CS250
VLSI Systems Design

Fall 2020

John Wawrzynek

with

Arya Reais-Parsi
### Implementation Alternatives

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full-custom:</strong></td>
<td>All circuits/transistors layouts optimized for application.</td>
</tr>
<tr>
<td><strong>Standard-cell:</strong></td>
<td>Arrays of small function blocks (gates, FFs) automatically placed and routed.</td>
</tr>
<tr>
<td><strong>Gate-array (structured):</strong></td>
<td>Partially prefabricated wafers customized with metal layers or vias.</td>
</tr>
<tr>
<td><strong>FPGA:</strong></td>
<td>Prefabricated chips customized with loadable latches or fuses.</td>
</tr>
<tr>
<td><strong>Microprocessor:</strong></td>
<td>Instruction set interpreter customized through software.</td>
</tr>
<tr>
<td><strong>Domain Specific Processor:</strong></td>
<td>Special instruction set interpreters (ex: DSP, NP, GPU, TPU).</td>
</tr>
</tbody>
</table>

By “ASIC”, most people mean “Standard-cell” based implementation.

**What are the important metrics of comparison?**
The Important Distinction

• “Instruction” Binding Time

  • When do we decide what operation needs to be performed?

  - **“Hardware”**
    - **Media:** Custom VLSI, Gate Array
    - **Binding Time:** first mask, metal masks
    - **One Time Prog.**
    - **Fabrication Time:**

  - **“Software”**
    - **FPGA, Processors**
      - **Config.:** load, every cycle

• **General Principles**
  
  *Earlier the decision is bound, the less area, delay/energy required for the implementation.*
  
  *Later the decision is bound, the more flexible the device.*
Full-Custom

- Circuit styles and transistors are custom sized and drawn to optimize die, size, power, performance.
- High NRE (non-recurring engineering) costs
  - Time-consuming and error prone layout
- Optimizing for small die can result in low per unit costs, extreme-low-power, or extreme-high-performance.
- Common for analog design.
- Requires full set of custom masks.
- High NRE usually restricts use to high-volume applications/markets or highly-constrained and cost insensitive markets.
Standard-Cell*

- Based around a set of pre-designed (and verified) cells
  - Ex: NANDs, NORs, Flip-Flops, counters, buffers, ...
- Each cell comes complete with:
  - layout (perhaps for different technology nodes and processes),
  - Simulation, delay, & power models.
- Chip layout is automatic, reducing NREs (usually no hand-layout).
- Requires full set of masks - nothing prefabricated.
- Non-optimal use of area and power, leading to higher per die costs than full-custom.
- Commonly used with other predesigned blocks (large memories, I/O blocks, etc.)
Modern ASIC Methodology and Flow

- **RTL Synthesis Based**
  
  HDL specifies design as combinational logic + state elements

  Cell instantiations needed for blocks not inferred by synthesis (typically RAM)

  Event simulation verifies RTL

  “Formal” verification compares logical structure of gate netlist to RTL

Place & route generates layout

Timing and power checked statically or dynamically

Layout verified with LVS and GDRC
Semi-Custom Chip Implementations

- Ex: standard practice in microprocessors was that data-paths were full-custom and control (instruction decode, pipeline control) in standard-cells. Now all generated with standard cells.

Control ("random") logic difficult to "regularize". Relatively small percentage of die area/power. Permits late binding of design changes.
Gate Array

- Store prefabricated wafers of “active” & gate layers & local interconnect, comprising, primarily, rows of transistors. Customize as needed with “back-end” metal processing (contact cuts, metal wires). Could use a different factory.

- CAD software understands how to make gates, but also possible to customize at the transistor circuit level.
Gate Array

- Shifts large portion of design and mask NRE to vendor.
- Shorter design and processing times, reduced time to market.
- Highly structured layout with fixed size transistors leads to large sub-circuits (ex: Flip-flops) and higher per die costs.
- Memory arrays are particularly inefficient, so often prefabricated, also:

  Sea-of-gates, structured ASIC, master-slice.
Field Programmable Gate Arrays

- Two-dimensional array of simple logic- and interconnection-blocks.
- Typical architecture: LUTs implement any function of n-inputs (n=3 in this case).
- Optional Flip-flop with each LUT.

- Fuses, EPROM, or Static RAM cells are used to store the “configuration”.
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Many FPGAs include special circuits to accelerate adder carry-chain and many special cores: RAMs, MAC, Enet, PCI, SERDES, ...
FPGA versus ASIC

- **ASIC**: Higher NRE costs (10’s of $M). Relatively Low cost per die (10’s of $ or less).
- **FPGAs**: Low NRE costs. Relatively low silicon efficiency ⇒ high cost per part (> 10’s of $ to 1000’s of $).
- **Cross-over volume** from cost effective FPGA design to ASIC was often in the 100K range.

