ADIUVO
ENGINEERING AND TRAINING, LTD.

All About HLS

Adam@AdiuvoEngineering.com

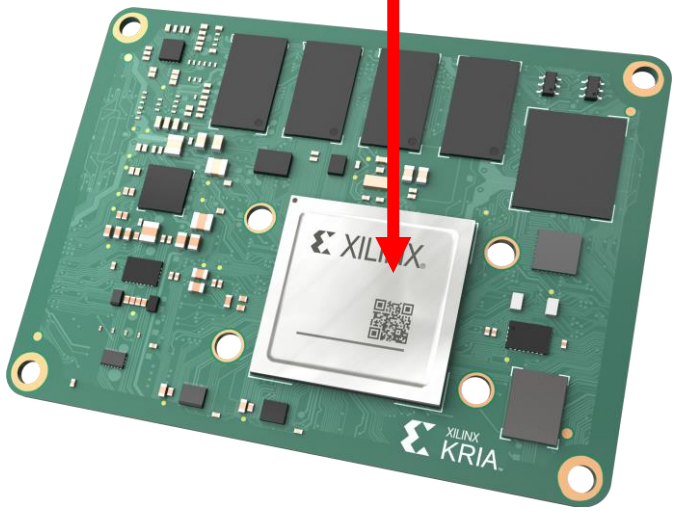
What is HLS

High Level Synthesis (HLS) enables generation of RTL modules from higher level language such as C, C++, OpenCL

Of course, SW engineers still consider these low-level languages.

HLS offers several benefits for the development of Signal / Data / Image processing algorithms

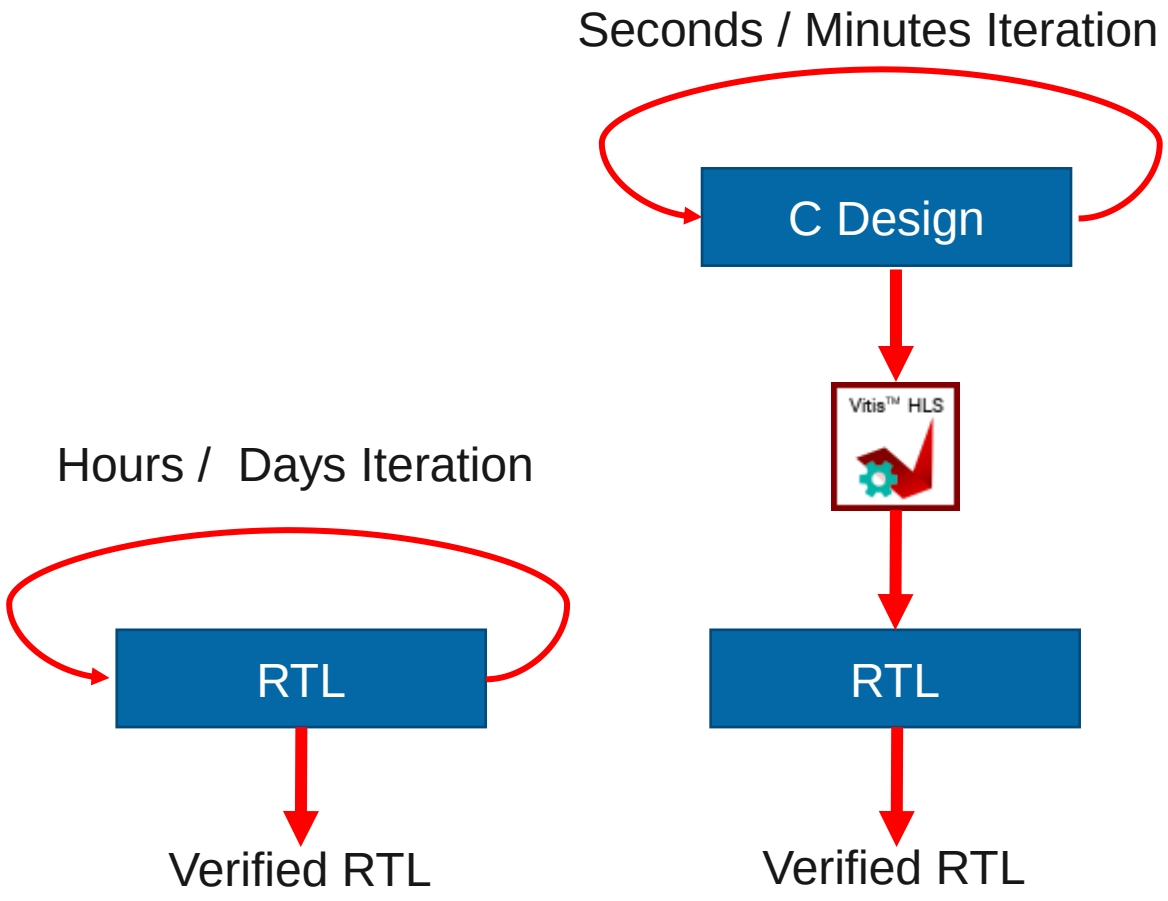
```
    error = set_point - sample;

    p = error * KP;
    i = i_prev + (error  * ts * KI);
    d = KD * ((error - error_prev) / ts);

    op = p+i+d;
    error_prev = error;
    if (op > pmax)  {
        i_prev = i_prev;
        op = pmax;
    }else{
        i_prev = i;
    }
    return op;
}
```



HLS Benefits

Developing in Higher Level language enables a faster iteration time.

- Development Time decreased as untimed language – No RTL / Behavioral level
- Increased level of abstraction – accelerates development
- Verification time is reduced as untimed Simulation



Creating HLS Solutions

Software written for CPUs and software written for FPGAs is fundamentally different

1. Not all C constructs can be synthesised
2. Learn about synthesizable C/C++ coding styles.
3. Need to focus on correct micro architecture
    1. Understand the producers and consumers
    2. Decompose the algorithm into small section which interconnect
    3. Understand throughput required for each element to achieve overall performance goals
4. Learn how to interpret the design reports

