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

GPU & FPGA module training: Part-2

Week-2: FPGA: Clock Frequency, Latency, Pipelining

Lecture-3: March 28th 2023

Varun Sharma

University of Wisconsin – Madison, USA

So Far…

- **FPGA and its architecture**
  - Registor/Flip-Flops, LUTs/Logic Cells, DSP, BRAMs
- **Parallelism in FPGA**
  - Scheduling, Pipelining, DataFlow

Today:

- FPGA: Clock Frequency, Latency, Pipelining
- Extracting Control Logic & Implementating I/O Ports
- Vivado HLS introduction

Clock Frequency

- Important metric to determine the choice of processor
- *In general*: High clock frequency means higher performance execution rate
  - Can be misleading

Fig. 1

Maximum clock frequency:

FPGA	500 MHz
Processor	2 GHz

Which is better?

Stages		Description
IF	**Instructions Fetch**	Get the instruction from program memory
ID	**Instruction decode**	Decode the instruction to determine the operation and the operators
EXE	**Execute**	Execute the instruction on the available hardware
MEM	**Memory Operation**	Fetch data for the next instruction using memory operations
WB	**Write Back**	Write the results of the instruction to local registers/global memory

Regardless of processor type: Execution instructions

Clock Frequency

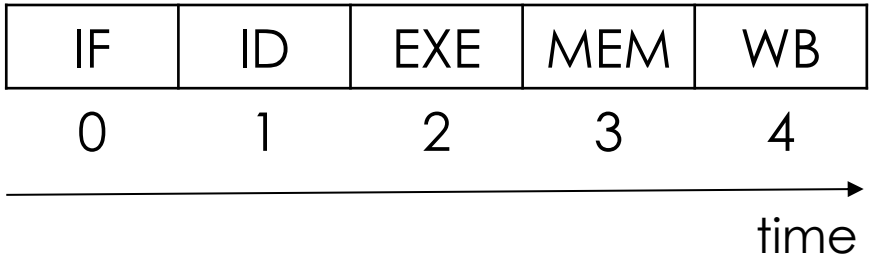
Fig. 2

FPGA	500 MHz
Processor	2 GHz

4 times better?

- A **processor** is able to execute any program on a common hardware platform
- The compiler, which has a built-in understanding of the processor architecture, compiles the user software into a set of instructions

Fig. 3



Processor instruction execution stages

Clock Frequency: FPGA

- **FPGA** does not execute all software on a common computation platform.
- BUT executes on custom circuit for that program
  - Therefore, any modification to program changes the circuit in the FPGA.

- Vivado HLS compiler does not need to account for overhead stages in the platform
  - Can find ways of maximizing instruction parallelism.

Fig. 4

Clock Frequency

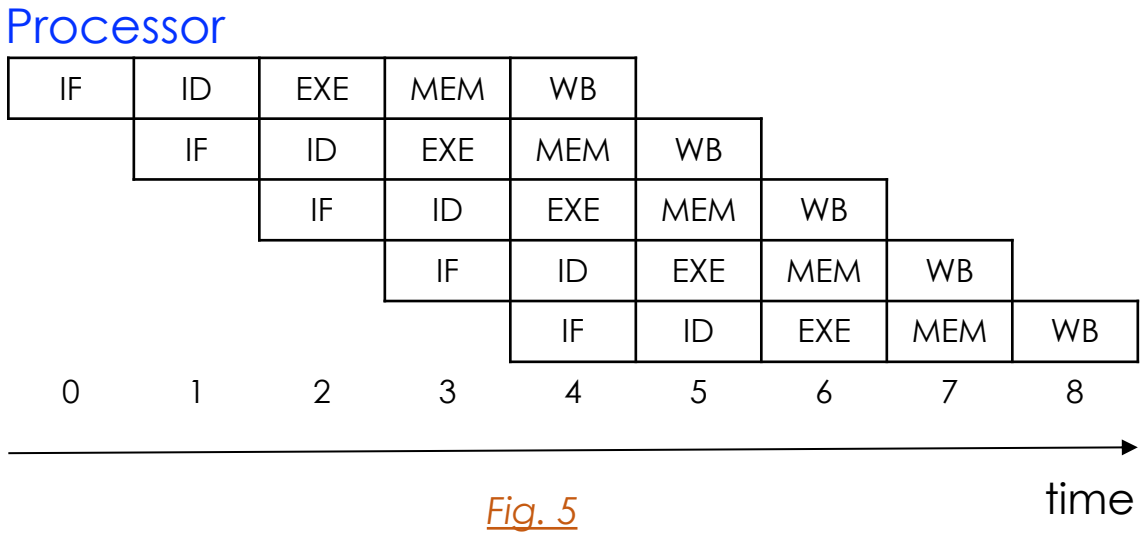


Fig. 5

< 9x faster

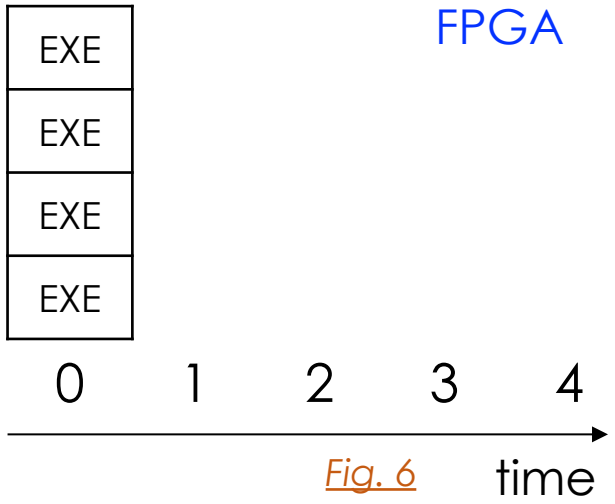


Fig. 6

Multiple instruction execution stages

FPGAs generally demonstrate at least 10x the performance

Approximate Power consumption = $\frac{1}{2}$ cF.V^2

FPGA is able to run at a lower clock frequency with maximum parallelism

- Thus lower power for same computational workload

Latency and Pipelining

Latency: number of clock cycles it takes to complete an instruction or set of instructions to generate an application result value

