

# Compute platforms for AI and Embedded Processing

Exercise sessions (= project)

# Convolution

input

4	1	2	5	8	3
6	0	7	4	1	2
3	1	9	5	2	4
7	2	6	9	7	8
3	8	1	0	2	3
8	7	5	6	2	1

kernel

2	4	1
3	1	5
0	6	2

\*

=

output


Makes use of local structure in image

# Convolution

input

4	1	2	5	8	3
6	0	7	4	1	2
3	1	9	5	2	4
7	2	6	9	7	8
3	8	1	0	2	3
8	7	5	6	2	1

kernel

2	4	1
3	1	5
0	6	2

\*

=

output

91			

Per output pixel

1. Elementwise multiplication
2. Addition of results

# Convolution

input

4	1	2	5	8	3
6	0	7	4	1	2
3	1	9	5	2	4
7	2	6	9	7	8
3	8	1	0	2	3
8	7	5	6	2	1

kernel

2	4	1
3	1	5
0	6	2

\*

=

output

91			
98			

# Convolution

input

4	1	2	5	8	3
6	0	7	4	1	2
3	1	9	5	2	4
7	2	6	9	7	8
3	8	1	0	2	3
8	7	5	6	2	1

\*

kernel

2	4	1
3	1	5
0	6	2

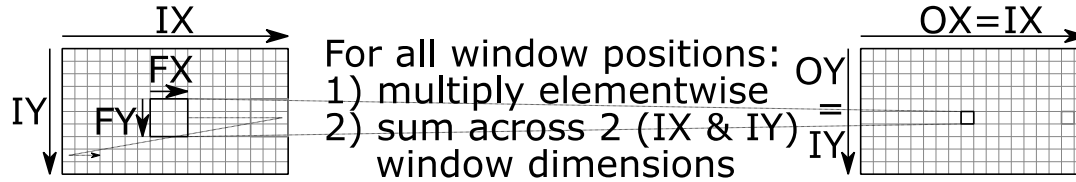
=

output

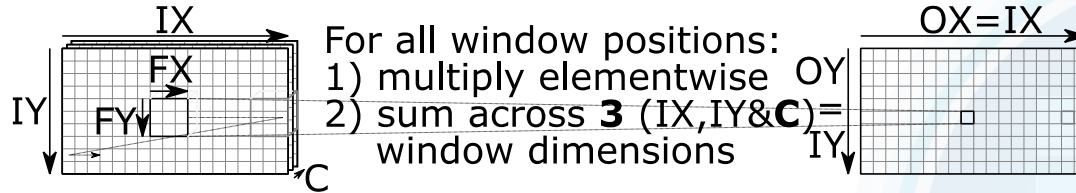
91	106	96	88
98	123	141	109
122	106	106	114
102	104	108	85

# Convolution → Convolutional layer

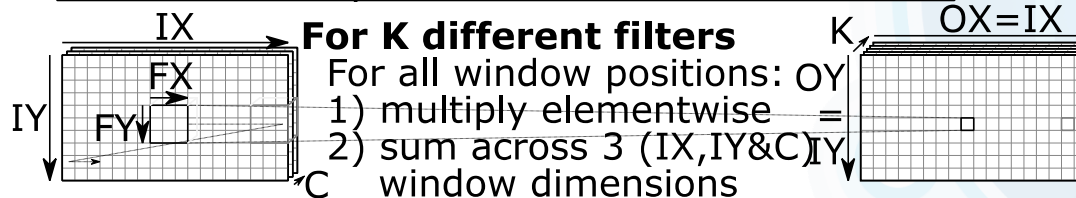
## Plain 2d convolution



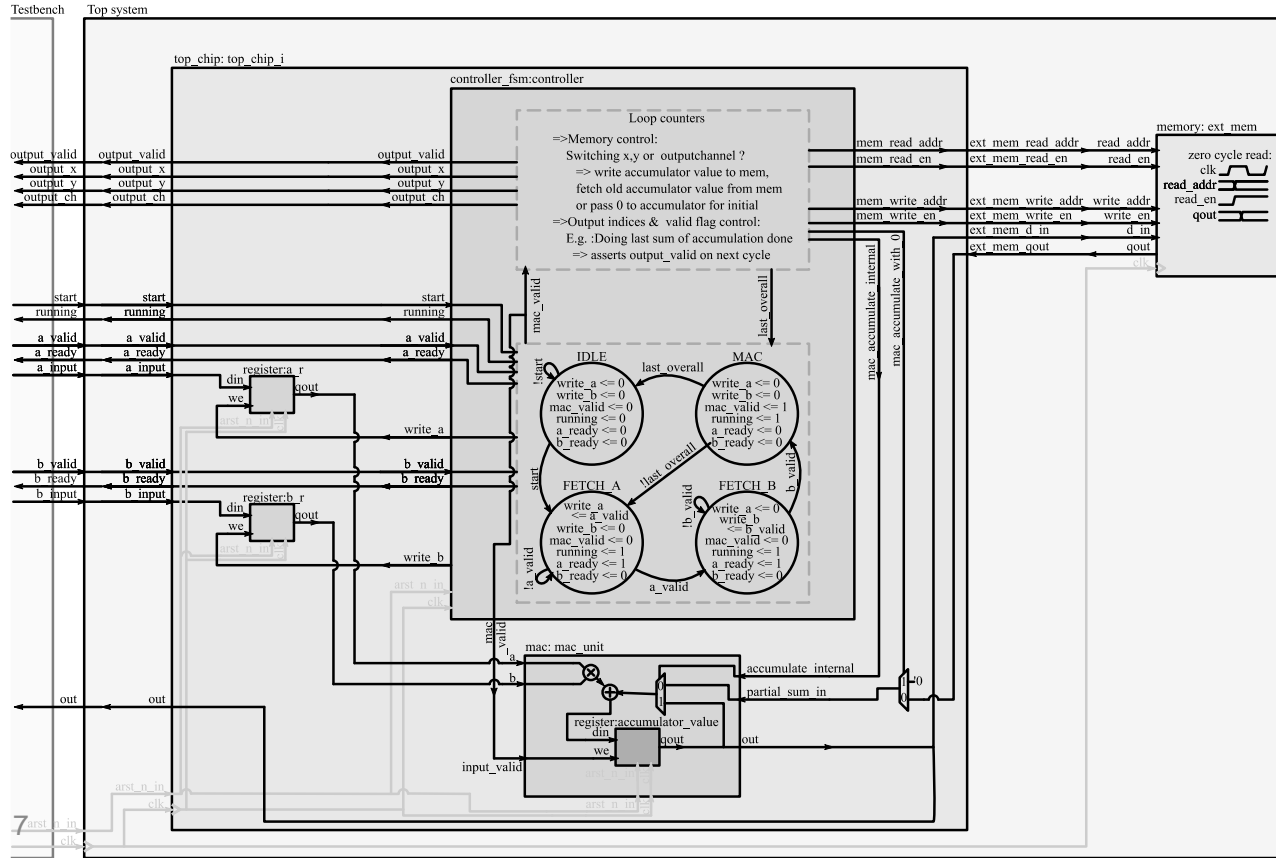
## Make input 3D: one more summation dimension ( $C$ )



