

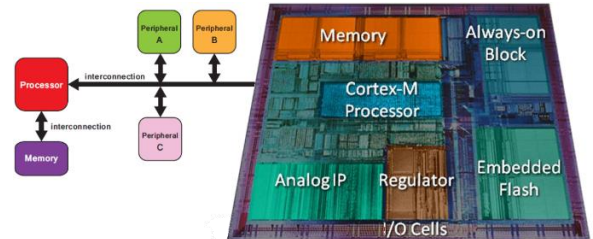
Lecture 1 How to design a SoC

Objective

- ...list the main characteristics of a SoC
- ...explain the main challenges of a SoC design
- ...list the main features of the Vivado IDE
- ...describe the general SoC design flow and the Vivado IDE Design flow

System on Chip

- Prozessor
- RAM
- Bussystem
- Speicher
- IP-Blöcke
- Interfaces und Peripherals zur Umwelt



Vorteile

- Kurze Entwicklungszeiten
- Sinkende Entwicklungskosten
- Kleinere Kosten für kleine bis mittlere Stückzahlen
- Grössere Flexibilität im Betrieb (updates, etc.)
- Hoher Grad von Parallelität in allen Stufen
- Tools/ Components/ Vendors / Hardware Beschreibung und Programmierung / Sprachen

Design Challenges

- Komplexität
- Heterogenität

Requirements

- Time to market
- Re-use

- Marktdruck
- Es kann nicht alles neu entwickelt werden
- Re-use von Komponenten und Teilsystemen

IP based Design

Parallel besser als Sequentiell

HW und SW parallel entwickeln

IP – Intellectual Property (dt: geistiges Eigentum)

Typen von IP-Blöcken:

Hard IP Cores

(black box in optimized layout form)
Bei FPGA sind diese bereits herstellerseitig unveränderbar auf Chip integriert.

Firm IP Cores

(Synthesized netlist, which can simulated and changed if needed)

Soft IP Cores

IP liegt bereits als synthetisierte Netzliste vor
(RTL design, user needs to synthesize and layout (P&R))
IP existiert in Form von Quell-Code in einer HWDL Sprache wie Verilog oder VHDL.

Sources of IP Cores

- Legacy in-house IP => z.B. IP von eigener Firma
- Third party IP
- New written in-house IP

Lecture 2 SoC Design Hardware

Objective

- ...describe the general architecture of the Zynq SoC
- ...describe the basic slice resources available in XILINX 7-Series FPGAs
- ...list and describe the various types of pads and special resources on FPGA and how they can be used
- ...use the syntax of design constraints and you will be able to define constraints for pad assignments and clock signal definition
- ...interpret timing information given in timing reports

Architecture Overview

Zynq-700 AP SoC:

PS -> ARM dual core A9
Multiple peripheral

PL ->

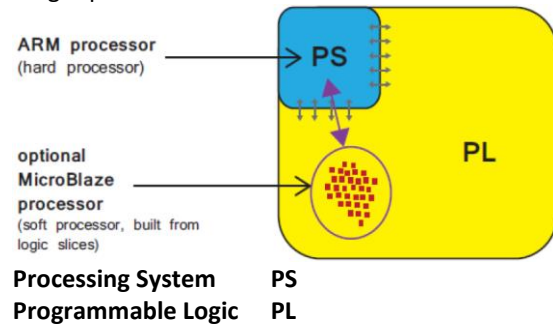
Analog -> 2 ADC (1MS/sec, 12 bit)
On-Chip Sensors (Temp. and Volt.)

PL in detail

LUT, Switch Matrix, Input/Output Blocks, Block RAM, DSP

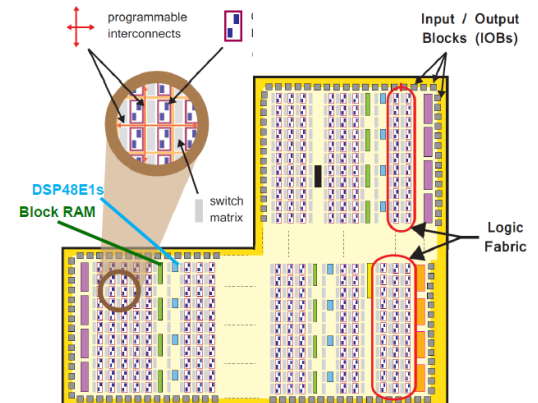
PL in detail

Example of Zynq



Logic Fabric

- CLB (Configurable Logic Block) -> (Contains two slices)
 - Slice Resource:
 - Four six-input Look-Up Tables (LUT) (RAM/SR in SLICEM)
 - Multiplexer
 - Carry chains (Channel for CARRY-Bit)
 - SRL (Shift Register Left)
 - Four flip-flops/latches
 - Four additional flip-flops
 - Resource for LUT, flip-flop
 - Type of slices:
 - SLICEM: Full slice
 - LUT can be used for logic and memory/SRL
 - Has wide multiplexers and carry chain
 - SLICEL: Logic and arithmetic only
 - LUT can only be used for logic (not memory)
 - Has wide multiplexers and carry chain
 - Switch Matrix
 - CLB is connected to switch matrix for routing to other FPGA resources



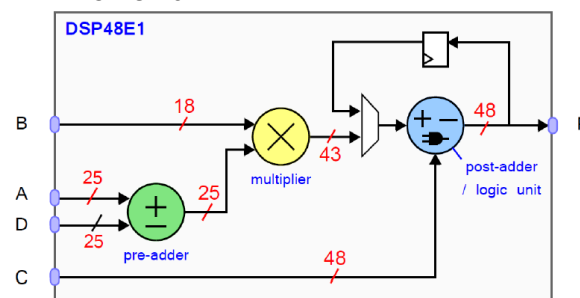
Input / Output Blocks (IOBs)

- High Range (HR)
 - up to 3.3V
- High Performance (HP)
 - up to 1.8V, designed for the highest performance, has ODELAY and DCI capability

Special Resources

- Block RAM
- DSP48E1s

DSP: DSP48E1 Slice

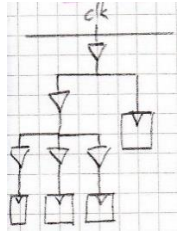


Kann auf der höchsten CLK-Frequenz betrieben werden.

