

## 2IN70 - Track 2: Real-Time Architectures

### Practical Training - Exercises 3 -

Today we will gain experience with running programs on the Freescale EVB9S12XF512E board and the  $\mu$ C/OS-II real-time operating system.

#### 3.1 Running a program on the board

In this exercise you will run your project from last week's Exercise 2.2 on the physical board. *It is important that you follow these steps in the given order, to prevent damaging the board!*

1. Make sure the switch SW4 is in **STAND ALONE** mode.
2. Connect the light sensor to the board, as indicated in Figure 1. Make sure to attach:
  - GND (black) wire on the sensor to ground on pin TP1 on the board
  - +5V (red) wire on the sensor to +5V on pin TP5 on the board
  - AIN (white) wire on the sensor to pin PAD14 on the board

*Double-check with your partner that the wires are correctly connected, to prevent destroying the board!*

3. Connect the programmer to the board **making sure that the cable is facing outwards**.
4. Connect the programmer to your computer via a USB port.
5. Connect the board to the power.
6. In the CodeWarrior window select the the **P&E USB BDB Multilink** target.
7. Build and run the project. You will notice that a very similar debugger window appears to when you ran the program in the simulator.
8. Start the program.
9. Observe the leds to verify that your program works.

If you are experiencing problems with your code, or if you want to verify that it is working correctly, you can run the project in the **exercise2.2.solution** directory.

#### Important

- I. Do not disconnect the board from power while the programmer is updating the code on the board!
- II. Always disconnect the board from power before connecting or disconnecting the light sensor!
- III. Always disconnect the board from power before connecting or disconnecting the USB connector!
- IV. Do not touch the components on the board!
- V. When disconnecting the board, make sure to follow steps 2 to 5 in reverse order!

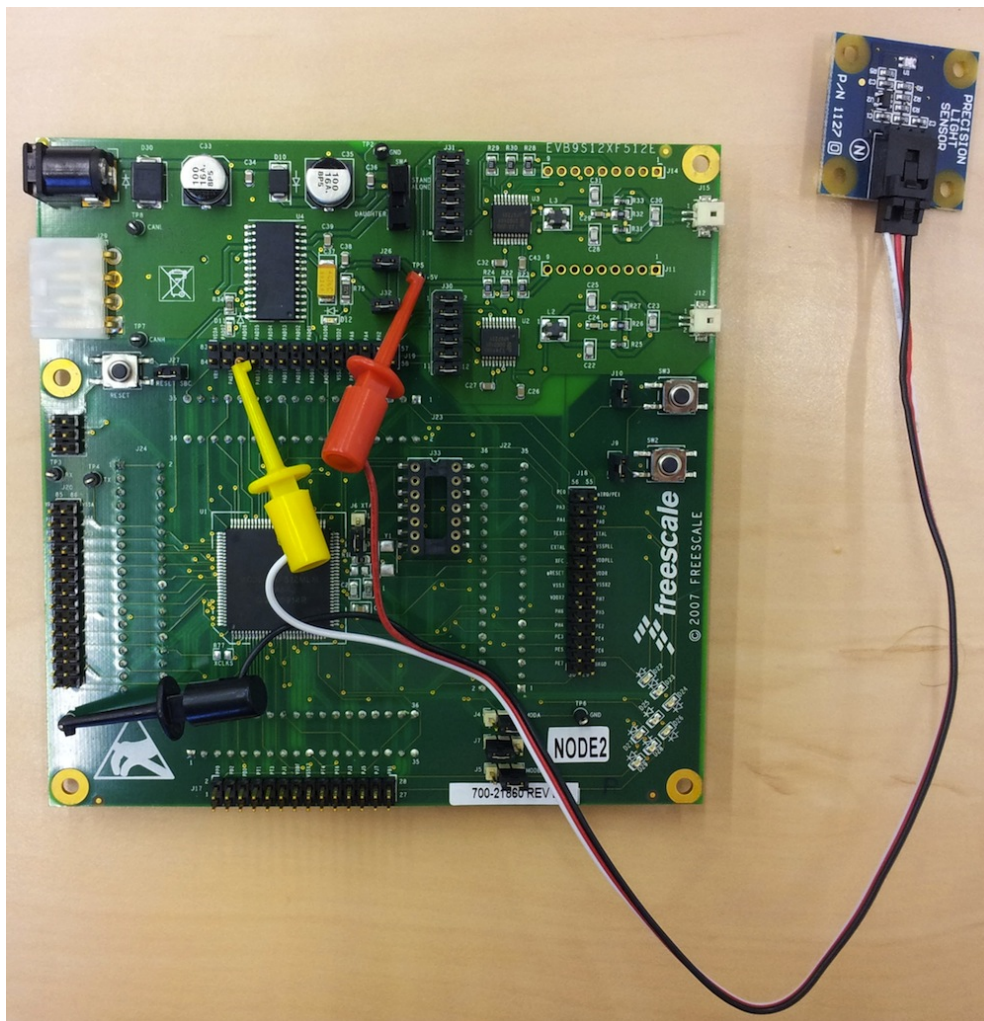


Figure 1: Connecting the light sensor to the EVB9S12XF512E board.

### 3.2 Dealing with preemption

In this exercise you will get acquainted with the  $\mu$ C/OS-II real-time operating system. We will be using  $\mu$ C/OS-II more often during the upcoming weeks. The CodeWarrior project for this exercise is in the directory `exercise3.2`.

1. You can register a periodic task with the `OSTaskCreatePeriodic()` function declared in `os_periodic_task.h`. It takes five arguments:
  - **function** is a pointer to the task function which specifies the work to be done by a job of the task,
  - **period** is the task period (expressed in number of ticks),
  - **offset** is the task phasing (or offset) (expressed in number of ticks),
  - **ptos** is the beginning of the task's stack (stack will be discussed later during the course, in these exercises the `ptos` argument will be given),
  - **prio** is the task priority (the lower the number the higher the priority).

2. **The `main.c` file contains two tasks. Register these tasks with the following parameters:**

Phasing	Period	Function	Priority
1ms	500ms	<b>Task1</b>	<b>Task1Prio</b>
0ms	500ms	<b>Task2</b>	<b>Task2Prio</b>

Note: check the timer frequency setting in the `CPUInitRTI()` function, but do not change this setting!

3. *Before running the program*, take a look at the implementation of the two tasks, and write down on a piece of paper how you expect the leds to behave.

4. Run the program on the board and verify your prediction in the previous step.

Note: Make sure that only one project is open in the CodeWarrior editor, otherwise it is difficult to see which code is uploaded to the board. Also, make sure that only one debugger window is open, otherwise they may interfere and prevent you from uploading/running the program on the board.

5. **Explain the problem with the current setup and propose a solution.**

Hint: take a look at the implementation of the `ATDReadChannel()` function, and keep in mind that a conversion from analog to digital (as implemented by `ATDReadChannel()`) takes 2.153ms.

Note: In this exercise the period, phasing and priority task parameters are fixed (consider them as part of the requirements) and cannot be modified.

6. **Implement your solution and copy the relevant code into your report.**

7. Run the program on the board and verify your solution.

8. **What are the consequences of using critical sections as resource access protocol? Describe one example where a critical section may result in undesired behavior.** Hint: think how a critical section in a lower priority task may impact a higher priority task.

9. (optional) Measure the execution time of `ATDReadChannel()` (expressed in CPU cycles and True Time), and describe how you measured it.