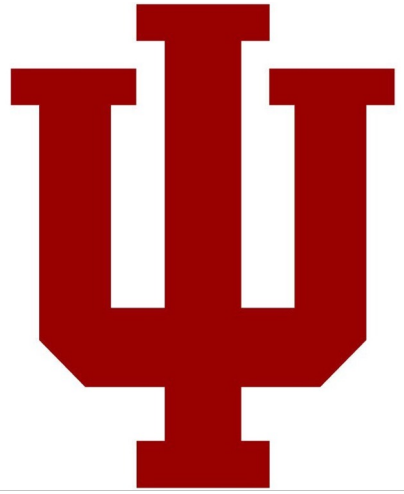
17: Pipelining II

ENGR 315: Hardware/Software CoDesign
Andrew Lukefahr & Grant Skipper
Indiana University



Some material taken from:
https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network
http://cs231n.github.io/neural-networks-1/

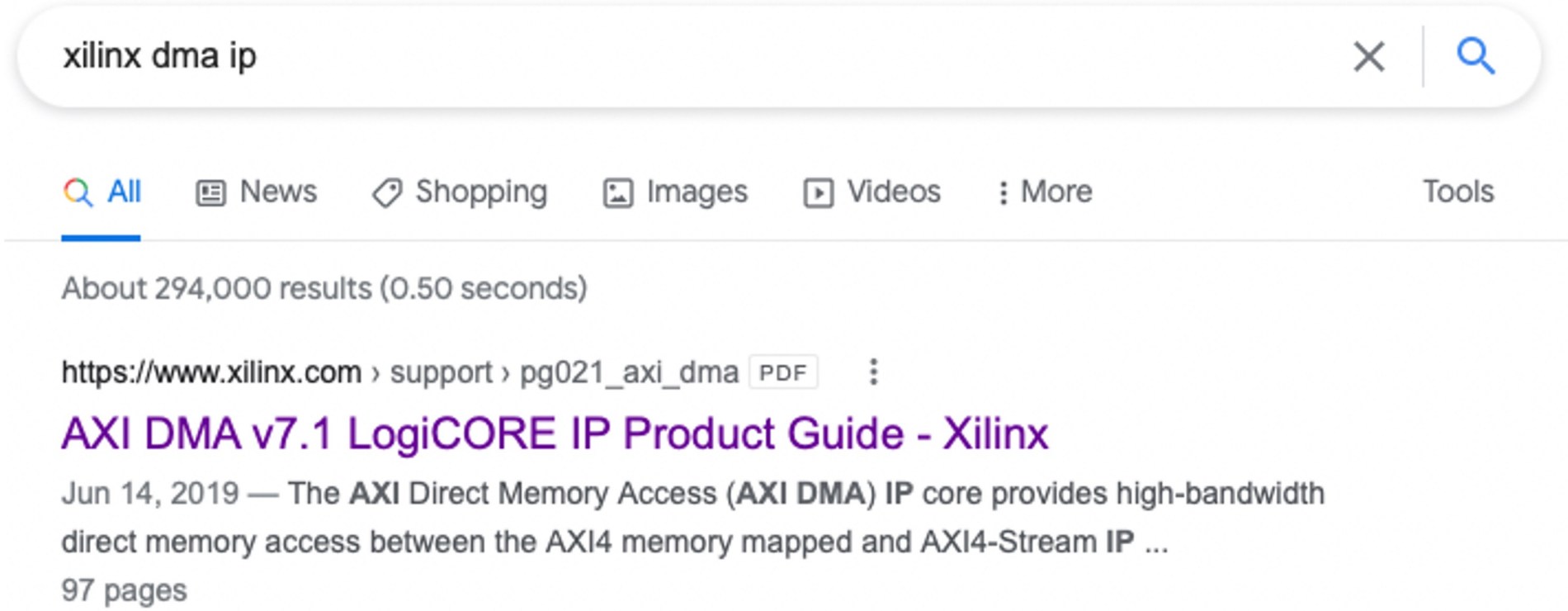
Announcements

- P6 out. Due Wednesday.
- No Class on Wednesday (Oct 26th)
- Exam Nov 6th

P6: Adds DMA + AXI-Stream to Popcount

- DMA
  - Add DMA engine to move data via AXI4-Full to AXI-Stream interface
- Popcount.sv:
  - Add AXI-Stream Interface
  - Keep AXI4-Lite Interface to read result

P7 – DMA from C



https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

P7 – DMA from C

Programming Sequence

Direct Register Mode (Simple DMA)

P7 – DMA from C

1. Start the MM2S channel running by setting the run/stop bit to 1 (MM2S_DMACR.RS = 1). The halted bit (DMASR.Halted) should deassert indicating the MM2S channel is running.
2. Skip
3. Write a valid source address to the MM2S_SA register.
4. Write the number of bytes to transfer in the MM2S_LENGTH register.

   The MM2S_LENGTH register must be written last.

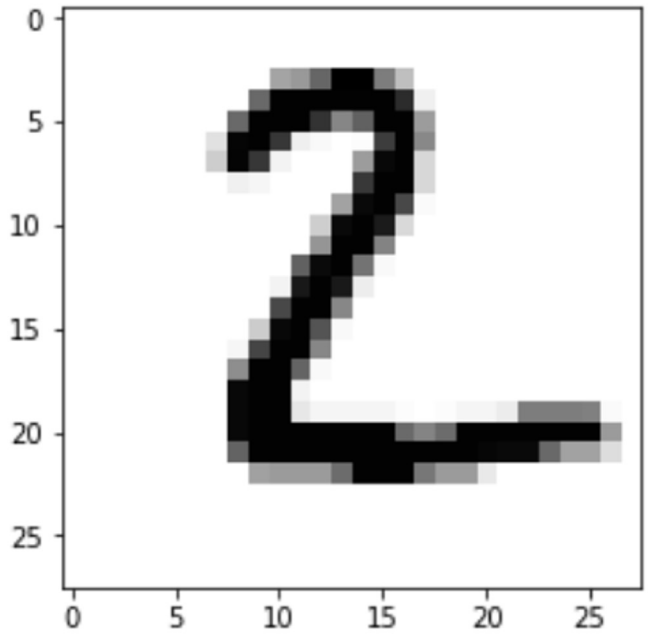
5. Wait until MM2S_DMASR.Idle==1 for completion

