TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Computer Engineering, Chair of VLSI Design, Diagnostics & Architecture

VHDL – SYNTHESIS

Dr. Thomas B. Preußer

thomas.preusser@tu-dresden.de

Dresden, July 5, 2014

Goals of this Lecture

- Use cases for FPGAs,
- Programmability of FPGAs,
- Illustrative Synthesis Mappings.

High-Performance Custom Computing

Option	Development Cost & Time	Mask Costs	Power Bill	Design Adaptability
SW on Supercomputers	low	none	huge	cheap
Full-Custom ASIC	huge	huge	low	extremely expensive
Semi-Custom ASIC	large	high	low	very expensive
PLD / FPGA	high	none	low	reasonable

High-Performance Custom Computing

Option	Development Cost & Time	Mask Costs	Power Bill	Design Adaptability
SW on Supercomputers	low	none	huge	cheap
Full-Custom ASIC	huge	huge	low	extremely expensive
Semi-Custom ASIC	large	high	low	very expensive
PLD / FPGA	high	none	low	reasonable

SW

- is ideal for flexible number-crunching applications, but
- requires an extensive infrastructure, and
- produces huge power bills.

High-Performance Custom Computing

Option	Development Cost & Time	Mask Costs	Power Bill	Design Adaptability
SW on Supercomputers	low	none	huge	cheap
Full-Custom ASIC	huge	huge	low	extremely expensive
Semi-Custom ASIC	large	high	low	very expensive
PLD / FPGA	high	none	low	reasonable

SW

- is ideal for flexible number-crunching applications, but
- requires an extensive infrastructure, and
- produces huge power bills.

ASICs

- are only affordable for *very* high-volume systems,
- do not match FPGA performance if budget dictates old technology, and
- are extremely hard and expensive to bugfix or adapt.

TECHNISCHE UNIVERSITÄT DRESDEN

High-Performance Custom Computing

Option	**Development Cost & Time**	**Mask Costs**	**Power Bill**	**Design Adaptability**
SW on Supercomputers	low	none	huge	cheap
Full-Custom ASIC	huge	huge	low	extremely expensive
Semi-Custom ASIC	large	high	low	very expensive
PLD / FPGA	high	none	low	reasonable

SW

- is ideal for flexible number-crunching applications, but
- requires an extensive infrastructure, and
- produces huge power bills.

ASICs

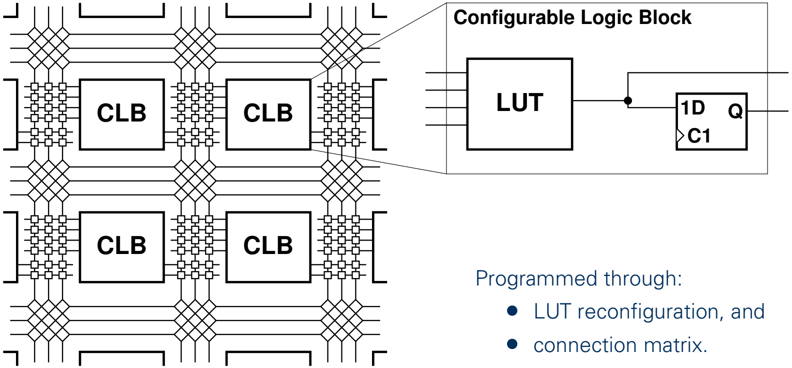
- are only affordable for *very* high-volume systems,
- do not match FPGA performance if budget dictates old technology, and
- are extremely hard and expensive to bugfix or adapt.

FPGAs

- offer tremendous performance gains for low power budgets,
- are suited for a mostly stable core computation, and
- exploit *bit-level* optimizations.

TECHNISCHE UNIVERSITÄT DRESDEN

FPGA – Field-Programmable Gate Array

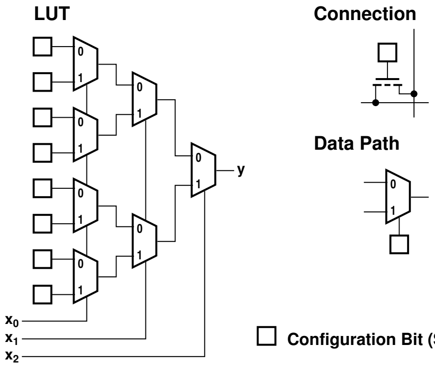


Programmed through:

- LUT reconfiguration, and
- connection matrix.