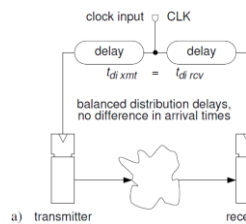
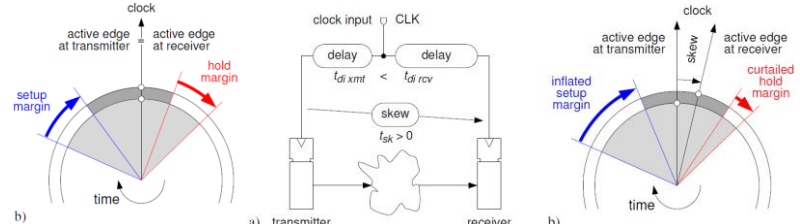
IDELAY and ODELAY:

The IDELAY or ODELAY will allow you to add an additional delay to the data so that you do not violate setup/hold timing of the input register (for IDELAY) or the off chip synchronous element (for ODELAY).

Clocks

**Problems:**

- Clock distribution delay t_{di} Time lag measured from when clock edge appears at clock source until state transition takes place. (Eingang CLK bis schlussendlich Flipflop schaltet)
- Clock skew t_{sk} time between first and last arrival of the same CLK edge at the flipflop. (Inaccuracy of same clock edge arriving at different locations within clock domain.)
- Clock jitter t_{jt} time variation of the CLK to the same flipflop (variation of voltage level detection, ...)
- Slack Difference between required time and arrival time, also called margin. Negative slack => timing problem

No skew**With skew**

Constraint

Hardware options have to be defined for each individual project

Constraint types*Synthesis Constraints*

Direct synthesis tool optimization techniques for particular design or piece of HDL code. Either embedded within VHDL code or defined in separate synthesis constraint file.

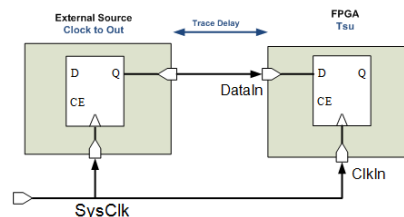
Implementation Constraints

Direct implementation tools to direct mapping, placement, timing or other guidelines. Generally placed in .xdc File. May exist in HDL.

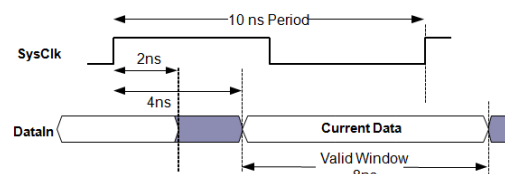
*.XDC

Constraints are applied sequentially. => order matters!

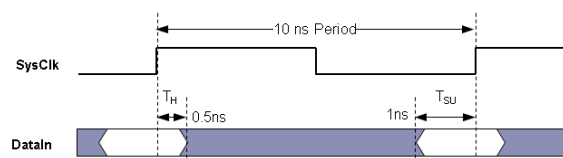
.xdc

**Example :****.xdc configuration 1**

```
create_clock -name SysClk -period 10 -waveform {0.000 5.000} [get_ports Clkin]
set_input_delay -clock SysClk 4 [get_ports DataIn]
set_input_delay -clock SysClk -min 2 [get_ports DataIn]
# -Maximum delay is used for the setup check
# -Minimum delay is used for the hold check
# Without the -max or -min option, the value supplied is used for both
```

Result 1 :**.xdc configuration 2**

```
create_clock -name SysClk -period 10 [get_ports Clkin]
set_output_delay -clock SysClk 1 [get_ports DataIn]
set_output_delay -clock SysClk -min -0.5 [get_ports DataIn]
#set_output_delay -max specifies the required setup time
#set_output_delay -min specifies the negative of the required hold time
```

Result 2:

Lecture 3 SoC Design using IP

Objective

- ...describe the syntax and use of functions, procedures, and packages in VHDL.
- ...to generate parametrized components and to instantiate them.
- ...to use the IP catalog and integrate IP in your design
- ...list the various types of AXI interfaces present on Zynq. You know the basic functionality and you can use AXI interfaces to connect your PL designs with the processing system.

The goal

- Manage the complexity -> Re-use proven IP
- Manage the heterogeneity -> Hierarchy!
- > HW/SW partitioning
- Don't delay market introduction -> Re-use IP
- > Modify existing IP
- > design only, what is really needed
- > High level design if possible

Don't invent the wheel => Re-use!!

Function in VHDL

- Is a hierarchical structure in VHDL, facilitating re-use.
- Are sub-programs having only inputs in their list of arguments and return one single value of a specified type.

Example: -- parity generator for odd parity, realized as function

```

library ieee;
use ieee.std_logic_1164.all;

entity parity_gen8 is
port( A      : in  std_ulogic_vector(3 downto 0);
      B      : in  std_ulogic_vector(7 downto 0);
      PUA, PUB : out std_ulogic);
end entity parity_gen8;

architecture RTL of parity_gen8 is
  function PARGEN(AVECT : std_ulogic_vector) return std_ulogic is
    variable PU_VAR : std_ulogic;
  begin
    PU_VAR := '1';
    for I in AVECT'range loop
      if AVECT(I) = '1' then PU_VAR := not PU_VAR;
    end if;
  end loop;
  return PU_VAR;
end function PARGEN;

-- function definition
-- begin of function

-- end of function
-- begin of architecture
begin
  PUA <= PARGEN(A);
  PUB <= PARGEN(B);
end architecture RTL;

-- end of architecture

```

Procedures in VHDL

Mostly equal to function, may not be synthesised rather do not use.

