Previous Lecture
An embedded system is nearly any computing system (other than a general-purpose computer) with the following characteristics:

- Single-functioned
  - Typically, is designed to perform predefined functions

- Tightly constrained
  - Tuned for low cost
  - Single-to-fewer components based
  - Performs functions fast enough
  - Consumes minimum power

- Reactive and real-time
  - Must continually monitor the desired environment and react to changes

- Hardware and software co-existence
Major Factors to be Considered

- Time-to-market
- Technology
- Performance
- Cost
- Power consumption
- Reliability
- Testability
- Availability of CAD tools, libraries, IP's
- ...
Simplified and General Embedded System Design Methodology

- Algorithm Functional Modeling
- Problem Partitioning
  - Sw Func. Model
  - SW/HW Interface
  - Hw Func. Model
- Sw. Dev
- Application Source Code
- Processor
  - SW/HW Interface
  - Application Specific Hardware
- Architectural Synthesis
- Logic/Physical Synthesis
Views and Levels of Abstractions

- Views
  - Behavioral
  - Structural
  - Physical

- Abstractions
  - Architectural level
  - Logic level
  - Geometrical level

Gajski and Kahn's Y-chart
Synthesis can be seen as a set of transformation between two axial views:

- Architectural synthesis
- Logic synthesis
- Physical synthesis

Gajski and Kahn’s Y-chart
Agenda

1. Why FPGA?
2. General FPGA Architecture
3. EDA For FPGAs
4. Xilinx EDA
5. Demonstrations
6. Partial Reconfiguration
Agenda

1. Why FPGA?
2. General FPGA Architecture
3. EDA For FPGAs
4. Xilinx EDA
5. Demonstrations
6. Partial Reconfiguration
What is FPGA?

- Integrated Circuit
- **Field-Programmable** Gate Array
  - designed to be configured by a customer or a designer **after manufacturing**
- Contains
  - an array of programmable logic blocks
  - a hierarchy of "reconfigurable interconnects" that allow the blocks to be "wired together"

https://en.wikipedia.org/wiki/Field-programmable_gate_array
Why FPGA?

- Reconfigurability/Flexibility
- High Performance
- Acceptable power consumption (xx Watts)
- Cost getting lower

---

https://www.actualtech.io/programmable-asic-will-change-infrastructure-investments/


© Akash Kumar
Where do I find FPGA?

INTEL® FPGA PAC N3000 FOR NETWORKING

Accelerates network traffic for up to 100Gbps to support low latency, high bandwidth 5G applications enabling custom tailored solutions for vRAN and Core network workloads with support of end-to-end industry standard and open source tools.

- **High Performance**
  - 1.1M logic elements
  - 144 MB DDR4

- **Low Latency**
  - 9GB DDR4
  - 4 x 25 Gbps

- **High Bandwidth**
  - 2 Intel® Ethernet Converged Network Adapter XL710 NIC for packet processing

- **Intelligent Offload**
  - 1/2 length, full height, PCIe card

© Akash Kumar
The Evolution of FPGA (Xilinx)
Integration in System Design

Programmable Systems usher in a new era of system design integration possibilities
Development Effort
Agenda

1. Why FPGA?
2. General FPGA Architecture
3. EDA For FPGAs
4. Xilinx EDA
5. Demonstrations
6. Partial Reconfiguration
Simplified FPGA Architecture

- **Functional Blocks**
  - Implement any logic function
    - \( F(a,b,c,d,e,f) = ab + cd + bde \)
    - \( F(a,b,c,d,e,f) = abcd + ef \)
    - \( F(a,b,c,d,e,f) = a + b + c + df \)
    - ....

