NYCU-EE ICLAB – Spring 2022

Midterm Project

Design: Time of Flight (TOF)

Data Preparation

1. Extract test data from TA's directory:

   % tar xvf ~iclabta01/Midterm_Project.tar

Design Description

■ ***Time of Flight (ToF)***

Time of Flight (ToF) is an important technique for distance measurement.

ToF measures the time taken by the light to travel back and forth between the target and the sensor.

A single photon detector can detect the photon and output a digital pulse when the photon hit its active region.

Suppose we have a single photon array, we can construct a 3D picture (4*4*255) by calculating each pixel's distance

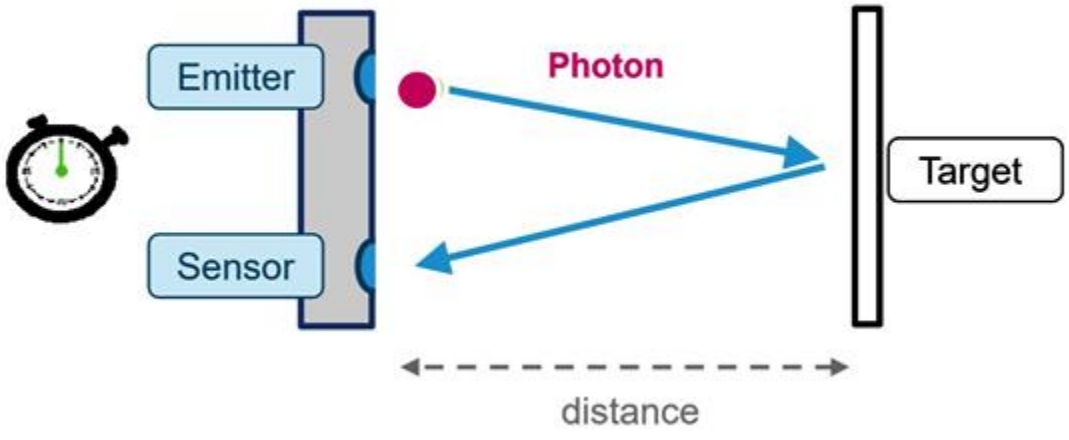


Fig 1. ToF distance measurement

(Ref. https://www.eettaiwan.com/20200505nt61-the-future-of-times-of-filight/)

■ *Project Description*

In this project, we are going to implement an ToF chip. The goal is to calculate how many cycles have passed from the laser (start) to the sensor (stop), calculate the distance of the object, and store it in DRAM.

In real world, there will be background noise, so we cannot calculate the distance using just one start and stop.

To have more accurate result, we will using multiple start and stop to calculate the actual distance. Below is an example of histogram generation. We will get the histogram and calculate the distance.

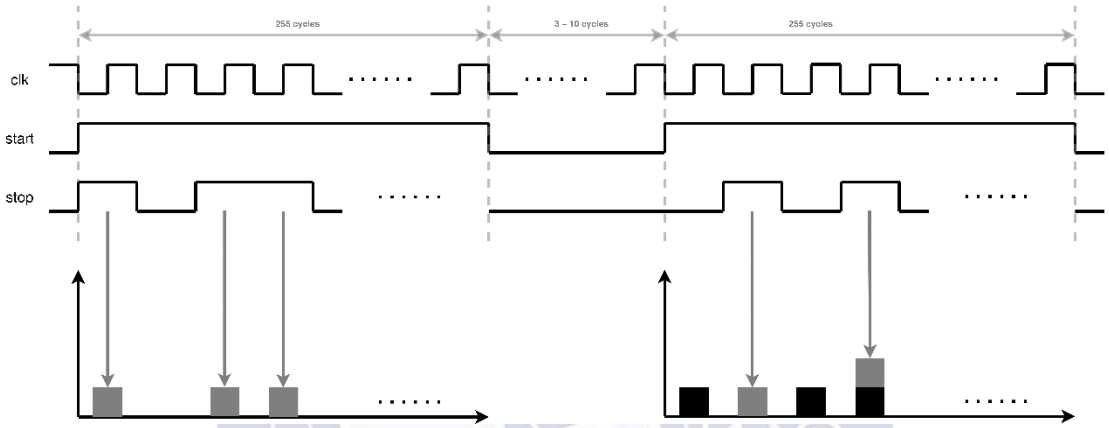


Fig 2. Input waveform (mode 0) & histogram example

They may be only a few starts in one measurement. Instead of finding peak in the histogram, we use a method called “window” to calculate the densest region in the histogram instead of finding the peak in histogram. (See Fig 3.)

