[Lab 5: Interrupt Driven Microblaze: System ]

[Blake Williams]

[10/13/2024]

**Summary**

Give a brief summary of the lab. You may include the requirements, and what activities you did to complete the lab.

This lab builds upon Lab 4 by combining the Timer driven system with an Interrupt driven system. This modification will further improve upon Timing accuracy as Interrupts are generally handled immediately. This means if an interrupt occurs then the handler is triggered instantaneously no matter what execution cycle the code is in. It also helps with CPU usage, as previous implementations utilized polling which probed the interrupt bit for changes. This takes CPU usage, and if more devices are added then delay will begin to occur. With interruptions only when something happens will the CPU handle it. Therefore, saving on CPU usage and further improving accuracy and decreasing delay.

**Design and Implementation**

For this lab the base code that was developed from lab4 remains mostly the same. The major changes come from adding in the interrupt system into the program. The general idea was to make an interrupt occur whenever the timer rolled over and restarted. To achieve this the provided code xtmrctr\_intr\_example.c was utilized. Here it details how to properly initialize a timer and interrupt controller using the Xilinx API. For enabling the Timer the following code was utilized from the example in lab 4 (status = XTmrCtr\_Initialize (**&**TimerCounter, TMRCTR\_DEVICE\_ID); ). This initializes the timer by utilizing the API instead of writing to memory addresses as done previously. Next, for enabling the Interrupt system we use status = XIntc\_Initialize(**&**InterruptController, INTC\_DEVICE\_ID); which initializes the Interrupt system, but what function will be called when an interrupt occurs? This is defined by the XTmrCtr\_SetHandler which calls the TimerCounterHandler when an interrupt occurs. After this Interrupts are enabled, then the Interrupt controller handler is registered to the exception table. Finally the timer options are set using the API, and the timer is started. Using this provided code, we are able to implement it into the previous Lab4 code by replacing the old probing of the interrupt bit to increment count++. Now then the count++ occurs inside the TimerCounterHandler which, as mentioned previously, is called when a interrupt happens. Along with the count++ moving into the TimerCounterHandler, the FSM\_tick() function call is also moved into this function. Now the State machine will tick when an interrupt occurs.

**Conclusion**

This lab taught a wide range of topics pertaining to Interrupt systems, but mainly utilizing the API provided. In the previous labs the only way to interface with the different systems was with manual writing and reading individual bits of a memory address. Now with this lab the Xilinx provided API was utilized and thought, which abstracts the writing to memory addresses away. Making it easier to understand the code, and interface with the different systems on the Arty board. Again this lab was a very well thought out and difficult lab. Which forced participants to understand how the systems work.