N-Body Accelerator

Isaac Trost (wit2102)
Kristian Nikolov (kdn2117)
Robert Pendergrast (rlp2153)
Moises Mata (mm6155)
Adib Khondoker (aak2250)

Project Overview

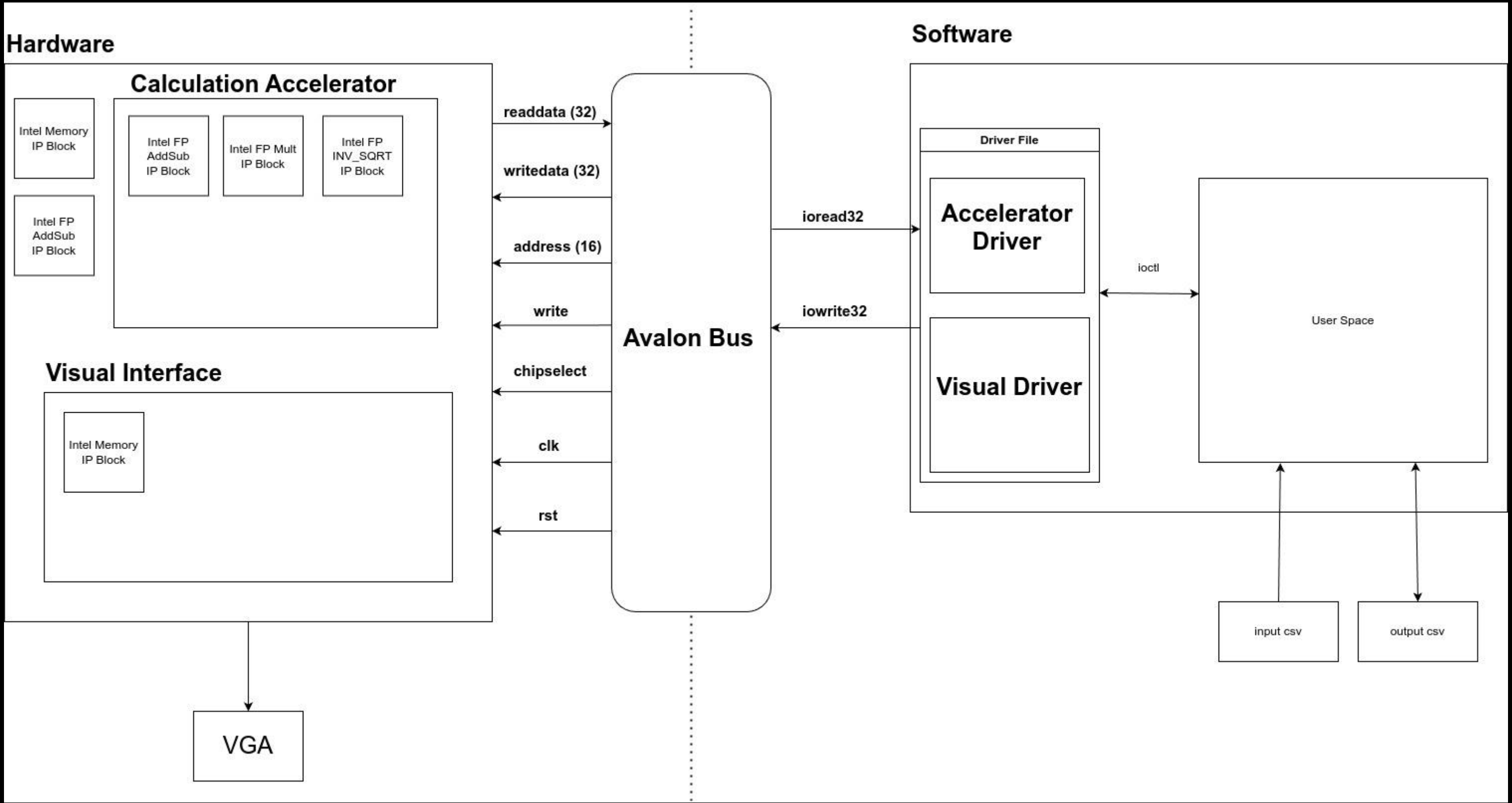
Background: N-Body calculations, used to determine the chaotic motions between *N* interacting masses, are critical in the fields of astrophysics, orbital dynamics, and molecular dynamics.

Problem: N-Body calculations are computationally expensive and SLOW, requiring many dividers and multipliers

GOAL: Develop a computational accelerator that can perform **N<512** Body simulations

Result... We fought the good fight and we came out victorious

High-Level System Design



nbody.sv

SW_READ_WRITE	Waits for software writes via bus (X,Y,VX,VY,M), IDLE STATE
CALC_ACCEL	Iterates body pairs, computes acceleration with getAccl, updates velocity
Update_POS	Update positions using Leapfrog step

GO, READ	Control Handshake
DONE	High when sim step completes
addr[15:9]	Select reg/mem
addr[8:0]	Select body index
write/readdata	32bit avalon bus IO

getaccel.sv

Signal	Width	Purpose
x1/y1	64	Position body 1
x2/y2	64	Position body 2
m2	64	Mass body 2 (pre mult by G)
ax/ay	64	Acc on body 1 from body 2
Clk, rst	1	Clock / synchronous reset

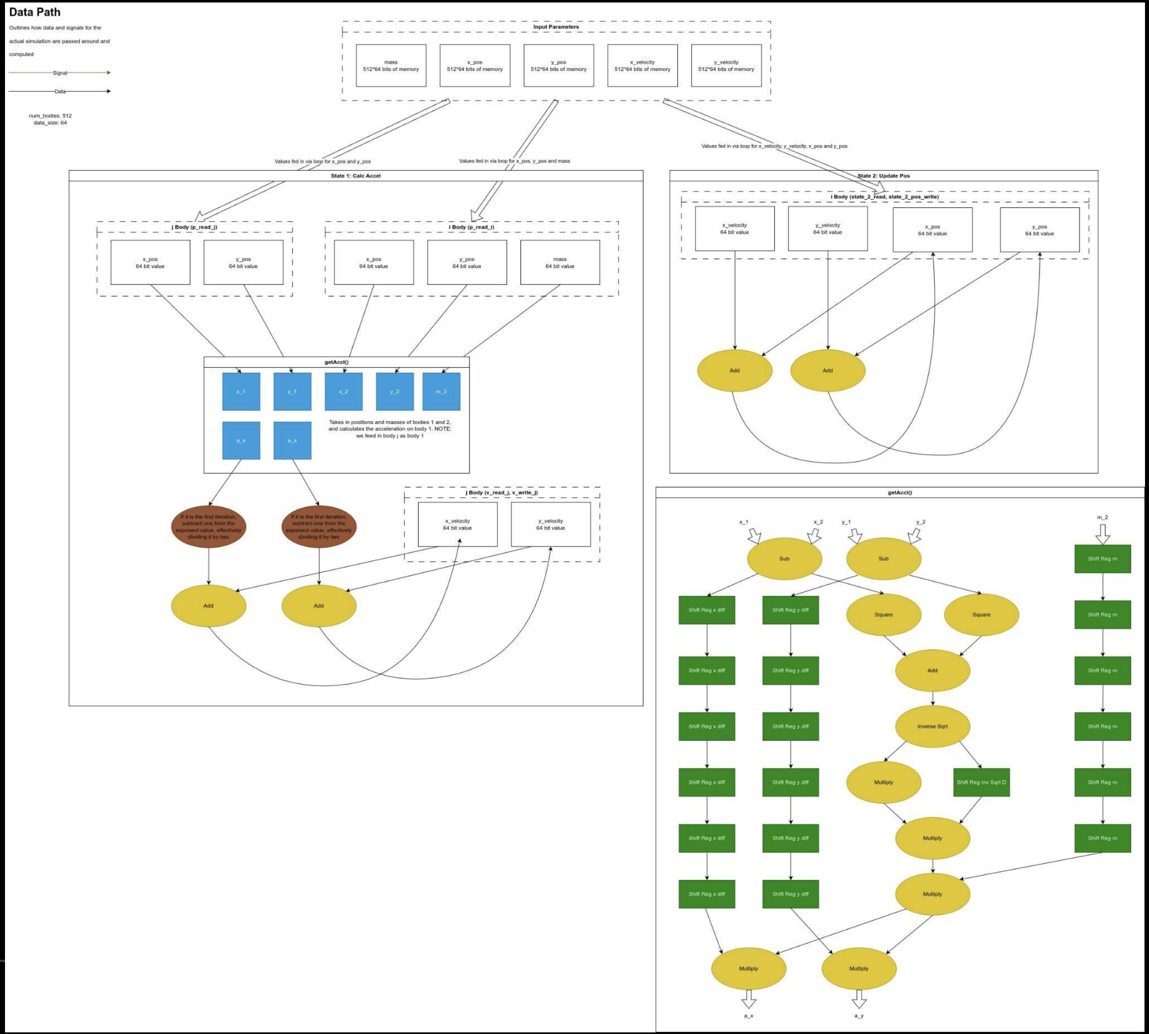
AddSub	3 units	Subtract, add
Mult	6 Units	Square, cube, scale
InvSqrt	1 unit	Inv sqr root
shift_reg	4 units	Align pipeline timing

