ATHENa – Automated Tool for Hardware Evaluation

Supported in part by the National Institute of Standards & Technology (NIST)

ATHENa Team

Venikata “Vinny” MS CpE student
Ekawat “Ice” PhD CpE student
Marcin PhD ECE student
John MS CpE student
Rajesh PhD ECE student
Michal PhD exchange student from Slovakia

Why Athena?

"The Greek goddess Athena was frequently called upon to settle disputes between the gods or various mortals. Athena Goddess of Wisdom was known for her superb logic and intellect. Her decisions were usually well-considered, highly ethical, and seldom motivated by self-interest."

from "Athena, Greek Goddess of Wisdom and Craftsmanship"

ATHENa – Automated Tool for Hardware Evaluation

Benchmarking open-source tool, written in Perl, aimed at an AUTOMATED generation of OPTIMIZED results for MULTIPLE hardware platforms

Currently under development at George Mason University.

Resources

• ATHENa website
  http://cryptography.gmu.edu/athena

• FPGA Embedded Resources web page
  available from the course web page
ATHENa Major Features (1)
- synthesis, implementation, and timing analysis in batch mode
- support for devices and tools of multiple FPGA vendors:
  - XILINX
  - ATERA
- generation of results for multiple families of FPGAs of a given vendor
- automated choice of a best-matching device within a given family

ATHENa Major Features (2)
- automated verification of designs through simulation in batch mode
- support for multi-core processing
- automated extraction and tabulation of results
- several optimization strategies aimed at finding
  - optimum options of tools
  - best target clock frequency
  - best starting point of placement

Generation of Results Facilitated by ATHENa
- batch mode of FPGA tools
- ease of extraction and tabulation of results
  - Text Reports, Excel, CSV (Comma-Separated Values)
  - optimized choice of tool options
  - GMU_optimization_1 strategy
How To Start Working With ATHENA?

**One-Time Tasks**

- Download and unzip ATHENA
  
  [Link](http://cryptography.gmu.edu/athena/)

- Read the Tutorial!

- Install the Required Tools
  
  (see Tutorial – Part 1 – Tools Installation)

- Run ATHENA_setup

**Repetitive Tasks**

- Prepare or modify your source files & source_list.txt

- Modify design.config.txt

  + possibly other configuration files

- Run ATHENA

---

**design.config.txt**

*Your Design*

- Directory containing synthesizable source files for the project
  
  `SOURCE_DIR = examples/sha256_rst`  
  
  `SOURCE_DIR_FILE = source_list.txt`

- A file list containing list of files in the order suitable for synthesis and implementation
  
  `SOURCE_LIST_FILE = source_list.txt`

- Project name
  
  `PROJECT_NAME = SHA256`

- Name of top level entity
  
  `TOP_LEVEL_ENTITY = sha256`

- Name of top level architecture
  
  `TOP_LEVEL_ARCH = rs_arch`

- Name of clock net
  
  `CLOCK_NET = clk`

---

**Timing Formulas**

- Timing Formulas
  
  `LATENCY = TCLK * 65`

  `THROUGHPUT = 512 / (TCLK * 65)`

---

**Application & Optimization Target**

*FPGA Families*

- Optimizing target = speed | area | balanced
  
  `OPTIMIZATION_TARGET = speed`

- Options = default | user
  
  `OPTIONS = default`

- Application = single_run | exhaustive_search | placement_search | frequency_search
  
