2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

SystemVerilog Real Models for an In-Memory Compute Design

Daniel Cross,

Sr. Principal Solutions Engineer

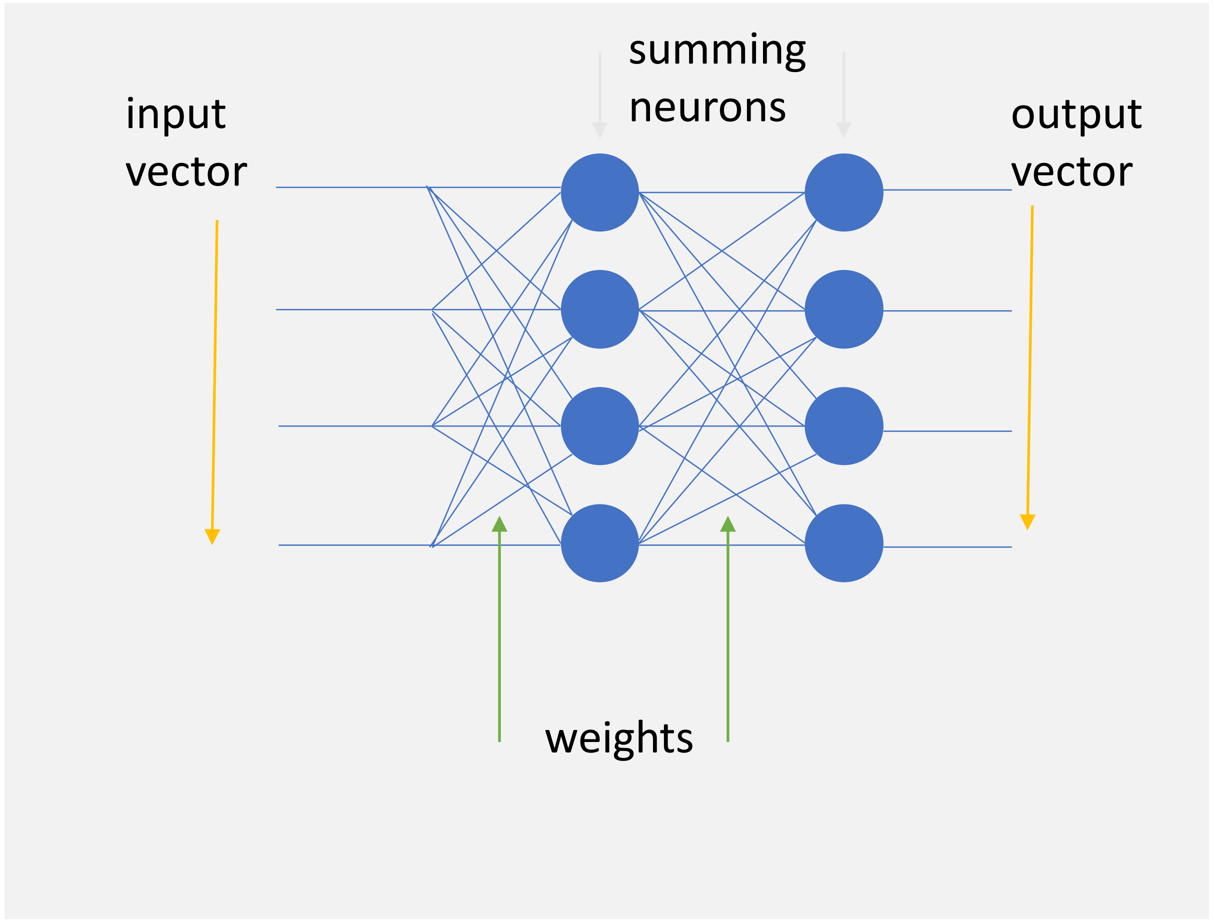
cadence®

accellera
SYSTEMS INITIATIVE

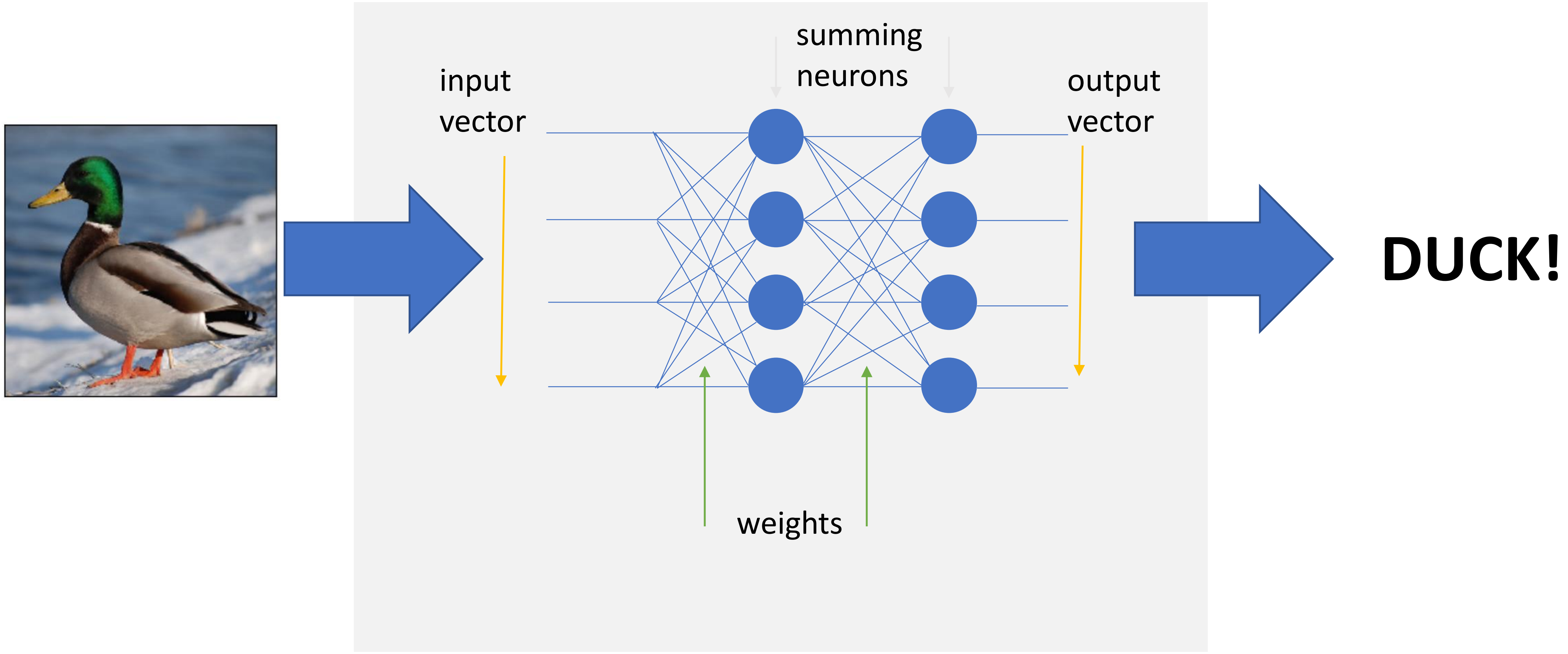
In-Memory Computing (IMC, CIM, etc.):

- What is it?
- Why is it needed for Machine Learning applications?
- How does it work?
- Is it really mixed-signal?
- How should we model it?

