# Traineeships in Advanced Computing for High Energy Physics (TAC-HEP)

**FPGA module training**

More on FPGAs

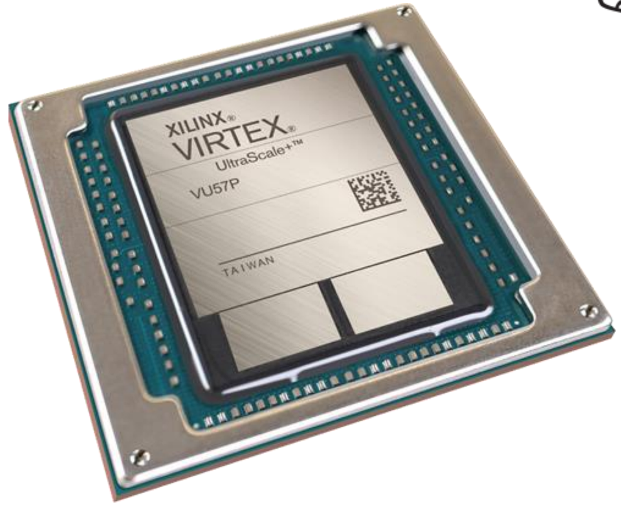
*Lecture-3: February 17th 2026*

Varun Sharma

University of Wisconsin – Madison, USA

## So Far…

- Motivation
- Comparison: FPGAs/ASICs/GPU/CPU
- Domain specific Accelerators



**Today:**

- More on FPGA's & HLS Setup

**FPGA:** **F**ield **P**rogrammable **G**ate **A**rray

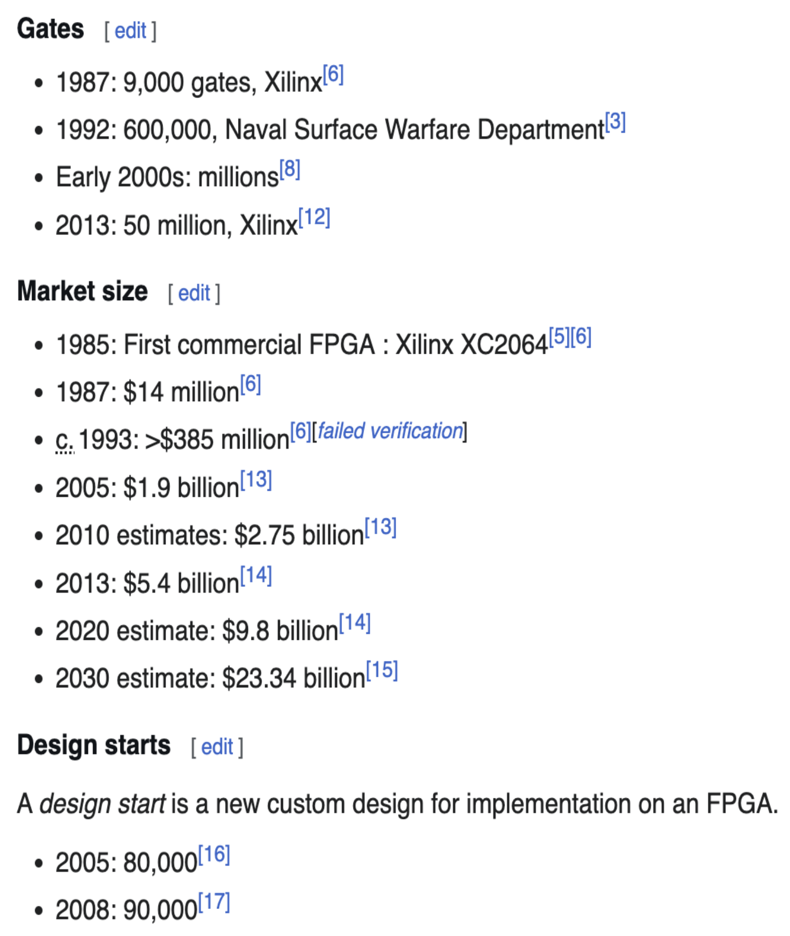
# Xilinx Field Programmable Gate Array

**Xilinx: All Programmable**

Software Defined, Hardware Optimized

You may know Xilinx because we invented the FPGA. Or maybe you know us because we turned the semiconductor world upside down and created the fabless model. With over 3500 patents and more than 60 industry firsts, we continue to pioneer new programmable technology putting our customers first. Today Xilinx's portfolio combines All Programmable devices in the categories of FPGAs, SoCs, and 3DICs, as well as All Programming models, including software-defined development environments. Our products are enabling smart, connected, and differentiated applications driven by 5G Wireless, Embedded Vision, Industrial IoT, and Cloud Computing.

First FPGA invented by Xilinx Inc. in 1985



**Gates** [ edit ]

- 1987: 9,000 gates, Xilinx[6]
- 1992: 600,000, Naval Surface Warfare Department[3]
- Early 2000s: millions[8]
- 2013: 50 million, Xilinx[12]

**Market size** [ edit ]

- 1985: First commercial FPGA : Xilinx XC2064[5][6]
- 1987: $14 million[6]
- c. 1993: >$385 million[6][*failed verification*]
- 2005: $1.9 billion[13]
- 2010 estimates: $2.75 billion[13]
- 2013: $5.4 billion[14]
- 2020 estimate: $9.8 billion[14]
- 2030 estimate: $23.34 billion[15]

**Design starts** [ edit ]

A *design start* is a new custom design for implementation on an FPGA.

- 2005: 80,000[16]
- 2008: 90,000[17]

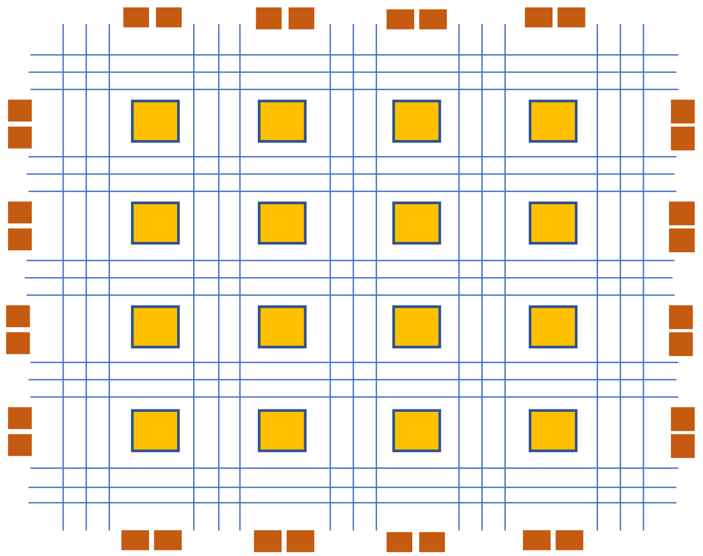
***Source:*** *https://en.wikipedia.org/wiki/Field-programmable_gate_array*

# FPGAs:

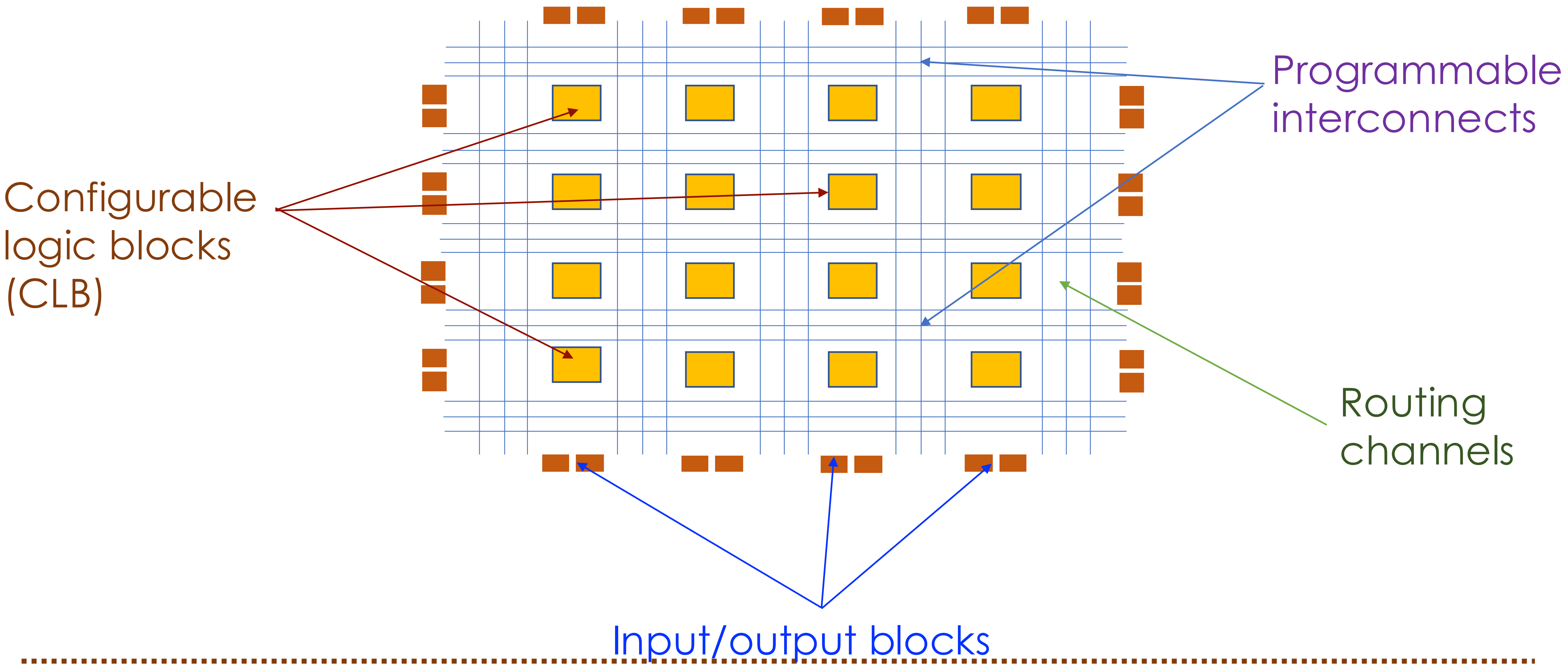
- **FPGAs** are an array of programmable logic blocks and memory elements that are connected together using programmable interconnect
    - Programmable hardware whose sub-component configuration can be changed even after fabrication: ***"field-programmable"***
    - Has 2D array of logic gates in its architecture: ***"Gate Array"***

- A silicon **'breadboard'** of configurable logic gates, memories, transceivers, Digital Signal Processors (DSPs), registers (flip flops)

- **Over decades**, FPGAs have gone from small arrays of PL and interconnect to massive arrays of PL and interconnect with on-chip memories, custom data paths, high speed I/O, & microprocessor cores all co-located on the same chip



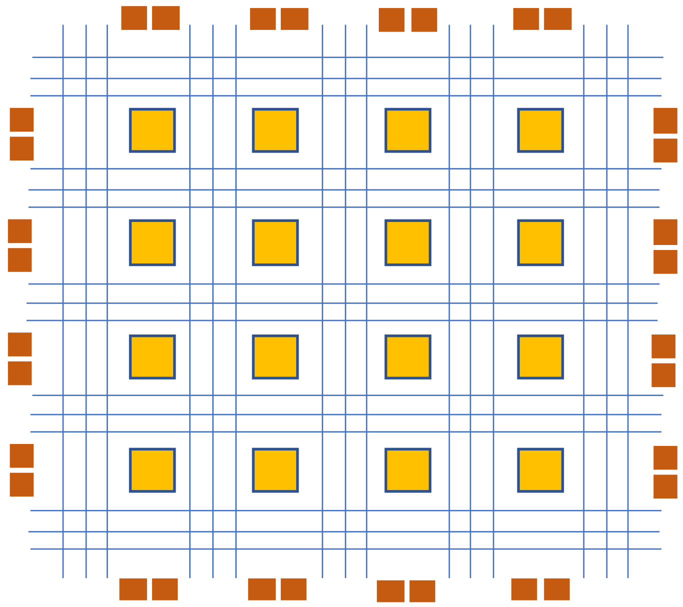
# FPGA Architecture



# FPGA Architecture

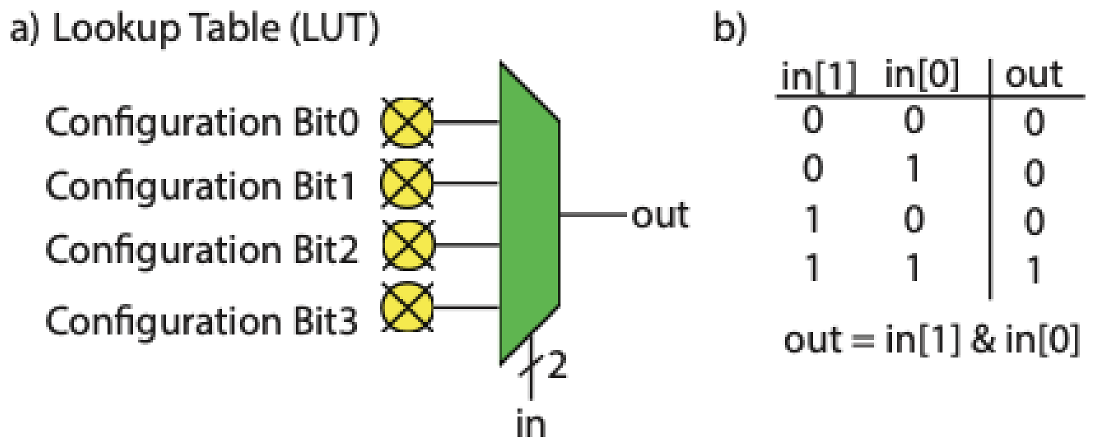
The basic structure of an FPGA is composed of:

- Static Random Access Memory (SRAM):
- Configurable Logic Blocks (CLBs)
  - Look-up table (LUT)
  - Flip-Flop (FF)
  - Multiplexers
  - DSP Blocks
  - Block Memories (BRAM)
- Wires: These elements connect elements to one another
- Input/Output (I/O) pads: These physically available ports get data in and out of the FPGA
- Clocking Resources



# Look-up Tables (LUTs)

- A memory where address signal are the inputs and the outputs are stored in the memory entries
- A typical LUT is an **n-input truth table** stored in SRAM
- Instead of computing logic function in real-time, LUTs store the output values for all possible input combinations
- When inputs are applied, the LUT retrieves the corresponding output from the memory

