About me

謝懷頡

HUAI-CHIEH HSIEH

Interesting domain

✓ Digital Circuit design
✓ Deep learning
✓ Programming

Education

National Taiwan University of Science and Technology
Electrical and Computer Engineering (ECE) — **2019/9 ~ 2023/1**

National Cheng Kung University
Program on Integrated Circuit Design (PICD) — **2023/1 ~ now**

National Taiwan University
Graduate Institute of Electronics Engineering (GIEE) — **2024/9 ~**

Skill

Software

- C/C++
- MATLAB
- Python

Hardware

- System Verilog
- Verilog

EDA Tools

- Simulation & Debug: VCS + Verdi/nWave
- Synthesis: Design Compiler
- APR: IC Compiler I 、II
- Power analysis: PrimeTime
- Linting tool: superlint
- Verification: Cadence JasperGold VIP
- CDC checking: Spyglass
- FPGA: Quartus/Vivado/Vitis

Contest & Award

- 16th HOLTEK MCU Innovation Competition | 2nd

- Graduation Project Competition | Excellent Award

Contest & Award

- Academic Excellence Award | 4 times

- 2024 IC Contest (**Graduate-level Cell-based**) | Class A

-> advance to the final

教育部智慧聯網技術與應用人才培育計劃
大學校院積體電路設計競賽證明
CERTIFICATE OF INTEGRATED CIRCUIT DESIGN CONTEST

茲證明 國立成功大學 (B00152)

謝懷頡 劉永勝

參加教育部113年度大學校院積體電路(IC)設計競賽
研究所標準元件數位電路設計組，成績達決賽錄取標準

總聯盟召集人

智慧晶片系統
整合推動聯盟

競賽主持人

Certificate

- Design Compiler

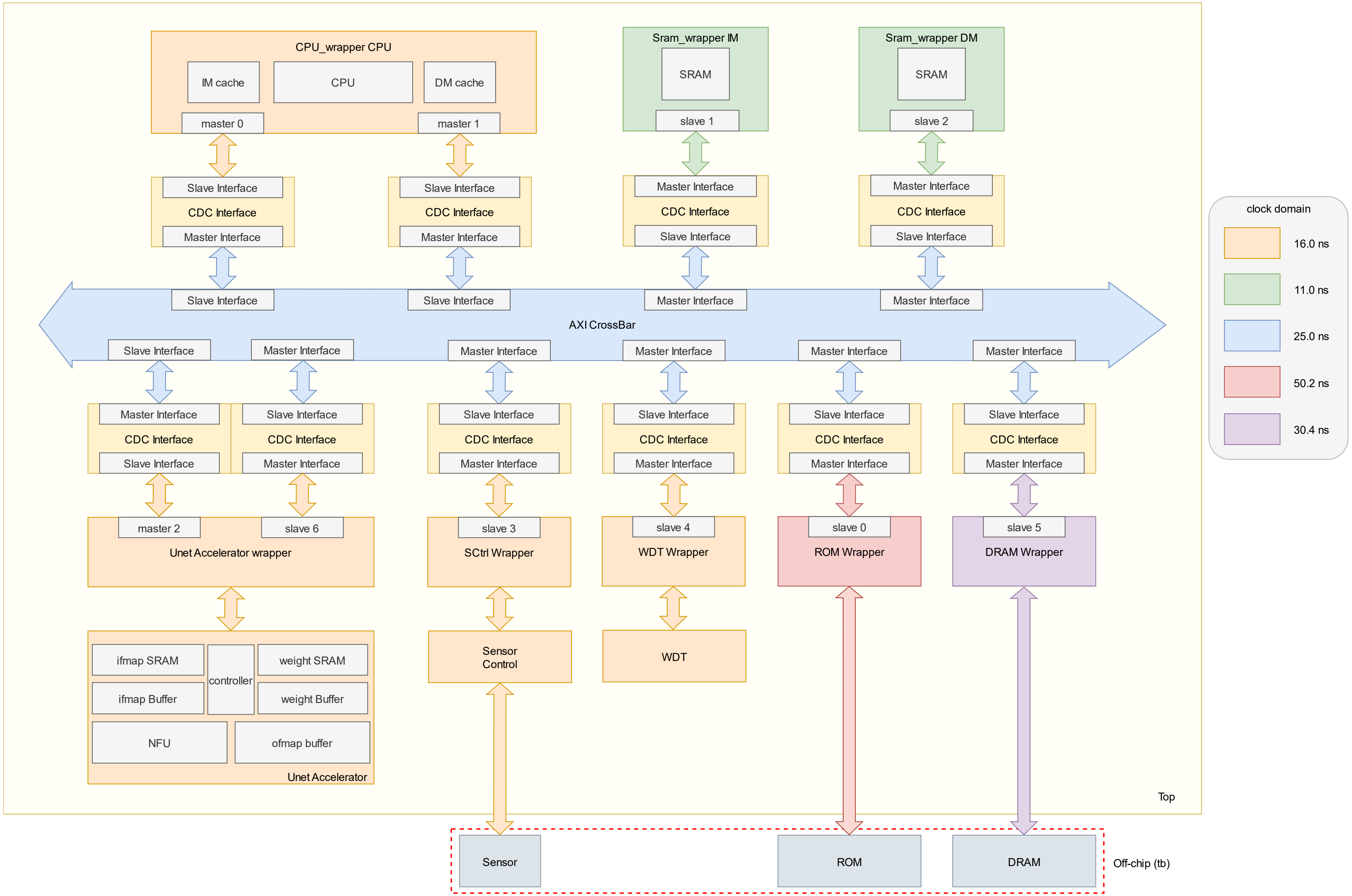
- IC Compiler I

- IC Compiler II

- MTK IC design program

Side Project – RISC-V CPU System

☐ Overview



Side Project – RISC-V CPU System

☐ Spec.

✓ CPU
- 5-stage pipelined RISC-V CPU
- ISA: RV32IM
- Handle 51 instructions

✓ Lite - AXI4
- In-order
- Burst length up to 128
- Verification through Cadence JasperGold VIP

✓ Clock Domain Crossing (CDC)
- Handle 5 clock domain between
  - ➢ CPU, AI Accelerator
  - ➢ SRAM
  - ➢ DRAM
  - ➢ ROM
  - ➢ AXI BUS
- Asynchronous FIFO
- Verification through Synopsys Spyglass

✓ One way direct mapped cache
- 64 entries
- 4 words per entry
- Write through

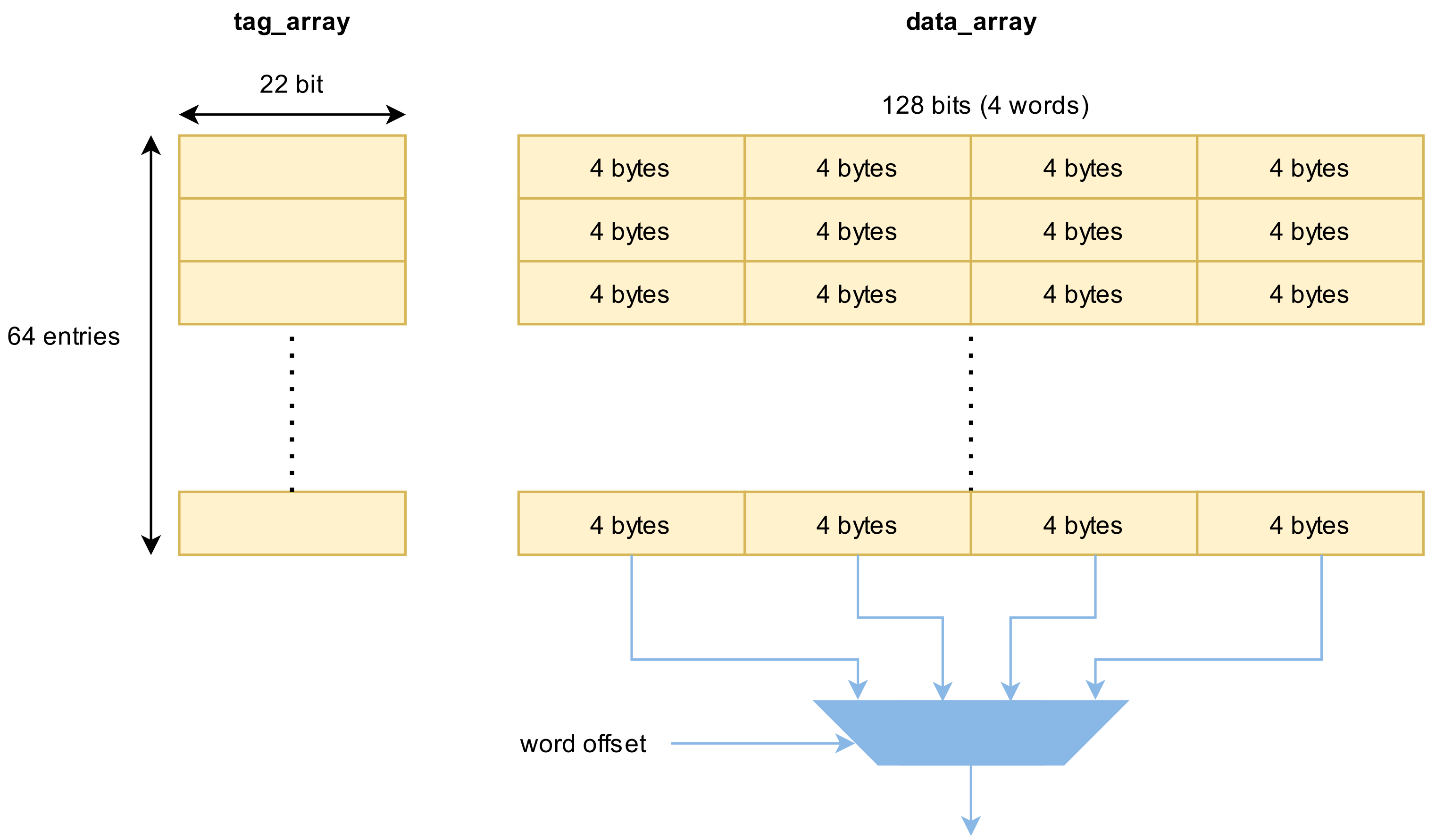
✓ Interrupt
- External interrupt – external I/O (Sensor controller)
- Timer interrupt – Watch Dog Timer (WDT)

✓ Booting
- The first program that the CPU will execute after reset.
- The booting program is stored in ROM.
- The booting program will move instructions and data from DRAM to IM and DM

Side Project – RISC-V CPU System

☐ One way direct mapped cache (L1)

31:10	9:4	3:2	1:0
tag	entry index	word offset	byte offset
22 bit	6 bit	2 bit	2 bit

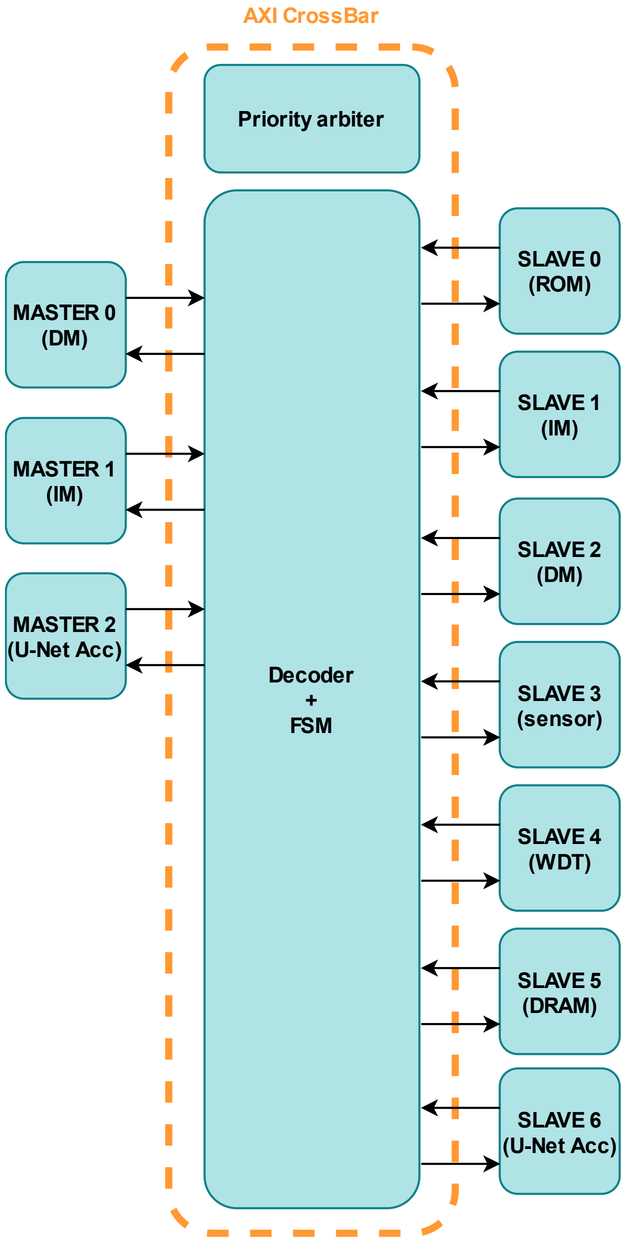


✓ One way direct mapped cache

- 1K bytes
- 4 words per entry
- 64 entries
- One-way direct mapping
- Read policy
  - ➢ Read miss: read allocate
- Write policy
  - ➢ Write hit: write through
  - ➢ Write miss: write no allocate