P8+ Accelerate Machine Learning

- Goal: Accelerate reference neural network
- Harder, more open-ended projects

Simple Neural Network

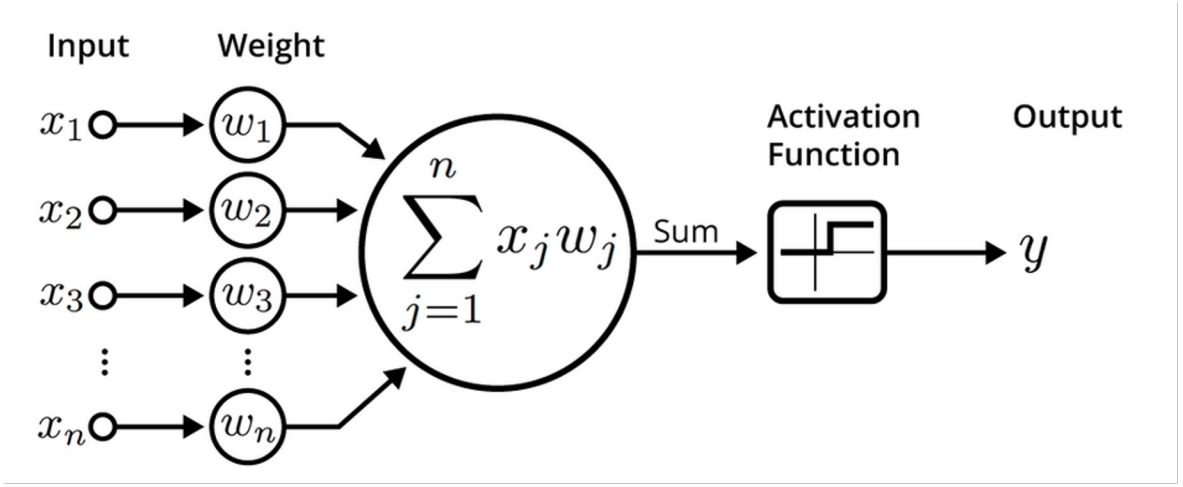
```
==============================
Index: 0
Image:
```




```
ML Classification Result: 2
Real Value: 2
Correct Result: True
==============================
```

- Takes in image of number
- Returns integer value
- How? artificial neural network

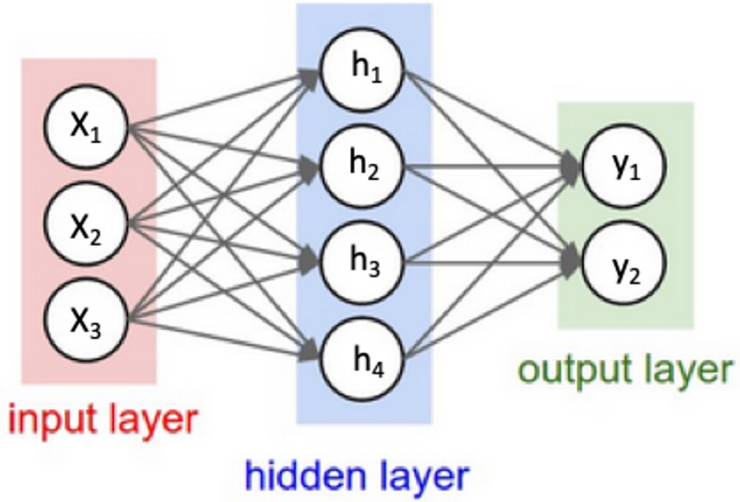
Python Neuron

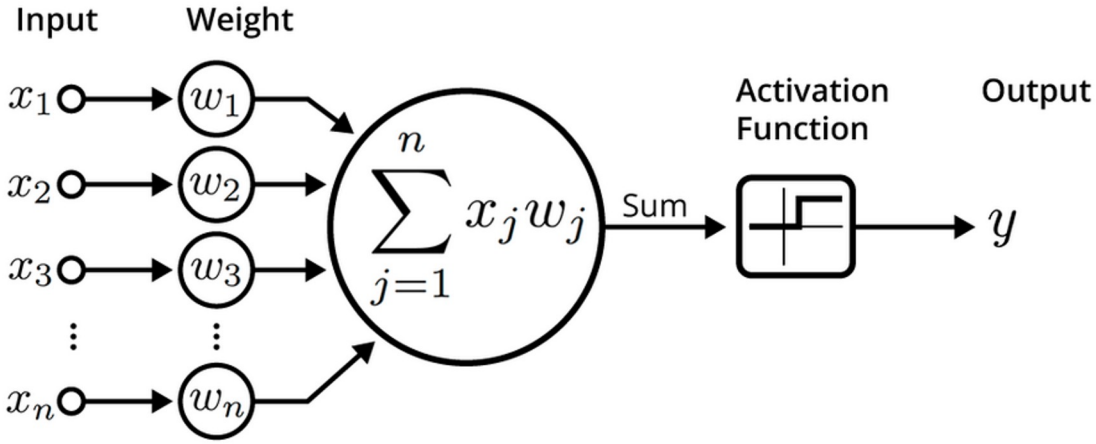



```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Activation functions.

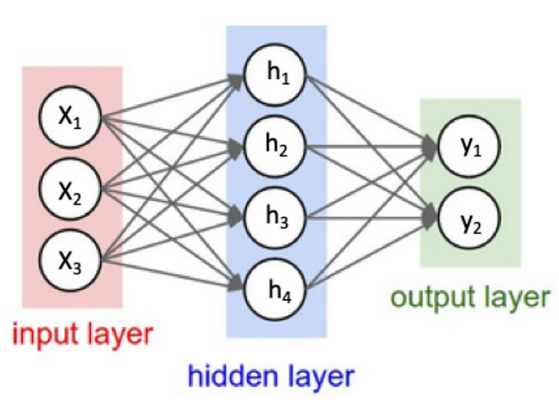
Python Neuron






```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Why focus on Dot Product?

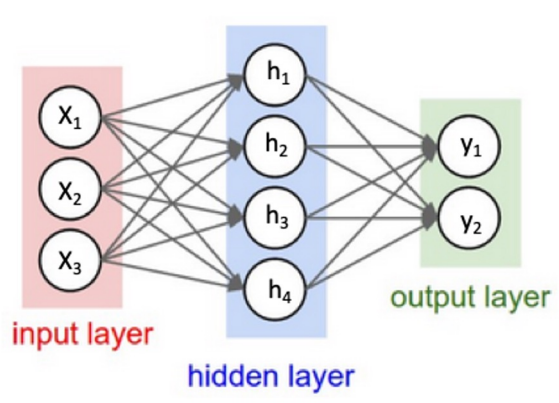



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Matrix Multiplication (Dot Product)

$$\begin{pmatrix} X_0 & X_1 \end{pmatrix} \times \begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix}$$