  `APPLICATION = single_run`

- Trim Mode = off | on | delete
  
  `TRIM_MODE = on`

---

*Commenting the next line removes all families of Xilinx*

- FPGA_VENDOR = xilinx

- Commenting the next line removes a given family
  
  `FPGA_VENDOR = spartan3`

- FPGA_DEVICES = {list of devices} | best_match | all

  `FPGA_DEVICES = best_match`

- SYN_CONSTRAINT_FILE = default
  
  `SYN_CONSTRAINT_FILE = default`

- IMP_CONSTRAINT_FILE = default
  
  `IMP_CONSTRAINT_FILE = default`

- REQ_SYN_FREQ = 120
  
  `REQ_SYN_FREQ = 120`

- REQ_IMP_FREQ = 100
  
  `REQ_IMP_FREQ = 100`

- MAX_SUC_TYPE_UTILIZATION = 0.8
  
  `MAX_SUC_TYPE_UTILIZATION = 0.8`

- MAX_MDL_UTILIZATION = 1
  
  `MAX_MDL_UTILIZATION = 1`

- TRIM_MODE = off | on | delete

  `TRIM_MODE = on`

  `MAX_PIN_UTILIZATION = 0.9`

END VENDOR
**design.config.txt**

**FPGA Families**

- Commenting the next line removes all families of Altera
- Commenting the next line removes a given family
  - FPGA_VENDOR = Xilinx
  - FPGA_FAMILY = Spartan6
  - FPGA_DEVICES = best_match
  - FPGA_hwCONFFile = default
  - FPGA_IPCONFFile = default
  - FPGA_REQ_IMP_FREQ = 120
  - FPGA_MAX_IMP_UTILIZATION = 0.8
  - FPGA_MAX_DSP_UTILIZATION = 0
  - FPGA_MAX_PIN_UTILIZATION = 0.8

**Library Files**

- FPGA_DEVICES = best_match
- FPGA.hwCONFFile = default
- FPGA要求的IPCONFFile = default
- FPGA_REQ_IMP_FREQ = 120
- FPGA_MAX_IMP_UTILIZATION = 0.8
- FPGA_MAX_DSP_UTILIZATION = 0
- FPGA_MAX_PIN_UTILIZATION = 0.8

**Library Files**

- FPGA_VENDOR = Xilinx
  - FAMILY = virtex5
- FPGA_VENDOR = Altera
  - FAMILY = arria

**Result Files**

- report_resource_utilization.txt
- report_timing.txt
- report_options.txt

**Library Files**

- device_lib/xilinx_device_lib.txt
- device_lib/altera_device_lib.txt
  - Files created during ATHENA setup
  - Characterize FPGA families and devices available in the version of Xilinx and Altera tools installed on your computer
  - Currently supported tool versions:
    - Xilinx WebPACK 9.1, 9.2, 10.1, 11.1, 11.5, 12.1, 12.2, 12.3
    - Altera Quartus II Web Edition 8.1, 8.2, 9.0, 9.1, 10.0
    - Altera Quartus II Subscription Edition 9.1, 10.0

**Library Files**

- device_lib/xilinx_device_lib.txt
- device_lib/altera_device_lib.txt
  - Commenting the next line removes a given family
  - FPGA_DEVICES = best_match
  - FPGA hwCONFFile = default
  - FPGA_IPCONFFile = default
  - FPGA_REQ_IMP_FREQ = 120
  - FPGA_MAX_IMP_UTILIZATION = 0.8
  - FPGA_MAX_DSP_UTILIZATION = 0
  - FPGA_MAX_PIN_UTILIZATION = 0.8

**Result Files**

- report_resource_utilization.txt
  - FAMILY = virtex5
  - FAMILY = arria

**Library Files**

- device_lib/xilinx_device_lib.txt
- device_lib/altera_device_lib.txt

**Result Files**

- report_resource_utilization.txt
- report_timing.txt
- report_options.txt
### Result Files

*report_execution_time.txt*

<table>
<thead>
<tr>
<th>Synthesis Time</th>
<th>Implementation Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1h 0m 39s</td>
<td>1h 1m 50s</td>
<td>2m 29s</td>
</tr>
</tbody>
</table>

