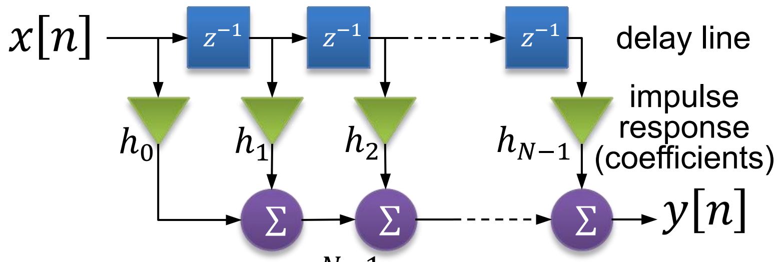
# Parallel Programming for FPGAs (part 1) Practice

# Lab1: Finite impulse response (FIR) filter

- 有限冲激响应滤波器
- $y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N]$

  $= \sum_{i=0}^{N} b_i \cdot x[n-i]$ (discrete convolution)
  - $x[n]$ is the input signal, $y[n]$ is the output signal
  - $N$ is the filter order
  - $b_i$ is the impulse response at the *i*th instant *(coefficient)*

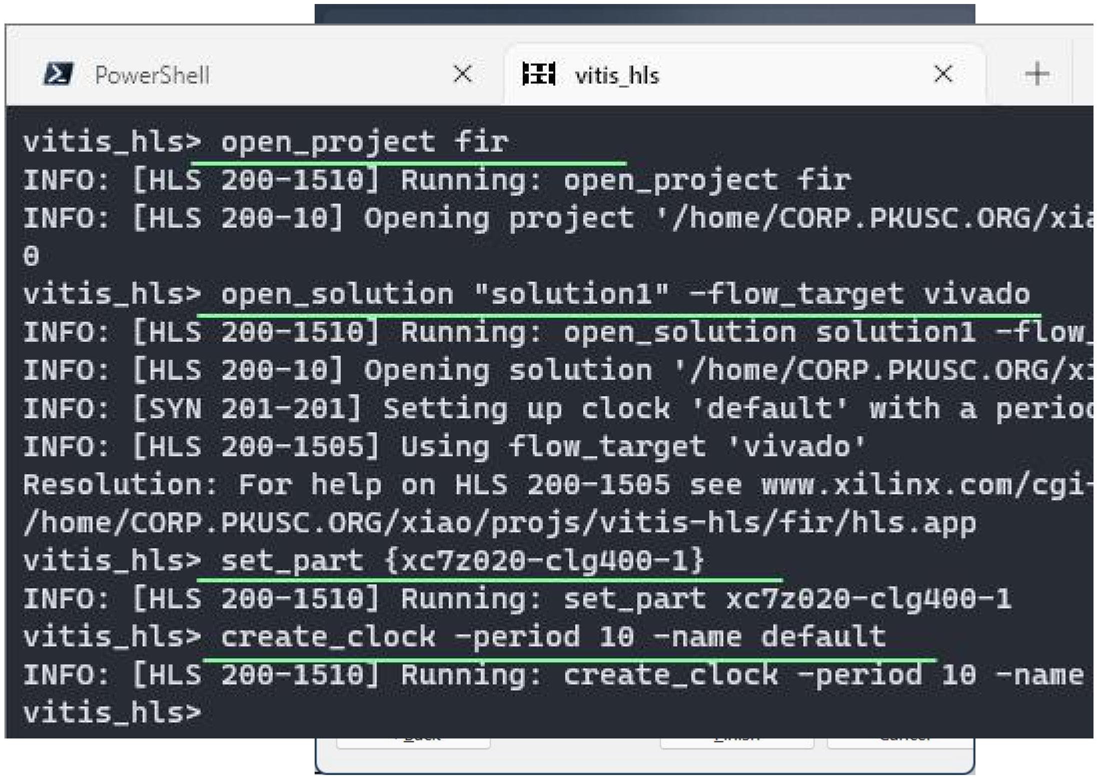


$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

# Create FIR project (1)

*GUI*

- Open *Vitis HLS* GUI
- Click `Create Project`
- Set project name
- Click `Next`s
  - until `Solution config.`
- Set Clock Period: 10 (ns)
- Set Part: xc7z020clg400-1



```
vitis_hls> open_project fir
INFO: [HLS 200-1510] Running: open_project fir
INFO: [HLS 200-10] Opening project '/home/CORP.PKUSC.ORG/xia
0
vitis_hls> open_solution "solution1" -flow_target vivado
INFO: [HLS 200-1510] Running: open_solution solution1 -flow_
INFO: [HLS 200-10] Opening solution '/home/CORP.PKUSC.ORG/x
INFO: [SYN 201-201] Setting up clock 'default' with a perio
INFO: [HLS 200-1505] Using flow_target 'vivado'
Resolution: For help on HLS 200-1505 see www.xilinx.com/cgi-
/home/CORP.PKUSC.ORG/xiao/projs/vitis-hls/fir/hls.app
vitis_hls> set_part {xc7z020-clg400-1}
INFO: [HLS 200-1510] Running: set_part xc7z020-clg400-1
vitis_hls> create_clock -period 10 -name default
INFO: [HLS 200-1510] Running: create_clock -period 10 -name
vitis_hls>
```

*Cmd*

- `source dir-to-vitis-hls/settings64.sh`
- `vitis_hls -i`: enter interactive mode
- type commands

# Create FIR project (2)

*GUI*

- Add Source files
  - fir.h, fir.cpp
- Add Test Bench
  - *.cpp, *.dat

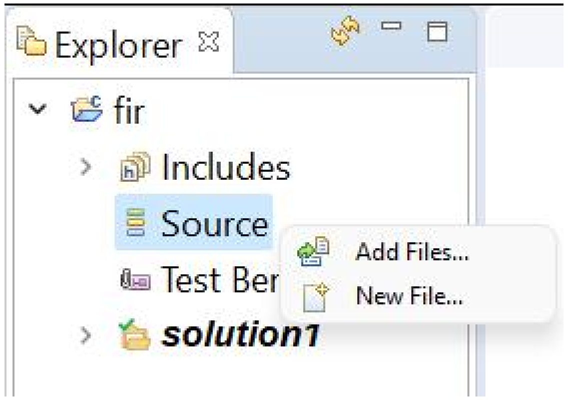
*CMD*

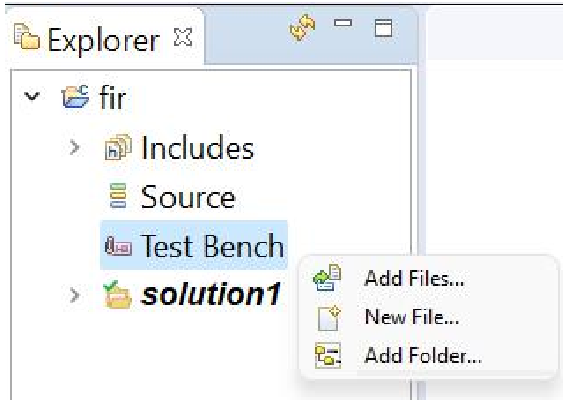
- type commands

```
add_files fir_srcs/fir.cpp
add_files fir_srcs/fir.h
add_files -tb fir_srcs/fir.cpp
add_files -tb fir_srcs/fir_tb.cpp
add_files -tb fir_srcs/input.dat
add_files -tb fir_srcs/out.gold.dat
```

*Source*

*test bench*





# Program design

- *fir.h*
  - macro, typedef
  - function declaration
- *fir.cpp*
  - function definition
- *fir_tb.cpp*
  - ground truth
  - kernel call