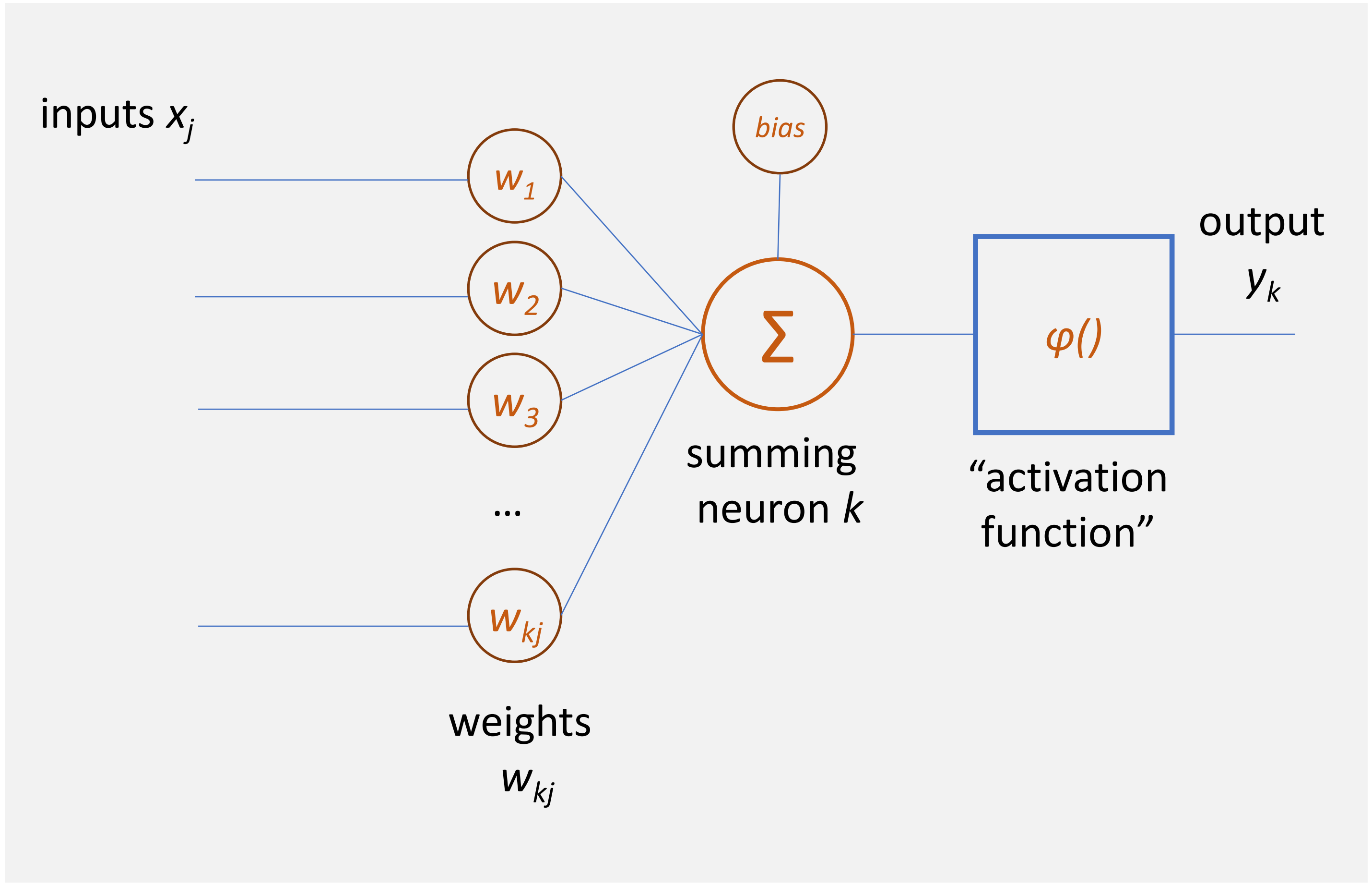
A Neural Network



A Neural Network



A Single Neuron



The output of Neuron k is given by

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

As a computational algorithm, this is a "Multiply and ACcumulate" or MAC operation.

Dedicated hardware in Graphics Processors is very efficient at performing this operation.

The limiting factor in terms of time and power consumption becomes moving the data around . . .

Estimates of Energy per Memory Access

• 10 – 50 nJ / access (write/read)	Trajkovic, Jelena & Veidenbaum, Alexander & Kejariwal, A.. (2008). “Improving SDRAM Access Energy Efficiency for Low-Power Embedded Systems.” *ACM Trans. Embedded Comput. Syst.*. 7. 10.1145/1347375.1347377.
• 640 pJ / 32b read	Murmann, “Mixed-Signal Techniques for Embedded Machine Learning Systems,” presented at CERN EP-ESE Electronics Seminar, Aug 2019
• 2 nJ / 64b access	Verma, et.al., “In-Memory Computing: Advances and Prospects,” IEEE Solid State Circuits Magazine, Summer 2019 Vol 11 No 3, p. 44
