

SYSTEM ON CHIP SPECIFICATION

16-BIT RISC MCU WITH UART

THOMAS NGUYEN  
CECS 460  
14 MAY 2019

DEPARTMENT OF COMPUTER ENGINEERING AND COMPUTER SCIENCE

This document contains information proprietary to the CSULB student that created the file – any reuse without adequate approval and documentation is prohibited

In submitting this file for class work at CSULB I am confirming that this is my work and the work of no one else.

In the event, other code sources are used, I will give document and give credit to the portion of code belonging to the author.

In submitting this project, I acknowledge that plagiarism in student project work is subject to dismissal from the class.

Copyright © 2019 Thomas Nguyen. All rights reserved.

INTENTIONALLY BLANK

Table Contents

[SECTION 1: INTRODUCTION 10](#_Toc8559833)

[1.1: PURPOSE 10](#_Toc8559834)

[SECTION 2: DOCUMENTS 11](#_Toc8559835)

[2.1: NEXYS 4 BOARD DATASHEET 11](#_Toc8559836)

[2.1.1: UART CONNECTIONS 11](#_Toc8559837)

[2.1.2: I/O Connections 12](#_Toc8559838)

[2.2: ARTIX-7 LIBRARY 13](#_Toc8559839)

[2.3: PICOBLAZE MICROPROCESSOR 13](#_Toc8559840)

[2.3.1: ARCHITECTURE 14](#_Toc8559841)

[2.3.2: INSTRUCTION SET 14](#_Toc8559842)

[SECTION 3: REQUIREMENTS 17](#_Toc8559843)

[3.1: INTERFACE REQUIREMENTS 17](#_Toc8559844)

[3.2: PHYSICAL REQUIREMENTS 18](#_Toc8559845)

[SECTION 4: TOP LEVEL IMPLEMENTATION 19](#_Toc8559846)

[4.1: DESCRIPTION 19](#_Toc8559847)

[4.2: BLOCK DIAGRAM 20](#_Toc8559848)

[4.3: DATA FLOW DESCRIPTION 20](#_Toc8559849)

[4.4: INPUT/OUTPUT 21](#_Toc8559850)

[4.4.1: SIGNAL NAMES 21](#_Toc8559851)

[4.4.2: PIN ASSIGNMENTS 21](#_Toc8559852)

[4.4.3: ELECTRICAL CHARACTERISTICS 21](#_Toc8559853)

[4.5: CLOCKS 21](#_Toc8559854)

[4.6: RESETS 22](#_Toc8559855)

[4.7: SOFTWARE 22](#_Toc8559856)

[4.7.1: DESCRIPTION 22](#_Toc8559857)

[4.7.2: SOFTWARE PLANNING DOCUMENT 22](#_Toc8559858)

[4.7.3: SOURCE CODE 26](#_Toc8559859)

[SECTION 5: EXTERNALLY ACQUIRED BLOCKS 27](#_Toc8559860)

[5.1: TRAMELBLAZE 27](#_Toc8559861)

[5.1.1: DESCRIPTION 27](#_Toc8559862)

[5.1.2: FUNCTIONAL REQUIREMENTS 27](#_Toc8559863)

[5.1.3: BLOCK DIAGRAM 27](#_Toc8559864)

[5.1.4: INPUT/OUTPUT 27](#_Toc8559865)

[SECTION 6: INTERNALLY DEVELOPED BLOCKS 28](#_Toc8559866)

[6.1: TSI 28](#_Toc8559867)

[6.1.1: DESCRIPTION 28](#_Toc8559868)

[6.1.2: BLOCK DIAGRAM 28](#_Toc8559869)

[6.1.3: INPUT/OUTPUT 28](#_Toc8559870)

[6.1.4: SOURCE CODE 28](#_Toc8559871)

[6.2: SOPC\_CORE 29](#_Toc8559872)

[6.2.1: DESCRIPTION 29](#_Toc8559873)

[6.2.2: BLOCK DIAGRAM 31](#_Toc8559874)

[6.2.3: INPUT/OUTPUT 32](#_Toc8559875)

[6.2.4: VERIFICATION 32](#_Toc8559876)

[6.2.5: SOURCE CODE 32](#_Toc8559877)

[6.3: RECEIVE ENGINE 33](#_Toc8559878)

[6.3.1: DESCRIPTION 33](#_Toc8559879)

[6.3.2: BLOCK DIAGRAM 34](#_Toc8559880)

[6.3.3: INPUT/OUTPUT 35](#_Toc8559881)

[6.3.4: VERIFICATION 35](#_Toc8559882)

[6.3.5: SOURCE CODE 35](#_Toc8559883)

[6.3.6: RECEIVE ENGINE CONTROL UNIT 36](#_Toc8559884)

[6.3.7: RECEIVE ENGINE DATAPATH 39](#_Toc8559885)

[6.4: TRANSMIT ENGINE 45](#_Toc8559886)

[6.4.1: DESCRIPTION 45](#_Toc8559887)

[6.4.2: BLOCK DIAGRAM 46](#_Toc8559888)

[6.4.3: INPUT/OUTPUT 47](#_Toc8559889)

[6.4.4: VERIFICATION 47](#_Toc8559890)

[6.4.5: SOURCE CODE 47](#_Toc8559891)

[6.4.6: SHIFT REGISTER 48](#_Toc8559892)

[6.5: EMBEDDED STATIC RAM 50](#_Toc8559893)

[6.5.1: DESCRIPTION 50](#_Toc8559894)

[6.5.2: BLOCK DIAGRAM 50](#_Toc8559895)