$$\begin{pmatrix} (X_0 * Y_0) + (X_1 * Y_1) \end{pmatrix}$$



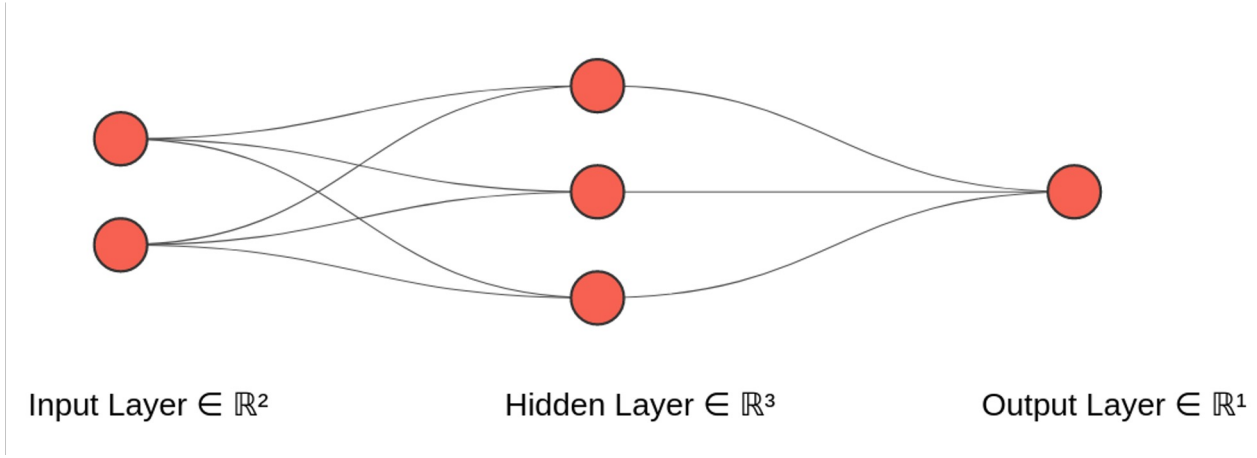
General Rules:

Result of Dot Product is a Scalar value!

Cannot take the dot product of two vectors of different lengths!

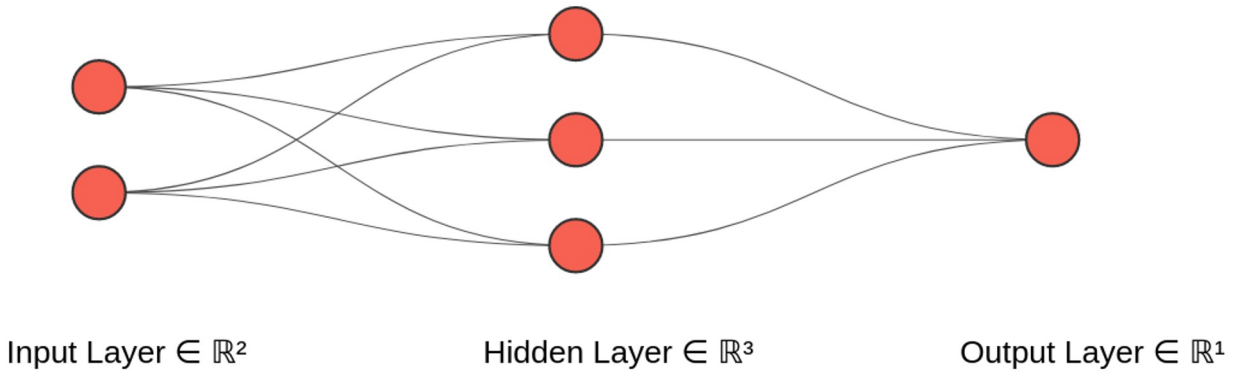
Matrix Multiplication (Dot Product)

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \times \begin{pmatrix} 4 & 1 & 5 \\ 3 & 6 & 2 \end{pmatrix}$$



Matrix Multiplication (Dot Product)

$$\begin{pmatrix} 2 & 3 \end{pmatrix} \times \begin{pmatrix} 4 & 1 & 5 \\ 3 & 6 & 2 \end{pmatrix}$$

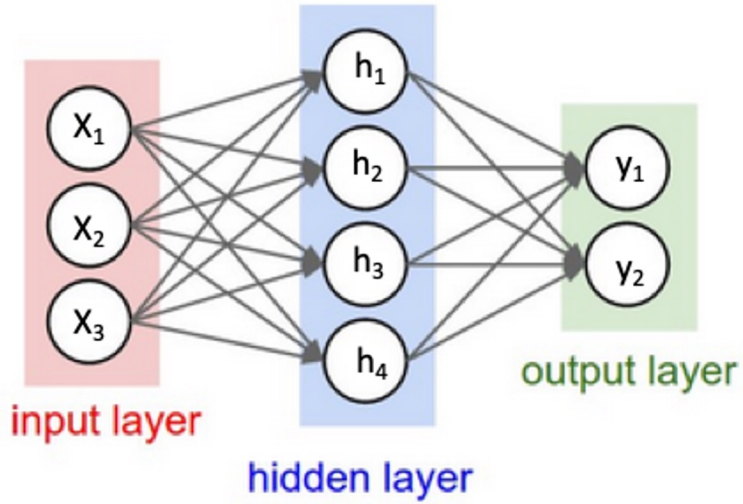


$$\begin{pmatrix} (2 * 4) + (3 * 3), (2 * 1) + (3 * 6), (5 * 2) + (3 * 2) \end{pmatrix}$$

$$\begin{pmatrix} 17, 20, 16 \end{pmatrix}$$

Matrix Multiplication (Dot Product)

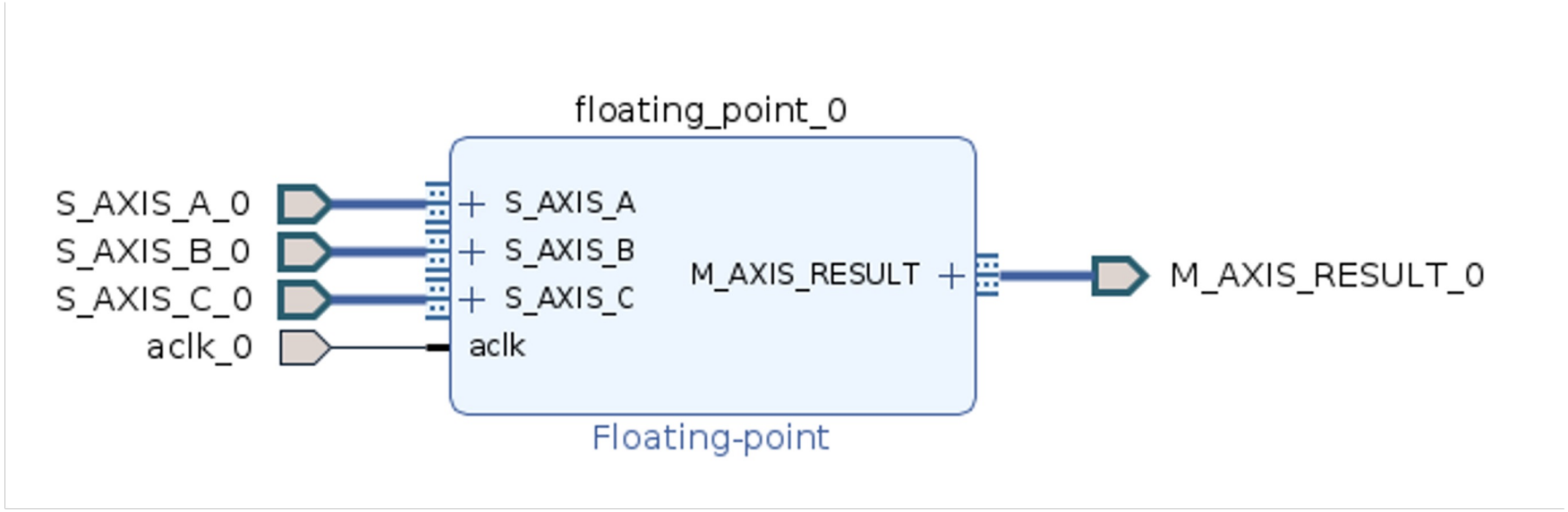
$$\begin{pmatrix} X_1 & X_2 & X_3 \end{pmatrix} \times \begin{pmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 \end{pmatrix}$$



