EG3205 Part II. Multiprocessor and Multicore Systems

Assignment 2

Contents

Introduction 1
Description 1
Marking Scheme 4
Submission Guidelines 4
Appendix 1: Running the application 5
Appendix 2: Assigning NIOS II processor IDs 5

Introduction

In Lab 5 you completed the experimental setup for a dual-core processor slave node for a distributed system using the CAN bus.

In this assignment, your goal is to implement the software of a shared-clock network design connecting a single-processor master node and a dual-processor slave node over a CAN bus. The master node will be a single Nios II processor, similar to the one made in Lab 4. These two nodes will be connected to a CAN bus, implemented through the use of CAN-SPI modules.

Description

In the dual-core processor node, one core, which is referred to as Event Triggered (ET) core, is able to handle event-triggered tasks. The other core, which is referred to as Time Triggered (TT) core, is able to handle time-triggered tasks.

The TT core will receive all the messages and take appropriate actions by executing tasks. These actions can include:

- display the message on a JTAG-UART
- Turn-on/off an LED associated with the TT core.

Inter-processor communication is achieved via a shared message buffer and the access of the two cores to the buffer is ensured by a mutex.

The ET core of the dual-processor node will be connected to the CAN bus as a slave. The other node on the CAN bus, which will act as a master, will be assumed to generate “bursts” of data

at random time intervals. Your ET core will receive and process these data. The dual-core processor design is intended for use in safety-critical embedded applications.

Task 1

To begin with, you need to develop the hardware design that you will be using, based on the material provided in Labs 4 and 5:

- The Nios II processor configuration on each board, as illustrated in Figure 1, should be captured in a separate Quartus II project using the Qsys tool.
- The boards must be connected via the CAN-SPI modules as you have previously done.

Once completed move on to Task 2.