### design.config.txt

**Functional Simulation (1)**

# Global Generics

### FPGA FAMILY

- FPGA FAMILY = Cyclone II

### VERIFICATION_DIR

- VERIFICATION_DIR = examples/sha256_rs/tb

### FPGA_DEVICES

- FPGA_DEVICES = best_match

### GLOBAL GENERICS

- mem_block_size = 1

### VERIFICATION_ONLY

- VERIFICATION_ONLY = off

### FUNCTIONAL_VERIFICATION

- FUNCTIONAL_VERIFICATION = on

### MAX_PIN_UTILIZATION

- MAX_PIN_UTILIZATION = 0.8

### MAX_MUL_UTILIZATION

- MAX_MUL_UTILIZATION = 1.0

### MAX_MEMORY_UTILIZATION

- MAX_MEMORY_UTILIZATION = 0.8

### REQ_IMP_FREQ

- REQ_IMP_FREQ = 120

### GLOBAL GENERICS

- mem_type = 0, 1

### ADDER

- ADDER_TYPE = ADD_SCCA_BASED, ADD_DSP_BASED

### MULTIPLIER

- MULTIPLIER_TYPE = MULTI_LOGIC_BASED, ADD_DSP_BASED

### MEMORY

- MEMORY_TYPE = M512, M4K, M9K, M20K

### GENERICS_END

**Functional Simulation (2)**

# MAX_TIME_FUNCTIONAL_VERIFICATION

- MAX_TIME_FUNCTIONAL_VERIFICATION = 0

# If blank, simulation will run untill it finishes

# no changes in signals, i.e., clock is stopped and no more inputs coming in

# Perform only verification (synthesis and implementation parameters are ignored)

# VERIFICATION ONLY = off | on

# VERIFICATION ONLY = on

### FUNCTIONAL_VERIFICATION_MODE

- FUNCTIONAL_VERIFICATION_MODE = off

### MAX_PIN_UTILIZATION

- MAX_PIN_UTILIZATION = 0.8

### MAX_MUL_UTILIZATION

- MAX_MUL_UTILIZATION = 1.0

### MAX_MEMORY_UTILIZATION

- MAX_MEMORY_UTILIZATION = 0.8

### REQ_IMP_FREQ

- REQ_IMP_FREQ =

### FPGA_DEVICES

- FPGA_DEVICES = best_match

### GLOBAL GENERICS

- mem_block_size = 1

### ADDER

- ADDER_TYPE = ADD_SCCA_BASED, ADD_DSP_BASED

### MULTIPLIER

- MULTIPLIER_TYPE = MULTI_LOGIC_BASED, ADD_DSP_BASED

### MEMORY

- MEMORY_TYPE = M512, M4K, M9K, M20K

### GENERICS_END

**ATHENA Example including embedded FPGA resources**

**test_circuit:**

![Diagram of test circuit with embedded FPGA resources]
Use of Embedded FPGA Resources in SHA-3 Candidates

### Xilinx FPGA Devices

<table>
<thead>
<tr>
<th>Technology</th>
<th>Low-cost</th>
<th>High-performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>120/150 nm</td>
<td>Virtex 2, 2 Pro</td>
<td></td>
</tr>
<tr>
<td>90 nm</td>
<td>Spartan 3</td>
<td>Virtex 4</td>
</tr>
<tr>
<td>65 nm</td>
<td>Spartan 5</td>
<td></td>
</tr>
<tr>
<td>45 nm</td>
<td>Spartan 6</td>
<td></td>
</tr>
<tr>
<td>40 nm</td>
<td>Virtex 6</td>
<td></td>
</tr>
</tbody>
</table>

### Altera FPGA Devices

<table>
<thead>
<tr>
<th>Technology</th>
<th>Low-cost</th>
<th>Mid-range</th>
<th>High-performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>130 nm</td>
<td>Cyclone</td>
<td>Stratix</td>
<td></td>
</tr>
<tr>
<td>90 nm</td>
<td>Cyclone II</td>
<td>Stratix II</td>
<td></td>
</tr>
<tr>
<td>65 nm</td>
<td>Cyclone III</td>
<td>Arria I</td>
<td>Stratix III</td>
</tr>
<tr>
<td>40 nm</td>
<td>Cyclone IV</td>
<td>Arria II</td>
<td>Stratix IV</td>
</tr>
</tbody>
</table>

### Basic Operations of 14 SHA-3 Candidates

<table>
<thead>
<tr>
<th>Hash Algorithm</th>
<th>DSP Adders</th>
<th>DSP Multipliers</th>
<th>Block Memories</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAKE</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>BMW</td>
<td>Yes</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cubehash</td>
<td>Yes</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ECHO</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Fugue</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Groestl</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Hamsi</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>JH</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Keccak</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Luffa</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SHA-2</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Shabal</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>SHA256</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>SIMD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Skein</td>
<td>Yes</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **NTT** – Number Theoretic Transform
- **GF MUL** – Galois Field multiplication
- **mADD** – integer multiplication, **mADDn** – multiprrend addition with n operands

