Traineeships in Advanced Computing for High Energy Physics (TAC-HEP)

GPU & FPGA module training: Part-2

Week-5: Vivado HLS: More pragmas and HLS *coding styles*

Lecture-10: April 19th 2023

Varun Sharma

University of Wisconsin – Madison, USA

So Far…

- **FPGA and its architecture**
  - Registor/Flip-Flops, LUTs/Logic Cells, DSP, BRAMs
  - Clock Frequency, Latency
  - Extracting control logic & Implementing I/O ports
- **Parallelism in FPGA**
  - Scheduling, Pipelining, DataFlow
- **Vivado HLS**
  - Introduction, Setup, Hands-on for GUI/CLI, Introduction to Pragmas
  - Different Pragmas and their effects on performance

Today:
- More pragmas
- Some more Good practices to write HLS codes
  - Does & Don'ts

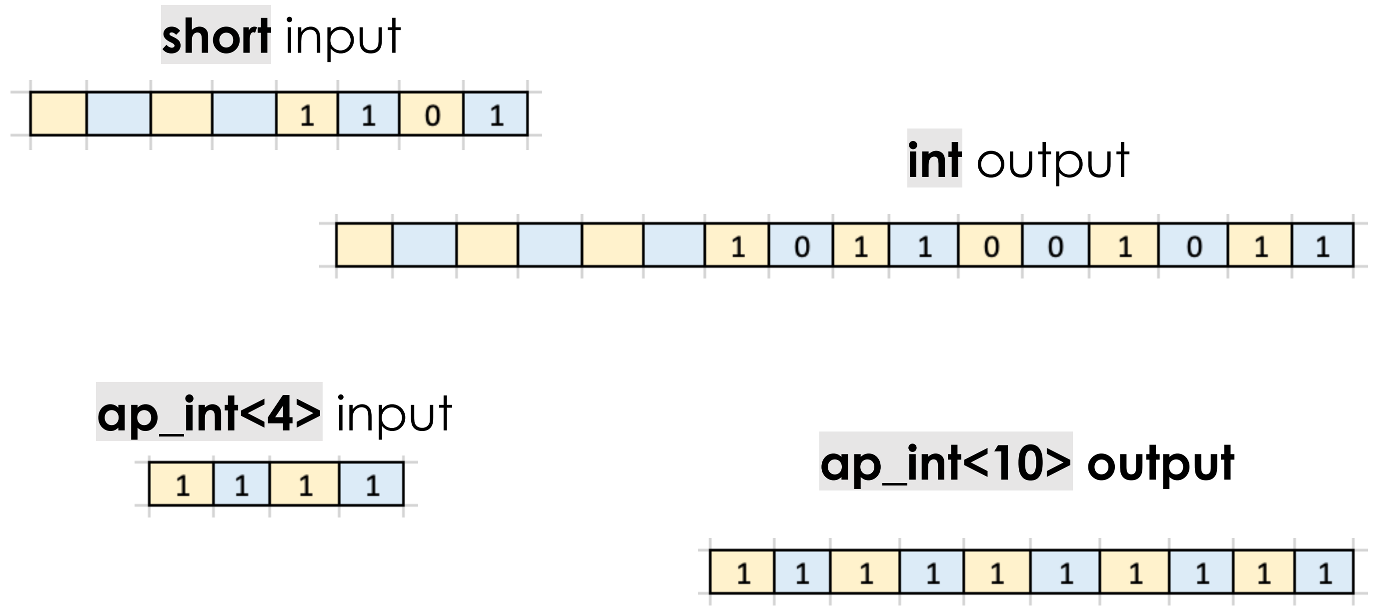
TAC-HEP 2023

Coding styles

Arbitrary precision

Creating hardware, it is useful to use more accurate bit-widths

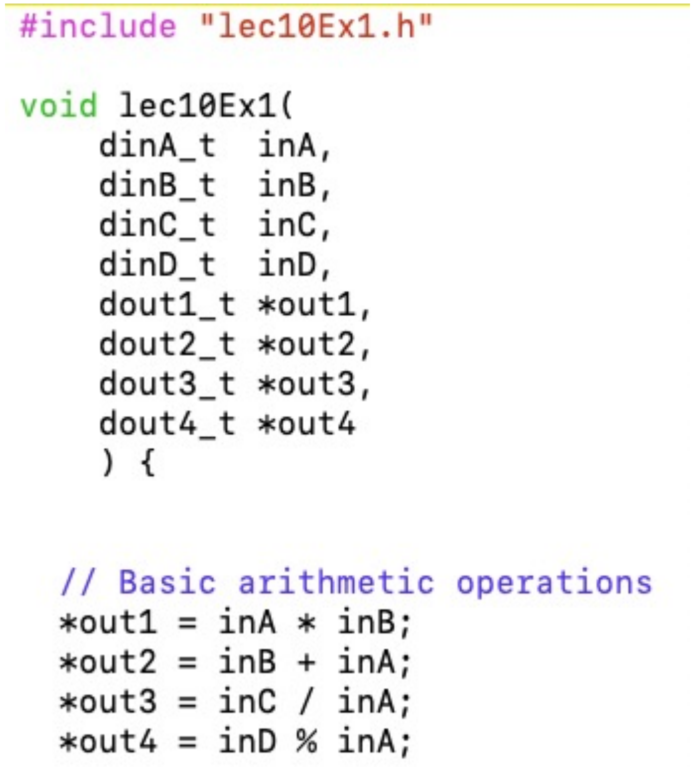
For ex: a case in which the input to a filter is 4-bit and the yielded results requires a maximum of 10-bits



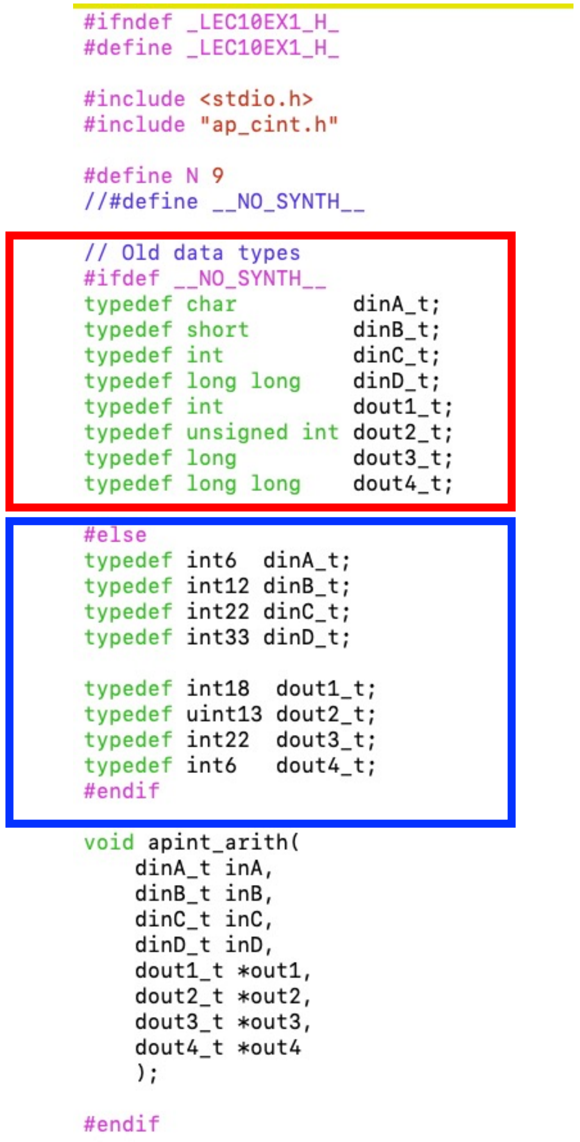
C/C++ data types	Bit-width
(unsigned) char	4
(unsigned) short	8
(unsigned) int	16
(unsigned) long	32
(unsigned) long long	64
float	32
double	64
IntN_t	N=8/16/32/64