[6.5.3: INPUT/OUTPUT 51](#_Toc8559896)

[6.5.4: VERIFICATION 51](#_Toc8559897)

[6.6: ASYNCHRONOUS-IN SYNCHRONOUS-OUT 53](#_Toc8559898)

[6.6.1: DEFINITION 53](#_Toc8559899)

[6.6.2: BLOCK DIAGRAM 53](#_Toc8559900)

[6.6.3: INPUT/OUTPUT 53](#_Toc8559901)

[6.6.4: VERIFICATION 53](#_Toc8559902)

[6.6.5: SOURCE CODE 53](#_Toc8559903)

[6.7: LOAD REGISTER 54](#_Toc8559904)

[6.7.1: DEFINITION 54](#_Toc8559905)

[6.7.2: BLOCK DIAGRAM 54](#_Toc8559906)

[6.7.3: INPUT/OUTPUT 54](#_Toc8559907)

[6.7.4: VERIFICATION 54](#_Toc8559908)

[6.7.5: SOURCE CODE 54](#_Toc8559909)

[6.8: POSEDGE DETECT 55](#_Toc8559910)

[6.8.1: DEFINITION 55](#_Toc8559911)

[6.8.2: BLOCK DIAGRAM 55](#_Toc8559912)

[6.8.3: INPUT/OUTPUT 55](#_Toc8559913)

[6.8.4: VERIFICATION 55](#_Toc8559914)

[6.8.5: SOURCE CODE 55](#_Toc8559915)

[6.9: ADDRESS DECODER 56](#_Toc8559916)

[6.9.1: DEFINITION 56](#_Toc8559917)

[6.9.2: BLOCK DIAGRAM 56](#_Toc8559918)

[6.9.3: INPUT/OUTPUT 56](#_Toc8559919)

[6.9.4: VERIFICATION 56](#_Toc8559920)

[6.9.5: SOURCE CODE 56](#_Toc8559921)

[6.10: BAUD DECODER 57](#_Toc8559922)

[6.10.1: DEFINITION 57](#_Toc8559923)

[6.10.2: INPUT/OUTPUT 57](#_Toc8559924)

[6.10.3: VERIFICATION 57](#_Toc8559925)

[6.10.4: SOURCE CODE 57](#_Toc8559926)

[6.9: SR FLOP 58](#_Toc8559927)

[6.11.1: DEFINITION 58](#_Toc8559928)

[6.11.2: BLOCK DIAGRAM 58](#_Toc8559929)

[6.11.3: INPUT/OUTPUT 58](#_Toc8559930)

[6.11.4: VERIFICATION 58](#_Toc8559931)

[6.11.5: SOURCE CODE 58](#_Toc8559932)

[SECTION 7: CHIP VERIFICATION PLAN 59](#_Toc8559933)

[7.1: SOPC CORE VERIFICATION 59](#_Toc8559934)

[7.2: RECEIVE ENGINE VERIFICATION 61](#_Toc8559935)

[7.3: TRANSMIT ENGINE VERIFICATION 62](#_Toc8559936)

[SECTION 8: CHIP LEVEL TEST 64](#_Toc8559937)

[APPENDIX: 67](#_Toc8559938)

[A.1: TOP LEVEL 68](#_Toc8559939)

[A.2: TECHNOLOGY SPECIFIC INSTATIATIONS 69](#_Toc8559940)

[A.3: SOPC CORE 70](#_Toc8559941)

[A.4: ASYNCHRONOUS-IN SYNCHRONOUS-OUT REGISTER 72](#_Toc8559942)

[A.5: ADDRESS DECODER 73](#_Toc8559943)

[A.6: SET RESET FLOP 74](#_Toc8559944)

[A.7: POSITIVE-EDGE DETECT 75](#_Toc8559945)

[A.8: LOAD REGISTER 76](#_Toc8559946)

[A.9: BAUD DECODER 77](#_Toc8559947)

[A.10: TRANSMIT ENGINE 78](#_Toc8559948)

[A.11: RECEIVE ENGINE 80](#_Toc8559949)

[A.12: RECEIVE ENGINE CONTROL UNIT 81](#_Toc8559950)

[A.13: RECEIVE ENGINE DATAPATH 83](#_Toc8559951)

[A.14: ASSEMBLY PROGRAM 85](#_Toc8559952)

Table of Figures

[Figure 1: Nexys 4 UART Connection 11](#_Toc8559953)

[Figure 2: Nexys 4 I/O Connections 12](#_Toc8559954)

[Figure 3: IBUF/IBUFG 13](#_Toc8559955)

[Figure 4: OBUF 13](#_Toc8559956)

[Figure 5: TramelBlaze Architecture 14](#_Toc8559957)

[Figure 6: Top Level Block Diagram 19](#_Toc8559958)

[Figure 7: Top Level Detailed Diagram 20](#_Toc8559959)

[Figure 8: Flowchart for Initialization and Main Loop 22](#_Toc8559960)

[Figure 9: TramelBlaze Top Level Block Diagram 27](#_Toc8559961)

[Figure 10: TSI Block Diagram 28](#_Toc8559962)

[Figure 11: SOPC\_CORE Block Design 29](#_Toc8559963)

[Figure 12: SOPC\_CORE Detailed Block Design 31](#_Toc8559964)

[Figure 13: RX\_Engine Block 33](#_Toc8559965)

[Figure 14: Receive Engine Detailed Block Design 34](#_Toc8559966)

[Figure 15: RX\_control FSM 36](#_Toc8559967)

[Figure 16: RX\_control verification 1 37](#_Toc8559968)

