Catapult – Lab 4: Hardware Accelerators

Margomenos Nikos

Exercise 1: Loop Acceleration

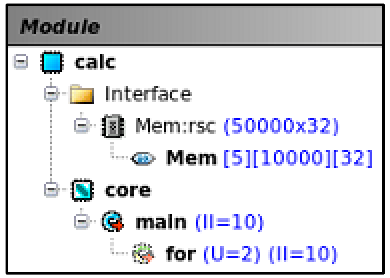
Code:

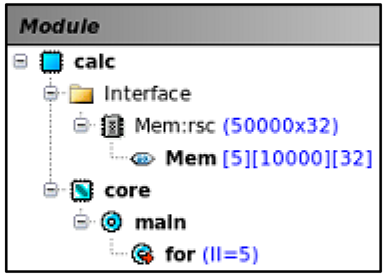
```
// Only 1 multiplier, 1 adder & 1 memory interface available
#pragma hls_design top
//void CCS_BLOCK(calc)(int Mem[5][N]){
void calc(int Mem[5][N]){
  for (int i=0; i<N; ++i){
    Mem[0][i] = Mem[1][i]*Mem[2][i] - Mem[3][i]*Mem[4][i];
  }
}
```

Settings & Results:

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
calc.v3 (extract)	50004	75006.00	50008	75012.00	4938.22	0.03
calc.v4 (extract)	**49999**	**74998.50**	**50000**	**75000.00**	**5687.38**	**0.02**

