CPUGEN TUTORIAL 2.00

gferrante@opencores.org

Getting started

1) Decompress cpugen.zip into a working directory (ex. C:\cpugen)

2) Add to windows PATH the working directory you have uncompressed files to

3) Altera users: from Altera site (www.altera.com)

3.1) Download and install Quartus II [ver. >= 3.0] (Altera FPGA development tool); ask for a free/commercial license

3.2) If you want to do VHDL simulations: download and install Modelsim AE (VHDL simulator); ask for a commercial license [ver. >= 5.7c]

4) Xilinx users: from Xilinx site (www.xilinx.com)

4.1) Download WebPack ISE [ver. >= 6.1i] (Xilinx FPGA development tool); ask for a free/commercial license

4.2) If you want to do VHDL simulations: download and install Modelsim XE (VHDL simulator); ask for a free/commercial license [ver. >= 5.7c]

Simulation notes: Quartus has a built-in waveform simulator; ISE needs a VHDL simulator.

NOTE: differences between cpugen rel 1.0 and 2.0 in CpuUpdate.txt

Applications

You will find some sample projects into the applications directory.
The VHDL cpu source code, assembler and rom/ram files are located in the relative design directory; other VHDL files used by the design are located in Components and Testbench directories too.

Each design has 3 implementations:

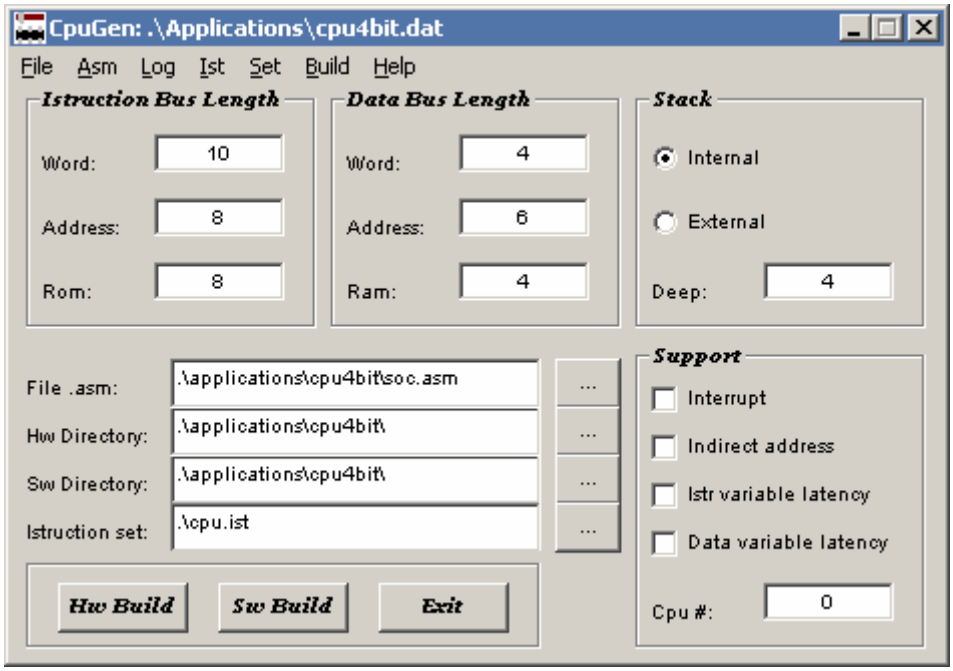
1) RTL design (vendor independent) :Testbench\testbench_apre.mpf or testbench_xpre.mpf
2) Altera design :Testbench\testbench_apost.mpf
3) Xilinx design :Testbench\testbench_xpost.mpf

Cpu4Bit

Goal:

- compare 4 digital input with 4 fixed thresolds and change a PWM output consequently
- use a microcontroller architecture inside a low cost FPGA device (< 10K gates device)
- use few logic elements and memory resources

Run **cpugen** and load Applications\cpu4bit.dat cpu configuration:



RTL design:

VHDL sources:

1) CPU

- Cpu#_utils.vhd
- Cpu#_iu.vhd
- Cpu#_cu.vhd
- Cpu#_du.vhd
- Cpu#_oa.vhd
- Cpu#_wd.vhd
- Cpu#.vhd

2) COMPONENTS

- Ctrl4cpu.vhd
- Pwm.vhd
- Utils_pkg.vhd
- Ram.vhd (only for RTL testbench; use ram.vin)
- Rom.vhd (only for RTL testbench; use rom.vin)

3) TESTBENCH (RTL)

- Cpu4bit.vhd
- Clock.vhd
- World.vhd
- Testbench_a.vhd

CPU 4 bit => Data Bus Word Length = 4 bit
IBUS = Instruction Words Length – 4 = 6
Instruction Words Length = 10 bit => instruction/data page size = 64 word = (2 ** IBUS)
Instruction Address Length = 8 bit => number of instruction pages = 4 = (2 ** (8 – IBUS))
Instruction Rom Length = 8 bit => max. number of code words = 256 = (2 ** 8) [x10]
Data Bus Address Length = 6 bit => number of data pages = 1 = (2 ** (6 – IBUS))
Data Ram Length = 4 bit => max. number of ram words = 16 = (2 ** 4) [x4]
Data Expansion = 0 bit (IBUS >= 4)
Stack = internal, deep = 4 (see Help/CpuCore: How to set stack deep)

Data address mapping (4 bit size):

- RAM 0 ... 0xF
- CTRL 0x10 = CTRL_IN = digital input; CTRL_OUT[0] = PWM enable
- `PWMPL` 0x12 = PWM period (low significative nibble)
- `PWMPH` 0x13 = PWM period (high significative nibble)
- `PWMLL` 0x14 = PWM low level (low significative nibble)
- `PWMLH` 0x15 = PWM low level (high significative nibble)

The cpu polls 4 digital inputs => It doesn't need Interrupt support.

Code structure:

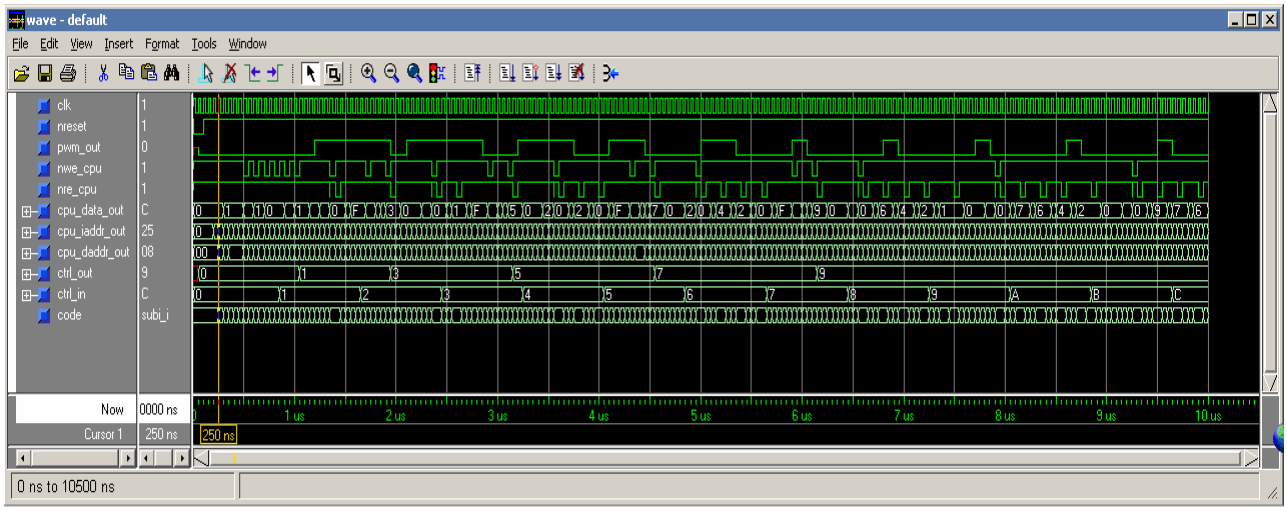
Instruction page 0 = polling loop
Instruction page 1 = registers initialization
Instruction page 2 = PWM setting

Now go to the "ASM" menu to view the source code, then "HW Build" to generate the VHDL CPU core and "SW Build" to generate Ram /Rom files. The "Log" menu shows the software compilation report.

If the generation process completes successfully, the VHDL simulation can start. The current testbench shows the PWM behaviour with a digital input ramp.

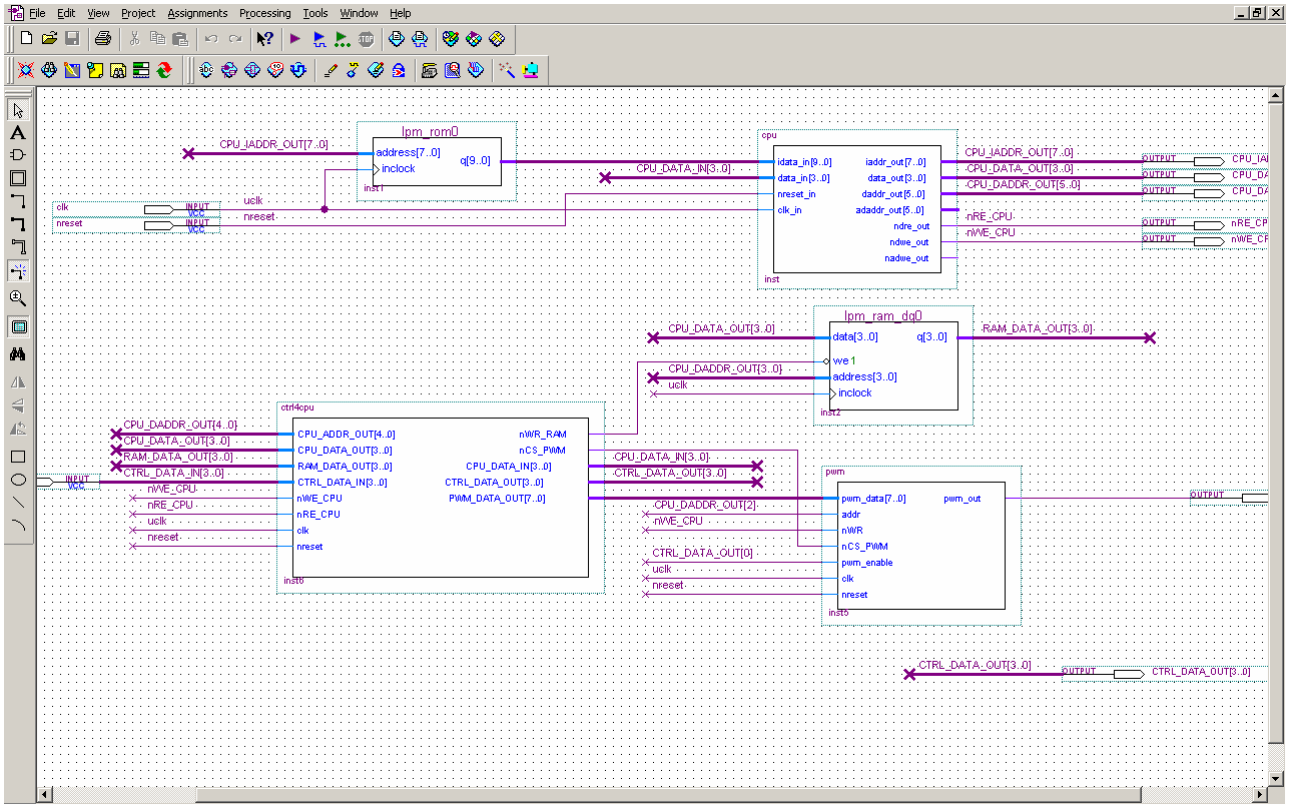
Run ModelSim and take the following actions:

1) File\Open\Project select the path for cpu4bit\Testbench\testbench_apre.mpf (or testbench_xpre.mpf)
2) Compile\Compile All
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ns
4) Simulate\Simulate Options: Default Run: 10000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_pre.do
7) Simulate\Run: Run 10 us (MACRO: sim_pre.do)



Altera Design:

Start cpu4bit.quartus in cpu4bit\Altera directory.