HLS Flow

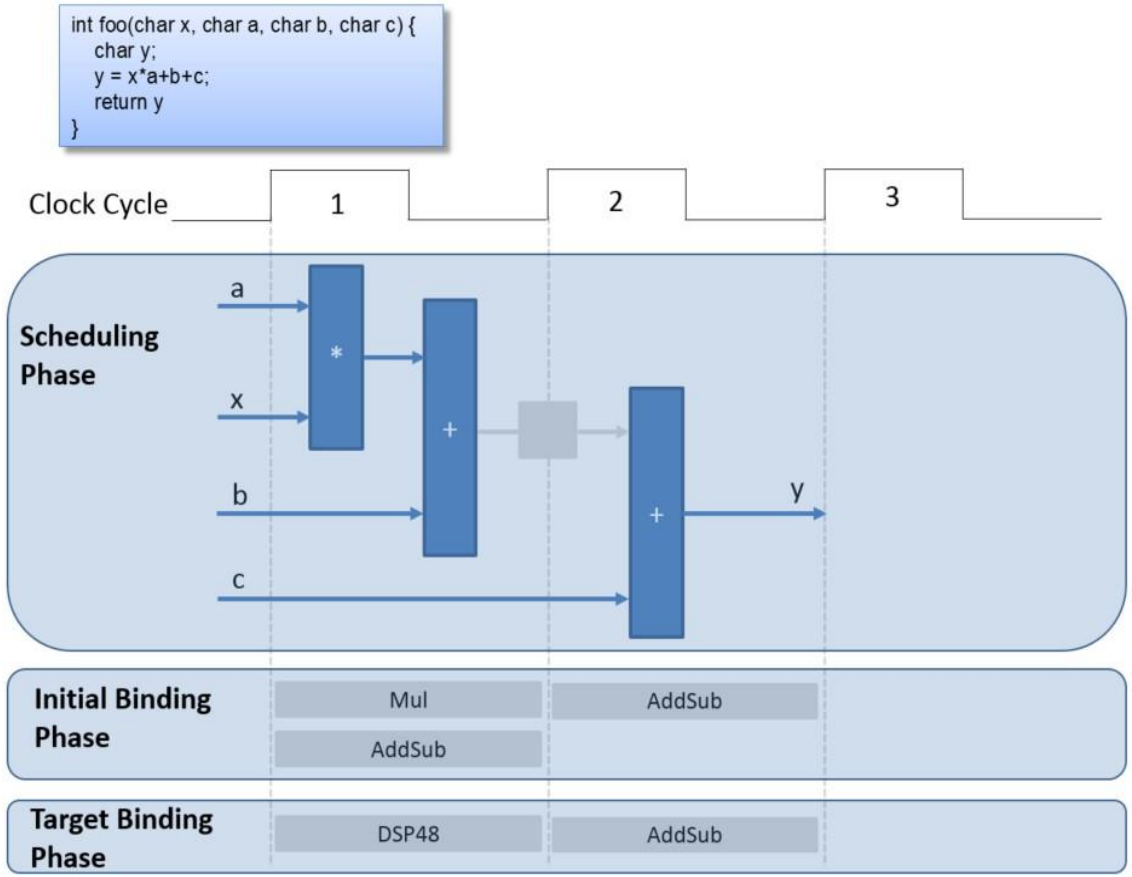
Create Solution, Simulate & Debug	Create the C Module and Test Bench – Leverage Libraries where possible Verify algorithm performance in SW
Synthesize	Convert C Module to RTL
Analyse & Optimize	Optimize Interfaces and Performance of synthesized implementation
Co Simulation	Simulation of the RTL module against the C Test Bench
Package the IP	Package for use in Vitis (XO) or Vivado (Xact IP)

Untimed to Timed

Scheduling
- Determines which operations occur during each clock cycle

Binding
- Determines which hardware resource implements each scheduled operation

Control logic extraction
- Extracts the control logic to create a finite state machine (FSM) that sequences the operations in the RTL design

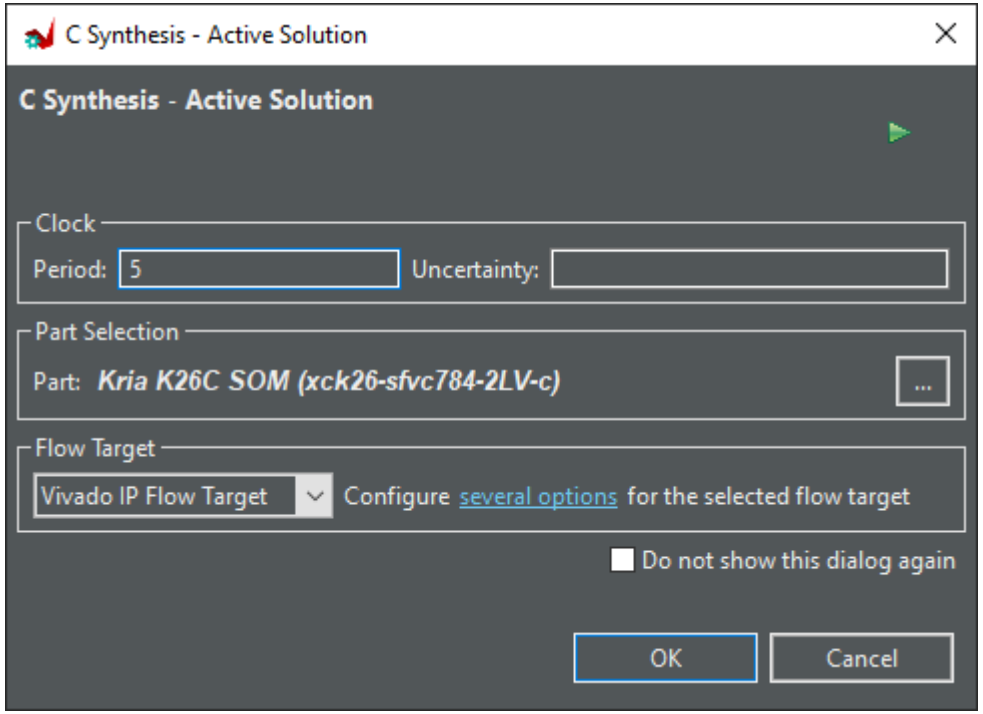


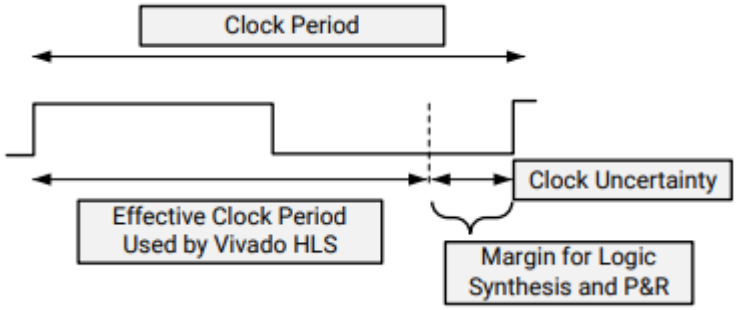
Clock Definition

Clock frequency & selected device used to determine timing of operations – Scheduling Phase

Uncertainty leaves time for final component placement and net routing

If left blank 27% is used





Interfacing

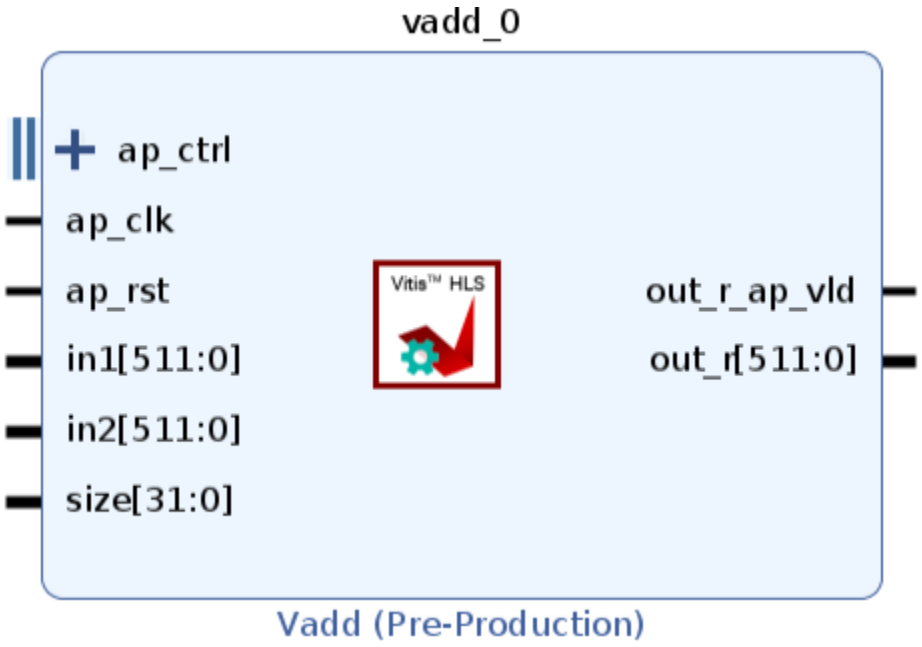
Interfacing depends on flow

Vitis - XO

- AXI – Data Accessed via Memory Map
- AXIS – Data Accessed via Stream
- AXI Lite – Register Access

Vivado – IP XACT

- Block Level – Controls Flow / Status
- Protocol Level – Controls



Vivado Interfacing

Instantiate appropriate interface to integrate with design

Do we want to start and stop block, synchronize or be data driven.

Are we controlling the block from SW

What about data interfaces.

Paradigm	Description	s
Memory	Data is accessed by the kernel through memory such as DDR, HBM, PLRAM/BRAM/ URAMSupported Interface Protocol	`ap_memory`, BRAM, AXI4 Memory Mapped (`m_axi`)
Stream	Supported InterfaceData is streamed into the kernel from another streaming source, such as video processor or another kernel, and can also be streamed out of the kernel.	`ap_fifo`, AXI4-Stream (`axis`)
Register	Data is accessed by the kernel through register interfaces performed by register reads and writes.	`ap_none`, `ap_hs`, `ap_ack`, `ap_ovld`, `ap_vld`, and AXI4-Lite adapter (`s_axilite`).

C-Argument Type	Supported Paradigms	Default Paradigm	Default Interface Protocol		
			Input	Output	Inout
Scalar variable (pass by value)	Register	Register	ap_none	N/A	N/A
Array	Memory, Stream	Memory	ap_memory	ap_memory	ap_memory
Pointer	Memory, Stream, Register	Register	ap_none	ap_vld	ap_ovld
Reference	Register	Register	ap_none	ap_vld	ap_vld
`hls::stream`	Stream	Stream	ap_fifo	ap_fifo	N/A

Interfacing Example

Synthesis Summary(solution1) | Schedule Viewer(solution1) | spi_main.cpp

```
64  int offset;
65  int x;
66  void spi_loop (data_type *din, data_type_qspi *dout, volatile ap_int<160> *device_id,  data_type *tx_fifo )
67  {
68      if (init == 0){ // configure the SPI driver
69          //init the QSP
70          //manual slave select, master mode, spi enable
71          WriteBuffer[0] = 0x86;
72          memcpy( dout + (AXI_SPI_CNTRL/4), WriteBuffer, ( sizeof(data_type)) );
73          init = 1;
74      }
75
76      WriteBuffer[0] = SF3_COMMAND_READ_ID;
77      slave_select[0] = 0;
78      memcpy (dout + (AXI_SPI_SSR/4), slave_select, sizeof(data_type_qspi) ); //assert SS
79      ap_wait_n(2);
80      memcpy( dout + (AXI_SPI_DTR/4), (data_type_qspi *) WriteBuffer, (20* sizeof(data_type)) );
81      slave_select[0] = 1;
82      ap_wait_n(2);
83      memcpy (dout + (AXI_SPI_SSR/4), slave_select, sizeof(data_type_qspi) ); //deassert SS
84      ap_wait_n(100);
85
86      offset = 0;
87      for(int x = 0; x<2; x++)
88      {
89          //read bytes from the ddr image location
90          memcpy( ddr_buffer + SF3_WRITE_EXTRA_BYTES, din + offset,  (DDR_RD_SIZE* sizeof(uint8_t)));
91          offset = offset + DDR_RD_SIZE;
92          //set up the command and address
93          ddr_buffer[0] = SF3_COMMAND_PAGE_PROGRAM;
94          ddr_buffer[1] = 0x55;
95          ddr_buffer[2] = 0xaa;
96          ddr_buffer[3] = 0x55;
```

Outline | Directive

- WriteBuffer
- ReadBuffer
- ddr_buffer
- slave_select
- init
- offset
- x
- spi_loop
  - HLS TOP name=spi_loop
  - din
  - HLS INTERFACE m_axi depth=1024 port=din offset=off
  - dout
  - HLS INTERFACE m_axi depth=1024 port=dout offset=off
  - device_id
  - tx_fifo
  - HLS INTERFACE ap_fifo port=tx_fifo
  - for Statement
    - for Statement
    - for Statement

Directives used to control interface types and optimization pragmas

C Simulation

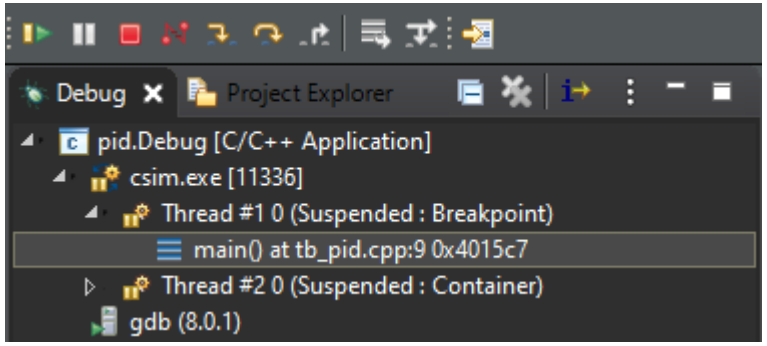
Synthesis Summary(solution1) | Schedule Viewer(solution1) | PID_csim.log | tb_pid.cpp

```c
#include "pid.h"

#include <stdio.h>
#define iterations 40

int main(void)
{

data_type set_point = -80.0;
data_type sample[iterations] = {-90.000,-88.988,-87.977,-86.966,-85.955,-84.946,-83.936,-82.928,-81.920,-80.912,-80.283,-79.926,
        -79.784,-79.774,-79.829,-79.898,-79.955,-79.993,-80.011,-80.017,-80.016,-80.010,-80.005,-80.002,-80.000,-79.999,
        -79.999,-79.999,-79.999,-80.000,-80.000,-80.000,-80.000,-80.000,-80.000,-80.000,-79.999,-80.000,-80.001,-80.000};



data_type kp = 19.6827; // w/k
data_type ki = 0.7420; // w/k/s
data_type kd = 0.0;
data_type op;

printf("testing cpp\r\n");


for (int i =0; i<iterations; i++){
    op = PID (set_point, kp, ki, kd, sample[i], 12.5, 40);
    printf("result %f\r\n",op);
}
return 0;
}
```