[Figure 17: RX\_control verification 2 38](#_Toc8559969)

[Figure 18: RX\_control verification 3 38](#_Toc8559970)

[Figure 19: RX\_DATAPATH Block Diagram 39](#_Toc8559971)

[Figure 20: RX\_Datapath Detailed Block Design 40](#_Toc8559972)

[Figure 21: RX\_Datapath Verification 1 41](#_Toc8559973)

[Figure 22: RX\_Datapath Verification 2 42](#_Toc8559974)

[Figure 23: RX\_Datapath Verification 3 42](#_Toc8559975)

[Figure 24: RX Shift Register Detailed Block Diagram 43](#_Toc8559976)

[Figure 25: RX Shift Register Verification 43](#_Toc8559977)

[Figure 26: Remap Register Detailed Block Diagram 44](#_Toc8559978)

[Figure 27: Remap Verification 44](#_Toc8559979)

[Figure 28: TX\_ENGINE Block Diagram 45](#_Toc8559980)

[Figure 29: TX\_ENGINE Detailed Block Diagram 46](#_Toc8559981)

[Figure 30: Shift Register Block Diagram 48](#_Toc8559982)

[Figure 31: Shift Register Detailed Block Diagram 48](#_Toc8559983)

[Figure 32: SRAM Block Diagram 50](#_Toc8559984)

[Figure 33: SRAM Verification 1 51](#_Toc8559985)

[Figure 34: SRAM Verification 2 51](#_Toc8559986)

[Figure 35: SRAM Verification 3 52](#_Toc8559987)

[Figure 36: SRAM Verification 4 52](#_Toc8559988)

[Figure 37: SRAM Verification 5 52](#_Toc8559989)

[Figure 38: AISO Block Detailed Diagram 53](#_Toc8559990)

[Figure 39: AISO Verification 53](#_Toc8559991)

[Figure 40: Load Register Block Detailed Diagram 54](#_Toc8559992)

[Figure 41: Load Register Verification 54](#_Toc8559993)

[Figure 42: Posedge Detect Detailed Block Diagram 55](#_Toc8559994)

[Figure 43: Posedge Detect Verification 55](#_Toc8559995)

[Figure 44: Address Decoder Block Diagram 56](#_Toc8559996)

[Figure 45: Address Decoder Verification 56](#_Toc8559997)

[Figure 46: Baud Decoder Block Diagram 57](#_Toc8559998)

[Figure 47: Baud Decoder Verification 57](#_Toc8559999)

[Figure 48: SR Flop Block Diagram 58](#_Toc8560000)

[Figure 49: SR Flop Verification 58](#_Toc8560001)

[Figure 50: RX\_Engine Verification 61](#_Toc8560002)

[Figure 51: Transmit Engine Verification 63](#_Toc8560003)

[Figure 52: Program Verification 1 64](#_Toc8560004)

[Figure 53: Program Verification 2 64](#_Toc8560005)

[Figure 54: Memory Test 1 65](#_Toc8560006)

[Figure 55: Memory Test 2 65](#_Toc8560007)

[Figure 56: Memory Test 3 66](#_Toc8560008)

[Figure 57: Dump Memory Test 66](#_Toc8560009)

Table of Tables

[Table 1: TramelBlaze Instruction Set 14](#_Toc8560010)

[Table 2: Baud Rate 17](#_Toc8560011)

[Table 3: Parity Control 17](#_Toc8560012)

[Table 4: Physical Buttons 18](#_Toc8560013)

[Table 5: Physical Switches 18](#_Toc8560014)

[Table 6: Physical Light Emitting Diodes 18](#_Toc8560015)

[Table 7: Top\_Level Signal Names 21](#_Toc8560016)

[Table 8: Top\_Level Pin Assignments 21](#_Toc8560017)

[Table 9: TramelBlaze Signal Names 27](#_Toc8560018)

[Table 10: TSI Signal Names 28](#_Toc8560019)

[Table 11: SOPC\_CORE Signal Names 32](#_Toc8560020)

[Table 12: RX Engine Signal Names 35](#_Toc8560021)

[Table 13: RX\_control Signal Names 37](#_Toc8560022)

[Table 14: RX\_Datapath Signal Names 41](#_Toc8560023)

[Table 15: RX Shift Register Signal Names 43](#_Toc8560024)

[Table 16: Remap Signal Names 44](#_Toc8560025)

[Table 17: TX\_ENGINE Signal Names 47](#_Toc8560026)

[Table 18: Shift Register Signal Names 49](#_Toc8560027)

[Table 19: SRAM Signal Names 51](#_Toc8560028)

[Table 20: AISO Signal Names 53](#_Toc8560029)

[Table 21: Load Register Signal Names 54](#_Toc8560030)

[Table 22: Posedge Detect Signal Names 55](#_Toc8560031)

[Table 23: Address Decoder Signal Names 56](#_Toc8560032)

[Table 24: Baud Decoder Signal Names 57](#_Toc8560033)

[Table 25: SR Flop Signal Names 58](#_Toc8560034)

# SECTION 1: INTRODUCTION

The Chip Specification Manual details all the information there is about the Universal Asynchronous Receiver and Transmitter (UART) and the 16-bit embedded system TramelBlaze. The embedded system, TramelBlaze, utilizing the UART is an emulator of the 8-bit PicoBlaze to work on the Nexys 4 Artix-7 FPGA. These two components are abstract layers making the System on a Programmable Chip (SOPC). The SOPC communicates to a specific device utilizing the UART protocol through the Technology Specific Instantiation (TSI) or commonly known as I/O PAD to use the I/O drivers from standard Artix-7 libraries.