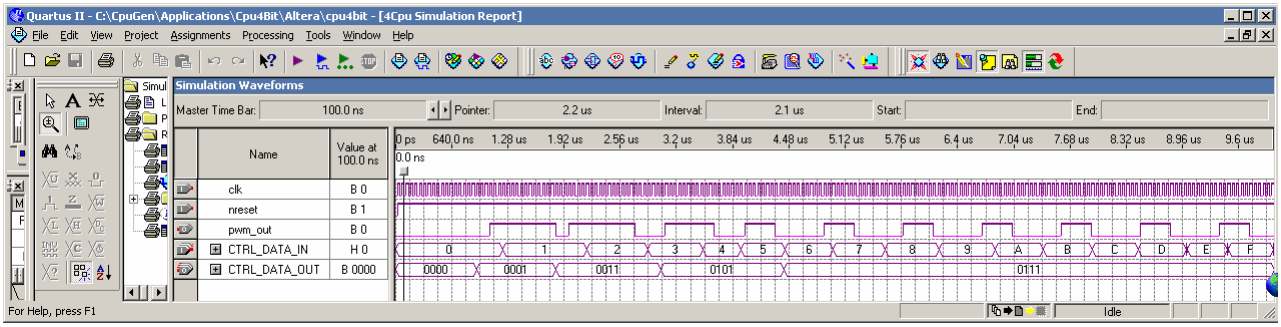


EAB (Embedded Array Blocks) are used for ram (ram.mif) and rom (rom.mif).
In Assignments\Device choice ACEX1K EP1K10TC100-1 device.
Then with Processing\Start Compilation Quartus synthetize the design.

Output:

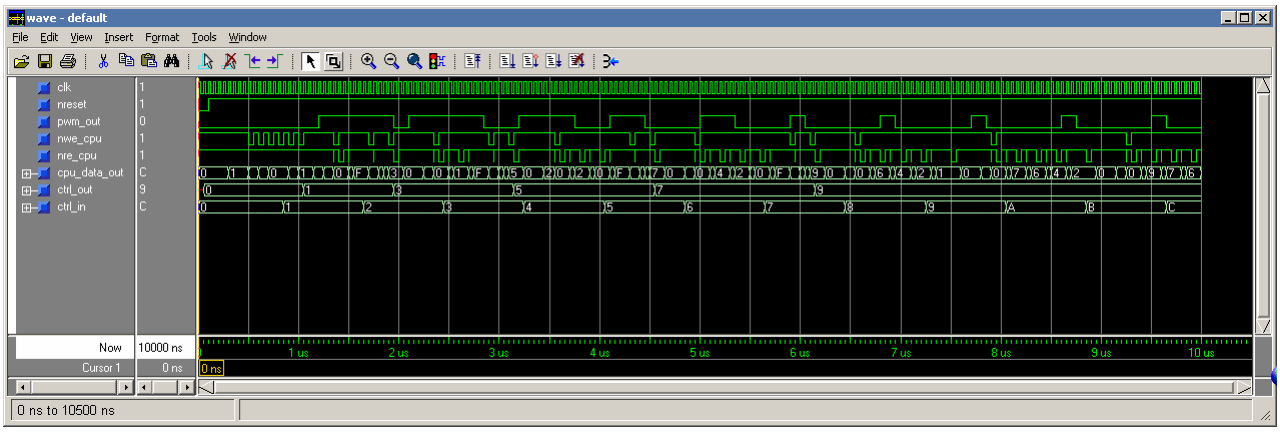
- Device EP1K10TC100-1 = 10K gates
- 384 / 576 (66%) = Logic Elements
- 2624 / 12288 (21%) = Memory bits
- 33.07 MHz = Max frequency clock

1) Simulate the design using Quartus Waveform simulator:



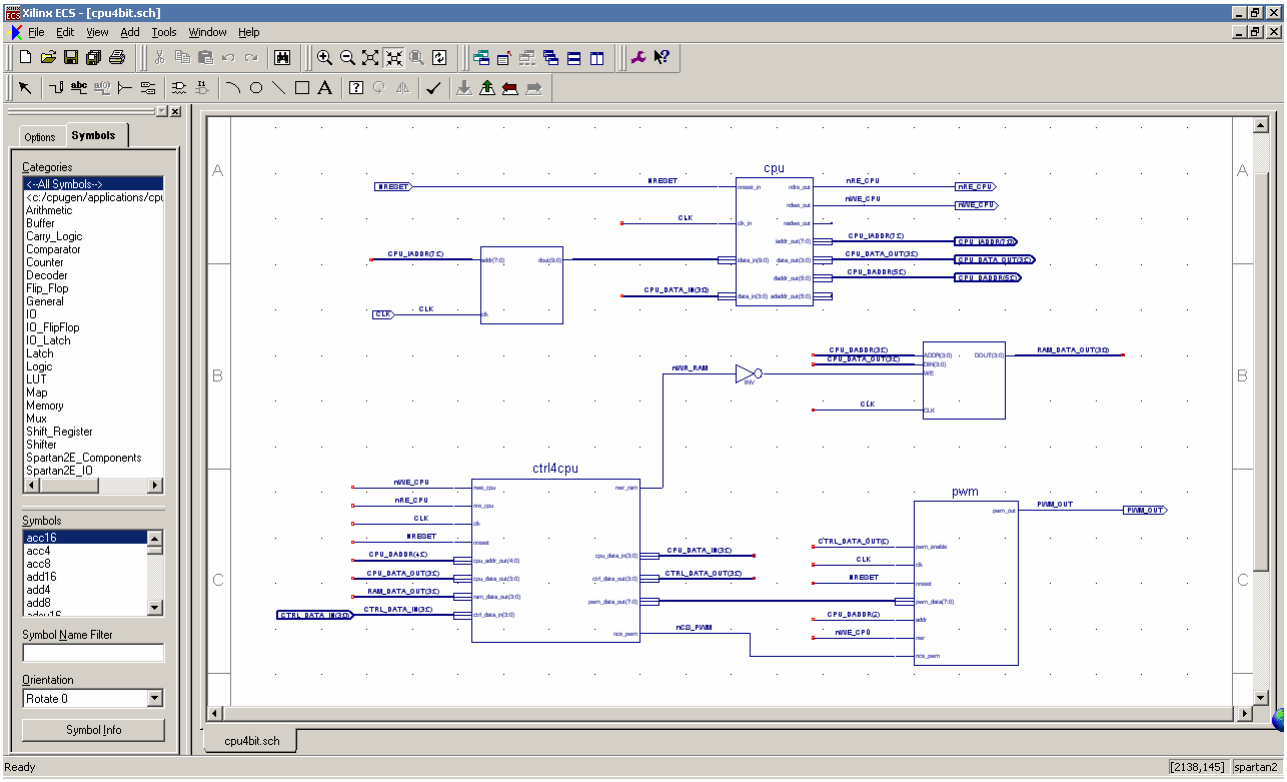
2) Simulate the design using Modelsim AE: (cpu4bit.vho = Quartus output)

1) File\Open\Project select the path for cpu4bit\Testbench\testbench_apost.mpf.
2) Compile\Compile All
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ns
4) Simulate\Simulate Options: Default Run: 10000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_post.do
7) Simulate\Run: Run 10 us (MACRO: sim_post.do)



Xilinx Design:

Start cpu4bit.npl in cpu4bit\Xilinx directory.



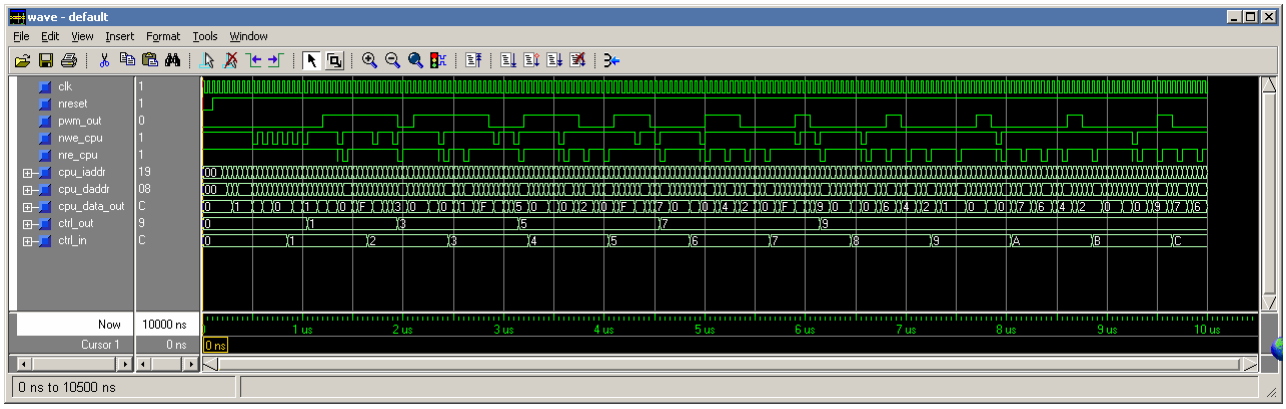
Memory Blocks are used for ram (ram.coe/ram.cfg) and rom (rom.coe/rom.cfg).
NOTE: It's necessary to update initialization files if you use a different project path directory (rom/ram.xco)
Remember to Regenerate Core, Update schematic and regenerate cpu4bit_timesim.vhd output when change hw/sw cpu.
I used a Spartan2 device: XC2S30-5cs144
Then, select cpu4bit and Generate the Programming file.

