ITM printing and IO-pin class

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

LPCXpresso 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.

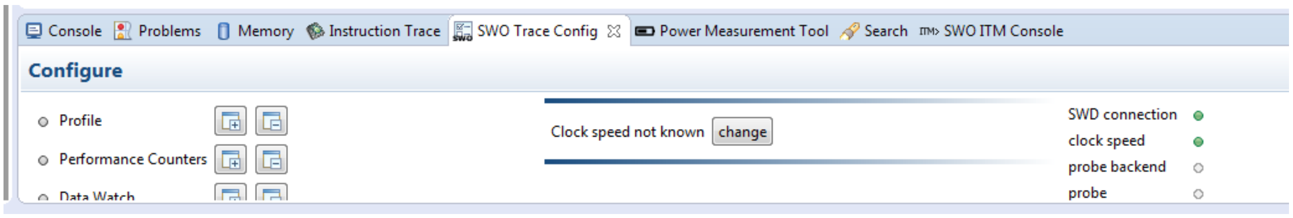
First some notes about enabling ITM and then further information about the assignment and implementation requirements.

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

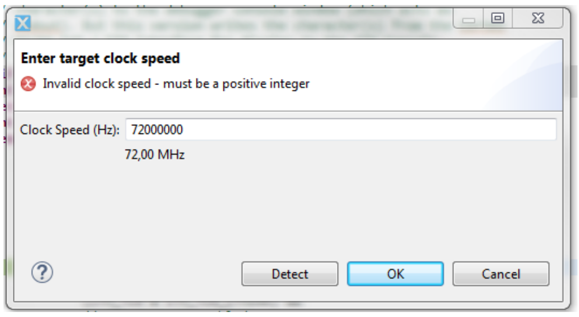
1. Add `ITM_write.c` and `ITM_write.h` to your project (see course workspace Documents/Sources)
2. Include ITM_write.h in each file that calls ITM functions
3. Call ITM_init() in your main after other hardware initialization routines have been called.
4. Call ITM_write() to write to the ITM console on 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 codes 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. **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.



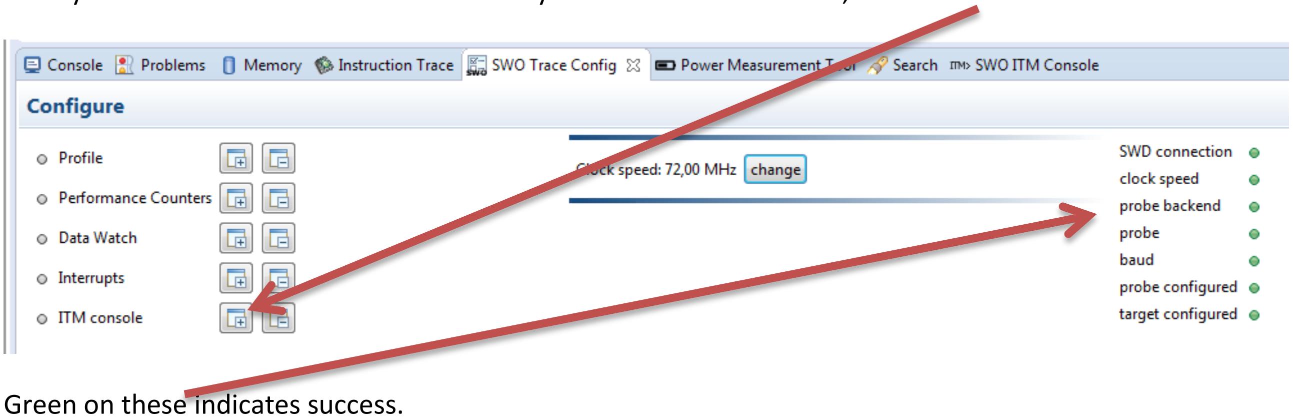
Enter target clock speed

Invalid clock speed - must be a positive integer

Clock Speed (Hz): 72000000

72,00 MHz

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

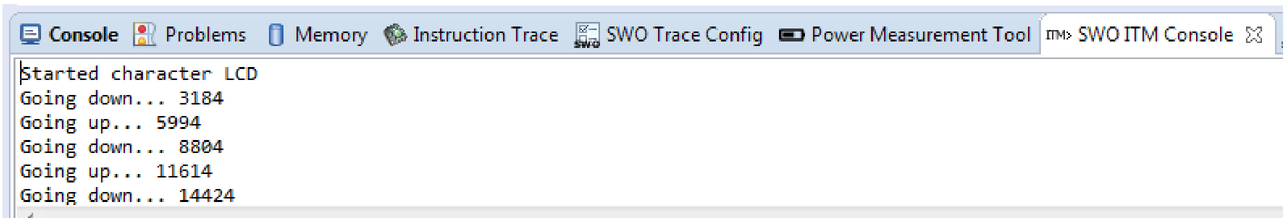


Configure

Clock speed: 72,00 MHz

Green on these indicates success.

Now when you switch to SWO ITM console you should see your printouts.



```
Started character LCD
Going down... 3184
Going up... 5994
Going down... 8804
Going up... 11614
Going down... 14424
```

Hint: First write a simple test program that enables ITM and prints something (for example "Hello World\n") on the screen. 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").

A simple delay function

Systick timer and a function that generates a fairly accurate delay. Systick timer is assumed to run at 1000 Hz (once per millisecond). See periph_blinky project for example on configuring the systick timer.

You can add this to your project if you need to generate delays.

```
static volatile int counter;

#ifdef __cplusplus
extern "C" {
#endif
/**
 * @brief   Handle interrupt from SysTick timer
 * @return  Nothing
 */
void SysTick_Handler(void)
{
	if(counter > 0) counter--;
}
#ifdef __cplusplus
}
#endif

void Sleep(int ms)
{
	counter = ms;
	while(counter > 0) {
		__WFI();
	}
}
```

Exercise 1

Implement a class that abstracts DigitalIoPins.

The class must have the following interface:

```
class DigitalIoPin {
public:
    DigitalIoPin(int pin, int port, bool input = true, bool pullup = true, bool invert = false);
    virtual ~DigitalIoPin();
    bool read();
    void write(bool value);
private:
    // add these as needed
};
```

Write a program that creates three DigitalIoPin objects, one for each on-board button. The buttons are connected to the following pins:

- SW1 → PIO 0.17
- SW2 → PIO 1.11
- SW3 → PIO 1.9

The program monitors the state of the pins and when SW1 is pressed program prints “SW1” on the ITM console. Implement similar printouts for all three buttons (SW2 pressed prints “SW2” etc.).

All three switches require that pullups are enabled. With pullups and the way the buttons are connected the button reads zero when button is pressed and one when not pressed. This can be changed by enabling the inverter on the IO-pins.

Create the objects as variables (do not use new!). For example:

```
DigitalIoPin sw1(0,17,true, true, false);
```

Exercise 2

Write a program that measures the length of the button presses with 10 ms granularity and prints the lengths on the ITM console.

Hint: Use the Sleep function above to generate accurate delays.

Since ITM_write uses C-style strings and we can't use iostream library due to limited amount of flash and RAM we need to use C library functions. Include <cstdio> to use snprint. 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
```

Exercise 3

Write a program that implements a menu where user can control the on-board leds with buttons.

SW1 and SW3 switch between leds and SW2 switches the led 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
```