## 1.1: PURPOSE

The purpose of this manual is to provide information for every part of a chip in order to recreate, use, and understand the SOPC and TSI. Chip level block diagrams, detailed block diagrams, I/O signal mapping and module verification will be provided to further the understanding of these two components. Utilizing the microprocessor, an assembly program can be used to demonstrate how the SOPC uses the UART protocol to receive or transmit data through the microprocessor and an embedded SRAM to a device terminal.

# SECTION 2: DOCUMENTS

The section entails the documents and hardware used to assist in designing the chip. The Nexys 4, the Artix-7 libraries and the PicoBlaze microprocessor are key interfaces to the SOPC in order to use the assembly program and the UART protocol.

## 2.1: NEXYS 4 BOARD DATASHEET

The datasheet details the specifics to use the Nexys 4 FPGA board for the SOPC. It includes the schematic for the UART connections and the basic I/O connections.

### 2.1.1: UART CONNECTIONS

Figure : Nexys 4 UART Connection

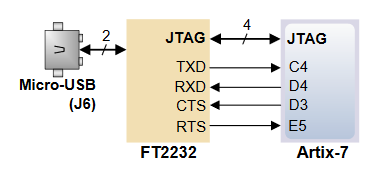


Figure 1 shows the necessary connections between the Nexys 4 Board and the Micro-USB to communicate using UART serial communication protocol.

### 2.1.2: I/O Connections

Figure : Nexys 4 I/O Connections

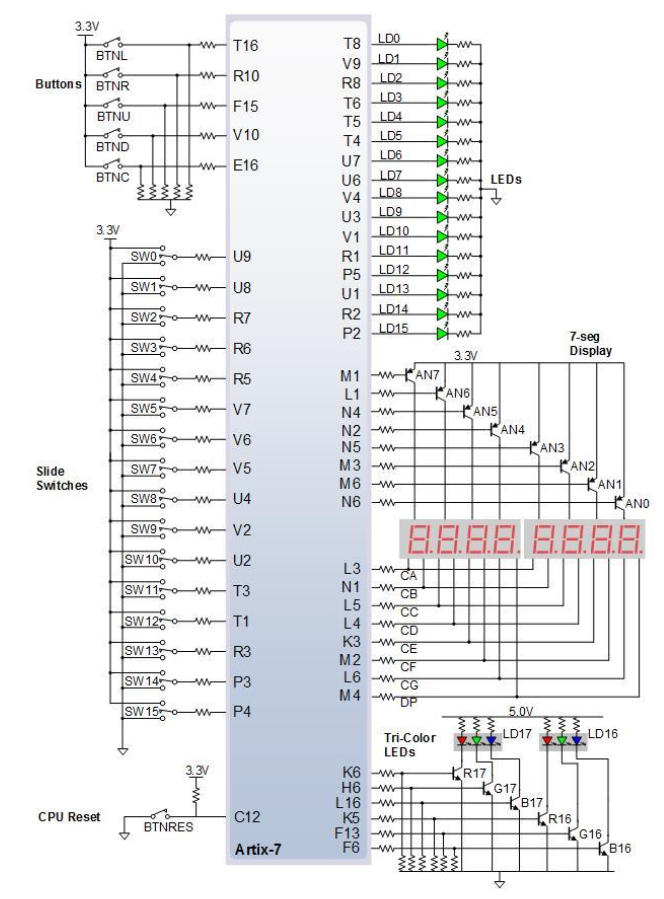


Figure 2 shows all the available I/O port connections possible on the Nexys 4 Artix-7 FPGA board. The necessary connections for the SOPC are the buttons, LEDs and slide switches.

## 2.2: ARTIX-7 LIBRARY

The standard library provides the simple essentials to implement the TSI for the SOPC input and output signals.

Figure : IBUF/IBUFG

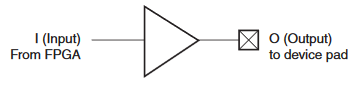


Figure 3 shows an input buffer used for all inputs. The IBUF is the universal input buffer while the IBUFG is for the clock input.

Figure : OBUF

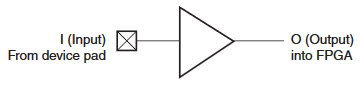


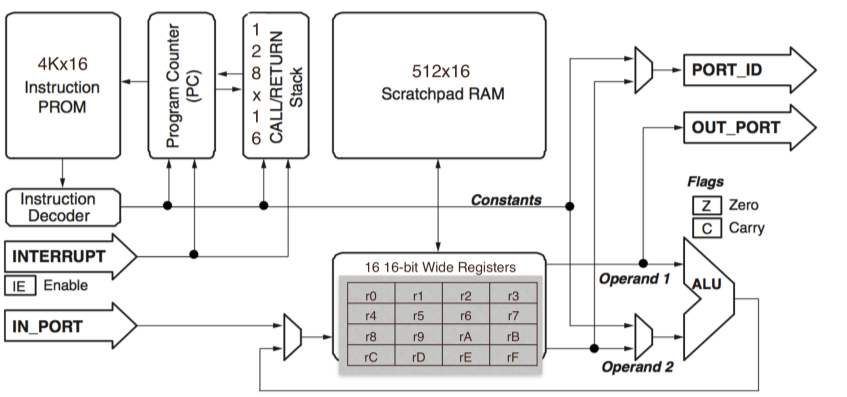
Figure 4 shows the universal output buffer for all output. It drives signals from the FPGA to external output pads.

