# Revision Log

|  |  |  |
| --- | --- | --- |
| Date | Revision | Change |
| 2014-12-15 | 1.00 | Initial revision |

Table of Contents

[Revision Log 1](#_Toc406448173)

[Description 2](#_Toc406448174)

[Project Directory Structure 2](#_Toc406448175)

[Environment 2](#_Toc406448176)

[ModelSim simulations 2](#_Toc406448177)

[Module simulations 2](#_Toc406448178)

[Top-level simulation 2](#_Toc406448179)

[Verification 3](#_Toc406448180)

[Fuse tests 3](#_Toc406448181)

[Z80 Assembly level tests 3](#_Toc406448182)

[Tools 3](#_Toc406448183)

[Integration 3](#_Toc406448184)

[Sample implementations 3](#_Toc406448185)

[Simple computer 4](#_Toc406448186)

[Sinclair ZX Spectrum 4](#_Toc406448187)

# Introduction

A-Z80 is a conceptual implementation of the venerable Zilog® Z80 processor targeted to synthesize and run on a modern FPGA device. It differs from the existing Z80 implementations in that it is designed from the ground-up through the schematics and low-level gates.

This design is capable of mimicking the actual Z80 CPU and it illustrates its inner workings.

The A-Z80 implementation strives to be internally structurally identical to the original Z80. Using this approach the model achieves a full cycle accuracy and has identical behavior for all documented and undocumented features (\*) not by explicitly hard-coding them but by mimicking their actual design.

Zilog Z80 CPU references are widely available so the CPU itself will not be described here.

You can read more about the conception and implementation of the A-Z80 on its home website: [www.baltazarstudios.com](http://www.baltazarstudios.com) .

This document focuses on the structure and mechanics of the project; it should help you understand it and incorporate it into your designs.

# Project Directory Structure

The project can be downloaded at OPENCORES as a SVN repo here: <http://opencores.org/project,a-z80> and also on Bitbucket here: <https://bitbucket.org/gdevic/a-z80> .

The following table describes the hierarchical directory structure.

|  |  |  |
| --- | --- | --- |
| Directory | Sub-directory | Description |
| cpu |  | Contains all core files for A-Z80 CPU |
|  | alu | Arithmetical and Logical Unit files |
|  | bus | Various bus-related files |
|  | control | Control unit files |
|  | registers | Register block files |
|  | toplevel | A-Z80 top level interfaces and projects |
| docs |  | Documentation and schematic images |
| host |  | Two implementations using the A-Z80 on Altera DE1 FPGA |
|  | basic | Basic computer containing UART mainly for testing and verification |
|  | zxspectrum | Sinclair ZX Spectrum implementation |
| resources |  | General project resources and scripts |
| tools |  | Build and testing utilities and misc. files |
|  | Arduino | Software for Arduino Mega dongle to interface with a Z80 |
|  | dongle | Results from running a dongle and simulation golden files |
|  | z80\_pla\_checker | Windows utility to test and create A-Z80 PLA tables |
|  | zmac | Z80 test and verification assembler files |

# Environment

Minimal set of tools needed to compile various parts of the project are:

* Altera Quartus II Web Edition (Free)
* ModelSim (Altera edition) – only if you would like to run module simulation (Free)
* Python 2.7 – only if you will change and compile CPU modules. All necessary files needed to include A-Z80 sources in your own project are included (Free)
* Microsoft Visual Studio 2010 SP1 – only if you want to recompile the z80\_pla\_checker tool yourself. This is normally not needed since the sources and precompiled executable are checked in with the project.

This project is developed and tested on a Windows 7 OS. Your mileage with it may vary if you use Linux.

All designs in this project are tested on the Altera FPGA DE1 board: <http://www.altera.com/education/univ/materials/boards/de1/unv-de1-board.html>



This board hosts Cyclone II EP2C20F484C7 and a number of useful peripherals including 512 KB SRAM.