Side Project – RISC-V CPU System

☐ AXI4



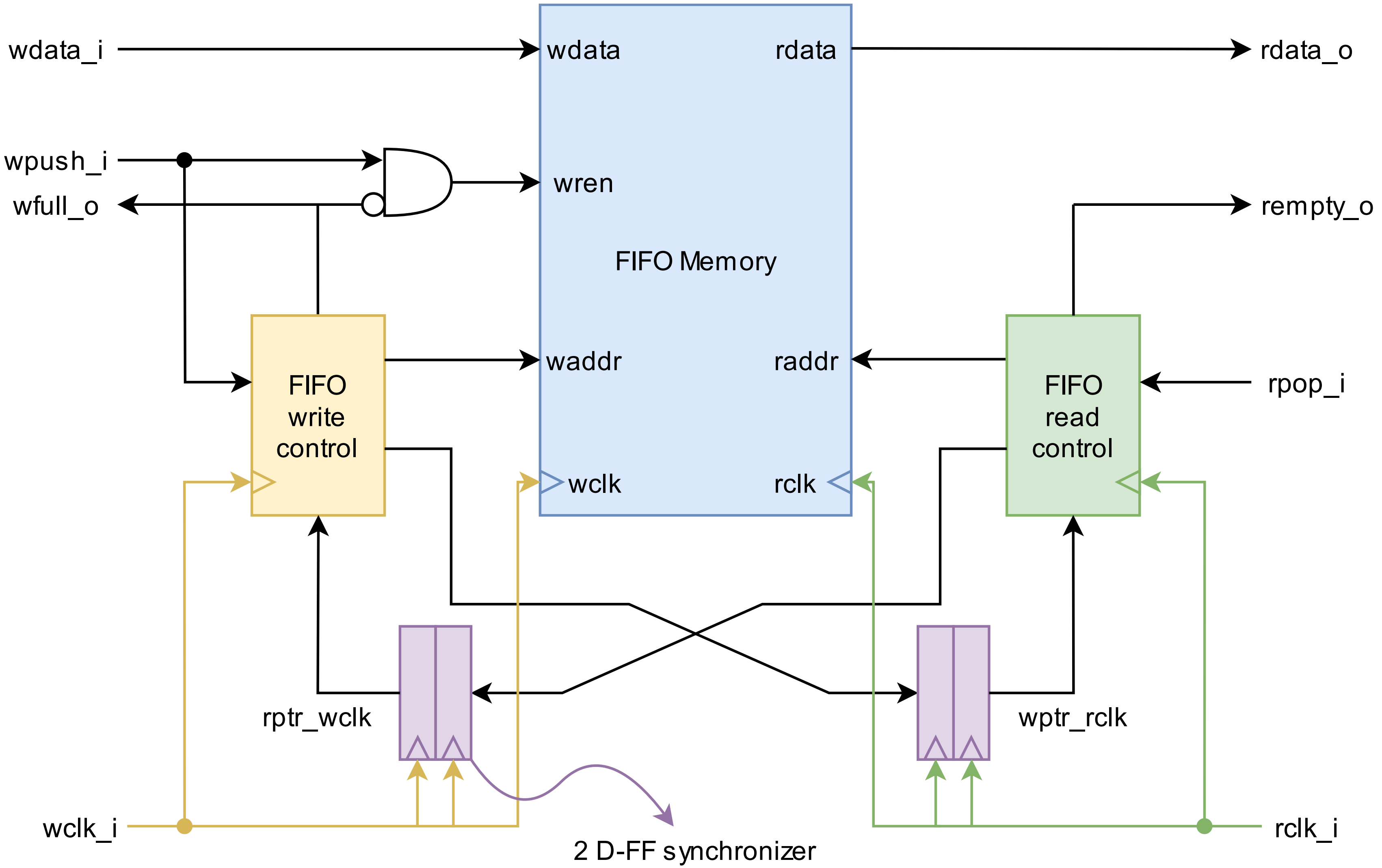
✓ AXI4

- In-order AXI
- Burst length up to 128
- Verification through JasperGold VIP
- Support
  - ✓ AxADDR
  - ✓ AxLEN (up to 128)
  - ✓ AxSIZE (4 byte)
  - ✓ AxBURST (INCR only)
  - ✓ AxVALID
  - ✓ AxREADY
  - ✓ xDATA
  - ✓ xSTRB
  - ✓ xVALID
  - ✓ xREADY
  - ✓ xRESP (OKAY only)

  (x : read/write)

Side Project – RISC-V CPU System

☐ Clock domain Crossing

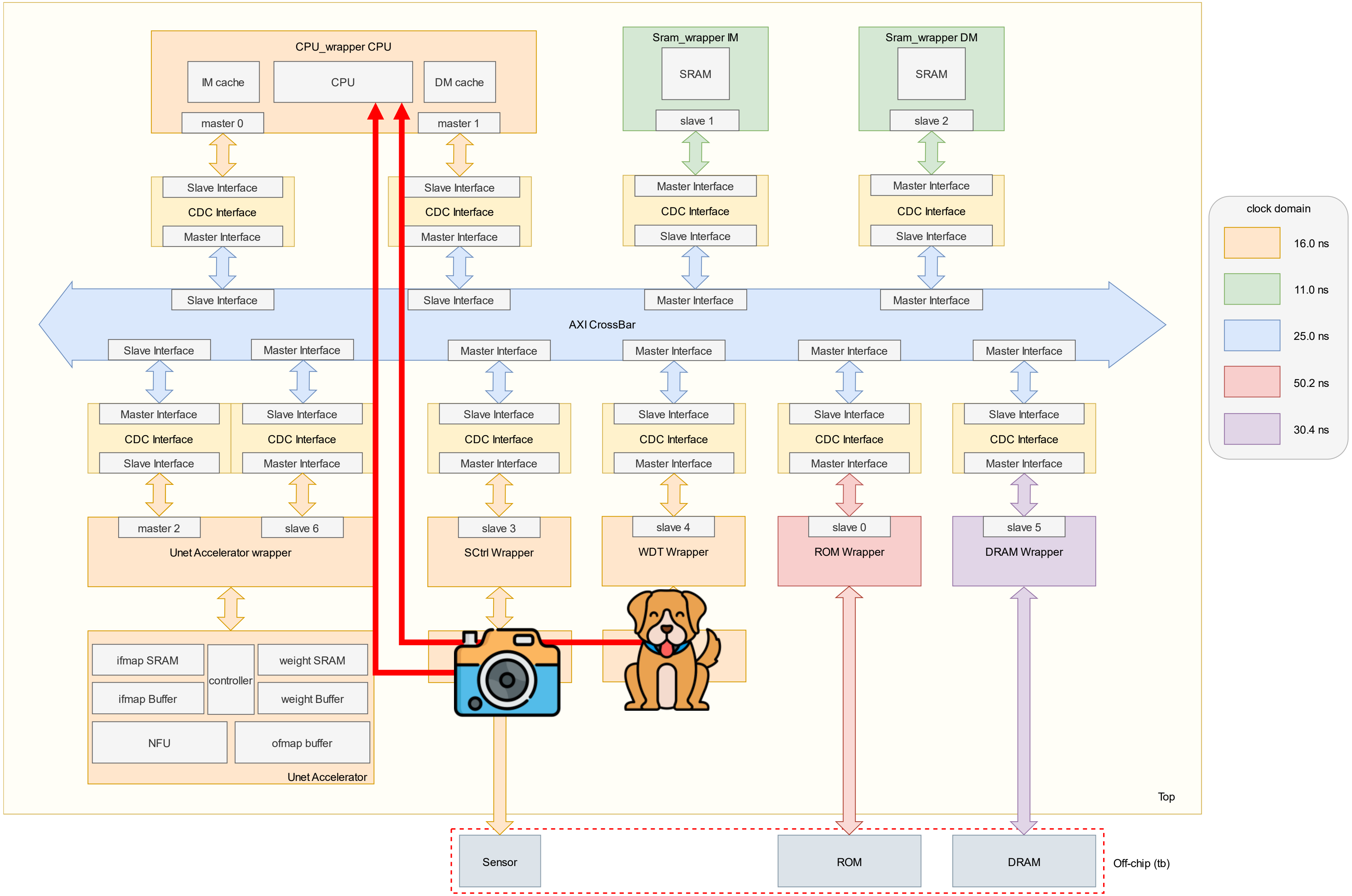


✓ Clock Domain Crossing(CDC)

- Use asynchronous FIFO
- Spyglass proved
- Address size = 4
- Data size = 32 bits