Debug View

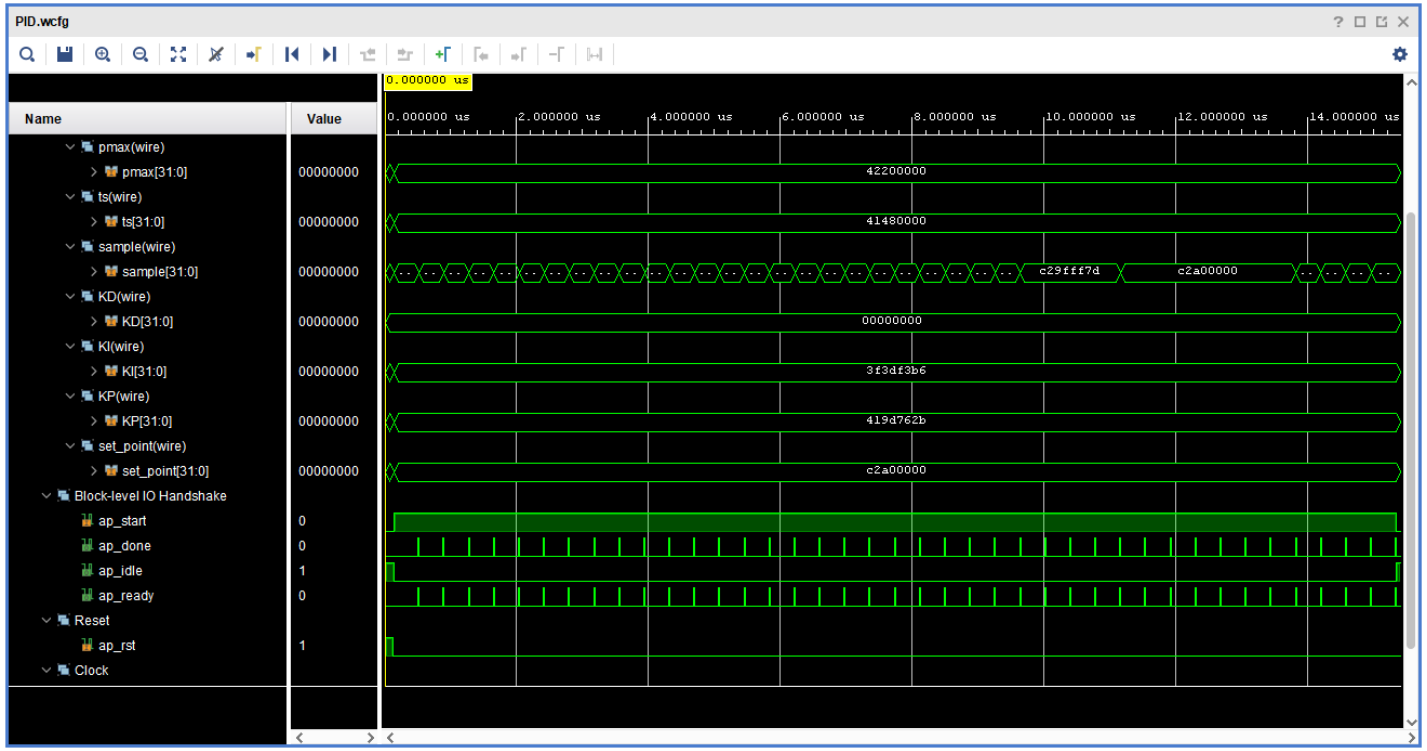
All features of a normal debugger

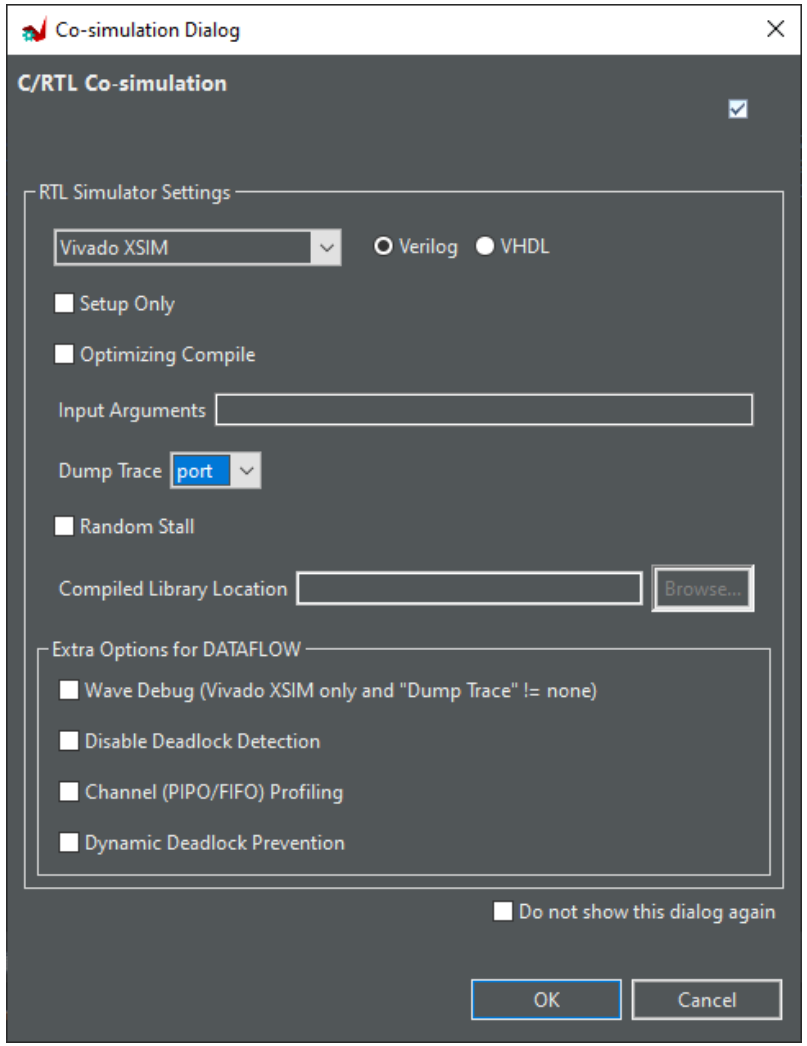
- Single Step
- Break point,
- Observe registers & Values



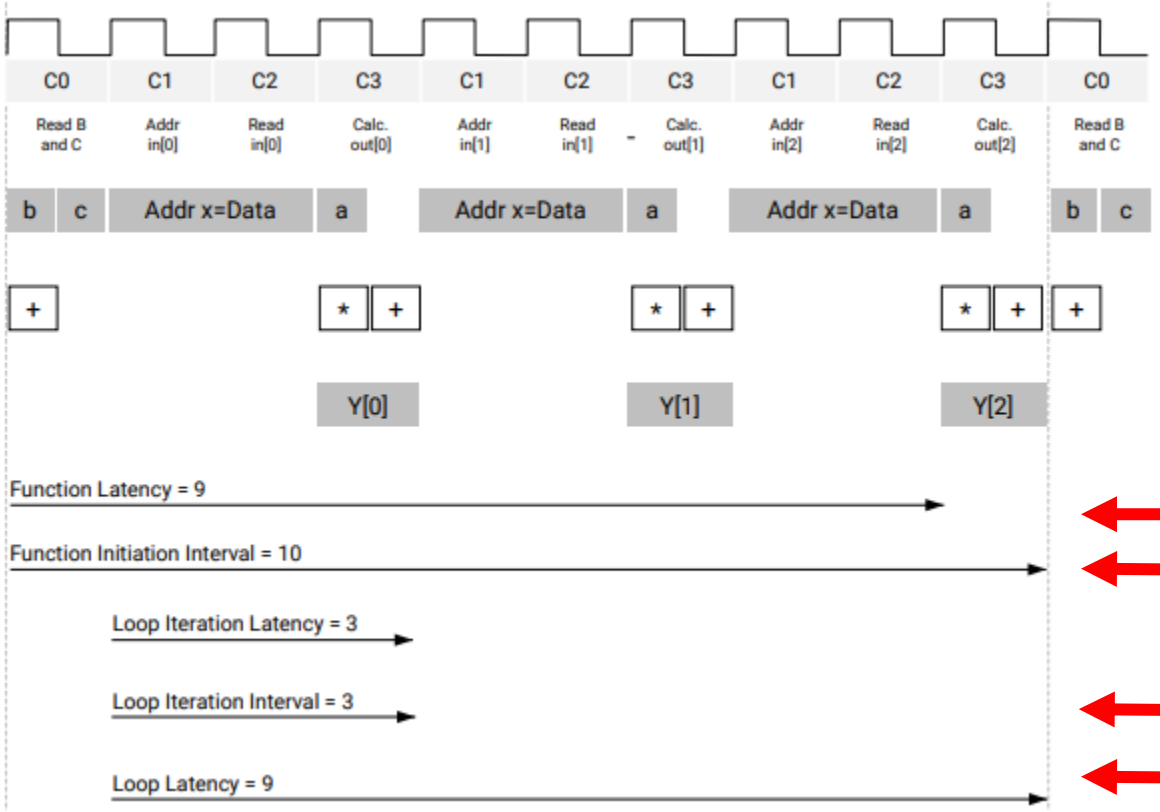
Co Simulation

Applies C test bench to the Generated RTL – Simulated using Xsim or preferred simulator