Output:

- Total Number Slice Registers: 115 out of 864 13%
- Number of 4 input LUTs: 392 out of 864 45%
- Number of occupied Slices: 226 out of 432 52%
- Clock Period 23.784 ns: Max Frequency = 42.045 MHz

1) Simulate using Modelsim XE: (cpu4bit_timesim.vhd = Xilinx output)

1) File\Open\Project select the path for cpu4bit\Testbench\testbench_xpost.mpf.
2) Compile\Compile All
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ns
4) Simulate\Simulate Options: Default Run: 10000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_post.do
7) Simulate\Run: Run 10 us (MACRO: sim_post.do)

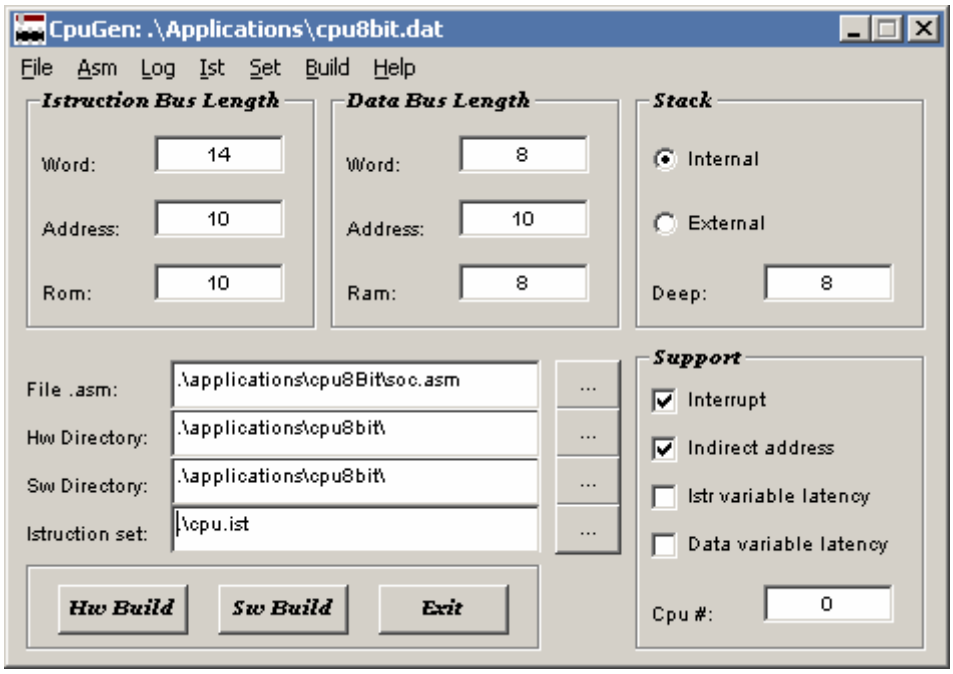


Cpu8Bit

Goal:

- check rising edges for 4 digital inputs (4 interrupt sources)
- control 8 bit ports (2 In and 2 Out ports => 32 I/O)
- control one Timer (1 interrupt source)
- send/receive data via a UART (3 interrupt source)

Run **cpugen** and load Applications\cpu8bit.dat cpu configuration:



RTL design:

VHDL sources:

1) CPU

- Cpu#_utils.vhd
- Cpu#_iu.vhd
- Cpu#_cu.vhd
- Cpu#_du.vhd
- Cpu#_oa.vhd
- Cpu#_wd.vhd
- Cpu#.vhd

2) COMPONENTS

- Ctrl8cpu.vhd
- Inout4reg.vhd
- Interrupt.vhd
- Timer.vhd
- Tx_uart.vhd
- Rx_uart.vhd
- Utils_pkg.vhd
- Ram.vhd (only for RTL testbench; use ram.vin)
- Rom.vhd (only for RTL testbench; use rom.vin)

3) TESTBENCH (RTL)

- Cpu8bit.vhd

- Clock.vhd
- World.vhd
- Testbench_a.vhd

CPU 8 bit => Data Bus Word Length = 8 bit
IBUS = Instruction Words Length – 4 = 10
Instruction Words Length = 14 bit => instruction/data page size = 1024 word = (2 ** IBUS)
Instruction Address Length = 10 bit => number of instruction pages = 1 = (2 ** (10 – IBUS))
Instruction Rom Length = 10 bit => max. number of code words = 1024 = (2 ** 10) [x14]
Data Bus Address Length = 10 bit => number of data pages = 1 = (2 ** (10 – IBUS))
Data Ram Length = 8 bit => max. number of ram words = 256 = (2 ** 8) [x8]
Data Expansion = 0 bit (IBUS >= 8)

Resources:

- 256 byte RAM (8)
- interrupt controller
- timer
- tx Uart
- rx Uart
- 2 input ports
- 2 output ports

Address mapping (8 bit size):

1. RAM 0 ... 0xFF
2. `INT_ADDR` 0x104
3. `REG_ADDR` 0x108
4. `TIMER_ADDR` 0x10C
5. `UART_ADDR` 0x110

2) INT_ADDR:

- `MASK/SOURCE [addr = 0x104]`
- `CLEAR[addr = 0x105]`

Bits:

- [3..0] = External interrupt input
- [4] = OVERRUN UART interrupt
- [5] = RX UART interrupt
- [6] = TX UART interrupt
- [7] = TIMER interrupt

`3)REG_ADDR:`

- `PORT0_IN/OUT [addr = 0x108]`
- `PORT1_IN/OUT [addr = 0x109]`

`4) TIMER_ADDR (input = current count, output = most significative byte counter):`

- prescaler register `[addr = 0x10C]`
- status register `[addr = 0x10D]`

`5) UART_ADDR`

- `TX/RX DATA REGISTER [addr = 0x110]`
- `BAUD RATE REGISTER [addr = 0x111]`

Used interrupt and indirect addressing support.

Stack = internal, deep = 8 (see Help/CpuCore: How to set stack deep)

Code structure:

1) Initialize parameters
2) Main Loop: indirect addressing operations.
3) ISR: check all interrupt sources (with priority order) and clear them.

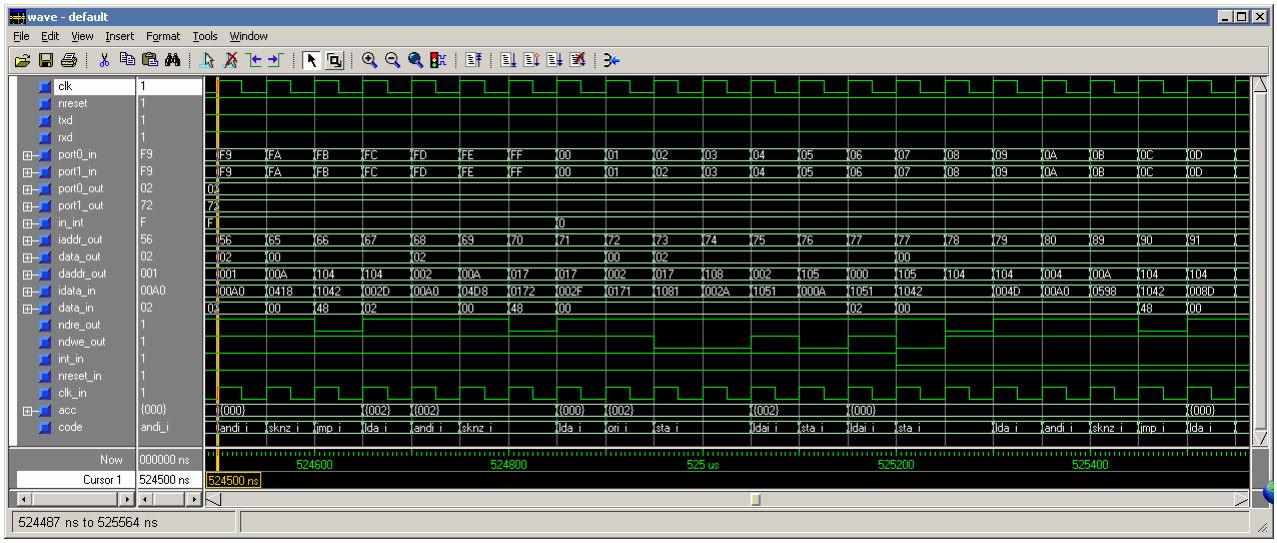
NOTE: The ISR address = num. of Instruction address – 8 => (10 bit Instruction address) = > 0x400h – 8 = 0x`3f8h`

Now choose “ASM” to view the source code, “HW Build” to generate the VHDL CPU core and “SW Build” to generate Ram /Rom files. The “Log” menu will show the software compilation report.

If the compilation result is correct, we can then start the VHDL simulation. The current testbench shows the signals behaviour for a test program.

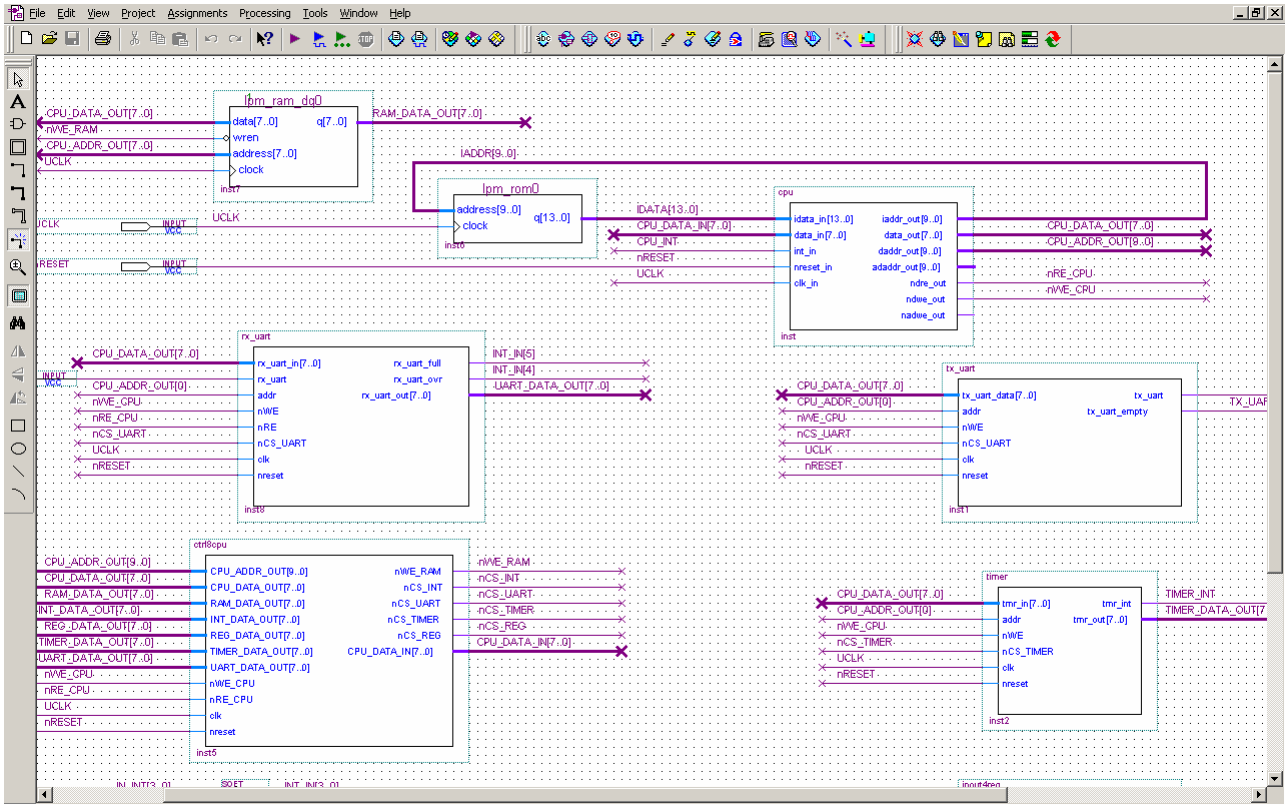
Run ModelSim and:

1) File\Open\Project select the path for cpu8bit\Testbench\testbench_apre.mpf (or testbench_xpre.mpf)
2) Compile\Compile All (MACRO: compile_apre.do/compile_xpre.do)
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ps
4) Simulate\Simulate Options: Default Run: 2 ms
5) View\All windows
6) In the wave window select: File\Load Format\Wave_pre.do
7) Simulate\Run: Run 2 ms (MACRO: sim_pre.do)



Altera Design:

Start cpu8bit.quartus in cpu8bit\Altera directory.