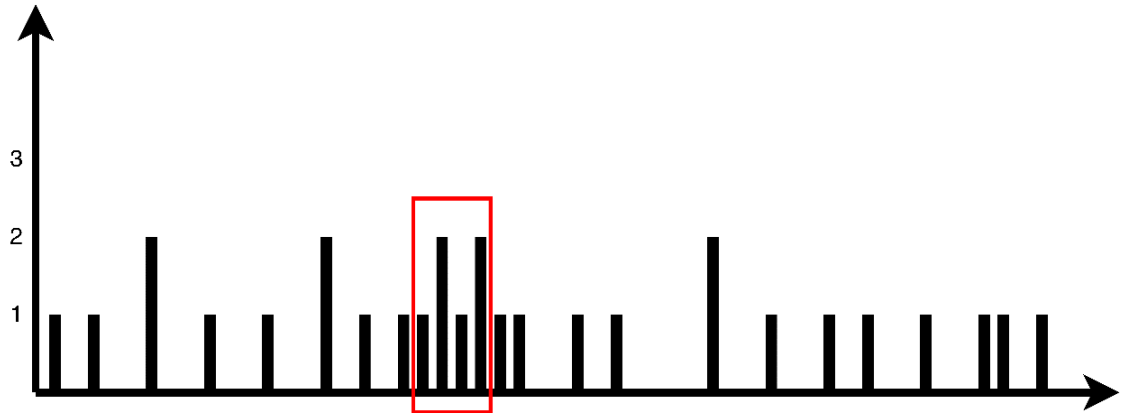
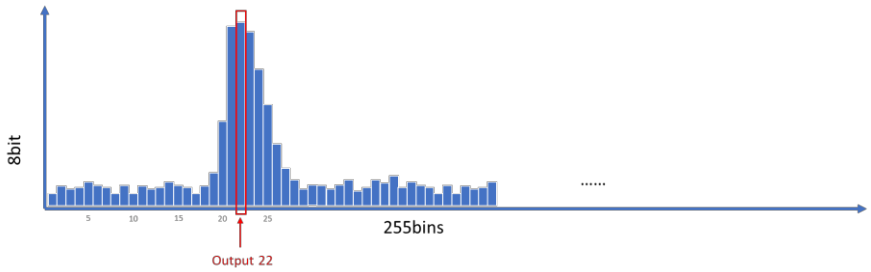


Fig 3. The reason why we need window method (window = 4)

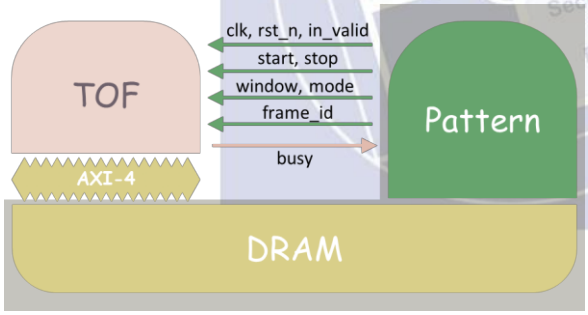
Window method finds N adjacent bins that have maximum sum, and output the first bins location. In this exercise, if multiple window results have the same height, output the first result



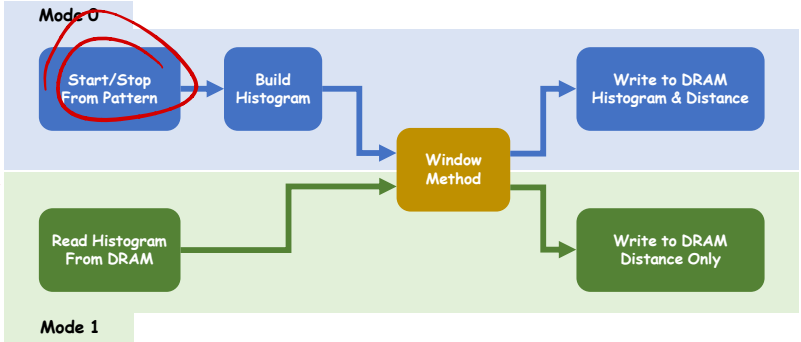
Pic2. Histogram & window method (window = 2'b00 => window = 1)

Pic3. window = 2'b11 => window = 8

System Architecture

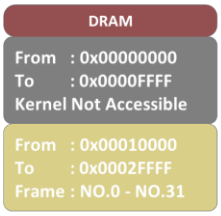


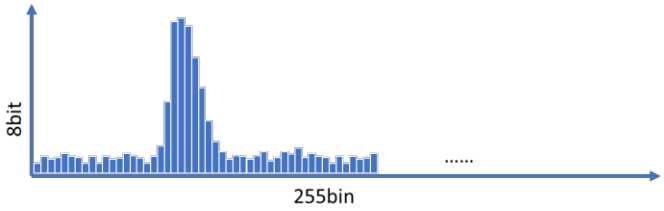
First, pattern will give the **frame_id**, **window** and **mode** in the first cycle when **in_valid** is high. Then, your chip should fetch frame data stored in DRAM via AXI-4 if mode = 1, or generate the histogram from input if mode = 0. While your design is dealing with current task, **busy** should be high. After all processing is done, and all data is written back to DRAM, **busy** should be pull low so that pattern could check the answer in DRAM and output cost is consistent with golden one. On the other hand, pattern can directly access DRAM for data checking and overwriting without any constrain. The DRAM behavior model (pseudo_DRAM.v) is provided by TA.