Floating-Point Multiply-Accumulate (FMAC)

- Math: a * b + c

Floating-Point Multiply in Hardware



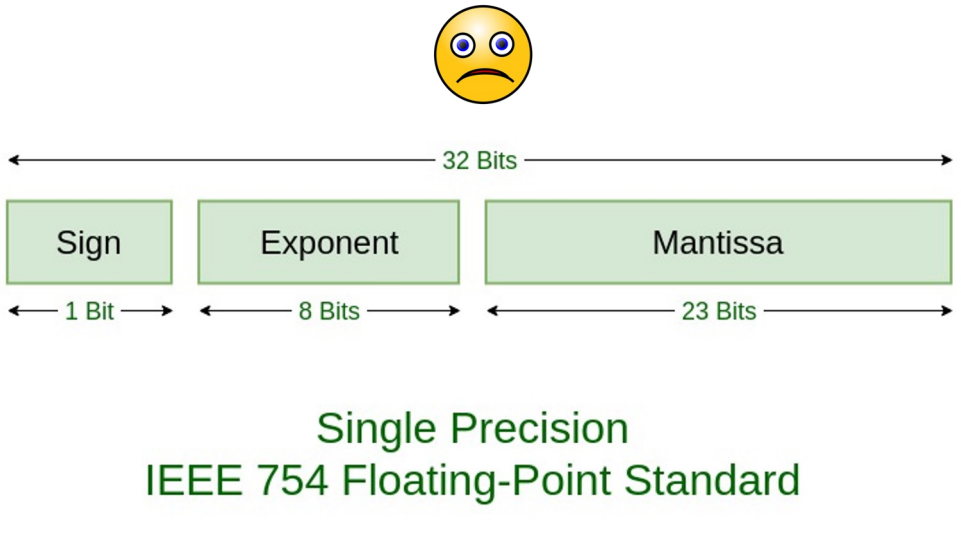
- result = a * b + c

Floating-Point math takes 8 cycles.

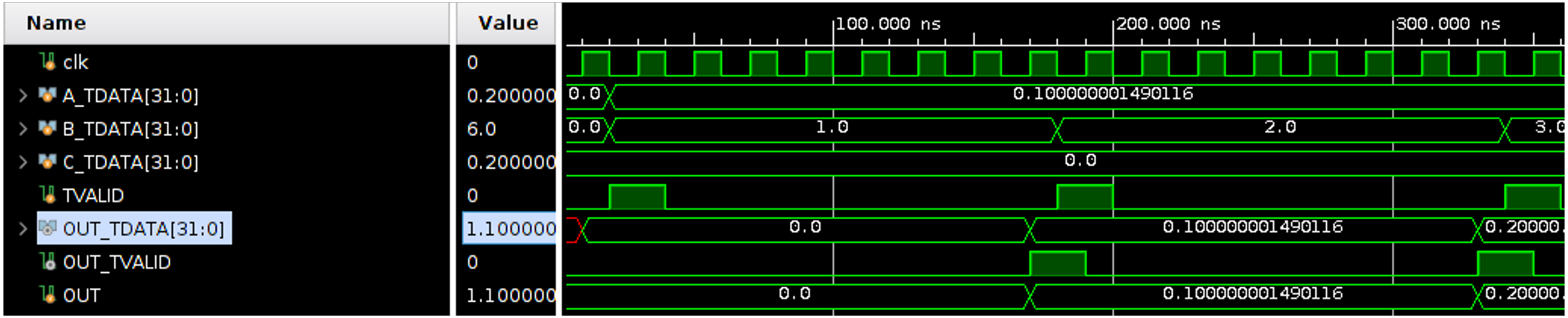
- Floating-Point is complicated.

[See also https://0.30000000000000004.com/] [15 zeroes]

- 8 cycles of complicated.



Demo Time


Name
Value
clk
A_TDATA[31:0]
B_TDATA[31:0]
C_TDATA[31:0]
TVALID
OUT_TDATA[31:0]
OUT_TVALID
OUT
0
0.200000
6.0
0.200000
0
1.100000
0
1.100000
100.000 ns
200.000 ns
300.000 ns
0.0
0.100000001490116
1.0
2.0
3.0
0.20000

Floating-Point math takes 8 cycles.

- Floating-Point is complicated.
- 8 cycles of complicated.
- How do we work around an 8 cycle latency?
- Pipelining!

How does Pipelining work in Math?

