Lab 6 - Taylor Cowley and Andrew Okazaki

- Programmable Interrupt Timer (PIT)
    - Register Descriptions
    - Timing Diagrams
    - Driver API
    - Bug Report

Programmable Interrupt Timer (PIT)

Register Descriptions

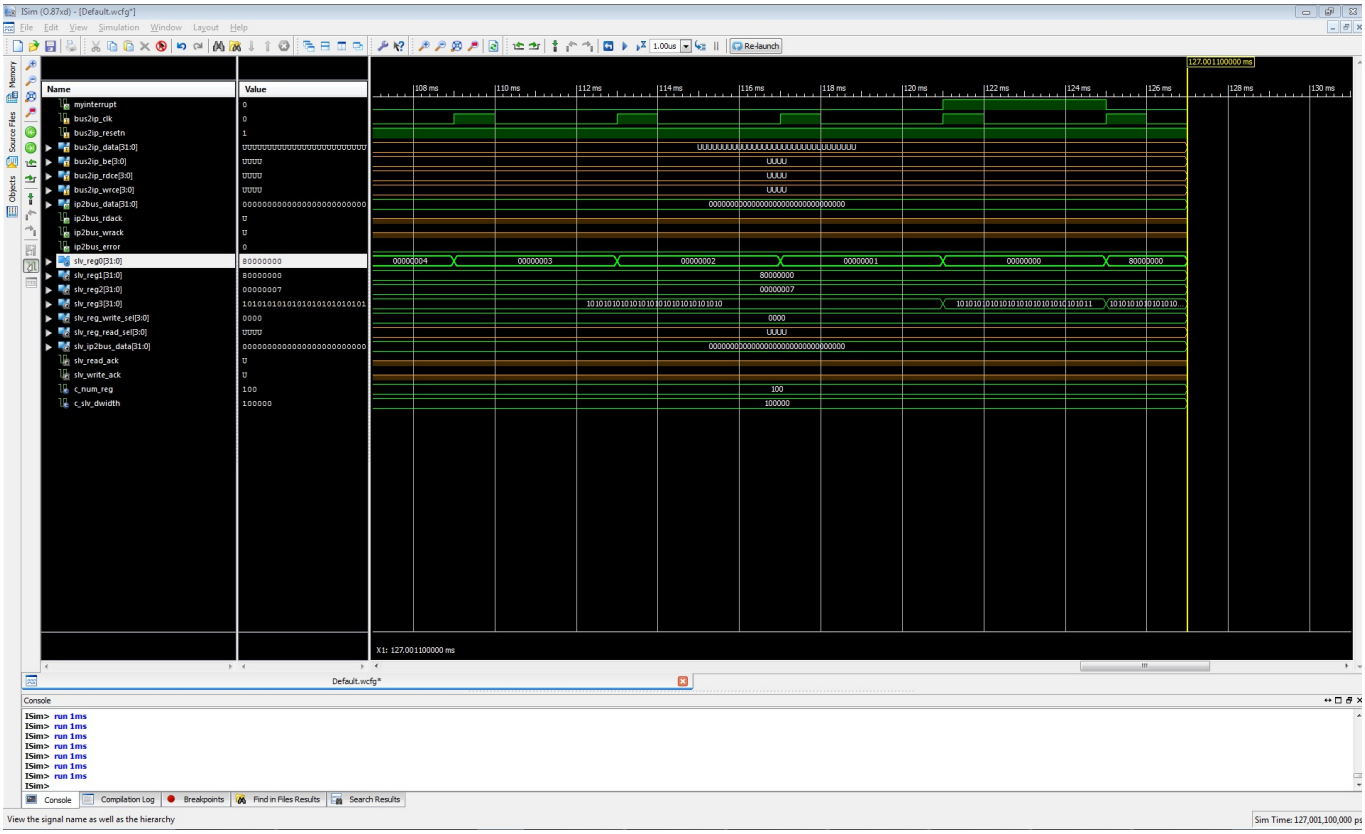
I our PIT we used four different registers named, slv_reg0, slv_reg1, slv_reg2, slv_reg3.

Register Name	Register Address (in xparameters.h)	Purpose
slv_reg0	0x7bc00000	Used as a 32 bit counter
slv_reg1	0x7bc00004	Contains the delay number
slv_reg2	0x7bc00008	The control register, if bit_0 = 1 allow it to decrement else do not. If bit_1 = 1 then show interrupts else do not show interrupts.,If bit_2 = 1 allow the value to reload once it hits 0 else do not reload.
slv_reg3	0x7bc00010	Bit 0 contains output to signal an

		interrupt. Bits 31-1 are a pre-set value used for debugging

Timing Diagram

This is a simulation of our PIT timer. Note at the top, because the count register has reached 1, and the control register's 1 bit is on, an interrupt is generated for one clock cycle



Driver API

The api was automatically generated in `pit.h` by the EDK. The important functions are listed below. BaseAddress should always be `XPAR_PIT_0_BASEADDR` (generated in `xparameters.h`), and the RegOffset should always be `0`.

```
// Used to write a value to reg0, the count register
#define PIT_mWriteSlaveReg0(BaseAddress, RegOffset, Value
```

```
) \
    Xil_Out32((BaseAddress) + (PIT_SLV_REG0_OFFSET) + (Re
gOffset), (Xuint32)(Value))

// Used to write a value to reg1, the reload register. A
higher number makes the PIT generate interrupts slower
#define PIT_mWriteSlaveReg1(BaseAddress, RegOffset, Value
) \
    Xil_Out32((BaseAddress) + (PIT_SLV_REG1_OFFSET) + (Re
gOffset), (Xuint32)(Value))

// Used to write a value to reg2, the control register. T
he bits that matter are listed in the above table
#define PIT_mWriteSlaveReg2(BaseAddress, RegOffset, Value
) \
    Xil_Out32((BaseAddress) + (PIT_SLV_REG2_OFFSET) + (Re
gOffset), (Xuint32)(Value))

// Used to write a value to reg3. Dangerous. Do not use.
#define PIT_mWriteSlaveReg3(BaseAddress, RegOffset, Value
) \
    Xil_Out32((BaseAddress) + (PIT_SLV_REG3_OFFSET) + (Re
gOffset), (Xuint32)(Value))


// The following are to read from the registers.
#define PIT_mReadSlaveReg0(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (PIT_SLV_REG0_OFFSET) + (Reg
```

```
Offset))
#define PIT_mReadSlaveReg1(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (PIT_SLV_REG1_OFFSET) + (Reg
Offset))
#define PIT_mReadSlaveReg2(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (PIT_SLV_REG2_OFFSET) + (Reg
Offset))
#define PIT_mReadSlaveReg3(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (PIT_SLV_REG3_OFFSET) + (Reg
Offset))
```

Bug Report

In our code we had a hard time with hardware often we would not include hardware that was needed. Often we did not know why or when some of the hardware modules were dropped from of our code. We ended up having to generate our hardware multiple times while trying to pass this lab off. The second error that we ran into was that our PIT interrupt timer was not generating an interrupt. This was a difficult problem but after simulating we were able to update the logic so that the pit would register a high value after a given amount of time. After knowing that the PIT timer was working correctly using it in our code took a while to figure out. But after using the generated C functions we were able to write and read checking that everything was working properly.