Performance Metrics



Used to define the optimization we might want to apply to the design

9 clocks to function latency

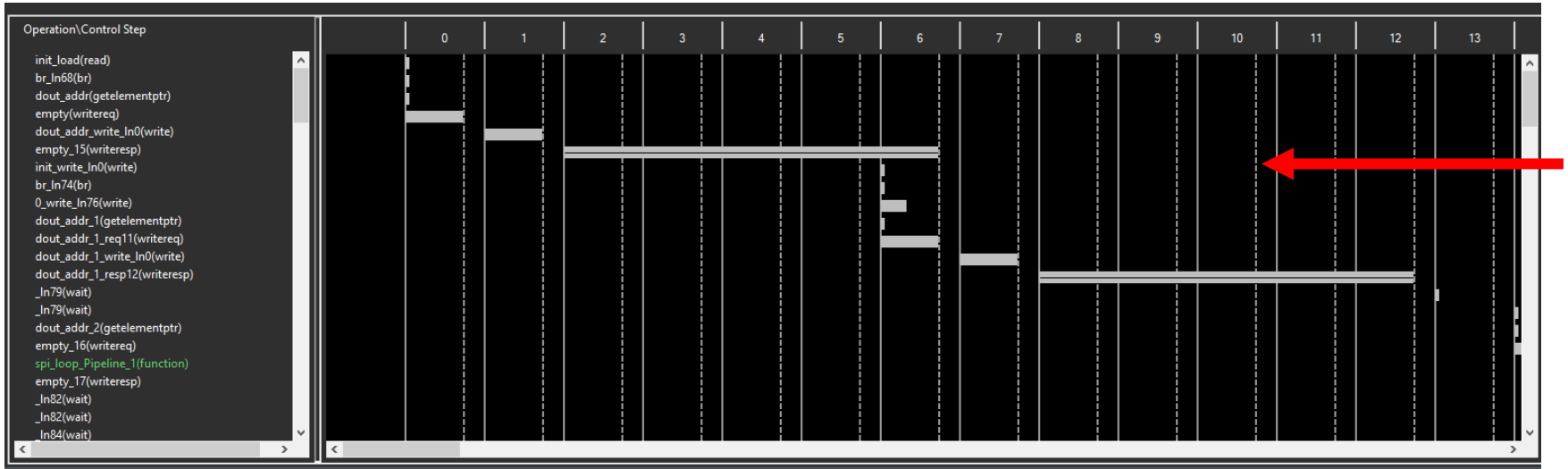
10 clocks between function initiation
Function Initiation Interval

3 Clocks latency per loop

3 Clocks Loop Initiation Interval

9 clocks to loop latency

Analysis View



Implementation information showing operations per clock cycle – Can cross probe to source

Console | Errors | Warnings | Guidance | Properties | Man Pages | Git Repositories | Modules/Loops

Synthesis | Cosimulation

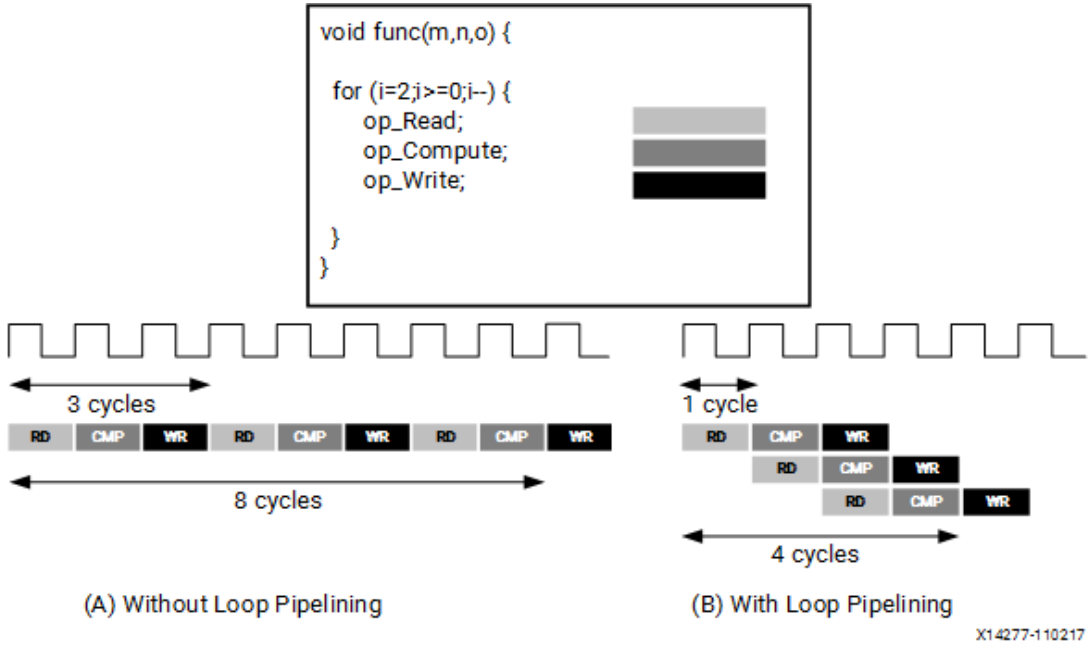
Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
spi_loop				-	947	9.470E3	-	948	-	no	8	0	1411	3620	0
spi_loop_Pipeline_1				-	8	80.000	-	8	-	no	0	0	17	68	0
Loop 1				-	6	60.000	3	1	5	yes	-	-	-	-	-
spi_loop_Pipeline_2				-	259	2.590E3	-	259	-	no	0	0	40	92	0
Loop 1				-	257	2.570E3	3	1	256	yes	-	-	-	-	-
spi_loop_Pipeline_VITIS_LOOP_101_5				-	262	2.620E3	-	262	-	no	0	0	21	65	0
VITIS_LOOP_101_5				-	260	2.600E3	2	1	260	yes	-	-	-	-	-
spi_loop_Pipeline_spi_loop_label0	II Violation			-	264	2.640E3	-	264	-	no	0	0	101	527	0
spi_loop_label0	II Violation	Memory Dependency	1	-	262	2.620E3	23	16	16	yes	-	-	-	-	-
spi_loop_Pipeline_5				-	7	70.000	-	7	-	no	0	0	17	69	0
Loop 1				-	5	50.000	3	1	4	yes	-	-	-	-	-

Implementation information – Loop Data

Kernel Optimization – Pipelining

By default, every iteration of a loop only starts when the previous iteration has finished

Pipelining the loop executes subsequent iterations in a pipelined manner

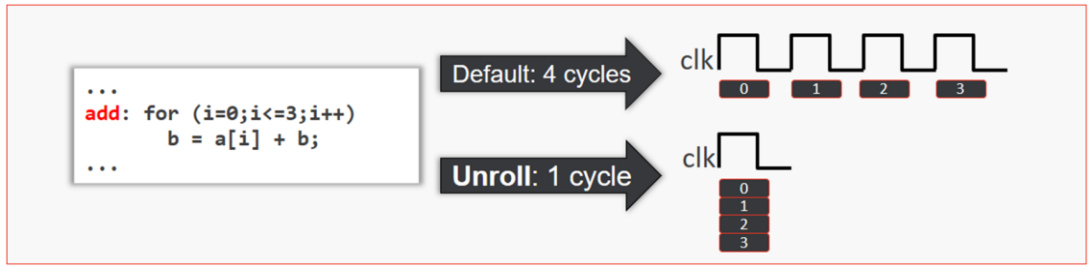


Kernel Optimization – Unrolling

Unrolling a loop enables the full parallelism

Full or Partial Unroll

Data dependencies in loops can impact the results of loop pipelining or unrolling



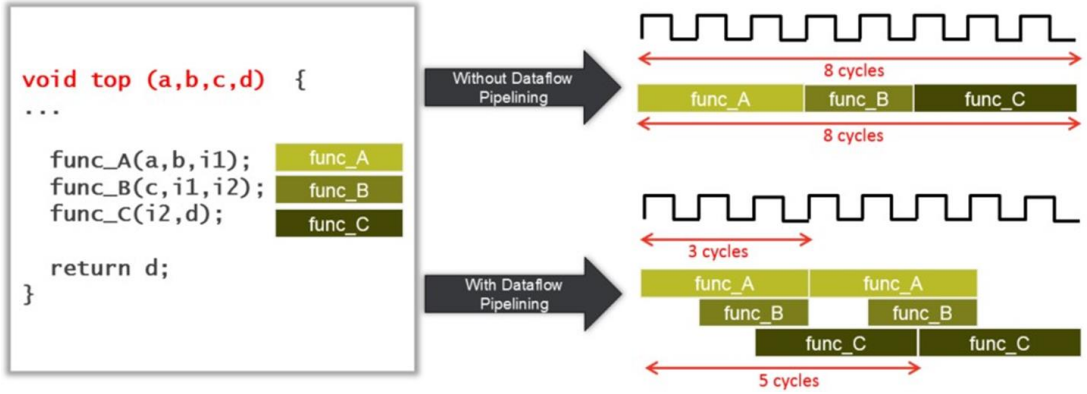
Kernel Optimization – DataFlow

Improve kernel performance by enabling task-level pipelining

Be careful of

Single producer-consumer violations.

- » Bypassing tasks.
- » Feedback between tasks.
- » Conditional execution of tasks.
- » Loops with multiple exit conditions or conditions defined within the loop

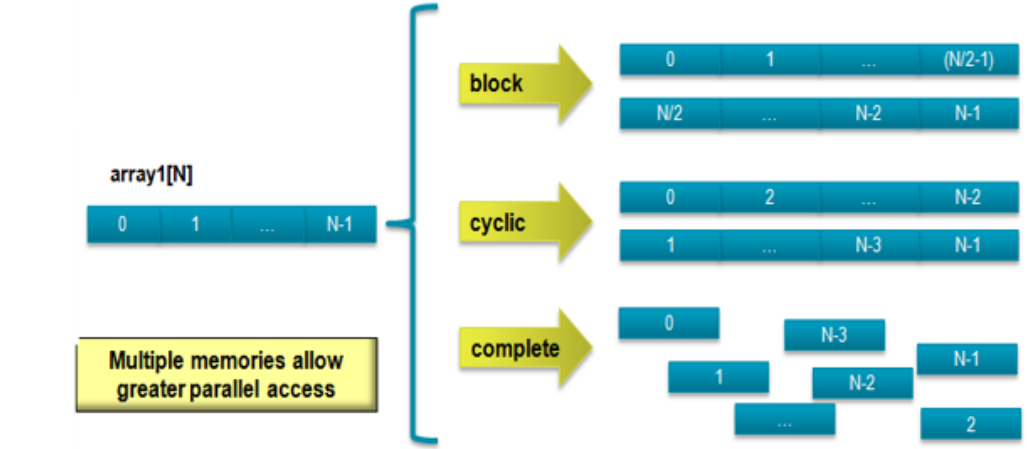


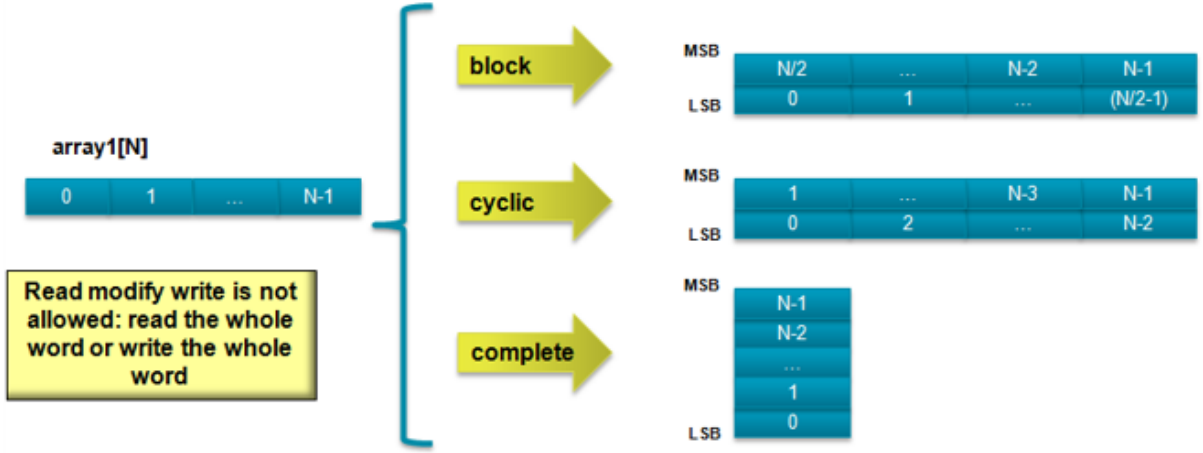
Kernel Optimization – Memory

Arrays – Performance bottlenecks as implemented in BRAM

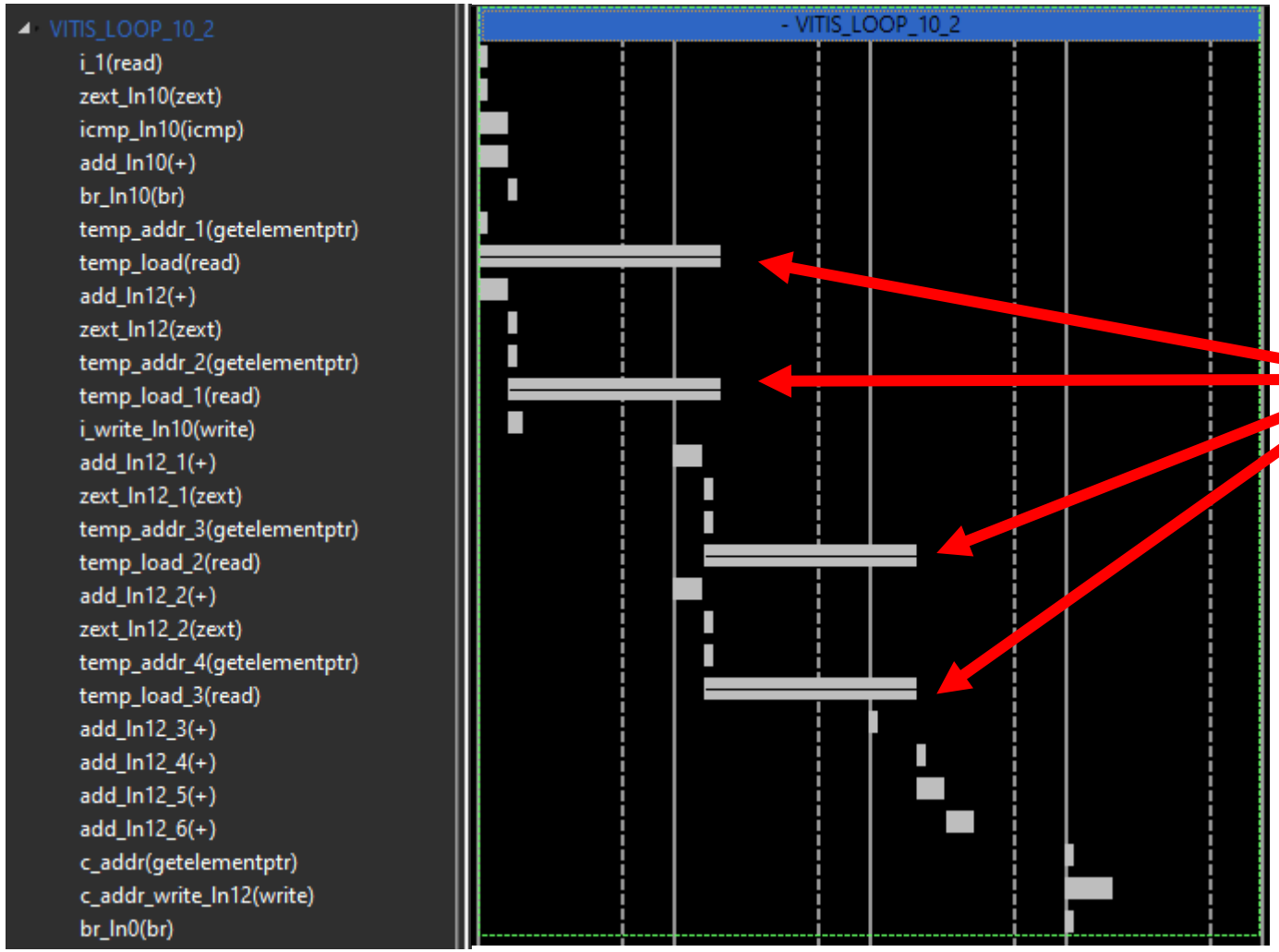
Limited BRAM access bandwidth, can heavily impact the overall performance