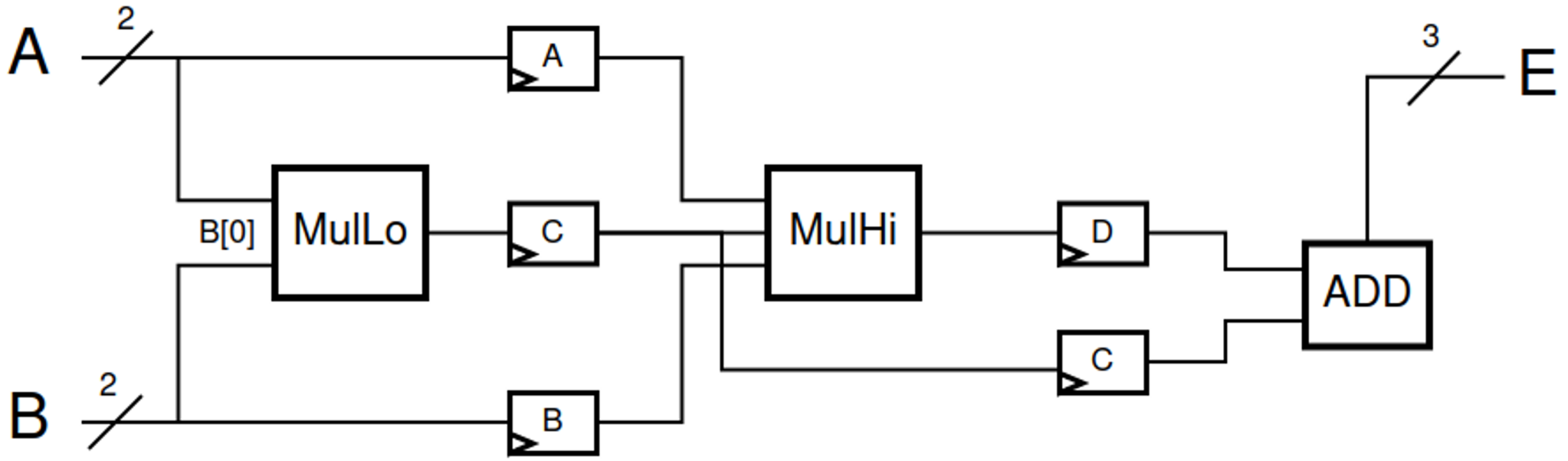
```
  2        10        10        10
x 3  =>  x 11  =>   x 1   +  x 10
  6       110        10   +   100 = 110
```

How does pipelining work in circuits?

```
  2        10        10        10
x 3  =>  x 11  =>   x 1    +  x 10
  6       110        10    +   100 = 110
```

Pipelining in hardware

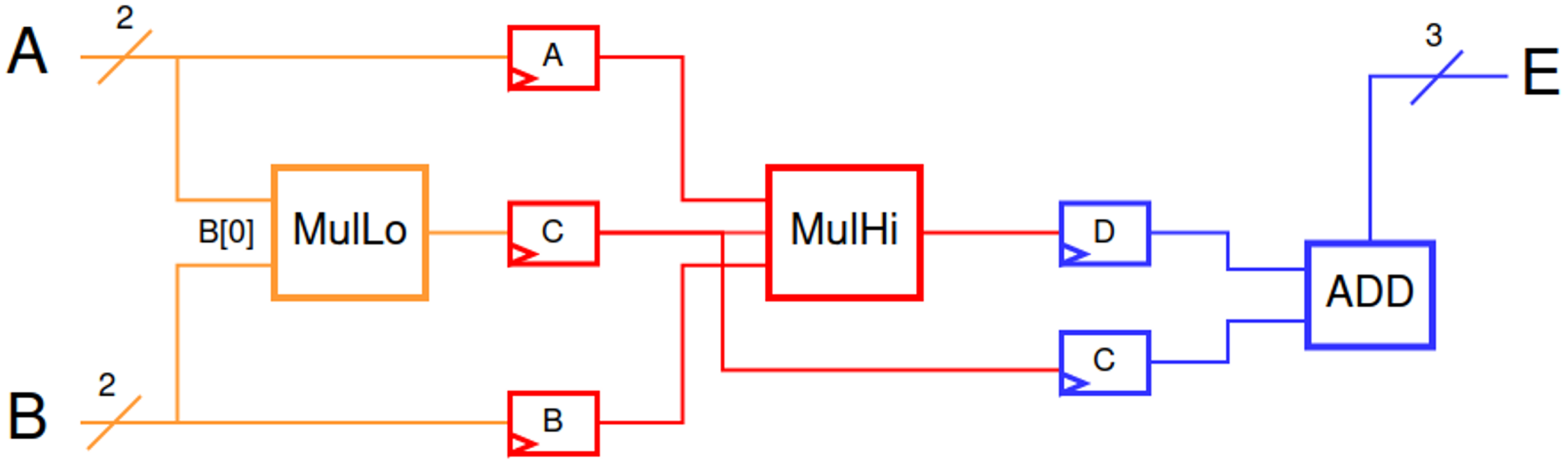
```
  2        10         10         10
x 3  =>  x 11   =>   x 1    +  x 10
  6        110        10    +   100  = 110
```



How many cycles does it take to execute this multiply?

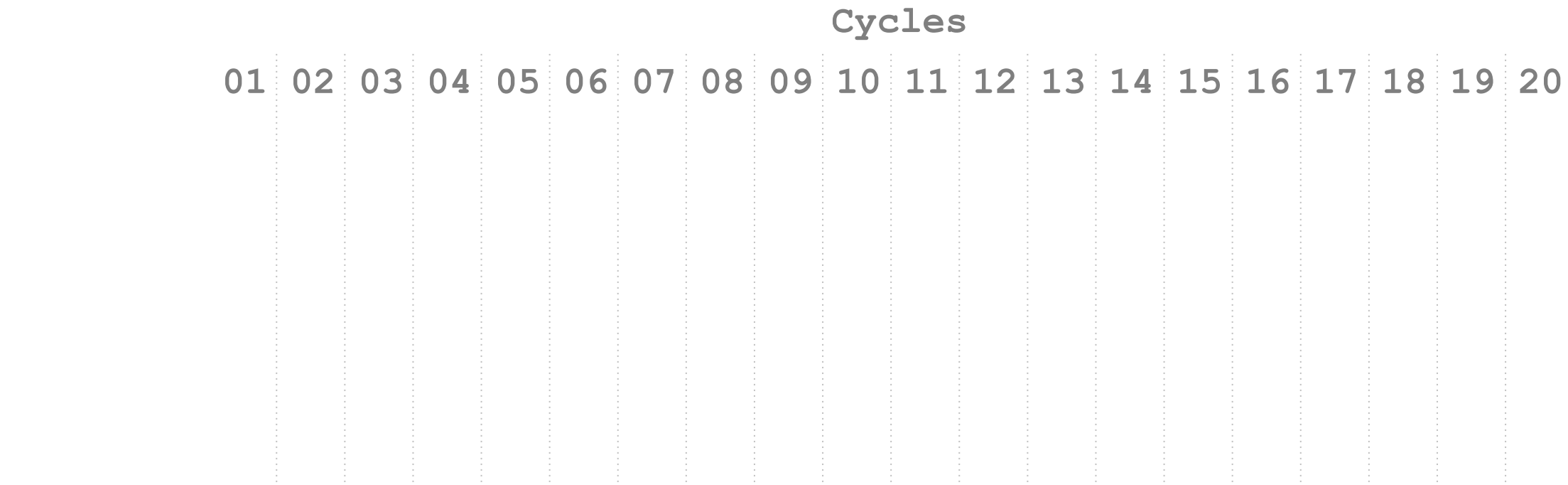
Pipelining in hardware

```
  2        10        10        10
x 3  =>  x 11  =>   x 1    +  x 10
  6        110       10     +  100  = 110
```