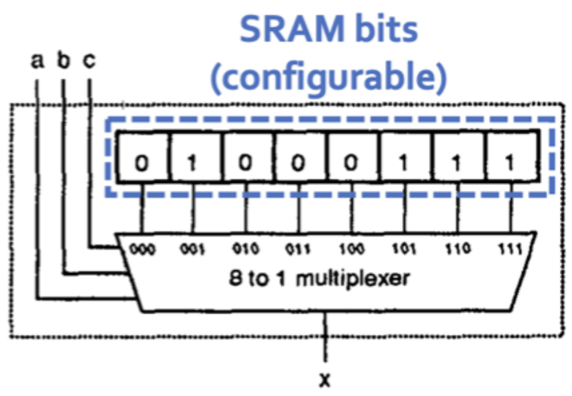


| in[1] | in[0] | out |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

out = in[1] & in[0]

| a | b | c | $x = ab + \bar{b}c$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

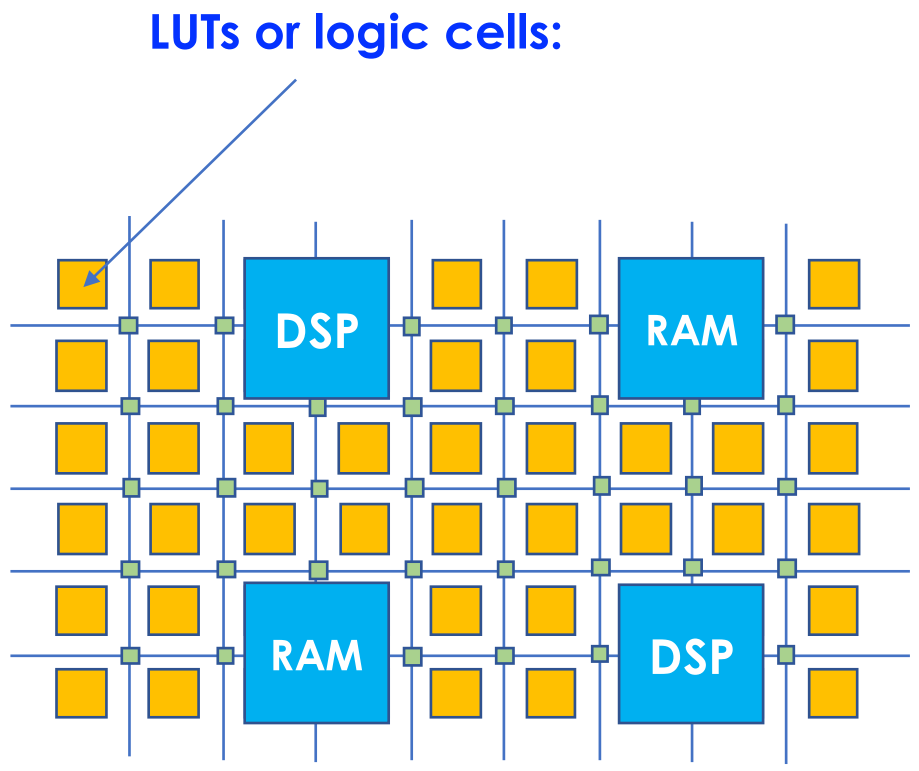
**Truth table**



**3-Input LUT**

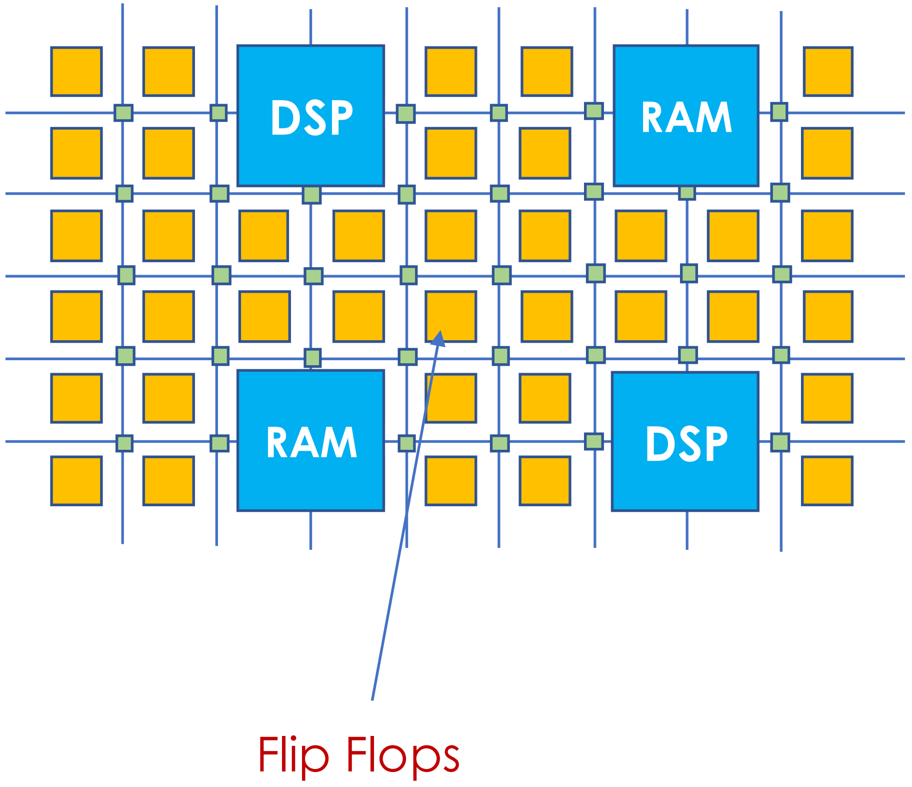
## LUTs

- Capable of performing any arbitrary functions on small bitwidth inputs (N), generally N ≤ 7
- Memory location accessed by LUTs: $2^N$
- It can be used as both a function compute engine and a data storage element
- LUTs can be combined to create more complex logic circuits.



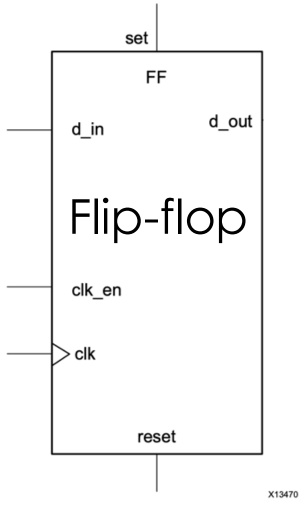
# Flip-Flops (FF)

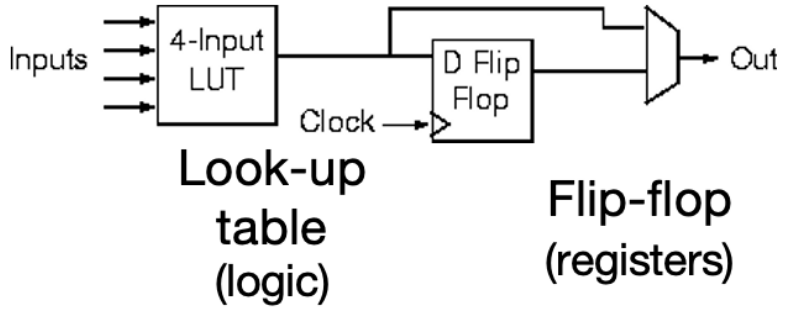
- Basic memory element, co-located with a LUT to assist in logic pipelining and data storage
- Its name comes from its ability to flip or flop between two stable states
- **Operation:** value at the data input port is latched and passed to the output on every pulse of the clock
- They can store data over time

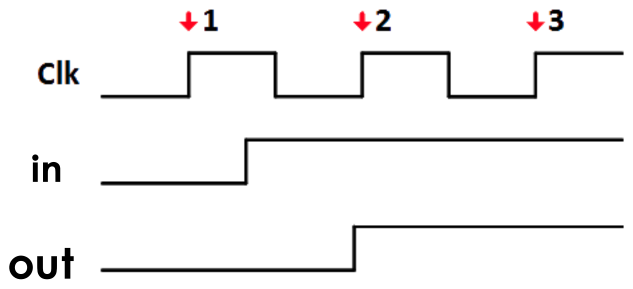


# FFs

- The clock is what allows a Flip-Flop to be used as a data storage element
- Any data storage elements are known as **sequential logic** or **registered logic**.
- Sequential logic operates on the *transitions* of a clock. Mostly on the rising edge (when the clock goes from 0 to 1).
- When a Flip-Flop sees a rising edge of the clock, it *registers* the data from the Input to the Output .
- CLBs typically contain several flip-flops that can be used to store intermediate results or outputs of logic functions







D Flip-Flop Input to Output

## Multiplexer

- Combinational logic circuits used to select one input from multiple inputs and pass it to the output based on a control signal
- Routing signals within the CLB
- FPGA interconnects rely on large multiplexers to route signals between logic blocks

## SRAMs

- These are memories that allows FPGAs to be configured and reconfigured so easily
- In SRAM-based FPGAs, configuration is stored in SRAM cells.
  - It determines how the logic blocks are connected, what function they perform, and how routing resources are configured

- They are inherently volatile.
  - Loses its data when power is turned off
  - ➢ Need to be configured every time they power on, usually done by loading configuration data from an external memory source

## Other Storage Elements: URAM

- Ultra RAM blocks are dual-port, synchronous 288Kb RAM with a fixed configuration
- Available in Xilinx's UltraScale+ devices

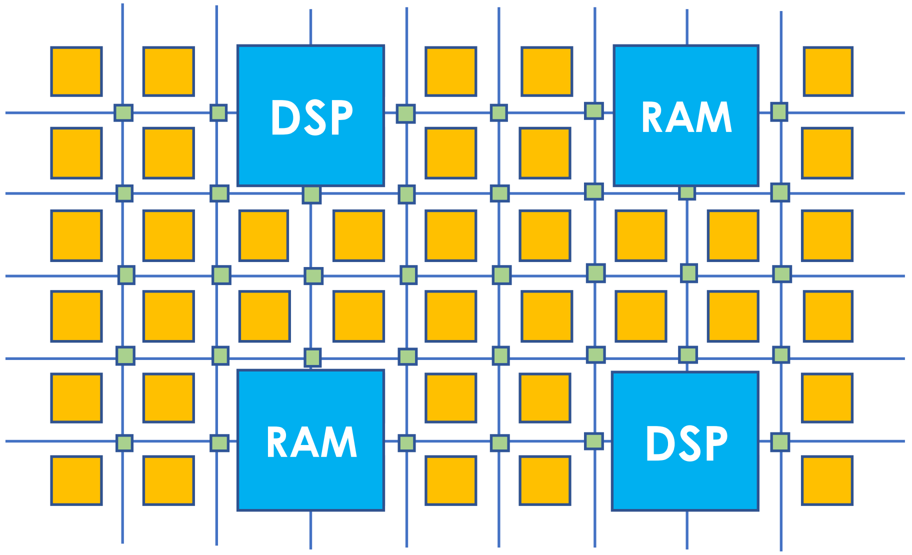
- Eight times more storage capacity than the BRAMs
- URAMs generally have higher latency access compared to BRAMs
- **Usage:** Large buffers, Video processing …

## Storage elements: Block RAM

BRAMs are used for storing large amounts of data in a FPGA

- Embedded memory elements that can be used to provide high speed data storage & retrieval

- BRAM is a dual-port RAM module instantiated to provide on-chip storage for a relatively large set of data
  - can hold either 18kb or 36 kb
  - **Multiple Block RAMs** can be used in parallel to create larger memory arrays or buffers.

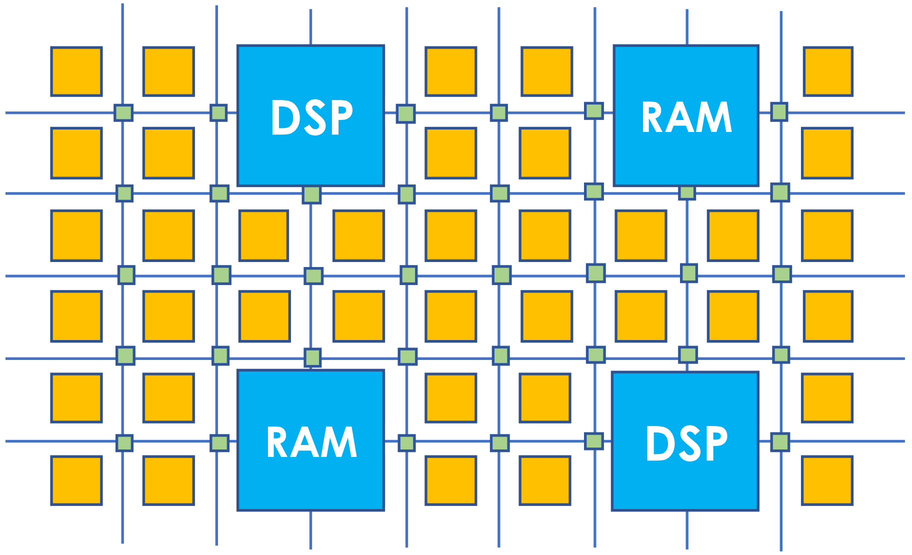


- Block RAMs are **dual-port**: two independent ports for **reading** and **writing** data simultaneously
  - Useful in applications like **FIFO buffers**, where one port handles writing data while the other port handles reading data

# FPGA Components: Storage elements

**LUTs as storage element:**

- They can be used as 64-b memories due to its structural flexibity
- Commonly referred to as distributed memories



- Fastest kind of memory available on the FPGA device, because it can be instantiated in any part of the fabric that improves the performance of the implemented circuit
- Memories using BRAMs more efficient than using LUTs