MultTime	11 cycles
AddTime	20 cycles
InvSqrtTime	27 Cycles

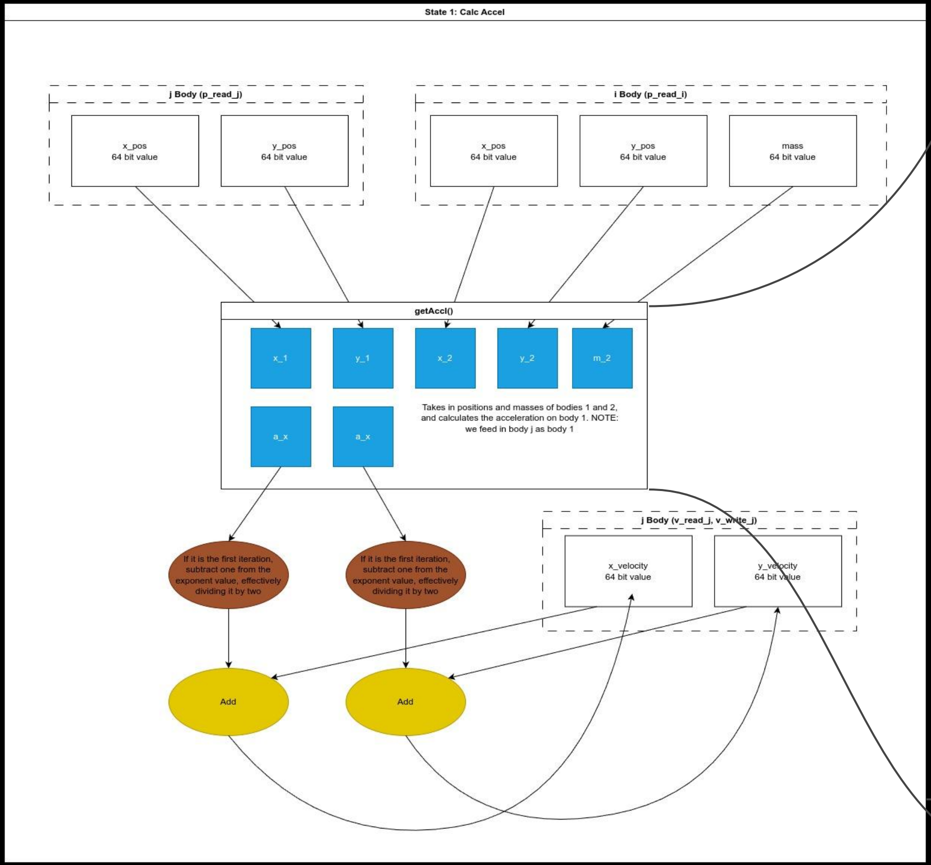
- Computes
  - $a_x = m_2 \cdot (x_1 - x_2) / |r|^3$
  - $a_y = m_2 \cdot (y_1 - y_2) / |r|^3$