• 20 pJ / bit	M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, San Francisco, CA, 2014, pp. 10-14, doi: 10.1109/ISSCC.2014.6757323.

In-Memory Compute for Power Optimization

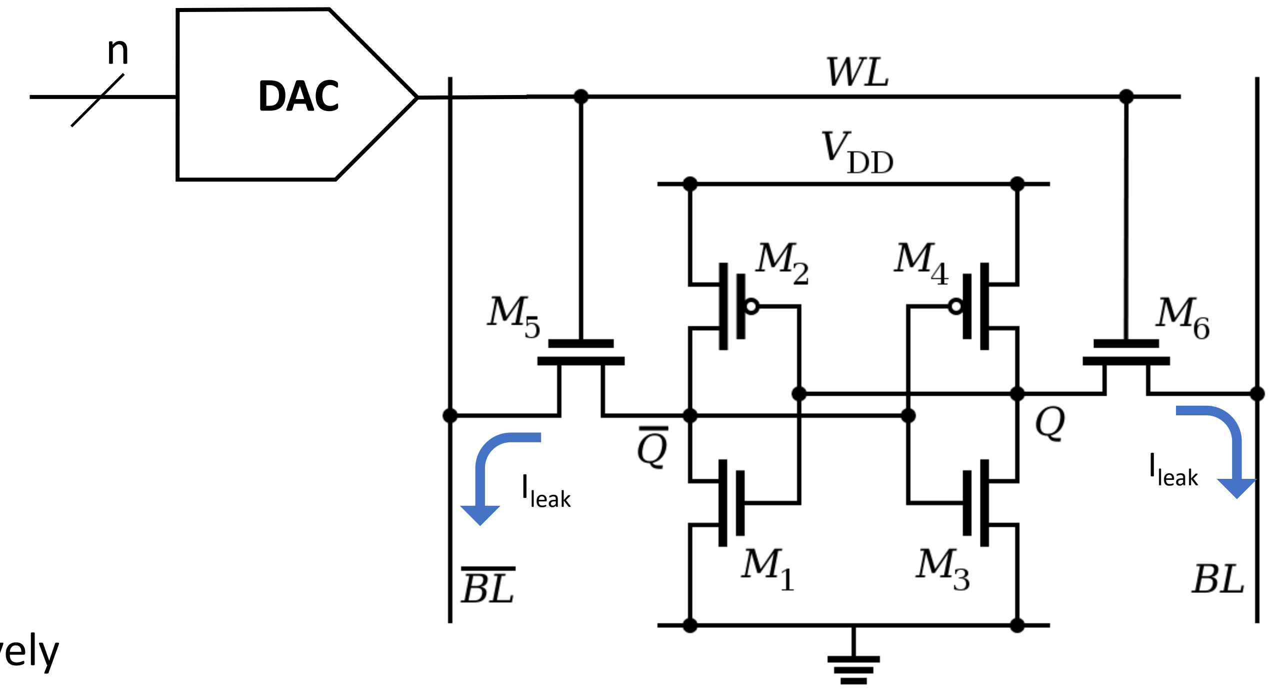
Remember the neuron equation: $y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$

If we can perform the needed MAC operations without moving the data, we can save energy.

IMC relies on using SRAM bit cells as tiny 1-bit multipliers. There is more than one strategy for doing this. We will focus on just one.

In-Memory Compute with SRAM

However, if instead of driving WL completely on we bias it in the active region of M5 and M6, the latch will leak current onto the bit-lines. The amount of leakage will be proportional to the bias voltage.



By driving the word-line with a DAC, we can effectively multiply the bitcell contents with a multi-bit value by sensing the relative amount of leakage current.