However, Verilog files that comprise A-Z80 should be synthesizable for other vendors such as Xilinx and their tool chains.

# ModelSim simulations

## Module simulations

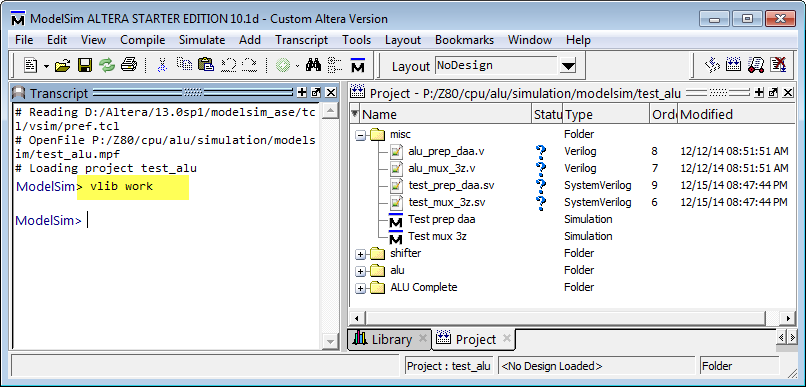
Each module in the “cpu” directory contains a ModelSim simulation project that verifies the functionality of each more important block. Before opening any project in ModelSim, you need to run “**modelsim\_setup.py**” script located in the project root directory. That script will set up relative file mappings to enable project to reside anywhere on your drive.

You may not need to run simulations since the project has already been tested. Should you want to do it, this chapter will show you how.

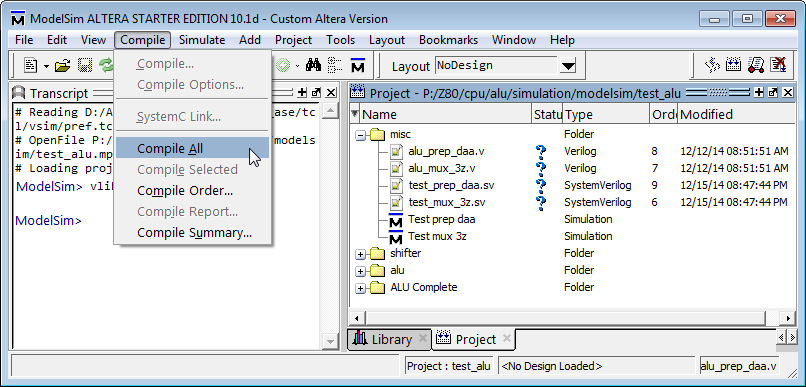
If you have installed and configured ModelSim properly, double-clicking on any **\*.mpf** file will open a project.

This particular example will illustrate setting up and starting a simulation in the **alu** module. Each module has its own ModelSim project.

Before you can compile simulation files, you need to create a library by typing “**vlib work**” as shown:



Next, select “**Compile->Compile All**” to compile all files that are part of a module simulation.



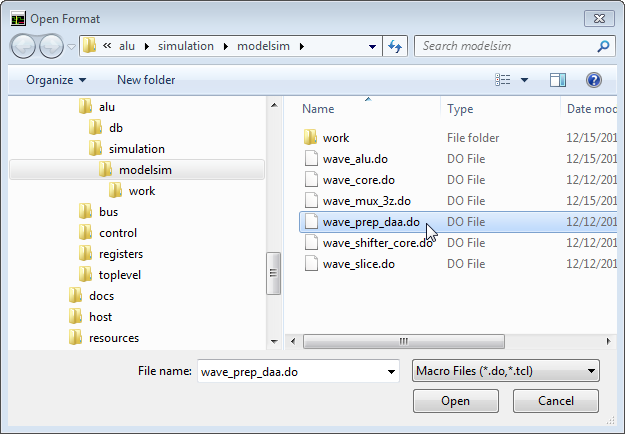
Each project has one or more Simulation Configurations, each configuration testing a specific block of logic. In addition, each configuration has its own wave file which you can load before you run a simulation. Wave files are customized for specific test.