## Convolutional layer convolution: add more filters



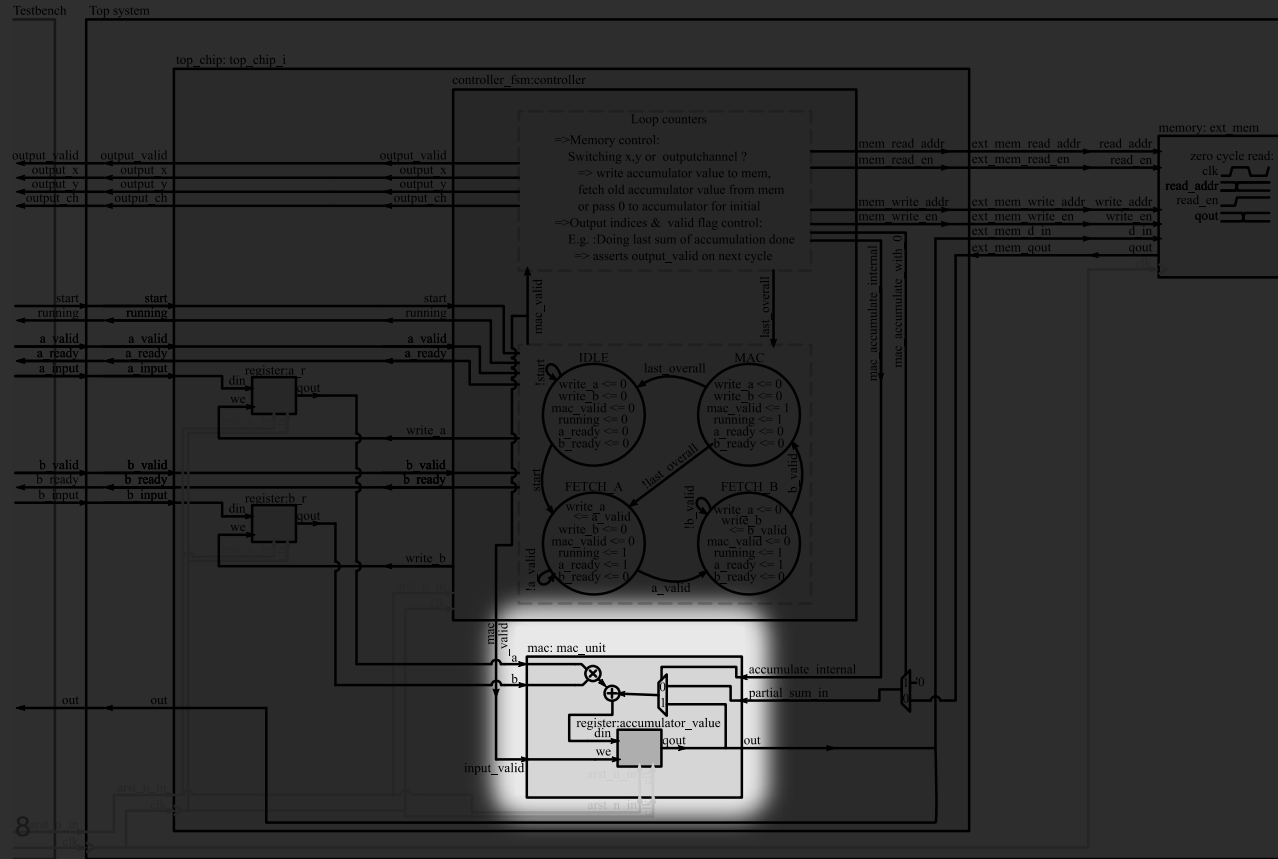
# The Device Under Test (DUT)



Slides will present the DUT as given as a starting point to you.

Keep of this what you want, adjust what you, start from scratch if you want ...