- **Routing Network/Switchbox**
  - Connect blocks

- **I/O Blocks**
  - Communicate with the outside world

- Everything is reconfigurable
FPGA Reconfiguration

- Bitstream ~ binary executable
- Contains the configuration data for all blocks

Download Bitstream
The most basic element of FPGA

Based on Lookup Table (LUT)

Differentiation between FPGAs

- Number of inputs to LUT
  - State-of-the-art: 6
- Some extra circuits
Consider a 4-input LUT

SRAM
- 16 cells ($2^4$)
- Implement Truth Table
- Any 4-input function
- $2^2^4$

Flip-flop (FF)
- Storage element for sequential circuit
A truth table is a mathematical table used in logic—specifically in connection with Boolean algebra, boolean functions, and propositional calculus—which sets out the functional values of logical expressions on each of their functional arguments, that is, for each combination of values taken by their logical variables (Enderton, 2001). (Wikipedia)

**AND Gate**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**OR Gate**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**NOR Gate**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
LUT Content

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( F = A + B + C + D \)
LUT Content

\[
F = A + B + C + D
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Xilinx Configurable Logic Block (CLB)

- Each CLB contains
  - 2 Slices
  - Carry-look-ahead unit
  - 8 Flip-flops
  - MUXes
- Each Slice contains 4 6-input LUTs
- Each LUT contains 2 5-input LUTs
Slice Internal Structure

- LUT
- Multiplexer to select inputs to the Carry Chain
- Carry Chain
- Multiplexer to select which signal to pass to the output
- Flip-flop
Additional Resources

- Block RAM
  - Store large data
- Carry-chain
  - Faster Adder with carry-look-ahead
- DSP (Digital Signal Processing)
  - Multiplication, MAC
- Processor
- ....

Additional Resources – Xilinx Zynq
Xilinx Virtex6
A functional block **input or output pin** can connect to some or all of the wiring segments in the channel adjacent to it via a connection block of programmable switches.

At every intersection of a horizontal channel and a vertical channel, there is a switch block.

- A set of **programmable switches** that allow some of the wire segments incident to the switch block to be connected to others.
- By turning on the appropriate switches, short wire segments can be connected together to form longer connections.
Routing Wires

- Some FPGAs contain routing architectures that include different lengths of wires.
- The length of a wire is the number of functional blocks it spans.
- Long wires introduces shorter delays for long interconnections since fewer switch blocks will be passed.
Xilinx Zynq Routing Architecture Example
Clock Tree Distribution

- Dedicated wires for clock
  - Reduce jitter
  - Low latency
  - High fan-out
- Multiple clock regions
  - Organized in vertical and horizontal tree-like topology
# Example of Commercial FPGAs from Xilinx

<table>
<thead>
<tr>
<th>Programmable Logic (PL)</th>
<th>Artix-7</th>
<th>Artix-7</th>
<th>Artix-7</th>
<th>Kintex-7</th>
<th>Kintex-7</th>
<th>Kintex-7</th>
<th>Kintex-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Cells</td>
<td>23K</td>
<td>55K</td>
<td>65K</td>
<td>28K</td>
<td>74K</td>
<td>85K</td>
<td>125K</td>
</tr>
<tr>
<td>Look-Up Tables (LUTs)</td>
<td>14,400</td>
<td>34,400</td>
<td>40,600</td>
<td>17,600</td>
<td>46,200</td>
<td>53,200</td>
<td>78,600</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>28,800</td>
<td>68,800</td>
<td>81,200</td>
<td>35,200</td>
<td>92,400</td>
<td>106,400</td>
<td>157,200</td>
</tr>
<tr>
<td>Total Block RAM (# 36Kb Blocks)</td>
<td>1.8Mb</td>
<td>2.5Mb</td>
<td>3.8Mb</td>
<td>2.1Mb</td>
<td>3.3Mb</td>
<td>4.9Mb</td>
<td>9.3Mb</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>66</td>
<td>120</td>
<td>170</td>
<td>80</td>
<td>160</td>
<td>220</td>
<td>400</td>
</tr>
<tr>
<td>PCI Express</td>
<td>Gen2 x4</td>
<td>Gen2 x4</td>
<td>Gen2 x4</td>
<td>Gen2 x4</td>
<td>Gen2 x8</td>
<td>Gen2 x8</td>
<td>Gen2 x8</td>
</tr>
</tbody>
</table>