In-Memory Compute with SRAM

M x N bitcell array

DAC resolution is j bits

ADC resolution is k bits

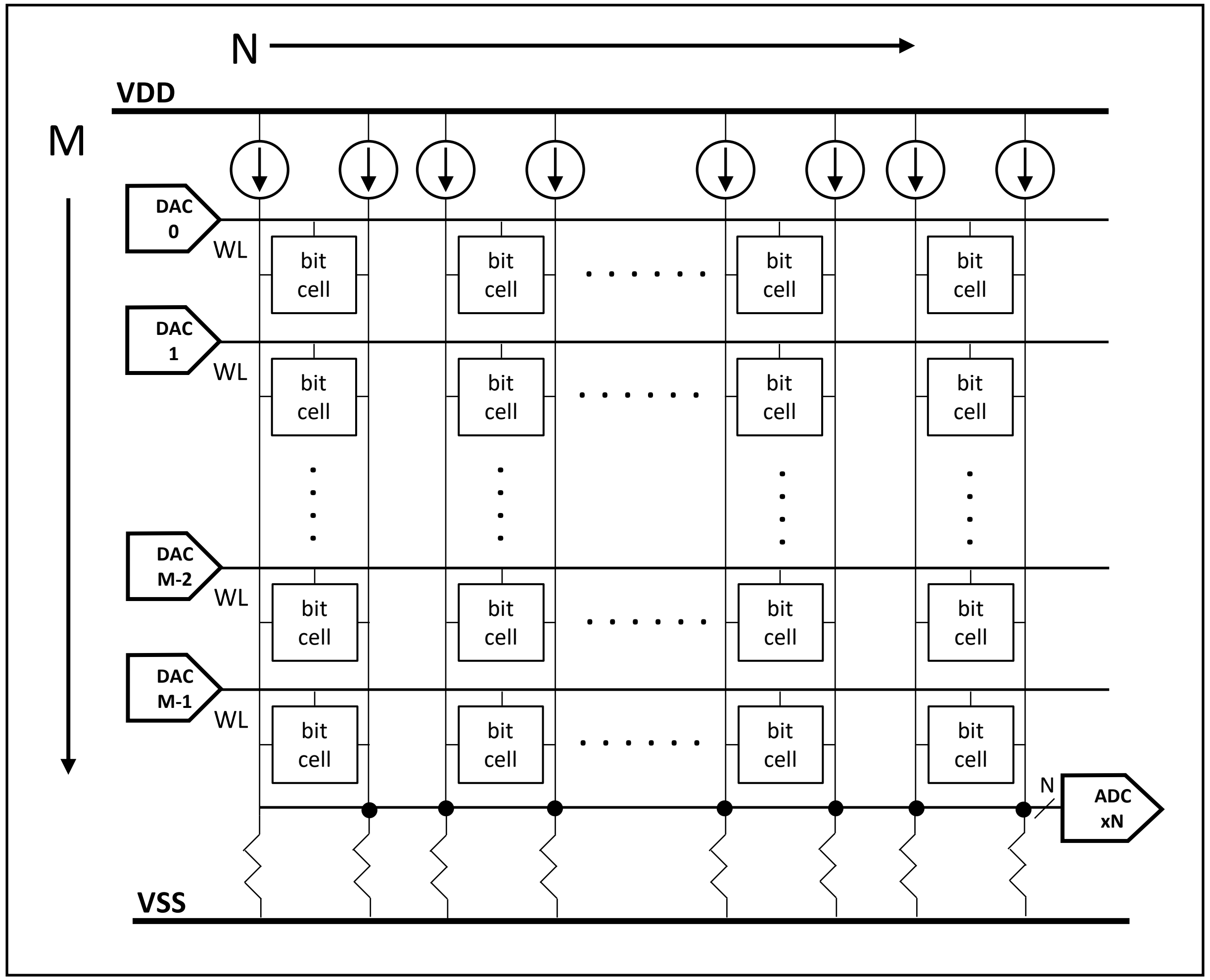
Currents sum on the vertical bit-lines (a '0' value stored in the bitcell will contribute a negative current)

Performs the needed MAC:
$\sum_{i=0}^{M} w_{ki} x_i$

The weights w_p are *N* bits wide

The input vectors x_i are *j* bits wide

The output is *N* x *k* bits wide



SystemVerilog Real IMC Bitcell (bitCell.sv)

For the word lines, we created a custom User Defined Nettype **`WLnet`** with the following structure:

```
typedef struct {
   enum {OFF, s0, s1, s2, s3, s4, s5, s6, s7, ON} state;
   }  t_WLnet;
```

The ON state is used for writing (or reading) the bitcell contents. The intermediate states s0-s7 correspond to values of the 3-bit WLDAC.

This design has at most one driver on each WL, but for future use we included a resolution function that resolves to the value of the greatest driver, with ON and OFF equivalent. Driving the net with ON and OFF resolves to X.

We use the EEnet UDN to model supply nets.

SystemVerilog Real IMC Bitcell (bitCell.sv)

```
module bitCell (
  inout EEnet vdd, vss, bL, bLb,
  input WLnet WL,
  input logic wEN, rEN
  );
```

The bitcell model has write-enable and read-enable controls in addition to the word line and differential bit lines.