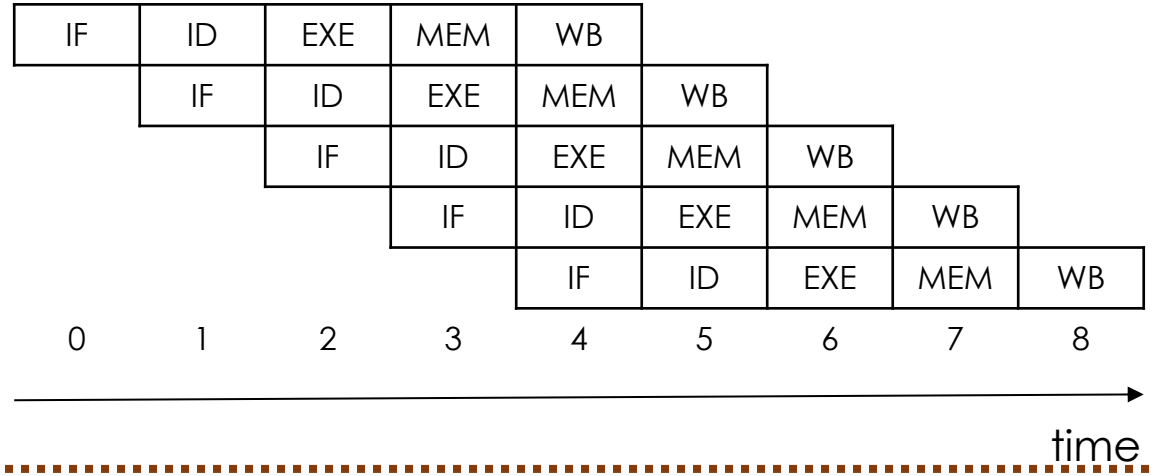
Fig. 7

Latency: **5 clk cyc**
For 5 set of instructions: **25 clk cycles**

Latency is another important key performance metric

Latency can be improved via pipelining

Fig. 8

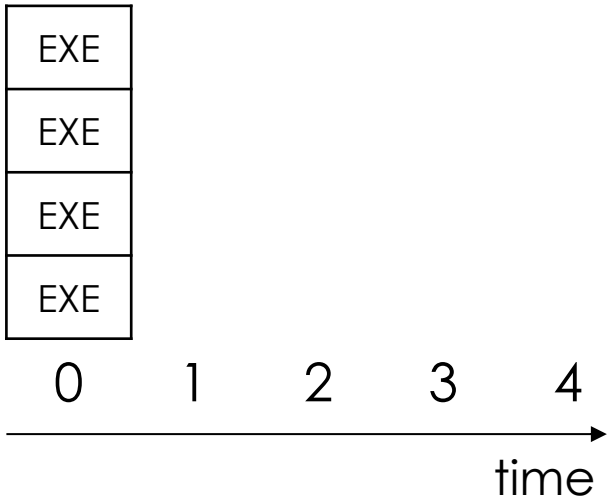


Latency: **5 clk cyc**
For 5 set of instructions with pipelining
9 clk cycles

Latency and Pipelining

Parallelism also plays an important role in reducing latency

Fig. 9



5 set of instructions

FPGA latency: 1 clock cycle

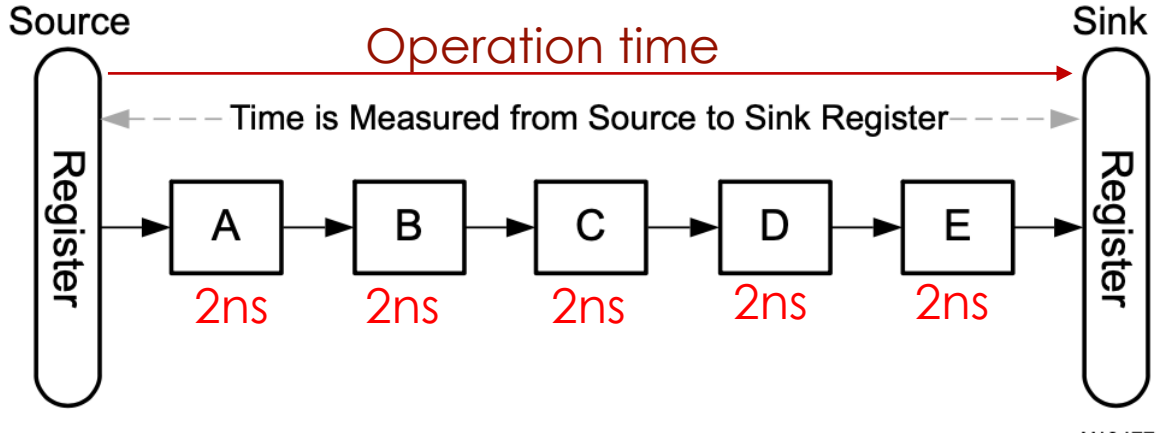
Do we need pipelining in One clock cycle latency of the FPGA?

- The reason for pipelining in an FPGA is to improve application performance

Latency and Pipelining

Reminder: *FPGA is a blank slate with building blocks that must be connected to implement an application*

Fig. 10



FPGA implementation without pipelining

Example:

- Each block takes 2 ns to execute
- Current design (5 stages of implementation): 10 ns
- Latency: 1 clock cycle
- Clock frequency: $\frac{1}{5 \times 2\ ns} = \frac{1}{10\ ns}$ =100 MHz

Clock Frequency: longest signal travel time between source and sink registers

Latency and Pipelining

Technique to avoid data dependencies and increase the level of parallelism

- Pipelining in an FPGA is the process of inserting more registers to break up large computation blocks into smaller segments.

- Partitioning of the computation increases the latency in absolute number of clock cycles but increases performance by allowing the custom circuit to run at a higher clock frequency

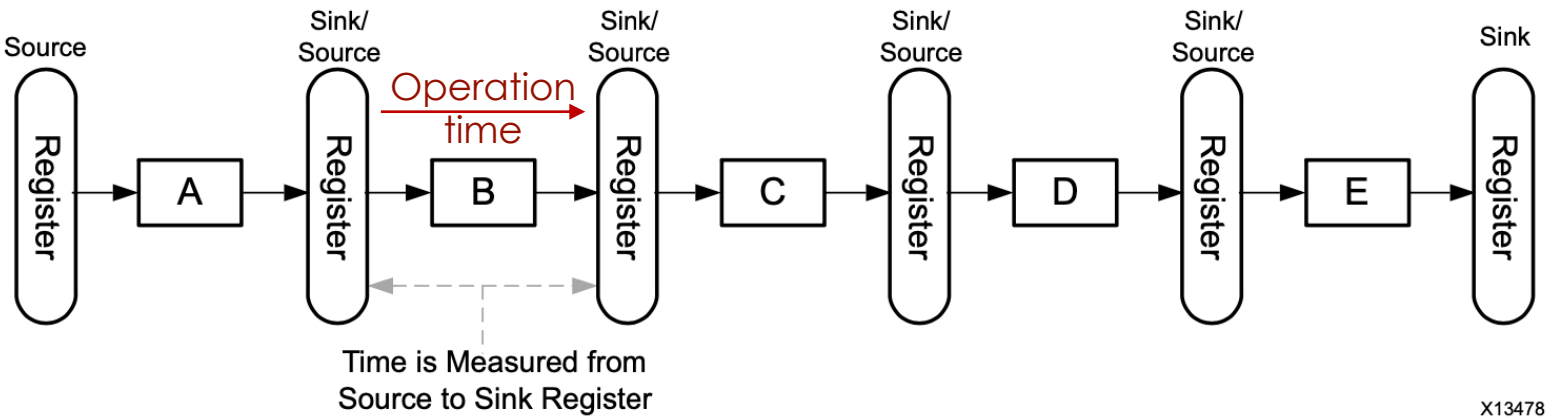


Fig. 11

FPGA implementation with pipelining

Latency and Pipelining

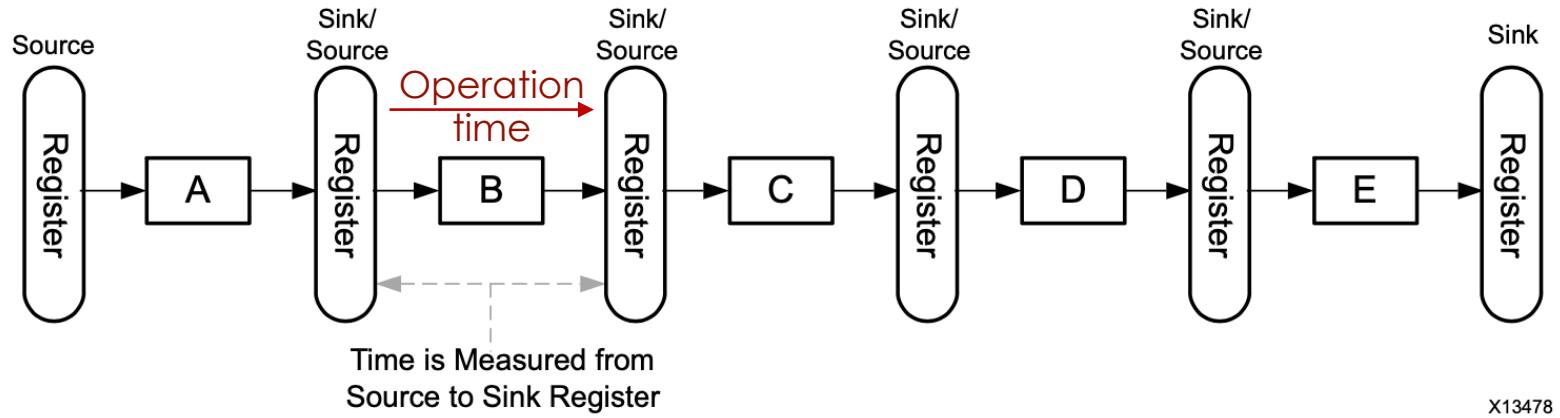


Fig. 12

FPGA implementation with pipelining

- Addition of registers reduces the timing requirement of the circuit from 10 ns to 2 ns,
- Results in a maximum clock frequency of 500 MHz.

