Bridge of Life Education

AAHLS Special Project Lab #B Presentation - Systolic Array

61275047H 黃聖崴

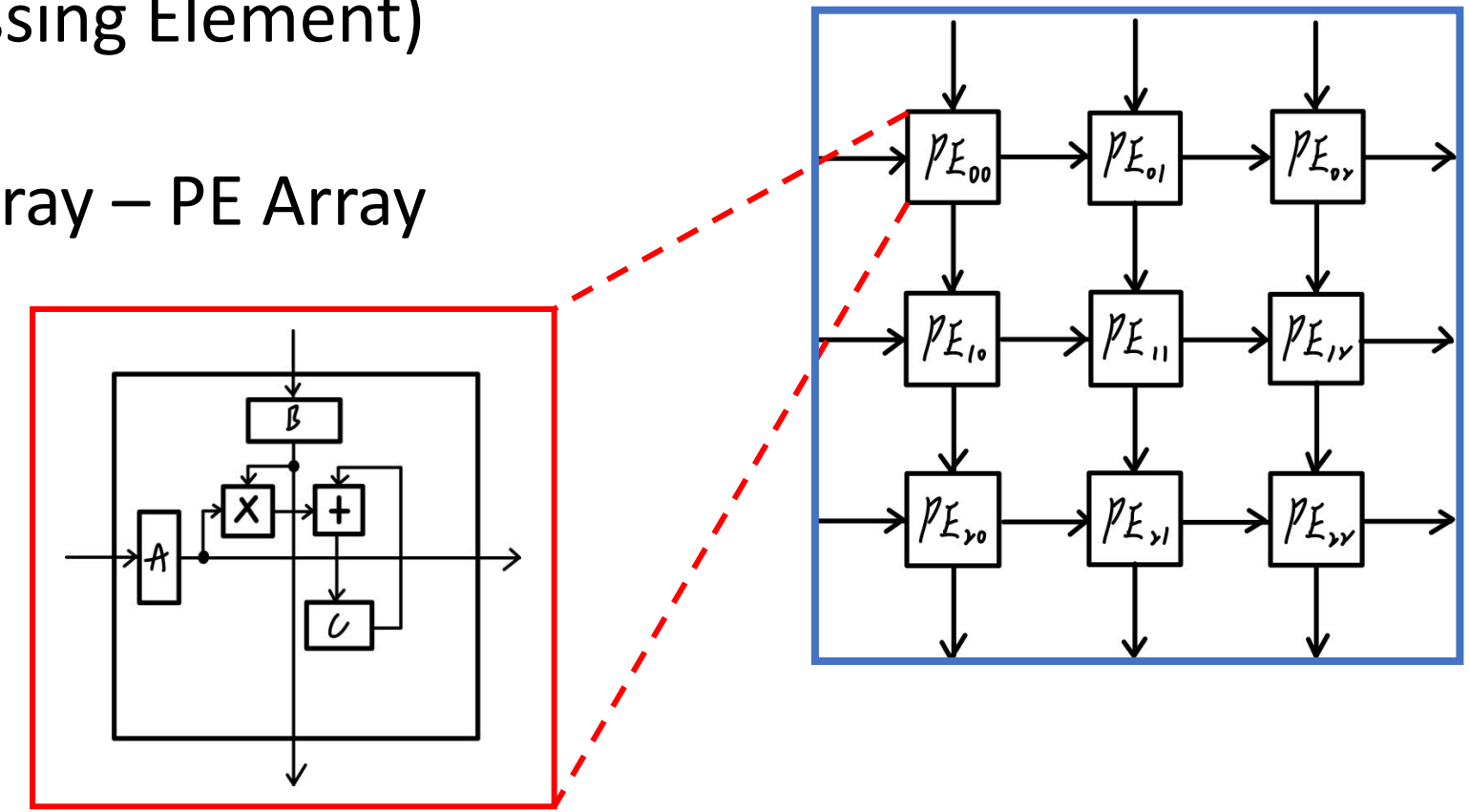
https://github.com/kevin33713371/2024_Fall_NTU_AAHLS_SP

Outline

- Introduction - Systolic Array
- Parameter Setting & Pragma Illustration
  - Implicit Type(Broadcasting)
  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)
- Broadcasting & Propagating Mechanism
  - Implicit Type(Broadcasting)
  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)
- Code Explanation
  - Implicit Type(Broadcasting)
  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)
- Timing Trace – Schedule Viewer
  - Implicit Type(Broadcasting)
  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)
- Time & Resource Analysis
  - Implicit Type(Broadcasting)
  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)
- Co-Simulation Timeline Trace
  - Implicit Type(Broadcasting)
  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)

Introduction – Systolic Array

- PE (Processing Element)
- Systolic Array – PE Array



Parameter Setting & Pragma Illustration

- Implicit Type(Broadcasting) – Parameter Setting
  - MAX_SIZE = 16 (Depends on the available DSP resources in the FPGA)
  - Local Buffer for AXI4 Master R/W
    - localA: MAX_SIZE * MAX_SIZE
    - localB: MAX_SIZE * MAX_SIZE
    - localC: MAX_SIZE * MAX_SIZE
    - localC = localA * localB
  - Systolic Array in this lab
    - PE Array with size of MAX_SIZE * MAX_SIZE

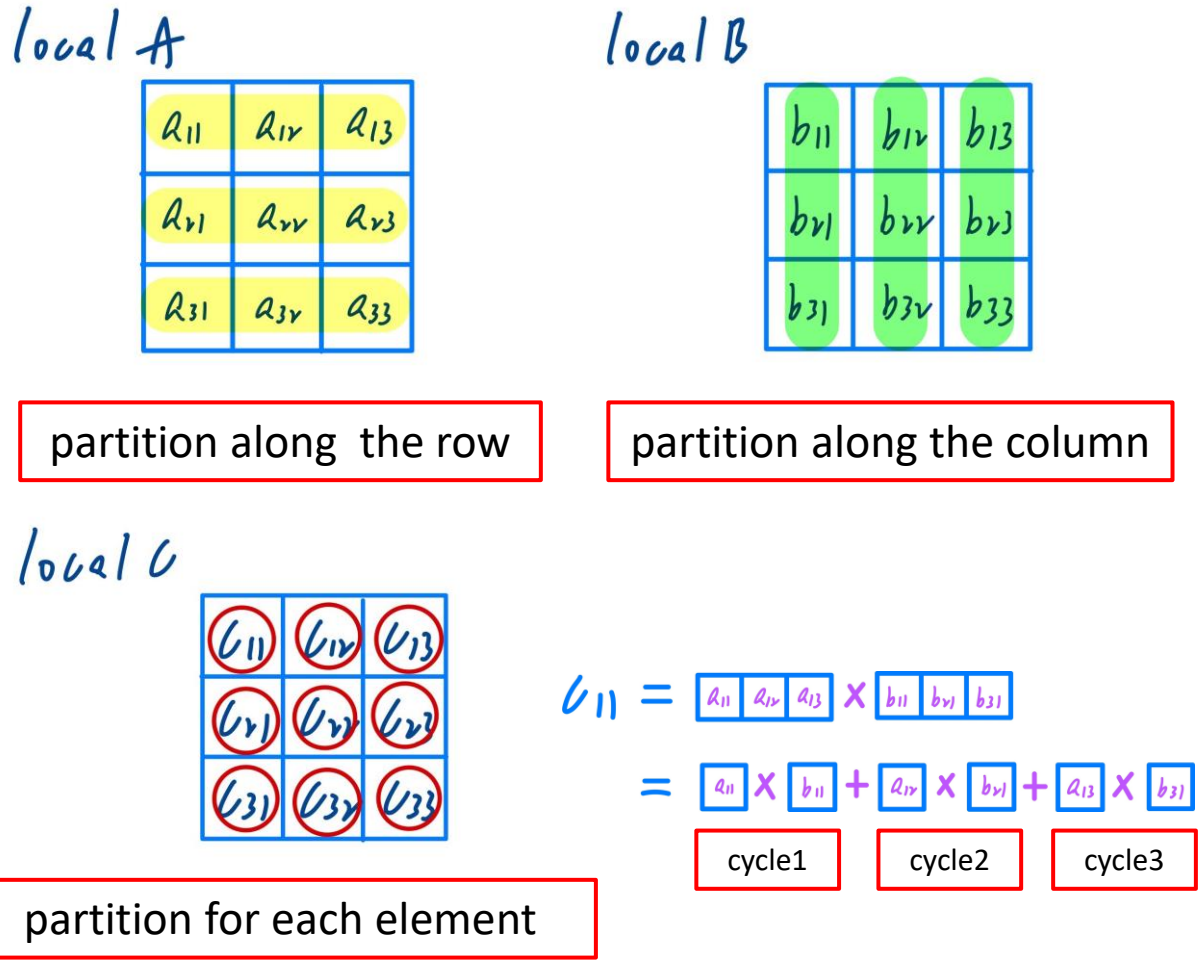
```
    // Local memory to store input and output matrices
    int localA[MAX_SIZE][MAX_SIZE];
#pragma HLS ARRAY_PARTITION variable = localA dim = 1 complete

    int localB[MAX_SIZE][MAX_SIZE];
#pragma HLS ARRAY_PARTITION variable = localB dim = 2 complete

    int localC[MAX_SIZE][MAX_SIZE];
#pragma HLS ARRAY_PARTITION variable = localC dim = 0 complete
```

Parameter Setting & Pragma Illustration

- Pragma
  - localA, localB, localC
    -> Array partition
  - Systolic1
    -> Unroll (create MAX_SIZE * MAX_SIZE to work in parallel)

- Every item in localC uses MAX_SIZE cycles to accumulate result

- Every element in localA and LocalB are broadcasted



Parameter Setting & Pragma Illustration

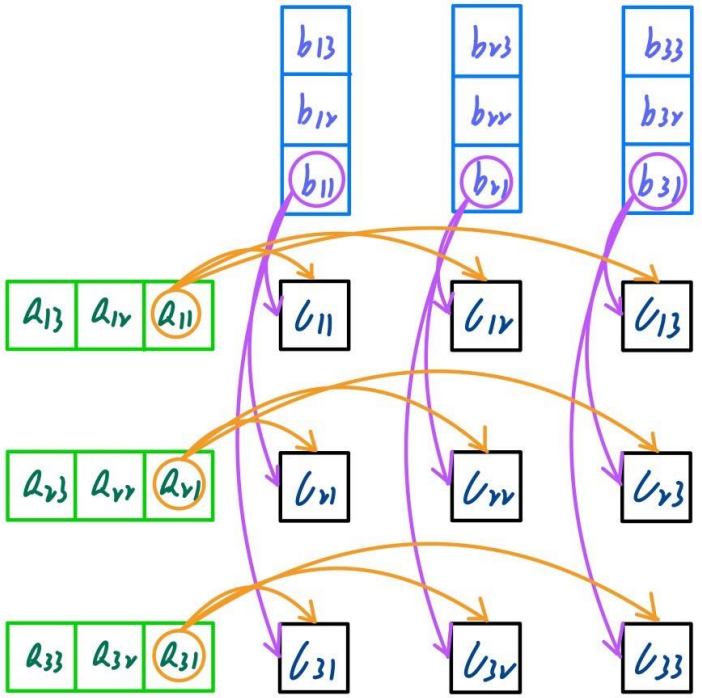
- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)
  - N = 16, M = 16 (has the same size as implicit type)
  - Local Buffer for AXI4 Master R/W
    - localA: N * N
    - localB: N * N
    - localC: N * N
  - Register in each Processing Element
    - inA: M * M
    - inB: M * M
  - Systolic Array in this lab
    - PE Array with size of M * M

```
// Local memory to store input and output matrices
int localA[N][N];
#pragma HLS ARRAY_PARTITION variable = localA dim = 2 complete

int localB[N][N];
#pragma HLS ARRAY_PARTITION variable = localB dim = 1 complete

int localC[N][N];
#pragma HLS ARRAY_PARTITION variable = localC dim = 0 complete

// Register in Systolic Array
int inA[M][M];
#pragma HLS ARRAY_PARTITION variable = inA dim=0 complete

int inB[M][M];
#pragma HLS ARRAY_PARTITION variable = inB dim=0 complete
```

Broadcasting Mechanism

- Every PE accumulates the partial sum from the inner product of two vectors.
- Array partitioned can be explained as turn an array into vectors