# I/O Blocks

There are specialised blocks for I/O

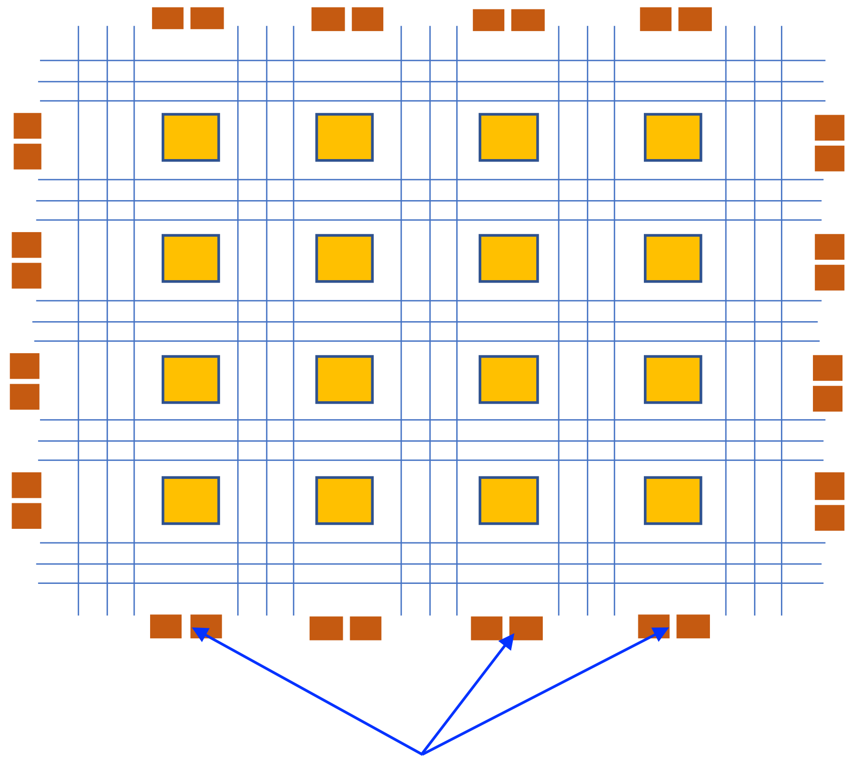
- Interface between FPGA & external devices
- Can be configured to handle a variety of voltage standards
- Making FPGAs popular in embedded systems and HEP triggers

Some of the I/O blocks are bidirectional – can be configured as input or outputs

Dynamically reconfigurable to accommodate changing requirements, such as switching b/w different voltage standards or signal types

Low power per Operation (relative to CPU/GPU)

Fig. 21


Input/output blocks

# I/O Blocks

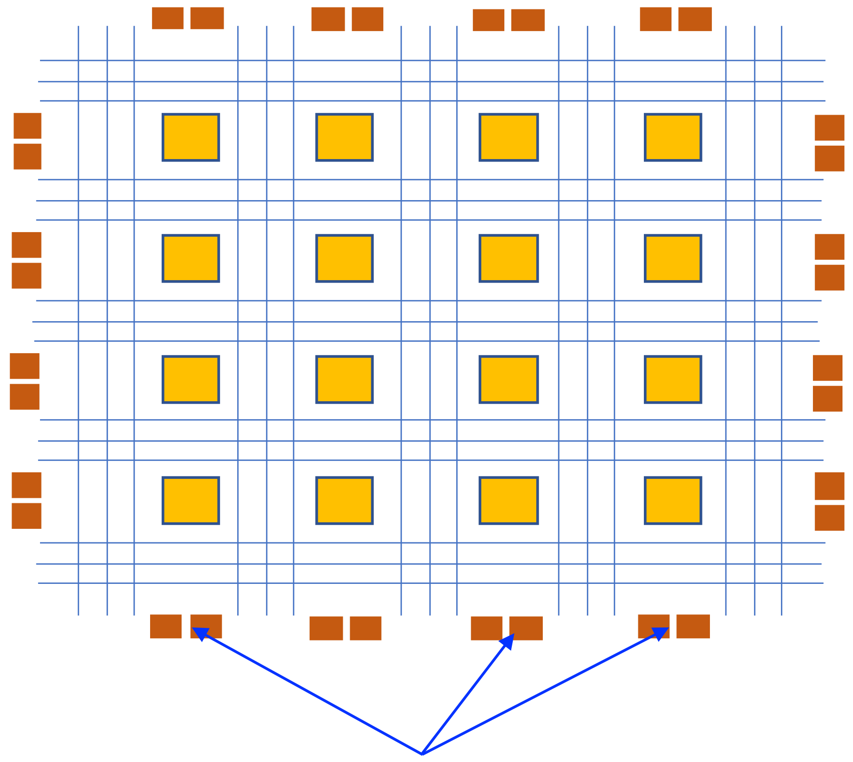
## Clocking & Synchronization:

- Used for clock input & output, essential for synchronization the FPGA with other devices
- Support clock domain crossing & buffering to ensure stable signal timing across different clock frequencies

## High speed transceivers

- with Tb/s total bandwidth PCIe
- (Multi) Gigabit Ethernet
- Infiniband

Fig. 21



## Multi-Gigabit Opto-electronics

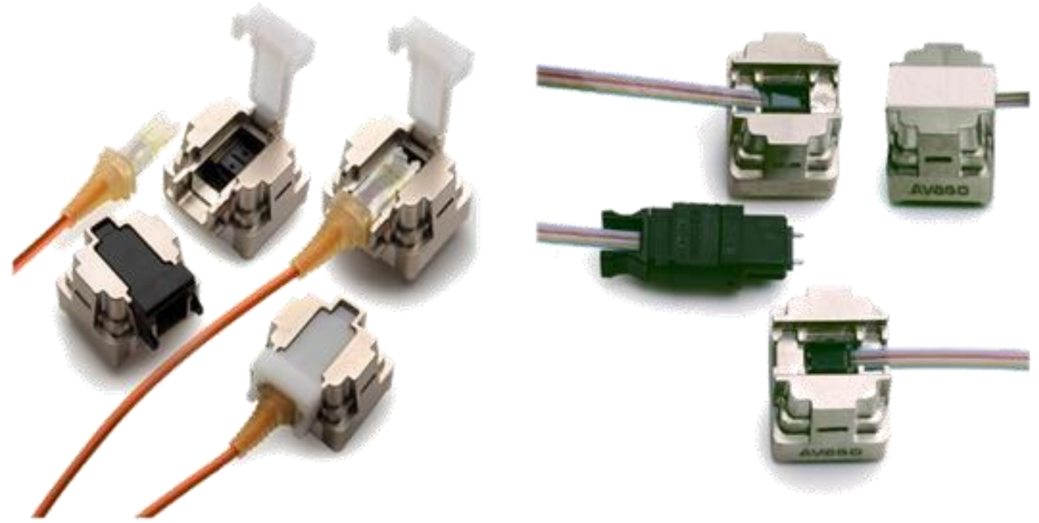


Figure 1. MiniPOD™ Transmitter and Receiver Modules with a) Round Cable and b) Flat Cable: shown with and without dust covers (White = Tx, Black = Rx).

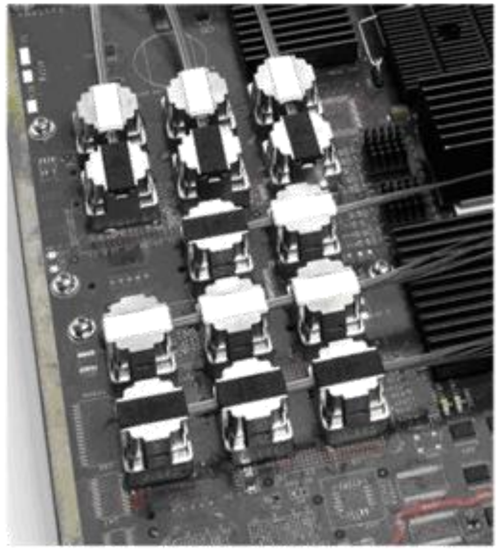
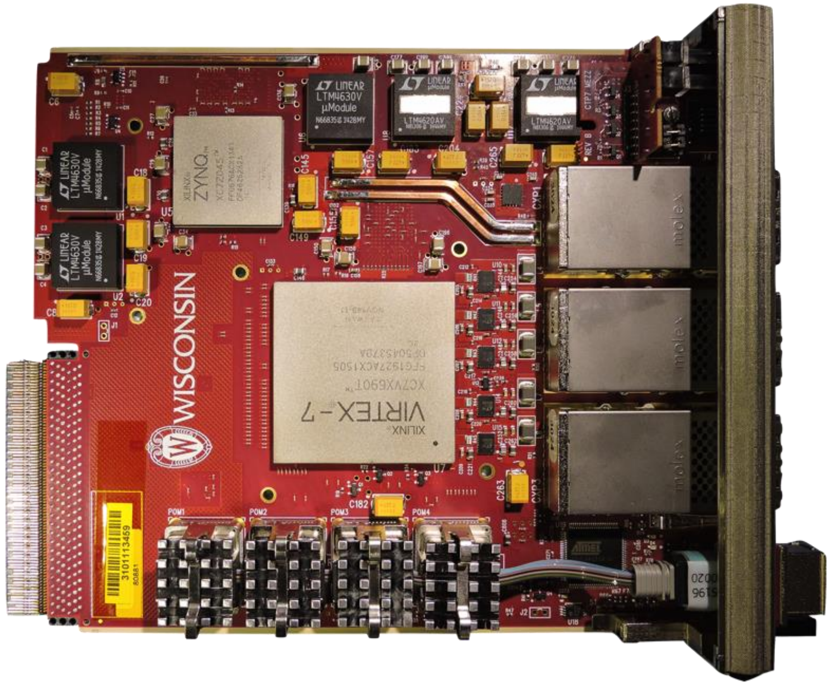


Figure 2. MiniPOD™ Transmitter and Receiver flat ribbon cable modules in a tiled arrangement example.



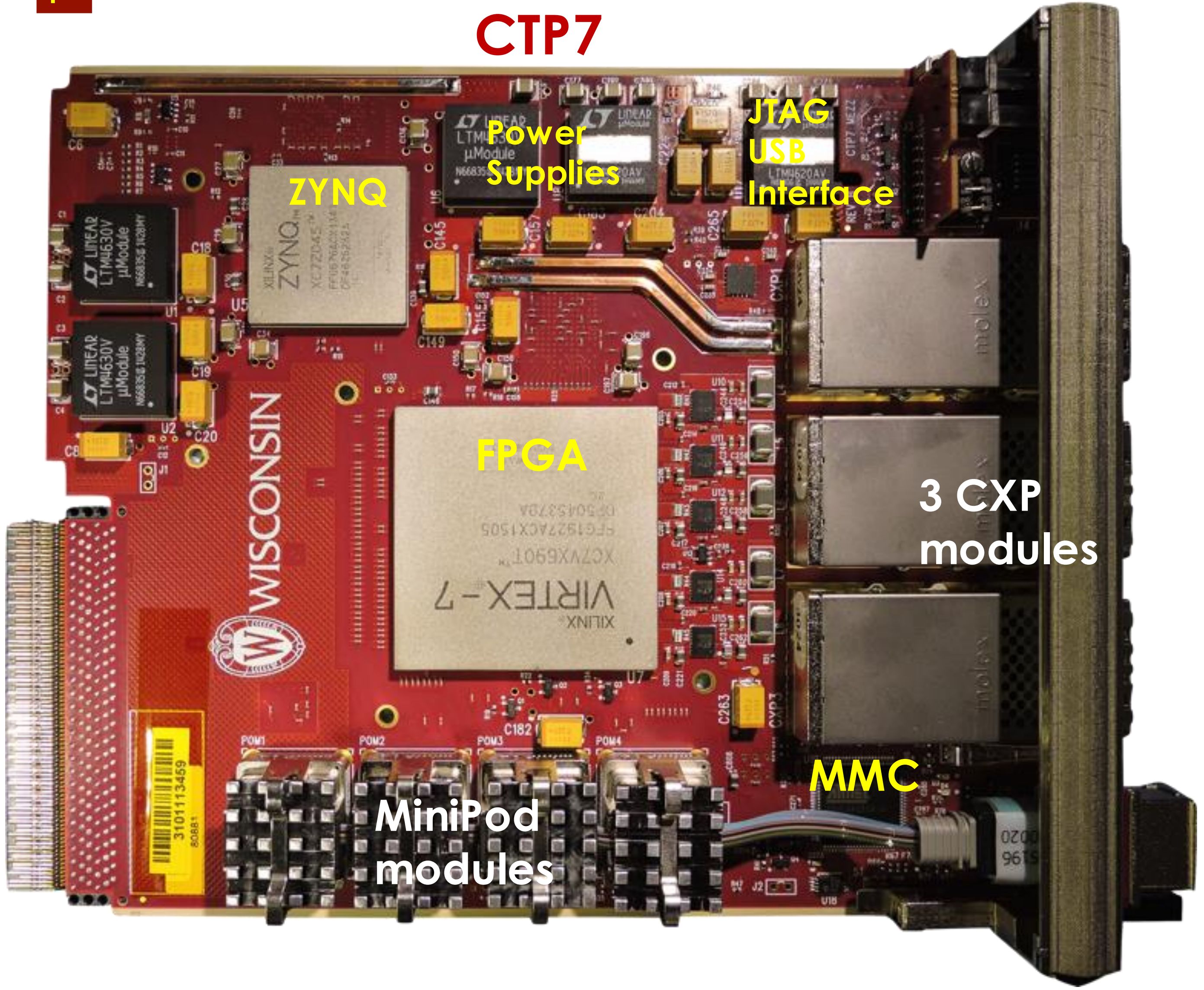
**Key Product Parameters**

The Avago Technologies MiniPOD™ modules operate at 850 nm and are compliant to the Multi-mode Fiber optical specs in clause 86 and relevant electrical specs in annex 86A of the IEEE 802.3ba specifications.

| Parameter | Value | Units | Notes |
|---|---|---|---|
| Data rate per lane | 10.3125 | Gbps | As per 802.3ba: 100GBASE-SR10 and nPPI specifications |
| Number of operational lanes | 12 | | 100GbE operation utilizes the middle ten lanes (Rx and Tx) of the 12 physically defined lanes |
| Link Length | 100<br>150 | m<br>m | OM3, 2000 MHzMHz•km 50 µm MMF<br>OM4, 4700 MHz•km 50 µm MMF |