But, there’s more to the story. What’s the value of “reconfigurability”?
System-on-chip (SOC)

- Brings together: standard cell blocks, custom analog blocks, processor cores, memory blocks, embedded FPGAs, ...
- Standardized on-chip buses (or hierarchical interconnect) permit “easy” integration of many blocks.
  - Ex: AMBA, Sonics, ...
- “IP Block” business model: Hard- or soft-cores available from third party designers.
- ARM, inc. is the shining example. Hard- and “synthesizable” RISC processors.
- ARM and other companies provide, Ethernet, USB controllers, analog functions, memory blocks, ...

- Pre-verified block designs, standard bus interfaces (or adapters) ease integration - lower NREs, shorten TTM.
FPGA Overview

- Basic idea: two-dimensional array of logic blocks and flip-flops with a means for the user to configure (program):

1. the interconnection between the logic blocks,
2. the function of each block.
Original FPGA

- Invented in 1985 by Ross Freeman after founding Xilinx

xc2064: 64 configurable logic blocks, 58 user input/outputs
xc2064

Figure 4. Configurable Logic Block

Figure 7a. General-Purpose Interconnect
Commercial FPGA Chips

Ball Grid Array (BGA) Flip-Chip Package
Why are FPGAs Interesting?

- Technical viewpoint:
  - For hardware/system-designers, like ASICs - only better: “Tape-out” new design every few minutes/hours.
  - “reconfigurability” or “reprogrammability” may offer other advantages over fixed logic?
  - In-field reprogramming? Dynamic reconfiguration? Self-modifying hardware, evolvable hardware?
Why are FPGAs Interesting?

- Staggering logic capacity growth (10000x):

<table>
<thead>
<tr>
<th>Year Introduced</th>
<th>Device</th>
<th>Logic Cells</th>
<th>“logic gate equivalents”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>XC2064</td>
<td>128</td>
<td>1024</td>
</tr>
<tr>
<td>2011</td>
<td>XC7V2000T</td>
<td>1,954,560</td>
<td>15,636,480</td>
</tr>
<tr>
<td>2019</td>
<td>VU13P</td>
<td>3,780,000</td>
<td></td>
</tr>
</tbody>
</table>

- Because of the regularity of their design, FPGAs have tracked Moore’s Law better than any other programmable device - similar to memory chips.
Why are FPGAs Interesting?

- Logic capacity now only part of the story: on-chip RAM, high-speed I/Os, “hard” function blocks, ...
- Modern FPGAs are “reconfigurable systems”
Early Computational Success

- FPGAs came to market for “glue logic”, but researchers soon found they had value as computing devices:

FPGA-based Computing Engines

- PAM (DEC PRL) (1991)
- SPLASH and SPLASH-II (Brown, SRC) (1991)
- CM-2X (SRC) (1993)
- PRISM and PRISM-II (Brown)
- ...

Applications

- encryption/decryption
- compression/decompression
- sequence/string matching
- sorting
- video and image processing
- physical system simulation

“Supercomputer level performance at orders of magnitude lower costs”
FPGAs are in widespread use

- Far more different designs are implemented in FPGA than in custom chips.
And in the Data-Center

Field programmable gate arrays (FPGAs) are taking computing to new heights by offering engineers the ability to program digital logic in the field on a chip many times — from anywhere. Here’s what a system-level integrated circuit like an FPGA can do for you.

- **Accelerating Artificial Intelligence**
  - Machine learning pulls specifics from mountains of data to predict and solve problems. FPGAs make this hyper-efficient, helping businesses parse data for cost savings and revenue growth by retrieving and classifying data in real-time.

- **Accelerating Networks**
  - More data than ever will arrive with 5G and Intel® FPGAs will help you process it faster. How? FPGA flexibility. By driving fiber deep speeds into the network, FPGAs will increase throughput for higher bandwidth.

- **Accelerating Databases**
  - FPGAs take technology to the edge and back. And they bring back tons of data. Then, used in databases, high-performance FPGAs can extract maximum value from data analytics.

- **Accelerating the Data Center**
  - Storage systems today need to be efficient and high-performance. FPGAs accelerate the data center with light-speed data transaction and storage processing to alleviate bottlenecks.

Using Intel® Arria 10 FPGAs, ZTE enhanced performance 10x to achieve a record-setting thousand images per second in face recognition with “theoretical high accuracy.”