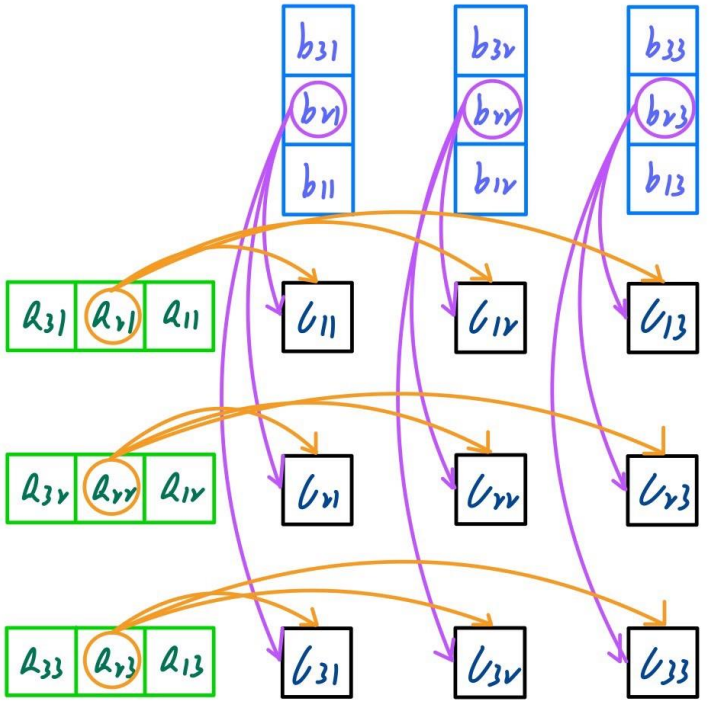
cycle1



Broadcasting Mechanism

- Every PE accumulates the partial sum from the inner product of two vectors.
- Array partitioned can be explained as turn an array into vectors

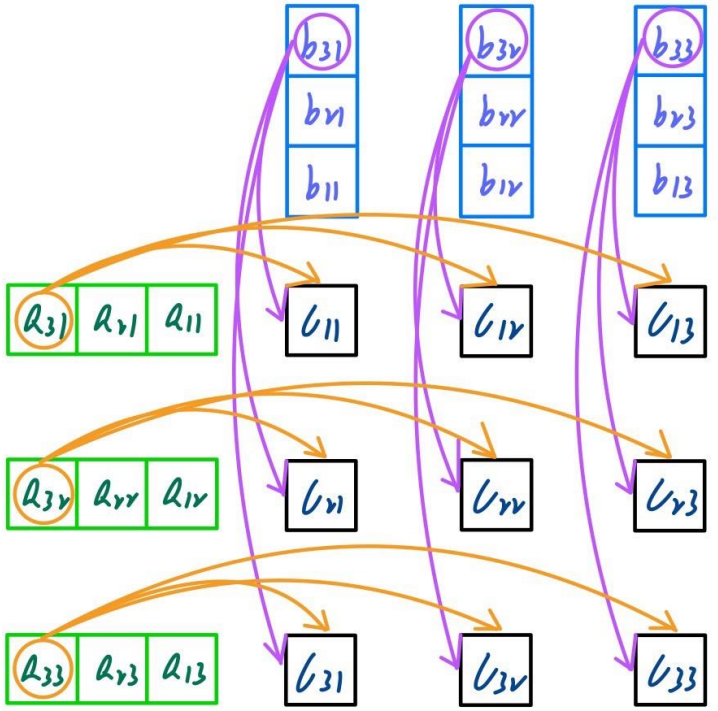
cycle2



Broadcasting Mechanism

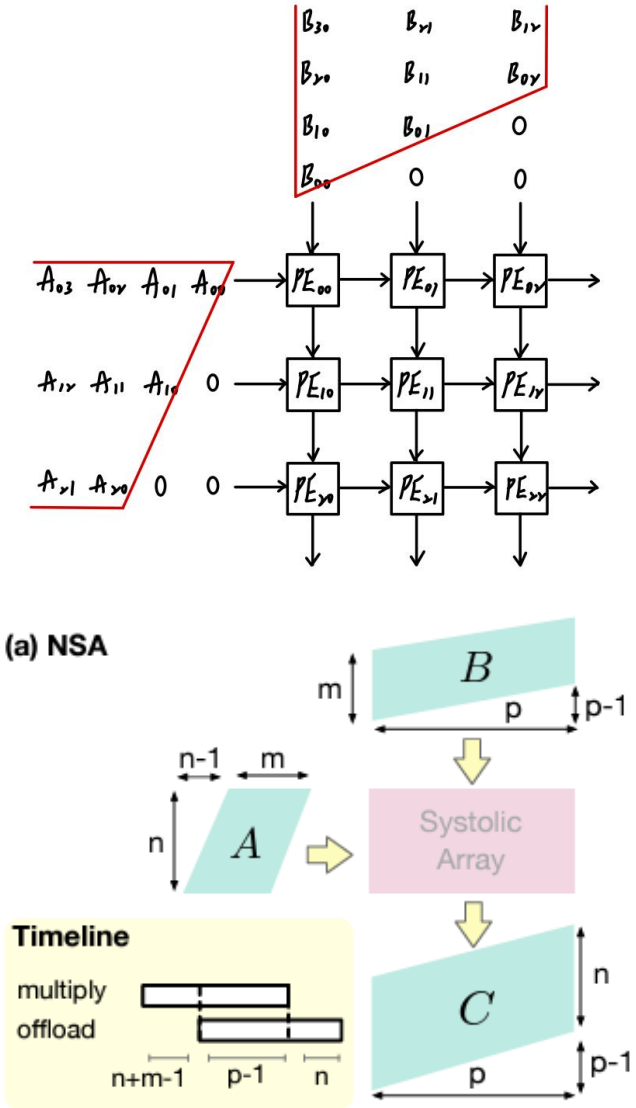
- Every PE accumulates the partial sum from the inner product of two vectors.
- Array partitioned can be explained as turn an array into vectors

cycle3



Propagating Mechanism

- The previously mentioned Systolic Array is not a true Systolic Array.
- A true Systolic Array used for matrix multiplication is shown in the image on the right.
- Non-Stationary Systolic Array (NSSA)



Parameter Setting & Pragma Illustration

- Protocol Definition for each I/O port
  - a - AXI4 Master
  - b - AXI4 Master
  - c - AXI4 Master
  - a_row - AXI4 Lite
  - a_col - AXI4 Lite
  - b_row - AXI4 Lite

Notice : port for AXI4 Master's transfer must define as volatile!

```
extern "C"{
void mmult(volatile int* a, // Read-Only Matrix A
           volatile int* b, // Read-Only Matrix B
           volatile int* c, // Output Result
           int a_row,     // Matrix A Row Size
           int a_col,     // Matrix A Col Size
           int b_col      // Matrix B Col Size
           ) {
    // AXI4 Master's depth must be a constant and should be adjusted according to MAX_SIZE * MAX_SIZE
    #pragma HLS INTERFACE m_axi port=a offset=slave bundle=gmem0 depth=256
    #pragma HLS INTERFACE m_axi port=b offset=slave bundle=gmem1 depth=256
    #pragma HLS INTERFACE m_axi port=c offset=slave bundle=gmem2 depth=256

    //AXI4 Lite's port must use the same bundle
    #pragma HLS INTERFACE s_axilite port=a bundle=control
    #pragma HLS INTERFACE s_axilite port=b bundle=control
    #pragma HLS INTERFACE s_axilite port=c bundle=control
    #pragma HLS INTERFACE s_axilite port=a_row bundle=control
    #pragma HLS INTERFACE s_axilite port=a_col bundle=control
    #pragma HLS INTERFACE s_axilite port=b_col bundle=control
    #pragma HLS INTERFACE s_axilite port=return bundle=control
```

Code Explanation – Implicit Type

```
// Initialization
init:
    for (int i = 0; i < MAX_SIZE; i++){
        #pragma HLS PIPELINE
        for (int j = 0; j < MAX_SIZE; j++){
            localC[i][j] = 0;
        }
    }
```

```
// Burst reads on input matrices from global memory
// Read Input A
readA:
    for (int loc = 0, i = 0, j = 0; loc < a_row * a_col; loc++, j++) {
        #pragma HLS LOOP_TRIPCOUNT min = c_size* c_size max = c_size * c_size
        #pragma HLS PIPELINE
        if (j == a_col) {
            i++;
            j = 0;
        }
        localA[i][j] = a[loc];
    }
```

```
// Read Input B
readB:
    for (int loc = 0, i = 0, j = 0; loc < b_row * b_col; loc++, j++) {
        #pragma HLS LOOP_TRIPCOUNT min = c_size * c_size max = c_size * c_size
        #pragma HLS PIPELINE
        if (j == b_col) {
            i++;
            j = 0;
        }
        localB[i][j] = b[loc];
    }
```

```
// Perform systolic matrix multiply
systolic1:
    for (int k = 0; k < a_col; k++) {
        #pragma HLS LOOP_TRIPCOUNT min = c_size max = c_size
    systolic2:
        for (int i = 0; i < MAX_SIZE; i++) {
            #pragma HLS UNROLL
        systolic3:
            for (int j = 0; j < MAX_SIZE; j++) {
                #pragma HLS UNROLL
                // Get previous sum
                int last = (k == 0) ? 0 : localC[i][j];

                // Update current sum
                // Handle boundary conditions
                int a_val = (i < a_row && k < a_col) ? localA[i][k] : 0;
                int b_val = (k < b_row && j < b_col) ? localB[k][j] : 0;
                int result = last + a_val * b_val;

                // Write back results
                localC[i][j] = result;
            }
        }
    }
```

```
// Burst write from output matrices to global memory
// Burst write from matrix C
writeC:
    for (int loc = 0, i = 0, j = 0; loc < c_row * c_col; loc++, j++) {
        #pragma HLS LOOP_TRIPCOUNT min = c_size* c_size max = c_size * c_size
        #pragma HLS PIPELINE
        if (j == c_col) {
            i++;
            j = 0;
        }
        c[loc] = localC[i][j];
        //printf("Write c[%d] = %d\n", loc, localC[i][j]); // Debug output
    }
```

Code Explanation – Explicit Type

```
// Initialization
init:
    for (int i = 0; i < M; i++){
        #pragma HLS PIPELINE
        for (int j = 0; j < M; j++){
            inA[i][j] = 0;
            inB[i][j] = 0;
            localC[i][j] = 0;
        }
    }
```

```
// Burst reads on input matrices from global memory
// Read Input A
readA:
    for (int loc = 0, i = 0, j = 0; loc < a_row * a_col; loc++, j++) {
        #pragma HLS LOOP_TRIPCOUNT min = c_size* c_size max = c_size * c_size
        #pragma HLS PIPELINE
        if (j == a_col) {
            i++;
            j = 0;
        }
        localA[i][j] = a[loc];
    }
```

```
// Read Input B
readB:
    for (int loc = 0, i = 0, j = 0; loc < b_row * b_col; loc++, j++) {
        #pragma HLS LOOP_TRIPCOUNT min = c_size * c_size max = c_size * c_size
        #pragma HLS PIPELINE
        if (j == b_col) {
            i++;
            j = 0;
        }
        localB[i][j] = b[loc];
    }
```

```
// Perform systolic matrix multiply
systolic1:
    for(int r = 0; r < N + 2 * M - 2; r++){
        #pragma HLS pipeline

        for (int i = 0; i < M; i++){
            for(int j = M - 1; j >= 1; j--){
                inA[i][j] = inA[i][j - 1];
            }
        }

        for (int i = M - 1; i >= 1; i--){
            for(int j = 0; j < M; j++){
                inB[i][j] = inB[i - 1][j];
            }
        }

        for (int i = 0; i < M; i++) {
            if(r >= i && r < i + N)
                inA[i][0] = localA[i][r - i];
            else
                inA[i][0] = 0;
        }

        for (int j = 0; j < M; j++) {
            if(r >= j && r < j + N)
                inB[0][j] = localB[r - j][j];
            else
                inB[0][j] = 0;
        }

        for(int i = 0; i < M; i++) {
            for(int j = 0; j < M; j++) {
                localC[i][j] += inA[i][j] * inB[i][j];
            }
        }
    }
```