## Trigger Processor Boards

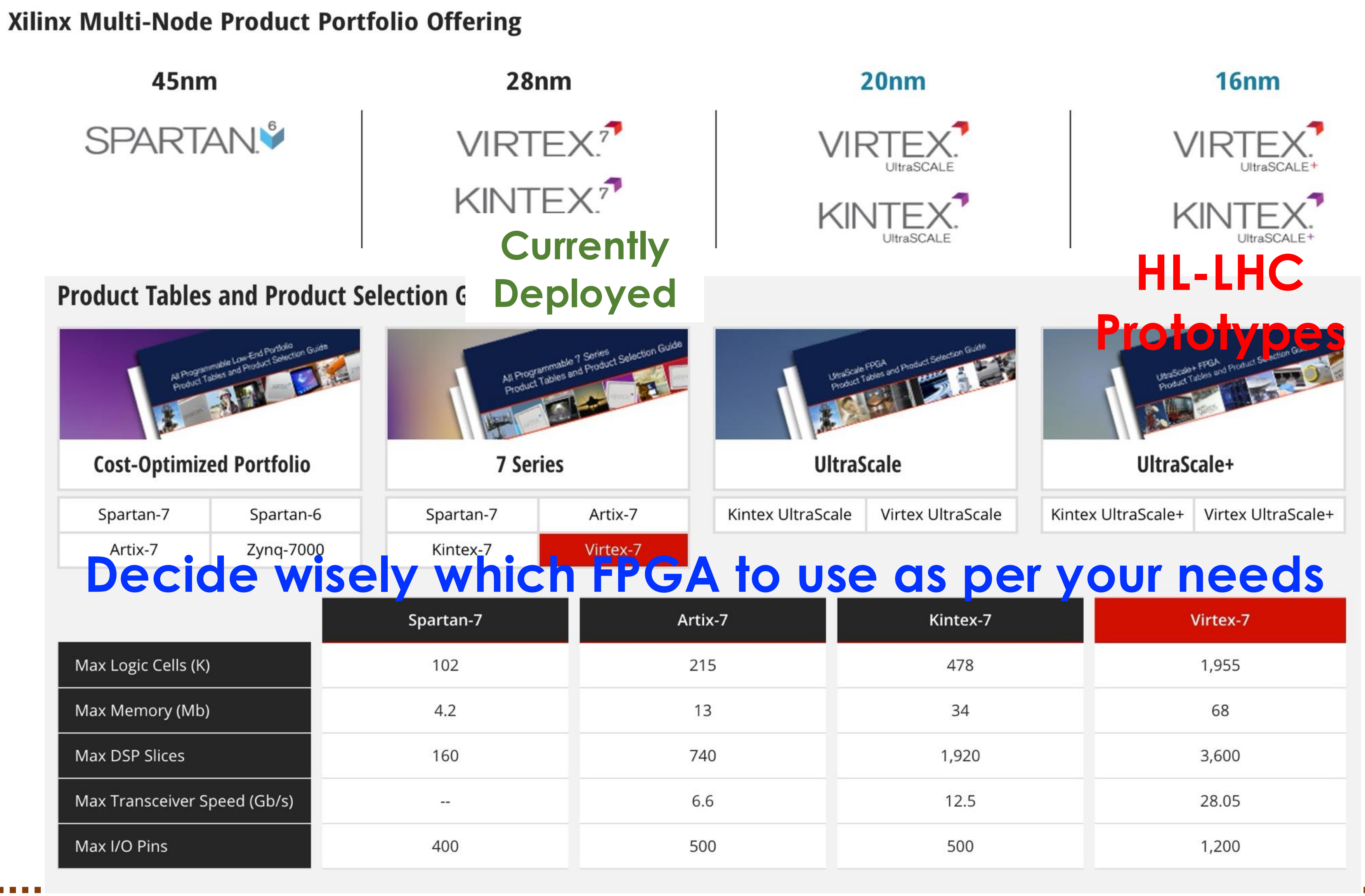
CTP7



**Calorimeter Trigger Processor (CTP7)**

- **CTP7 (Layer-1) – mTCA Single Virtex 7 FPGA, 67 optical inputs, 48 outputs, 12 RX/TX backplane**
  - Virtex 7 allows 10 Gb/s link speed on 3 CXP(36 TX & 36 RX) and 4 MiniPODs (31 RX & 12 TX)
  - ZYNQ processor running Xilinx PetaLinux for service tasks, including virtual JTAG cable

# Xilinx FPGAs – Phase-1 choice: V7 690T



**Xilinx Multi-Node Product Portfolio Offering**

| 45nm | 28nm | 20nm | 16nm |
|---|---|---|---|
| SPARTAN 6 | VIRTEX 7, KINTEX 7 | VIRTEX UltraSCALE, KINTEX UltraSCALE | VIRTEX UltraSCALE+, KINTEX UltraSCALE+ |

**Currently Deployed**

**HL-LHC Prototypes**

**Product Tables and Product Selection G**

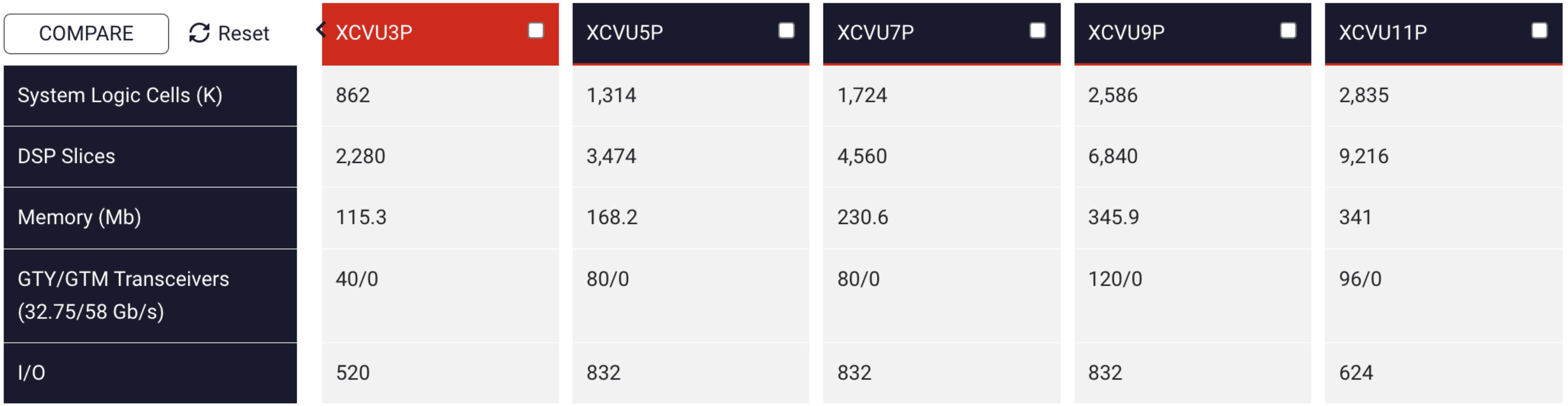
| Cost-Optimized Portfolio | | 7 Series | | UltraScale | | UltraScale+ | |
|---|---|---|---|---|---|---|---|
| Spartan-7 | Spartan-6 | Spartan-7 | Artix-7 | Kintex UltraScale | Virtex UltraScale | Kintex UltraScale+ | Virtex UltraScale+ |
| Artix-7 | Zynq-7000 | Kintex-7 | Virtex-7 | | | | |

**Decide wisely which FPGA to use as per your needs**

| | Spartan-7 | Artix-7 | Kintex-7 | Virtex-7 |
|---|---|---|---|---|
| Max Logic Cells (K) | 102 | 215 | 478 | 1,955 |
| Max Memory (Mb) | 4.2 | 13 | 34 | 68 |
| Max DSP Slices | 160 | 740 | 1,920 | 3,600 |
| Max Transceiver Speed (Gb/s) | -- | 6.6 | 12.5 | 28.05 |
| Max I/O Pins | 400 | 500 | 500 | 1,200 |

**Speed grade: maximum propagation delay** for critical paths in the FPGA fabric or I/O operations

## Xilinx Virtex Ultra Scale+ Product Table



COMPARE | Reset

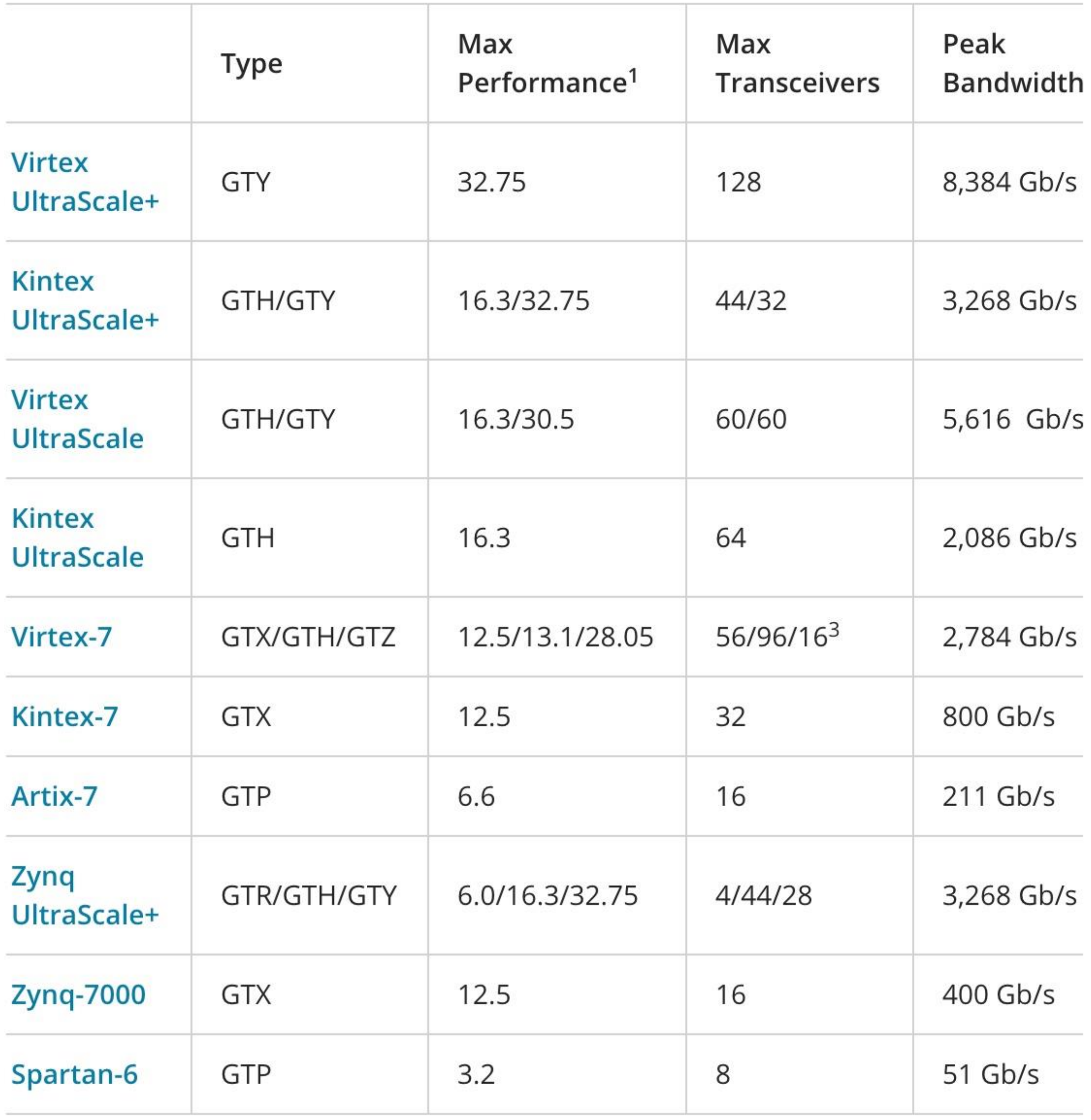
| | XCVU3P | XCVU5P | XCVU7P | XCVU9P | XCVU11P |
|---|---|---|---|---|---|
| System Logic Cells (K) | 862 | 1,314 | 1,724 | 2,586 | 2,835 |
| DSP Slices | 2,280 | 3,474 | 4,560 | 6,840 | 9,216 |
| Memory (Mb) | 115.3 | 168.2 | 230.6 | 345.9 | 341 |
| GTY/GTM Transceivers (32.75/58 Gb/s) | 40/0 | 80/0 | 80/0 | 120/0 | 96/0 |
| I/O | 520 | 832 | 832 | 832 | 624 |

*Source*: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html#productTable

Decide wisely which FPGA to use as per your needs

# Multi-gigabit-per-second serial links

HL-LHC



| | Type | Max Performance[1] | Max Transceivers | Peak Bandwidth |
|---|---|---|---|---|
| Virtex UltraScale+ | GTY | 32.75 | 128 | 8,384 Gb/s |
| Kintex UltraScale+ | GTH/GTY | 16.3/32.75 | 44/32 | 3,268 Gb/s |
| Virtex UltraScale | GTH/GTY | 16.3/30.5 | 60/60 | 5,616 Gb/s |
| Kintex UltraScale | GTH | 16.3 | 64 | 2,086 Gb/s |
| Virtex-7 | GTX/GTH/GTZ | 12.5/13.1/28.05 | 56/96/16[3] | 2,784 Gb/s |
| Kintex-7 | GTX | 12.5 | 32 | 800 Gb/s |
| Artix-7 | GTP | 6.6 | 16 | 211 Gb/s |
| Zynq UltraScale+ | GTR/GTH/GTY | 6.0/16.3/32.75 | 4/44/28 | 3,268 Gb/s |
| Zynq-7000 | GTX | 12.5 | 16 | 400 Gb/s |
| Spartan-6 | GTP | 3.2 | 8 | 51 Gb/s |