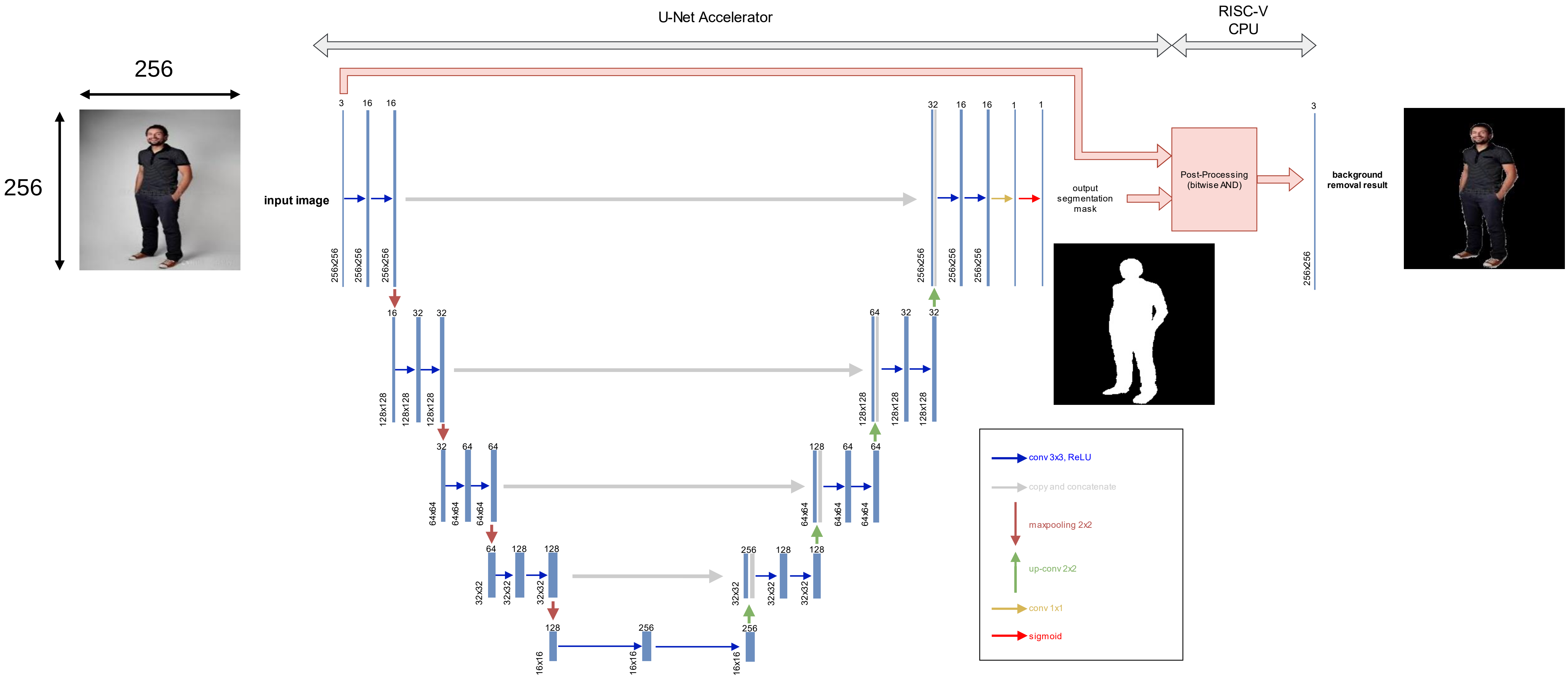
Side Project – RISC-V CPU System

- Interrupt
  - Timer
  - External



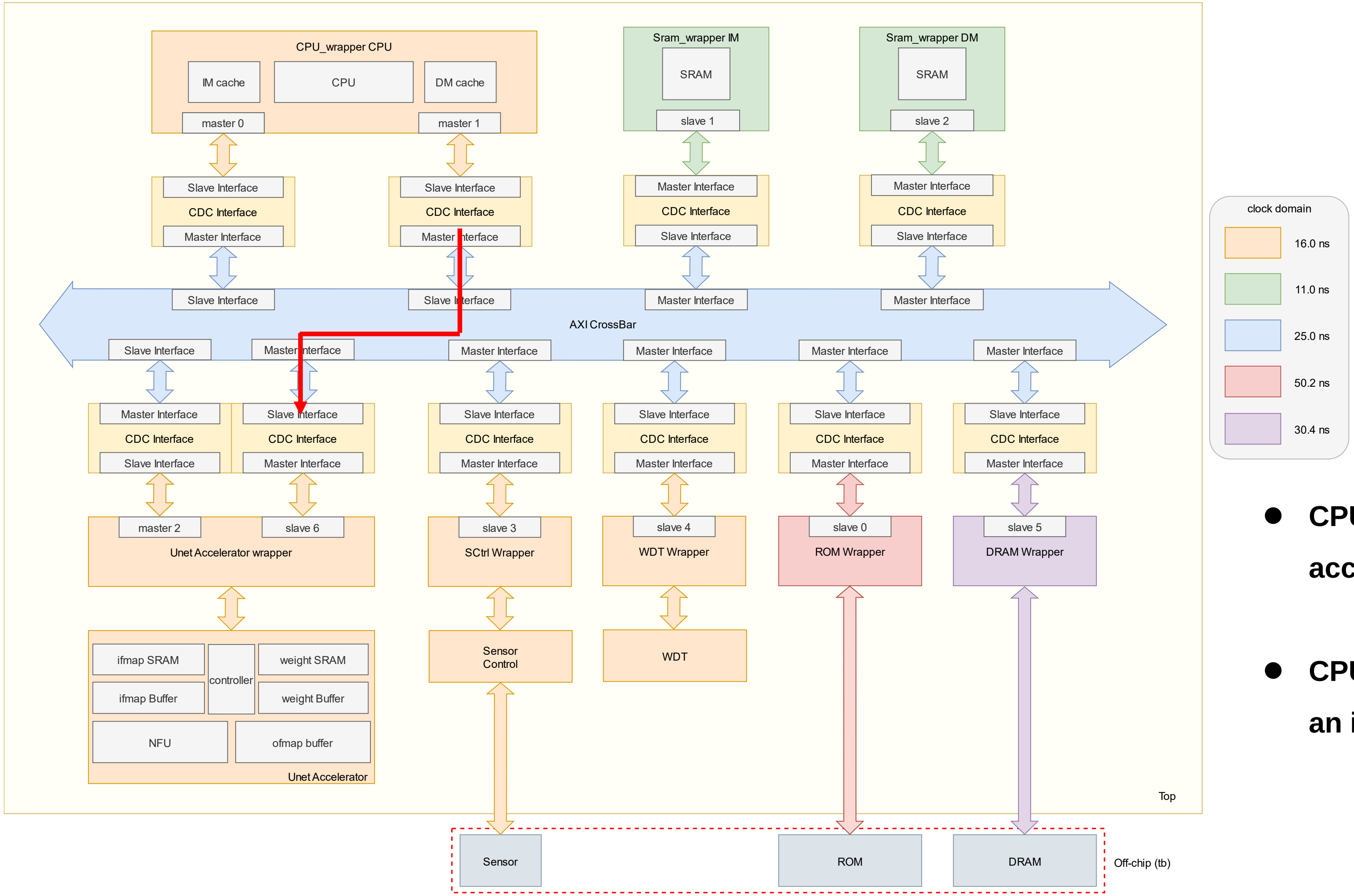
Side Project – RISC-V CPU System

☐ Application of U-Net: Image Background Removal



Side Project – RISC-V CPU System

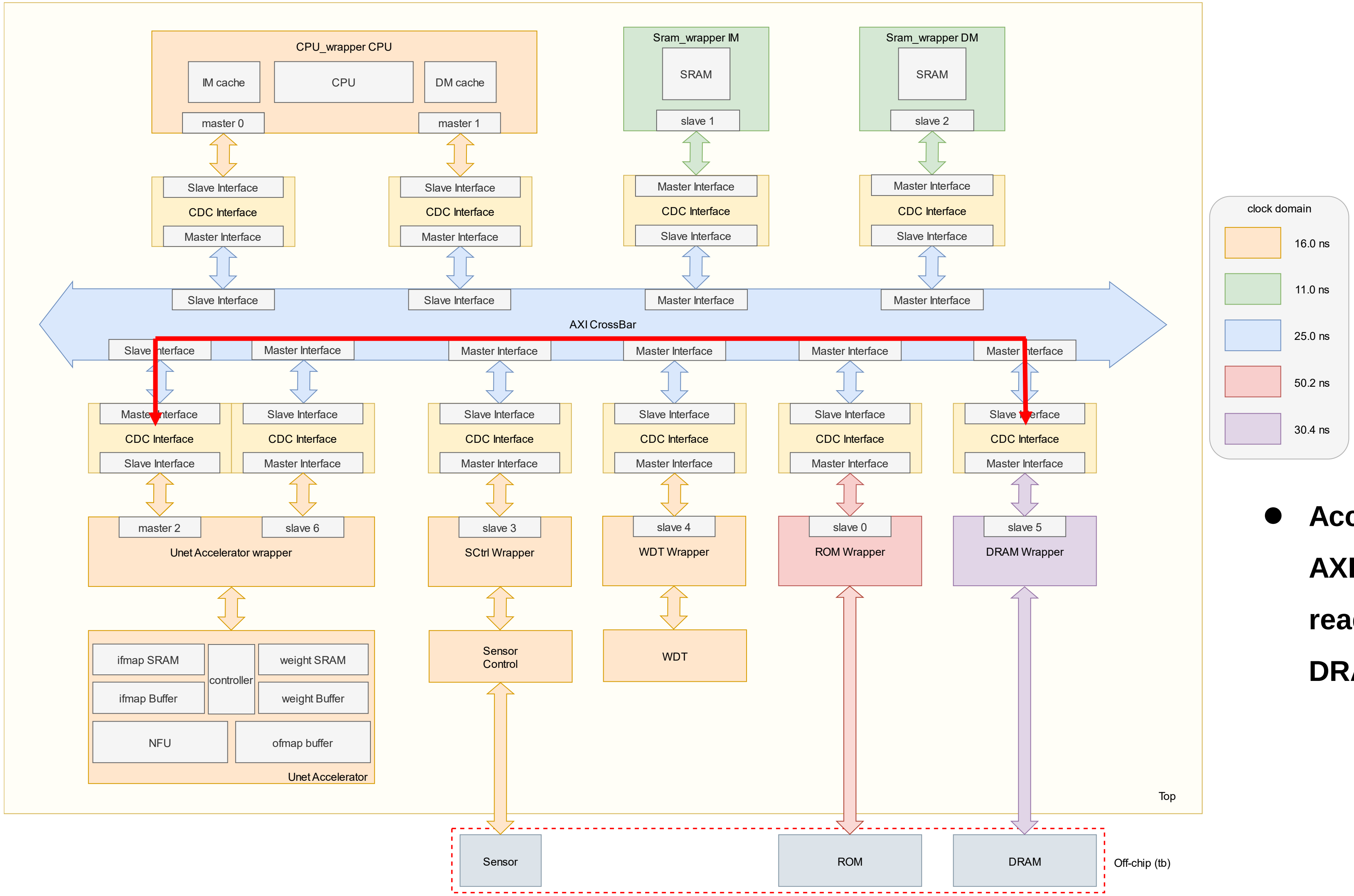
◻ U-Net Accelerator data flow



- **CPU calls the accelerator to start.**
- **CPU starts to wait for an interrupt.**

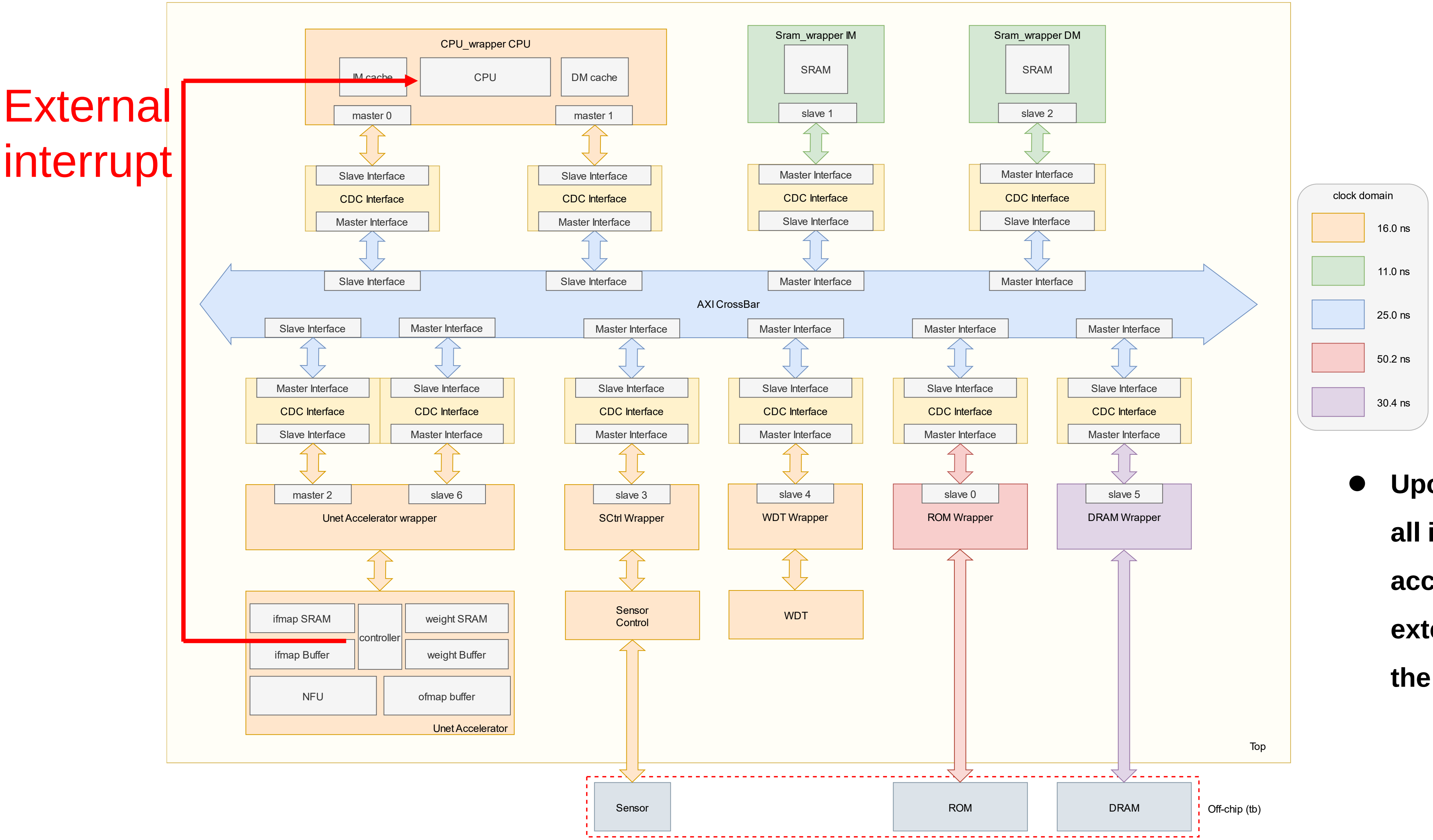
Side Project – RISC-V CPU System

☐ U-Net Accelerator data flow



- **Accelerator acts as an AXI master, directly reading and writing DRAM data.**

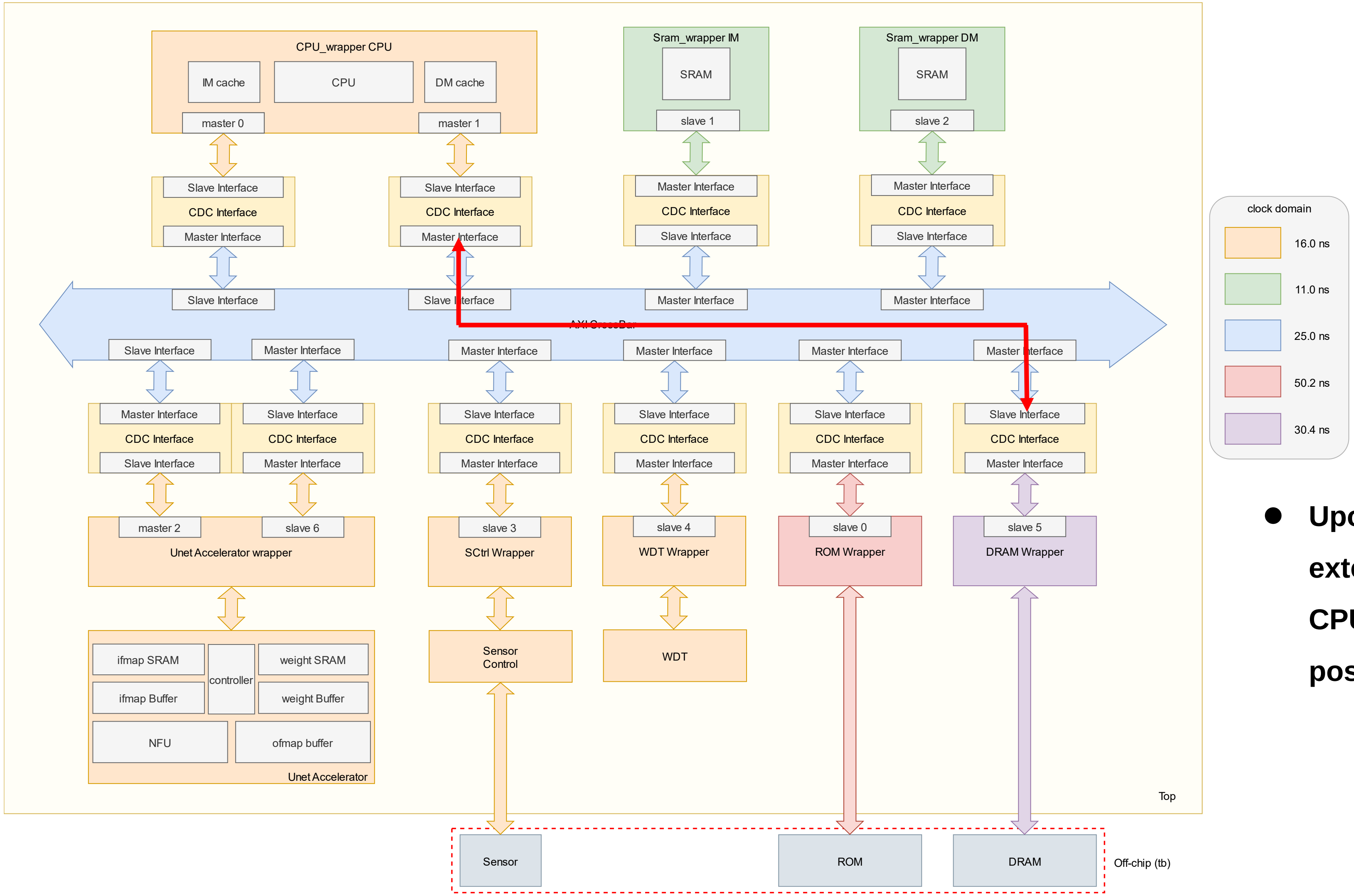
☐ U-Net Accelerator data flow



- **Upon completion of all inference, the accelerator sends an external interrupt to the CPU.**

Side Project – RISC-V CPU System

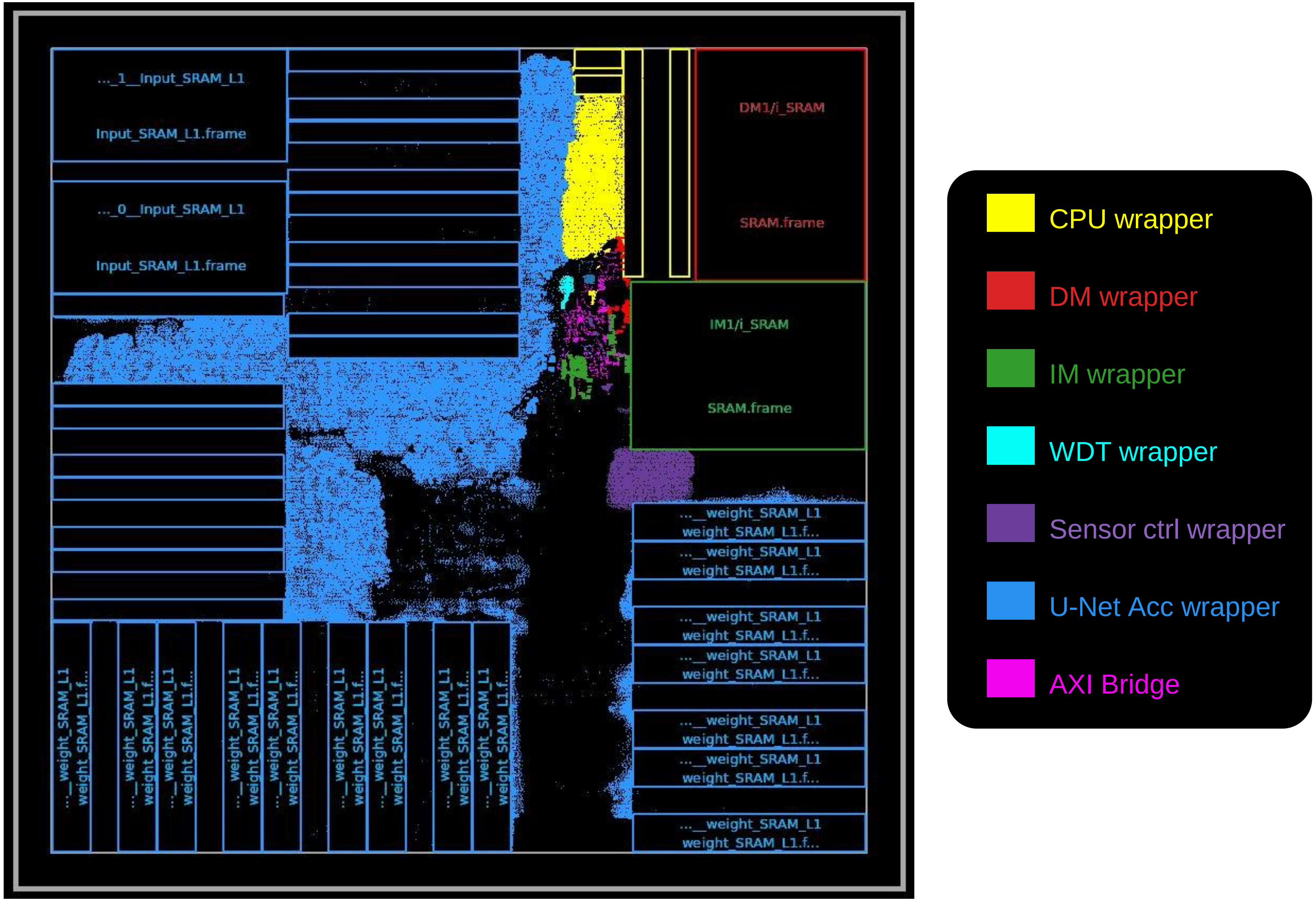
U-Net Accelerator data flow



- **Upon receiving an external interrupt, the CPU start to do image post-processing.**

Side Project – RISC-V CPU System

- APR result (U18)

