

# NYCU-EE IC LAB – Spring2023

## Midterm project

### Design: Gray-level co-occurrence matrix (GLCM)

#### Data Preparation

1. Extract test data from TA's directory:

```
% tar xvf ~iclabta01/Midterm_Project_2023_spring.tar
```

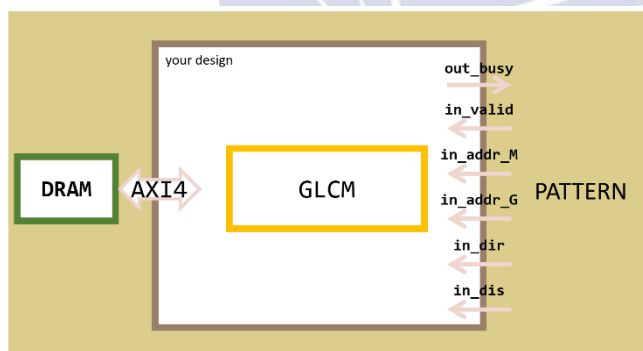
**\*\* Notice that you can only extract this file from the new servers. Please go through the “New server SOP.pdf” file to get into your account on the new servers.**

#### Basic Concept

Gray-Level Co-occurrence Matrix (GLCM) is a statistical method used to describe texture in image processing. It is a 2D matrix that counts the number of times different gray-level combinations occur in an image. The matrix can be calculated by analyzing the relationship between pixels in an image at a specific angle and distance. The resulting matrix can be used to extract features that describe the texture, such as contrast, homogeneity, and energy.

In this project, you are going to calculate the GLCM of a given matrix. The input matrix and computed result will both be read from or written to the DRAM through the AXI4 protocol.

#### System Architecture



The architecture shown on the left-hand side is the overall system architecture. Pattern will send control signal through IO ports when **in\_valid** is high.

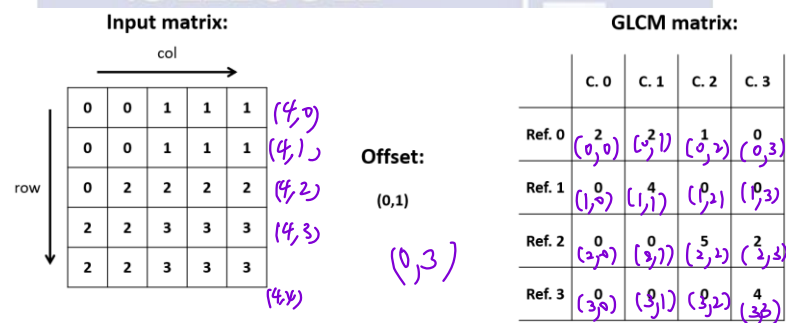
Your design should use the given control signals to fetch the corresponding matrix in DRAM through AXI4 protocol. After that you should calculate the GLCM and write the result back to DRAM. Then you should notify the pattern that you have completed the calculation by pulling **out\_valid** high.

#### GLCM

In this lab, your design should calculate the  $32 \times 32$  GLCM of a  $16 \times 16$  input matrix. The gray level value of the input matrix will be in the form of 5-bit data (ranged

from 0 to 31), and the value of the GLCM matrix will be in the form of 8-bit data (ranged from 0 to 255). In order to calculate the GLCM of a matrix, you will be given the direction (**in\_dir**) and the distance (**in\_dis**) of the target vector. The following steps show how to calculate the GLCM of a matrix:

1. Define the offset for the co-occurrence calculation. The offset is a 2D vector that represents the distance and direction between the pixels being analyzed. For example, an offset of (offset[row], offset[col]) = (0,1) means that you are analyzing the co-occurrence of a pixel with the one to its right.
2. Iterate over all the pixels in the matrix. In each iteration, extract the value of the pixel at position (row, col) and the value of the pixel at position (row+offset[row], col+offset[col]). Then increment the value of the corresponding cell in the GLCM matrix. Make sure to handle the case when the co-occurrence pixel falls outside the image.
3. The column direction of the GLCM matrix represent the gray level value of the original pixel, and the row direction of the GLCM matrix represent the gray level value of the co-occurrence pixel.



### AXI4 protocol

The AMBA AXI protocol is targeted at high-performance, high-frequency system designs and includes a number of features that make it suitable for a high-speed submicron interconnect.

In this exercise, the signal transaction between your design and DRAMs should follow the AXI4 protocol. The figure below illustrates an example of AXI4 signal waveform for read burst and write burst.