Arbitrary precision

Lets see with an example how much we can improve if we use precise data-types



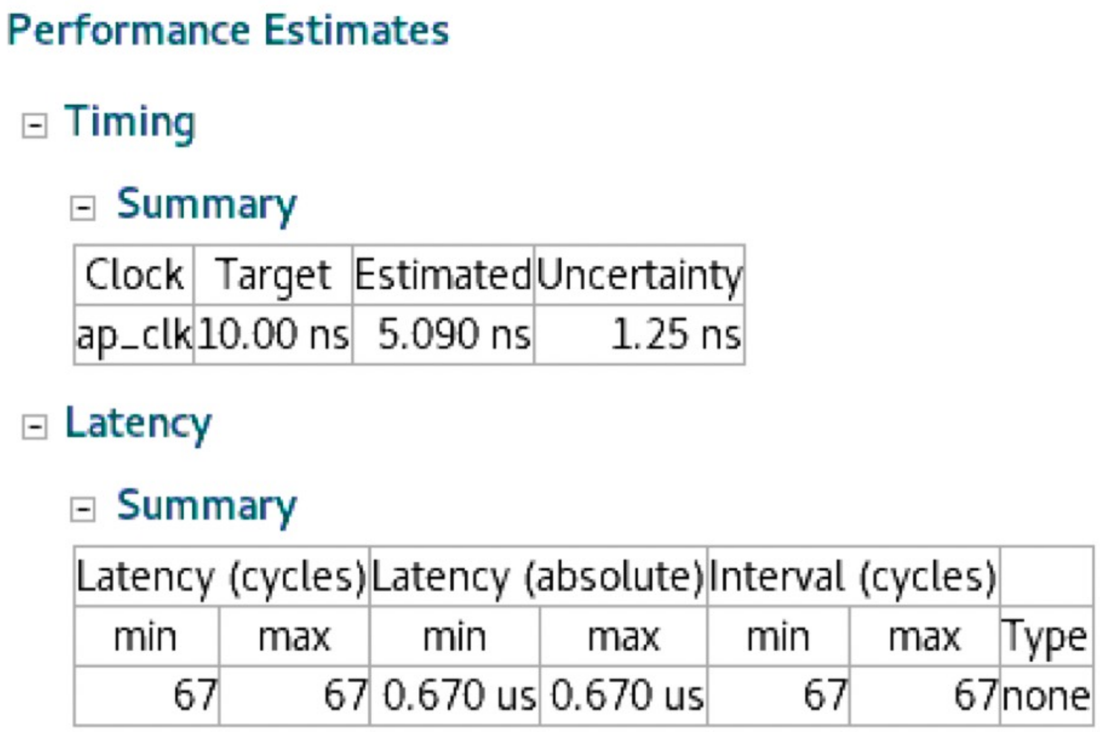
```
#include "lec10Ex1.h"

void lec10Ex1(
    dinA_t  inA,
    dinB_t  inB,
    dinC_t  inC,
    dinD_t  inD,
    dout1_t *out1,
    dout2_t *out2,
    dout3_t *out3,
    dout4_t *out4
    ) {


 // Basic arithmetic operations
 *out1 = inA * inB;
 *out2 = inB + inA;
 *out3 = inC / inA;
 *out4 = inD % inA;
```



```
#ifndef _LEC10EX1_H_
#define _LEC10EX1_H_

#include <stdio.h>
#include "ap_cint.h"

#define N 9
//#define __NO_SYNTH__

// Old data types
#ifdef __NO_SYNTH__
typedef char              dinA_t;
typedef short             dinB_t;
typedef int               dinC_t;
typedef long long         dinD_t;
typedef int               dout1_t;
typedef unsigned int      dout2_t;
typedef long              dout3_t;
typedef long long         dout4_t;
#else
typedef int6  dinA_t;
typedef int12 dinB_t;
typedef int22 dinC_t;
typedef int33 dinD_t;

typedef int18  dout1_t;
typedef uint13 dout2_t;
typedef int22  dout3_t;
typedef int6   dout4_t;
#endif

void apint_arith(
    dinA_t inA,
    dinB_t inB,
    dinC_t inC,
    dinD_t inD,
    dout1_t *out1,
    dout2_t *out2,
    dout3_t *out3,
    dout4_t *out4
    );

#endif
```

Arbitrary precision

Without arbitrary precision



Performance Estimates

⊟ Timing

⊟ Summary

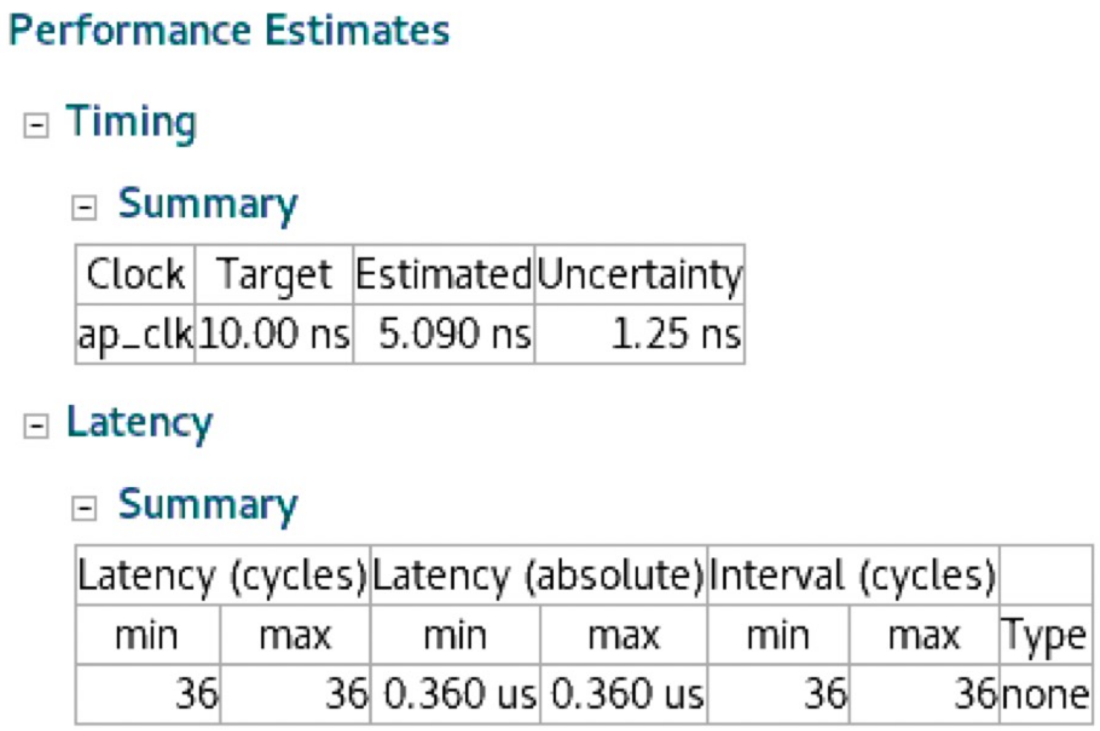
Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.090 ns	1.25 ns

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
67	67	0.670 us	0.670 us	67	67	none

With arbitrary precision



Performance Estimates

⊟ Timing

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.090 ns	1.25 ns

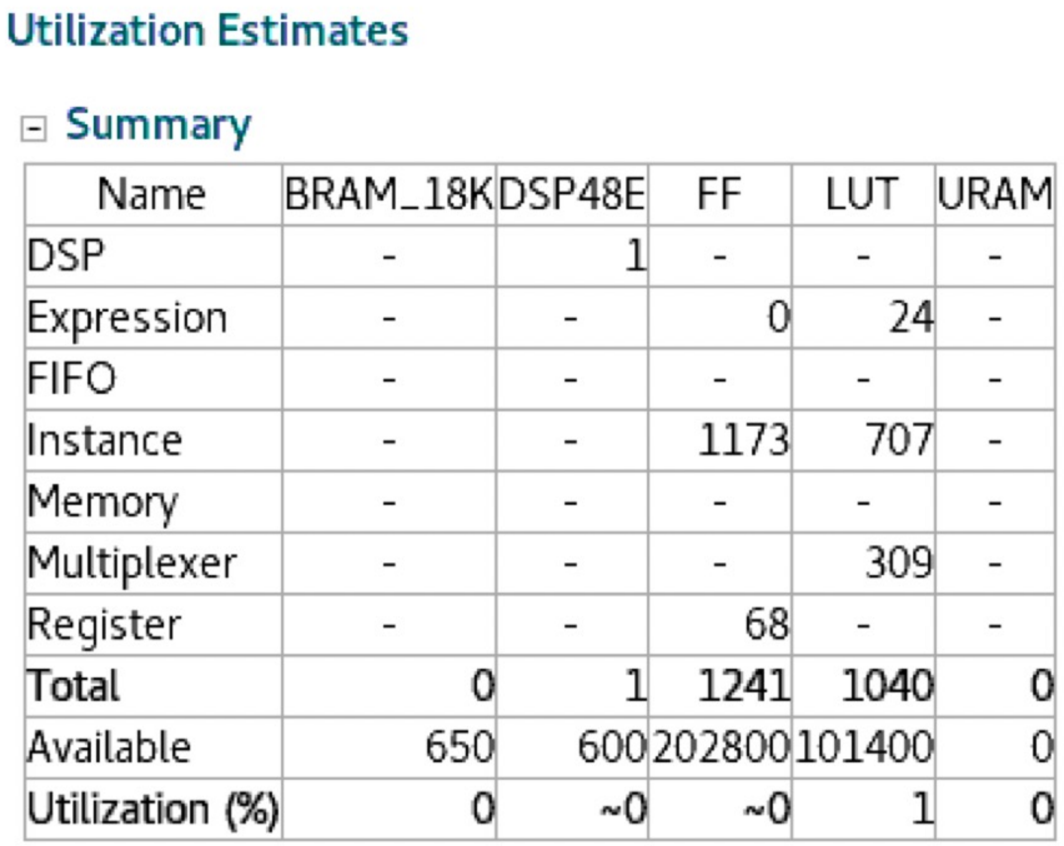
⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
36	36	0.360 us	0.360 us	36	36	none

Arbitrary precision

Without arbitrary precision

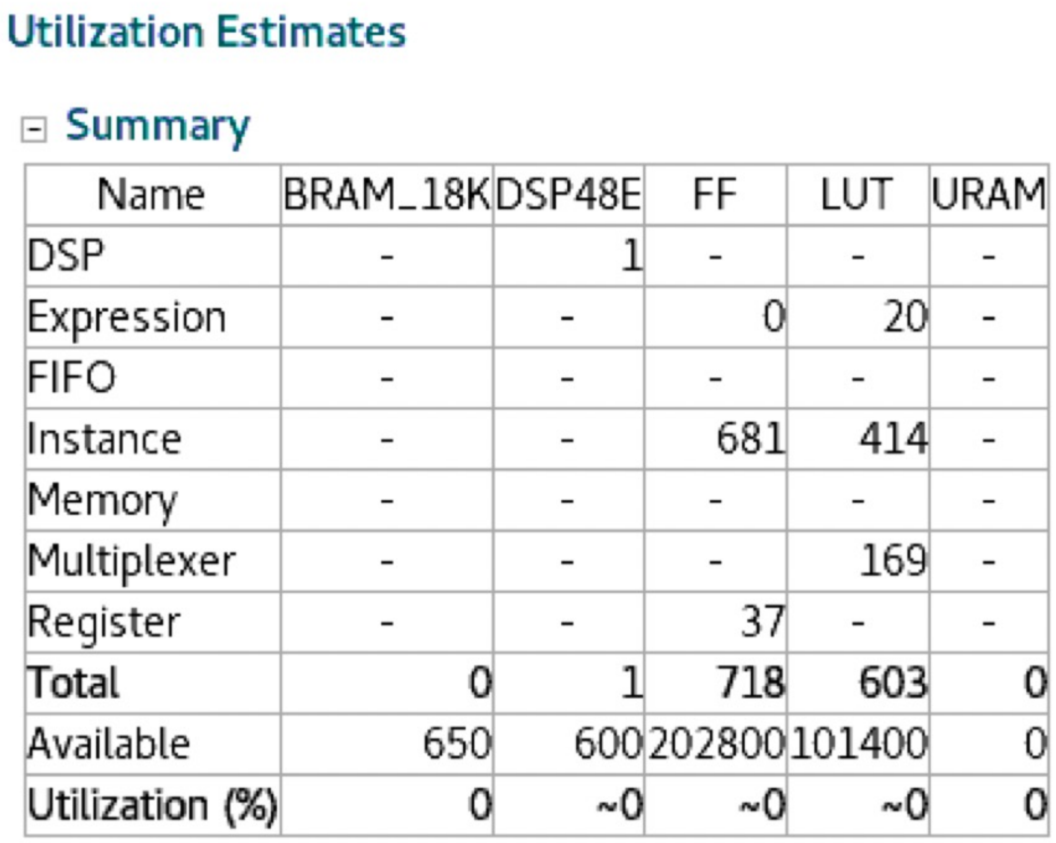


Utilization Estimates

⊟ **Summary**

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	24	-
FIFO	-	-	-	-	-
Instance	-	-	1173	707	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	309	-
Register	-	-	68	-	-
Total	**0**	**1**	**1241**	**1040**	**0**
Available	650	600	202800	101400	0
Utilization (%)	**0**	**~0**	**~0**	**1**	**0**

