INTRO

Router

SpaceWire Link

-Host

SpaceWire Node

-RX

-TX

-FSM

RESULTS

Conclusions

Router

The router contains 4 hosts, a finite state machine and 16 MUXs, the FSM takes in data from the hosts and determines which inputs and outputs go to the MUXs. This is done by using the signals generated by the host. When the host receives data, it sends a request signal to the router FSM and if no other host is sending data then the request signal is processed, and the host can send data. The FSM looks at the data that is in the Receiver buffer register and that first byte of data determines which host the data is going to. If the lowest 2 bits are “00”, “01”, “10” or “11” then it will send the data to the first, second, third or fourth host respectively. Once this is done the signals are sent to each MUX within the router. These signals consist of the address of the host that is sending data and the host that is receiving the data. The signals that are going through the MUXs are signals that are generated by the hosts which communicate when data is ready to be grabbed and the bytes of data that are being sent between the hosts.

FSM(Router)

Only one connection can be established at a time, so a priority is in place that give the highest priority to the first host then the second then third and finally the fourth host. When a request is received from a host then the FSM checks to see if the router is currently transmitting data, if not then a connection is established. If there is a connection already, then the router ignores the request until the current transmission has ended. This is determined when the 9th data bit in the transmitting hosts Receiver Buffer Register starts with a ‘1’. The ‘1’ signals that the transmission is complete, and the connection can be ended. This is done by putting the FSM into the End\_Trans state. Here all signals that are used are set back to zero and then the FSM goes back to waiting for the next host to request connection.

After the request is received, then the FSM moves into the Address state, here a “busy” signal is set to ‘1’ to indicated that the router is in the process of sending information. Then the FSM checks the signal “inuse(3:0)” to see which host is sending the data. If inuse(0) is equal to one then the Data from the first host is checked to see where the data is going. This is done for inuse(1) and so on. In this state, all the signals for the MUXs are created. There are only two main types of signals, the first is the address of the sending host and the second is the address of the receiving host. After the address are setup then the FSM goes back to the Wait\_for state where it allows data to be sent back and forth between the hosts until connection is terminated.

Host

Each host consists of a Receiver FSM, Transmitter FSM and two registers, one for the incoming data and one for outgoing data, to store the data between links. The registers each store 9 bits of data and are told when to load by the two FSMs in the host.

The Receiver FSM consists of 3 states. The default state is the “Buffer Ready” state where is waits until “bufferwrite” is high then it goes to the “ldData” state. In this state the Receiver Buffer Register is loaded with the data from the Receiver in the SpaceWire link. It also sends out a signal “ldTx” which tells the other host it is communicating with that the data is ready to be loaded. The last signal in this state is the “req” signal which request transmission to be setup between the two hosts. This state changes when the router allows communication and sends the “established” signal. When that signal is received then the FSM will go to the “Start” state (needs a better name) where it waits for a “TxDone” signal from the other host. This tell the Receiver FSM that the data was grabbed in the other host and that it can load the next data.

The Transmitter FSM consists of four states. It starts in the Start state where it waits for a “ldTx” signal from the other hosts Receiver. This tells it that it can go to the “LdData” state and load the data onto the Transmitter Buffer Register. It then waits for the “TxReady” signal, from the Transmitter in the SpaceWire link, to go high. This means the Transmitter is looking for the data to grab. It then goes to the “TxWrt” state where it asserts the “TxWrite” signal to the transmitter to say that the data is ready. It then goes to the “TxDone” state where it asserts the “TxDone signal to tell the other host that it is ready for new data.

The Host created here does not consist of the Time-codes storage and transmission throughout the system.

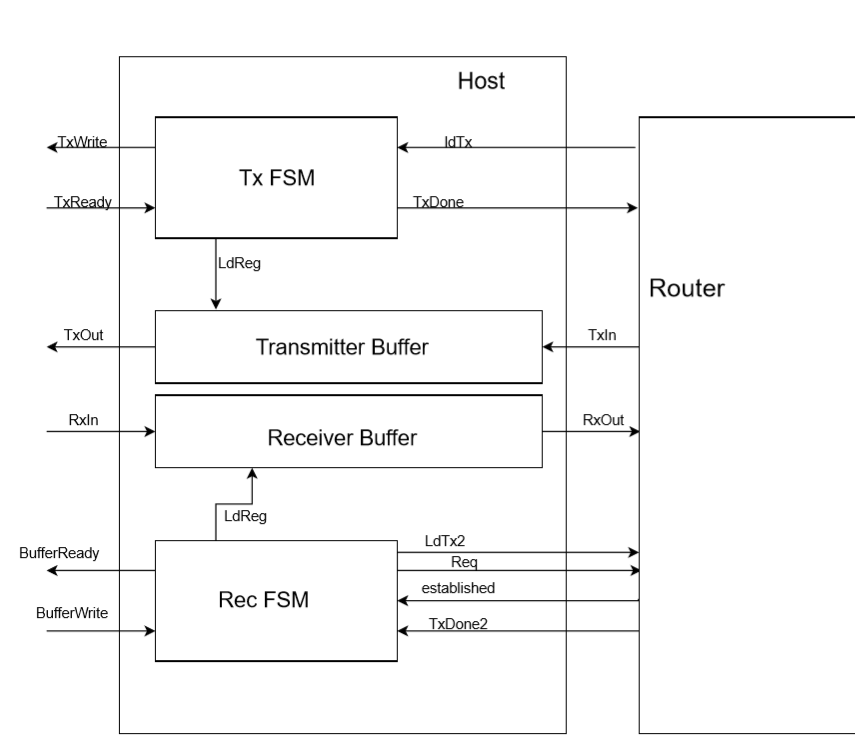
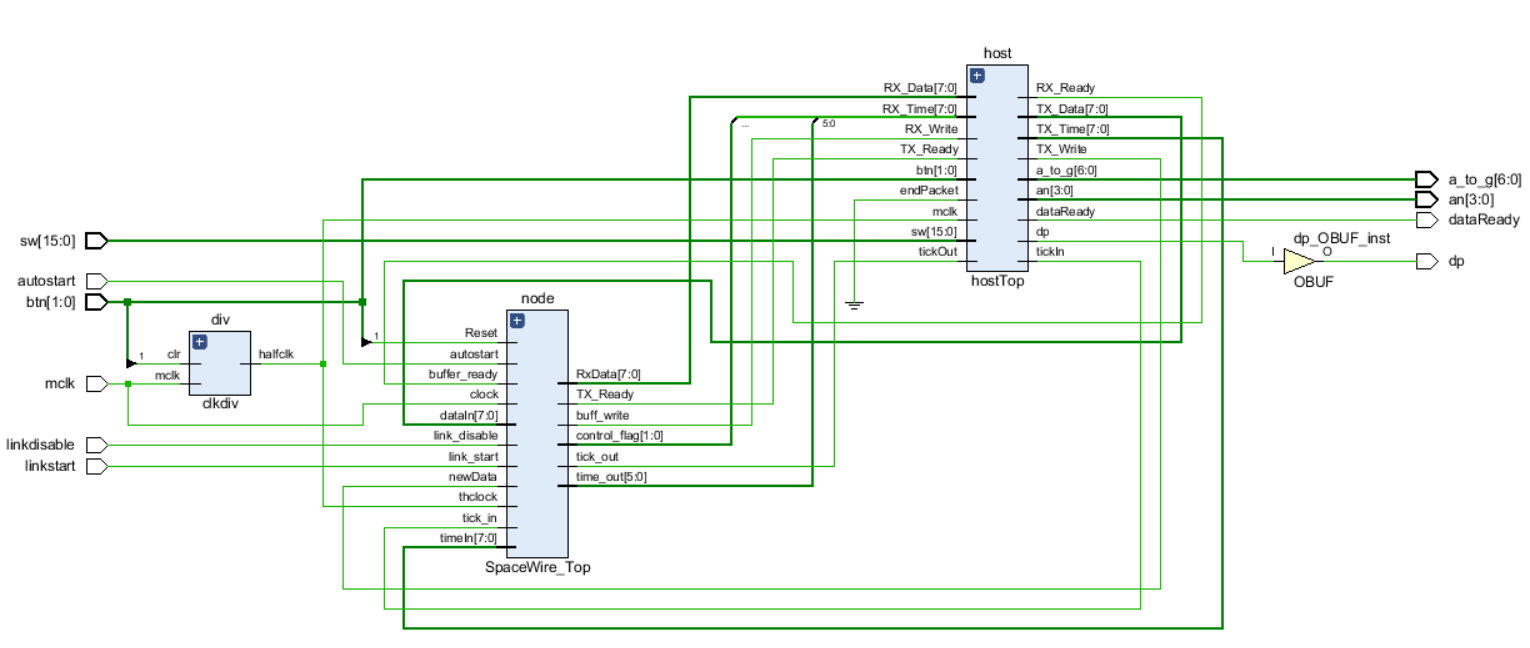


Figure: The Host connected to the router

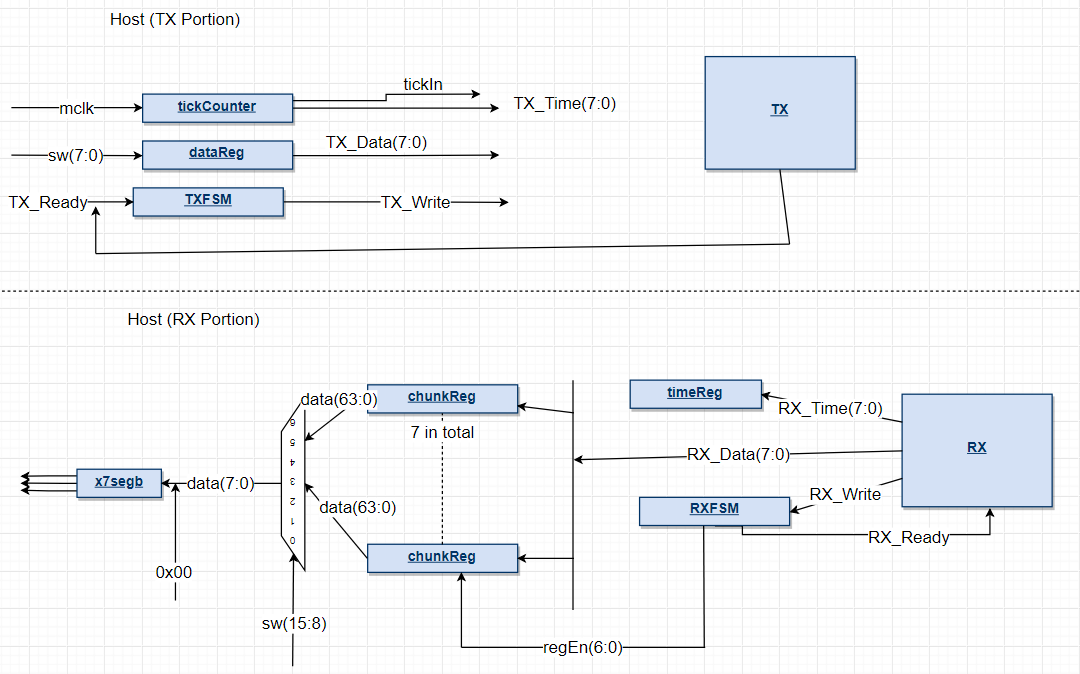
SpaceWire Link



The SpaceWire link consists of two major components. A SpaceWire Node, containing a Receiver and Transmitter, and a Host to stimulate the TX and hold data from the RX. It was found in implementation that a 50 MHz clock was needed to sync communication between the TX and RX so a divider was implemented for the TX and Host. In actual usage, this host would be the system the module is connected to and which has the task of asserting the correct signals to the node.

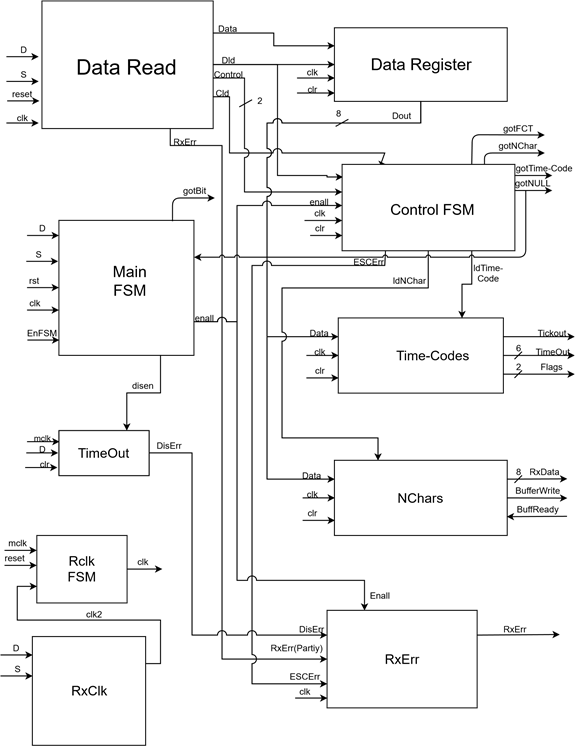
SpaceWire Host

The implementation of the host is that it’s broken into two categories of components. The components either interface with the Transmitter or the Receiver and have their own FSM for both. The job of the TX half is to place data on the line from the buttons, time-codes from a counter, and assert the TX\_Write signal in the proper state when writing. The job of the RX half is to assert RX\_Read when the storage is available, change which register the data is going into, have space available for 56 data char, and project them to the MUX where the user specifies what storage slot they want to view. Using 64-bit wide chunk registers connected to 7 slots of the MUX, the user can select what char they want to display out of the 56 available.



SpaceWire Receiver

The receiver takes the Din and Sin lines to create characters based on the parity bit, control flag, and the remaining two bits (control character) or the remaining eight bits (data character) which is dependent on the control flag, 0 for a data character and 1 for a control character. The receiver creates signals used by the Finite State Machine of the space wire link. It also creates signals that are sent to the host system, RxData and a control flag if an N-Char is received or control-flags and Time out data if a Time-Code is received. The receiver also runs on its own clock cycle which is generated by exclusive oring the Din and Sin lines and creating a clock to be used.



**Data Read**

Shift Registers

The shift registers contain ten registers that load the from the data strobe line every clock cycle. The data shifts over every clock cycle into the next register and the data is lost after the tenth register because by then it has been used and is no longer needed. There are ten registers because the max number of bits that need to be held is ten. This is because when the parity is checked the parity bit, control flag and either the eight data bits or two control characters are read, which is either four bits (P, C, C0, C1) or ten bits (P, C, D0, D1, D2, D3, D4, D5, D6, D7). The values being read are controlled by the finite state machine which will grab the correct data from the registers when needed.

FSM

The finite state machine controls how the data and control characters are grabbed and used throughout the receiver. When the finite state machine is reset if defaults to the beginning state S0. As soon as the reset goes low the FSM starts to look for a change on the data or strobe lines coming into the receiver. Once this has happened the FSM will move to the next state S1 and load the first character which will be the parity bit. Then it immediately goes to the next state S2 to load the control flag which determines if a control character (2 bits) or data character (8 bits) is coming. It the control flag is a ‘0’ then the FSM will go to state S7 if it is a ‘1’ then the FSM will go to state S3. State S7 will enable a counter to count to six and when the counter is done it will enable the state machine to go to state S8 where it will load the last bit of the data character. The FSM will go directly from state S8 to state S9 where it will load the parity bit. It then goes to state S10 where it loads the control flag for the next character. In state S10 the FSM will check the parity of the previous data character, the parity bit and the control flag that was loaded. If the receiver is enabled to look for a parity error, then if the parity has an even number of ones the FSM will move to state S11 which will set RxErr high which will be read by the Finite State Machine of the Space Wire link. If the FSM is not allowed to look for parity errors, then it will check the control character and if it is a ‘1’ then it will go to state S3 and if it is a ‘0’ then it will go back to state S7. Once in state S3 the first control character will be loaded and then go to state S4. Once in this state it will load the second control character and move to state S5 where the parity bit will be loaded. After the parity bit is loaded the FSM will move to state S6 where the FSM will load the control flag and check for a parity error if it can. If it can then it will check the bits of the control character, control flag and the parity bit. If there is an even number of ones, then it will move to state S11. If the number of ones is odd, then it will check the control flag. If it is a ‘0’ it will move to state S7, if it is a ‘1’ then it will move to state S3. The FSM will stay in this loop until it is reset.

*Outputs*

In every state but state S0 the registers are enabled to be loaded with data by the FSM. In state S6 the signal getData2 is set high to indicate that a control character is ready to be read from the third and fourth registers (qt (3:2)). The control characters are retrieved here instead of when they are first loaded two states earlier because the parity must be checked before they are acted upon. When in state S7 the FSM enables a counter to count to “110” to enable the registers to load most of the data character. When in state S10 the FSM sends a signal to an 8 bit register and loads the data character onto it. This is done two states after the data character is first loaded because the parity needs to be checked before the data character is acted upon. In state S11 the FSM is sending a signal that a parity error has occurred and that the link needs to be reset.

Data Register

The data register is loaded with the last eight bits of the shift registers when the Finite State machine from the Data Read component sends the signal to say that the data is ready. This allows the Receiver to hold the eight bits that will be put on either the RxData line or the Time\_Code/Tick\_out lines depending if the data character is part of an N-Char or Time-Code. It is saved on the register because if it were to go directly to the Time-Codes component or N-Char component when needed there would be an extra clock cycle causing it to be shifted one more time. The Data Register could be replaced if one more shift register in the Data Read component was added and the last eight bits (10:3) were read.

Timeout

This component reads the Din line and if the value on the line does not change for 850 ns then a disconnect error has occurred. This component runs on the space wire link clock cycle, not the receivers clock cycle. This is looked for when the Receiver has reached or passed the gotBit state.

Shift Registers, Disconnect Error

The Shift register component within the Timeout component contains two shift registers which load and shift the Data value over. On every clock cycle the two values in the registers are compared and if they are the same for 850 ns then the disconnect error component detects it and sends a signal out that the error has occurred.

Control FSM

The Control FSM reads the control character when the FSM from the Data Read component sends the signal that the control charter is loaded. It starts in the Start state where it waits until a control character is ready to be read. This is done by reading the signal from the Data Read component that the control character has been received and is ready. The signals coming into the control FSM are cld, which is the signal that says a control character is ready, dld, which is the signal saying that a data character has been received, enall, which allows the component to look for values other than NULLs, and Control, which is the two bits that make up the control character. When in the start state the FSM looks for the cld or dld to be high. If cld is high, then it will look to see if enall is also high. If enall is high, then it will look to see what the value of Control is. If Control is “00” then an FCT has been received and the FSM moves to the FCT state. If Control is “11” then an ESC has been received and the FSM moves to the ESC state. If enall is low, then only an ESC can be read because the Control FSM can only look for NULLs (ESC followed by an FCT). If while in the start state and enall is high, and dld is high then an N-Char has been received and the state machine moves to the NChar state. While in the FCT state the gotFCT signal is sent out of the Receiver and on the next clock cycle the Contorl FSM goes back to the Start state. While in the ESC state if a control character is received then if enall is high, the control character will be checked. If the control character is “11”, “01” or “10” then an invalid sequence has been sent and it moves to the Error state. If enall is not high, then if the control is “00” then the FSM moves to the NULL state. If any other control character is received while enall is low, then the FSM will go back to the start state. If while in the ESC state and dld goes high, and enall is high, then a time-code is received, and the FSM goes to the TimeCode state. If both cld and dld are low while in the ESC state, then it will stay in the ESC state until one of the signals go high. While in the Error state an escape error is sent out of the Receiver which will cause the Receiver to get reset. While in the NULLs state the signal gotNULL is sent out of the Receiver, also a signal is sent to the Main FSM of the receiver to signal that a null was received. While in the TimeCode state the FSM sends a signal that enables the Time-Codes component and goes back to the Start state on the next clock cycle. While in the NChar state a signal is sent to enable the NChar component, the FSM goes back to the start state on the next clock cycle.

Main FSM

The main FSM of the Receiver had four states, the Reset state, Enable state, GotBit state and gotNULL state. When the receiver is reset it will move to the Reset state. While in the reset state the FSM will wait for the enable signal to go high and when this happens it will move to the Enable state. While in the enable state, a signal, disen, will be set to high which enables the Receiver to look for a disconnect error. When the Din or Sin lines change the FSM will move to the GotBit state. While in the GotBit state, the control FSM is looking for a NULL, when one is received the Main FSM will receive a signal from the control FSM to signal that a NULL had been received and will move to the gotNULL state. While in the GotNULL state, the enall signal is sent out to allow the control FSM to look for something other than NULLs and allow parity errors and ESC errors to occur.

RxClk and ClkFSM

The RxClk takes the Din and Sin lines and exclusive ors the values together to create a signal that is sent to the ClkFSM which detects the rising and falling edges of the RxClk to create a clock signal that is used by the rest of the receiver. The clkFSM detects the rising and falling edges of the RxClk by using the main clock that is used by the space wire link. It has four states, zero, rising, one, and falling. When the component is reset it will move to the zero state because no data will be entering the receiver so the RxClk will be zero. While in the zero state if the RxClk is high then it will move to the rising state but if the signal is low then it will stay in the zero state. When in the rising state the FSM will move immediately to the one state. When in the one state if the RxClk is low then it will move to the falling state or else it will stay in the one state. When in the falling state it will immediately move to the zero state. While in the rising or falling state, the clock used by the receiver will be created from the FSM. This means that the receiver clock can only run at half the speed of the rest of the space wire link.

RxErr

The RxErr component takes in the escape error, disconnect Error and parity error and turns them into a RxErr when needed. While the main FSM is in the GotBit and GotNULL states the disconnect error is read and if it goes high, then RxErr is sent out of the receiver. While in the GotNULL state if a parity error or Escape error is received the the RxErr is sent out of the receiver.

NChar

The Nchar component takes in a data character and sends it out of the receiver when ready. When the receiver is cleared the RxData line is set to 0, the BuffWrite and Credit Error is set to 0 as well. On the rising edge of the clock if the component is told that an NChar has been received it will check to see if the host system sent a signal that it is ready for an NChar. If the host system did not say it was ready, then a credit error has been received and is sent out of the receiver. If the host system is ready, then the data character stored on the data register will be put on the RxData line and the BuffReady signal will be asserted to tell the host system that the data is ready. If no NChar has been received, then the buffer write signal is kept low and the RxData line is set to zero.

Time-Code

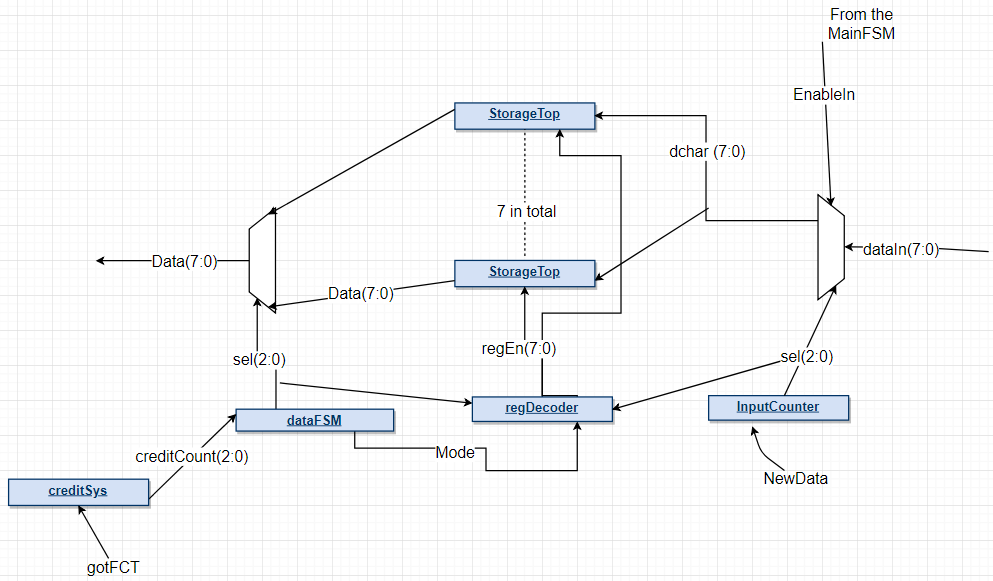
The time-code component occurs when a data character is received after an escape control character. When the time-code is reset the tick out signal is set to low, and the time out and flags lines are set to zeros. On the clock edge if the control FSM detects a time-code then if tells the time-codes component, when this happens the bottom six bits are put on the time out lines and the top two bits are put on the flag lines. Then the tick out signal is asserted to tell the host system a time code has been received. If no time code has been received, then the output lines are kept at zero.

SpaceWire Transmitter

Following is a breakdown of the different components in the Transmitter and their functions.

First are some assumptions made about the system. First is that the data from the host system will come in parallel format, 8 bits at a time in the format LSB-MSB. Otherwise, a bit reversal would have to be implemented because the receiver expects this format. These are held in the storage system and then a ‘0’ is appended to the leftmost bit.

Storage Top – The storage system is made up of 7 storage blocks connected to a MUX and deMUX and sequenced by separate counters. This FSM also controls how many words are transmitted based on numbers of FCTs received (Up to 7 bundles of 8 words, 56 words at maximum). Inside each storage block is a chunk shift register that groups 8 bits together and shifts them. The regDecoder sends enables that allows this system to shift in data and send it out.

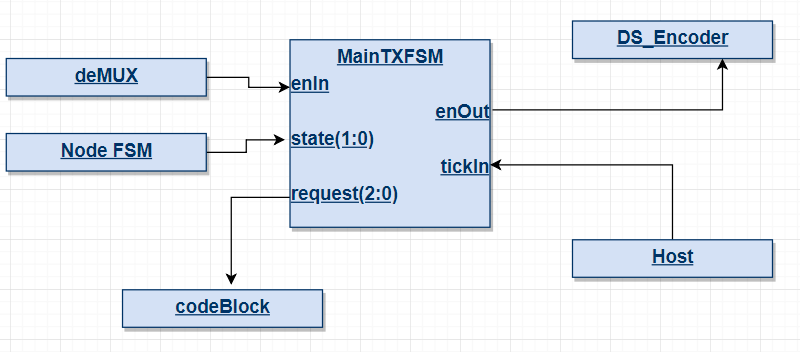


Input counter – This is a counter dependent on the DataReady signal coming in with a fresh set of bits. The new data assertions will increment to 8 to fill each storage block and then switches the deMUX to start filling the next block. There is also an enabling signal for the output MUX counter to prevent 8 words from being reserved by FCT when there aren’t enough available. This enable goes high after the first block is filled.

dataFSM- A FSM that begins on the fctReady signal from the deMUX counter and controls what outputs are leaving the storage blocks. It dictates whether the system is reading or writing based on the timing between new data availability. If new data hasn’t been asserted after 8 cycles, the system flips into writing mode and the block acts on the FCTs its counted. This FSM also regulates the register decoder that enables and disables the storage blocks.

regDecoder- Takes in the selects from the outFSM and the inputCounter and outputs based on the mode of operation. The decoded output will enable and disable the chunk registers to move and output data.

creditSys- Counts the number of FCTs received and translates to how many data words are desired from the storage system. These credits are used by the dataFSM to coordinate the outputs.



Main FSM – The central transmitter FSM has the job of enabling and resetting certain portions of the transmitter based on the current state provided by the FSM of the whole system. These states in order are

* Clearing the whole system and not allowing any communication
* Having NULLS be generated and sent to the receiver
* Allowing FCTs to come in from the receiver and incrementing counter based on number received. This will influence the storage system and how many blocks it is allowed to unload.
* Enabling the storage system and inputs from the host system. Will have to support both data characters and control characters being generated at the same time. Provide signal to MUX to shift between these two types of signals.
  + Sub-state is when Tick\_In is asserted by the system FSM. An ESC will have be generated followed by a Time-Code. This has priority over all other characters, so response has to be immediate.

Control Character Block – Based on inputs from FSM, block creates 2-bit codes that are appended with 6 zeroes to the right (or left, determined by convenience) and a ‘1’ on the leftmost bit to create a 9-bit signal. The requests come from the main FSM and are

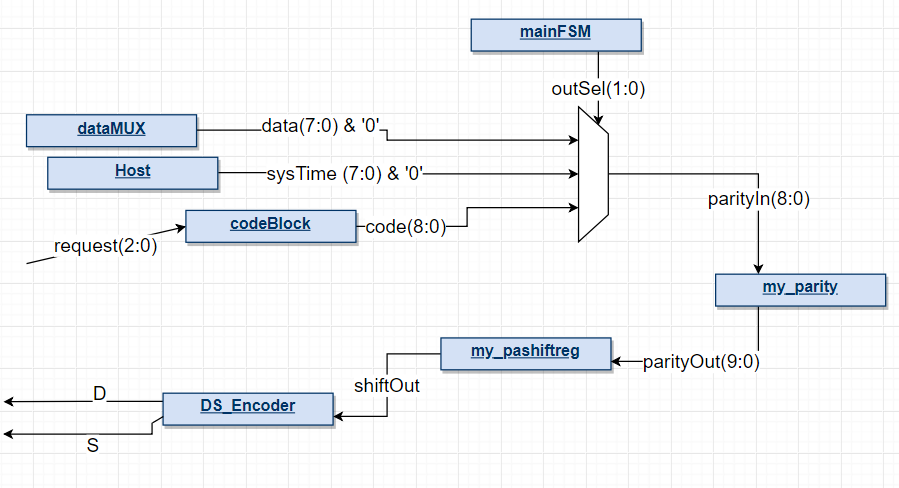
001-Flow Control Token => “00”+000000

010-End of Packet => “01”+000000

011-Error End of Packet => “10”+000000

100-ESC Code =>“11”+0000000

101-NULL (FCT+ESC) =>”110100”+00



Parity block – Works to add either a parity bit to the left-most bit of words based on the control bit of the current character (1 for control characters, 0 for data characters) and the 8 bit data or control bits + “00” content of the previous character. The protocol asks for odd-parity in this 10 bit long zone. The implemented solution is a register holding the two most recent characters and creating logic based around on their content and type.

10 to 1 Register – Shift register that grabs data every 10 cycles and shifts one out at a time. The shift is left and the first three bits transmitted should be Parity, Control Bit, and then data LSB. A counter would anticipate new data and allow all bits to be transmitted before grabbing new data. Some arguments are used to determine if 10 bits needed to be transferred (data words), 4 bits (FCTs/Control Char) or 6 bits (NULL).

DS Encoding – The D signal output is a wire from the serial register. The Strobe signal is generated by an internal FSM and depends on the 2 most recent D signals.

Finite State Machine of the SpaceWire Link

*Timer*

The Timer consists of four inputs and 2 outputs and it is designed to run on a 100 MHz clock cycle. The timer has two different timers in it, one is a 6.4 us timer and the other is a 12.8 us timer. The 6.4 us timer starts when the “E” input is active high. A signal count and a signal wrap are used in the if statement that increments the count value each clock cycle. The count value goes up to 640 which is “1010000000” in binary. When 640 is reached the wrap, value is set to high and count is kept at 640 until the timer is reset. The wrap signal is attached to the clk\_out output which is connected to the FSM. A signal count2 and wrap2 is used for the 12.8 us timer. When the input “En” is set high then the 12.8 us timer is started. On each clock cycle the count2 value is incremented until it has reached 1280 which is “1010000000” in binary. When this value is reached the wrap, value is set to high and the count value is set at 1280 to keep the value high. The wrap2 value is then set high and set equal to the clk\_out2 value which connects to the FSM. If the timer is reset, then both count values and wrap values are set back to zero.

*Tick too Pulse*

The TickTPulse waits for a gotNULL to be sent from the receiver and changes the one clock cycle tick into a pulse signal that is sent to the FSM. This is done because the FSM looks for a gotNULL signal from state 2 to state 4. The receiver will only send a single clock tick value for gotNULL when it is received so the TickTPulse will set the value high whenever the receiver sends just one tick. The TickTPulse is reset whenever the FSM is in the first state. This is done by the FSM sending a reset value whenever it is in the Error Reset state.

*Finite State Machine*

The FSM uses six states, S1-S6, to go through the processes of the FSM as well as a signal y that corresponds to the state that the FSM is in. The state machine uses an if statement that first checks if the state machine is being reset. If it is then the state machine is reset, then it will go back to the first state which is the Error Reset state and will also reset the timer by setting the resettimer high. If the state machine is not being reset, then on the clock tick up the if statement will check the state it is currently in.

When in the Error Reset state (S1) then it will check to see if the 6.4 us timer has completed. The input timer1, which is the 6.4 us timer, is connected to the clk\_out from the timer. If the timer has completed, then the FSM will reset the timer and then move to the Error Wait State (S2). If the 6.4 us timer has not completed, then is will stay in the Error Reset state.

When in the Error Wait state (S2) then it will first check to see if the FSM is receiving an Rx\_Err, gotFCT, gotTimeCodes or gotNChar and will go back to the first state. If none of those signals are received then it will check to see if timer2, which is the 12.8 us timer, is high. If it is then the timer will be reset and move to the Ready State (S3). If the 12.8 us timer is not complete, then it will stay in the Error Wait state.

When in the Ready State (S3) the resettimer is set low so the timer can be started. It then checks if the FSM is receiving an Rx\_Err, gotFCT, gotTimeCodes or gotNChar, if it is then it will go back to the first state. If the FSM does not receive any of those values, then it will check if the LinkEnable is high. If it is then the FSM will go to the Started State (S4), or else it will stay in the Ready state.

When in the Started State (S4) it will check if the gotNULL value is low, if it is then it will see if the FSM is receiving an Rx\_Err, gotFCT, gotTimeCodes or gotNChar and will go back to the first state. If none of those signals are received then it will check if the timer2 (12.8 us) is high, if it is then it will reset the timer and go back to the Error Reset state. If the timer is not high, then it will stay in the Started state. If gotNULL is not 0 then it will check to see if the FSM is receiving an Rx\_Err, gotFCT, gotTimeCodes or gotNChar and will go back to the first state if it is. If not, then the timer will be reset and move to the Connecting state (S5).

When in the Connecting state it will check to see if the FSM is receiving a gotFCT, if it is then it will go to the Running state (S6). If not then it will check to see if the FSM is receiving an Rx\_Err, gotTimeCodes or gotNChar and will go back to the first state if it is. If not then it will check to see if timer2 (12.8 us) is high, if it is then it will time out and go back to the first state. If the timer2 is not complete, then it will stay in the Connecting state.

If the FSM is in the Running state, then it will stay in the Running state until an Rx\_Err, CreditError or LinkDisable is high.

*Outputs*

In the Error Reset state, the Receiver and Transmitter are both reset and the entimer1 is high which enables the 6.4 us timer.

In the Error Wait state, the transmitter is reset and the receiver is enabled. The entimer2 is high which enables the 12.8 us timer.

In the Ready state the transmitter is reset, and the receiver is enabled.

In the Started state both the transmitter and receiver are enabled. The entimer2 is set high to start the 12.8 us timer and the FSM tells the transmitter to send Nulls.

In the Connecting state both the transmitter and receiver are enabled. The entimer2 is set high to start the 12.8 us timer and the FSM tells the transmitter to send Nulls and FCTs.

In the Running state both the transmitter and receiver are enabled. The FSM tells the transmitter to send nulls, FCTs, NChars and Time Codes.

*Top File*

The top file uses five signals between the timer and FSM. There are two enables (e1 and e2) which connect entimer1 and entimer2 to E and En respectively. There are also two signals that connect the clock signals (t1 and t2) which connects timer1 and timer2 to clk\_out and clk\_out2 respectively. There is also the signal r which connects the resettimer and resetn. Three other signals are used to compute the LinkEnable. Link is equal to (AutoStart and gotNULL), link2 is equal to (linkstart or link), and link3 is equal to (not linkdisable and link2. Then link3 is the signal that is connected to LinkEnable in the portmap for the FSM. The top file also uses two signals between the FSM and the TickTPulse. The first is gotNULL1 which is the gotNULL value that is looked for in states two through four in the FSM. There is also the signal reset1 which connects ResetTick on the FSM and reset on the TickTPulse.

Results:

The simulatiion of the Router with the 4 hosts was never tested with the SpaceWire Link because the Link was never finished so the Router was only tested by attempting to emulate the Links in simulation. The simulation only got as far as getting one host to send data to another and then ending the communication with an EOP marker. It was never tested to see if the router can do continuous communication by one host after another.

When simulating the SpaceWire node, the state of the FSM of the SpaceWire Link is denoted as the “y” below the gotNULL signal to show which state the link is in. After the link is enabled then the Receiver is looking for a Null from the Transmitter. Once this Null is received the FSM is told which can be seen bellow on the gotNULL signal. Once the Null is received then the Receiver is looking for an FCT from the transmitter. For every FCT that is received by the Receiver, a signal is sent to the FSM and Transmitter seen at the top of Figure X. Once the FCT has been received then the FSM goes into the running state where it allows the Reciever and Transmitter to run as intended. In this mode the Transmitter sends Nulls until it is prompted to send different data, the Nulls allow communication to continue. In the Figure you can see that after the FCTs are send two Nulls are send because the transmitter is waiting for the data to be ready. Once the data is ready the Transmitter grabs it and sends the data to the Receiver where it is processed as N-Chars. The N-Char is sent for as long as possible for testing purposes to see the data being sent.

Case 1: Data with Even Number of 1s

The initial link is successful and the Receiver gets consistent data of 0x093. This continues until a Time-Code needs sending and is shifted into the parity block. This change in the pipeline causes the parity of the data char to flip for one cycle. If the shift register happens to grab this inconsistent value and shift it out, it will trigger a Parity Error in the Receiver and end communication.

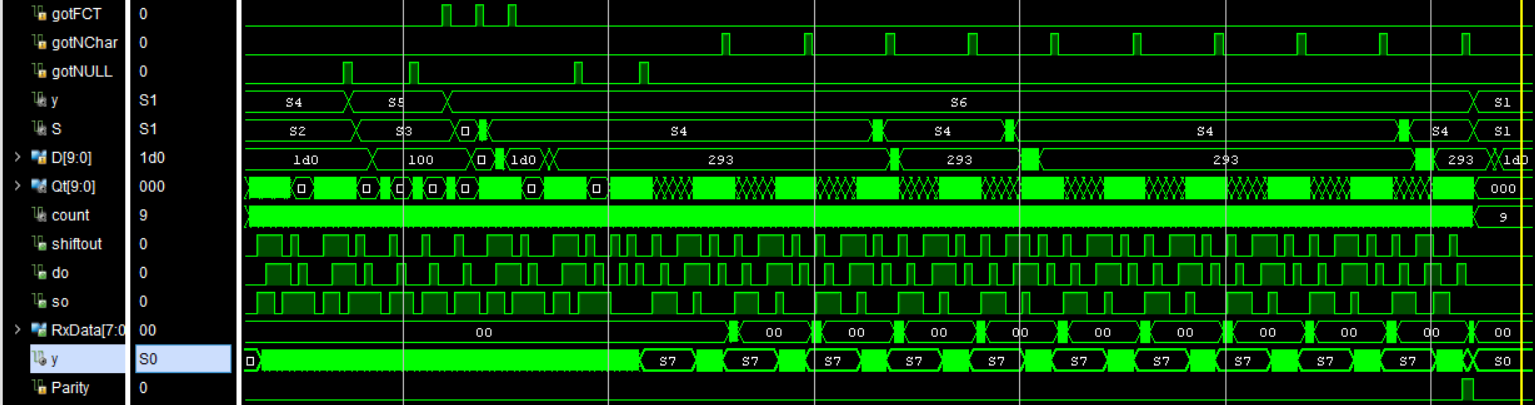


Figure X

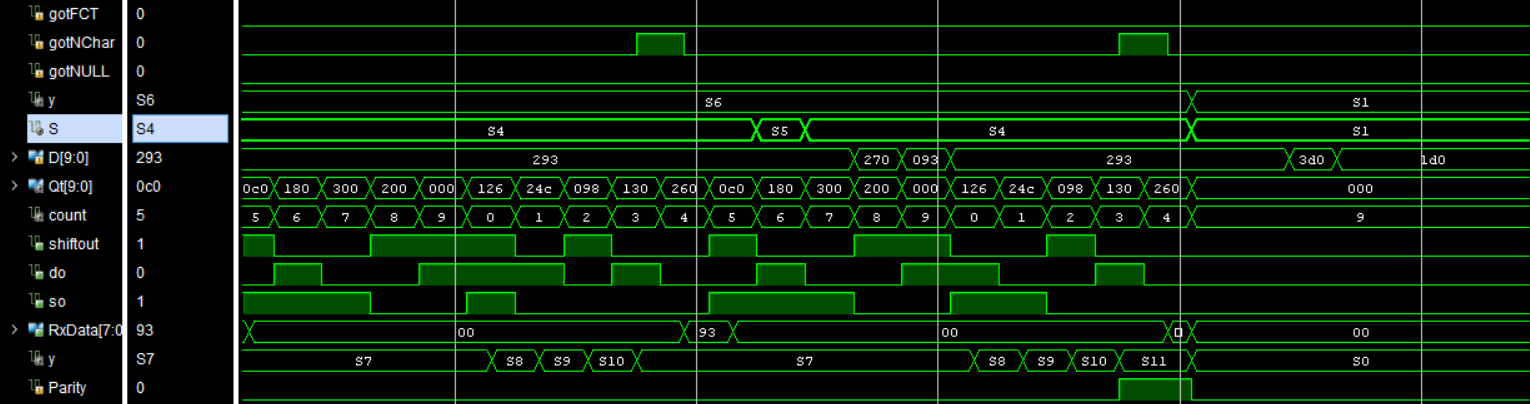
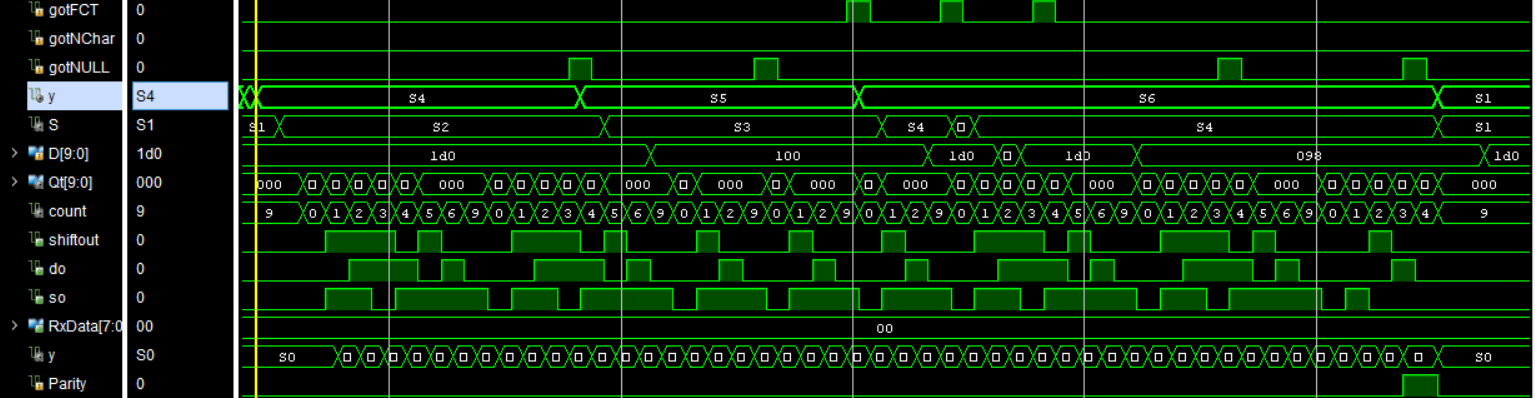


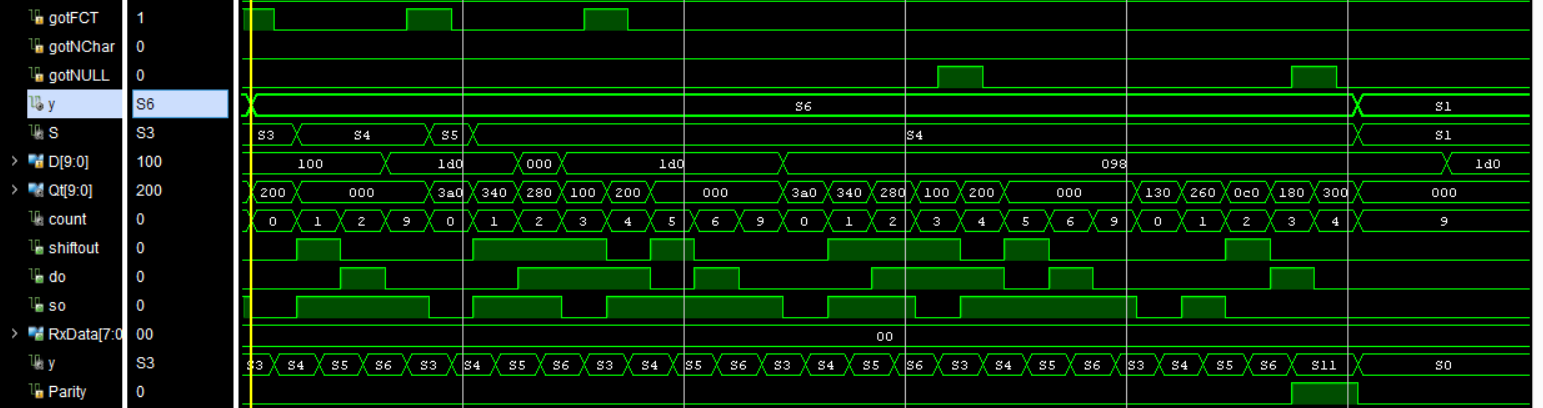
Figure Y

As soon as the Receiver loads in the 0x293 it didn’t expect, it asserts the error signal and prompts communication end.

Case 2: Data with Odd Number of 1s

The initial NULL sends are successful but the first D-Char sent is a data failure. The parity check computes a NULL - Dchar parity of ‘1’ for one cycle which is what's expected by the Receiver. After this, a Dchar-Dchar is computed as ‘0’ for the duration until the communication ends. The shift register never had an opportunity to grab 0x298 for the one cycle it was available, instead sending 0x098 which the Receiver wasn’t expecting.





In these particular snips, a 0x000 had been loaded in place of 0x298 due to some bad timing concerning the Time-Codes but the same process of shifting out 0x098 instead of 0x298 can be observed.

Conclusion

The main item to conclude on is how to improve the system and have it run more efficiently. The first step is solving the issue of the Transmitter not being to store a set of data to be shifted so none is lost. The main goal of this is to capture the needed data that may only last one cycle. Another point of consideration is that the Transmitter and Receiver have to be synced up with their clocks so the data can grabbed correctly. If one is significantly slower then bits could be lost. If the node is tested to have full function then the last step before doing router work is to have a good stimulating host. There’s a lot required of the host so all of the proper functions must be implemented as a minimum.