Lab6
Simple Convolution system

Speaker: Eric

Instructor: Lih-Yih Chiou

Date: 2021/03/31

EE
IPHPLAB VLSI Design LAB

A simple convolution system

- Purpose: do a 3 × 3 convolution and pooling

256 × 256
Padding = 1
-> 258 × 258

*

-1	-1	-1
0	0	0
1	1	1

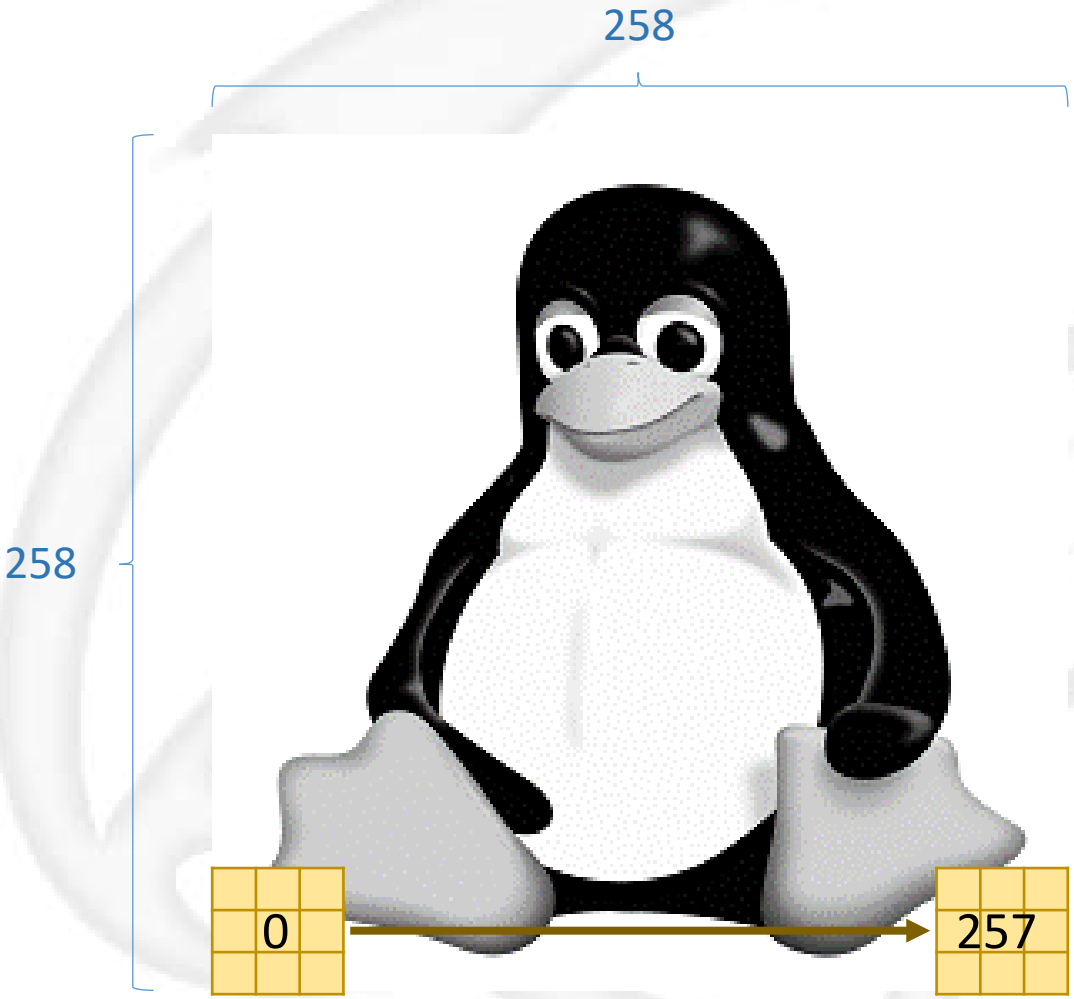
3 × 3

=

256 × 256

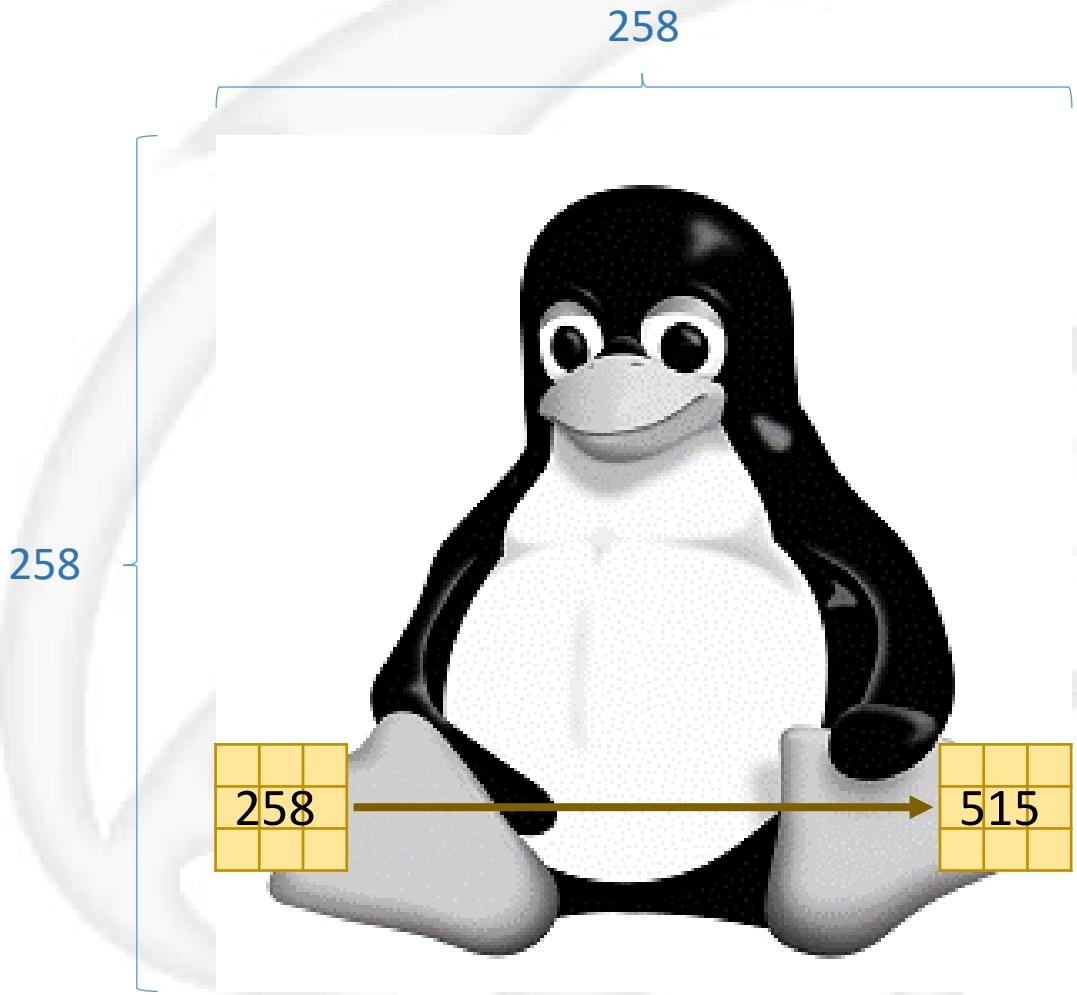
A simple convolution system

- Review: convolution process



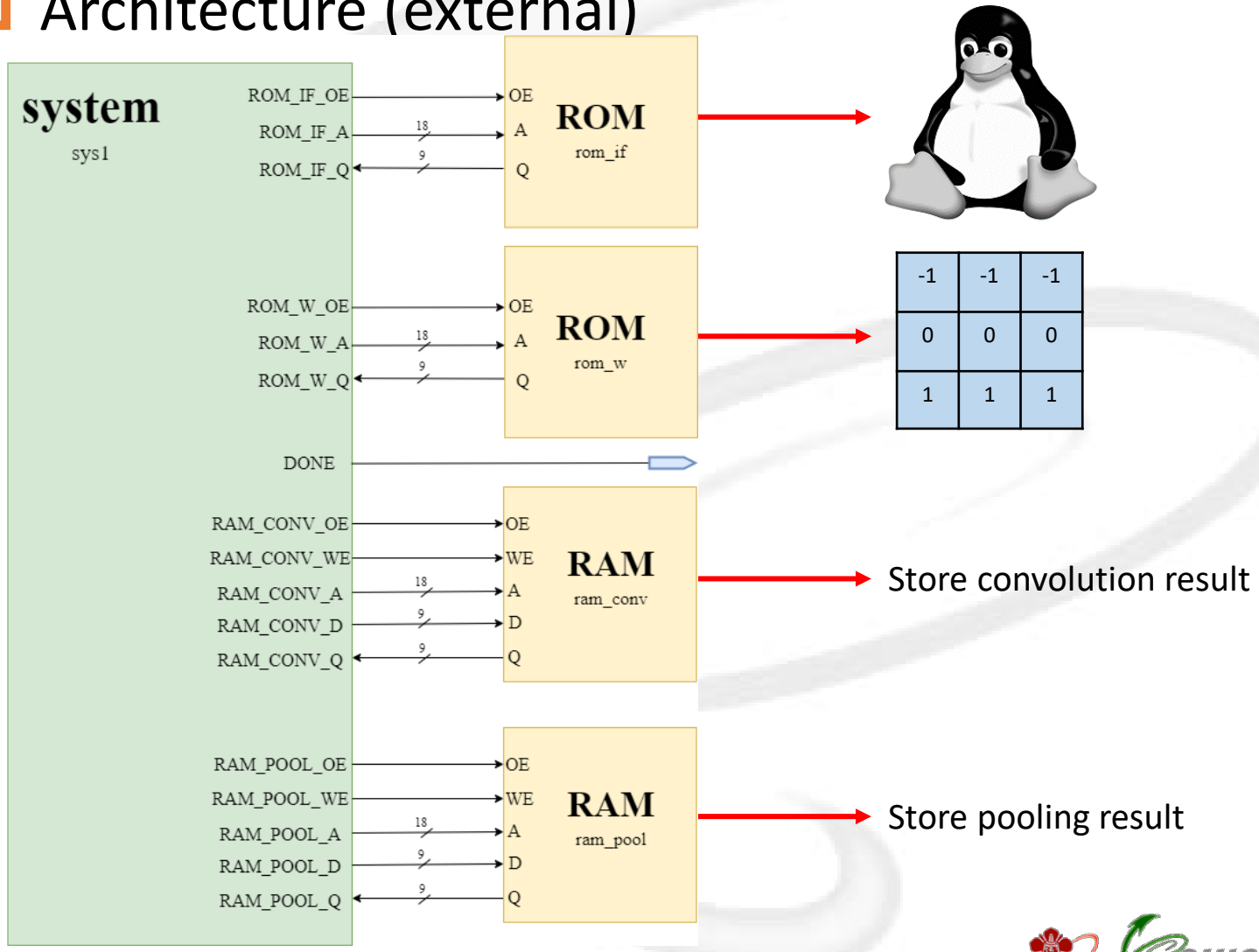
A simple convolution system

