UART, SRAM & TSI

CSULB CECS 460

Cristian Lopez

May 14, 2019

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Table of Contents

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

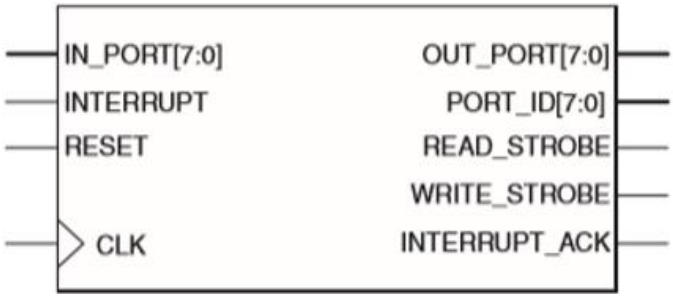
Introduction

For the Receive engine part of the UART project, we made a FPGA sent a message to the computer using a terminal, in this case I used real term since it included much functionality than Putty. The message we displayed with the FPGA was a prompt and a line counter that would display the number of characters we would be echoing. We achieved to display this message by writing the hexadecimal number of the specific letter we wanted to display. The line counter would turn the line number in binary to a hexadecimal value in ascii which is displayed in the terminal

External Documents

PicoBlaze

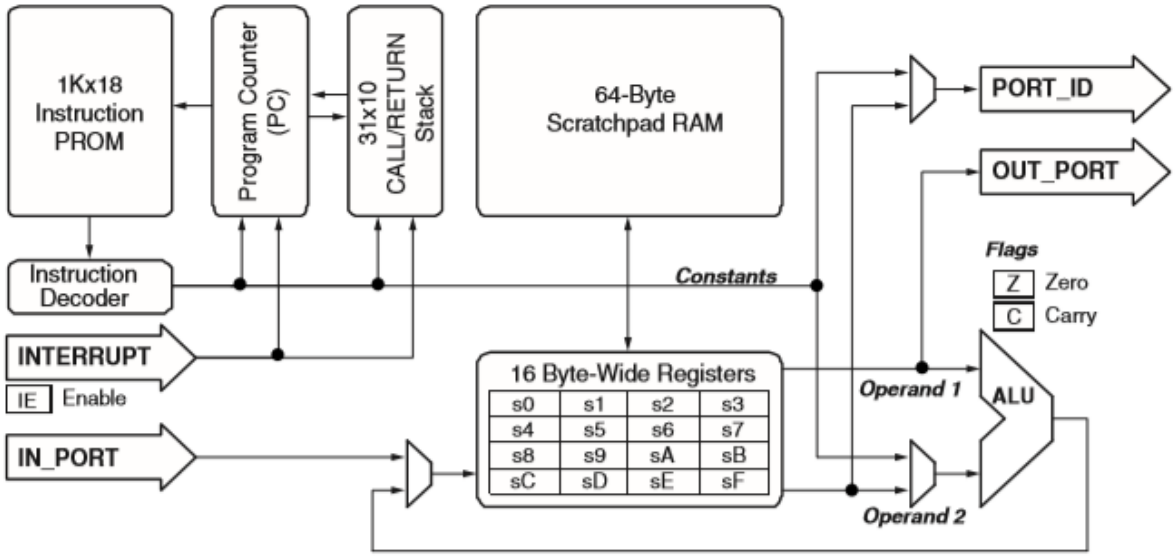
The PicoBlaze microcontroller is a compact, capable, and cost-effective fully embedded 8-bit RISC microcontroller core optimized for the Spartan3, VirtexII, and Virtex-II Pro FPGA families. The PicoBlaze microcontroller provides cost-efficient microcontroller-based control and simple data processing.



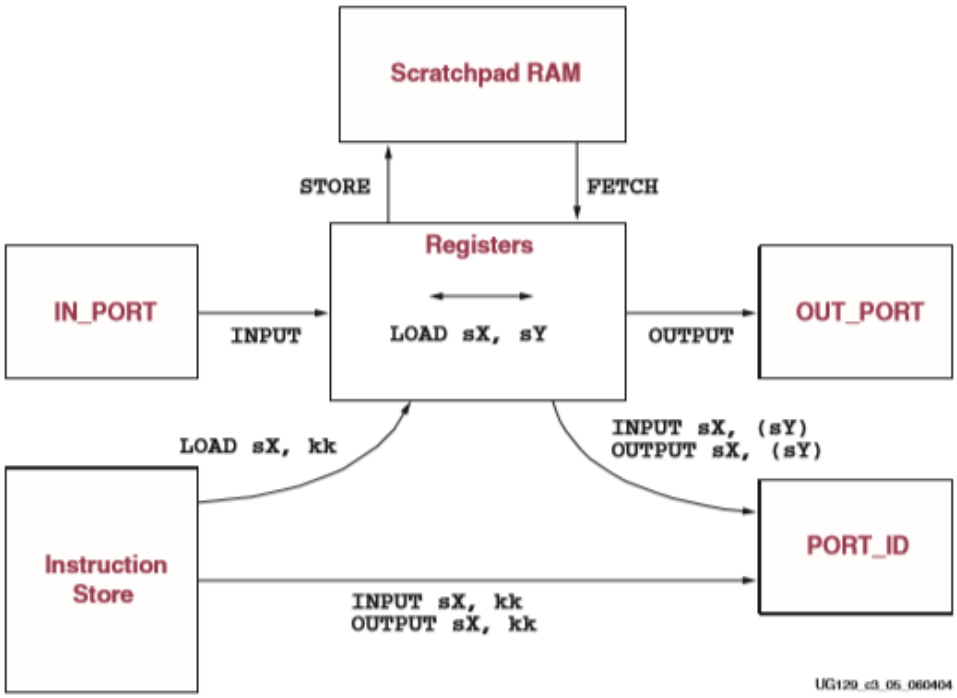
The PicoBlaze microcontroller supports the following features:

• 16 byte-wide general-purpose data registers
• 1K instructions of programmable on-chip program store, automatically loaded during FPGA configuration
• Byte-wide Arithmetic Logic Unit (ALU) with CARRY and ZERO indicator flags
• 64-byte internal scratchpad RAM
• 256 input and 256 output ports for easy expansion and enhancement
• Automatic 31-location CALL/RETURN stack
• Predictable performance, always two clock cycles per instruction, up to 200 MHz or 100 MIPS in a Virtex-II Pro FPGA
• Fast interrupt response; worst-case 5 clock cycles
• Optimized for Xilinx Spartan-3, Virtex-II, and Virtex-II Pro FPGA architectures—just 96 slices and 0.5 to 1 block RAM
• Assembler, instruction-set simulator support

Data movement between various resources is an essential microcontroller function. The next picture shows the various PicoBlaze instructions to move data.



Data movement



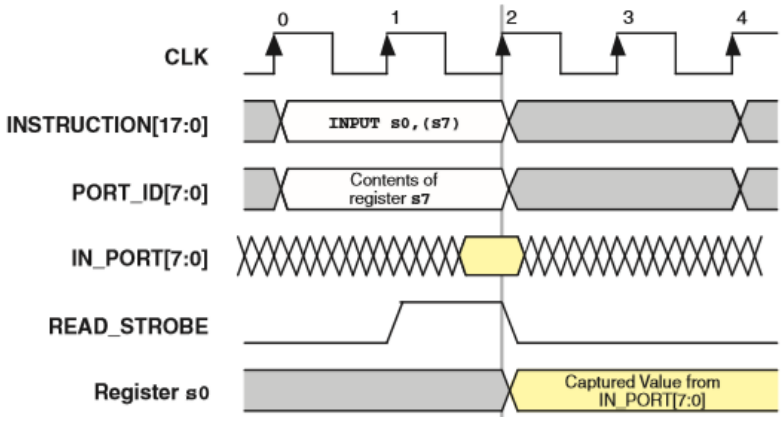
PicoBlaze Address Spaces and Related Instructions

As shown in the next image, the PicoBlaze microcontroller has five distinct address spaces. Specific instructions operate on each of the address spaces.

Address Space	Size (Depth x Width)	Addressing Modes	Instructions that Operate on Address Space
Instruction	1Kx18	Direct	• JUMP • CALL • RETURN • RETURNI • INTERRUPT event • RESET event All others increment the PC to the next location
Register File	16x8	Direct	• LOAD • AND • OR • XOR • TEST (read only) • ADD • ADDCY • SUB • SUBCY • COMPARE (read only) • SR0 • SR1 • SRX • SRA • RR • SL0 • SL1 • SLX • SLA • RL • INPUT • OUTPUT (read only) • STORE (read only) • FETCH
Scratchpad RAM	64x8	Direct Indirect	• STORE • FETCH
I/O	256x8	Direct Indirect	• INPUT • OUTPUT
CALL/RETURN Stack	31x10	N/A	• CALL • Enabled INTERRUPT event • RETURN • RETURNI • RESET event

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

The INPUT operation asserts the associated READ_STROBE output pulse on the second cycle of the two-cycle INPUT cycle, as shown in the next picture. This port indicates that the PicoBlaze microcontroller has acquired the data. READ_STROBE is critical when reading data from a FIFO, acknowledging receipt of data as shown . The PicoBlaze captures the value on IN_PORT[7:0] into register s0 on this clock edge.



Nexys 4 Datasheet

FPGA Board used for this project is a Nexys 4. This board is made by diligent and it contains the Artix 7. In the next image we can see the board with all of its I/O.

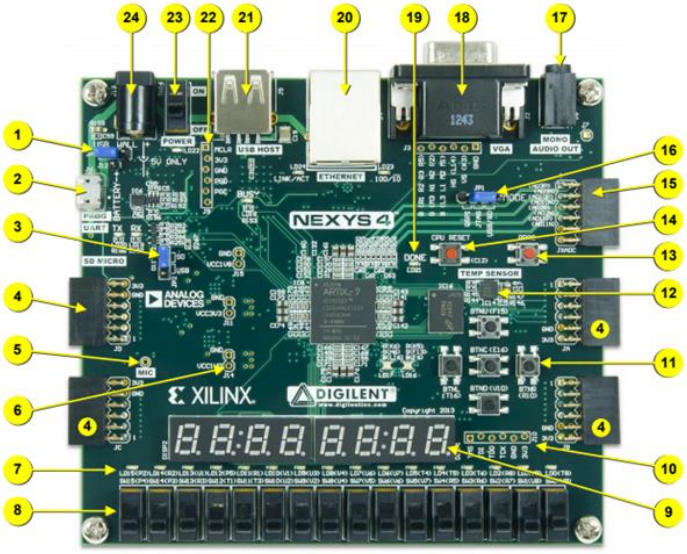


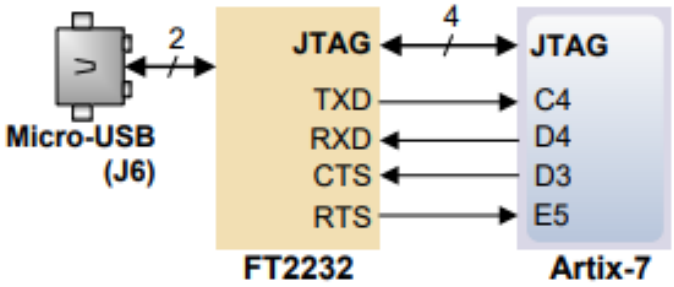
Figure 1. Nexys 4 board features

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod port (XADC)
4	Pmod port(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

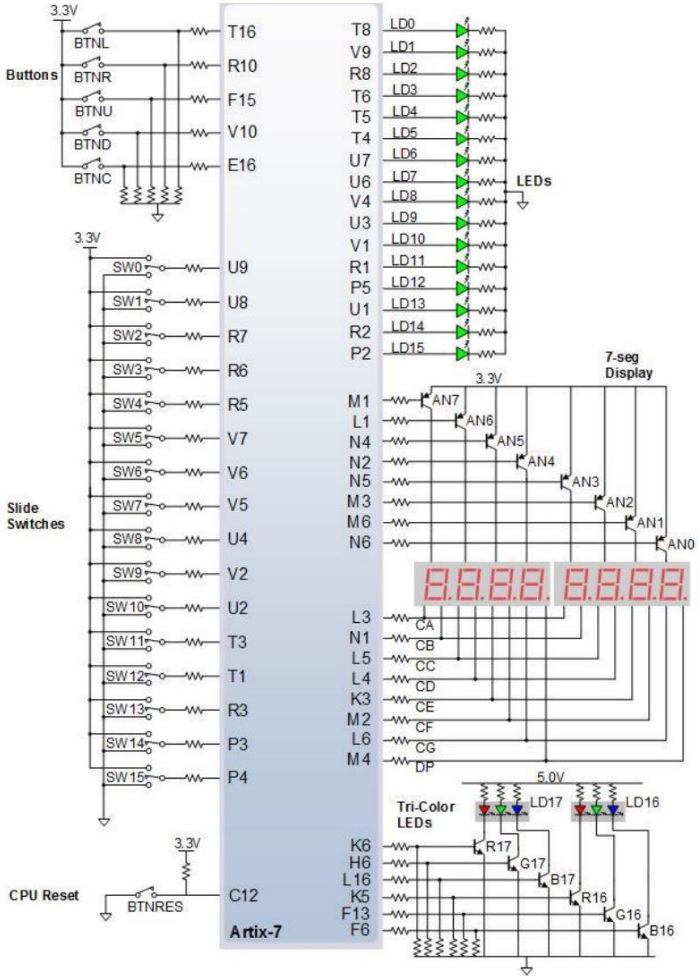
USB-UART Bridge connection

The Nexys 4 includes an FTDI FT2232HQ bridge (attached to connector J6) that allows us use PC applications to communicate with the board using standard Windows COM port commands. The combination of the features into a single device allows the Nexys 4 to be programmed, communicated with via UART, and powered from a computer attached with a single Micro USB cable.
The connections between the FT2232HQ and the Artix-7 are shown in the next picture:



Nexys 4 basic I/O

The Nexys 4 board includes two tri-color LEDs, sixteen slide switches, six push buttons, sixteen individual LEDs, and an eight-digit seven-segment display, as shown in following picture. The pushbuttons and slide switches are connected to the FPGA via series resistors to prevent damage from short circuits. The five pushbuttons arranged in a plus-sign configuration are "momentary" switches that normally generate a low output when they are at rest, and a high output only when they are pressed. In the next picture you can the diagram of the I/O of the Nexys 4.

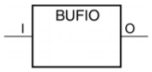


Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Xilinx Artix 7 FPGA Library

BUFIO

Primitive: Local Clock Buffer for I/O



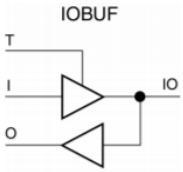
This design element is simply a clock-in, clock-out buffer. It drives a dedicated clock net within the I/O column, independent of the global clock resources. Thus, these elements are ideally suited for source-synchronous data capture (forwarded/receiver clock distribution). They can be driven by a dedicated MRCC I/O located in the same clock region or a BUFMRCE/BUFMR component capable of clocking multiple clock regions. The BUFIO can only drive I/O components within the bank in which they exist. These elements cannot directly drive logic resources (CLB, block RAM, etc.) because the I/O clock network only reaches the I/O column.

Port Descriptions

Port	Type	Width	Function
I	Input	1	Input port to clock buffer. Connect this to an IBUFG connected to a top-level port or an associated BUFMR buffer.
O	Output	1	Output port from clock buffer. Connect this to the clock inputs to synchronous I/O components like the ISERDESE2, OSERDESE2, IDDR, ODDR or register connected directly to an I/O port (inferred or instantiated).

IOBUF

Primitive: Bi-Directional Buffer



The design element is a bidirectional single-ended I/O Buffer used to connect internal logic to an external bidirectional pin.

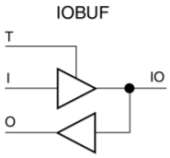
Port Descriptions

Port	Direction	Width	Function
O	Output	1	Buffer output
IO	In/out	1	Buffer In/out
I	Input	1	Buffer input
T	Input	1	3-State enable input

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

IOBUF

Primitive: Bi-Directional Buffer



The design element is a bidirectional single-ended I/O Buffer used to connect internal logic to an external bidirectional pin.

Port Descriptions

Port	Direction	Width	Function
O	Output	1	Buffer output
IO	In/out	1	Buffer In/out
I	Input	1	Buffer input
T	Input	1	3-State enable input

Ascii Table

Hexadecimal and ascii conversions was one of the important aspects of the project since a message was required to be showed using the processor. The next diagram is one of the tools used for conversion of hex and ascii notation.

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	sp
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

WUSB4™

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Requirements

Software Requirements

1) Walk LEDS while in the main
2) Test memory
    i) AAAAs, 5555s, Address, Address Bar
3) Display a banner when starting out of reset and provide a prompt.
4) Echo characters pressed to the terminal, except for the ones with special function.
5) Store characters in SRAM
6) Dump contents of SRAM to the terminal when % is pressed.
7) <CR> sends the cursor to the start of the next line and displays the prompt.
8) <BS> erases the character in front of the cursor but should not erase prompt.
9) "*" results in outputting my home town followed by a newline and a prompt.
10) "@" results in outputting of the number of characters received since reset followed by a newline and a prompt.

Hardware Requirements

This design has is focused on creating a Universal Asynchronous Receiver and Transmitter, which is divided in two big portions. The Transmit engine receives an 8-bit input and sends the bit one bit at a time depending on the specified baud rate. The Receive Engine also takes the same control from the baud rate to receive a bit at a time and gets asserted in shift register. The processor also plays a big role receiving the data from the Receive engine and processing it to execute specific instruction.

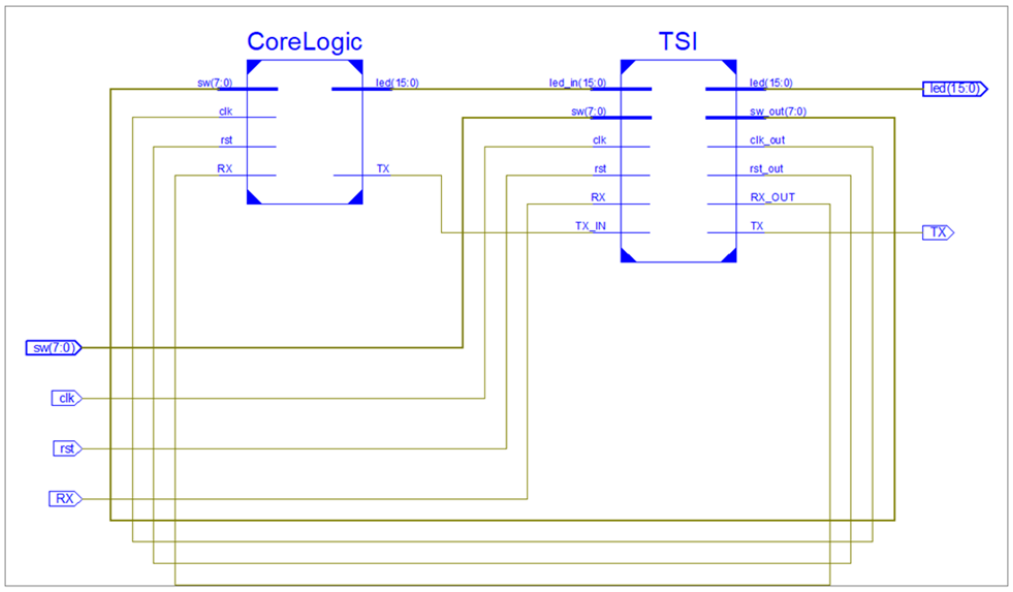
Top Level Design

Description

The Top-level Design is formed by the Core logic, which includes the major blocks of the design. These are the Picoblaze processor, the Static RAM and the UART, which is formed by the Receive and Transmit engines. They contain the most important blocks of the design then the Technology Specific buffers the I/O to the outside world, which contains the electrical and timing requirements of the external interface. This was achieved by looking into the Xilinx library for the Artix 7 FPGA board(Nexys 4).

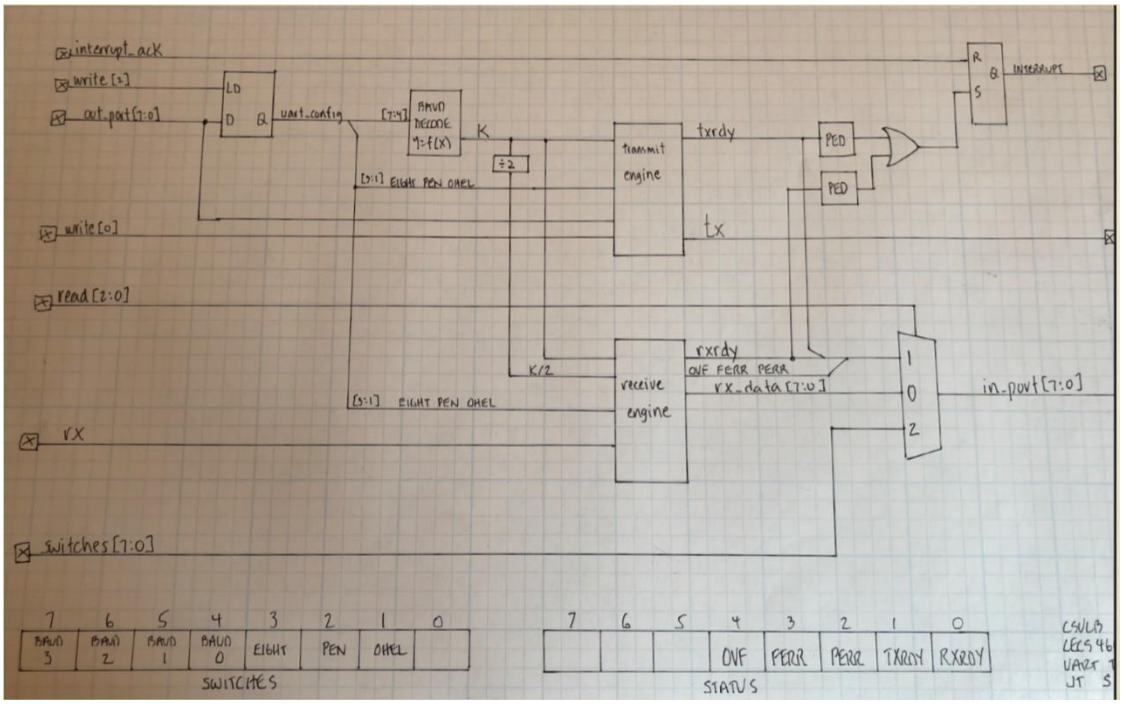
Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Block Diagram



Data Flow Description

Data received comes into the RX input and goes in straight in to the Receive engine. When it is in there the bits gets remapped into an 8-bit frame of data that gets used by the processor, which is also feed the input from the switches and the flags for the receive engine. After the data gets processed, we send it to the transmit engine, in here the data gets spit out one bit a time depending on the baud rate set. The OUT_PORT from the processor also brings the data back from the switches and it gets feed into the decoder for the baud rate. You can see this complex design in the next picture that was given by the professor Tramel.



Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

I/O

Signal Names

i) Inputs:
   (1) Clk – Sourced clock for design
   (2) Baud Switches – 4 Baud rate switches that give the ability to set 12 different baud rate values.
   (3) EIGHT – Enables 8 or 7 switches for data transmission and receiving.
   (4) PEN – Enables parity.
   (5) OHEL - Parity is high when switch is high or low when down.
   (6) RX – Input for UART to receive data.

ii) Outputs
   (1) TX – Sends serial data out by the given baud rate.
   (2) LEDS – Rotate LEDs while in main loop

Pin Assignments

Signal Name	PIN Assigned
CLK	LOC = "E3"
RST	LOC=J15
SW<1>	LOC=L16
SW<2>	LOC=M13
SW<3>	LOC=R15
SW<4>	LOC=R17
SW<5>	LOC=T18
SW<6>	LOC=U18
SW<7>	LOC=R13
LED<0>	LOC=H17
LED<1>	LOC=K15
LED<2>	LOC=J13
LED<3>	LOC=N14
LED<4>	LOC=R18
LED<5>	LOC=V17
LED<6>	LOC=U17
LED<7>	LOC=U16
LED<8>	LOC=V16
LED<9>	LOC=T15
LED<10>	LOC=U14
LED<11>	LOC=T16
LED<12>	LOC=V15
LED<13>	LOC=V14
LED<14>	LOC=V12
LED<15>	LOC=V11
RX	LOC=C4
TX	LOC=D4

Electrical Characteristics

The UART chip created did not require any high amounts of power, therefore we were able to get away using the low power mode given by the Xilinx library. The design also uses a slow slew rate to avoid ringing since we want to keep the integrity of the data and the UART is not fast where we would require a faster rate of change in voltage.

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Clocks

The clock source for this project came from the Nexys 4 board. It has a frequency of 100MHz with clock period of 10 Micro-seconds.

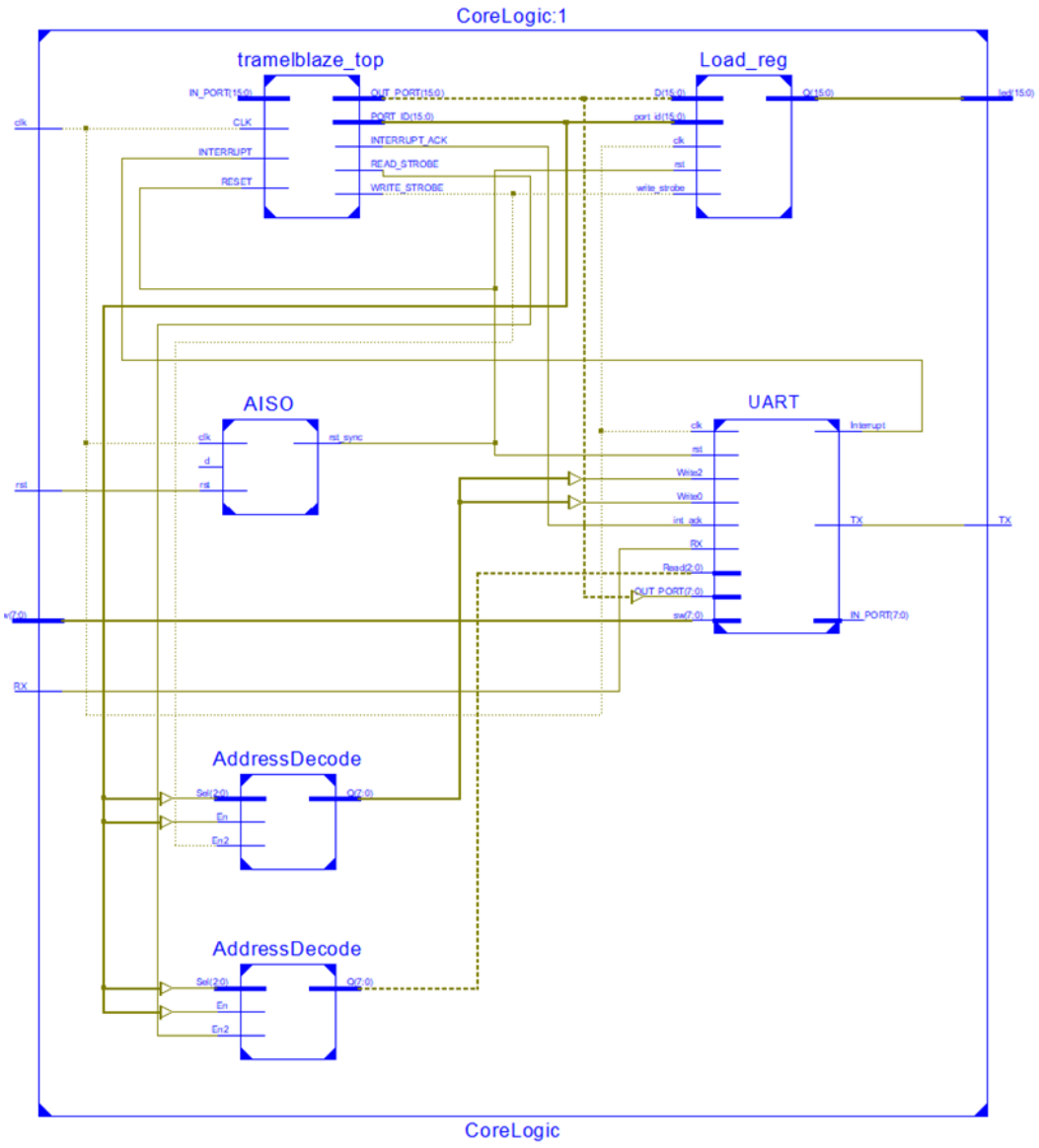
Reset

The asynchronous reset was source from one of the push buttons, then was sent through a pair of flip flops to achieve a synchronous output for the design. This is done to avoid setting reset in a metastable state, since the push button is not synchronized with the clock.

Software

Description

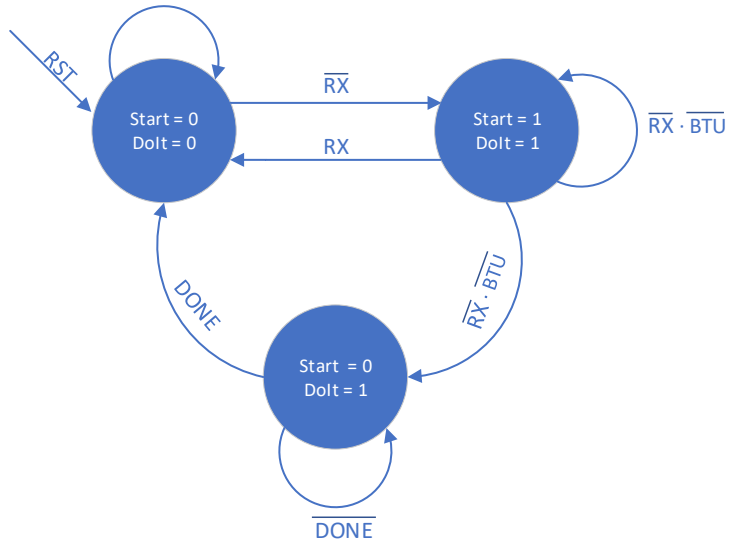
The goal for this project is to generate a UART chip that is also used by a processor to do some straightforward operations, the top level description of the design shows the connections that were necessary to connect the UART with the Processor and I/O.



Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Detailed Software Planning document

State Machine used to count the number of bits received in the received engine



Specific clock counts to achieve correct baud rate with the given input from 4 switches

baud[3:0]	rate	bit time	Eng Not	N3/N4 Count
0000	300	0.003333333	3.3333 ms	333,333
0001	1200	0.000833333	833.33 us	83,333
0010	2400	0.000416667	416.66 us	41,667
0011	4800	0.000208333	208.33 us	20,833
0100	9600	0.000104167	104.16 us	10,417
0101	19200	5.20833E-05	52.083 us	5,208
0110	38400	2.60417E-05	26.041 us	2,604
0111	57600	1.73611E-05	17.361 us	1,736
1000	115200	8.68056E-06	8.6806 us	868
1001	230400	4.34028E-06	4.3403 us	434
1010	460800	2.17014E-06	2.1701 us	217
1011	921600	1.08507E-06	1.0851 us	109

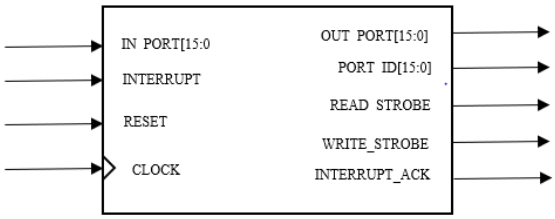
Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Processor (TramelBlaze)

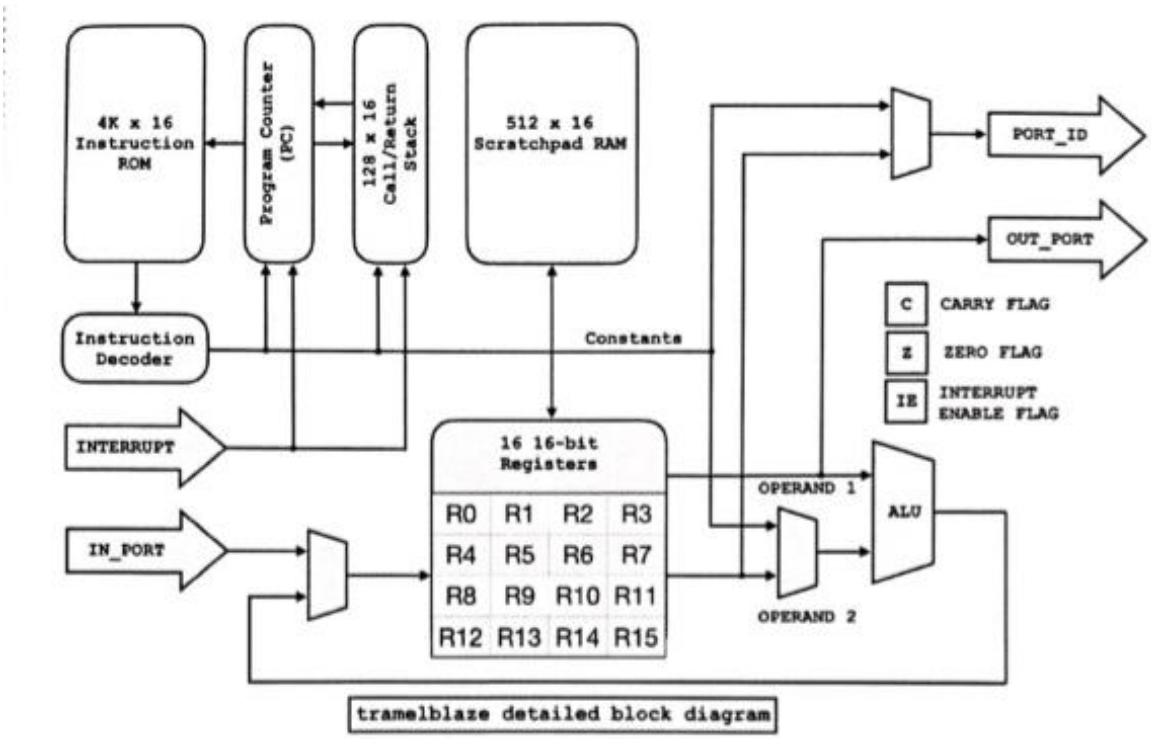
Description

For the processor of the design we used an emulator of the Pico blaze, called TramelBlaze give by the instructor. The PicoBlaze was an 8-bit RISC processor, and the only difference would be that the emulator is 16-bits. Since the PicoBlaze processor was meant to be used by the Nexys3, we use an updated version for the Nexys4 with an added feature. The goal for the processor in this UART project is to process the data it's sending out to the transmit engine. It takes care of computing the binary to hex for the counting, and it also fetches the hex values that are stored in the Scratchpad for the letters to output a message in the terminal.

Top Level



Detailed Block Diagram



Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

I/O

i) Inputs:
   (1) IN_PORT: 16-bit input for the processor.
   (2) INTERRUPT: triggers the interrupt service routine.
   (3) RESET: Asynchronous reset that comes from the AISO.
   (4) CLOCK: 100MHz form board.

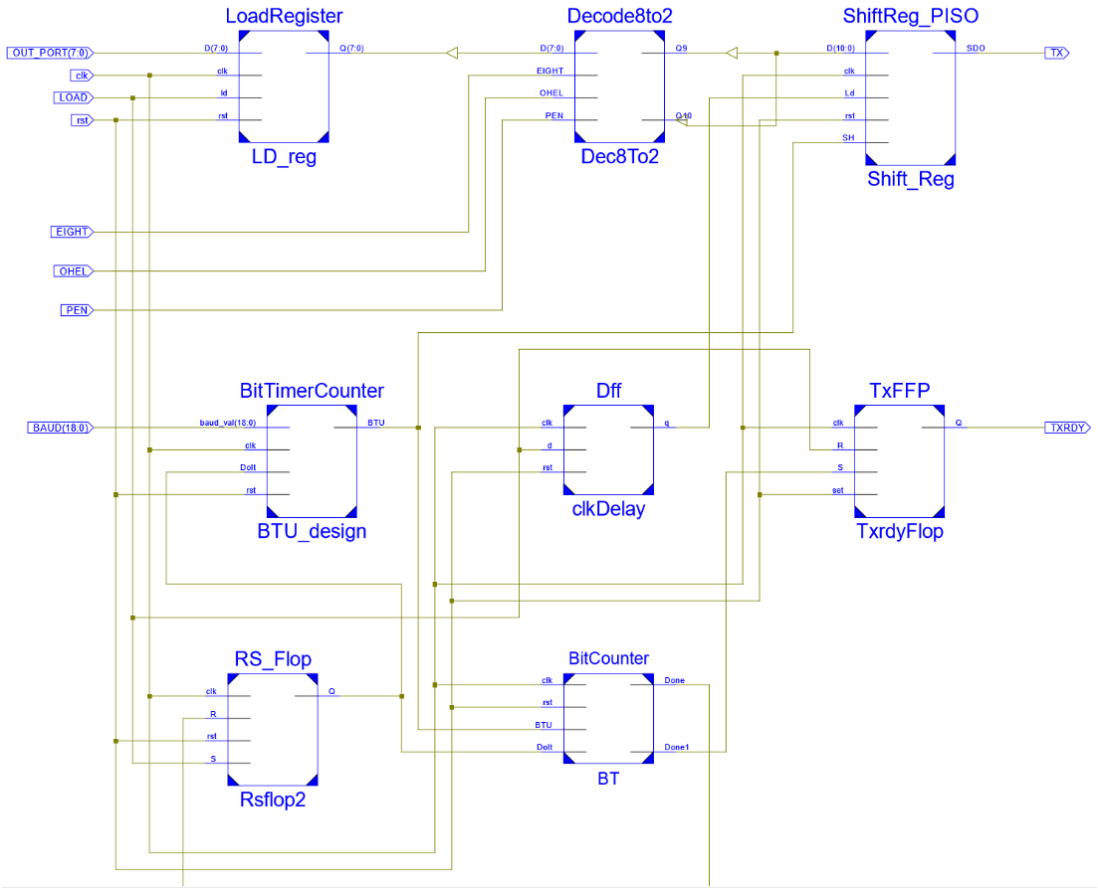
ii) Outputs:
   (1) OUT_PORT: 16-bit output form the processor.
   (2) PORT_ID: 16 bits for identification of output port.
   (3) READ_STROBE: goes high when an output is in process.
   (4) WRITE_STROBE: goes high when an input is in process.
   (5) INTERRUPT_ACK: acknowledges when an interrupt is received.

Transmit Engine

Description

The transmit engine took inputs for a specific baud rate, which range from 300 to 921600, this value was calculated by counting the number of clock ticks from the nexys4. For an example, the 9600 baud rate we counted to a value of 10417 with the 100Mhz clock. The data to transmit comes into the OUT_PORT which was sent from the processor, in this case we used the TramelBlaze, which was given to us. The processor would send a write strobe signal to the load register, then the engine would start to process the data and sent it out from the shift register to the TX port.

Detailed Block Diagram



Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

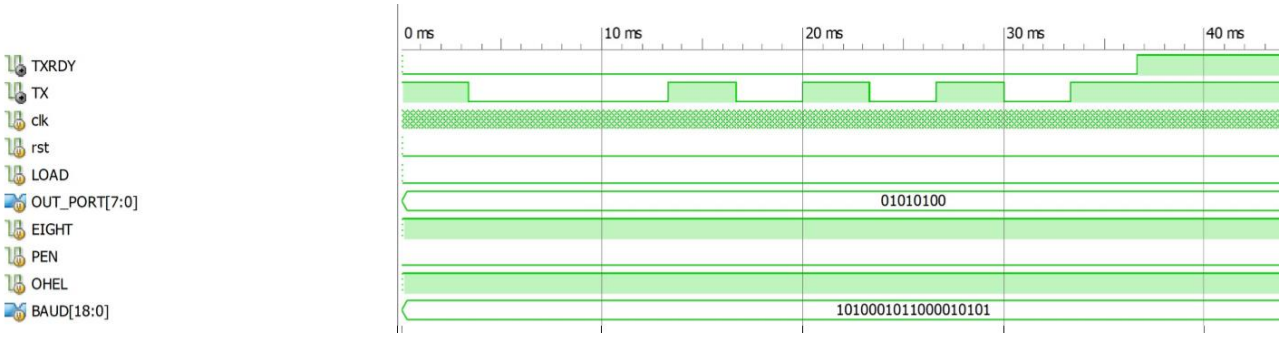
I/O

i) Inputs:
   (1) clk - 100MHz from board.
   (2) rst - Asynchronous input that goes into the AISO to get a synchronous output for the reset for the design.
   (3) EIGHT - sets a 7 or 8 bit to transfer data.
   (4) PEN - enable parity.
   (5) OHEL - Odd parity when switch is high, even parity when switch is low.
   (6) LOAD - signal to start shifting the data out
   (7) BAUD – speed to shift the data out the TX port

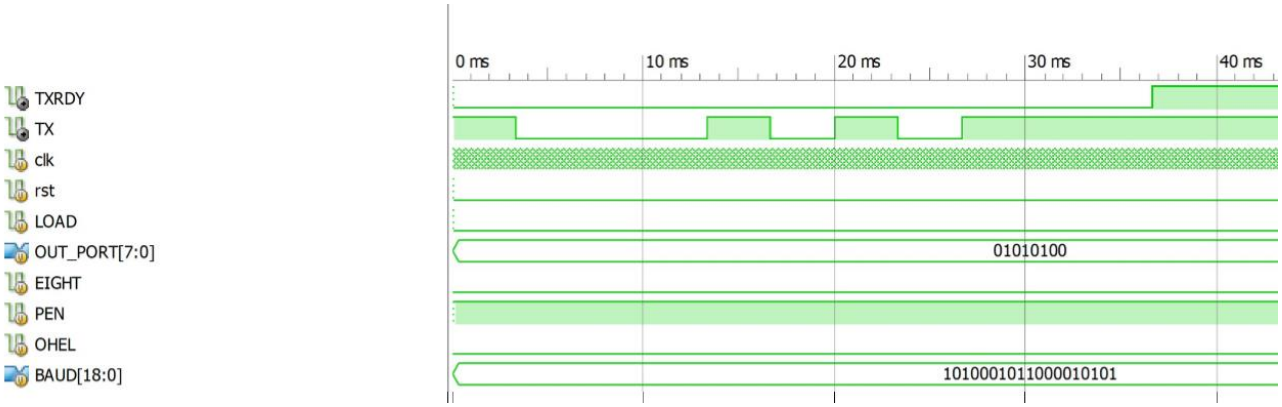
ii) Outputs:
   (1) TX – Transmits the data bits
   (2) TXRDY – flag for when we start to send the data

Verification

In the verification for the transmit engine I sent 0x54 to the OUT_PORT and set load active for one clock tick. TXRDY is asserted when coming off reset, then its ready to send the data out, that's when the shift register starts sending the least significant bits of the data through the TX port. In this case we see the start bit(0), and the stop bit(1) at the end followed by the MSB(0).



In the next waveform the parity bit was exercised with the choice of 7 bits of data. The OUT_PORT had the same choice data 0x54 but since it's 8 bits, the MSB gets cutoff and replaced by the parity bit(1), because odd parity was enabled.



Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

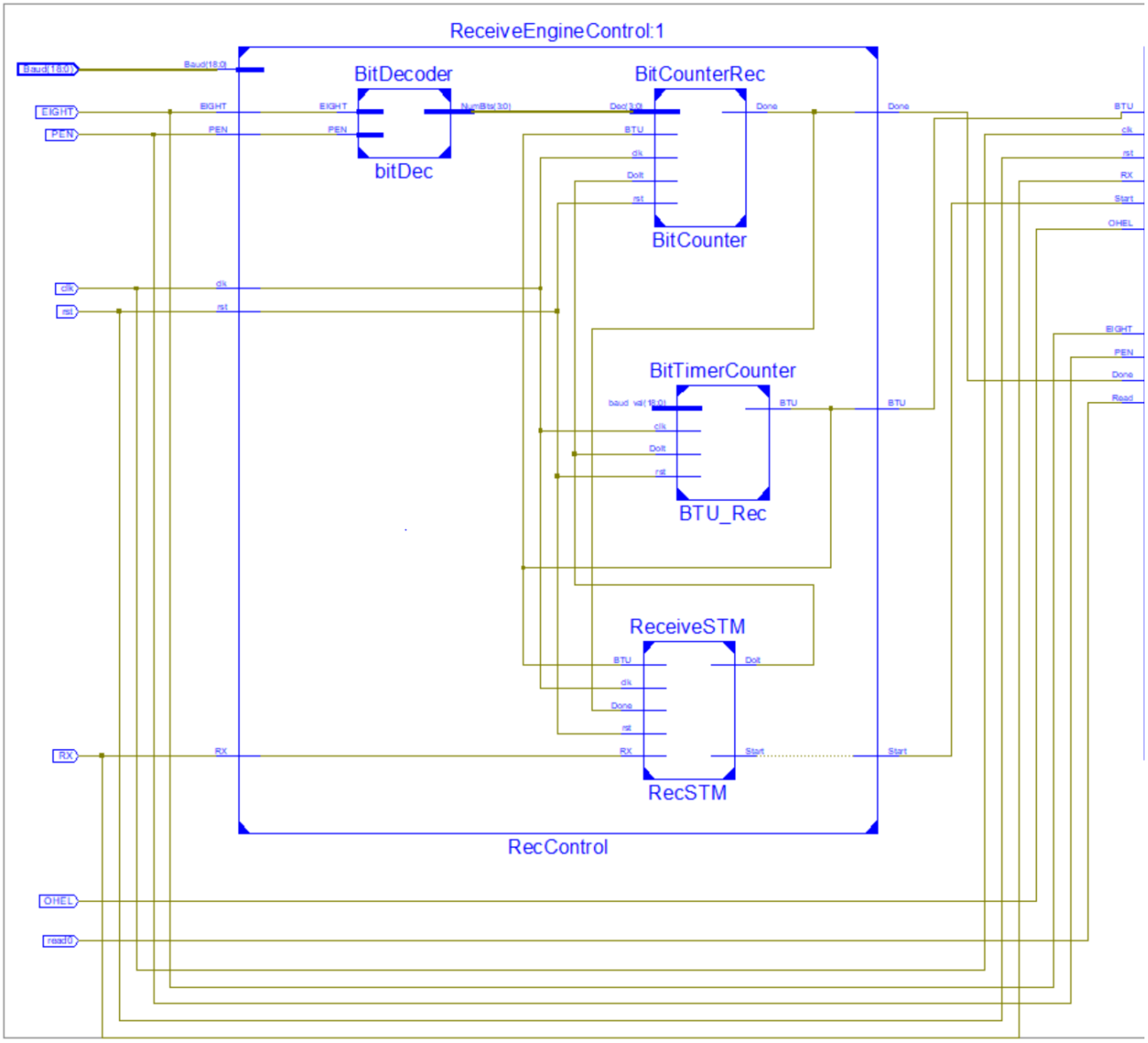
Receive Engine

Description

The receive engine receives data at a specific baud rate set by the decode of the switches. The data is receiving by the RX line, then that data is sent to a Shift register, which then is sent to a remap that aligns the data to be correctly readable for our processor. The process of reading the specific number of bits is controlled by the receive engine control, it reads 8 or seven bits depending on the EIGHT switch. The engine control also has a state machine that waits for a start bit and changes the state of START and DoIt, which are used for the Datapath, which is in charge of remapping the data and sending it to the processor.

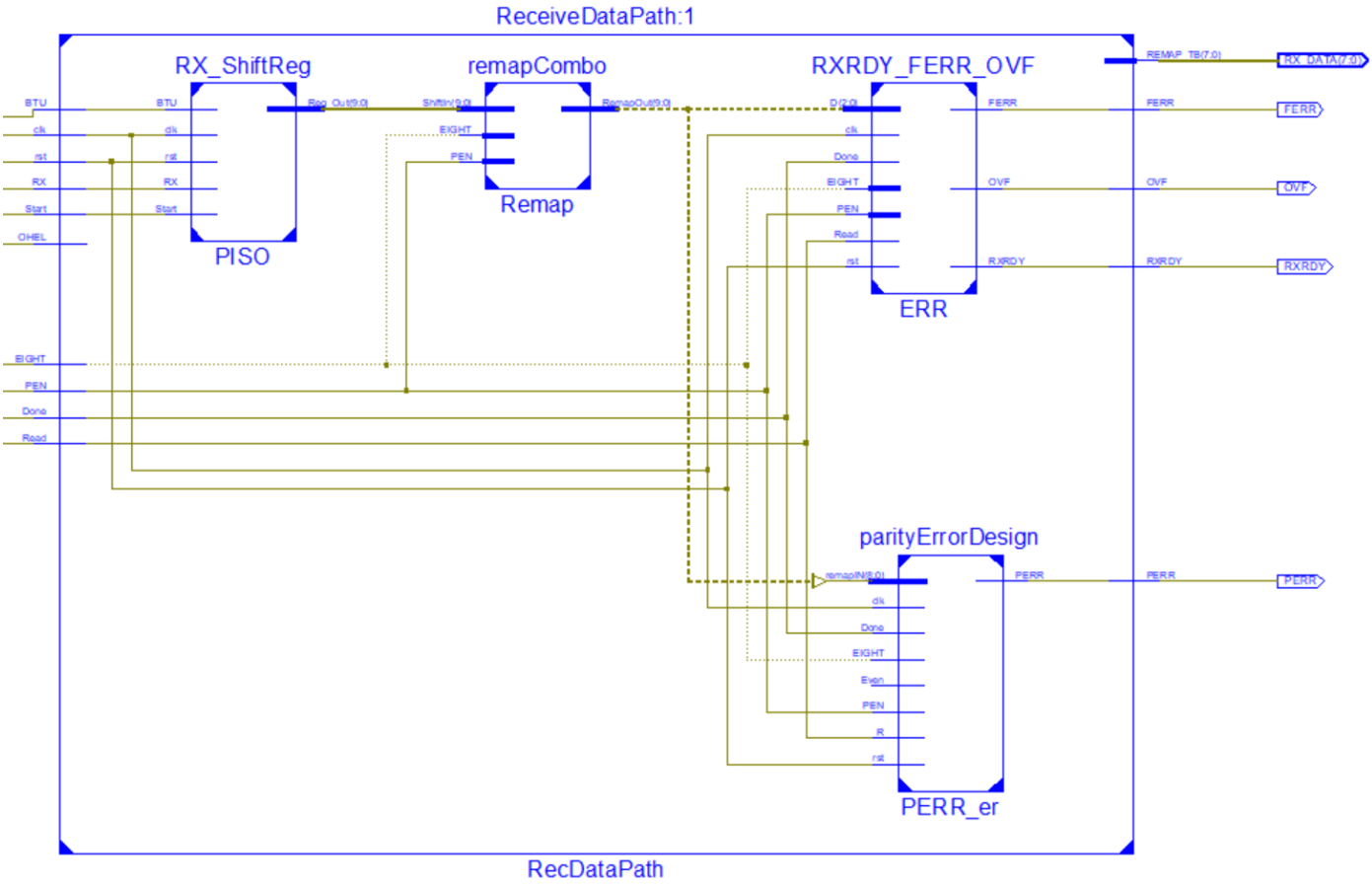
Detailed Block Diagram

Part 1: Receive Engine Control



Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Part 2: Data Path



I/O

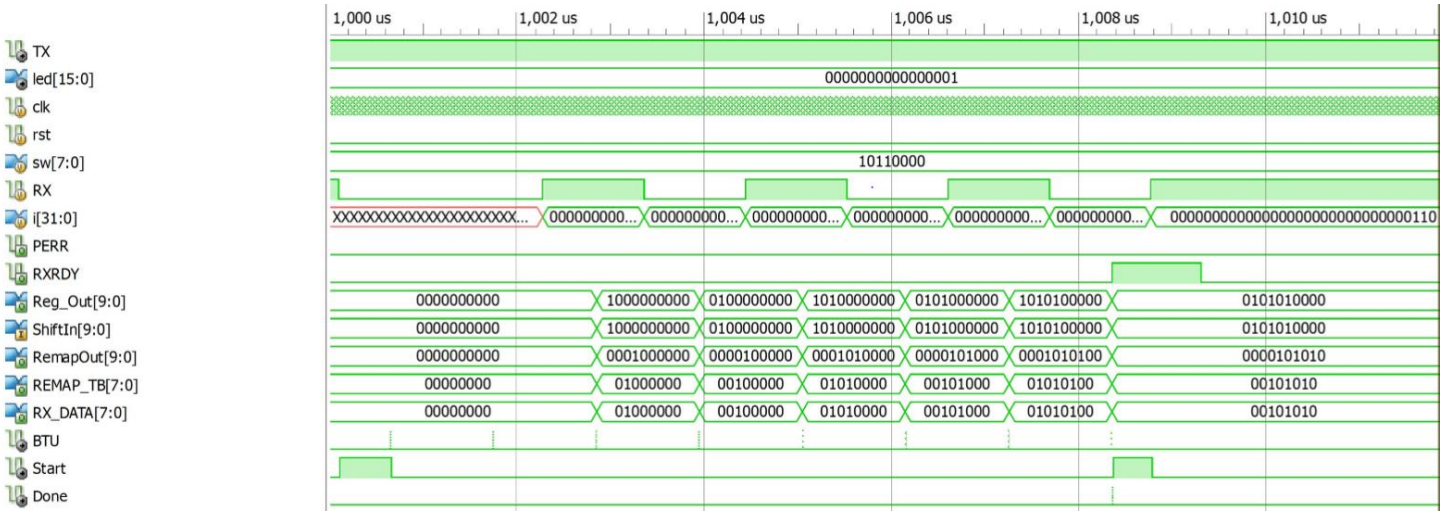
i) Inputs
   (1) Baud – Decoded value set by the switches
   (2) EIGHT – Ability to read inputs in 7- or 8-bit frames
   (3) PEN – Parity Enable switch
   (4) Clk – Clock sourced by the Nexys4
   (5) Rst – synchronous inputs for reset
   (6) RX – Input to receive the data, one bit at a time, specified by the baud Rate
   (7) OHEL – Switch that sets odd parity while high, and even parity while low
   (8) Read – input that clears the flags for overflow, framing, and parity error.

ii) Outputs
   (1) RX_DATA –Data that was remapped to be sent to the processor for execution
   (2) FERR – Framing error flag
   (3) OVF – Overflow Flag
   (4) RXRDY – Receive ready flag
   (5) PERR – Parity error

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Verification

For the verification of the Transmit engine, I sent a 0x54 in the RX line with serial communication after the start bit was set low , every bit after was sent every 1.085 usec. Then RX sent the bits into the shift register, that can be seen in the Reg_Out port which is sending the data into the remap to assert the binary data correctly depending on parity enable, or if we sending 8 or 7 bits. After the data is asserted correctly we sent to the processor in the RX_DATA Port.



Static Random-Access Memory

Description

The SRAM is used to store the characters echoed to the screen, then they get dumped out to the screen when the % character is pressed. The processor sends the characters to the SRAM to be saved using Port_ID 15 as the Write Enable since we only write a max of 32k values. The Address where the data is stored is set by the remaining values of Port_ID 0 – 14, and followed by the data which came from the OUT_PORT of the processor. After the data is stored in the memory, we wait until the data needs to be seen by the processor, which is specified by the address port then sent in Dout. Using the External memory allows us to have memory not depending in the processor but also be usable by it. We are able to store much more data, which gives us a great amount of flexibility using this approach.

Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

Detailed Block Diagram

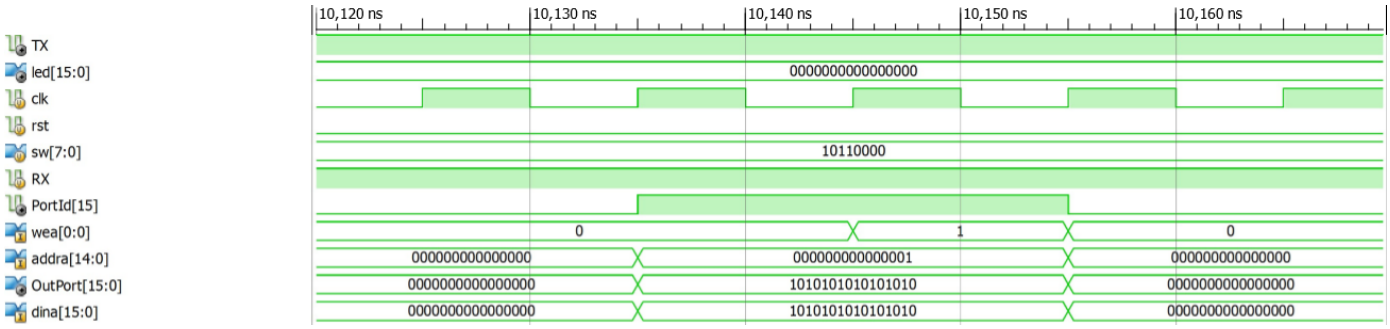
I/O

iii) Inputs
- (1) Clk – Clock sourced by the Nexys4
- (2) WEA – Write enable to write to ram.
- (3) ADDR – Address where data is going to be stored.
- (4) Din – Data sent to ram

iv) Outputs
- (1) Dout – Data transmitted out of memory to processor.

Verification

The simulation portion proved the contents of the memory for the memory test. In the first case the memory gets loaded with AAAA, then the contents get changed to 5555. To load the contents in the memory we can see that PORT_ID 15 goes high for 2 clock ticks, which is an output characteristic by our processor, then the write enable goes high for a clock cycle and the data from the OUT_PORT gets copied to the Din port and stored in the memory by the specified address.



In the next picture we can see the actual values of the SRAM being loaded for AAAA test after boot, verifying the data writing in the memory,

Address: | Columns: auto | Address Radix:

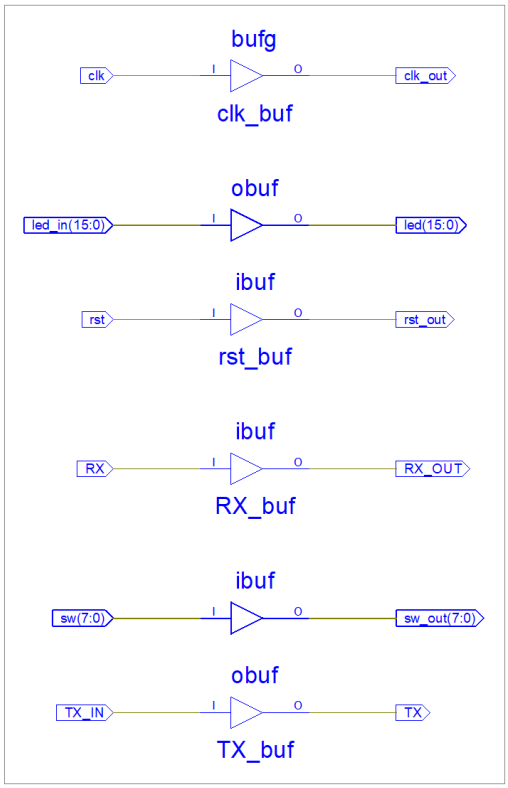
	0	1	2	3	4	5	6	7
0x108	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA
0x110	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA
0x118	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA
0x120	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA
0x128	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA
0x130	0000	0000	0000	0000	0000	0000	0000	0000
0x138	0000	0000	0000	0000	0000	0000	0000	0000

Technology Specific Instantiation

Description

Technology Specific Instantiations (TSI) contains all references to the target technology library. All communication to or from the I/O of the device pass through the TSI. The core logic contains the major functionalities of the design, this block contains the UART, which is formed by the Transmit and receive engine, this block also contains the Processor and the Static RAM. The TSI is simply a buffer for the inputs and outputs so they do not a directly interface to the outside world. This helps in keeping track of each I/O of the chip, since they must meet the electrical and timing requirements of the external interface.

Detailed Block Diagram



Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

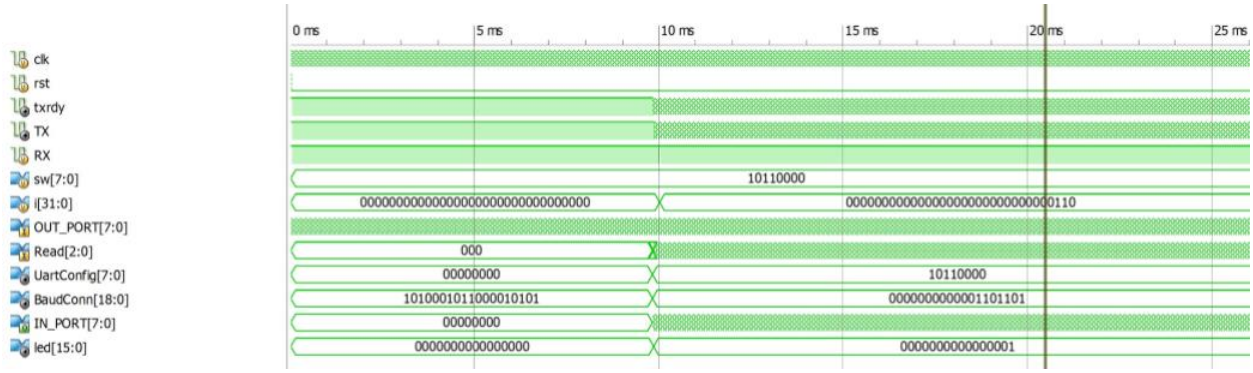
I/O

i) Inputs
   (1) Clk – Clock sourced by the Nexys 4,
   (2) led_in – Output from the processor to rotate leds.
   (3) Rst – synchronous inputs for reset
   (4) RX – Input to receive the data, one bit at a time, specified by the baud Rate
   (5) Sw – Input buffer from the Switches of Nexys 4
   (6) TX_IN- input buffered to the TSI from the CoreLogic.

ii) Outputs
   (1) Clk out – Clock buffered to the design
   (2) led – Output buffered to the Leds from the board.
   (3) Rst out – buffered input for reset
   (4) RX OUT– buffered from the received signals of the board
   (5) Sw – Output buffer from the Switches of Nexys 4
   (6) TX - Output from the CoreLogic to the external device.

Verification

The next waveform proves the design has the same characteristics from the previous verifications of the Receive and Transmit engines. The UART configuration is kept as the same value entered from the Switches, so the Baud rate gets set to the highest value as was previously proven. After the first memory test is completed, we can se the Transmit Engine sending out characters displaying the contents of the memory for the verification. The device keeps the same characteristics from the original core logic which is expected since the signals are just buffered though the TSI.

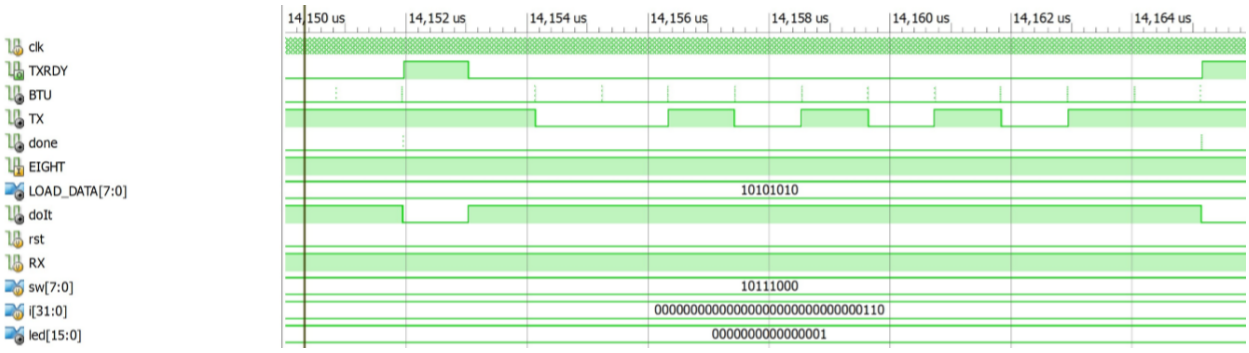


Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

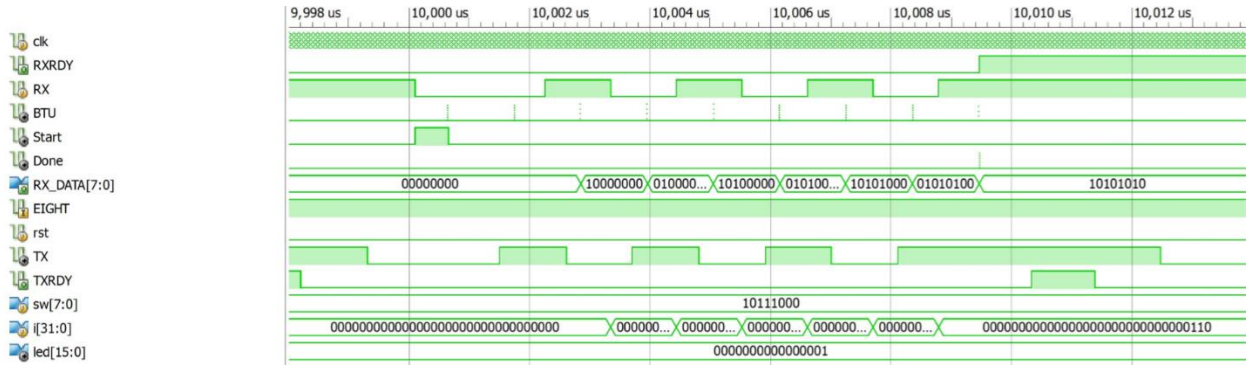
Chip Level Verification

The chip created for this project was the UART, which is made by the transmit and engine. In this verification is will prove the UART is able to interface with the processor and it is able to output data sent to the shift register of the Transmit engine. I will also prove the ability to read data sent into the UART by the receive engine.

Outputting the AAAAs memory test, (01010101) 8-bit mode. TXRDY and doIt go active.



Sending bits to the RX line(01010101) MSB first. Remapped data in RX_DATA is arranged correctly for processor to use.

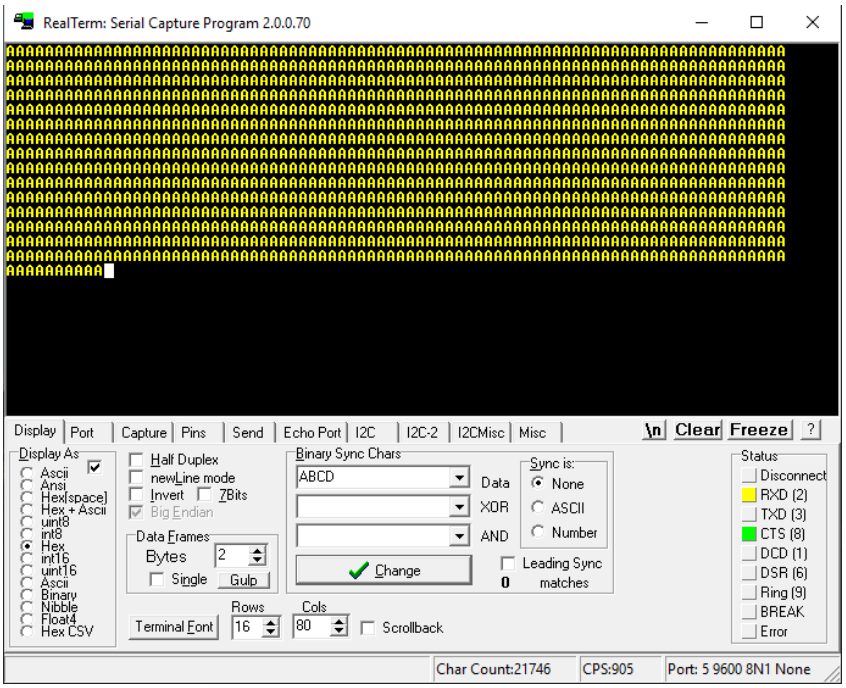


Prepared by:	Date:	Document Number and Filename:	Revision:
Cristian Lopez	May 14, 2019	CECS490_ChipSpec.pdf	1

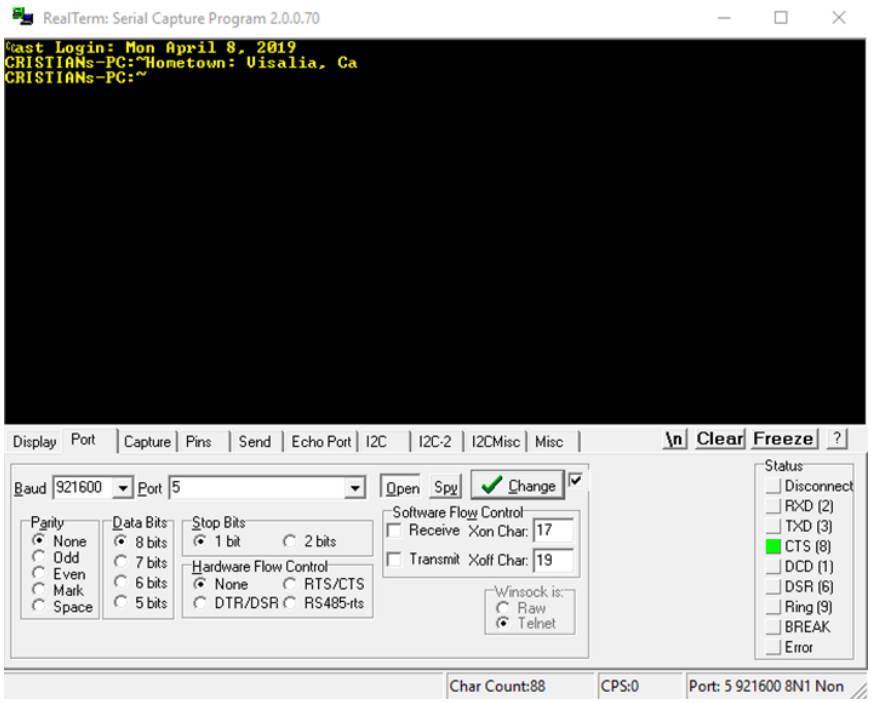
Chip Level Test

The chip was tested using real term at 9600 and 921600 which was the fastest available. All the requirements of the project were tested and proven in the upcoming pictures.

Printing results of AAAAs memory test after boot

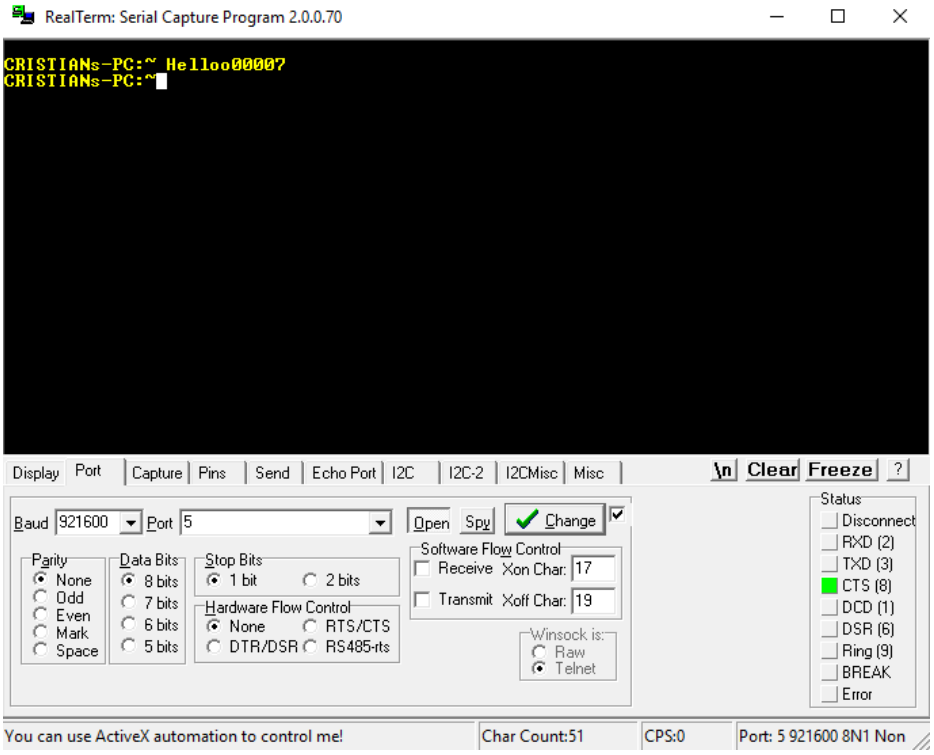


Hometown is shown after asterisk is pressed ""*



Prepared by: Cristian Lopez	Date: May 14, 2019	Document Number and Filename: CECS490_ChipSpec.pdf	Revision: 1

Number of characters entered get shown after @ is pressed.



Characters in SRAM get dumped after '%' is pressed.