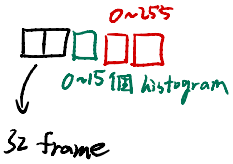
Memory Mapping

- ***Data in DRAM***





0x00010000-0x000100FE: Histogram
0x000100FF: Distant for that pixel
0x00010?XX: 16 Histograms form 1 frame
0x0001?XXX: 16 Frames
0x0002?XXX: another 16 Frames



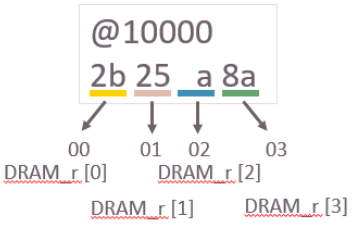
- ***Mapping Example***

../00_TESTBED/DRAM/DRAM.dat

```
@10000
2b 25 a 8a
@10004
2b a8 29 34
@10008
bf 8a 5 79
@1000c
6d c5 29 a
@10010
73 a7 a 94
@10014
85 62 42 a4
@10018
ea dd 80 8b
@1001c
26 45 4a 1c
```

Variable in pseudo_DRAM.v

reg [7:0] DRAM_r [0:196607]; (Address from 00000000 to 0002FFFF)



DRAM_r		
Address	[7:4]	[3:0]
[0]	2	B
[1]	2	5
[2]	0	A
[3]	8	A
[4]	2	B
[5]	A	8
[6]	2	9
[7]	3	4

- ***Fetch DRAM***

Given **frame_id** = 5, that means your need to fetch DRAM 0x0001_5000 ~ 0x0001_5FFE with AXI4 protocol.

Note that Memory Address is Byte Offset as shown in above example. Each address is mapped to single byte.

Please refer to NOTE_2022_SPRING for more detail information of DRAM.

You should **generate the SRAM** to store the histogram from input or DRAM. TA will check if you truly use memory to store the histogram, and if **NOT**, you will **Fail** the demo.

Control Interface

Inputs

1. You will receive 1 number **frame_id** [4:0]. The frame_id will be the index when access the DRAM
2. Input signal **window, frame_id** and **mode** are valid in the first cycle when **in_valid** is high, otherwise would be unknown value.
3. Input signal **start** and **stop** are valid when **in_valid** is high, otherwise would be unknown value.

1. **Start** will be high for 255 cycles, and there will be 4 ~ 255 **start** in one **in_valid**, to form an 8-bit height histogram.
2. There will be 3~10 cycles between each start and the first cycle of in_valid.

4. When stop is high, add one to the histogram's corresponding bin. That means stop can be high for consecutive cycles. (see Fig 5.)
5. There is **only 1 reset** before the first pattern, thus, your design must be able to reset automatically.
6. All inputs will be changed at clock **negative** edge.
7. The next input pattern will come in 3 ~ 10 cycles after **busy** falls.

Name	Width	Functional Description
rst_n	1	Asynchronous reset and active-low
clk	1	Clock for TOF chip
in_valid	1	When in_valid is high, all the other three input is valid.
start	1	Start counting cycle
stop	16	Calculate the time between start and stop and store the time in histogram, 16 stops are independent
window	2	2'b00: window = 1, find the highest bin in histogram 2'b01: window = 2, find 2 adjacent bins that have maximum sum 2'b10: window = 4, find 4 adjacent bins that have maximum sum 2'b11: window = 8, find 8 adjacent bins that have maximum sum
mode	1	1'b0: generate histogram from input 1'b1: read histogram from DRAM
frame_id	5	Index of frame in DRAM, range in No. 0 ~ No. 31

Outputs

1. The only output signal of the design is 1 bit busy.
2. In mode0, you should generate histogram from input, and write histogram and distance result to DRAM.
3. In mode1, you should read histogram from DRAM and calculate the distance, then store the distance back to DRAM.
4. After **busy is pulled low, pattern will check the correctness of that frame value inside DRAM.**
5. All outputs are synchronized at clock **positive** edge.
6. **busy** should be low after initial reset.
7. **busy** should not be raised when **in_valid** is high.
   The test pattern will check whether your data in DRAM is correct or not at the first clock **negative** edge after busy pulled low.