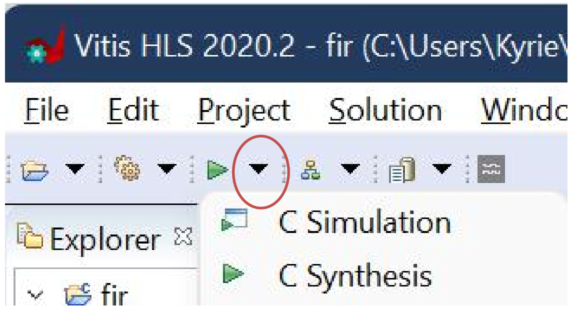
```
// fir.h
#define N 11
typedef int coef_t;
typedef int data_t;
typedef int acc_t;
void fir(data_t *y, data_t x);
```

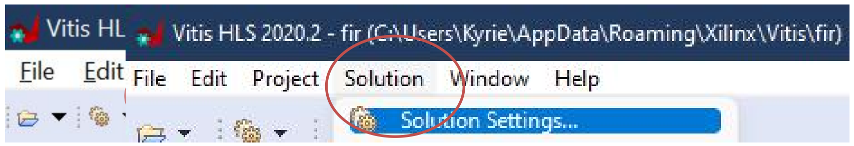
```
#include "fir.h"
void fir(data_t *y, data_t x) {
  coef_t c[N] = {53, 0, -91, 0, 313,
500, 313, 0, -91, 0, 53};
  static data_t shift_reg[N];
  acc_t acc = 0;
Shift_Accum_Loop:
  for (int i=N-1; i>=0; i--) {
    if (i==0) {
      acc += x * c[0];
      shift_reg[0] = x;
    } else {
      shift_reg[i] = shift_reg[i-1];
      acc += shift_reg[i] * c[i];
    }
   }
*y = acc;
}
```

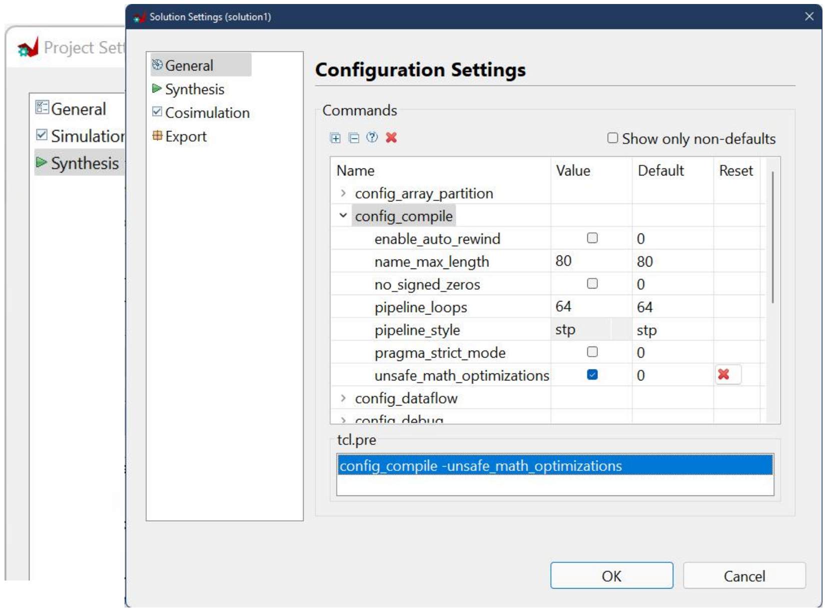
# C-Simulation, C-Synthesis (1)

*GUI*

- Run C-Simulation
- Set top function
- Set synthesis configuration
  - enable unsafe math optimizations


Vitis HLS 2020.2 - fir (C:\Users\Kyrie\
File Edit Project Solution Windo
Explorer
fir
C Simulation
C Synthesis


Vitis HLS 2020.2 - fir (C:\Users\Kyrie\AppData\Roaming\Xilinx\Vitis\fir)
File Edit Project Solution Window Help
Solution Settings...



Configuration Settings

Commands

☐ Show only non-defaults

| Name | Value | Default | Reset |
|---|---|---|---|
| › config_array_partition | | | |
| ⌄ config_compile | | | |
| enable_auto_rewind | ☐ | 0 | |
| name_max_length | 80 | 80 | |
| no_signed_zeros | ☐ | 0 | |
| pipeline_loops | 64 | 64 | |
| pipeline_style | stp | stp | |
| pragma_strict_mode | ☐ | 0 | |
| unsafe_math_optimizations | ☑ | 0 | ✖ |
| › config_dataflow | | | |
| › config_debug | | | |

tcl.pre

```
config_compile -unsafe_math_optimizations
```

- Run C Synthesis

## C-Simulation, C-Synthesis (2)

*CMD*

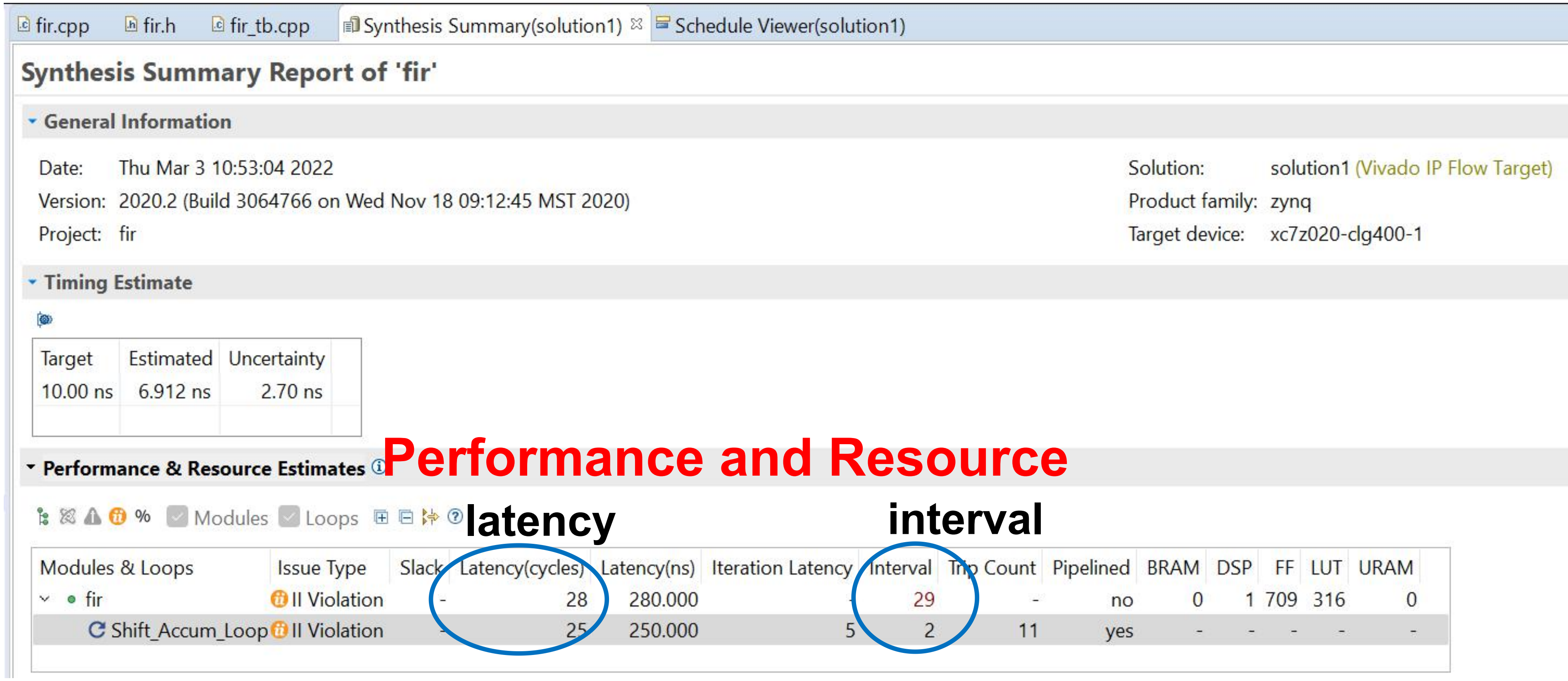
- type commands

```
csim_design

set_top fir

config_compile -unsafe_math_optimizations

csynth_design
```