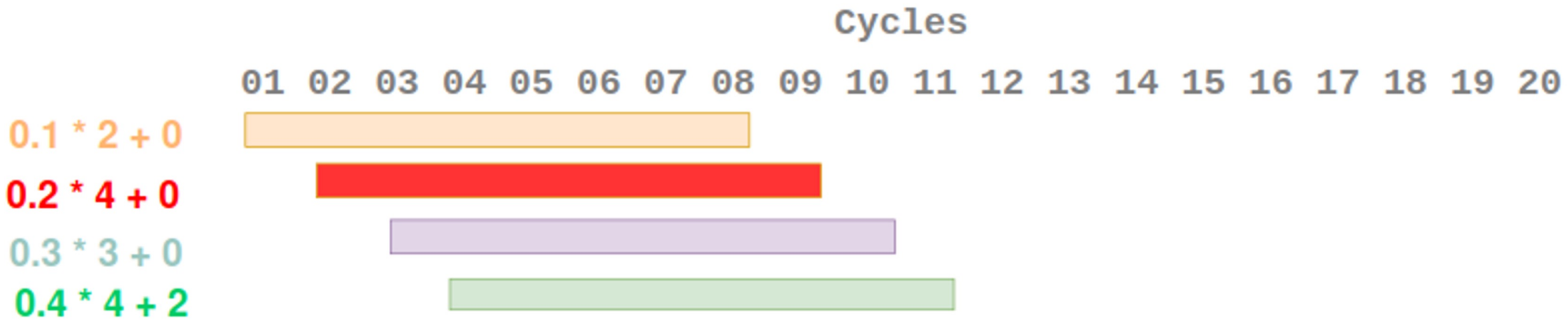


How many cycles does it take to execute this multiply?

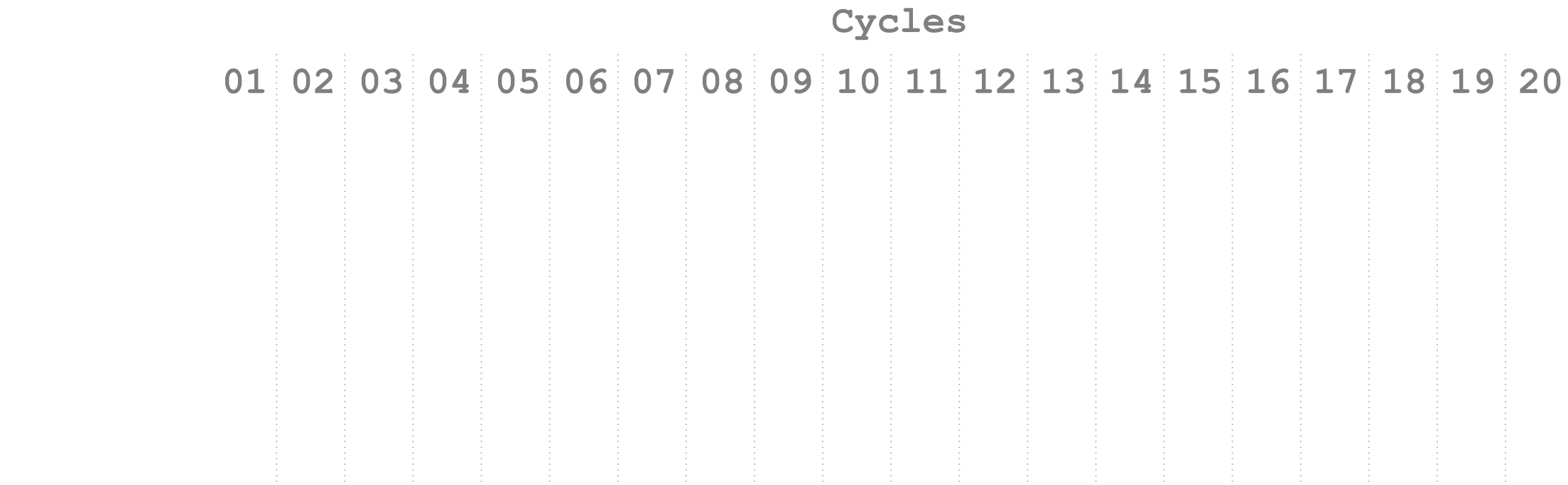
FMAC Pipelining



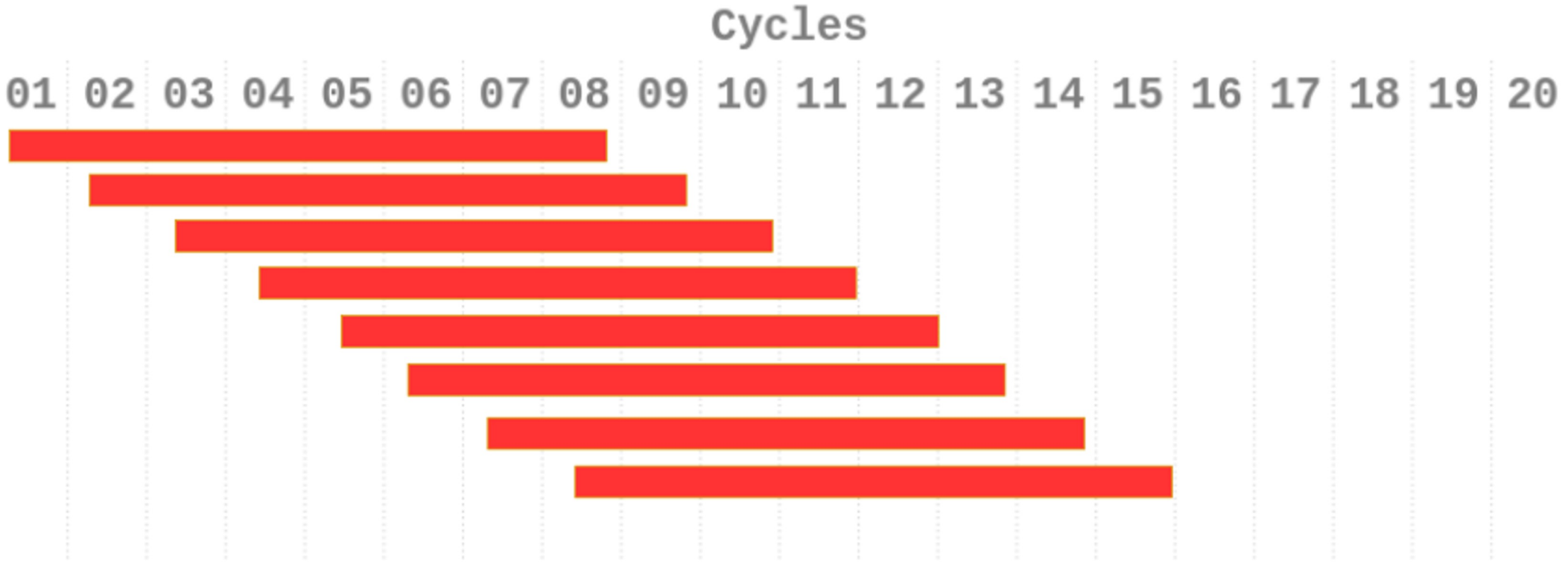
FMAC Pipelining



FMAC Pipelining



FMAC Pipelining



How long does one multiply take?