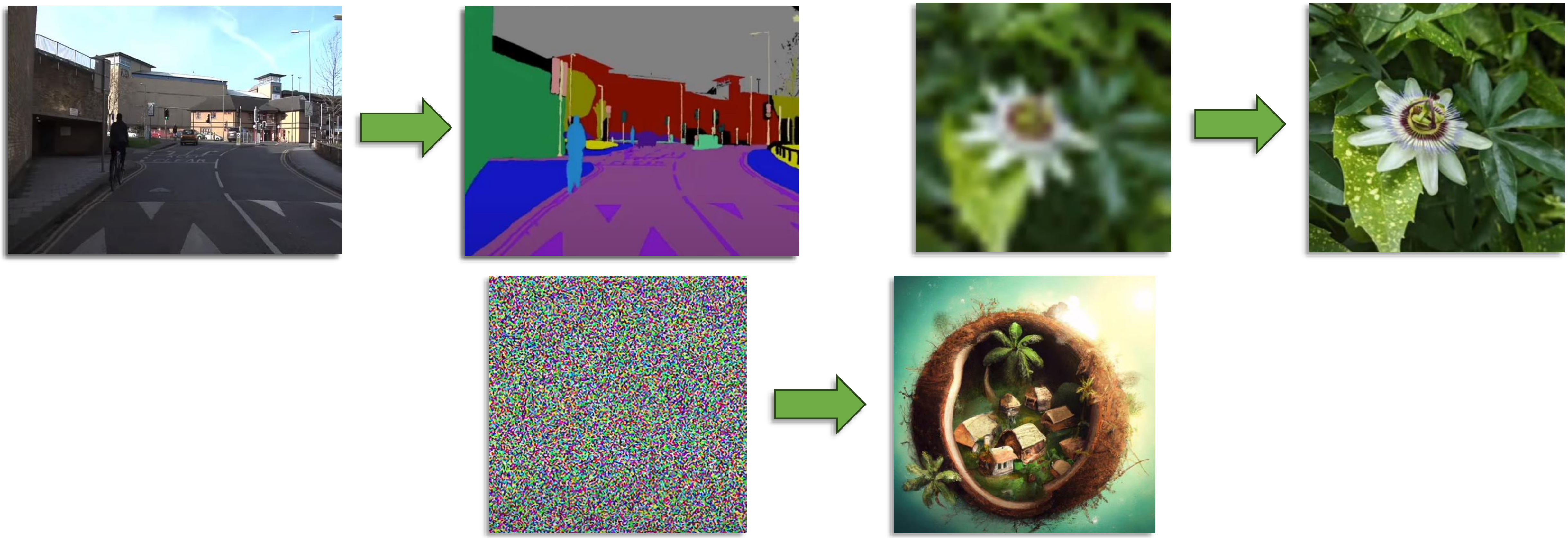


Thank you for listening ☺

Appendix

Application background & AI model

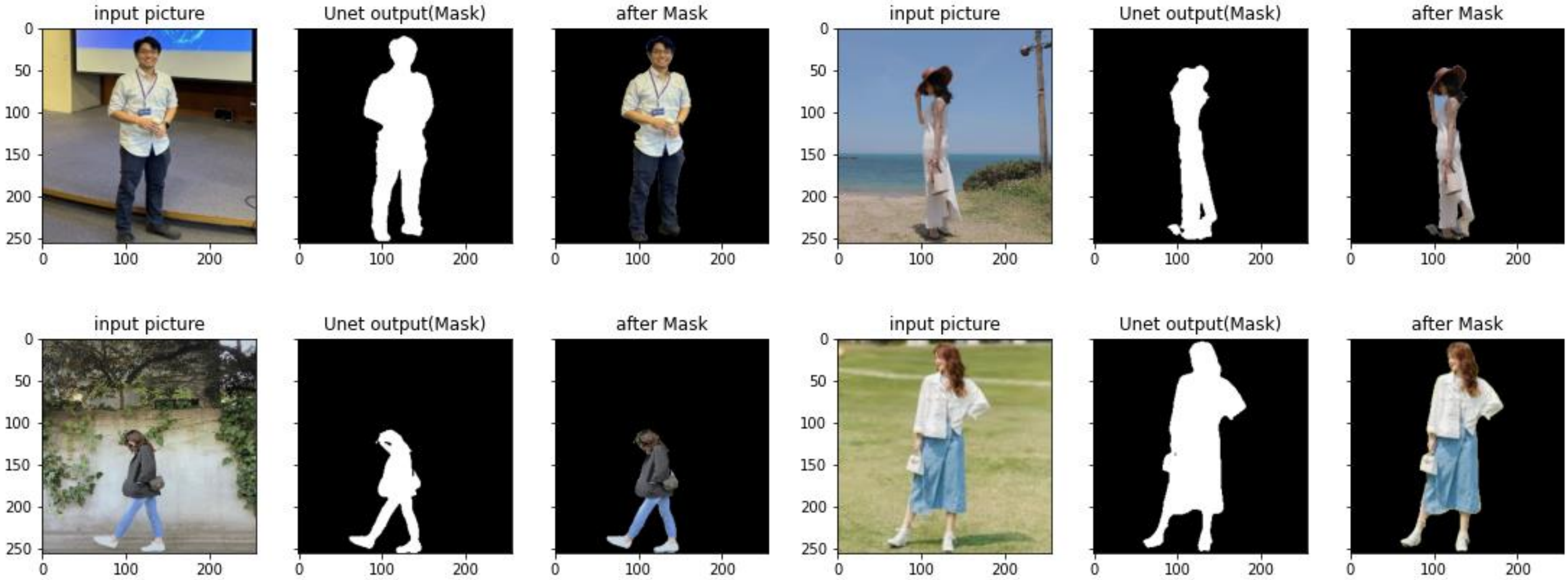
- Introduction to U-Net
  - The U-Net paper was introduced in 2015, initially focused on medical image segmentation.
  - It has become popular not only in medical image analysis but also in various computer vision applications.



[Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. http://arxiv.org/abs/1505.04597]

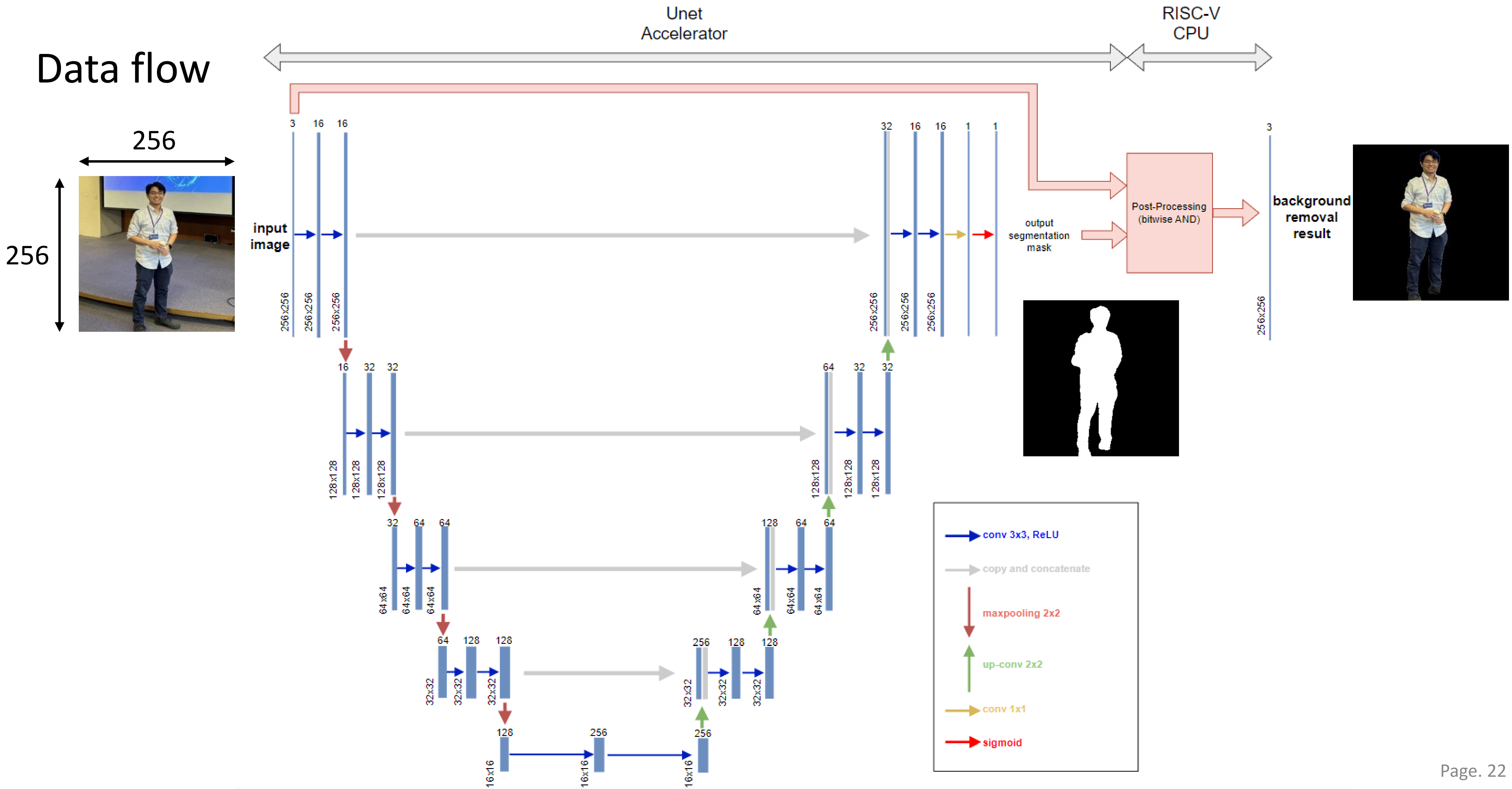
Application background & AI model

- Application
  - **Image Background Removal**



AI model

- Data flow



Application background & AI model

- Model Quantization

$$r = S(q - Z)$$

r : real number
q : integer
S : scale
Z : zero point

input/output:uint8
weight:int8

- Calculate convolution in integer

$$r_3 = \sum r_1 r_2 \Rightarrow S_3(q_3 - Z_3) = \sum S_1(q_1 - Z_1)\, S_2(q_2 - Z_2)$$

$$\Rightarrow q_3 = Z_3 + M \sum S_1(q_1 - Z_1)\, S_2(q_2 - Z_2)$$

$$M = \frac{S_1 S_2}{S_3} = 2^{-n} M_0$$

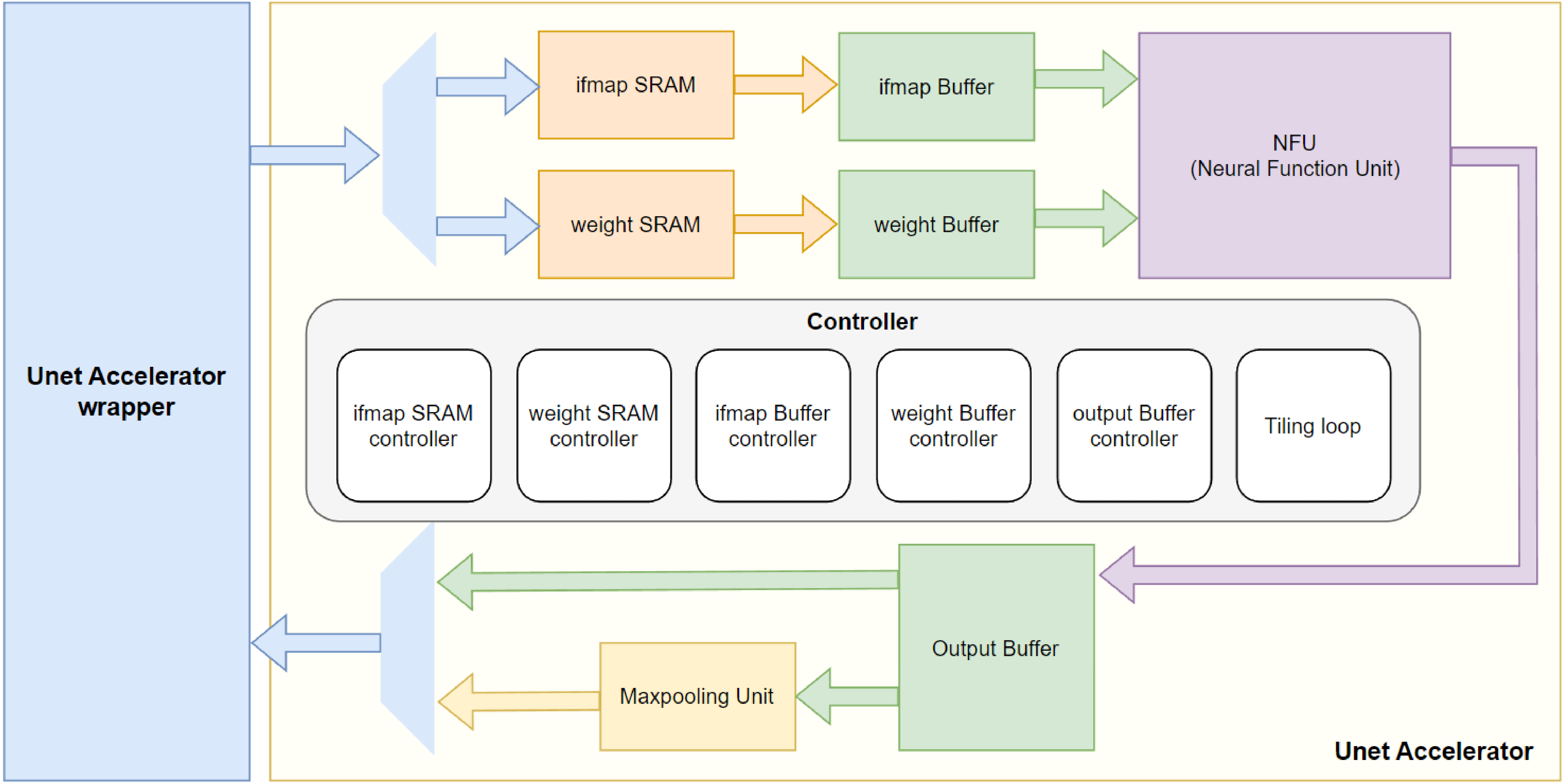
right shift n bit

fixed point multiplication

[Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2017). *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. http://arxiv.org/abs/1712.05877]