- Review: convolution process



A simple convolution system

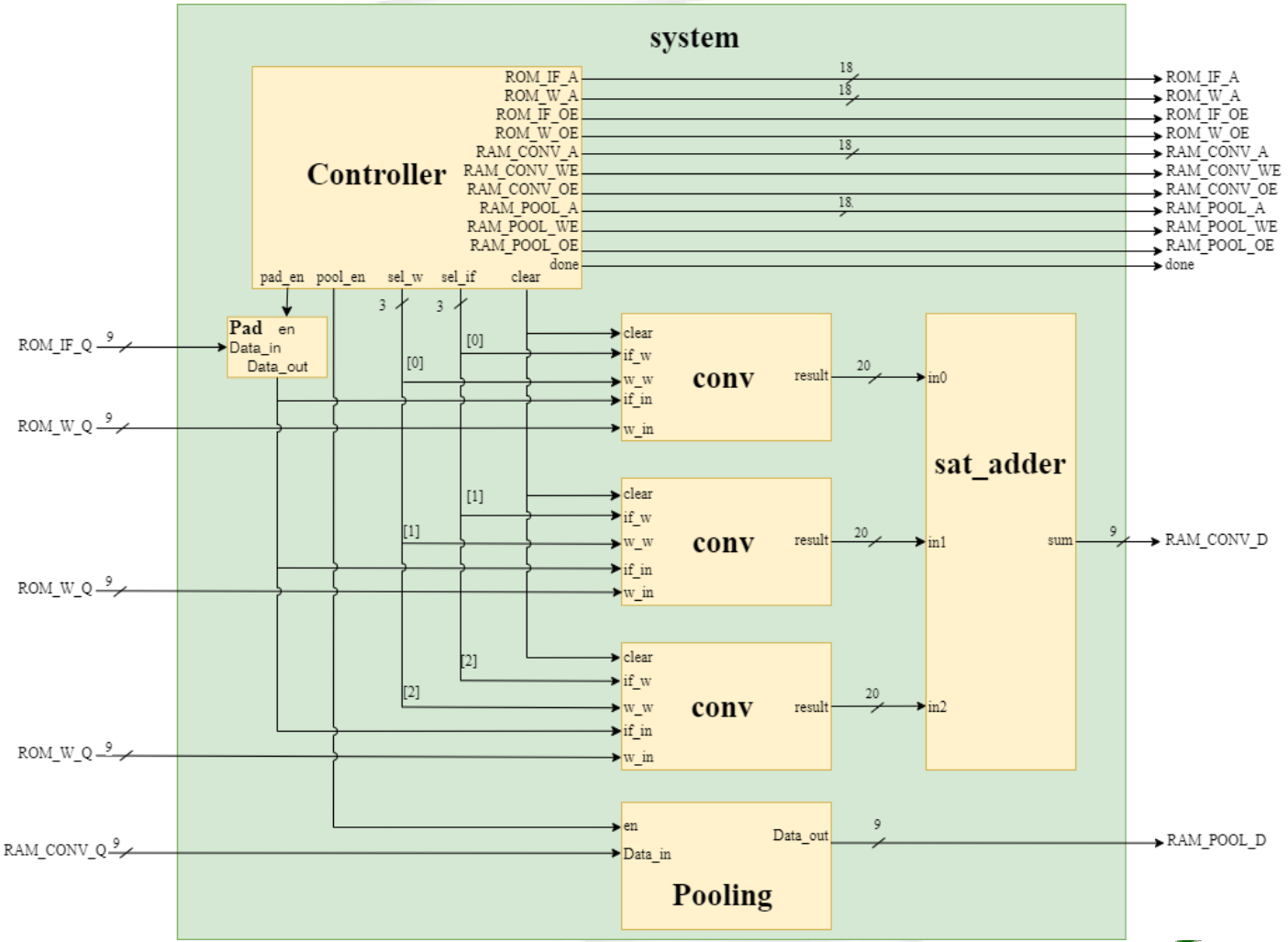
- Architecture (external)



*Clock pin and reset pin is ignored in this graph

A simple convolution system

- Architecture (internal)