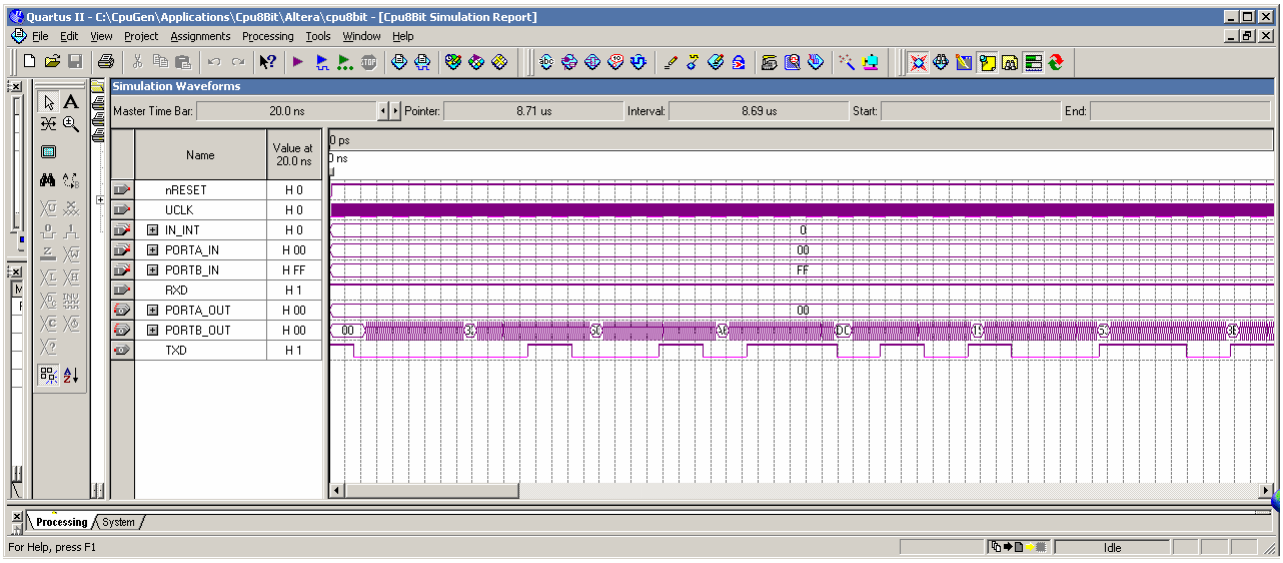


EAB (Embedded Array Blocks) are used for ram (ram.mif) and rom (rom.mif).
NOTE: both ram e rom EAB have to be created with single clock and q output port not registered.
In Assignments\Device set CYCLONE device EP1C3T100C8.
Then with Processing\Start Compilation Quartus synthetize the design.

Output:

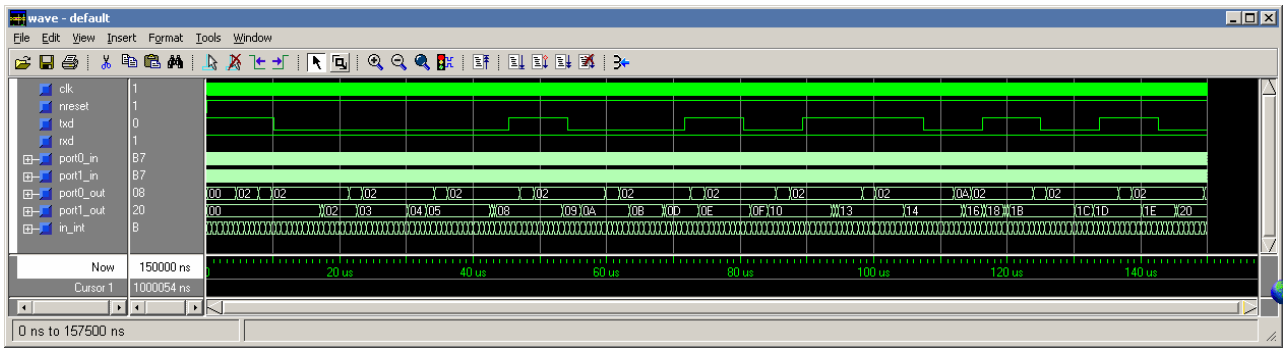
- 935 / 2910 (32%) = Logic Elements
- 16384 / 59904 (27%) = Memory bits
- 44.12 MHz = Max frequency clock

1) Simulate using Quartus Waveform simulator:



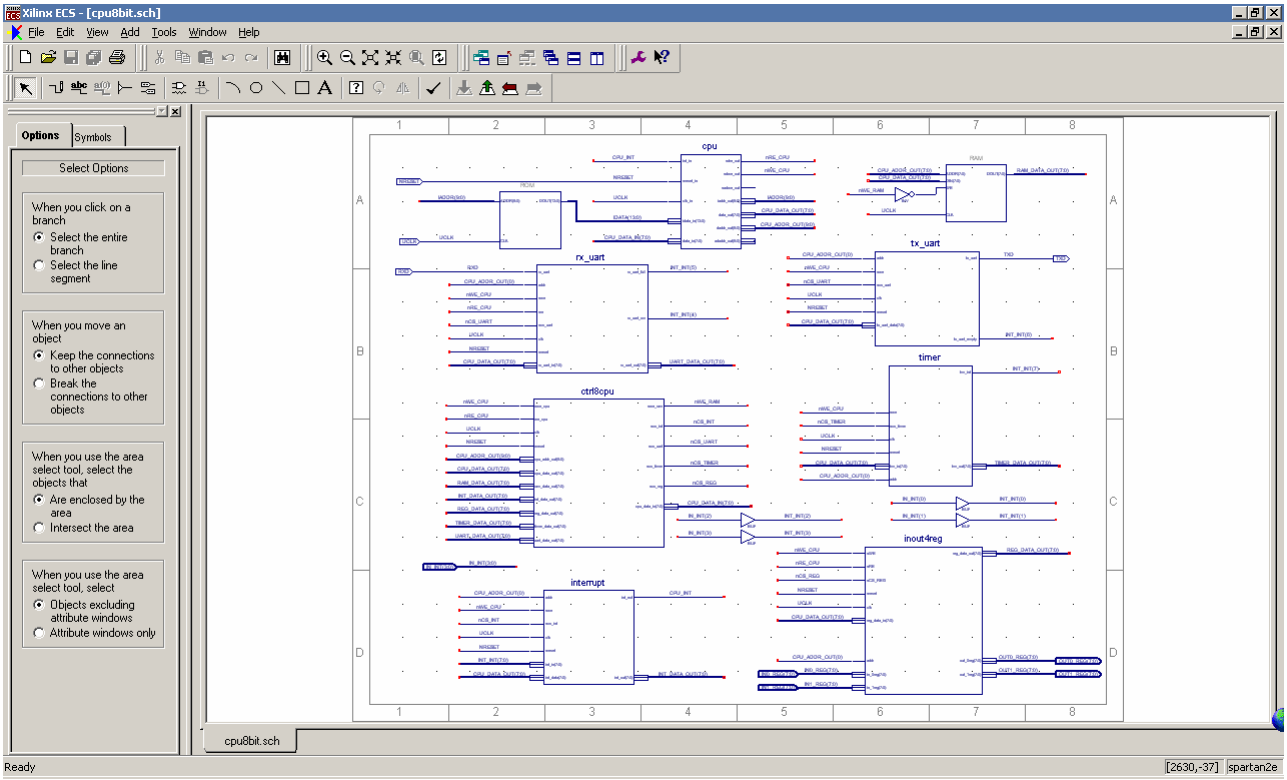
2) Simulate using Modelsim AE: (cpu8bit.vho = Quartus output)

1) File\Open\Project select the path for cpu8bit\Testbench\testbench_apost.mpf.
2) Compile\Compile All (MACRO: compile_apost.do)
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ps
4) Simulate\Simulate Options: Default Run: 10000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_post.do
7) Simulate\Run: Run 100 us (MACRO: sim_post.do)



Xilinx Design:

Start cpu8bit.npl in cpu8bit\Xilinx directory.



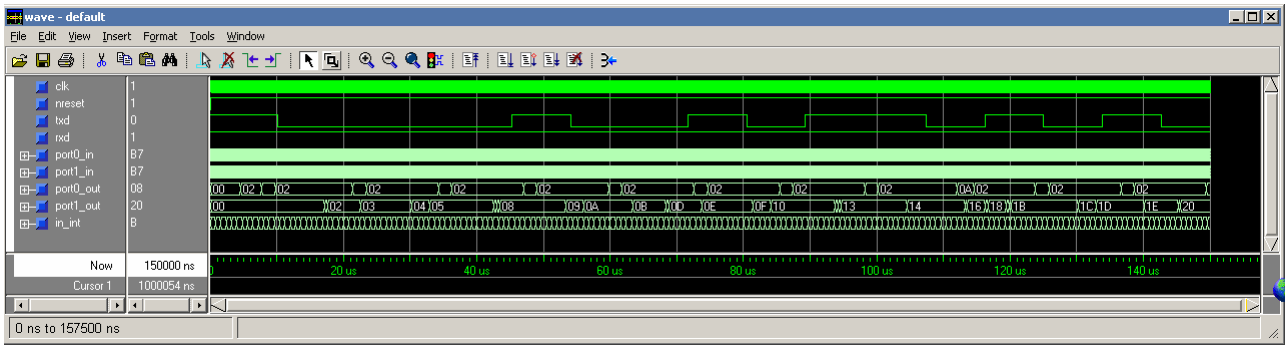
Memory Blocks are used for ram (ram.coe/ram.cfg) and rom (rom.coe/rom.cfg).
NOTE: It's necessary to update initialization files if you use a different project path directory (rom/ram.xco)
Remember to Regenerate Core, Update schematic and regenerate cpu8bit_timesim.vhd output when change hw/sw cpu.
I used a Spartan2 device: XC2S50e-7tq144
Then, select cpu8bit and Generate Programming file.