Propagating

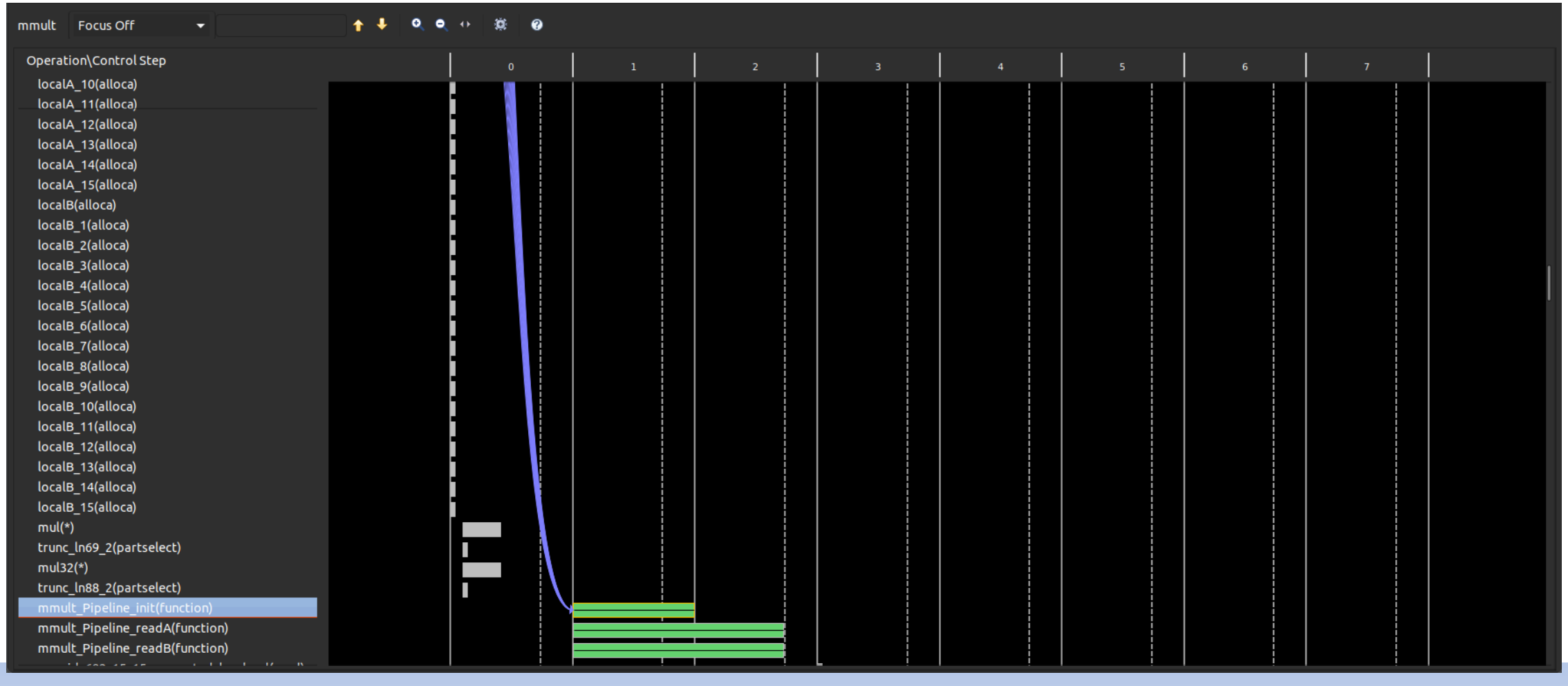
Load

Compute

```
// Burst write from output matrices to global memory
// Burst write from matrix C
writeC:
    for (int loc = 0, i = 0, j = 0; loc < c_row * c_col; loc++, j++) {
        #pragma HLS LOOP_TRIPCOUNT min = c_size* c_size max = c_size * c_size
        #pragma HLS PIPELINE
        if (j == c_col) {
            i++;
            j = 0;
        }
        c[loc] = localC[i][j];
        //printf("Write c[%d] = %d\n", loc, localC[i][j]); // Debug output
    }
```

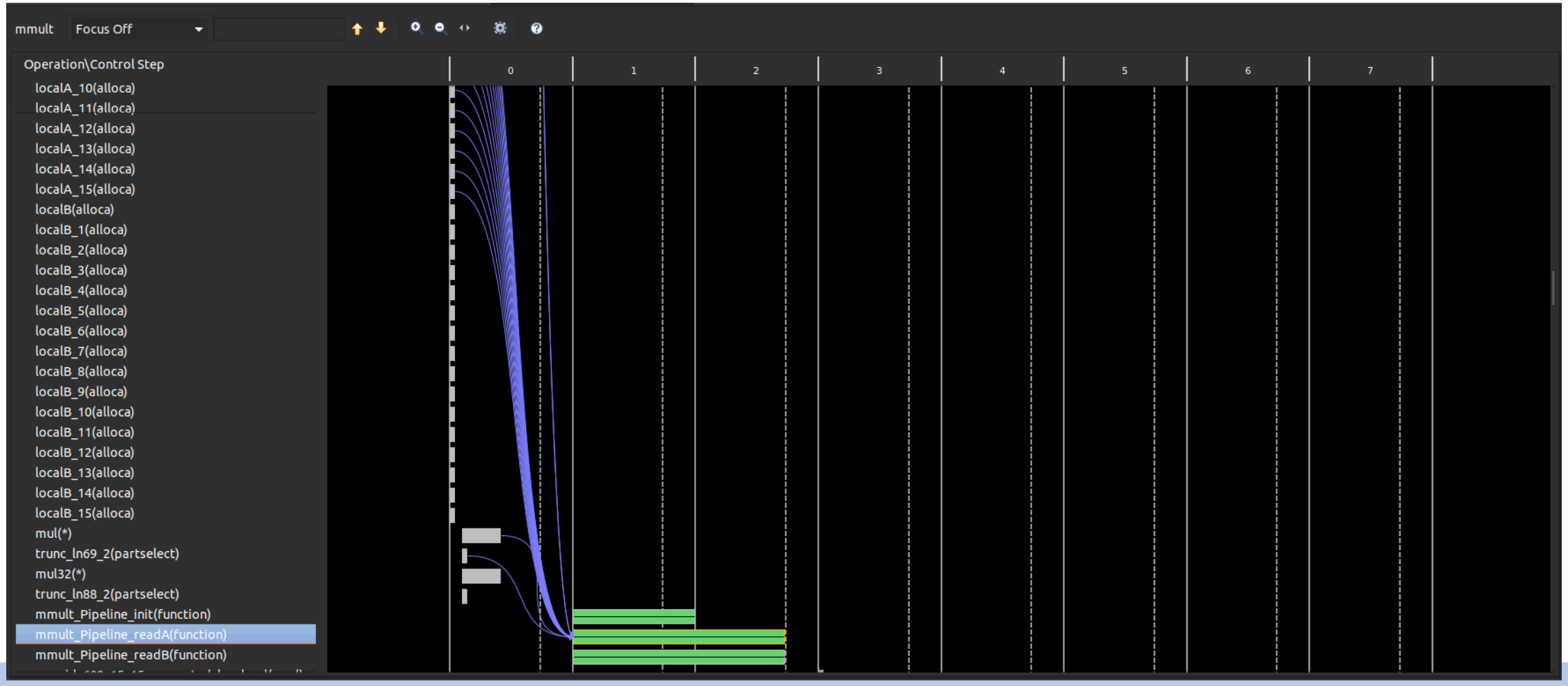
Timing Trace – Schedule Viewer

- Implicit Type(Broadcasting)



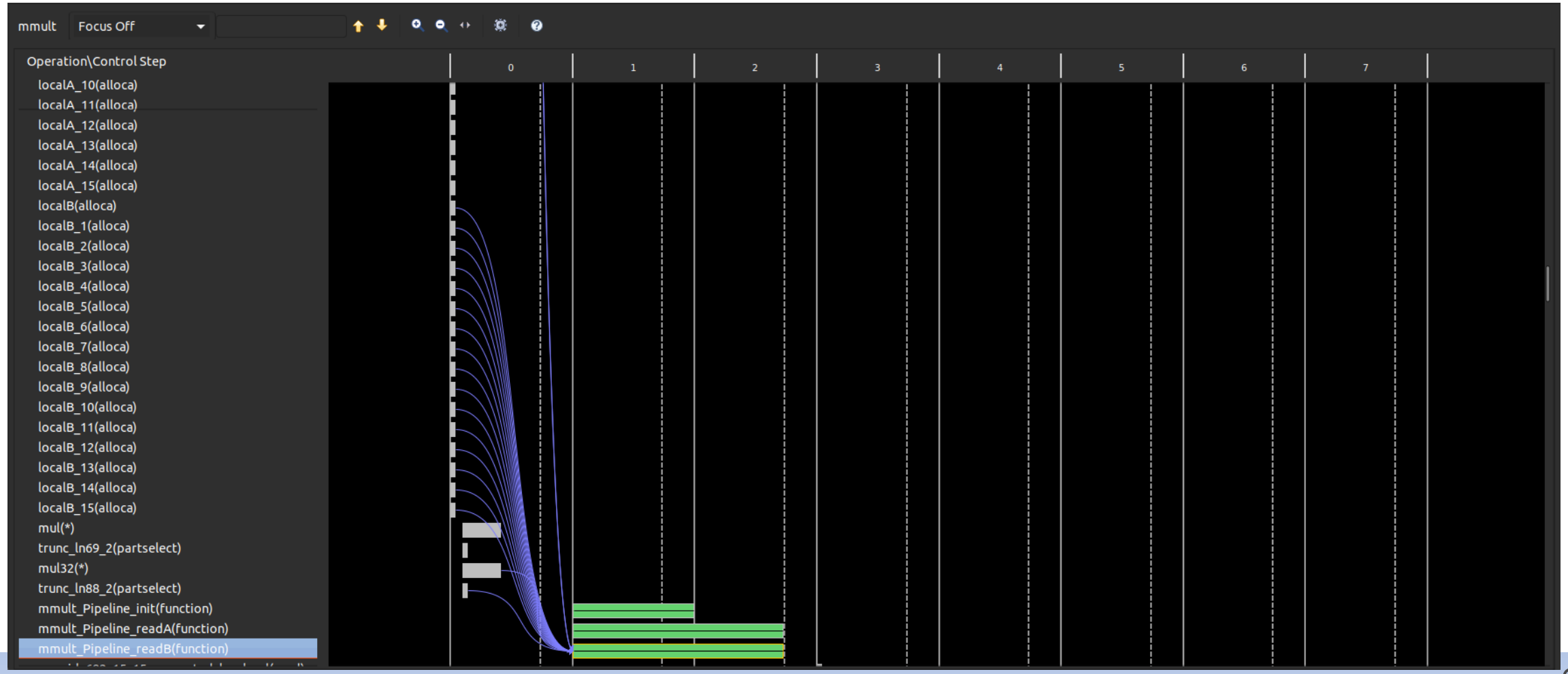
Timing Trace – Schedule Viewer

- Implicit Type(Broadcasting)



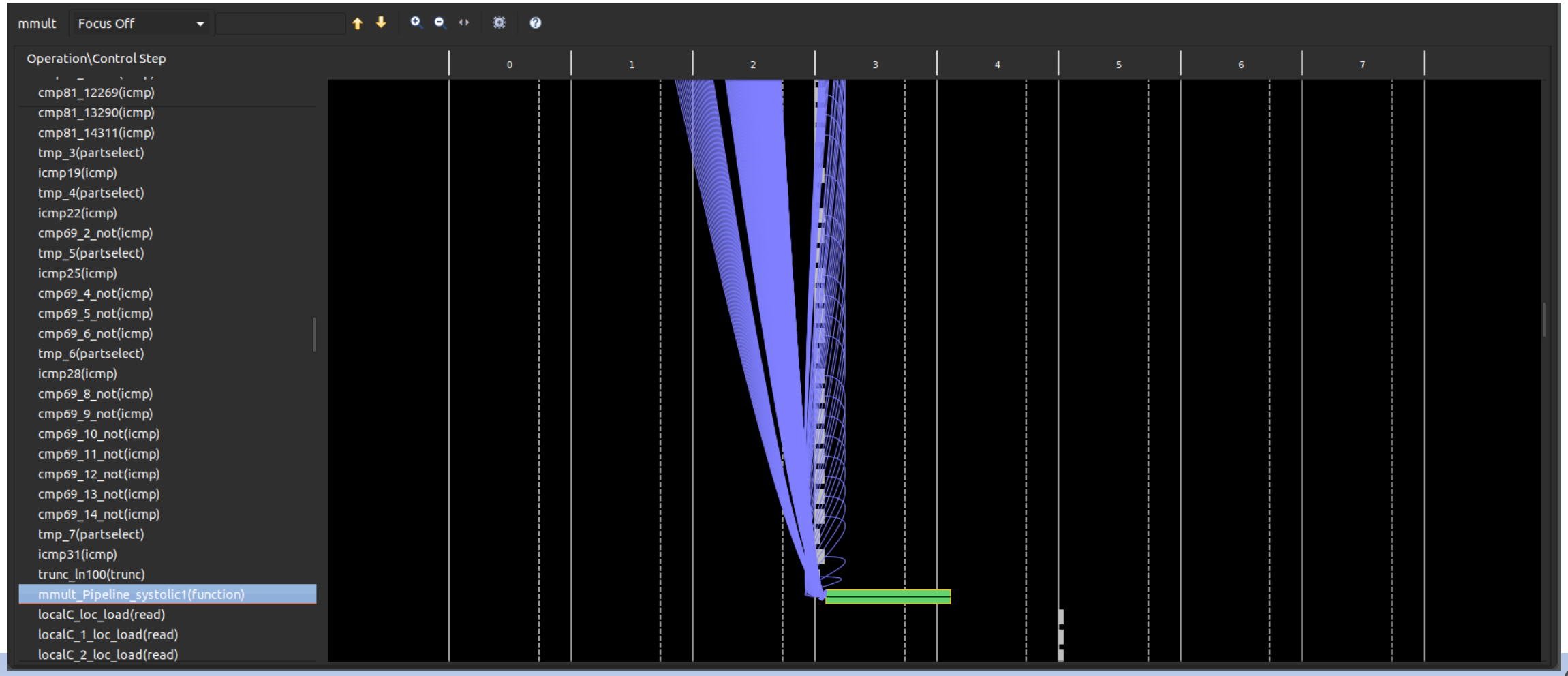
Timing Trace – Schedule Viewer

- Implicit Type(Broadcasting)