Supply currents are tracked based on the operating mode of the bitcell:

```
parameter real iWrite = 1e-6;
parameter real iCalc = 1e-8;
parameter real iLeak = 1e-11;
real iSupply;

assign vdd = '{`wrealZState, ((supplyOn == 1'b1) ? -iSupply : 0.0), 0};
assign vss = '{`wrealZState, ((supplyOn == 1'b1) ?  iSupply : 0.0), 0};
```

The EEnets used for vdd and vss sum up the dynamic active currents for the entire memory to help with system power budgeting.

SystemVerilog Real IMC Bitcell (bitCell.sv)

A logic value stores the contents of the bitcell:

```
logic content = 1’bx;
```

Leakage currents that map to the s0-s7 states of WL are defined (in an array):

```
real scaleFactor [0:7] = '{0e-6, 10e-6, 20e-6, 30e-6, 40e-6, 50e-6, 60e-6, 70e-6};
```

The bitcell enters a non-standby mode on rising edges of **`wEN`** or **`rEN`**, depending upon the value of the word-line held in **`WL.state`**:

OFF:

```
if ((WL.state == OFF) || (WL.state == 1'bx)) begin  // bit cell OFF
       iSupply = iLeak;
       blI = 0.0;
       blbI = 0.0;
    end
```

SystemVerilog Real IMC Bitcell (bitCell.sv)

ON: (direct write or read)

```
else if (WL.state == ON) begin   // Logic Write ON. Set content holding register
       if (wEN == 1'b1) begin
          iSupply = iWrite;
          content = ((bL.V - bLb.V) == `wrealXState) ? 1'bx : ((bL.V - bLb.V) > blThd) ? 1’b1
                         : ((bL.V - bLb.V) < -blThd) ? 1'b0 : 1'bx ;
          blI = 0.0;
          blbI = 0.0;
       end
       else begin // must be (rEN == 1'b1)
          iSupply = iCalc;
          blI  = (content == 1) ? scaleFactor[7] : -scaleFactor[7]; // Logic read; currents are max/min
          blbI = (content == 1) ? -scaleFactor[7] : scaleFactor[7]; // depending on register contents
       end
    end
```

For a write cycle, supply current is set to `iWrite`, and the values on the bit-lines are compared to a threshold value and 1, 0, or X is stored in `content`.

For a read cycle, supply current is set to `iCalc`, and the currents to be driven onto the bit-lines is set to max or min depending on the stored contents.

SystemVerilog Real IMC Bitcell (bitCell.sv)

Intermediate state (s0-s7) -- Multiplication:

```
else begin
       // WL is a DAC state
       blI = scaleFactor[WL.state - 1]*(content ? 1 : -1);   // bitLine current depends
       blbI = -scaleFactor[WL.state - 1]*(content ? 1 : -1); // on DAC and contents
       iSupply = iCalc;
    end
```

Supply current is **`iCalc`**. The bit-line currents are set by indexing the **`scaleFactor`** array with the enumerated state integer equivalent.

At the falling edge of **`wEN`**/**`rEN`**, **`iSupply`** is set back to **`iLeak`**, and the bit-line current drive is set to 0.

```
always @ (negedge wEN or negedge rEN) begin
    if ((wEN === 1'b0) && (rEN === 1'b0))
       iSupply = iLeak;  // Write / Read cycle ended, set supply current back to background
    blI = 0.0;
    blbI = 0.0;
 end
```

SystemVerilog Real IMC Bitcell (bitCell.sv)

The bit-line current gets assigned to the **bL** nets:

```
assign bL  = '{`wrealZState, blI, 0};  // contribute bitLine current
assign bLb = '{`wrealZState, blbI, 0}; // contribute bitLineB current
```

Some extra logic corrupts the contents if supply is insufficient:

```
always_comb begin
    if ( (vdd.V >= (vddNom*0.9)) && (vdd.V <= (vddNom*1.1)) )
       vddOn = 1'b1;
    else
       vddOn = 1'b0;
    if ( (vss.V >= (vssNom-10e-3)) && (vss.V <= (vssNom+10e-3)) )
       vssOn = 1'b1;
    else
       vssOn = 1'b0;
    if ((vddOn == 1'b1) && (vssOn == 1'b1))
       supplyOn = 1'b1;
    else
       supplyOn = 1'b0;
 end

 always @ (supplyOn) begin
    if (supplyOn == 1'b0)
       content = 1'bx;
  end
```

SystemVerilog Real IMC WL DAC (WLDAC.sv)

```
module WLDAC (
  inout EEnet vdd, vss,
  input logic [2:0] dacIn,
  input logic WLdrv,
  output WLnet WL
);
```

`WLdrv=1` sets the DAC state to ON for direct write/read to the memory.
The internal state of the DAC must match the possible states for the WLnet:

```
enum {OFF, s0, s1, s2, s3, s4, s5, s6, s7, ON} WLint;
```

There is logic to determine the supply state (not shown), and **`iSupply`** is set accordingly:

```
parameter real iActive = 1e-8; // DAC on Q-current
always @ (supplyOn) begin
    if (supplyOn == 1'b1)
       iSupply = iActive;
    else
       iSupply = 0.0;
  end
```