- In addition, by separating the computation into separate register-bounded regions, each block is allowed to always be busy, which positively impacts the application **throughput**

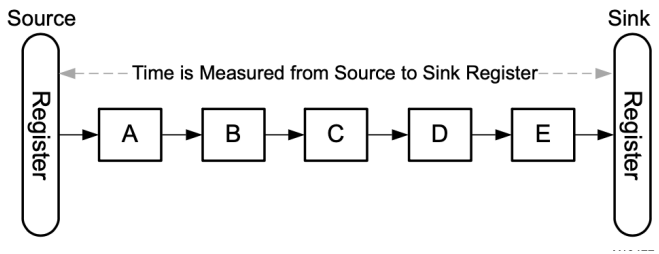
The latency caused by pipelining is one of the trade-offs to consider during FPGA design

Throughput

- Another another metric used to determine overall performance of an implementation

- Number of clock cycles it takes for the processing logic to accept the next input data sample
- Throughput changes with clock frequency

Fig. 13



10 ns between input samples

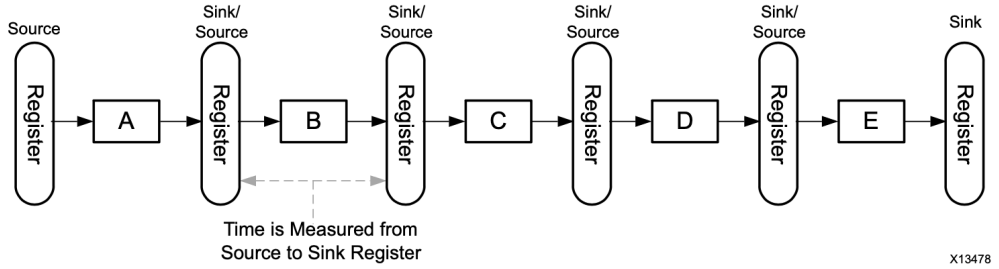


Fig. 14

2 ns between input samples

Second implementation has higher performance, because it can accept a higher input data rate

TAC-HEP 2023

Control Logic & Implementation

Reminder from Last week

E.g.: Scheduling phases for a simple code

```
int foo(char x, char a, char b, char c) {
 char y;
 y = x*a+b+c;
 return y;
}
```

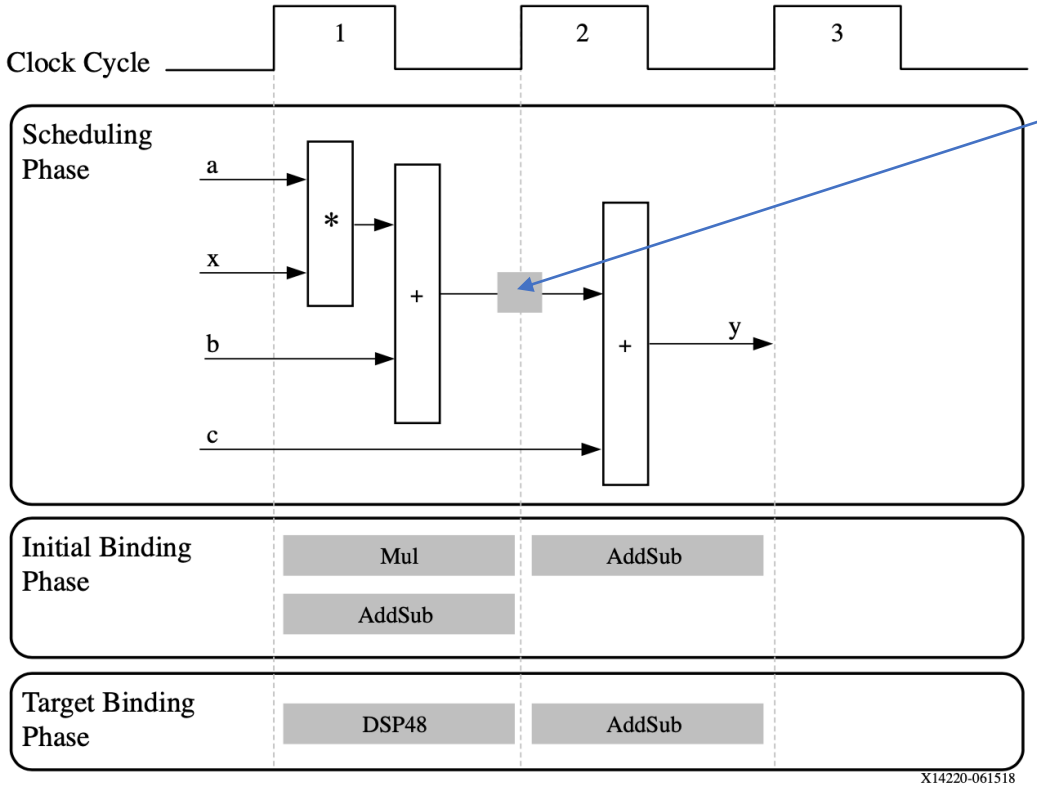


Fig. 15

First cycle: reads x, *a*, and *b* data ports
Second cycle: reads data port c & generates output *y*

Internal register storing a variable

First cycle: Multiplication and the first addition
Second cycle: Second addition and output generation

Extracting control logic & Implementing I/O

Let's talk about a slightly more complex example:

- Same operation inside a for-loop and two of function arguments as arrays

Fig. 16

```
void foo(int in[3], char a, char b, char c, int out[3]) {
  int x,y;
  for(int i = 0; i < 3; i++){
    x = in[i];
    y = a*x + b + c;
    out[i] = y;
  }
}
```

- Resulting design executes logic inside the for-loop 3-times when code is scheduled
- HLS extracts the control logic from the C code & creates FSM in the RTL design to sequence these operations.
- HLS implements the top-level function arguments as ports in the final RTL design

Extracting control logic & Implementing I/O

- Scalar variable of type *char* maps into a standard 8-bit data bus port
- Array arguments, such as *in* and *out*, contain an entire collection of data

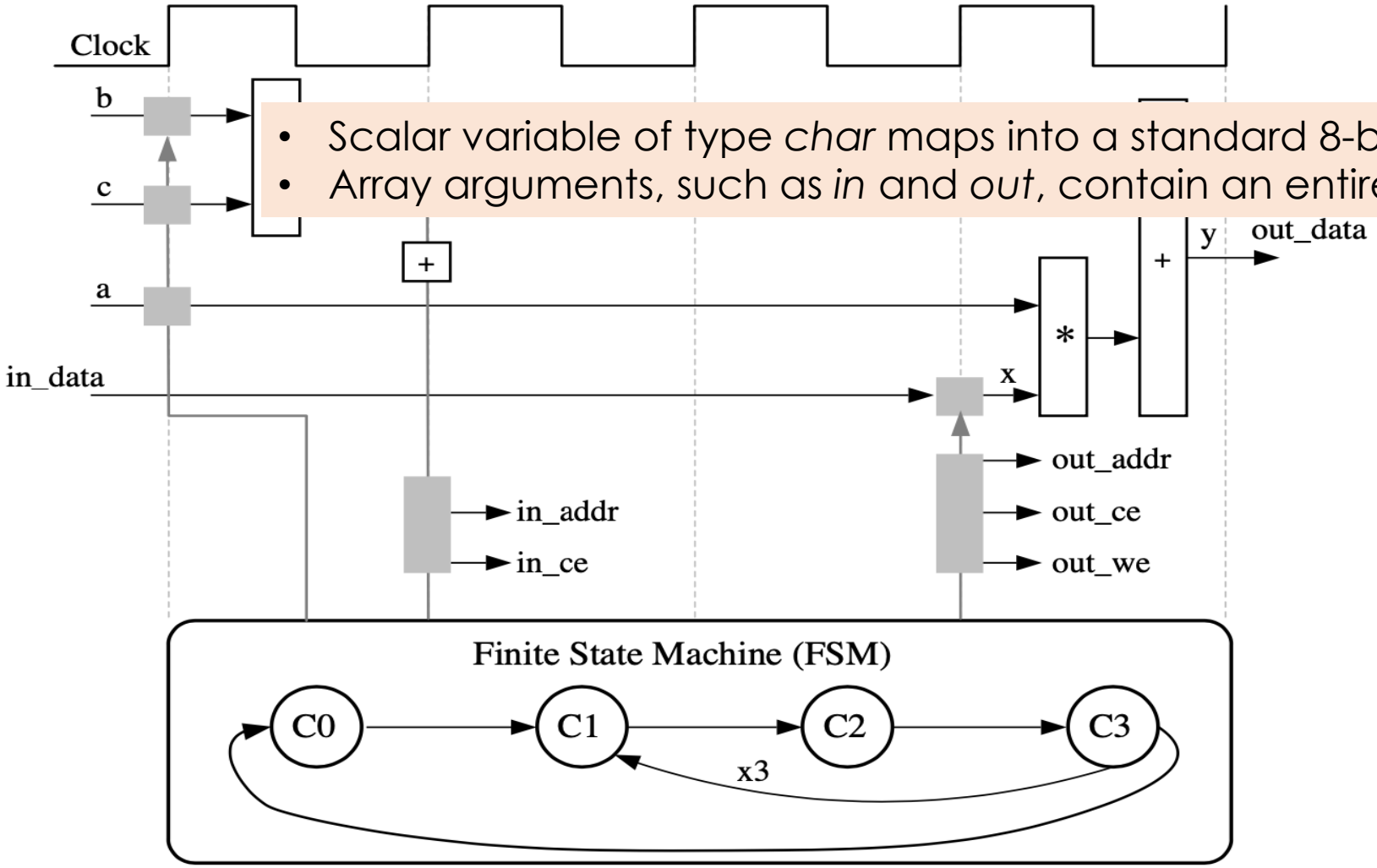


Fig. 17

Extracting control logic & Implementing I/O

- In HLS, arrays are synthesized into BRAM by default
  - Other possible options: FIFOs, distributed RAM, and individual registers
- Arrays as arguments in the top-level function: HLS assumes BRAM to be outside the top-level function
  - Creates ports to access a BRAM outside the design, such as data ports, address ports, and any required chip-enable or write-enable signals.
- The FSM controls when the registers store data and controls the state of any I/O control signals.
- The FSM starts in the state ***C0***. On the next clock, it enters state ***C1***, then state ***C2***, and then state ***C3***.
  - Returns to state ***C1*** (and ***C2***, ***C3***) a total of three times before returning to state ***C0***