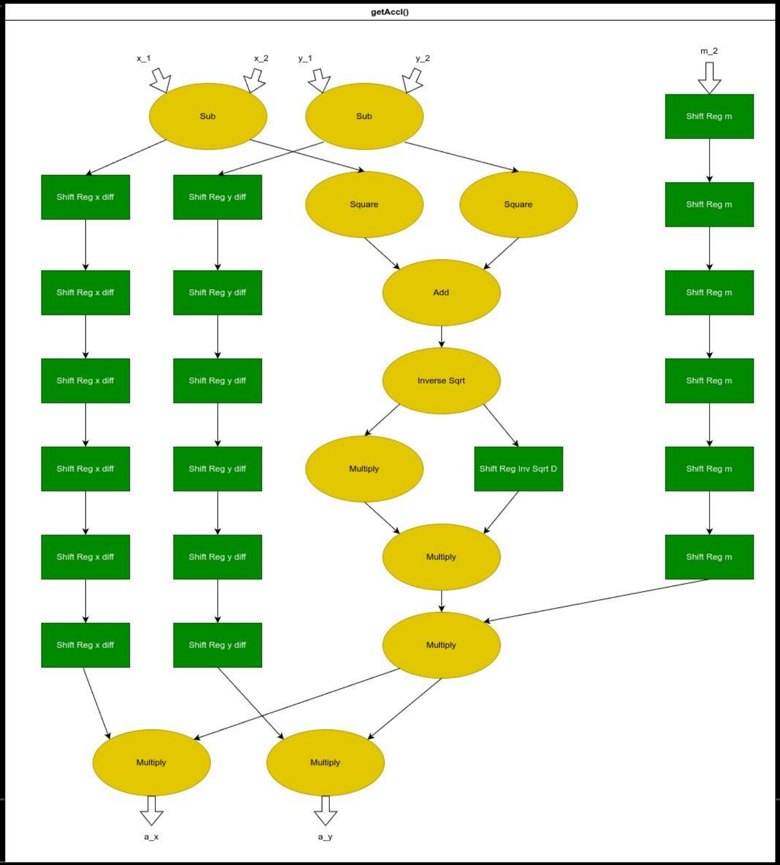
Accelerator Logic

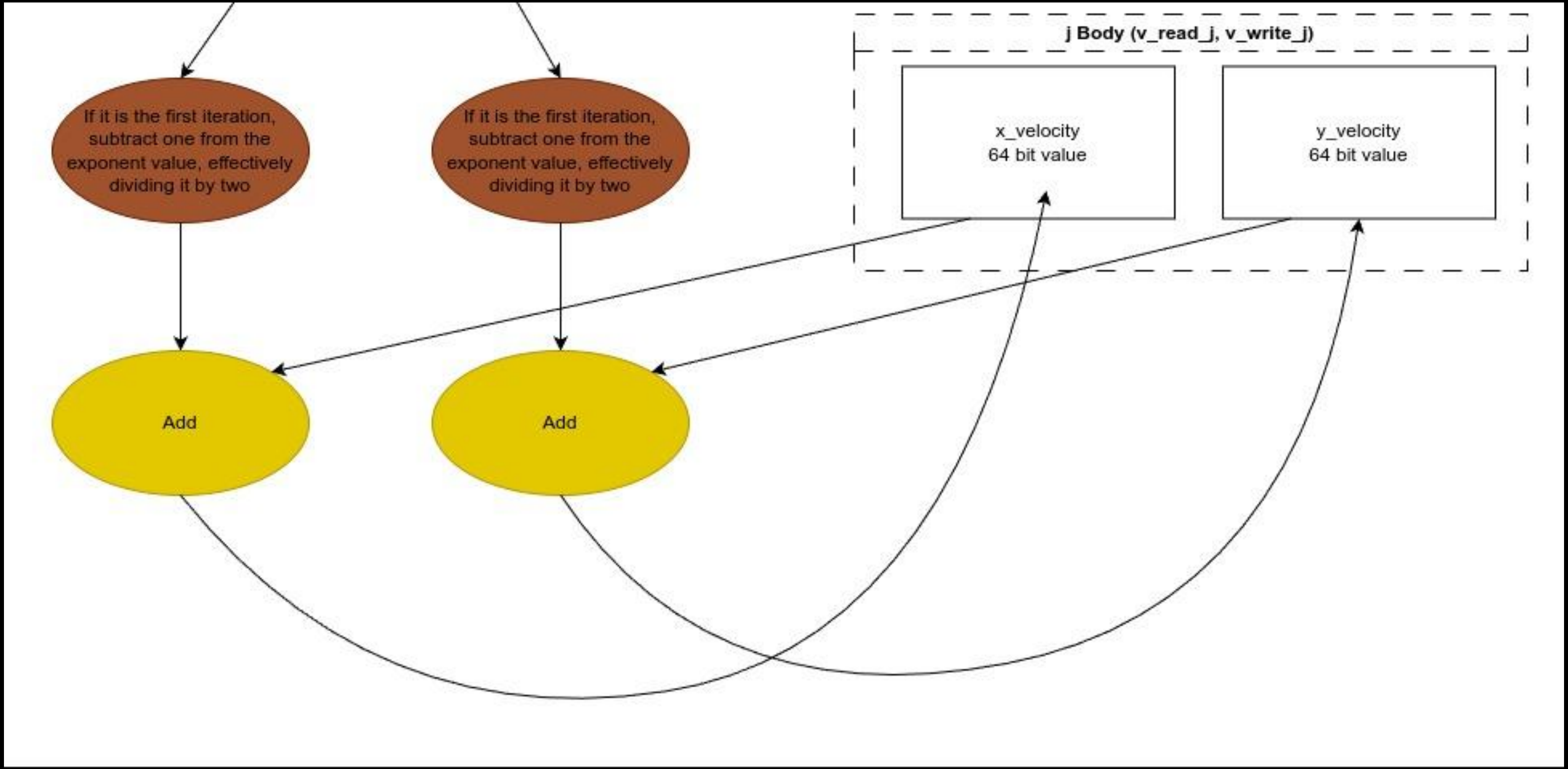
- SW Read/Write
- Calculate Acceleration and update velocity
- Update position