# The Device Under Test (DUT)

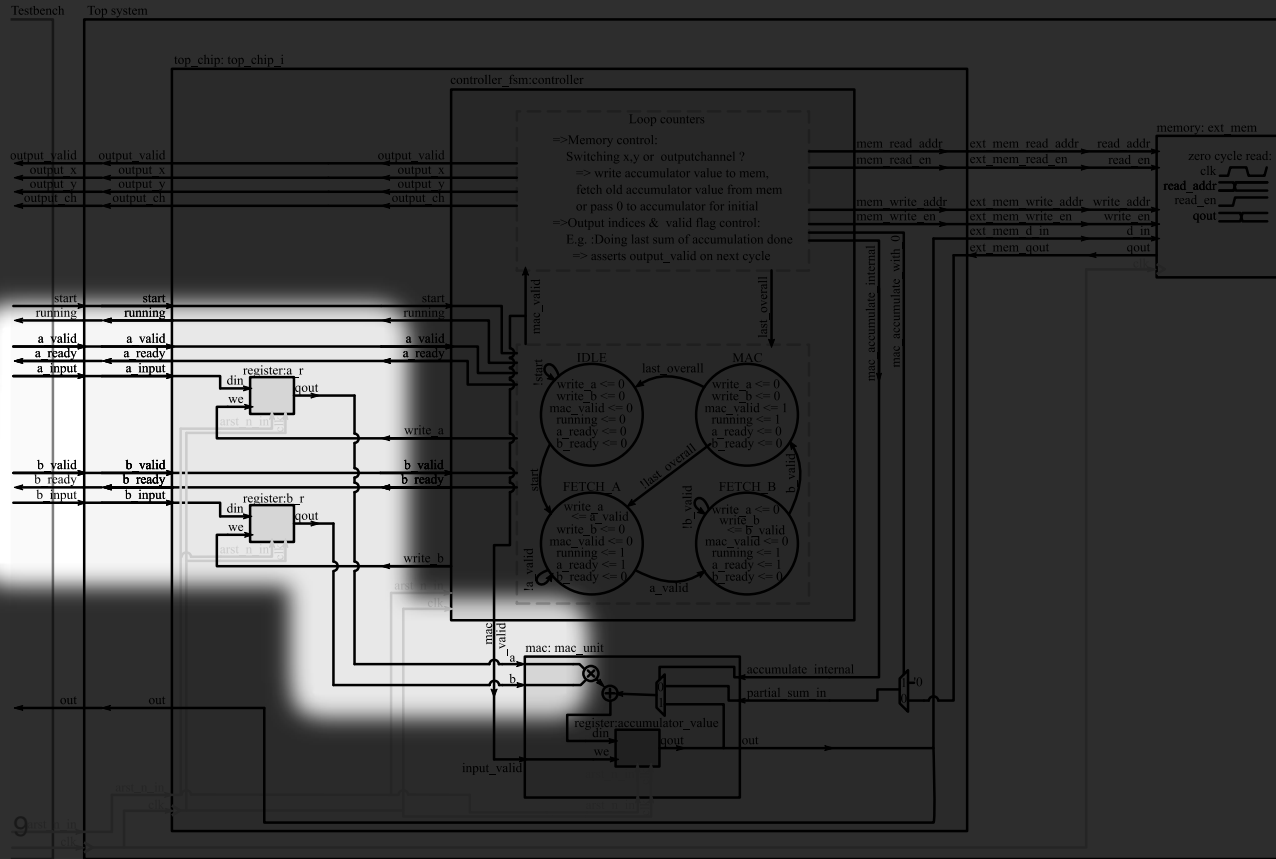


## MAC unit

- Does actual Multiply ACcumulate operations
- Needs to be fed data

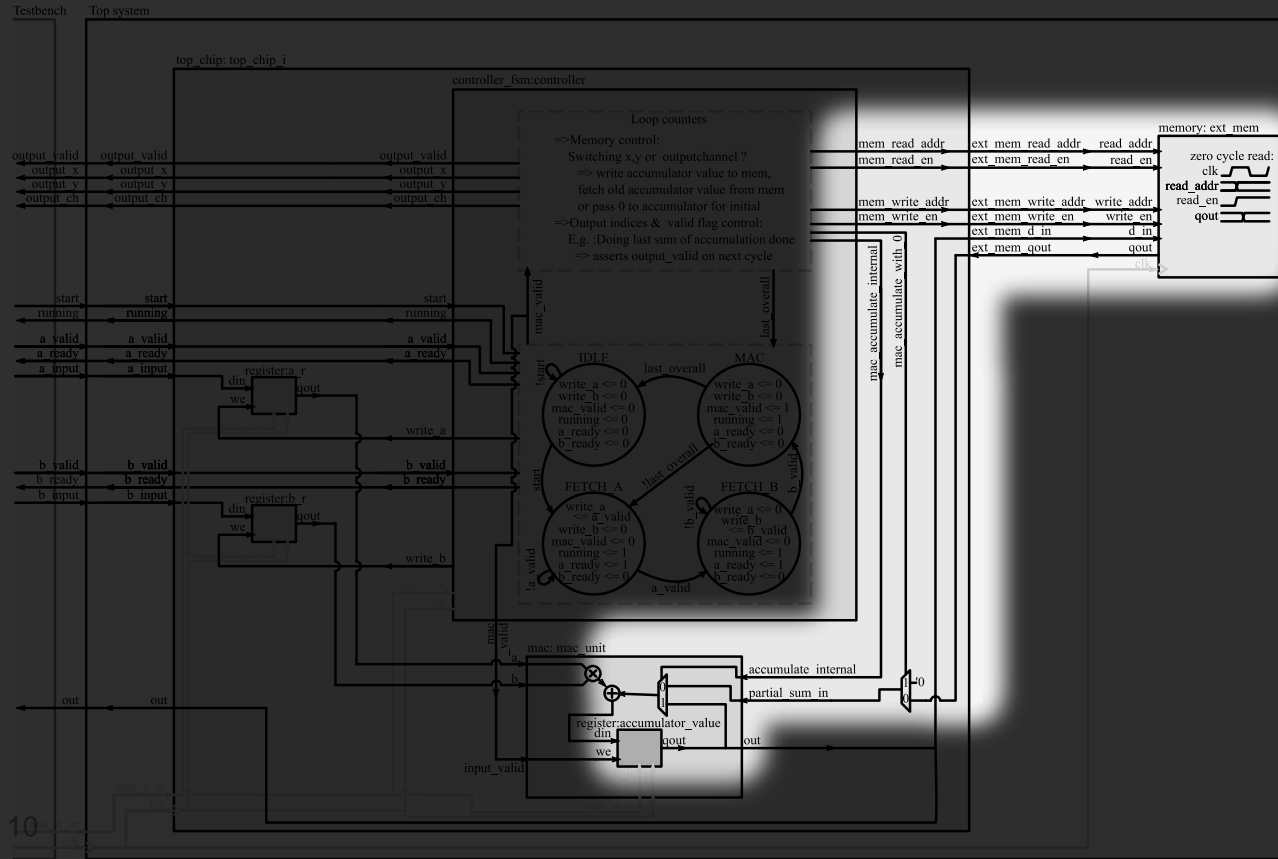


# The Device Under Test (DUT)



**Multiplication factors  
come from testbench,  
after being stored in  
registers first**

# The Device Under Test (DUT)



- Partial sums come from**
- **Nowhere for the first MAC: it is 0**
  - **Accumulator value inside the register in the MAC unit**
  - **External memory if the partial sum to continue accumulating to is no longer in the MAC's register**

The diagram illustrates a hardware accelerator architecture for a neural network layer, consisting of three main components: a top chip, a controller FSM, and a memory block.

**Top Chip (top\_chip\_i):** This block interfaces with the controller FSM and the memory. It provides input data (a, b) and control signals (start, running, valid, ready, input, output). It also receives output data (x, y, ch) and status signals (output\_valid, output\_x, output\_y, output\_ch).

**Controller FSM (controller\_fsm/controller):** This block contains a state machine and logic for memory control. The state machine has four states: IDLE, MAC, FETCH A, and FETCH B. The transitions are triggered by 'last overall' signals. The logic includes memory control (switching x, y or output channel), writing accumulator values to memory, and fetching old accumulator values from memory. It also asserts output\_valid on the next cycle.

**Memory (memory\_ext\_mem):** This block stores data and provides read/write addresses and data. It includes signals for zero cycle read, read address, read enable, and read data (qout).