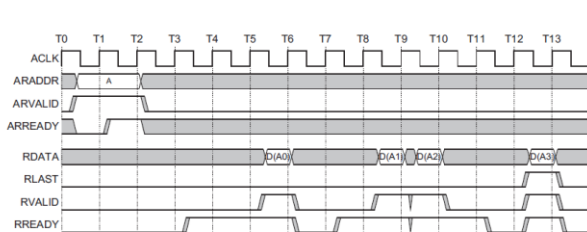


Figure 1-4 Read burst

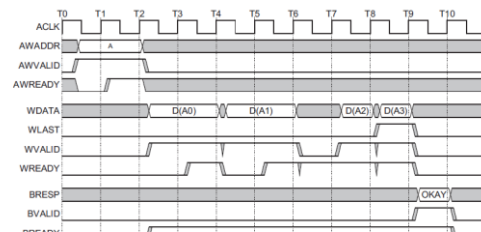
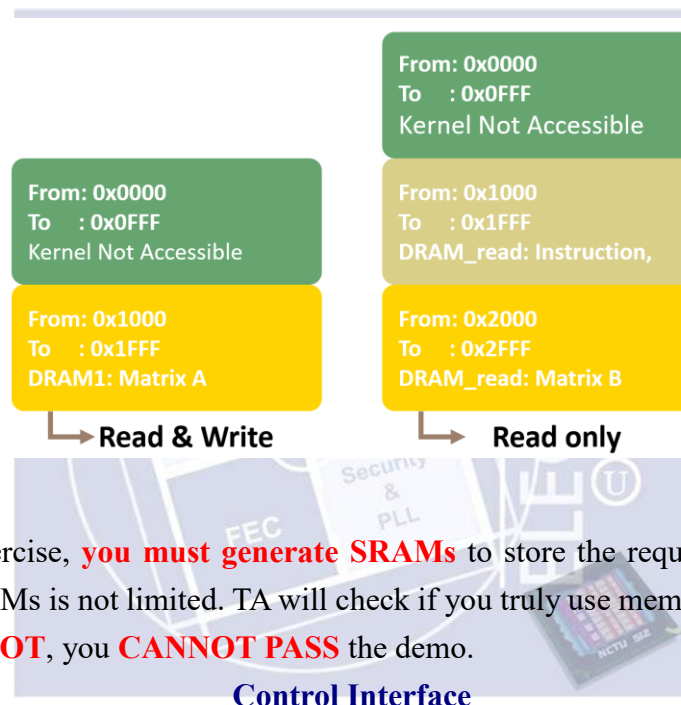


Figure 1-6 Write burst

### Memory mapping

The memory mapping method is displayed in the image. There is only one DRAM and it stores both the input matrix and GLCM, but in different memory addresses. The memory addresses from **0x000 to 0x0FFF** are reserved for kernel usage and are not accessible. All the data stored in the DRAM, including the input matrix and GLCM, are in the form of **1-byte (8-bit) data**, located in the addresses from 0x1000 to 0x2FFF. Note that the read/write data port has a width of 32 bits, so four address data must be read to form a functional signal. The input matrix can be retrieved from the readable addresses **0x1000 to 0x1FFF**, while the calculated GLCM should be written back to the readable and writable addresses **0x2000 to 0x2FFF**. The data format is stored in raster scan order.



In this exercise, **you must generate SRAMs** to store the required data and the number of SRAMs is not limited. TA will check if you truly use memory for the circuit operations. If **NOT**, you **CANNOT PASS** the demo.

### Control Interface

#### Inputs

I/O	Signal name	Bits	Description
Input	clk	1	Digital circuit clock signal
Input	rst_n	1	Digital circuit reset signal
Input	in_valid	1	High when in_addr_M, in_addr_G, in_dir, and in_dis signals are valid.
Input	in_addr_M	32	Address of the input matrix.
Input	in_addr_G	32	Address of the GLCM matrix.
Input	in_dir & in_dis	2 & 4	Direction and distance of the offset vector. in_dir = 2'b01 → offset vector = (in_dis×1 , 0);

			$\text{in\_dir} = 2'b10 \rightarrow \text{offset vector} = (0, \text{in\_dis} \times 1);$ $\text{in\_dir} = 2'b11 \rightarrow \text{offset vector} = (\text{in\_dis} \times 1, \text{in\_dis} \times 1);$ $*** (\text{row}, \text{col}) >>> \text{LSB1:col}, \text{LSB0:row}$ The distance value will not exceed the size of the input matrix.
--	--	--	---

## Output

1. All your results should be written back to DRAM. The test pattern will check whether your data in DRAM is correct or not at the first clock **negative** edge after **out\_valid** pulled high.
2. All outputs are synchronized at the clock **positive** edge.
3. **out\_valid** should be low after the initial reset.
4. **out\_valid** should not be raised when **in\_valid** is high.

## AXI interface

In this project, the signal naming rule for AXI4 related signals is: AXI4 signal name (lower case) + **\_m** (suffix).