Extracting control logic & Implementing I/O

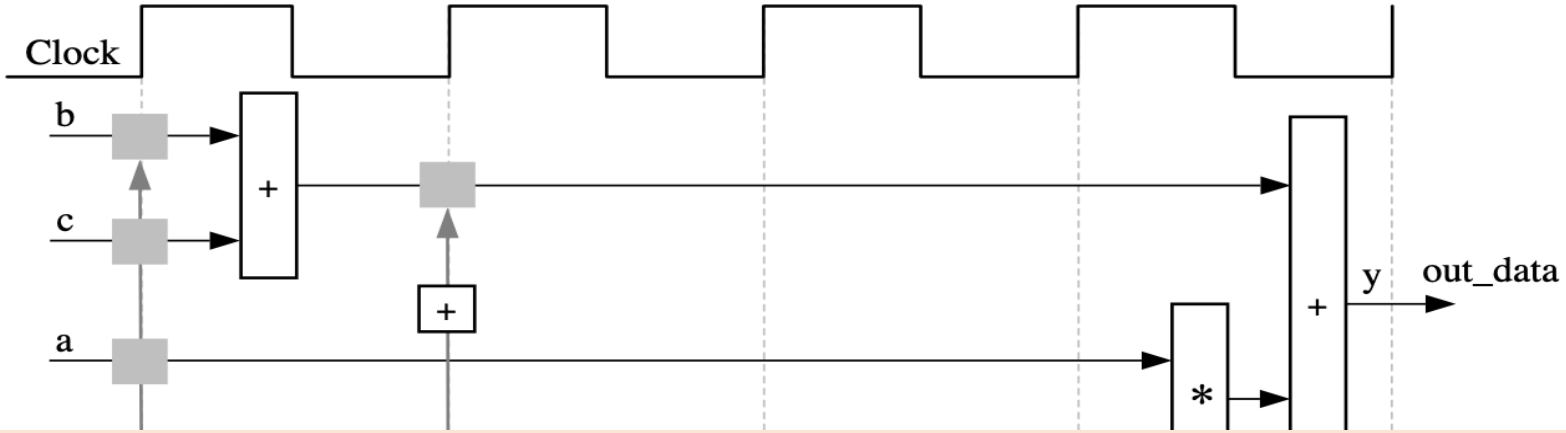
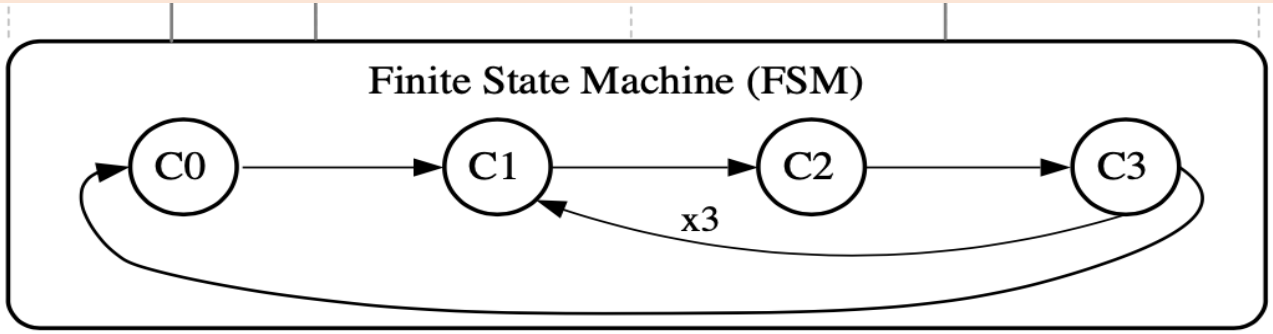


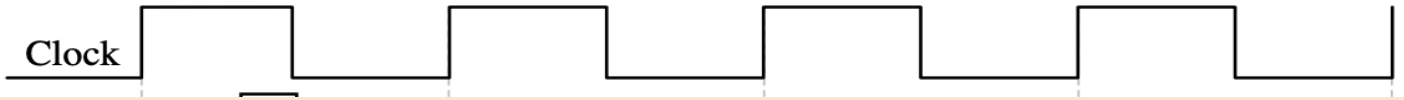
Fig. 18

C0

- Design requires the addition of *b* and *c* only one time
- HLS moves the *addition* operation outside the for-loop and into state C0
- Reuses the results of *addition*, each time the design enters state *C3*



Extracting control logic & Implementing I/O



C1:

- The FSM generates the address for the first element in state C1
- An adder increments to keep track of how many times the design must iterate around states C1, C2, and C3

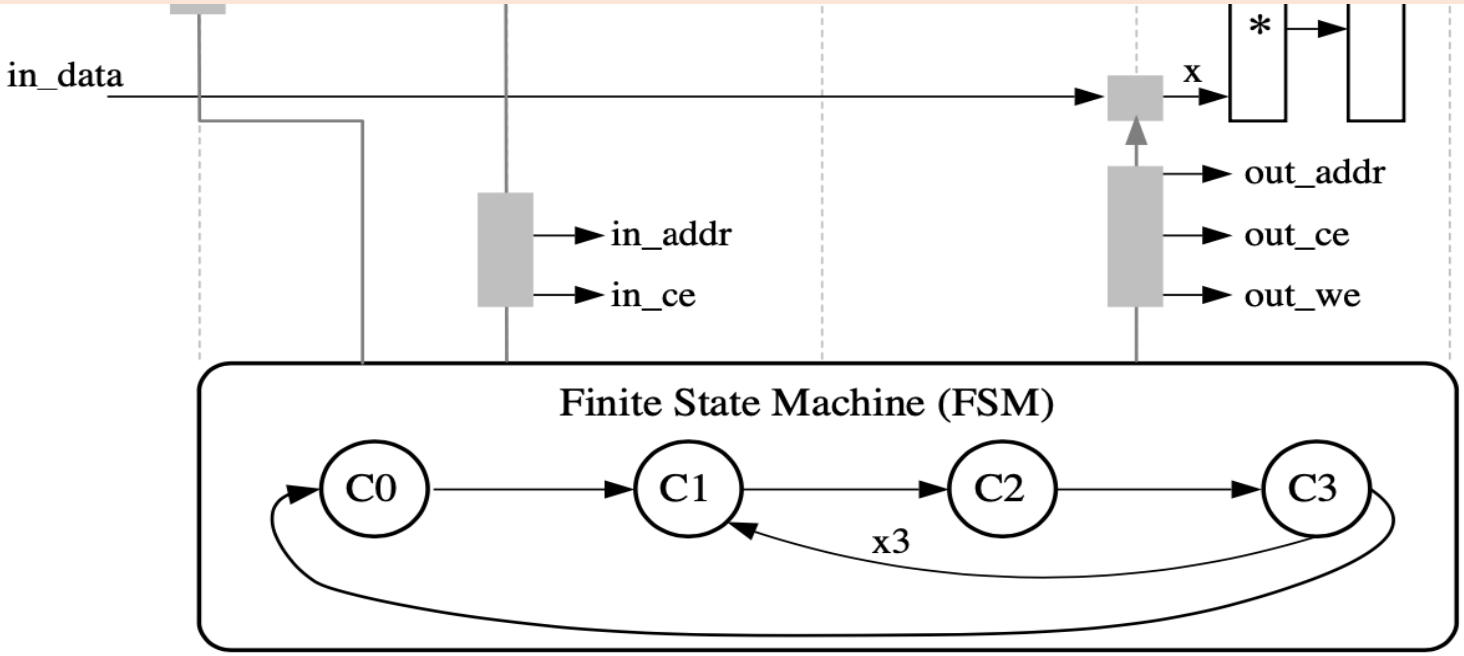
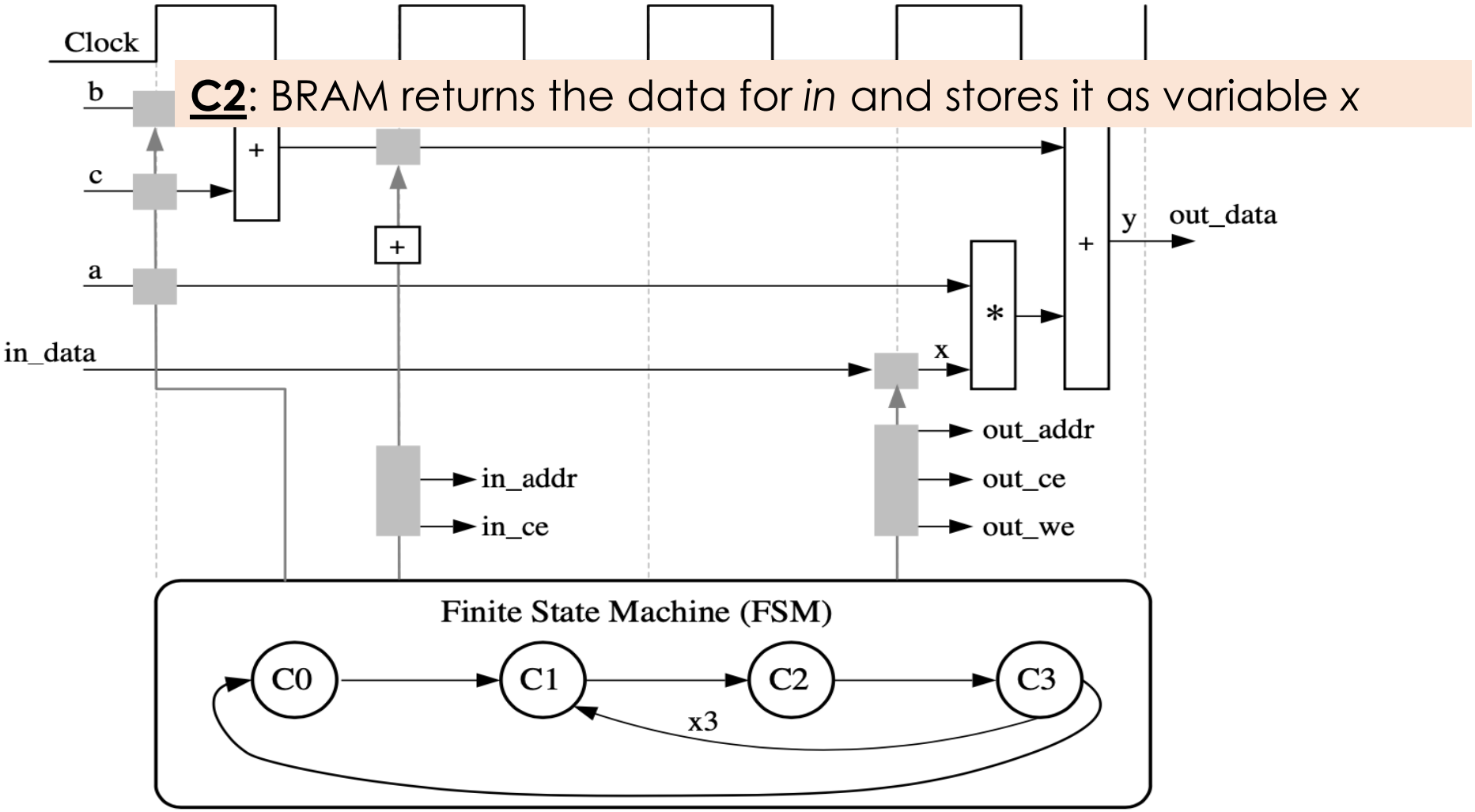