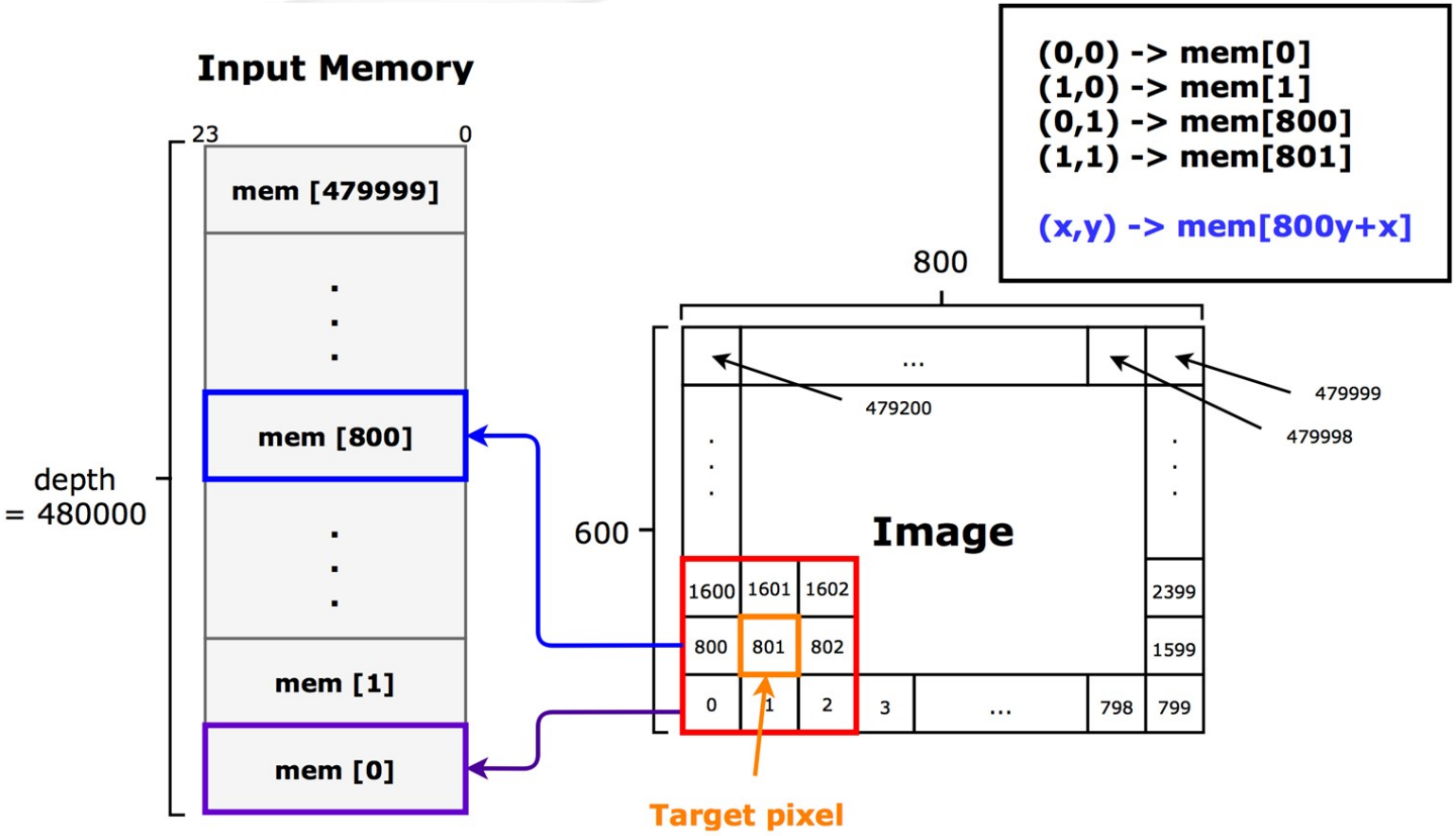
*Clock pin and reset pin is ignored in this graph

A simple convolution system

- Controller:
  - Control the address, enable signal of memories
  - Control the write enable and clear signal of convolution module
  - Control the enable signal of pooling module
  - Control the enable signal of padding module
  - If the process is finished, make done=1 to stop the simulation.
  - Make sure that your data has store in RAM.
- Conv:
  - Same as Lab5 mac module
  - There will be 3 Conv to complete a 3×3 convolution
- Sat_adder:
  - Add 3 input data. If $\begin{cases} sum > 255, & output\ 255 \\ sum < 0, & output\ 0 \\ else, & output\ sum \end{cases}$
- Pad
  - Padding the input feature map
- Pooling:
  - Do maximum pooling to convolution result
- ROM:
  - rom_if: Store the input feature map
  - rom_w: Store the weight data
- RAM:
  - ram_conv: Store conv result
  - ram_pool: Store pooling result

Image Format

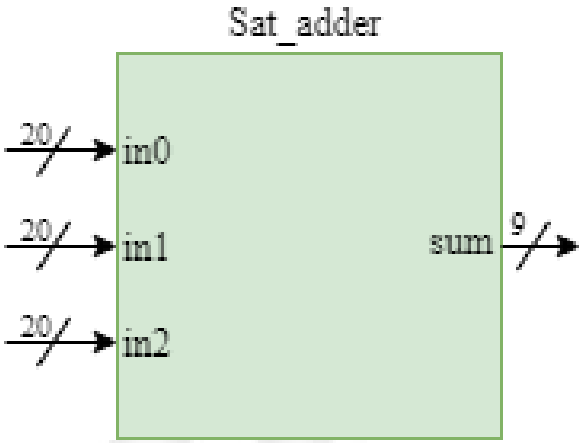
- Image(Here we take 800×600 picture for example)



VLSI Design LAB

Hardware Architecture – Sat_adder

- Add three convolution results
- Do activation function “ReLU”
- Since the data precision, consider the saturation.