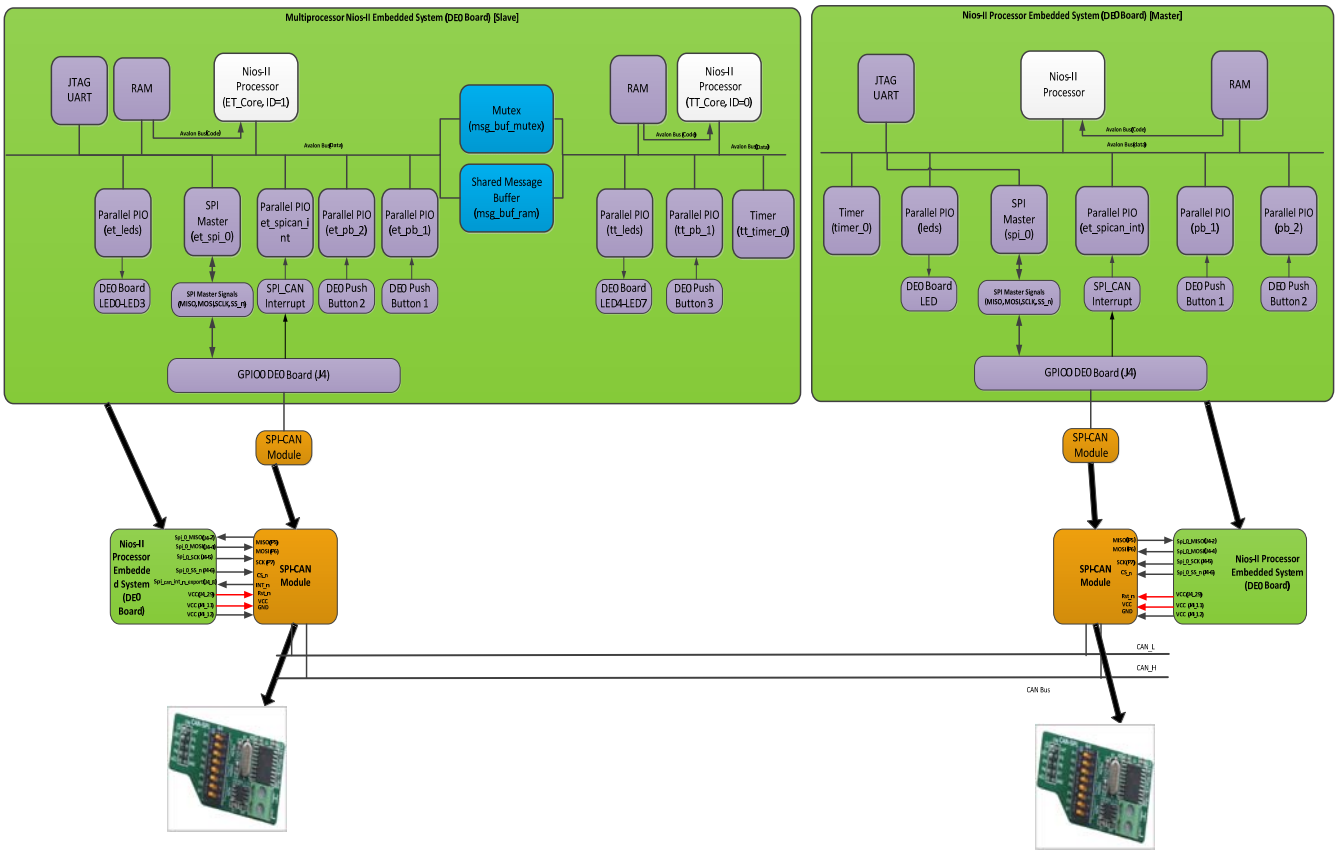


Figure 1

Task 2

Your second task is to develop the necessary application code and apply the software to the hardware as required to create a shared-clock distributed system that implements the operational scenario described below:

1) The Heartbeat LEDs on both boards blinking continuously as follows:
   (i) LED2 - on Master Board,
   (ii) LED2 - on Slave Board (ET core) and
   (iii) LED6 - on Slave Board (TT core).
2) When Button0 on the master board is pressed, LED0 on the slave board (ET core) blinks.
3) When Button0 on the master board is pressed, LED5 on the slave board (TT core) switches ON/OFF.
4) When Button0 on the slave board (ET core) is pressed, LED0 on the master board blinks.

You are expected to demonstrate a successful system operation on the hardware setup showing the above outcomes.

Task 3

Your third task is to analyse the software code of the provided solution for the Master node **app_master_burst_data** and Slave node from Lab 5 and understand the functions performed by the scheduler in making the application run. Carry out the following two sub-tasks:

3.1 Give details of the system-level implementation of the network.

a. Summarise briefly the features of the CAN protocol.
b. Outline the implementation of the shared clock scheduler over CAN.
c. Summarise the tasks of the master and slave nodes.
d. Describe the roles of the two cores on the slave node.
e. Illustrate your answer with a top-level diagram.

3.2 The communication between the two nodes involves sending a message from the master to the slave and receiving an acknowledgement message. Carefully go through the communication process so that you understand how the nodes will be talking to each other and what tasks are being carried out.

a. Work out the communication process between the master and the slave (ET core) node and represent the sequence of communication steps graphically.

b. Describe the following main communication stages and specify the contents of the exchanged messages at each of them.

   (i) Startup

   (ii) Normal Operations: no Button is pressed

   (iii) Master Button0 Press

   (iv) Slave Button0 Press

c. Identify and briefly discuss the C code that is involved in the above communication stages.

Task 4

Write a report with the following sections:

- Introduction.
- Qsys designs for the master and slave boards.
- Description of the application software developed for the Master node.
- Answers to tasks 3.1 and 3.2.
- Conclusions.

Provide photographic evidence to demonstrate correct operation of the distributed system setup.

The report should not exceed 15 pages.

Marking Scheme

Task 1 – 10%

Task 2 – 20%

Task 3 – 40%

Task 4 – 30%

Submission Guidelines

Submission deadline: by 12 pm on the 21st December, Friday.

