Project 4

Due March 14, 2018 at 11:59 PM

For this project you will be working with a partner. Note: all members of the group are not guaranteed equal credit. You will be using Logisim for this project. Submit a README.txt, and interactive grading signup with your circuit file compressed together in a tgz (tar gzip) file. Only one member of the group needs to submit the tgz file. Your README file must have your name, SID number, partner's name, partner's SID number, a brief description of what works/doesn't, and a list of sources you used for designing of your circuit (you do not need to list the book or lecture notes it is assumed these have been used). You may use any of the built in components of Logisim. All class projects will be run through MOSS like software to determine if students have excessively collaborated. Excessive collaboration, or failure to list external sources will result in the matter being referred to Student Judicial Affairs

You will be developing a Network Interface Controller (NIC). You will need to complete at least the parallel interface and transmitter portion of the NIC for full credit. The receiver portion will be extra credit. The NIC you will be developing will have a 32-byte FIFO for both receive and transmit. The NIC will have an 8-bit parallel interface to communicate with the micro-processor. The micro-processor side parallel interface includes a $\overline{CS}$, $\overline{O}$, $\overline{W}$, $\overline{INT}$, 2 dedicated address lines ADDR, the 8 data in lines DATAIN, and the 8 data out lines DATAOUT. The other I/O include a CLK, RX, and TX. The NIC transfers data frames from 2 to 32 bytes in length. Each 16-bit word that is transmitted is transferred serially least significant bit first with Single Error Correction Double Error Detection (SECDED) encoding; each word will be 22-bits in length for the additional check bits. The serial transmission is a Non-Return-to-Zero (NRZ) format. Since the NRZ format is used, bit stuffing is implemented whenever there are five consecutive ones or zeros to be transmitted (e.g. 00000000 would be sent as 000001000). Six consecutive ones or zeros received in the middle of a frame is an error. Each data frame has a 22-bit parity word over all data. The sequential communication is initiated with a start of frame sequence of six 1's followed by one 0, 1 to 16 data words, the 22-bit parity word and concluded with a stop of frame sequence of six 0's. Table 1 shows the NIC frame format. When the TX is idle the TX outputs a logic 0.

Table 1. NIC Frame Format (Ignoring Stuff Bits).

Bit Location	Description
0..6	Start Frame (0111111)
7..28	Data Word 0
29.. FS•22+6	Data Word 1-15 (Depends on Frame Size in Words)
FS•22+7..FS•22+28	CRC
FS•22+29.. FS•22+34	Stop Frame (000000)

Registers

Status/Command (Address 0)

The register accessed at address 0 depends upon transaction type (read or write).

	7	6	5	4	3	2	1	0
Read	FE	SDE	OR	NF	TXI	TBNF	DR	ENA
Write	FEC	SDEC	ORC	NFC			TF	ENA

FE – Frame Error, this bit is set if the stop bit is 1 instead of 0, or if there are missing stuff bit. It is cleared upon the disabling of the NIC or if a 1 is written to the FEC.

SDE – SECDED Error, this bit is set if any of the received data or parity words have a double error. It is cleared upon the disabling of the NIC or if a 1 is written to the SDEC.

OR – Overrun Error, this bit is set if the RX FIFO is overrun with a new frame. It is cleared upon the disabling of the NIC or if a 1 is written to the ORC.

NF – Noise Flag, this bit is set if noise is detected on the line. Noise is detected if the center sample of the bit doesn't match the first and last. It is cleared upon the disabling of the NIC or if a 1 is written to the NFC.

TXI – Transmitter Idle, this bit is set if the transmitter is not outputting anything. This bit is cleared when the transmitter is shifting out data.

TBNF – Transmit buffer not full, this bit is set if the transmit buffer is not full. This bit is cleared if when the TX FIFO becomes full.

DR – Data Ready, this bit is set if there is data in the RX FIFO. This bit is cleared if all the data has been read out of the RX FIFO.

TF – Transmit Frame, when a 1 is written to the TF bit the current contents of the TX FIFO are sent as a frame. If no data is in the TX FIFO the TX request is ignored.

ENA – Enable, this bit is set if the NIC is enabled, it is cleared if the NIC is disabled. This bit can be written at any time.

Interrupt Mask (Address 1)

The register accessed at address 1 is the interrupt mask register.

	7	6	5	4	3	2	1	0
Read	FEIM	SDEIM	ORIM	NIM	TXIIM	TBNFIM	DRIM	
Write	FEIM	SDEIM	ORIM	NIM	TXIIM	TBNFIM	DRIM	

FEIM – Frame Error Interrupt Mask, this bit masks the Frame Error. If it is cleared, a frame error will not generate an interrupt.

SDEIM – SECDED Error Interrupt Mask, this bit masks the SECDED Error. If it is cleared, a SECDED error will not generate an interrupt.

ORIM – Overrun Error Interrupt Mask, this bit masks the Overrun Error. If it is cleared, an overrun will not generate an interrupt.

NIM – Noise Interrupt Mask, this bit masks the Noise Flag. If it is cleared, a detection of noise will not generate an interrupt.

TXIIM – Transmitter Idle Interrupt Mask, this bit masks the Transmitter Idle Flag. If it is cleared, the transmitter going idle will not generate an interrupt.

TBNFIM – Transmit Buffer Not Full Interrupt Mask, this bit masks Transmit Buffer Not Full Flag. If it is cleared, the transmit buffer not being full will not generate an interrupt.

DRIM – Data Ready Interrupt Mask, this bit masks the Data Ready Flag. If it is cleared, data being ready in the RX FIFO will not generate an interrupt.

Data (Address 2)

The register access at address 2 depends upon transaction type (read or write). Upon read the next valid data byte will be read from the RX FIFO. Writing to address 2 will place a byte into the TX FIFO unless the FIFO is full.

Baud Rate Divisor (Address 3)

The Baud Rate Divisor is an 8-bit register that is the clock divisor for the serial clock. The Baud Rate Divisor can only be written if the NIC is disabled, and must be between 3 and 15. Values less than three or greater than fifteen should not update the Baud Rate Divisor.

Parallel Interface Timing

The parallel interface is asynchronous to the NIC clock, and each transaction (read or write) is guaranteed to last longer than one clock cycle. The a $\overline{CS}$ line will go low at or prior to the $\overline{O}$ or $\overline{W}$ signals going low. The data must be made available for reads on the following clock edge of the NIC. Figure 1 shows the timing diagram for a read where DATA is the DATAOUT from the NIC, and Figure 2 shows the timing diagram for a write where DATA is the DATAIN to the NIC.

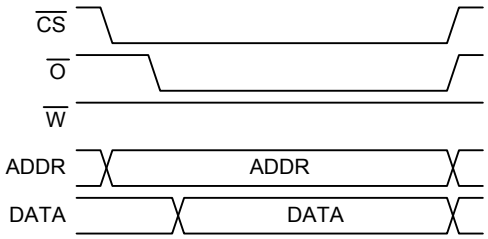

Figure 1. Data Read

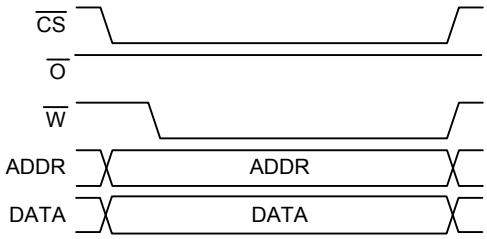

Figure 2. Data Write

Serial Interface

Each serial bit is NRZ and will have a bit stuffed for five bits of the same type in a row. On the receive side bits are sampled three times and the data value is based upon consensus. The sampling times depend upon the Baud Rate Divisor value. Table 2 shows the sampling times for each bit, where SX_Y is read as sample Y of bit X. Noise is detected if sample 2 does not match sample 1 and 3. The receiver should be able to resynchronize using an advance or delay of one cycle depending upon disagreement of the samples. If sample 1 and 2 agree, but disagree with sample 3, then it is possible that the third sample should have occurred earlier. This means that the sampling should be advanced by one cycle and sample 3 should become the new sample 2. If sample 2 and 3 agree but disagree with sample 1, then sample 1 may have been sampled too early. This means that the sampling should be delayed one cycle before the next bit sampling phase begins.

Table 2. Bit Sampling Time by Baud Rate Divisor

BRD\Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	$S1_1$	$S1_2$	$S1_3$	$S2_1$	$S2_2$	$S2_3$	$S3_1$	$S3_2$	$S3_3$	$S4_1$	$S4_2$	$S4_3$	$S5_1$	$S5_2$	$S5_3$
4		$S1_1$	$S1_2$	$S1_3$		$S2_1$	$S2_2$	$S2_3$		$S3_1$	$S3_2$	$S3_3$		$S4_1$	$S4_2$
5	$S1_1$		$S1_2$		$S1_3$	$S2_1$		$S2_2$		$S2_3$	$S3_1$		$S3_2$		$S3_3$
6		$S1_1$		$S1_2$		$S1_3$		$S2_1$		$S2_2$		$S2_3$		$S3_1$	
7	$S1_1$			$S1_2$			$S1_3$	$S2_1$			$S2_2$			$S2_3$	
8		$S1_1$			$S1_2$			$S1_3$		$S2_1$			$S2_2$		
9	$S1_1$				$S1_2$				$S1_3$	$S2_1$				$S2_2$	
10		$S1_1$				$S1_2$				$S1_3$		$S2_1$			
11	$S1_1$					$S1_2$					$S1_3$	$S2_1$			
12		$S1_1$					$S1_2$					$S1_3$		$S2_1$	
13	$S1_1$						$S1_2$						$S1_3$	$S2_1$	
14		$S1_1$						$S1_2$						$S1_3$	
15	$S1_1$							$S1_2$							$S1_3$

Extra Credit

Extra credit is only available for the groups that correctly implement the NIC transmitter in this project. For the extra credit, implement the NIC receiver that samples each bit three times and can synchronize with other NICs that are of slightly different data rates, this will require the NIC to resynchronize mid frame reception. Single bit errors in a data word will be need to be corrected, and a single word with a double bit error should be correctable as well.

Suggested Approach

Transmitter:

1. Build a clock divisor that will divide the clock down based upon the Baud Rate Divisor. This should output a high signal for one clock every BRDth clock cycle. This signal should be used as an enable for memory that is running at this clock rate, and shouldn't be used as clock for the memory devices.
2. Build a bit-stuffing transmitter. This should have inputs to signal if a new data bit, or special sequence (start frame, stop frame) is being input into the bit stuffer. The output should be the TX pin of the bit stuffed sequence. As there will be some finite amount of memory for a FIFO, there should be an output signal to notify if a new data bit or special sequence can be input.
3. Build a FIFO that can store 16, 22-bit words (this general structure might be reused multiple times with slightly different amounts of values and widths). This will FIFO should have the ability to shift a word in, shift a word out, load a specified number of values in parallel, and unload all in parallel. This FIFO structure will likely used multiple times, and have different parts used. For example writing into the FIFO will shift in to the transmit buffer, then a parallel unload may transfer all contents to a secondary FIFO that is used as a shift buffer for transmission. The secondary FIFO may then use the shift out to update a shift register for the final bit shifting.
4. Build a SECDED encoder that takes in 16-bit data value and encodes it as 22-bit word.
5. Build a transmitter from the other components that you have constructed that will input one 16-bit word at a time to write into its buffer. The transmitter should have a signal specifying if a new word is being written, and a signal to begin transmission of a frame. The transmitter should also have a clock signal, the clock divisor amount, and possibly and enable input. The transmitter should have signals to notify if the transmit buffer is full, if the transmitter is idle, and the transmit serial pin.

UART/Parallel Interface:

1. Build logic to be able to detect the first cycle of a read or write request. This is necessary because you don't want your NIC to believe that multiple writes (or reads) have occurred just because the $\overline{CS}$ and $\overline{O}$ (or $\overline{W}$) remain low for multiple cycles.
2. Build logic to implement the Interrupt Mask register. This should be able to read/written at any time.

3. Build logic to implement the Baud Rate Divisor, so that it can only be updated with valid values, and so that it can only be updated while the NIC is disabled. Make sure that the BRD can be read and written from the parallel interface.
4. Build logic to enable/disable the NIC through the Command/Status register.
5. Build logic to implement the Data register. This will need to be able to buffer every other byte. This will allow the writing of only 16-bit values to the transmitter. Insert the transmitter that will be written to with the 16-bit values with every other write.
6. Build logic to implement the Transmit Frame command to the command register. This should do a write of a 16-bit value first if one byte is buffered. The MSB of the 16-bit value should be all zero if this occurs. After the 16-bit value is written, the Transmit Frame should be sent. If no byte is buffered, then the Transmit Frame command should be sent to the transmitter immediately.
7. Build logic to implement the TXI and TBNF flags in the Status register.

Receiver:

1. Build logic to do the bit sampling. This unit should have a RX pin, a clock, and divisor inputs. The outputs should be a flag to signal that there is a new data bit, the value of the bit, and a noise flag. The bit sampler should be able to resynchronize by advancing or delaying sampling as specified in the previous section.
2. Build logic to do the bit "destuffing". This unit should be able to use the bit sampler to detect stuff bits, and to detect special sequences like Start Frame and Stop Frame. The destuffer should be able to detect idle condition (seven or more consecutive 0's). It should also be able to detect an error condition (seven or more consecutive 1's). The destuffer should obviously also be able to detect a valid data bit.
3. Build a SECDED decoder. This should correct the received data if there is a single error, or output that there was a uncorrectable error. There may be detected single errors that may signal a larger number of errors (the bit to be corrected is larger than 21).
4. Build a receiver that can detect the begin of a frame, and store the bits of the frame, and then the stop of frame. This should be able to assemble the words from the individual bits. Correct the words as they are received, or store them as all zero if an uncorrectable error is detected. You will likely need to keep track of which were received correctly and which were corrupted for later. Copy the received frame to a receive buffer that be read 16-bits at a time (make sure to ignore the parity word).
5. Add noise detection (and clearing), this will occur if any noise is detected during the receipt of a frame.
6. Add overrun detection (and clearing). This will occur when there are buffered words still when a new frame is received.
7. Add word error correction to the receiver that will replace the all zeros with the XOR of all other received words.
8. Add SECDED error detection (and clearing). This occurs if an uncorrectable error is detected.
9. Add frame error detection (and clearing). This occurs if during a frame more than one word error is detected, a Start Frame is received before a Stop Frame, or a Stop Frame is received in the middle of a frame word (during bits 7..21 of the word).
10. Integrate the receiver into the rest of the NIC.