# Observe reports (1)

*GUI*

- Open *Analysis* view – right top
- Look at *Synthesis Summary Report*



Synthesis Summary Report of 'fir'

General Information

| | | | |
|---|---|---|---|
| Date: | Thu Mar 3 10:53:04 2022 | Solution: | solution1 (Vivado IP Flow Target) |
| Version: | 2020.2 (Build 3064766 on Wed Nov 18 09:12:45 MST 2020) | Product family: | zynq |
| Project: | fir | Target device: | xc7z020-clg400-1 |

Timing Estimate

| Target | Estimated | Uncertainty |
|---|---|---|
| 10.00 ns | 6.912 ns | 2.70 ns |

Performance & Resource Estimates

**Performance and Resource**

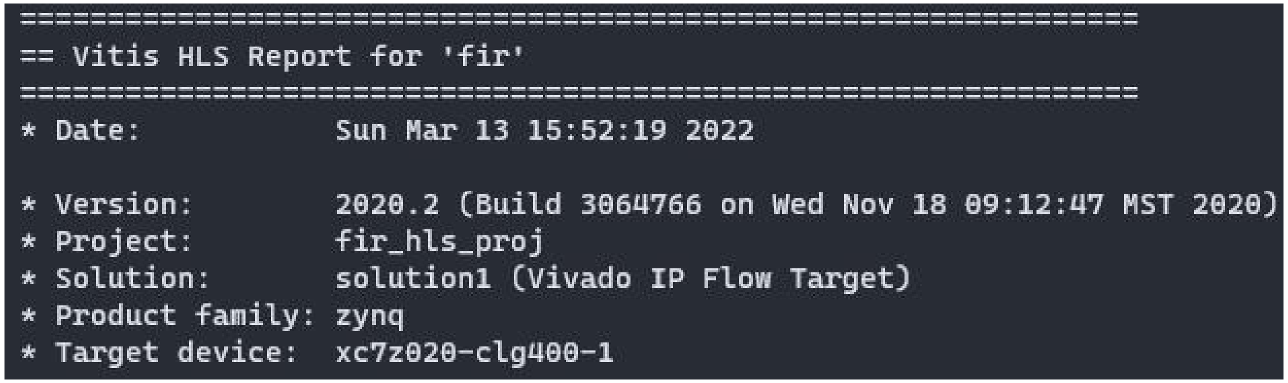
**latency** **interval**

| Modules & Loops | Issue Type | Slack | Latency(cycles) | Latency(ns) | Iteration Latency | Interval | Trip Count | Pipelined | BRAM | DSP | FF | LUT | URAM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fir | II Violation | - | 28 | 280.000 | - | 29 | - | no | 0 | 1 | 709 | 316 | 0 |
| Shift_Accum_Loop | II Violation | - | 25 | 250.000 | 5 | 2 | 11 | yes | - | - | - | - | - |

# Observe reports (2)

*CMD*

- cd to the vitis-hls project directory
- cd to the solution directory
- cd to `syn` directory
- cd to `report` directory
- open `[top function]_csynth.rpt`

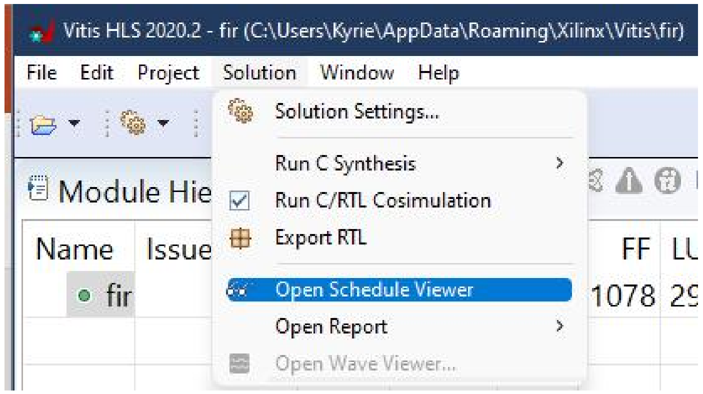


```
================================================================
== Vitis HLS Report for 'fir'
================================================================
* Date:           Sun Mar 13 15:52:19 2022

* Version:        2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)
* Project:        fir_hls_proj
* Solution:       solution1 (Vivado IP Flow Target)
* Product family: zynq
* Target device:  xc7z020-clg400-1
```

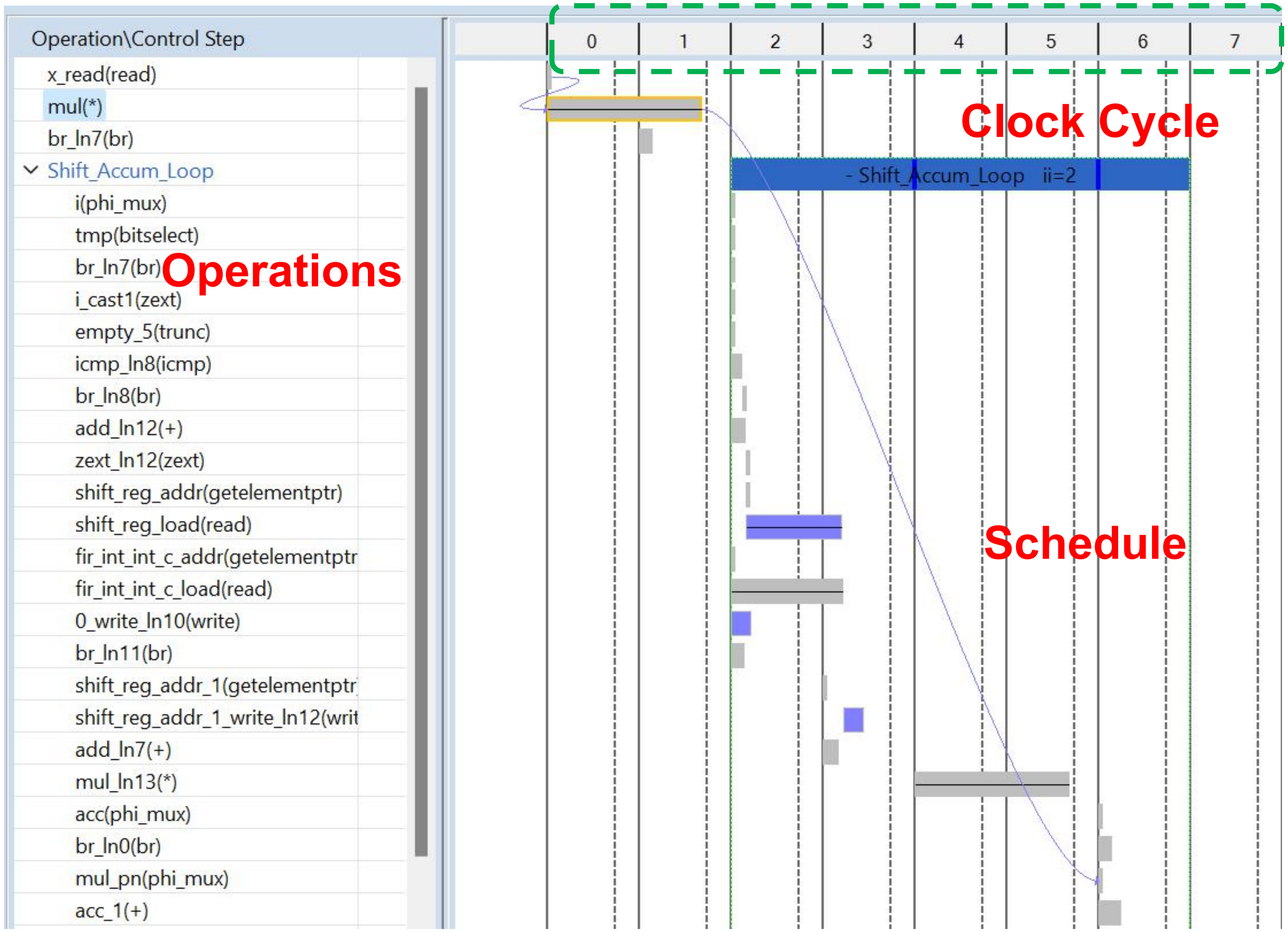
## Observe Schedule Viewer (1)

