Lab 3 – Vivado AXI Timer and Interrupts

Leomar Durán

Summary

Introduction

Discussion

interrupt controller tut 2D.c

The unmodified *interrupt_controller_tut_2D.c* project has five module functions. Namely, these are:

- `BTN_Intr_Handler(void *):void`,
- `TMR_Intr_Handler(void *):void`,
- `main(void):int`,
- `InterruptSystemSetup(XScuGic *):int`, and
- `IntcInitFunction(u16, XTmrCtr *, XGpio *):int`.

BTN Intr Handler

The `BTN_Intr_Handler(void *):void` module function handles button interrupts by increasing the LED counter by the buttons pressed. Specifically, it performs Algorithm 1.

Algorithm 1: BTN INTR HANDLER

Input: unused instance pointer.
DISABLE button interrupts on `axi_gpio_0`.
if *there are other interrupts from `axi_gpio_0` not from channel* 1 **then**
  return to the caller.
READ in the value of the button.
ADD the value of the button to the LED counter data.
WRITE the LED counter data to channel 1 on `axi_gpio_1`.
CLEAR the button interrupt flag on `axi_gpio_0`.
REËNABLE button interrupts on `axi_gpio_0`.

TMR Intr Handler

The `TMR_Intr_Handler(void *):void` module function handles timer interrupts, by periodically incrementing the LED counter. Specifically it performs the Algorithm 2.

Algorithm 2: TMR INTR HANDLER

Input: unused data pointer
if *the timer expires* **then**
    if *this is the expiration #*3 **then**
        STOP the timer counter.
        RESET the expiration count.
        INCREMENT the LED counter data.
        WRITE the LED counter data to channel 1 on `axi_gpio_1`.
        RESET the timer counter.
        START the timer counter.
    else
        INCREMENT the expiration count.

Main

The `main(void):int` module function orchestrates all of the operations necessary to run the program. In this case, it initializes the peripheral devices (the LEDs and buttons) and the corresponding GPIOs. It also starts the timer and initially enables the interrupts.

Then it polls.

Algorithm 3: MAIN

Output: 0 on success; `XST_FAILURE` if there is either an error initializing the peripherals or the timer.

INITIALIZE the instance for `api_gpio_1`, the GPIO for LEDs.
if *the initialization was not successful* **then**
  return failure.
INITIALIZE the instance for `api_gpio_0`, the GPIO for the push buttons.
if *the initialization was not successful* **then**
  return failure.
SET the DDR for the LEDs to all outputs.
SET the DDR for the buttons to all inputs.

INITIALIZE the instance for the timer.
if *the initialization was not successful* **then**
  return failure.
ATTACH `TMR_Intr_Handler(void *):void` to handle interrupts on the instance for the timer, bound to the instance for the timer.
SET the compare value of the timer to `0xF8000000`.
SET the timer options to interrupt mode and to reset upon hitting the compare value.

CALL the initialization function `IntcInitFunction(u16, XTmrCtr *, XGpio *):int` using the interrupt controller `xlconcat_0`, the instance for the timer, and the instance for `api_gpio_1`.
if *the initialization was not successful* **then**
  return failure.
START the timer counter.

POLL indefinitely.
return 0 for success.

InterruptSystemSetup

The `InterruptSystemSetup(XScuGic *):int` module function sets enables the button interrupts and sets up the exception handler to the primary interrupt handler. Specif-

ically, it performs Algorithm 4.

Algorithm 4: INTERRUPTSYSTEMSETUP

Input: driver instance data.
Output: `XST_SUCCESS` always.

ENABLE button interrupts on `axi_gpio_0`.
CONFIRM enabling button interrupts with the global enable.

REGISTER the primary interrupt handler, bound to the driver instance data.
ENABLE the exception handling.

return success.

IntcInitFunction

The `IntcInitFunction(u16, XTmrCtr *, XGpio *):int` module function does most of the setting up of the interrupt controller `xlconcat_0`. Specifically, it per-

forms Algorithm 5.

Algorithm 5: INTCINITFUNCTION

Input: ID of the device to configure, instance for the timer, instance for a GPIO.
Output: `XSL_SUCCESS` on success; `XSL_FAILURE` if there is an error connecting the GPIO or timer to the handler.

LOOK UP the interrupt controller.
INITIALIZE a driver instance data using the configuration for the interrupt controller.
if *the initialization was not successful* **then**
    return failure.

CALL the initialization function `InterruptSystemSetup(XScuGic *):int` using the interrupt controller driver instance data.
if *the initialization was not successful* **then**
    return failure.

PUT in the driver instance data, a connection from the GPIO interrupt to `BTN_Intr_Handler(void *):void`, bound to the instance of GPIO.
if *the initialization was not successful* **then**
    return failure.

PUT in the driver instance data, a connection from the timer interrupt to `TMR_Intr_Handler(void *):void`, bound to the instance of timer.
if *the initialization was not successful* **then**
    return failure.

ENABLE GPIO interrupts.
CONFIRM enabling GPIO interrupts with the global enable.

ENABLE GPIO interrupts on the the driver instance data.
ENABLE Timer interrupts on the the driver instance data.

---

return success.

Conclusions
Appendices