With arbitrary precision



Utilization Estimates

⊟ **Summary**

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	20	-
FIFO	-	-	-	-	-
Instance	-	-	681	414	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	169	-
Register	-	-	37	-	-
Total	**0**	**1**	**718**	**603**	**0**
Available	650	600	202800	101400	0
Utilization (%)	**0**	**~0**	**~0**	**~0**	**0**

Uninitialized/Unused Variable

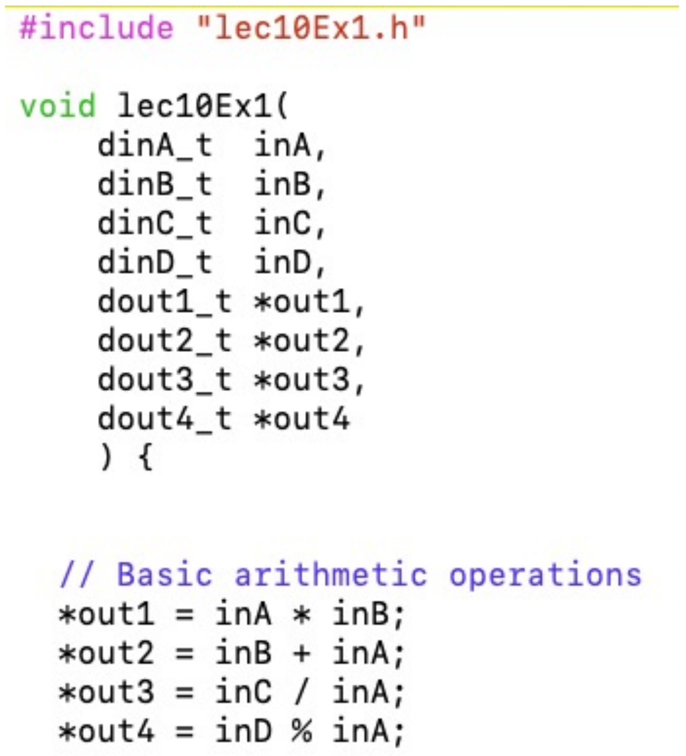
- Uninitialized variables are a result of a poor coding style in which the designer does not initialize variables to 0

- Variable *out* is not an issue: assigned before it is ever read.
- Issue due to variable *in*: used in a computation before it is assigned a value
- Some compilers may automatically assign 0 to *in* at the point of declaration
- **HLS does not use this type of solution**

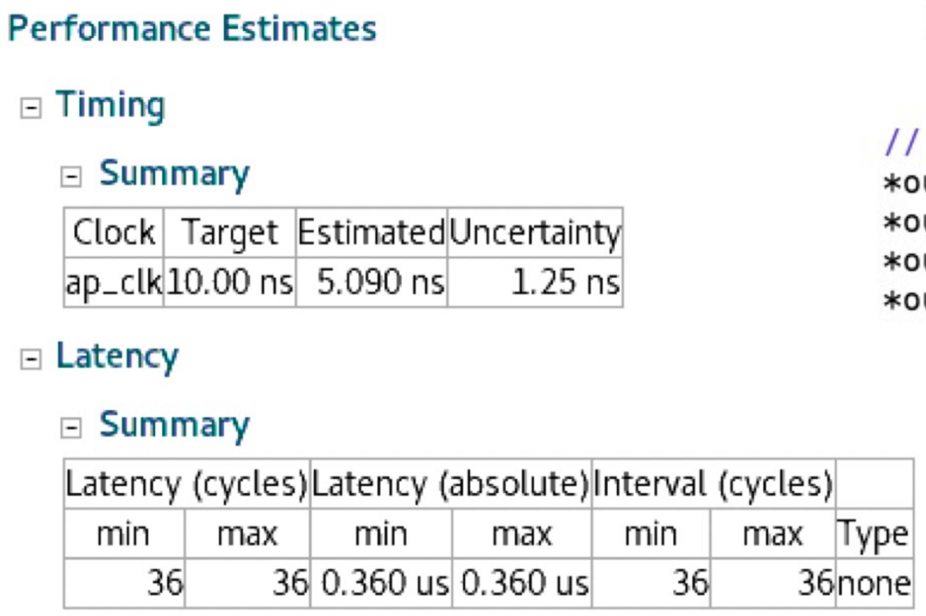
```
int in[N];
int out[N];
...

for(auto i=0; i<N; i=+2){
  out[i]= in[i];
}
```

- Any undefined behavior can be optimized out of the resulting implementation
  - Reduce circuit to nothing
  - Example: piece of code which is not affecting output

Un-used code



```
#include "lec10Ex1.h"

void lec10Ex1(
    dinA_t  inA,
    dinB_t  inB,
    dinC_t  inC,
    dinD_t  inD,
    dout1_t *out1,
    dout2_t *out2,
    dout3_t *out3,
    dout4_t *out4
    ) {


// Basic arithmetic operations
*out1 = inA * inB;
*out2 = inB + inA;
*out3 = inC / inA;
*out4 = inD % inA;
```



Performance Estimates

Timing

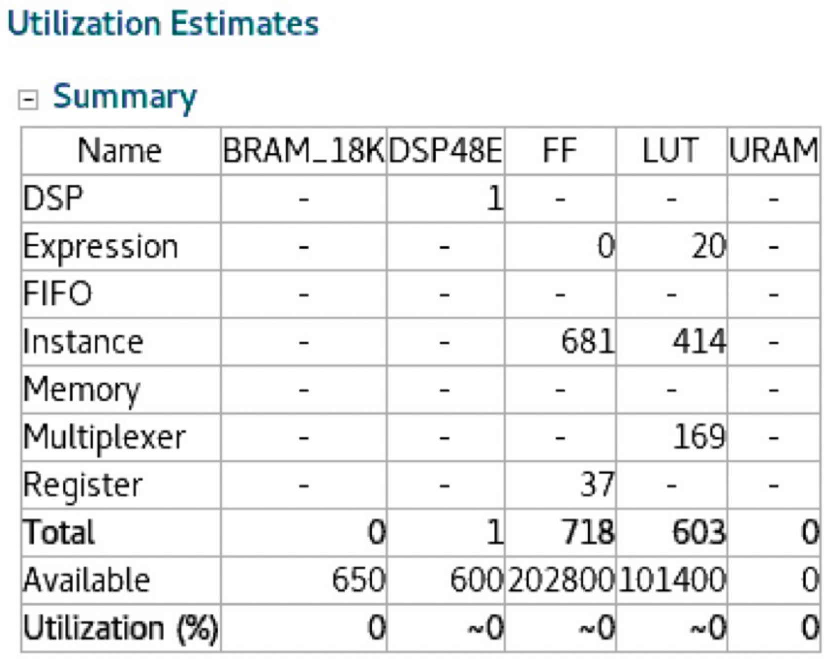
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.090 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
36	36	0.360 us	0.360 us	36	36	none