**MAC Unit (mac: mac unit):** This block performs the MAC (Multiply-Accumulate) operation. It takes inputs a and b, multiplies them, and accumulates the result. It includes a register for the accumulator value and a partial sum in.

**Signal Traces:** The diagram shows various signal traces, including data signals (a, b, x, y, ch), control signals (start, running, valid, ready, input, output), and status signals (output\_valid, output\_x, output\_y, output\_ch). It also shows internal signals like 'mac\_valid', 'last overall', 'mac accumulate internal', and 'partial sum in'.

**State Machine Diagram:** The state machine diagram shows the following states and transitions:

- IDLE:** write a <= 0, write b <= 0, mac\_valid <= 0, running <= 0, a\_ready <= 0, b\_ready <= 0. Transitions: 'last overall' to MAC, 'start' to FETCH A.
- MAC:** write a <= 0, write b <= 0, mac\_valid <= 1, running <= 1, a\_ready <= 0, b\_ready <= 0. Transitions: 'last overall' to IDLE, 'last overall' to FETCH B.
- FETCH A:** write a <= a\_valid, write b <= 0, mac\_valid <= 0, running <= 1, a\_ready <= 1, b\_ready <= 0. Transitions: 'last overall' to IDLE, 'last overall' to FETCH B.
- FETCH B:** write a <= 0, write b <= b\_valid, mac\_valid <= 0, running <= 1, a\_ready <= 1, b\_ready <= 1. Transitions: 'last overall' to IDLE, 'last overall' to FETCH A.

**FSM controls fetching of inputs and when the MAC unit is active (=accumulating)**

[illegible]

**define the order in which all the MACs are executed (=dataflow)**

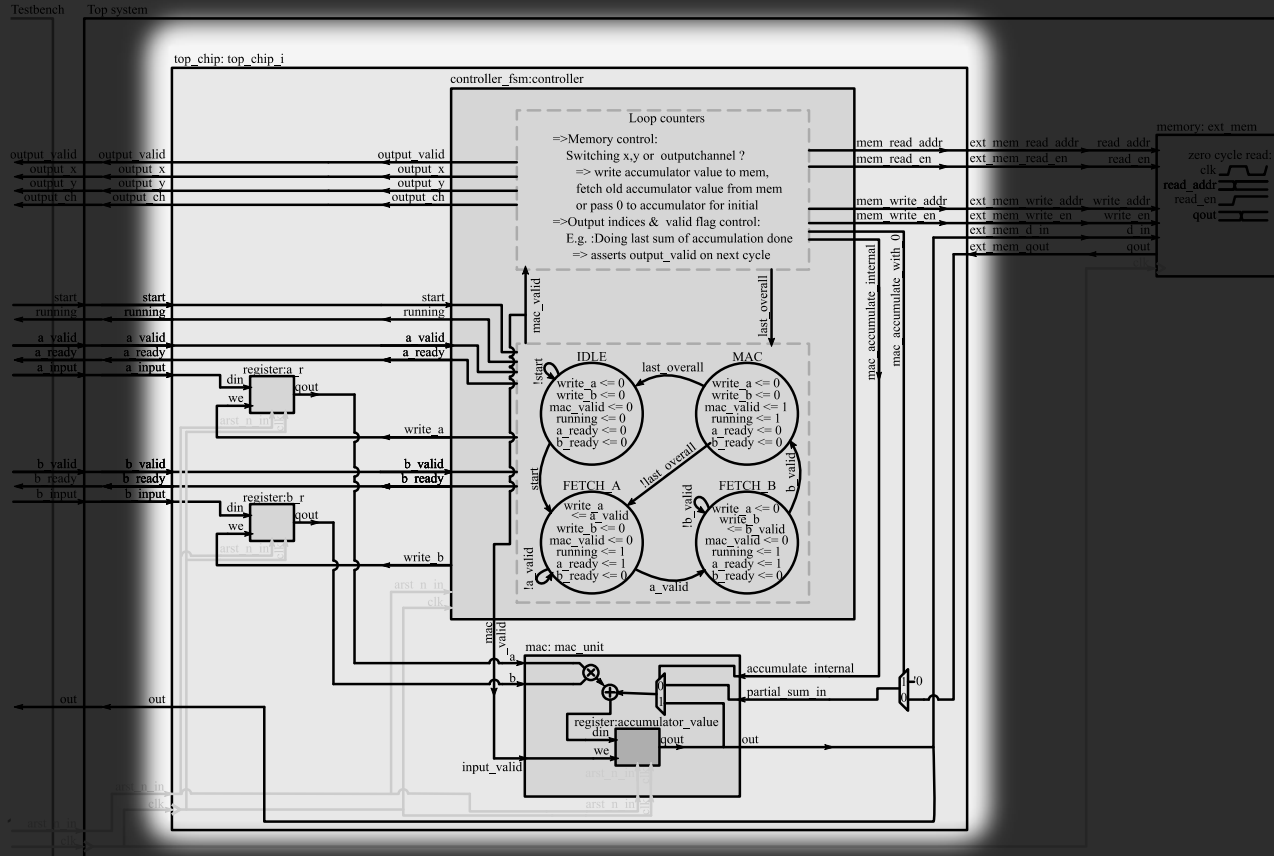
## Through loop counters that increment/reset each other