Frequency:	666.67	MHz
Period:	1.5	ns



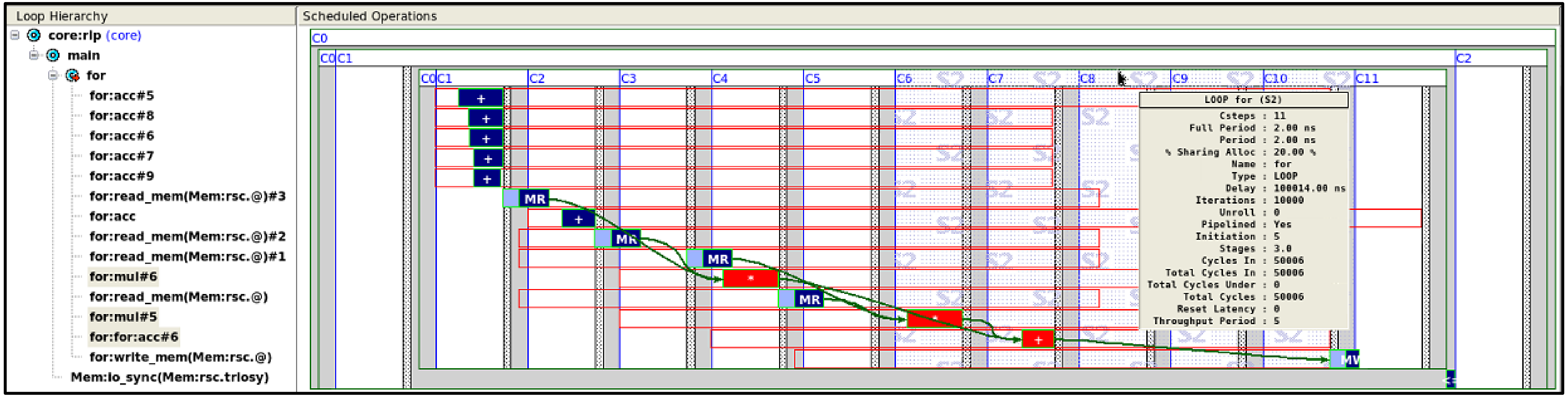


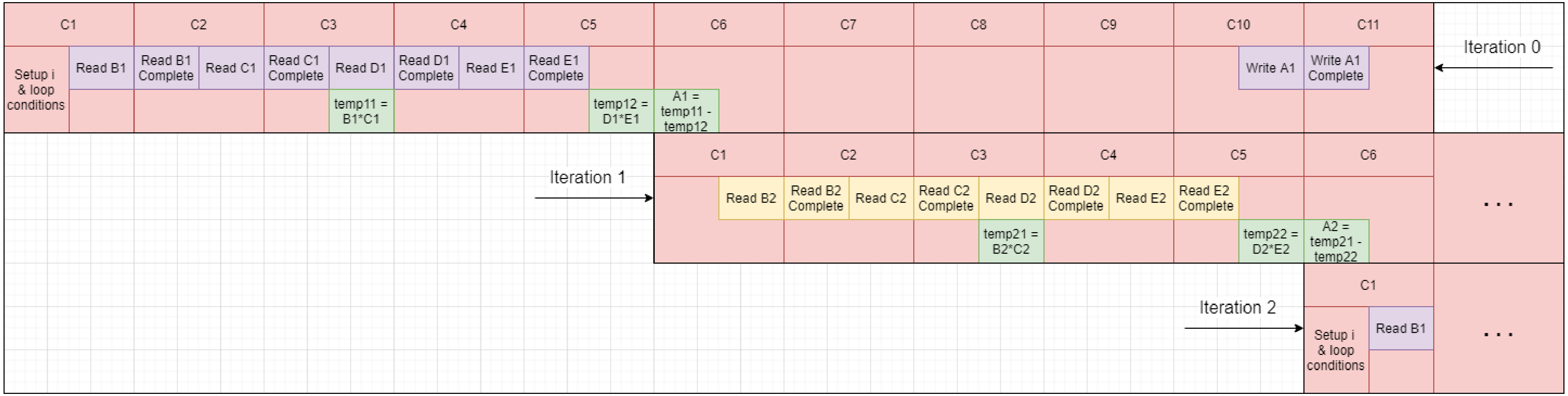
<u>#Note:</u> A pipeline with ii = 5 cannot be applied to the main loop

Highest Attainable Throughput:

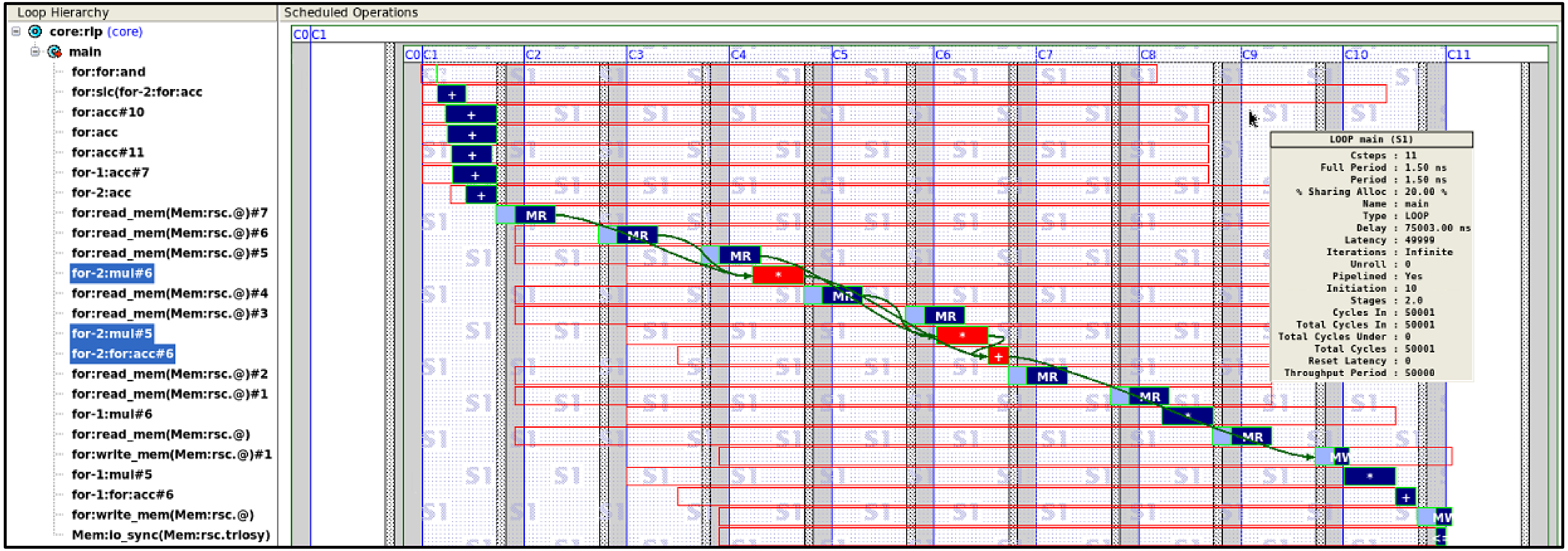
- Available Hardware:
  - One single-port memory
  - One multiplier
  - One adder
- Limitations:
  - There are 4 read and 1 write operations, for a total of 5 consecutive memory accesses
- Best-case scenario:
  - Theoretical pipeline with **<u>ii = 5</u>**
  - Practically implements **<u>unroll = 2</u>** & **<u>ii = 10</u>**

Exercise 1: Scheduling (ii = 5 & unroll = 0)





Exercise 1: Scheduling (ii = 10 & unroll = 2)



	C1		C2		C3		C4		C5		C6		C7		C8		C9		C10		C11		
Iteration 0	Setup i & loop conditions	Read B1	Read B1 Complete	Read C1	Read C1 Complete	Read D1	Read D1 Complete	Read E1	Read E1 Complete	Read B2	Read B2 Complete	Read C2	Read C2 Complete	Read D2	Read D2 Complete	Read E2	Read E2 Complete	Write A1	Write A1 Complete	Write A2	Write A2 Complete		
						temp11 = B1*C1				temp12 = D1*E1	A1 = temp11 - temp12			temp21 = B2*C2				temp22 = D2*E2	A2 = temp21 - temp22				
Iteration 1																					C1		
																					Setup i & loop conditions	Read B1	. . .

