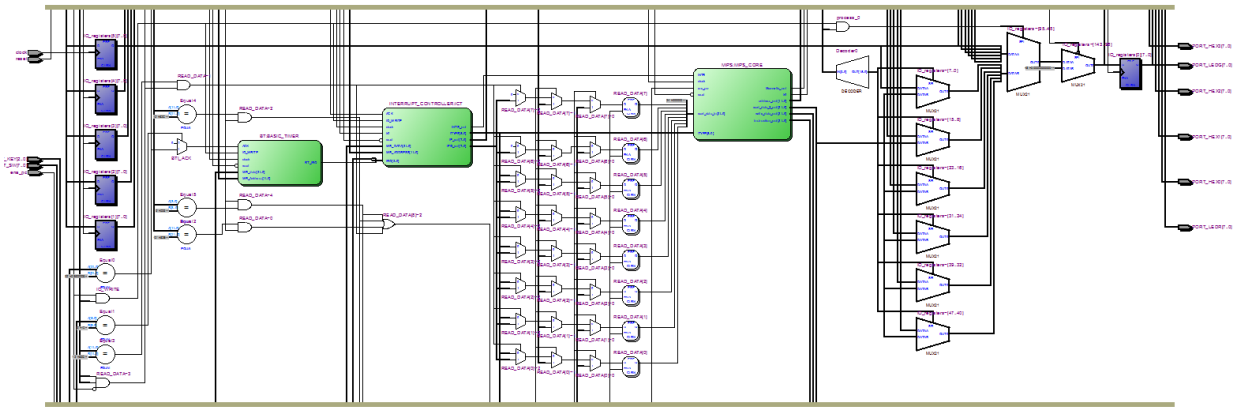
FINAL PROJECT

Design

The following block diagrams represent the system's design as shown in Quartus RTL viewer:

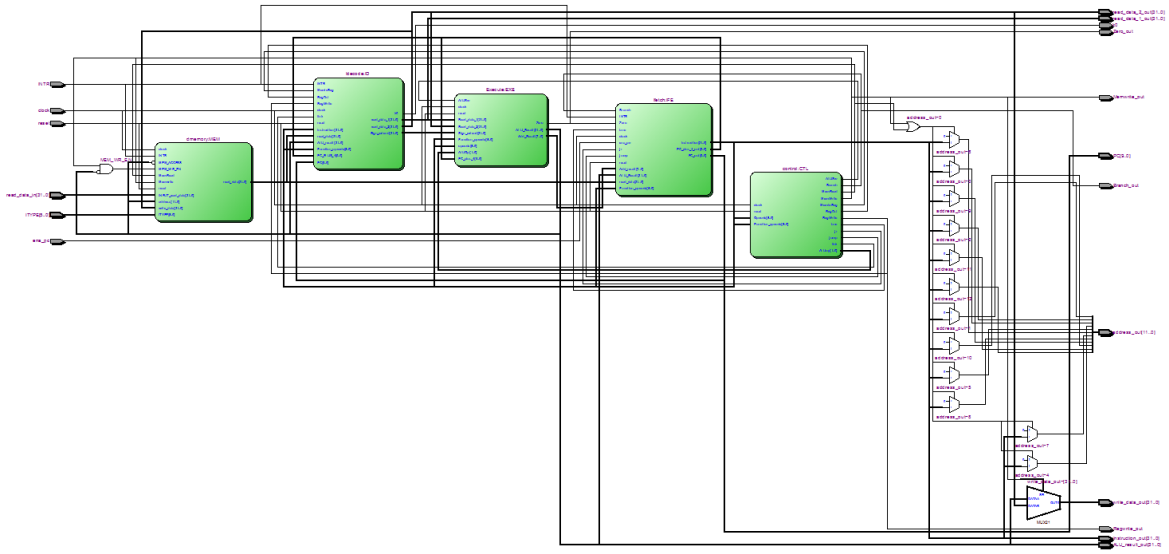


DESCRIPTION:

This is an overview of the design, which operates as a fully functional MCU Micro Controller Powered by a complete- ISA-enabled Single-Cycle MIPS processor, being fed a memory component with data and a MIPS Assembly program, and syncs fetching the program, decoding, executing and reading/ writing to memory if needed and handling Interrupt Service Routines

MIPS CPU core:

RTL diagram:

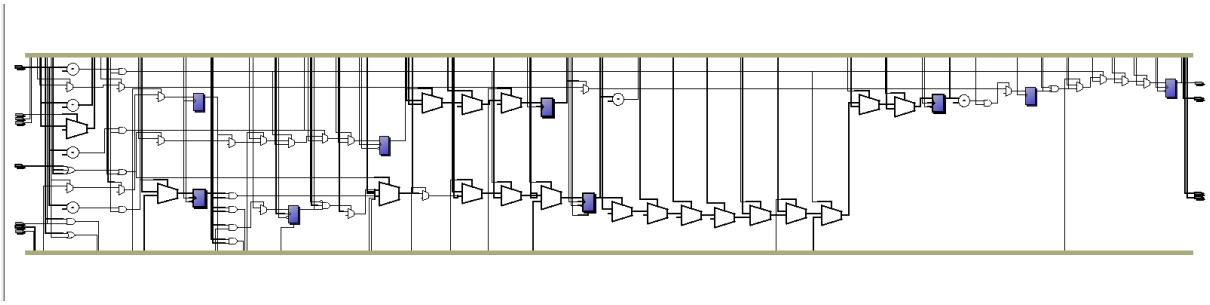


DESCRIPTION:

a complete- ISA-enabled Single-Cycle MIPS processor, being fed a memory component with data and a MIPS Assembly program, and syncs fetching the program, decoding, executing and reading/ writing to memory if needed. The Core is interrupt-enabled. Controlled by hardware inputs, an Interrupt signal may cause the CPU to perform a JAL command emulation, sending the PC to the Interrupt Service Routine address stored in memory, ACK'ing when it returns from said routine.

Interrupt Controller:

RTL diagram:

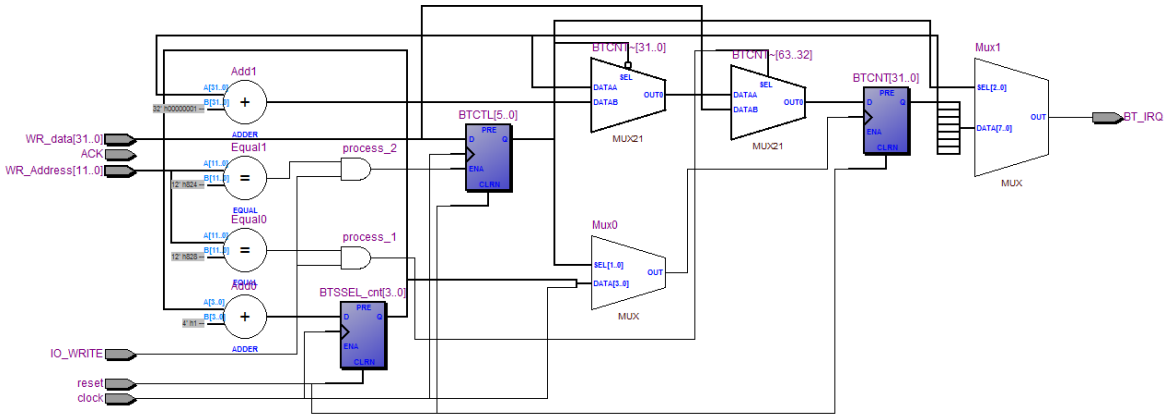


DESCRIPTION:

The Interrupt Controller module is driven by keys-input on the Cyclone V hardware, is responsible to decode,prioritize and trigger the correct Interrupt signals for each trigger- either from Hardware inputs or from the Basic timer module.