*GUI*

- Open Schedule Viewr



# Observe Schedule Viewer (2)



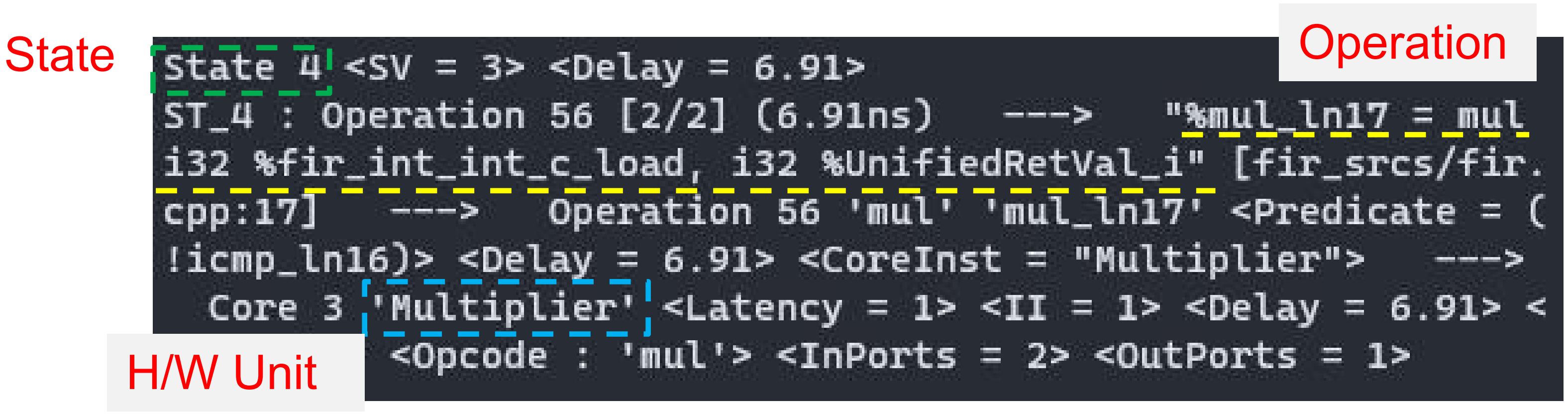
Operation\Control Step

x_read(read)
mul(*)
br_ln7(br)
Shift_Accum_Loop
i(phi_mux)
tmp(bitselect)
br_ln7(br)
i_cast1(zext)
empty_5(trunc)
icmp_ln8(icmp)
br_ln8(br)
add_ln12(+)
zext_ln12(zext)
shift_reg_addr(getelementptr)
shift_reg_load(read)
fir_int_int_c_addr(getelementptr
fir_int_int_c_load(read)
0_write_ln10(write)
br_ln11(br)
shift_reg_addr_1(getelementptr
shift_reg_addr_1_write_ln12(writ
add_ln7(+)
mul_ln13(*)
acc(phi_mux)
br_ln0(br)
mul_pn(phi_mux)
acc_1(+)

# Observe Schedule Viewer (3)

*CMD*

- cd to solution directory
- cd to `.autopilot/db/` directory
- open `[top_function].verbose.sched.rpt`
  - FSM



```
State 4 <SV = 3> <Delay = 6.91>
ST_4 : Operation 56 [2/2] (6.91ns)   --->   "%mul_ln17 = mul
i32 %fir_int_int_c_load, i32 %UnifiedRetVal_i" [fir_srcs/fir.
cpp:17]   --->   Operation 56 'mul' 'mul_ln17' <Predicate = (
!icmp_ln16)> <Delay = 6.91> <CoreInst = "Multiplier">   --->
  Core 3 'Multiplier' <Latency = 1> <II = 1> <Delay = 6.91> <
         <Opcode : 'mul'> <InPorts = 2> <OutPorts = 1>
```

# Lab2: Matrix Vector Multiplication

- MVM: $V_{out}[i] = \sum_k M[i][k] * V_{in}[k]$

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} \odot \begin{bmatrix}x\\y\\z\end{bmatrix} = \begin{bmatrix}1\cdot x + 0\cdot x + 0\cdot x\\0\cdot y + 1\cdot y + 0\cdot y\\0\cdot z + 0\cdot z + 1\cdot z\end{bmatrix} = \begin{bmatrix}x\\y\\z\end{bmatrix}$$

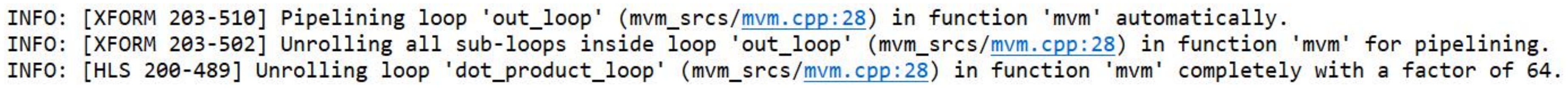
# Create baseline

- Please create and configure the project as you did in FIR example.
- Implement a simple baseline.
- Pass C-Simulation and run Synthesis

```
// mvm.h
typedef float in_t;
typedef float out_t;
#define N 64
// mvm.cpp
void mvm(in_t M[N][N], in_t V_In[N], out_t V_Out[N]) {
out_loop:
  for (int i = 0; i < N; i++) {
    out_t sum = 0;
dot_product_loop:
    for (int j = 0; j < N; j++) {
      sum += V_In[j] * M[i][j];
    }
    V_Out[i] = sum;
  }
}
```

# Analyze MVM Baseline (1)

- What optimizations does *Vitis HLS* apply automatically?
  - Look at the *Vitis HLS Console* or *log information.*

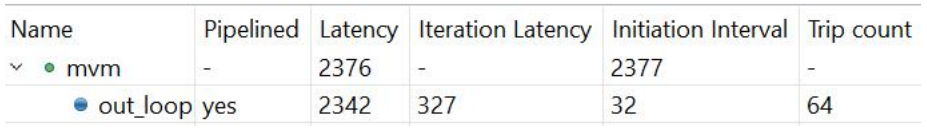


```
INFO: [XFORM 203-510] Pipelining loop 'out_loop' (mvm_srcs/mvm.cpp:28) in function 'mvm' automatically.
INFO: [XFORM 203-502] Unrolling all sub-loops inside loop 'out_loop' (mvm_srcs/mvm.cpp:28) in function 'mvm' for pipelining.
INFO: [HLS 200-489] Unrolling loop 'dot_product_loop' (mvm_srcs/mvm.cpp:28) in function 'mvm' completely with a factor of 64.
```

- Pipeline “out_loop”, fully unroll “dot_product_loop”