FPGAs offer the flexibility, performance, and scalability needed for cost-effective 5G solutions.

High-performance computing with FPGAs leads to reduced latency in software algorithms to deliver real-time analysis of collected data.

Intel® Stratix 10 FPGAs’ hard and floating point digital processing with Intel® Xeon processors offer higher-performance, lower-latency implementation than centralized and network-based storage.

Image credit: Copyright © 2018 Microsoft Corporation. All rights reserved.
FPGA Variations

- Families of FPGA’s differ in:
  - physical means of implementing user programmability,
  - arrangement of interconnection wires, and
  - the basic functionality of the logic blocks.

- Most significant difference is in the method for providing flexible blocks and connections:
  - Anti-fuse based (ex: Actel)
    - Non-volatile, relatively small
    - fixed (non-reprogrammable)
  - Dual-gate non-volatile memory technology (similar to flash) has also been used:
    - repro programmable
    - larger area
    - more special processing steps
User Programmability

- Latch-based (Xilinx, Altera, ...)
  - Latches are used to:
    1. make or break cross-point connections in the interconnect
    2. define the function of the logic blocks
    3. set user options:
       - within the logic blocks
       - in the input/output blocks
       - global reset/clock
       - “Configuration bit stream” can be loaded under user control

+ reconfigurable
- volatile
- relatively large
- n-type only switch?
Idealized FPGA Logic Block

- 4-input look up table (LUT)
- implements combinational logic functions
- Register (Flip-flop)
- optionally stores output of LUT

Function defined by configuration bit-stream
4-LUT Implementation

- n-bit LUT is implemented as a $2^n \times 1$ memory:
  - inputs choose one of $2^n$ memory locations.
  - memory locations (latches) are normally loaded with values from user’s configuration bit stream.
  - Inputs to mux control are the CLB inputs.
  - Result is a general purpose “logic gate”.
- n-LUT can implement any function of n inputs!

Latches programmed as part of configuration bit-stream
LUT as general logic gate

- An n-lut is a direct implementation of a function truth-table.
- Each latch location holds the value of the function corresponding to one input combination.

**Example: 2-input functions**

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

A 2-lut Implements any function of 2 inputs.

**Example: 4-lut**

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>F(0,0,0,0)</th>
<th>F(0,0,0,1)</th>
<th>F(0,0,1,0)</th>
<th>F(0,0,1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>store in 1st latch</td>
<td>store in 2nd latch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other means for general logic has been tried over the years (particularly while the original patent was in effect).
FPGA Generic Design Flow

- **Design Entry:**
  - HDL (hardware description languages: Verilog, VHDL)

- **Design Implementation:**
  - Logic synthesis (in case of using HDL entry) followed by,
  - Partition, place, and route to create configuration bit-stream file

- **Design verification:**
  - Optionally use simulator to check function,
  - Load design onto FPGA device (cable connects PC to development board), optional “logic scope” on FPGA
  - check operation at full speed in real environment.

https://digitalsystemdesign.in/fpga-implementation-step-by-step/
Clocks have dedicated wires (low skew)

Vdd, GND, and global resets also all “prewired”.
Circuit combinational logic must be “covered” by 4-input 1-output LUTs.

Flip-flops from circuit must map to FPGA flip-flops. (Best to preserve “closeness” to CL to minimize wiring.)

Best placement in general attempts to minimize wiring.
Circuit combinational logic must be “covered” by 4-input 1-output LUTs.

Flip-flops from circuit must map to FPGA flip-flops.
(Best to preserve “closeness” to CL to minimize wiring.)
Best placement in general attempts to minimize wiring.
Example Partition, Placement, and Route

Two partitions. Each has single output, no more than 4 inputs, and no more than 1 flip-flop. In this case, inverter goes in both partitions.

Note: the partition can be arbitrarily large as long as it has not more than 4 inputs and 1 output, and no more than 1 flip-flop.
Xilinx FPGAs (interconnect detail)
Colors represent different types of resources:

- Logic
- Block RAM
- DSP (ALUs)
- Clocking
- I/O
- Serial I/O + PCI

A routing fabric runs throughout the chip to wire everything together.
Configurable Logic Blocks (CLBs)

*Slices define regular connections to the switching fabric, and to slices in CLBs above and below it on the die.*
Primitive: 5-input Look Up Tables (LUTs)

Computes any 5-input logic function.

Timing is independent of function.

Latches set during configuration.
Virtex 6-LUTs: Composition of 5-LUTs

May be used as one 6-input LUT (D6 out) ...