### DSP ADDERS & MULTIPLIERS
DSP Adders

- **BLAKE**
  - 32-bit or 64-bit Addition

- **BMW**
  - 32-bit or 64-bit Multioperand Addition

- **CubeHash**
  - 32-bit Addition

- **SHA-2**
  - 32-bit or 64-bit Multioperand Addition

DSP Adders

- **Shabal**
  - 32-bit Addition

- **SIMD**
  - 32-bit Multioperand Addition

- **Skein**
  - 64-bit Addition

BLOCK MEMORIES

- **Hamsi**
  - ROM in message expansion
    
    \[
    8 \times 4 \times 256 \times 32 = 256 \text{ kbit in Hamsi-256} \\
    16 \times 8 \times 256 \times 32 = 1 \text{ Mbit in Hamsi-512}
    \]

- **Keccak, JH, SHA-2**
  - Round constants only

- **BLAKE**
  - Permutation

- **SIMD**
  - Twiddle Factors

PRELIMINARY RESULTS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Architecture</th>
<th>Max CL Frequ [MHz]</th>
<th>Throughput [Mbit/s]</th>
<th>Area [Kb, Bits, Byte, Bit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CubeHash</td>
<td>Base</td>
<td>256.41</td>
<td>394.08</td>
<td>795.0 32</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skein</td>
<td>Base</td>
<td>104.54</td>
<td>282.0</td>
<td>1463.0 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>58.15</td>
<td>170.15</td>
<td>1111.0 64</td>
</tr>
<tr>
<td>BMW</td>
<td>Base</td>
<td>13.28</td>
<td>1250.7</td>
<td>4400.0 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>5.79</td>
<td>700.16</td>
<td>2057.0 144</td>
</tr>
<tr>
<td>Shabal</td>
<td>Base</td>
<td>146.47</td>
<td>2067.68</td>
<td>375.0 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>239.16</td>
<td>421.0</td>
<td>286.0 30</td>
</tr>
<tr>
<td>SIMD</td>
<td>Base</td>
<td>40.38</td>
<td>1059.48</td>
<td>3520.0 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>13.73</td>
<td>1819.0</td>
<td>788.0 96</td>
</tr>
</tbody>
</table>

- ✔ - Throughput increases
- ✗ - Throughput decreases
  (most likely as a result of design error)
Block Memory & Adders

| Algorithm | Architecture | Bit Size [B] | Throughput [B/Sec] | Area [ULB Shores | Block Memory & Adders

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Architecture</th>
<th>Bit Size [B]</th>
<th>Throughput [B/Sec]</th>
<th>Area [ULB Shores</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECHO</td>
<td>Embedded</td>
<td>23.00</td>
<td>356.00</td>
<td>✔</td>
</tr>
<tr>
<td>SHA-2</td>
<td>Embedded</td>
<td>120.00</td>
<td>640.00</td>
<td>✔</td>
</tr>
<tr>
<td>Blake</td>
<td>Embedded</td>
<td>190.00</td>
<td>1100.00</td>
<td>✔</td>
</tr>
</tbody>
</table>

✔ - Throughput increases
✗ - Throughput decreases

(Throughput decreases most likely as a result of design error)

Block Memories used to implement T-boxes/S-boxes

- ECHO, SHA-vite-3
  - AES-Sboxes (8x8)
  - AES-Tboxes (8x32)

- Fugue
  - AES-Sboxes (8x8)
  - Fugue-Tboxes (8x128)

- Groestl
  - AES-Sboxes (8x8)
  - Groestl-Tboxes (8x64)

AES Input, internal state, and output

128 bits = 16 bytes

AES Round

SubBytes

ShiftRows

MixColumns

S-box and Inversion in GF(2^8)

Hardware

S-box 8 x 8

ROM

8-bit address

2^8 bits

2^8 words

8-bit output

direct logic
SubBytes Look-up Table

SubBytes Look-up Table

AES SubBytes

ShiftRows

MixColumns

AES MixColumns

AddRoundKey

- simple bitwise addition (xor) of round keys
S-box Based Implementation of AES Round

T-box Based Implementation of AES Round

Fast implementation of the entire round (1)

\[
\begin{align*}
& e_2 = T_0[a_0,2] \oplus T_1[a_1,3] \oplus T_2[a_2,0] \oplus T_3[a_3,1] \oplus K_2 \\
& \begin{bmatrix}
  e_0 \\
  e_1 \\
  e_2 \\
  e_3 \\
\end{bmatrix} = \begin{bmatrix}
  T_0[a_0,1] & T_1[a_1,0] & T_2[a_2,1] & T_3[a_3,0]
\end{bmatrix}
\end{align*}
\]