FPGA – Field-Programmable Gate Array

Configurability



□ **Configuration Bit (SRAM Cell)**

Special-Purpose Hardware Structures

- IO pads (obviously a must),
- memory blocks,
- multipliers (DSP blocks),
- high-speed transceivers,
- PCIe endpoints, . . .

They implement frequently needed functionality whose *soft* implementation in the reconfigurable fabric is little efficient:

- occupying many general-purpose resources, or
- being too slow.

FPGA – Field-Programmable Gate Array

FPGA Design vs. Software

Configurable Hardware:

- greater design effort (VHDL, Verilog),
- $10\times$ lower clock frequency than standard CPUs,

+ fine-grained custom concurrency,

+ high computational power per Watt,

+ no instruction overhead.

[Xilinx Inc.]

FPGAs offer:

- affordable custom hardware performance (beyond a mass market),
- revisable designs (prototyping, communication protocols), and
- tremendous low-level concurrency.

Hints for Synthesis

- Good solutions demand a good overview!
  Structure $\rightarrow$ Block Diagrams
  Behavior $\rightarrow$ SM-Charts
- Design preferably on a high abstraction level.
  The synthesis tool usually does a better technology mapping.
- Buffer asynchronous inputs (external, other clock domains) with FFs!
  Unless there is *definitely* only *one* I/O-register-path,
  inconsistencies due to run-time differences may occur.
- FPGA: It is better to use more FFs instead of complex logic.
  Every logic cell has one FF.
  One-hot state encoding is usually favorable for non-trivial state machines.

Meta-Values

. . . are reduced in the physical design:

- to '0'|'1' for internal binary signals, and
- to '0'|'1'|' Z' for output drivers.

Do *not* assign 'U'|'W' to signals in synthesizable code. 'U' is valid in initializers.

'L'|' H' silently map to '0'|'1' , respectively, *but* this is bad style.

'−'|' X' are mapped to '0'|'1' *by the choice of the synthesis tool.*
Use Don't Cares intensively! They:

- reveal the use of seemingly irrelevant signal values by computing 'X' instead of a normal but wrong signal value in the simulation.
- document irrelevant signal states, and
- open optimization opportunities for the synthesis tool.

Coping with Meta-Values

Use meta-values so as to assist the debugging through simulation!

There are library functions to help with the expanded value range:

use IEEE.std_logic_1164.**all**;

Function	Simulation	Synthesis
Is_X(s)	s maps unambiguously to ’0’ or ’1’	(false)
to_X01	Maps to ’0’, ’1’ or ’X’ (’U’, ’X’, ’W’, ’-’, ’Z’)	Identity
to_X01Z	Maps to ’0’, ’1’, ’X’ (’U’, ’X’, ’W’, ’-’) or ’Z’	Identity

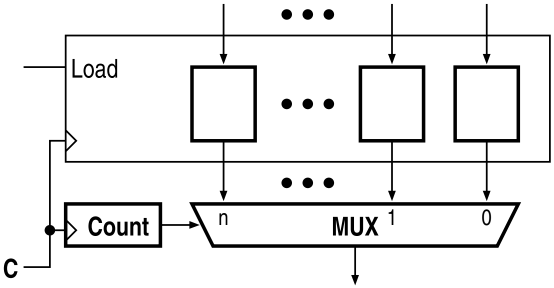
select and **case** statements and other control flow structures can often be greatly simplified using these functions.

others branches often help as well.

