

Hardware Design & Test Flow from Behavior Level Modeling

Wonyong Sung

EDA Tools

EVERY transformation
requires a verification

► Specification

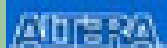
- Simulink
 - Model Based Design

► Transformation

- Synplify DSP
 - DSP Synthesis
 - RTL Implementation
- Synplify PRO
 - RTL Synthesis
 - Gate Implementation

- Xilinx ICE, Altera Quartus
 - Gate Place&Route
 - FPGA Implementation





► Verification

- Simulink
 - Model Simulation
- ModelSim, ActiveHDL,...
 - RTL Simulation
 - Gate Level Simulation
 - Timing Analysis
 - Formal Verification
- FPGA Board
 - Board Simulation



Synplicity

Simply Better Results

Inference vs. Instantiation

- Given that synthesis and implementation are implicitly tool and technology dependent, there often exist a need to trade-off maximum code portability against design optimization within a target technology.

$Z \leq A + B ;$
The "+" operator infers that a generic Adder be built, without consideration of device level optimization.

Code portability is maintained.

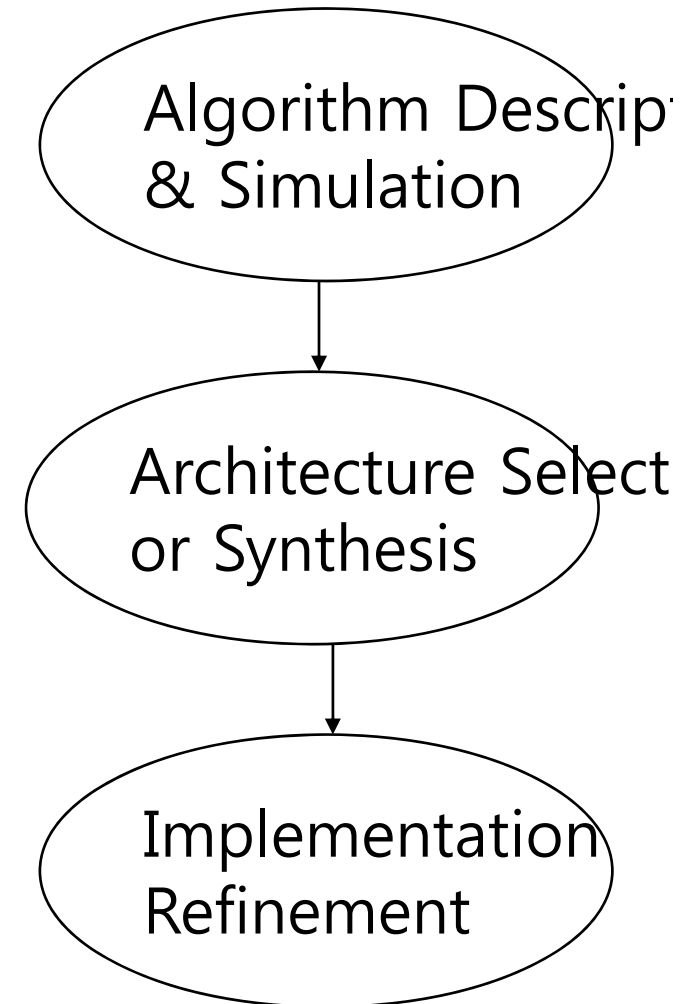


```
component  
ADSUB8  
port (...  
...end component  
This is in effect a direct call to a library macro within the target technology.
```

Device optimization is achieved.