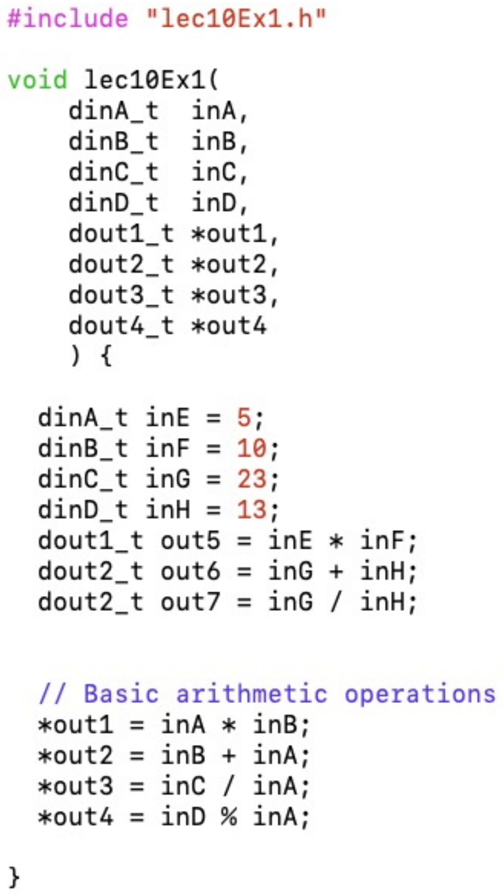


Utilization Estimates

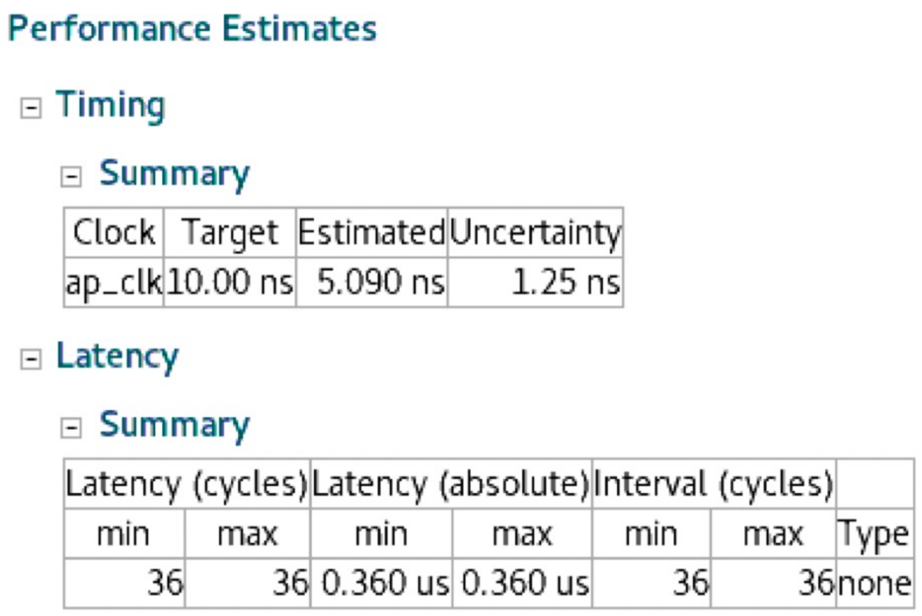
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	20	-
FIFO	-	-	-	-	-
Instance	-	-	681	414	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	169	-
Register	-	-	37	-	-
Total	0	1	718	603	0
Available	650	600	202800	101400	0
Utilization (%)	0	~0	~0	~0	0

Un-used code



```
#include "lec10Ex1.h"

void lec10Ex1(
    dinA_t  inA,
    dinB_t  inB,
    dinC_t  inC,
    dinD_t  inD,
    dout1_t *out1,
    dout2_t *out2,
    dout3_t *out3,
    dout4_t *out4
    ) {

  dinA_t inE = 5;
  dinB_t inF = 10;
  dinC_t inG = 23;
  dinD_t inH = 13;
  dout1_t out5 = inE * inF;
  dout2_t out6 = inG + inH;
  dout2_t out7 = inG / inH;


  // Basic arithmetic operations
  *out1 = inA * inB;
  *out2 = inB + inA;
  *out3 = inC / inA;
  *out4 = inD % inA;

}
```



Performance Estimates

Timing

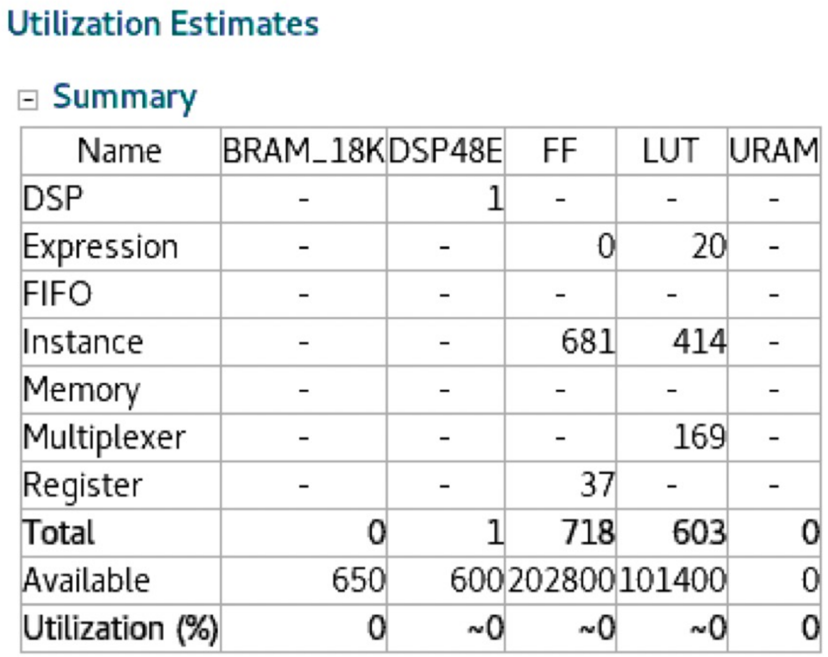
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.090 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
36	36	0.360 us	0.360 us	36	36	none

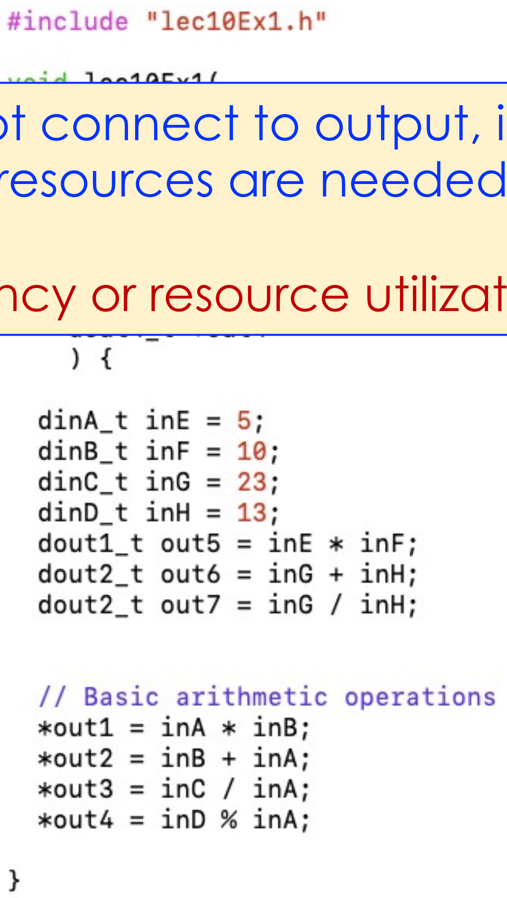


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	20	-
FIFO	-	-	-	-	-
Instance	-	-	681	414	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	169	-
Register	-	-	37	-	-
Total	0	1	718	603	0
Available	650	600	202800	101400	0
Utilization (%)	0	~0	~0	~0	0

Un-used code

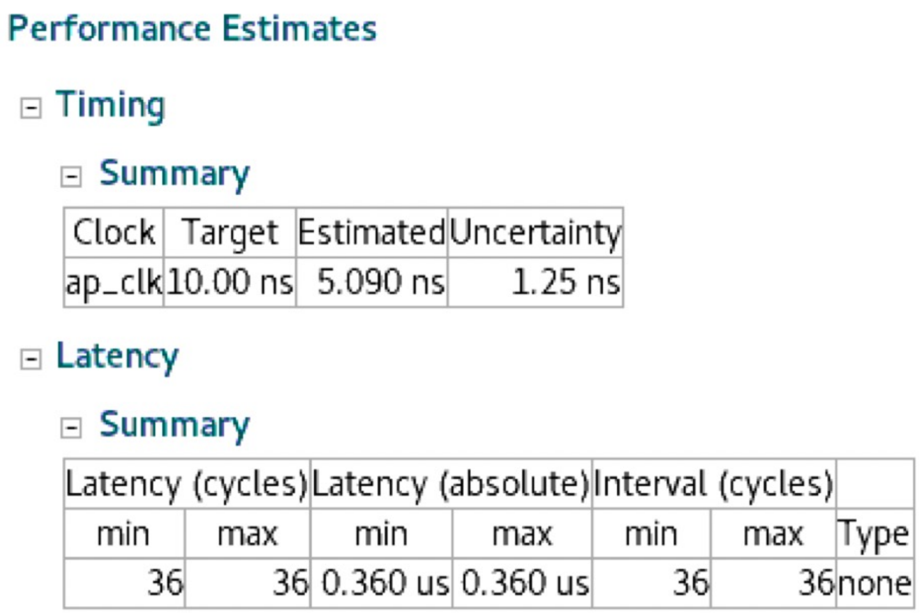


```
#include "lec10Ex1.h"

void lec10Ex1(
```

If any piece of code iss not connect to output, it will not be considered in RTL design and thus no resources are needed

No change in timing, latency or resource utilization is expected

```
     ) {

  dinA_t inE = 5;
  dinB_t inF = 10;
  dinC_t inG = 23;
  dinD_t inH = 13;
  dout1_t out5 = inE * inF;
  dout2_t out6 = inG + inH;
  dout2_t out7 = inG / inH;


  // Basic arithmetic operations
  *out1 = inA * inB;
  *out2 = inB + inA;
  *out3 = inC / inA;
  *out4 = inD % inA;

}
```



Performance Estimates

Timing

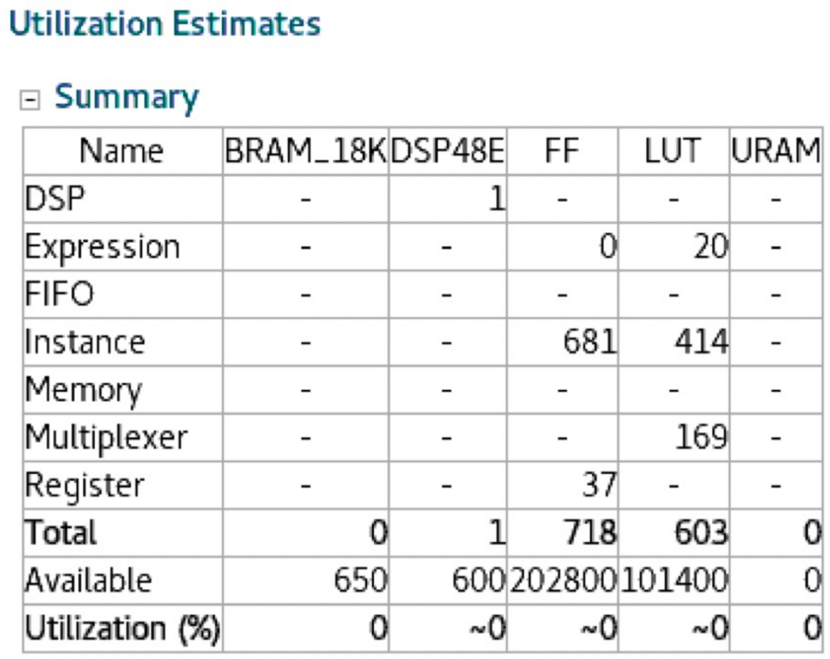
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.090 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
36	36	0.360 us	0.360 us	36	36	none