Packages in VHDL

Manage complexity, facilitate re-use

Notice: Constants, Types, Components, Functions and Procedures that are used at various places in one or several projects may be grouped into packages.

```

library ieee;
use ieee.std_logic_1164.all;

package parity_package is
  constant nibble : integer; -- Declaration and initialization optional
  constant word : integer; -- Declaration
  function PAR_GEN(AVECT : std_ulogic_vector) return std_ulogic;
end package parity_package;

package body parity_package is
  function PAR_GEN(AVECT : std_ulogic_vector) return std_ulogic is
    variable PU_VAR : std_ulogic;
  begin
    PU_VAR := '1';
    for I in AVECT'range loop
      if AVECT(I) = '1' then
        PU_VAR := not PU_VAR;
      end if;
    end loop;
    return PU_VAR;
  end function PAR_GEN;

  constant nibble : integer := 4; -- Declaration and initialization
  constant word : integer := 8; -- Declaration and initialization
  ..... -- may contain more functions, procedures, etc.
end package body parity_package;

```

Use packages in VHDL module with the use statement:

```
use work.my_package.all;
```

Distributed RAM
VHDL Code

Main Properties: Value stored in LUT's

- Write synchronous/ read: asynchronous
- Large area, fast access only for small structures

Block RAM in VHDL
Code

Main Properties: Values stored in dedicated RAM blocks at 36kB

- Write synchronous/ read: synchronous
- Optimized area, fast for larger structures

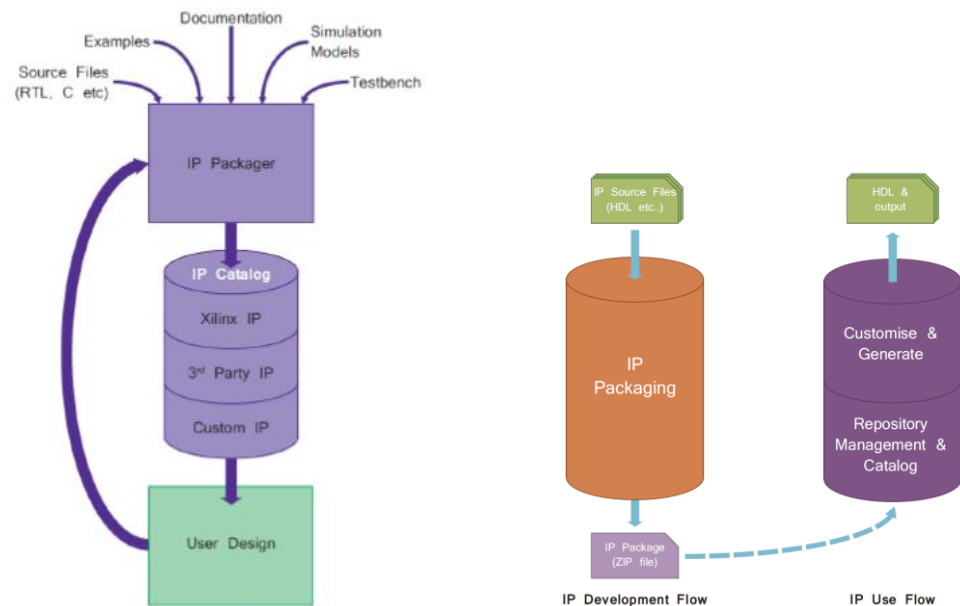
Distributed vs. Block Ram

Block RAM ist in den Chip integriertes RAM und wird daher mit speziellen RAM-Blöcken à 36kB realisiert.

Distributed RAM wird dagegen aus den Logikressourcen des FPGA gebildet. Wenn das RAM in VHDL so implementiert wird, dass asynchron gelesen werden kann, dann implementiert der Synthesizer automatisch so, dass der Speicher in Distributed RAM angelegt wird. Wird der Speicher so implementiert, dass bei else, also synchron gelesen wird, dann reagiert der Synthesizer automatisch so, dass der Speicher in Block RAM angelegt wird!

XILINX IP Catalog-
Overview (left)

Vivado IP-
Organisation (right)

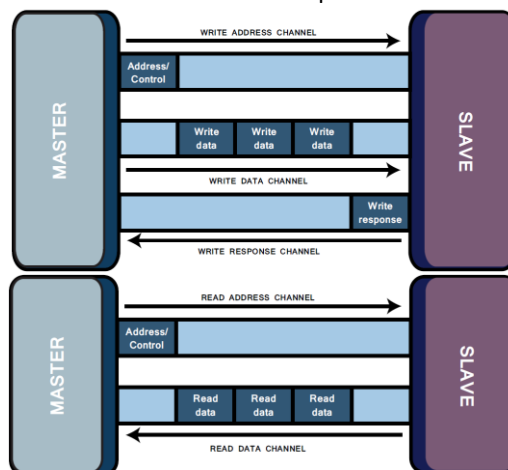


➤ interfaces

AXI – Bus

AXI Interface: Advanced eXtensible Interface
Impact today: de facto standard for on-chip communication
ASX14 write channel

AXI4 read channel



AXI-Bus types

AXI4: Memory-mapped links; Address/Data Burst
AXI4-Lite: Simplified link; Address/Data Word
AXI4-Stream: For high-speed streaming. Dir. Burst

Lecture 4

Custom IP Blocks design

Objective

System Design

...list at least 4 criteria that will guide your system design

- *Satisfy customer/ business needs/ Use hierarchical approach/ Performance ...*

HW/SW Partitioning

...describe a typical design flow for hardware – software code sign and the hardware part in general

- *SW as long as requirements are satisfied*
- *SW in general for OS, GUI, user application*
- *HW if speed/latency requires parallelism*
- *HW in general for comp. intensive data flow*

IP Block Creation

...list the steps involved in creating custom IP. Create reusable IP blocks. You will know the difference between OOC and Global synthesis.

- *look up below at "IP Block Creation"*
 - o *OOC: IP not re-synthesized in context*
 - o *Global: IP re-synthesized*

IP Block Integration

...describe the mechanism for using IP in a new context and connect it to a processor.

...describe IP life cycle.

...integrate IP in your design.

- *IP integrator in block design, instantiation in RTL design*
- *Version numbering: mayor#.minor#.revision#*

General Approach

General Procedure:

Design system architecture from top-down perspective

Define top level interface and parameter

In general:

Use a high abstraction level -> SW

Hardware/Software Partitioning

Definition:

Refers to the process of deciding, for each subsystem, whether the required functionality is more advantageously implemented in hardware or in software.