Output:

- Total Number Slice Registers: 377 out of 1,536 24%
- Total Number 4 input LUTs: 947 out of 1,536 61%
- Number of occupied Slices: 562 out of 768 73%
- Clock Period 38.376 ns: Max Frequency = 26.058 MHz

1) Simulate using Modelsim XE: (cpu8bit_timesim.vhd = Xilinx output)

1) File\Open\Project select the path for cpu8bit\Testbench\testbench_xpost.mpf.
2) Compile\Compile All
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ps
4) Simulate\Simulate Options: Default Run: 10000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_post.do
7) Simulate\Run: Run 40 us

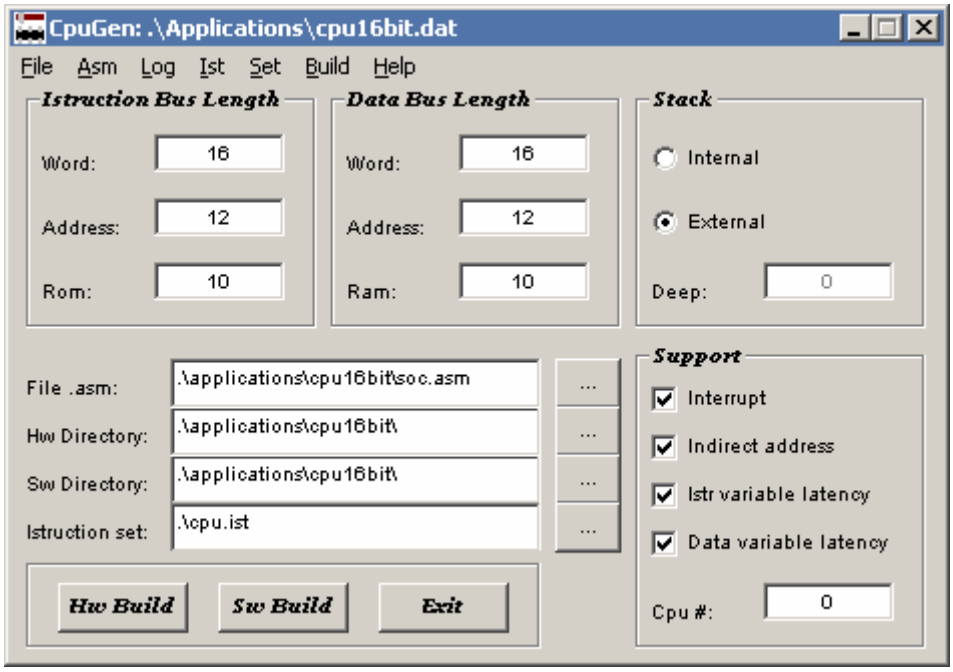


Cpu16Bit

Goal:

- use a 16 bit data/Instruction bus
- use the same memory for code and data (Harvard to Von Neuman Architecture)
- control one external level interrupt source
- control one external slow peripheral memory mapped (wait states)
- implement the stack with vendor dependent memory blocks

Run **cpugen** and load Applications\cpu16bit.dat cpu configuration:



RTL design:

VHDL sources:

1) CPU

- Cpu#_utils.vhd
- Cpu#_iu.vhd
- Cpu#_cu.vhd
- Cpu#_du.vhd
- Cpu#_oa.vhd
- Cpu#_wd.vhd
- Cpu#.vhd

2) COMPONENTS

- Ctrl16cpu.vhd
- H2v.vhd
- Waitstategen.vhd
- Stack_if.vhd
- Utils_pkg.vhd
- Ram.vhd (only for RTL testbench; use rom.vin)
- Sram.vhd (only for RTL testbench)
- Stack_t.vhd (only for RTL testbench)

3) TESTBENCH (RTL)

- Cpu16bit.vhd
- Clock.vhd
- World.vhd
- Testbench_a.vhd

CPU 16 bit => Data Bus Word Length = 16 bit
IBUS = Instruction Words Length – 4 = 12
Instruction Words Length = 16 bit => instruction/data page size = 4096 word = (2 ** IBUS)
Instruction Address Length = 12 bit => number of instruction pages = 1 = (2 ** (12 – IBUS))
Instruction Rom Length = 10 bit => max. number of code words = 1024 = (2 ** 10) [x16]
Data Bus Address Length = 12 bit => number of data pages = 1 = (2 ** (12 – IBUS))
Data Ram Length = 10 bit => max. number of ram words = 1024 = (2 ** 10) [x16]
Data Expansion = 4 bit (IBUS < 16)

Resources:

- 1024 word RAM [x16] for code (512 low) and data (512 high)
- 256 word [x12] of external stack

Address mapping (16 bit size):

1. CODE 0 ... 0x1FF
2. DATA 0…0x1F8
3. INTERRUPT 0x3F8…0x3FF
4. EXTERNAL 0x200 ... 0x3FF

Number of wait states for external access = 7
Used interrupt, indirect addressing, data and instruction variable latency support.
Stack = external, deep = 256 (see Help/CpuCore: How to set stack deep)

Code structure:

1. Read external data
2. Update it
3. Save it and write back
4. ISR: count interrupt rise edges

NOTE:

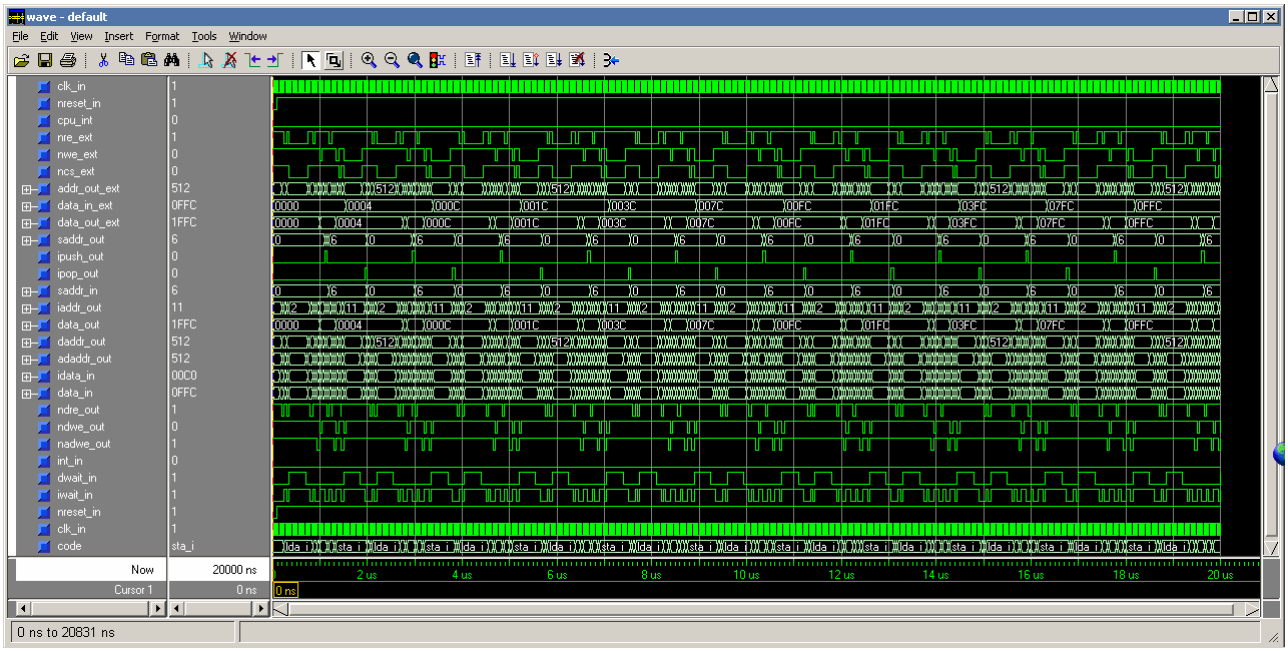
1. The ISR address = num. of Instruction address – 8 => (12 bit Instruction address) = > 0x1000h – 8 = 0xFF8h
2. Using the same memory, for data and code, at the moment it's not possible to initialize static ram data (init file = rom file)

Now go to "ASM" to view the source code, "HW Build" to generate the VHDL CPU core and "SW Build" to generate Ram /Rom files. The "Log" menu shows the software compilation report.

If the compilation ends successfully, we can start the VHDL simulation. The current testbench shows signals behaviour with a test program.

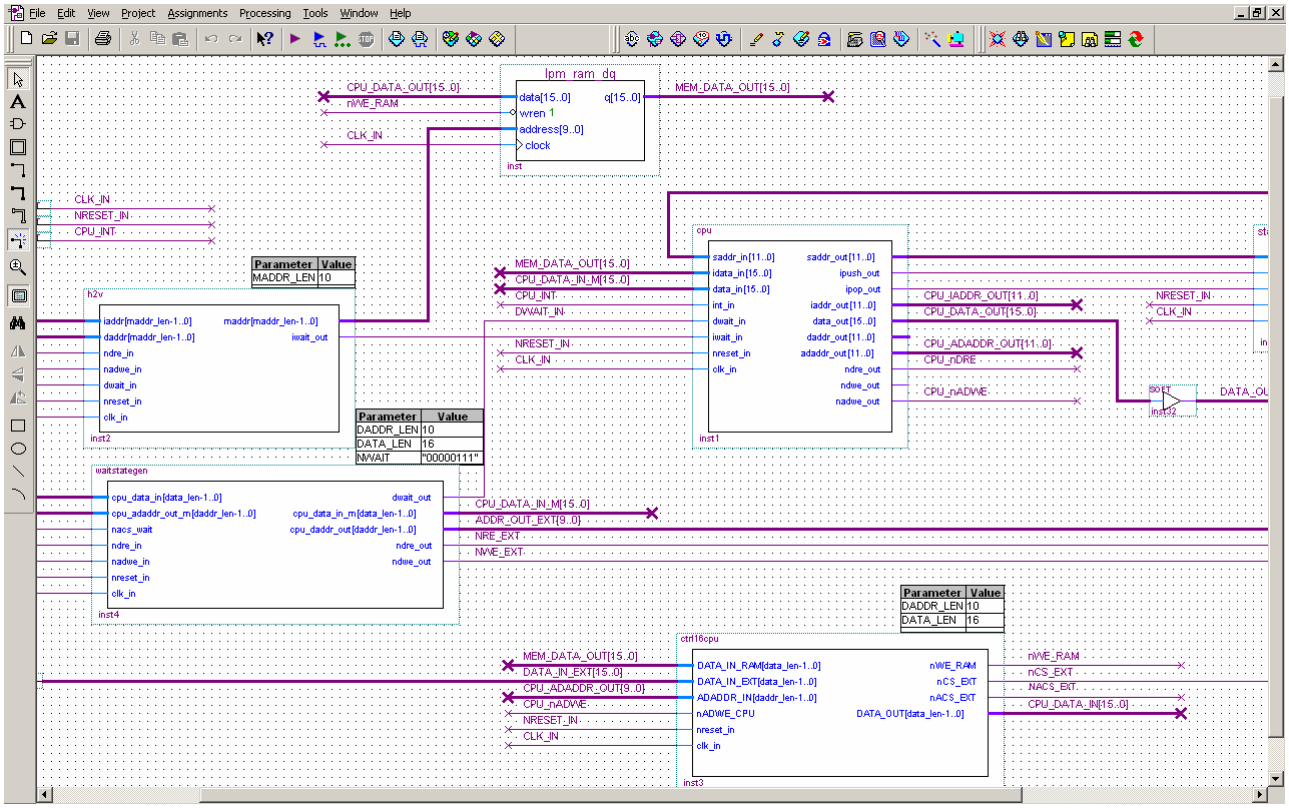
Run ModelSim and:

1) File\Open\Project select the path for cpu16bit\Testbench\testbench_apre.mpf (or testbench_xpre.mpf)
2) Compile\Compile All
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ps
4) Simulate\Simulate Options: Default Run: 20000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_pre.do
7) Simulate\Run: Run 20 us



Altera Design:

Start cpu16bit.quartus in cpu16bit\Altera directory.