HL-LHC

$\longleftarrow$

25 Gbps

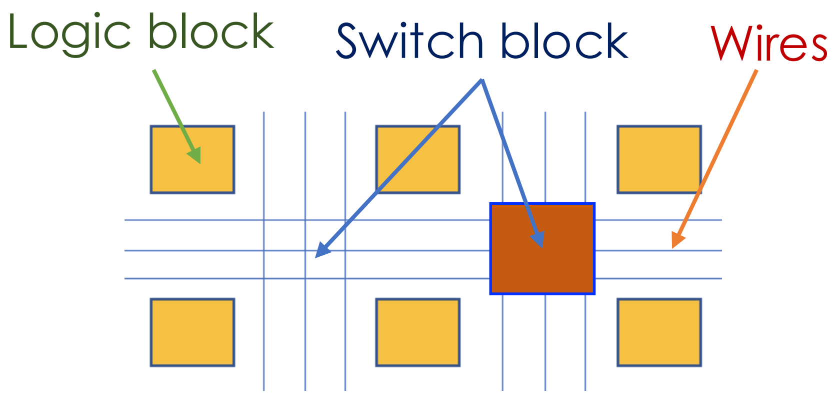
LHC

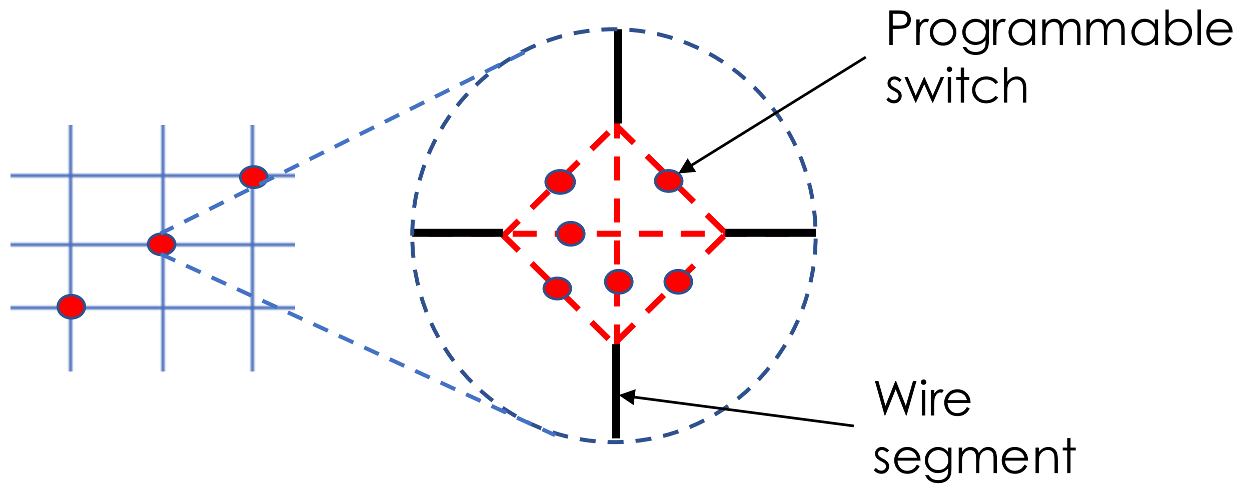
$\longrightarrow$

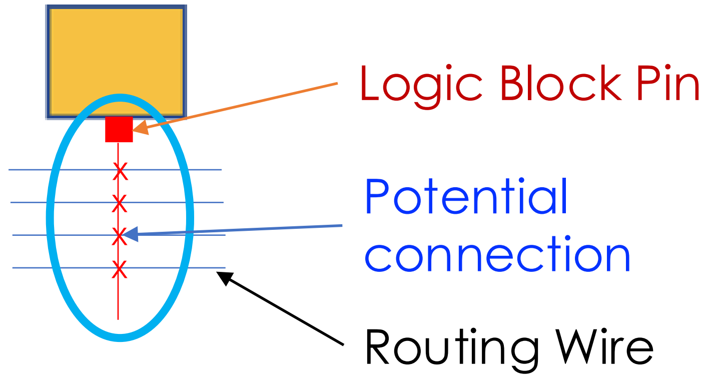
10 Gbps

# FPGA Components: Routing

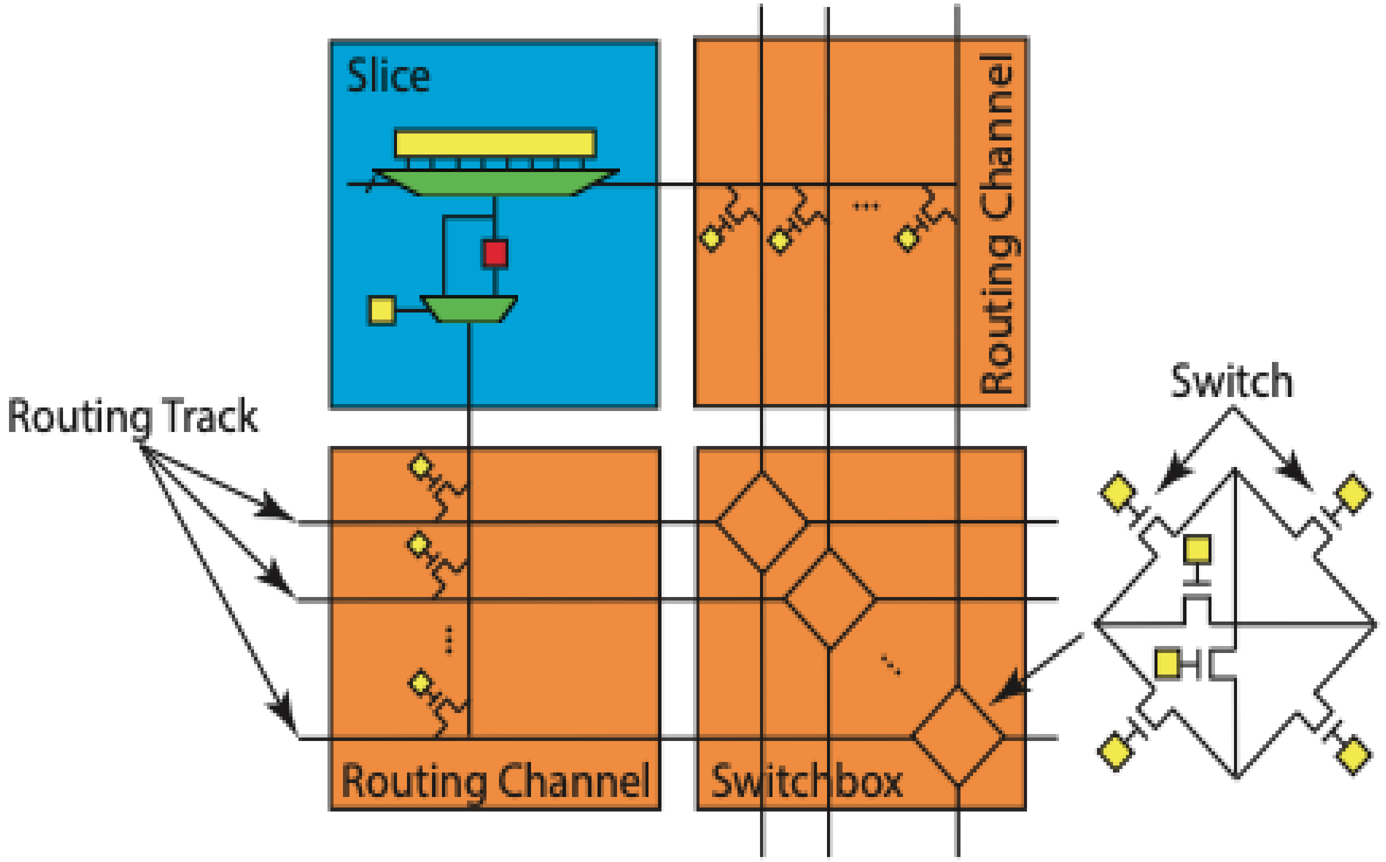
- Between rows and columns of logic blocks are wiring channels
- These are programmable – a logic block pin can be connected to one of many wiring tracks through programmable switch
- Xilinx FPGA have dedicated switch block circuits for routing (flexible)
- Each wiring segment can be connected in one of many ways







# FPGA Components: Routing

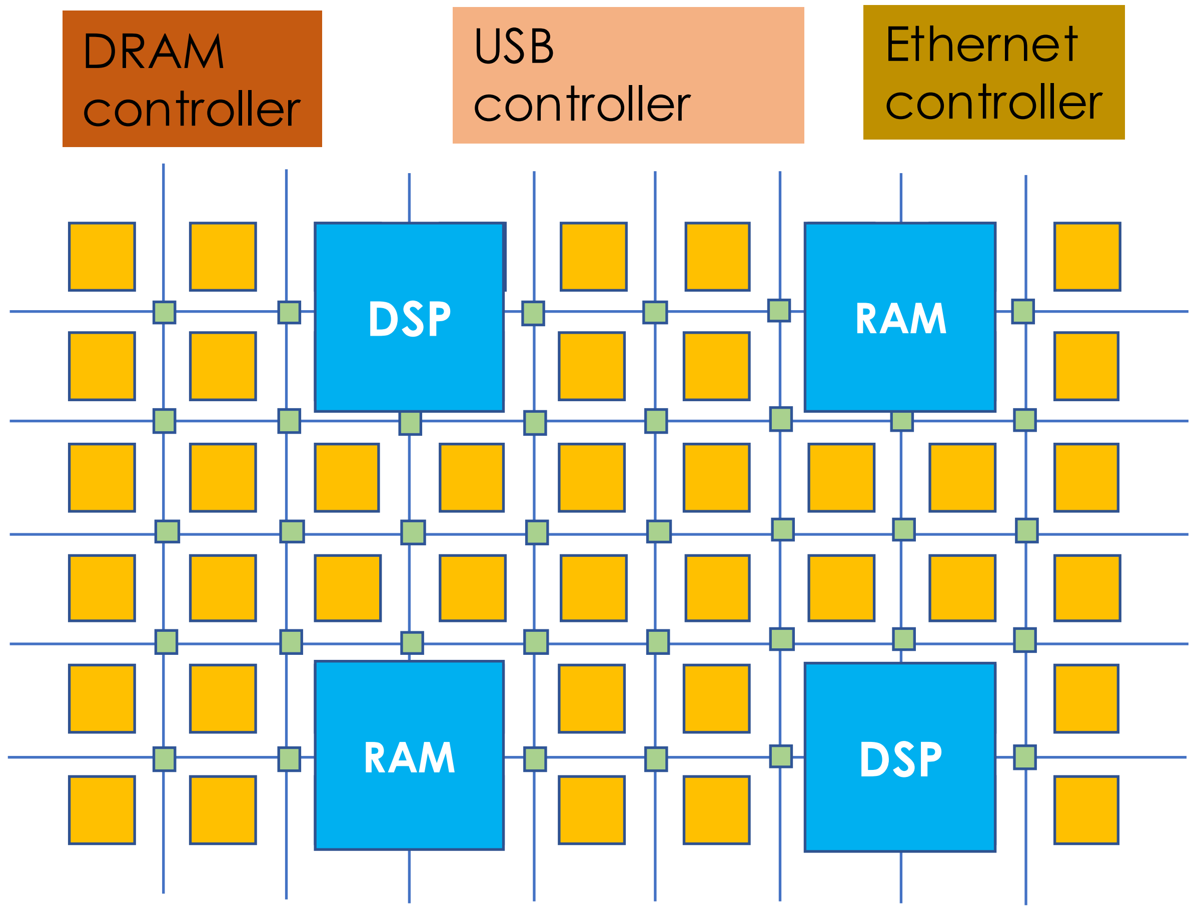


- Simple slice with a LUT and a FF
- Slices are connected to one another using a routing channel & switchbox
- These two provide a programmable interconnect that provide the data movement b/w slices.

- The switchbox has many switches (typically implemented as pass transistors) that allow for arbitrary wiring configurations between the different routing tracks in the routing tracks adjacent to the switchbox

**The main advantage and attraction of FPGA comes from the programmable interconnect – more so than the programmable logic.**

# FPGA Architecture



# Digital Signal Processing (DSP) Blocks

Built in components for fast arithmetic operation optimized for DSP operations

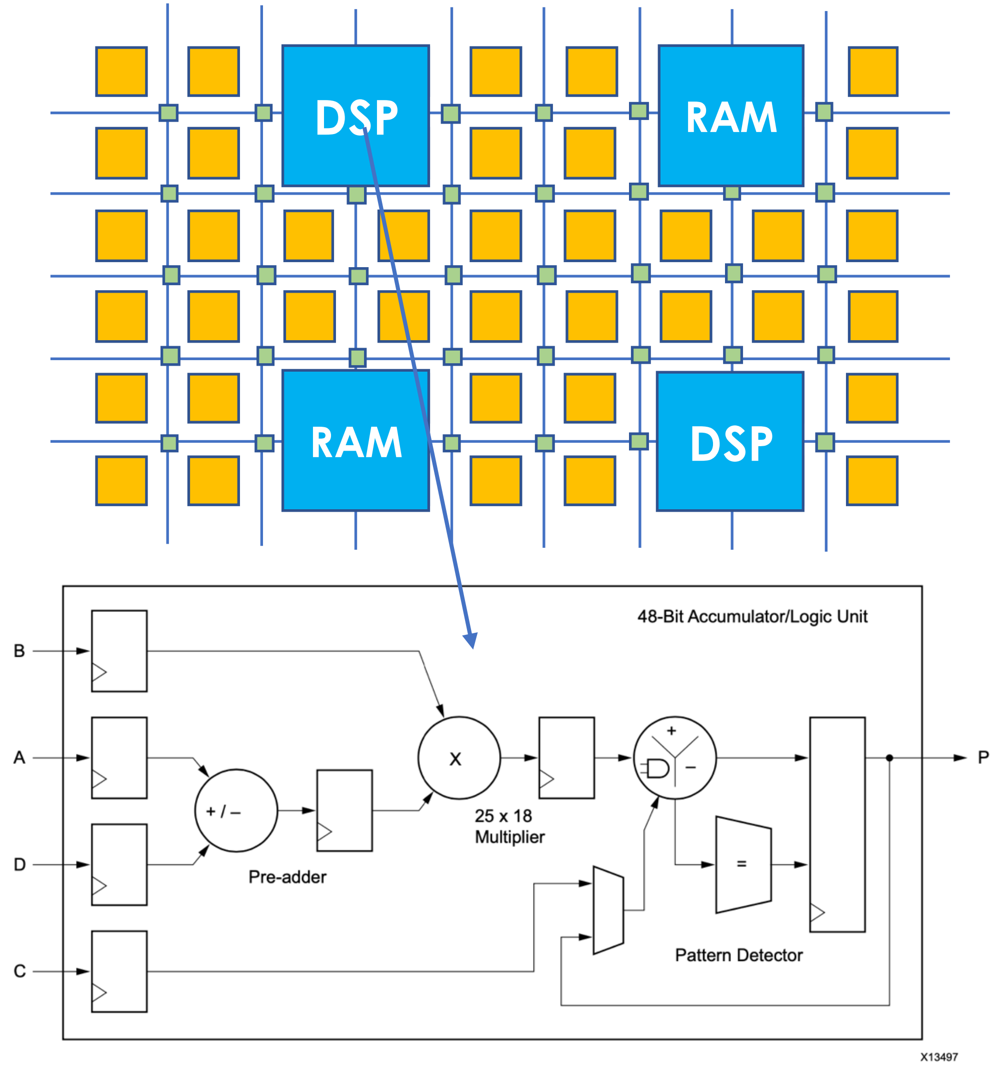
- Most complex computational block available in a FPGA
- Helps in accelerating signal processing algos

