ITM_write.h in each file that calls ITM functions
3. Call ITM_init() in your main after other hardware initialization routines have been called.
4. Call ITM_write() to write to the ITM console on the debugger. Note that the function prints C-style strings (null terminated) not C++ string objects. You can safely add the files in to a C++ project. Both code and header have been written to support C and C++.

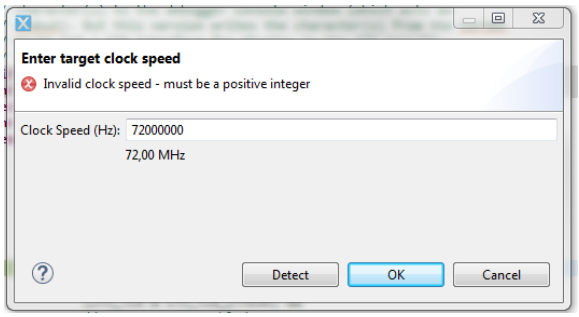
When your program is ready to run you need to add ITM console into debug view to see the printouts. First thing to do is to configure is SWO clock rate. The clock rate can be automatically detected after the clock has been enabled. You must call ITM_init() to enable ITM!

The simplest way to setup clock rate is to start the application which enables the clock and then start clock detection. Enable detection when your program is running.

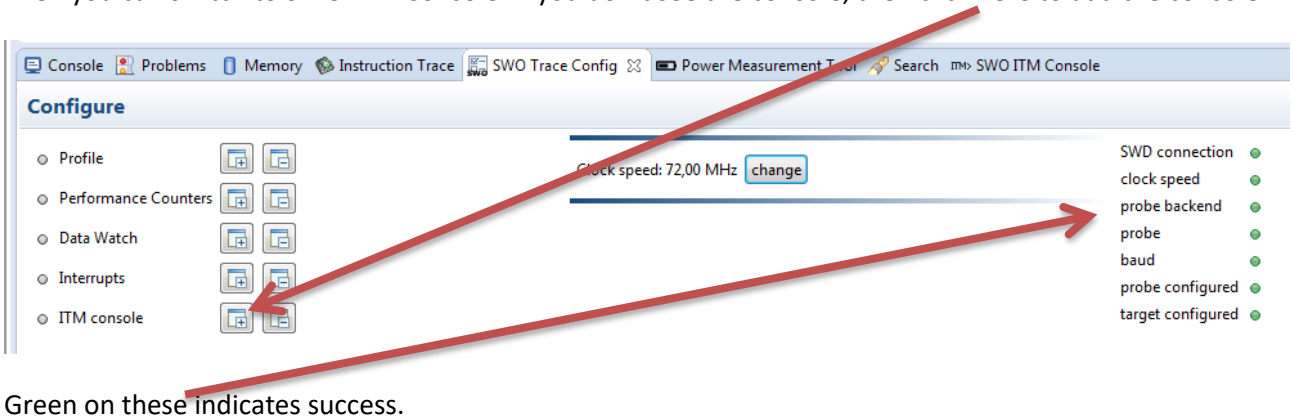
In the debug consoles window go to **SWO Trace Config** and click on **change**.



You will see the following window. Click **Detect** and clock speed should be set 72 MHz. If the clock speed is not detected then check that you have called ITM_init after the chip and board initializations. Press OK to close the window.

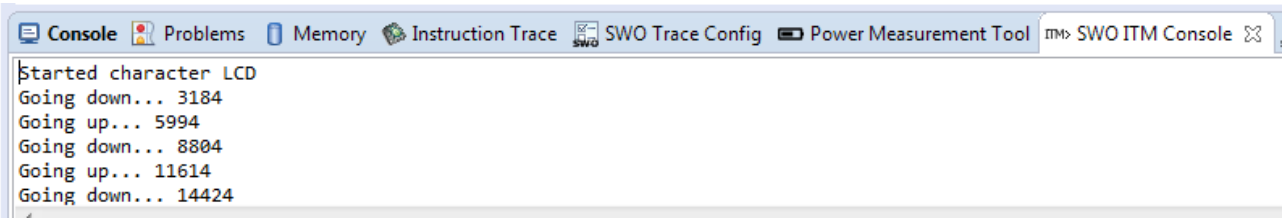


Then you can switch to SWO ITM Console. If you don't see the console, then click here to add the console.