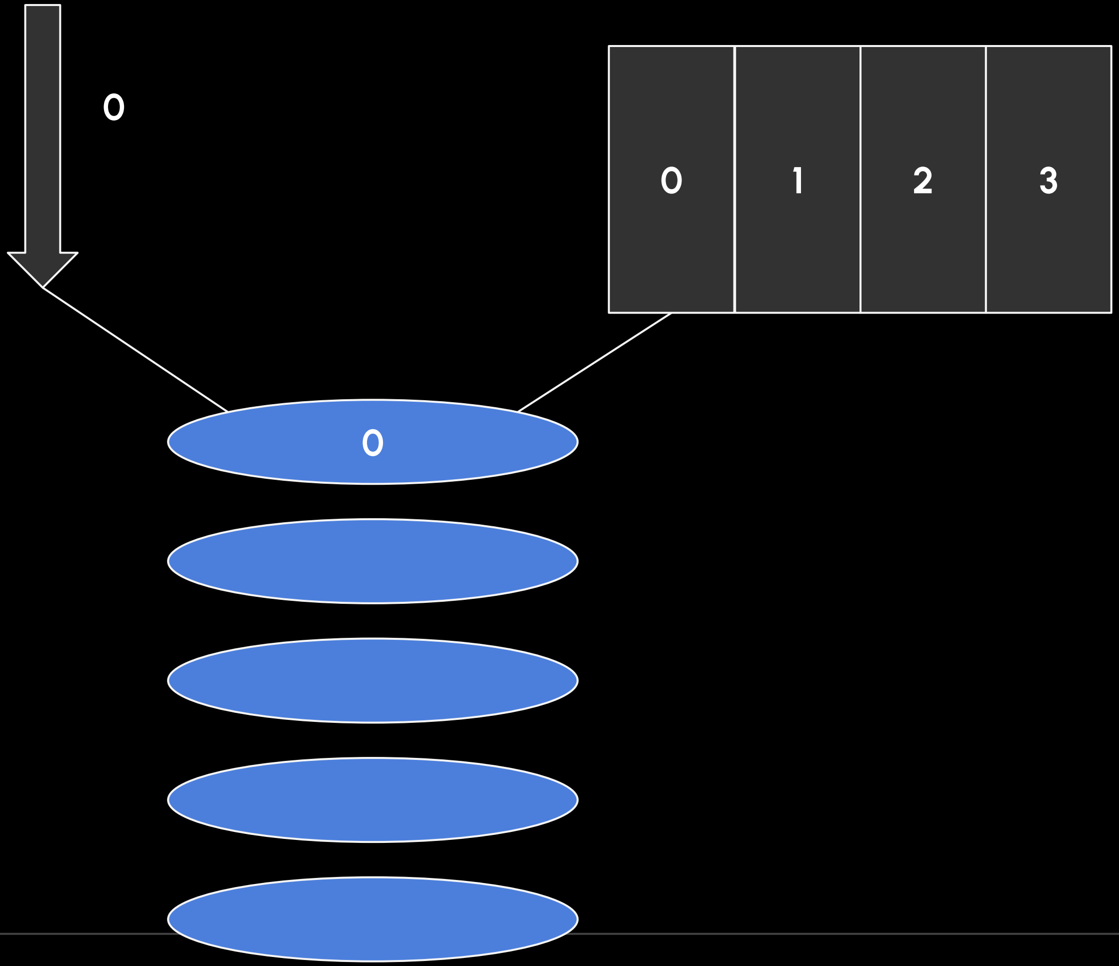


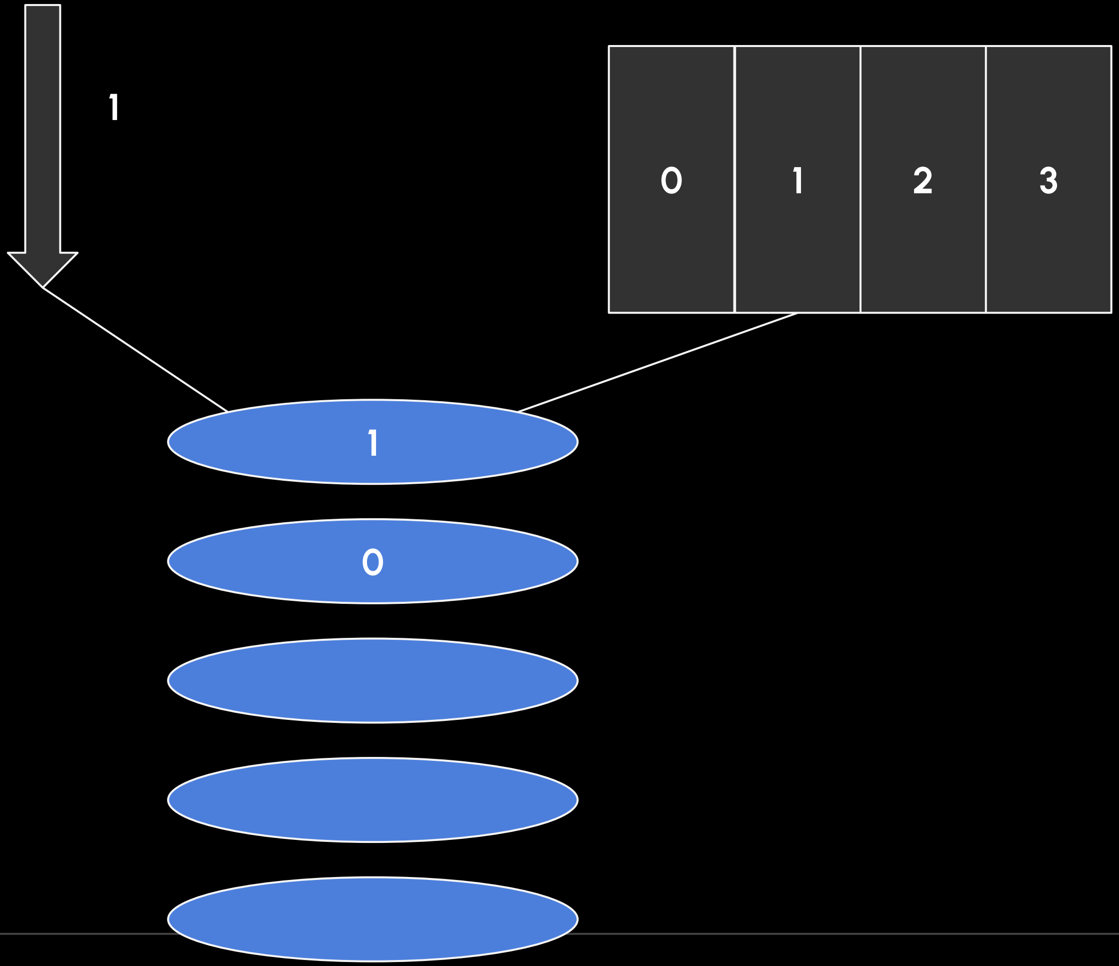
Calculating Acceleration and Velocity

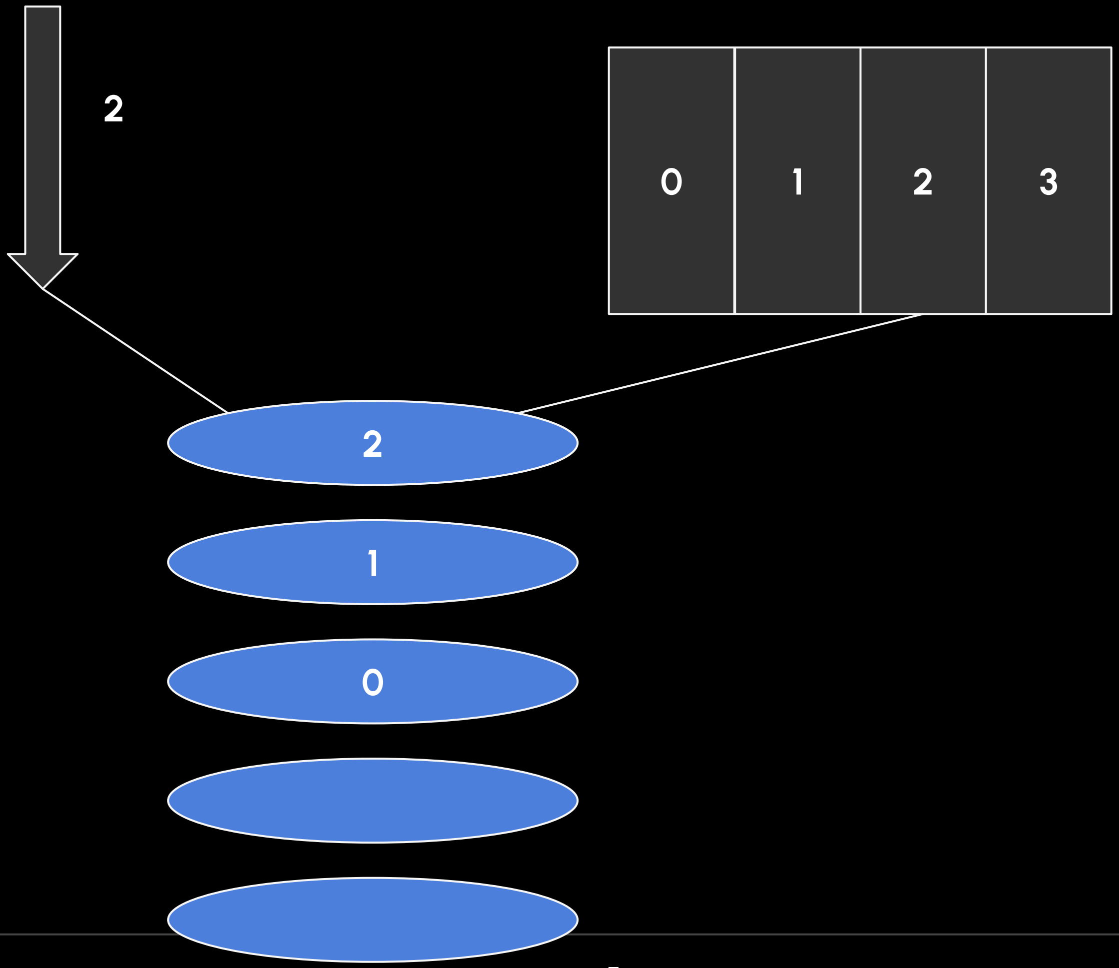


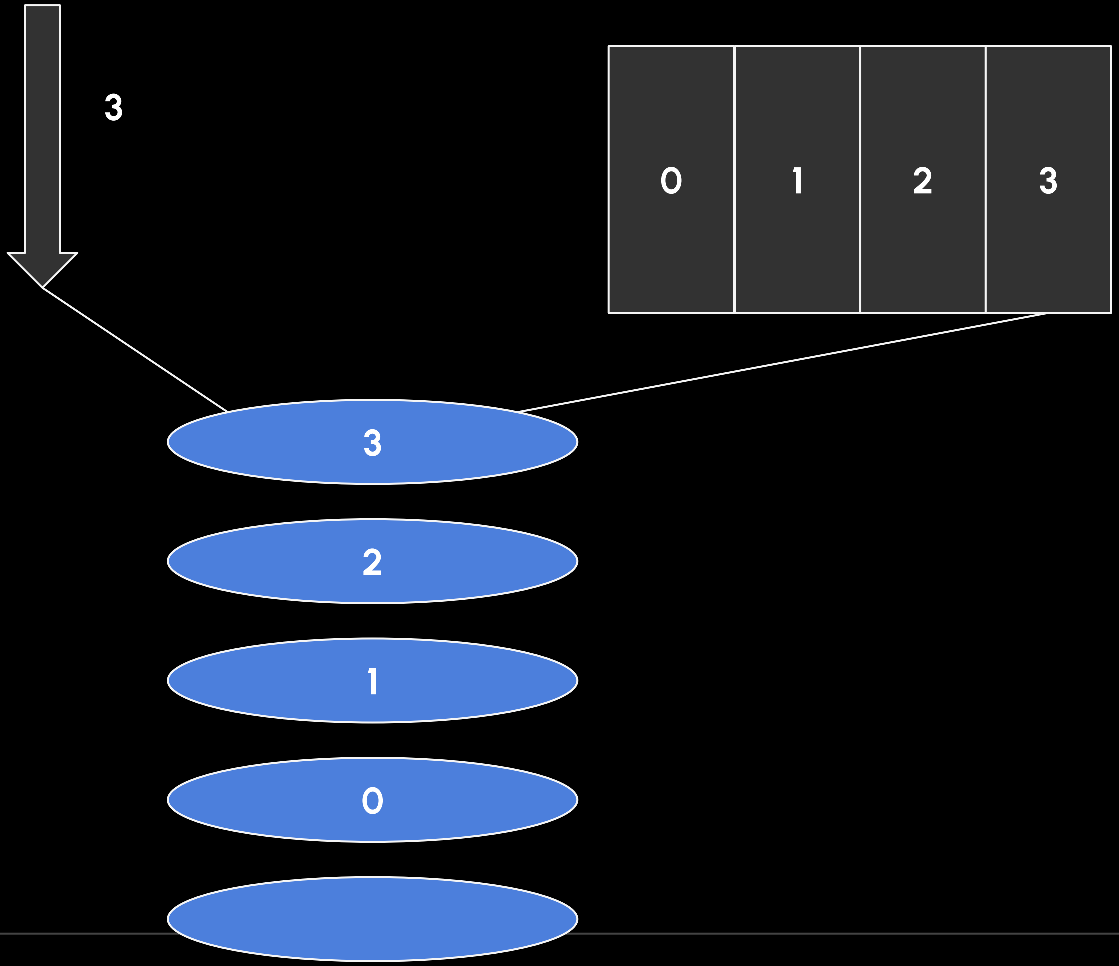


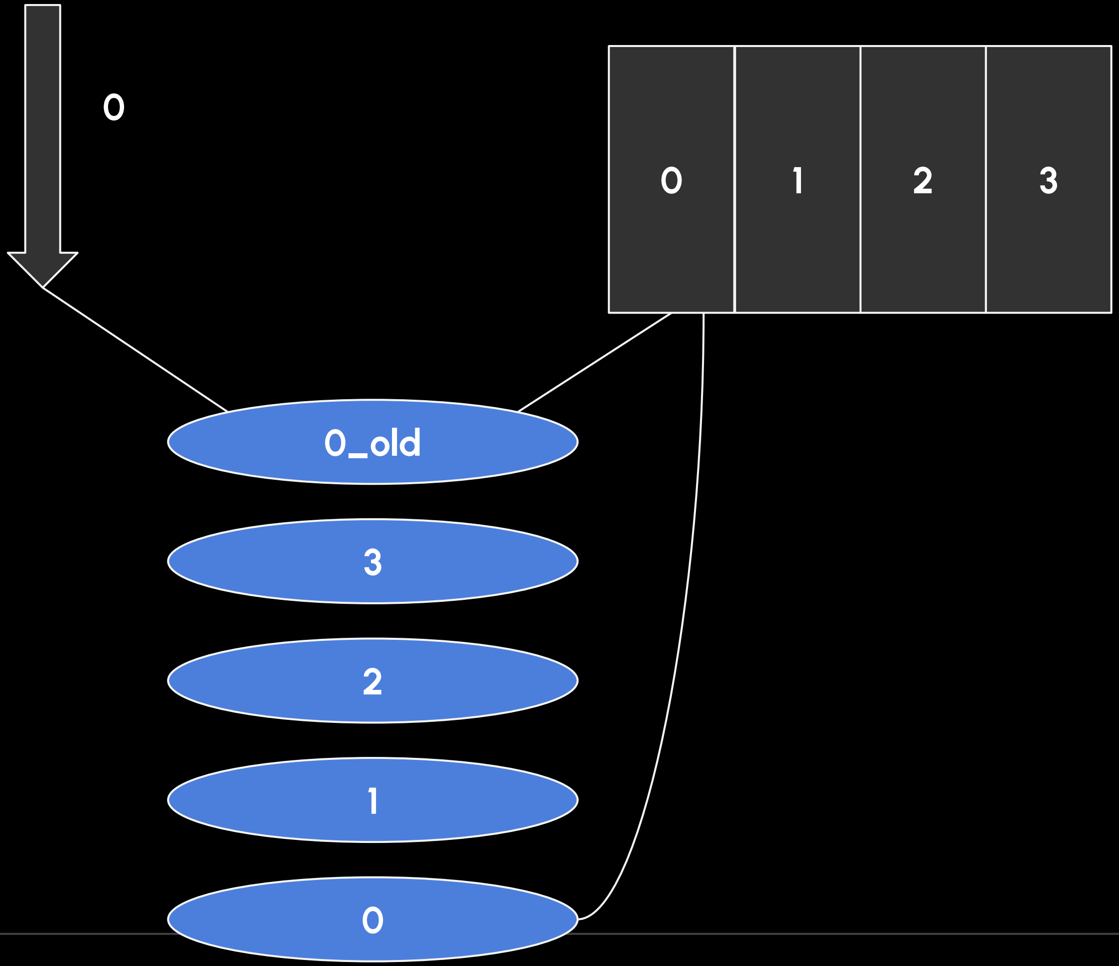

j Body (v_read_j, v_write_j)
x_velocity
64 bit value
y_velocity
64 bit value
If it is the first iteration, subtract one from the exponent value, effectively dividing it by two
If it is the first iteration, subtract one from the exponent value, effectively dividing it by two
Add
Add


