CECS 460 Chip Specification

Dustin Nguyen

5/10/2020

Chip Specification

Prepared By:	Date:	Document Number and Filename:	Revision:
N. Dustin	May 10, 2020	UART Specification	1

Table of Contents

1 Introduction

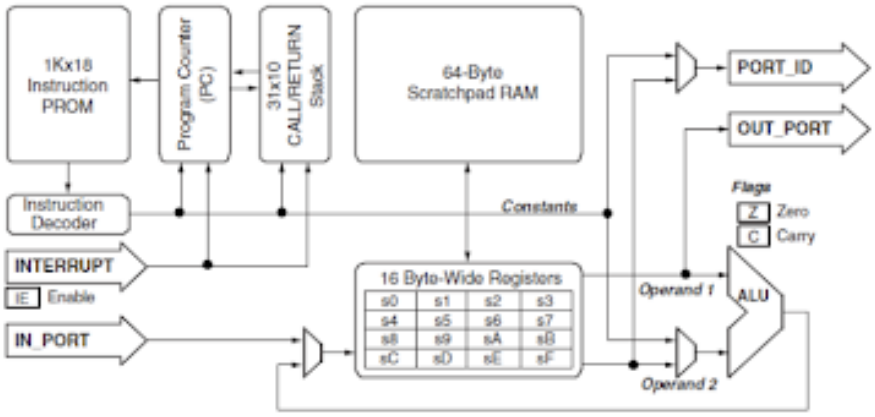
This document provides the chips specification for the Universal Asynchronous Receiver Transmitter (UART) developed in the System-on-Chip Design course taught by John Tramel. It uses a microprocessor, the TramelBlaze, to process its assembly code into functions for the UART. The UART is composed of the Transmit Engine (TX) and the Receive Engine (RX) as the main means of transmitting and receiving data. It is also supported with registers, state machines, Positive Edge Detect (PED), and Asynchronous-In-Synchronous-Out Reset (AISO) to help if function correctly.

1.1 Purpose

To provide a UART transmission protocol with a selectable baud rate, bits transmitted, parity enable/disable, and an odd/even parity bit

2 External Documents

2.1 PicoBlaze



The PicoBlaze and TramelBlaze share almost the exact same architecture with the only difference being that the TramelBlaze's instruction Rom is 4Kx16, the stack RAM is 128x16, the scratchpad RAM is 512x16, and the bus lines are 16 bits.

Instruction	Description	Function	ZERO	CARRY
ADD sX, kk	Add register sX with literal kk	sX ← sX + kk	?	?
ADD sX, sY	Add register sX with register sY	sX ← sX + sY	?	?
ADDCY sX, kk (ADDC)	Add register sX with literal kk with CARRY bit	sX ← sX + kk + CARRY	?	?
ADDCY sX, sY (ADDC)	Add register sX with register sY with CARRY bit	sX ← sX + sY + CARRY	?	?
AND sX, kk	Bitwise AND register sX with literal kk	sX ← sX AND kk	?	0
AND sX, sY	Bitwise AND register sX with register sY	sX ← sX AND sY	?	0
CALL aaa	Unconditionally call subroutine at aaa	TOS ← PC PC ← aaa	-	-
CALL C, aaa	If CARRY flag set, call subroutine at aaa	If CARRY=1, (TOS ← PC, PC ← aaa)	-	-
CALL NC, aaa	If CARRY flag not set, call subroutine at aaa	If CARRY=0, (TOS ← PC, PC ← aaa)	-	-
CALL NZ, aaa	If ZERO flag not set, call subroutine at aaa	If ZERO=0, (TOS ← PC, PC ← aaa)	-	-

Instruction	Description	Function	ZERO	CARRY
CALL Z, aaa	If ZERO flag set, call subroutine at aaa	If ZERO=1, [TOS ← PC, PC ← aaa]	-	-
COMPARE sX, kk (COMP)	Compare register sX with literal kk. Set CARRY and ZERO flags as appropriate. Registers are unaffected.	If sX=kk, ZERO ← 1 If sX<kk, CARRY ← 1	?	?
COMPARE sX, sY (COMP)	Compare register sX with register sY. Set CARRY and ZERO flags as appropriate. Registers are unaffected.	If sX=sY, ZERO ← 1 If sX<sY, CARRY ← 1	?	?
DISABLE INTERRUPT (DINT)	Disable interrupt input	INTERRUPT_ENABLE ← 0	-	-
ENABLE INTERRUPT (EINT)	Enable interrupt input	INTERRUPT_ENABLE ← 1	-	-
Interrupt Event	Asynchronous interrupt input. Preserve flags and PC. Clear INTERRUPT_ENABLE flag. Jump to interrupt vector at address 3FF.	Preserved ZERO ← ZERO Preserved CARRY ← CARRY INTERRUPT_ENABLE ← 0 TOS ← PC PC ← 3FF	-	-
FETCH sX, (sY) (FETCH sX, sY)	Read scratchpad RAM location pointed to by register sY into register sX	sX ← RAM[(sY)]	-	-
FETCH sX, ss	Read scratchpad RAM location ss into register sX	sX ← RAM[ss]	-	-
INPUT sX, (sY) (IN sX, sY)	Read value on input port location pointed to by register sY into register sX	PORT_ID ← sY sX ← IN_PORT	-	-
INPUT sX, pp (IN)	Read value on input port location pp into register sX	PORT_ID ← pp sX ← IN_PORT	-	-
JUMP aaa	Unconditionally jump to aaa	PC ← aaa	-	-
JUMP C, aaa	If CARRY flag set, jump to aaa	If CARRY=1, PC ← aaa	-	-
JUMP NC, aaa	If CARRY flag not set, jump to aaa	If CARRY=0, PC ← aaa	-	-
JUMP NZ, aaa	If ZERO flag not set, jump to aaa	If ZERO=0, PC ← aaa	-	-
JUMP Z, aaa	If ZERO flag set, jump to aaa	If ZERO=1, PC ← aaa	-	-
LOAD sX, kk	Load register sX with literal kk	sX ← kk	-	-
LOAD sX, sY	Load register sX with register sY	sX ← sY	-	-
OR sX, kk	Bitwise OR register sX with literal kk	sX ← sX OR kk	?	0
OR sX, sY	Bitwise OR register sX with register sY	sX ← sX OR sY	?	0
OUTPUT sX, (sY) (OUT sX, sY)	Write register sX to output port location pointed to by register sY	PORT_ID ← sY OUT_PORT ← sX	-	-
OUTPUT sX, pp (OUT sX, pp)	Write register sX to output port location pp	PORT_ID ← pp OUT_PORT ← sX	-	-
RETURN (RET)	Unconditionally return from subroutine	PC ← TOS+1	-	-

Instruction	Description	Function	ZERO	CARRY
RETURN C (RET C)	If CARRY flag set, return from subroutine	If CARRY=1, PC ← TOS+1	-	-
RETURN NC (RET NC)	If CARRY flag not set, return from subroutine	If CARRY=0, PC ← TOS+1	-	-
RETURN NZ (RET NZ)	If ZERO flag not set, return from subroutine	If ZERO=0, PC ← TOS+1	-	-
RETURN Z (RET Z)	If ZERO flag set, return from subroutine	If ZERO=1, PC ← TOS+1	-	-
RETURNI DISABLE (RETI DISABLE)	Return from interrupt service routine. Interrupt remains disabled.	PC ← TOS ZERO ← Preserved ZERO CARRY ← Preserved CARRY INTERRUPT_ENABLE ← 0	?	?
RETURNI ENABLE (RETI ENABLE)	Return from interrupt service routine. Re-enable interrupt.	PC ← TOS ZERO ← Preserved ZERO CARRY ← Preserved CARRY INTERRUPT_ENABLE ← 1	?	?
RL sX	Rotate register sX left	sX ← [sX[6:0],sX[7]] CARRY ← sX[7]	?	?
RR sX	Rotate register sX right	sX ← [sX[0],sX[7:1]] CARRY ← sX[0]	?	?
SL0 sX	Shift register sX left, zero fill	sX ← [sX[6:0],0] CARRY ← sX[7]	?	?
SL1 sX	Shift register sX left, one fill	sX ← [sX[6:0],1] CARRY ← sX[7]	0	?
SLA sX	Shift register sX left through all bits, including CARRY	sX ← [sX[6:0],CARRY] CARRY ← sX[7]	?	?
SLX sX	Shift register sX left. Bit sX[0] is unaffected.	sX ← [sX[6:0],sX[0]] CARRY ← sX[7]	?	?
SR0 sX	Shift register sX right, zero fill	sX ← [0,sX[7:1]] CARRY ← sX[0]	?	?
SR1 sX	Shift register sX right, one fill	sX ← [1,sX[7:1]] CARRY ← sX[0]	0	?
SRA sX	Shift register sX right through all bits, including CARRY	sX ← [CARRY,sX[7:1]] CARRY ← sX[0]	?	?
SRX sX	Arithmetic shift register sX right. Sign extend sX. Bit sX[7] is unaffected.	sX ← [sX[7],sX[7:1]] CARRY ← sX[0]	?	?
STORE sX, (sY) (STORE sX, sY)	Write register sX to scratchpad RAM location pointed to by register sY	RAM[sY)] ← sX	-	-
STORE sX, ss	Write register sX to scratchpad RAM location ss	RAM[ss] ← sX	-	-

Instruction	Description	Function	ZERO	CARRY
SUB sX, kk	Subtract literal kk from register sX	sX ← sX – kk	?	?
SUB sX, sY	Subtract register sY from register sX	sX ← sX – sY	?	?
SUBCY sX, kk (SUBC)	Subtract literal kk from register sX with CARRY (borrow)	sX ← sX – kk - CARRY	?	?
SUBCY sX, sY (SUBC)	Subtract register sY from register sX with CARRY (borrow)	sX ← sX – sY - CARRY	?	?
TEST sX, kk	Test bits in register sX against literal kk. Update CARRY and ZERO flags. Registers are unaffected.	If (sX AND kk) = 0, ZERO ← 1 CARRY ← odd parity of (sX AND kk)	?	?
TEST sX, sY	Test bits in register sX against register sX. Update CARRY and ZERO flags. Registers are unaffected.	If (sX AND sY) = 0, ZERO ← 1 CARRY ← odd parity of (sX AND kk)	?	?
XOR sX, kk	Bitwise XOR register sX with literal kk	sX ← sX XOR kk	?	0
XOR sX, sY	Bitwise XOR register sX with register sY	sX ← sX XOR sY	?	0

Table for PicoBlaze and TramelBlaze Instruction Set

2.2 Nexys 4

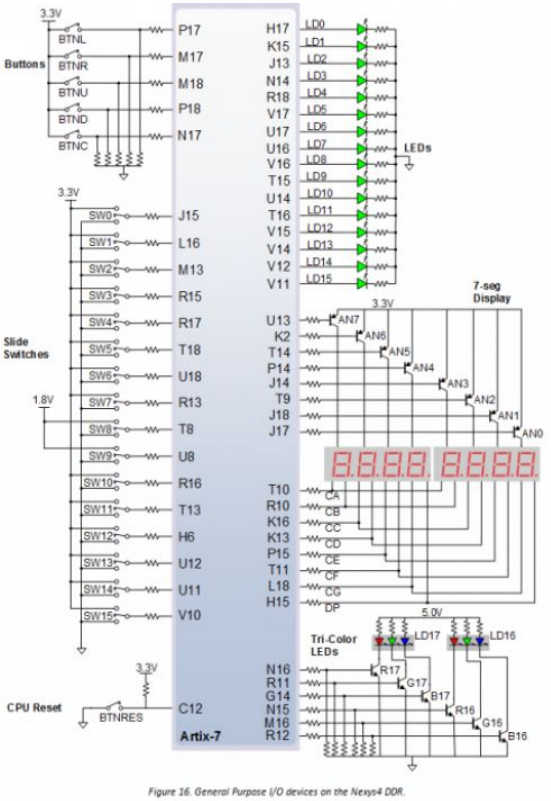


Figure 16. General Purpose I/O devices on the Nexys4 DDR.

2.3 ASCII Table

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

3 Requirements

3.1 Interface Requirements

The design is made up of the UART and the TramelBlaze which work together to utilize the inputs and outputs from the Nexys 4 board. The design has 11 selectable Baud rates through the Nexys 4 on board switches, a 3 bit parity control, a parity enable/disable, and an odd or even parity select. It can also serially output through the TX port and indicate the microprocessor with a Txrdy signal

3.2 Hardware Requirements

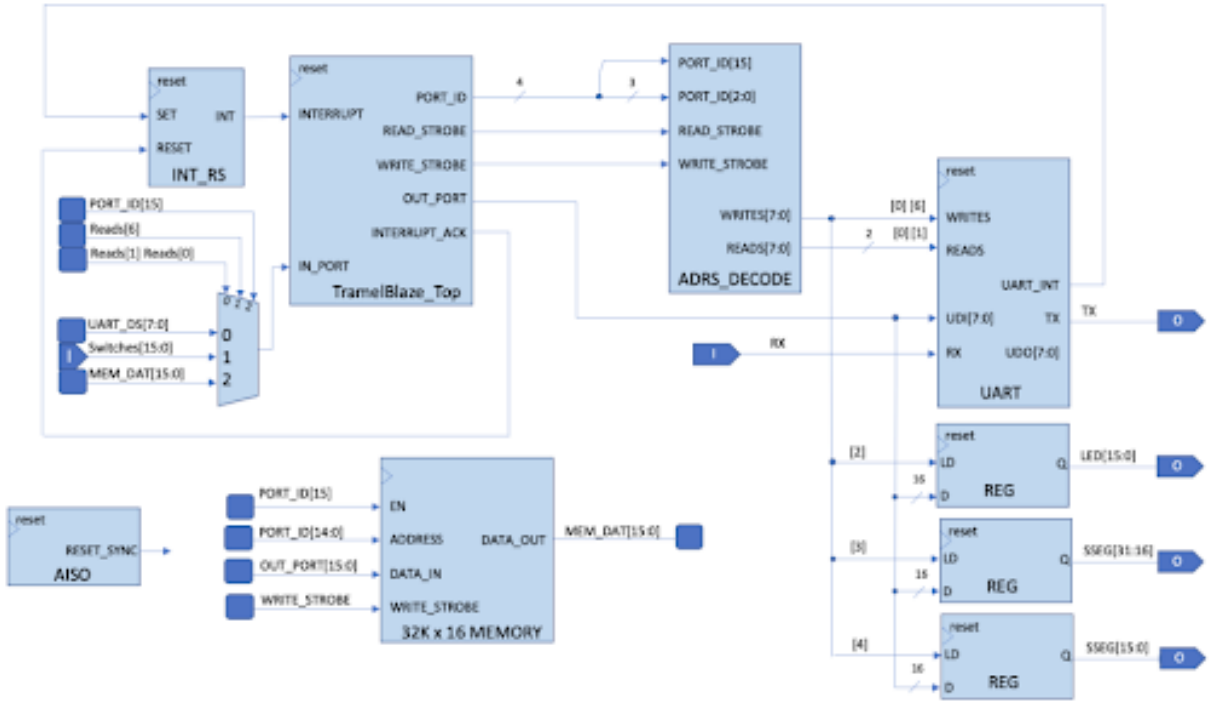
We are using the 7 switches and 1 button on the Nexy4 board. The button on the upper top of the board is used for the reset signal. The switches 7 to 4 are used for baud rate selection and switches 3 to 1 are used for parity checking. Switch 3 will enable eight bit data, switch 2 will enable the parity bit, and switch 1 will check if it is an odd or even parity bit.

4 Top Level Design

4.1 Description

The top level consists of the entire digital design utilizing the UART and TramelBlaze. The SOPC CORE contains the UART and TramelBlaze and the TSI contains needed libraries for the design to function. I/O pases first through the TSI then flows to the SOPC. The function of the UART is for it to display a banner stored in the TramelBlaze ROM which can then be interacted through user input on the Nexys4 Board.

4.2 Block Diagram



4.3 Data Flow Description

The main reset signal is first delivered through the AISO module and then to the rest of the design to synchronize all resets and prevent metastability. The UART engine consists of the TX engine for transmitting data and the RX engine for receiving data. It uses the TramelBlaze to execute the instructions given from the UART as well as driving the LEDs on the Nexys4 board. The address decoder will decode the read and write strobe data and write to the memory locations.

4.4 I/O

Signal Names and Pin Assignments

Signal	From	To	Description
w_CLK	TSI	SOPC_CORE	100MHz Crystal Oscillator
w_RST	TSI	SOPC_CORE	System Reset
w_RX	TSI	SOPC_CORE	Rx Line from USB
w_SW	TSI	SOPC_CORE	UART controls from Switches
w_TX	SOPC_CORE	TSI	Tx Line to USB
w_LED	SOPC_CORE	TSI	LED outputs

Table 7: Top Level Signal Names

Input Signals	Assignment	Output Signals	Assignment
SYS_CLK	V10	o_TX	N18
SYS_RST	A8	o_LED[7]	T11
i_RX	N17	o_LED[6]	R11
i_SW[6]	T5	o_LED[5]	N11
i_SW[5]	V8	o_LED[4]	M11
i_SW[4]	U8	o_LED[3]	V15
i_SW[3]	N8	o_LED[2]	U15
i_SW[2]	M8	o_LED[1]	V16
i_SW[1]	V9	o_LED[0]	U16
i_SW[0]	T9		

Table 8: Top Level Pin Assignment

4.5 Software

UART assembly code (pg.)

5 Externally Developed Blocks

5.1 TramelBlaze

The TramelBlaze is a 16-bit microprocessor that emulates the PicoBlaze. It uses a 4Kx16 bit ROM instruction memory where it reads and executes assembly code. We use the TramelBlaze alongside the UART engine to communicate with the Serial Terminal and display the ASCII values.

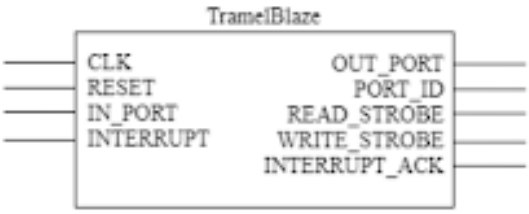


Figure 7: TramelBlaze Block Diagram

I/O

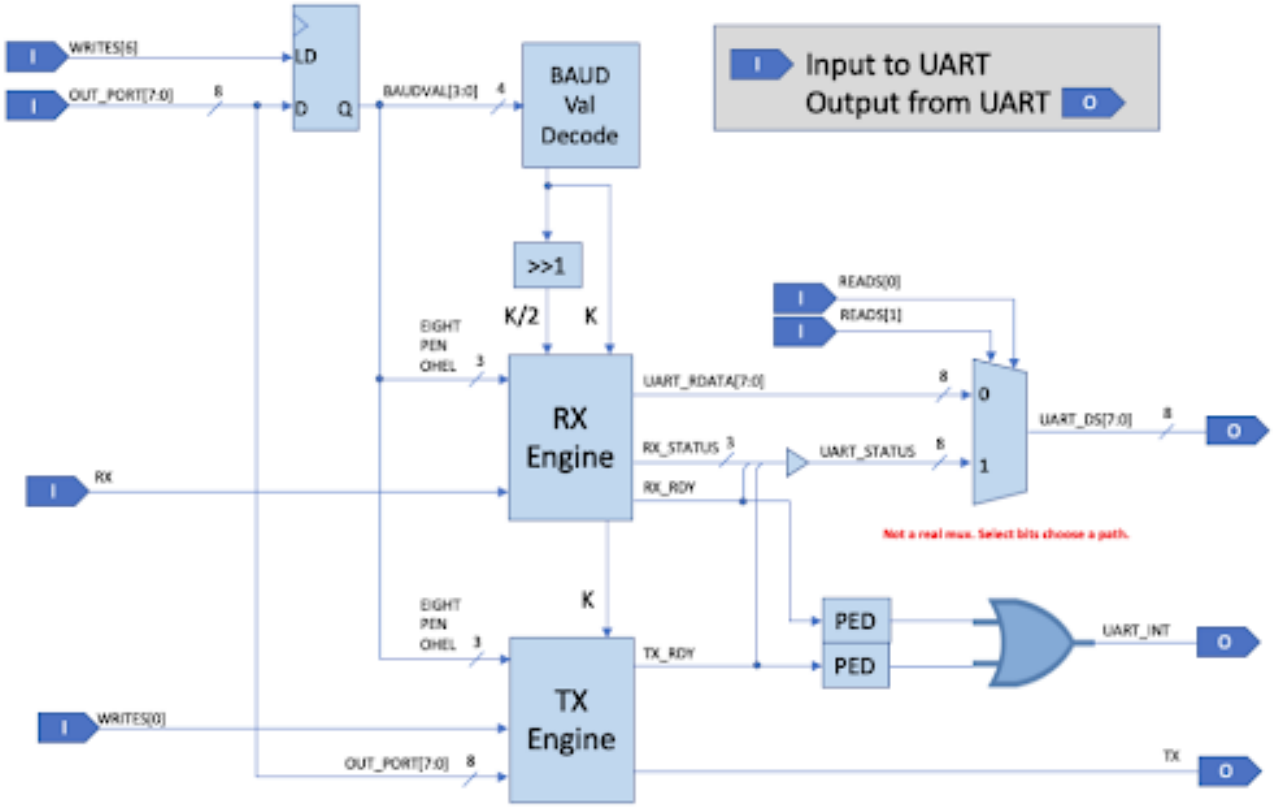
Signal	Size (bit)	I/O	Connected to
CLK	1	I	100MHz Crystal Oscillator
RESET	1	I	AISO_RST
INTERRUPT	1	I	RS_FLOP
IN_PORT	16	I	UART_TOP
OUT_PORT	16	O	UART_TOP
PORT_ID	16	O	Address Decoder
INTERRUPT_ACK	1	O	RS_FLOP
READ_STROBE	1	O	UART_TOP
WRITE_STROBE	1	O	UART_TOP

Table 9: TramelBlaze I/O

6 Internally Developed Blocks

6.1 Universal Asynchronous Receiver Transmitter (UART)

The UART is a combination of the TX and RX engine that acts together as a serial communication protocol. The 3 major parts of the UART are the Transmission Engine, Receive engine, and the Baud Decoder. The Transmission and Receive engine are responsible for the transmitting and receiving the data while the Buad decoder will give a baud rate for a specific bit time to keep the transmission and receive engine synchronized. The UART also produces interrupts that are acknowledged by the TramelBlaze for when each engine is ready.



Inputs:

1. Writes[6] - Signal from the address decode to load the UART
2. Out_port[7:0] - 8 bit signal for the onboard switches
3. RX - Signal indicating that the RX engine can begin receiving serial data
4. Writes[0] - Signal from the address decode to the TX engine load
5. Reads[1:0] - Signal for the select of the RX engine

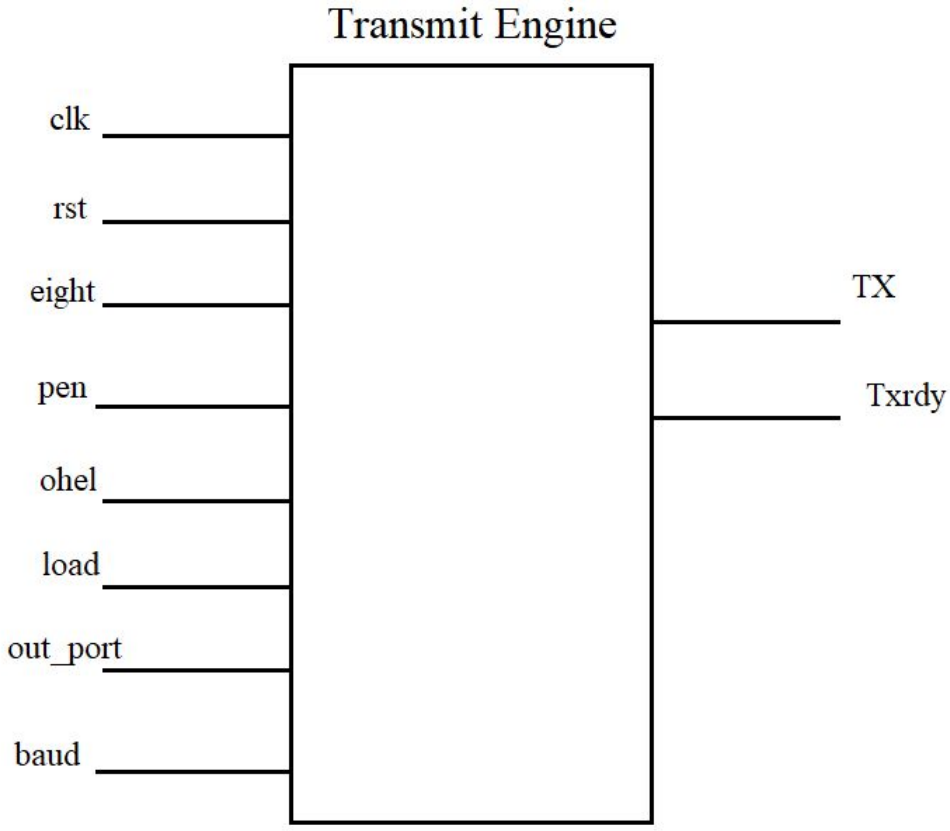
Output:

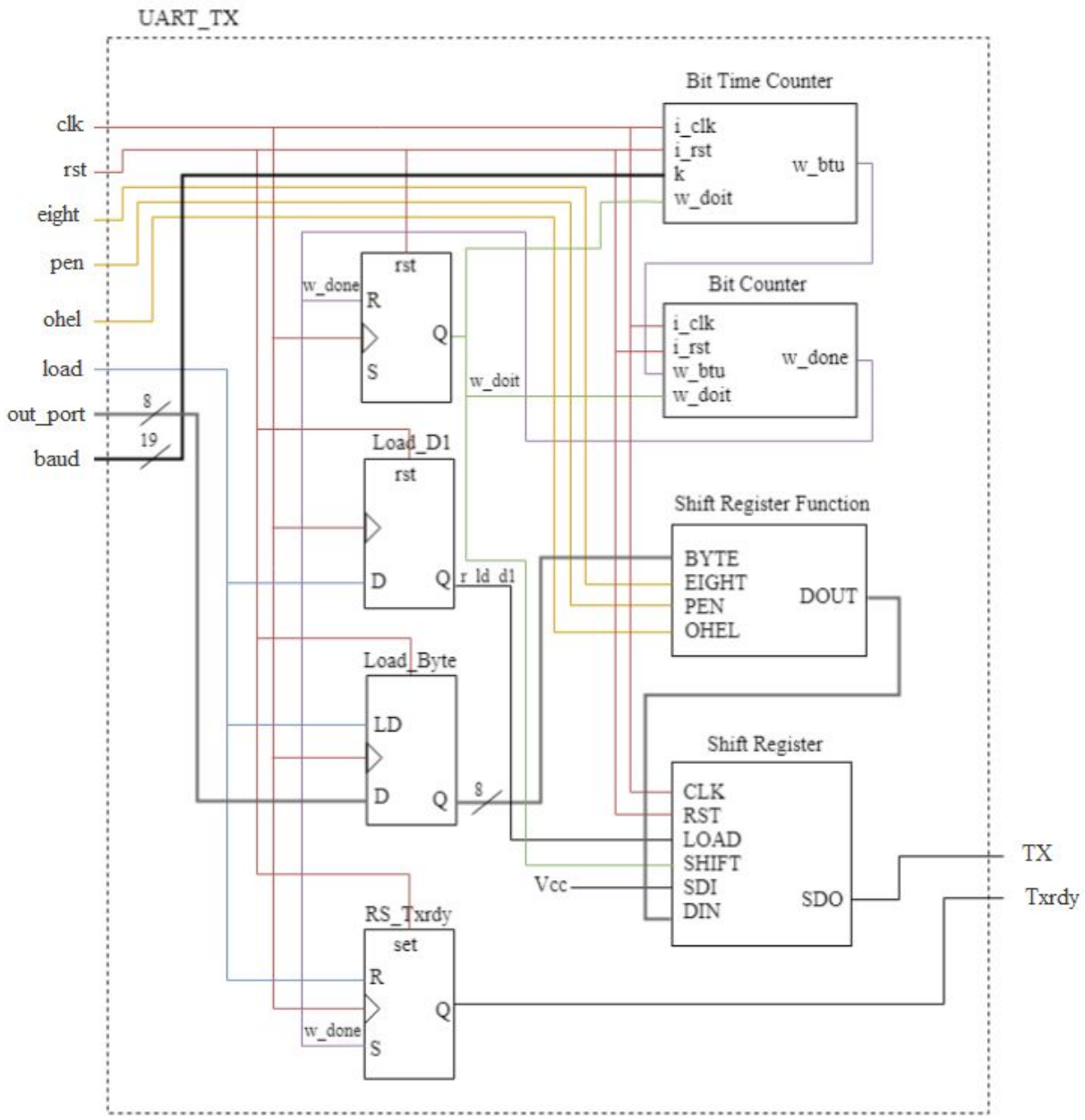
1. UART_DS[7:0] - 8 bit signal going to the TramelBlaze
2. UART_INT - Signal that goes through the RS flop of the TramelBlaze
3. TX - Signal with transmission serial data going to the terminal program

6.1.1 Transmit Engine

The transmission engine serves as a communication protocol from one device to another without clock crossing. The transmit engine is responsible for serial shifting data through the transmit line to a receiving device. There are 4 components that make up the transmit engine; the baud decoder, bit time counter, bit counter, and shift register/ The baud decoder first determines which frequency for the data transmission. The signal from there then goes to the bit time counter to generate a pulse. Each time a signal is sent it is counted while being shifted by the shift register one by one.

Block Diagram





Input:

1. Clk - clk
2. Rst - to return to a known state
3. Eight - Switches that choose the mode of the shift register
4. Pen - Switches that choose the mode of the shift register
5. Ohel - Switches that choose the mode of the shift register
6. Load - load signal

7. Out_port - inputs from the onboard switches
8. Baud - controls the baud rate at which the bits are generated

Output

1. TX - Outputs the message
2. Txrdy - feeds to the PED to indicate a signal.

Register Map

Txrdy	RS_Flop	Indicates that the Transmition is ready
[18:0] Count	Bit_time_counter	Buad Rate Counter
[3:0] BTC	Bit_counter	Bit time Counter
loadL	Load_D1_Flop	Sets a baud rate that delays the time the data takes to reach the shift register as well as delaying the w_doit as well
out_port	Load_Byte_Flop	Stores the TramelBlaze input data

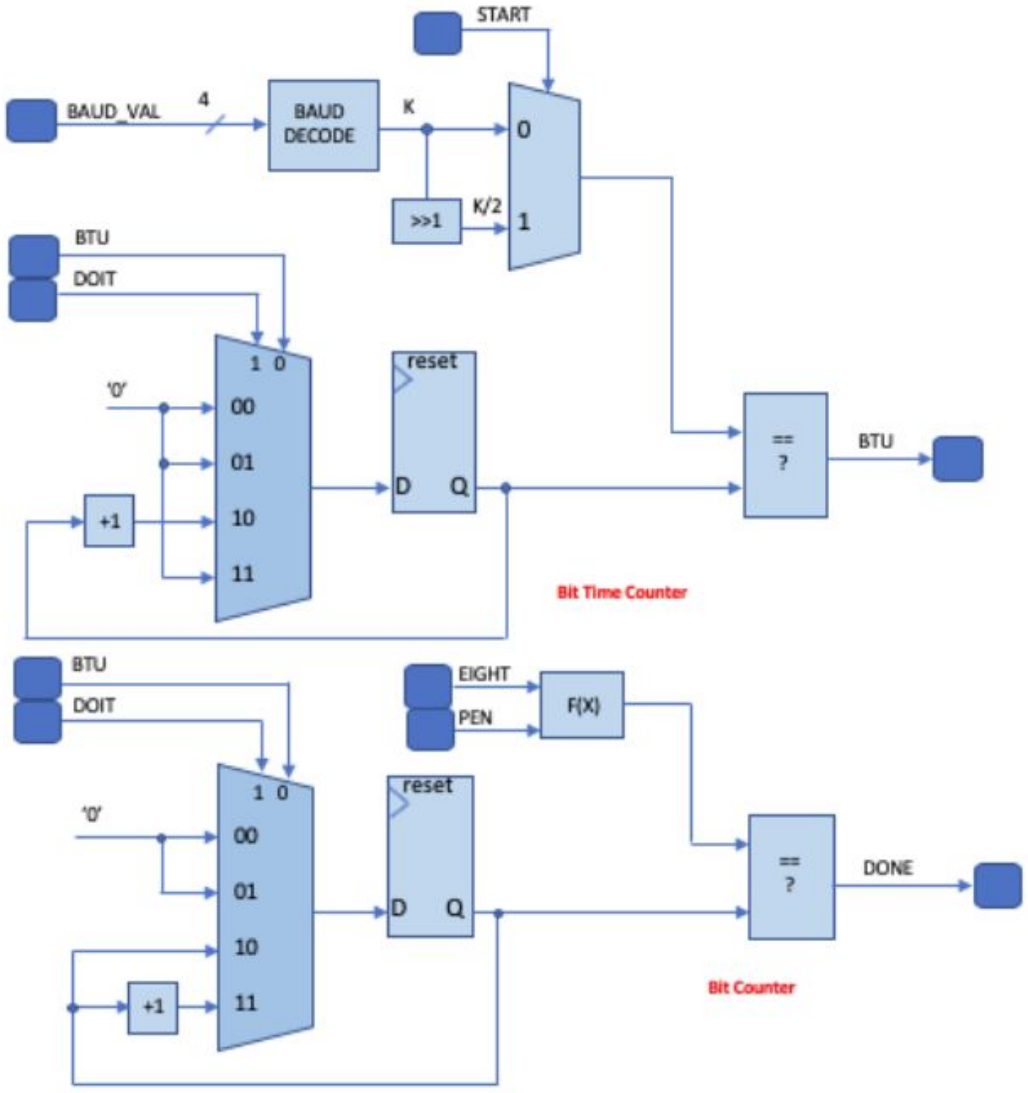
Verification

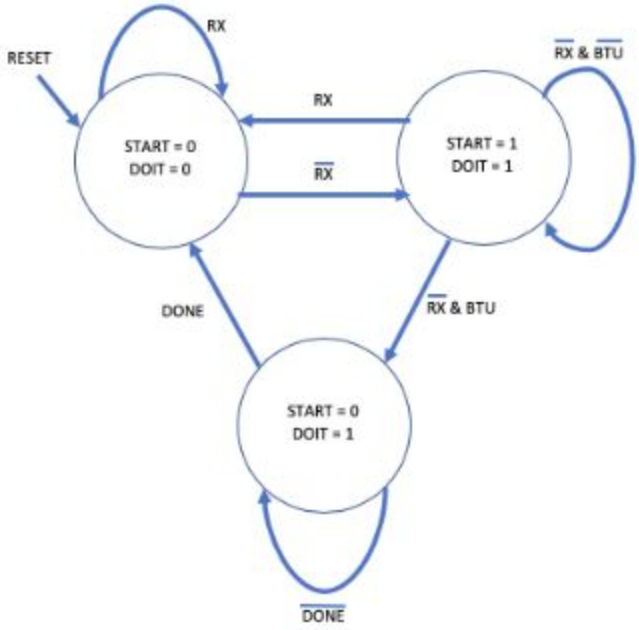
The Transmit Engine is able to shift the data in accordance to the parity bits and baud rate through the Tx line

6.1.2 Receive Engine

The receive engine synchronizes the data being collected with the Tx engine. The receive built from a finite State machine and a shift register to be able to receive the serial data. The receive engine continually looks for a 1 to 0 transition to indicate the start bit. Data is then collected from that point with the first 2 values dictating what happens in the UART protocol control.

Block Diagram





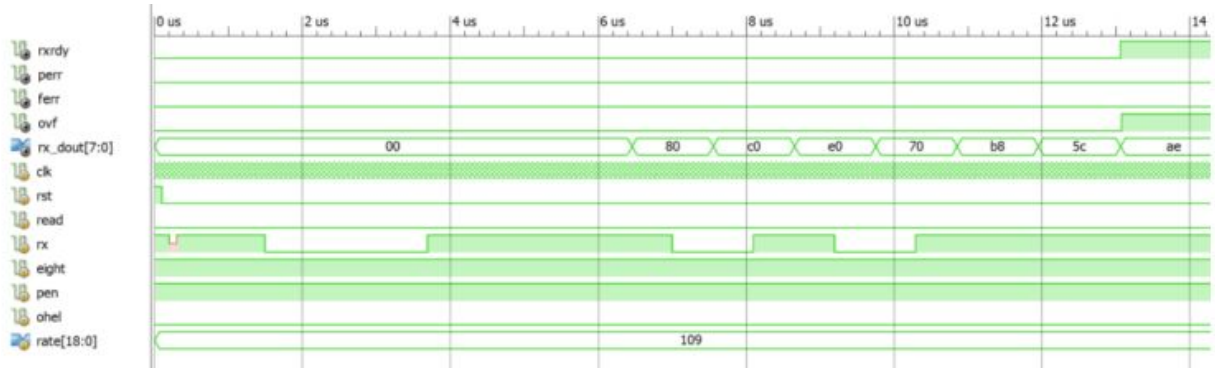
Input:

1. Start - Indicates the receive engine to start recording the values being sent
2. Baud Value - Determines the time delay to take while receiving data
3. BTU - Counts the amount of bits being receive
4. Doit - Indicates when the receive engine will start
5. Eight - Determines the mode of the receive engine
6. Pen - Determines the mode of the receive engine

Output

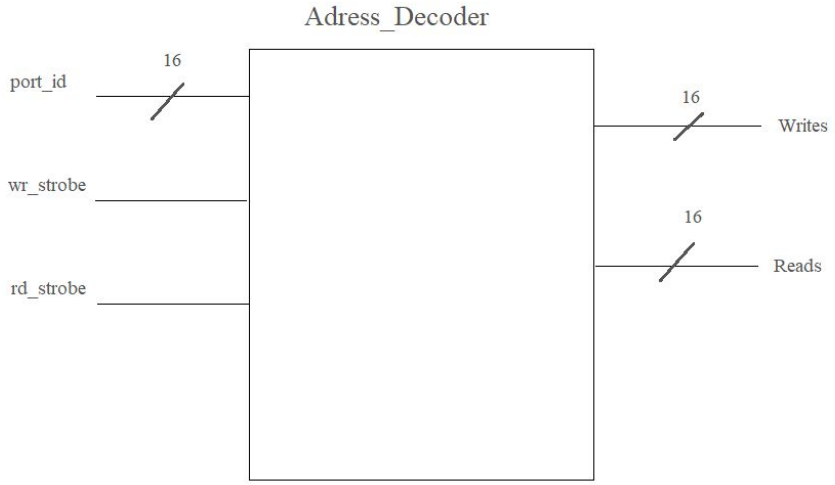
1. BTU - returns the amount of bits counted
2. Done - returns that the receive engine is done recording the data

Verification



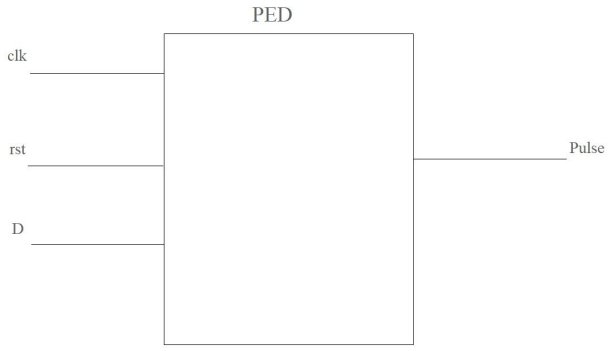
6.2 Address Decoder

The address decoder is a combinational block that decodes the port ID from the TramelBlaze into 2 16-bit read or write strobe outputs. If it reads a read or write it duplicates the signal and turns the bit position of the current port ID to the respective position.



6.3 Positive Edge Detector (PED)

A PED is a module that detects a rising edge of a signal and outputs a one clock period signal in response. It is used to indicate a change in signal, such as a high to low or a low to high shift.



6.4 Asynchronous In Synchronous Out Reset (AISO Reset)

This module's purpose is to synchronize the reset of the system to apply to all modules at the same time. The reason is that if the system were to reset un synchronized, metastability would occur giving off incorrect data.

7 Chip Level Verification

7.1 Transmit Engine

The transmit engine’s job is to continuously shift out 16-bit data that is assigned by the Tramelblaze. The specifications of the data itself are controlled by whether the transmit engine sees a load, the size of the data, and the parity of the data. In the simulation we mark the start of the transmission with low to high state. After which is followed by 7-8 bits of data a parity bit

and an end bit. We can see that the data is being constantly shifted through the TX line which should be identically mirrored by the receiving engine.

My board was not currently working at the time but to test whether the board is actually transmitting the correct data we use the USB cable to connect to another computer and the program Realterm to see the serial transmitted data. By adjusting the baud rate, bit data, parity, and stop bit we can synchronize the RealTerm to be able to receive our transmitted data and display what is being transmitted to verify the proper operation of our transmit engine.

7.2 Receive Engine

The receive engine is in conjunction with the transmit engine to be able to receive the data being sent and translate it to coherent data. To test the functions we have to make sure that we can correctly transmit the data we want to send. The way we do it is we first transmit data using a specific requirement (even bit parity disable) to a text file. We then get the data from the file for the receive engine to read. Then we just verify that the data transmitted is the same that we are receiving.

8 Chip Level Test

To successfully show that we have created a working UART that can transmit and receive data we were required to make a program that would display a banner and based on user input, display certain messages. Using Realterm we hooked up the Nexys4 board to a computer so that the data can be read. If the input was a <cr> then we should have the cursor start a new line. If the input is a <bs> then it acts as a backspace deleting the charter in front of the cursor.

An input of '*' will display the hometown followed by a new line and an input of '@' will display the number of characters that has been received so far.