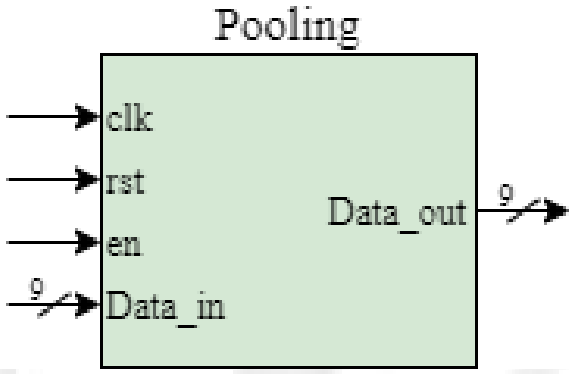
Port	Description
in0	Conv0 result
in1	Conv1 result
in2	Conv2 result
sum	Adder result

$$\begin{cases} sum > 255, & output\ 255 \\ sum < 0, & output\ 0 \\ else, & output\ sum \end{cases}$$

VLSI Design LAB

Hardware Architecture – Pooling

- Do max pooling operation. (signed arithmetic)
- Read 4 convolution results from ram_conv, then write the maximal value back to ram_pool.



Port	Description
clk	Clock signal
rst	Reset signal
en	Max pooling enable
Data_in	Data input
Data_out	Data output

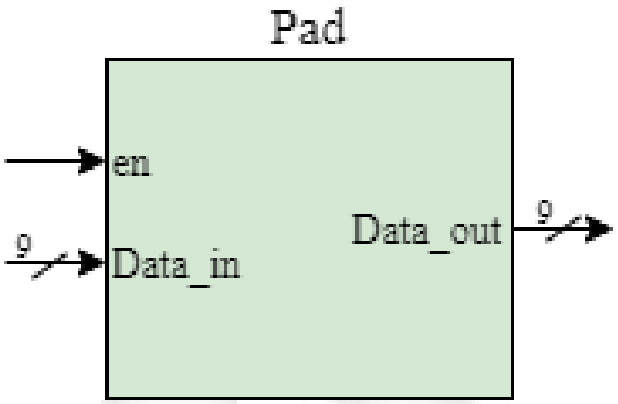
Stride = 2

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

VLSI Design LAB

Hardware Architecture – Pad

- Padding operation
- If en is high, the output is 9'h0ff, otherwise the output is input data.
- Set the enable signal in controller.

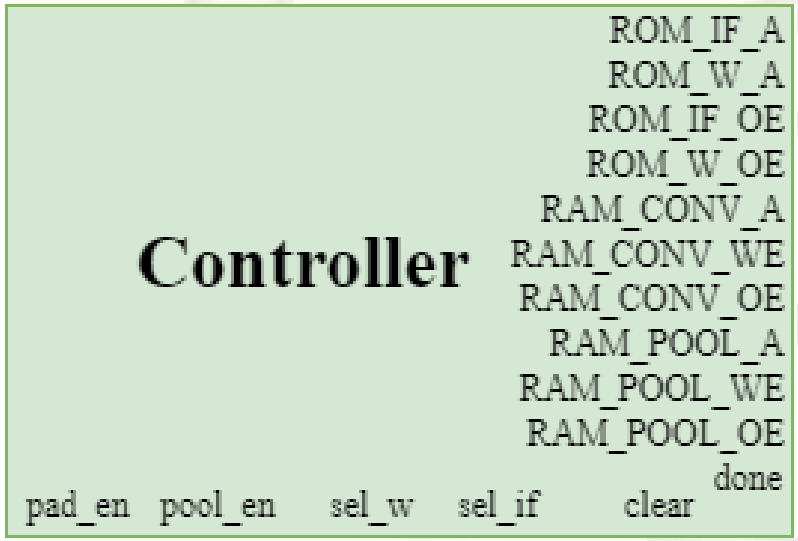


Port	Description
en	Enable signal
Data_in	Data input
Data_out	Data output

VLSI Design LAB

Hardware Architecture – Controller

- Control the system working procedure
- Use FSM to control



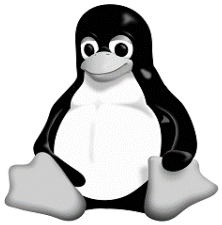
Port	Description
ROM_IF_A	IFM ROM address
ROM_IF_OE	IFM ROM read enable
ROM_W_A	Weight ROM address
ROM_W_OE	Weight ROM read enable
RAM_CONV_A	Conv RAM address
RAM_CONV_WE	Conv RAM write enable
RAM_CONV_OE	Conv RAM read enable
RAM_POOL_A	Pool RAM address
RAM_POOL_WE	Pool RAM write enable
RAM_POOL_OE	Pool RAM read enable
pad_en	Padding enable
pool_en	Pooling enable
Sel_w	Conv weight selection
Sel_if	Conv IFM data selection
clear	Conv block clear signal
done	System done