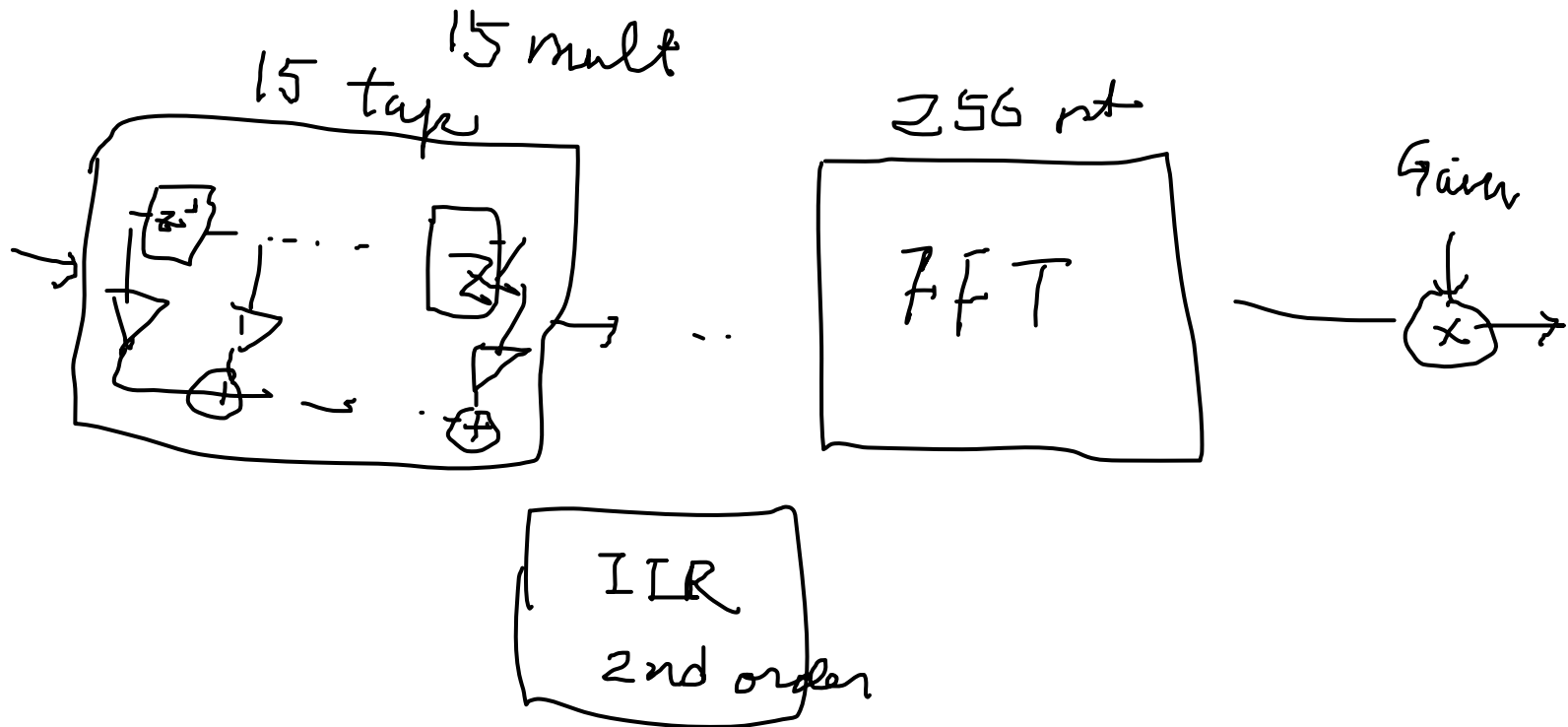
1. Pure Top-down Approach

- You don't care the down flow when you conduct algorithm design
- Pros: Portable, can choose different targets easily.
- Cons: Inefficient design results, Takes much time for synthesis (too much design space)



Pure top-down design

- Scheduling and binding:
 - Map several operations into one hardware block (binding); these operations should not be conducted at the same time slot (scheduling).
 - In FPGA, we need scheduling and binding considering the interconnection delays.
- Floating-point to fixed-point conversion
 - Algorithms are represented in floating-point, but hardware blocks are fixed-point, and fixed-width memory.
 - In FPGA, for example, we need to try to use DSP48 blocks for filtering.



$$f_{\text{system}} = 100 \text{ MHz}$$

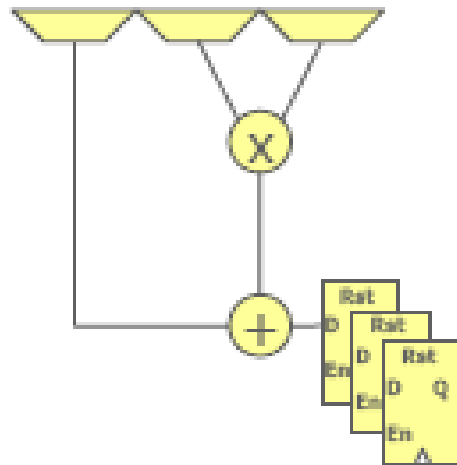
$$f_{\text{sample}} = 10 \text{ MHz}$$

How many resources?