→ are aware when a fully accumulated sum is ready → drives output information signal

→ know what partial sum the MAC unit should use → drives those muxes

➔ know when to store partial sum for later use, and when to fetch it ➔ drives external memory control

# The Device Under Test (DUT)

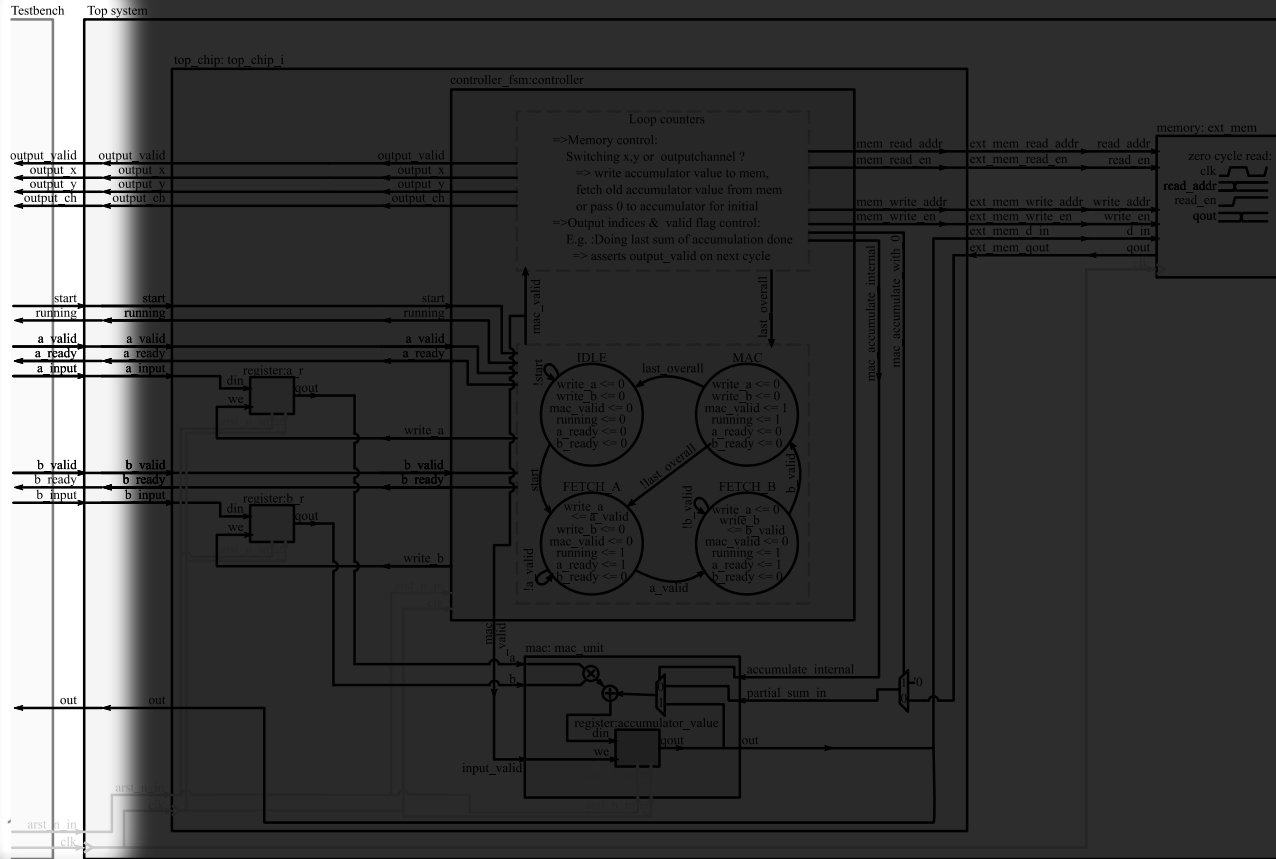


## Top chip:

- Everything inside is considered on chip → counts for area constraint
- Everything going in/out is external communication → has high energy cost