VLSI Design LAB

Hardware Architecture – Memory

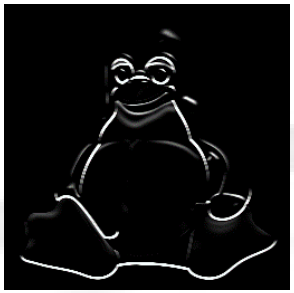
- Memory data storage

Input feature map
$0 \sim (256 \times 256 - 1)$



rom_if

Convolution result
$0 \sim (256 \times 256 - 1)$



ram_conv

Weight data:
$0 \sim 8$

[6]	[7]	[8]
[3]	[4]	[5]
[0]	[1]	[2]

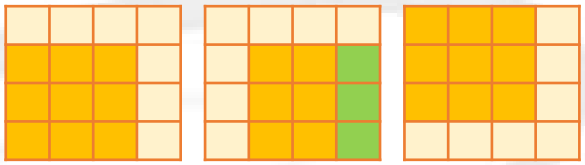
rom_w

Pooling result
$0 \sim (128 \times 128 - 1)$

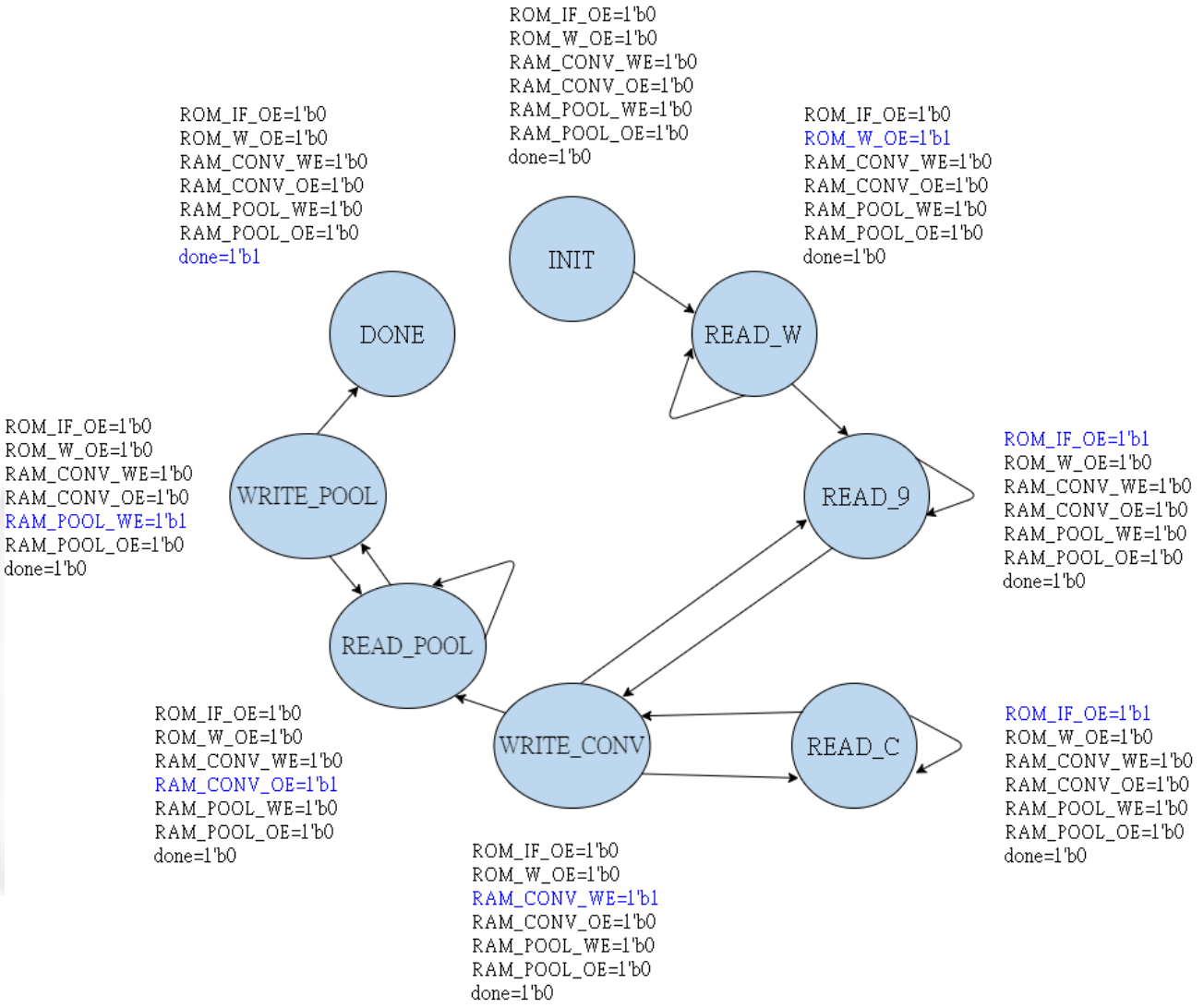
ram_pool

Hardware Architecture – Controller (1/5)