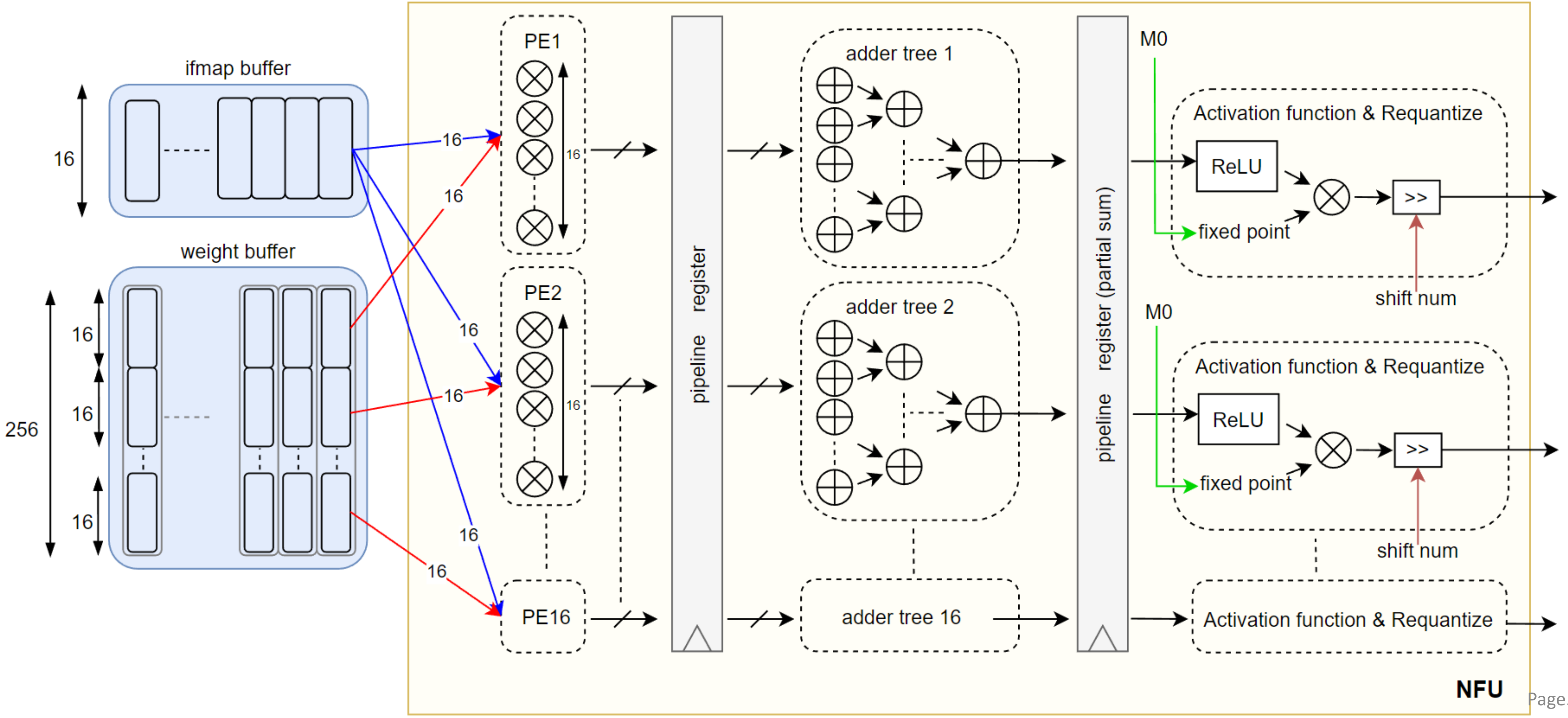
AI model

- Unet accelerator architecture



AI model

- Unet accelerator – NFU(Neural Function Unit)



Distinct Specification

- Support 3 kinds of convolution and maxpooling
    1. 1x1 convolution
    2. 3x3 convolution
    3. up-convolution
    4. 2x2 maxpooling
- Frame/sec (U18 process)
    1. only Unet accelerator : 2 Frame/sec
    2. whole system : 0.23 Frame/sec

Distinct Specifications

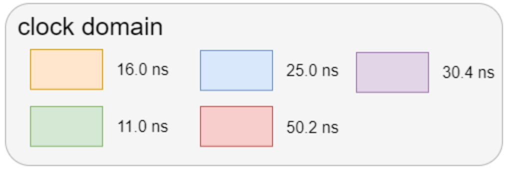
- Size of Storage Units
  - Units directly connected on AXI

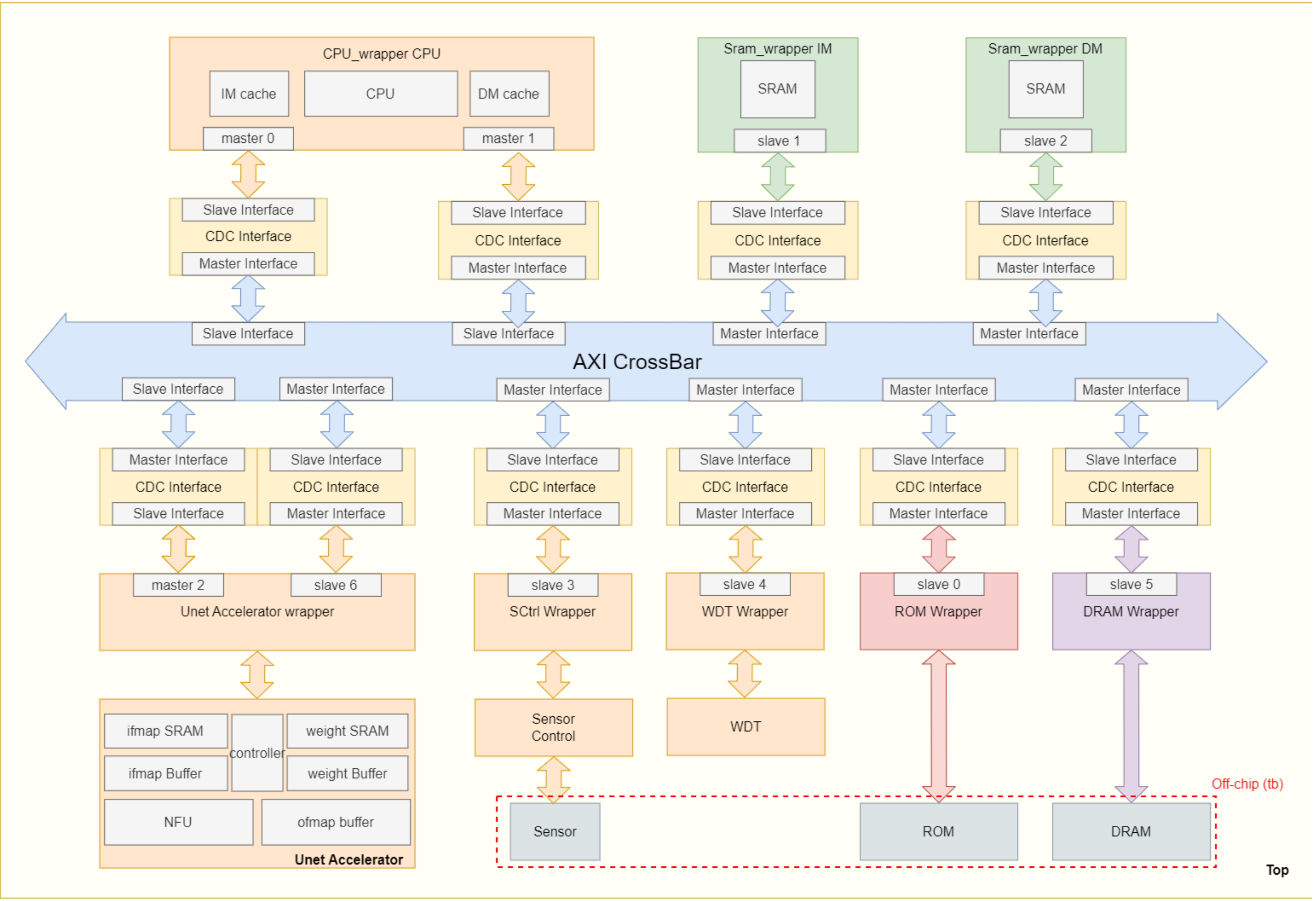
DRAM	ROM	Sensor	IM	DM
33MB	16KB	200KB	64KB	64KB

  - Units within the accelerator

	Ifmap	Weight	Ofmap
Buffer	16KB	32KB	1KB
SRAM	128KB	72KB	None

System Architecture

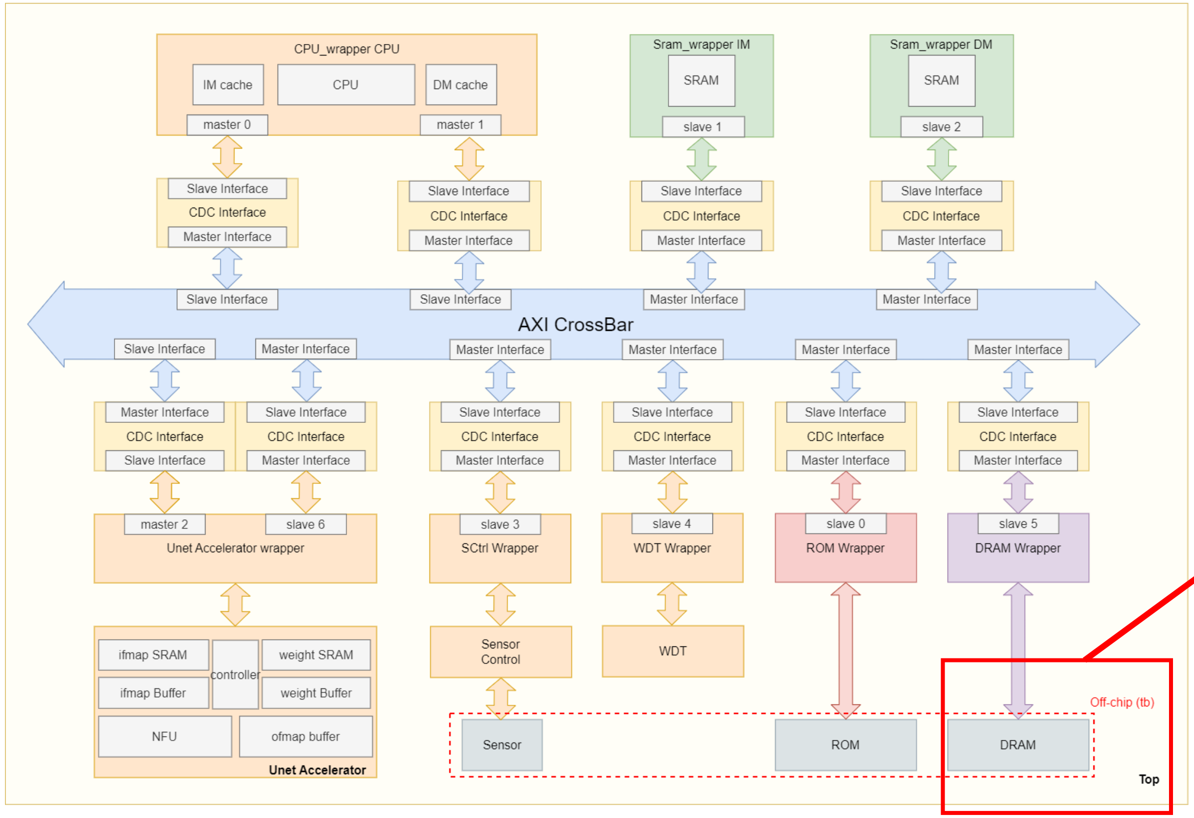


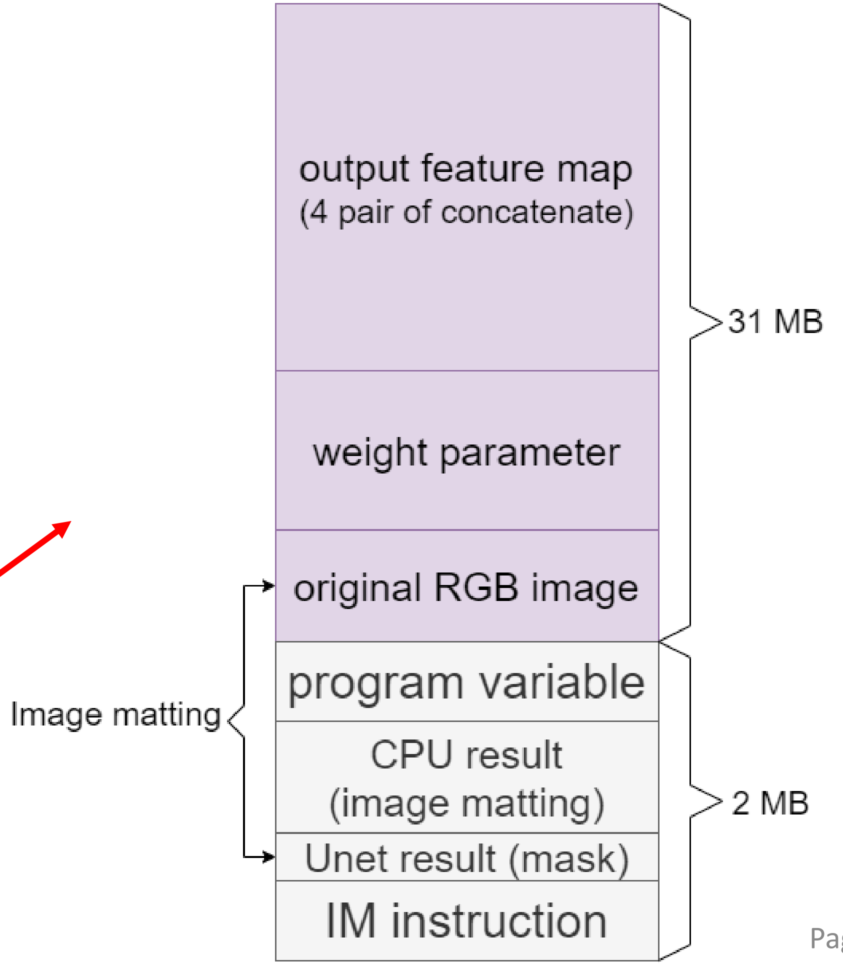


- CPU
  - 32-bit 5-stage RISC-V
  - RV32I、M extension
  - IM, DM Cache
- AXI4
  - Read Burst length up to 16
- ROM
  - Store booting program
- DRAM
  - Store program and accelerator data

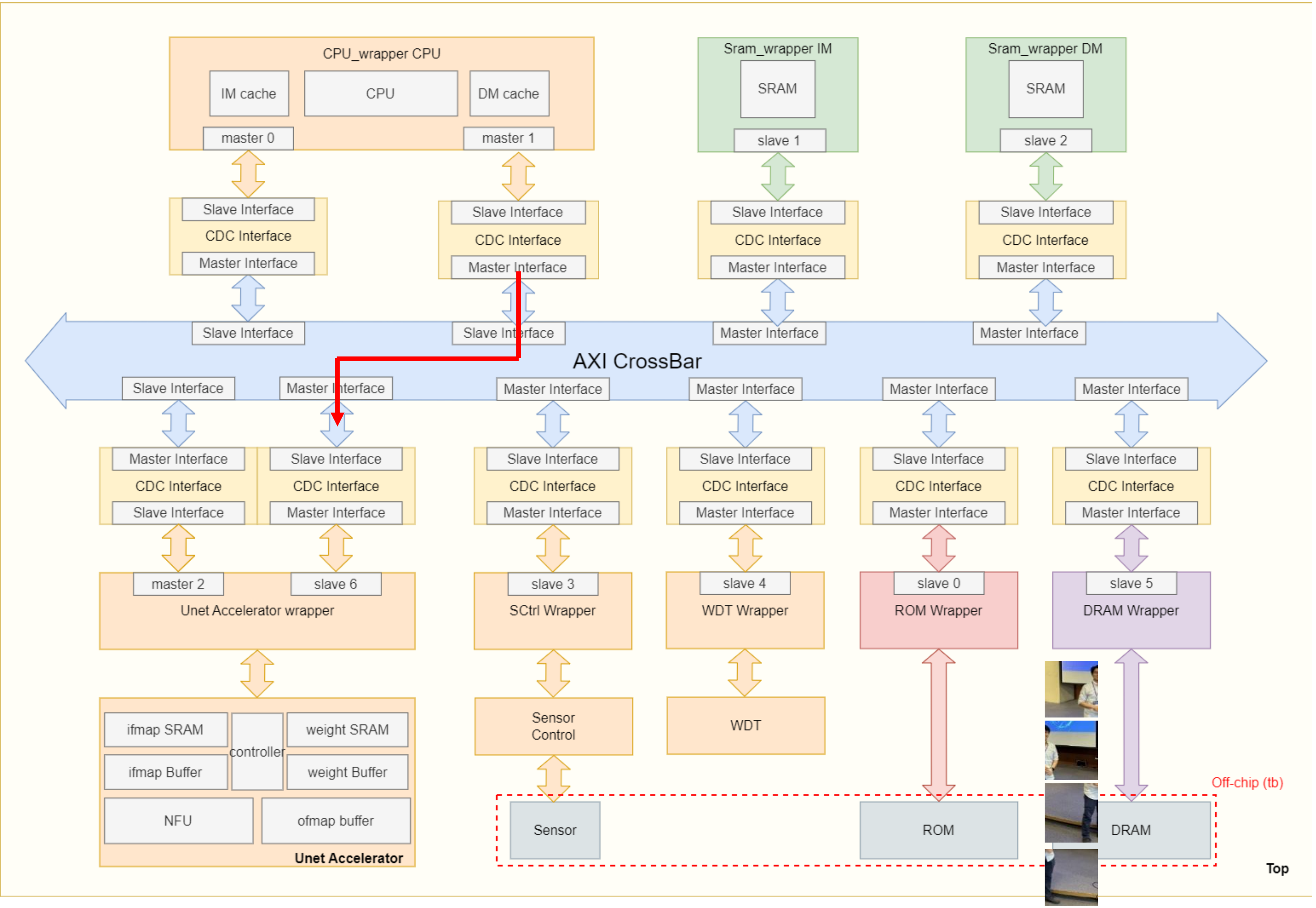
System Architecture – DRAM mapping

1. Loading the RGB image and weight parameter into DRAM at first.
2. Due to Unet concatenation property, DRAM region is required to store the previous output feature map.



