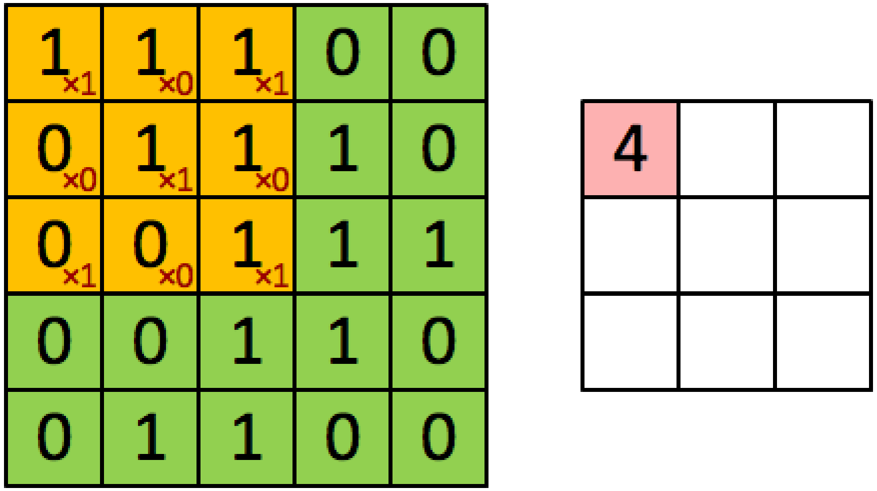
COE 1195: Advanced Digital Design

Lab 5 – 2D Image Convolution

Dr. Amr Mahmoud

Exercise:
2D Convolutional Accelerator using Vivado HLS

2D Convolution

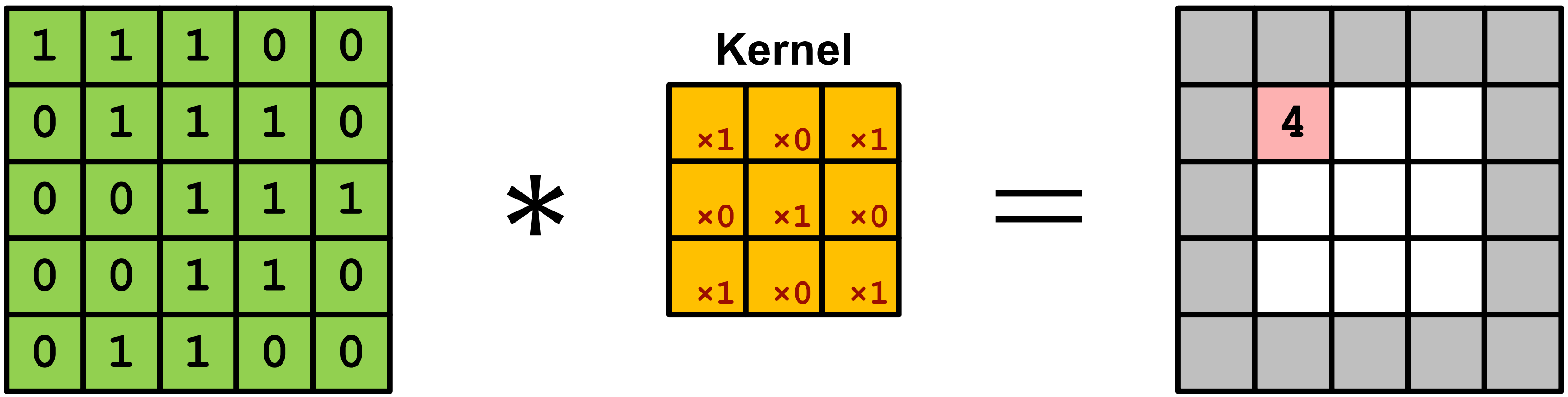


1 ×1	1 ×0	1 ×1	0	0
0 ×0	1 ×1	1 ×0	1	0
0 ×1	0 ×0	1 ×1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature



Input Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

*

Kernel

×1	×0	×1
×0	×1	×0
×1	×0	×1

=

Output Image

	4			

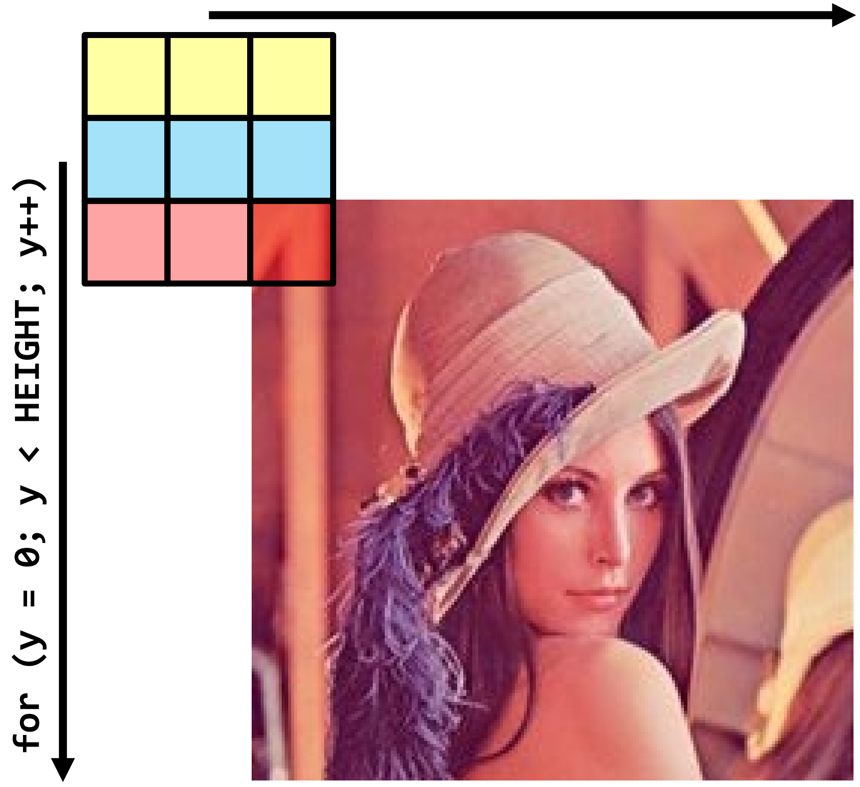
Access Pattern

- Software Approach
  - Two nested loops
  - Less efficient in hardware
  - Pixels are *read* more than once

- Stream Approach
  - One single loop
  - Efficient in hardware
  - Pixels are *read* only once

Software Approach

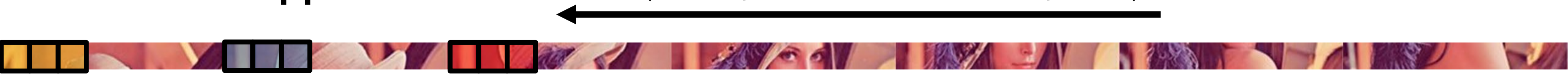
```
for (x = 0; x < WIDTH; x++)
```



```
for (y = 0; y < HEIGHT; y++)
```

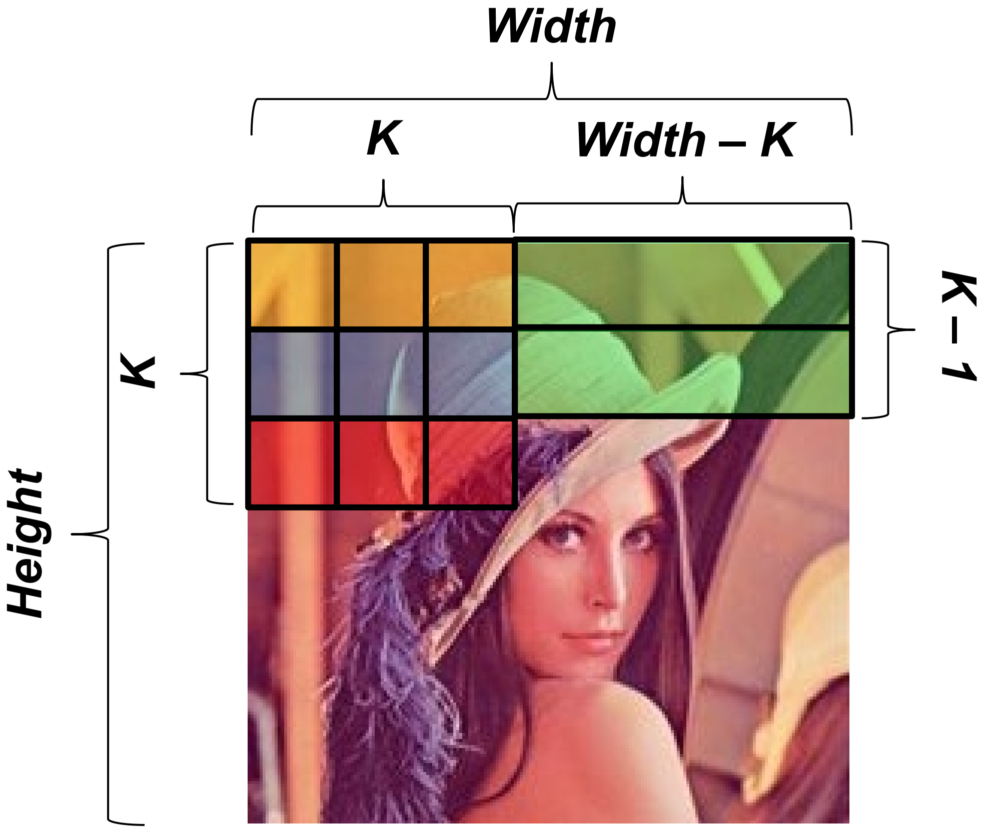
Stream Approach

```
for (i = 0; i < WIDTH*HEIGHT; i++)
```

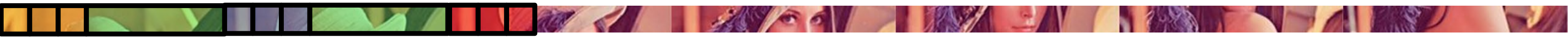


Memory Path (1/3)

- Kernel
  - Need ***K*** **×** ***K*** registers
  - Hold values to compute kernel to produce output pixel
- Line Buffer
  - Need ***K* – *1*** line buffers of length ***Width* – *K***
  - Temporarily store image line to access previous pixels without re-accessing from stream source