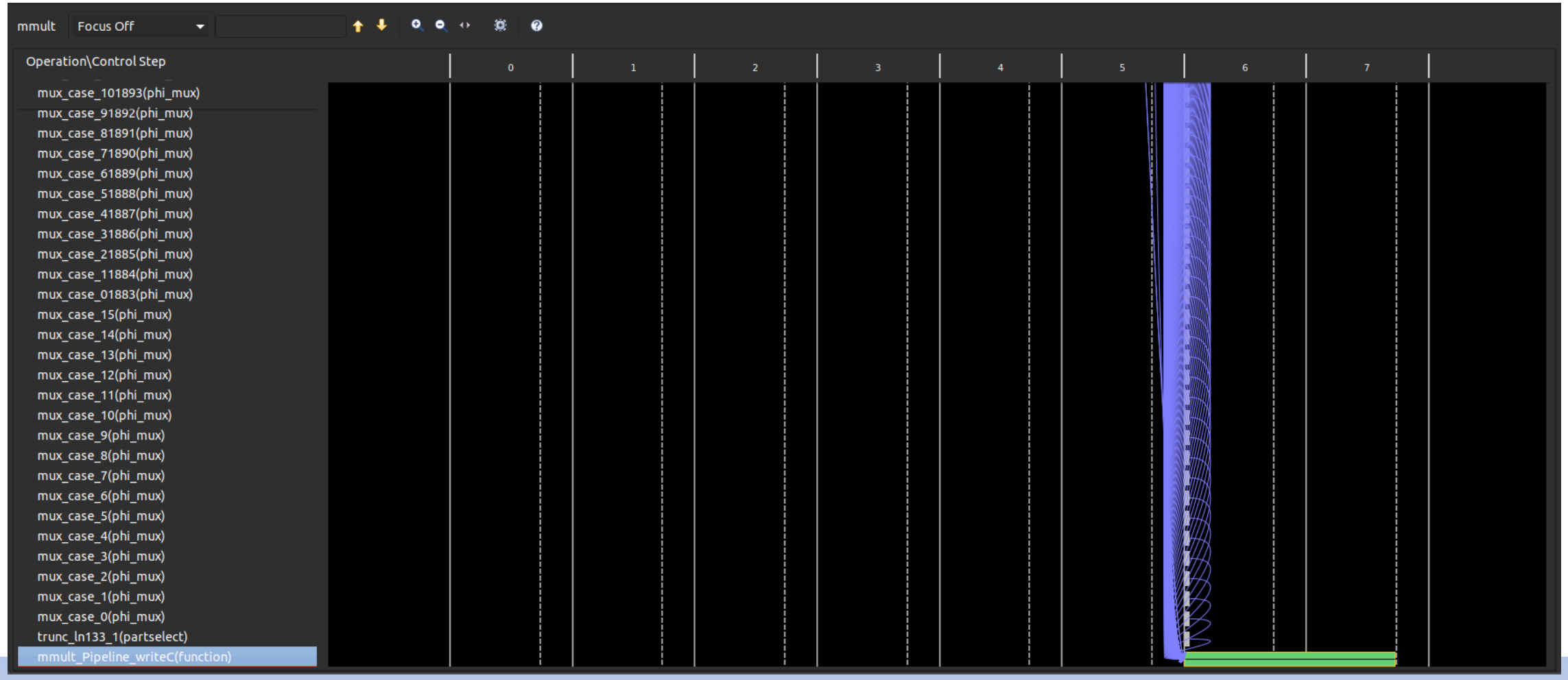
Timing Trace – Schedule Viewer

- Implicit Type(Broadcasting)



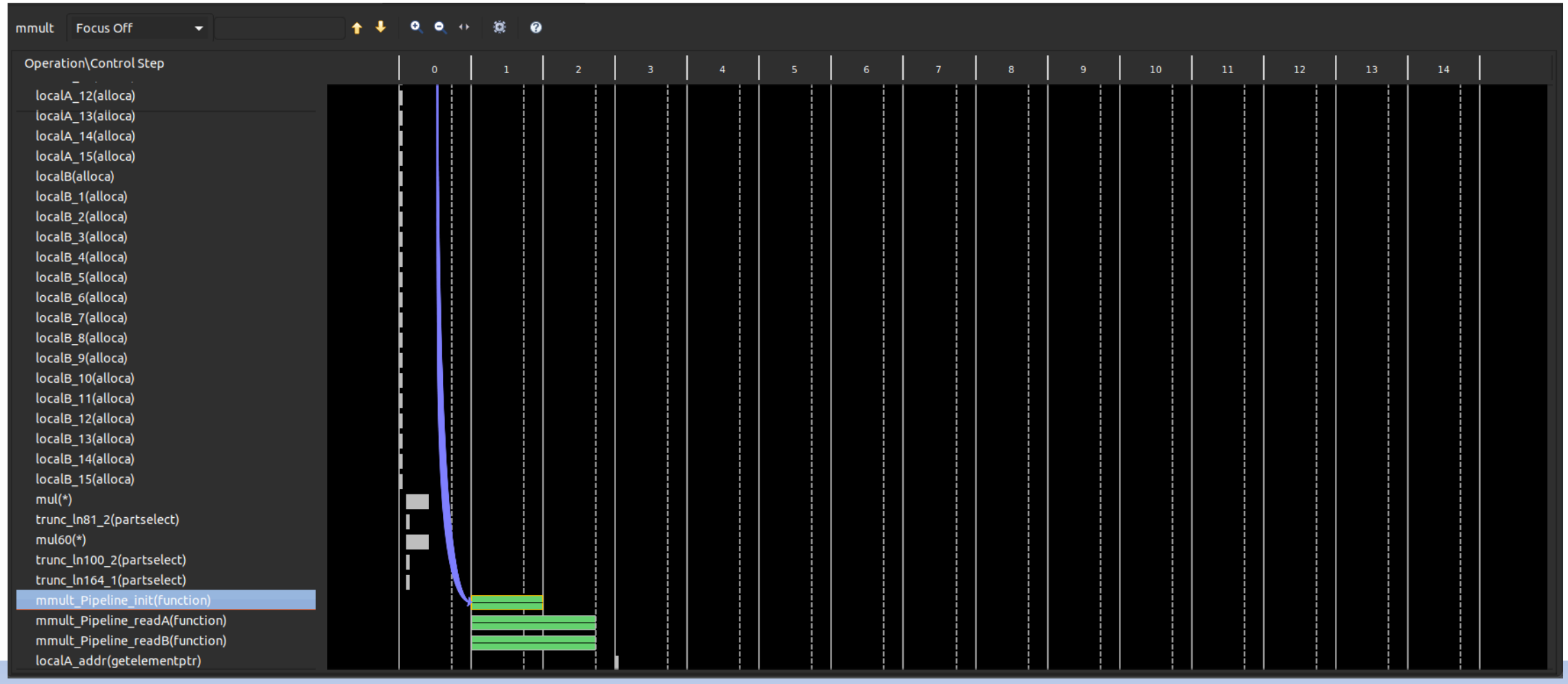
Timing Trace – Schedule Viewer

- Implicit Type(Broadcasting)



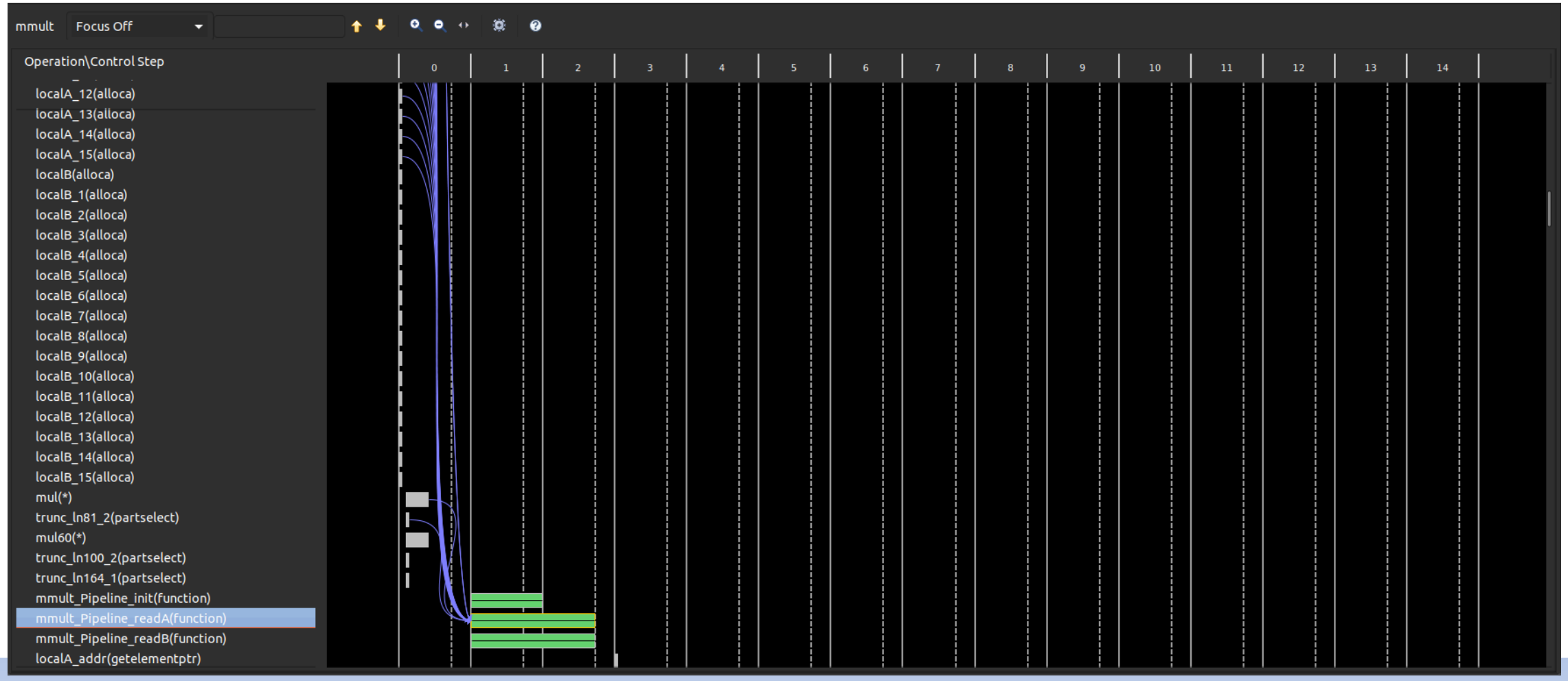
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



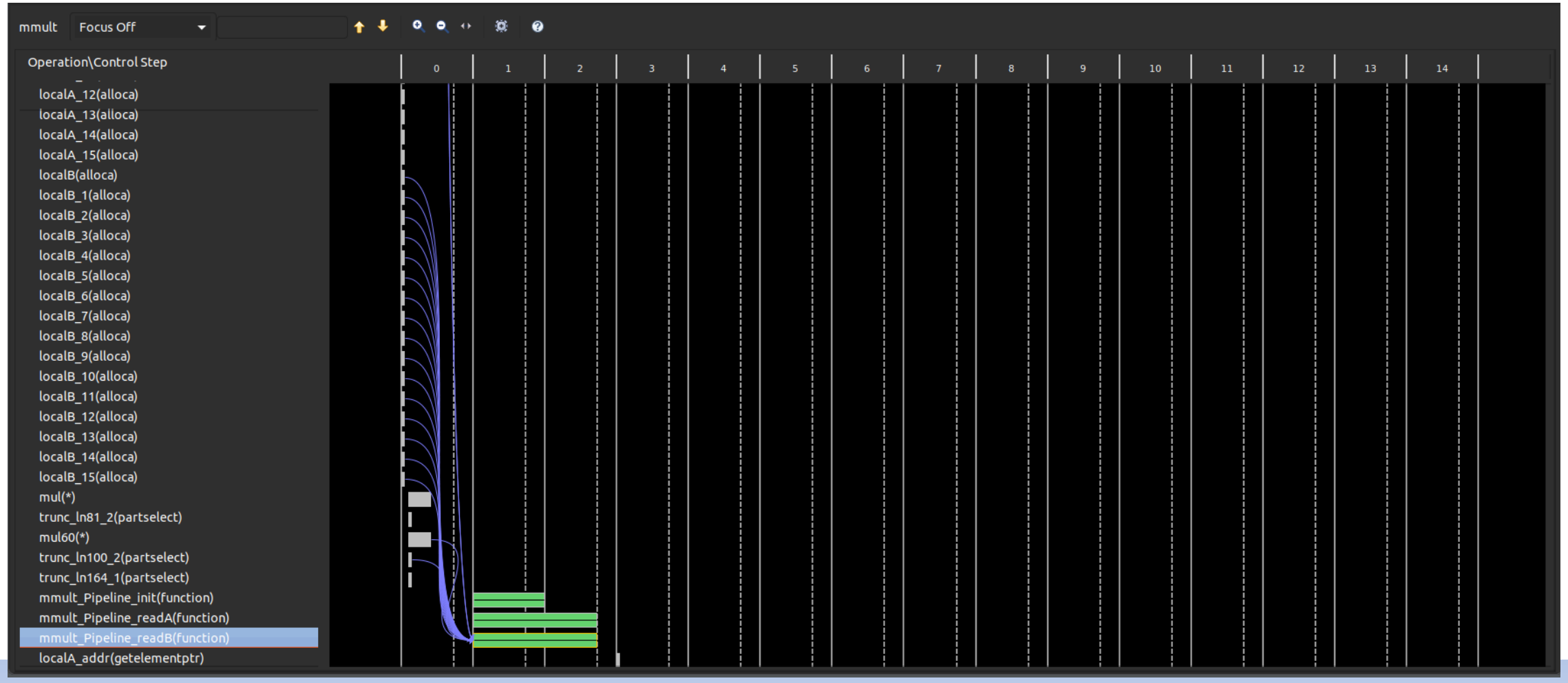
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