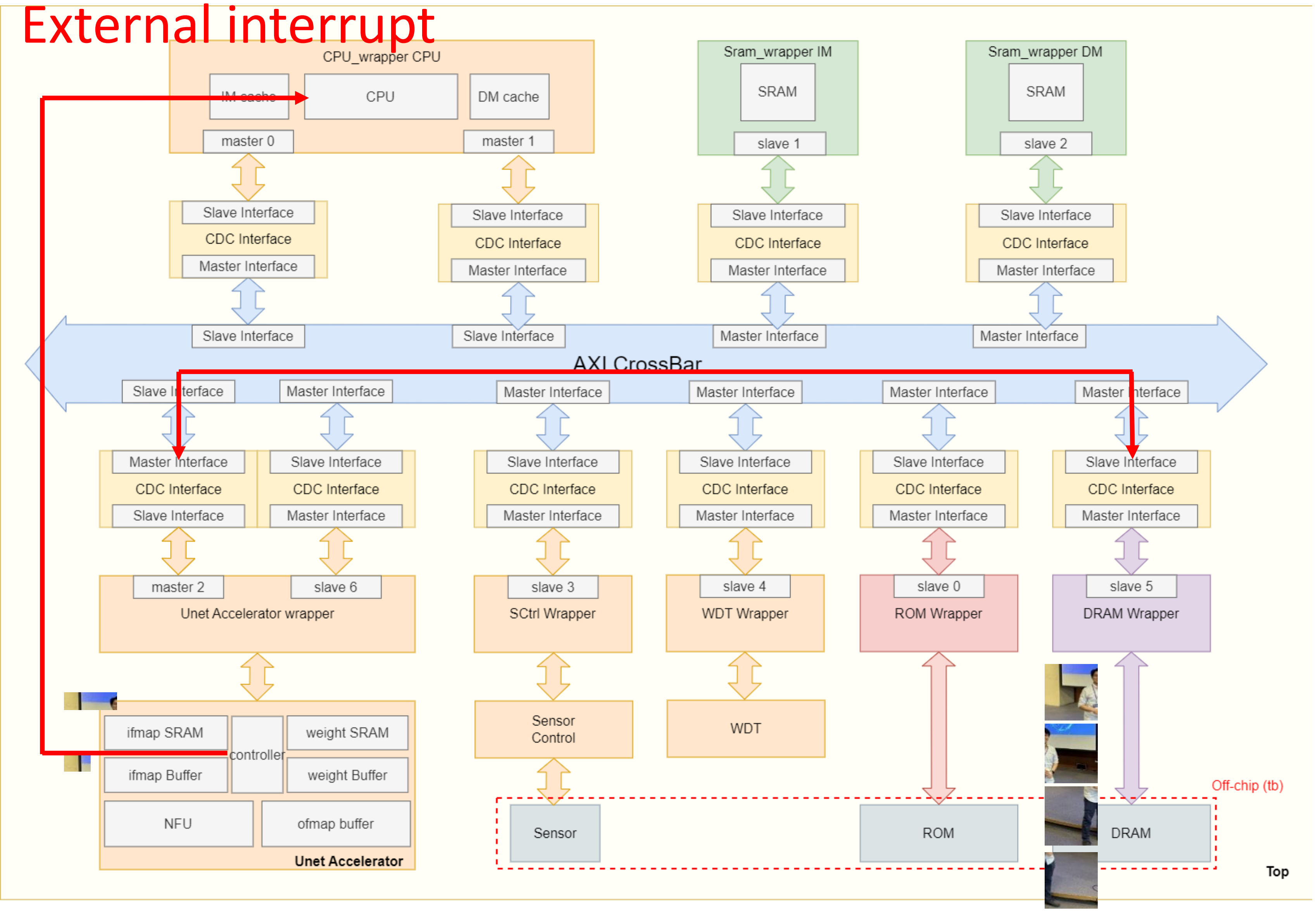


System Architecture – CPU call accelerator to start



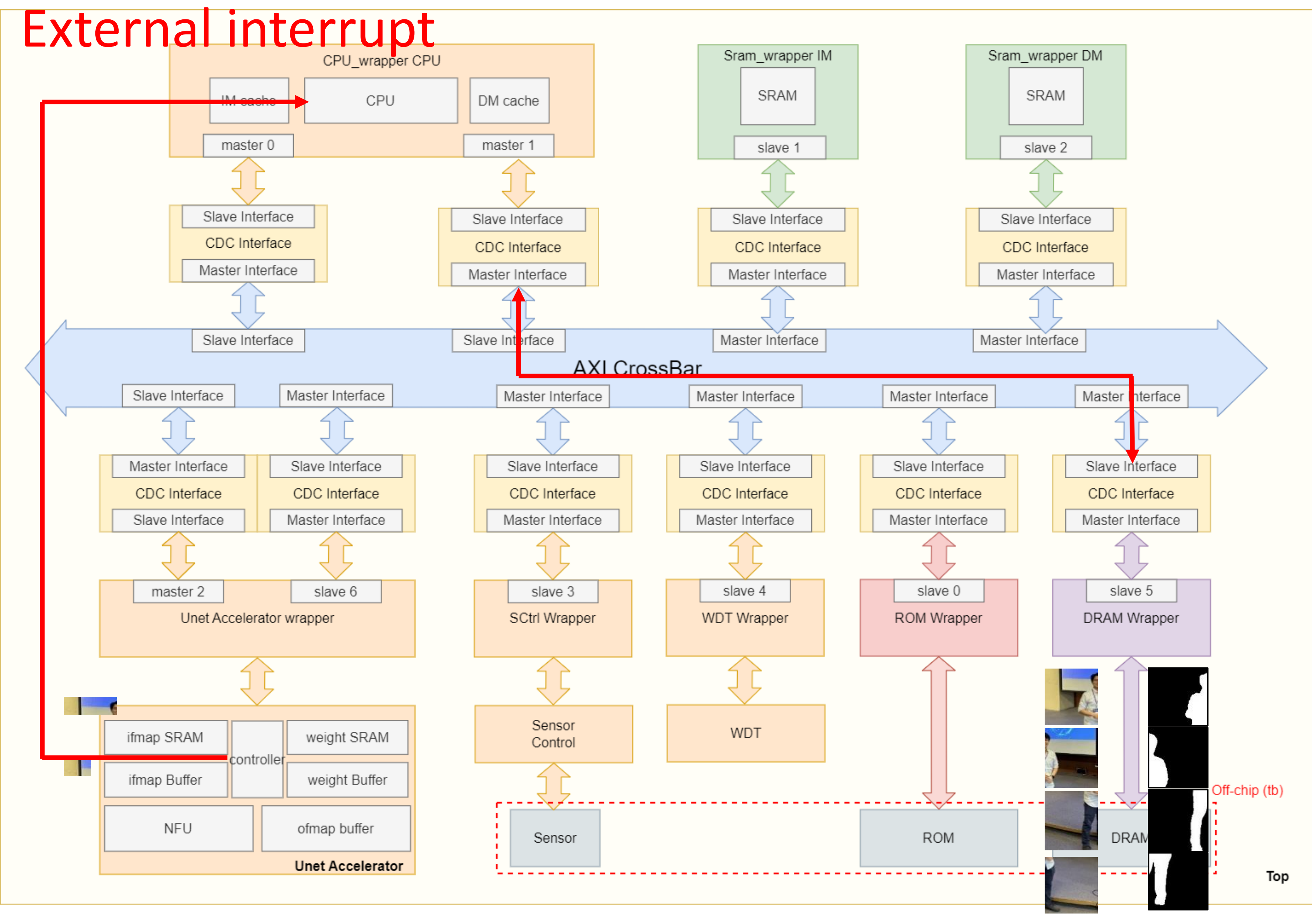
- The CPU will initiate Unet accelerator to begin inference via the AIX write channel.
- CPU wait for interrupt.

System Architecture – Accelerator call external interrupt



- Accelerator acts as an AXI master, directly reading and writing DRAM data.
- Upon completion of all inference, the accelerator sends an external interrupt to the CPU.

System Architecture – CPU start to image matting

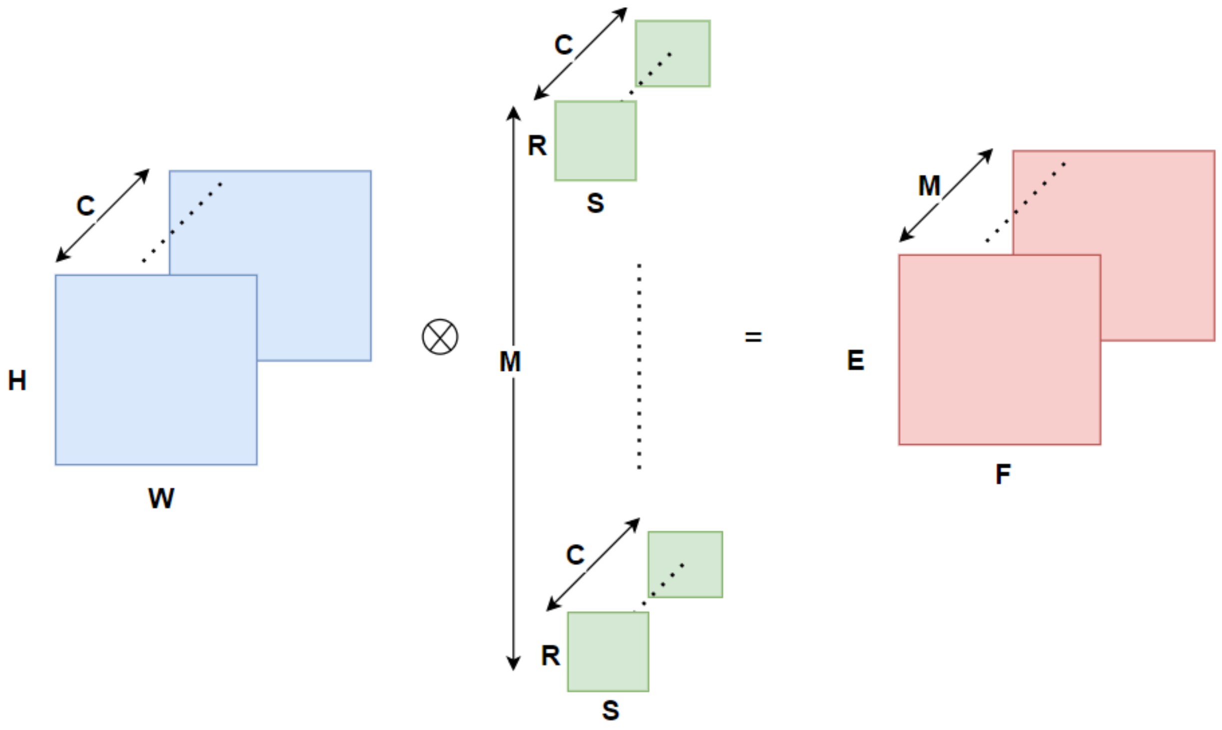


- Upon receiving an external interrupt, the CPU start to do image matting.
- The grayscale mask and original RGB data already stored in DRAM.

Data flow

- Direct Convolution
    - Assume U is 1, P is 1 E=H, F=W.

Ofmap parameters:
E = F = (H-R+2P+U)/U, if H=W.
Where *P* is #padding, *U* is stride.

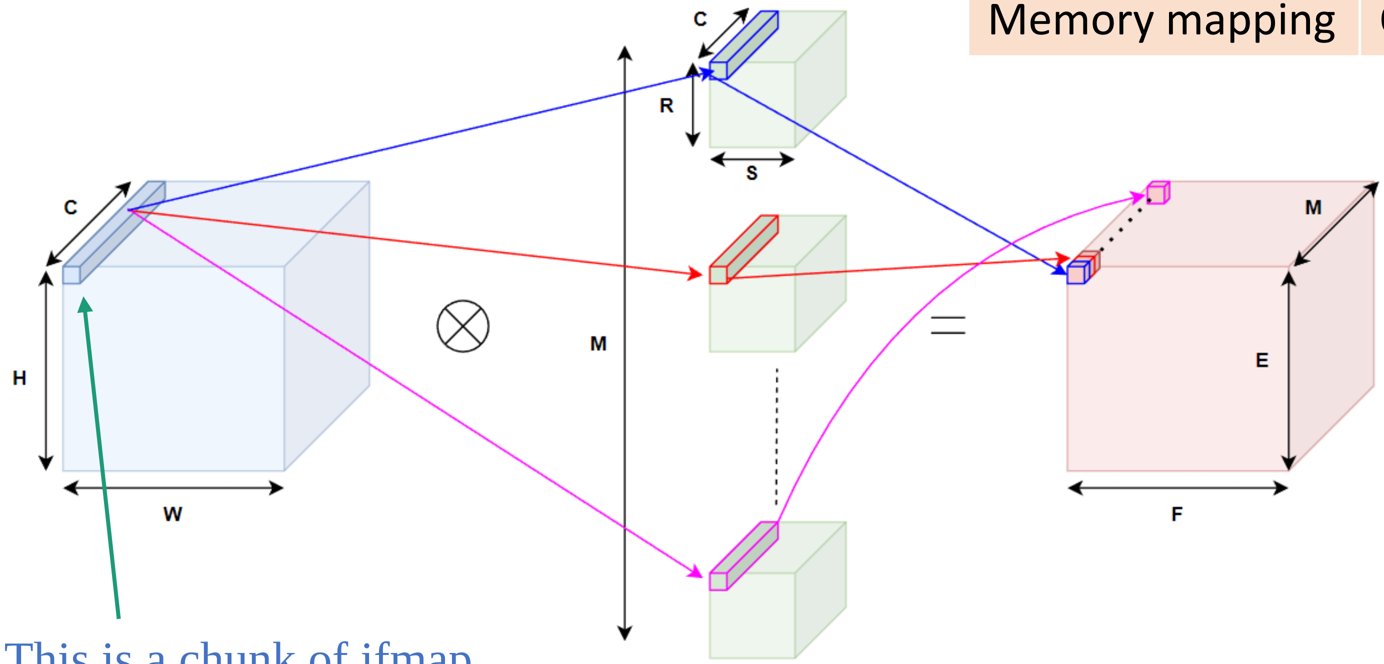


Shape Parameter	Description
N	batch size of 3D fmaps
M	# of 3D filters / # of ofmap channels
C	# of ifmap/filter channels
H/W	ifmap plane height/width
R/S	filter plane height/width
E/F	ofmap plane height/width

Data flow

- Input stationary data flow to reduce ifmap data movement
  - Same chunk of ifmap is reused multiple times across M channel.