SystemVerilog Real IMC WL DAC (WLDAC.sv)

```
// Main DAC function
parameter real iSettle = 1e-6; // current needed to update DAC
 always @ (dacIn or WLdrv or supplyOn) begin
    if ((WLdrv == 1'b0) || (supplyOn == 1'b0)) begin
       // drive is logic 0, or DAC is OFF
       WLint = OFF;
       iChange = 0.0;
    end
    if (WLdrv == 1'b1) begin
       // drive is logic 1
       WLint = ON;
       iChange = iSettle;
       #10ps iChange = 0.0;
    end
    else begin // WLdrv is X or Z
       WLint = OFF;
       WLint = WLint.next(int'(dacIn+1));
       iChange = iSettle;
       #10ps iChange = 0.0;
    end
 end
```

DAC idle states
slewing current is 0

DAC memory write/read state
slewing current is high

DAC “analog” operation, sets a state s0 – s7
slewing current is high

SystemVerilog Real IMC WL DAC (WLDAC.sv)

Assign the output and supply ports:

```
assign WL = '{WLint};
assign vdd = '{`wrealZState, -iSupply-iChange, 0.0};
assign vss = '{`wrealZState,  iSupply+iChange, 0.0};
```

Memory Structure – bitcell Array

parameterized

```
module bitCellRow # (COLUMNS=8)
(
  inout EEnet VDD, VSS,
  inout EEnet  bL [COLUMNS-1:0], bLb [COLUMNS-1:0],
  input  WLnet WL,
  input  logic wEN, rEN
);
  genvar i;
  generate // a row of bitcells COLUMN wide
     for (i=0; i<COLUMNS; i++) begin : Columns
        bitCell xBitCell (VDD, VSS, bL[i], bLb[i], WL, wEN, rEN);
     end
  endgenerate

endmodule
```

Each row has **`COLUMN`** instances of **`bitCell`**

Memory Structure – bitcell Array

parameterized

```
module bitCellArray
# (ROWS=8, COLUMNS=8)
(
  inout EEnet VDD, VSS,
  input logic [2:0] DACIN [ROWS-1:0],
  input logic [ROWS-1:0] WLdrv,
  inout EEnet bL [COLUMNS-1:0], bLb [COLUMNS-1:0],
  input logic wEN, rEN
);
  genvar i, j;
  WLnet WL [ROWS-1:0];

  generate
     for (i=0; i<ROWS; i++) begin : Rows
        WLDAC xWLDAC (VDD, VSS, DACIN[i], WLdrv[i], WL[i]); // One driving DAC per row
        bitCellRow #(COLUMNS) xbitCellRow (VDD, VSS, bL[COLUMNS-1:0], bLb[COLUMNS-1:0], WL[i], wEN, rEN);
     end
  endgenerate

endmodule
```

The array is built from **`ROWS`** instances of **`bitCellRow`** and **`WLDAC`**
elements of **`bL`** and **`bLb`** span the rows

Memory Structure – top level (imcRAM.sv)

```
for (k=0; k<COLUMNS; k++) begin : bitLineTerminations

     VRsrcG #(.tr(1e-10), .Kinc(1e-9)) bLres (
       .P(bL[k]),
       .vval( ),
       .rval(rTerm),
       .imeas(ibL[k])
     );

     VRsrcG #(.tr(1e-10), .Kinc(1e-9)) bLbres (
       .P(bLb[k]),
       .vval( ),
       .rval(rTerm),
       .imeas(ibLb[k])
     );
  end


  always @ (negedge rEN) begin
     for (n=0; n<COLUMNS; n++) begin
        // The real summed value is discretize to a 1-bit value per column
        bLreadInt[n] = ((bL[n].V - bLb[n].V) > (0.0)) ? 1'b1 : 1'b0;
     end
  end
```

(Head-end current sources not modeled)