1. Partition – Separates into different BRAMS
2. Reshape – Allows combination of words



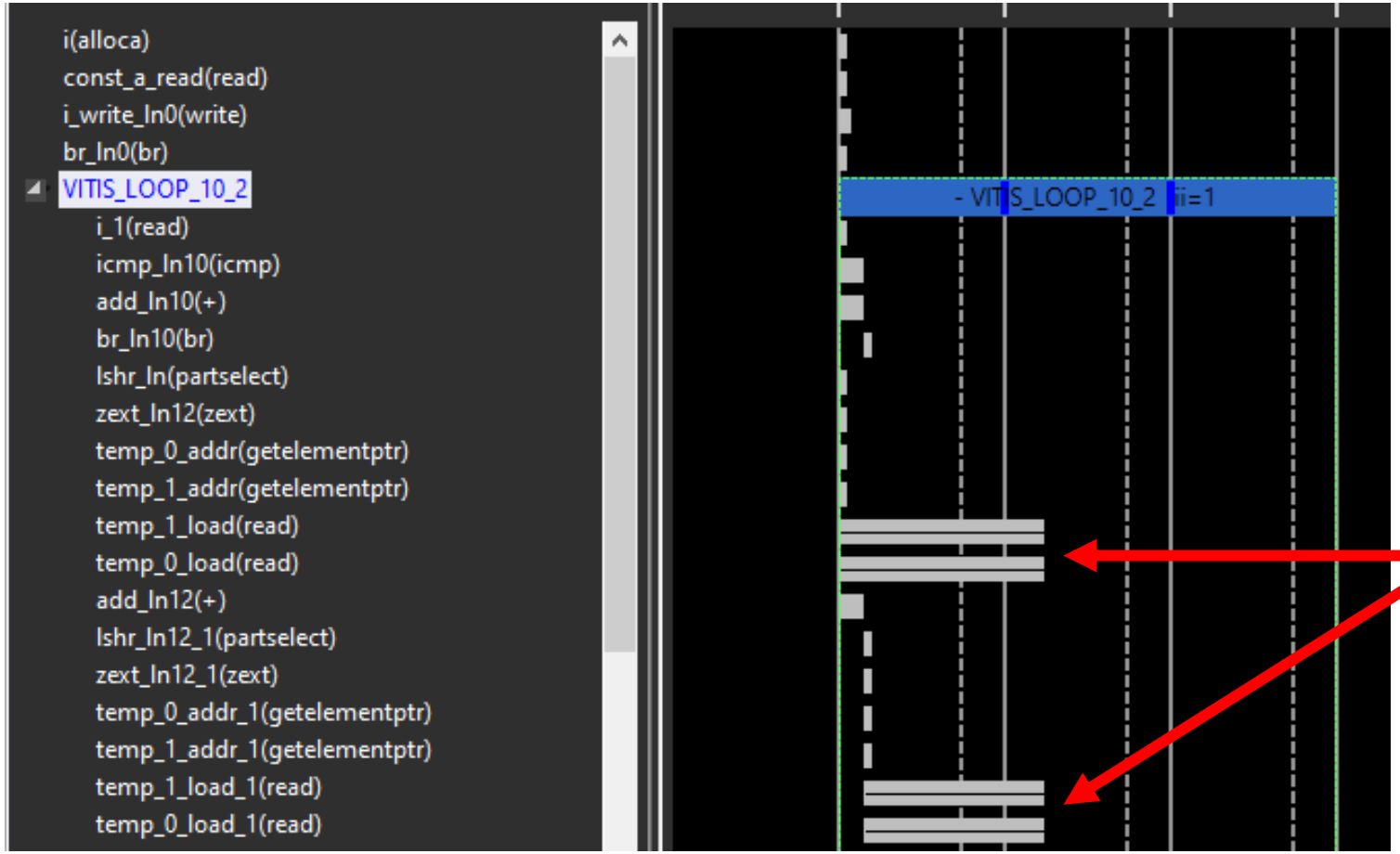


BRAM Optimization



Pre-Optimization – Sequential Access

BRAM Optimization



Post Optimization – Parallel Access

Kernel Optimization - Pragmas

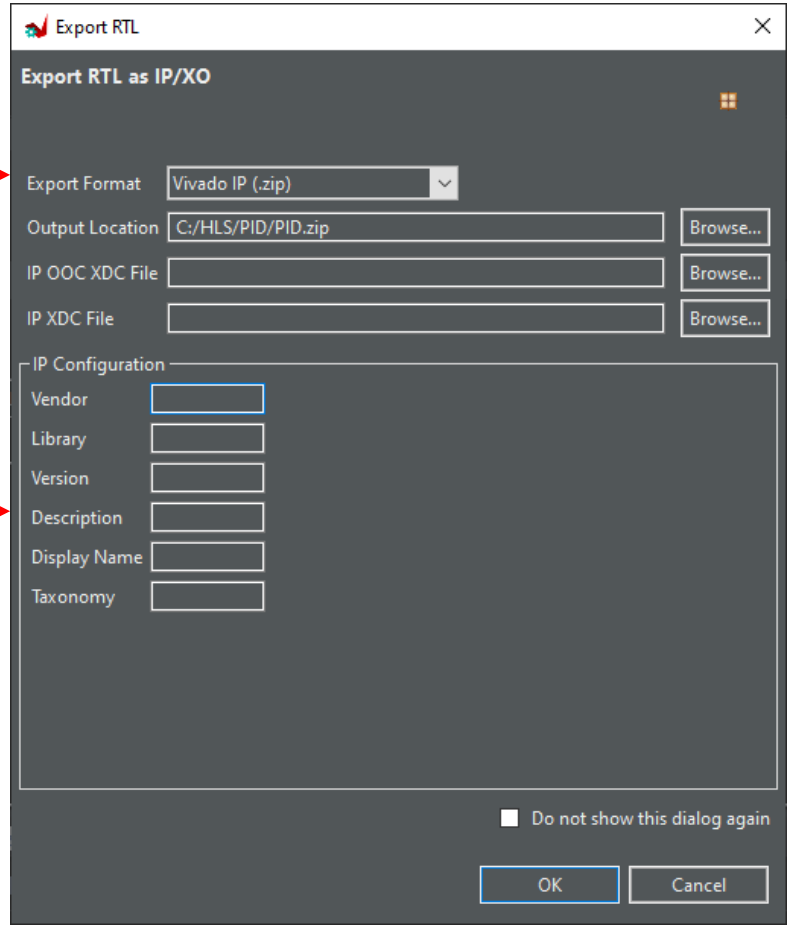
Optimization	C/C++
Pipeline	#pragma HLS PIPELINE
Unroll	#pragma HLS UNROLL
DataFlow	#pragma HLS DATAFLOW
Memory	#pragma HLS ARRAY_PARTITION

Further information can be found at
https://www.xilinx.com/html_docs/xilinx2020_1/vitis_doc/optimizingperformance.html#fhe1553474153030