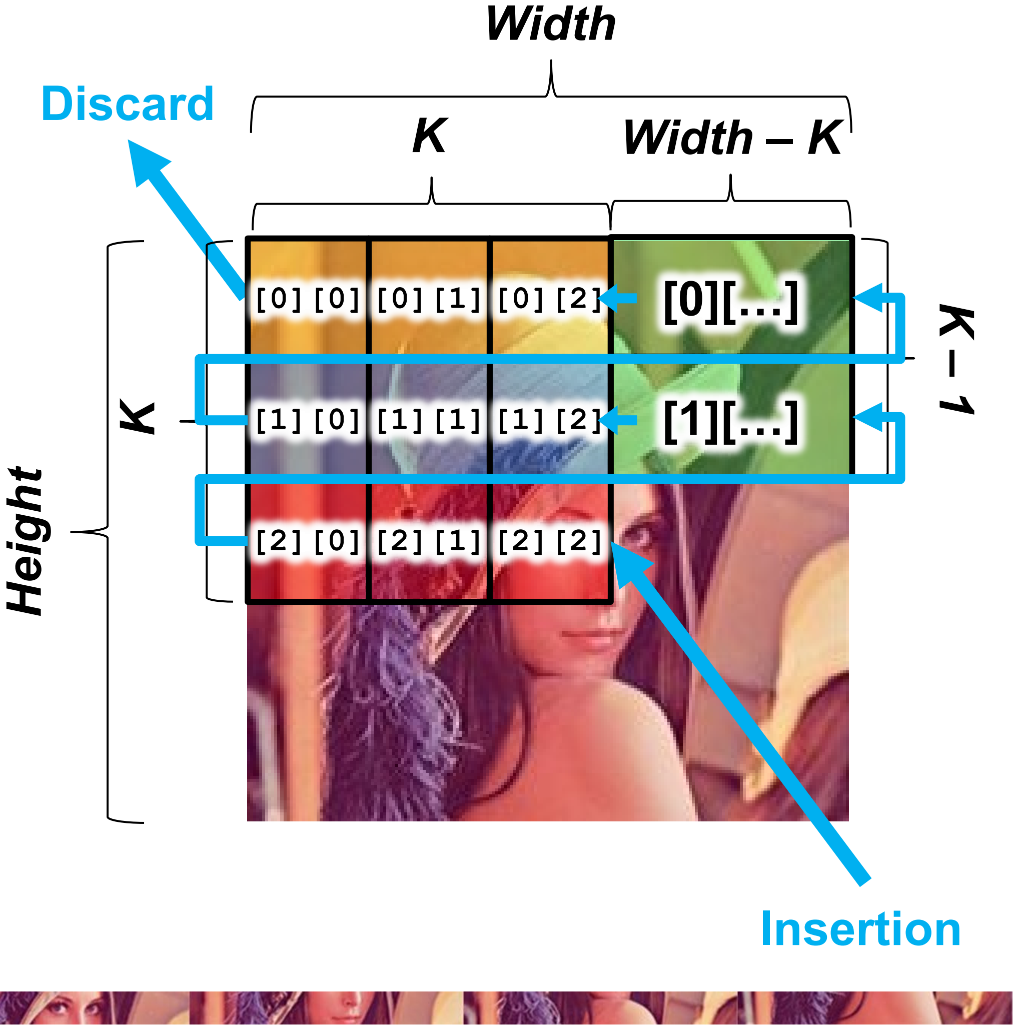


Stream Approach



Memory Path (2/3)

- Kernel
  - Need ***K* × *K*** registers
  - Hold values to compute kernel to produce output pixel
- Line Buffer
  - Need ***K* – 1** line buffers of length ***Width – K***
  - Temporarily store image line to access previous pixels without re-accessing from stream source