	Ofmap	Ifmap	Filter
Memory mapping	O[E][F][M]	I[H][W][C]	W[R][S][M][C]



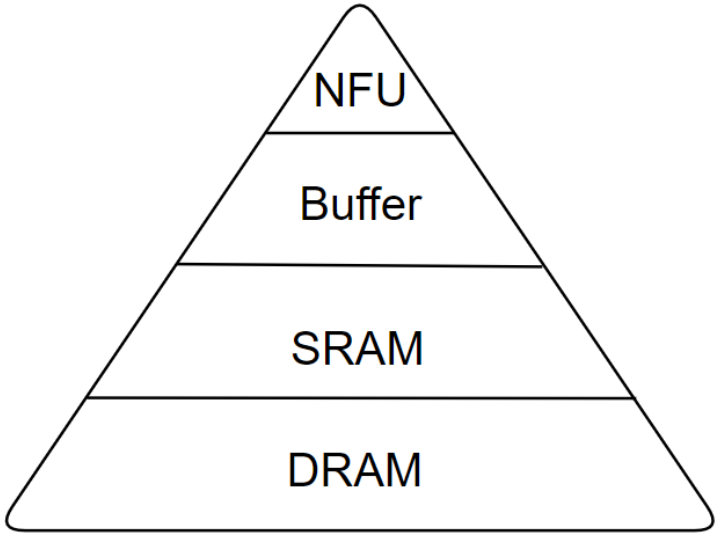
This is a chunk of ifmap

Loop nest without tiling

```
for h in range(H):
    for w in range(W):
        e = h
        f = w
        for r in range(R):
            for s in range(S):
                for m in range(M):
                    for c in range(C):
                        O[e][f][m] += W[r][s][m][c] * I[r+h][s+w][c]
```

Data flow

- Loop tiling to alleviate data movement.
- Memory hierarchy
  - Buffer level
  - SRAM level



		INPUT				
	layer	(H) hight	(W)wdith	(C)channel	size	KB
conv	1	256	256	3	196608	192
conv	2	256	256	16	1048576	1024
maxpool	3			0	0	0
conv	4	128	128	16	262144	256
conv	5	128	128	32	524288	512
maxpool	6	0	0	0	0	0
conv	7	64	64	32	131072	128
conv	8	64	64	64	262144	256
maxpool	9	0	0	0	0	0
conv	10	32	32	64	65536	64
conv	11	32	32	128	131072	128
maxpool	12	0	0	0	0	0
conv	13	16	16	128	32768	32
conv	14	16	16	256	65536	64

Layer2 input is 1024KB!

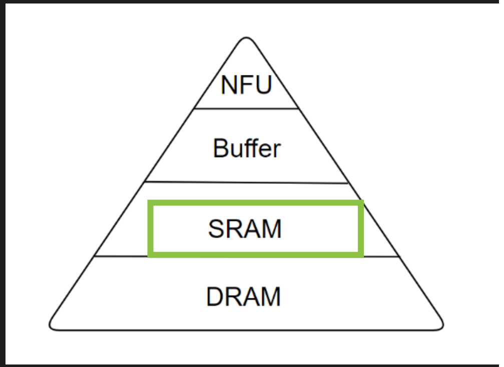
Layer14 weight is 576KB!

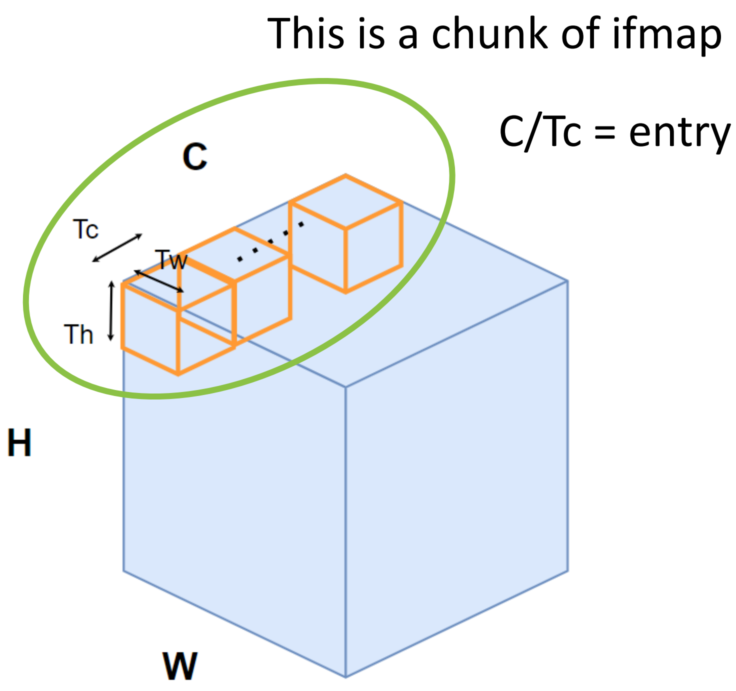
		WEIGHT						
	layer	kernel size	(R) hight	(S)wdith	(C)channel	(M)filter	size	KB
conv	1	3	3	3	3	16	432	0.421875
conv	2	3	3	3	16	16	2304	2.25
maxpool	3		0	0		0	0	0
conv	4	3	3	3	16	32	4608	4.5
conv	5	3	3	3	32	32	9216	9
maxpool	6		0	0		0	0	0
conv	7	3	3	3	32	64	18432	18
conv	8	3	3	3	64	64	36864	36
maxpool	9		0	0		0	0	0
conv	10	3	3	3	64	128	73728	72
conv	11	3	3	3	128	128	147456	144
maxpool	12		0	0		0	0	0
conv	13	3	3	3	128	256	294912	288
conv	14	3	3	3	256	256	589824	576

Data flow

- Ifmap SRAM tiling, tiling size Th,Tw,C.

```
for(int hh=0; hh<H; hh+=(Th)) begin
    for(int ww=0; ww<W; ww+=(Tw)) begin
        ifmap_sram = load_ifmap_sram(); // load ifmap chunk into ifmap SRAM

        for(int mmm=0; mmm<M; mmm+=Tmm) begin
            weight_sram = load_weight_sram(); // load weight chunk into weight SRAM

            for(int h=hh; h<hh+Th; h+=stride) begin
                for(int w=ww; w<ww+Tw; w+=stride) begin
                    ifmap_buffer = load_ifmap_buffer(); // load a sliding window of ifmap into buffer
                    mm_end = (M<Tmm) ? M : (Tmm+mmm);

                    for(int mm=mmm; mm < mm_end; mm+=Tm) begin
                        weight_buffer = load_weight_buffer(); // load a sliding window of weight into buffer

                        // NFU begins to do MAC operation
                        for(int r=0; r<R; r++) begin
                            for(int s=0; s<S; s++) begin
                                for(int i=0; i<entry; i++) begin
                                    // COMPUTATION
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

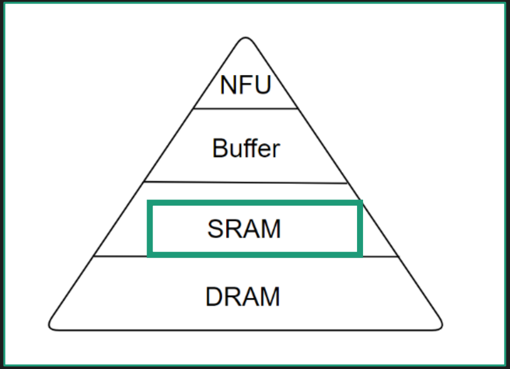




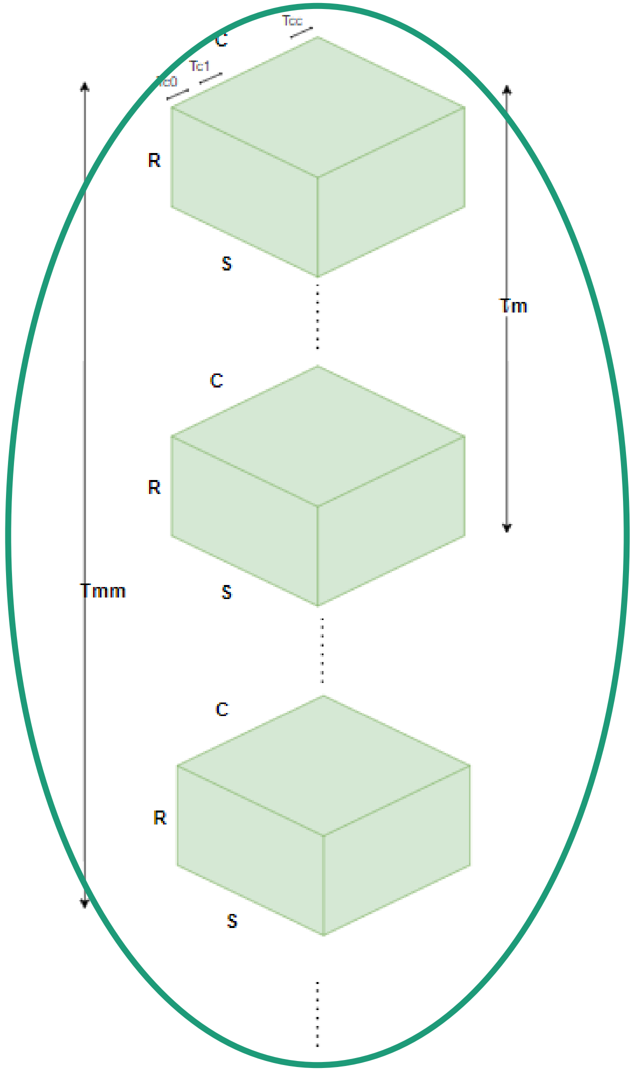
Data flow

- Wight SRAM tiling, tiling size R,S,C,Tmm.

```
for(int hh=0; hh<H; hh+=(Th)) begin
    for(int ww=0; ww<W; ww+=(Tw)) begin
        ifmap_sram = load_ifmap_sram(); // load ifmap chunk into ifmap SRAM

        for(int mmm=0; mmm<M; mmm+=Tmm) begin
            weight_sram = load_weight_sram(); // load weight chunk into weight SRAM

            for(int h=hh; h<hh+Th; h+=stride) begin
                for(int w=ww; w<ww+Tw; w+=stride) begin
                    ifmap_buffer = load_ifmap_buffer(); // load a sliding window of ifmap into buffer
                    mm_end = (M<Tmm) ? M : (Tmm+mmm);

                    for(int mm=mmm; mm < mm_end; mm+=Tm) begin
                        weight_buffer = load_weight_buffer(); // load a sliding window of weight into buffer

                        // NFU begins to do MAC operation
                        for(int r=0; r<R; r++) begin
                            for(int s=0; s<S; s++) begin
                                for(int i=0; i<entry; i++) begin
                                    // COMPUTATION
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```



