SystemVerilog Real Models for an In-Memory Compute Design

Daniel Cross
Cadence Design Systems
+1-512-342-5560
12301 Research Blvd Ste 200
Austin, TX 78759
danielcr@cadence.com

***Abstract*-Artificial Intelligence / Machine Learning (AI/ML) applications rely on repeated Multiply and Accumulate (MAC) operations for training and inference. While many energy efficient circuits for performing the MAC operations exist, for many AI/ML systems the limiting factor for total power consumption is the cost of moving data from memory to the calculating unit. In-Memory Compute (IMC) is a strategy that seeks to reduce the cost of moving data by doing computation within the memory unit itself. Many IMC solutions are fundamentally mixed-signal in nature, and accurate prediction of system behavior and power consumption often requires expensive and time consuming analog simulations. In this work, a set of real valued SystemVerilog models for emulating the behavior of an IMC design are presented. System components modelled include the Static RAM bit cell, DAC, and ADC. These models provide a way to simulate a large IMC system with nearly analog accuracy but at nearly digital speed, and provide full-system power consumption estimates under realistic compute load scenarios.**

I. INTRODUCTION

A. Neurons for Artificial Intelligence

Circuits for Artificial Intelligence and Machine Learning (AI/ML) are composed primarily of Convolutional Neural Networks (CNNs). The fundamental building block of CNNs is the neuron, whose operation can be represented mathematically as

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right),$$ [5]

in which an array of input values x_j are multiplied by a set of weights w_{kj}, the results summed, and the sum passed through a so-called activation function φ() to determine the neuron's response y_k. This is a Multiply and ACcumulate (MAC) operation.

B. Impetus for In-Memory-Computing

Due to the importance and centrality of the MAC operation in AI/ML, many performance-optimized implementations of MAC circuits have been designed in recent years. These developments have shifted the performance bottleneck in AI/ML systems to other necessary processes, particularly the process of moving weights and computation values between memory and the MAC core. A brief survey of published literature produced the estimates of the amount of energy consumed by memory access that are collected in Table I.

TABLE I
ESTIMATES OF ENERGY PER MEMORY ACCESS

Energy per Access (J)	Size of Access (bits)	Read (R) or Write (W)	Source
10n – 50n		R/W	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.
640p	32	R	Murmann, "Mixed-Signal Techniques for Embedded Machine Learning Systems," presented at CERN EP-ESE Electronics Seminar, Aug 2019
2n	64		Verma, et.al., "In-Memory Computing: Advances and Prospects," IEEE Solid State Circuits Magazine, Summer 2019 Vol 11 No 3, p. 44
20p	1		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.

Thus, we can estimate that a single bit memory access requires on the order of 25 pJ of energy.

In order to optimize the total system power consumption, some designers are focusing on methods of performing the MAC operations directly within the memory. This is termed In-Memory Computing (IMC), or alternately Computing In Memory (CIM). This paper reviews one particular type of circuit used for IMC, demonstrates that it is a mixed-signal circuit, and describes how a Real Number Modeling (RNM) methodology can be used to perform detailed functional simulations of a system containing IMC with performance approaching that of a pure digital simulation. We first present the circuit to be modeled, then describe SystemVerilog language constructs that model the circuit behavior, and present simulation results and performance data indicating that both the level of detail and simulation throughput are sufficient for verifying IMC AI/ML systems and predicting their power consumption.

II. CIRCUIT TO BE MODELED

A. Description of Operation

The representative IMC subsystem to be modeled (from [3]) is depicted in block diagram form in Figure 1. It is a modified Static Random Access Memory (SRAM) in which the word lines (horizontal) are driven by Digital to Analog Converters (DACs) to values between and including the supply and ground, and the bit lines (vertical) are headed by current sources and terminated with resistors, in addition to the usual line buffers (not shown).

