TramelBlaze System-On-Chip Specification

Aaron Mai
Technical Report CECS 460
December 12, 2020

College of Computer Engineering and Science
California State University, Long Beach

aaron.mai@student.csulb.edu

Abstract

Within this report I will describe and define the chip specification of the TramelBlaze SOC. I will be using a methodology and style heavily influenced by Tramel's chip specification template from his time at Valence Semiconductor, UCI's style guide for SOC specifications, the 8051 Microcontroller data sheet as well as other various datasheets and chip specifications. What I hope to accomplish is to document our full UART, a project culminating from our past work: the transmit + receive engine. Within this chip specification, descriptions of code, functions, and diagrams as well as commentary of my thought process will be provided in a chip spec style that I hope to "own" and develop/refine throughout my engineering career. Enjoy the read, best Aaron.

Prepared by	Loc/Dept	Date	Document Number and File Name	Revision :3
Aaron Mai	CECS	December 11, 2020	Full UART Chip Spec.	

TABLE OF CONTENTS

1 Introduction

This document will provide the chip specification for the Universal Asynchronous Receiver (UART) that has been developed in our senior SOC design course CECS 460, taught by the expertise of John Tramel. Our beloved professor for the course created the microprocessor that runs this show, dubbed the TramelBlaze, the microprocessor processes assembly code into functions for our UART. The UART is composed of two engines: The Transmit (TX) engine and the Recieve Engine(RX). Indicative of their names the former transmits data while the latter receives data. Our design will be built and verified by key features and building blocks such as registers, state machines, a Positive Edge Detect (PED) module, and a Asynchronous-In-Synchronous-Out Reset (AISO).

1.1 Purpose

The objectives we are to achieve will be to create a functioning UART transmission that consists of several key features. Features will be listed below.

1. Have options for selectable baud rate
2. Have options for bits transmitted
3. Have an option for parity enable/disable
4. Have an option for odd/even parity bit

2 External Documents

2.1 PicoBlaze User Guide

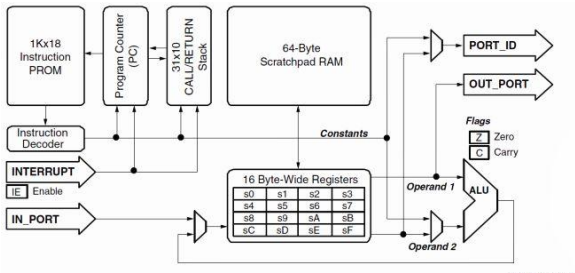


Figure 1-1: **PicoBlaze Embedded Microcontroller Block Diagram**

2.1 PicoBlaze User Guide (cont.)

Picoblaze block diagram is shown prior.
The Picoblaze is the foundation of the TramelBlaze and both share many similarities and identical architecture. However the key difference is that the TramelBlaze's instruction ROM will contain 4Kx16, its stack RAM is 512x16, and its bus lines are 16 bits. Further information on the picoblaze can be found within the xilinx documentation web page. (https://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf)

Picoblaze Instruction Set Table Shown Below

Table 3-1: **PicoBlaze Instruction Set (alphabetical listing)**

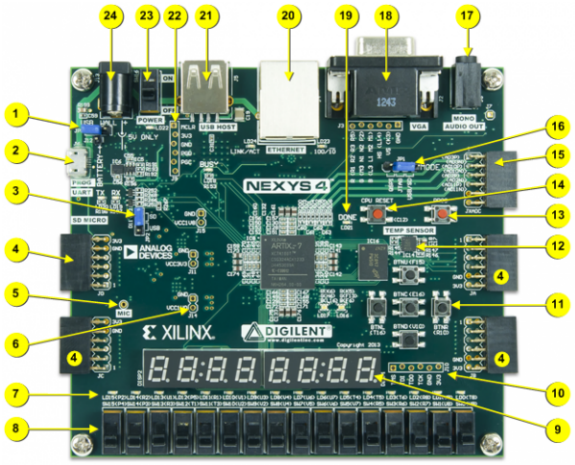
Instruction	Description	Function	ZERO	CARRY
ADD sX, kk	Add register sX with literal kk	sX ← sX + kk	?	?
ADD sX, sY	Add register sX with register sY	sX ← sX + sY	?	?
ADDCY sX, kk (ADDC)	Add register sX with literal kk with CARRY bit	sX ← sX + kk + CARRY	?	?
ADDCY sX, sY (ADDC)	Add register sX with register 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}	-	-
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	-	-

2.1 PicoBlaze User Guide (cont.)

Instruction	Description	Function	ZERO	C
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	?	
OR sX, sY	Bitwise OR register sX with register sY	sX ← sX OR sY	?	
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	-	
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	-	

Instruction	Description	Function	ZERO	CARRY
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	-	-
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	?	?

2.2 Nexys4 Artix7 Documentation



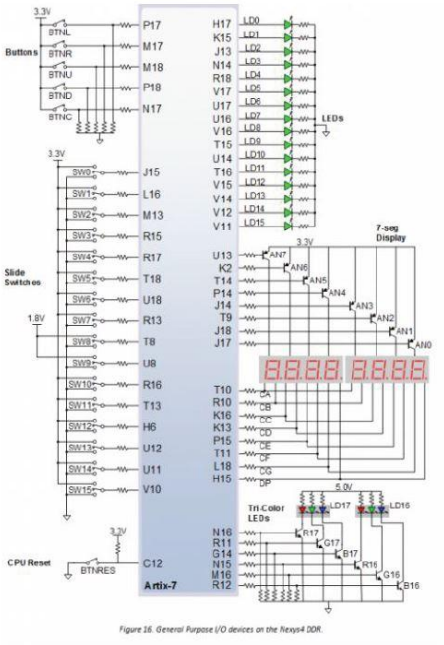


Figure 1. shown above is the NEXYS4 Artix7, the board which we will run our UART protocol on

Figure 2. GPIO schematic of the Board, documents buttons, switches, cpu reset, rgb leds, 7seg display, leds

2.2 Nexys4 Artix7 Documentation (Cont.)

Further documentation can be found on the Digilent webpage. (https://reference.digilentinc.com/reference/programmable-logic/nexys-4/start)

2.3 ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Full ASCII table can be found in the link below (*http://www.asciitable.com/*)

3.1 Requirements

Our requirements will be split into 2 subcategories: Interface & Hardware. Interface will preface hardware specifications in our design.

3.1 Interface Requirements

The design consists of the UART and the TramelBlaze, which work in tandem and conjunction in order to utilize the inputs and outputs (I/O) from the Nexys 4 Artix 7 board. The design will consist of eleven selectable Baud rates through the Nexys 4 on board switches, a three bit parity control, a parity enable/disable, and an odd or even parity select. An additional feature is serial output through the TX port and indicate the microprocessor with a Txrdy signal.

3.2 Hardware Requirements

We are implementing and using the on board seven switches and one button. The upper onboard button is used in our design for the reset signal. Switches 7 to 4 are used for baud rate selection. Switches 3 to 1 are used for parity checking. Switch 3 will enable eight bit data. Switch 2 will enable the parity bit. Switch 1 will check if it is an odd or even parity bit.

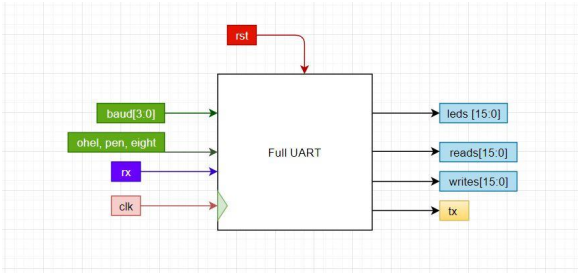
BTN	Reset signal
SW1	Parity check/ Even or Odd check
SW2	Parity check/Enable Parity
SW3	Parity check/ Enable 8 bit data
SW4	Baud rate
SW5	Baud rate
SW6	Baud rate
SW7	Baud rate

4 Top Level

4.1 Description

The top level contains the entire digital design utilizing both the TramelBlaze and UART engine. 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 (<cr> <bs> * @) on the Nexys4 Board. Furthermore the design will be outputting to the program Realterm. We will discuss the program later in the UART engine section.

4.2 Block Diagram



Inputs

Writes[6] - Signal from the address decode to load the UART

Out_port[7:0] - 8 bit signal for the onboard switches

RX - Signal indicating that the RX engine can begin receiving serial data

Writes[0] - Signal from the address decode to the TX engine load

Reads[1:0] - Signal for the select of the RX engine

Outputs

UART_DS[7:0] - 8 bit signal going to the TramelBlaze

UART_INT - Signal that goes through the RS flop of the TramelBlaze

TX - Signal with transmission serial data going to the terminal program

4.3 Data Flow Description

These sections will be further developed in the UART section 6.1 along with a detailed top level diagram. To preface that section though, in this design our reset signal will be utilizing our AISO module. Our UART engine will utilize our past projects, the tx and rx engine to transmit and receive data. The engine utilizes the Tramelblaze to both drive the LEDS as well as execute instructions from the UART. Simultaneously our address decoder module indicative of its name will decode the read and write strobe data and write to correct memory paths.

4.4.2 Signal Names + Pin assignments

Signal Name	Pin Location
clk	E3
ohel	L16
pen	M13
eight	R15
baud[0]	R17
baud[1]	T18
baud[2]	U18
baud[3]	R13
rst	N17
Tx	D4
Rx	C4
leds[0]	H17
leds[1]	K15
leds[2]	J13
leds[3]	N14

Signal Name	Pin Location
leds[4]	R18
leds[5]	V17
leds[6]	U17
leds[7]	U16
leds[8]	V16
leds[9]	T15
leds[10]	U14
leds[11]	T16
leds[12]	V15
leds[13]	V14
leds[14]	V12
leds[15]	V11
----------	----------
----------	----------
----------	----------

4.4.1 I/O

4.5 Software

```
;Aaron Mai
;Full UART Assembly Code
;CECS 460 Fall 2020

;=================================================
;          ASCII & Constant DECLARATIONS
;=================================================
ascii_null      EQU   0000
ascii_BS        EQU   0008  ; backspace
ascii_LF        EQU   000A  ; <LF>
ascii_CR        EQU   000D  ; <CR>
ascii_aster     EQU   002A  ; *
ascii_AT        EQU   0040  ; @
ascii_DOT       EQU   002E  ; .
ascii_EQ        EQU   003D  ; =
ascii_DASH      EQU   002D  ; - dash
ascii_COL       EQU   003A  ; colon
ascii_SP        EQU   0020  ; space

ascii_0         EQU   0030
ascii_1         EQU   0031
ascii_2         EQU   0032
ascii_3         EQU   0033
ascii_4         EQU   0034
ascii_5         EQU   0035
ascii_6         EQU   0036
ascii_7         EQU   0037
ascii_8         EQU   0038
ascii_9         EQU   0039
ascii_A         EQU   0041
ascii_B         EQU   0042
ascii_C         EQU   0043
ascii_D         EQU   0044
ascii_E         EQU   0045
ascii_F         EQU   0046
ascii_G         EQU   0047
ascii_H         EQU   0048
```

```
ascii_I         EQU   0049
ascii_J         EQU   004A
ascii_K         EQU   004B
ascii_L         EQU   004C
ascii_M         EQU   004D
ascii_N         EQU   004E
ascii_O         EQU   004F
ascii_P         EQU   0050
ascii_Q         EQU   0051
ascii_R         EQU   0052
ascii_S         EQU   0053
ascii_T         EQU   0054
ascii_U         EQU   0055
ascii_V         EQU   0056
ascii_W         EQU   0057
ascii_X         EQU   0058
ascii_Y         EQU   0059
ascii_Z         EQU   005A

FORTY           EQU   0028
ZERO            EQU   0000
ONE             EQU   0001

;=================================================
;          ADDRESSES FOR OUTPUTS
;=================================================
BAN_START          EQU     0000
BAN_END            EQU     0098

PROMPT_START     EQU     0098
PROMPT_END       EQU     00A4

HOMETW_START     EQU     00A4
HOMETW_END       EQU     00BD

BACKSPCE_START   EQU     00BD
BACKSPCE_END     EQU     00C0

CRLF_START       EQU     00C0
CRLF_END         EQU     00C2
```

```
;=================================================
;          REGISTER DECLARATIONS
;=================================================
CHAR_REG        EQU     R0      ; holds temporary ascii characters
STATUS          EQU     R1      ; holds status signals
DATA            EQU     R2      ; holds received data
POINTER         EQU     R3      ; holds memory address
LED_COUNT       EQU     R4      ; holds LED delay
DELAY_COUNT     EQU     R5      ; holds delay value
LEDS            EQU     R6      ; holds LED value
CONFIG          EQU     R8
CASE            EQU     RC      ; holds current case based on user's input
CHAR_COUNT      EQU     RD      ; holds number of inputted characters

;=================================================
;          INITIALIZATIONS
;=================================================
START
        LOAD    CHAR_REG,       ZERO
        LOAD    CHAR_COUNT,     ZERO
        LOAD    STATUS,         ZERO
        LOAD    DATA,           ZERO
        LOAD    CASE,           ONE
        LOAD    CONFIG,         ZERO

        LOAD    LED_COUNT,      ZERO
        LOAD    LEDS,           ONE
        LOAD    DELAY_COUNT,    ZERO

        CALL    banner_init
        CALL    prompt_init
        CALL    hometown_init
        CALL    bs_init
        CALL    crlf_init

        LOAD    POINTER,      0000
        ENINT
```

```
;==================================================
;          MAIN LOOP TO WALK ONBOARD LEDS
;==================================================
MAIN
      INPUT CONFIG, 0002
      OUTPUT CONFIG, 0006
      ADD LED_COUNT, ONE
      ADDC DELAY_COUNT, ZERO
      COMP DELAY_COUNT, 000F  ;delay register compare value
      JUMPC LED_OUT

      LOAD LED_COUNT, ZERO
      LOAD DELAY_COUNT, ZERO
      RL LEDS                       ;rotate LEDs

LED_OUT
      OUTPUT LEDS, 0002       ;output LEDS
      JUMP MAIN

;==================================================
;           INTERRUPT SERVICE ROUTINE
;==================================================
        ADDRESS 0300
ISR
        INPUT   STATUS, 0001
        AND     STATUS, 0003

        COMP    STATUS, 0003
        JUMPZ   GOT_BOTH

        COMP    STATUS, 0002        ; check if TxRdy is high
        CALLZ   GOT_TXRDY

        COMP    STATUS, 0001        ; check if RxRdy is high
        CALLZ   GOT_RXRDY

        RETEN

GOT_BOTH
        CALL GOT_TXRDY
```

```
        CALL GOT_RXRDY
        RETEN

;=================================================
;           BIN_TO_ASCII SUBROUTINE
;=================================================
BINARY_TO_ASCII
        LOAD    RE, CHAR_COUNT


        LOAD    RD, 000A
        CALL    FIND_IT
        ADD     RB, 0030        ;CONVERT TO ASCII
        STORE   RB, 00C2        ;STORE TO MEMORY

        ADD     RE, 0030        ;CONVERT TO ASCII
        STORE   RE, 00C3        ;STORE TO MEMORY


        RETURN                  ;DONE

;=================================================
;                  FIND_IT SUBROUTINE
;=================================================
FIND_IT
        LOAD    RB, ZERO
NOT_DONE
        SUB     RE, RD
        JUMPC   RESTORE
        ADD     RB, ONE
        JUMP    NOT_DONE

RESTORE
        ADD     RE, RD
        RETURN

;=================================================
;                  BANNER_INIT SUBROUTINE
;=================================================
banner_init
```

```
        LOAD    CHAR_REG, ascii_DASH
x50     STORE   CHAR_REG,       POINTER
        ADD     POINTER,    ONE
        ADD     CHAR_COUNT, ONE
        COMP    CHAR_COUNT, 0031    ; 50 dashes
        JUMPC   x50
        LOAD    CHAR_COUNT, ZERO

        COMP    POINTER,    005F
        JUMPNC  finish_banner

        LOAD    CHAR_REG, ascii_CR
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_LF
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        ; 5 spaces
        LOAD    CHAR_REG, ascii_DOT
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE
```

```
        LOAD    CHAR_REG, ascii_A     ;Using my nickname AMAI
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_M
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_A
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_I
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE

        LOAD    CHAR_REG, ascii_CR
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_LF
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001
; END OF NAME
        ; 5 spaces
        LOAD    CHAR_REG, ascii_DOT
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001
```

```
LOAD    CHAR_REG, ascii_SP
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_C
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_E
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_C
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_S
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_SP
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_4
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_6
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_0
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_CR
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001
```

```
LOAD    CHAR_REG, ascii_LF
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_DOT
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_SP
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_SP
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_SP
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_SP
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_F
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_U
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_L
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001

LOAD    CHAR_REG, ascii_L
STORE   CHAR_REG, POINTER
ADD     POINTER, 0001
```

```
        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_U
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_A
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_R
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_T
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_CR
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_LF
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_DASH
        JUMP    x50
finish_banner


        LOAD    CHAR_REG, ascii_CR
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE
        LOAD    CHAR_REG, ascii_LF
        STORE   CHAR_REG, POINTER
        ADD     POINTER, ONE
```

```
        LOAD    CHAR_COUNT, 0000

        RETURN

;=================================================
;                  PROMPT_INIT SUBROUTINE
;=================================================
prompt_init

        LOAD    CHAR_REG, ascii_E
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_N
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_T
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_E
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_R
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_A
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
```

```
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_K
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_E
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_Y
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_COL
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        RETURN
;=================================================
;               HOMETOWN_INIT SUBROUTINE
;=================================================
hometown_init

        LOAD    CHAR_REG, ascii_M
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_Y
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_H
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001
```

```
        LOAD    CHAR_REG, ascii_O
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_M
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_E
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_T
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_O
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_W
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_N
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_I
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_S
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001
```

```
        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_I       ;hometown is industry
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_N
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_D
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_U
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_S
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_T
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_R
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

            LOAD    CHAR_REG, ascii_Y
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_CR
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001
```

```
        LOAD    CHAR_REG, ascii_LF
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        RETURN
;=================================================
;                  BACKSPACE_INIT SUBROUTINE
;=================================================
bs_init

        LOAD    CHAR_REG, ascii_BS
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_SP
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_BS
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        RETURN
;=================================================
;                  NEWLINE_INIT SUBROUTINE
;=================================================
crlf_init

        LOAD    CHAR_REG, ascii_CR
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_REG, ascii_LF
        STORE   CHAR_REG, POINTER
        ADD     POINTER, 0001

        RETURN

;=================================================
;                  TxRdy SUBROUTINE
```

```
; Retrieves data from memory based on current case
;===================================================
GOT_TXRDY
        COMP    CASE,    ZERO           ; case 0
        RETURNZ

        FETCH   CHAR_REG, POINTER       ; fetch data from pointer
        OUTPUT  CHAR_REG, ZERO          ; output the data
        ADD     POINTER,  ONE

        COMP    CASE,    0001           ; case 1
        JUMPZ   PRINT_BANNER            ; prints banner

        COMP    CASE,    0002           ; case 2
        JUMPZ   PRINT_PROMPT            ; prints prompt

        COMP    CASE,    0003           ; case 3
        JUMPZ   PRINT_HOMETOWN          ; prints hometown

        COMP    CASE,    0004           ; case 4
        JUMPZ   PRINT_BS                ; prints backspace

        COMP    CASE,    0005           ; case 5
        JUMPZ   PRINT_CRLF              ; prints new line

        COMP    CASE,    0006           ; case 6
        JUMPZ   PRINT_COUNT                     ; prints character count

        RETURN

PRINT_BS
        COMP    POINTER, BACKSPCE_END
        RETURNC
        LOAD    CASE, ZERO
        RETURN

PRINT_BANNER
        COMP        POINTER, BAN_END
        RETURNC
        LOAD        CASE, 0002
```

```
        RETURN

PRINT_PROMPT
        COMP        POINTER, PROMPT_END
        RETURNC
        LOAD        CASE, 0000
        RETURN

PRINT_HOMETOWN

        COMP        POINTER, HOMETW_END
        RETURNC
        LOAD        POINTER, PROMPT_START
        LOAD        CASE, 0002
        RETURN

PRINT_CRLF
        COMP        POINTER, CRLF_END
        RETURNC
        LOAD        POINTER, PROMPT_START
        LOAD        CASE, 0002
        RETURN

PRINT_COUNT
        COMP    POINTER, 00C4
        RETURNC
        LOAD    POINTER, CRLF_START
        LOAD    CASE, 0005
        RETURN

;=================================================
;                 RxRdy SUBROUTINE
;     Sets case and pointer based on user input
;=================================================
GOT_RXRDY
        COMP        CASE, ZERO      ; z if it is true
        RETURNNZ
            ; only enter this subroutine if the user is inputting chars
            ; based on those characters, different cases are set
```

```
        INPUT       DATA, ZERO
        COMP        DATA, ZERO
        RETURNZ

        COMP        DATA, ascii_aster
        JUMPZ       SET_HOMETOWN

        COMP        DATA, ascii_BS
        JUMPZ       SET_BS

        COMP        DATA, ascii_CR
        JUMPZ       SET_CRLF

        COMP        DATA, ascii_AT
        JUMPZ       SET_AT

        ; Only allowing forty characters
        ADD         CHAR_COUNT, ONE
        OUTPUT      DATA,       ZERO
        COMP        CHAR_COUNT, FORTY
        JUMPZ       SET_CRLF
        RETURN


SET_HOMETOWN
        LOAD        CASE, 0003
        LOAD        POINTER, HOMETW_START
        LOAD        CHAR_REG, ascii_null
        OUTPUT      CHAR_REG, ZERO
        LOAD        CHAR_COUNT, ZERO
        RETURN

SET_BS
        COMP        CHAR_COUNT, ZERO
        RETURNZ
        LOAD        CASE, 0004
        LOAD        POINTER, BACKSPCE_START
        LOAD        CHAR_REG, ascii_null
        OUTPUT      CHAR_REG, ZERO
        SUB         CHAR_COUNT, ONE
```

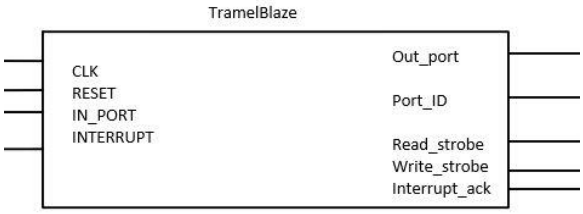
```
        RETURN

SET_CRLF

        LOAD        CASE, 0005
        LOAD        POINTER, CRLF_START
        LOAD        CHAR_REG, ascii_null
        OUTPUT      CHAR_REG, ZERO
        LOAD        CHAR_COUNT, ZERO
        RETURN

SET_AT
        CALL        BINARY_TO_ASCII
        LOAD        CASE, 0006
        LOAD        POINTER, 00C2
        LOAD        CHAR_REG, ascii_null
        OUTPUT      CHAR_REG, ZERO
        LOAD        CHAR_COUNT, ZERO
        RETURN

;================================================
; INTERRUPT SERVICE ROUTINE VECTORED THRU 0FFE
;================================================

        ADDRESS 0FFE
        JUMP    ISR
        END
```

5. Externally Developed Blocks

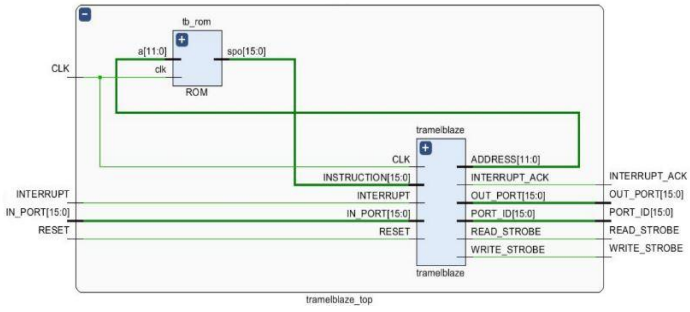
5.1 TramelBlaze

In the tx engine the purpose of the TramelBlaze is to control the signalling to the transmit engine. The signal txrdy will be used to trigger the TramelBlaze interrupt, and it will select one character and output to the transmit engine word input. The characters are outputted in ASCII before outputting to the transmit engine. 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.

Block diagram of Tramel Blaze



Detailed block diagram of Tramel Blaze

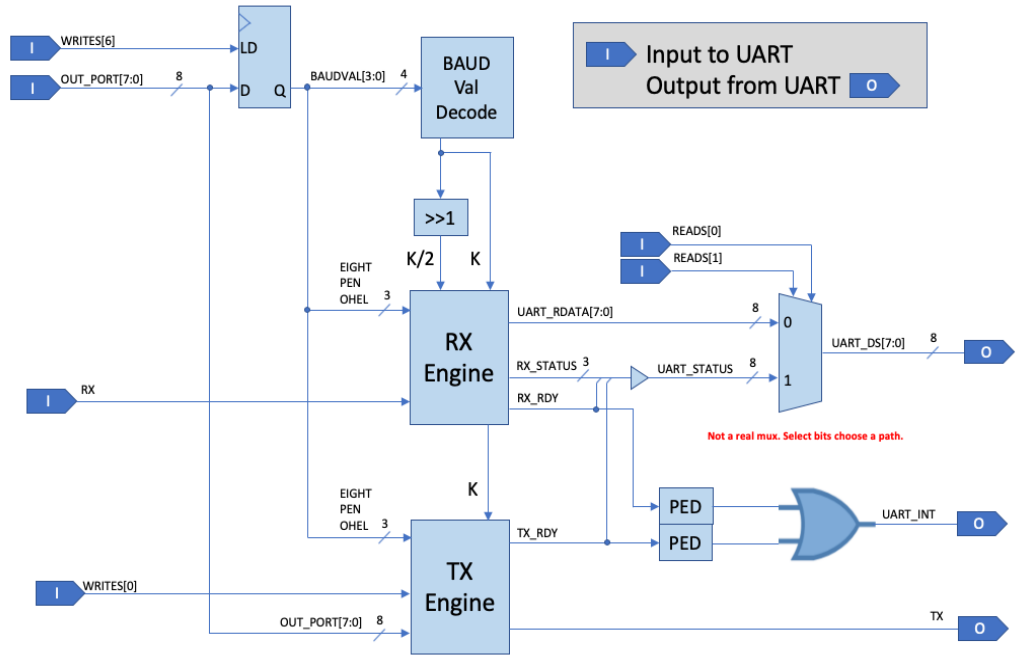


I/O chart of the Tramel blaze

Signal	Bits	I/O	Connected
CLK	1	Input	100 MHz Nexys 4 Board Clock
INTERRUPT	1	Input	UART Interrupt
IN_PORT	16	Input	UART Data Out
RESET	1	Input	AISO Synchronized Reset
INTERRUPT_ACK	1	Output	R Input of Interrupt SR Flop
OUT_PORT	16	Output	To UART Data In
PORT_ID	16	Output	Address Decoder Input
READ_STROBE	1	Output	Address Decoder Input
WRITE_STROBE	1	Output	Address Decoder Input

6 Internally Developed Blocks

UART Top Level Shown Below



6. Universal Asynchronous Receiver Transmitter (UART)

Our UART receiver transmitter is the product of the TX and RX engine working together in tandem. In conjunction they serve and act together as our serial communication protocol. To reiterate the Transmission and Receive engine are responsible for transmitting/receiving data, while that is happening our baud decoder module will be assigned a specific bit time to keep both the tx and rx engine synchronized. The UART design not only uses those parts for serial communication but also produces interrupts (via 2 PED) that communicate with the tramelblaze that will signal when each or either engine is ready.

Inputs

Writes[6] - Signal from the address decode to load the UART

Out_port[7:0] - 8 bit signal for the onboard switches

RX - Signal indicating that the RX engine can begin receiving serial data

Writes[0] - Signal from the address decode to the TX engine load

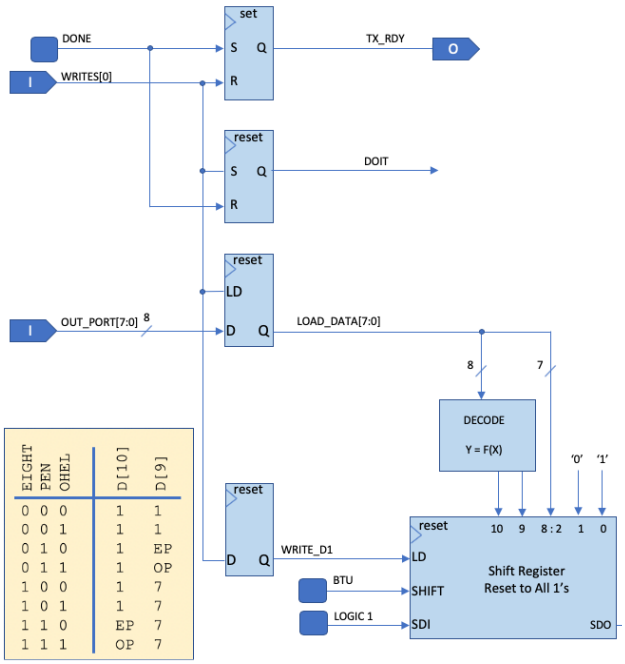
Reads[1:0] - Signal for the select of the RX engine

Outputs

UART_DS[7:0] - 8 bit signal going to the TramelBlaze

UART_INT - Signal that goes through the RS flop of the TramelBlaze

TX - Signal with transmission serial data going to the terminal program



Transmit Engine Block Diagram Shown Above

6.2 Transmit Engine

The Transmit Engine is fed data from the Tramel blaze, then transmits the data serially through the FPGA UART port where it should be received by the host machine. The Transmit engine's TxRdy signal is an interrupt that is implemented to send a ready flag whenever it is read to send a set of serial data. The transmit engines function by either sending 7 or 8 bits depending on the option you choose. Another function also has a polarity bit option, where you can choose to include a polarity bit as well as choose whether the polarity bit will be even or odd. Sequentially after the TxRdy signal has gone high, a load signal will determine when the data from the Tramelblaze is correctly loaded into the Transmit Engine's shift register. The decoded baud signal is then used to generate the correct timing through the two comparator setups. Utilizing this timing, the shift registers will move data to the right and the LSB (least significant bit) is used for our tx output.

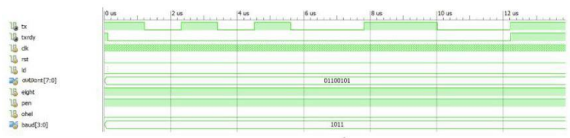
Inputs

clk – The clk is the 100MHz clock supplied by the FPGA
rst – Global reset used to initialize the design or restart operation
baud – A four-bit code used to determine comparator value which generates baud rate
eight – Determines if data sent is 7-bits (low) or 8-bits (high)
pen – Determines if polarity bit generation is enabled
ohel – Determines type of polarity generated (odd high/even low)

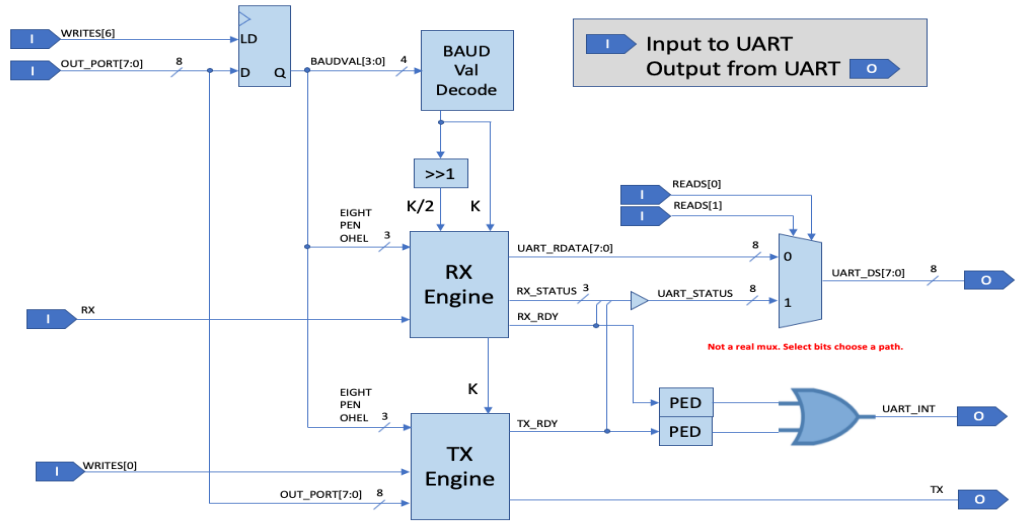
Outputs

tx – Single bit of transmit data
leds – Signal used to drive on board LEDs

Verification



Waveform verification showing that TX engine is able to shift the data in accordance to the parity bits + baud rate through the tx engine

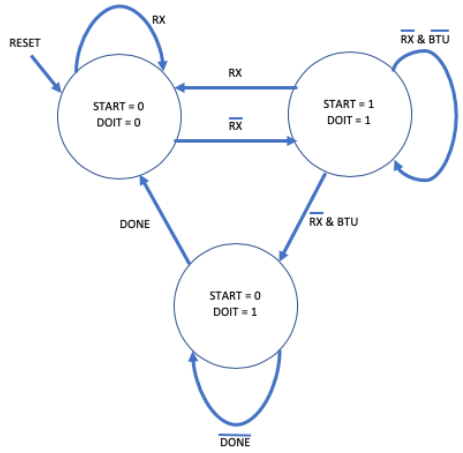


Receive Engine Block Diagram shown above

6.1.2 Recieve Engine

The receive engine utilizes a fsm (finite state machine) as well as a shift register in order to receive the necessary serial data. Our RX engine is designed to constantly,without interruption, look for a transition of 1 to 0 to identify and indicate our start bit. Then data is collected from that point with the first two values determining what happens next within our UART Protocol.

Receive Engine FSM Diagram shown below.



Inputs

Start - Indicates the receive engine to start recording the values being sent

Baud Value - Determines the time delay to take while receiving data

BTU - Counts the amount of bits being receive

Doit - Indicates when the receive engine will start

Eight - Determines the mode of the receive engine

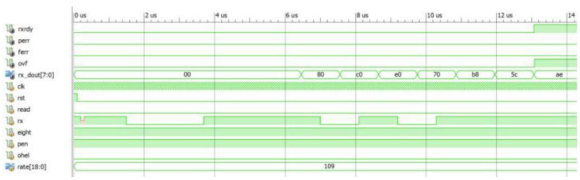
Pen - Determines the mode of the receive engine

Outputs

BTU - returns the amount of bits counted

Done - returns that the receive engine is done recording the data

Verification

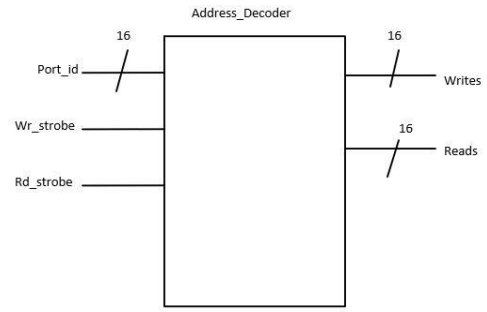


Waveform verification showing that RX engine is checking for transitions of 1 to 0

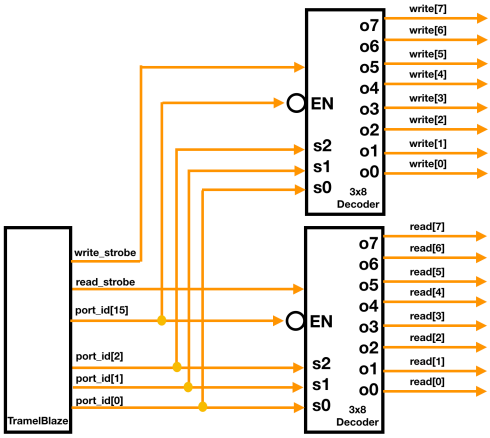
6.2 Address Decoder

The TramelBlaze needs to access data from a variety of sources within the SOC. Therefore an address decoder is required for the design to generate the necessary write and read enables. The address decoder is a combinational block that decodes the port ID from the TramelBlaze into two 16-bit read or write strobe outputs. (Note: 8 read and 8 write enables respectively). 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.

Address Decoder Combo-Block Diagram



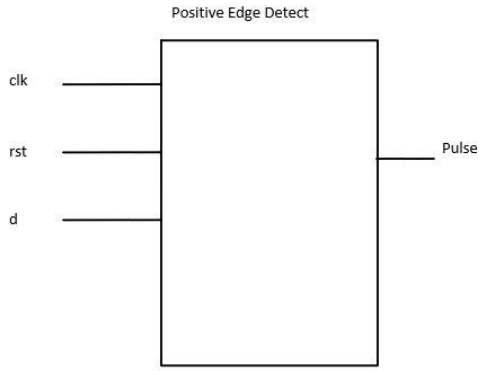
Address Decoder implementation example



6.3 Positive Edge Detect

The positive edge detect module implemented in our design detects whenever a signal goes low to high (rising edge) and will then output a one clock period signal (pulse) following. Its use cases are to indicate whenever there is a change in signal, again either high to low or low to high.

Positive Edge Detect Block Diagram



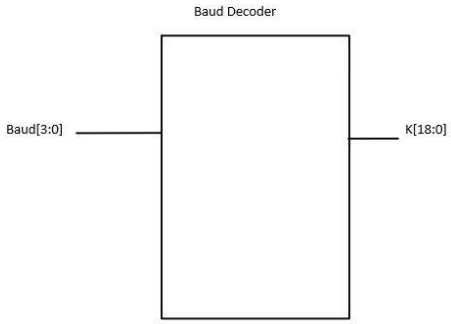
6.4 Baud Decoder

Our baud decoder will be utilizing a 4-bit input and will output a 19-bit wide signal. This signal will then generate the baud timing necessary for the transmit engine. The input for our baud decoder in this project will be our on-board switches and different switches will be a different desired baud value for our transmit engine to operate at. The design will be driven by the FPGAs 100 MHZ clock and the k values that have been generated will help drive our clock-divider circuit within our tx module.

Baud Rate Select Values Table **k bits = 19*

baud[3:0]	Rate	Eng not (bit time)	k
0000	300	3.3333 ms	333,333
0001	1200	833.33 us	83,333
0010	2400	416.66 us	41,667
0011	4800	208.33 us	20,8333
0100	9600	104.16 us	10,417
0101	19200	52.083 us	5,208
0110	38400	26.041 us	2,604
0111	57600	17.361 us	1,736
1000	115200	8.6806 us	868
1001	230400	4.3403 us	434
1010	460800	2.1701 us	217
1011	921600	1.0851 us	109

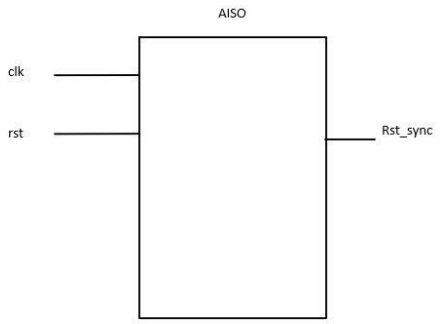
Baud Decoder Block diagram



6.5 Asynchronous In Synchronous Out Reset (AISO)

Our Asynchronous In Synchronous Out Reset, or to keep it sweet and short (AISO) module applies a synched reset across all modules. It is a great module that helps prevent unsynced data, clock domains crossing, and metastability.

AISO Block diagram



7 Chip Level Verification + Test

Onboard

To test that we have created a working UART that can transmit and receive data our program would display a banner (In this case I used my nickname Amai) and based on user input (e.g <cr> <bs> * @), 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 character in front of the cursor. A input of '*' will display my hometown: Industry followed by a new line and an input of '@' will display the number of characters that has been received so far.