**=Top\_chip + external memories/FIFOs**

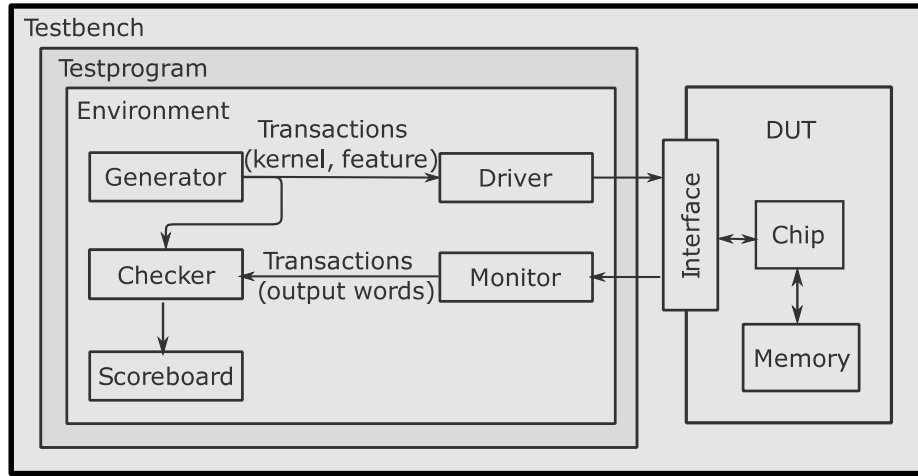
# The Device Under Test (DUT)



Connection  
top\_system <-> testbench:

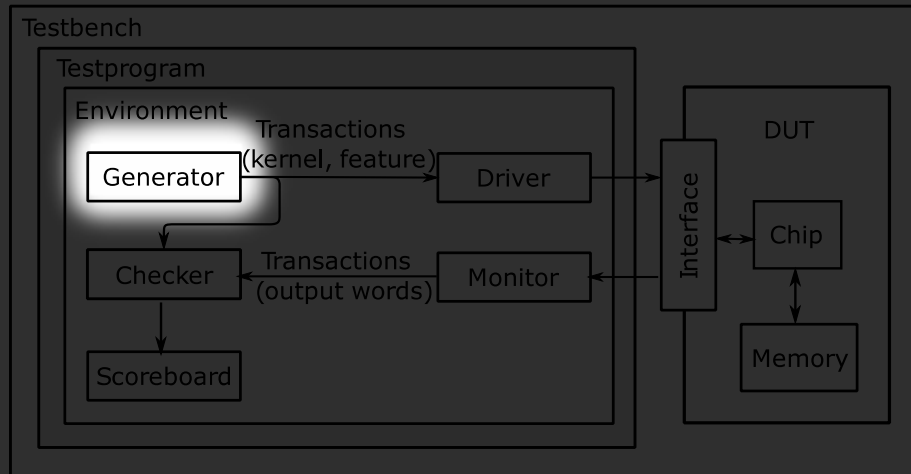
Constrained to 48 bits,  
excluding handshaking  
signals like valid & ready,  
and output\_x/y/ch

# Testbench





# Testbench



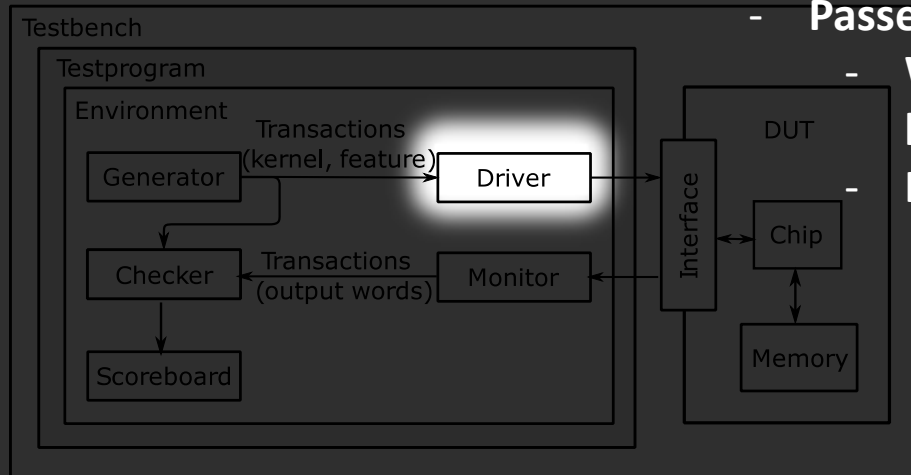
## Generator:

- Generates random inputs
- Encapsulates them as a transaction
- Sends this transaction to Driver & Checker (through mailboxes)

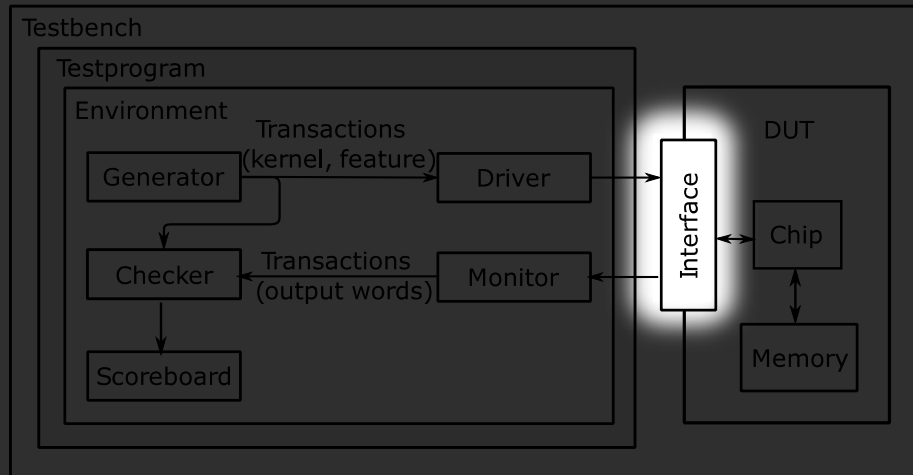
# Testbench

## Driver:

- Receives transaction with inputs from generator (through mailbox)
- Passes this data to chip
  - With respect for handshaking and other low level control
  - In the order that the chip expects it
    - As per dataflow set by loop counter logic
    - ➔ To change dataflow, change loop counter logic and driver together, so that sender (driver) and receiver (chip) still agree on order of inputs