0
0
1
2
3
0

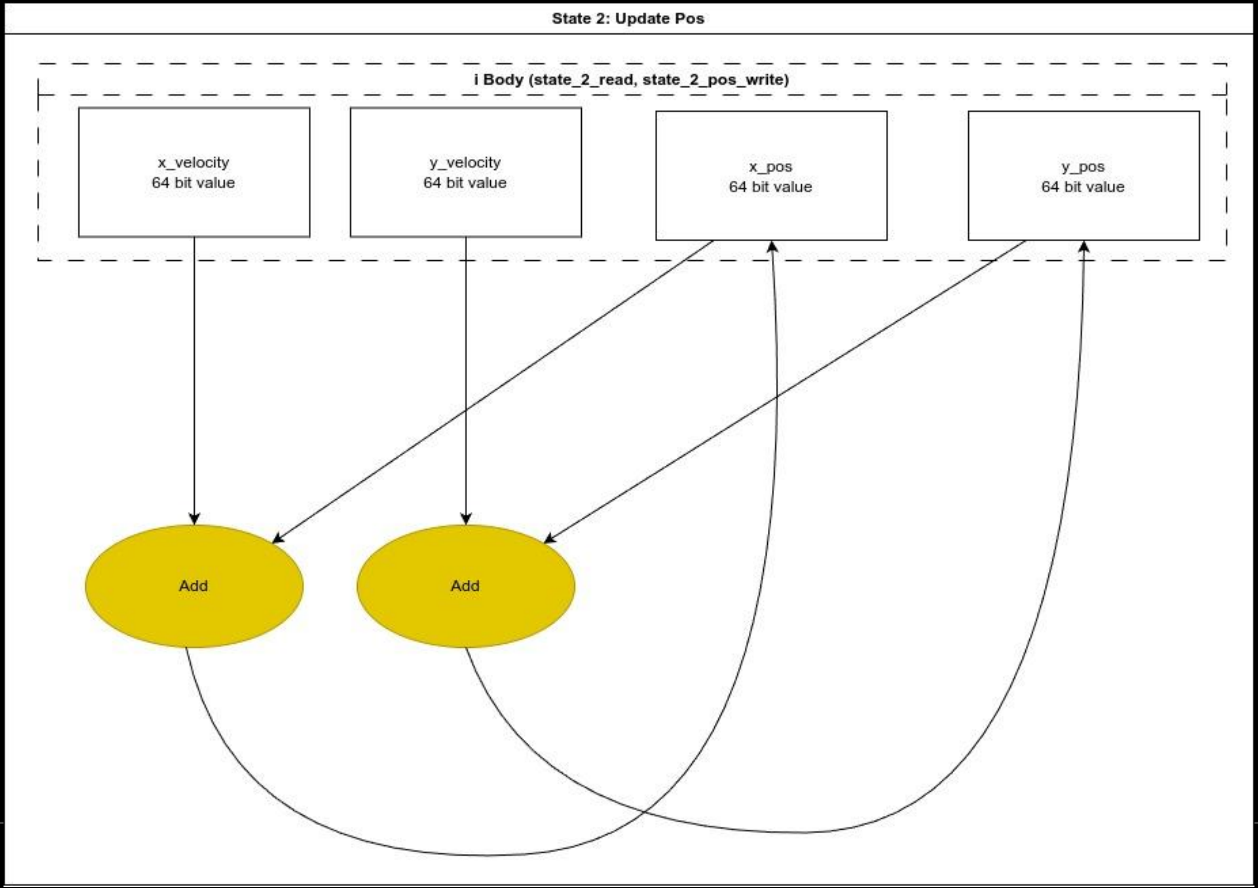

1
0
1
2
3
1
0


2
0
1
2
3
2
1
0


3
0
1
2
3
3
2
1
0

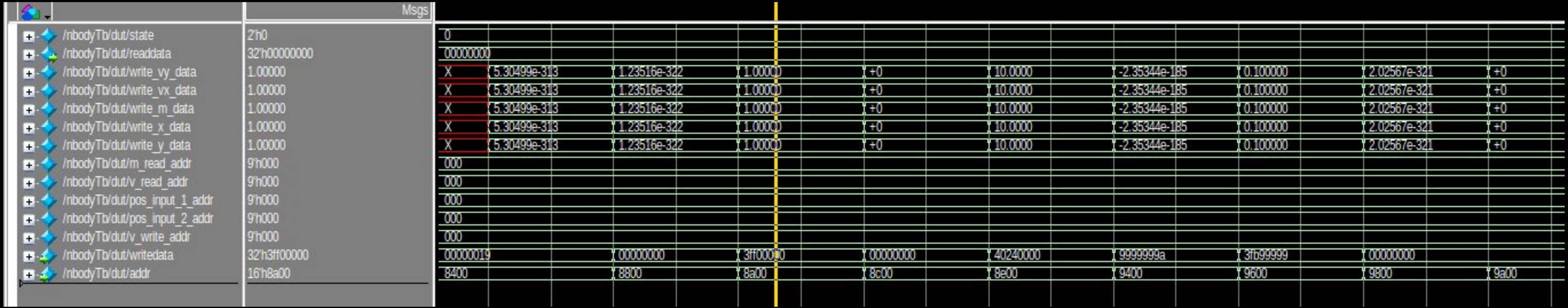

0
0
1
2
3
0_old
3
2
1
0

Update Position

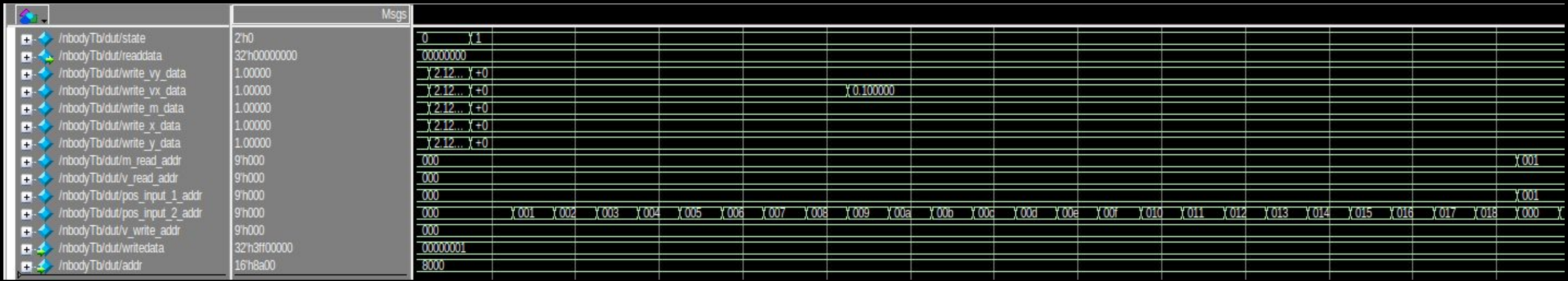


Hardware Validation

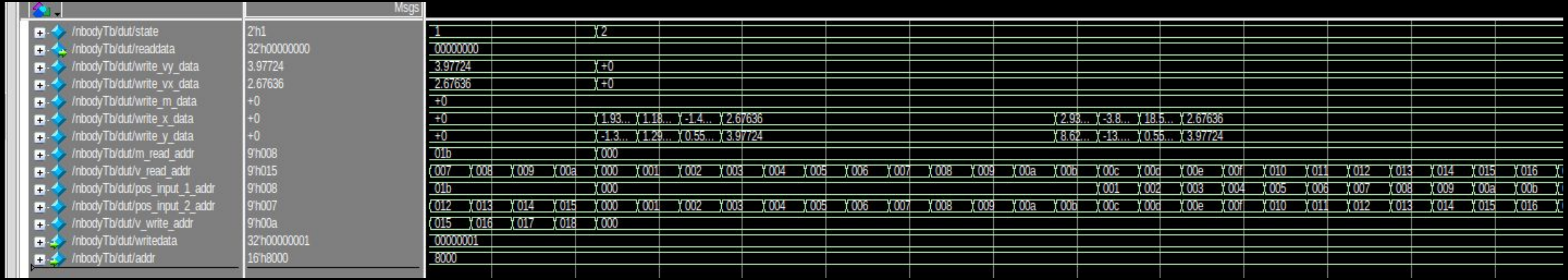
State 0: Read/Write (Input)