# Analyze MVM Baseline (2)

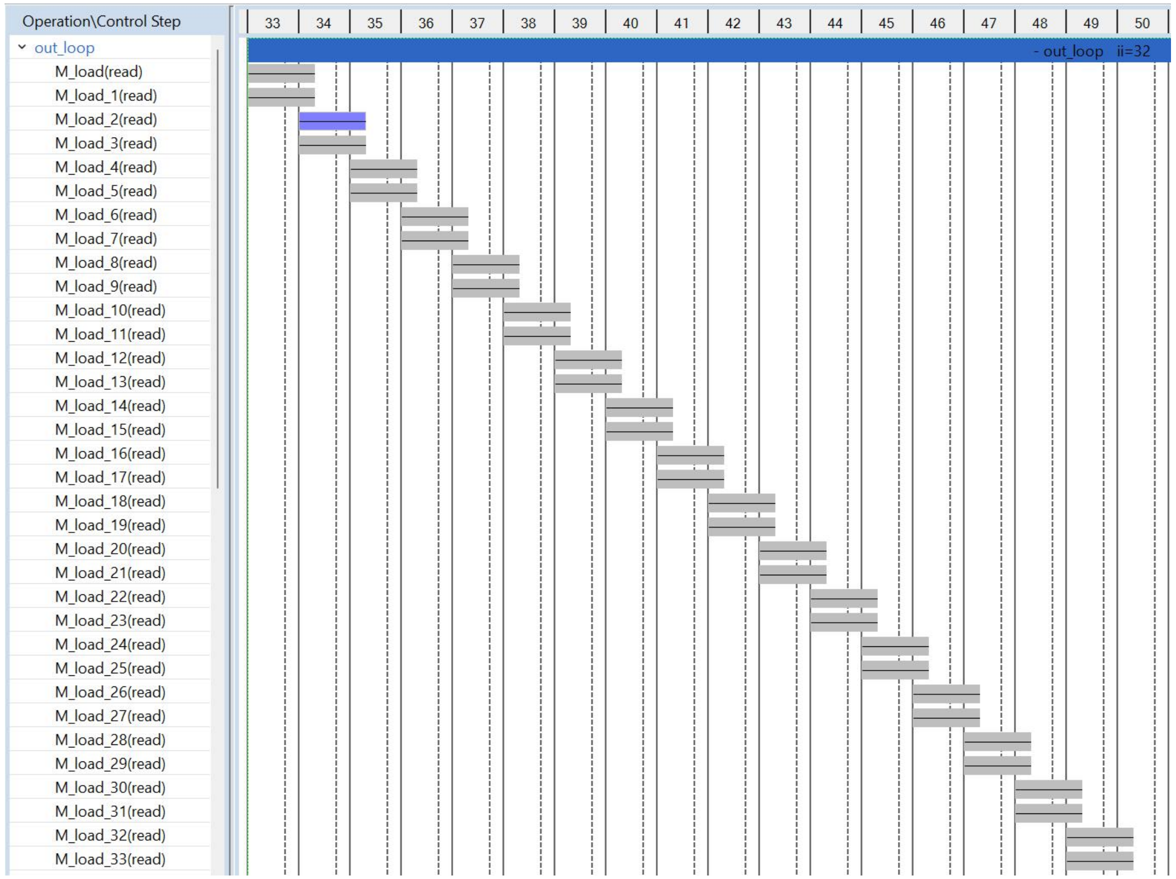
- Why the II is 32?



| Name | Pipelined | Latency | Iteration Latency | Initiation Interval | Trip count |
|---|---|---|---|---|---|
| mvm | - | 2376 | - | 2377 | - |
| out_loop | yes | 2342 | 327 | 32 | 64 |

  – Look at the *Schedule Viewer*

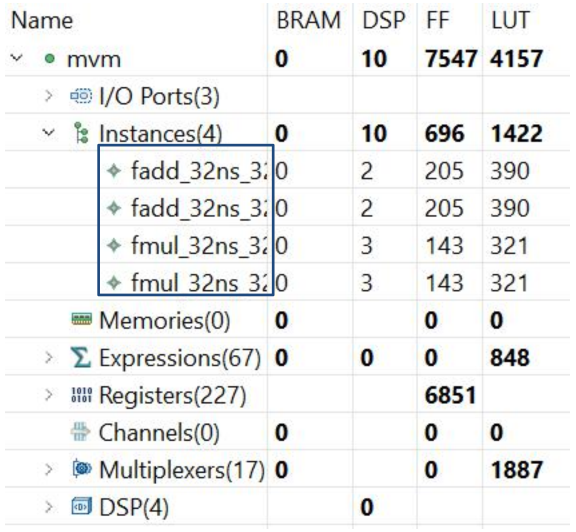
# Analyze MVM Baseline (3)



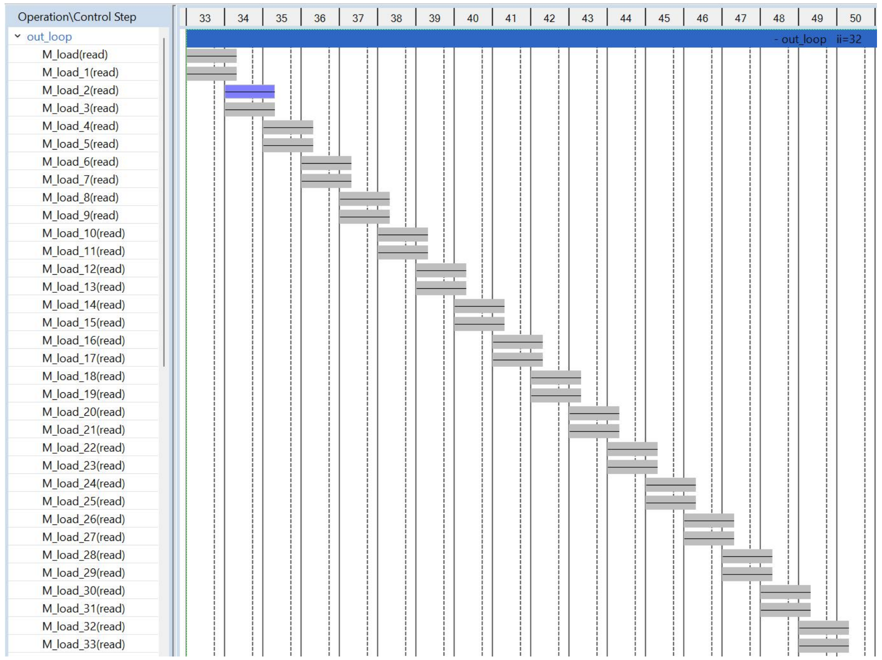
- It take 32 cycles to read V_In[] and M[i][]

# Analyze MVM Baseline (4)

- Why two *fadds* and two *fmuls*?



| Name | BRAM | DSP | FF | LUT |
|---|---|---|---|---|
| mvm | **0** | **10** | **7547** | **4157** |
| I/O Ports(3) | | | | |
| Instances(4) | **0** | **10** | **696** | **1422** |
| fadd_32ns_3: | 0 | 2 | 205 | 390 |
| fadd_32ns_3: | 0 | 2 | 205 | 390 |
| fmul_32ns_3: | 0 | 3 | 143 | 321 |
| fmul_32ns_3: | 0 | 3 | 143 | 321 |
| Memories(0) | **0** | | **0** | **0** |
| Expressions(67) | **0** | **0** | **0** | **848** |
| Registers(227) | | | **6851** | |
| Channels(0) | **0** | | **0** | **0** |
| Multiplexers(17) | **0** | | **0** | **1887** |
| DSP(4) | | **0** | | |



```
Operation\Control Step   33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
out_loop                                                    - out_loop  ii=32
  M_load(read)
  M_load_1(read)
  M_load_2(read)
  M_load_3(read)
  M_load_4(read)
  M_load_5(read)
  M_load_6(read)
  M_load_7(read)
  M_load_8(read)
  M_load_9(read)
  M_load_10(read)
  M_load_11(read)
  M_load_12(read)
  M_load_13(read)
  M_load_14(read)
  M_load_15(read)
  M_load_16(read)
  M_load_17(read)
  M_load_18(read)
  M_load_19(read)
  M_load_20(read)
  M_load_21(read)
  M_load_22(read)
  M_load_23(read)
  M_load_24(read)
  M_load_25(read)
  M_load_26(read)
  M_load_27(read)
  M_load_28(read)
  M_load_29(read)
  M_load_30(read)
  M_load_31(read)
  M_load_32(read)
  M_load_33(read)
```

  - Read 2 M[i][]s and do *fmuls* and *fadds* for them.
  - HLS executes two *fadds* in parallel, and so as *fmuls*.

# Quantization with AP_Fixed

- Use the “shortest” data type under precision requirements.

```
// mvm.h
#include "ap_fixed.h"
typedef ap_ufixed<8, 5> in_t;
```

Choose an appropriate `out_t`

```
// [xxxxx.xxx] * [xxxxx.xxx] * 64 = [16b.6b]
typedef ap_ufixed<22, 16> out_t;
```

*Please try to optimize the design* to satisfy the constraints: #DSP<10, #BRAM <100.

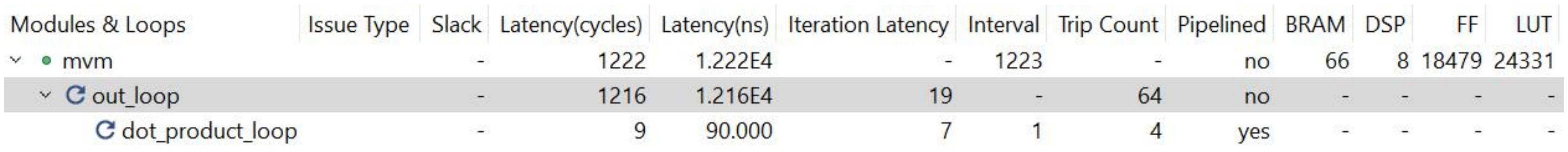
What’s your best latency?

$\leq 2000$ ?

# Optimized design

```
void mvm(in_t M[N][N], in_t V_In[N], out_t V_Out[N]) {
#pragma HLS interface m_axi port=M
#pragma HLS interface m_axi port=V_In
#pragma HLS interface m_axi port=V_Out

#pragma HLS array_partition variable=V_In cyclic factor=16
#pragma HLS array_partition variable=M dim=2 cyclic factor=16

out_loop:
  for (int i = 0; i < N; i++) {
    out_t sum = 0;
dot_product_loop:
    for (int j = 0; j < N; j+=16) {
#pragma HLS pipeline II=1
      for (int jj = 0; jj < 16; jj++)
#pragma HLS unroll
        sum += V_In[j+jj] * M[i][j+jj];
    }
    V_Out[i] = sum;
  }
}
```

***Try this optimized design.***



| Modules & Loops | Issue Type | Slack | Latency(cycles) | Latency(ns) | Iteration Latency | Interval | Trip Count | Pipelined | BRAM | DSP | FF | LUT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mvm | | - | 1222 | 1.222E4 | - | 1223 | - | no | 66 | 8 | 18479 | 24331 |
| out_loop | | - | 1216 | 1.216E4 | 19 | - | 64 | no | - | - | - | - |
| dot_product_loop | | - | 9 | 90.000 | 7 | 1 | 4 | yes | - | - | - | - |

# Design Analysis (1)

```
void mvm(in_t M[N][N], in_t V_In[N], out_t V_Out[N]) {
#pragma HLS interface m_axi port=M
#pragma HLS interface m_axi port=V_In
#pragma HLS interface m_axi port=V_Out

#pragma HLS array_partition variable=V_In cyclic factor=16
#pragma HLS array_partition variable=M dim=2 cyclic factor=16

out_loop:
  for (int i = 0; i < N; i++) {
    out_t sum = 0;
dot_product_loop:
    for (int j = 0; j < N; j+=16) {
#pragma HLS pipeline II=1
      for (int jj = 0; jj < 16; jj++)
#pragma HLS unroll
        sum += V_In[j+jj] * M[i][j+jj];
    }
    V_Out[i] = sum;
  }
}
```

Why array_partition with the configuration "*cyclic factor=16*"?

To provides enough ports (16) for the unrolled innermost loop. Finishing reads in one cycle allows *II=1*.

# Design Analysis (2)

```
void mvm(in_t M[N][N], in_t V_In[N], out_t V_Out[N]) {
#pragma HLS interface m_axi port=M
#pragma HLS interface m_axi port=V_In
#pragma HLS interface m_axi port=V_Out

#pragma HLS array_partition variable=V_In cyclic factor=16
#pragma HLS array_partition variable=M dim=2 cyclic factor=16

out_loop:
  for (int i = 0; i < N; i++) {
    out_t sum = 0;
dot_product_loop:
    for (int j = 0; j < N; j+=16) {
#pragma HLS pipeline II=1
      for (int jj = 0; jj < 16; jj++)
#pragma HLS unroll
        sum += V_In[j+jj] * M[i][j+jj];
    }
    V_Out[i] = sum;
  }
}
```

Why split dot_product_loop into two loops (j and jj)? (***Tiling!***)

To pipeline the partially unrolled loop. (Equivalent to pipeline + partial unroll.)

*Tiling looks more explicit.*

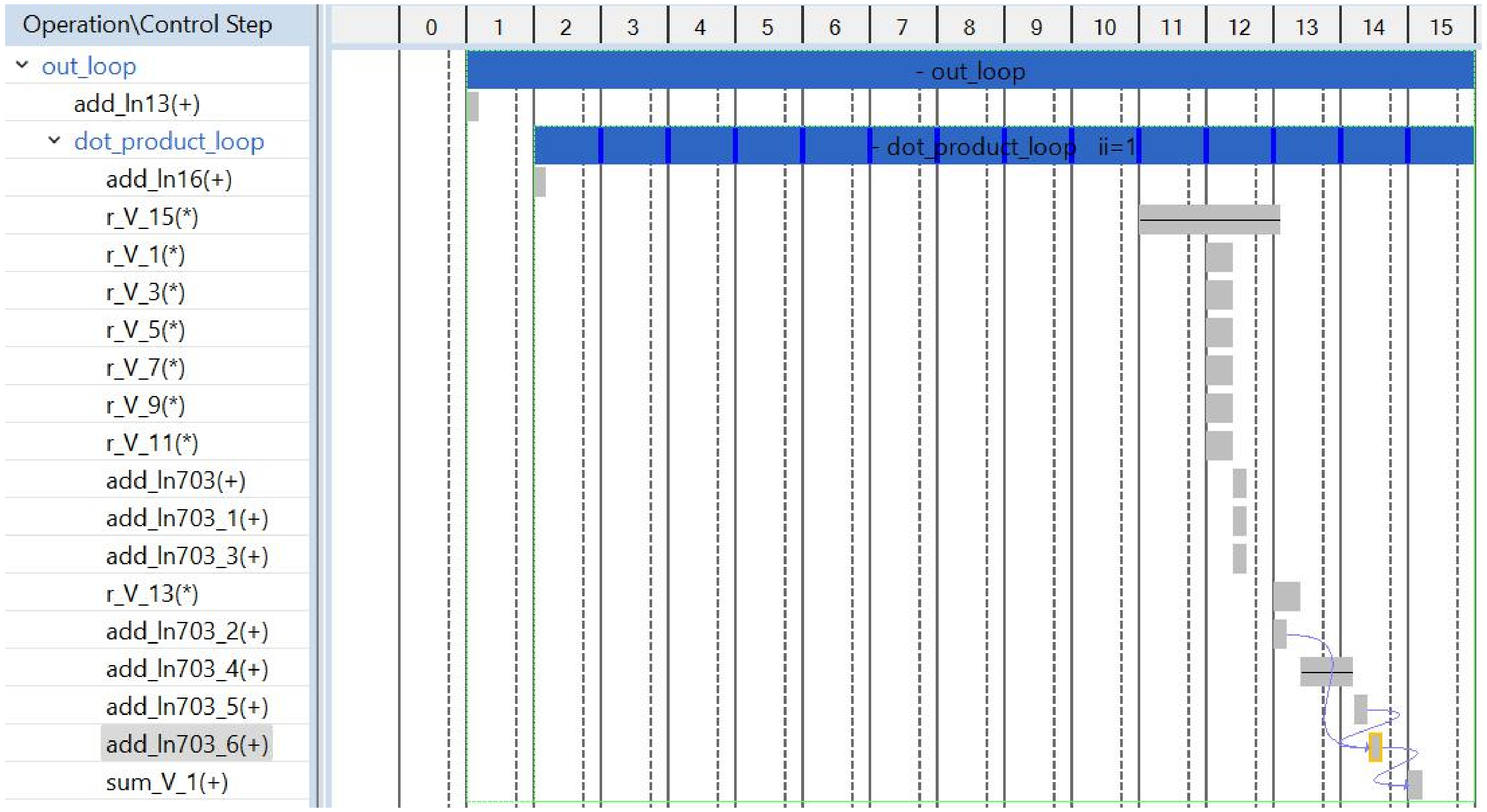
# Design Analysis (3)

```
void mvm(in_t M[N][N], in_t V_In[N], out_t V_Out[N]) {
#pragma HLS interface m_axi port=M
#pragma HLS interface m_axi port=V_In
#pragma HLS interface m_axi port=V_Out

#pragma HLS array_partition variable=V_In cyclic factor=16
#pragma HLS array_partition variable=M dim=2 cyclic factor=16

out_loop:
  for (int i = 0; i < N; i++) {
    out_t sum = 0;
dot_product_loop:
    for (int j = 0; j < N; j+=16) {
#pragma HLS pipeline II=1
      for (int jj = 0; jj < 16; jj++)
#pragma HLS unroll
        sum += V_In[j+jj] * M[i][j+jj];
    }
    V_Out[i] = sum;
  }
}
```