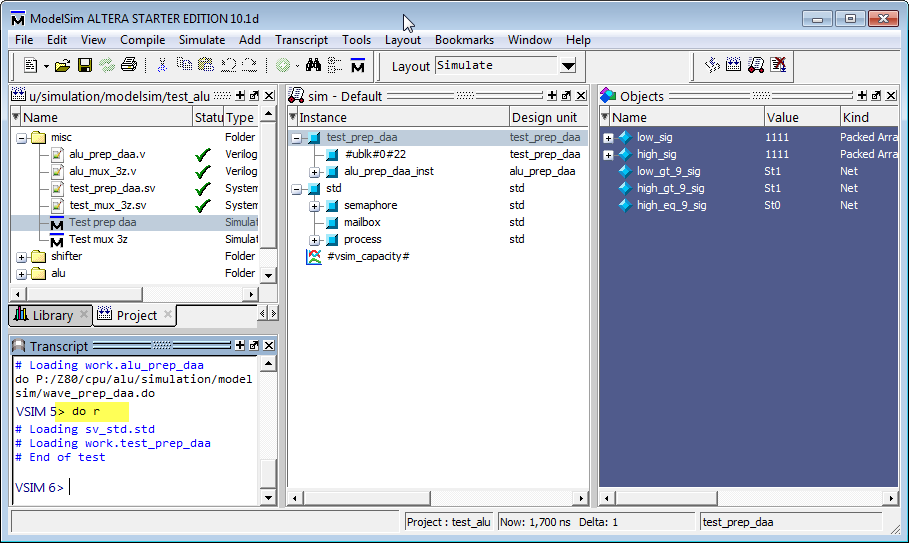
In this example, we will run “Test prep daa” configuration. DAA is a Z80 instruction that adjusts accumulator for a decimal operation. It requires calculating the adjustment addend based on the result of a previous operation. Hence, this test is written to verify the correctness of that calculation.

Each test configuration has a main test file – always written in *System Verilog*, a file with the extension **\*.sv,** to run the test. The file to run “Test prep daa” configuration is “**test\_prep\_daa.sv**”.

Double-click on the “Test prep daa” configuration and your simulation should be loaded.

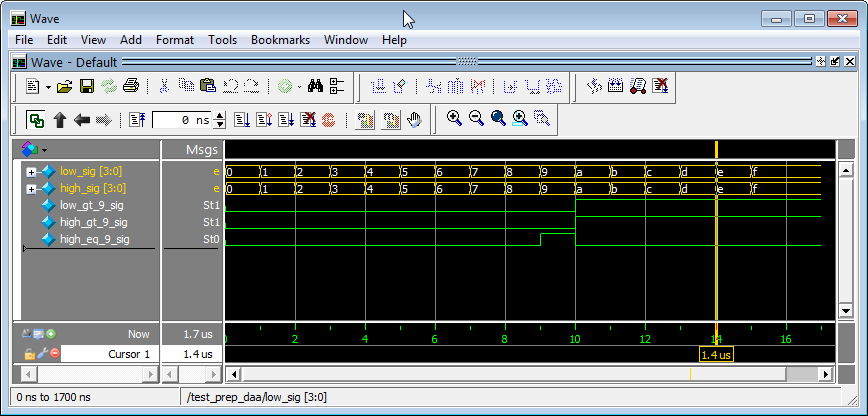
Open the wave window if it is not already visible and select File->Open to load a wave file as shown:





There is another shortcut to run a simulation: each ModelSim directory contains a small file with the name “r” that contains command “**restart -f ; run -all**”. Run, or rerun, a test simply by typing “**do r**” as shown above.

After running this particular example, you should see the waveform of the DAA preparation block:



This is a very simple example, but the method of running simulations on other configurations and modules is the same.

Each main test file (for example “**test\_prep\_daa.sv**” contains a set of **assert()** statements to verify signal correctness.