## 2.3: PICOBLAZE MICROPROCESSOR

The PicoBlaze is an 8-bit RISC microprocessor from Xilinx for use in their FPGA and CPLD products. The PicoBlaze is compatible with some of the Xilinx FPGAs. It was made for the Spartan-3, Virtex-II and Virtex-II Pro families. The processor comes with license cores that are free to use for Xilinx products. In order for the PicoBlaze to be compatible with the Artix-7, John Tramel, the professor of CECS 460, developed an emulator to run the PicoBlaze as a 16-bit RISC microprocessor. The PicoBlaze utilizes assembly program instructions inside the instruction ROM to fetch, decode and execute register-based instructions. The PicoBlaze consists of a 1K x 18 PROM, 64 word scratchpad RAM, and 31 word call/return stack. The TramelBlaze ups the memories to 4K x 16 PROM, 512 x 16 scratchpad, 128 x 16 call/return stack. Both have 16 16-bit wide registers.

### 2.3.1: ARCHITECTURE

Figure : TramelBlaze Architecture



The PicoBlaze architecture is no different from the TramelBlaze with the exception of the different memory sizes.

### 2.3.2: INSTRUCTION SET

Table : TramelBlaze Instruction Set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 | ? | ? |

*Table 1: TramelBlaze Instruction Set (continue)*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction** | **Description** | **Function** | **ZERO** | **CARRY** |
| 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 | - | - |
| 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, 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 ← pp | - | - |
| 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 | - | - |
| 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] | ? | ? |

*Table 1: TramelBlaze Instruction Set (continue)*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction** | **Description** | **Function** | **ZERO** | **CARRY** |
| 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] | 1 | ? |
| 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 ← {0, sX[7:1]}  CARRY ← sX[0] | 1 | ? |
| SRA sX | Shift register sX right through all bits, including CARRY | sX ← {CARRY, sX[7:1]}  CARRY ← sX[0] | ? | ? |
| SRX sX | Shift register sX right. Bit sX[0] 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 | ? | ? |
| 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 sY. Update CARRY and ZERO flags. Registers are unaffected | If(sX AND sY) = 0, ZERO ← 1  CARRY ← odd parity of (sX AND sY) | ? | ? |
| 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 |

|  |  |
| --- | --- |
| **Operand** | **Description** |
| sX | One of 16 possible register locations ranging from s0 through sF or specified as a literal |
| sY | One of 16 possible register locations ranging from s0 through sF or specified as a literal |
| aaa | 12-bit address, specified either as a literal or a four-digit hexadecimal value ranging from 0000 to FFFF or a labeled location. |
| kk | 16-bit immediate constant, specified as a literal or a two-digit hexadecimal value ranging from 00 to FF or specified as a literal |
| pp | 16-bit port address, specified as a literal or a two-digit hexadecimal value ranging from 00 to FF or specified as a literal |
| ss | 9-bit scratchpad RAM address, specified either as a literal or a three-digit hexadecimal value ranging from 000 to 1FF or specified as a literal |
| RAM[n] | Contents of scratchpad RAM at location n |
| TOS | Value stored at top of stack |

# SECTION 3: REQUIREMENTS

## 3.1: INTERFACE REQUIREMENTS

The input and output of the SOPC core comes from and to the Nexys 4 Artix-7 FPGA board through a Technology Specific Instantiation (TSI). To communicate with each other, the design has eleven baud rates to choose from using four onboard switches of the FPGA. Parity is controlled by another three switches. The parity control contains a switch for eight-bit, parity enable and odd/even parity select. These are based on and compatible with industrial standards.

Table : Baud Rate

|  |  |  |
| --- | --- | --- |
| **Switches [7:4]** | **Bit Time** | **Baud Rate** |
| 0000 | 3.3333 ms | 300 |
| 0001 | 833.33 us | 1200 |
| 0010 | 416.66 us | 2400 |
| 0011 | 208.33 us | 4800 |
| 0100 | 104.16 us | 9600 |
| 0101 | 52.083 us | 19200 |
| 0110 | 26.041 us | 38400 |
| 0111 | 17.361 us | 57600 |
| 1000 | 8.6806 us | 115200 |
| 1001 | 4.3403 us | 230400 |
| 1010 | 2.1701 us | 460800 |
| 1011 | 1.0851 us | 921600 |

Table : Parity Control

|  |  |  |
| --- | --- | --- |
| Switches [3:1] | Bits | Parity |
| 000 | 7 | None |
| 001 | 7 | None |
| 010 | 7 | Even |
| 011 | 7 | Odd |
| 100 | 8 | None |
| 101 | 8 | None |
| 110 | 8 | Even |
| 111 | 8 | Odd |

## 3.2: PHYSICAL REQUIREMENTS

The SOPC utilizes seven switches out of sixteen from the FPGA board to control the baud rate and parity and one button out of five for reset. The switches are in a specific order for a register within the SOPC. The switches [7:1] are used. Switches [7:4] is for baud rate control. Switches [3:1] is for parity control. The LED[15:0] are used to show that the board is on and actually running outside of the UART terminal.