1-bit ADC for read-out is the same for product and content reading

Results

```
 Loading matrix value A =
67
97
bd
d6
a9
a2
0b
1d
```

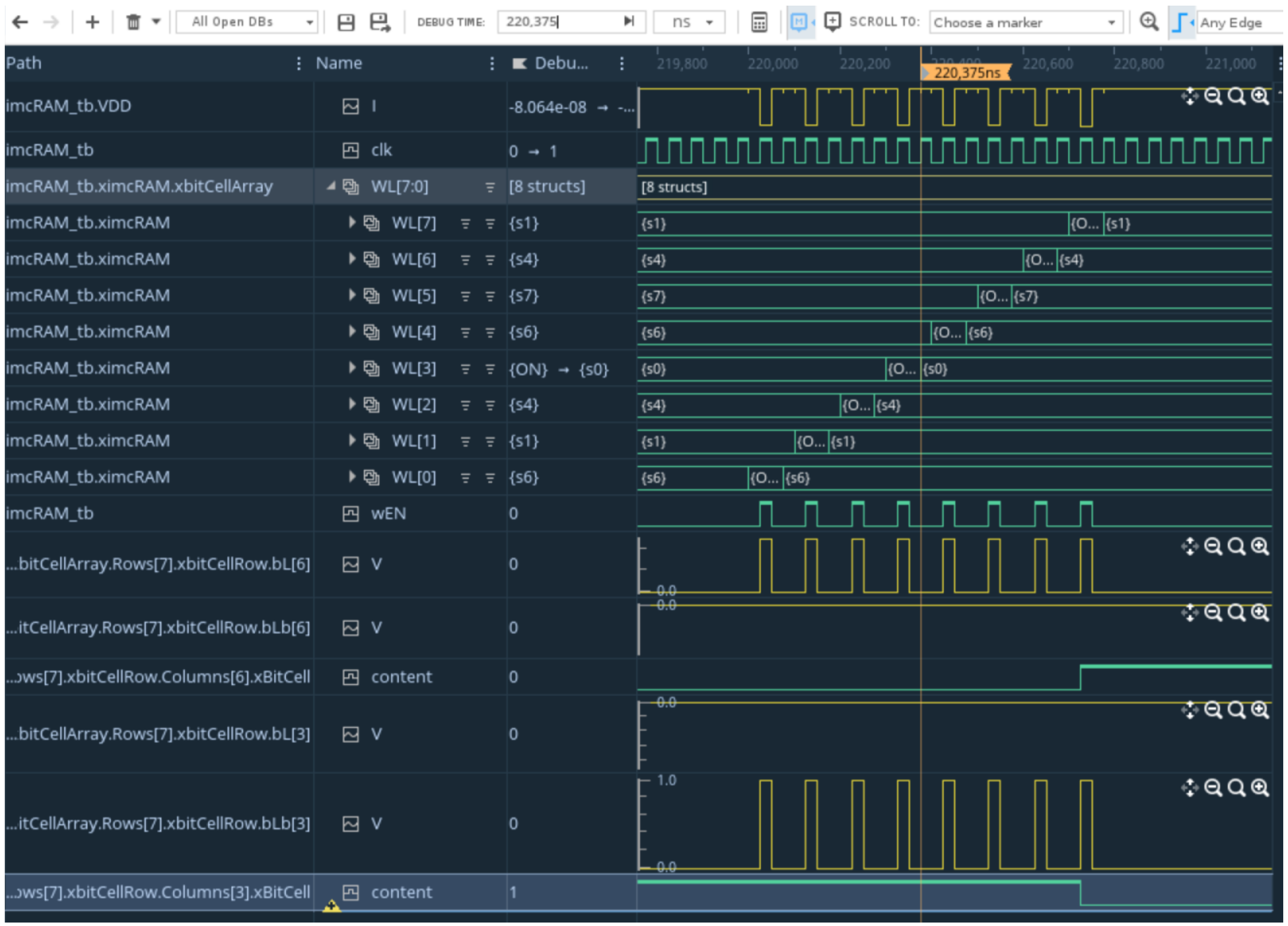
```
 Multiplying by vector value B =
2
2
5
6
0
5
7
0
Got result C = 87
```

- Total simulated time : 250 µs,
- Tasks simulated : load memory, 24 MAC operations, read out memory contents
- CPU time : under 1 second
- Memory used : 245M

Waveforms

write
cycle

row 7
col 6



DEBUG TIME: 220,375 ns | SCROLL TO: Choose a marker | Any Edge

Cursor: 220,375ns

Path	Name	Debu...
imcRAM_tb.VDD	I	-8.064e-08 → -...
imcRAM_tb	clk	0 → 1
imcRAM_tb.ximcRAM.xbitCellArray	WL[7:0]	[8 structs]
imcRAM_tb.ximcRAM	WL[7]	{s1}
imcRAM_tb.ximcRAM	WL[6]	{s4}
imcRAM_tb.ximcRAM	WL[5]	{s7}
imcRAM_tb.ximcRAM	WL[4]	{s6}
imcRAM_tb.ximcRAM	WL[3]	{ON} → {s0}
imcRAM_tb.ximcRAM	WL[2]	{s4}
imcRAM_tb.ximcRAM	WL[1]	{s1}
imcRAM_tb.ximcRAM	WL[0]	{s6}
imcRAM_tb	wEN	0
...bitCellArray.Rows[7].xbitCellRow.bL[6]	V	0
...itCellArray.Rows[7].xbitCellRow.bLb[6]	V	0
...ows[7].xbitCellRow.Columns[6].xBitCell	content	0
...bitCellArray.Rows[7].xbitCellRow.bL[3]	V	0
...itCellArray.Rows[7].xbitCellRow.bLb[3]	V	0
...ows[7].xbitCellRow.Columns[3].xBitCell	content	1