Most simulations run for the predetermined number of clocks. The exceptions are top-level simulations (in the directory “**cpu\toplevel\simulation\modelsim**”) and basic host simulation (in the directory “**host\basic\simulation\modelsim**”). These simulations need to be stopped manually since they simply continue to execute given Z80 executable code.

## Top-level simulations

Two top-level simulations are designed to load an arbitrary Z80 assembly level code and execute it. A simple unidirectional UART model is provided to dump the output to the ModelSim console as well as to simulate the behavior of a synthesized design when run on the actual DE1 hardware.

|  |  |
| --- | --- |
| Module | Simulation project |
| Toplevel | cpu\toplevel\simulation\modelsim\test\_top.mpf |
| Basic host | host\basic\simulation\modelsim\test\_host.mpf |

Those simulation configurations can run any Z80 code, and several sample tests can be found in the directory “**tools\zmac**” along with the ZMAC assembler and few batch scripts that simplify the compilation and test setup.

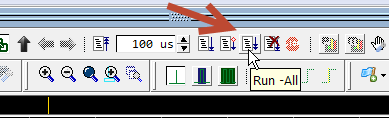
For this example, we will compile and run “Hello, world” test. Its source file is “**tools\zmac\hello\_world.asm**”. (Those Z80 test files are roughly based on the CP/M and BDOS interface for text printing)

Two MS DOS batch files are used to compile and run any of the tests in that folder (you can also create and run your own tests as well):

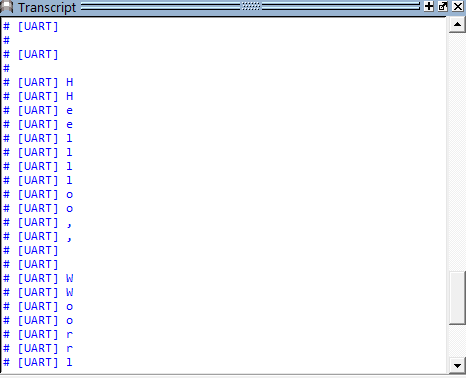
|  |  |
| --- | --- |
| Batch file | Description |
| tools\zmac\make\_modelsim.bat | Compiles and generates executable code for a ModelSim test at “cpu\toplevel\simulation\modelsim\test\_top.mpf”, for “test\_top” configuration. |
| tools\zmac\make\_fpga.bat | 1. Compiles and generates executable code in Intel HEX file format to be included into the target FPGA data file for basic host “host\basic\ host\_board.qpf” 2. Also generates executable code for the basic host ModelSim test at “host\basic\simulation\modelsim\ test\_host.mpf” |

You can simply drag and drop an assembly file (\*.asm) into one of the batch files and it will do compile them and copy them to proper directory after which you only need to recompile a project.

Drag and drop “**hello\_world.asm**” onto the “**make\_modelsim.bat**” and start the simulation. Shortly you should see the output.



After you see the text being written to the UART, you can stop the simulation.



# Verification

## Fuse tests

Fuse is a set of tests to verify Z80 instruction behavior.

## Z80 Assembly level tests

Folder tools/zmac contain several Z80 assembly level tests.

# Tools

PLA checker tool is a test utility to verify and create…

# Integration

This section describes how to integrate the A-Z80 CPU into your own project.

The method is tested with Altera design tools (Quartus), but it should be relatively easy for someone skilled in the art to use other vendor such is Xilinx.