Figure 2 depicts a single bit cell of Figure 1 in detail. For normal operation, writing into the bit cell is accomplished by asserting a value (and its complement) on the bit-lines (BL and $\overline{\text{BL}}$) and asserting the word-line (WL). Reading the stored value is done by raising WL while sensing BL and $\overline{\text{BL}}$. However, if instead of driving WL completely on (all the way to the supply) 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 product of the bias voltage and the value stored in the bit cell. Thus, by driving the word line with a DAC, we can effectively multiply the stored bit cell contents with a multi-bit value (the DAC input), and sense the results by measuring the relative amount of leakage current. Since all of the bit cells in a column share a bit line, their leakage currents (representing the products of interest) will sum together, thereby performing the accumulation part of the MAC operation. The total summed current on each bit line flows in the terminating resistor of that bit line, converting the MAC result into a voltage that can be rendered back into the digital domain with an Analog to Digital Converter (ADC), as shown in the lower right of Figure 1.

We assume an M x N bitcell array with DAC resolution j bits and ADC resolution k bits. Currents will sum on the vertical bit-lines where a '0' value stored in the bit cell will contribute a negative current to the bit line and a '1' value stored in the bit cell will contribute a positive current. This performs the needed MAC function (shown here without the activation function φ and using *i* as the summation index):

$$\sum_{i=0}^{M} w_{ki} x_i$$

The weights w_{ki} are *N* bits wide, the input vectors x_i are *j* bits wide, and the output is *N* x *k* bits wide.

For additional details of operation, the reader is referred to [3].

B. Design Considerations

The current sources at the top of the bit lines must source enough current to provide the maximum leakage sink current, which occurs when all the stored values in a column are '0' and the DAC is at its maximum value. The termination resistors should be sized such that the sourced current (assuming balanced '1' and '0' stored values in the column) produce a voltage at the center of the input range of the ADC. Word line drive for read and write operations can be provided by a special DAC setting, or by a separate line driver in parallel with the DAC. Similarly, bit line write drive can be provided by special purpose settings of the current sources, or by separate drivers in parallel.

III. SystemVerilog Code for Modeling Circuit Functions

To demonstrate the capability and advantages of using RNM for behavioral analysis and verification of IMC for AI/ML, we developed a set of RNM models in SystemVerilog that we then used to construct and simulate a sample IMC memory of the type described above.

A. User Defined Nettypes used for the IMC Models

The RNM models for this demonstration rely on the User Defined Nettype (UDN) capability defined in the 2012 version of the SystemVerilog standard [6]. UDNs are powerful modeling tools since they are treated as nets by the simulator, can carry nearly any data type or combination of data types, and can support a user defined function that can resolve the value of the net in the presence of multiple drivers. The latter is termed a User Defined Resolution function (UDR).

One UDN we call WLnet was designed specifically for the purpose of modeling the DAC-driven word lines. Its SystemVerilog definition is shown here.

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

The UDR is shown below.

```
function automatic t_WLnet res_WLnet (input t_WLnet driver[]);
  integer j, k, m;
  k = 0;

  foreach (driver[i]) begin
     k++;
  end

  res_WLnet.state = driver[0].state;

  if (k > 0) begin
    for (j=0;j<=k;j++) begin
       for (m=0;m<=k;m++) begin
          if (j != m) begin
             if (driver[j].state == OFF) begin
                if (driver[m].state != ON)
                   res_WLnet.state =  driver[j].state;
                else
                   res_WLnet.state =  'X;
             end
             if (driver[j].state == ON) begin
                if (driver[m].state != OFF)
                   res_WLnet.state =  driver[j].state;
                else
                   res_WLnet.state =  'X;
             end
             else if (driver[j].state > driver[m].state) begin
                 res_WLnet.state =  driver[j].state;
             end
             else begin
                 res_WLnet.state =  driver[m].state;
             end
          end
       end
    end
  end

endfunction : res_WLnet
```

The operation of the UDR is summarized in Table II.

Table II
SUMMARY OF WLNET UDR OPERATION

Driving Condition	Net Resolution
More than one driver = ON	ON
One or more drivers = ON, one or more drivers = OFF	X
Exactly one driver = ON, no drivers = OFF	ON
No drivers = ON, one or more drivers = OFF	OFF
One or more drivers = s*, no drivers ON or OFF	Greatest s* value driver wins

The resolution function was included to allow modeling of a separate read/write line driver in parallel with the DAC (as mentioned previously and shown in [3]), although that design approach was not followed for this demonstration. If separate line drivers were used, they would drive the WLnets with ON, OFF, or no drive while the DACs would drive the intermediate s* states.