DSP Synthesis Transformations

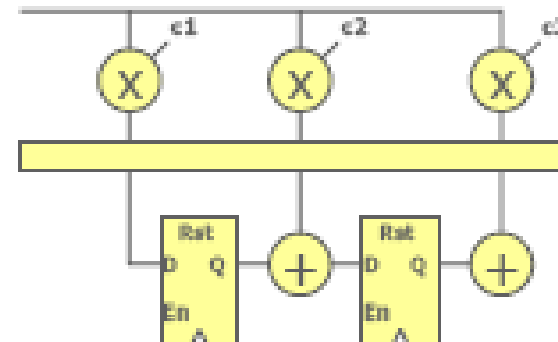
System-level Retiming

- System level re-timing/pipeline insertions
- Improves performance
- Reduces area



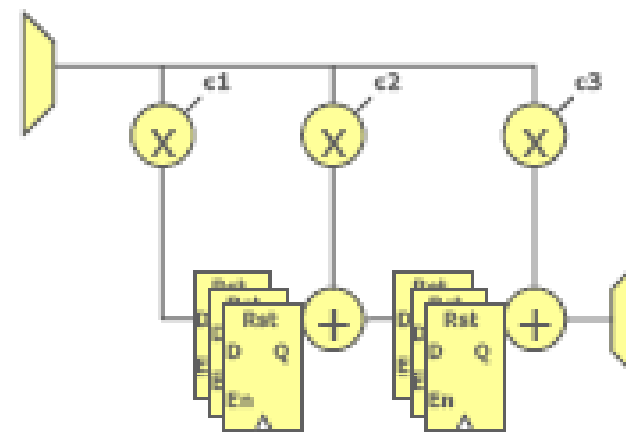
Automatic Multi-channel capability

- Creates a multi-channel system from a single channel spec
- Quickly determine optimal number of channels before implementation



Folding

- Fast, accurate speed-area tradeoffs
- Shares expensive resources like multipliers
- Enables optimization of high-volume designs for cost



Simplicity

Simply Better Results

2. Top-Down Interactive Design (Instantiation approach)

- Uses the blocks that are well supported in lower level designs
 - Xilinx FPGA contains 18*18 multipliers,
 - What if your fixed-point design requires 20*12 bit multipliers?
 - Fixed-point optimization SW may consider that 20*12 will require less number of gates (this is true if we implement them using CLB's).
 - FIR filter compiler, FFT,
- Pros:
 - Design space for architecture synthesis is local – easy to optimize by parameter change.
 - Optimized to certain targets.
 - Easy to apply.
- Cons:
 - Less efficient scheduling and binding results
 - Less portable
 - Not always applicable, if you cannot find blocks.

AccelDSP

- M code (Matlab code) to scheduled (synthesizable, optimized for a given throughput) VHDL
- Fixed-point design support
- Use AccelWare DSP IP

System Generator

- Provides fixed-point simulation model to Simulink
- Provides synthesized IP's
 - Blockset: Xilinx supplied blocks that can be mapped to FPGAs very efficiently. They are usually parameterized.
 - FIR filter, FFT, adaptive filter, ...
 - Core generator
 - Produces blocks optimized in placement and routing.

Coregen Module Instantiation

- Generally speaking, Core generator is useful tool for xilinx architecture base design something like memory, fifo etc. This module will be need to be instantiated. It will generate function simulation model and implementation file. During synthesis, the instantiated module will treat block box, but implementation tool will merger.

```
architecture RTL of My_latch is  
component RAM16X1  
port ( A0,A1, A2, A3, WE, D : in std_logic ;  
        O : out std_logic ;  
end component ;  
begin  
uut : RAM16X1 port map ( A0 =>low, A1=>low,  
        A2 =>low, A3 =>low, WE=>My_We,  
        DI=>My_Di,  
        Do=>My_Do ) ;
```

...