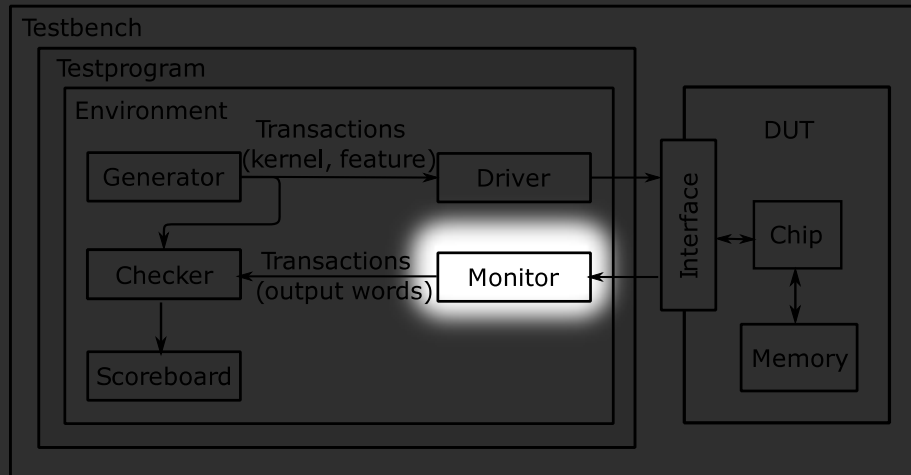
# Testbench



## Interface:

- Lists all signals to/from chip
- Declares input and output latencies after/before clock with a clocking block
- ➔ drive and read all synchronous signals through the clocking block
- In testbench, use `@(intf_i.cb)` instead of `@(posedge clk)`

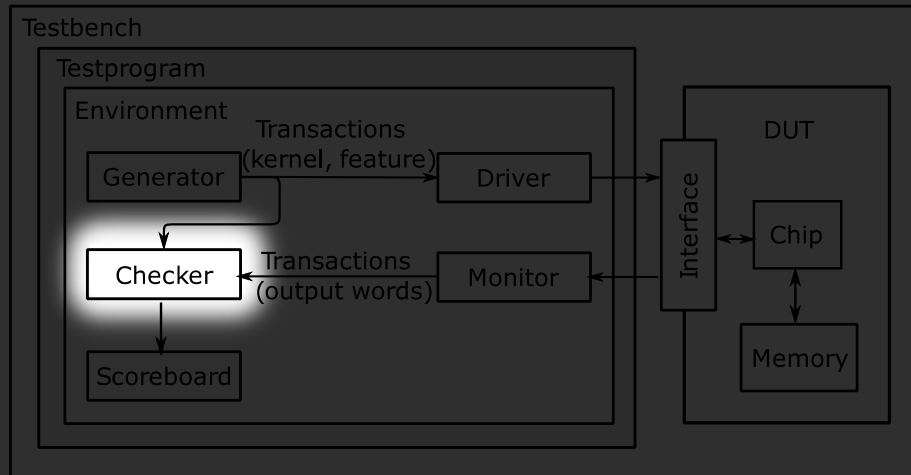
# Testbench



## Monitor:

- Reads outputs from chip
  - With respect for handshaking and other low level control
- Encapsulates these outputs in a transaction
- Sends this transaction to the checker (through a mailbox)

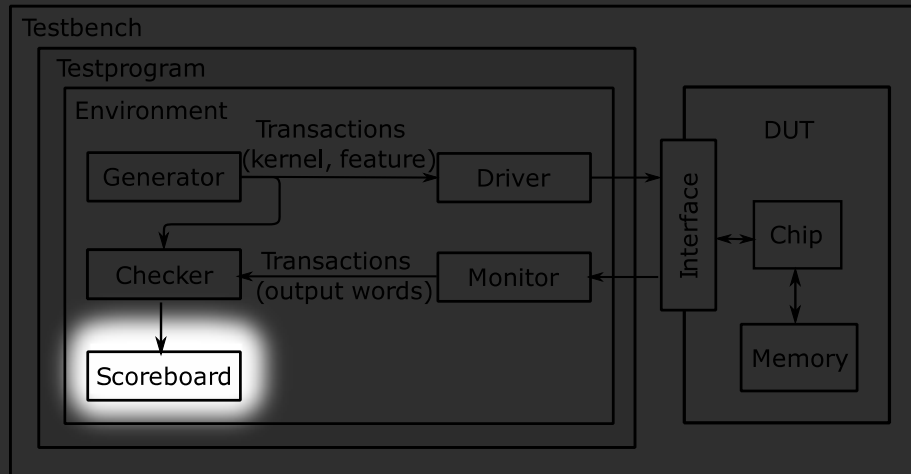
# Testbench



## Checker:

- Receives inputs from generator as a transaction (through mailbox)
- Uses golden reference code to calculate expected outputs
- Receives actual outputs as a transaction from monitor (through mailbox)
- Compares expected vs actual output
- and notifies the scoreboard of it