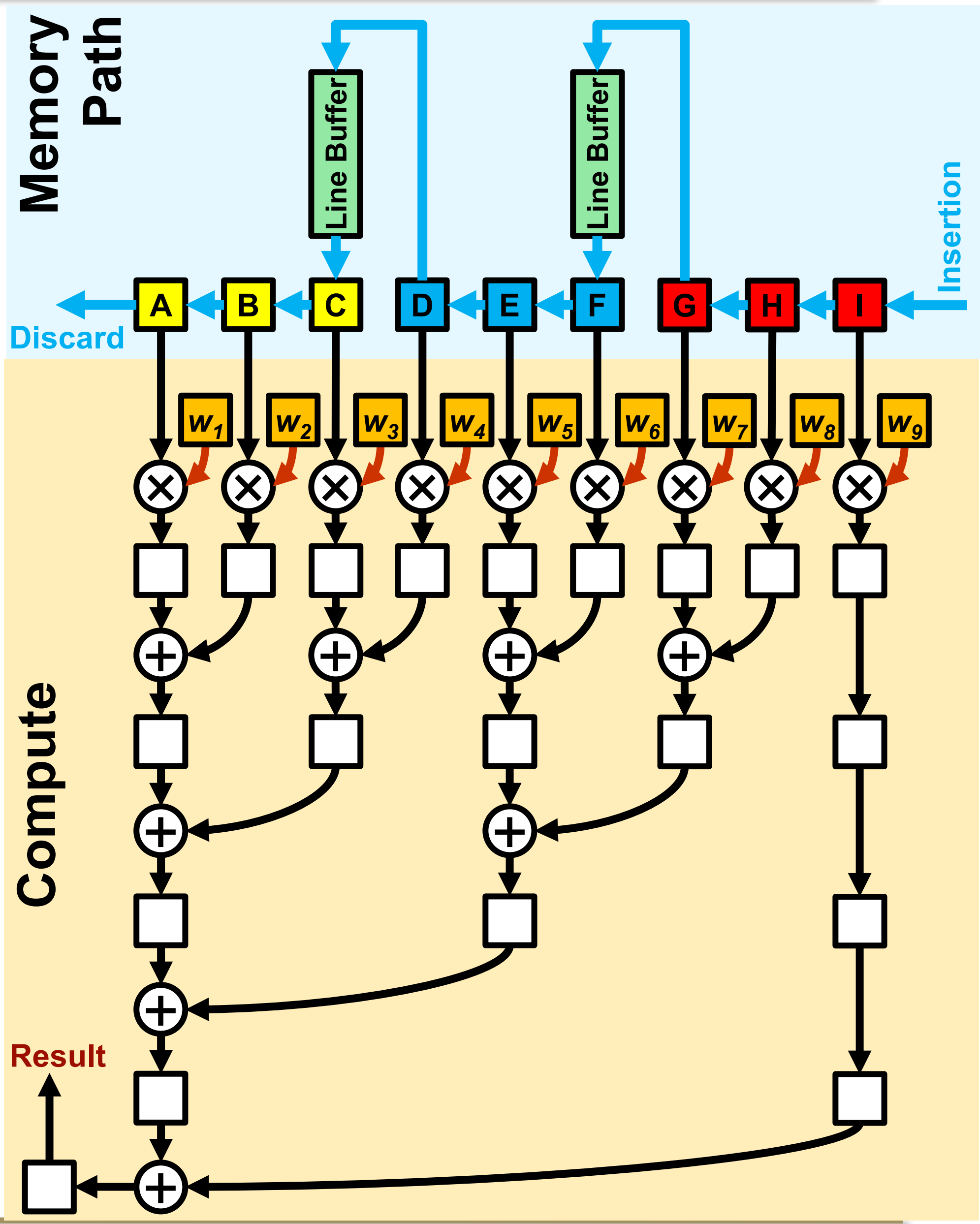
Stream Approach

Data Path

- Multiplications
  - Unrolled
    - *K* × *K* multiplications per iteration
- Additions
  - Implicit reduction
    - Pipelined binary tree