C/Tc = entry

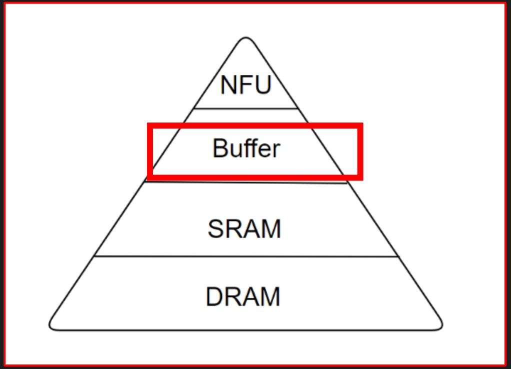


This is a chunk of weight

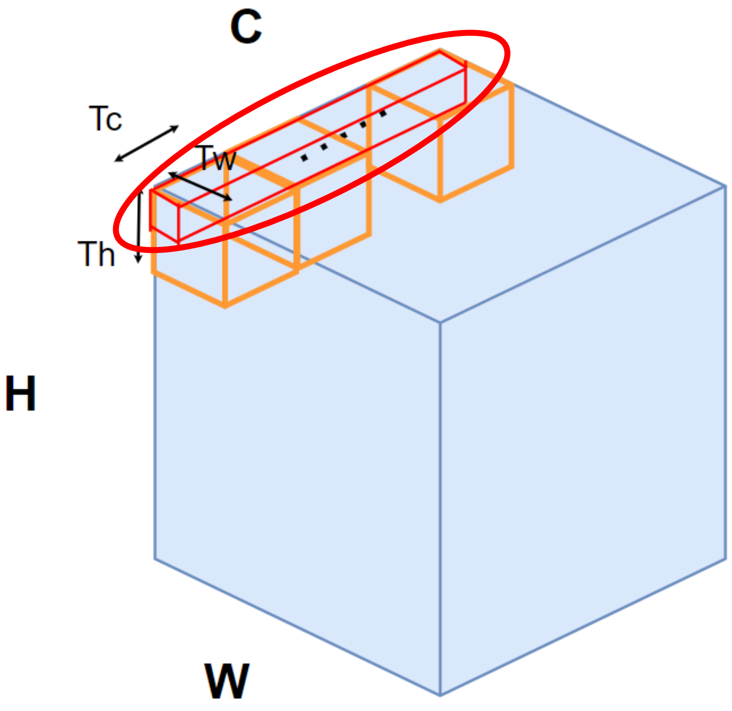
Data flow

- Ifmap buffer tiling, tiling size R,S,C.

```
for(int hh=0; hh<H; hh+=(Th)) begin
    for(int ww=0; ww<W; ww+=(Tw)) begin
        ifmap_sram = load_ifmap_sram(); // load ifmap chunk into ifmap SRAM

        for(int mmm=0; mmm<M; mmm+=Tmm) begin
            weight_sram = load_weight_sram(); // load weight chunk into weight SRAM

            for(int h=hh; h<hh+Th; h+=stride) begin
                for(int w=ww; w<ww+Tw; w+=stride) begin
                    ifmap_buffer = load_ifmap_buffer(); // load a sliding window of ifmap into buffer
                    mm_end = (M<Tmm) ? M : (Tmm+mmm);

                    for(int mm=mmm; mm < mm_end; mm+=Tm) begin
                        weight_buffer = load_weight_buffer(); // load a sliding window of weight into buffer

                        // NFU begins to do MAC operation
                        for(int r=0; r<R; r++) begin
                            for(int s=0; s<S; s++) begin
                                for(int i=0; i<entry; i++) begin
                                    // COMPUTATION
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```



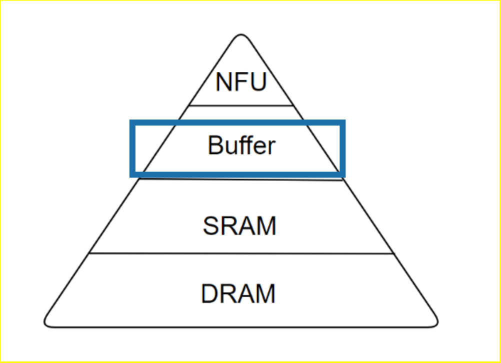
This is a sliding window in a ifmap chunk



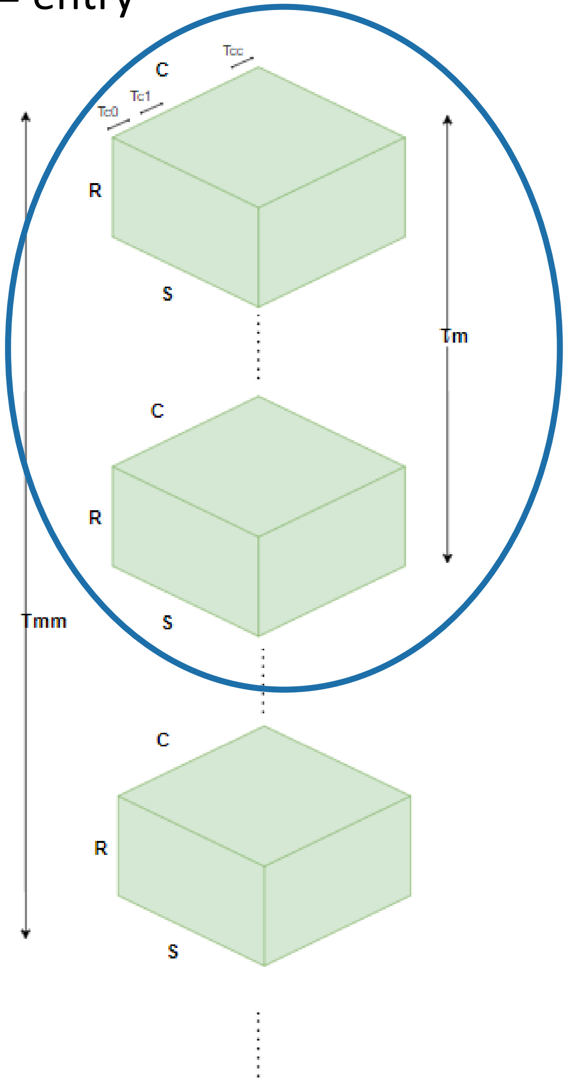
Data flow

- Wight buffer tiling, tiling size R,S,C,Tm.

```
for(int hh=0; hh<H; hh+=(Th)) begin
    for(int ww=0; ww<W; ww+=(Tw)) begin
        ifmap_sram = load_ifmap_sram(); // load ifmap chunk into ifmap SRAM

        for(int mmm=0; mmm<M; mmm+=Tmm) begin
            weight_sram = load_weight_sram(); // load weight chunk into weight SRAM

            for(int h=hh; h<hh+Th; h+=stride) begin
                for(int w=ww; w<ww+Tw; w+=stride) begin
                    ifmap_buffer = load_ifmap_buffer(); // load a sliding window of ifmap into buffer
                    mm_end = (M<Tmm) ? M : (Tmm+mmm);

                    for(int mm=mmm; mm < mm_end; mm+=Tm) begin
                        weight_buffer = load_weight_buffer(); // load a sliding window of weight into buffer

                        // NFU begins to do MAC operation
                        for(int r=0; r<R; r++) begin
                            for(int s=0; s<S; s++) begin
                                for(int i=0; i<entry; i++) begin
                                    // COMPUTATION
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```



C/Tc = entry



This is a sliding window in a weight chunk

Data flow

- Loop nest with tiling.

```
for(int hh=0; hh<H; hh+=(Th)) begin
    for(int ww=0; ww<W; ww+=(Tw)) begin
        ifmap_sram = load_ifmap_sram(); // load ifmap chunk into ifmap SRAM    1

        for(int mmm=0; mmm<M; mmm+=Tmm) begin
            weight_sram = load_weight_sram(); // load weight chunk into weight SRAM    2

            for(int h=hh; h<hh+Th; h+=stride) begin
                for(int w=ww; w<ww+Tw; w+=stride) begin
                    ifmap_buffer = load_ifmap_buffer(); // load a sliding window of ifmap into buffer    3
                    mm_end = (M<Tmm) ? M : (Tmm+mmm);

                    for(int mm=mmm; mm < mm_end; mm+=Tm) begin
                        weight_buffer = load_weight_buffer(); // load a sliding window of weight into buffer
                                                                                                  4
                        // NFU begins to do MAC operation
                        for(int r=0; r<R; r++) begin
                            for(int s=0; s<S; s++) begin
                                for(int i=0; i<entry; i++) begin
                                    // COMPUTATION
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

Partial sums are accumulated in a sliding window
A sliding window contains whole C dimension.

The operation changing frequency from large to small is 4,3,2,1. Therefore, once ifmap chunk is loaded into SRAM, it will not move until it has done all the computation with M filters.

C/Tc = entry

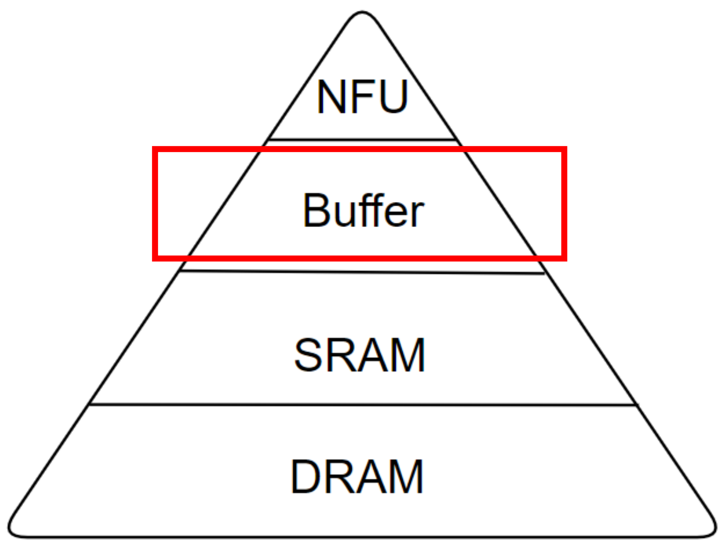
Data flow

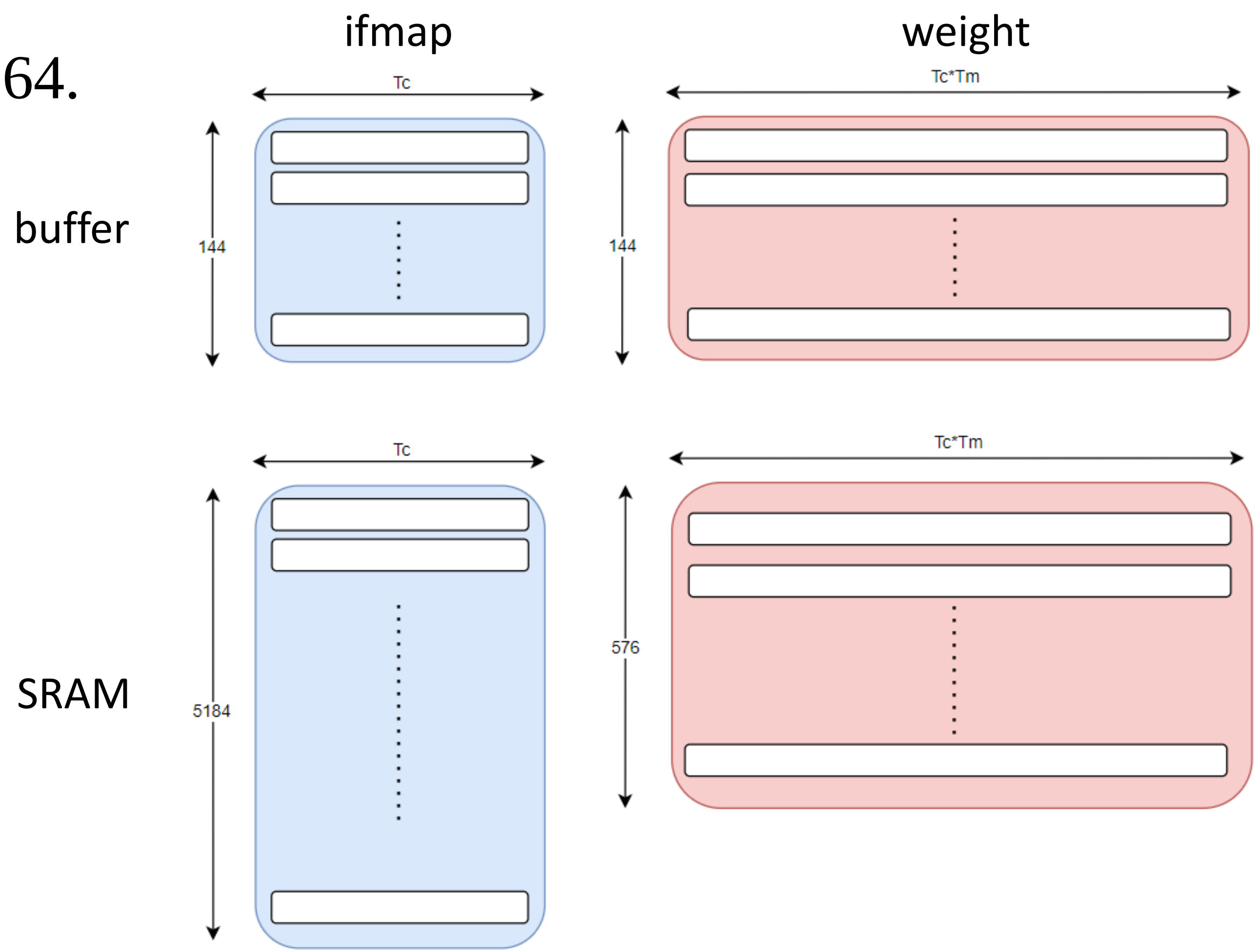
- Storage unit size(total=2.25+81+36+144KB = 263.25KB)
  - Select Tm = 16, Tc = 16.
  - Th = Tw =16. Tmm = 64.

Storage Unit	Ifmap	Filter(MemGen)	Ofmap
Buffer	R*S*C 3*3*256=9/4KB Macro:128 (bits/byte)*144(words) #Macro = 1	R*S*Tm*C 3*3*16*256=36KB Macro: 128(bits/byte)*144(words) #Macros = 16	Tm*(Th-2)+Tm*2=288 bytes (16*16+16+16)
SRAM	Th*Tw*C 18*18*256=81KB Macro:128byte(bits/byte)*5184(words) #Macro = 2	R*S*Tmm*C 3*3*64*256=144KB Macro: 16byte*(C/Tc)*(Tmm/Tm)*R*S =128byte(bits/byte)*576(words) #Macros = 16	
DRAM	H*W*C	R*S*M*C	E*F*M

Data flow

- Storage unit size(total=2.25+81+36+144KB = 263.25KB)
  - Select Tm = 16, Tc = 16.
  - Th = Tw =16. Th=Tw=16. Tmm = 64.

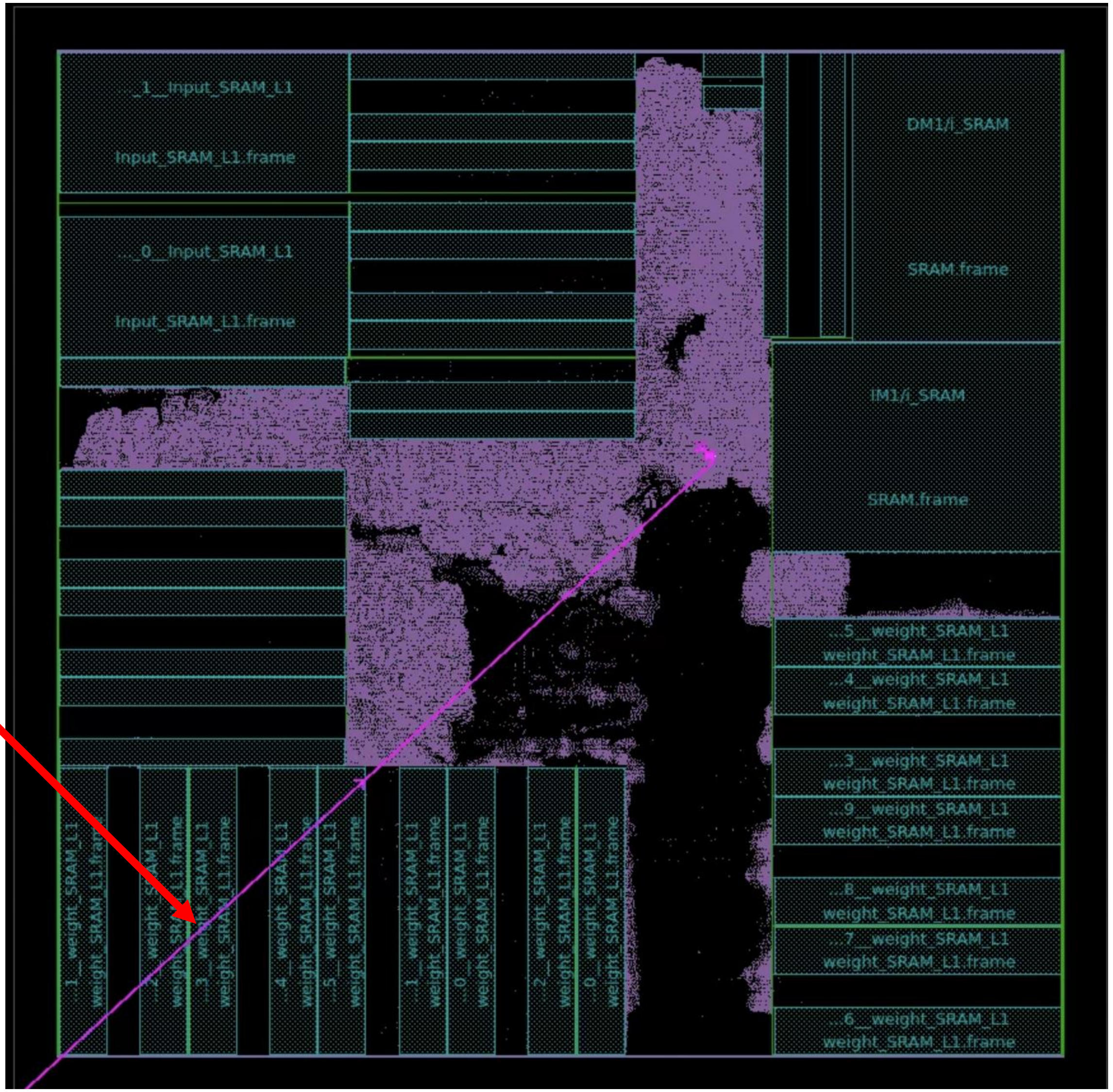




Simulation Results

- RTL
- Synthesis
- APR

critical path



Result