## Write Address Channel

Signal	Source	Description
<del>AWADDR[31:0]</del>	Master	Write address ID. This signal is the identification tag for the write address group of signals. (In this project, we only use this to recognize master, reordering method is not supported)
AWADDR[31:0]	Master	Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
<del>AWLEN[31:0]</del>	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
<del>AWSIZE[31:0]</del>	Master	Burst size. This signal indicates the size of each transfer in the burst. (In this project, we only support 3'b010 which is 4 Bytes (matched with Data Bus-width) in each transfer)
<del>AWSTRIDE[31:0]</del>	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. (only INCR(2'b01) is supported in this Project)
<del>AWVALID</del>	Master	Write address valid. This signal indicates that valid write address and control information are available: 1 = address and control information available 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, <b>AWREADY</b> , goes HIGH.
<b>AWREADY</b>	Slave	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

## Write Data Channel

Signal	Source	Description
WDATA[31:0]	Master	Write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide. (In this project, we only support 32 bit data width: WDATA[31:0])
WLAST	Master	Write last. This signal indicates the last transfer in a write burst.
WVALID	Master	Write valid. This signal indicates that valid write data and strobes are available: 1 = write data and strobes available 0 = write data and strobes not available.
WREADY	Slave	Write ready. This signal indicates that the slave can accept the write data: 1 = slave ready 0 = slave not ready.

256 → 4 → 64

1024 → 16  
- 4 -

## Write Response Channel

Signal	Source	Description
BID[3:0]	Slave	Response ID. The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
BRESP[1:0]	Slave	Write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (only OKAY(2'b00) is supported in this Project)
BVALID	Slave	Write response valid. This signal indicates that a valid write response is available: 1 = write response available 0 = write response not available.
✓ <del>ARREADY</del>	Master	Response ready. This signal indicates that the master can accept the response information. 1 = master ready 0 = master not ready.

## Read Address Channel

Signal	Source	Description
✓ ARID[3:0]	Master	Read address ID. This signal is the identification tag for the read address group of signals. (In this project, we only use this to recognize master, reordering method is not supported)
○ ARADDR[31:0]	Master	Read address. The read address bus gives the initial address of a read burst transaction.
○ ARLEN[3:0]	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
✓ ARSIZE[2:0]	Master	Burst size. This signal indicates the size of each transfer in the burst. (In this project, we only support 3'b010 which is 4 Bytes (matched with Data Bus-width) in each transfer)
✓ ARBURST[1:0]	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. (only INCR(2'b01) is supported in this Project)
✓ ARVALID	Master	Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, <b>ARREADY</b> , is high. 1 = address and control information valid 0 = address and control information not valid.
input ARREADY	Slave	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

## Read Data Channel

Signal	Source	Description
in RID[3:0]	Slave	Read ID tag. This signal is the ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
in RDATA[31:0]	Slave	Read data. The read data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide. (In this project, we only support 32 bit data width: WDATA[31:0])
in RRESP[1:0]	Slave	Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (only OKAY(2'b00) is supported in this Project)
in RLAST	Slave	Read last. This signal indicates the last transfer in a read burst.
in RVALID	Slave	Read valid. This signal indicates that the required read data is available and the read transfer can complete: 1 = read data available 0 = read data not available.
✓ RREADY	Master	Read ready. This signal indicates that the master can accept the read data and response information: 1 = master ready 0 = master not ready.

### Specifications

1. Top module name: **GLCM** (File name: GLCM.v)
2. The design is **asynchronous** reset and **active-low** architecture.
3. All the data are in **unsigned** format.
4. The latency of your design (latency between PENABLE and PREADY) in each pattern should not be larger than **100,000** cycles. (during functionality test)
5. The maximum clock period is set to **20ns**, and **you can determine the clock**



**period by yourself.**

6. 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.
7. The output loading is set to 0.05.
8. The synthesis result of data type cannot include any **LATCH** (in syn.log).
9. After synthesis, you can check GLCM\_SYS.area, GLCM\_SYS.timing and GLCM\_SYS.resource.
10. The slack in the end of GLCM\_SYS.timing should be **non-negative** and the result should be **MET**.
11. 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.
12. You **CANNOT PASS** the demo if there are **timing violation** messages in the gate level simulation WITHOUT notimingchecks option.
13. No **ERROR** is allowed in every simulation/synthesis.
14. The latency between BVALID and BREADY, RREADY and RVALID should not be larger than **300** cycles.
15. DRAM must be declared in PATTERN, **do not declare it in your DESIGN**.
16. Don't use any wire/reg/submodule/parameter name called \*error\*, \*latch\*, \*congratulation\* or \*fail\* otherwise you will fail the lab. Note: \* means any char in front of or behind the word. e.g: error\_note is forbidden.
17. Don't write Chinese comments or other language comments in the file you turned in.
18. Verilog commands  

```
//synopsys dc_script_begin, //synopsys dc_script_end  
//synopsys translate_off, //synopsys translate_on
```