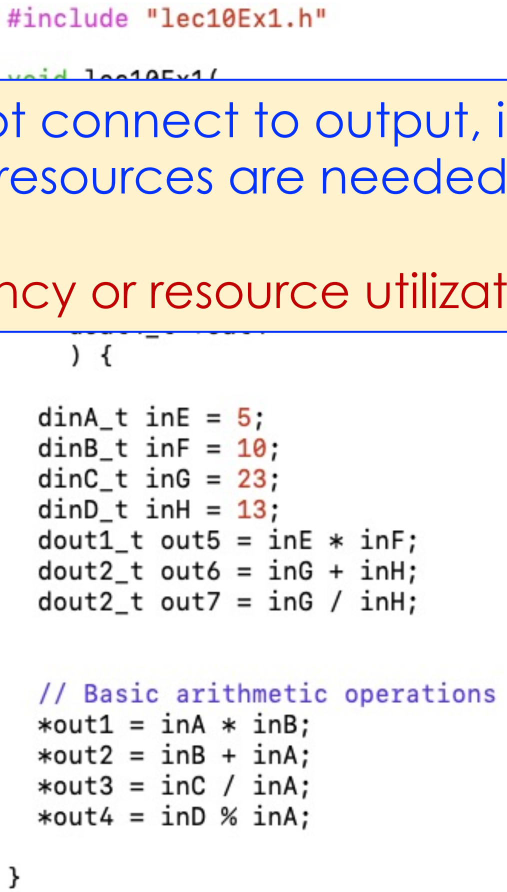


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	20	-
FIFO	-	-	-	-	-
Instance	-	-	681	414	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	169	-
Register	-	-	37	-	-
Total	0	1	718	603	0
Available	650	600	202800	101400	0
Utilization (%)	0	~0	~0	~0	0

Un-used code

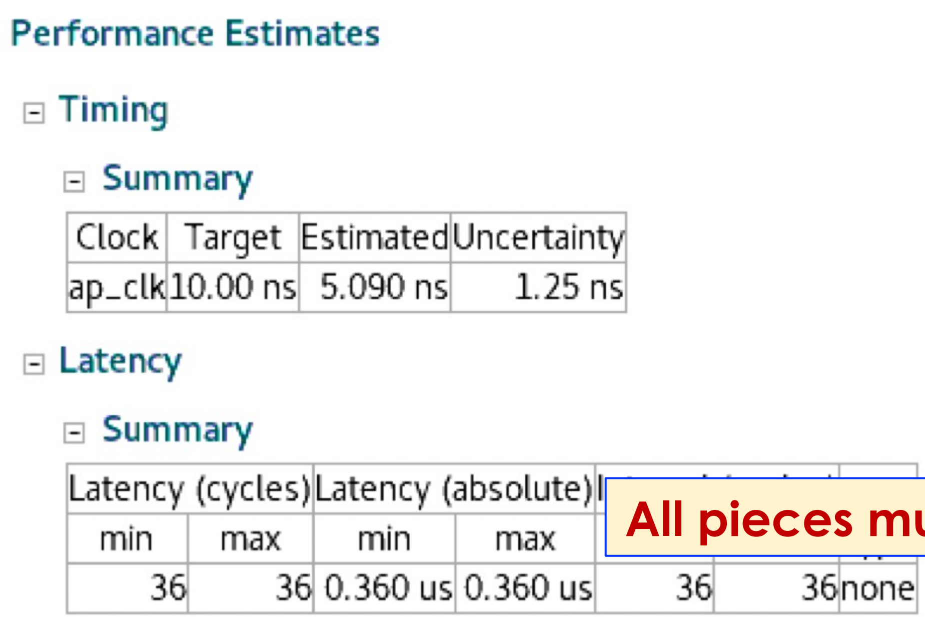


```
#include "lec10Ex1.h"

void lec10Ex1(
```

If any piece of code iss not connect to output, it will not be considered in RTL design and thus no resources are needed

No change in timing, latency or resource utilization is expected

```
    ) {

    dinA_t inE = 5;
    dinB_t inF = 10;
    dinC_t inG = 23;
    dinD_t inH = 13;
    dout1_t out5 = inE * inF;
    dout2_t out6 = inG + inH;
    dout2_t out7 = inG / inH;


    // Basic arithmetic operations
    *out1 = inA * inB;
    *out2 = inB + inA;
    *out3 = inC / inA;
    *out4 = inD % inA;

}
```



Performance Estimates

⊟ Timing

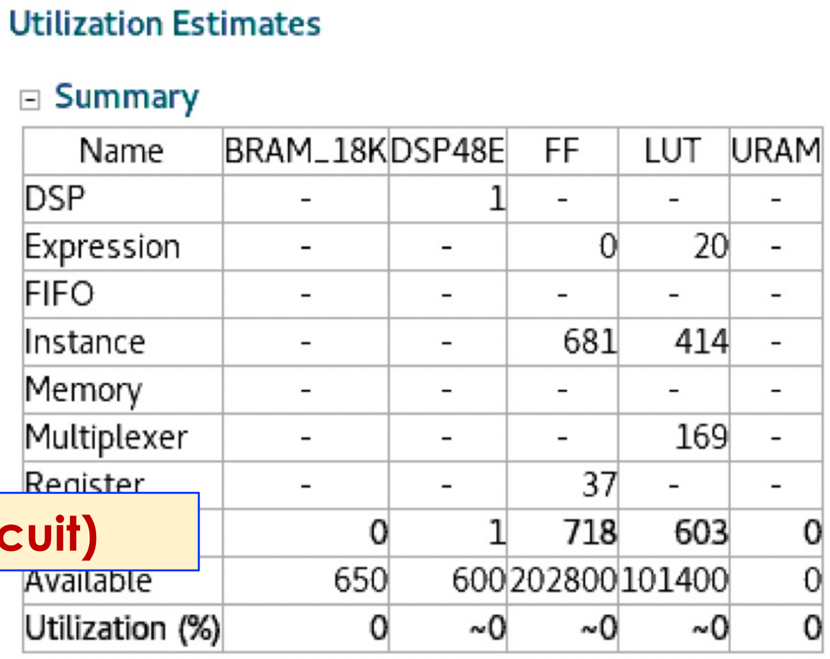
⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.090 ns	1.25 ns

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)				
min	max	min	max			
36	36	0.360 us	0.360 us	36	36	none



Utilization Estimates

⊟ Summary

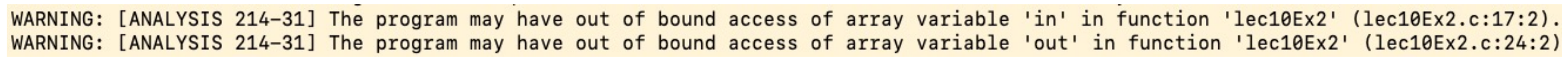
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	20	-
FIFO	-	-	-	-	-
Instance	-	-	681	414	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	169	-
Register	-	-	37	-	-
	0	1	718	603	0
Available	650	600	202800	101400	0
Utilization (%)	0	~0	~0	~0	0

All pieces must be connected (closed circuit)

Out-of-Bounds Memory Access

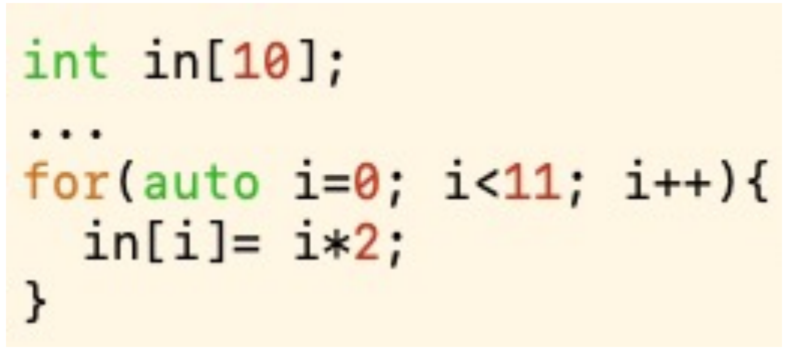
- Memory accesses are expressed either as operations on an array or as operations on an external memory through pointers

- Although the result is functionally incorrect, this kind of error does not usually result in a program crash



```
WARNING: [ANALYSIS 214-31] The program may have out of bound access of array variable 'in' in function 'lec10Ex2' (lec10Ex2.c:17:2).
WARNING: [ANALYSIS 214-31] The program may have out of bound access of array variable 'out' in function 'lec10Ex2' (lec10Ex2.c:24:2)
```

Out-of-bound memory access



```
int in[10];
...
for(auto i=0; i<11; i++){
  in[i]= i*2;
}
```

- Accessing an invalid address triggers a series of events that result in an irrecoverable runtime error in the generated circuit

- HLS implementation assumes that the software algorithm was properly verified, error recovery logic is not included in the generated FPGA implementation

To avoid such situations (out-of-bounds/uninitialized variales) it is recommended that the tool is executed through a dynamic code checker such as valgrind or similar

TAC-HEP 2023

Composite Data Types

Vivado HLS supports composite data types for synthesis:

- Struct
- Enum
- unions

Structs

- When structs are used as arguments to the top-level function
- Ports created by synthesis are a direct reflection of the struct members.
  - Scalar members: standard scalar ports
  - Arrays: memory ports

Struct is used as both:

- A pass-by-value argument (from *i_val* to the return of *o_val*)
- A pointer (**i_pt* to **o_pt*)