- Optimized for high performance **multiplication** and **accumulation**
    - Eg: p = a x (b + d) + c
    - Foundation for many signal processing algorithms, like **filtering**, **transformations**, **convolutions**…

- Faster and more efficient than using LUTs for these types of operations

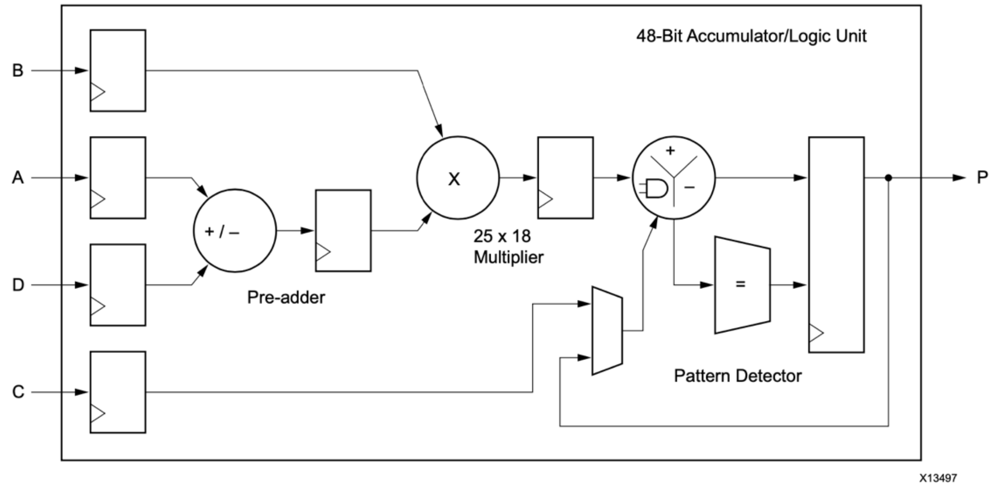
- Often most scarce in available resources

Fig. 14

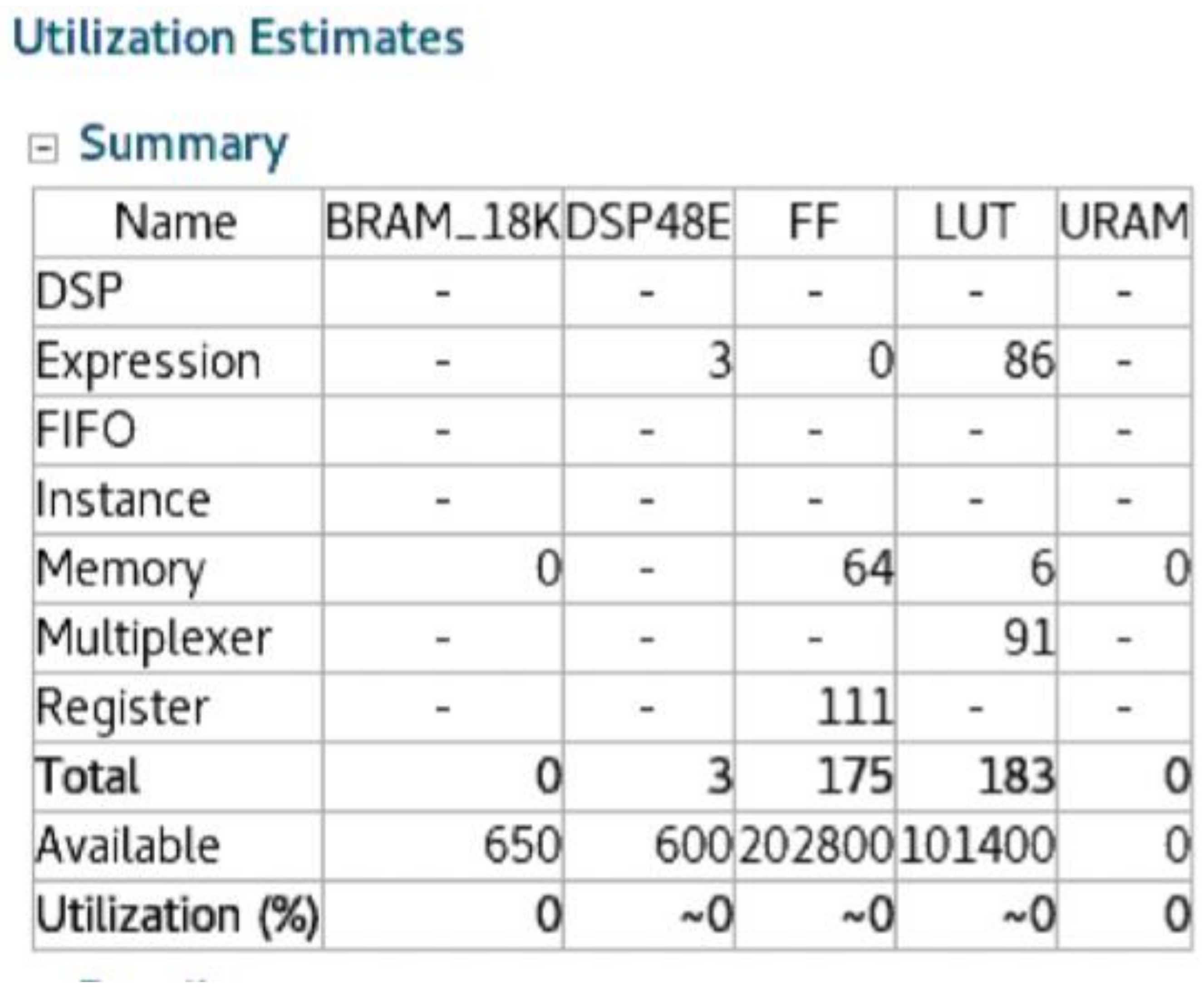


## DSP

- Each DSP block typically contains a **multiply-accumulate unit**,
  - Allows a multiplication to be directly followed by an accumulation operation (multiplying two values and adding the result to an accumulator)
  - Highly efficient and minimizes the need for multiple steps in traditional processing
- These blocks support various data widths, typically ranging from **8 bits** to **64 bits** or more, and support both **signed** and **unsigned** arithmetic



# Example: Resource Utilization



Utilization Estimates

⊟ Summary

| Name | BRAM_18K | DSP48E | FF | LUT | URAM |
|---|---|---|---|---|---|
| DSP | - | - | - | - | - |
| Expression | - | 3 | 0 | 86 | - |
| FIFO | - | - | - | - | - |
| Instance | - | - | - | - | - |
| Memory | 0 | - | 64 | 6 | 0 |
| Multiplexer | - | - | - | 91 | - |
| Register | - | - | 111 | - | - |
| **Total** | 0 | 3 | 175 | 183 | 0 |
| Available | 650 | 600 | 202800 | 101400 | 0 |
| Utilization (%) | 0 | ~0 | ~0 | ~0 | 0 |

# Clocking Resources

Clocking resources in FPGAs are essential components that manage **clock signals**, ensuring proper timing and synchronization across various logic blocks

- FPGAs include specialized **clock management circuits** to distribute and modify clock signals efficiently, enabling high-performance designs

- Global Clock Networks
  - Distribute clocks efficiently across the entire FPGA
- Phase-locked loops (PLLs) for driving the FPGA fabric at different clock rates
  - Adjusts the clock frequency by **multiplying** or **dividing** an input clock
  - Reduces **jitter** and maintains **clock stability**
  - Used in **frequency synthesis** and **clock recovery**

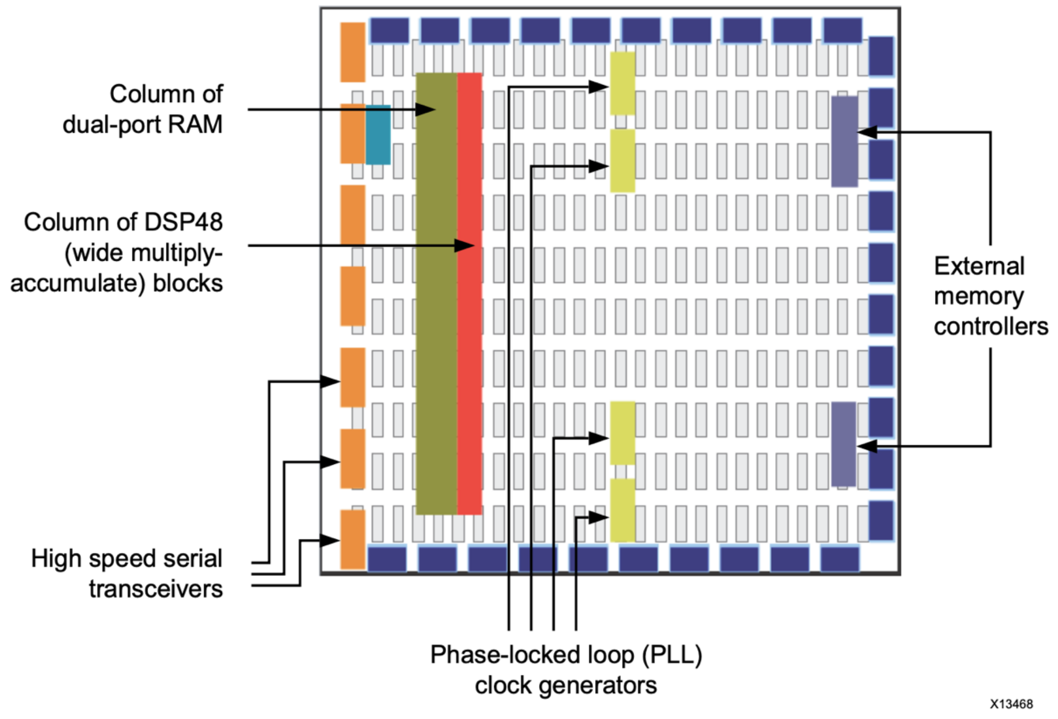

Column of dual-port RAM
Column of DSP48 (wide multiply-accumulate) blocks
High speed serial transceivers
Phase-locked loop (PLL) clock generators
External memory controllers
X13468

Fig. 9

# Why are FPGAs fast

Fine-grained/resource parallelism

- Use the many resources to work on different parts of the problem simultaneously
- Allows us to achieve low latency

Most problems have at least some sequential aspect, limiting how low latency we can go

- But we can still take advantage of it with…

*Fig. 22: Like a production line for data…*

Pipeline parallelism

- Use the register pipeline to work on different data simultaneously
- Allows us to achieve high throughput

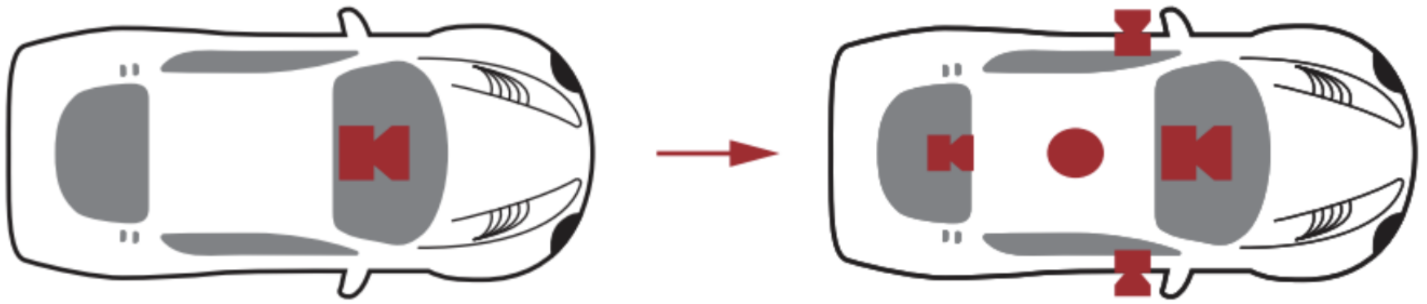
## More Advanced Architectures

- Embedded FPGA System on Chip (SoC)

- High Bandwidth Memory (HBM) on Xilinx FPGA
  - A theoretical bandwidth up to 460 GB/s

- ACAP: Adaptive Compute Acceleration Platform
  - *A fully software-programmable, heterogeneous compute platform that **combines Scalar Engines, Adaptable Engines**, and **Intelligent Engines** to achieve dramatic performance improvements of up to 20X over today’s fastest FPGA implementations and over 100X over today's fastest CPU implementations—for Data Center, wired network, 5G wireless, and automotive driver assist applications.*

# ACAP Application



| |
|---|
| 1x HD Camera |
| ~10W |

| |
|---|
| Sensor Fusion |
| 4x HD Cameras |
| Radar |
| Ultrasound |
| LIDAR |
| Machine Learning |
| ~10W |

WP505_13_092818

Xilinx ACAP Devices enable sensor fusion in small power envelopes

## Setup (connection to login)

- Connect to login machine:
  - `ssh -X -Y <username>@login.hep.wisc.edu`

  - `mkdir /nfs_scratch/`whoami`` (If directory exist, go to next bullet)
  - `cd /nfs_scratch/`whoami``