Basic Timer:

RTL diagram:



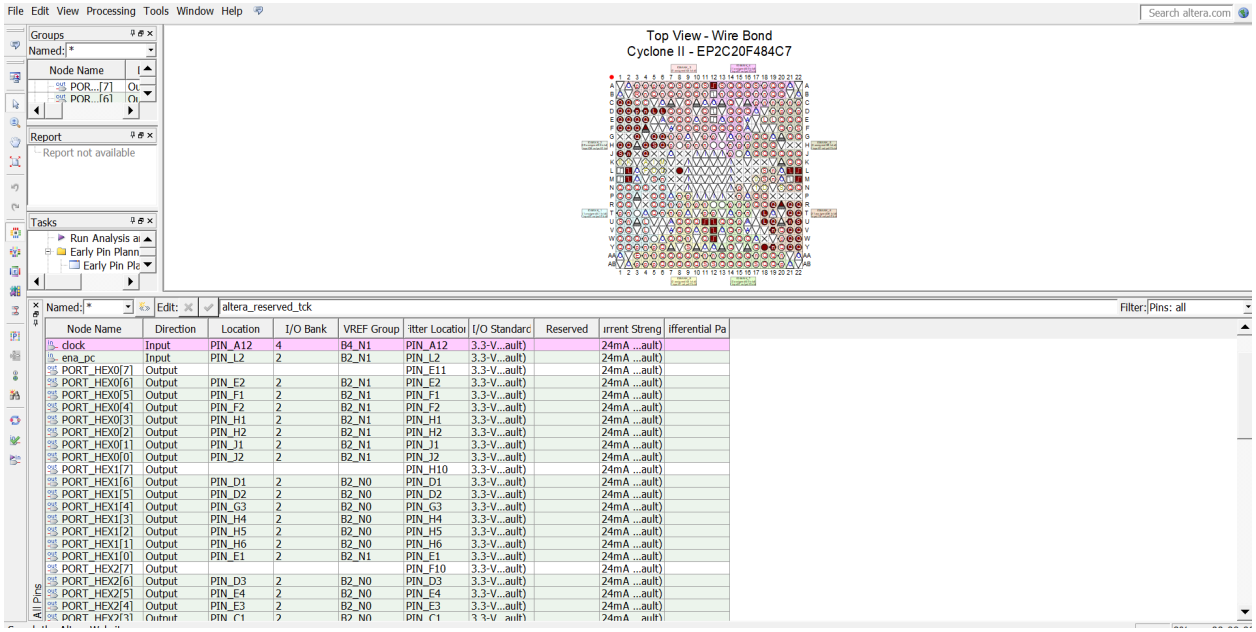
DESCRIPTION:

The basic timer is an Interrupt enabled module which offers the user some advanced clock related functionality including: clock based delay ,custom frequency recurring interrupts, and a clock based "sleep mode". The BT module consists of a divisible clock, a custom trigger counter, and a "BT flag" output in order to cause an interrupt procedure in the MIPS core.

Logic Usage

Flow Summary	
Flow Status	Successful - Sun Aug 28 16:58:07 2022
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	project1
Top-level Entity Name	TOP
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	7,095 / 18,752 (38 %)
Total combinational functions	3,970 / 18,752 (21 %)
Dedicated logic registers	4,693 / 18,752 (25 %)
Total registers	4693
Total pins	62 / 315 (20 %)
Total virtual pins	0
Total memory bits	37,696 / 239,616 (16 %)
Embedded Multiplier 9-bit elements	6 / 52 (12 %)
Total PLLs	0 / 4 (0 %)