Analog Mixed Signal (AMS) / XADC(2)

2x 12 bit, MSPS ADCs with up to 17 Differential Inputs

AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config

<table>
<thead>
<tr>
<th>Speed Grades</th>
<th>Commercial</th>
<th>Extended</th>
<th>Industrial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1</td>
<td>-1</td>
<td>-2, -2L</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td>-2, -L</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>-1, -2</td>
<td>-1, -2, -1L</td>
<td>-1, -2, -2L</td>
</tr>
</tbody>
</table>

© Akash Kumar
## Example of Commercial FPGAs from Xilinx

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Part Number</th>
<th>Cost-Optimized Devices</th>
<th>Mid-Range Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z-7007S</td>
<td>XC7Z027S</td>
<td>Single-Core</td>
<td>Z-7030</td>
</tr>
<tr>
<td>Z-7012S</td>
<td>XC7Z027S</td>
<td>ARM® Cortex™-A9 MPCore™</td>
<td>Z-7035</td>
</tr>
<tr>
<td>Z-7014S</td>
<td>XC7Z027S</td>
<td>Up to 766MHz</td>
<td>Z-7045</td>
</tr>
<tr>
<td>Z-7010</td>
<td>XC7Z015</td>
<td>Dual-Core ARM</td>
<td>Z-7100</td>
</tr>
<tr>
<td>Z-7015</td>
<td>XC7Z015</td>
<td>Cortex-A9 MPCore</td>
<td></td>
</tr>
<tr>
<td>Z-7020</td>
<td>XC7Z020</td>
<td>Up to 866MHz</td>
<td></td>
</tr>
</tbody>
</table>

### Processor Core
- **NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor**

### Processor Extensions
- **L1 Cache**: 32KB Instruction, 32KB Data per processor
- **L2 Cache**: 512KB
- **On-Chip Memory**: 256KB

### External Memory Support
- DDR3, DDR3L, DDR2, LPDDR2

### DMA Channels
- 8 (4 dedicated to PL)

### Peripherals
- 2x UART, 2x CAN 2.0B, 2x I²C, 2x SPI, 4x 32b GPIO

### Security
- RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot

### Processing System to Programmable Logic Interface Ports
- 2x AXI 32b Master, 2x AXI 32b Slave
- 4x AXI 64b/32b Memory
- AXI 64b ACP
- 16 Interrupts
Agenda

1. Why FPGA?
2. General FPGA Architecture
3. EDA For FPGAs
4. Xilinx EDA
5. FPGA Platform Examples
6. Partial Reconfiguration
FPGA Design Flow

Architecture Independent

1. Design the circuit
   - Design the circuit using VHDL or Verilog in Xilinx ISE or Vivado.

2. Synthesis
   - Execute FPGA synthesys tools to convert the HDL codes into netlists.

3. Technology Mapping
   - Execute FPGA technology mapping tool to map the netlists into available FPGA resources.

Architecture Dependent

4. Placement and Routing
   - Execute FPGA Placement & Routing tools to place and connect the resources mapping done in the previous step onto the FPGA.

5. Generate Bitstreams
   - The configuration data for the FPGA is generated in the form of bitstream.

6. Reconfigure FPGA
   - Load the bitstream to the FPGA to run the circuit designed in the first step.
Synthesis

- Architectural synthesis
- Logic synthesis
Design should meet the following requirements
- Design must function at the specified speed
- Design must fit in the targeted device

Using synthesis tools, you can
- Use its area report to estimate device utilization
  - Generally accurate because the synthesis tool creates the logic from your code and maps your design into the FPGA
- Use its timing report to estimate performance.
  - They are estimated because they are based on the logic level delays from the cell libraries and estimated wire-load models.
  - This report is an estimate of how close you are to your timing goal; however, they are not the actual timing for your design.
Design Constraints

- The constraints are used to instruct the EDA tools
  - The desired clock frequency
  - The locations of input/output signals, i.e., which pins of the FPGA that these signals should be connected to
  - Timing characteristics on some specific paths
  - The specific locations on the FPGA that some modules should be placed inside (floorplanning)
- Xilinx Design Constraint (.xdc) file written in TCL language
Technology Mapping

- Technology step restructures the primitive logic gates, generated from the logic optimization step, into sets of 4-input functional blocks.

