To simulate a Verilog design before implementation, it is necessary to use a digital logic simulator. The simulation could be verilog-based *presynthesis* and gate-level *postsynthesis* simulation. These simulators use event-driven simulation to give accurate results.

To model the parallelism and concurrent operation of real circuits, it is necessary to use an *event-driven* (ED) simulator. An event is defined as a logic-value change in a net at some instant in time.
Example

- In a basic ED simulator, a list or a queue is maintained that contains every net for which an event occurred at some instant of time.
- After this event list is built, it is traversed and a new list is built, called the gate queue.
- Whenever an event occurs on an input net to a gate, the simulator must simulate the gate to determine if the gate output net undergoes a corresponding event.
- The event queue and the gate queue are alternately formed and processed and the simulation is over when the queues are empty.
Example of even-driven simulation

[Example from Thornton & Reese]
Example of event-driven simulation

Contrast the output to the output of traditional simulation

[Example from Thornton & Reese]
Presynthesis vs. postsynthesis simulation

(a) Pre-synthesis Verilog functional simulation

The example below is zero-delay as no delays are specified.

always @*
begin
  C = ~B;
  Y = C & A;
end

(b) Post-synthesis simulation of Implementation X

Timing (and glitches!) depend on implementation technology

(c) Post-synthesis simulation of Implementation Y

4x1 Memory
(4 locations, each location has 1 bit)

[Example from Thornton & Reese]
Megafunctions

- Using megafunctions instead of coding your own logic saves valuable design time.

- Megafunctions include the library of parameterized modules (LPM) and Altera device-specific megafunctions.

- QuartusII integrated synthesis, recognize certain types of HDL code and automatically infer the appropriate megafunction.

- For some designs, generic HDL code can provide better results than instantiating a megafunction (e.g., for simple addition or subtraction functions, use the + or – symbol instead of an LPM function. Instantiating an LPM function for simple arithmetic operations can result in a less efficient result because the function is hard coded and the synthesis algorithms cannot take advantage of basic logic optimizations.)
Examples of Megafunctions

MegaWizard Plug-In Manager [page 2a]

Which megafuction would you like to customize?
Select a megafuction from the list below

- Altera SOPC Builder
- Arithmetic
- ALTACCUMULATE
- ALTECC
- ALTFADD_SUB
- ALTF_DIV
- ALTFMULT
- ALTMMULT
- ALTMULT_ACCUM (MAC)
- ALTMULT_ADD
- ALT_SORT
- LPM_ABS
- LPM_ADD_SUB
- LPM_COMPARE
- LPM_COUNTER
- LPM_DIVIDE
- LPM_MULT
- PARALLEL_ADD
- Communications
- DSP
- Error Detection/Correction
- Filters
- Signal Generation
- Transforms

Which device family will you be using?
- Cyclone II

Which type of output file do you want to create?
- AHDL
- VHDL
- Verilog HDL

What name do you want for the output file?
- c:\altera\71\quaruls\traffic\ 

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user library specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

- Cancel
- Back
- Next
- Finish
Example: ROM
Example: Floating point multiplier
The Nios II soft processor

- 32 bit soft processor from Altera
- 82 instructions
- Up to 256 custom instructions
- Optional multiply and divide depending on the flavor
- Comes in three flavors (number for Cyclone II implementations):
  - *Economy*: emphasizes minimum size ~700 L.E and ~17 DMIPS.
  - *Standard*: performance/size balance ~1400 L.E and ~54 DMIPS
  - *Fast*: best performance ~1800 L.E and ~92 DMIPS

<table>
<thead>
<tr>
<th>Nios II</th>
<th>Nios II/s “Standard”</th>
<th>Nios II/e “Economy”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>6-stage</td>
<td>5-stage</td>
</tr>
<tr>
<td>Max Frequency</td>
<td>200 MHz</td>
<td>180 MHz</td>
</tr>
<tr>
<td>Max D-MIPS</td>
<td>225</td>
<td>130</td>
</tr>
<tr>
<td>Size (4-input LUTs)</td>
<td>1800</td>
<td>1200</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Dynamic</td>
<td>Static</td>
</tr>
<tr>
<td>I-Cache</td>
<td>Up to 64K</td>
<td>Up to 64K</td>
</tr>
<tr>
<td>D-Cache</td>
<td>Up to 64K</td>
<td>no</td>
</tr>
</tbody>
</table>

1. Characteristics in Stratix II 90nm FPGA
# Nios II flavors

![Nios II Processor GUI](image)

<table>
<thead>
<tr>
<th>Nios II</th>
<th>Nios II/e</th>
<th>Nios II/s</th>
<th>Nios II/f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
<td>Cyclone II</td>
<td>Cyclone II</td>
<td>Cyclone II</td>
</tr>
<tr>
<td>System</td>
<td>50.0 MHz</td>
<td>50.0 MHz</td>
<td>50.0 MHz</td>
</tr>
<tr>
<td>CPU</td>
<td>32-bit</td>
<td>32-bit</td>
<td>32-bit</td>
</tr>
<tr>
<td>Instruction Cache</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Hardware Multiply</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Hardware Divide</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Performance at 50 MHz</td>
<td>Up to 5 DMIPS</td>
<td>Up to 25 DMIPS</td>
<td>Up to 51 DMIPS</td>
</tr>
<tr>
<td>Logic Usage</td>
<td>800-700 LEs</td>
<td>1200-1400 LEs</td>
<td>1400-1800 LEs</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>Two M4Ks (or equv.)</td>
<td>Two M4Ks + cache</td>
<td>Three M4Ks + cache</td>
</tr>
<tr>
<td>Reset Vector</td>
<td>Memory:</td>
<td>Offset:</td>
<td>0x0</td>
</tr>
<tr>
<td>Exception Vector</td>
<td>Memory:</td>
<td>Offset:</td>
<td>0x20</td>
</tr>
</tbody>
</table>

**Warning:** Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor.
Nios II processor system
Creating Nios based systems using SOPC and program it using IDE

SOPC builder

Nios II IDE

Reconfigurable Computing
S. Reda, Brown University