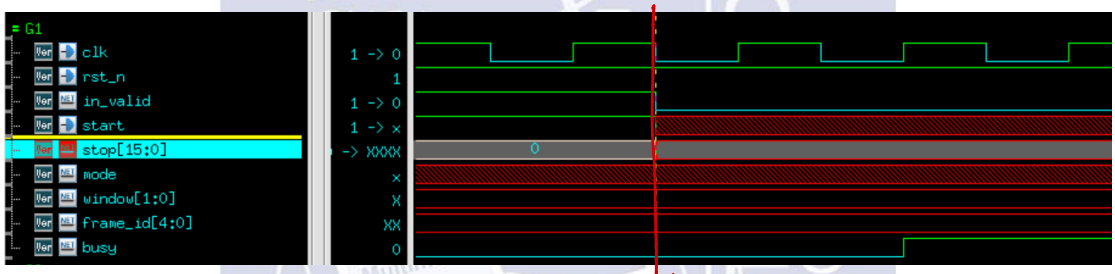


Fig 1. Busy pulls high after in_valid pull low

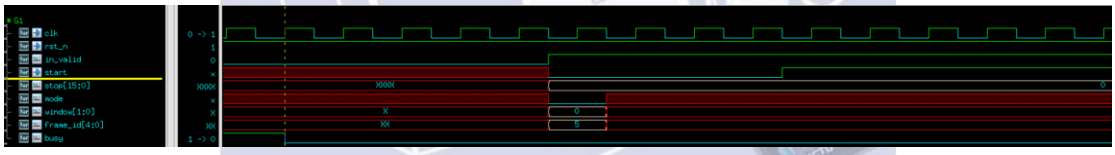


Fig 2. Busy pulls low after right data store in DRAM, and pattern gives new input after 3 ~ 10 cycles

AXI4 IO

Please refer to Midterm_Project_Pre-Released_2022_SPRING

Specifications

Top module

1. Top module name: **TOF** (File name: **TOF.v**)

Reset

2. It is **asynchronous** reset and **active-low** architecture.

Latency

3. The latency of your design in each pattern should not be larger than **1,000,000** cycles.
4. The maximum clock period is set to **20ns**, and **you can determine the clock period by yourself**.

Synthesis

5. The input delay and the output delay should be **half** of the clock period. For example, if clock period is 10ns, the input delay and the output delay will be 5ns.
6. The output loading is set to 0.05.
7. The synthesis result of data type cannot include any **LATCH** (check in syn.log).
8. The slack in the end of TOF.timing should be **non-negative** and the result should be **MET**.
9. In this project, you should modify the syn.tcl (e.g. link library for your memory.db) by yourself. compile_ultra will be used to synthesis. Since memories are used in this project, the syn.tcl in Lab05 may be a good reference one.
10. Note that maximum synthesis time could not be over 1 hours (Normally, this design would not over 1 hours, TA will give 1.5 hours while demo)

Supplement

11. All the data are in **unsigned** format.
12. No **ERROR** is allowed in every simulation/synthesis.

Grading Policy

1. The performance is determined by area, latency and clock period of your design. The smaller the performance index is, the higher score you can get.

   Score = Functionality (60%) + Performance (40%)

   Functionality (60%): Random test
   Performance (40%):
   Area * Latency * clock period = Area * total clock cycles * $(\text{clock period})^2$
   The clock period will affect the TOF resolution, so the clock period is squared when calculating the performance

2. Performance points can be obtained when Functionality pass
3. 2nd demo will get 30% from total score.

TA will demo your design with 2 different patterns, one is for functionality, the other is for performance.
For functionality, the DRAM latency is range from 1 to 10, not fixed.
For performance, the DRAM latency is 300 ~ 500.