Fig. 19

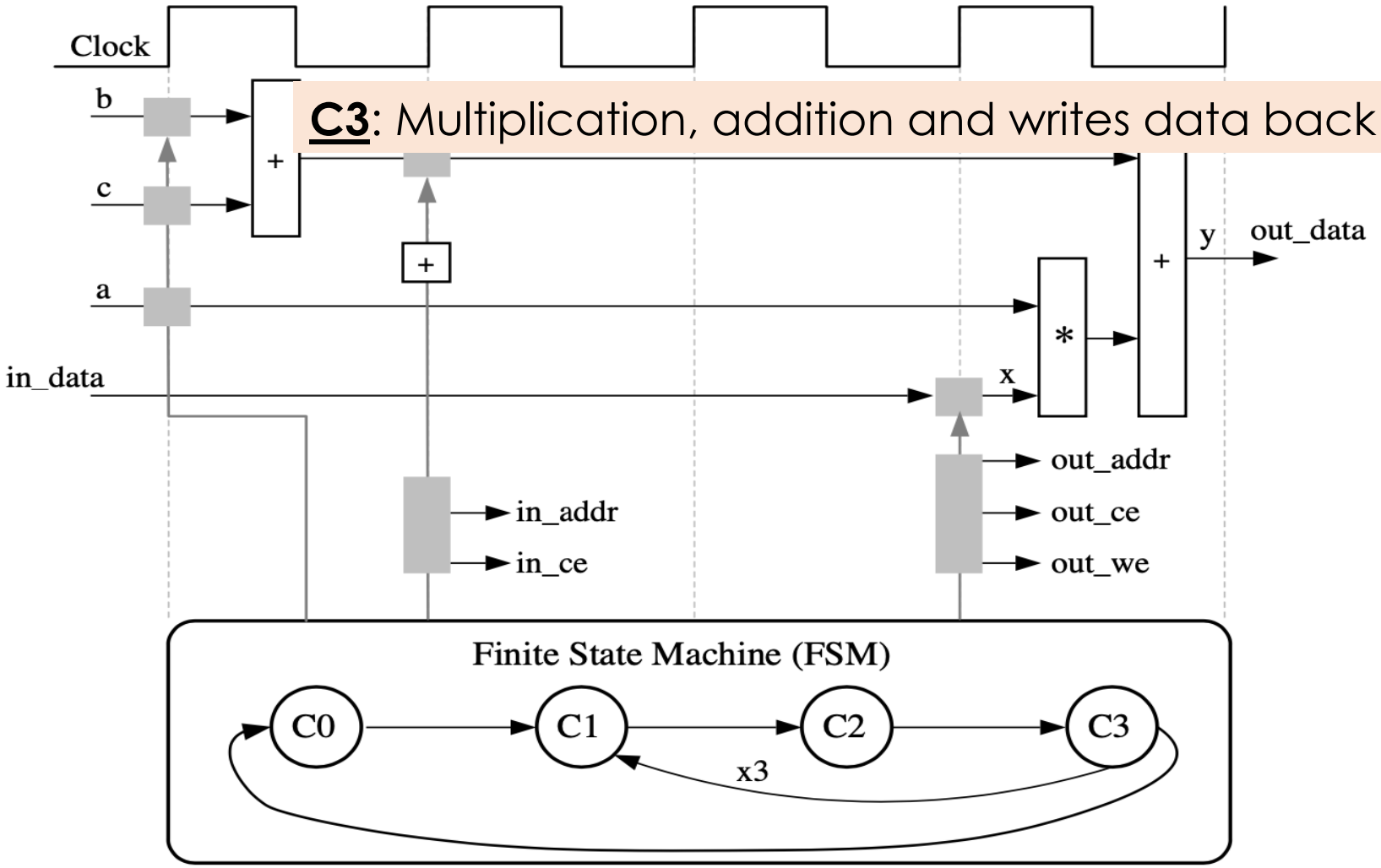
Extracting control logic & Implementing I/O



C2: BRAM returns the data for *in* and stores it as variable x

Fig. 20

Extracting control logic & Implementing I/O



C3: Multiplication, addition and writes data back

Fig. 21

Latency & Initiation Interval

Complete cycle-by-cycle execution for the code, including the states for each clock cycle, read operations, computation operations, and write operations