How long does 8 multiplies take?

Latency vs. Throughput

- **Latency**: How long does an individual operation take to complete?

- **Throughput**: How many operations can you complete per second (or per cycle)?

Latency vs. Throughput

- **Latency**: How long does an individual operation take to complete?

Latency is the ***time*** required to perform some action or to produce a unit of result.

- **Throughput**: How many operations can you complete per second (or per cycle)?

Throughput is the **number of actions** executed or units produced per unit of time.

Pipelining

- FMAC takes 8 cycles for 1 value
- But can accept a new value every cycle.

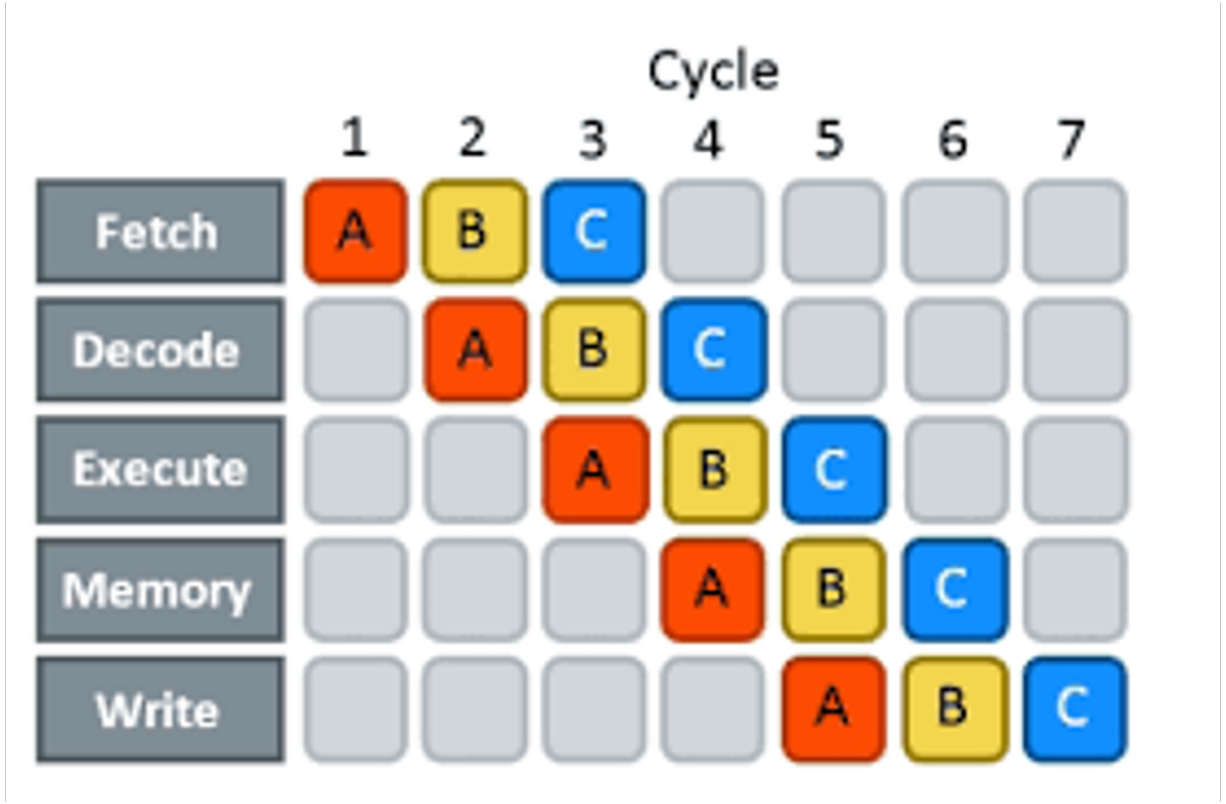
- What is Latency:
- What is Throughput:

Pipelining

- FMAC takes 8 cycles for 1 value
- But can accept a new value every cycle.

- What is Latency: 8 cycles / value
- What is Throughput: 1 value / cycle

Pipelining in the wild



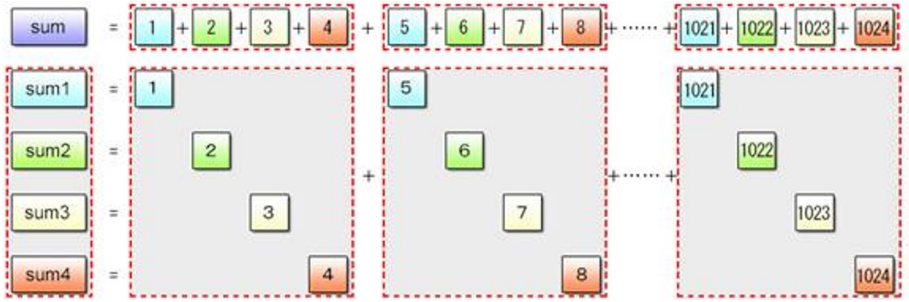
Pipelining in the wild

SIMD: Single Instruction Multiple Data

https://replit.com/@GrantSkipper/PipelineSIMD
make all && make run

Modern architectures implement a **Vector Processor (VPU)** which can be accessed through SIMD/Vector instructions (ISA extension).

Vector units are implemented with… pipelining!



Each stage of the pipeline operates on different elements of a vector.

Can greatly reduce control flow and expensive loops in code!***

Intel Intrinsics: https://db.in.tum.de/~finis/x86-intrin-cheatsheet-v2.1.pdf
Aarch64 Intrinsics: https://developer.arm.com/architectures/instruction-sets/intrinsics/#q=dot

Recall: Matrix Multiplication (Dot Product)

Alternative Dot Computations

Multiply-Accumulate Dot Computations

Python Time

```
weights = np.array( [[1,2,3,4],[5,6,7,8],[9,10,11,12]], dtype=np.float32)
inputs = np.array([[0.1,0.2,0.3]], dtype=np.float32)
outputs = np.dot(inputs, weights)
```

```
Input                  Weights                          Output
[0.1 0.2 0.3]    .     [1. 2. 3. 4.]          =  [3.8000002 4.4       5.        5.6000004]
                       [5. 6. 7. 8.]
                       [ 9. 10. 11. 12.]
```

```
Input                    Weights                          Output
[[0.1 0.2 0.3]]    .     [1. 2. 3. 4.]          =  [3.8000002 4.4       5.        5.6000004]
                         [5. 6. 7. 8.]
                         [ 9. 10. 11. 12.]
```