Are only allowed during the usage of including and setting designware IPs, **other design compiler optimizations are forbidden.**
19. Using the above commands are allowed, however **any error messages** during synthesize and simulation, regardless of the result will lead to failure in this lab.
20. The synthesize time cannot exceed **4 hours**.
21. Any form of display or printing information in verilog design is forbidden. You may use this methodology during debugging, but the file you turn in **should not contain any coding that is not synthesizable.**
22. The input matrix has the size of **16 × 16**, where each pixels' gray value are in the range of **0~31**. Therefore the size of the GLCM is **32 × 32**, and the value of each pixel is ranged from **0 to 255**.

## Grading Policy

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

**Functionality: 70%**

**Performance: (Total Latency)<sup>1.2</sup> x Area x Clock Period: 30%**

(in this performance metric latency means cycle count)

TA will demo your design with 2 different patterns, one is for functionality check, the other is for performance evaluation.

**For functionality check, the DRAM latency is in between 1 to 20, not fixed.**

**For performance evaluation, the DRAM latency is above 100, not fixed.**

Both PATTERNS will include some instructions that has temporal locality or spatial locality.

## Note

1. Please upload your design file, clk period file and memory files on new e3 platform before **12:00 noon**.

**Due Day :** 1<sup>st</sup> Demo : April 17<sup>th</sup> 2<sup>nd</sup> Demo : April 19<sup>th</sup>

**RTL design :** GLCM.v, (?? is your account number)

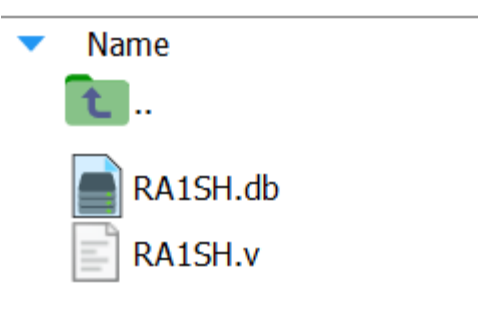
**Memory File :** MEMORY.v  
MEMORY.db

(?? is your account number).

**File List for Customized SRAM:** file\_list.f

Example:

Given your memory name is RA1SH512 and your submitted memory file RA1SH.v

in file_list.f	in 04_MEM
<pre>1 ../04_MEM/RA1SH.v</pre>	

Then, change directory to 09\_SUBMIT and

```
linux01 [Midterm_project/09_SUBMIT]% ./01_submit 20.0
```

Note that **you can provide multiple memory specs** in this lab.

If the uploaded files violate the naming rule, you will get **5 deduct points**.

**If you omit any file, you will fail the demo.**

2. Template folders and reference commands:

01\_RTL/ (for RTL simulation)

**./01\_run**

02\_SYN/ (for Synthesis)

**./01\_run\_dc**

**Remember to modify .tcl to fit your memory db name.**

(Check the design which contains **Latch** and **Error** or not in **syn.log**)

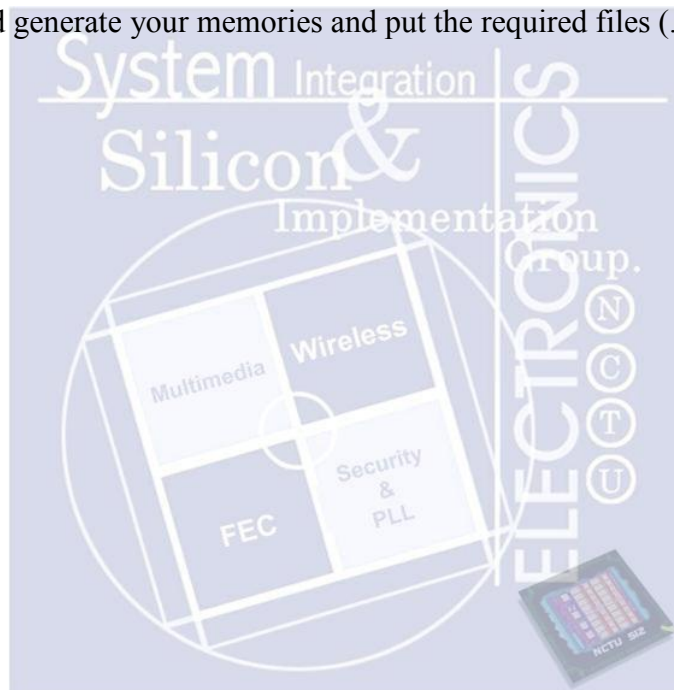
(Check the design's timing in /Report/GLCM.timing to see if the slack is **MET**)

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.





## Waveform Example