Each T_i table can be implemented using a 256 x 32 bit ROM

Look-up Tables T

static const u32 T0[256] =
0x6363a5U, 0x8f7084U,
0x777799U, 0x7b7b8dU,
0xff2b2dU, 0x4b5b5b5bU,
0x6f6b6bU, 0x1c5c5c5cU,
0x000000U, 0x10101010U,
0x0e6767a9U, 0x5b2b2b2bU,
0x7ee1e1e1U, 0x6b5d5d7dU,
0x4e5e5e5eU, 0x7e7e7e7eU,
0x8e8e8e8eU, 0x8f8f8f8fU,
0x8e7e7e7eU, 0x8f8f8f8fU,

. . . . . . . . . . . . .
Implementing AES Round Using T-box Tables

Encryption XOR Network

T_i,j

j=0..3, i=0..3

K_j

j=0..3

Output

Input

Implementing AES Round Using T-box Tables

Encryption XOR Network

T_i,j

j=0..3, i=0..3

K_j

j=0..3

Output

Input

Generic Multiplier (1)

entity mult is
generic;
    vendor : integer := XILINX = 0, ALTERA = 1;
    multiplier_type : integer := MUL_LOGIC_BASED = 0, MUL_DEDICATED = 1;
    WIDTH : integer := 8;
port(a : in std_logic_vector (WIDTH-1 downto 0);
b : in std_logic_vector (WIDTH-1 downto 0);
s : out std_logic_vector (WIDTH-1 downto 0))
end mult;

Generic Multiplier (2)

architecture mult of mult is
begin
    xil_dsp_mult_gen : if (multiplier_type = MUL_DEDICATED and vendor = XILINX) generate
        mult_xil : entity work.mult(xilinx_dsp) generic map (WIDTH => WIDTH);
    end generate;

    alt_dsp_mult_gen : if (multiplier_type = MUL_DEDICATED and vendor = ALTERA) generate
        mult_alt : entity work.mult(altera_dsp) generic map (WIDTH => WIDTH);
    end generate;

    xil_logic_mult_gen : if (multiplier_type = MUL_LOGIC_BASED and vendor = XILINX) generate
        mult_xil : entity work.mult(xilinx_logic) generic map (WIDTH => WIDTH);
    end generate;

    alt_logic_mult_gen : if (multiplier_type = MUL_LOGIC_BASED and vendor = ALTERA) generate
        mult_alt : entity work.mult(altera_logic) generic map (WIDTH => WIDTH);
    end generate;
end mult;
Generic Multiplier (3)

architecture xilinx_logic of mult is
begin
temp1 := STD_LOGIC_VECTOR(unsigned(a) * unsigned(b));
s <= temp1(WIDTH-1 downto 0);
end xilinx_logic;

architecture xilinx_dsp of mult is
begin
temp2 := STD_LOGIC_VECTOR(unsigned(a) * unsigned(b));
s <= temp2(WIDTH-1 downto 0);
end xilinx_dsp;

Generic Multiplier (4)

architecture altera_logic of mult is
begin
temp := STD_LOGIC_VECTOR(unsigned(a) * unsigned(b));
s <= temp(WIDTH-1 downto 0);
end altera_logic;

architecture altera_dsp of mult is
begin
temp := STD_LOGIC_VECTOR(unsigned(a) * unsigned(b));
s <= temp(WIDTH-1 downto 0);
end altera_dsp;

FPGA Embedded Resources

Embedded Multipliers

Multipliers in Spartan 3
### Number of Multipliers per Spartan 3 Device

<table>
<thead>
<tr>
<th>Device</th>
<th>Multiplier Columns</th>
<th>Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S50</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>XC3S200</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XC3S400</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>XC3S1000</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>XC3S1500</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>XC3S2000</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>XC3S4000</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>XC3S5000</td>
<td>4</td>
<td>104</td>
</tr>
</tbody>
</table>

### Combinational and Registered Multiplier

![Diagram](image)

### Dedicated Multiplier Block

![Diagram](image)

### Interface of a Dedicated Multiplier

![Diagram](image)

### Embedded Multiplier Block Overview

Each Cyclone II has one to three columns of embedded multipliers.