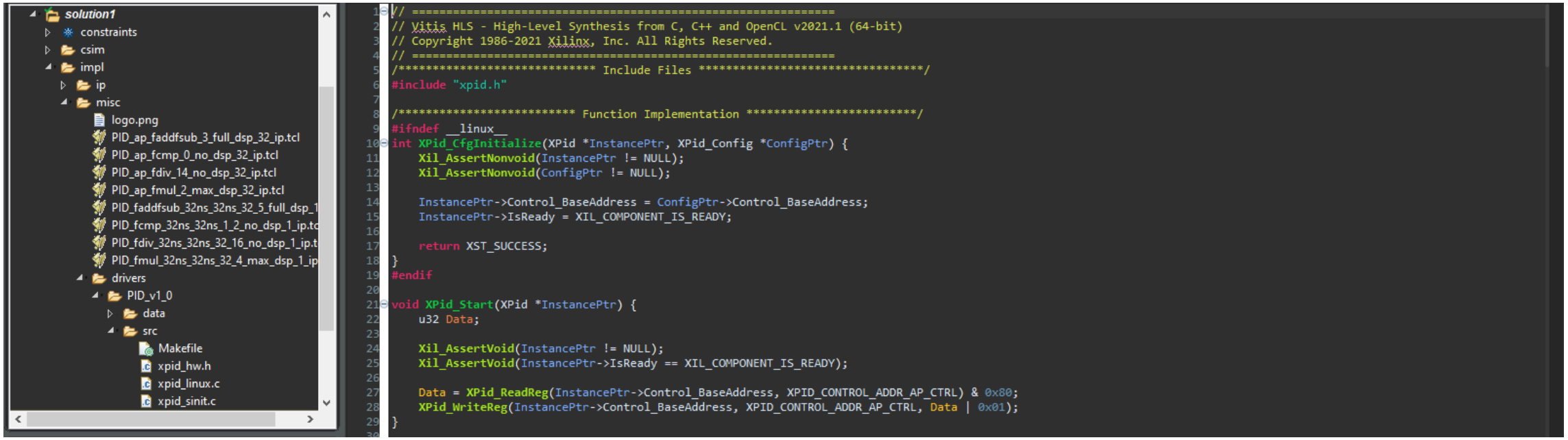
Exporting

Vitis or Vivado Flow

IP Information for Library and developing company



Software Development



All IP which is exported comes with a software drivers, if AXI interfaces are used.

This can be useful for controlling the IP if connected to Processing System or MicroBlaze etc.

Useful Concepts

Need	Structure	Comment
Wait for an input signal as a trigger.	ap_wait_until(SIGNAL)	Use ap_unit<1> to define signal input
Wait for a defined number of clock cycles	ap_wait_n(DELAY)	Delay in clock cycles
Generate an output trigger signal	volatile bool *TRIG	Needs to be defined as volatile to ensure immediate output

Example

```
#include "vitis_source.h"

void foo( ap_uint<32> volatile *reg,   op_type volatile *raw_line)
{
#pragma HLS interface ap_none port=raw_line
#pragma HLS interface s_axilite port=reg bundle=ctrl
#pragma HLS interface s_axilite port=return bundle=ctrl

        ap_uint<32> local_reg = *reg;


       if (local_reg.test(0)) {
                *raw_line = 1;
                local_reg.clear(0);
                ap_wait_n(50);

       }

        *reg = local_reg;
        *raw_line = 0;
        ap_wait_n(1);
}
```

Raw_line – Boolean volatile Ptr

Raw_line – Interface Type define ap_none

When LSB of Register set

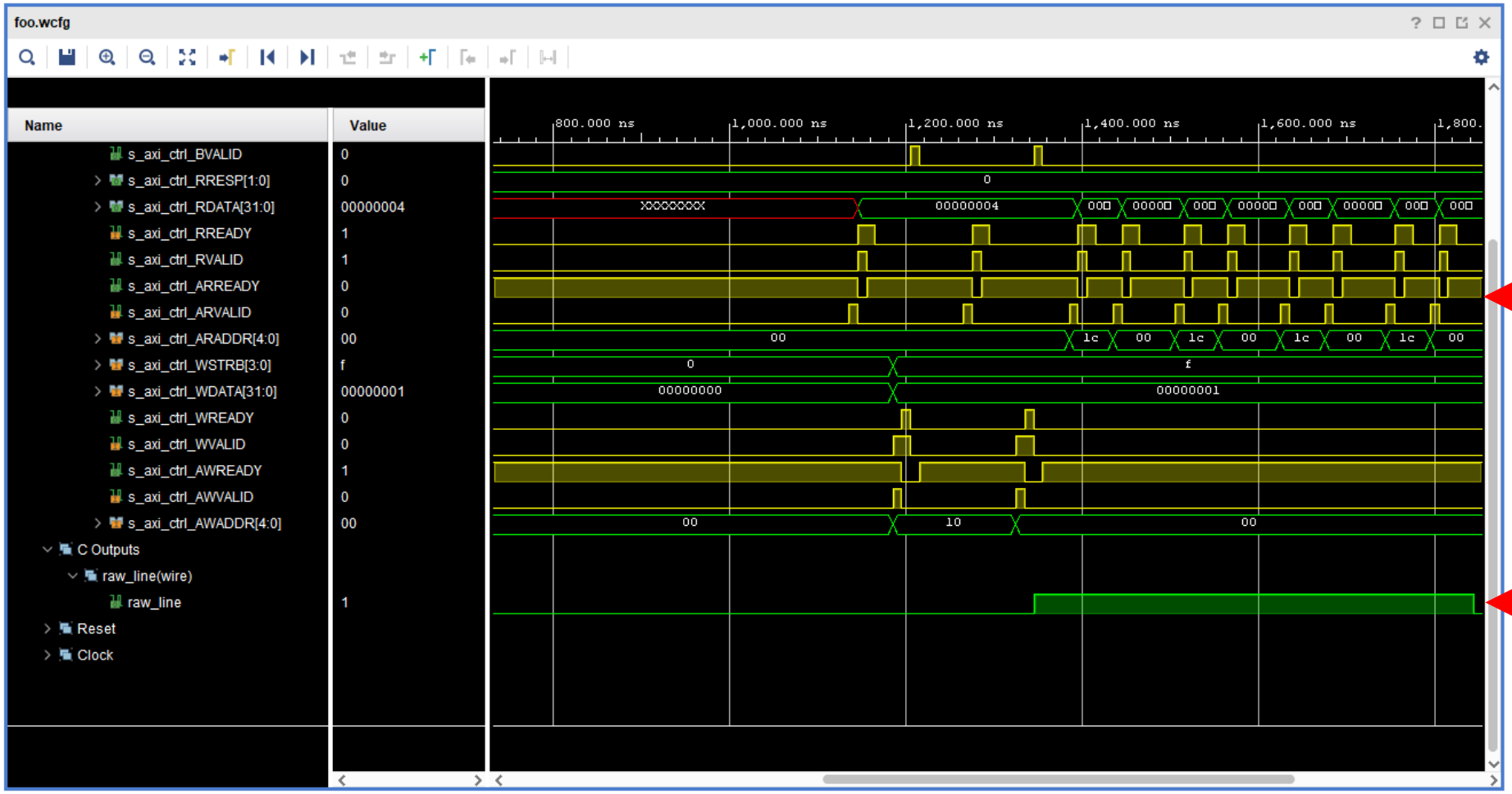
Set raw_line to 1

Wait for 50 clock cycles

Set raw_line to 0

Wait for 1 clock cycle

Co-Simulation



AXI Write to REG

50 CLK wide pulse

ADIUVO
ENGINEERING AND TRAINING, LTD.

HUD Example

PID Example

ADIUVO
ENGINEERING AND TRAINING, LTD.

Questions ?

Example code Available at

https://github.com/ATaylorCEngFIET/basys3_pong

https://github.com/ATaylorCEngFIET/MZ_402_PID_HLS

ADIUVO

ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com

adam@adiuvoengineering.com