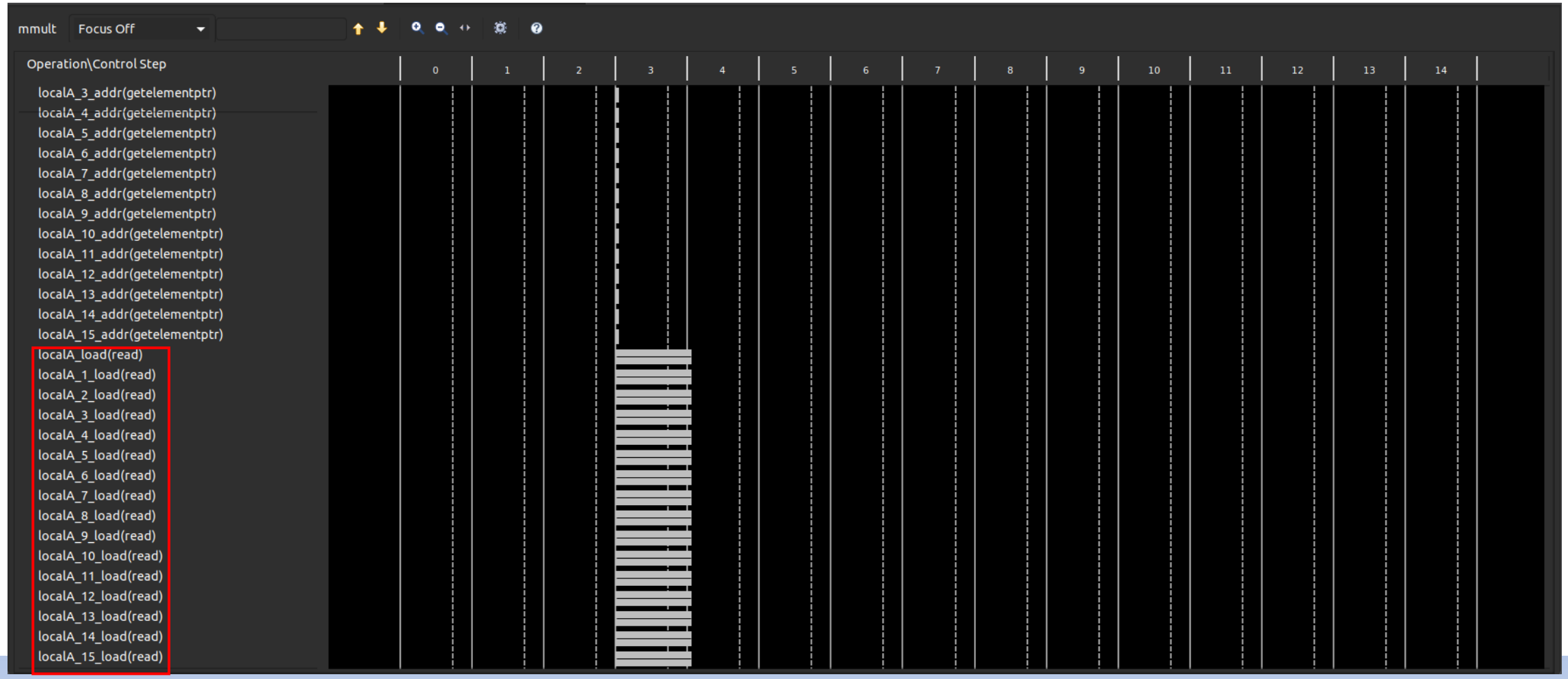
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



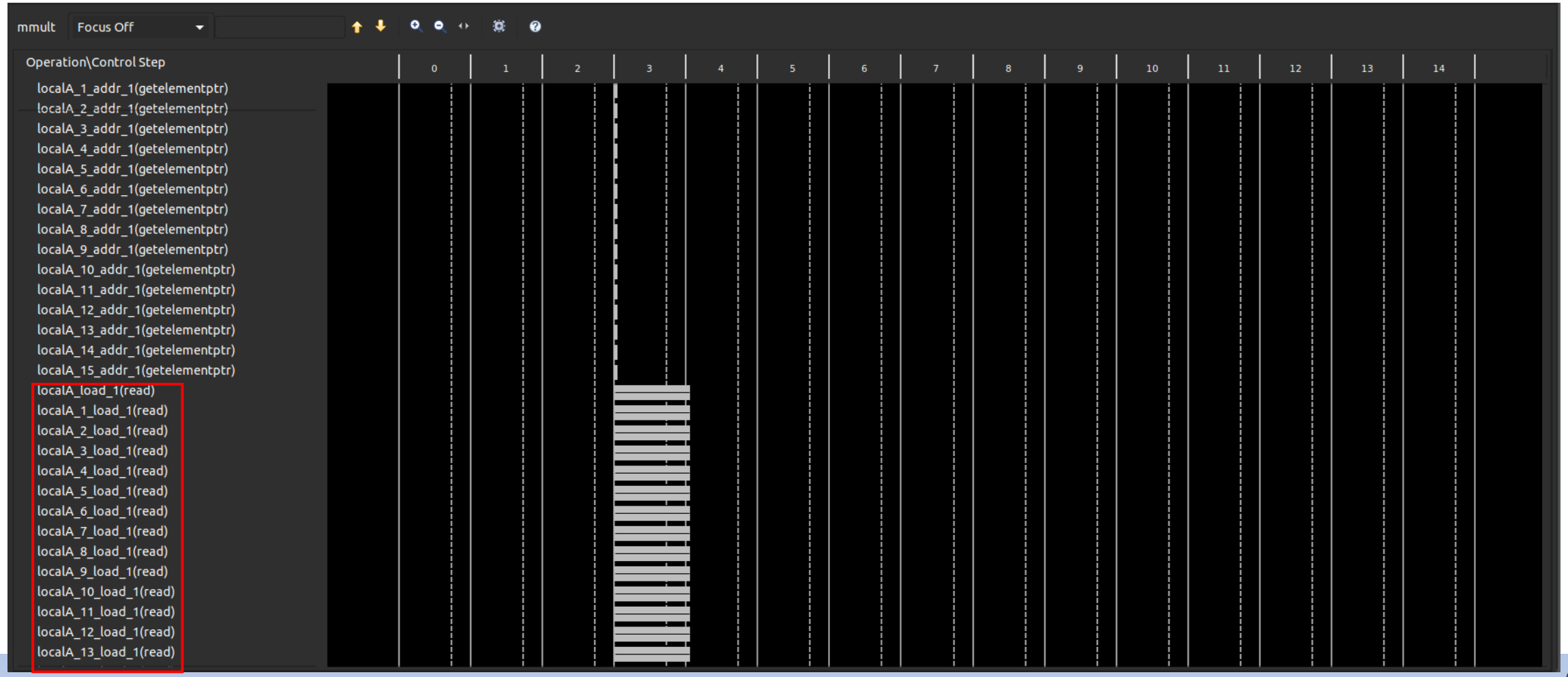
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



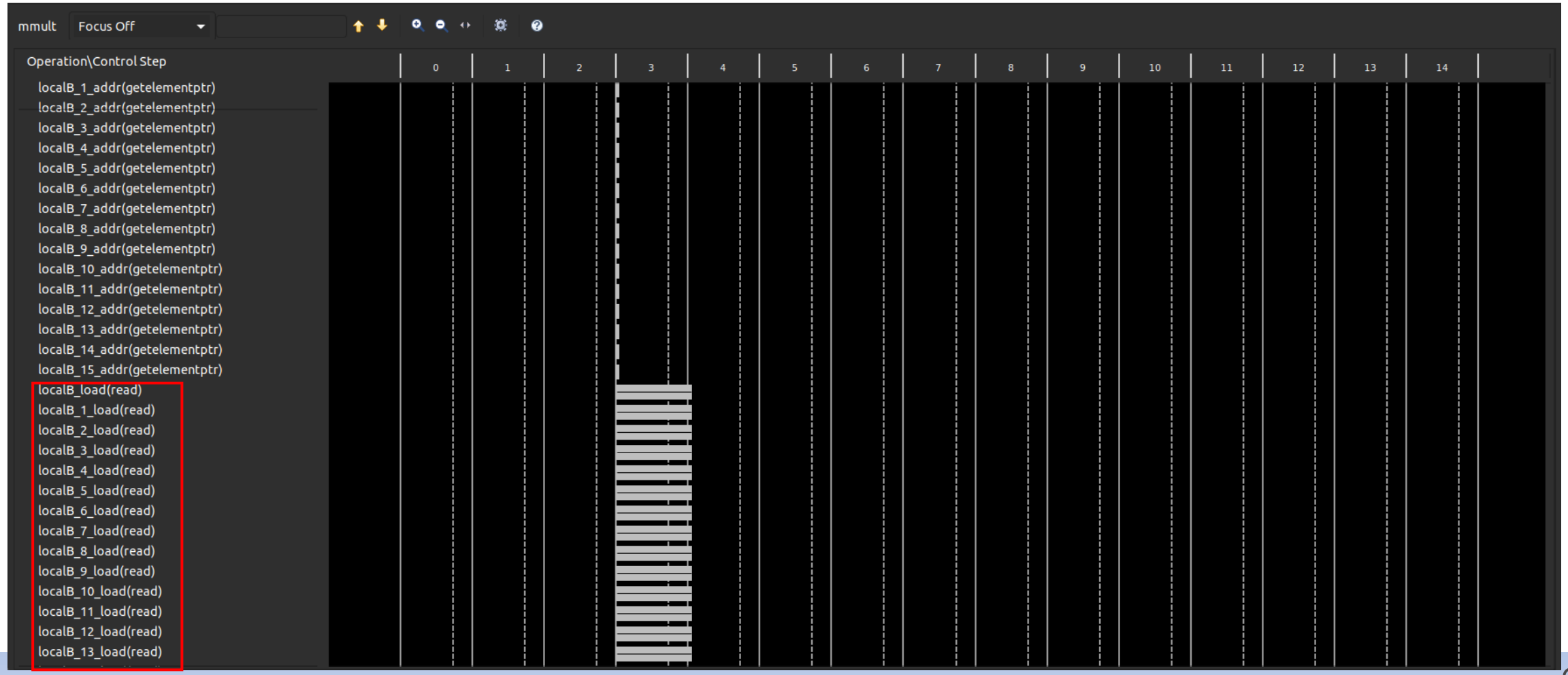
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