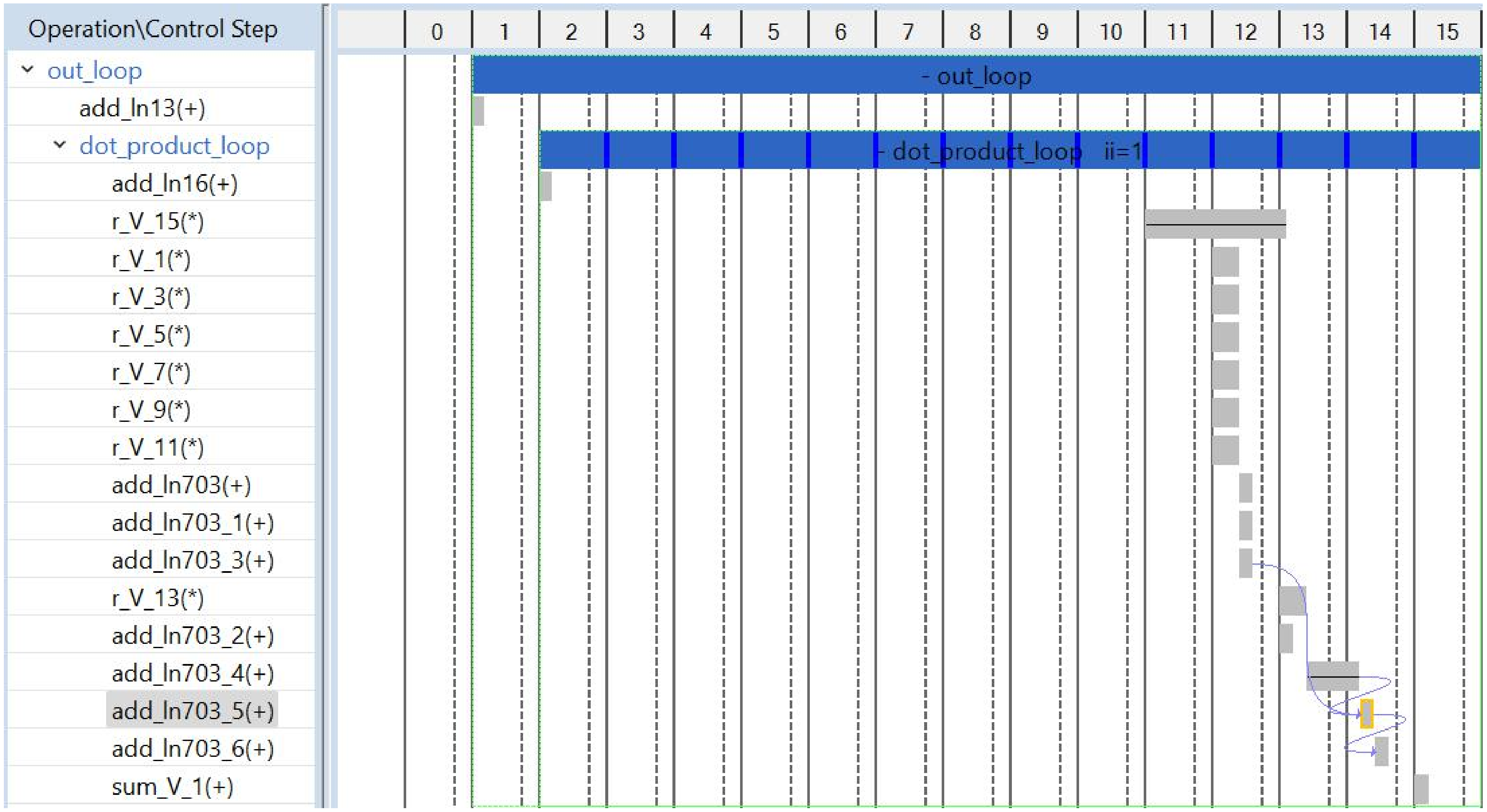
How does the produced hardware implement 16 additions in the innermost loop?

Look at the *Schedule Viewer*.