Modeling of the current summing on the bit lines, as well as power consumption (current drain) on the supply ports (vdd, vss) was accomplished using the EEnet (Electrical Equivalent net) UDN developed by Cadence Design Systems ([7],[8]). This UDN has independent real elements V, I, and R that represent respectively voltage, current, and resistance of an electrical element. The reader is referred to [7], [8] for detailed information on the EEnet.

B. Modeling the Bit Cell

Supply current consumption for various states of operation are parameterized within the bit cell model, so that the effects of low-level circuit design decisions on total system power can be assessed. The model parameters are combined based on the dynamic operation of the bit cell, and the total is assigned to the `vdd` and `vss` EEnets. Code is shown here.

```
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};
```

A portion of the SystemVerilog behavioral description of the bit cell is as follows. Note that `iSupply`, the variable assign to the "`I`" element of the `vdd` and `vss` EEnets, is assigned the parameter values of `iWrite`, `iCalc`, or `iLeak` depending upon the state of the bit cell (continued in the next block of code as well).

```
module bitCell (
  inout EEnet vdd, vss, bL, bLb,
  input WLnet WL,
  input logic wEN, rEN
  );
logic content = 1'bx;
real scaleFactor [0:7] = '{0e-6, 10e-6, 20e-6, 30e-6, 40e-6, 50e-6, 60e-6, 70e-6};
// Write or Read cycle
always @ (posedge wEN or posedge rEN) begin
  // write-enable and read-enable are mutually exclusive
  #1ps          // delay for proper sampling of values
  if ((WL.state == OFF) || (WL.state == 1'bx)) begin  // bit cell OFF
    iSupply = iLeak;
    blI = 0.0;
    blbI = 0.0;
  end
  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;
  // Logic read; currents are max/min
  // depending on register contents
           blI  = (content == 1) ? scaleFactor[7] : -scaleFactor[7];
           blbI = (content == 1) ? -scaleFactor[7] : scaleFactor[7];
        end
  end
```

A logic value stores the contents of the bitcell (`content`). Leakage currents that map to the s0-s7 states of WL are defined in an array `scaleFactor`. A write cycle begins when the state on the word line is ON and the write enable bit `wEN` is set. The supply current is set to the value defined in `iWrite`, and the voltage values (`bL.V` and `bLb.V`) on the bit-lines are compared to a threshold value `blThd` and 1, 0, or X is stored in `content`. A read cycle occurs when the read enable bit `rEN` is set. The supply current is set to `iCalc`, and the currents to be driven onto the bit lines is set to max or min (`scaleFactor[7]`) depending on the stored contents.

If the read enable is asserted but the word line is in one of the intermediate states s0 – s7, the bit cell is being used as a multiplier, as modeled by this bit of SystemVerilog:

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

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

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

The bit line currents are set by indexing the `scaleFactor` array with the enumerated state integer equivalent. Supply current is set to `iCalc`. In the last two lines, the bit line currents set during a multiplication, normal read cycle, or standby event get assigned to the I (current) elements of the `bL` and `bLb` nets. The `wrealZState value assigned to the V (voltage) elements will allow the EEnet resolution function to determine the final voltage on the bit lines once the current contributions of all the bit cells are summed and the termination resistor is connected.

At the falling edge of `wEN` or `rEN`, `iSupply` is set to `iLeak`, and the bit line current drive is set to 0 (shown above).

C. Modeling the DAC

Since power optimization is one of the primary reasons for using IMC, a focus of our modeling for this demonstration is accurate representation of power supply currents in all the blocks. To realistically model the DAC supply current, we assumed that the DAC would consume most of its current while changing state. To model this behavior, we included parameters in the DAC model for power-on idle current (`iActive`) and output updating current (`iSettle`). Code that uses these parameters is shown here.

```
enum {OFF, s0, s1, s2, s3, s4, s5, s6, s7, ON} WLint;
always @ (supplyOn) begin
     if (supplyOn == 1'b1)
        iSupply = iActive;
     else
        iSupply = 0.0;
   end
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
     else 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 = WLint.next(int'(dacIn+1));
        iChange = iSettle;
        #10ps iChange = 0.0;
     end
  end
assign WL = '{WLint};
assign vdd = '{`wrealZState, -iSupply-iChange, 0.0};
assign vss = '{`wrealZState,  iSupply+iChange, 0.0};
```