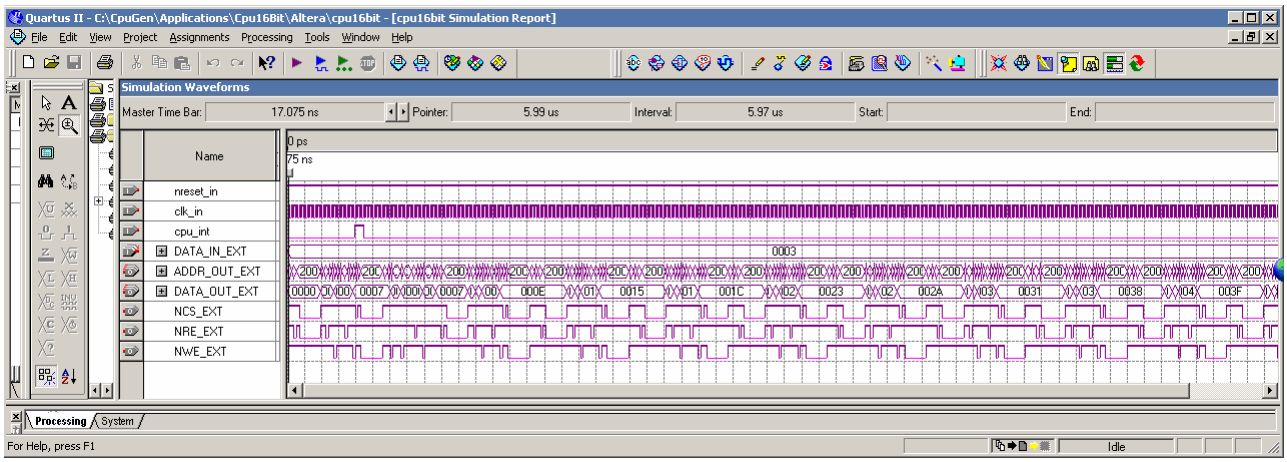


EAB (Embedded Array Blocks) are used for ram data/code (rom.mif) and stack.
NOTE: both ram EAB have to be created with single clock and q output port not registered.
In Assignments\Device set CYCLONE device EP1C3T100C8.
Then with Processing\Start Compilation Quartus synthetize the design.

Output:

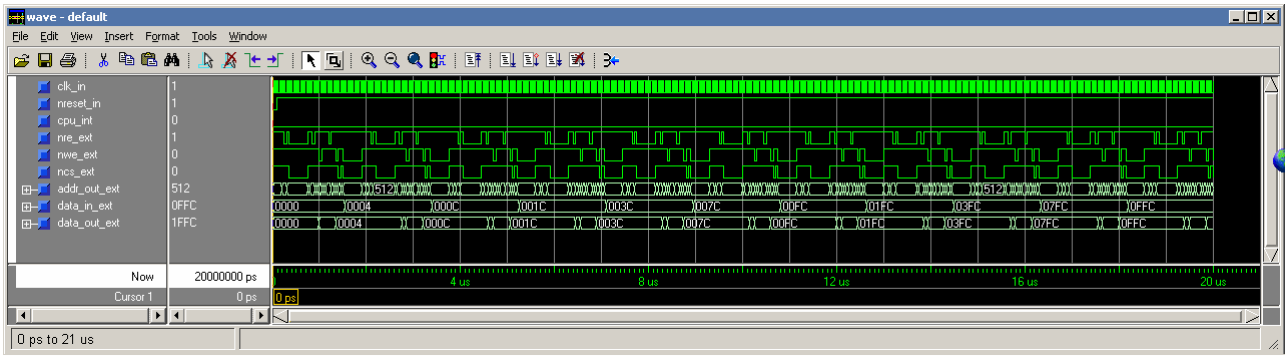
- 739 / 2910 (25%) = Logic Elements
- 18944 / 59904 (31%) = Memory bits
- 46.49 MHz = Max frequency clock

1) Simulate using Quartus Waveform simulator:



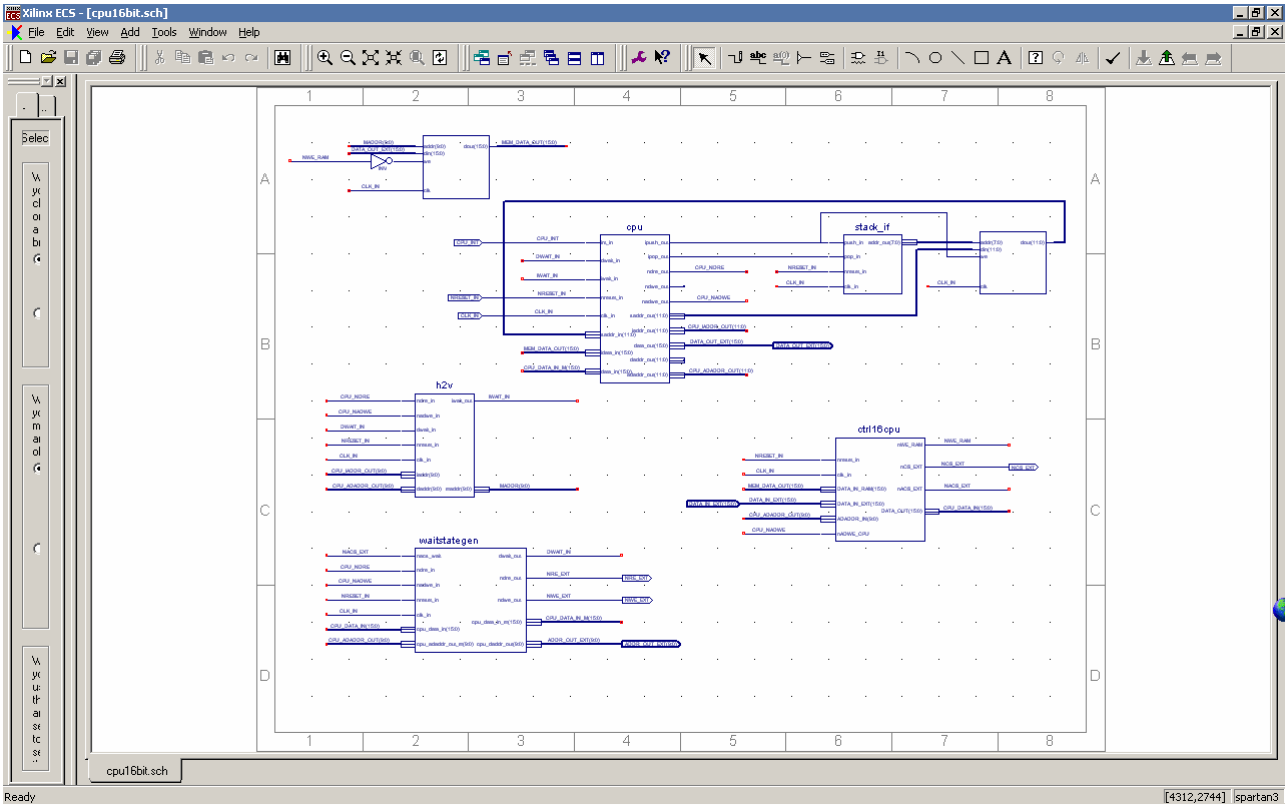
2) Simulate using Modelsim AE: (cpu16bit.vho = Quartus output)

1) File\Open\Project select the path for cpu16bit\Testbench\testbench_apost.mpf.
2) Compile\Compile All (MACRO: compile_apost.do)
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ps
4) Simulate\Simulate Options: Default Run: 20000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_post.do
7) Simulate\Run: Run 20 us (MACRO: sim_post.do)



Xilinx Design:

Start cpu16bit.npl in cpu16bit\Xilinx directory.



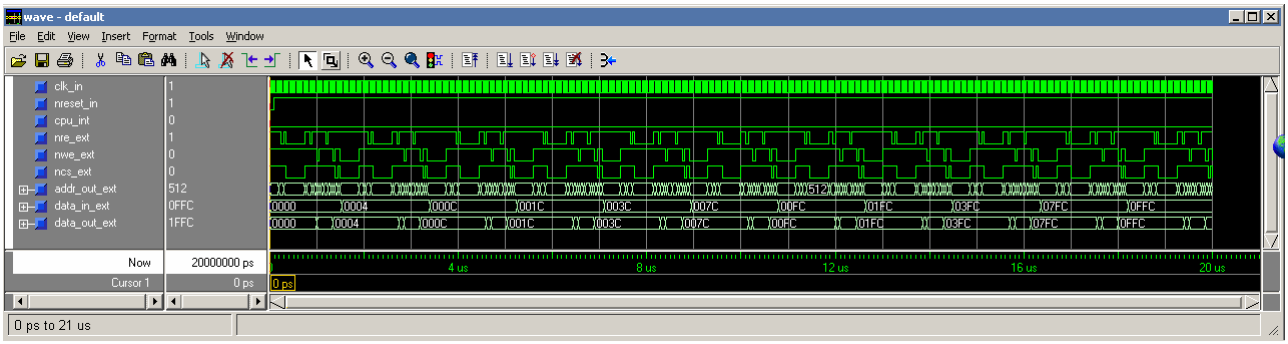
Memory Blocks are used for ram (rom.coe/rom.cfg).
NOTE: It's necessary to update initialization files if you use a different project path directory (rom.xco)
Remember to Regenerate Core, Update schematic and regenerate cpu16bit_timesim.vhd output when change hw/sw cpu.
I selected the Spartan3 device: XC3S50-4pq208
Then select cpu16bit and Generate Programming file.

