Queues

In the following exercises register all queues, semaphores and mutexes to the queue registry.

Note that debug serial port must be protected against simultaneous access (printf, DEBUGP, and BoardUART-functions all use the same UART)!

Exercise 1

Write a program with three tasks and a queue of 5 integers:

Task 1 reads lines from debug serial port and counts the number of characters on the line up to but not including '\n' or '\r' at the end line. Task then sends the number of characters to the back of the queue.

Task 2 monitors SW1(PIO 0.17) and when button is pressed sends -1 to the back the queue

Task 3 waits on the queue and calculates the sum of integers received form the queue. When -1 is received the task prints: "You have typed %d characters" where %d is the number of characters. When task has printed the total it clears the total.

Exercise 2

Write a program with three tasks and a queue of 20 integers:

Task 1 sends random numbers to the back of the queue with an interval that randomly varies between 100 and 500 ms.

Task 2 monitors SW1(PIO 0.17) and when button is pressed sends 112 to the **front** of the queue.

Task 3 waits on the queue and prints the number it receives. If number is 112 the task prints "Help me" after the number. Then task sleeps for 300 ms and goes to wait for more data from the queue. Task must sleep only when 112 is received. Sleeping simulates a case where data processing takes longer than average.

Exercise 3

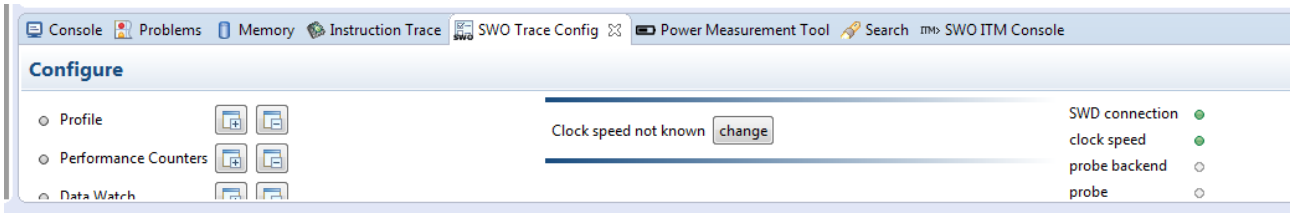
Write a program that implements debug printing task that uses ITM macrocell for printing debug data to the debugger console window. First some notes about enabling ITM and then further information about the assignment and implementation requirements.

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

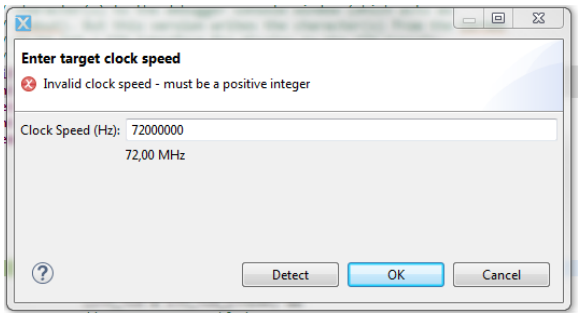
1. Add `ITM_write.c` and `ITM_write.h` to your project
2. Include ITM_write.h in each file that calls ITM functions
3. Call ITM_init() in your main after other hardware initialization routines have been called.
4. Call ITM_write() to write to the ITM console on the debugger. Note that the function prints C-style strings (nul terminated) not C++ string objects.

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.

