Worcester Polytechnic Institute
Electrical and Computer Engineering Department
Methodologies for System Level Design and Modeling - ECE 5723
Online Offering – Fall 2020

First Name: Zhenyuan
Last Name: Liu

Grade:

Problem 1. ______/ 30

Problem 2. ______/ 30

Problem 3. ______/ 40

Total: ______/ 100

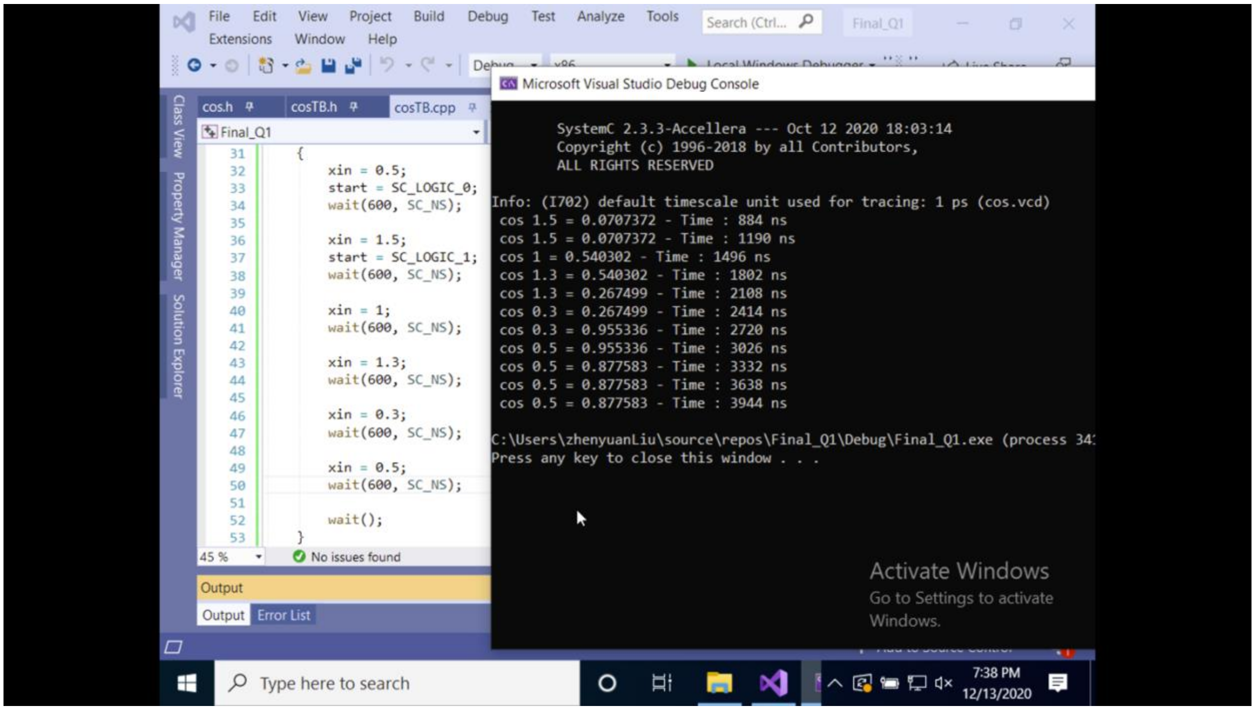
THIS IS AN OPEN BOOK OPEN NOTE EXAM
YOU MUST SHOW COMPLETE WORK ON ALL PROBLEMS

Please Sign:

I have worked on this test alone and have not received any help from my classmates, instructors, and students elsewhere.

Name: zhenyuan Liu Signature:

1. The systemC code and the testbench of the cosine approximation are included in the Final_Q1 folder. The testing inputs are in radius, 0 < xin < 1.57. The best iteration number of this calculation is 2.



```
{
    xin = 0.5;
    start = SC_LOGIC_0;
    wait(600, SC_NS);

    xin = 1.5;
    start = SC_LOGIC_1;
    wait(600, SC_NS);

    xin = 1;
    wait(600, SC_NS);

    xin = 1.3;
    wait(600, SC_NS);

    xin = 0.3;
    wait(600, SC_NS);

    xin = 0.5;
    wait(600, SC_NS);

    wait();
}
```

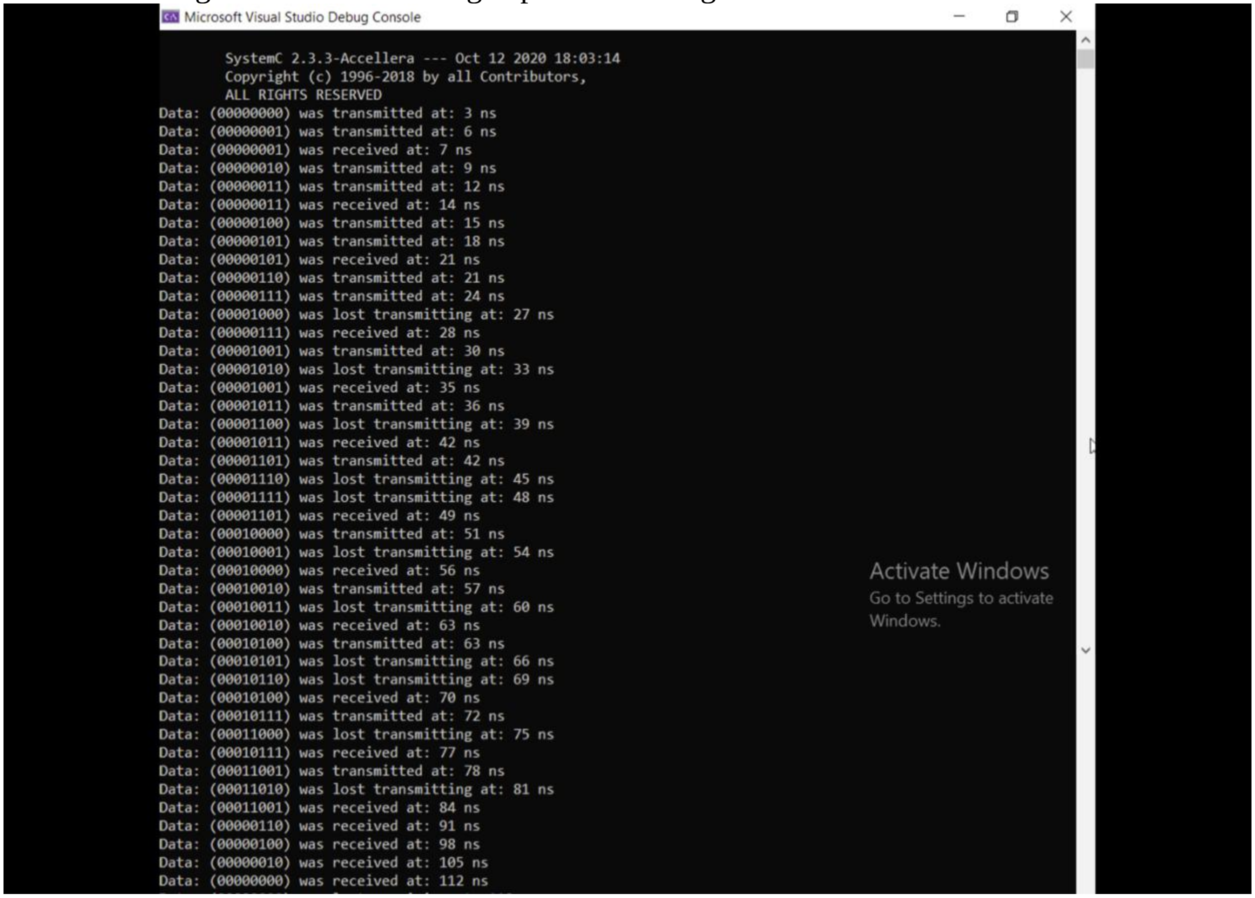
```
        SystemC 2.3.3-Accellera --- Oct 12 2020 18:03:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED

Info: (I702) default timescale unit used for tracing: 1 ps (cos.vcd)
 cos 1.5 = 0.0707372 - Time : 884 ns
 cos 1.5 = 0.0707372 - Time : 1190 ns
 cos 1 = 0.540302 - Time : 1496 ns
 cos 1.3 = 0.540302 - Time : 1802 ns
 cos 1.3 = 0.267499 - Time : 2108 ns
 cos 0.3 = 0.267499 - Time : 2414 ns
 cos 0.3 = 0.955336 - Time : 2720 ns
 cos 0.5 = 0.955336 - Time : 3026 ns
 cos 0.5 = 0.877583 - Time : 3332 ns
 cos 0.5 = 0.877583 - Time : 3638 ns
 cos 0.5 = 0.877583 - Time : 3944 ns

C:\Users\zhenyuanLiu\source\repos\Final_Q1\Debug\Final_Q1.exe (process 34
Press any key to close this window . . .
```

Figure 1. Test results of the cosine approximation calculation in SystemC. When xin was ‘0.5’ and when start was ‘0’, the calculation did not start. However, after start was ‘1’, the calculation successfully output 5 different inputs with the best iteration number of 2. When time was at 1802ns, the first iteration of cos(1.3) was incorrect, the second iteration of cos(1.3) was correct, which was 0.267499. When time was at 3026ns, the first iteration of cos(0.5) was incorrect, the second iteration of cos(0.5) was correct, which was 0.877583.

2. All the code are included inside of folder Final_Q2. Question 2 implemented a non-blocking buffer channel using export on the target side.



Microsoft Visual Studio Debug Console

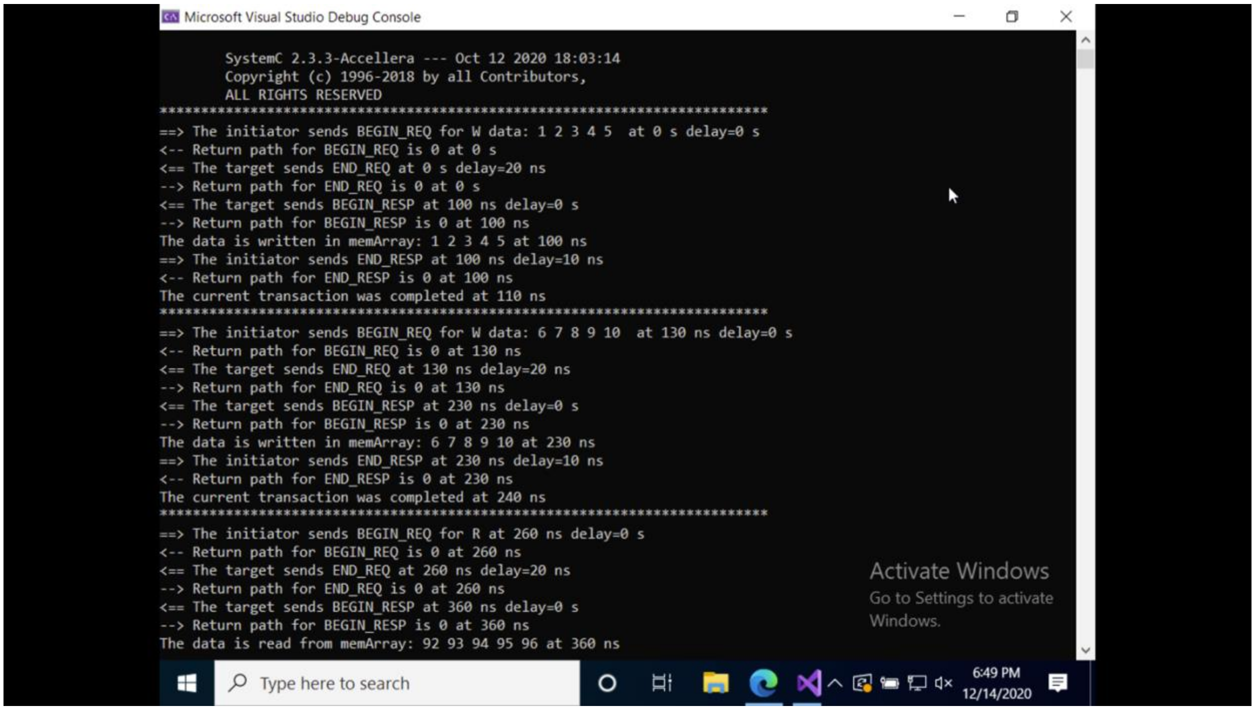
Activate Windows
Go to Settings to activate Windows.

```
        SystemC 2.3.3-Accellera --- Oct 12 2020 18:03:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
Data: (00000000) was transmitted at: 3 ns
Data: (00000001) was transmitted at: 6 ns
Data: (00000001) was received at: 7 ns
Data: (00000010) was transmitted at: 9 ns
Data: (00000011) was transmitted at: 12 ns
Data: (00000011) was received at: 14 ns
Data: (00000100) was transmitted at: 15 ns
Data: (00000101) was transmitted at: 18 ns
Data: (00000101) was received at: 21 ns
Data: (00000110) was transmitted at: 21 ns
Data: (00000111) was transmitted at: 24 ns
Data: (00001000) was lost transmitting at: 27 ns
Data: (00000111) was received at: 28 ns
Data: (00001001) was transmitted at: 30 ns
Data: (00001010) was lost transmitting at: 33 ns
Data: (00001001) was received at: 35 ns
Data: (00001011) was transmitted at: 36 ns
Data: (00001100) was lost transmitting at: 39 ns
Data: (00001011) was received at: 42 ns
Data: (00001101) was transmitted at: 42 ns
Data: (00001110) was lost transmitting at: 45 ns
Data: (00001111) was lost transmitting at: 48 ns
Data: (00001101) was received at: 49 ns
Data: (00010000) was transmitted at: 51 ns
Data: (00010001) was lost transmitting at: 54 ns
Data: (00010000) was received at: 56 ns
Data: (00010010) was transmitted at: 57 ns
Data: (00010011) was lost transmitting at: 60 ns
Data: (00010010) was received at: 63 ns
Data: (00010100) was transmitted at: 63 ns
Data: (00010101) was lost transmitting at: 66 ns
Data: (00010110) was lost transmitting at: 69 ns
Data: (00010100) was received at: 70 ns
Data: (00010111) was transmitted at: 72 ns
Data: (00011000) was lost transmitting at: 75 ns
Data: (00010111) was received at: 77 ns
Data: (00011001) was transmitted at: 78 ns
Data: (00011010) was lost transmitting at: 81 ns
Data: (00011001) was received at: 84 ns
Data: (00000110) was received at: 91 ns
Data: (00000100) was received at: 98 ns
Data: (00000010) was received at: 105 ns
Data: (00000000) was received at: 112 ns
```

Figure 2. This test results proves that this buffer has non-blocking channel. When data is failed to transmit or receive, it showed output error, such as at 27ns, 33ns, and 39ns.

3.

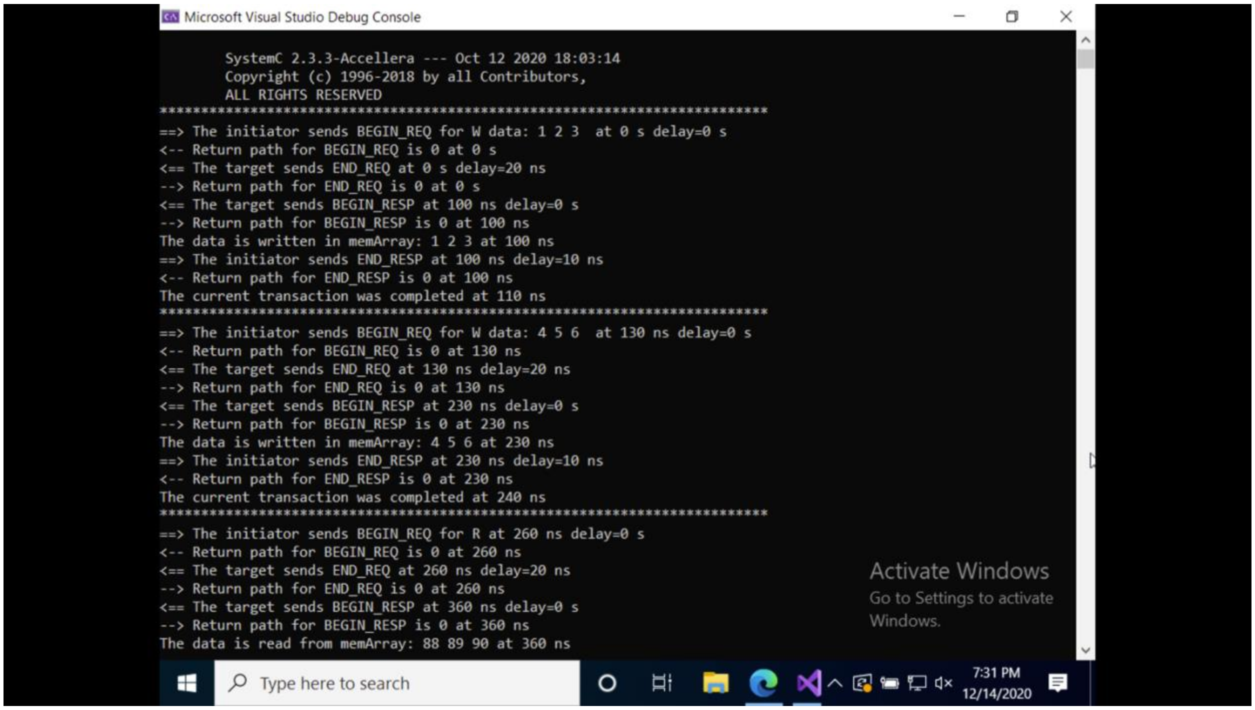
a. For read or write, the imitator send (forward) begin request, then goes into target, the target issues TLM accept (return path), the initiator now knows that the TLM has been accepted, and knows that the initiator wants to read or write.
   After a delay, the target send (backward) end request to the initiator, because the target has completely seen a request and understood the request so the target is ready to go to the next step. Then the initiator issues the accept (return path).
   After a delay, the target issues (backward) begin respond and the initiator issues accept from the return path to notify the target that the initiator is ready to read or write data.
   After a delay, the data will be either read or written after the begin respond.
   After a delay, and after the data has been successfully transmitted, the initiator will issue (forward) end respond to the target. The target will issue (backward) accepted to indicate that this transaction is completed. Shown in figure 3.


Microsoft Visual Studio Debug Console
Activate Windows
Go to Settings to activate Windows.
Type here to search
6:49 PM
12/14/2020

```
        SystemC 2.3.3-Accellera --- Oct 12 2020 18:03:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
***************************************************************************
==> The initiator sends BEGIN_REQ for W data: 1 2 3 4 5  at 0 s delay=0 s
<-- Return path for BEGIN_REQ is 0 at 0 s
<== The target sends END_REQ at 0 s delay=20 ns
--> Return path for END_REQ is 0 at 0 s
<== The target sends BEGIN_RESP at 100 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 100 ns
The data is written in memArray: 1 2 3 4 5 at 100 ns
==> The initiator sends END_RESP at 100 ns delay=10 ns
<-- Return path for END_RESP is 0 at 100 ns
The current transaction was completed at 110 ns
***************************************************************************
==> The initiator sends BEGIN_REQ for W data: 6 7 8 9 10  at 130 ns delay=0 s
<-- Return path for BEGIN_REQ is 0 at 130 ns
<== The target sends END_REQ at 130 ns delay=20 ns
--> Return path for END_REQ is 0 at 130 ns
<== The target sends BEGIN_RESP at 230 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 230 ns
The data is written in memArray: 6 7 8 9 10 at 230 ns
==> The initiator sends END_RESP at 230 ns delay=10 ns
<-- Return path for END_RESP is 0 at 230 ns
The current transaction was completed at 240 ns
***************************************************************************
==> The initiator sends BEGIN_REQ for R at 260 ns delay=0 s
<-- Return path for BEGIN_REQ is 0 at 260 ns
<== The target sends END_REQ at 260 ns delay=20 ns
--> Return path for END_REQ is 0 at 260 ns
<== The target sends BEGIN_RESP at 360 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 360 ns
The data is read from memArray: 92 93 94 95 96 at 360 ns
```

Figure 3. Result of example 6 (nbFwdBwdMemoryRW: 4 calls) in the lecture series on TLM implements a non-blocking communication using 4 calls.

b. Set generic payload attributes to transfer 3 streams of data (instead of 5) in each transaction transportation between initiator and target, shown in Figure 4.

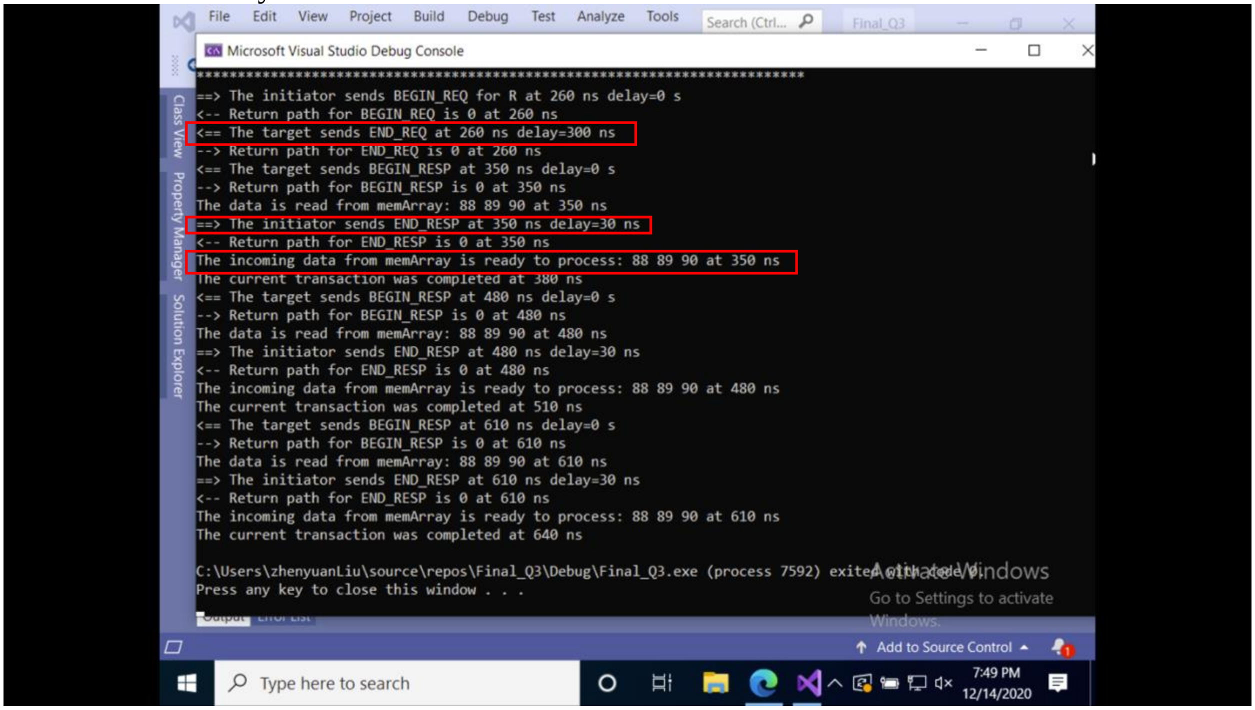


Microsoft Visual Studio Debug Console
Activate Windows
Go to Settings to activate Windows.
Type here to search
7:31 PM
12/14/2020

```
        SystemC 2.3.3-Accellera --- Oct 12 2020 18:03:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
**************************************************************************
==> The initiator sends BEGIN_REQ for W data: 1 2 3  at 0 s delay=0 s
<-- Return path for BEGIN_REQ is 0 at 0 s
<== The target sends END_REQ at 0 s delay=20 ns
--> Return path for END_REQ is 0 at 0 s
<== The target sends BEGIN_RESP at 100 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 100 ns
The data is written in memArray: 1 2 3 at 100 ns
==> The initiator sends END_RESP at 100 ns delay=10 ns
<-- Return path for END_RESP is 0 at 100 ns
The current transaction was completed at 110 ns
**************************************************************************
==> The initiator sends BEGIN_REQ for W data: 4 5 6  at 130 ns delay=0 s
<-- Return path for BEGIN_REQ is 0 at 130 ns
<== The target sends END_REQ at 130 ns delay=20 ns
--> Return path for END_REQ is 0 at 130 ns
<== The target sends BEGIN_RESP at 230 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 230 ns
The data is written in memArray: 4 5 6 at 230 ns
==> The initiator sends END_RESP at 230 ns delay=10 ns
<-- Return path for END_RESP is 0 at 230 ns
The current transaction was completed at 240 ns
**************************************************************************
==> The initiator sends BEGIN_REQ for R at 260 ns delay=0 s
<-- Return path for BEGIN_REQ is 0 at 260 ns
<== The target sends END_REQ at 260 ns delay=20 ns
--> Return path for END_REQ is 0 at 260 ns
<== The target sends BEGIN_RESP at 360 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 360 ns
The data is read from memArray: 88 89 90 at 360 ns
```

Figure 4. Result of nbFwdBwdMemoryRW (4 calls) which changed the generic payload to transfer 3 streams of data instead of 5 streams of data.

c. The code is implemented.
d. The timing has changed.
e. Analyze the new result.



```
**************************************************************************
==> The initiator sends BEGIN_REQ for R at 260 ns delay=0 s
<-- Return path for BEGIN_REQ is 0 at 260 ns
<== The target sends END_REQ at 260 ns delay=300 ns
--> Return path for END_REQ is 0 at 260 ns
<== The target sends BEGIN_RESP at 350 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 350 ns
The data is read from memArray: 88 89 90 at 350 ns
==> The initiator sends END_RESP at 350 ns delay=30 ns
<-- Return path for END_RESP is 0 at 350 ns
The incoming data from memArray is ready to process: 88 89 90 at 350 ns
The current transaction was completed at 380 ns
<== The target sends BEGIN_RESP at 480 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 480 ns
The data is read from memArray: 88 89 90 at 480 ns
==> The initiator sends END_RESP at 480 ns delay=30 ns
<-- Return path for END_RESP is 0 at 480 ns
The incoming data from memArray is ready to process: 88 89 90 at 480 ns
The current transaction was completed at 510 ns
<== The target sends BEGIN_RESP at 610 ns delay=0 s
--> Return path for BEGIN_RESP is 0 at 610 ns
The data is read from memArray: 88 89 90 at 610 ns
==> The initiator sends END_RESP at 610 ns delay=30 ns
<-- Return path for END_RESP is 0 at 610 ns
The incoming data from memArray is ready to process: 88 89 90 at 610 ns
The current transaction was completed at 640 ns

C:\Users\zhenyuanLiu\source\repos\Final_Q3\Debug\Final_Q3.exe (process 7592) exited with code 0.
Press any key to close this window . . .
```

Figure 5. The first red box proved that the latency of target is 300ns. The second red box proved that the response accept delay is 30ns. The third red box proved that the request accept delay is 50ns. The code worked perfectly without the PEQ.