Guidelines Software:

- General purpose sequential processing task
- Operation system
- User application
- GUI's

Guidelines Hardware:

- Computationally, intensive data flow parts of the design
- Software algorithms exhibiting significant parallelism (coprocessor model)

Design Flow

Refer at the graphic in Lecture 6

IP Block Creation

What elements belong to an IP?

- Design Source
- Simulation Models
- Documentation
- Example Designs
- Example Test Benches

Output product

IP synthesized without context -> OOC (Out-Of-Context)

IP Lifecycle

Pre-Production => Production => Discontinued => Superseded (ersetzt) => Removed

Lecture 5

Verifying your SoC I: Design phase

Objective

- ...indicate at least two methods for assuring good circuits during specification phase
- ...say what exhaustive verification means
- ...list at least 4 measures for assuring good circuits during design phase
- ...tell the difference between static and dynamic verification
- ...use file operations in VHDL and to write test benches with file input
- ...debug code and FPGA circuits

Verification & Specification Phase

⇒ *Is what I am asking for what is really needed?*

Formal Specification

Unsuitable for communication with customer and management

Rapid Prototyping (virtual prototyping):

Can be implemented as software code on computer, microprocessor or DSP

Design Phase

⇒ *Have I indeed designed what I have asked for?*

Adequate (angemessene) strategy:

- Dynamic verification (simulation, debugging): Apply stimuli, observe response.
- Static verification (code inspection, static timing analysis, DRC, ERC, LVS, etc.): independent of waveform, at time t=0

Properties to be verified:

- *Parametric properties*: physical properties measured in V, mW, pF, °C, ns, etc.
- *Functional properties*: output = F(inputs, state) -> dynamic verification
-> exhaustive verification (verification of all possibilities) is not possible

=> Recommendation: Functionality and parametric properties are best checked separately.

A realistic compromise of Functional Verification:

(1) Assertion Based Verification

Safety checks before debugging:

- Memory addresses outside legal range
- State machine entering error states
- Unforeseen input values
- Numeric over-/underflow

```
assert condition report "message" severity
error;
```

- Condition: any Boolean expression
- Report "message": string message will be displayed in console window during simulation. If omitted, standard message will be written on simulation screen.
- Severity: optional statement. Following severity levels are defined: note, warning, error, failure. Failure normally aborts simulation

Assertion in VHDL -> assert condition report "message" severity error;
Vivado synthesizer ignores assertions

(2) Test Cases

1. Apply stimuli that make DUT work in all regular regions of operation
2. Stimuli for likely cases (division by very small number, under-/overflow)
3. Apply known exceptional cases in control flow
4. Apply stimuli collected from real-world services
5. Apply randomly selected test cases

(3) Separate Design and Test

Organize manpower into two independent teams for circuit design and testing

(4) Virtual Prototyping

Virtual prototype can be tested in real environment

Evaluation of Response

Pure visual inspection is not acceptable for SoC design for reasons of efficiency and quality.
Automatic procedure

Functional Verification Methods

- | | | |
|--------------------|---|---|
| Simulation | + | very good observability & debug ability |
| | - | no real time, slow |
| Emulation | + | faster than simulation |
| | - | Effort for implementing model on hardware |
| Prototyping | + | real time feasible |
| | - | Limited observability & debug ability |

Simulation Flow	- Behavioural Simulation: runs on HDL -> fast/ verify functionality only/ No timing information is considered - Post Synthesis Simulation: Structural simulation/ Verifies synthesized netlist/ Timing possible, but unusual (no routing) - Post Implementation Simulation: Functional and timing simulation possible/ Most precise, but slowest
How to find a bug	1. Bug sensitization (Bug anstossen) 2. Bug must propagate to observable nodes 3. Bug observation: result must show difference to logic value of correct circuit ➔ Simulation: stepping through code Break point -> to code line Conditional Breakpoint: finding specific event in simulation
Virtual Input/Output (VIO)	SoC (and FPGA) device in principle only accessible through pins at interface level => with VIO IP internal nodes can be defined as inputs or outputs => inputs can be driven during debug session in real time and also the VIO outputs can be analysed during debug session.
Integrated Logic Analyser (ILA)	Used for monitoring internal programmable logic signals for post-analysis.

Lecture 6 Verifying your SoC II: Production phase

During testing: Can I tell intact circuits from malfunctioning ones?

Objectives

After completing this lecture and the lab, you will be able to ...

...list at least three reasons for verification of System on Chip circuits

...describe at least two fault models

...apply fault collapsing for given vectors

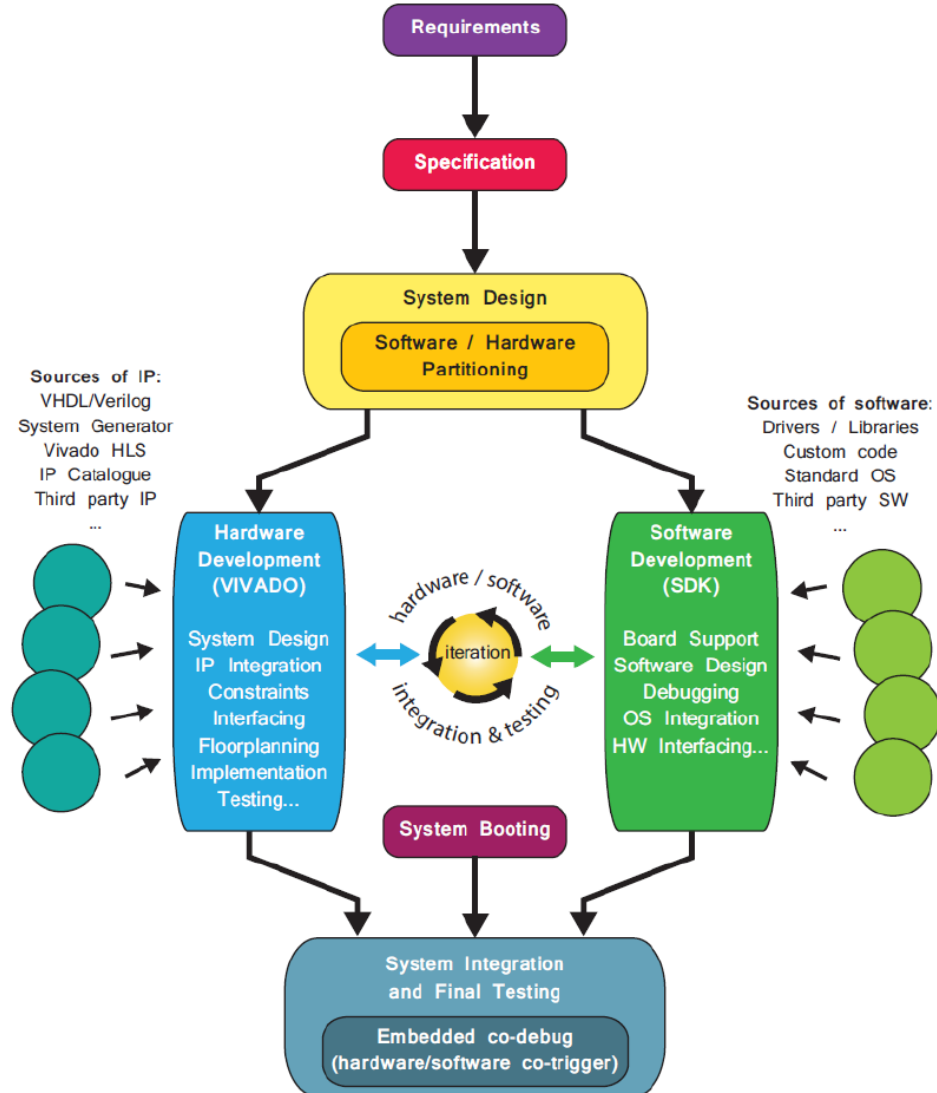
...describe IDDq test and its advantages