(We assume one functional block contains only one logic element in this example)
Synthesis + Technology Mapping Netlist
Placement and Routing

- The placement step finds physical locations for functional blocks.
- The routing step finds physical routes for logic connections.
PnR Considerations

- The overall goal when placing and routing your design is fast implementation and high quality results. However, depending on the situation and your design, you may not always accomplish this goal.
- Earlier in the design cycle, run time is generally more important than the quality of results, and later in the design cycle, it’s reversed.
- If the targeted device is highly utilized, the routing may become congested, and your design may be difficult to route. In this case, the placer and router may take longer to meet your timing requirements.
- If design constraints are strict, it may take longer to correctly place and route your design, and meet the specified timing.
- Placement and Routing report gives your accurate device utilization.
PnR Results

After PnR

After Synthesis
Agenda

1. Why FPGA?
2. General FPGA Architecture
3. EDA For FPGAs
4. Xilinx EDA
5. Demonstrations
6. Partial Reconfiguration
When the first FPGA was invented, Xilinx did not focus on design automation tools to ease the logic design process. They provided the physical design tool called XACT 1.0, later known as Xilinx Device Editor. The designers could have access to all of the FPGA's logic and routing resources. The functionality of the chip had to be defined through manual manipulations of these resources.

Ten years later, Xilinx acquired NeoCAD to gain access to its advanced FPGA implementation technology. The designers could be able to describe the design in higher-level language, Verilog or VHDL, instruct the tools to automatically map to FPGA, place and route the design and finally generate the bitstream.

Xilinx kept on developing the set of tools with the name ISE (Integrated Synthesis Environment) for 15 years.

It is only until 2012 when Vivado Design Suites was introduced to replace ISE. The High Level Synthesis (HLS) feature (developed by autoESL and later acquired by Xilinx) is integrated tightly to the design flow to synthesize the design from the compatible C code.
Xilinx Vivado EDA Tools
Xilinx Vivado – Traditional Flow with HDL
Xilinx Vivado – Electronic System-Level (ESL) Design
Xilinx Vivado HLS
Xilinx SDSoC (Software-defined System-on-Chip)
Reference Materials
Agenda

1. Why FPGA?
2. General FPGA Architecture
3. EDA For FPGAs
4. Xilinx EDA
5. Demonstrations
6. Partial Reconfiguration
Zynq-7000 SoC Features

- Dual ARM® Cortex™-A9 MPCore™ with CoreSight™
- 32 KB Instruction, 32 KB Data per processor L1 Cache
- 512 KB unified L2 Cache
- 256 KB On-Chip Memory
- 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO
- 2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO on-chip peripherals
- 28K logic cells (4400 logic slices, each with four 6-input LUTs and 8 flip-flops)
- 240 Kbits of fast block RAM
- Four clock management tiles, each with phase-locked loop (PLL)
- 80 DSP slices
- Internal clock speeds exceeding 450MHz
- 2x 12 bit, 1 MSPS On-chip analog-to-digital converter (XADC)
Ultra96

- **Features**
  - Xilinx Zynq UltraScale+ MPSoC ZU3EG SBVA484
  - Micron 2 GB (512M x32) LPDDR4 Memory
  - Delkin 16 GB MicroSD card + adapter
    - Pre-loaded with Embedded Linux plus Enlightenment Desktop
  - Wi-Fi / Bluetooth
  - Mini DisplayPort (MiniDP or mDP)
  - 1x USB 3.0 Type Micro-B upstream port
  - 2x USB 3.0 Type A, 1x USB 2.0 HS Mezzanine downstream ports
  - 40-pin 96Boards Low-speed expansion header
  - 60-pin 96Boards High speed expansion header
  - 85mm x 54mm form factor
  - Linaro 96Boards Consumer Edition compatible