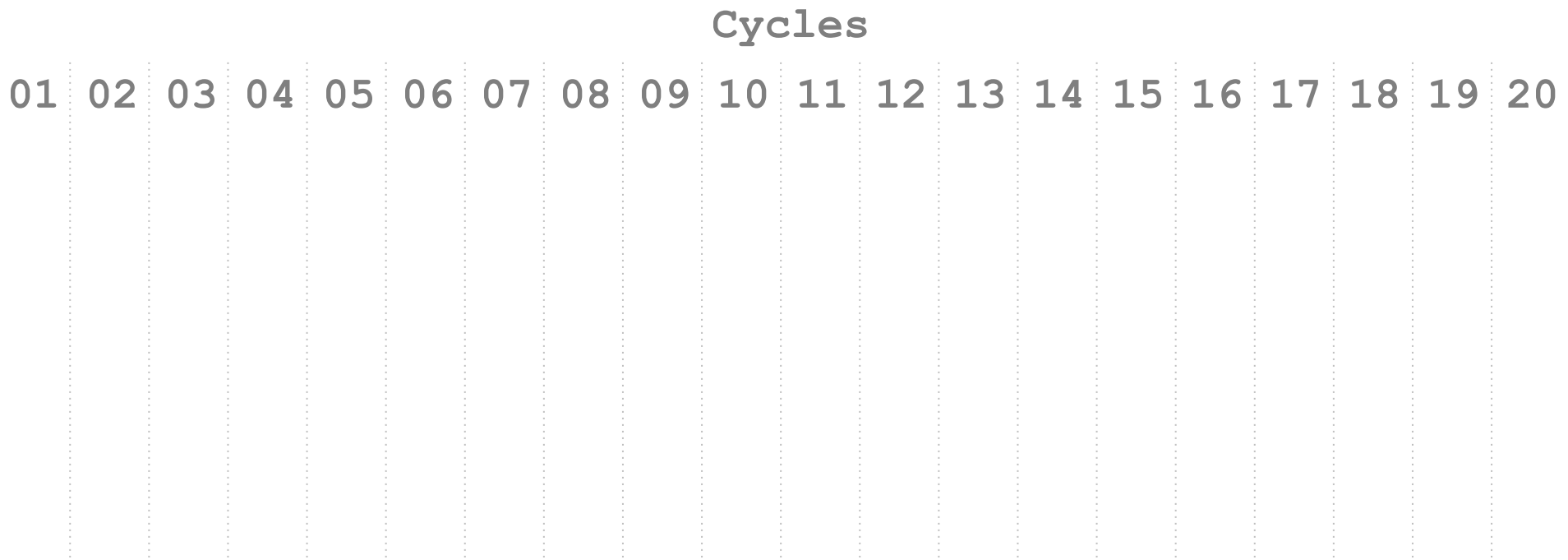
Mult-Accum Dot

```
Inputs (Shape):
 (1, 3)
Output (Shape):
 (1, 4)
Weights (Shape):
 (3, 4)
```

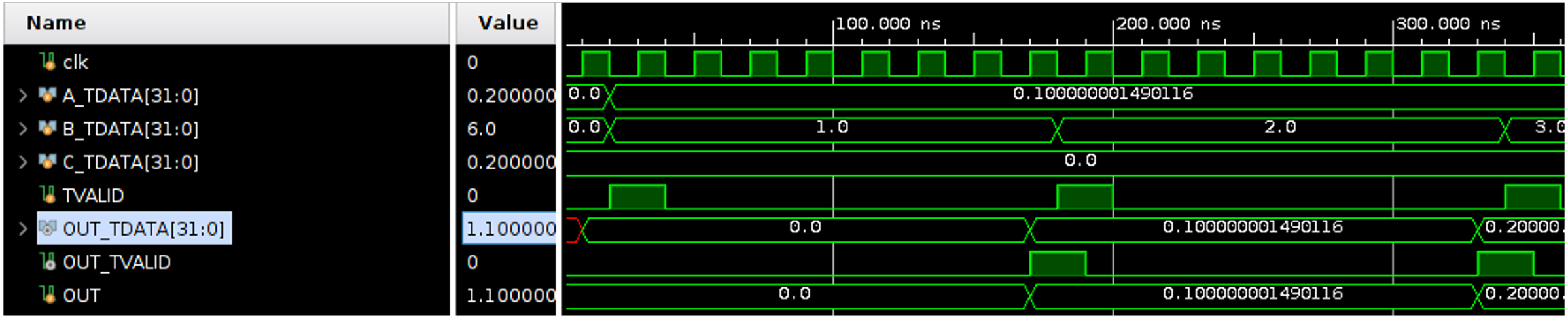
```python
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

```
Input                    Weights                          Output
[[0.1 0.2 0.3]]    .     [1. 2. 3. 4.]          =  [3.8000002 4.4       5.        5.6000004]
                         [5. 6. 7. 8.]
                         [ 9. 10. 11. 12.]
```

Dependencies



Demo Time


Name
Value
clk
A_TDATA[31:0]
B_TDATA[31:0]
C_TDATA[31:0]
TVALID
OUT_TDATA[31:0]
OUT_TVALID
OUT
0
0.200000
6.0
0.200000
0
1.100000
0
1.100000
100.000 ns
200.000 ns
300.000 ns
0.0
0.100000001490116
1.0
2.0
3.0
0.0
0.0
0.100000001490116
0.0
0.100000001490116

Next Time: More on Dependencies

Latency on Pipelined FMAC

- Solution: Stall at the end of a row.
- Drain the pipeline.

Hardware Parallelism

- CPU: 1 Floating-Point Unit
- FPGA?

Finding Parallelism

- Some some computation that doesn’t depend on other computation’s results
- Shared Inputs are OK.

Next Time: Can we use 2+ FMACs?

Parallize Alternative Dot Computations?

Can we parallelize Dot?

```
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

Can we parallelize Dot?

```
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

```
def par_pydot(inputs, weights):
    par_inputs = [inputs[:,::2], inputs[:,1::2]]
    par_weights = [weights[::2,:], weights[1::2,:]]

    par_outputs = [pydot(par_inputs[0], par_weights[0]),
                   pydot(par_inputs[1], par_weights[1])]

    outputs = par_outputs[0] + par_outputs[1]
    return outputs
```

Can we parallelize Dot?

```
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

```
def par_pydot(inputs, weights):
    par_inputs = [inputs[:,::2], inputs[:,1::2]]
    par_weights = [weights[::2,:], weights[1::2,:]]

    par_outputs = [pydot(par_inputs[0], par_weights[0]),
                   pydot(par_inputs[1], par_weights[1])]

    outputs = par_outputs[0] + par_outputs[1]
    return outputs
```

19: Hardware Acceleration III

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University