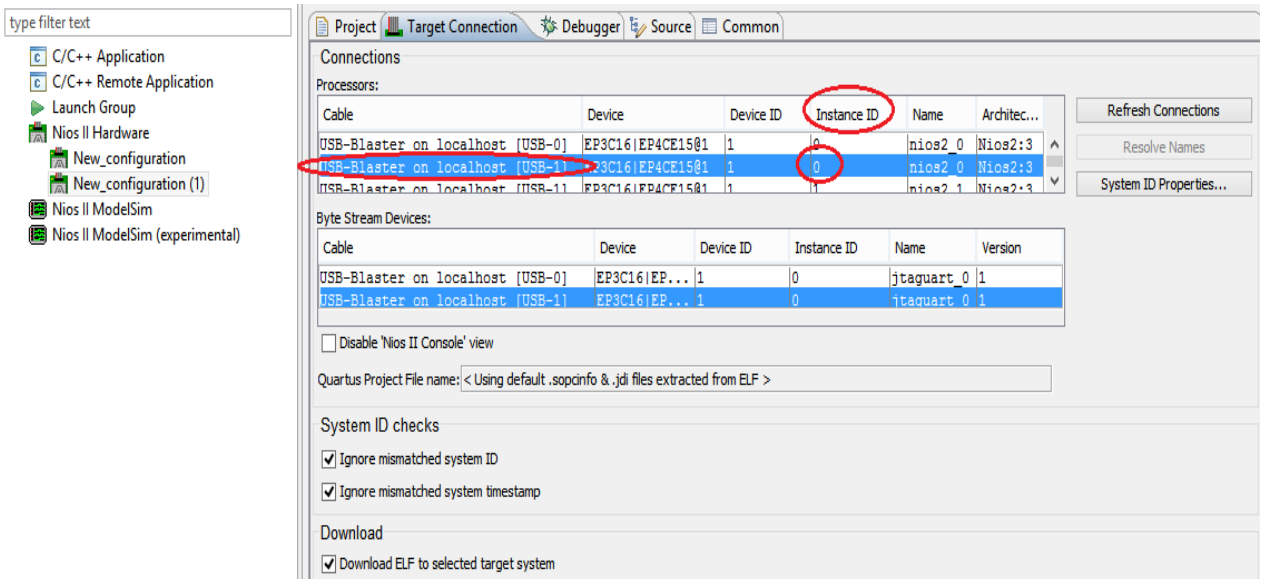
Submit the report and photos / videos as well as any code that you have developed as part of Task 2 in the form of Quartus II projects to Blackboard as one zipped file. Indicate the root directory in which your Quartus II projects reside.

Appendix 1: Running the application

In this exercise, you must download three ELF files to the three NIOS II processors (one file for each processor, one on the master and two for each core on the slave).

To download a file to the target board, use **Run As** by right clicking on the particular project.

- Be careful about the Blaster USB port while downloading your program on the target DE0 board. The screenshot below shows several USB blaster options for downloading programs on the three different processors.
- Each USB blaster will provide you access for downloading the application to a processor.
- In addition, make sure that you have selected the right instance of the Nios II processor while programing the dual-core processor system.



When running the file from the Run As tab in Eclipse, you will need to pick the correct processors **and** the correct core on which to run each project.

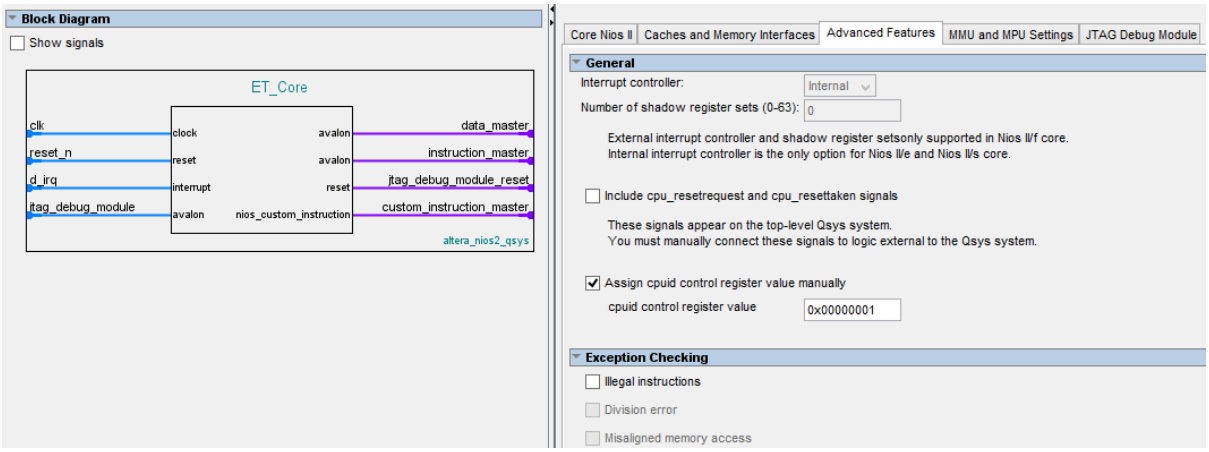
Appendix 2: Assigning NIOS II processor IDs

It is important to assign and acquire the processor IDs in multiprocessor systems. In such

systems all the processors must have unique ID numbers. For instance in the figure bellow we manually assign the processor ID 0x00000001 to the ET core.

In software we can read the processor ID by using this instruction:

cpu_id = ALT_CPU_CPU_ID_VALUE;



Prof. Tanya Vladimirova

Assignment issued: 27 November 2018

Revised: 12 December 2018