Synplify DSP

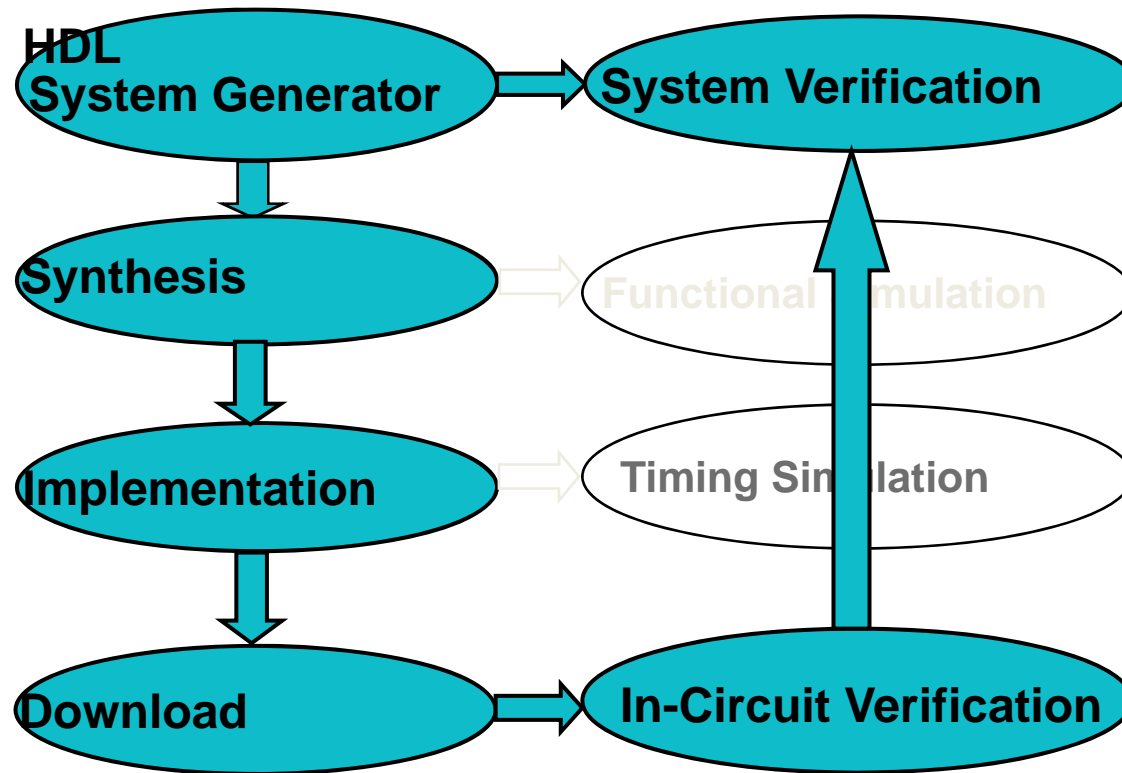
- Similar functionality with AccelDSP/System Generator
- Fixed-point support, various transformations for optimized RTL code generation
- Multiple target support (Actel, Texas Instruments DSP, not attached to Xilinx)

Test of Your Design

- To err is human.
- Behavioral level simulation is much faster than RTL or gate level simulations.
- What are the disadvantages?
 - Less accuracy (bit-accurate?).
 - Timing inaccuracy (cycle accurate?)
- For cycle accurate, bit accurate, and timing accurate, you need simulation with back-annotated design.
- This means that you may find bugs at the lower level! This is against our assumption of top-down design flow.

Hardware in the loop – hardware assisted simulation

MATLAB/Simulink



Files Used

- Configuration file
- VHDL
- IP
- Constraints File



Black Box



JTAG
Compilation

ChipScope (1)

- Software based logic analyzer
 - Get results on the computer
- Put a logic analyzer right into the FPGA
 - ICON – Connects FPGA to software
 - ILA – Does the actual analysis, FPGA implemented logic analyzer
 - VIO – virtual input and output (push button)
- More flexible than the bench analyzers
 - Can create busses
 - Advanced triggering support

ChipScope (2)

- Steps to use ChipScope
 - Generate an ICON
 - Generate an ILA, VIO
 - Connect the ILA to the ICON
 - Synthesize, and implement your design
 - With the ILA and ICON
 - Program the CaLinx board
 - Run the ChipScope Pro Analyzer
 - Runs over the JTAG, not Slave Serial connection!

ChipScope (3)

- Logic Analyzer Similarities/Differences
 - Triggering is similar
 - Can be set to show waves before trigger
 - Can trigger on repeated or combined events
 - Data/Trigger can be MUCH bigger
 - Up to 256bits wide
 - As many samples as Block RAM on the FPGA
 - Data is captured synchronously
 - Can't look at clocks
 - Much easier to view waveforms

ChipScope (4)

- ChipScope is useful to verify
 - In this lab we're using it just to make absolutely sure
- You will **NEED** ChipScope
 - You cannot debug a large design (i.e. your project) without it
 - Bench analyzers won't show enough signals
- It helps to master use of HP Logic Analyzer and ChipScope early on.
 - You want to have ready knowledge of both tools for when you're working on the project.

ChipScope (4)

- Detailed ChipScope Tutorial
 - <http://inst.eecs.berkeley.edu/~cs150/sp06/Documents.php#Tutorials>
- Get used to reading technical documents and tutorials.
 - It's a useful and necessary job skill for engineers.