# ECE 5723 Homework 5

By Bruce Huynh

11/11/2021

**Introduction:**

In this assignment we are using abstract communication channels to facilitate and simulate the handshaking of signals between modules. Specifically, we are using it to help the communication of one transmitter that connects to four receivers.

**SystemC Description:**

The description can be found in the .zip file. However, I will mention a few significant parts here. The code was made using bits of both the FiFoPutGet codes and Multiway SharedBus codes. The main changes that had to be made were that we needed each *head* and *elems* variables to be arrays so that they are separate from other targets. For example:

```
queueContents[target][(head[target] + elems[target]) % size] = data;
```

The description of the channels, delays, transmitter, and receivers are all done with templates. This is so the user has free reign to manipulate the size and type of each object. Examples:

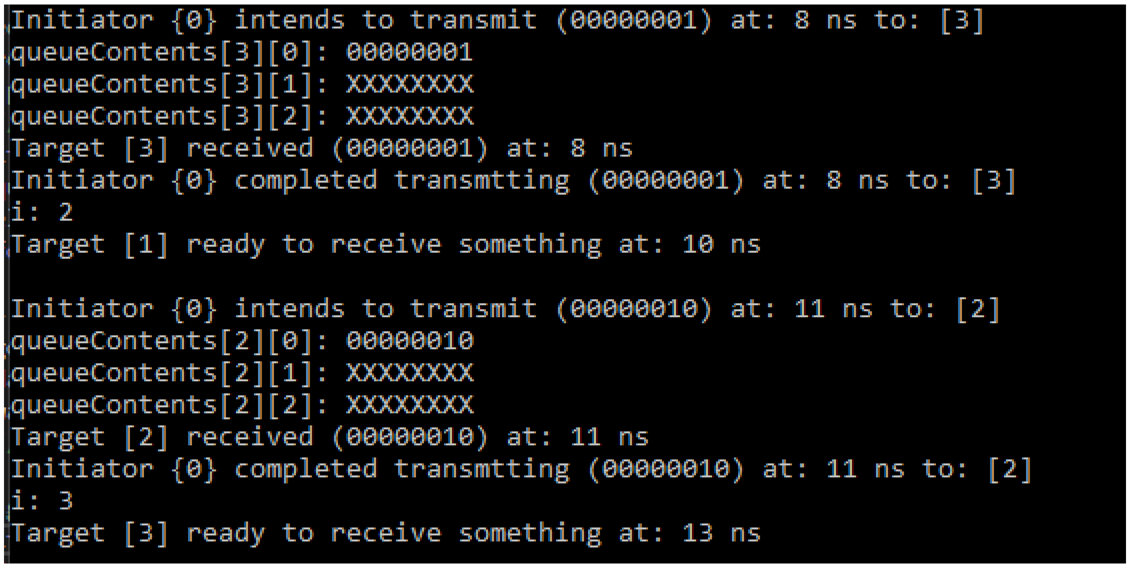
```
template <int N, int F>
SC_MODULE (initiators) {
```

```
template <int N, int F>
SC_MODULE(targets) {
```

```
template <class T, int numOfInitiators, int numOfTargets, int Max>
class sharedFIFO : public put_if<T>, public get_if<T>
{
```

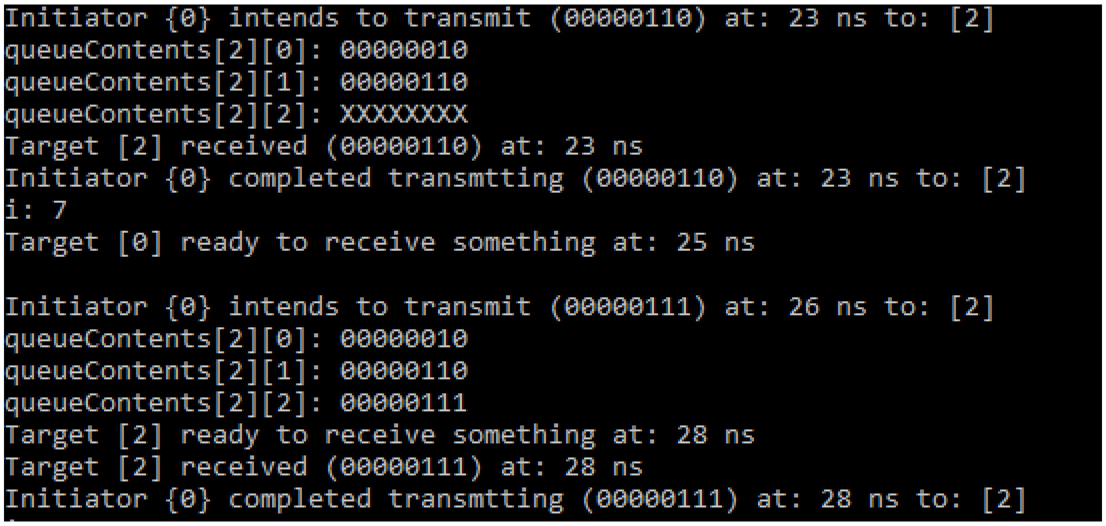
**Testbench:**

In this testbench, we are mainly going to be following target 2 for the sake of simplicity but we will also observe how the transmitter only transmits data to its specified target as well. In my testbench, each target has an array of three sc_lv<8>.



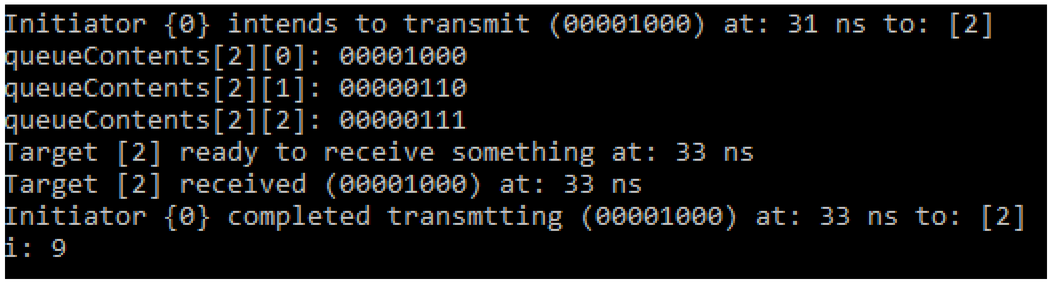
```
Initiator {0} intends to transmit (00000001) at: 8 ns to: [3]
queueContents[3][0]: 00000001
queueContents[3][1]: XXXXXXXX
queueContents[3][2]: XXXXXXXX
Target [3] received (00000001) at: 8 ns
Initiator {0} completed transmtting (00000001) at: 8 ns to: [3]
i: 2
Target [1] ready to receive something at: 10 ns

Initiator {0} intends to transmit (00000010) at: 11 ns to: [2]
queueContents[2][0]: 00000010
queueContents[2][1]: XXXXXXXX
queueContents[2][2]: XXXXXXXX
Target [2] received (00000010) at: 11 ns
Initiator {0} completed transmtting (00000010) at: 11 ns to: [2]
i: 3
Target [3] ready to receive something at: 13 ns
```

In this screenshot, we can see that the transmitter can transmit data to two different targets without it affecting one another.



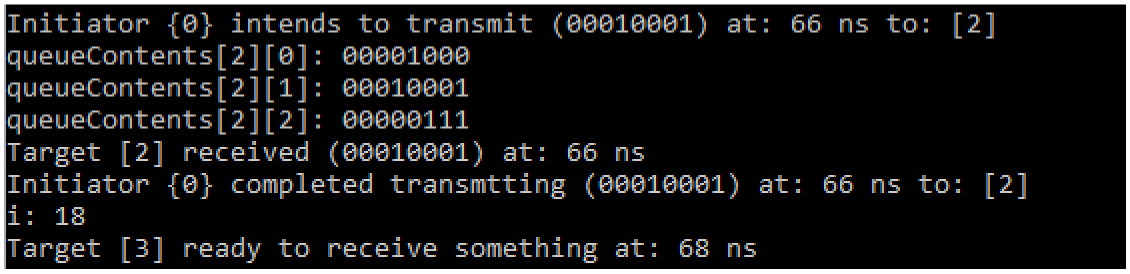
```
Initiator {0} intends to transmit (00000110) at: 23 ns to: [2]
queueContents[2][0]: 00000010
queueContents[2][1]: 00000110
queueContents[2][2]: XXXXXXXX
Target [2] received (00000110) at: 23 ns
Initiator {0} completed transmtting (00000110) at: 23 ns to: [2]
i: 7
Target [0] ready to receive something at: 25 ns

Initiator {0} intends to transmit (00000111) at: 26 ns to: [2]
queueContents[2][0]: 00000010
queueContents[2][1]: 00000110
queueContents[2][2]: 00000111
Target [2] ready to receive something at: 28 ns
Target [2] received (00000111) at: 28 ns
Initiator {0} completed transmtting (00000111) at: 28 ns to: [2]
```

This next screenshot shows that the FIFO keeps adding the new data after the first one.



```
Initiator {0} intends to transmit (00001000) at: 31 ns to: [2]
queueContents[2][0]: 00001000
queueContents[2][1]: 00000110
queueContents[2][2]: 00000111
Target [2] ready to receive something at: 33 ns
Target [2] received (00001000) at: 33 ns
Initiator {0} completed transmtting (00001000) at: 33 ns to: [2]
i: 9
```

Since “00000010” was the first data in, it is the first data to get out. It gets replaced with the new data which is “00001000” which confirms the functionality of our FIFO. But just to be sure, we test it again.



```
Initiator {0} intends to transmit (00010001) at: 66 ns to: [2]
queueContents[2][0]: 00001000
queueContents[2][1]: 00010001
queueContents[2][2]: 00000111
Target [2] received (00010001) at: 66 ns
Initiator {0} completed transmtting (00010001) at: 66 ns to: [2]
i: 18
Target [3] ready to receive something at: 68 ns
```

Since “00000110” is the next oldest piece of data, it gets moved out and replaced by “00010001”.

**Conclusion:**

This homework assignment gave us a glimpse into abstract channels which will allow us to better simulate and view handshaking between modules. We will use these techniques in the future to aid with further system-level design and simulation.