- Struct element A results in a 16-bit port
- Struct element B results in a RAM port, accessing 4 elements

There are no limitations in the size or complexity of structs that can be synthesized by Vivado HLS

```
typedef struct {
   unsigned short A;
   unsigned char B[N];
} data_t;

data_t struct_port(data_t i_val, data_t *i_pt, data_t *o_pt);

data_t struct_port(
  data_t  i_val,
  data_t  *i_pt,
  data_t  *o_pt
  ){
   data_t  o_val;
   int i;
   // Transfer pass-by-value structs
   o_val.A = i_val.A+2;
   for (i=0;i<4;i++) {
      o_val.B[i] = i_val.B[i]+2;
   }
   // Transfer pointer structs
   o_pt->A = i_pt->A+3;
   for (i=0;i<4;i++) {
      o_pt->B[i] = i_pt->B[i]+3;
   }
   return o_val;
}
```

TAC-HEP 2023

HLS Pragmas

Pragmas by type

Type	Attributes	
Kernel Optimization	pragma HLS allocation pragma HLS expression_balance pragma HLS latency	pragma HLS reset pragma HLS resource pragma HLS stable
Function Inlining	pragma HLS inline pragma HLS function_instantiate	
Interface Synthesis	pragma HLS interface	
Task-level Pipeline	pragma HLS dataflow pragma HLS stream	
Pipeline	pragma HLS pipeline pragma HLS occurrence	
Loop Unrolling	pragma HLS unroll pragma HLS dependence	
Loop Optimization	pragma HLS loop_flatten pragma HLS loop_merge	pragma HLS loop_tripcount
Array Optimization	pragma HLS array_map pragma HLS array_partition	pragma HLS array_reshape
Structure Packing	pragma HLS data_pack	

Pragma HLS interface

- **C/C++ based design:** Input & outputs are performed in zero time through function arguments
- **RTL design:** same I/O operations must be performed through a port in the design interface & typically operates using a specific I/O protocol
- INTERFACE pragma specifies how RTL ports are created from the function definitions during interface synthesis
- When top level function is synthesized: the arguments (or parameters) to the functions are synthesized into RTL posts
  - *This process is called interface synthesis*
- Lets try to understand more with an example

Interface Synthesis overview

```
#include "sum_io.h"
dout_t sum_io(din_t in1, din_t in2, dio_t *sum) {
   dout_t temp;
   *sum = in1 + in2 + *sum;
   temp = in1 + in2;
   return  temp;
}
```

- Two inputs *n1* & *n2*
- A pointer *sum* that is read from and written to
- A function *return*, the value of *temp*

Default interface settings will synthesize the design into a RTL block with ports as shown:

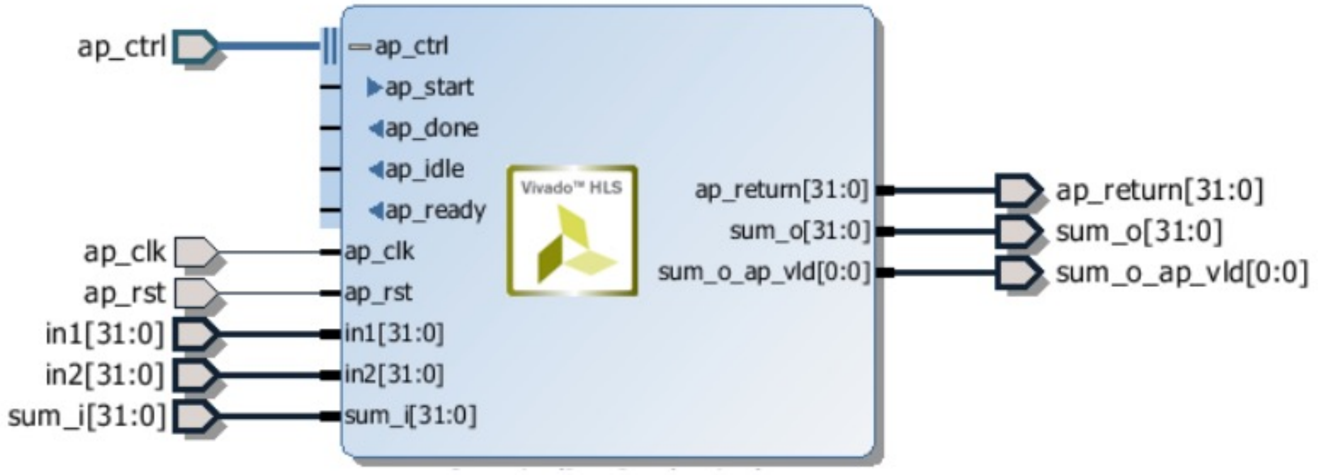


Fig-1

Interface Synthesis overview

Three types of ports in the design:

- **Clock & reset ports:** *ap_clk* and *ap_rst*
  - If the design takes more than 1 clock cycle to complete

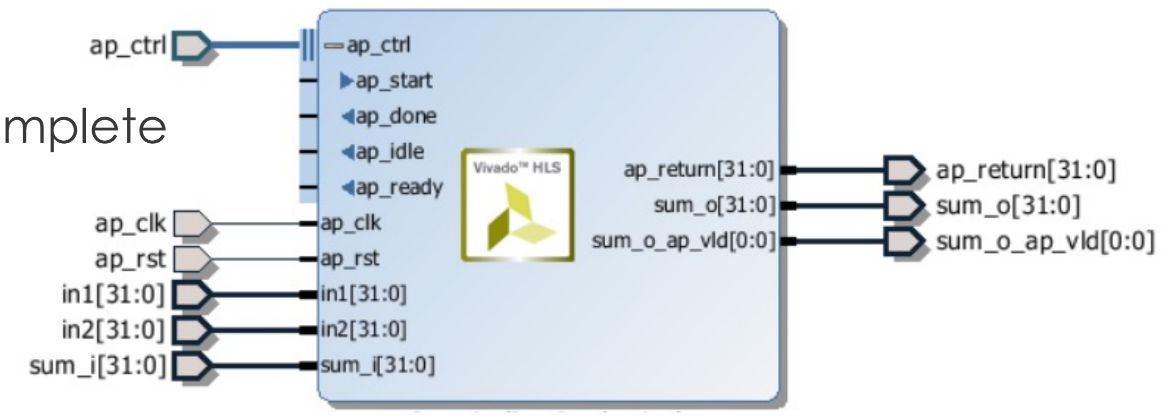


Fig-2

- **Block-level interface protocol:**
  - Added by default & control the block
  - Independent to anyport-level protocol
  - *ap_start*: Control when block can start processing data
  - *ap_ready*: when ready to accept new input
  - *ap_idle*: if the design is idle
  - *ap_done*: completed operation

- **Port level interface portocols:** *in1*, *in2*, *sum_i*, *sum_o*, *sum_o_ap_vld*, and *ap_return*
  - Final group of signals
  - Created for each argument in the top-level function & the function return
  - After block-level protocol has been used to start the operation of block, port level I/O protocols are used to sequence data in and out of the block

Port-Level Interface Protocol

- By default, input pass-by-value arguments and pointers are implemented as simple wire ports with no associated handshaking signal
  - Ex: Input ports are implemented without an I/O protocol, only a data port (data is held stable until it is read)
- By default, output pointers are implemented with an associated output valid signal *(sum_o_ap_vld)* to indicate when the output data is valid
  - No I/O protocol associated with the output port, it is difficult to know when to read the data
  - It is always a good idea to use an I/O protocol on an output
- Function arguments that are both read from & writes to are split into separate input & output ports
  - Ex: sum is implemented as input port sum_i and output port sum_o with associated I/O protocol port sum_o_ap_vld
- Function with a return value, an output port *ap_return* is implemented to provide the return value

- Completion of one transaction: the block-level protocols indicate the function is complete with the *ap_done* signal.
  - Also indicates the data on port *ap_return* is valid and can be read