Each embedded multiplier can be configured to support:

- One 18 x 18 multiplier
- Two 9 x 9 multipliers
### Number of Embedded Multipliers

<table>
<thead>
<tr>
<th>Device</th>
<th>Embedded Multipliers</th>
<th>9 × 9 Multipliers (1)</th>
<th>18 × 18 Multipliers (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP2C35</td>
<td>13</td>
<td>26</td>
<td>13</td>
</tr>
<tr>
<td>EP2C8</td>
<td>18</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>EP2C20</td>
<td>26</td>
<td>52</td>
<td>26</td>
</tr>
<tr>
<td>EP2C35</td>
<td>35</td>
<td>70</td>
<td>35</td>
</tr>
<tr>
<td>EP2C50</td>
<td>86</td>
<td>172</td>
<td>86</td>
</tr>
<tr>
<td>EP2C70</td>
<td>150</td>
<td>300</td>
<td>150</td>
</tr>
</tbody>
</table>

### Multiplier Block Architecture

- **Data A**
  - **Data B**
  - **Data Out**
  - **Embedded Multiplier Block**

### Two Multiplier Types

- **2x2 multiplier**
- **1x1 multiplier**

### Multiplier Stage

- **Signals** `a` and `b` are used to identify the signed and unsigned inputs.

<table>
<thead>
<tr>
<th>Data A</th>
<th>Data B</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned</td>
<td>Low</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Signed</td>
<td>High</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Signed</td>
<td>High</td>
<td>Signed</td>
</tr>
</tbody>
</table>

### 3 Ways to Use Dedicated Hardware

- **Three (3) ways to use dedicated (embedded) hardware**
  - **Inference**
  - **Instantiation**
  - **CoreGen in Xilinx**
    - MegaWizard Plug-In Manager in Altera

### Inferred Multiplier

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity mult18x18 is
  generic (word_size : natural := 18;
           signed_mult : boolean := true);
  port (clk : in std_logic;
         a   : in std_logic_vector(word_size-1 downto 0);
         b   : in std_logic_vector(word_size-1 downto 0);
         c   : out std_logic_vector(2*word_size-1 downto 0));
end entity mult18x18;
architecture infer of mult18x18 is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if signed_mult then
        c <= std_logic_vector(signed(a) * signed(b));
      else
        c <= std_logic_vector(unsigned(a) * unsigned(b));
      end if;
    end if;
  end process;
  end architecture infer;
```
Forcing a particular implementation in VHDL

Synthesis tool: Xilinx XST

Attribute MULT_STYLE: string;
Attribute MULT_STYLE of c: signal is block;

Allowed values of the attribute:
- block – dedicated multiplier
- lut - LUT-based multiplier
- pipe_block – pipelined dedicated multiplier
- pipe_lut – pipelined LUT-based multiplier
- auto – automatic choice by the synthesis tool

Instantiation for Spartan 3 FPGAs

VHDL Instantiation Template

```
-- Component Declaration for MULT1X18 should be placed
-- after architecture statement but before begin keyword
component MULT1X18
  port ( P : out STD_LOGIC_VECTOR (35 downto 0);
       A : in STD_LOGIC_VECTOR (17 downto 0);
       S : in STD_LOGIC_VECTOR (17 downto 0));
end component;

-- Component Instantiation for MULT1X18 should be placed
-- in architecture after the begin keyword
MULT1X18 INSTANCE_NAME : MULT1X18
  port map ( P => user_P,
            A => user_A,
            B => user_B);
```

Xilinx XtremeDSP

- Starting with Virtex 4 family, Xilinx introduced DSP48 block for high-speed DSP on FPGAs
- Essentially a multiply-accumulate core with many other features
- Now also in Spartan-3A, Spartan 6, Virtex 5, and Virtex 6

DSP48 Slice: Virtex 4

Simplified Form of DSP48