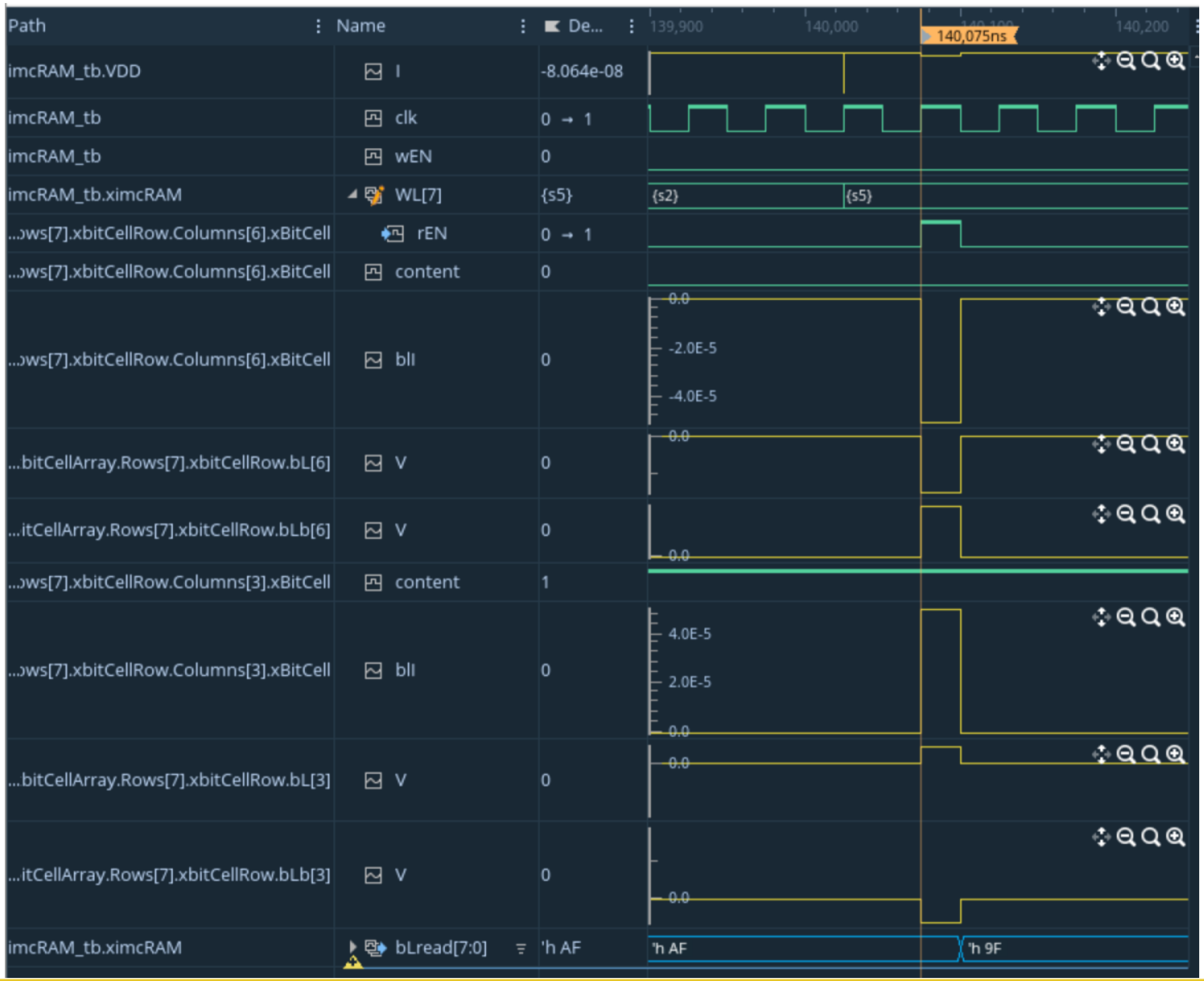
Waveforms

MAC cycle

WL7 drive

row 7 col 6

row 7 col 3



Path	Name	De...
imcRAM_tb.VDD	I	-8.064e-08
imcRAM_tb	clk	0 → 1
imcRAM_tb	wEN	0
imcRAM_tb.ximcRAM	WL[7]	{s5}
...ows[7].xbitCellRow.Columns[6].xBitCell	rEN	0 → 1
...ows[7].xbitCellRow.Columns[6].xBitCell	content	0
...ows[7].xbitCellRow.Columns[6].xBitCell	bll	0
...bitCellArray.Rows[7].xbitCellRow.bL[6]	V	0
...itCellArray.Rows[7].xbitCellRow.bLb[6]	V	0
...ows[7].xbitCellRow.Columns[3].xBitCell	content	1
...ows[7].xbitCellRow.Columns[3].xBitCell	bll	0
...bitCellArray.Rows[7].xbitCellRow.bL[3]	V	0
...itCellArray.Rows[7].xbitCellRow.bLb[3]	V	0
imcRAM_tb.ximcRAM	bLread[7:0]	'h AF

Conclusion

- In-Memory Computing can save power in Machine Learning applications
- IMC circuits are mixed-signal in nature and can be modeled using Real Numbers in SystemVerilog
- Real IMC models can capture fine details of behavior, including current summing, analog delays, and peak and average power consumption
- The real valued IMC model demonstrated acceptable simulation speed.
- Simulation time will scale linearly with memory size (unlike a device-level electrical simulation, which might scale exponentially).

References / Acknowledgements

Verma, et.al., “In-Memory Computing: Advances and Prospects,” IEEE Solid State Circuits Magazine, Summer 2019 Vol 11 No 3, p. 44 (original inspiration)

A. S. Glassner, Deep learning: A visual approach. San Francisco, CA: No Starch Press, Inc, 2021. (General neural network information)

R. Sanborn, R. Mitra, Z. Fan, “Best Practices for Verifying Mixed-Signal Systems”, Cadence Application Notes, USA and Canada, 2018. (for EEnet information) (*https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/technology-on-tour/secured/analog-mixed-signal/07-best-practices-for-verifying-ms-systems-sanborn-mitra-fan-cp.pdf*)

Paul McLellan’s blog *Breakfast Bytes*, (*https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes*) particularly these entries:

- How Is Google So Good at Recognizing Cats?
- Embedded Neural Network Summit—How to Build a Silicon Brain

Questions

- Can I get your full demonstration to run on my own?

  https://support.cadence.com/apex/ArticleAttachmentPortal?id=a1O3w00000AC1EREA1

  (Cadence Support Account required)

- How do these models compare in behavior and/or simulation performance with a schematic version?

  We didn't have (or build) a schematic version to compare to – sorry!

- Other Questions?