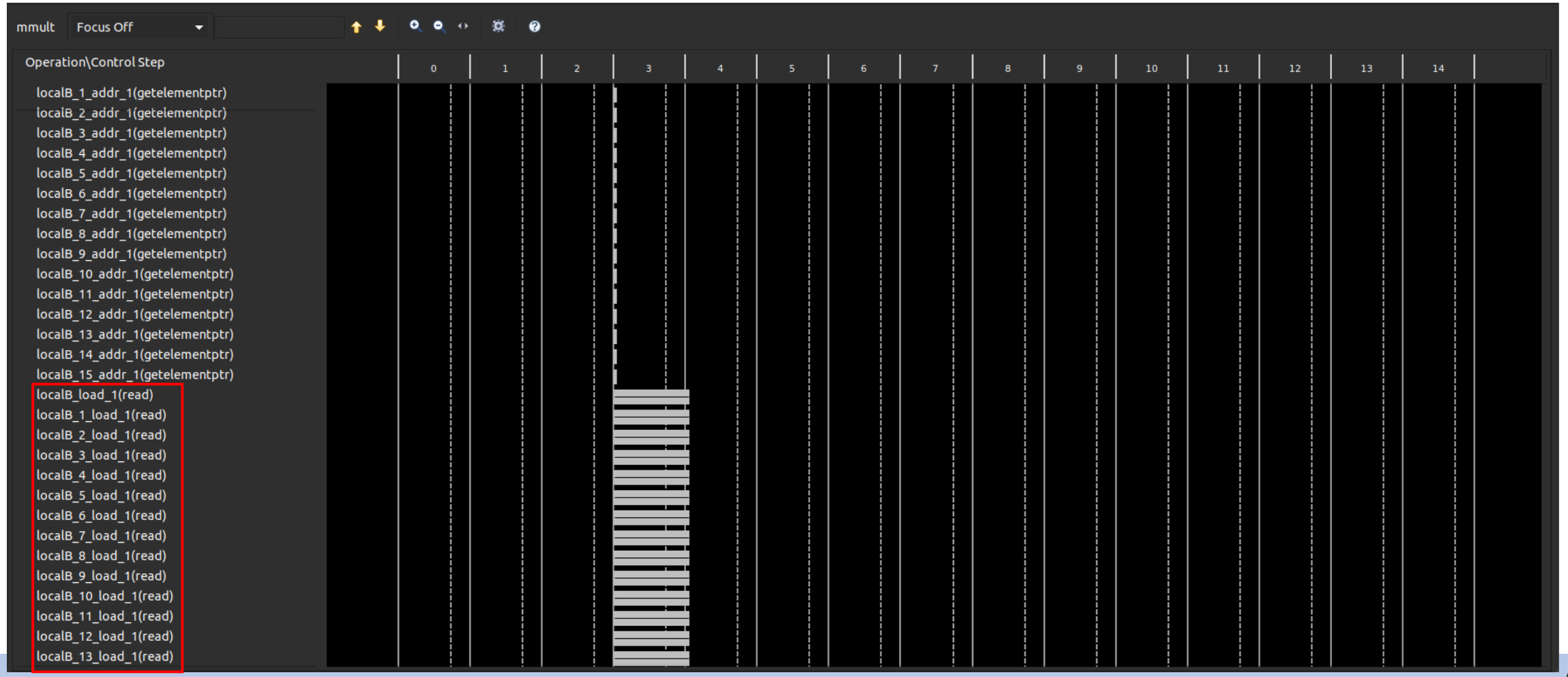
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



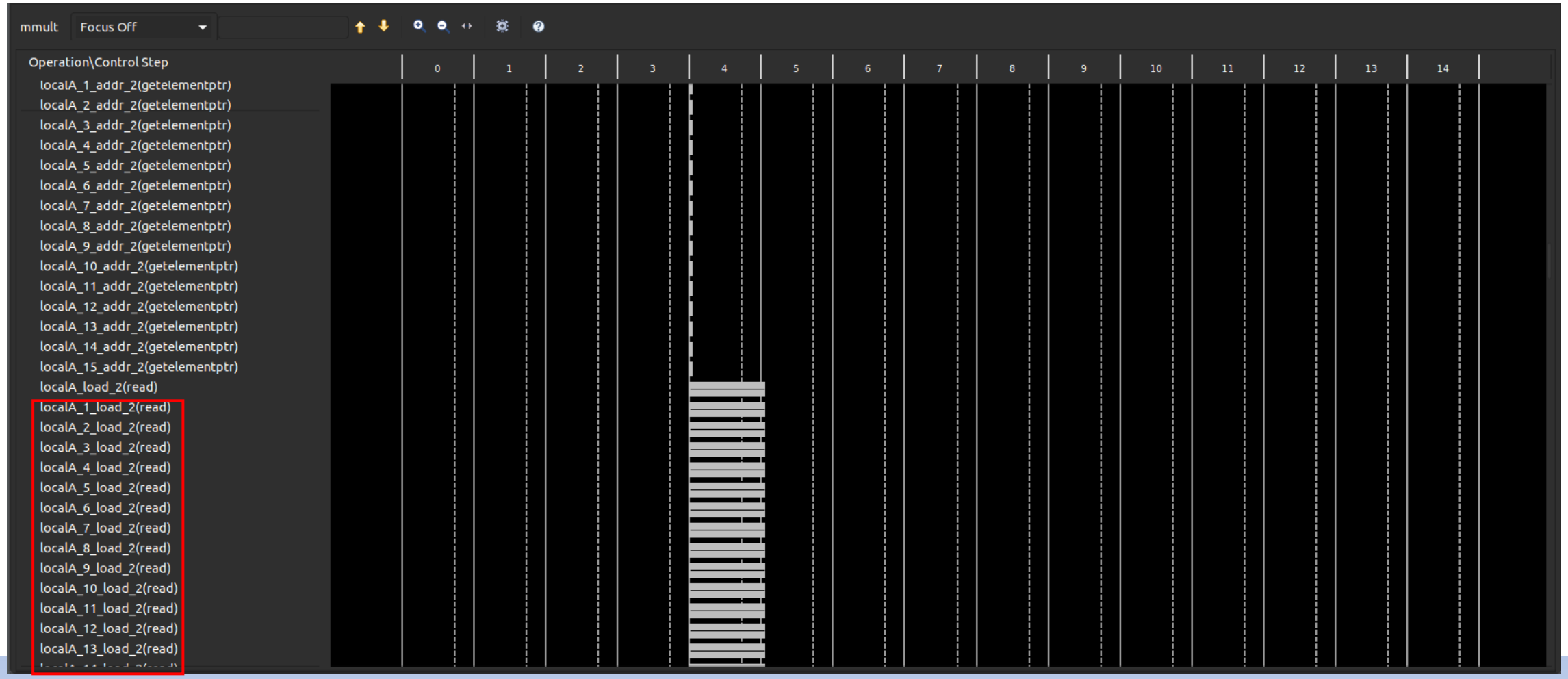
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



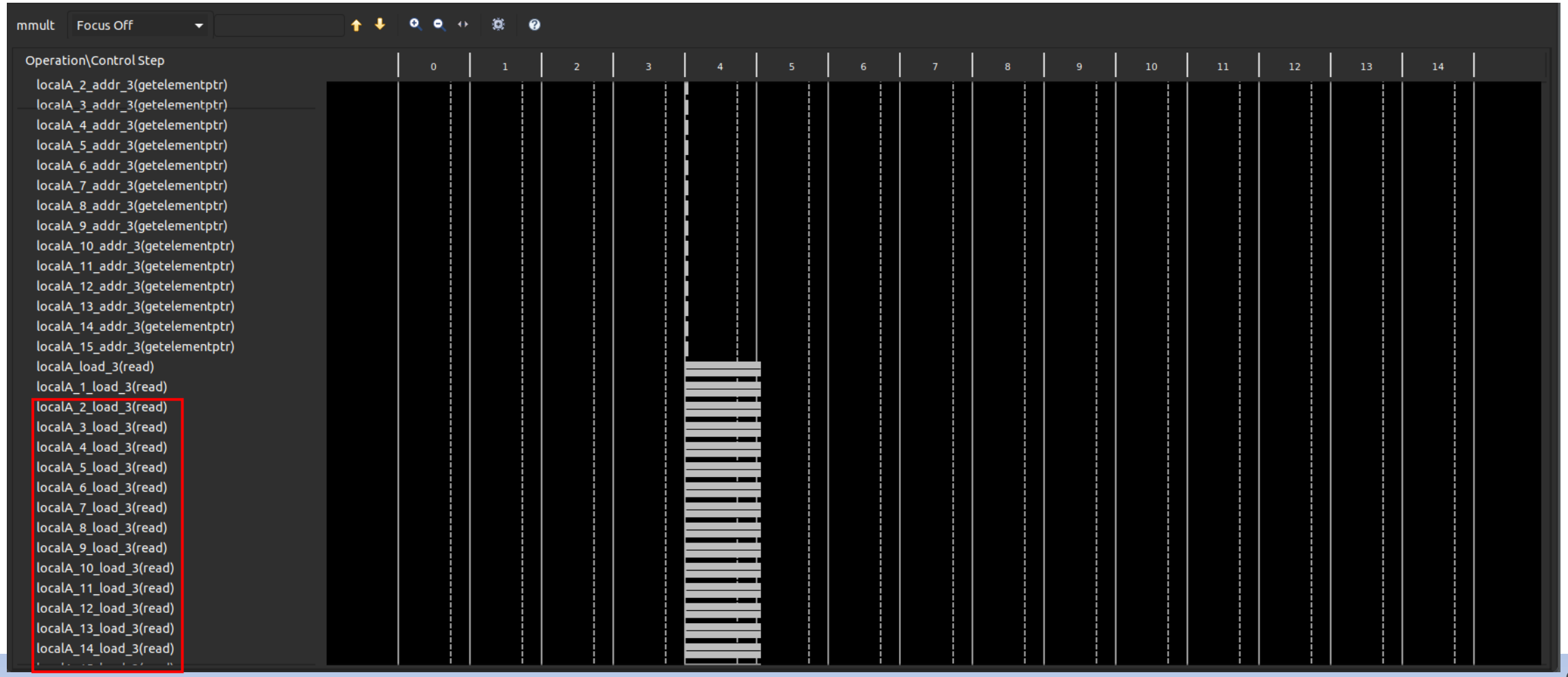
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



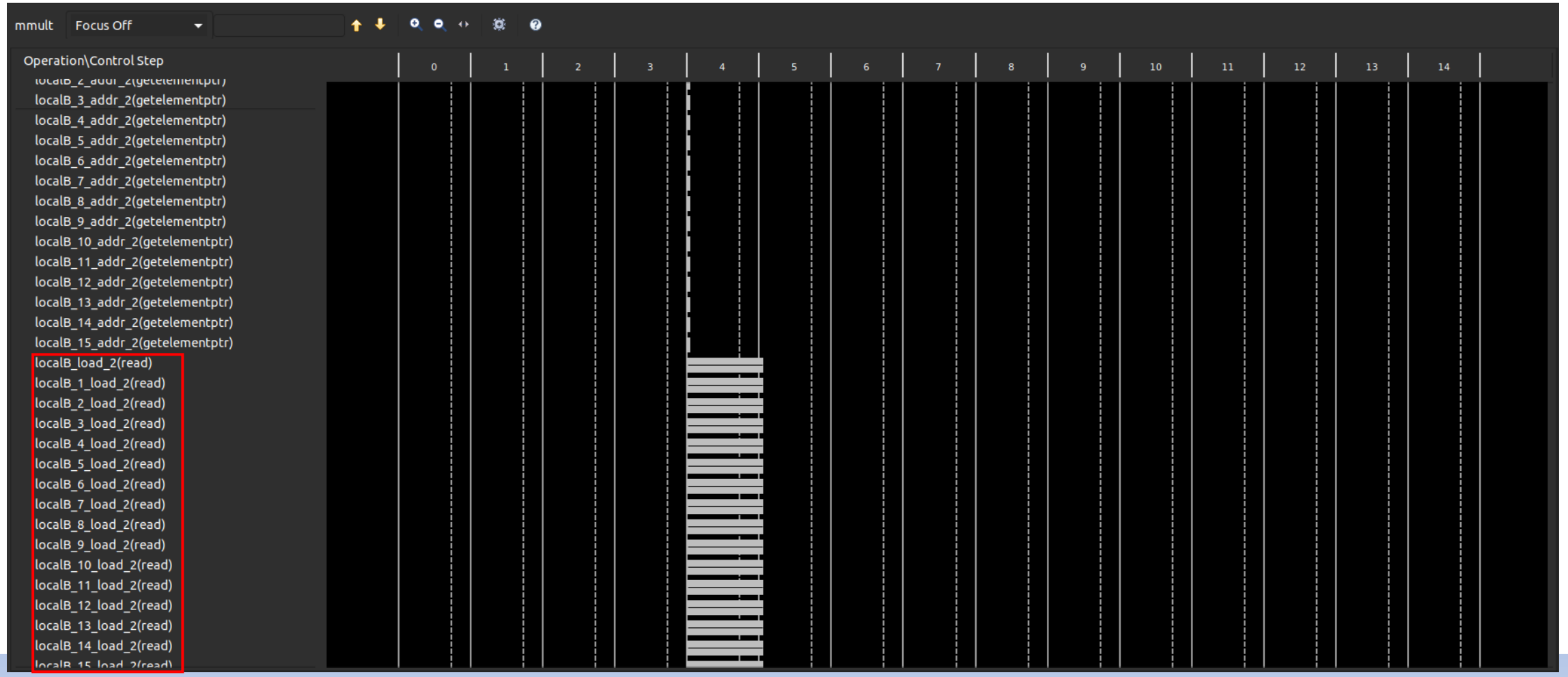
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