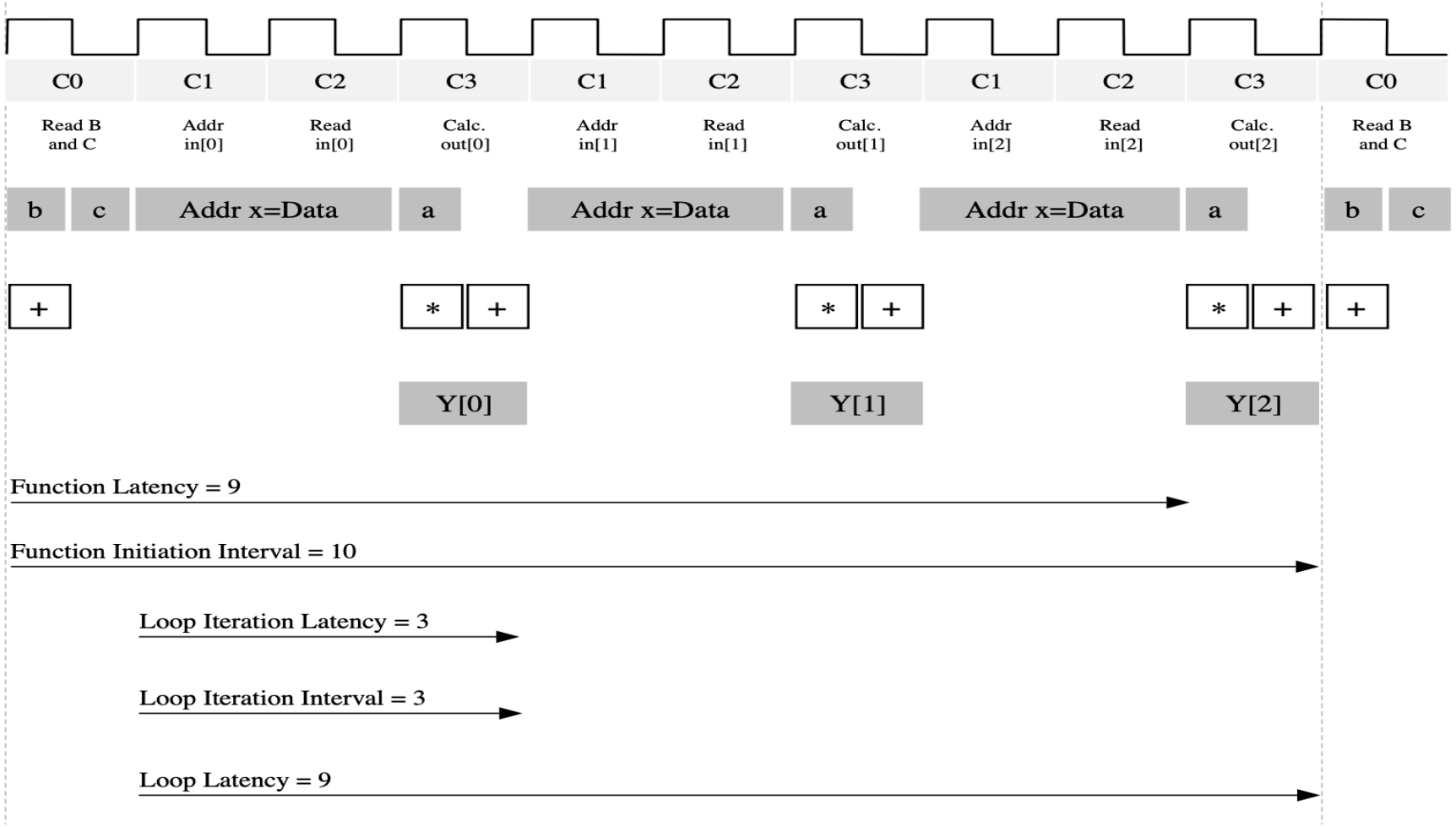


Fig. 22

Latency & Initiation Interval

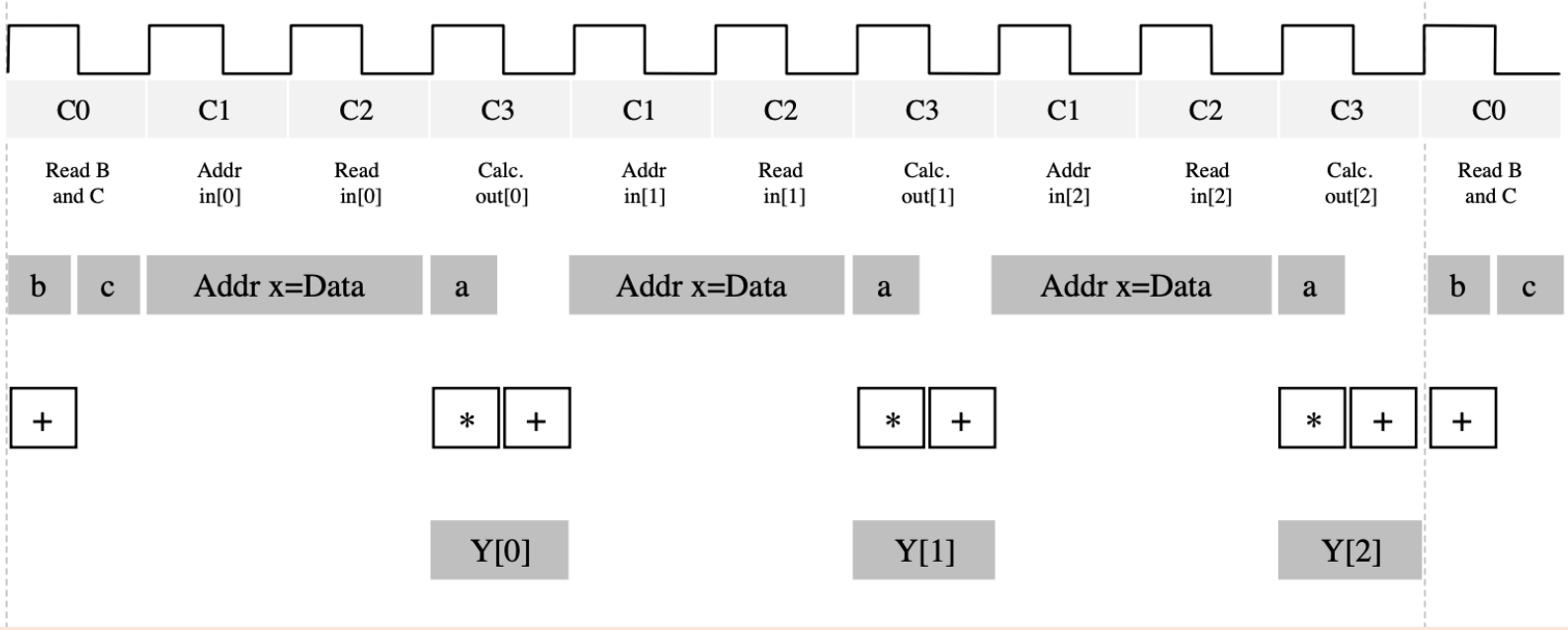


Fig. 23

- **Latency:** Function takes 9 clock cycles to output all values
  - When the output is an array: the latency is measured to the last array value output
- **Initiation Interval (II):** 10, i.e., it takes 10 clock cycles before the function can initiate a new set of input reads and start to process the next set of input data
- **Loop iteration latency:** The latency of each loop iteration is 3 clock cycles
- **Loop II:** The interval is 3.
- **Loop latency:** The latency is 9 clock cycles

TAC-HEP 2023

Vivado HLS

Vivado HLS

- The Xilinx Vivado HLS tool synthesizes a C function into an IP block that you can integrate into a hardware system
- Tightly integrated with the rest of the Xilinx design tools and provides comprehensive language support and features for creating the optimal implementation for your C algorithm

- **Following is the Vivado HLS design flow:**
  1. Compile, execute (simulate), and debug the C algorithm
  2. Synthesize the C algorithm into an RTL implementation, optionally using user optimization directives
  3. Generate comprehensive reports and analyze the design
  4. Verify the RTL implementation using a pushbutton flow
  5. Package the RTL implementation into a selection of IP formats