...describe procedure and requirements for scan test

...list components required for BIST and describe BIST procedure

...list components required for boundary scan test and describe functionality

Design Flow



Defects

Defect: A defect is a physical source of fault

Fault: Circuit malfunction

Common defects: Processing defects (mask misalignment, dust on wafer, ...), Material defects, Packaging failures, Short/open circuits, Missing, wrong or reversed components

Faults are expensive

The cost of a fault increases by a factor of ten for every production step -> The earlier, the better!!

Fault models

Models: stuck-at fault; bridging fault; delay fault

(Fault assumptions: single fault, only one fault occurs in a circuit)

Stuck-at-fault: signal level stuck at Low or High-Level, because of any reason (Short circuit, soldering bridge, failure during process through crystallization or dirt)



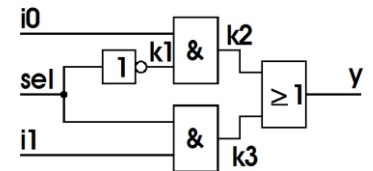
Bridging fault: Two circuit nodes are electrically connected when they shouldn't be.

Delay fault: Sequential circuit will exhibit timing error, because path delay too long

Example
2:1 Multiplexer

Testvector	Sel	I1	I0	Y	K1: s-a-0	K1: s-a-1	K2: s-a-0	K2: s-a-1	K3: s-a-0	K3: s-a-1
TV0	0	0	0	0	-	-	-	1	-	1
TV1	0	0	1	1	0	-	0	-	-	-
TV2	0	1	0	0	-	-	-	1	-	1
TV3	0	1	1	1	0	-	0	-	-	-
TV4	1	0	0	0	-	-	-	1	-	1
TV5	1	0	1	0	-	1	-	1	-	1
TV6	1	1	0	1	-	-	-	-	0	-
TV7	1	1	1	1	-	-	-	-	0	-

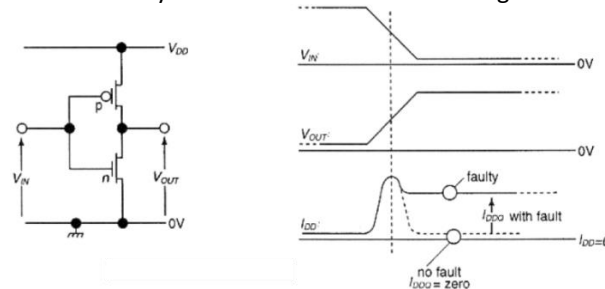
If only the presence of a fault is of interest, but not, where exactly it is, complete test is normally not required. A subset of test vectors is sufficient = Fault Collapsing.
In this example TestVector1, TV5 and TV7 discover all stuck-at faults



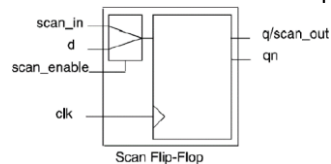
Design for Test

IDDq: Current Drain-Drain-Query

detect fault by measure the current -> much greater as expected -> fault is existing

**Scan path**

modification are required: scan FF, additional path, pads

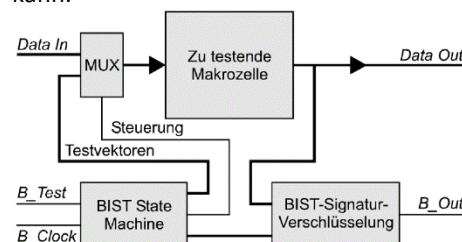


Operation:

Scan mode:	Load register
Normal mode:	run system
Scan mode:	read scan register

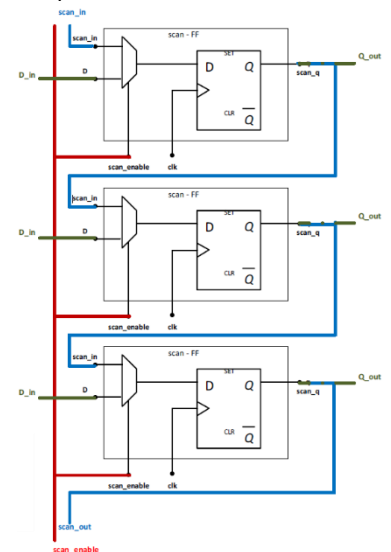
Built In Self Test (BIST)

Dabei erhält jede Makrozelle eine zusätzliche Logik zugeordnet, mit der dieser Selbsttest durchgeführt werden kann.