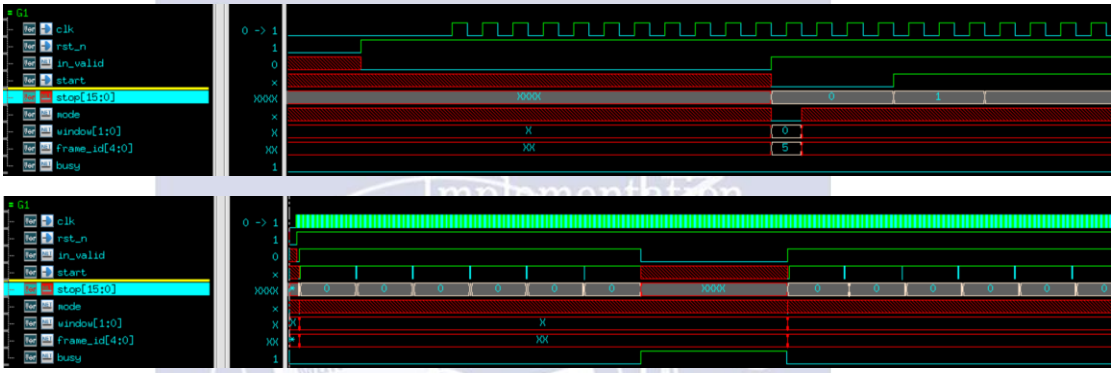
Note

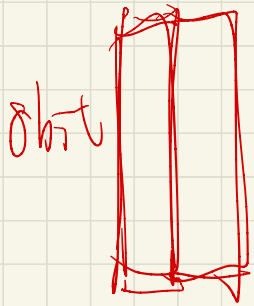
1. Please upload the following files on new e3 platform before **23:59**.
   Due Day: **1st Demo: 04/26** **2nd Demo: 05/03**
   RTL design: **TOF_iclab???**, (??? is your account number)
   Memory File: **MEMORY_NAME_iclab???.v**
   MEMORY_NAME_iclab???.db
   File List for Customized SRAM: **file_list_iclab???.f**
   Clock period: **clock_period_iclab???.txt** **(ex: 20.0_iclab099.txt)**
   Example:
   Given your memory name is RA1SH512, and your submitted memory file RA1SH512_iclab099.v
   Modified in file_list_iclab099.f
   ../04_MEM/RA1SH512.v

   Note that you can provide **multiple** memory specs in this lab.
   If the uploaded files violating the naming rule, you will get **5 deduct points**.
   If you omit any file, you will fail demo.
2. Template folders and reference commands:
   01_RTL/ (for RTL simulation) **./01_run**
   02_SYN/ (for Synthesis) **./01_run_dc**
   Remember modify.tcl to fit your memory db name (refer to Lab05)
   (Check the design which contains **Latch** and **Error** or not in **syn.log**)
   (Check the design's timing in /Report/TOF.timing to see if the slack is **MET**)
   03_GATE_SIM/ (Gate Level simulation) **./01_run**
   You can key in **./09_clean_up** to clear all log files and dump files in each folder
   04_MEM/ (Memory location)
   You should generate your memories and put the required files (.v and .db) here.

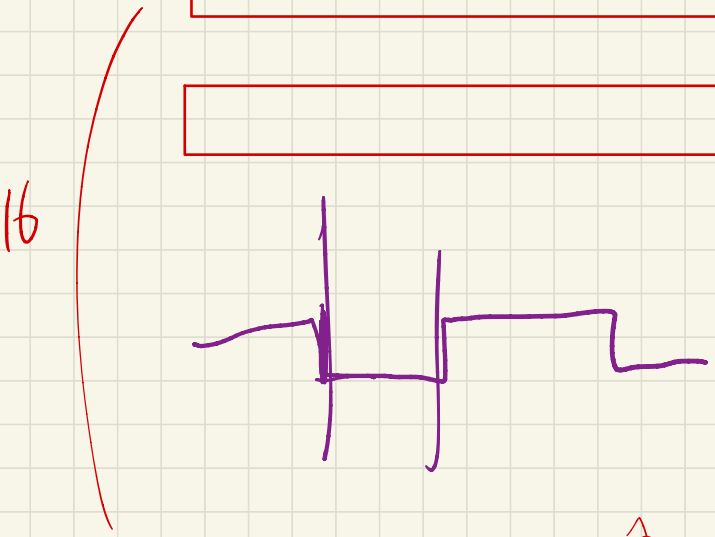
Hint

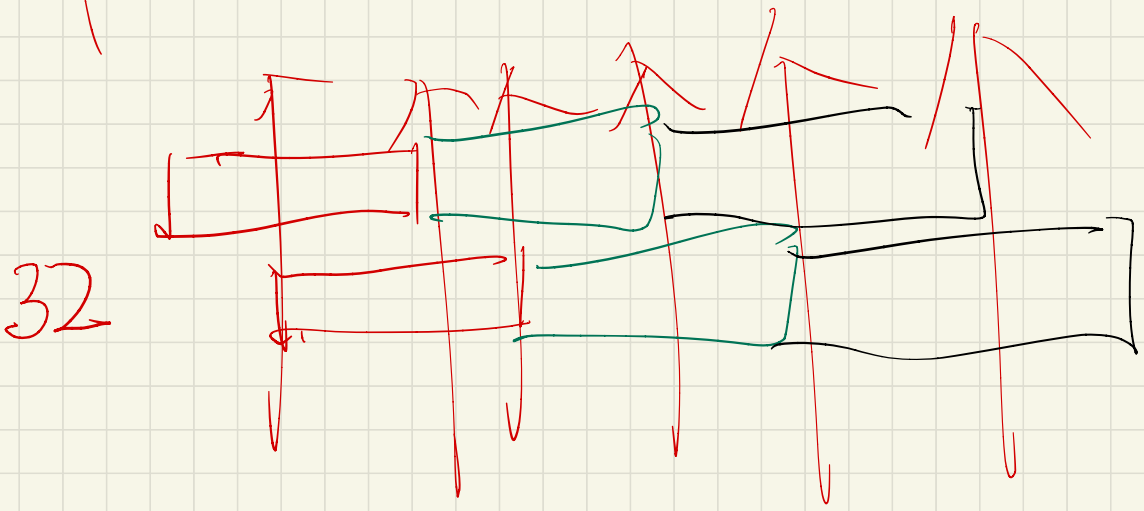
1. **Recommend development flow:**
   I. Establish a PATTERN for testing your circuit
   II. Ensure your design can fetch data from DRAM with AXI4
   III. Ensure your design can write back to DRAM with AXI4
   IV. Design the Memory usage of whole algorithm
   V. Design the Architecture of circuit and FSM for each part
   VI. Verify circuit functionality with more test case
   VII. Find out bottleneck and do some optimization
   VIII. Optimize latency – Algorithm and Architecture
   IX. Optimize area – Memory and Circuit Design
2. Some Waveforms