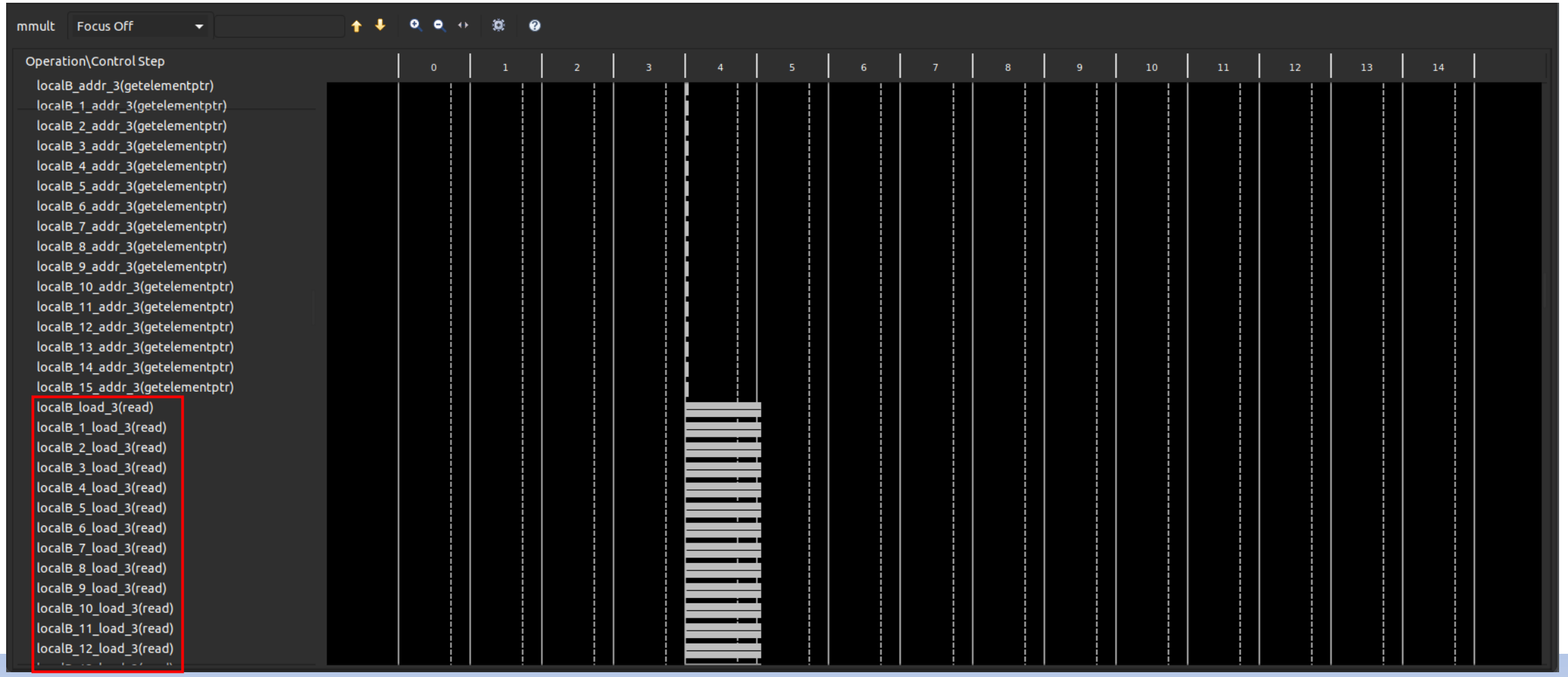
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



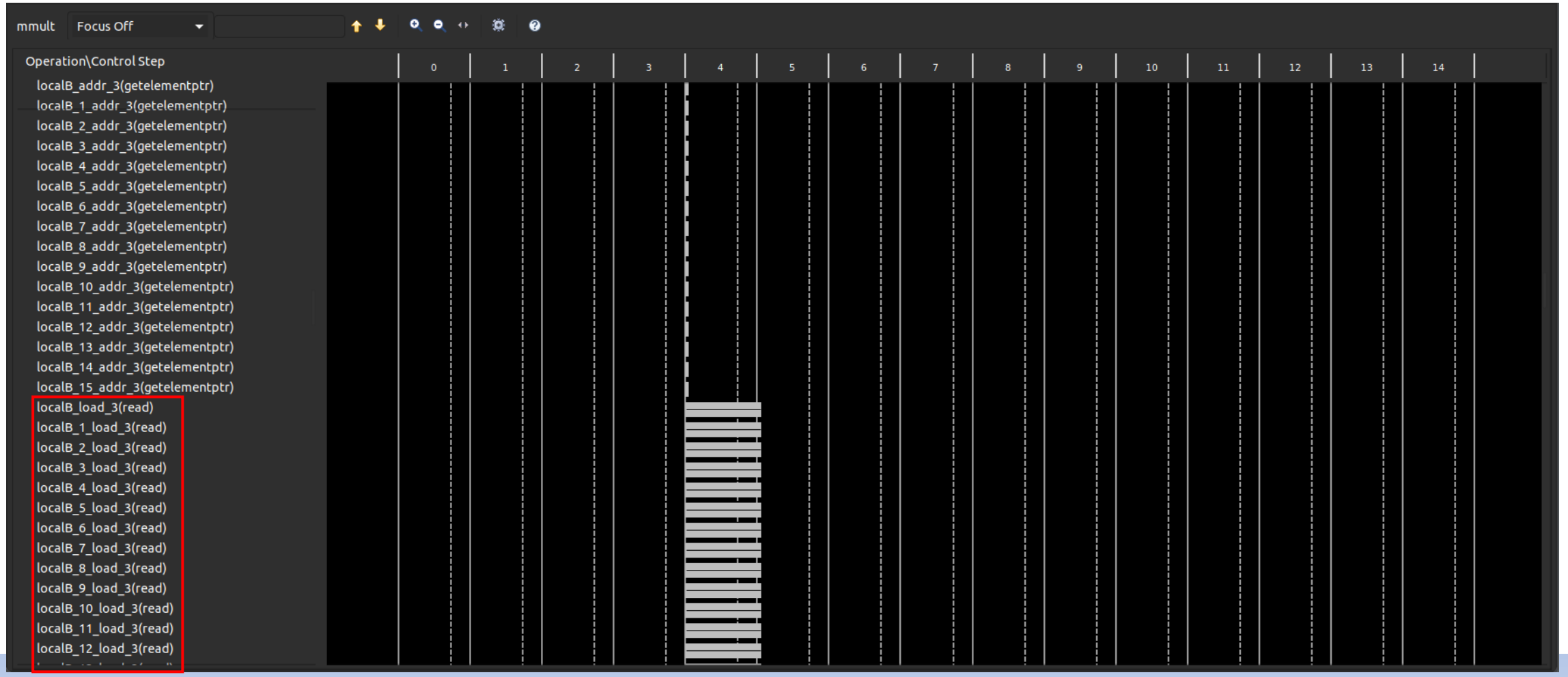
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



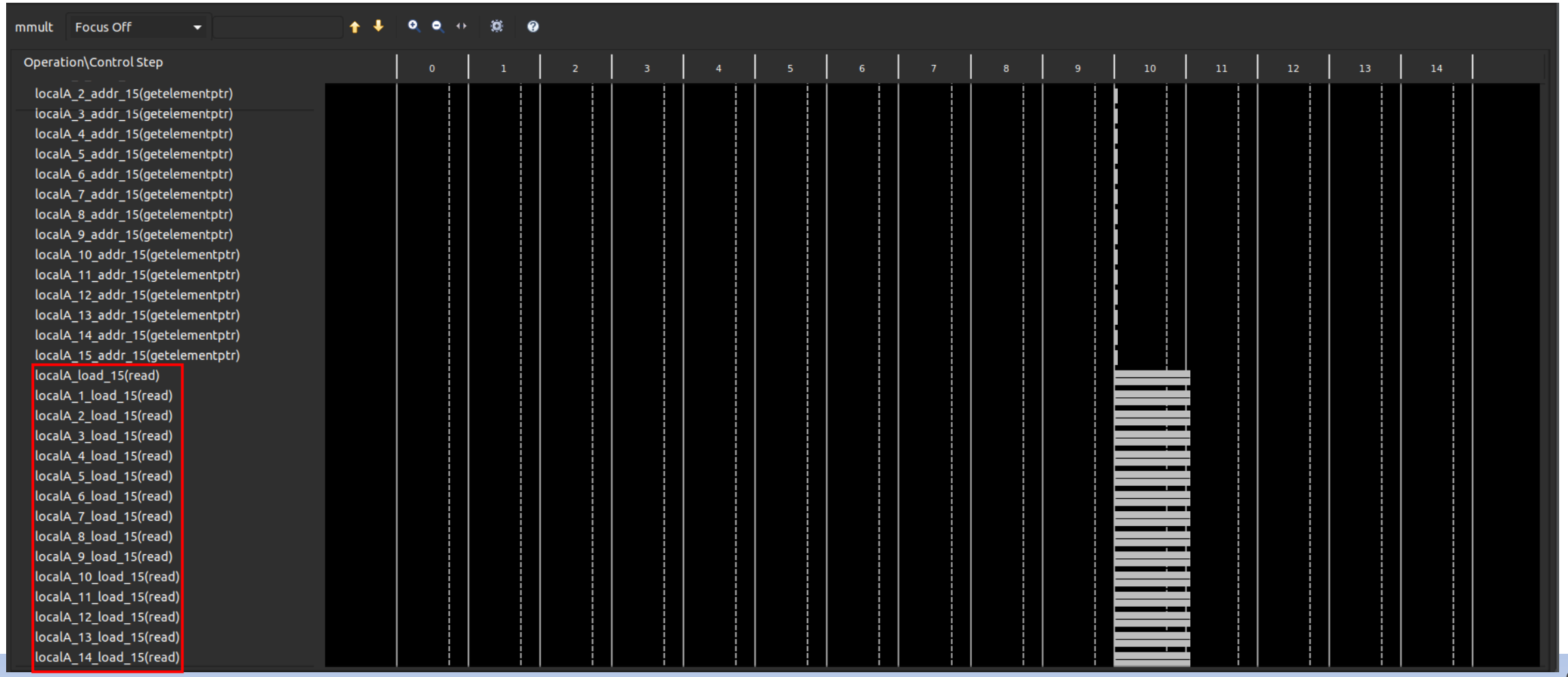
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



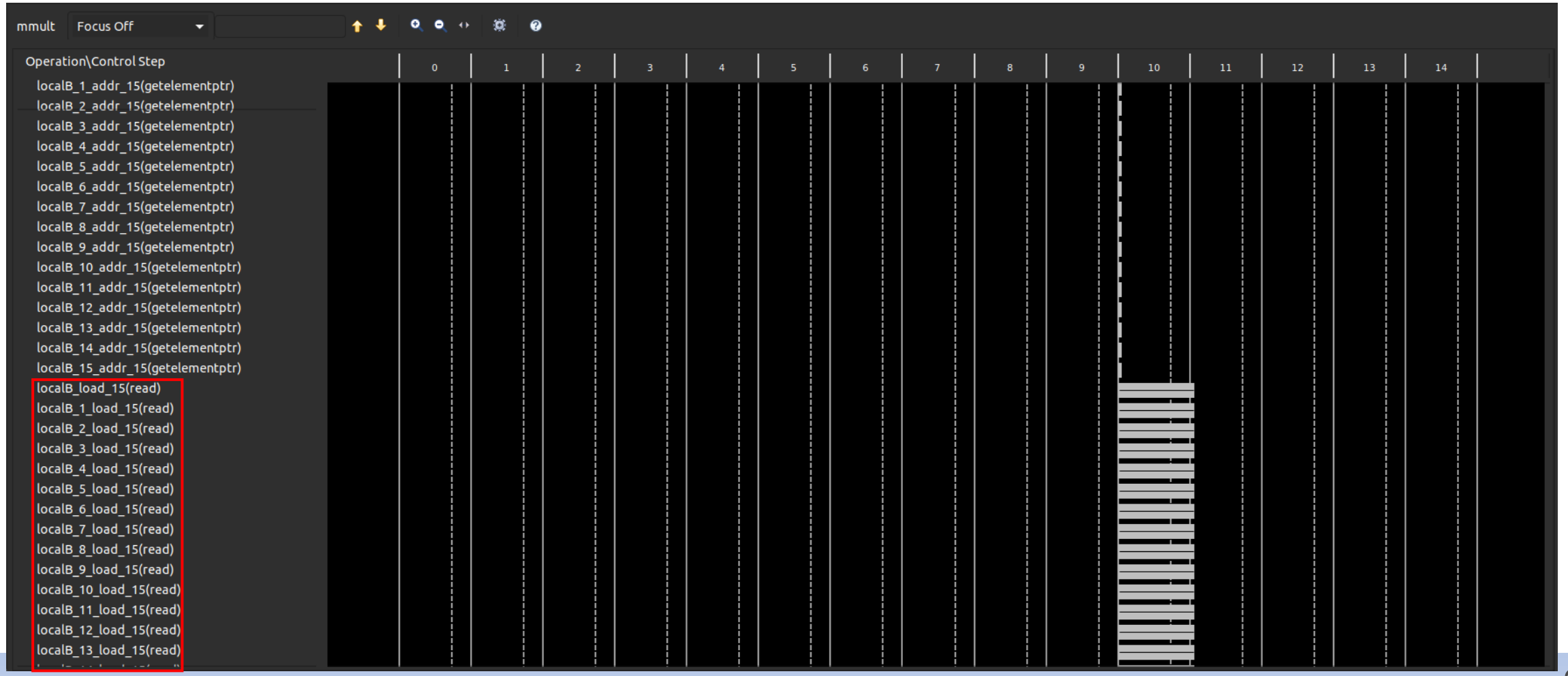
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