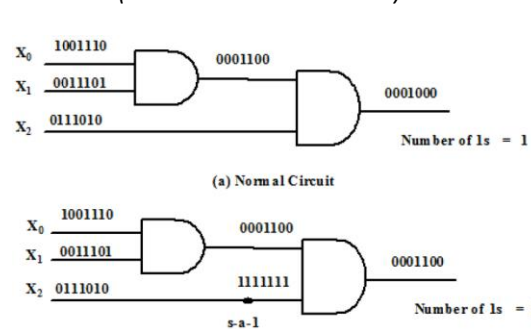
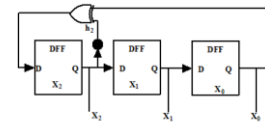


Normalbetrieb: Data In -> Zu testende Makrozelle -> Data Out

Testbetrieb: Testvektoren (MUX geschaltet) -> Zu testende Makrozelle -> BIST-Signatur-Verschlüsselung -> wertet Testergebnis aus -> gibt Signal B_Out aus



Vorteil: Intern kann man es so schnell laufen lassen wie das System kann! Zudem kann das System in Echtzeit überwacht werden. Ausgewertet werden dann z.B. die Anzahl von 1en (=> siehe Abb. unten links).

LFSR (3 FF, 1 XOR): 2^n-1 TV

d	x2	x1	x0
1	0	0	1
1	1	0	0
1	1	1	0
0	1	1	1
1	0	1	1
...	...	...	...

BIST => Input test pattern generation:

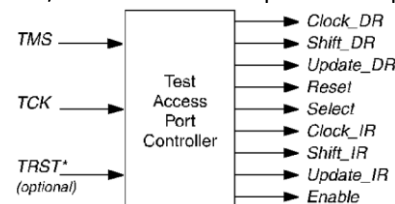
Input Test-Patterns werden mithilfe von on-Chip State Machines generiert (=> siehe Abbildung oben rechts). So müssen die LUT's nicht im ROM gespeichert werden. Der Vorteil in diesen Generatoren liegt hier darin, dass das ROM zuerst auch noch implementiert werden und zudem auch nochmals einzeln getestet werden müsste.

Boundary Scan

Each Input/output needs to be replaced by BS-Cell. Each component needs additional hardware components

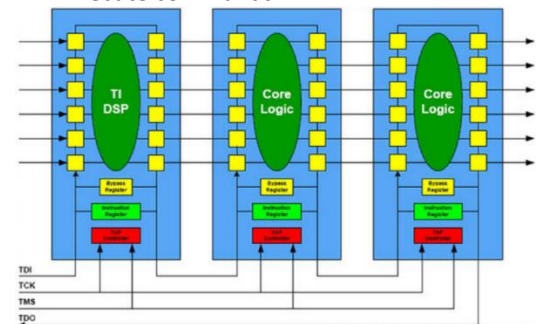
Boundary Scan: Circuit Requirements

TMS: sets TAP controller into test mode
TCK: Test clock, independent from sys clk
TRST: optional reset for TAP controller
TDI/TDO: Test data input and output



Boundary Scan: In System Design

1. Load instruction register
2. Execute commands



Lecture 7

IP Block Design using HLS

Objectives

After completing this lecture and the lab, you will be able to ...

- ...explain, why you should use HLS
- ...explain, where HLS makes sense, and where manual RTL coding is better used.
- ...explain, how HLS supports a designer in managing the main design challenges (time to market, complexity, heterogeneity)
- ...name the major steps in the HLS design flow and to run a HLS design.
- ...understand the main metrics applied for performance evaluation.
- ...apply directives and know, how directives influence the result of your design.

What is high-level

Synthesis (HLS)

Why should we use HLS

Creates an RTL implementation from C level source

Many options: smaller designs, faster designs, optional designs

Time to market: spend time for careful design of functionality -> let tool to implement Design accelerator (Beschleuniger): HLS

- Generates HDL directly from C Code.
- Method for creating functional modules or IP blocks for reuse

Important side effect:

Mechanism for generating and assessing variations on the hardware implementation => fast and easy way to find best architecture.

Manage Complexity: HLS = method for raising **abstraction level**

High Level: Algorithm in C, C++, SystemC code

Behavioral: Algorithm in VHDL or Verilog

RTL: hides technology details, but still describes registers and combinational logic between registers

Structural: instantiating, configuring, connecting hardware elements

Manage Heterogeneity: HLS = skip the design part, which includes largest number of tools and methods

Benefit of Raising Abstraction Level

- Simpler direction:** Description of circuit becomes simpler. Description is done in an environment, which is known to designers. Functionality is separated from implementation.
- Simpler comparison:** Implementation process can be directed by designer with simple directives instead of re-writing code. Source code does not fix architecture
- Simpler description:** High level synthesis allows for rapid generation of implementation variants. Architecture variations can be created and compared quickly.

Summary

Raising Productivity

Why NOT use: not suitable for later modification, not suitable for best performance

Principles of HLS

Profiling: measures where the CPU is spending it's cycles on a function-by-function or task-by-task basis and informs system designer which software routine may be a candidate to hardware-accelerate

➔ Similar to timing analysis in HW

Profiling

Keep it in SW:

- Not in critical path
- Enough "free" cycles
- Easier to code in software than in hardware
 - Uses math library functions

Move to hardware

- Programmable logic co-processor
- Customized to user's needs
- Excellent for iterative and pipelined processing

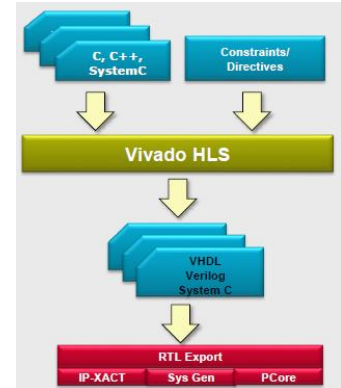
HLS inputs

Source file (HLS source languages: C, C++, SystemC)
 Test bench (c-test bench for hole project)
 Golden reference (expected response => ideally, prepared set of output values as response to specific input values)

Generating Hardware out of C-Code

Interface Synthesis manual process
 Inferred from source code
 Algorithm synthesis extraction of data path and control
 Scheduling and binding
 Optimization