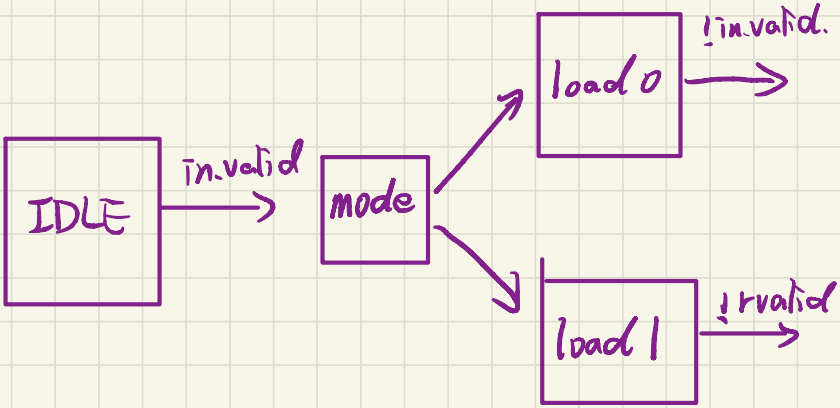



8bit

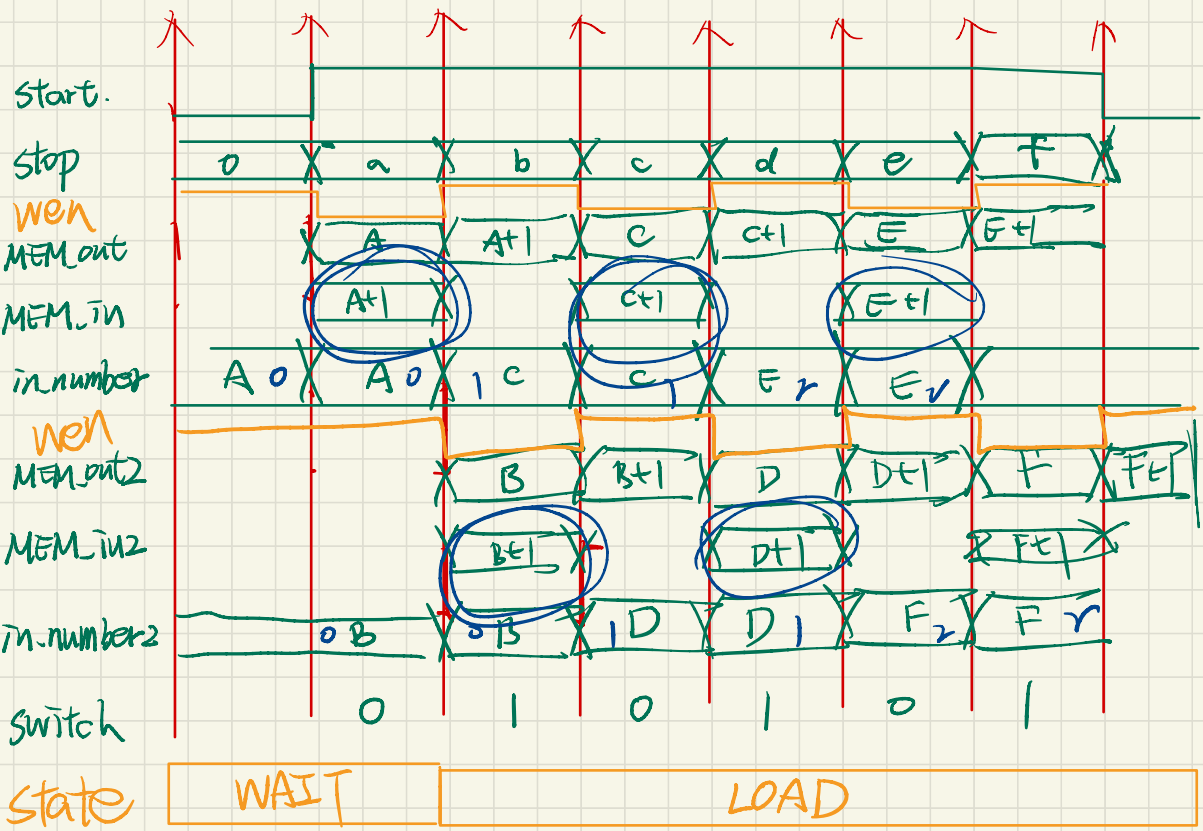

255


16

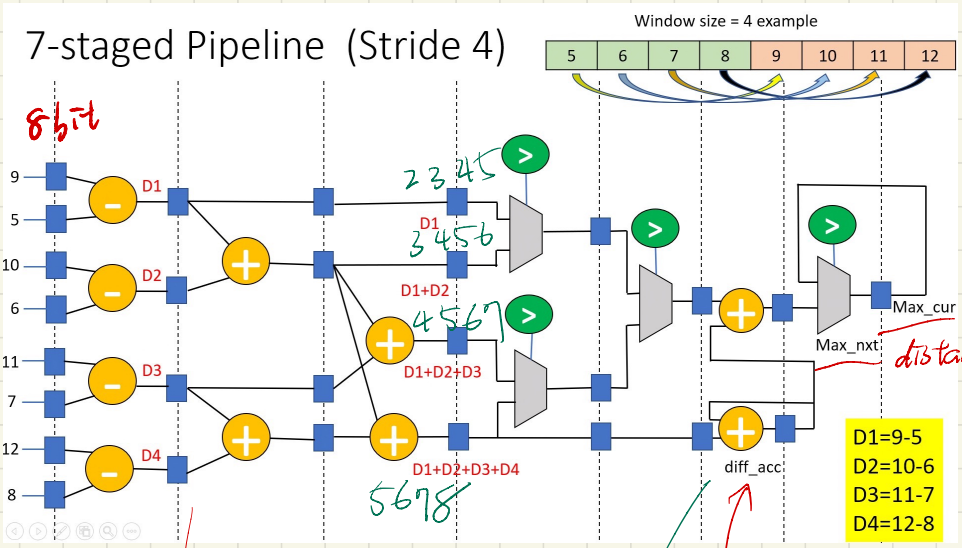

32


IDLE
in.valid
mode
load 0
!in.valid.
load 1
!rvalid

window 1 2 4 8



127:120 119:112
111:104 103:96
95:88 87:80
79:72 71:64
63:56 55:48
47:40 39:32
31:24 23:16
15:8 7:0



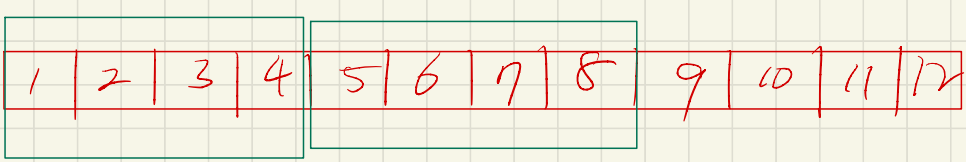
前值 2bit 3bit 4bit

initial (1+2+3+4)

next (5-1)+(6-2)

原本的1234 (+7-3)+(8-4)