TECHNISCHE UNIVERSITÄT DRESDEN

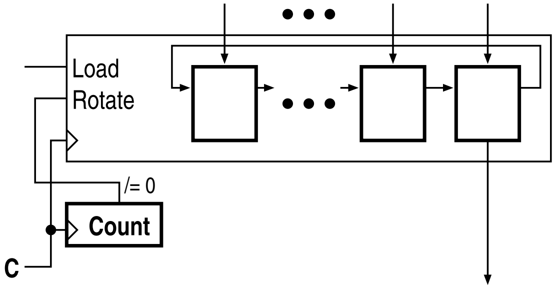
Example: Iteration over Array

- Array access by index increment implies a complex multiplexer circuit or enable logic.



Example: Iteration over Array

- Often better:
  Simple regular structure with the help of a shift register.



Application:
Parallelization, Serialization, Iteration over digits (arithmetic), . . .

Example: Wide Gates and Comparison

Keep implementations maintainable by the use of array attributes:

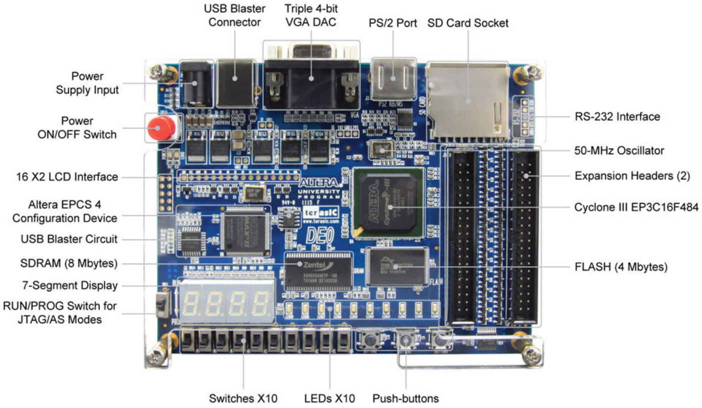
```
-- Wide OR
y <= '1' when x /= (x'range => '0') else '0';

-- Wide AND
y <= '1' when x  = (x'range => '1') else '0';

-- Wide XOR
process(x)
  variable t : std_logic;
begin
  t := '0';  -- Neutral initialization
  for i in x'range loop
    t := t xor x(i);
  end loop;
  y <= t;    -- Final result assignment (signal)
end process;

-- Comparator
y <= '1' when x = to_unsigned(42, x'length) else '0';
```

Altera DE0 Board

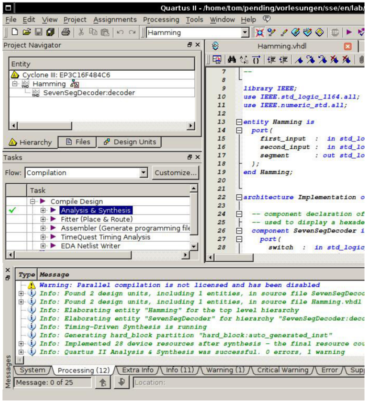


Quartus Main Window

Project Navigator
- design structure
- file selection

Task Window
- start synthesis
- progress display

Message Log
- process outputs



```
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Hamming is
  port(
    first_input  :  in std_lo
    second_input :  in std_lo
    segment      : out std_lo
  );
end Hamming;


architecture Implementation o

  -- component declaration of
  -- used to display a hexade
  component SevenSegDecoder i
    port(
      switch  :  in std_logic
```

```
Warning: Parallel compilation is not licensed and has been disabled
Info: Found 2 design units, including 1 entities, in source file SevenSegDecod
Info: Found 2 design units, including 1 entities, in source file Hamming.vhdl
Info: Elaborating entity "Hamming" for the top level hierarchy
Info: Elaborating entity "SevenSegDecoder" for hierarchy "SevenSegDecoder:deco
Info: Timing-Driven Synthesis is running
Info: Generating hard_block partition "hard_block:auto_generated_inst"
Info: Implemented 28 device resources after synthesis - the final resource cou
Info: Quartus II Analysis & Synthesis was successful. 0 errors, 1 warning
```

Editor

Report Window

Essential Quartus Tasks

Settings
- Adjust project settings,
- Associate testbenches.

Pin Planner
- Assign FPGA pins to the ports of the top-level module.

Compilation
- Synthesize the design.

TimeQuest Timing Analyzer
- Define timing constraints, e.g. clocks.

Programmer
- Configure the FPGA.