Lab 4: Interrupt with Cortex-A9 Processors on DE10

In this lab, you will interface with the generic interrupt controller (GIC) of the ARM Cortex-A9 processor on the DE10 standard board. The document titled Using the ARM Generic Interrupt Controller, posted to Blackboard, will serve as a good reference source for the provided GIC configuration code. Additionally, sections 2.10.4 & 3.4 of the DE10-Standard Computer System Manual describe how to utilize interrupts for the Pushbutton Key Parallel Port.

To help you focus on the core features of interrupts, an example design, called GIC example, is provided for this lab. You will reuse some of the functions in your project. Modify them as needed.

Part I: Study the GIC example

- Download GIC_example.zip from Bb and unzip it to a new folder, e.g. Lab4.
- Make a new project in the Monitor program. During the process, add the given C files (only) to the project. Make sure the top-level file interrupt_example.c appears first on the list when adding files. Header (.h) files should stay in the project folder or a path searchable by the compiler. This is required for all projects. When you reach the last screen illustrated in Figure 1, make sure to choose **Exceptions** in the Linker Section Presets drop-down menu.
- Study the provided C functions line by line and how the functions interface with one another.
- Compile, download, and test the code on the board. Take a picture of the board showing 3 on HEX0.

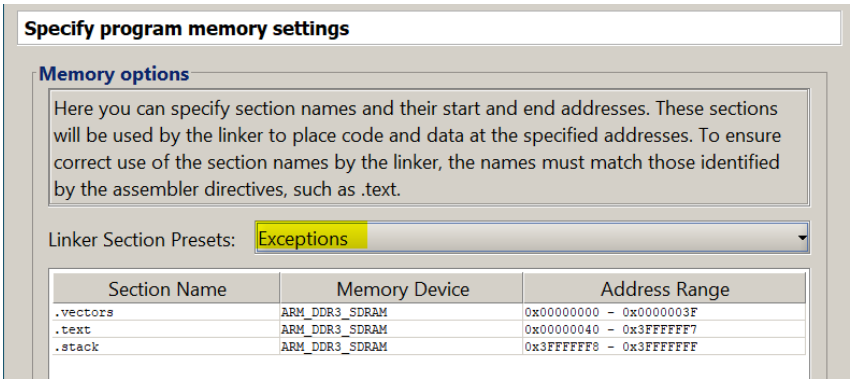

Figure 1: Choose Exceptions for Linker Section Presets

Part II:

In this part, you will modify the main() function and the pushbutton_ISR() to complete the specification provided below. To communicate between main() and the ISR, use global variables. Other functions in the GIC example code remain as provided.

1. In the while (1) loop of the **main () function,** implement an up/down counter and display the counter value on the 10 LEDRs. At the beginning, the counter starts from 0 and counts up or down when a global variable called *direction* is 0 or 1, respectively. When

the counter value reaches the maximum value that can be represented by 10 LEDRs, i.e. 0x3FF, it restarts from 0. When it reaches 0 when counting down, it restarts from 0x3FF. A global variable called *run* pauses the counter when run = 0 and resumes the counter when run = 1. The counter changes behavior depending on which KEY is pressed, as indicated by the value of a global variable *KEY_num.*

- KEY_num = 1 (KEY0): The counter value resets to 0 when *direction* = 0, or 0x3FF when *direction* 1.
- KEY_num = 2 (KEY1): The counter switches between running and pausing.
- KEY_num = 3 (KEY2): The counter switches between counting up and down.
- KEY_num = 4 (KEY3): The LEDRs go blank. The counter continues to count. Pressing any other key will bring the LEDRs back on.

2. Modify the pushbutton_ISR() to add the following functions:
   - Assigns the variable *KEY_num* 1/2/3/4 when KEY0/1/2/3, respectively, is pressed to let main() know which KEY has been pressed.
   - Toggles the *run* variable when KEY1 is pressed.
   - Toggles the *direction* variable when KEY2 is pressed.

Your pushbutton_ISR() also displays the KEY value on the HEX0, which is the original function of the GIC_example.

Suggestions: 1.) One of many possible solutions is to use a switch (KEY_num) statement in the main while (1) loop. Assign case 0 for the common behavior. 2.) Use a delay loop so human eyes can see the changing value shown on the LEDRs.

What to submit:

The picture from Part I and the entire project folder for Part II.