For normal writing of logic values to the memory row, drive the `WLdrv` input to 0 or 1. The DAC model sets its `WLint` enumerated variable to `OFF` or `ON`. To use the memory row to multiply, float `WLdrv` (set to X or Z), and the DAC uses the `enum.next()` built-in method to select an appropriate analog output state based on the value of `dacIn`. Finally, `WLint` is assigned to the output `WL`. In each branch of the code (except OFF), the variable `iChange` is set to the value of `iSettle` for 10 ps. The `iChange` value is summed with the (relatively) static value of `iSupply` and assigned to the "`I`" elements of `vdd` and `vss`. This modeling style allows for a very detailed representation of the dynamic power consumption of the application while maintaining the simple enumerated representation of the DAC output.

D. Other Models, Assembling the Memory

Space constraints prohibit presentation of models of other system components. A uniform modeling philosophy, in keeping with that presented here, carries throughout the demonstration subsystem. Multiple `generate` statements are used to replicate the bit cell into an arbitrary number of rows and columns, as shown here.

```
// Build bitCell Rows first for inclusion into the array
// COLUMNS is a parameter

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

// This assembles rows of bitcells with WLDACs into an IMC array.
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;
  WLnet WL [ROWS-1:0];

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

IV. RESULTS

The model implemented omits the head-end current sources for simplicity, so the center voltage and thus the threshold for the comparators (1 bit ADCs) is 0V.

Figure 3 shows some signals in the demonstration SRAM model during a write cycle. The resolution function of the EEnet used to represent the supply (VDD) provides summation of all current flow out of the supply. The figure shows a small peak of current as the word line driving DACs update (seen as the nets WL[7:0] update), and then a much larger peak as the write current of all the bit cells activated during that cycle write the input to their contents. During the write cycle, the bit lines bL and bLb are driven to 0V or 1V (VDD) in accordance with the data to be stored. Near the bottom right of the figure, one can see two bit cell content holders updating.

Figure 4 shows some signals in the demonstration SRAM model during a MAC (read) cycle. Once again, the EEnet sums up the total current at VDD. For this operation, the DAC update peak is larger than the read current, which is primarily contributed to by the leakage current from each bit cell proportional to the word line bias driven by the DACs. The two bit cells (from column 6 and column 3) illustrated in the figure are both in row 7 and share a