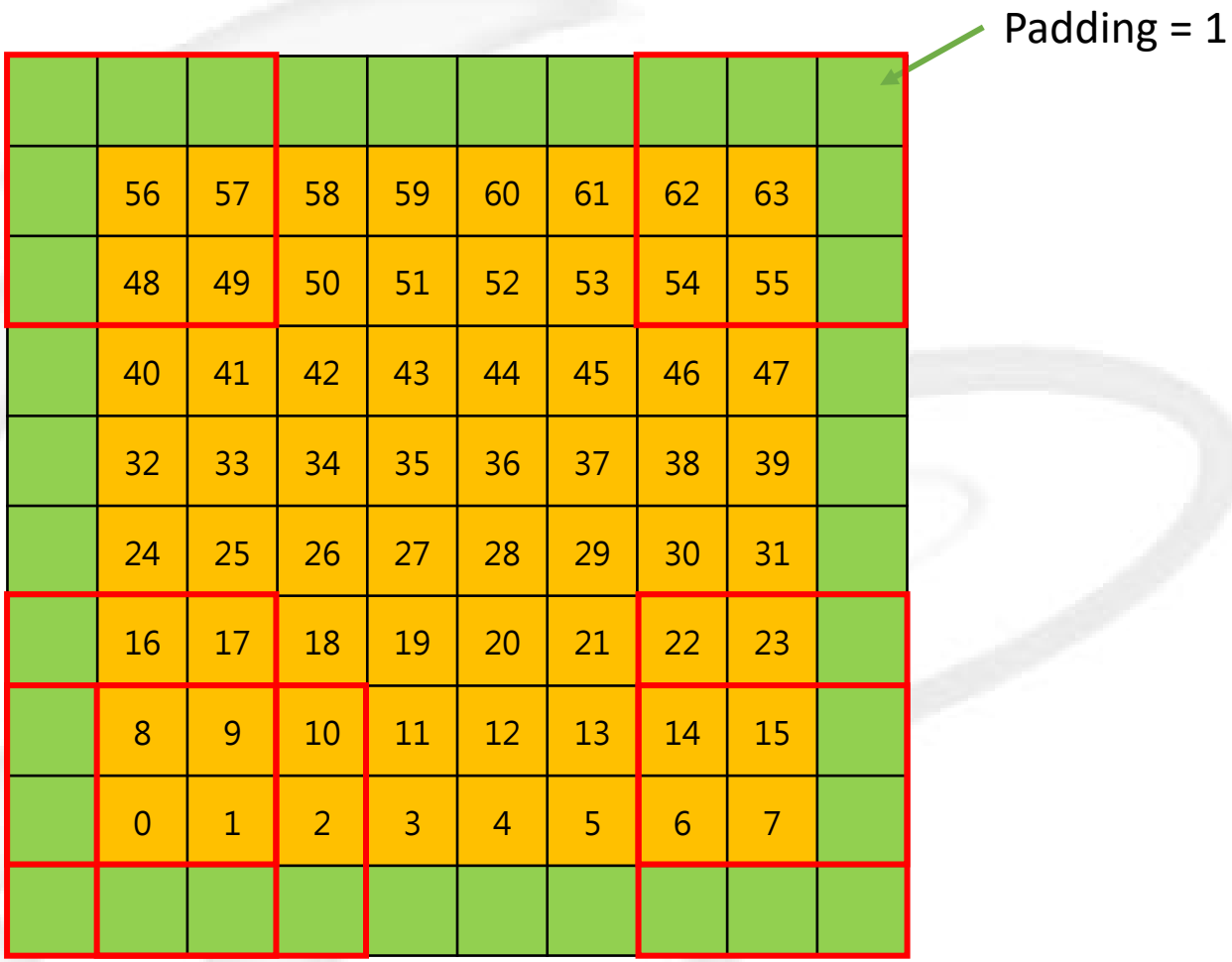
- All operations initiated at the positive edge trigger
- Control state:
  - INIT
    - Initial state
  - READ_W
    - Read 9 weights from ROM
  - READ_9
    - Read 9 input pixels for next row
  - READ_C
    - Read next 3 input pixels for the same row
  - WRITE_CONV
    - Write convolution result to CONV RAM
  - READ_POOL
    - Read 4 convolution results for pooling operation.
  - WRITE_POOL
    - Write maximum pooling result to POOL RAM
  - Done
    - Finish and stop the process



Hardware Architecture – Controller (2/5)



Hardware Architecture – Controller (3/5)



Hardware Architecture – Controller (4/5)

- For states with multiple cycles, you can…

```
always@(posedge clk or posedge rst)begin
if(rst)begin
    counter<=4'd0;
end
else if(state==S1)begin
    counter<=counter+4'd1;
end
end
```

```
always@(*)begin
case(state)
S1:begin
    if(counter==4'd10)
        n_state=S2;
    else
        n_state=S1;
    end
S2:begin
    n_state=S2;
    end
endcase
end
```