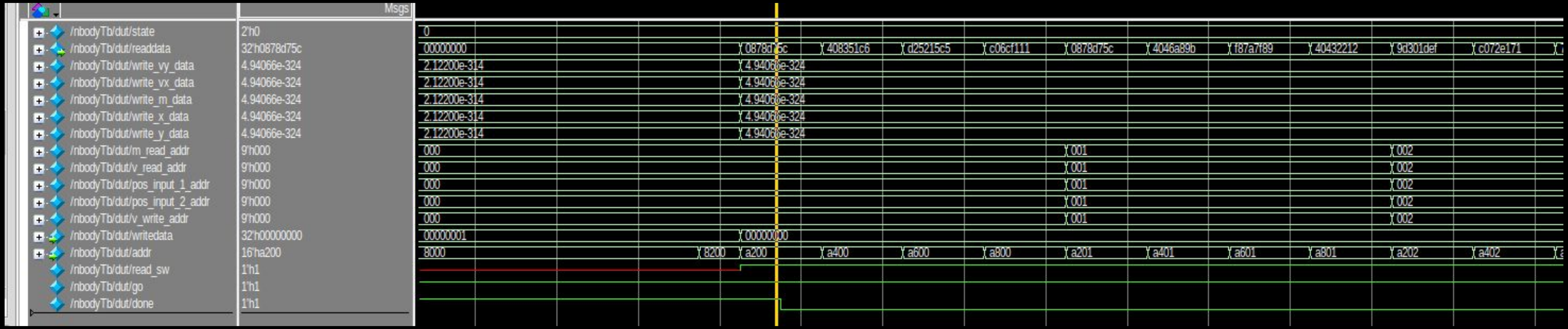
State 1: Begin acceleration computation



State 2: Begin position update



State 0: Reading output values



-
-

Testbench Final Results

```
# Time=                    62575: Te
# For body nbodyTb             0
# X = 618.221696
# Y = -231.533425
# For body nbodyTb             1
# X = 45.317231
# Y = 38.266204
# For body nbodyTb             2
# X = -302.090238
# Y = 108.575161
# Time=                   122835: Te
# For body nbodyTb             0
# X = 1529.887500
# Y = -583.881837
# For body nbodyTb             1
# X = 112.031482
# Y = 109.574936
# For body nbodyTb             2
# X = -775.501705
# Y = 268.047184
# ** Note: $finish    : nbodyTb
#    Time: 123095 ps  Iteration
```

Software

Accelerator Software Interface

32-bit Memory-Mapped Interface with **16 Bit addresses:**

| 0000000 | 000000000 | -> 9 Bits for Body Number 0-511

Parameter **Body Number**

Parameter Mappings:

0x44 - X_LOW
0x45 - X_UPPER

0x46 - Y_LOW
0x47 - Y_UPPER

0x4C - VY_LOW
0x4D - VY_UPPER

0x48 - M_LOW
0x49 - M_UPPER

0x4A - VX_LOW
0x4B - VX_UPPER

Read Addresses:

0x51 - Read_X_LOW
0x52 - Read_X_UPPER

0x53 - Read_Y_LOW
0x54 - Read_Y_UPPER

EX:
```
#define X_ADDR_LOW(base, body) (base) + ((body<<2) + (0x44 << 11))
```

Accelerator Driver

WRITE POSITIONS

```
typedef struct {
 double x, y, vx, vy,
m;
 int n;
} body_t;
```

WRITE PARAMETERS

```
typedef struct {
 int N;
 int gap;
} nbody_sim_config_t;
```

READ POSITIONS

```
typedef struct{
 double x, y;
 int n;
} body_pos_t;
```

NOTE - Positions and parameters are written and read as upper 32 and lower 32 bits separately:

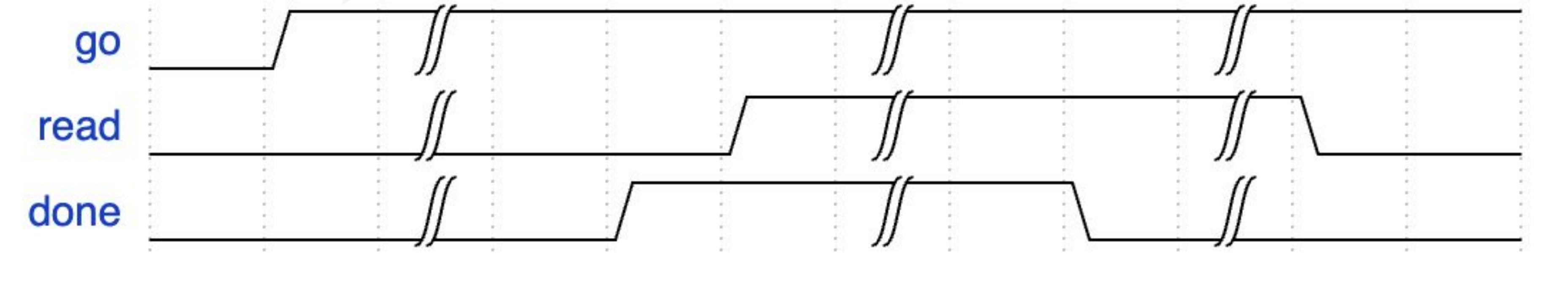
```
memcpy(&x_bits, &body_parameters->x, sizeof(uint64_t));
iowrite32(x_bits[0], X_ADDR_LOW(dev.virtbase, i));
iowrite32(x_bits[1], X_ADDR_HIGH(dev.virtbase, i));
```