Output:

- Total Number Slice Registers: 244 out of 1536 15%
- Total Number 4 input LUTs: 1012 out of 1536 65%
- Number of occupied Slices: 541 out of 78 60%
- Clock Period 19.578ns: Max Frequency = 51.076 MHz

1) Simulate using Modelsim XE: (cpu16bit_timesim.vhd = Xilinx output)

1) File\Open\Project select the path for cpu16bit\Testbench\testbench_xpost.mpf.
2) Compile\Compile All
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ps
4) Simulate\Simulate Options: Default Run: 20000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_post.do
7) Simulate\Run: Run 20 us

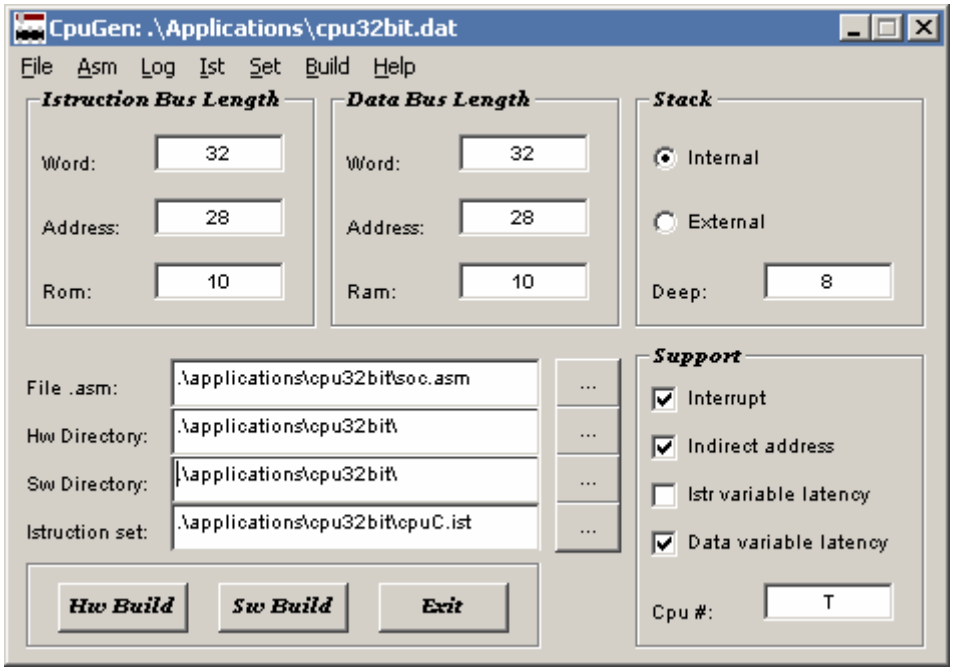


Cpu32Bit

Goal:

- implement a 32 bit coprocessor with custom instructions (Harvard Architecture)

Run **cpugen** and load Applications\cpu32bit.dat cpu configuration:



RTL design:

VHDL sources:

1) CPU

- CpuT_utils.vhd => CpuC_utils.vhd *
- CpuT_iu.vhd => CpuC_iu.vhd
- CpuT_cu.vhd => CpuC_cu.vhd *
- CpuT_du.vhd => CpuC_du.vhd *
- CpuT_oa.vhd => CpuC_oa.vhd
- CpuT_wd.vhd => CpuC_wd.vhd
- CpuT.vhd => CpuC.vhd

2) COMPONENTS

- Utils_pkg.vhd (only for RTL testbench)
- Ram.vhd (only for RTL testbench)
- Rom.vhd (only for RTL testbench)

3) TESTBENCH (RTL)

- Clock.vhd
- World.vhd
- Testbench_a.vhd

CPU 32 bit => Data Bus Word Length = 32 bit
IBUS = Instruction Words Length – 4 = 28
Instruction Words Length = 32 bit => instruction/data page size = 256 Mword = (2 ** IBUS)
Instruction Address Length = 28 bit => number of instruction pages = 1 = (2 ** (28 – IBUS))

Instruction Rom Length = 10 bit => max. number of code words = 1024 = (2 ** 10) [x32]
Data Bus Address Length = 28 bit => number of data pages = 1 = (2 ** (28 – IBUS))
Data Ram Length = 10 bit => max. number of ram words = 1024 = (2 ** 10) [x32]
Data Expansion = 4 bit (IBUS < 32)

Use Cpu ID =T as a template Cpu to implement a custom one (ID = C):

- insert new instructions:
  - SUM WITH SIGN
  - SUB WITH SIGN
  - MUL WITH/WITHOUT SIGN (64 bits result)
  - SHIFT LEFT/RIGHT OF N BITS
  - MOVE HIGH WORD MUL RESULT TO ACC
  - SKIP IF OVERFLOW
- change encoding instructions from 4 to 5 bits => Data Expansion = 5 bit
- update assembler instructions (CpuC.ist) *

The CpuC needs 2 more internal registers (ACC_NUM = 3):

- HIGH WORD MUL RESULT
- OVERFLOW FLAG BIT

Compare * file to see the differences:

- CpuT_utils.vhd with CpuC_utils.vhd
- CpuT_cu.vhd with CpuC_cu.vhd
- CpuT_du.vhd with CpuC_du.vhd
- Cpu.ist with CpuC.ist

Resources:

- 1024 word ROM [x32] for code
- 1024 word RAM [x32] for data

Code structure:

Compute max. N so that N! < (2 ** 32)

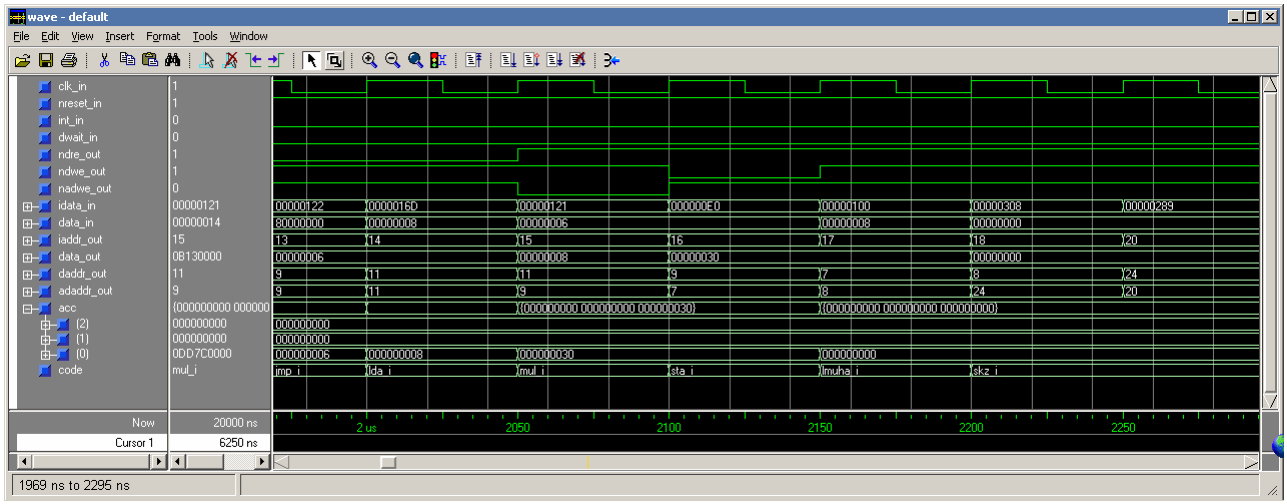
Note: instead of ram/rom block can be used dual port ram to load/save data/code from an external processor (ram.bin, rom.bin are the formats to use).

Now go to "ASM" to view the source code, "HW Build" to generate the Template VHDL CPU core and "SW Build" to generate Ram /Rom files. The "Log" menu shows the software compilation report.

If the compilation was successful, we can start the VHDL simulation. The current testbench shows the signals behaviour with a test program.

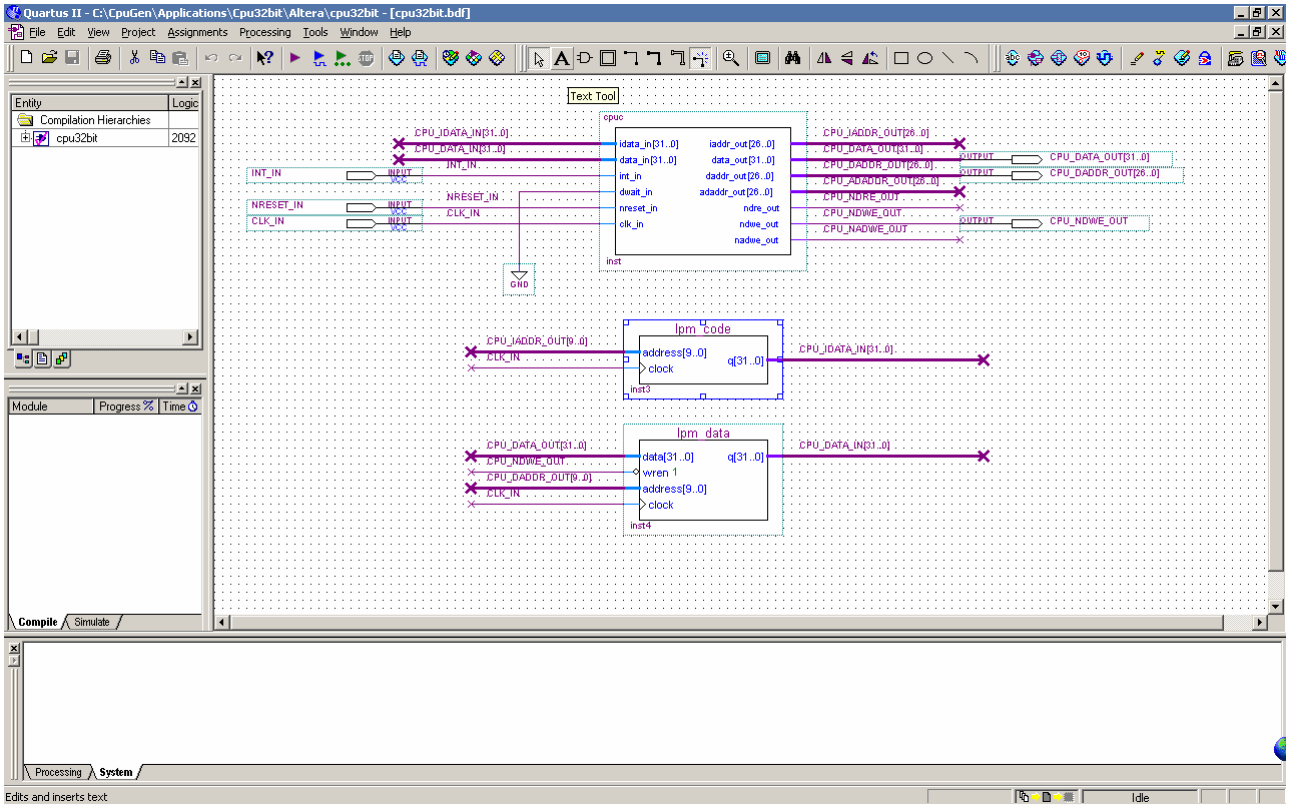
Run ModelSim and:

1) File\Open\Project select the path for cpu32bit\Testbench\testbench_apre.mpf (or testbench_xpre.mpf)
2) Compile\Compile All
3) Simulate\Simulate: Load work\testbench\struct; Simulator resolution: ps
4) Simulate\Simulate Options: Default Run: 10000 ns
5) View\All windows
6) In the wave window select: File\Load Format\Wave_pre.do
7) Simulate\Run: Run 10 us



Altera Design:

Start cpu32bit.quartus in cpu32bit\Altera directory.