Table : Physical Buttons

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BTNU** | **BTND** | **BTNC** | **BTNL** | **BTNR** |
| SYS\_RESET | N/A | N/A | N/A | N/A |

Table : Physical Switches

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SW7** | **SW6** | **SW5** | **SW4** | **SW3** | **SW2** | **SW1** | **SW0** |
| i\_SW[7] | i\_SW[6] | i\_SW[5] | i\_SW[4] | i\_SW[3] | i\_SW[2] | i\_SW[1] | N/A |

Table : Physical Light Emitting Diodes

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **LED15** | **LED14** | **LED13** | **LED12** | **LED11** | **LED10** | **LED9** | **LED8** |
| o\_LED[15] | o\_LED[14] | o\_LED[13] | o\_LED[12] | o\_LED[11] | o\_LED[10] | o\_LED[9] | o\_LED[8] |
| **LED7** | **LED6** | **LED5** | **LED4** | **LED3** | **LED2** | **LED1** | **LED0** |
| o\_LED[7] | o\_LED[6] | o\_LED[5] | o\_LED[4] | o\_LED[3] | o\_LED[2] | o\_LED[1] | o\_LED[0] |

# SECTION 4: TOP LEVEL IMPLEMENTATION

## 4.1: DESCRIPTION

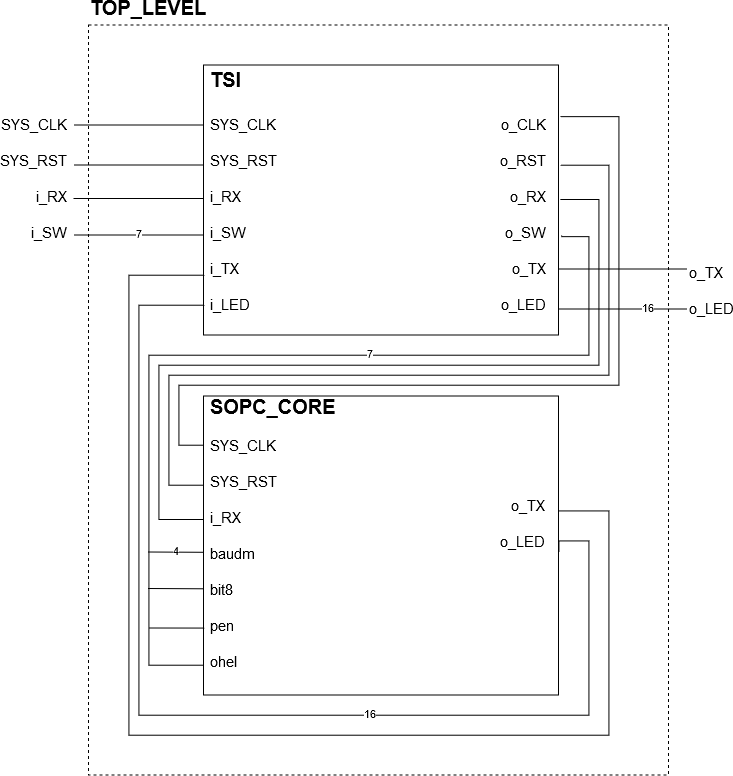
The top level of the design is to implements the connection between the SOPC\_CORE and the TSI. The SOPC\_CORE contains the microprocessor and UART engine program. The program inside the microprocessor communicates with the SOPC\_CORE through UART. The TSI contains references to the Artix 7 FPGA libraries and buffers all incoming input and output data from the hardware and software. I/O, including clock and reset, go through a buffer that is the TSI before heading out or into the SOPC\_CORE. Input go through an IBUF or IBUFG and output go through an OBUF. Output signals are driven from the FPGA to external devices.

Figure : Top Level Block Diagram



## 4.2: BLOCK DIAGRAM

Figure : Top Level Detailed Diagram



## 4.3: DATA FLOW DESCRIPTION

The baudm is an input that determines the baud rate or transfer rate of data to and from the hardware. LEDs receive data to light up.

## 4.4: INPUT/OUTPUT

### 4.4.1: SIGNAL NAMES

Table : Top\_Level Signal Names

|  |  |  |
| --- | --- | --- |
| **Signal** | **From** | **To** |
| SYS\_CLK | 100MHz Crystal Oscillator | SOPC\_CORE |
| SYS\_RESET | BTNU | SOPC\_CORE |
| i\_SW | FPGA Switches | SOPC\_CORE |
| i\_RX | FPGA RX | SOPC\_CORE |
| o\_TX | SOPC\_CORE | FPGA TX |
| [15:0] o\_LED | SOPC\_CORE | FPGA LEDs |

### 4.4.2: PIN ASSIGNMENTS

Table : Top\_Level Pin Assignments

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Pin Location** | **Signal Name** | **Pin Location** |
| SYS\_CLK | E3 | o\_LED[3] | N14 |
| SYS\_RST | M18 | o\_LED[4] | R18 |
| i\_SW[1] | J15 | o\_LED[5] | V17 |
| i\_SW[2] | L16 | o\_LED[6] | U17 |
| i\_SW[3] | M13 | o\_LED[7] | U16 |
| i\_SW[4] | R15 | o\_LED[8] | V16 |
| i\_SW[5] | R17 | o\_LED[9] | T15 |
| i\_SW[6] | T18 | o\_LED[10] | U14 |
| i\_SW[7] | U18 | o\_LED[11] | T16 |
| o\_TX | D4 | o\_LED[12] | V15 |
| i\_RX | C4 | o\_LED[13] | V14 |
| o\_LED[0] | H17 | o\_LED[14] | V12 |
| o\_LED[1] | K15 | o\_LED[15] | V11 |
| o\_LED[2] | J13 |  |  |

### 4.4.3: ELECTRICAL CHARACTERISTICS

* Buttons
  + 3.3V as logical 1
  + 0V as logical 0
* Switches
  + 1.8V as logical 1
  + 0V as logical 0

## 4.5: CLOCKS

The Nexys 4 Artix-7 FPGA board has a single 100MHz crystal oscillator clock.