Exercise 2: Mean Filter Acceleration

Default Code:

```
// Dual-port memories available
#pragma hls_design top
//void CCS_BLOCK(mean_filter)(int img[N][M], int out [N][M]){
void mean_filter(int img[N][M], int out [N][M]){
  // Solution 1
  int kernel[5];
  // scan the image row by row
  ROW:for (int i=0; i<N; ++i) {
    // scan each row pixel by pixel from left to right
    COL:for (int j=0; j<M; ++j) {
      // get values of the pixels in the kernel
      kernel[0] = (j>1)   ? img[i][j-2] : 0;
      kernel[1] = (j>0)   ? img[i][j-1] : 0;
      kernel[2] = img[i][j];
      kernel[3] = (j<M-1) ? img[i][j+1] : 0;
      kernel[4] = (j<M-2) ? img[i][j+2] : 0;
      // compute the mean
      out[i][j] = (kernel[0]+kernel[1]+kernel[2]+kernel[3]+kernel[4]) / 5;
    }
  }
}
```

Improved Code:

```
// Dual-port memories available
#pragma hls_design top
//void CCS_BLOCK(mean_filter)(int img[N][M], int out[N][M]){
void mean_filter(int img[N][M], int out[N][M]){
  // Solution 2
  int kernel[6];
  // scan the image row by row
  ROW:for (int i=0; i<N; ++i) {
    // scan each row pixel by pixel from left to right
    COL:for (int j=-2; j<M; j+=2) {
      // Shift kernel
      kernel[0] = (j>=0)  ? kernel[2]   : 0;
      kernel[1] = (j>=0)  ? kernel[3]   : 0;
      kernel[2] = (j>=0)  ? kernel[4]   : 0;
      kernel[3] = (j>=0)  ? kernel[5]   : 0;
      kernel[4] = (j<M-2) ? img[i][j+2] : 0;
      kernel[5] = (j<M-3) ? img[i][j+3] : 0;
      // compute the mean
      if (j>=0){
        out[i][j]   = (kernel[0]+kernel[1]+kernel[2]+kernel[3]+kernel[4]) / 5;
        out[i][j+1] = (kernel[1]+kernel[2]+kernel[3]+kernel[4]+kernel[5]) / 5;
      }
    }
  }
}
```

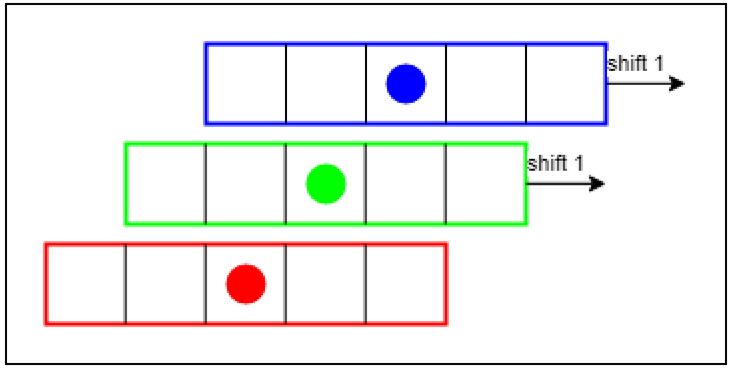
Exercise 2: Mean Filter Acceleration

Settings/Results (default code):

- Limiting factors:
  - Available resources **are** a factor:
    - 5 read operations from a dual-port memory, for a total of 3 consecutive cycles for memory accesses
  - Dependencies **are not** a factor:
    - 1 write operation but to a different memory, therefore no feedback path is created
- Best-case scenario:
  - Pipeline with ii = 3 wastes memory resources (only 1 read @ 3rd cycle)
  - **Unroll = 2** allows 10 read operations in 5 cycles, and so it can be pipelined with **ii = 5**

Settings/Results (improved code):

- Reduced read operations:
  - Neighboring pixels have overlapping kernels that do not need to be read from scratch in each iteration



  - With each shift, one pixel is read and one pixel is written. The dual-port memories allow for 2 shifts in 1 cycle, with a pipeline of **ii = 1** (no unroll)
  - When changing to a new row, an extra cycle is required for the first kernel