C-Simulation

Vivado HLS Design Flow

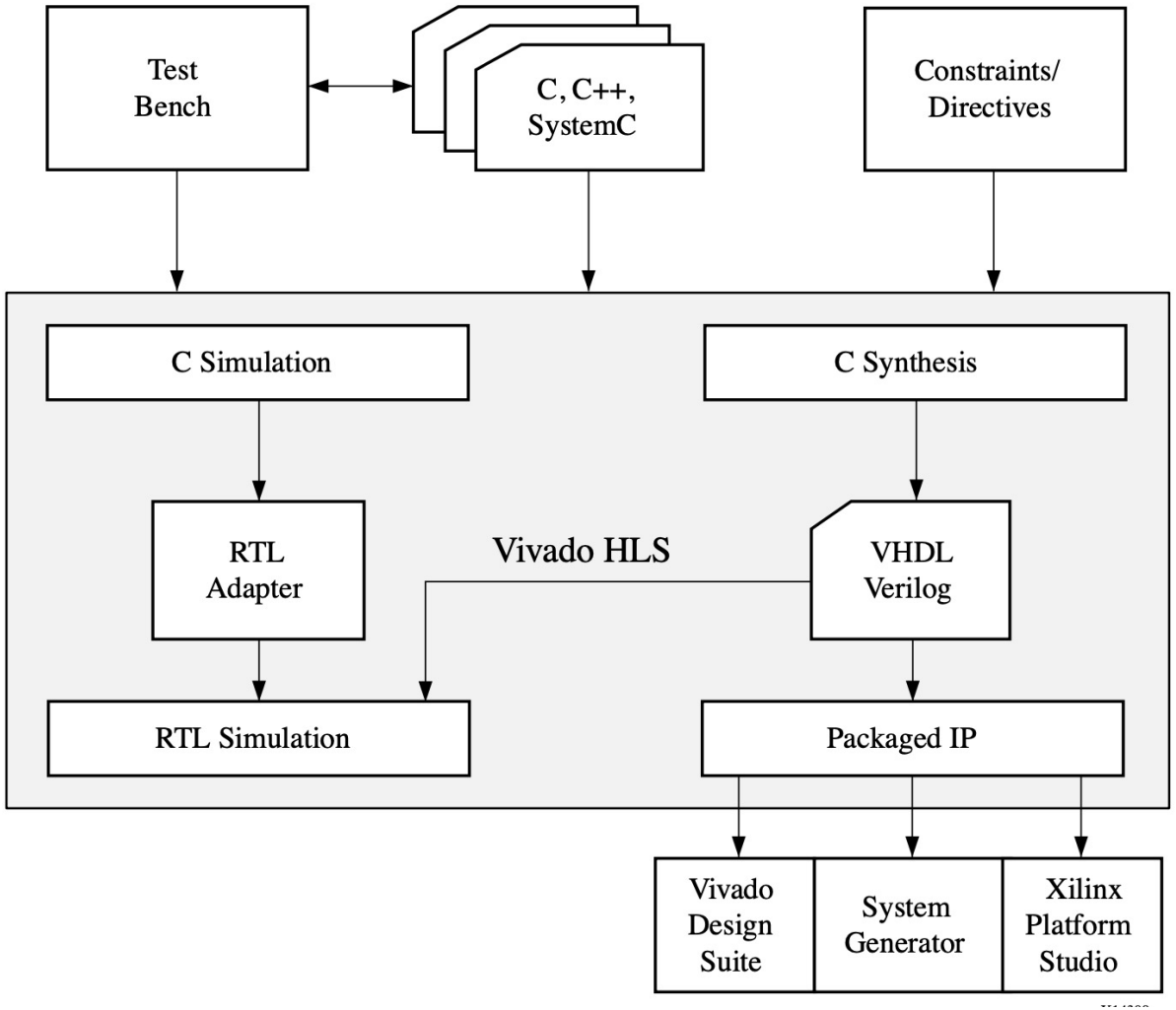


Fig. 24

TAC-HEP 2023

HLS Setup

- Xilinx Vivado HLS has a graphical user interface that we intend to use
- The goal is to run vivado_hls on *cmstrigger02* machine but be able to do so remotely
- So, we want to display the *cmstrigger02* 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

Connecting to cmstrigger02

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

- From 'login' machine connect to 'cmstrigger02' machine - All of you should have access
  - ssh cmstrigger02
  - mkdir /nfs_scratch/`whoami` (If directory exist, go to next bullet)
  - cd /nfs_scratch/`whoami`

VNC Server setup

One time only

- 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

Setting direct tunnelling

One time only

- Add to your (laptop or computer) ~/.ssh/config

```
Host *
  ControlPath ~/.ssh/control/%C
  ControlMaster auto

Host cmstrigger02-via-login
    User varuns
    HostName cmstrigger02.hep.wisc.edu
    ProxyCommand ssh login.hep.wisc.edu nc %h %p

Host *.wisc.edu
    User varuns
```

Replace "varuns" with your <username>

- If all is done correctly, following command should directly take you to *cmstrigger02* machine (enter passwd twice)
  - ssh cmstrigger02-via-login

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 *cmstrigger02.hep.wisc.edu:1*, where **:1** is your display

```
[varuns@cmstrigger02 tac-hep-fpga]$ vncserver -localhost -geometry 1024x768


New 'cmstrigger02.hep.wisc.edu:1 (varuns)' desktop is cmstrigger02.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/cmstrigger02.hep.wisc.edu:1.log
```

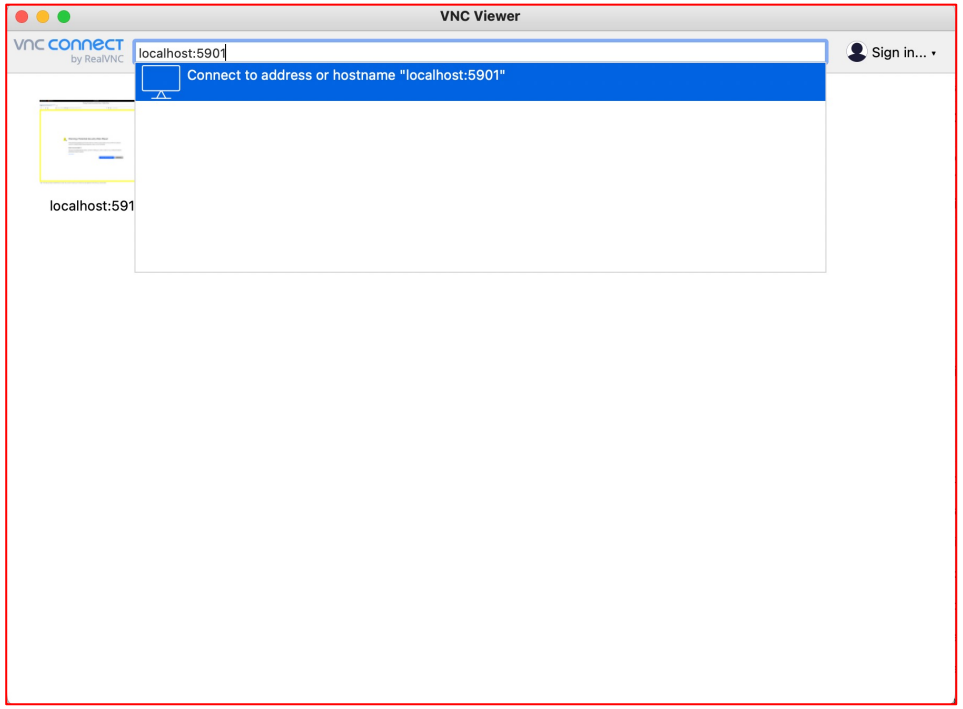
- We use an IP forwarding tunnel to *cmstrigger02.hep.wisc.edu* to see your *cmstrigger02* display on your laptop/desktop. The command to make that magic is:
  - ssh varuns@cmstrigger02-via-login -L5901:localhost:5901
  - Make sure you change "varuns" to your user name, and "5901" to (5900 + your display number), say 5903, if vncserver told you 3!

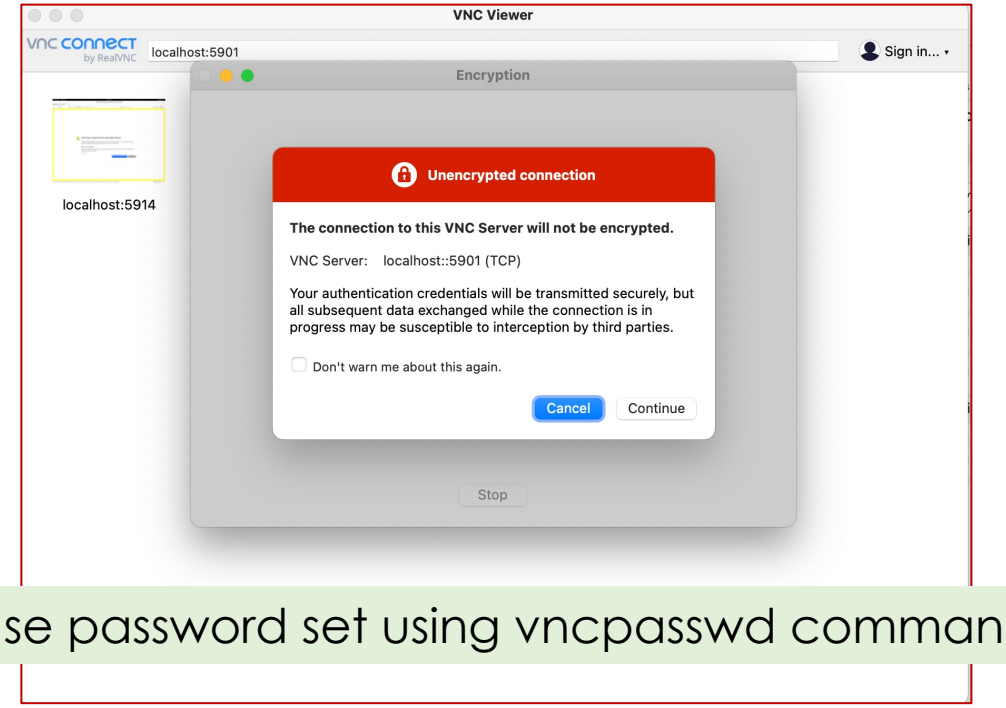
- You can kill your VNC server (:3) using the command:
  - vncserver -kill :1