Pin planner

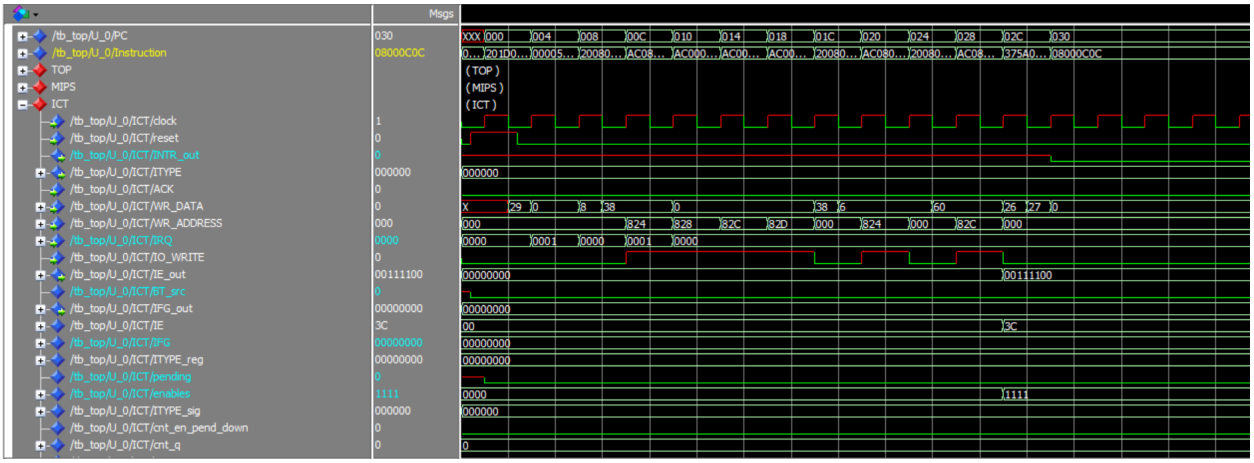


Pins were assigned to the relevant I/O interfaces and Interrupt triggers. The above diagram show the layout on the device.

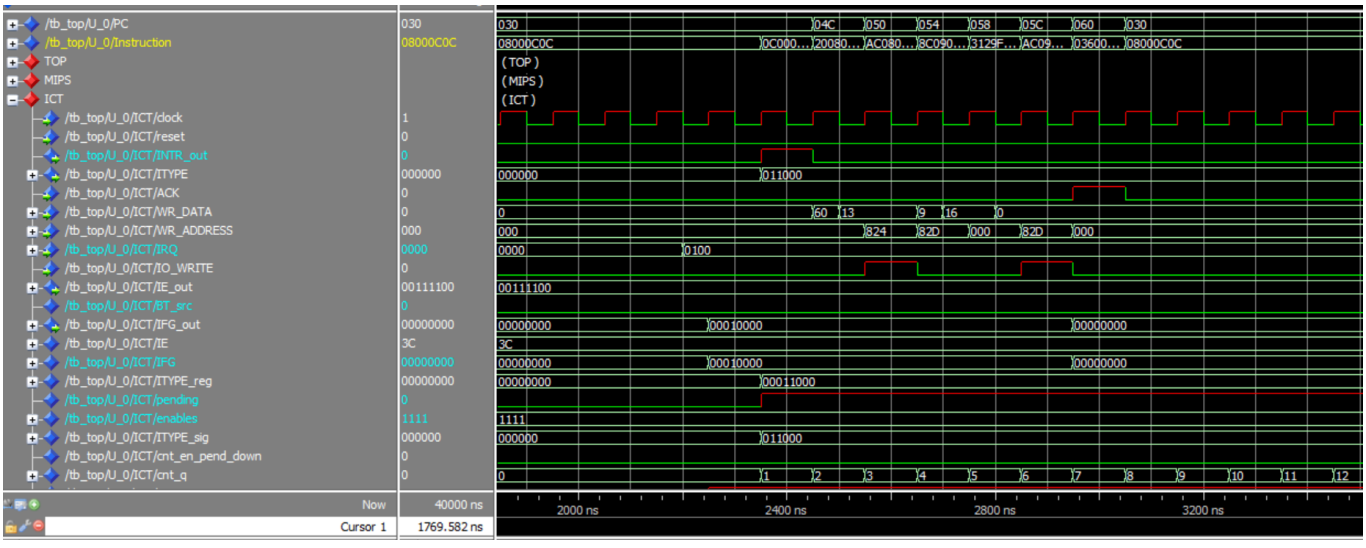
Signal waveform:

This is the waveform representation as given by the MODELSIM simulation:

This example spans from reset to the infinite loop with just one “jump” instruction.