## 4.6: RESETS

Every register and flop within the SOPC utilize a single reset that is generated by a synchronized signal through the use of an asynchronous in synchronous out (AISO) hardware model. This is to prevent any possibility of metastability happening during reset due to unsynced signals.

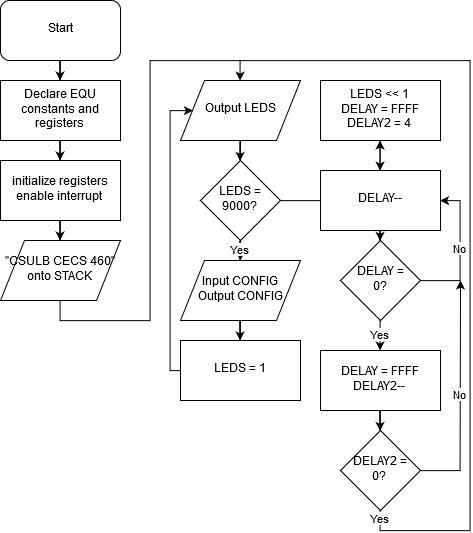
## 4.7: SOFTWARE

### 4.7.1: DESCRIPTION

The assembly program that the microprocessor executes is programmed into the PROM. The program is an interface to allow the UART protocol to communicate with other devices outside of the SOPC. The program first does a memory test on bootup and is followed by a banner. After the banner a prompt is initiated to allow the user to input characters from the keyboard. Any key press will echo back what the user input. There are specific characters with added functions when received by the UART. The ‘@’ key displays the number of characters received since reset, the ‘\*” key displays the author’s hometown and the ‘%’ key dumps the memory again. The memory takes in all received data and stores them inside the embedded RAM to be accessed and dumped. Only on hardware reset will cause the embedded RAM to be emptied.

### 4.7.2: SOFTWARE PLANNING DOCUMENT

Figure : Flowchart for Initialization and Main Loop



Pseudo Code for ISR:  
Start @ Address 0500  
read status flag  
obtain TXRDY flag and check if high  
if TXRDY flag is high  
 call TXRDY function  
obtain RXRDY flag and check if high  
if RXRDY flag is high  
 call RXRDY function

Pseudo Code for TXRDY Function:  
if DISPLAY register is 8  
 jump to MEM\_TEST0 function  
else if 9  
 jump to MEM\_TEST1 function  
else if A  
 jump to MEM\_TEST2 function  
else if B  
 jump to MEM\_TEST3 function  
else if C  
 jump to MEM\_TEST4 function  
else if D  
 jump to MEM\_TEST5 function  
else if DISPLAY register is 1  
 jump to BANNER\_O function  
else if DISPLAY register is 2  
 jump to PROMPT\_O function  
else if DISPLAY register is 3  
 jump to HOMETOWN\_O function  
else if DISPLAY register is 4  
 jump to CHAR\_COUNT\_O function  
else if DISPLAY register is 5  
 jump to BACKSPACE\_O function  
else if DISPLAY register is 6  
 jump to CAR\_RET\_O function  
else if DISPLAY register is 7  
 jump to MEM\_DUMP function  
fetch from memory pointed by stack pointer  
output memory contents @location  
increment stack pointer

Pseudo Code for RXRDY Function:  
if STATUS is equal to (Status & 1C)   
 call ERROR function  
check RX input  
if RX is null  
 return to ISR  
if RX is the same as RX old  
 jump to RXFB  
if RX is hex 2A (asterisk)  
 set DISPLAY to 3  
 set stack pointer to 9  
else if RX is hex 40 (@)  
 convert character count binary to ascii and store into memory  
 set DISPLAY to 4  
 set stack pointer to hex F0  
else if RX is hex 7F (backspace)  
 if character counter is 0  
 return to ISR  
 set DISPLAY to 5  
 set stack pointer to hex 16  
 subtract from character counter  
 subtract from SRAM pointer  
else if RX is hex D (carriage return)  
 set DISPLAY to 6  
 set stack pointer to hex 14  
else if RX is hex 25 (percent)  
 set DISPLAY to 7  
 set stack pointer to hex 8000  
else  
 increment character counter  
 output RX to SRAM  
 increment SRAM pointer  
 output RX to terminal  
 set RX old to RX  
 jump to CAR\_RET\_I  
return to ISR

Pseudo Code for Memory Test:  
initialize stack pointer  
initialize DISPLAY register to 8  
load temporary register with AAAA  
output to memory address pointed by stack pointer  
increment stack pointer  
output content at location in memory  
reset stack pointer to initialized value  
repeat with 5555  
initialize stack pointer  
load address location into temporary register  
output address to memory address pointed by stack pointer  
increment stack pointer  
output content at location in memory  
reset stack pointer to 0  
load DISPLAY register to 0  
return to ISR

Pseudo Code for BANNER\_O Function:  
for stack pointer < 11  
 output from memory location pointed by stack pointer  
set stack pointer to 0  
store string characters of prompt onto stack starting at stack point 0  
store string characters of hometown onto stack  
store string characters of carriage return and line feed onto stack  
store string characters of backspace onto stack  
set DISPLAY reg to 0  
return to ISR

Pseudo Code for PROMPT\_O Function:  
for stack pointer < A  
 output from memory location pointed by stack pointer  
set stack pointer to 0  
set DISPLAY reg to 0  
return to ISR

Pseudo Code for HOMETOWN\_O Function:  
for stack pointer < 16  
 output from memory location pointed by stack pointer  
set stack pointer to 0  
set DISPLAY reg to 0  
return to ISR

Pseudo Code for CHAR\_COUNT\_O Function:  
for stack pointer < 57  
 output from memory location pointed by stack pointer  
set stack pointer to 0  
set DISPLAY reg to 6  
return to ISR