RTL Port timing

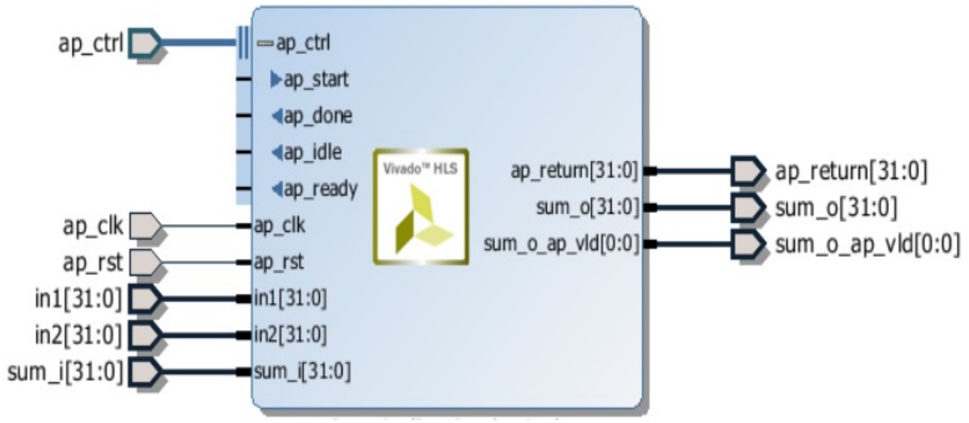


Fig-3

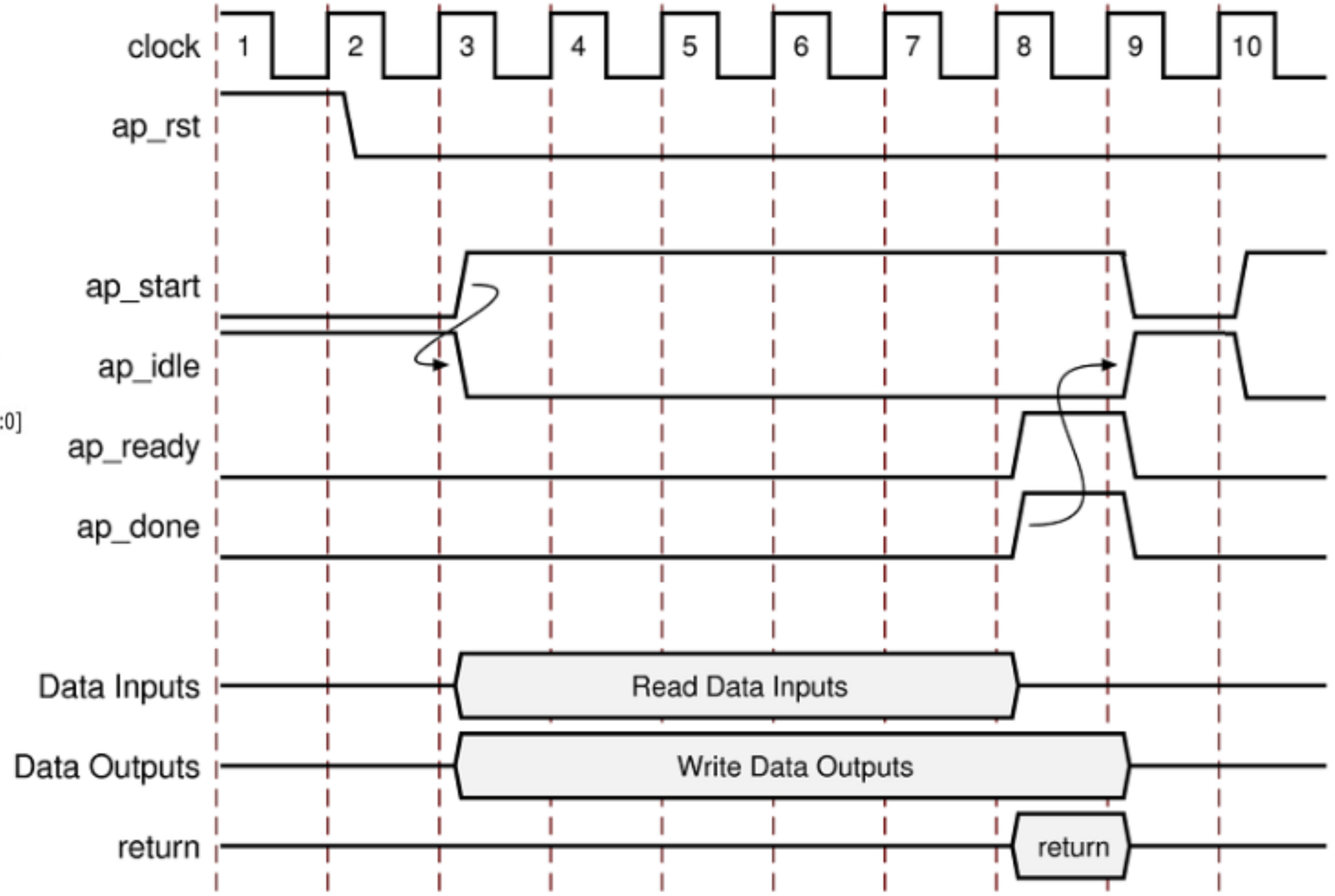


Fig-4

RTL Port timing

- Design starts: *ap_start* is High
- *ap_idle* signal goes Low indicating design is operating
- Input data is read at any CLK after the first cycl.
- HLS schedules when the reads occur
- *ap_ready* signal is asserted high when all inputs have been read
- When output sum is calculated, the associated output handshake (*sum_o_ap_vld*) indicates that the data is valid
- When the function completes, *ap_done* is asserted. This also indicates that the data on *ap_return* is valid
- Port *ap_idle* is High indicating design is waiting start again

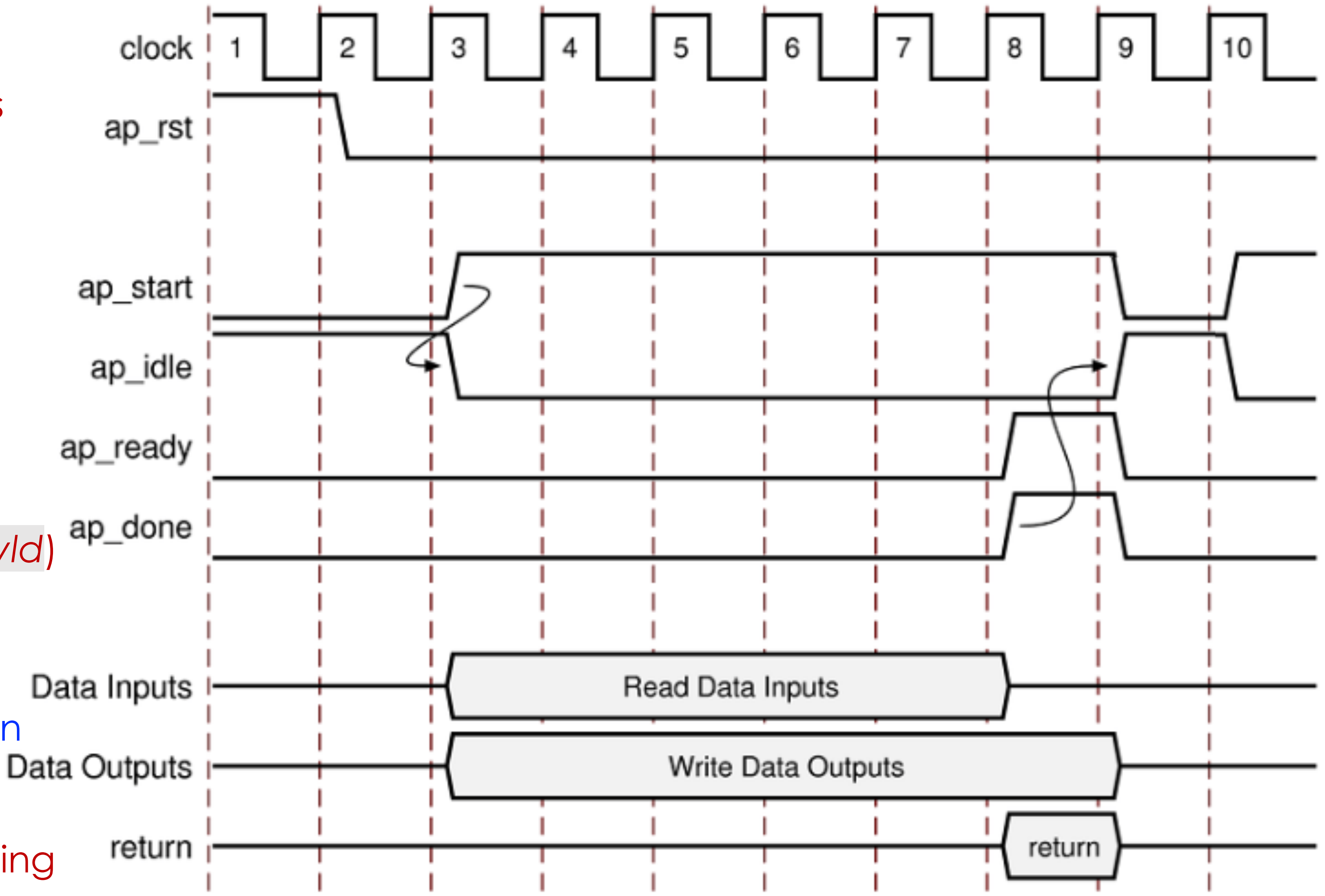


Fig-5

Interface Synthesis I/O Protocols

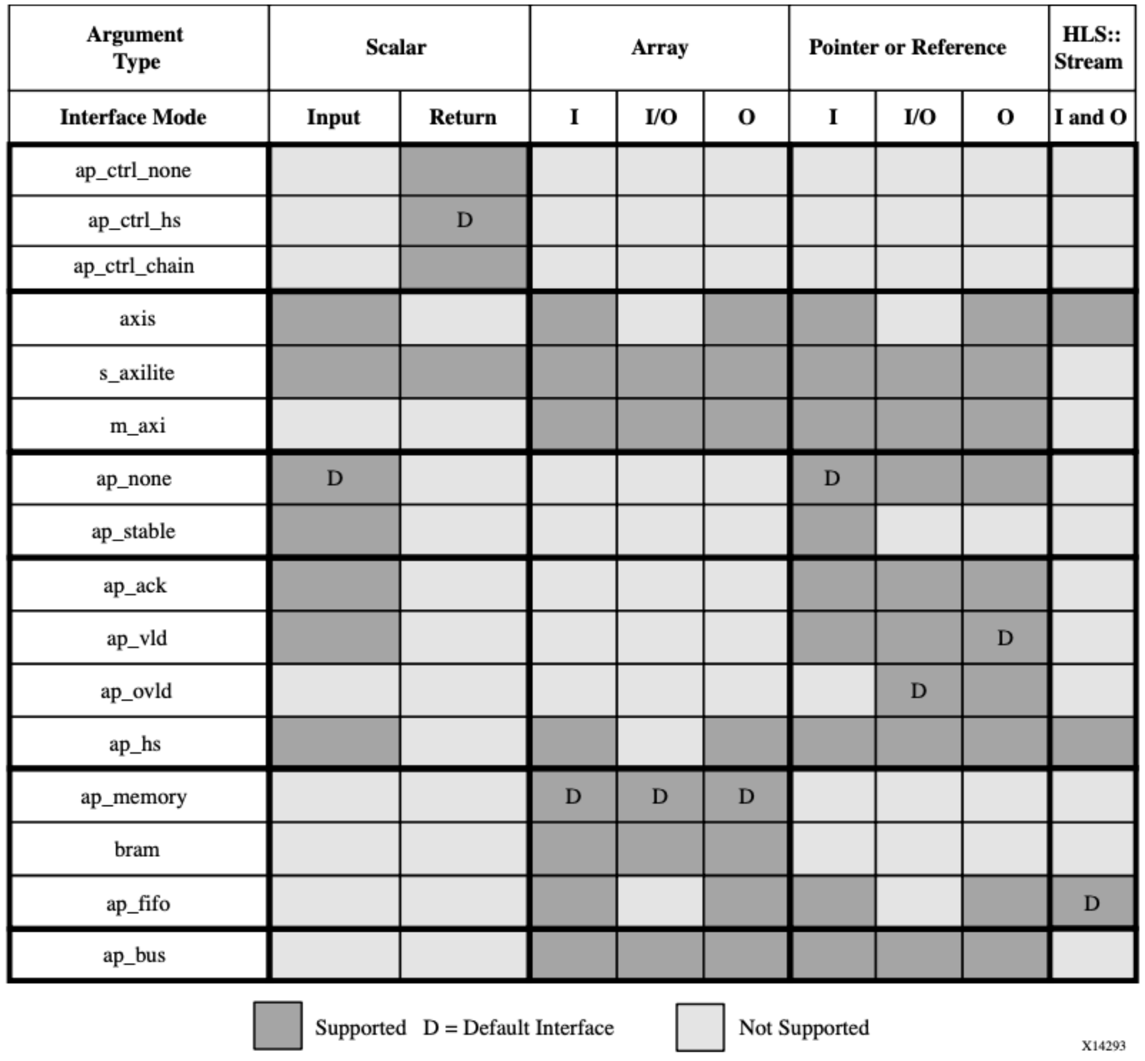
The type of interfaces that are created by interface synthesis depends on the type of C/C++ argument