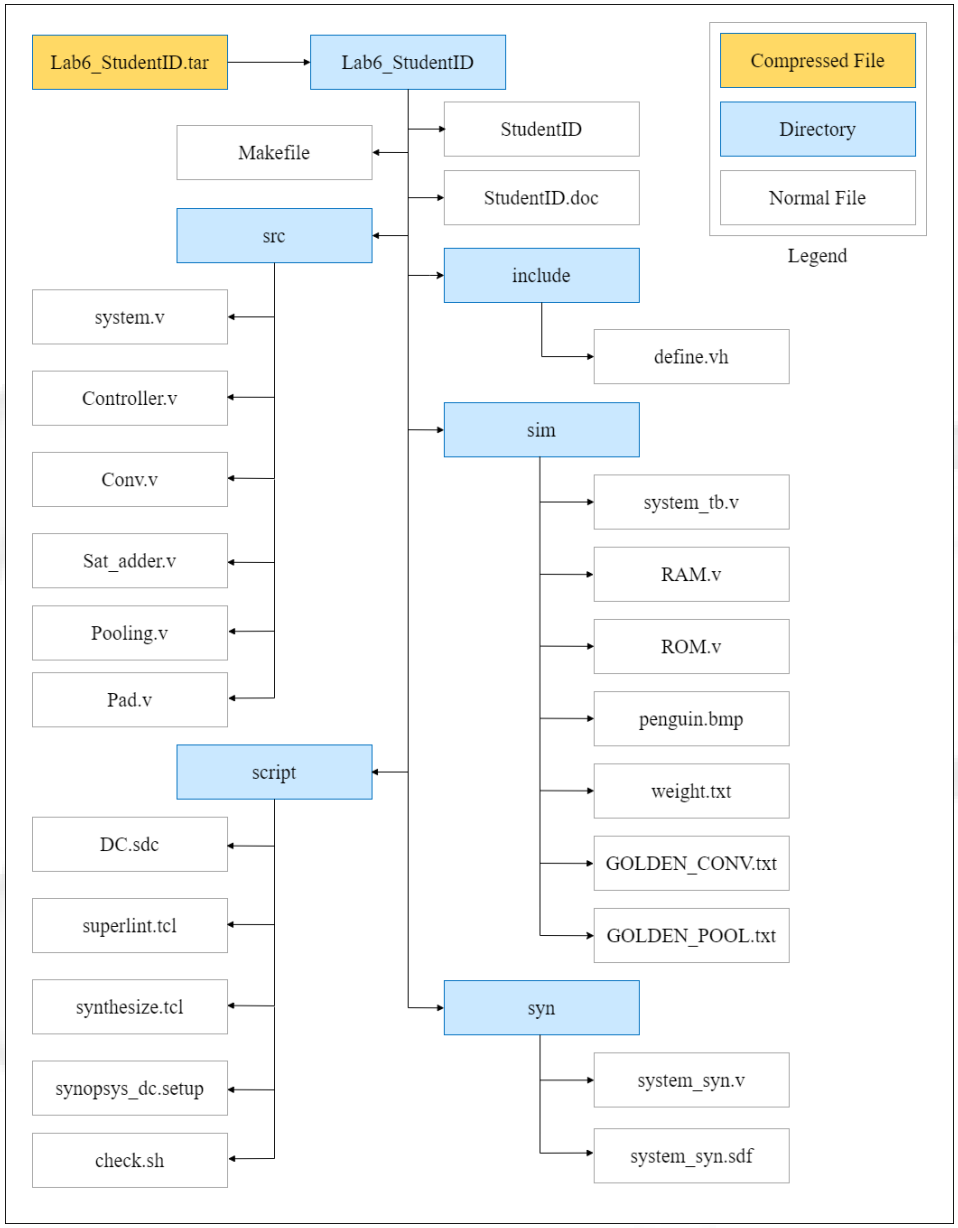
Hardware Architecture – Controller (5/5)

- Use the shift register convolution module in Lab5.
- Consider the boundary condition to handle the padding problem.
- Control the ROM address to give the right data to the convolution modules.
- The ROM has only “1” output data port. Therefore, you should distribute the data to three convolution modules, respectively, by controlling selection signals
- Testbench will output your image in RAM.
- You can design your own states.

Lab6 (1/4)

- Due day: 2021-05-05(Wed), 15:00
- Remark
  1. All file should be named in English.
  2. Use Makefile command to compress the file
  3. Please upload the Lab6_StudentID.tar to Moodle,
     EX: Lab6_E240xxxxx.tar

Lab6 (2/4) - File hierarchy



Lab6 (3/4) - Grading

- Pre-sim – 45%
  - Explain how the FSM & Padding works
  - Explain the system operations with waveforms
  - Get all credits with all results pass, or some credits will be lost.
- Post-sim – 30%
  - Get all credits with all results pass, or some credits will be lost.
  - Faster simulation time, more credits get.
- Superlint – 10%
  - Show the Superlint coverage of your code
- Demo – 15%
  - Demo the Pre-sim/Post-sim procedure to TAs
  - Depends on your realization about this system

Lab6 (4/4) - Makefile command

Description	Command
Run RTL Convolution simulation	make rtl0
Run RTL Pooling simulation	make rtl1
Run RTL simulation	make rtl_full
Run post-synthesis simulation	make syn_full
Dump waveform (no array)	make {rtlX, syn_full} FSDB=1
Dump waveform (with array)	make {rtlX, syn_full} FSDB=2
Open nWave without file pollution	make nWave
Open Superlint without file pollution	make superlint
Open DesignVision without file pollution	make dv
Synthesize your RTL code	make synthesize
Check correctness of your file structure	make check
Compress your homework to tar format	make tar
Count the total lines of your code	wc –l ./src/* ./include/*