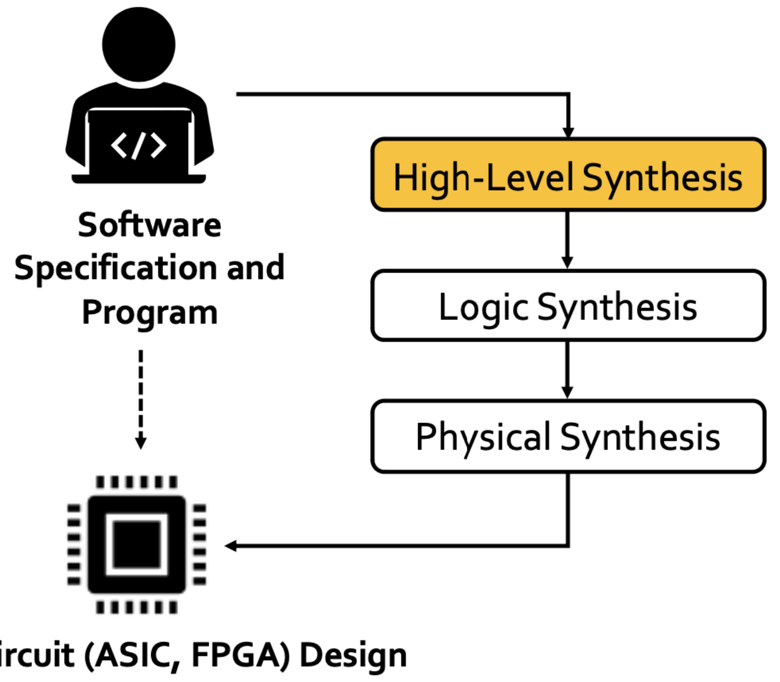
**HLS Setup**

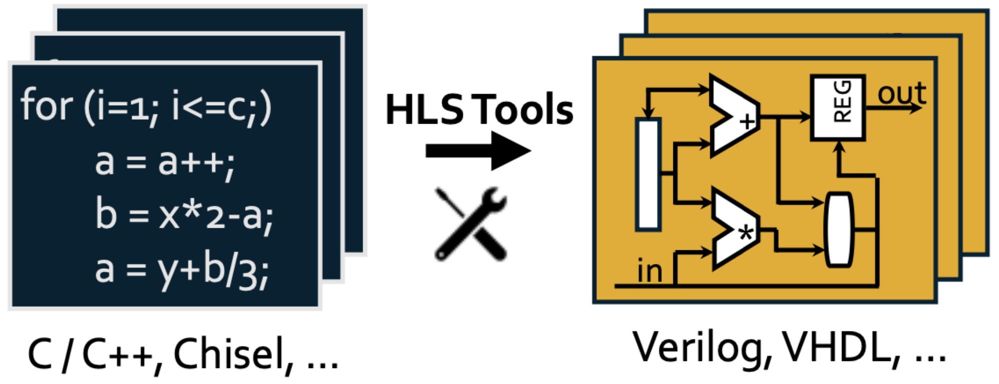
- Xilinx **Vitis/Vivado** HLS has a graphical user interface that we intend to use
- The goal is to run ***vivado_hls*** or ***vitis_hls*** on ***login*** machine but be able to do so remotely
- So, we want to display the ***login*** screen on your desktop (Mac or Windows or Linux)
- In principle one can use X-Windows directly. However, that will be very slow over WAN
- Therefore, we suggest using a VNC server on cmstrigger02 and a remote machine

- Follow instructions at: https://github.com/varuns23/TAC-HEP-FPGA/tree/main/hls-setup

## What is HLS (High-Level Synthesis)

HLS is an automated design process that transforms a high-level functional specification to an optimized register-transfer level (RTL) descriptions for efficient hardware implementation


Software Specification and Program
High-Level Synthesis
Logic Synthesis
Physical Synthesis
Circuit (ASIC, FPGA) Design

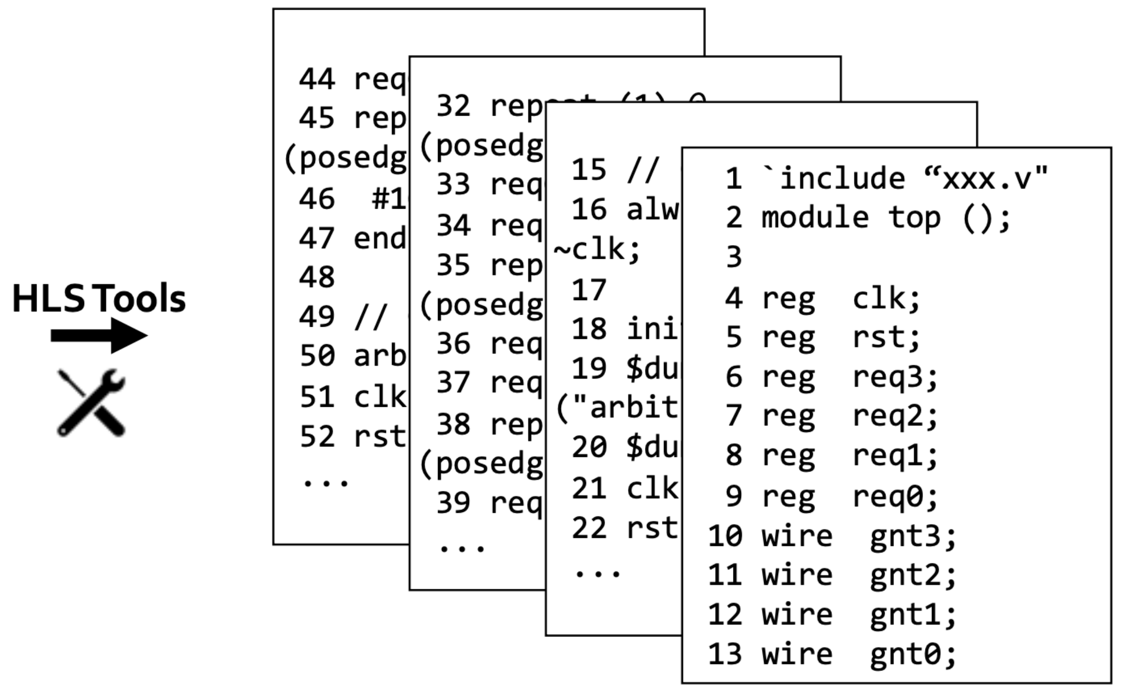


```
for (i=1; i<=c;)
        a = a++;
        b = x*2-a;
        a = y+b/3;
```

# What is HLS

```
for(int h = 0; h < H; h++ )
  for(int w = 0; w < W; w++)
    for(int m = 0; m < K; m++)
      for(int n = 0; n < K; n++)
        ...
```

Behavioral-level:
Expressive and Concise



```
`include “xxx.v"
module top ();

reg  clk;
reg  rst;
reg  req3;
reg  req2;
reg  req1;
reg  req0;
wire  gnt3;
wire  gnt2;
wire  gnt1;
wire  gnt0;
```

# Why use HLS?

- Productivity
  - Lower design complexity and faster simulation speed
  - Ease of use
- Portability
  - Single source -> multiple implementations (different target devices)
- Permutability
  - Much more optimization opportunities at higher level
  - Rapid design space exploration

## Summary

- The information does not cover all the details about the FPGA architecture

- Rather to a concise report of some useful information needed to understand the HLS reports and successfully use and leverage the HLS directives, many of which very specifically target modern FPGA architectural features.

TAC-HEP 2026

# Questions?

TAC-HEP 2026

# Extra Slides

## Connecting to “login” machines

- Connect to login machine:
  - ssh -X -Y <username>@login.hep.wisc.edu
  - mkdir /scratch/\`whoami\`   (If directory exist, go to next bullet)
  - cd /scratch/\`whoami\`

# VNC Server setup

- Log into cmstrigger02
- Set your VNC password using the linux command: vncpasswd
  - **Do NOT use an important password** here, as it is NOT secure
- Follow this instruction at http://red.ht/1fSVIUc to set up your X-Windows session
- Namely, you need to create a file ~/.vnc/xstartup with content:

```
#!/bin/sh
# Uncomment the following two lines for normal desktop:
# unset SESSION_MANAGER
# exec /etc/X11/xinit/xinitrc
[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
#xsetroot -solid grey
#vncconfig -iconic &
#xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
#twm &
if test -z "$DBUS_SESSION_BUS_ADDRESS" ; then
    eval `dbus-launch --sh-syntax ?exit-with-session`
    echo "D-BUS per-session daemon address is: \
    $DBUS_SESSION_BUS_ADDRESS"
fi
exec gnome-session
```

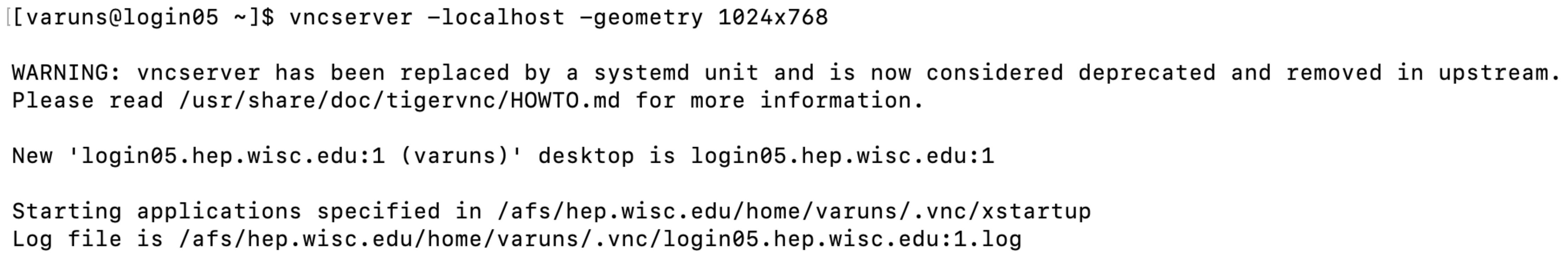
Can be copied from above link as well

- You need to set execute permission for the startup file
  - chmod +x ~/.vnc/xstartup

## IP Port forwarding

- Start the VNC server - you do this command after you stopped vncserver by hand or otherwise, using:
  - vncserver -localhost -geometry 1024x768

- This command, vncserver, tells you the number of your X-Windows Display, example ***login.hep.wisc.edu:1***, where **:1** is your display



```
[varuns@login05 ~]$ vncserver -localhost -geometry 1024x768

WARNING: vncserver has been replaced by a systemd unit and is now considered deprecated and removed in upstream.
Please read /usr/share/doc/tigervnc/HOWTO.md for more information.

New 'login05.hep.wisc.edu:1 (varuns)' desktop is login05.hep.wisc.edu:1

Starting applications specified in /afs/hep.wisc.edu/home/varuns/.vnc/xstartup
Log file is /afs/hep.wisc.edu/home/varuns/.vnc/login05.hep.wisc.edu:1.log
```

Ignore the warning

# IP Port forwarding

- Start the VNC server - you do this command after you stopped vncserver by hand or otherwise, using:
  - vncserver -localhost -geometry 1024x768

- This command, vncserver, tells you the number of your X-Windows Display, example ***login.hep.wisc.edu:1***, where **:1** is your display

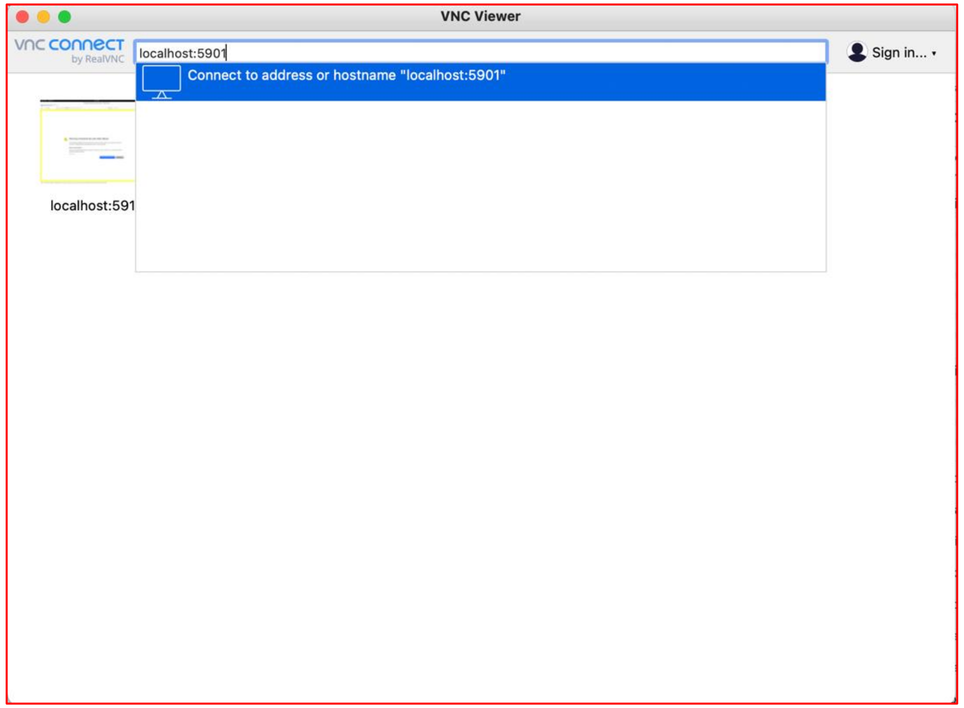
- ssh <username>@login.hep.wisc.edu -L5902:localhost:5902 [In separate terminal tab]
- Make sure you change "<username> to your user name, and "5902" to (5900 + your display number), say 5903, if vncserver told you 3!

- You can kill your VNC server (:2) using the command:
  - vncserver –kill :2
  - vncserver –list (check active servers and kill unnecessary ones)

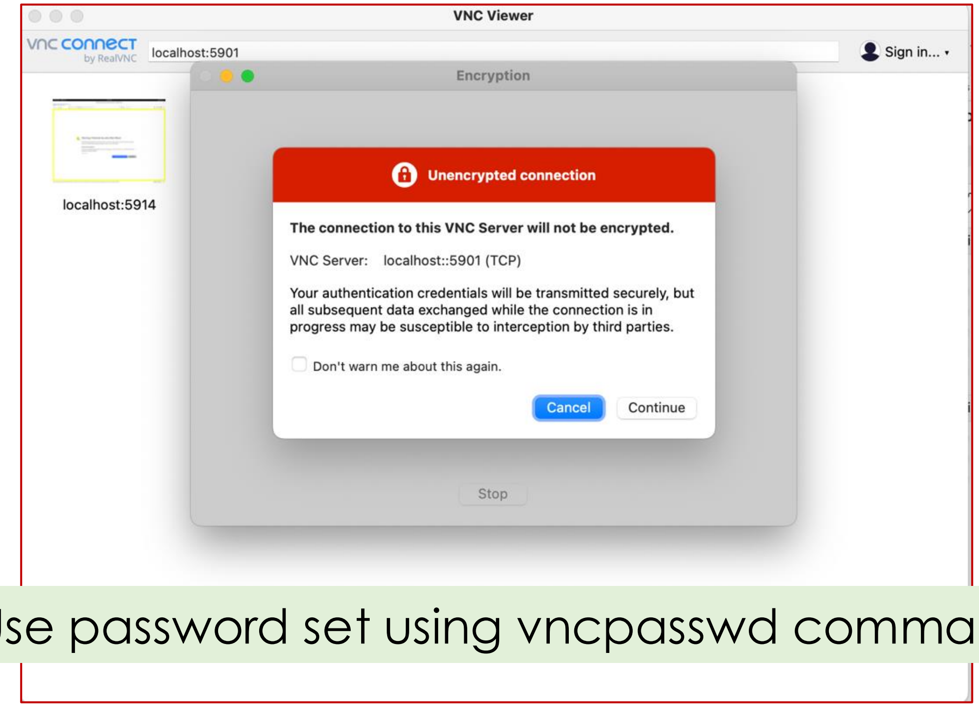
# Remote desktop client

One time only

- Download VNC viewer: [https://www.realvnc.com/en/connect/download/viewer/](https://www.realvnc.com/en/connect/download/viewer/)
  - You can choose any other remote desktop client but this is one of the stable one that I have used



Connect to address or hostname "localhost:5901"



Unencrypted connection

The connection to this VNC Server will not be encrypted.