=5678



1+2+3+4 → 5-1

2+3+4+5 → 6-2

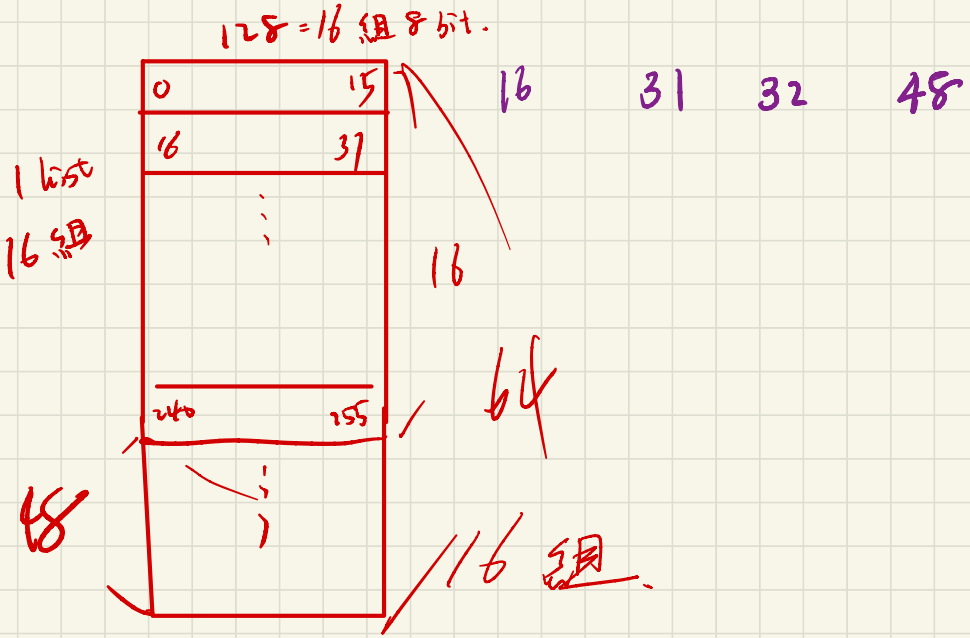
3+4+5+6 → 7-3

4+5+6+7 → 8-4

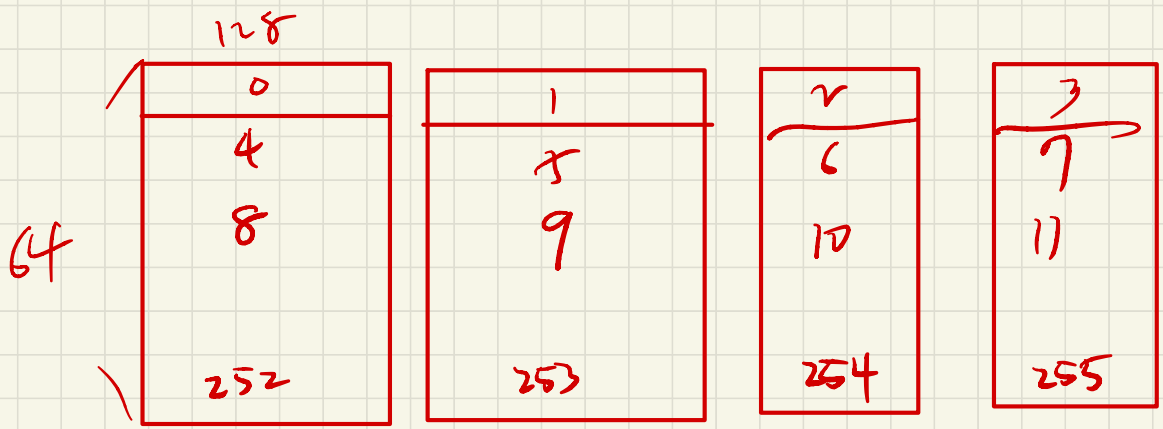
5+6+7+8

比視的較大

mode 1

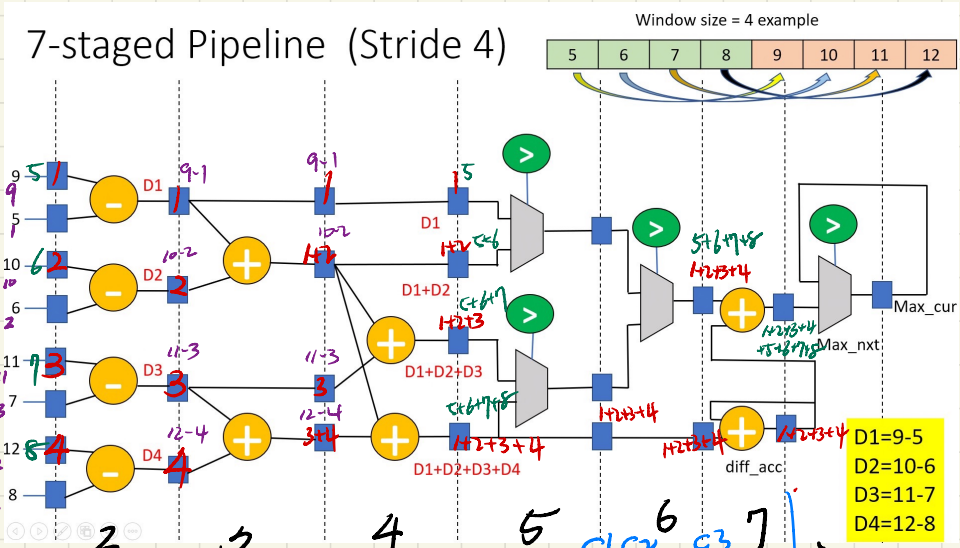

128=16 组 8 bit.
0
15
16
31
32
48
16
16
240
255
64
48
16 组.

mode 0.


128
0
4
8
252
1
5
9
253
2
6
10
254
3
7
11
255
64

1 2 3 4 5 6 7 8

2 3 4 5 6 7 8 9



42

53

windows 1

5-4 1

6-5 2-1

7-6 3-2

8-7 4-3

-1+3

-1-2+3+4

1 2

2 3

3 4

windows 2

5-3 1

6-4 2

7-5 3-1

8-6 4-2

diff_acc = 3+4

7+8

window = 8

9-1 5. 1

10-2 6. 2

11-3 7. 3

12-4 8. 4

45 207 70 41

R1 = 0

R2 = 0

R3 = 45

R4 = 207

R5 = 45

R6 = 207

R7 = 70

R8 = 41

D1 = 45

D2 = 207

D3 = 25

D4 = -186

S1 = 45

S2 = 252

S3 = 25

S4 = -141

S5 = 45

S6 = 252

S7 = 207

S8 = -111

142 - 70 = 72

53 - 188 = -135

42 - 142 = -100

0 - 53 = -53

142 + 188 = 330

72 72

-63 -63

-200 -163

-153 -216