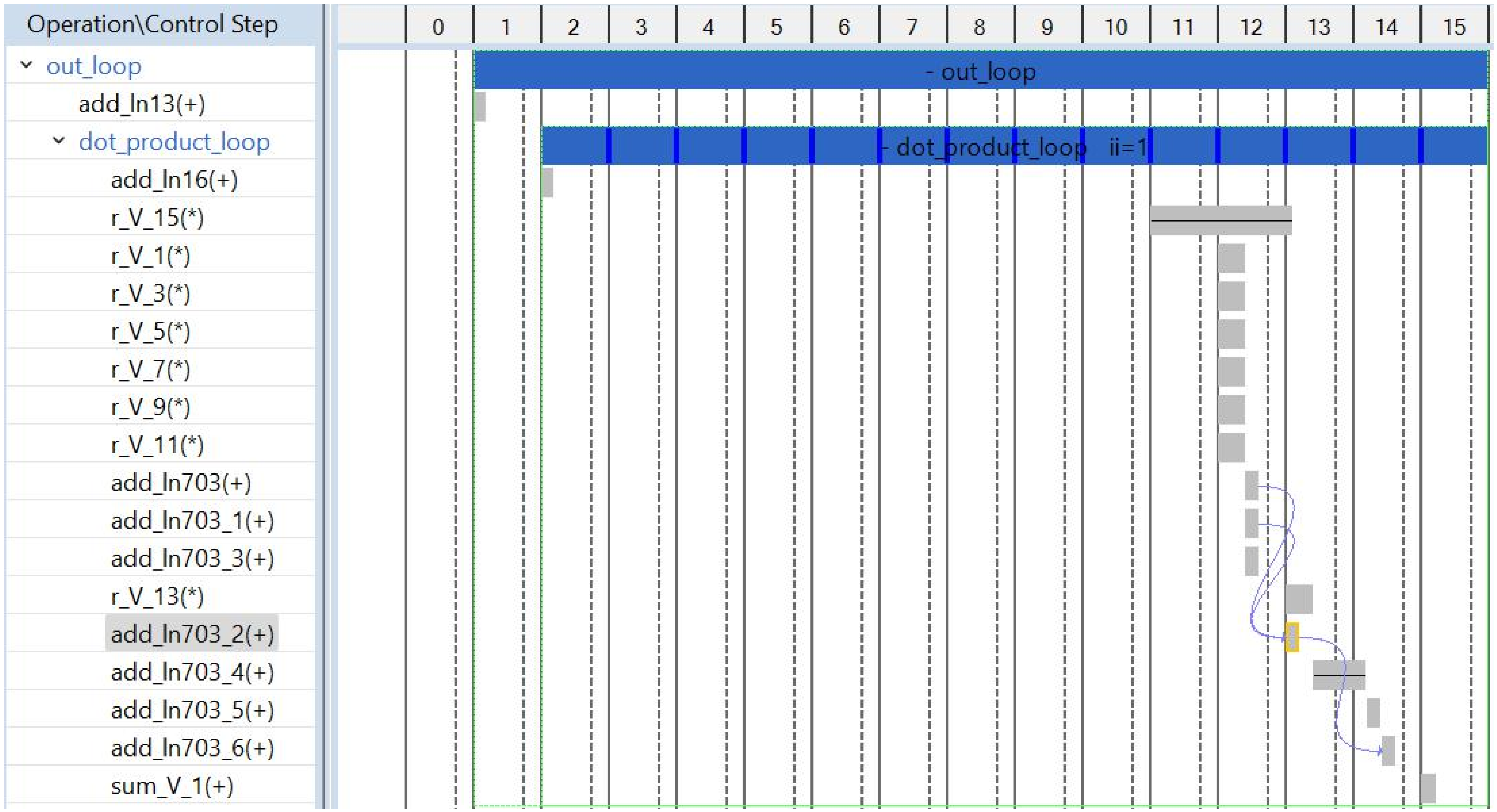
# Design Analysis (4)



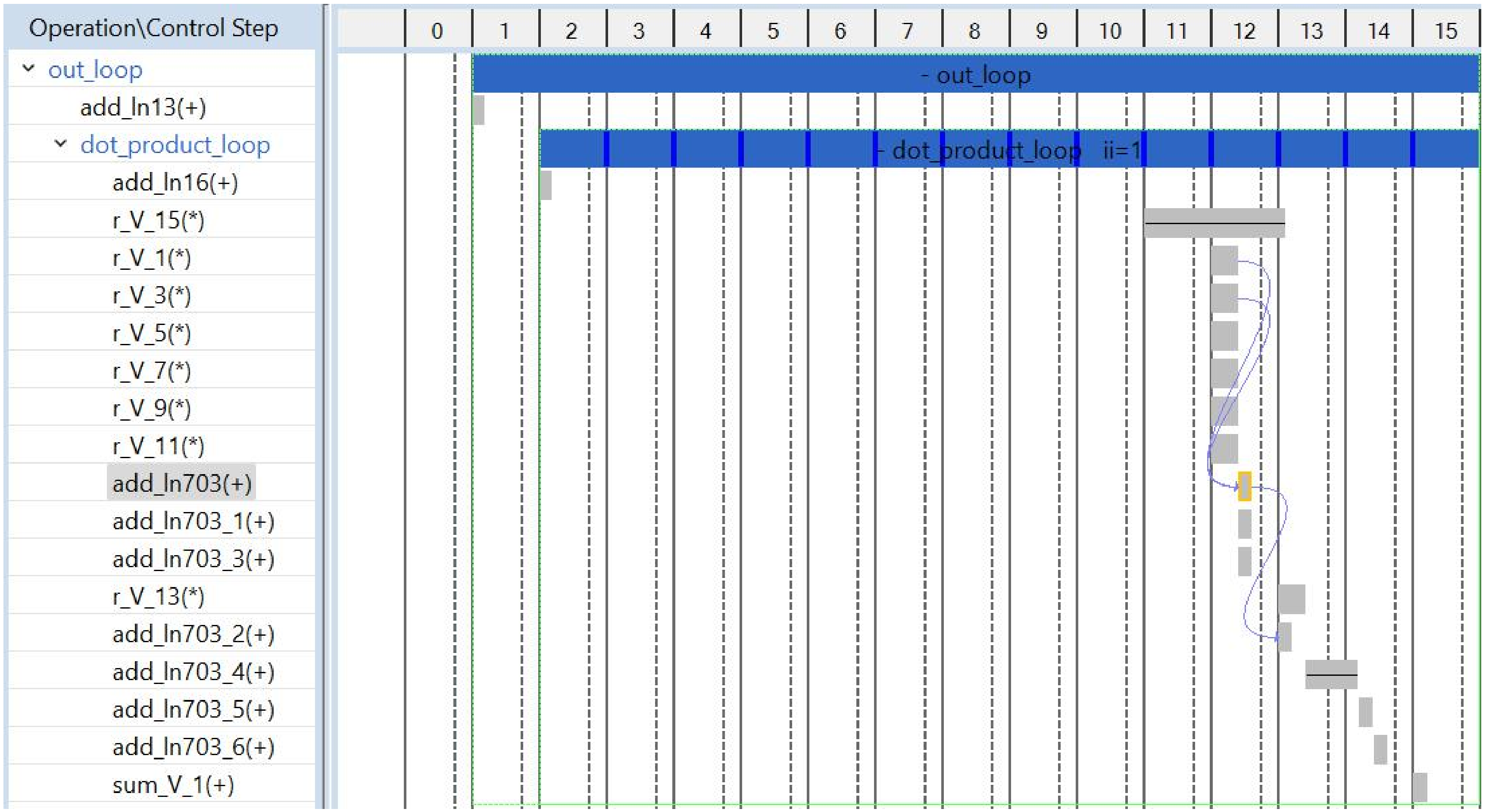
# Design Analysis (4)



# Design Analysis (4)



# Design Analysis (4)



Operation\Control Step

- out_loop
  - add_ln13(+)
  - dot_product_loop
    - add_ln16(+)
    - r_V_15(*)
    - r_V_1(*)
    - r_V_3(*)
    - r_V_5(*)
    - r_V_7(*)
    - r_V_9(*)
    - r_V_11(*)
    - add_ln703(+)
    - add_ln703_1(+)
    - add_ln703_3(+)
    - r_V_13(*)
    - add_ln703_2(+)
    - add_ln703_4(+)
    - add_ln703_5(+)
    - add_ln703_6(+)
    - sum_V_1(+)

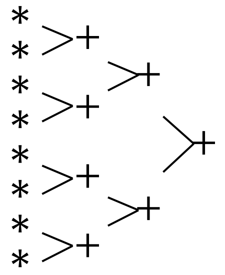
# Design Analysis (5)

```
void mvm(in_t M[N][N], in_t V_In[N], out_t V_Out[N]) {
#pragma HLS interface m_axi port=M
#pragma HLS interface m_axi port=V_In
#pragma HLS interface m_axi port=V_Out

#pragma HLS array_partition variable=V_In cyclic factor=16
#pragma HLS array_partition variable=M dim=2 cyclic factor=16

out_loop:
  for (int i = 0; i < N; i++) {
    out_t sum = 0;
dot_product_loop:
    for (int j = 0; j < N; j+=16) {
#pragma HLS pipeline II=1
      for (int jj = 0; jj < 16; jj++)
#pragma HLS unroll
        sum += V_In[j+jj] * M[i][j+jj];
    }
    V_Out[i] = sum;
  }
}
```

How does the produced hardware implement 16 additions in the innermost loop?

Look at the *Schedule Viewer*.

Reduction tree



Why?

- better latency.
- adder is cheap.