... or as two 5-input LUTS (D6 and D5)
The simplest view of a slice

Four 6-LUTs

Four Flip-Flops

Switching fabric may see combinational and registered outputs.

An actual Virtex slice adds many small features to this simplified diagram. We show them one by one ...
Two 7-LUTs per slice ...

Extra multiplexers (F7AMUX, F7BMUX)
Extra inputs (AX and CX)
Or one 8-LUTs per slice ...

Third multiplexer (F8MUX)

Third input (BX)
Extra muxes to chose LUT option ... 

From eight 5-LUTs ... to one 8-LUT.

Combinational or registered outs.

Flip-flops unused by LUTs can be used standalone.
We can map ripple-carry addition onto carry-chain block.

The carry-chain block also useful for speeding up other adder structures and counters.
Putting it all together ... a SLICEL.

The previous slides explain all SLICEL features.

About 50% of the are SLICELs.

The other slices are SLICEMs, and have extra features.
Recall: 5-LUT architecture ...

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>1</td>
</tr>
<tr>
<td>00001</td>
<td>0</td>
</tr>
<tr>
<td>00010</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11101</td>
<td>0</td>
</tr>
<tr>
<td>11110</td>
<td>0</td>
</tr>
<tr>
<td>11111</td>
<td>1</td>
</tr>
</tbody>
</table>

32 Latches. Configured to 1 or 0.

Some parts of a logic design need many state elements.

SLICEMs replace normal 5-LUTs with circuits that can act like 5-LUTs, but can alternatively use the 32 latches as RAM, ROM, shift registers.
Virtex DSP48E Slice

Efficient implementation of multiply, add, bit-wise logical.
Xilinx Virtex-5

Memory resources:

Flip-flops in Logic blocks

Distributed RAM using LUTs among the CLBs.

Block RAMs (in 4 columns)
A SLICEM 6-LUT ...

Normal 6-LUT inputs.

Memory data input

Normal 5/6-LUT outputs.

Memory data input.

Control output for chaining LUTs to make larger memories.

Memory write address

Synchronous write / asynchronous read
SLICEM adds memory features to LUTs, + muxes.
Distributed RAM Primitives

All are built from a single slice or less.

Remember, though, that the SLICEM LUT is naturally only 1 read and 1 write port.
Block RAM Overview

- 36K bits of data total, can be configured as:
  - 2 independent 18Kb RAMs, or one 36Kb RAM.
- Each 36Kb block RAM can be configured as:
  - 64Kx1 (when cascaded with an adjacent 36Kb block RAM),
    32Kx1, 16Kx2, 8Kx4, 4Kx9, 2Kx18, or 1Kx36 memory.
- Each 18Kb block RAM can be configured as:
  - 16Kx1, 8Kx2, 4Kx4, 2Kx9, or 1Kx18 memory.
- Write and Read are synchronous operations.
- The two ports are symmetrical and totally independent (can have different clocks), sharing only the stored data.
- Each port can be configured in one of the available widths, independent of the other port. The read port width can be different from the write port width for each port.
- The memory content can be initialized or cleared by the configuration bitstream.
Ultra-RAM Blocks

Table 2-1: Block RAM and UltraRAM Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Block RAM</th>
<th>UltraRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clocking</td>
<td>Two clocks</td>
<td>Single clock</td>
</tr>
<tr>
<td>Built-in FIFO</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data width</td>
<td>Configurable (1, 2, 4, 9, 18, 36, 72)</td>
<td>Fixed (72-bits)</td>
</tr>
<tr>
<td>Modes</td>
<td>SDP and TDP</td>
<td>Two ports, each can independently read or write (a superset of SDP)</td>
</tr>
<tr>
<td>ECC</td>
<td>64-bit SECDED</td>
<td>64-bit SECDED</td>
</tr>
<tr>
<td></td>
<td>Supported in 64-bit SDP only (one ECC decoder for port A and one ECC encoder for port B)</td>
<td>One set of complete ECC logic for each port to enable independent ECC operations (ECC encoder and decoder for both ports)</td>
</tr>
<tr>
<td>Cascade</td>
<td>• Cascade output only (input cascade implemented via logic resources)</td>
<td>• Cascade both input and output (with global address decoding)</td>
</tr>
<tr>
<td></td>
<td>• Cascade within a single clock region</td>
<td>• Cascade across clock regions in a column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cascade across several columns with minimal logic resources</td>
</tr>
<tr>
<td>Power savings</td>
<td>One mode via manual signal assertion</td>
<td>One mode via manual signal assertion</td>
</tr>
</tbody>
</table>