Evaluation	Area:	HW resources, cost	
	Speed:	Clock frequency:	switching frequency
	Latency:	number of clock cycles from input sample to valid output sample.	
Export	Throughput:	rate at which circuit can process data	
	Power consumption		
	I / O requirements:	number and complexity	
	Generated Code: VHDL / Verilog / SystemC		
	Preferred IP-XACT Format others SysGen, PCore		

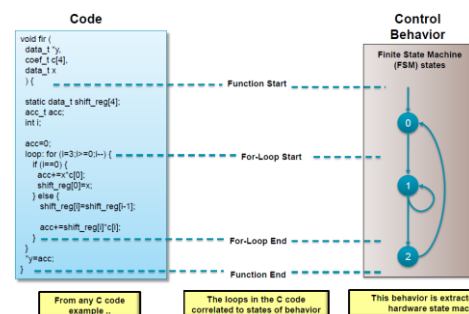


Interface synthesis	Manual process	specified manually
	Interface synthesis	inferred from code

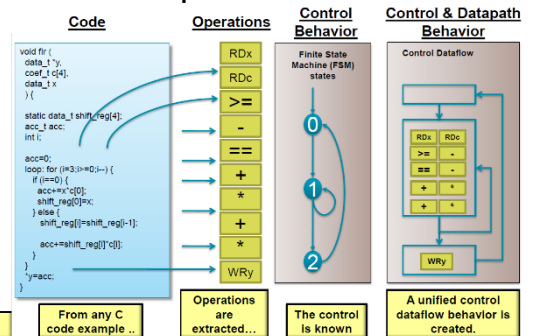
- Algorithm Synthesis
- 1. Extraction of data path and control**
 - 2. Scheduling and binding** (Generate RTL statements from C code and associate for each its duration in terms of clock cycles. Associate physical resources of target device. Iterate if needed)
 - 3. Optimization** (Constraints: clock or resource utilization; Directives: how to treat loops, arrays, etc)

How does HLS work in Vivado

Control Extraction

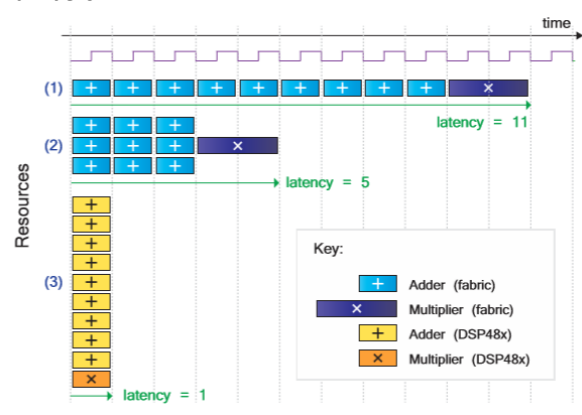


Control & Datapath Extraction



Scheduling & Binding

Example: calculate average of array input consisting of 10 numbers.



Solution 1:

- Latency: 11 cycles
- Throughput: $f_{clk} / 11$
- Resources: 1 add, 1 mult (both in fabric)

Solution 2:

- Latency: 5 cycles
- Throughput: $f_{clk} / 5$
- Resources: 3 add, 1 mult (both in fabric)

Solution 3:

- Latency: 1 cycles
- Throughput: f_{clk}
- Resources: 9 DSP48

Lecture 8

Objectives

Modelling of digital Filters

FIR Filter structure

You can draw the basic FIR filter structures and you know how the structure influences amount of required resources for the realization of a FIR filter

Hardware Requirements

You can list the required hardware elements and you can indicate the number of required elements for various structures.

You know what word length is required for the various stages in the filter.

Filter Hardware in VHDL

You can describe FIR filters in VHDL.

symbols

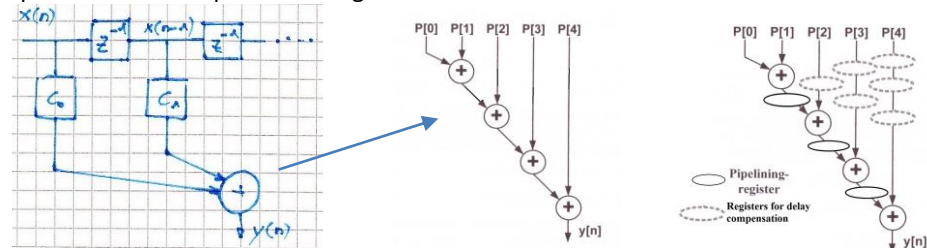
N Order or Filter

j input word length

q_x adder output word length

Filter Structures

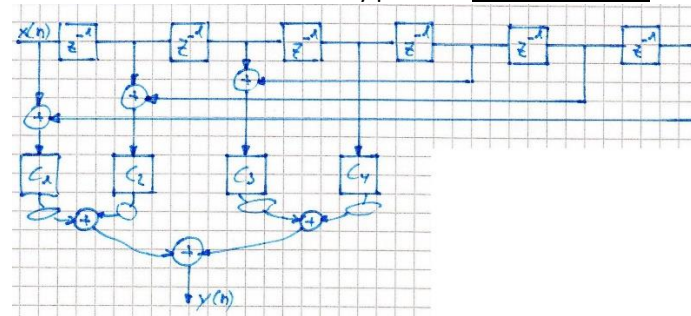
➤ Direct form



$$\#FF = N * j$$

Disadvantages: Long adder tree generates long propagation delay times.
Adder tree may produce timing problems.