```
Adder Out = (Z ± (X + Y + CIN))
```

[Diagram of DSP48 Slice]

[Diagram of Simplified DSP48]
Choosing Inputs to DSP Adder

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PCin</td>
<td>001</td>
<td>AxB</td>
<td>01</td>
<td>AxB</td>
<td>01</td>
</tr>
<tr>
<td>F</td>
<td>010</td>
<td>Inv</td>
<td>10</td>
<td>F</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>011</td>
<td>C</td>
<td>11</td>
<td>AxB</td>
<td>11</td>
</tr>
<tr>
<td>ShEx</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ShEx</td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ P = \text{Adder Out} = (Z \pm (X + Y + CIN)) \]

New in Virtex 5 Compared to Virtex 4

Stratix III DSP Unit

Memory Types

- Memory
  - RAM
  - ROM

- Memory
  - Single port
  - Dual port

- Memory
  - With asynchronous read
  - With synchronous read
**Memory Types in Xilinx**

- Distributed (MLUT-based)
- Block RAM-based (BRAM-based)
- Inferred
- Instantiated
- Manually Using Core Generator

**Memory Types in Altera**

- Distributed (ALUT-based, Stratix III onwards)
- Memory block-based
- Small size (512)
- Medium size (4K, 9K, 20K) (144K, 512K)
- Inferred
- Instantiated
- Manually Using MegaWizard Plug-In Manager

**Inference vs. Instantiation**

There are two methods to handle RAMs: instantiation and inference. Many FPGA families provide technology-specific RAMs that you can instantiate in your HDL source code. The software supports instantiation, but you can also set up your source code so that it infers the RAMs. The following table sums up the pros and cons of the two approaches.

<table>
<thead>
<tr>
<th>Inference in Synthesis</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>Advantages</td>
</tr>
<tr>
<td>Portable coding style</td>
<td>Most efficient use of the RAM primitives of a specific technology</td>
</tr>
<tr>
<td>Automatic timing-driven synthesis</td>
<td>Supports all kinds of RAMs</td>
</tr>
<tr>
<td>No additional tool dependencies</td>
<td></td>
</tr>
</tbody>
</table>

**Block RAM**

- Most efficient memory implementation
  - Dedicated blocks of memory
- Ideal for most memory requirements
  - 4 to 104 memory blocks
  - 16 kbits = 18,432 bits per block (16 k without parity bits)
- Use multiple blocks for larger memories
- Builds both single and true dual-port RAMs
- Synchronous write and read (different from distributed RAM)

**Block RAM can have various configurations (port aspect ratios)**

- 16k x 1
- 8k x 2
- 4k x 4
- 2k x (8+1)
- 1024 x (16+2)
Block RAM Port Aspect Ratios

<table>
<thead>
<tr>
<th>D/Ox Bus Width</th>
<th>D/Ox Bus Depth</th>
<th>Total Data Path Width</th>
<th>ADDR Bus Width</th>
<th>No. of Addressable Locations</th>
<th>Block RAM Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1/4</td>
<td>16,384</td>
<td>16,384</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1/4</td>
<td>8,192</td>
<td>8,192</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>16,384</td>
<td>16,384</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
<td>11</td>
<td>22,409</td>
<td>18,432</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>16</td>
<td>10</td>
<td>10,054</td>
<td>18,432</td>
</tr>
</tbody>
</table>

Single-Port Block RAM

Dual-Port Block RAM

Block RAM library components

<table>
<thead>
<tr>
<th>Component</th>
<th>Data Cells</th>
<th>Parity Cells</th>
<th>Address Bus</th>
<th>Data Bus</th>
<th>Parity Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMB16_52</td>