D: Default interface mode for each type

I: Input arguments, which are only read

O: Output arguments, which are only written to

I/O: Input/Output arguments, which are both read and written



Argument Type	Scalar		Array			Pointer or Reference			HLS:: Stream
Interface Mode	Input	Return	I	I/O	O	I	I/O	O	I and O
ap_ctrl_none	□	▨	□	□	□	□	□	□	□
ap_ctrl_hs	□	▨ D	□	□	□	□	□	□	□
ap_ctrl_chain	□	▨	□	□	□	□	□	□	□
axis	▨	□	▨	□	▨	▨	□	▨	▨
s_axilite	▨	▨	▨	▨	▨	▨	▨	▨	□
m_axi	□	□	▨	▨	▨	▨	▨	▨	□
ap_none	▨ D	□	□	□	□	▨ D	▨	▨	□
ap_stable	▨	□	□	□	□	▨	□	□	□
ap_ack	▨	□	□	□	□	▨	▨	▨	□
ap_vld	▨	□	□	□	□	▨	▨	▨ D	□
ap_ovld	□	□	□	□	□	□	▨ D	▨	□
ap_hs	▨	□	▨	□	▨	▨	▨	▨	▨
ap_memory	□	□	▨ D	▨ D	▨ D	□	□	□	□
bram	□	□	▨	▨	▨	□	□	□	□
ap_fifo	□	□	▨	□	▨	▨	□	▨	▨ D
ap_bus	□	□	▨	▨	▨	▨	▨	▨	□

▨ Supported D = Default Interface □ Not Supported

X14293

Fig-6

Pragma HLS interface: Syntax

#pragma HLS interface **<mode>** port=<name> bundle=<string> register \
register_mode=<mode> depth=<int> offset=<string> clock=<string> name=<string> \
num_read_outstanding=<int> num_write_outstanding=<int> max_read_burst_length=<int> \
max_write_burst_length=<int>

<mode>: Specifies the interface protocol mode for function arguments, global variables used by the function, or the block-level control protocols

- *ap_none:* No protocol. The interface is a data port
- *ap_stable*: No protocol. The interface is a data port. The HLS tool assumes the data port is always stable after reset, which allows internal optimizations to remove unnecessary registers
- *ap_vld*: Implements the data port with an associated valid port to indicate when the data is valid for reading or writing
- *ap_ack*: Implements the data port with an associated acknowledge port to acknowledge that the data was read or written
- *ap_hs*: Implements the data port with associated valid and acknowledge ports to provide a two-way handshake to indicate when the data is valid for reading and writing and to acknowledge that the data was read or written
- *ap_ovld*: Implements the output data port with an associated valid port to indicate when the data is valid for reading or writing

Pragma HLS interface: Syntax

```
#pragma HLS interface <mode> port=<name> bundle=<string> register  \
register_mode=<mode> depth=<int> offset=<string>  clock=<string> name=<string>  \
num_read_outstanding=<int> num_write_outstanding=<int>  max_read_burst_length=<int> \
max_write_burst_length=<int>
```

<mode>: Specifies the interface protocol mode for function arguments, global variables used by the function, or the block-level control protocols

- *ap_fifo*: Implements the port with a standard FIFO interface using data I/O ports with associated active-Low FIFO empty and full ports
- *ap_bus*: Implements pointer and pass-by-reference ports as a bus interface.
- *ap_memory*: Implements array arguments as a standard RAM interface
- *axis*: Implements all ports as an AXI4-Stream interface
- *s_axilite*: Implements all ports as an AXI4-Lite interface
- *m_axi*: Implements all ports as an AXI4 interface
- *ap_ctrl_none*: No block-level I/O protocol
- *ap_ctrl_hs*: Implements a set of block-level control ports to start the design operation and to indicate when the design is idle, done, and ready for new input data
- *ap_ctrl_chain*: Implements a set of block-level control ports to start the design operation, continue operation & indicate when the design is idle, done, & ready for new input data

Pragma HLS interface: Syntax

```
#pragma HLS interface <mode> port=<name> bundle=<string> register  \
register_mode=<mode> depth=<int> offset=<string>  clock=<string> name=<string>  \
num_read_outstanding=<int> num_write_outstanding=<int>  max_read_burst_length=<int> \
max_write_burst_length=<int>
```

port=<name>: Specifies the name of the function argument, function return, or global variable which the INTERFACE pragma applies to

bundle=<string>: Groups function arguments into AXI interface ports

register: An optional keyword to register the signal and any relevant protocol signals, and causes the signals to persist until at least the last cycle of the function execution.

- Ap_none, ap_ack, ap_vld, ap_ovld, ap_hs, ap_stable, axis, s_axilite

Pragma HLS interface: Example

```
void example(int A[50], int B[50]) {
  #pragma HLS INTERFACE axis port=A
  #pragma HLS INTERFACE axis port=B
  int i;
  for(i = 0; i < 50; i++){
    B[i] = A[i] + 5;
  }
}
```

Both function arguments are implemented using an AXI4-Stream interface

```
#pragma HLS interface ap_ctrl_none port=return
```

Turns off block-level I/O protocols, and is assigned to the function return value

```
#pragma HLS interface ap_vld register port=InData
```

The function argument *InData* is specified to use the *ap_vld* interface, and also indicates the input should be registered

```
#pragma HLS interface ap_memory port=lookup_table
```

This exposes the global variable *lookup_table* as a port on the RTL design, with an *ap_memory* interface

Assignment Week-5

1. Do exercise mention on slide-24
2. A matrix multiplication using two for loops and compare results for pragma loop_flatten & unroll
3. Write a simple program doing arithmetic operations(+, -, *, /, %) between two variable use of arbitrary precision to compare results between stand c/c++ data types and using ap_(u)int<N>
4. Write a program using an array with N(=10/15/20) elements and then restructure the code with a struct having N-data member. Compare the results of two programs

TAC-HEP 2023

Questions?

Acknowledgement

Lectures are compiled using content from Xilinx's public pages/examples or different user guides

TAC-HEP 2023

Additional material

Assignment submission

- Where to submit:
  - https://pages.hep.wisc.edu/~varuns/assignments/TAC-HEP/

- Use your login machine credentials

- Submit one file per week

- Try to submit by following week's Tuesday

Correct Time

From 03.28.2023 onwards

- Tuesdays: 9:00-10:00 CT / 10:00-11:00 ET / 16:00-17:00 CET
- Wednesday: 11:00-12:00 CT / 12:00-13:00 ET / 18:00-19:00 CET

Jargons

- **ICs - Integrated chip:** assembly of hundreds of millions of transistors on a minor chip
- **PCB:** Printed Circuit Board
- **LUT - Look Up Table aka 'logic'** - generic functions on small bitwidth inputs. Combine many to build the algorithm
- **FF - Flip Flops** - control the flow of data with the clock pulse. Used to build the pipeline and achieve high throughput
- **DSP - Digital Signal Processor** - performs multiplication and other arithmetic in the FPGA
- **BRAM - Block RAM** - hardened RAM resource. More efficient memories than using LUTs for more than a few elements
- **PCIe or PCI-E - Peripheral Component Interconnect Express**: is a serial expansion bus standard for connecting a computer to one or more peripheral devices
- **InfiniBand** is a computer networking communications standard used in high-performance computing that features very high throughput and very low latency
- **HLS -** High Level Synthesis - compiler for C, C++, SystemC into FPGA IP cores
- **DRCs -** Design Rule Checks
- **HDL** - Hardware Description Language - low level language for describing circuits
- **RTL** - Register Transfer Level - the very low level description of the function and connection of logic gates
- **FIFO** – First In First Out memory
- **Latency** - time between starting processing and receiving the result
  - Measured in clock cycles or seconds
- **II - Initiation Interval** - time from accepting first input to accepting next input

Assignment Week-3

- Use target device: **xc7k160tfbg484-2**
- Clock period of 10ns

1. Execute the code (lec5Ex2.tcl) using CLI (slide-25) and compare the results with GUI results for C-Simulation, C-Synthesis

2. Vary following parameters for two cases: high and very high values and compare with 1 for both CLI and GUI
   - Variable: “samples”
   - Variable: “N”

3. Run example lec3Ex2a

Assignment Week-4

1. Do a matrix multiplication of two 1-dimensional arrays – A[N]*B[N], where N > 5
   a) Report synthesis results without any pragma directives
   b) Add as many pragma directives possible
      i. Report any conflicts (if reported in logs) between two pragmas

2. Compare the analysis perspective (Performance) for different case shared today
3. For Array_partitioning, instead of using complete, use block and cyclic with different factors