- **Target Applications**
  - Artificial Intelligence
  - Machine Learning
  - IoT/Cloud connectivity for add-on sensors
PYNQ is an open-source project from Xilinx® that makes it easy to design embedded systems with Xilinx Zynq® Systems on Chips (SoCs).

Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors in Zynq to build more capable and exciting embedded systems.

PYNQ users can now create high performance embedded applications with:
- parallel hardware execution
- high frame-rate video processing
- hardware accelerated algorithms
- real-time signal processing
- high bandwidth IO
- low latency control

http://www.pynq.io/
Agenda

1. Why FPGA?
2. General FPGA Architecture
3. EDA For FPGAs
4. Xilinx EDA
5. Demonstrations
6. Partial Reconfiguration
Reconfiguration Latency

The uncompressed/unencrypted bitstream sizes of some FPGAs and the respective configuration latencies using the fastest interface.
Context Switching

- In GPP, the average direct context switch cost is only 3.8 µs
- It is 2300 times faster than the fastest reconfiguration time for the smallest FPGA
Partial Reconfiguration

- Reconfiguring part of the FPGA without affecting other components

- Benefits
  - Use smaller FPGA
  - If device is large enough, hide the reconfiguration latency by having cache-like mechanism to preload the accelerators to the available slots
PR Design Flow

- **Design the circuit**: Design the circuit using VHDL or Verilog in Xilinx ISE or Vivado.
- **Synthesis**: Execute FPGA synthesys tools to convert the HDL codes into netlists.
- **Technology Mapping**: Execute FPGA technology mapping tool to map the netlists into available FPGA resources.
- **Placement and Routing**: Execute FPGA Placement & Routing tools to place and connect the resources mapping done in the previous step onto the FPGA.
- **Generate Bitstreams**: The configuration data for the FPGA is generated in the form of bitstream.
- **Reconfigure FPGA**: Load the bitstream to the FPGA to run the circuit designed in the first step.
Floorplanning

- The constraints are used to instruct the EDA tools
  - The desired clock frequency
  - The locations of input/output signals, i.e., which pins of the FPGA that these signals should be connected to
  - Timing characteristics on some specific paths
  - The specific locations on the FPGA that some modules should be placed inside (floorplanning)

```bash
# Placement constraint for a module
create_pblock pblock_rp
add_cells_to_pblock [get_pblocks pblock_rp] [get_cells -quiet [list rp]]
resize_pblock [get_pblocks pblock_rp] -add [SLICE_X0Y0:SLICE_X2Y4]
resize_pblock [get_pblocks pblock_rp] -add [DSP40_X0Y0:DSP40_X0Y17]
resize_pblock [get_pblocks pblock_rp] -add [RAMB18_X1Y0:RAMB18_X1Y17]
resize_pblock [get_pblocks pblock_rp] -add [RAMB36_X1Y0:RAMB36_X1Y5]
```
Floorplanning

8 PRRs

15 PRRs
Automatic Floorplanner
What’s else?

Summary

- In FPGA, almost everything is reconfigurable
- The most basic element is LUT
- Routing resources take significant area of FPGA
- Designing with FPGA is getting much easier with EDA tools
- Partial reconfiguration is key
“Homework”

- Why can one LUT implement $2^2^n$ functions? (n is the number of inputs)
- FPGA Vendors? What do they do?
- Which FPGA devices/families are being offered by Xilinx?
- How many ways are there to reconfigure Xilinx FPGA? What are the corresponding speeds?
- Can FPGA be used for IoT?
- Install Xilinx SDSoC (you may request for 30-day evaluation before a license server is setup for you to access)