Figure 2-1: UltraRAM URAM288_BASE Primitive
# State-of-the-Art - Xilinx FPGAs

<table>
<thead>
<tr>
<th>Device Name</th>
<th>VU3P</th>
<th>VUSP</th>
<th>VU7P</th>
<th>VU9P</th>
<th>VU11P</th>
<th>VU13P</th>
<th>VU27P</th>
<th>VU29P</th>
<th>VU31P</th>
<th>VU33P</th>
<th>VU35P</th>
<th>VU37P</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Logic Cells (K)</td>
<td>862</td>
<td>1,314</td>
<td>1,724</td>
<td>2,586</td>
<td>2,835</td>
<td>3,780</td>
<td>2,835</td>
<td>3,780</td>
<td>962</td>
<td>962</td>
<td>1,907</td>
<td>2,852</td>
</tr>
<tr>
<td>CLB Flip-Flops (K)</td>
<td>788</td>
<td>1,201</td>
<td>1,576</td>
<td>2,364</td>
<td>2,592</td>
<td>3,456</td>
<td>2,592</td>
<td>3,456</td>
<td>879</td>
<td>879</td>
<td>1,743</td>
<td>2,607</td>
</tr>
<tr>
<td>CLB LUTs (K)</td>
<td>394</td>
<td>601</td>
<td>788</td>
<td>1,182</td>
<td>1,296</td>
<td>1,728</td>
<td>1,296</td>
<td>1,728</td>
<td>440</td>
<td>440</td>
<td>872</td>
<td>1,304</td>
</tr>
<tr>
<td>Max. Dist. RAM (Mb)</td>
<td>12.0</td>
<td>18.3</td>
<td>24.1</td>
<td>36.1</td>
<td>36.2</td>
<td>48.3</td>
<td>36.2</td>
<td>48.3</td>
<td>12.5</td>
<td>12.5</td>
<td>24.6</td>
<td>36.7</td>
</tr>
<tr>
<td>Total Block RAM (Mb)</td>
<td>25.3</td>
<td>36.0</td>
<td>50.6</td>
<td>75.9</td>
<td>70.9</td>
<td>94.5</td>
<td>70.9</td>
<td>94.5</td>
<td>23.6</td>
<td>23.6</td>
<td>47.3</td>
<td>70.9</td>
</tr>
<tr>
<td>UltraRAM (Mb)</td>
<td>90.0</td>
<td>132.2</td>
<td>180.0</td>
<td>270.0</td>
<td>270.0</td>
<td>360.0</td>
<td>270.0</td>
<td>360.0</td>
<td>90.0</td>
<td>90.0</td>
<td>180.0</td>
<td>270.0</td>
</tr>
<tr>
<td>HBM DRAM (GB)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>HBM AXI Interfaces</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Clock Mgmt Tiles (CMTs)</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>2,280</td>
<td>3,474</td>
<td>4,560</td>
<td>6,840</td>
<td>9,216</td>
<td>12,288</td>
<td>9,216</td>
<td>12,288</td>
<td>2,880</td>
<td>2,880</td>
<td>5,952</td>
<td>9,024</td>
</tr>
<tr>
<td>Peak INT8 DSP (TOP/s)</td>
<td>7.1</td>
<td>10.8</td>
<td>14.2</td>
<td>21.3</td>
<td>28.7</td>
<td>38.3</td>
<td>28.7</td>
<td>38.3</td>
<td>8.9</td>
<td>8.9</td>
<td>18.6</td>
<td>28.1</td>
</tr>
<tr>
<td>PCIe® Gen3 x16</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>PCIe Gen3 x16/Gen4 x8 / CCIX(1)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>150G Interlaken</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>100G Ethernet w/ KR4 RS-FEC</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>12</td>
<td>11</td>
<td>15</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Max. Single-Ended HP I/Os</td>
<td>520</td>
<td>832</td>
<td>832</td>
<td>832</td>
<td>624</td>
<td>832</td>
<td>520</td>
<td>676</td>
<td>208</td>
<td>208</td>
<td>416</td>
<td>624</td>
</tr>
<tr>
<td>GTy 32.75Gb/s Transceivers</td>
<td>40</td>
<td>80</td>
<td>80</td>
<td>120</td>
<td>96</td>
<td>128</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>64</td>
<td>96</td>
</tr>
<tr>
<td>GTM 58Gb/s PAM4 Transceivers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100G / 50G KP4 FEC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extended(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
</tr>
</tbody>
</table>

**Virtex Ultra-scale**