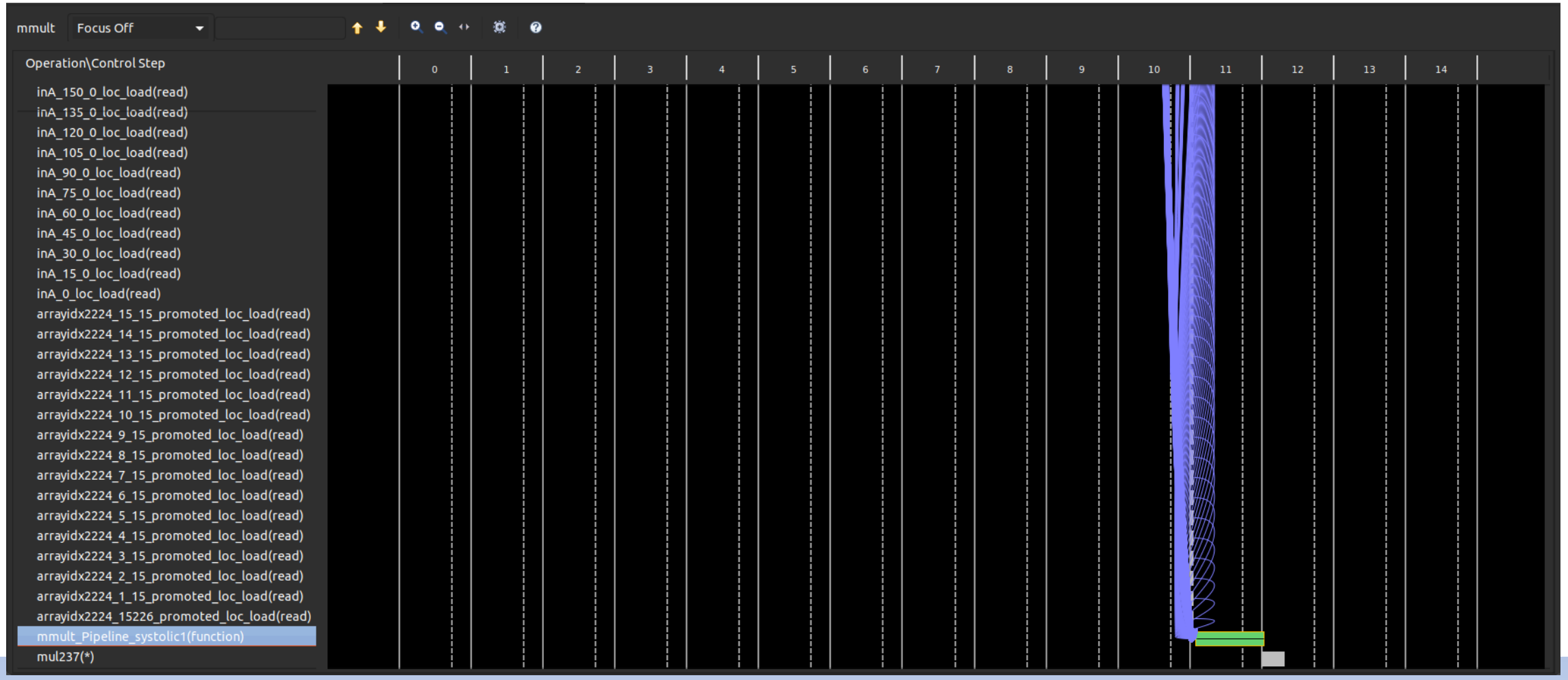
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



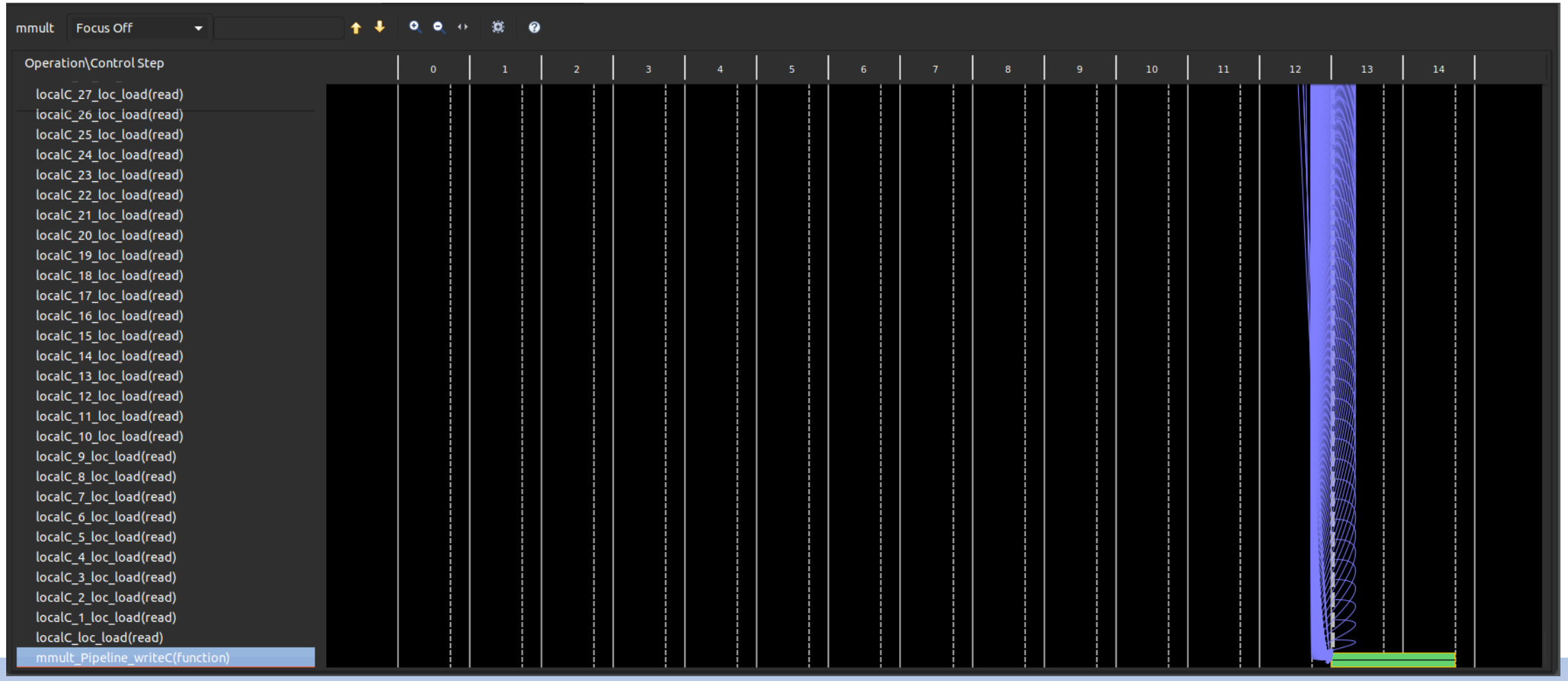
Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



Timing Trace – Schedule Viewer

- Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)



Time and Resource Analysis

The following information can be obtained from the simulation results of the Hardware Emulation's hw-link (Vitis Analyzer -> HLS Synthesis).

- Time Analysis
  - Implicit Type(Broadcasting)

Name	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined
mmult	685	2.283E3		686		no
mmult_Pipeline_init	18	59.994		18		no
init	16	53.328	1	1	16	yes
mmult_Pipeline_readA	328	1.093E3		328		no
readA	257	857.000	3	1	256	yes
mmult_Pipeline_readB	328	1.093E3		328		no
readB	257	857.000	3	1	256	yes
mmult_Pipeline_systolic1	21	69.993		21		no
systolic1	19	63.327	5	1	16	yes
mmult_Pipeline_writeC	325	1.083E3		325		no
writeC	256	853.000	2	1	256	yes

  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)

Name	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined
mmult						no
mmult_Pipeline_init	18	59.994		18		no
init	16	53.328	1	1	16	yes
mmult_Pipeline_readA	328	1.093E3		328		no
readA	257	857.000	3	1	256	yes
mmult_Pipeline_readB	328	1.093E3		328		no
readB	257	857.000	3	1	256	yes
mmult_Pipeline_systolic1	50	167.000		50		no
systolic1	48	160.000	4	1	46	yes
mmult_Pipeline_writeC						no
writeC			2	1		yes

Time and Resource Analysis

The following information can be obtained from the simulation results of the Hardware Emulation's hw-link (Vitis Analyzer -> HLS Synthesis).

- Resource Analysis
  - Implicit Type(Broadcasting)

Name	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)
mmult	0	0	768	12	72858	4	46649	5
mmult_Pipeline_init	0	0	0	0	7	~0	49	~0
init								
mmult_Pipeline_readA	0	0	0	0	203	~0	621	~0
readA								
mmult_Pipeline_readB	0	0	0	0	203	~0	621	~0
readB								
mmult_Pipeline_systolic1	0	0	759	12	59323	3	34079	3
systolic1								
mmult_Pipeline_writeC	0	0	0	0	200	~0	1718	~0
writeC								

  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)

Name	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)
mmult	0	0	771	12	96362	5	45003	5
mmult_Pipeline_init	0	0	0	0	7	~0	49	~0
init								
mmult_Pipeline_readA	0	0	0	0	203	~0	621	~0
readA								
mmult_Pipeline_readB	0	0	0	0	203	~0	621	~0
readB								
mmult_Pipeline_systolic1	0	0	762	12	74694	4	32553	3
systolic1								
mmult_Pipeline_writeC	0	0	0	0	200	~0	1718	~0
writeC								

Co – Simulation Timeline Trace

- Latency
  - Implicit Type(Broadcasting)

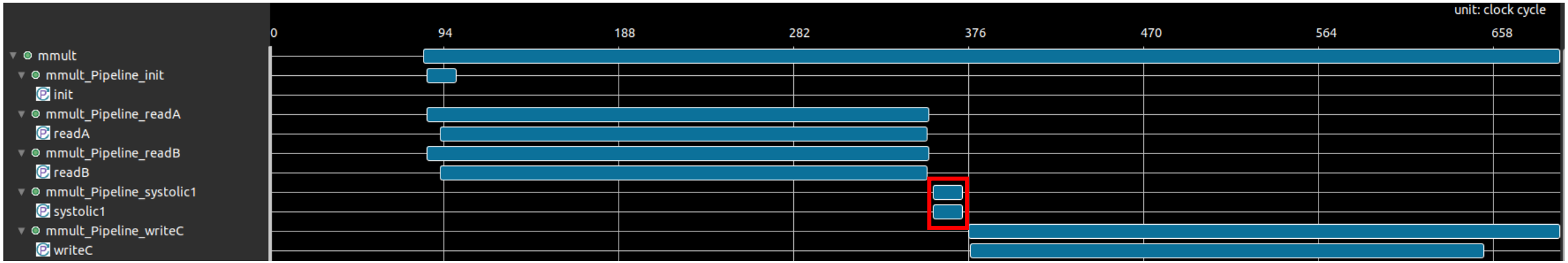
Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
mmult				609	609	609
mmult_Pipeline_init				16	16	16
mmult_Pipeline_readA				269	269	269
mmult_Pipeline_readB				269	269	269
mmult_Pipeline_systolic1				16	16	16
mmult_Pipeline_writeC				317	317	317

  - Explicit Type(Propagating) – Non-Stationary Systolic Array(NSSA)

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
mmult				647	647	647
mmult_Pipeline_init				16	16	16
mmult_Pipeline_readA				269	269	269
mmult_Pipeline_readB				269	269	269
mmult_Pipeline_systolic1				47	47	47
mmult_Pipeline_writeC				317	317	317

Co – Simulation Timeline Trace

- Implicit Type(Broadcasting)



- Explicit Type(Propagating) – Non-Stationary Systolic