Green on these indicates success.

Now when you switch to SWO ITM console you should see your printouts. **What you see depends of course on what you print. You will only see what you print with ITM-print.**



Hint: First write a simple test program that enables ITM and prints something (for example "Hello World\n") on the ITM console. When that works the you can move on to the exercises. Remember that output is buffered so characters are displayed only after you print a newline character ("\n").

Exercise 1

Write a program that measures the length of the button presses with 10 ms granularity and prints the lengths on the ITM console. You will also need to implement an ITM output class to make printing to ITM-console a bit easier. Use the class from exercise 1 and configure all buttons as inverting.

ITM_write uses C-style strings. We can't use iostream library due to limited amount of flash but we can **implement a wrapper class that initializes ITM in the constructor and provides overloaded functions for printing C and C++-style strings and integers.** The public member function of your wrapper class must be called **print**. The constructor must handle the initialization of ITM hardware.

To convert integers (or any numeric values) to string you can use std::to_string() or snprintf. Include <cstdio> to use snprintf. See the following link for help on using snprintf.

http://www.cplusplus.com/reference/cstdio/snprintf/

The program prints for example:

```
SW1 pressed for 730 ms
SW1 pressed for 930 ms
SW2 pressed for 1730 ms
SW3 pressed for 1200 ms
```

Hint: Use the Sleep function from Lab1 to generate accurate delays.

Exercise 2

Write a program that implements a menu where user can control the on-board leds with buttons. The menu is printed to ITM console. Use the object from exercise 2.

Use your IO-pin object from exercise 1 to control the on-board leds and to read buttons. Improve the IO-pin class so that inverter can be enabled also when a pin is configured as on output. Note that there is no hardware inverter for outputs so you need to do the inverting in the software (invert the value before writing to the pin). Configure the leds so that when you write “true” the led goes on and when you write “false” the led goes off.

SW1 and SW3 move the arrow in the menu and SW2 switches the led that the arrow points on or off (toggles the led). The current status must also be printed.

Example:

```
Select led:
Red   OFF <--
Green OFF
Blue  OFF
```
<SW2 pressed, red toggles, menu is reprinted>

```
Select led:
Red   ON  <--
Green OFF
Blue  OFF
```
<SW3 pressed, “cursor” moves, menu is reprinted>>

```
Select led:
Red   ON
Green OFF <--
Blue  OFF
```
<SW2 pressed, green toggles, menu is reprinted>

```
Select led:
Red   ON
Green ON <--
Blue  OFF
```

Exercise 3

1. Find the addresses of CoreDebug and DWT (see below).
2. Write a program that measures timing jitter of Sleep() function.

LPC1549 has a built-in high-resolution timer that can be used to measure timing with clock cycle accuracy. To start the timer, you need to execute the following statements:

```
CoreDebug->DEMCR |= 1 << 24;
DWT->CTRL |= 1;
```

To get the current counter value you can read the counter register:

```
value = DWT->CYCCNT
```

What to do:

1. Read button state.
2. If not pressed go back to 1.
3. Read the counter value.
4. Execute Sleep(1).
5. After the sleep ends immediately read the counter again and record the difference in an array.
6. Wait until the button is released
7. Sleep for 50 ms
8. If 50 samples collected then stop else repeat from step 1

Collect 50 samples in the array - you need to press the button 50 times! Sort the array, print the array using ITM and finally print average sleep time both in counter cycles and in microseconds.

You need to copy the result of your test to a document that you attach to your answer. The document must be in **pdf-format**.