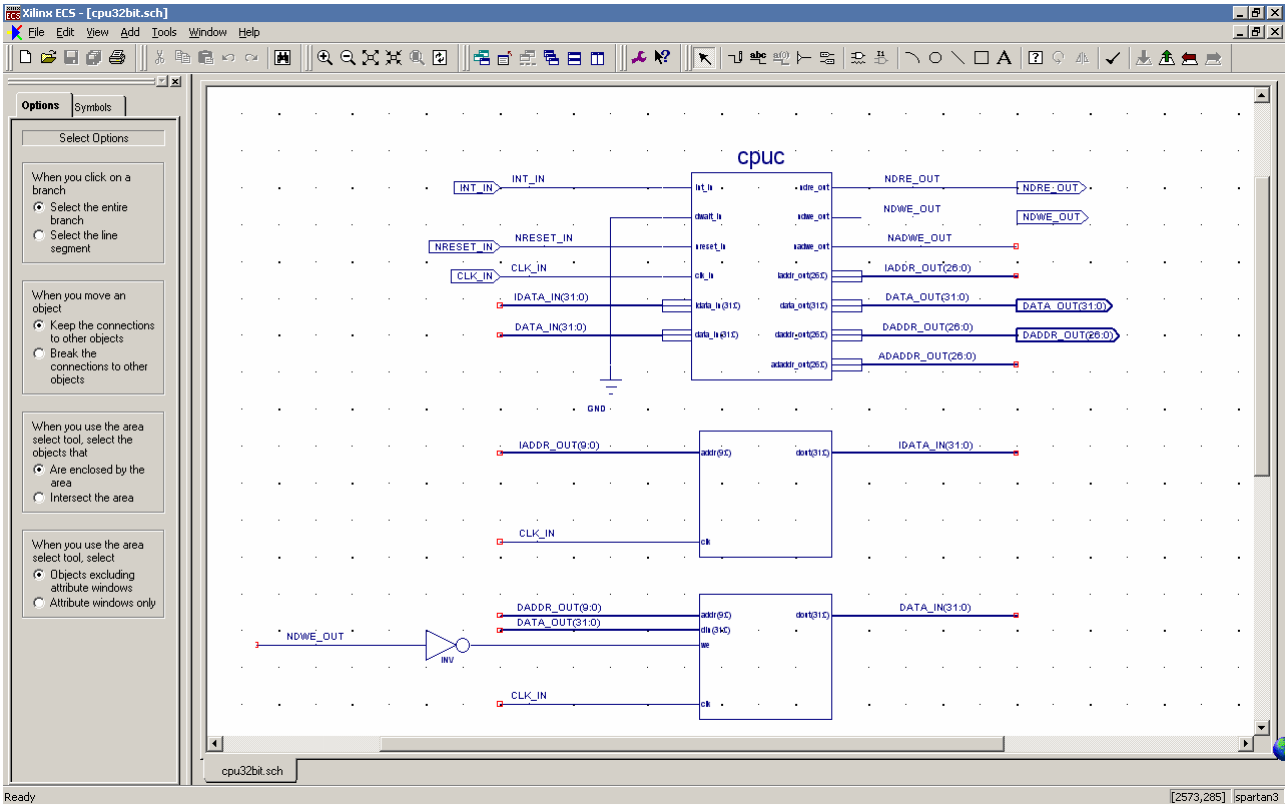
EAB (Embedded Array Blocks) are used for ram data (ram.mif) and code (rom.mif).
NOTE: both ram EAB have to be created with single clock and q output port not registered.
In Assignments\Device set Stratix device EP1S10F780C6.
Then with Processing\Start Compilation Quartus synthetize the design.

Output:

- 2092 / 10570 (19%) = Logic Elements
- 65536 / 920448 (7%) = Memory bits
- 38.00 MHz = Max frequency clock

Xilinx Design:

Start cpu32bit.npl in cpu32bit\Xilinx directory.



Memory Blocks are used for ram (ram.coe/ram.cfg) and rom (rom.coe/rom.cfg).
NOTE: It's necessary to update initialization files if you use a different project path directory (ram/rom.xco)
Remember to Regenerate Core, Update schematic and regenerate cpu32bit_timesim.vhd output when change hw/sw cpu.
I selected the Spartan3 device: XC3S200-4pq208
Then select cpu32bit and Generate Programming file.

Output:

- Total Number Slice Registers: 602 out of 3840 15%
- Total Number 4 input LUTs: 2962 out of 3840 77%
- Number of occupied Slices: 1577 out of 1920 82%
- Clock Period 27.569ns: Max Frequency = 30.469 MHz

NiosApex

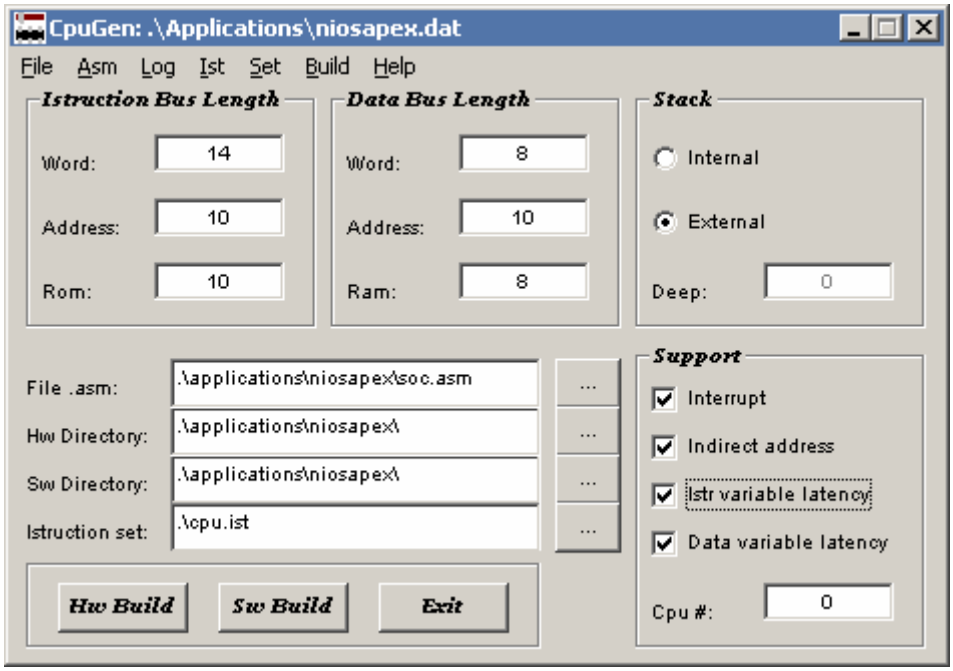
Goal:

- Test design for Nios APEX EP20K200EFC484-2X evaluation board

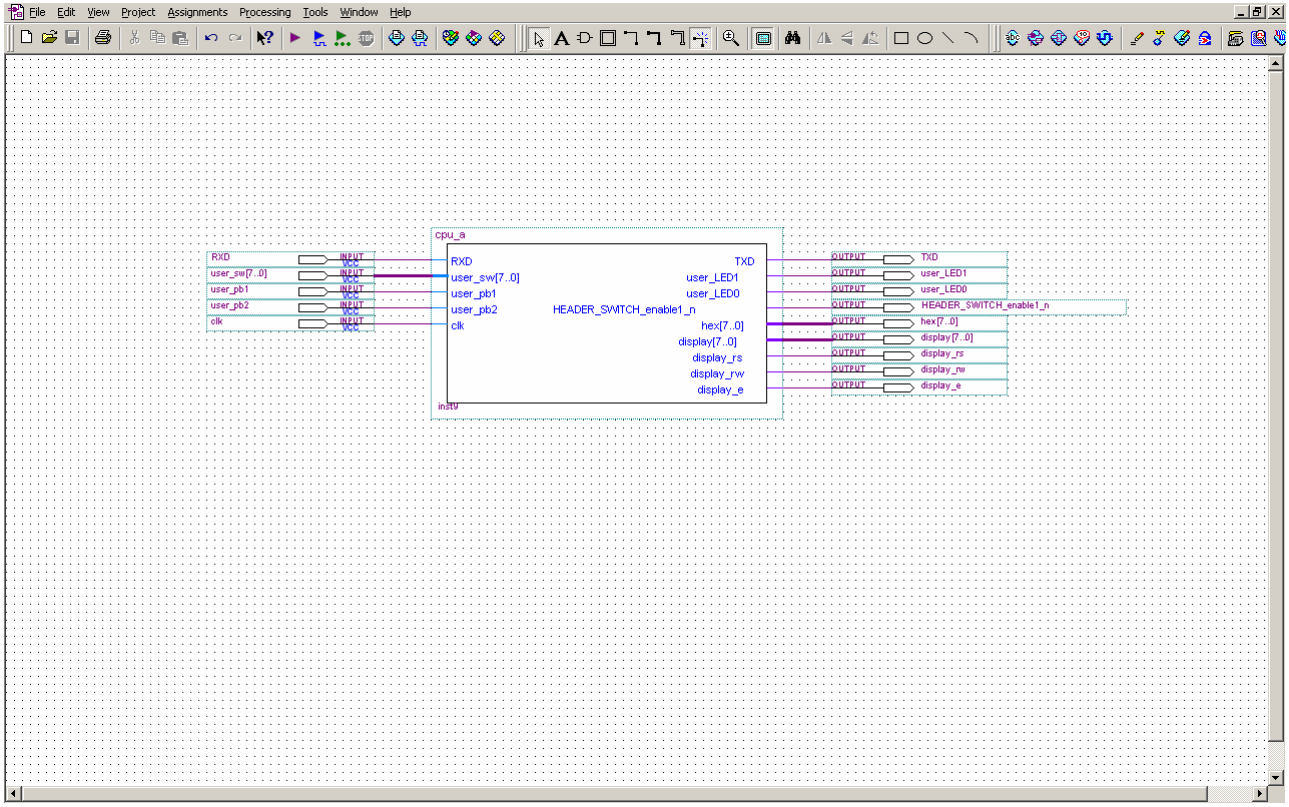
Features:

- 8 bit data/instruction cpu
- Von Neuman Architecture
- data bus with wait states
- interrupt controller
- 1 timer
- 1 uart
- LCD 2x16

Run **cpugen** and load Applications\niosapex.dat cpu configuration:



Start NiosApex.quartus in Applications\NiosApex directory.



1) connect an LCD display 2x16 to the board
2) connect a serial cable (115200,8,N,1) and run a terminal program (ex. Hyperterminal)
3) connect the byteblaster to the jtag/parallel port
4) turn on the board
5) with Tools/programmer load the binary (niosapex.sof)

Program:

1) “Hello” string should appear on the terminal, then each character sent will be received + 1
2) an interrupt timer counter will send an hexadecimal byte to the LCD about each 0.5 sec

Bibliography:

1) VHDL - CookBook: P.Ashenden
2) VHDL - Primer: J.Bhasker
3) Digital Design and Modeling with VHDL and Synthesis: K.C.Chang
4) Component Design By Example: Ben Cohen
5) VHDL and AHDL Digital System Implementation: Frank Scarpino
6) ARM system-on-chip architecture (second edition): Steve Furber
7) Computer Architecture, a quantitative approach (third edition): Hennessy & Patterson
8) PIC16/17 microcontroller DataBook
9) C Language: Kernighan & Ritchie

Legal Notice