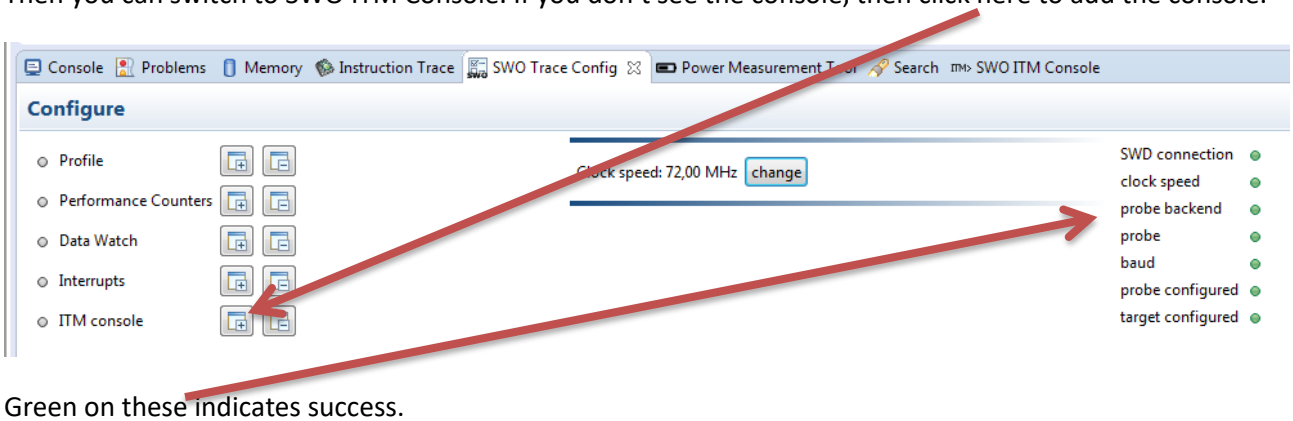
In the debug consoles window go to **SWO Trace Config** and click on **change**.



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 SWO setup is in proper place. Press OK to close the window.

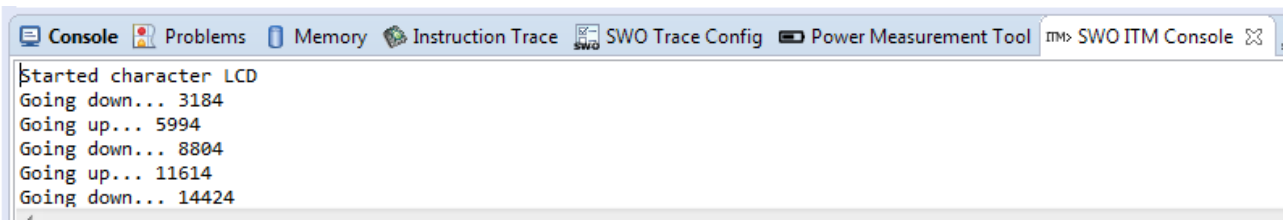


Then you can switch to SWO ITM Console. If you don't see the console, then click here to add the console.



Green on these indicates success.

Now when you switch to SWO ITM console you should see your printouts. The actual text that you see depends naturally on what you are printing.



Hint: First write a simple test program that enables ITM and prints something with ITM_write. When that works move on to the actual implementation.

Implementation

Write a program that has (at least) three tasks: debug print task with priority idle+1, two other tasks with priority idle+2. Debug print task must handle all ITM debug console printing. Other tasks communicate the text to print by sending data to debug queue.

One high priority task reads the debug serial port and when a word is received it prints the number characters in the word plus a time stamp to ITM debug console. The received characters must be echoed back to the debug serial port. End of word is determined by white space (space, tab, newline, cr, etc.). Any white space at the beginning of the word must be ignored and not counted to the word length.

The other high priority task monitors SW1 and when SW1 is pressed the task measures the length of press and prints it to ITM debug console.

Debug print task waits on a queue. Queue items contain **a constant string** and three numbers. The string must be a literal or other type of string that does not change at run time. The string is a format string that is accepted by printf/sprinf. When debug() is called (see examples below) it constructs a debug event and sends it to queue. For example we could call:

```
debug("Received cmd: %c at %d\n", rcv[0], xTaskGetTickCount(), 0);
```

Note that since the time stamp is stored in the event it does not matter if the line is printed much later. Since queue maintains ordering of the items the debug messages will print out in exactly the same order as they were send to queue.

Note that the implementation below is far from complete. It is provided as an example of partial solution. Feel free to use this or create your own implementation.

```
void debug(const char *format, uint32_t d1, uint32_t d2, uint32_t d3);


struct debugEvent {
    const char *format;
    uint32_t data[3];
};



void debugTask(void *pvParameters)
{
    char buffer[64];
    debugEvent e;

    // this is not complete! how do we know which queue to wait on?

    while (1) {
        // read queue
        xQueueReceive(syslog_q, &e, portMAX_DELAY);
        snprintf(buffer, 64, e.format, e.data[0], e.data[1], e.data[2]);
        ITM_write(buffer);
    }
}
```