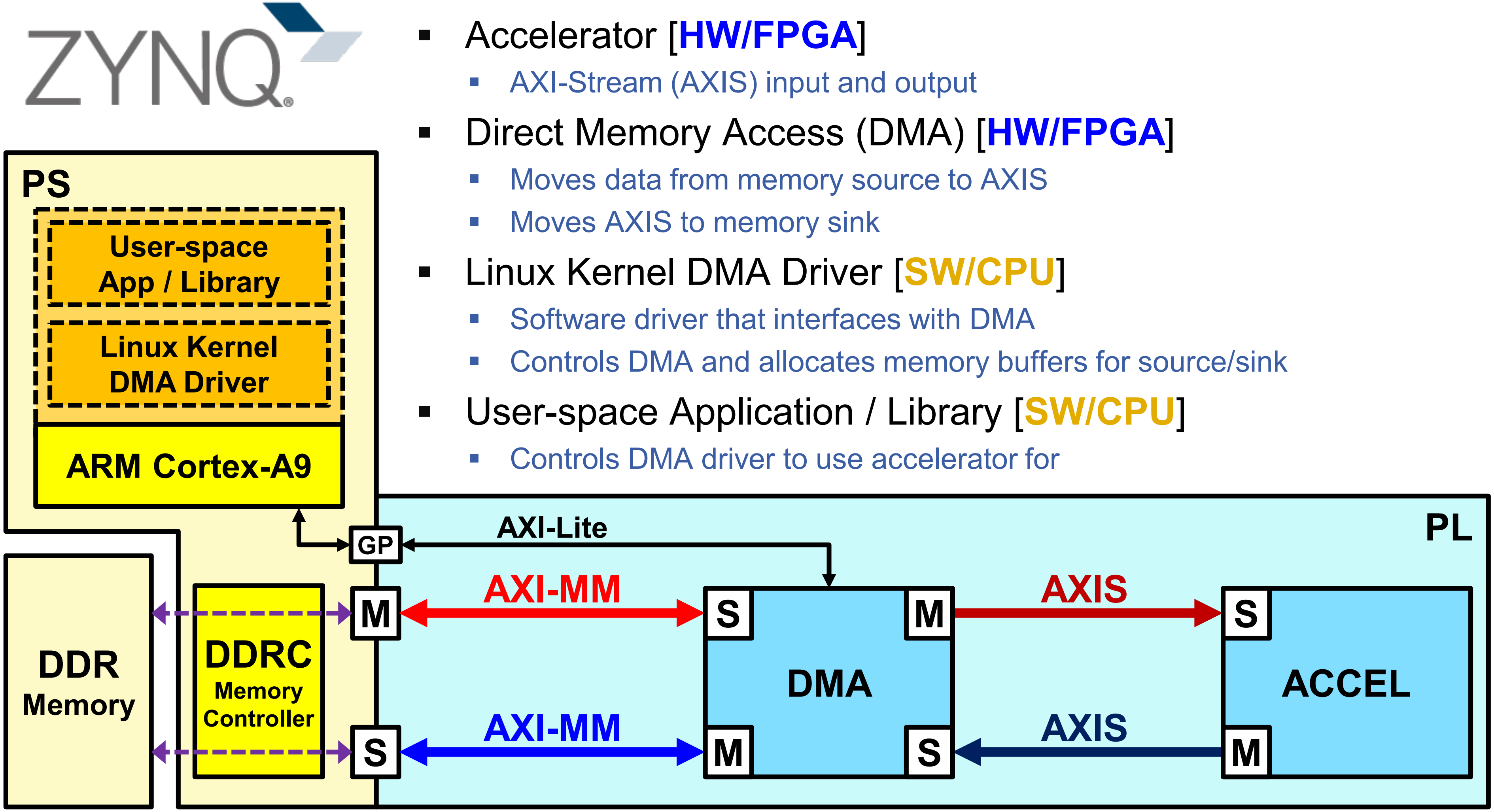
A	B	C
D	E	F
G	H	I

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9



Acceleration Framework

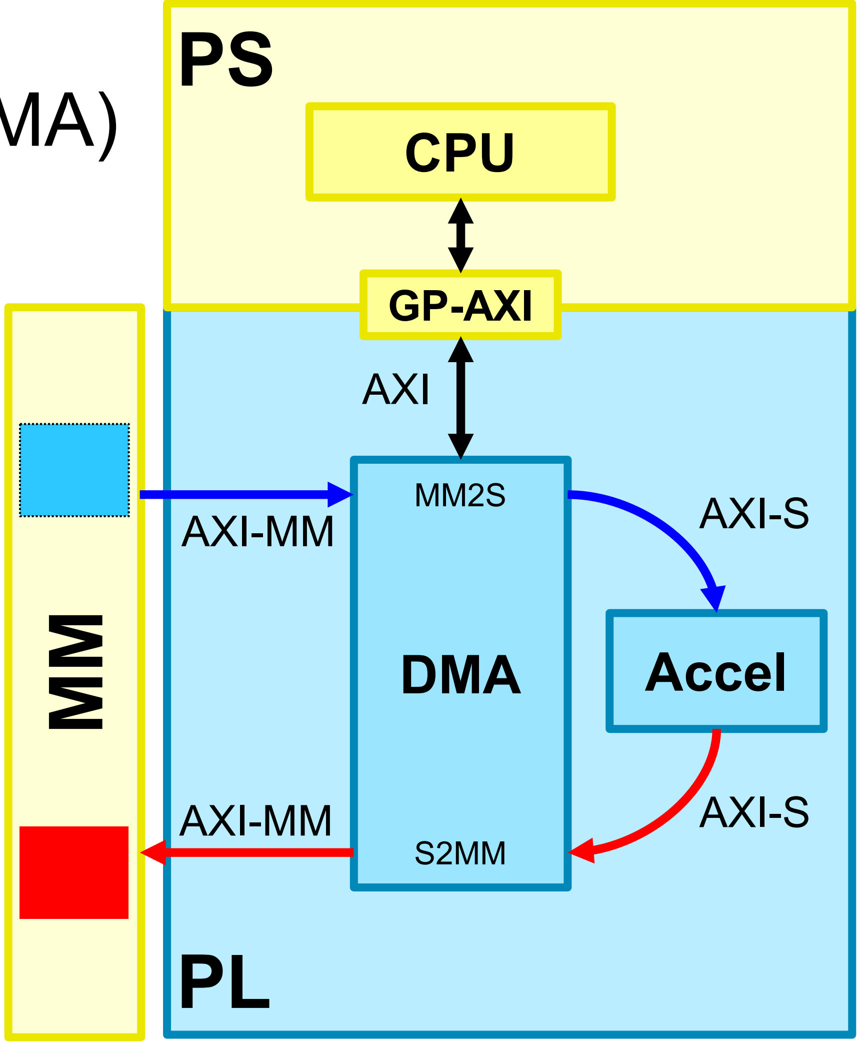
- Accelerator [**HW/FPGA**]
  - AXI-Stream (AXIS) input and output
- Direct Memory Access (DMA) [**HW/FPGA**]
  - Moves data from memory source to AXIS
  - Moves AXIS to memory sink
- Linux Kernel DMA Driver [**SW/CPU**]
  - Software driver that interfaces with DMA
  - Controls DMA and allocates memory buffers for source/sink
- User-space Application / Library [**SW/CPU**]
  - Controls DMA driver to use accelerator for



DMA Architectures

DMA Architectures

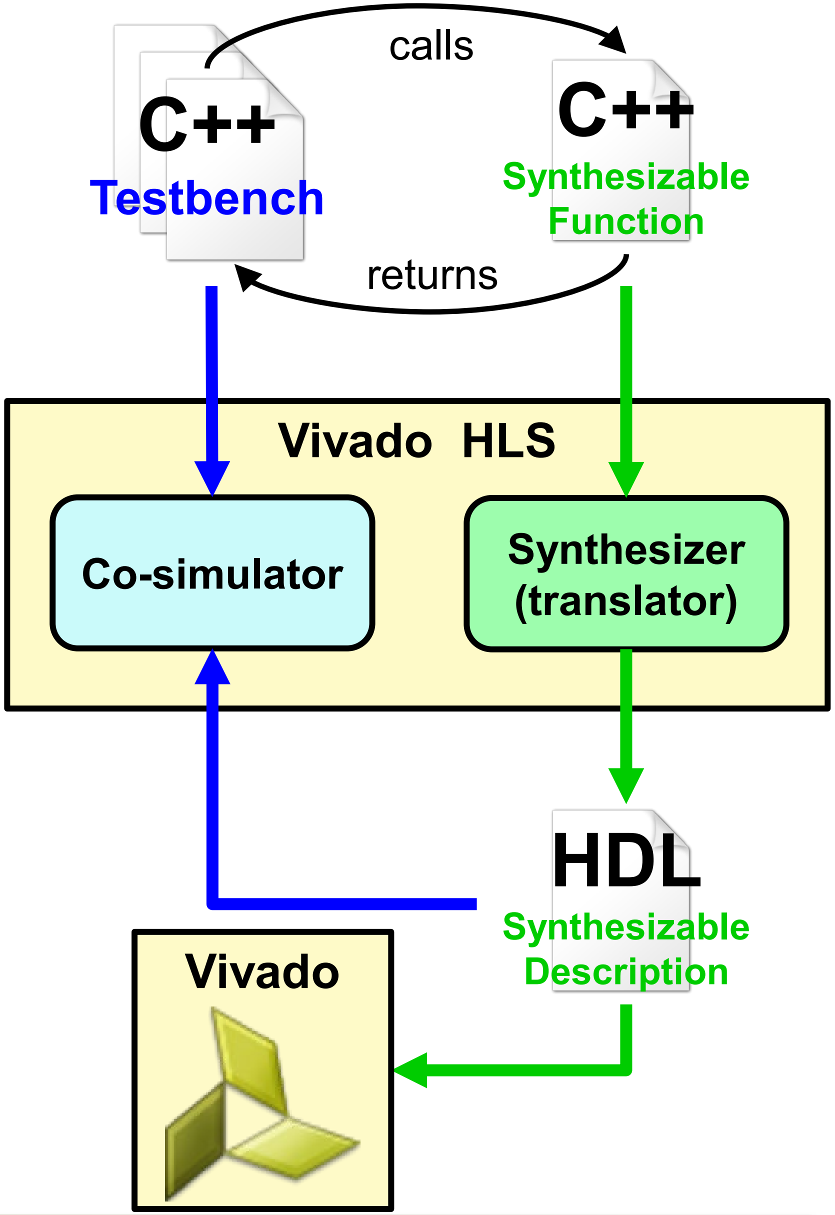
- Direct Memory Access (DMA)
  - "Data Mover" Behavior
    - Asynchronously move data from MM/stream to stream/MM
  - Register Space
    - Control/Status (for controller)
    - MM2S/S2MM addresses
    - MM2S/S2MM lengths
  - Operations
    - Stream data in MM starting at *MM2S address* of *MM2S length*
    - Stream data of *S2MM length* to MM starting at *S2MM address*



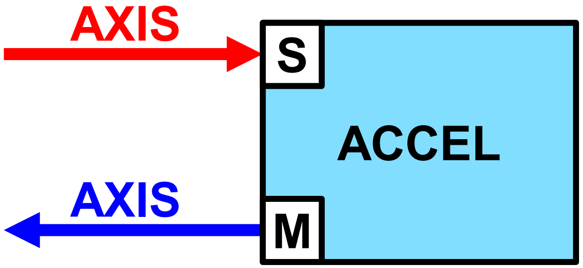
Vivado HLS

Vivado HLS

- HLS translator
  - Input: **synthesizable function**
    - C++ function
    - HLS directives and rules
  - Output: HDL
- Co-simulator
  - **Testbench** (C++) that calls synthesizable function
  - Executes testbench and simulates translated synthesizable function on PC



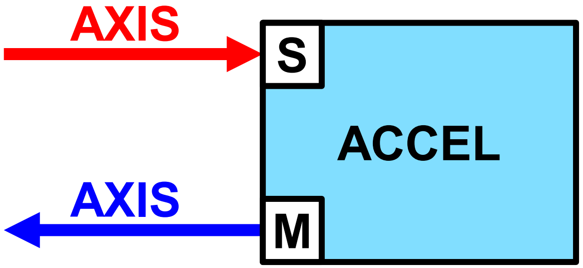
INTERFACE (1/2)



- AXI-Stream Interfaces
  - Hint to translator that arguments are AXIS interfaces

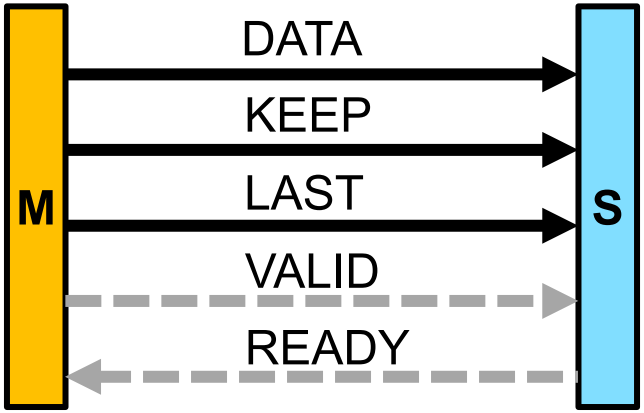
```
void hw_conv(stream_t &sin, stream_t &sout) {
#   pragma HLS INTERFACE ap_ctrl_none port=return
#   pragma HLS INTERFACE axis port=sin
#   pragma HLS INTERFACE axis port=sout
    ...
}
```

INTERFACE (2/2)



- AXI-Stream Beats
  - Contain **Data**, **Keep**, **Last**, **Valid**, **Ready**
  - Read (>>) and write (<<)
  - **Data** (value), **Keep** (byte strobe), **Last** (last beat) are explicit
  - **Valid** and **Ready** are implicit on >> and <<

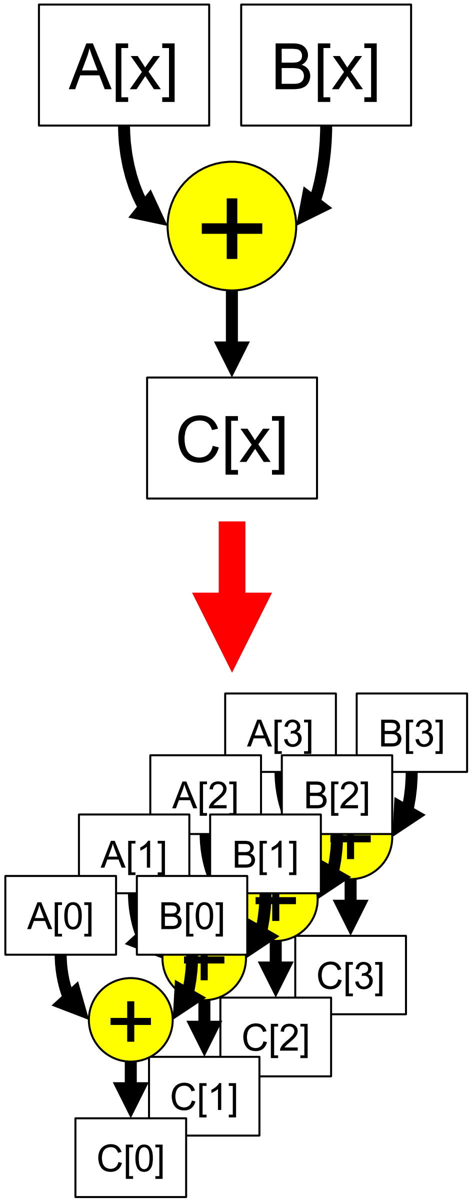
```
... // read beat from sin
beat_t beat;
sin >> beat;
value = beat.data(7, 0);
... // write beat to sout
beat_t beat;
beat.data(7, 0) = value;
sout << beat;
...
```



UNROLL

- Parallelize Loop Iterations
  - Without unrolling
    - Iterations occur one at a time (*N* cycles each)
    - Trade-off: less area, less bandwidth
  - With unrolling
    - Iterations occur in parallel
    - Trade-off: more area, more bandwidth
  - `factor` ≜ unrolling factor (# of iterations)

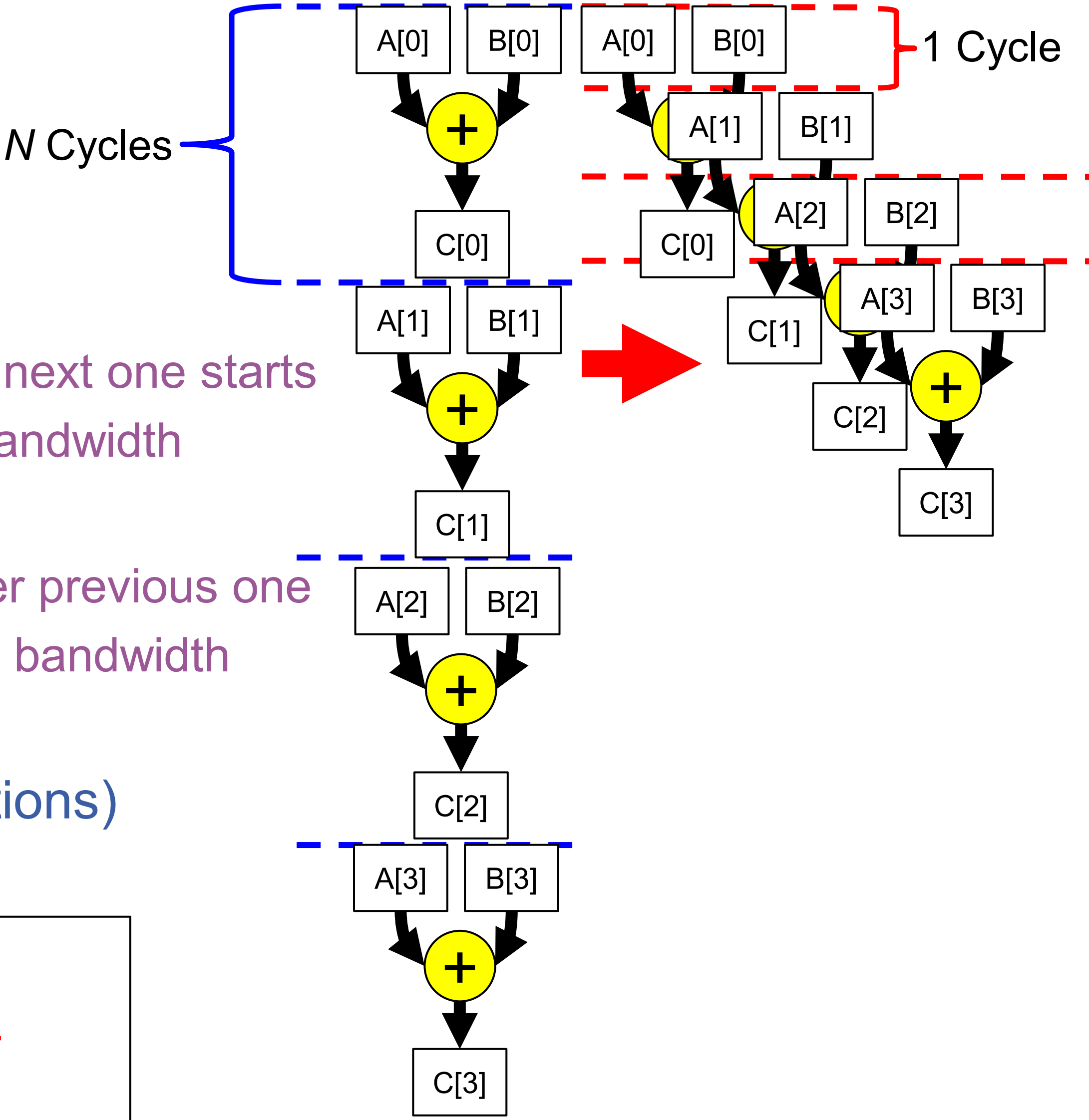
```
for (int x; x < 3; x++) {
#  pragma HLS UNROLL factor=4
   C[x] = A[x] + B[X]
}
```



PIPELINE

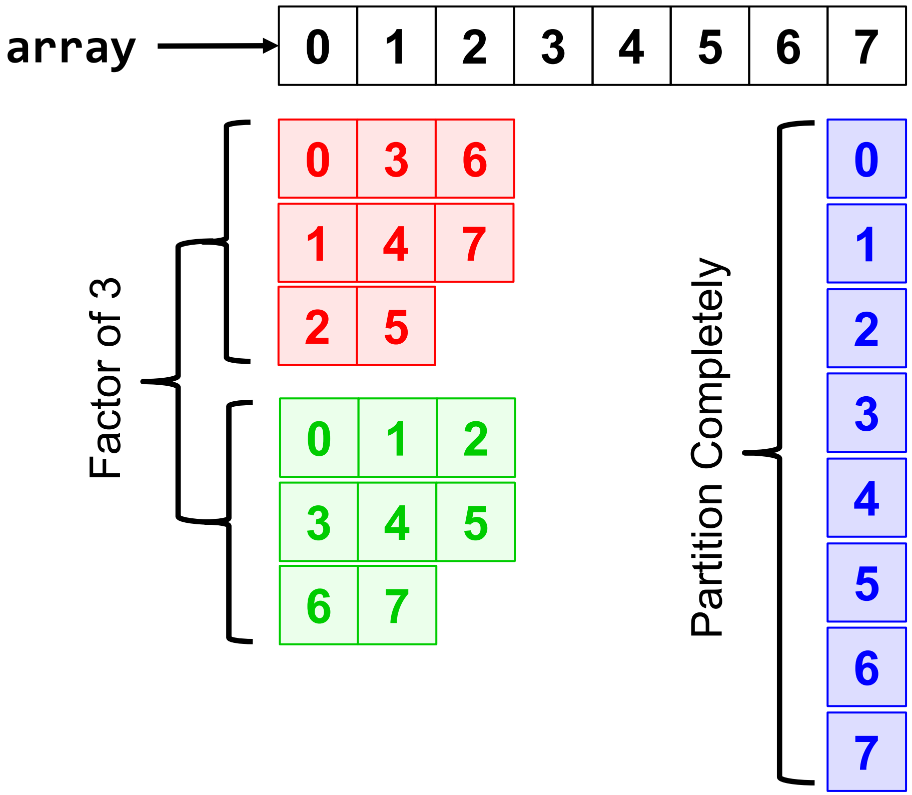
- Pipeline Loop Iterations
  - Without pipelining
    - Iteration must finish before next one starts
    - Trade-off: less area, less bandwidth
  - With pipelining
    - Iteration starts *II* cycles after previous one
    - Trade-off: more area, more bandwidth
  - `II` ≜ initiation interval
    (# of cycles between iterations)

```
for (int x; x < 3; x++) {
#   pragma HLS PIPELINE II=1
    C[x] = A[x] + B[X]
}
```



ARRAY_PARTITION

- Partition Array into smaller arrays
  or individual elements
  - Cyclic (partition into sub-arrays with cyclic assignment)
  - Block (partition into sub-arrays)
  - Complete (partition into individual elements)
- Factor
- Dimensionality



```
char array[8];
#pragma HLS ARRAY_PARTITION variable=array cyclic factor=3 dim=0
#pragma HLS ARRAY_PARTITION variable=array block factor=3 dim=0
#pragma HLS ARRAY_PARTITION variable=array complete dim=0
```