word line, which is driven to state 5 (signal WL[7] in the figure). The leakage currents for each bit cell, represented by the blI signals, are both 5 uA, but since the bit in column 6 has a 0 stored, its leakage is negative. Notice that the bit line voltages (bL and bLb) depend on the leakage contributed by all the bit cells in their respective columns, and are different values (although their signs match that of the individual currents, that is just a coincidence). The final readout of the MAC operation, after passing through the row of comparators, is 0x9F. Notice that bit 6 is 0 and bit 3 is 1. This matches with the polarity of the column 6 bit line voltage (bL[6] = -0.28V) and the polarity of the column 3 bit line (bL[3] = 0.0625V).

The simulation test bench loaded a random 8x8 matrix of values ("weights") into the memory early in the simulation, then proceeded to multiply that by a series of random 3x8 vectors ("inputs") to produce a series of 1x8 output values. In one run, the 8x8 matrix was { x67, x97, xbd, xd6, xa9, xa2, x0b, x1d }, and as an example this was multiplied by { 2, 2, 5, 6, 0, 5, 7, 0 } with a result of x87.

The total simulated time was 250 µs, which enough to perform loading of the memory, 24 MAC operations, and reading out of the memory contents, ran in under 1 second and used 245M total core memory.

V. Conclusion

In-Memory Computing can be effective and 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. A real valued IMC model was presented and its operation described. These modeling methods have wide applicability. The real valued IMC model demonstrated acceptable simulation speed.

VI. References