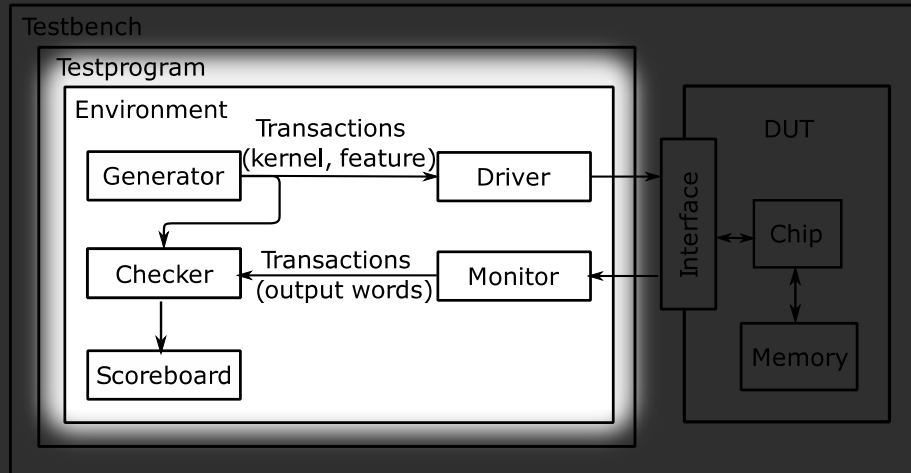
# Testbench



**Scoreboard:**

**keeps track of number of (passed and failed tests)**

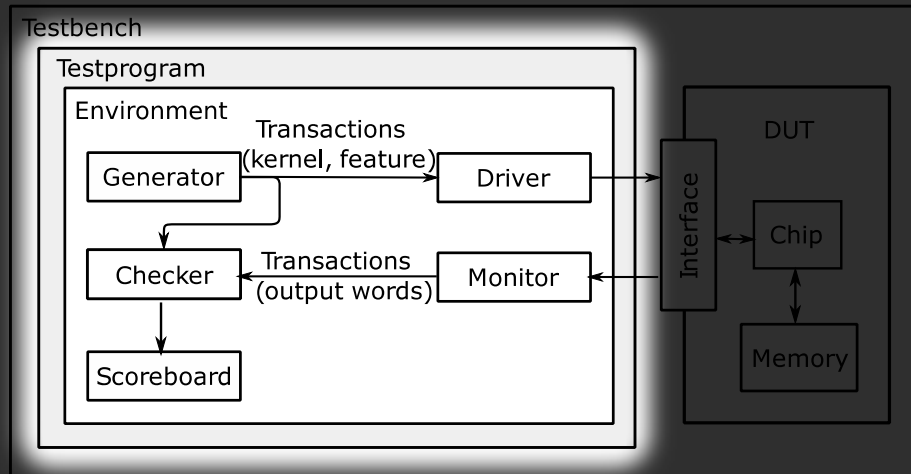
# Testbench



## Environment:

- Instantiates all of the above (except interface)
- Instantiates mailboxes
- Starts all of the above

# Testbench

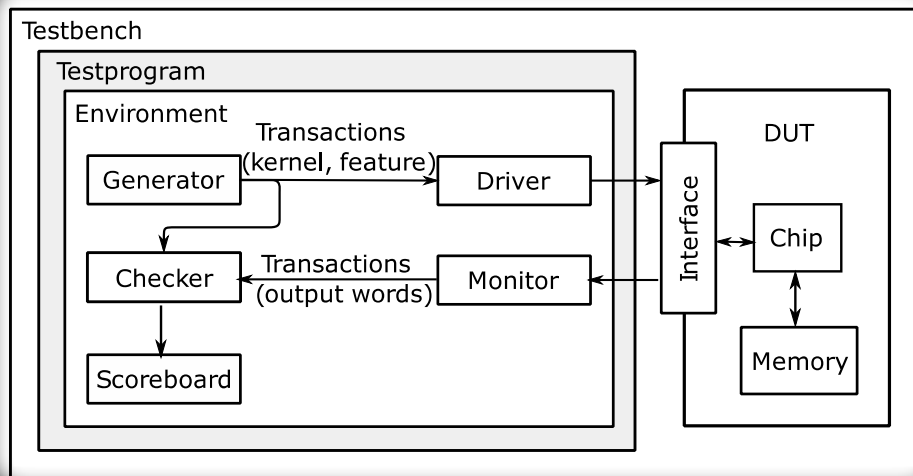


## Testprogram:

- Instantiates and starts environment



# Testbench



## testbench:

- Absolute top level
- Instantiates testprogram, DUT, and interface between them
- Generates clock

# Target

---

 Minimize Energy X Latency

# Subtarget: latency

- ⌚ Clock period, defined by critical path
  - =Maximum number of sequential 32b adders / 16b multipliers between two registers (or external inputs/outputs)
- ⌚ X
- ⌚ # clock cycles
  - Can be estimated as  $\frac{\#MACs\_to\_be\_done}{\#MAC\_units\_in\_design \cdot their\_utilization}$
- ⌚ **Use provided adder and multiplier building blocks and adjust clock period in testbench → TB will error if clock period set to low, report latency otherwise**

# Subtarget: energy

---

- ⚙️ Energy caused by:
  - MACs: unavoidable constant cost
- ⚙️ On-chip memory/FIFO transfers: 0.1/bit read/written
- ⚙️ External communication: 1/bit read/written
  - Testbench: check bottom of intf.sv, update when necessary
  - external memory/FIFO: accounted for if you use provided building blocks

# Constraints

- Testbench <-> DUT bandwidth
  - 48 data bits (see above)
- Area of top\_chip:
  - On-chip memory/FIFO  $\geq 256$  words: 1/bit
  - On-chip memory/FIFO  $< 256$  words: 17/bit
  - Registers: 17/bit
  - 16x16b multiplier: 5800
  - 32+32b adder: 1000
- When using provided memory, registers, FIFO, adder and multiplier building blocks: calculated and reported automatically by simulation

# Practicalities

---

 See assignment

# Competition

- Once you have obtained the target scores (area, latency, energy), submit the forms:
  - <https://forms.office.com/r/dCsxypwVWx>
- Results & ranking will be shown here:
  - [https://kuleuven-my.sharepoint.com/:x:/g/personal/kodai\\_ueyoshi\\_kuleuven\\_be/ER55kho73lZGoUymMOpw\\_OQB057s6Md0DRx9CfAshxVWu7g?e=Zf4uS1](https://kuleuven-my.sharepoint.com/:x:/g/personal/kodai_ueyoshi_kuleuven_be/ER55kho73lZGoUymMOpw_OQB057s6Md0DRx9CfAshxVWu7g?e=Zf4uS1)
- If you want to remove/correct submission, send an email to [kodai.ueyoshi@kuleuven.be](mailto:kodai.ueyoshi@kuleuven.be).

Submission form:



Results sheet:

