Wisig Trainee

Indian Institute of Technology Hyderabad

Email: velicharla@outlook.com



Date:19-03-2023

#### Hls ASSIGNMENT-2

Repeat the experiments in Assignment 1 with the following three independent changes.

- **1.** Use 32bit inputs and figure out the what the bit width should be for your design using your understanding of digital design and arithmetic operations.
- **2.** Use arbitrary precision data type with bit width of 4 for integer and 24 for fractional part of the inputs and figure out what the bit width of the output should be for your design.
- 3. Use HLS stream blocking interfaces for the ports in your design.

#### Record the following observations.

- **1.** Change in resource consumption from Assignment 1 and Assignment 2.1 above. If it changed, what and why it changed? Did the timing report change? Include all code, entire synthesis report, simulation output and C/RTL co-simulation output and report.
- **2.** Change in resource consumption from Assignment 2.1 to Assignment 2.2. If it changed, what and why it changed? Did the timing report change? Include all code, entire synthesis report, simulation output, and C/RTL co-simulation output and report.
- **3.** Repeat Assignment 2.1 and 2.2 with the change mentioned in Assignment 3. Analyse change in timing report from Assignment 1, 2.1, 2.2, and report your understanding of why it changed. Include all code, entire synthesis report, simulation output, and C/RTL co-simulation output and report.

# 1.

# Design:

#### Multiplier function:

```
In the last of the last of
                                                                                                                                                                                                                                                                                              le hls_arbitary_tb.cpp
                1 #include "hls arbitary.h"
                 2⊖ void multiplier(dinA_t a, dinB_t b, doutC_t *c)
                                    {
                 3
                 4
                                                                          #pragma HLS INTERFACE ap none port=a
                 5
                                                                         #pragma HLS INTERFACE ap_none port=b
                                                                         #pragma HLS INTERFACE ap_none port=c
                 6
                 7
                                                                         *c = a * b;
                8
                9
        10
```

#### Header file:

```
Image: Index of the i
```

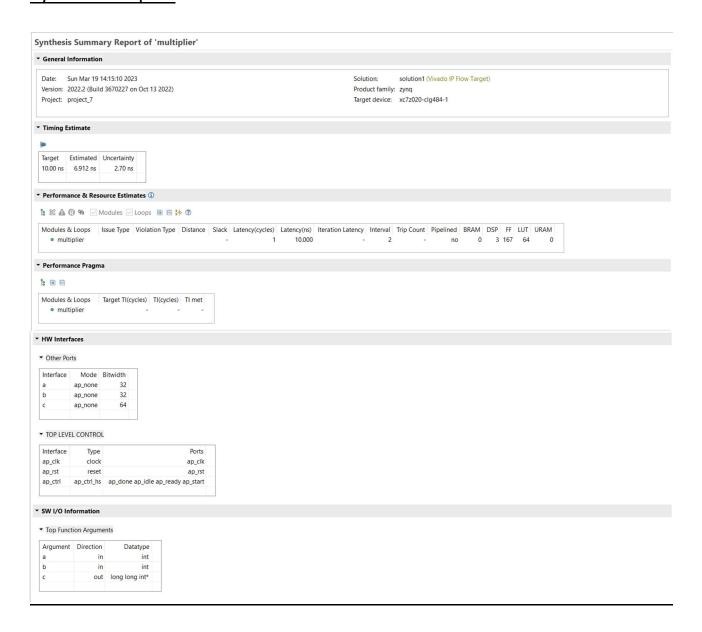
### Test Bench:

```
    hls_arbitary_tb.cpp 

    □

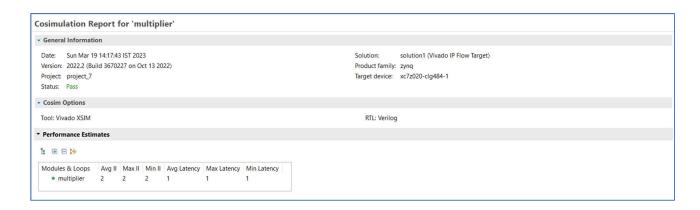
hls_arbitary.cpp
                  h hls_arbitary.h
 1 #include "hls_arbitary.h"
 2⊖int main() {
        doutC_t result;
 3
 4 // Define test inputs
      dinA_t test_inputs[10][2] = {
            {0x00000011, 0x00000001},
 6
            {0x00000011, 0x00000002},
 7
            {0x00000011, 0x00000003},
 8
 9
            {0x00000011, 0x00000004},
            {0x00000011, 0x00000005},
10
            {0x00000011, 0x00000006},
11
            {0x00000011, 0x00000007},
12
13
            {0x00000011, 0x00000008},
            {0x00000011, 0x00000009},
14
15
            {0x00000011, 0x0000000A}
16
       };
       for (int i = 0; i < 10; i++)
17
18
19
            dinA t a = test inputs[i][0];
20
            dinB_t b = test_inputs[i][1];
21
            multiplier(a, b, & result);
            cout << a << "*"<< b << "=" << result <<endl;
22
23
        // printf("%d * %d = %d\n",a,b,result);
24
      }
25
       return 0;
26 }
 27
```

```
hls_arbitary.cpp
 2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../oneDrive/Desktop/vitis/hls_arbitary_tb.cpp in debug mode
   Compiling ../../../../OneDrive/Desktop/vitis/hls_arbitary.cpp in debug mode
   Generating csim.exe
 6 17*1=17
 717*2=34
 8 17*3=51
 9 17*4=68
1017*5=85
11 17*6=102
12 17*7=119
13 17*8=136
1417*9=153
15 17*10=170
16 INFO: [SIM 1] CSim done with 0 errors.
```





```
/// Intra-Transaction Progress: Completed Transaction / Total Tran
```



- As the bit width increases, more hardware resources are required to process the larger data. This can lead to an increase in resource consumption, such as a larger number of flip-flops, DSP, LUT's
- With larger data sizes, there may be an increase in data movement between different hardware modules, such as between memory and processing units. This increase in data movement can result in longer latencies and slower timing

# 2. Design:

#### Multiplier function:

```
In the last of the last of
                                                                                                                                                                                                                                                                                                         le his fixedpoint tb.cpp
             1 #include "hls fixedpoint.h"
                2⊖ void multiplier(dinA t a, dinB t b, doutC t *c)
                3 {
                                                                    #pragma HLS INTERFACE ap none port=a
               4
                                                                    #pragma HLS INTERFACE ap none port=b
                5
                                                                    #pragma HLS INTERFACE ap none port=c
               6
                                                                    *c = a * b;
                7
              8 }
              9
       10
```

#### Header file:

```
Implication of the implicat
```

## Test Bench:

```
le his fixedpoint.cpp
                    multiplier_csim.log
                                        Synthesis Summary(solution1)
                                                                      ( C
  1 #include"hls_fixedpoint.h"
  2⊖int main()
  3 {
        doutC_t result;
  4
  5
        dinA t test inputs[10][2] = {
                  {4.125,3.264589},
  6
  7
                  {4.56215,2.4568912},
                  {3.69845632,2.456891},
  8
  9
                  {3.69845632,2.456891},
 10
                  {1.25698456,3.569874},
 11
                  {1.4567896512,5.623366},
 12
                  {5.6684258,6.54566321},
 13
                  {6.2255620,5.636115},
 14
                  {4.6985230,2.5698523},
 15
                  {1.563995,3.2656343}
 16
            };
 17
           for (int i = 0; i < 10; i++)
 18
 19
               dinA_t a=test_inputs[i][0];
               dinB_t b=test_inputs[i][1];
 20
 21
                multiplier(a,b,&result);
              std::cout<< a << "*"<< b << "="<< result<<std::endl;
 22
              //printf("%d * %d= %d\n",a,b,result);
 23
 24
 25
           return 0;
 26 }
 27
```

# C simulation printed output:

```
multiplier_csim.log 🛭 🗊 Synthesis Summary(solution1)
hls_fixedpoint.cpp
                                                      Co-simulation Report(solutio...
 2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
     Compiling ../../../../OneDrive/Desktop/vitis/hls_fixedpoint_tb.cpp in debug mode
     Compiling ../../../../oneDrive/Desktop/vitis/hls_fixedpoint.cpp in debug mode
     Generating csim.exe
 64.125*3.26459=13.4664
 74.56215*2.45689=11.2087
 83.69846*2.45689=9.0867
 93.69846*2.45689=9.0867
101.25698*3.56987=4.48728
111.45679*5.62337=8.19206
125.66843*6.54566=37.1036
136.22556*5.63611=35.088
144.69852*2.56985=12.0745
151.56399*3.26563=5.10744
16 INFO: [SIM 1] CSim done with 0 errors.
18
```



```
/// Intra-Transaction Progress: Completed Transaction / Total Transaction * 100% // Intra-Transaction Progress: Measured Latency / Latency Estimation * 100% // RTL Simulation: * 110 (10.00%) @ "Intra-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time" // RTL Simulation: * 1 10 (10.00%) @ "155000" // RTL Simulation: * 1 10 (10.00%) @ "155000" // RTL Simulation: * 1 10 (10.00%) @ "155000" // RTL Simulation: * 2 10 (10.00%) @ "155000" // RTL Simulation: * 3 10 (10.00%) @ "255000" // RTL Simulation: * 3 10 (10.00%) @ "255000" // RTL Simulation: * 4 10 (10.00%) @ "255000" // RTL Simulation: * 5 10 (10.00%) @ "255000" // RTL Simulation: * 5 10 (10.00%) @ "315000" // RTL Simulation: * 7 10 (100.00%) @ "315000" // RTL Simulation: * 7 10 (100.00%) @ "315000" // RTL Simulation: * 8 10 (100.00%) @ "375000" // RTL Simulation: * 9 10 (100.00%) @ "375000" // RTL Simulation: * 9 10 (100.00%) @ "375000" // RTL Simulation: * 9 10 (100.00%) @ "375000" // RTL Simulation: * 9 10 (100.00%) @ "375000" // RTL Simulation: * 9 10 (100.00%) @ "375000" // RTL Simulation: * 9 10 (100.00%) @ "375000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 9 10 (100.00%) @ "355000" // RTL Simulation: * 10 (100.00%) @ "355000" // RTL
```



- fixed-point arithmetic can be more hardware-efficient than arbitrary arithmetic.
   This is because fixed-point arithmetic uses a fixed number of bits to represent both the integer and fractional parts of a number
- In terms of timing requirements, fixed-point arithmetic can offer faster computation times than arbitrary precision arithmetic.

3.

### Design:

#### Multiplier function:

```
    hls_stream1.cpp 
    □ hls_stream1_tb.cpp

                                                                        hls_stream1.h
                                                                                          multiply_cs
Synthesis Summary(solution1)
1 #include"hls_stream1.h"
  2<sup>©</sup> void multiply(hls::stream<dinA_t> &A, hls::stream<dinB_t>& B,hls::stream<doutC_t> &C)
  3 {
  4
         dinA_t a;
  5
         dinB t b;
  6
         doutC_t result;
  7
         a=A.read();
  8
         b=B.read();
  9
         result=a*b;
 10
         C.write(result);
 11
```

#### Header file:

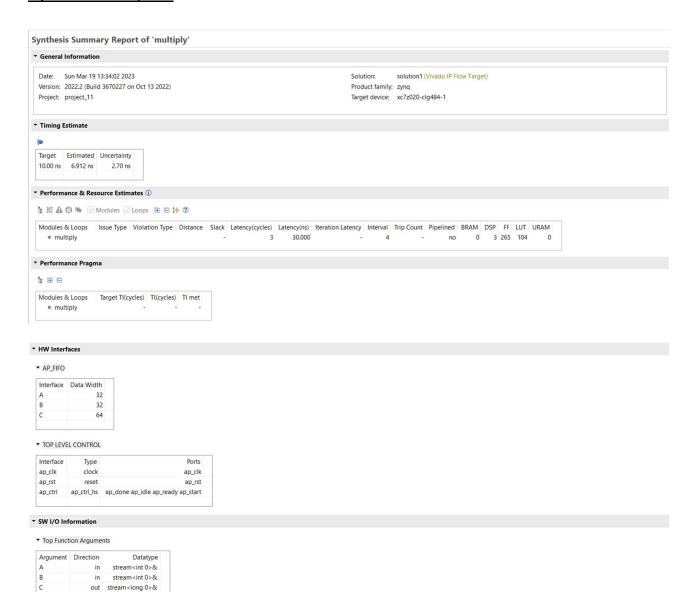
## <u>Test Bench:</u>

```
    hls_stream1_tb.cpp 
    hls_stream1.h 
    multiply_csim.log

lack hls_stream1.cpp
  1 #include"hls_stream1.h"
  2⊖int main()
  3 {
        hls::stream<dinA_t> a;
 4
 5
        hls::stream<dinB_t> b;
        hls::stream<doutC_t> c;
 6
  7
        dinA t test inputs[10][2] = {
  8
                  {0x00000011,0x00000001},
  9
                  {0x00000011,0x00000002},
 10
                  {0x00000011,0x00000003},
 11
                  {0x00000011,0x00000004},
 12
                  {0x00000011,0x00000005},
13
                  {0x00000011,0x00000006},
14
                  {0x00000011,0x00000007},
15
                  {0x00000011,0x000000008},
16
                  {0x00000011,0x00000009},
17
                  {0x00000011,0x0000000A},
 18
           };
 19
           for (int i = 0; i < 10; i++)
20
21
                a.write(test_inputs[i][0]);
22
                b.write(test_inputs[i][1]);
                multiply(a,b,c);
23
24
                //out >> result;
25
              std::cout<< test_inputs[i][0] << "*"<< test_inputs[i][1] << "="<<c.read()<<std::endl;
 26
27 }
```

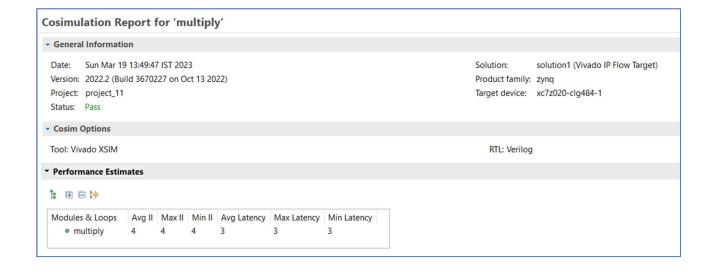
```
    multiply_csim.log 
    □ Synthesis Summary(solution1)

lack hls_stream1.cpp
              le hls_stream1_tb.cpp
                               hls_stream1.h
 2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
    Compiling ../../../../OneDrive/Desktop/vitis/hls_stream1_tb.cpp in debug mode
     Compiling ../../../../oneDrive/Desktop/vitis/hls_stream1.cpp in debug mode
    Generating csim.exe
 6 17*1=17
 717*2=34
 8 17*3=51
 9 17*4=68
10 17*5=85
11 17*6=102
12 17*7=119
13 17*8=136
1417*9=153
15 17*10=170
16 INFO [HLS SIM]: The maximum depth reached by any hls::stream() instance in the design is 1
17 INFO: [SIM 1] CSim done with 0 errors.
19
```





```
// // Intra-Transaction Progress: Completed Transaction / Total Transaction / Total Transaction Progress: Measured Latency / Latency Estimation ** 190% // // RTL Simulation: ** "Intra-Transaction Progress" (** Intra-Transaction Progress** [** Intra-Transaction Progress** [** Intra-Transaction Progress**] (** "Simulation Time" // // RTL Simulation: ** 1 10 [100.00%] (** 175000" // RTL Simulation: ** 1 10 [100.00%] (** 175000" // RTL Simulation: ** 2 10 [100.00%] (** 175000" // RTL Simulation: ** 3 / 10 [100.00%] (** 175000" // RTL Simulation: ** 4 10 [100.00%] (** 175000" // RTL Simulation: ** 4 10 [100.00%] (** 175000" // RTL Simulation: ** 5 / 10 [100.00%] (** 175000" // RTL Simulation: ** 5 / 10 [100.00%] (** 175000" // RTL Simulation: ** 7 / 10 [100.00%] (** 175000" // RTL Simulation: ** 7 / 10 [100.00%] (** 175000" // RTL Simulation: ** 8 / 10 [100.00%] (** 175000" // RTL Simulation: ** 8 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (** 175000" // RTL Simulation: ** 9 / 10 [100.00%] (**
```



4.

### Design:

#### Multiplier function:

```
hls_stream2_tb.cpp
                    Synthesis Summary(solution1)
                                                 Co-simulation Report(solution1)
                                                                                hls_stream2.h
  1 #include"hls_stream2.h"
  2@ void multiply(hls::stream<dinA_t> &A, hls::stream<dinB_t>& B,hls::stream<doutC_t> &C)
 3 {
 4
        dinA_t a;
 5
        dinB_t b;
        doutC_t result;
 6
  7
        a=A.read();
        b=B.read();
 9
        result=a*b;
10
        C.write(result);
11 }
12
13
```

#### Header file:

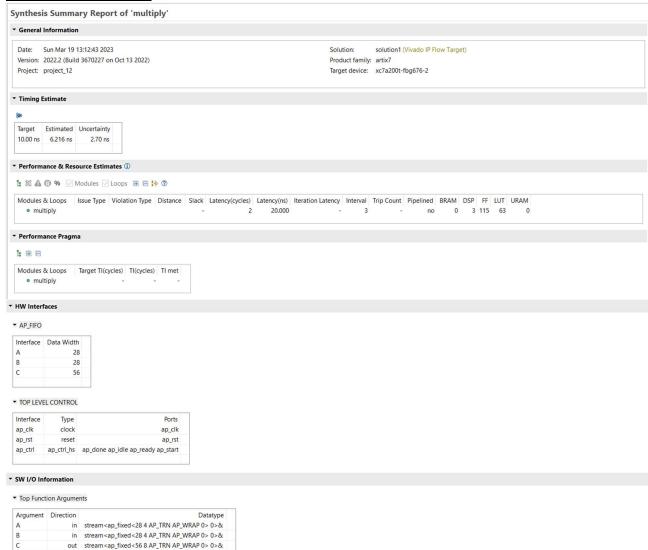
#### Test Bench:

```
hls_stream2.h
                hls_stream2.cpp

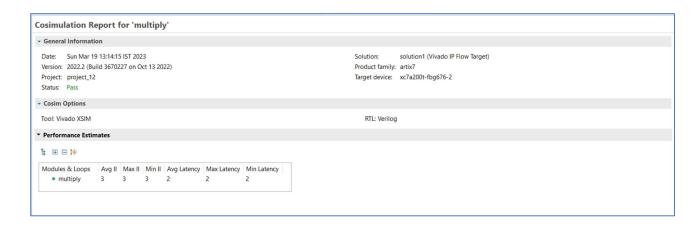
    hls_stream2_tb.cpp 
    □ Synthesis Summary(solution1)

                                                                                    Co-simulation Report(solution
1 #include"hls_stream2.h"
  2⊖int main()
  3 {
        hls::stream<dinA_t> a;
  4
  5
        hls::stream<dinB t> b;
  6
        hls::stream<doutC t> c;
  7
         dinA_t test_inputs[10][2] = {
  8
                  {4.125,3.264589},
  9
                  {4.56215,2.4568912},
 10
                  {3.69845632,2.456891},
 11
                  {3.69845632,2.456891},
 12
                  {1.25698456,3.569874},
                  {1.4567896512,5.623366},
 13
 14
                  {5.6684258,6.54566321},
 15
                  {6.2255620,5.636115},
 16
                  {4.6985230,2.5698523},
 17
                  {1.563995,3.2656343}
 18
           };
           for (int i = 0; i < 10; i++)
 19
 20
                a.write(test_inputs[i][0]);
 21
                b.write(test_inputs[i][1]);
 22
 23
                multiply(a,b,c);
 24
                //out >> result;
 25
              std::cout<< test_inputs[i][0] << "*"<< test_inputs[i][1] << "="<<c.read()<<std::endl;
 26
 27 }
 28
```

```
h hls_stream2.h
           le hls_stream2.cpp
                       2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
 3 Compiling ../../../../OneDrive/Desktop/vitis/hls_stream2_tb.cpp in debug mode
 4 Compiling ../../../../OneDrive/Desktop/vitis/hls_stream2.cpp in debug mode
   Generating csim.exe
 64.125*3.26459=13.4664
 74.56215*2.45689=11.2087
 83.69846*2.45689=9.0867
93.69846*2.45689=9.0867
101.25698*3.56987=4.48728
111.45679*5.62337=8.19206
125.66843*6.54566=37.1036
136.22556*5.63611=35.088
144.69852*2.56985=12.0745
151.56399*3.26563=5.10744
16 INFO [HLS SIM]: The maximum depth reached by any hls::stream() instance in the design is 1
17 INFO: [SIM 1] CSim done with 0 errors.
19
```







- HLS stream interface, particularly when it comes to timing requirements. Specifically, because the stream interface is designed to process data in a pipelined fashion, it may require more complex timing considerations than traditional fixed or arbitrary data types. This can make it more difficult to ensure that the design meets timing requirements, particularly when working with high-speed data streams.
- HLS stream interface can help reduce resource consumption, particularly when dealing with large data sets. This is because the stream interface enables data to be processed in a pipelined fashion, which can help reduce the amount of storage and processing resources required to handle large data sets.
- stream interface can also help to reduce the overall memory footprint of a design, since data can be streamed in and out

#### Github:

https://github.com/velicharlagokulkumar/vitis-hls/tree/main/project\_7 https://github.com/velicharlagokulkumar/vitis-hls/tree/main/project\_8 https://github.com/velicharlagokulkumar/vitis-hls/tree/main/project\_11 https://github.com/velicharlagokulkumar/vitis-hls/tree/main/project\_12