[1] 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.

[2] Murmann, "Mixed-Signal Techniques for Embedded Machine Learning Systems," presented at CERN EP-ESE Electronics Seminar, Aug 2019

[3] Verma, et.al., "In-Memory Computing: Advances and Prospects," IEEE Solid State Circuits Magazine, Summer 2019 Vol 11 No 3, p. 44

[4] 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.

[5] A. S. Glassner, *Deep learning: A visual approach*. San Francisco, CA: No Starch Press, Inc, 2021.

[6] "IEEE 1800-2012 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language", IEEE Computer Society, USA, 2013.

[7] A. Caicedo and S. Fritz, "Enabling Digital Mixed-Signal Verification of Loading Effects in Power Regulation using SystemVerilog User-Defined Nettype", *DVCON Europe*, 2019.

[8] R. Sanborn, R. Mitra, Z. Fan, "Best Practices for Verifying Mixed-Signal Systems", *Cadence Application Notes*, USA and Canada, 2018. (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, retrieved 9/16/2022).

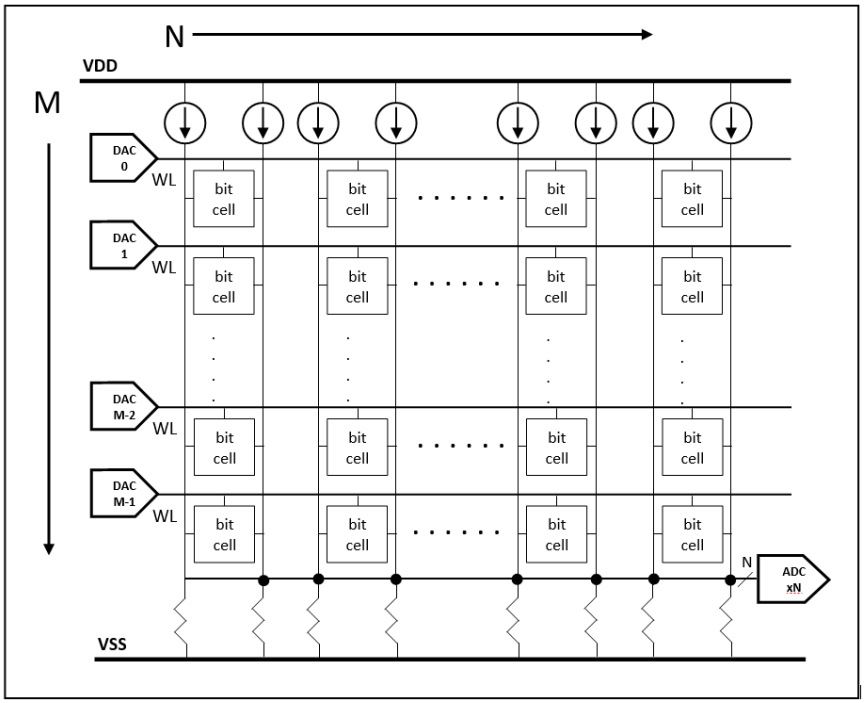

Figure 1. Diagram of the IMC subsystem to be modeled (based on [3]).

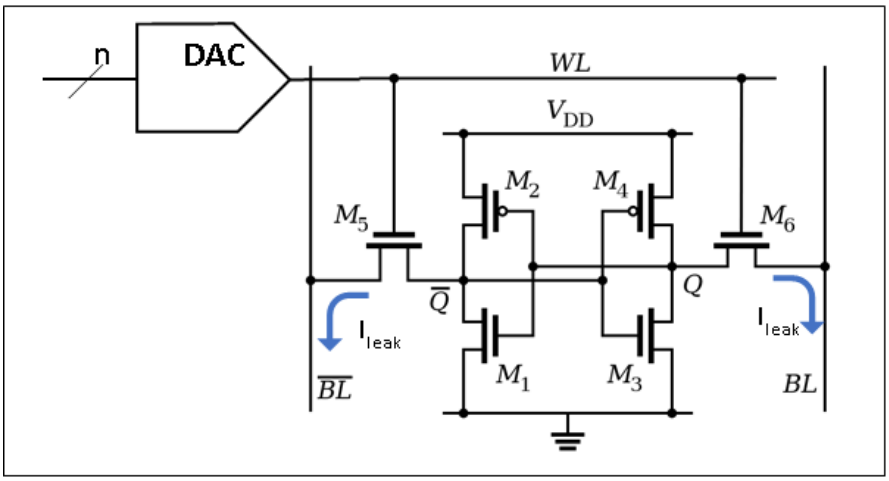

Figure 2. Schematic diagram of a single SRAM bit cell with DAC drive to the word line and showing the path of leakage currents.

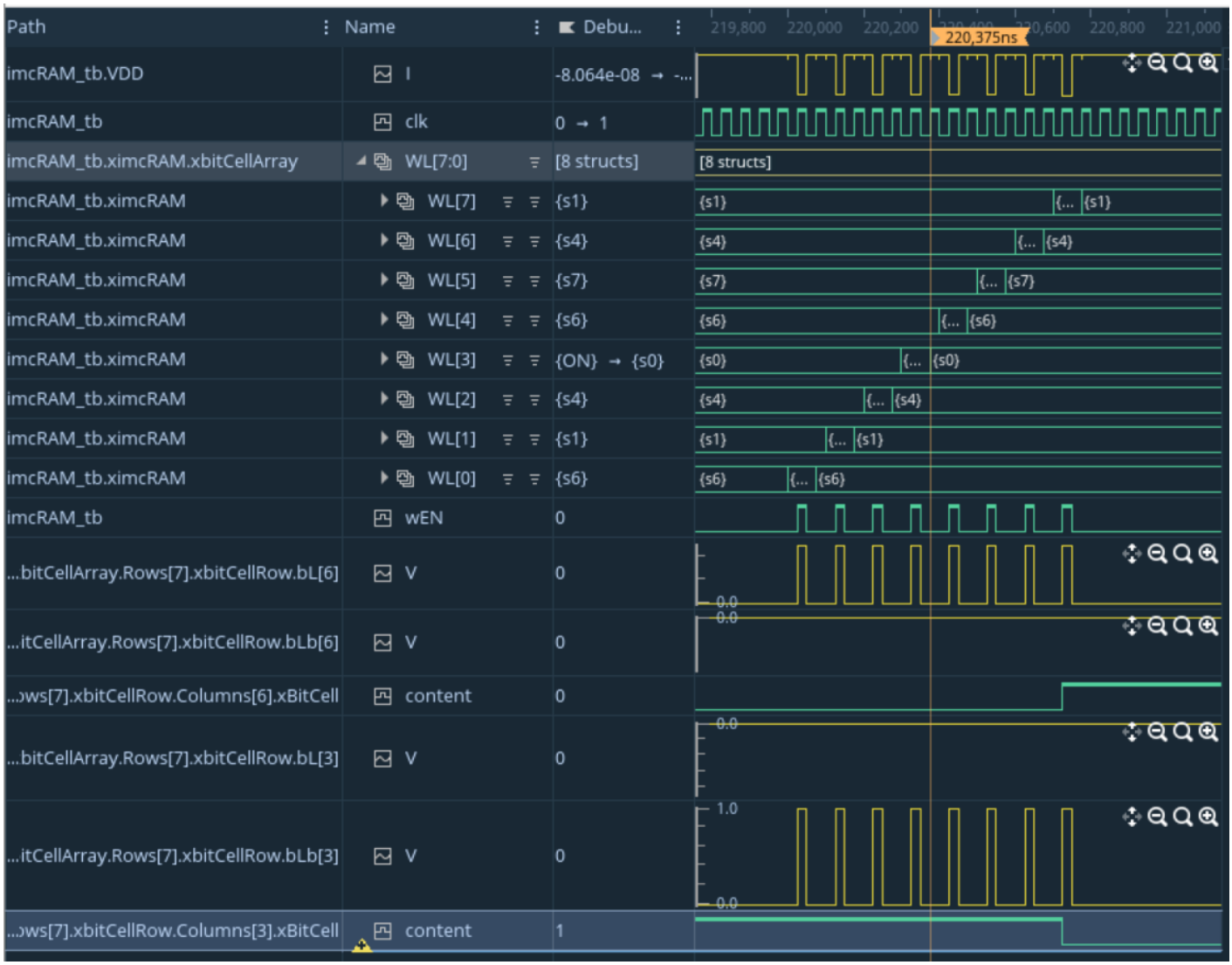

Figure 3. Some waveforms from a write cycle. VDD.I is amps, bL[*].V is volts.

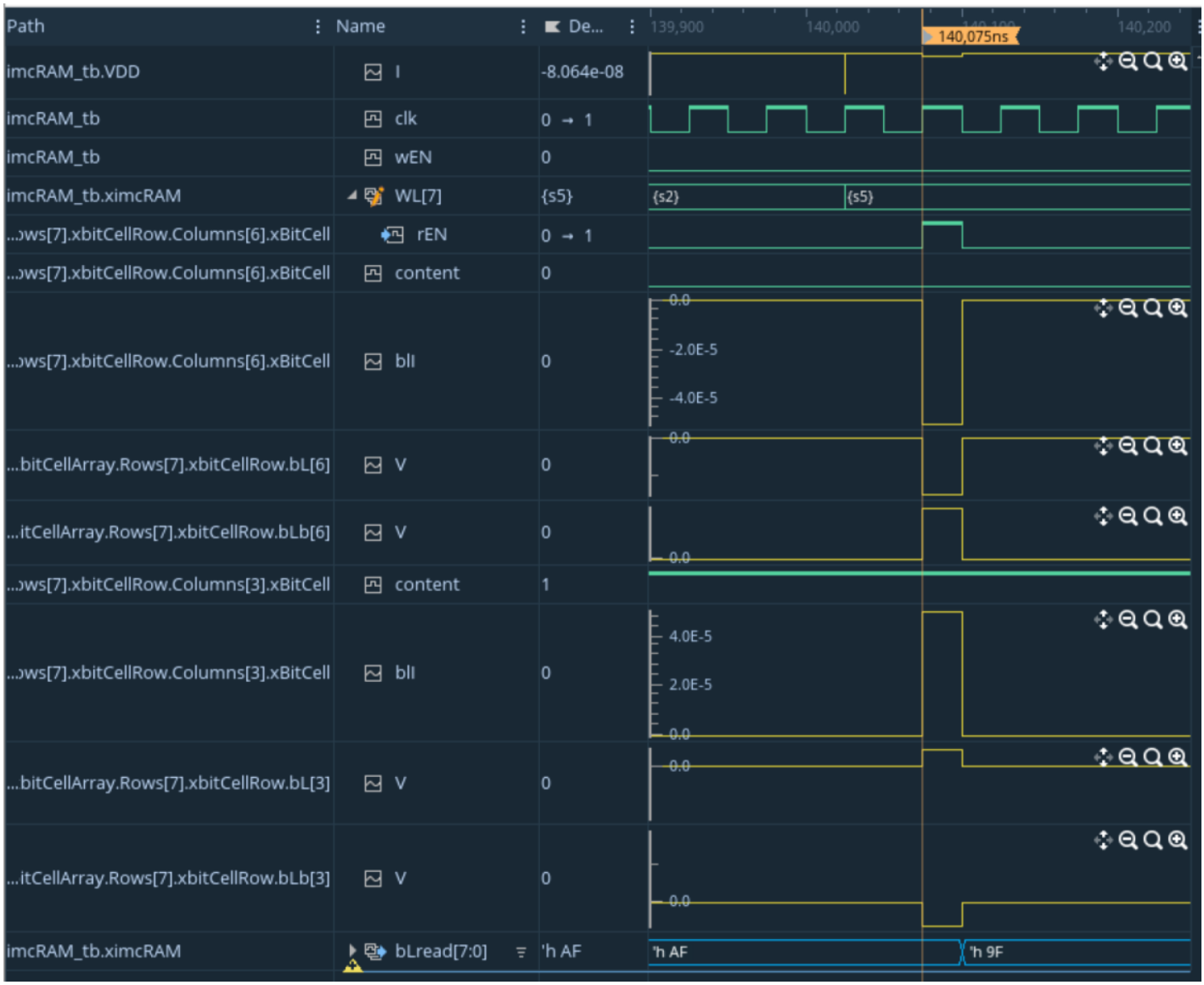

Figure 4. Some waveforms from a MAC cycle.