<td>16,384</td>
<td>-</td>
<td>(13:0)</td>
<td>(15:0)</td>
<td>-</td>
</tr>
<tr>
<td>RAMB16_52</td>
<td>8,192</td>
<td>-</td>
<td>(12:0)</td>
<td>(1:0)</td>
<td>-</td>
</tr>
<tr>
<td>RAMB16_64</td>
<td>4,096</td>
<td>-</td>
<td>(11:0)</td>
<td>(3:0)</td>
<td>-</td>
</tr>
<tr>
<td>RAMB16_128</td>
<td>2,048</td>
<td>2</td>
<td>(10:0)</td>
<td>(7:0)</td>
<td>(6:0)</td>
</tr>
<tr>
<td>RAMB16_512</td>
<td>1,024</td>
<td>15</td>
<td>(9:0)</td>
<td>(15:0)</td>
<td>(1:0)</td>
</tr>
<tr>
<td>RAMB16_512</td>
<td>512</td>
<td>32</td>
<td>4</td>
<td>(8:0)</td>
<td>(31:0)</td>
</tr>
</tbody>
</table>

Cyclone II Memory Blocks

The embedded memory structure consists of columns of M4K memory blocks that can be configured as RAM, first-in-first-out (FIFO) buffers, and ROM.

Memory Modes

The M4K memory blocks support the following modes:

- Single-port RAM (RAM:1-Port)
- Simple dual-port RAM (RAM: 2-Port)
- True dual-port RAM (RAM:2-Port)
- Tri-port RAM (RAM:3-Port)
- Single-port ROM (ROM:1-Port)
- Dual-port ROM (ROM:2-Port)
Single-Port ROM

- The address lines of the ROM are registered.
- The outputs can be registered or unregistered.
- A .mif file is used to initialize the ROM contents.

Stratix II TriMatrix Memory

<table>
<thead>
<tr>
<th>Feature</th>
<th>R612 Blocks</th>
<th>MA6 Blocks</th>
<th>HI-RAM Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum performance</td>
<td>500 MHz</td>
<td>550 MHz</td>
<td>650 MHz</td>
</tr>
<tr>
<td>Total ROM bits (including parity bits)</td>
<td>8128</td>
<td>16384</td>
<td>32768</td>
</tr>
<tr>
<td>Configurations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512 x 1</td>
<td>4K x 1</td>
<td>64K x 8</td>
<td></td>
</tr>
<tr>
<td>256 x 2</td>
<td>2K x 2</td>
<td>32K x 9</td>
<td></td>
</tr>
<tr>
<td>128 x 4</td>
<td>1K x 4</td>
<td>64K x 16</td>
<td></td>
</tr>
<tr>
<td>64 x 8</td>
<td>512 x 8</td>
<td>128K x 32</td>
<td></td>
</tr>
<tr>
<td>32 x 16</td>
<td>256 x 16</td>
<td>256K x 64</td>
<td></td>
</tr>
<tr>
<td>16 x 32</td>
<td>128 x 32</td>
<td>512K x 128</td>
<td></td>
</tr>
<tr>
<td>8 x 64</td>
<td>64 x 64</td>
<td>1M x 256</td>
<td></td>
</tr>
<tr>
<td>4 x 128</td>
<td>32 x 128</td>
<td>2M x 512</td>
<td></td>
</tr>
<tr>
<td>2 x 256</td>
<td>16 x 256</td>
<td>4M x 1024</td>
<td></td>
</tr>
<tr>
<td>1 x 512</td>
<td>8 x 512</td>
<td>8M x 2048</td>
<td></td>
</tr>
</tbody>
</table>

Stratix III & Stratix IV TriMatrix Memory

<table>
<thead>
<tr>
<th>Feature</th>
<th>MA60</th>
<th>HI-RAM Blocks</th>
<th>HI-RAM Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum performance</td>
<td>600 MHz</td>
<td>650 MHz</td>
<td>700 MHz</td>
</tr>
<tr>
<td>Total memory bits (including parity bits)</td>
<td>640 (in ROM mode) or 650 (in other modes)</td>
<td>650</td>
<td>650</td>
</tr>
<tr>
<td>Configurations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 x 0</td>
<td>4 x 1</td>
<td>16 x 8</td>
<td></td>
</tr>
<tr>
<td>16 x 10</td>
<td>2 x 10</td>
<td>16 x 10</td>
<td></td>
</tr>
<tr>
<td>16 x 16</td>
<td>1 x 16</td>
<td>16 x 16</td>
<td></td>
</tr>
<tr>
<td>16 x 32</td>
<td>1 x 32</td>
<td>16 x 32</td>
<td></td>
</tr>
<tr>
<td>16 x 64</td>
<td>1 x 64</td>
<td>16 x 64</td>
<td></td>
</tr>
<tr>
<td>16 x 128</td>
<td>1 x 128</td>
<td>16 x 128</td>
<td></td>
</tr>
<tr>
<td>16 x 256</td>
<td>1 x 256</td>
<td>16 x 256</td>
<td></td>
</tr>
</tbody>
</table>

Stratix II & III Shift-Register Memory Configuration