VNC Server: localhost::5901 (TCP)

Your authentication credentials will be transmitted securely, but all subsequent data exchanged while the connection is in progress may be susceptible to interception by third parties.

☐ Don't warn me about this again.

Cancel Continue

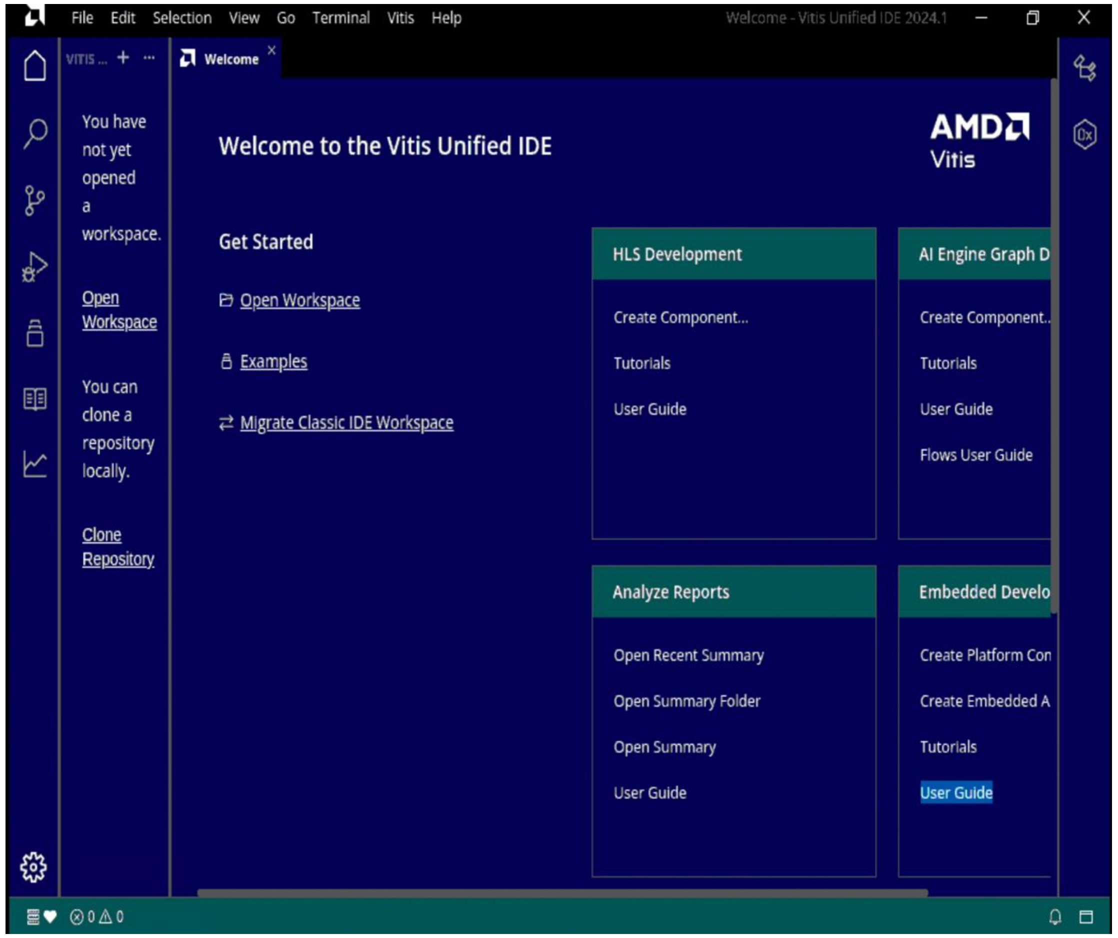
Use password set using vncpasswd command

# Connected

**Everytime**

## Summary

- ssh varuns@login.hep.wisc.edu -L5901:localhost:5901
  - Or whatever *:1* display number
  - Sometimes you may need to run vncserver -localhost -geometry 1024x768 again to start new vnc server
- Connect to VNC server (remote desktop) client
- Open terminal
  - Source /opt/Xilinx/Vitis/2024.1/settings64.sh
  - vitis_hls


File Edit Selection View Go Terminal Vitis Help
Welcome - Vitis Unified IDE 2024.1
VITIS ...
Welcome

You have not yet opened a workspace.

Open Workspace

You can clone a repository locally.

Clone Repository

Welcome to the Vitis Unified IDE

AMD Vitis

Get Started

Open Workspace

Examples

Migrate Classic IDE Workspace

HLS Development

Create Component...

Tutorials

User Guide

AI Engine Graph D

Create Component...

Tutorials

User Guide

Flows User Guide

Analyze Reports

Open Recent Summary

Open Summary Folder

Open Summary

User Guide

Embedded Develo

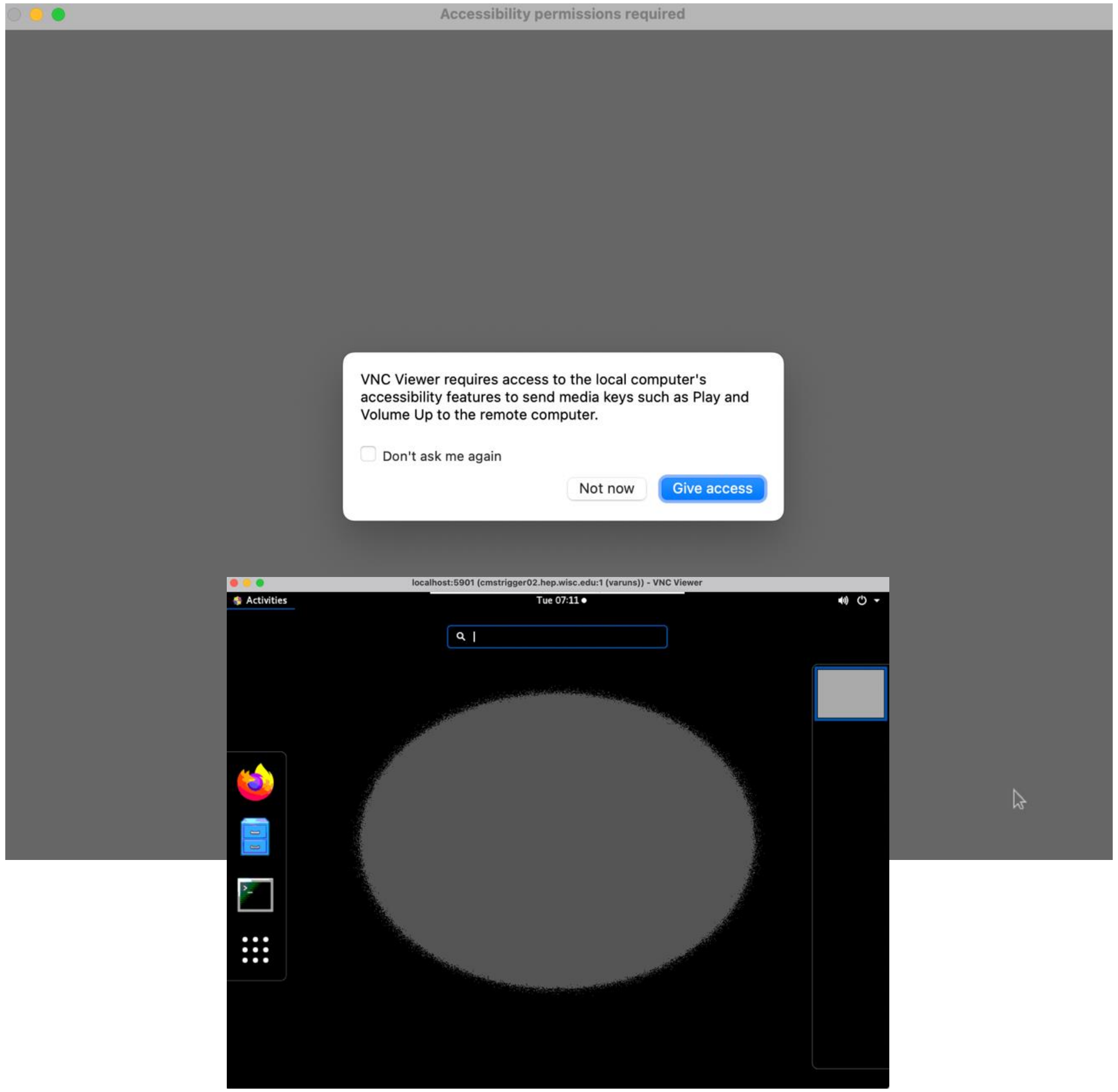
Create Platform Con

Create Embedded A

Tutorials

User Guide

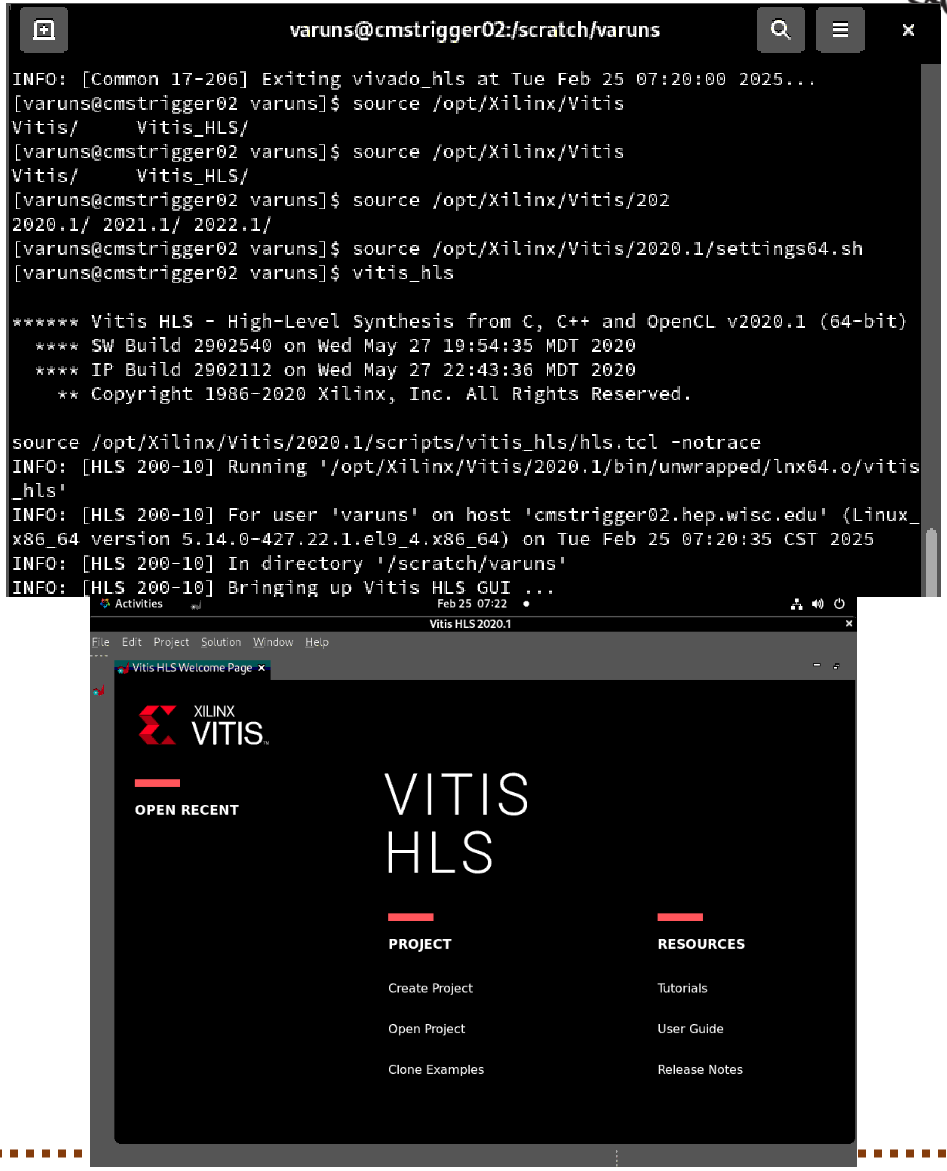
# Connected


Accessibility permissions required
localhost:5901 (cmstrigger02.hep.wisc.edu:1 (varuns)) - VNC Viewer
Activities
Tue 07:11 ●

VNC Viewer requires access to the local computer's accessibility features to send media keys such as Play and Volume Up to the remote computer.

Don't ask me again

Not now | Give access



```
INFO: [Common 17-206] Exiting vivado_hls at Tue Feb 25 07:20:00 2025...
[varuns@cmstrigger02 varuns]$ source /opt/Xilinx/Vitis
Vitis/     Vitis_HLS/
[varuns@cmstrigger02 varuns]$ source /opt/Xilinx/Vitis
Vitis/     Vitis_HLS/
[varuns@cmstrigger02 varuns]$ source /opt/Xilinx/Vitis/202
2020.1/ 2021.1/ 2022.1/
[varuns@cmstrigger02 varuns]$ source /opt/Xilinx/Vitis/2020.1/settings64.sh
[varuns@cmstrigger02 varuns]$ vitis_hls

****** Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2020.1 (64-bit)
  **** SW Build 2902540 on Wed May 27 19:54:35 MDT 2020
  **** IP Build 2902112 on Wed May 27 22:43:36 MDT 2020
    ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.

source /opt/Xilinx/Vitis/2020.1/scripts/vitis_hls/hls.tcl -notrace
INFO: [HLS 200-10] Running '/opt/Xilinx/Vitis/2020.1/bin/unwrapped/lnx64.o/vitis
_hls'
INFO: [HLS 200-10] For user 'varuns' on host 'cmstrigger02.hep.wisc.edu' (Linux_
x86_64 version 5.14.0-427.22.1.el9_4.x86_64) on Tue Feb 25 07:20:35 CST 2025
INFO: [HLS 200-10] In directory '/scratch/varuns'
INFO: [HLS 200-10] Bringing up Vitis HLS GUI ...
```