The integration involves adding all relevant source files. This table lists them:

|  |
| --- |
| cpu/alu/alu\_slice.v  cpu/alu/alu\_shifter\_core.v  cpu/alu/alu\_select.v  cpu/alu/alu\_prep\_daa.v  cpu/alu/alu\_mux\_8.v  cpu/alu/alu\_mux\_4.v  cpu/alu/alu\_mux\_3z.v  cpu/alu/alu\_mux\_2z.v  cpu/alu/alu\_mux\_2.v  cpu/alu/alu\_flags.v  cpu/alu/alu\_core.v  cpu/alu/alu\_control.v  cpu/alu/alu\_bit\_select.v  cpu/alu/alu.v  cpu/bus/bus\_switch.sv  cpu/bus/inc\_dec\_2bit.v  cpu/bus/inc\_dec.v  cpu/bus/data\_switch\_mask.v  cpu/bus/data\_switch.v  cpu/bus/data\_pins.v  cpu/bus/control\_pins\_n.v  cpu/bus/bus\_control.v  cpu/bus/address\_pins.v  cpu/bus/address\_latch.v  cpu/bus/address\_mux.v  cpu/control/sequencer.v  cpu/control/resets.v  cpu/control/ir.v  cpu/control/interrupts.v  cpu/control/decode\_state.v  cpu/control/clk\_delay.v  cpu/control/pin\_control.v  cpu/control/pla\_decode.sv  cpu/control/memory\_ifc.v  cpu/control/execute.sv  cpu/registers/reg\_latch.v  cpu/registers/reg\_file.v  cpu/registers/reg\_control.v  cpu/toplevel/z80\_top\_direct\_n.sv |

In addition, any one of the two sample implementations (*basic host* and *zxspectrum*) provide a good starting point and a working example.

# Pinout

The top-level file “**cpu\toplevel\z80\_top\_direct\_n.sv**” exports the following interface:

|  |
| --- |
| module z80\_top\_direct\_n(  output wire nM1,  output wire nMREQ,  output wire nIORQ,  output wire nRD,  output wire nWR,  output wire nRFSH,  output wire nHALT,  output wire nBUSACK,  input wire nWAIT,  input wire nINT,  input wire nNMI,  input wire nRESET,  input wire nBUSRQ,  input wire CLK,  output wire [15:0] A,  inout wire [7:0] D  ); |

The pinout is 100% identical to the Zilog Z80 package; bus timings also implement tri-state signaling. (While this is admittedly not optimal for an FPGA implementation, the goal of the project is to mimic the actual Z80 silicon).

Your system design should simply include all necessary files and instantiate a “**z80\_top\_direct\_n**” module.

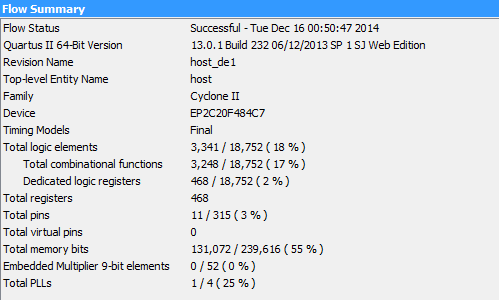
# Sample implementations

Two working sample implementations are included.

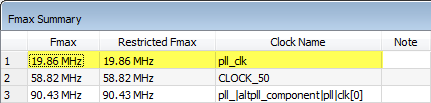
Warning: The synthesis and fMax results as shown might vary depending on the tool versions, applied timing constraints and the exact configuration.

## Simple host

This is the synthesis result of a simple host design on Altera DE1 board:

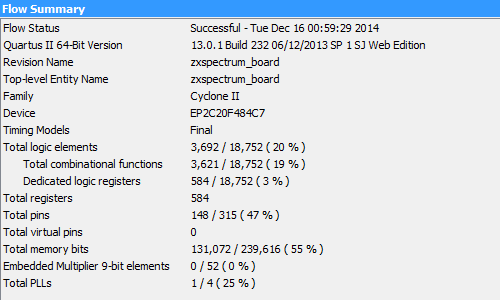


Since the CPU CLK is derived from the *pll\_clk*, the effective A-Z80 fmax for this compilation is 19.86 MHz.



## Sinclair ZX Spectrum

This is the synthesis result of a Sinclair ZX Spectrum host design on Altera DE1 board:



The effective A-Z80 *clk\_cpu* fmax for this compilation is 10.65 MHz.