Control Flow

SW Sets GO high to start sim

Upon receiving DONE,
Software Asserts Read

After DONE is lowered, software
reads the positions from memory

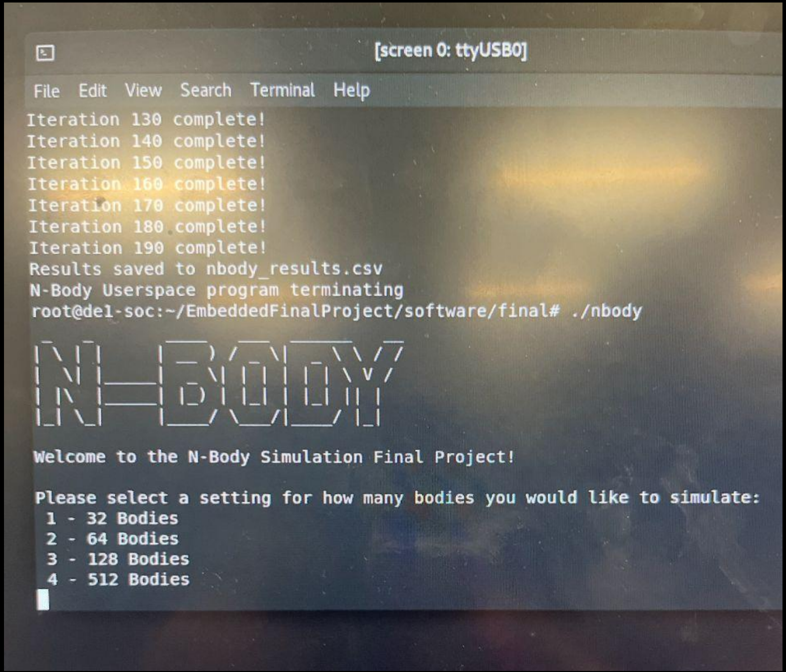


After performing *GAP*
iterations,
Hardware writes DONE signal

DONE goes
down after READ
is raised

User Logic

- Accelerator reads selected input CSV from userspace.
- Simulation parameters determined by user
  - Gap & Iterations
- After acceleration, positions for each iteration is written to output csv file



But wait! There's more

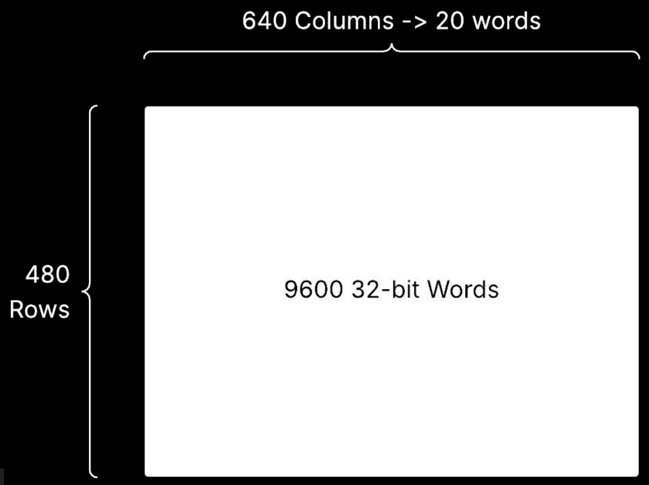
Display Hardware Interface

Memory mapped VGA Frame Buffer Module:

32 bit words accessed through 15 bit memory addresses, which maps to 32 pixels on the screen

- The read address passed on to memory is calculated as so below, and uses both vcount and hcount with an appropriate offset, to ensure the address is passed to memory in time

```
assign vcount_32 = {22'b0, vcount};
assign vcount_x_512 = vcount_32 << 8;
assign vcount_x_128 = vcount_32 << 10;
assign hcount_32 = (hcount > 11'd1300) ? 32'd1300 : {21'b0, hcount};
assign vcountx20 = vcount_x_128 + vcount_x_512;
assign placecounter = vcountx20 + hcount_32 + 32'd2;
assign rdaddress = placecounter[20:6];
```



Display Software Interface

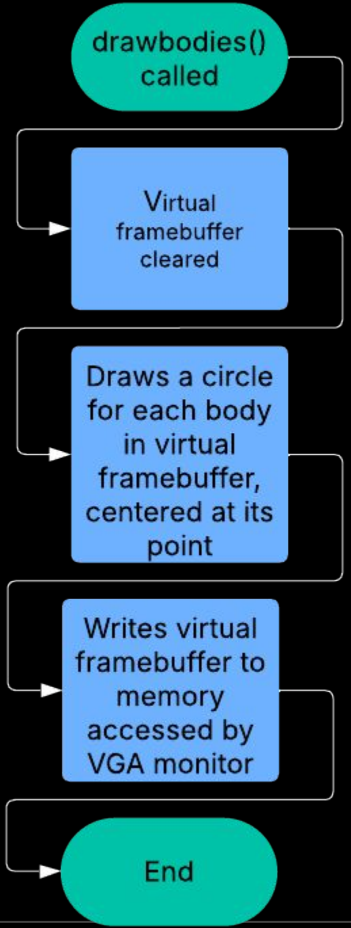
Kernel Module with specific functions to facilitate N-Body animations

IOCTLS

`WRITE_PROPERTIES - Copies the current state for all bodies in the timestep from user to kernel space. Writes each body as a circle into the framebuffer, which is written to memory at the end.`

`CLEAR_SCREEN - Clears both the virtual framebuffer and the corresponding area in memory`

`FILL_SCREEN - Writes a pattern directly to framebuffer memory. Used in debugging to display a filled screen, checker pattern.`



Display Software Interface

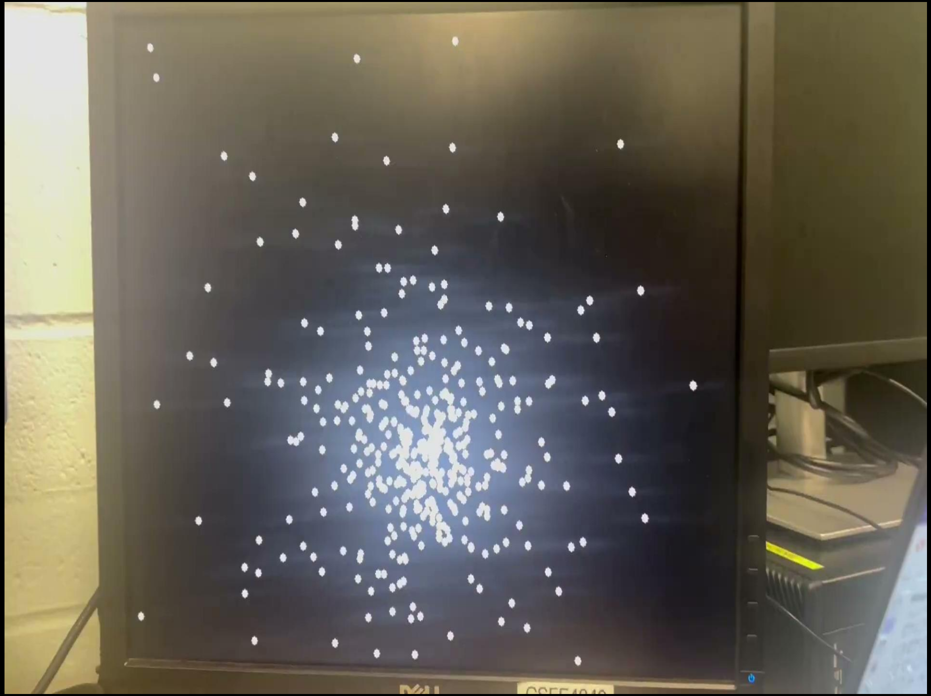
Userspace program that allows playback from CSV with adjustable play speed

```
static void convert_coordinates(float
nbody_x, float nbody_y,short
*display_x, short *display_y) {

    *display_x = (short)((nbody_x +
500.0) / 1000.0 * (DISPLAY_WIDTH));
    *display_y = (short)((nbody_y +
500.0) / 1000.0 * (DISPLAY_HEIGHT));

}
```

`After this coordinate conversion, values are sent to the driver through the ioctl call!`



DEMO