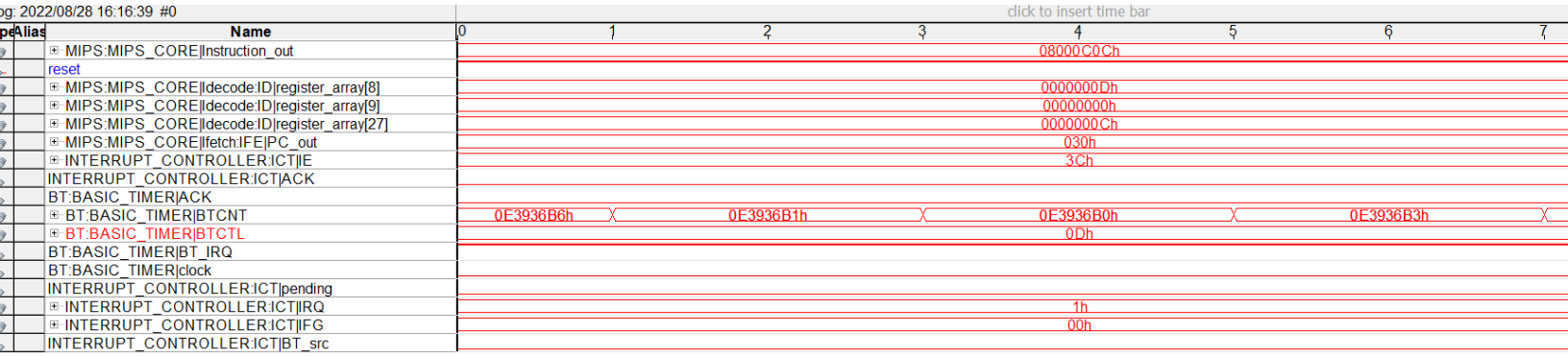
This one shows an Interrupt triggered by a key press.



Key press, followed by the rise of IRQ signal, raising the relevant flag and finally the actual interrupt request.
The return from the ISR, back to the waiting Loop can be observed as well.

QUARTUS SignalTap waveform:

This is the waveform representation as given by the QUARTUS Signaltap:



Basic timer upcount caught on SignalTap.

Current instruction is “jump” in the main segment’s infinite loop.

The upcount is visible in the BTCNT register.

BT_IRQ shows that the basic timer interrupt bit is set to 1.

Timing

We analyzed the timing quest from the compilation report and got the following results:

Fmax analysis:

The following table shows the maximal clock frequency:

	Fmax	Restricted Fmax	Clock Name	Note
1	18.13 MHz	18.13 MHz	clock	

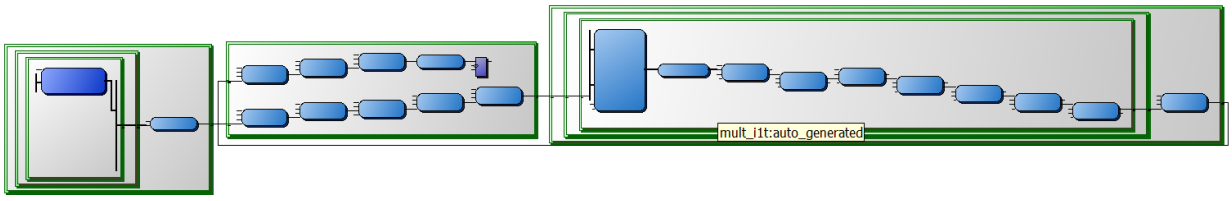
Critical path:

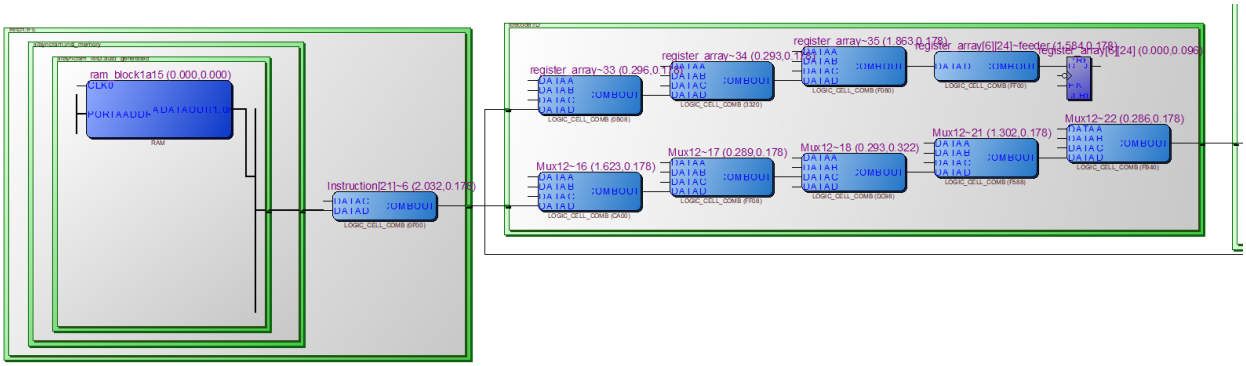
Critical path analysis:

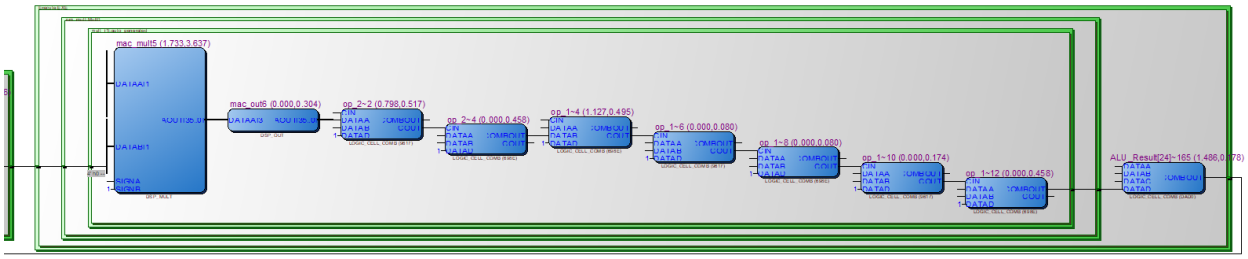
The following table shows the systems longest combinatorial (critical) path, as shown in time quest analysis:

Now, unsurprisingly, we can derive that the bottleneck is in a MULT instruction including reading from ram and propagating through the multiplier module. It is the one responsible for the Execute stage of the operation.

it is quite unexpected - as we would think that a peripheral module would require the most intense computation time. In real life, most of the time we would consider memory and I/O access as the longest path.







Optimizations:

Given the above insights, the ultimate step is to optimize the code in order to further the performance. The following was done accordingly:

- the clock period was changed to the optimal frequency:

$$t_{clk} = \frac{1}{f_{max}} = \frac{1}{18.13MHz} = 5.515 \times 10^{-8} \sim 5.55ns$$

-unnecessary registers were removed

-unnecessary outputs were converted to signals

-unnecessary clocks were removed

-latches were replaced

In addition, we added an analysis of critical path and frequency limiting operations for each component:

The critical path inside the adder block can be seen in the green box below

PLL

According to the above Fmax analysis, it was reasonable to use PLL to achieve the desired clock frequency.

Clock configuration:

Critical path analysis:

We acheaved the following values in previous stages :

$$t_{clk}[ns] = \frac{1}{f_{max}} = \frac{1}{18.13MHz} = 55.15 \times 10^{-9}[sec]$$

$$= 5.515[ns] \sim 5.55ns$$

$$f_{clk} = 5.55^{-1} \sim 18MHz$$

$$\sim 50 \times \frac{9}{25} MHz$$

Now, a good approximation would be achieved by using a 9 fold clock multiplier alongside a 25 fold clock divider.

Therefor the PLL was configured as such: