Using Hard Macros To Reduce FPGA Compilation Time

BYU Department of Electrical & Computer Engineering

Christopher Lavin, Marc Padilla, Subhrashankha Ghosh,
Brent Nelson, Brad Hutchings, and Michael Wirthlin

CHREC NSF Center for High-Performance Reconfigurable Computing

Paper Summary

Conventional Xilinx Flow

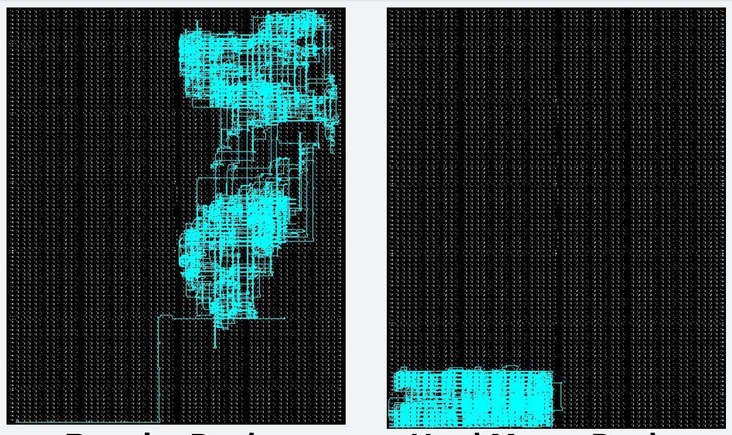
VS.

A Hard Macro-based Design Flow

- FPGA compilation is very time consuming
- *Hard Macros can accelerate the design process by 3X*
- Propose a new design flow based on hard macros—enabling fast FPGA compilation

What is a Hard Macro?

- Hard Macros are pre-placed pre-routed design blocks created for a specific FPGA family
- All elements of a hard macro move together to maintain the same 'shape' as it is moves
- In Xilinx FPGAs, hard macros exist at the primitive (NCD/XDL) level. Xilinx uses NMC/XDL files to represent them
- Hard macros are most effective when they are compact (see below)

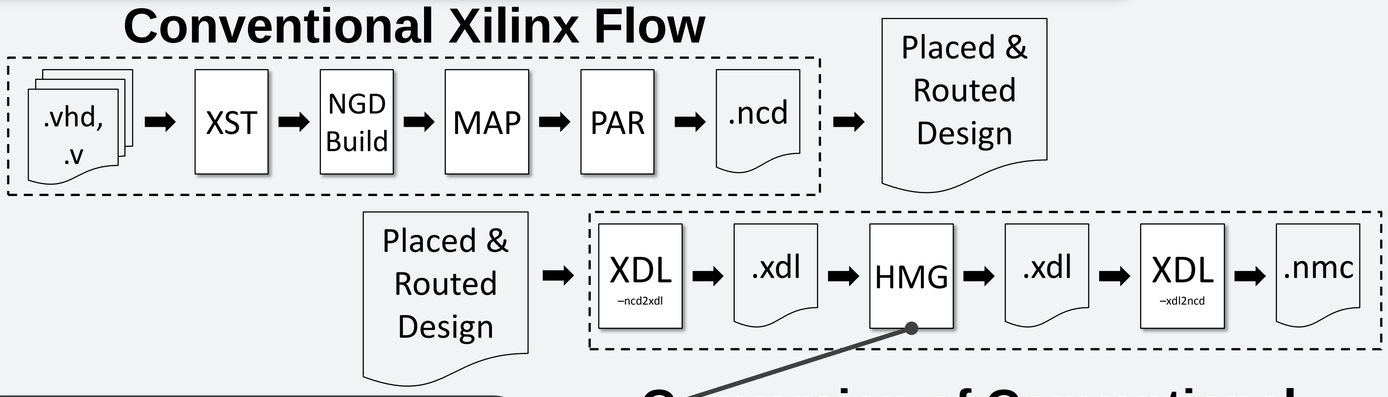


Regular Design

Hard Macro Design

How Do You Make A Hard Macro?

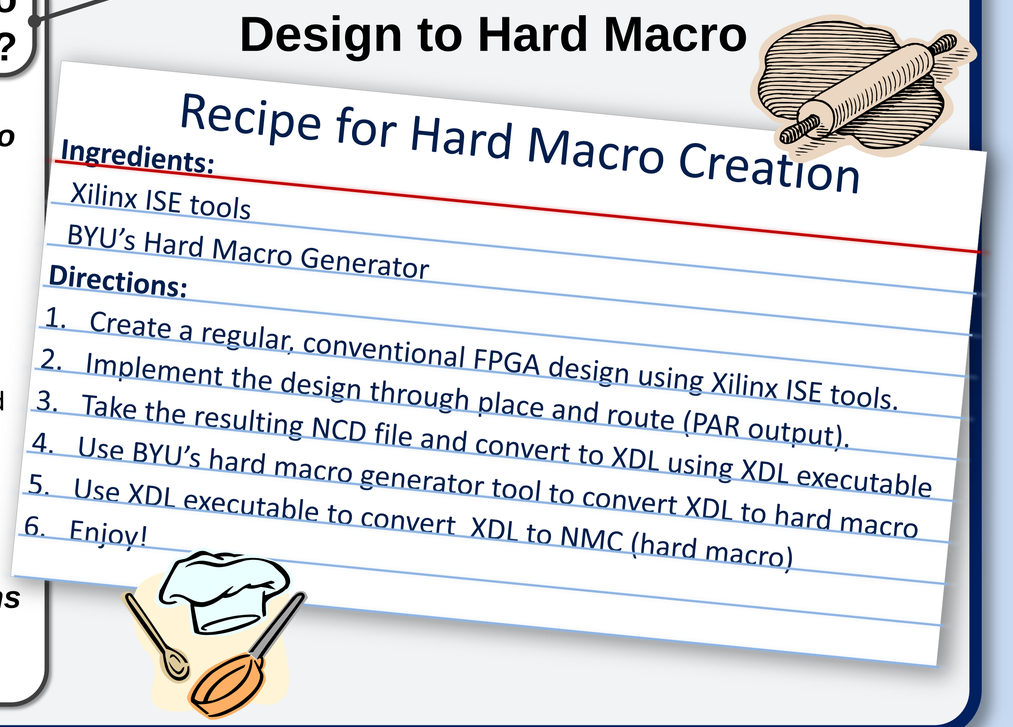
Conventional Xilinx Flow



What Does the Hard Macro Generator (HMG) Do?

- ***Several challenges exist to convert a conventional design to a hard macro:***
  - Xilinx has several undocumented, unsupported features, such as:
    - No ground or power nets
    - No TIEOFF primitives
    - Only one port per net
    - …
  - The HMG identifies these issues and properly addresses them to create valid hard macros
  - The HMG also locates all IOBs in a design and replaces them with appropriate Hard Macro ports
- ***Follows correct XDL conventions for representing a hard macro using BYU's XDL framework***

Conversion of Conventional Design to Hard Macro



How Do Hard Macros Make FPGA Compilation *Fast*?

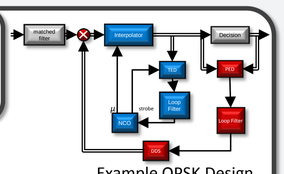
Benefits of Hard Macros

- ***No need for synthesis (XST), technology mapping (NGDBuild), or packing (MAP)***
  - Skipping these steps saves significant part of build time
- ***Potentially faster placement times***
  - Only placing *10's* hard macros instead of *1000's* of primitives
- ***Faster routing times***
  - Hard macros contain all necessary internal routing
  - Hard macro designs only need routing to connect hard macros to each other and to IOs
- ***Offer significant design reuse***
  - Once a hard macro is built, it can be reused in many other designs without the need to rebuild it

Hard Macro Experiments

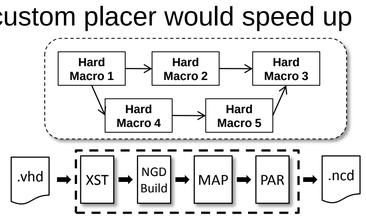
- ***Experiment #1: Determine Xilinx hard macro support***
  - Can Xilinx tools implement designs created purely of hard macros?
- ***Experiment #2: Determine obtainable speedup of a hard macro-based flow***
  - Hand place hard macros to get accurate timing of the routing

Experiment #1: Xilinx Hard Macro Support



Example QPSK Design

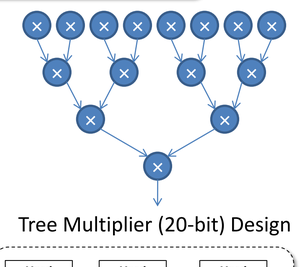
- ***Three different results***
  1. Design placed/routed fine but took longer than regular design
  2. PAR failed, could not find a valid placement
  3. Placement succeeded, but no valid routing could be found
- ***Conclusions***
  - Xilinx PAR is not suitable for hard macro-based designs
    - Placement of hard macros is source of problems
  - Implies creation of our own hard macro-optimized placer
    - Experiment #2 tests if custom placer would speed up compilation time



Conventional Xilinx Flow

Experiment #2: Obtainable Speedup

- ***Place Hard Macros by hand to determine obtainable speedup***
- ***Conclusions***
  - 3X obtainable speedup
  - Custom router will also increase speedup



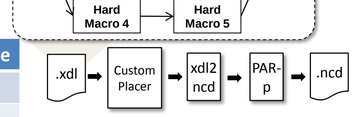
Tree Multiplier (20-bit) Design

Conventional Designs (Baseline Runtimes)

Design	XST	NGDBuild	MAP	PAR	Total Runtime
Mult-tree	46.7s	9.0s	17.7s	64.2s	137.5s
Heterogeneous	80.2s	4.9s	10.4s	35.5s	131.0s

Hard Macro-based Designs

Design	Custom Placer	XDL2NCD	PAR –p (router)	Total Runtime	Speedup (over baseline)
Mult-tree	4.3s	14.3s	25.7s	44.3s	**3.1X**
Heterogeneous	4.3s	12.1s	25.5s	41.9s	**3.1X**



- XST/NGDBuild/Map = 0 secs, PAR reduced
- Assume placer can run very quickly (only a few blocks vs. 1000's)
- XDL design assembler will be fast (trivial operation)

What Kind of Design Flow Will Use Hard Macros?

HM*Flow*