Exercise 2: Mean Filter Acceleration

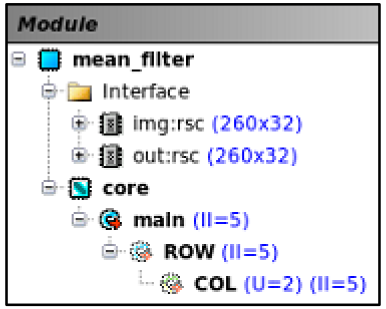
Best throughput (default code):

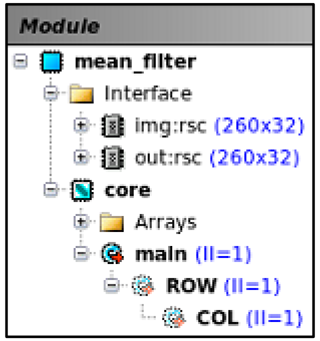
- Expected results:
  - 10(rows) * 26(columns) = 260 pixels
  - 260 * 5(kernel) = 1300 reads
  - 1300 / 2(reads/cycle) = 650 cycles
- Total throughput: **650 cycles**

Best throughput (improved code):

- Expected results:
  - 10(rows) * 26(columns) = 260 pixels
  - 260 / 2(reads/cycle) = 130 cycles
  - 10(rows) * 1(cycle/row) = 10 cycles
- Total throughput: **140 cycles**

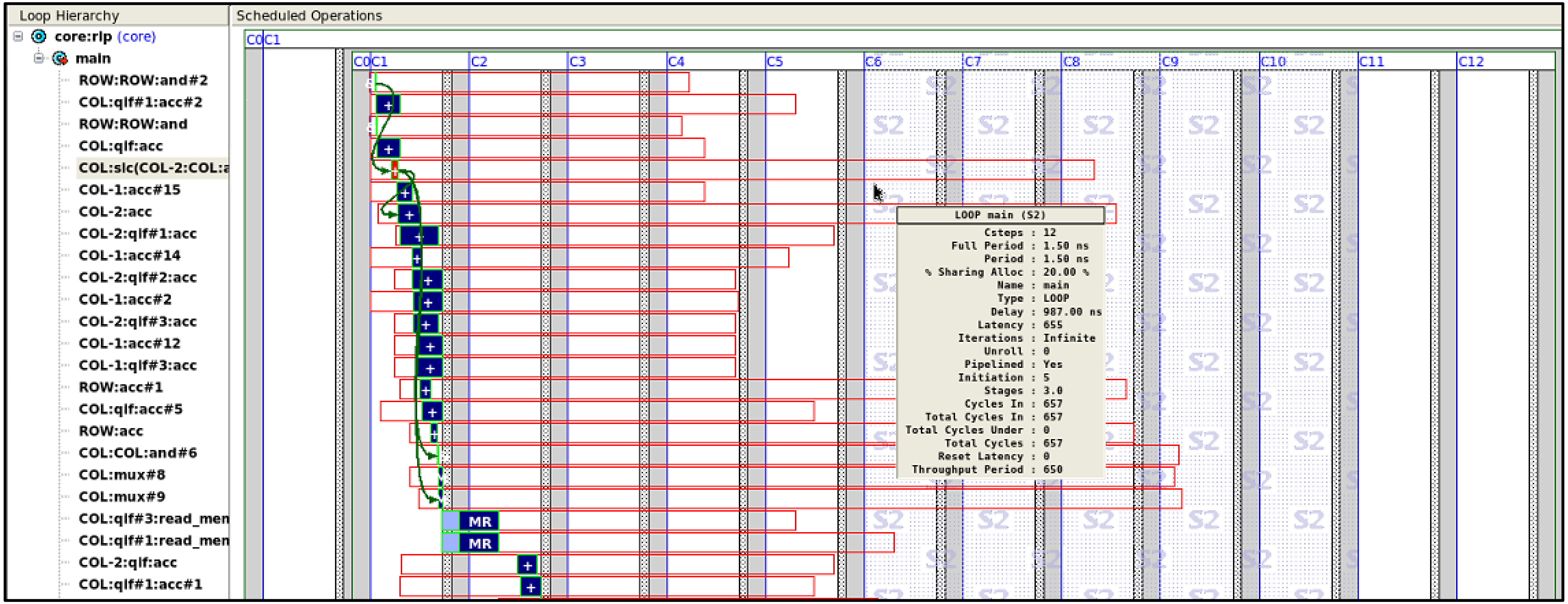
Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
mean_filter.v4 (extract)	655	982.50	650	975.00	7493.97	-0.08





Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
mean_filter.v2 (extract)	144	216.00	140	210.00	9689.76	0.01

Exercise 2a: Scheduling (ii = 5 & unroll = 2)



Exercise 2b: Scheduling (ii = 1)