Remote desktop client

One time only

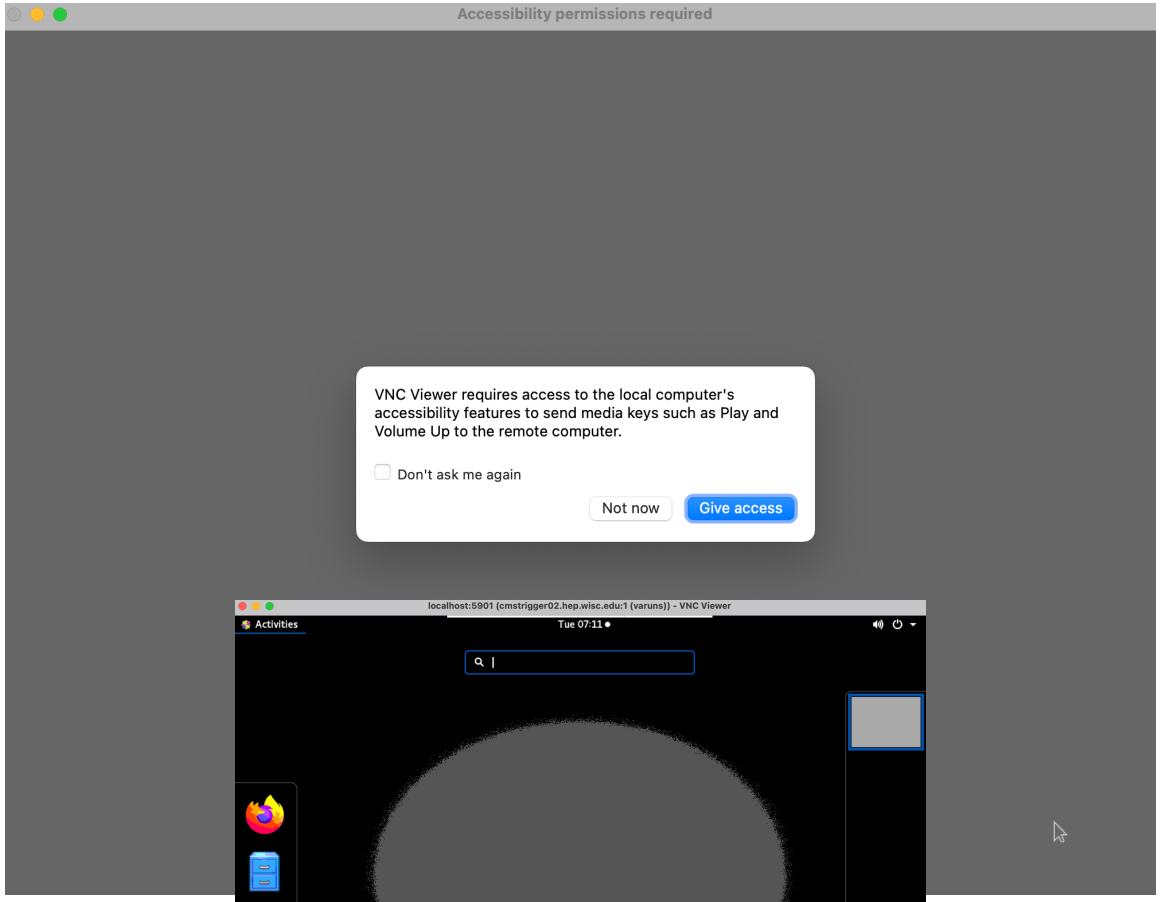
- Download VNC viewer: https://www.realvnc.com/en/connect/download/viewer/
  - You can choose any other remote desktop client but this is one of the stable one that I have used





Use password set using vncpasswd command

Connected...


Accessibility permissions required
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

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

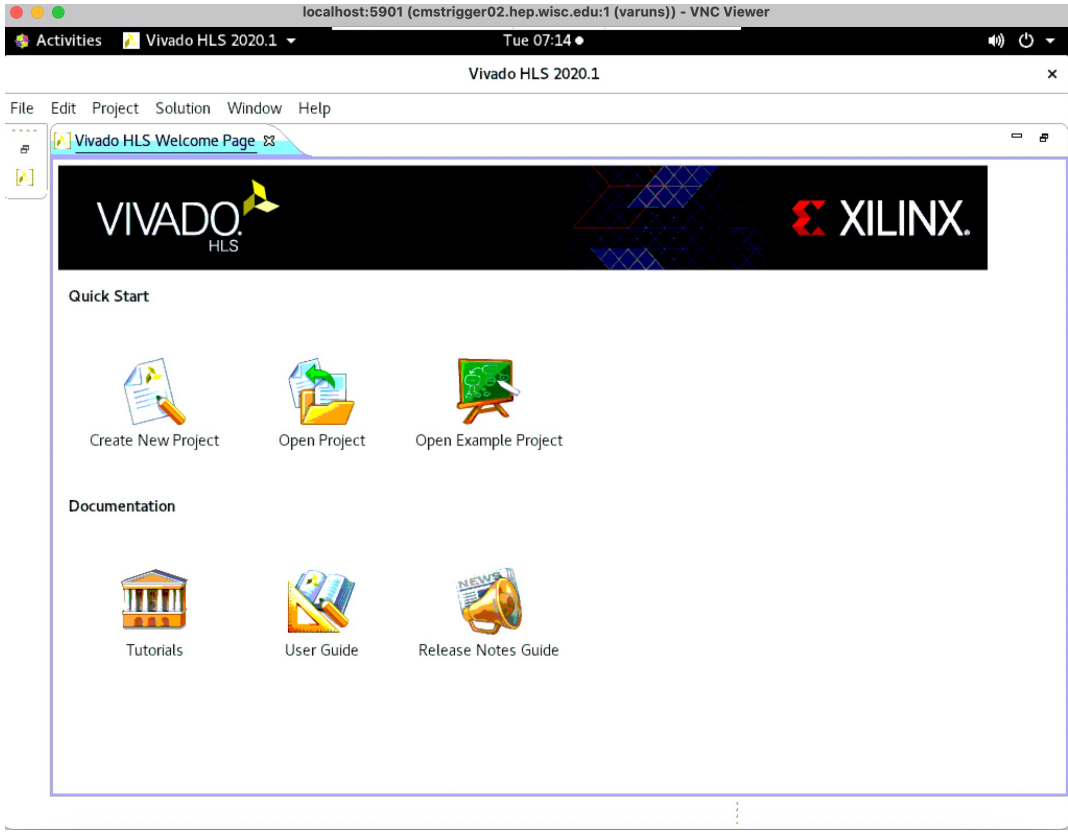

localhost:5901 (cmstrigger02.hep.wisc.edu:1 (varuns)) - VNC Viewer
Activities
Terminal
Tue 07:13
varuns@cmstrigger02:/afs/hep.wisc.edu/user/varuns
File Edit View Search Terminal Help
[varuns@cmstrigger02 varuns]$ source /opt/Xilinx/Vivado/2020.1/settings64.sh
[varuns@cmstrigger02 varuns]$ vivado_hls
VIVADO HLS
2020.1
XILINX
Copyright 1986-2020 Xilinx, Inc.
All Rights Reserved.

All set for hands-on

Everytime

Summary

- ssh varuns@cmstrigger02-via-login -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/Vivado/2020.1/settings64.sh
- vivado_hls



Homework: You are able to connect and bring this screen
Let me know in case of any issue

TAC-HEP 2023

Questions?

TAC-HEP 2023

Additional material

Correct Time

From 03.28.2023 onwards

- Tuesdays: 9:00-10:00 CT / 10:00-11:00 ET / 16:00-17:00 CET
- Wednesday: 11:00-12:00 CT / 12:00-13:00 ET / 18:00-19:00 CET

Jargons

- **ICs - Integrated chip:** assembly of hundreds of millions of transistors on a minor chip
- **PCB:** Printed Circuit Board
- **LUT - Look Up Table aka 'logic'** - generic functions on small bitwidth inputs. Combine many to build the algorithm
- **FF - Flip Flops** - control the flow of data with the clock pulse. Used to build the pipeline and achieve high throughput
- **DSP - Digital Signal Processor** - performs multiplication and other arithmetic in the FPGA
- **BRAM - Block RAM** - hardened RAM resource. More efficient memories than using LUTs for more than a few elements
- **PCIe or PCI-E - Peripheral Component Interconnect Express**: is a serial expansion bus standard for connecting a computer to one or more peripheral devices
- **InfiniBand** is a computer networking communications standard used in high-performance computing that features very high throughput and very low latency
- **HLS** - High Level Synthesis - compiler for C, C++, SystemC into FPGA IP cores
- **HDL** - Hardware Description Language - low level language for describing circuits
- **RTL** - Register Transfer Level - the very low level description of the function and connection of logic gates
- **FIFO** – First In First Out memory
- **Latency** - time between starting processing and receiving the result
  - Measured in clock cycles or seconds
- **II - Initiation Interval** - time from accepting first input to accepting next input