➤ linear phase structure



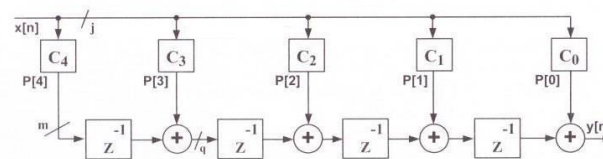
$$\#FF = N * j$$

Symmetric coefficients $c(n) = c(N-n)$

Benefit: linear phase

Constant group delay

➤ Transposed form



Disadvantage: Register size increases towards output

$$\#FF = N * q_x$$

Hardware

Requirement

Delay Elements: realized as register

Normal registers may also be realized as RAM cells

Number of delay elements (registers) = N ($\neq \#FF$: $\#FF$ depends on register width and Filter order)

Adders: per default only 2 inputs are available

Long adder tree generates long propagation delay (may produce timing problems)

Multipliers: use IP Block

Implement by yourself

Control logic: synchronization

Provide data samples at sampling rate, not at processing rate

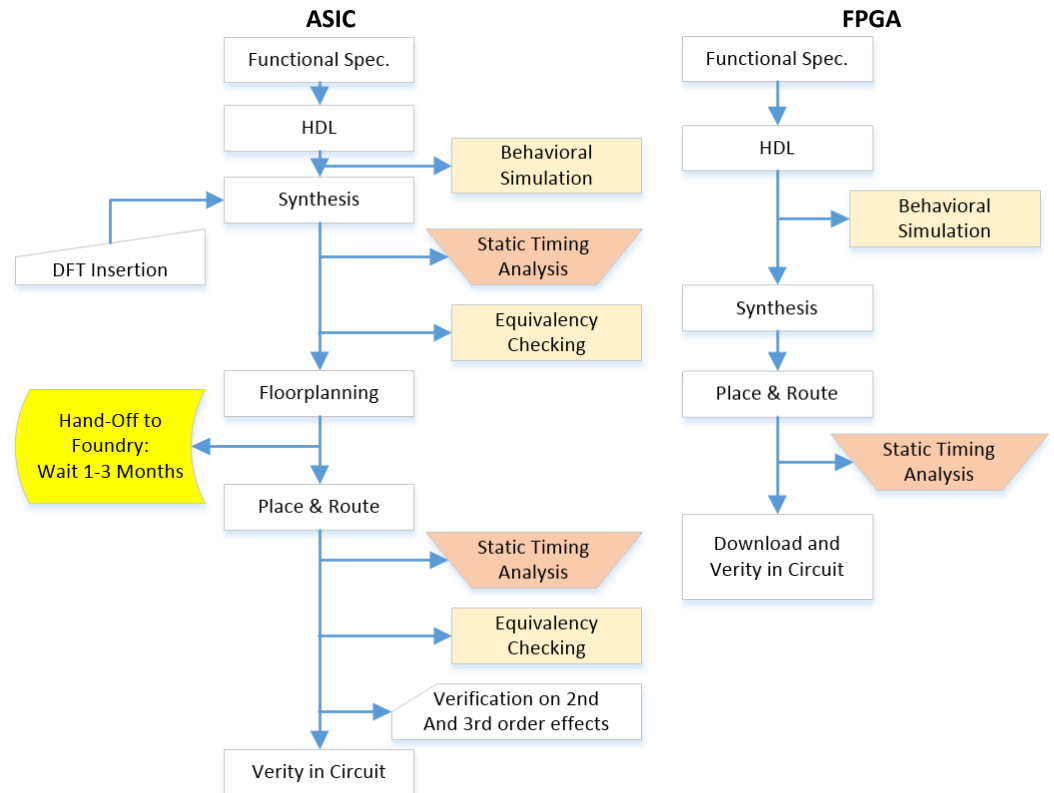
Lecture 9

SoC ASIC Design

Objectives

You know the typical FPGA design flow
 You know the typical ASIC design flow
 You know the reasons, why the two design flows are not the same
 You know the meaning of the individual steps in the design flow for FPGA design as well as for ASIC design

Design Flow Comparison ASIC vs. FPGA



Design Entry

No difference between FPGA and ASIC

Synthesis for FPGA

Select target device (FPGA family, type), tool, behavioural struct in ...

Synthesis for ASIC

+ more choice, more freedom

- expensive infrastructure

- Select designhouse
- Select design tolls (very expensive)
- Select technology: performance, MHz, gates/mm², but does not determines # of gate, # of IO's
- Select digital call library: use one library (only one possible per block, because of their different height.))
- Add constraints

Synthesis (Test logic)

FPGA has built in scan chain

ASIC no built in test

- Testlogic is generated and added during synthesis
- Replace FF by scan FF's
- Support tools available

(Verification)

ASIC synthesis creates new hardware

- Equivalency checking (netlist before, after synthesis)
- Static timing analysis (simulate timing with setup- and hold-times at selected clock frequency)

- Floorplanning
- Schematic placement of major blocks (pads, analog, digital)
- desired aspect ratio of chip
 - bonding pad locations
 - desired placement of analog and digital building blocks, etc.

This step exists for FPGA as well, but is not explicitly called floorplanning. Vivado distinguishes between timing constraints and physical constraints. Setting physical constraints = floorplanning

FPGA Constraints file for pad definition:

```
# CLK source 100 MHz
set_property PACKAGE_PIN Y9 [get_ports clk_pin]
set_property IOSTANDARD LVCMOS33 [get_ports clk_pin]
```

ASIC D Defining aspect ratio:

```
floorPlan -r A D L B R T
```

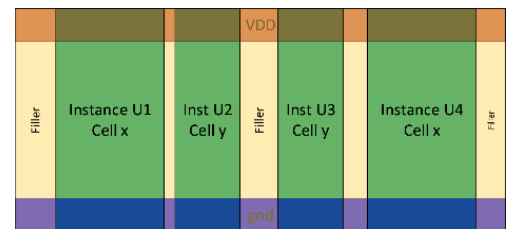
A: aspect ratio; D: density;
L,B,R,T: margin from outside to core box Left, Bottom, Right, Top

- Clock Tree
- FPGA** FPGA = fixed Hardware. Clock Tree is included, tested, and characterized.

ASIC Clock Tree is vital for functionality, speed, and power consumption of chip.
- Clock tree defined in specification file:
Defines parameters such as phase delay, input transition time, clock skew, etc.
Clock tree is synthesized by using buffers from a list of buffer cells defined in the specification file.

- Filler cells
- Filler cells are used to fill up standard cell rows. Necessary for production reasons
- (Continuity of N-well and implant layers, see Analog Microelectronics).

After placing filler cells, standard cell rows shall no longer have «holes».



- Summary

FPGA production Test

FPGA chips are tested. No need for test program development and production test. Download bitfile and verify functionality of circuit by running application test program.

ASIC production test program

Each ASIC is a new device and has to be tested prior to packaging and shipping to customer. ASIC test development is an important task. Test program has a huge impact on quality and cost.