Pseudo Code for BACKSPACE\_O Function:  
for stack pointer < 19  
 output from memory location pointed by stack pointer  
set stack pointer to 0  
set temporary register = RX  
set DISPLAY reg to 0  
return to ISR

Pseudo Code for CAR\_RET\_O Function:  
for stack pointer < 16  
 output from memory location pointed by stack pointer  
set stack pointer to 0  
set DISPLAY reg to 2  
return to ISR

Pseudo Code for DUMP\_MEM\_O Function:  
for stack pointer < SRAM pointer value  
 input from SRAM memory  
 output from memory location pointed by stack pointer   
set stack pointer to 0  
set DISPLAY reg to 0  
return to ISR

### 4.7.3: SOURCE CODE

See [A.14: ASSEMBLY PROGRAM](#_Toc8414840)

# SECTION 5: EXTERNALLY ACQUIRED BLOCKS

## 5.1: TRAMELBLAZE

### 5.1.1: DESCRIPTION

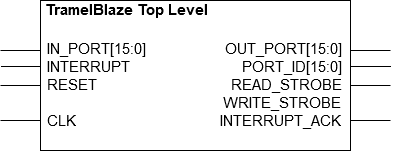
The TramelBlaze is a 16-bit emulator of the 8-bit PicoBlaze in order to use the PicoBlaze ISA in newer versions of the Nexys FPGA boards. It provides all the functions of the original PicoBlaze.

### 5.1.2: FUNCTIONAL REQUIREMENTS

The TramelBlaze requires the creation of the 128x16 Stack RAM and 512x16 Scratch RAM and then the use of an assembler to convert assembly code into a .coe file for the 4096x16 Instruction ROM to tell what the rest of the hardware to do. There is also a 16x16 array register.

### 5.1.3: BLOCK DIAGRAM

Figure : TramelBlaze Top Level Block Diagram



### 5.1.4: INPUT/OUTPUT

Table : TramelBlaze Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| CLK | [15:0] OUT\_PORT |
| RESET | [15:0] PORT\_ID |
| [15:0] IN\_PORT | READ\_STROBE |
| INTERRUPT | WRITE\_STROBE |
|  | INTERRUPT\_ACK |

# SECTION 6: INTERNALLY DEVELOPED BLOCKS

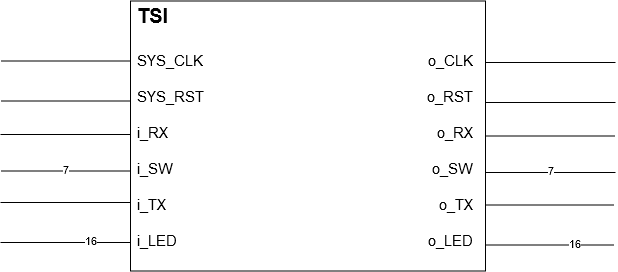
## 6.1: TSI

### 6.1.1: DESCRIPTION

The technology specific instantiation (TSI) is an I/O PAD. The I/O PADs interface with the pins of the device. They use the library of the device and serve as an input and output buffer of the device to external devices.

### 6.1.2: BLOCK DIAGRAM

Figure : TSI Block Diagram



### 6.1.3: INPUT/OUTPUT

Table : TSI Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| SYS\_CLK | o\_CLK |
| SYS\_RESET | o\_RESET |
| i\_RX | o\_RX |
| [7:0] i\_SW | [7:0] o\_SW |
| i\_TX | o\_TX |
| [15:0] i\_LED | [15:0] o\_LED |

### 6.1.4: SOURCE CODE

See [A.2: TECHNOLOGY SPECIFIC INSTATIATIONS](#_Toc8444575)

## 6.2: SOPC\_CORE

### 6.2.1: DESCRIPTION

The SOPC\_CORE exists to implement a processor that receives and transfers bits of data. The current design implements the full Universal Asynchronous Receiver Transmitter (UART) and works in unison with the TramelBlaze to receive and transmit data via terminal. The switches of the UART controls the baud rate at which data is being transferred and received and the parity control.

The SOPC\_CORE is made up of the UART RX and TX Engine, TramelBlaze, Baud Decoder, Address Decoder, Asynchronous-In-Synchronous-Out Register, Positive Edge Detect, LEDs, SR flop and load register.

Figure : SOPC\_CORE Block Design

SOPC\_CORE

4

16

uart\_rxd\_out

baudm

bit8

leds

pen

ohel

clk100mhz

reset

uart\_txd\_in

rx\_engine – This is the UART Receive Engine. It receives data bits input by the user. It generates interrupts to the microprocessor and the amount and speed of data bits it receives are changed based on the parity control and baud switches.

tx\_engine – This is the UART Transmit Engine. It transmits data bits, generates interrupts to the microprocessor and the amount and speed of data bits transmitted are changed based on the parity control and baud switches.

TramelBlaze – A 16-bit microprocessor that is emulating an 8-bit microprocessor PicoBlaze. The microprocessor is externally developed by John Tramel to run and execute program-based assembly instructions from the 4K x 16 PROM.

baud\_dec – This is the baud rate decoder. It has eleven types of baud rate to select from based on switch [7:4] of the FPGA starting from the slowest to fastest.

AISO\_register – The Asynchronous-In-Synchronous-Out register allows an input signal to become synchronous across the entire hardware. This register is used to bring reset to a synchronous known state in order to prevent metastability.

posedge\_detect – This allows an active signal to be caught for 1 clock cycle and no more than that.

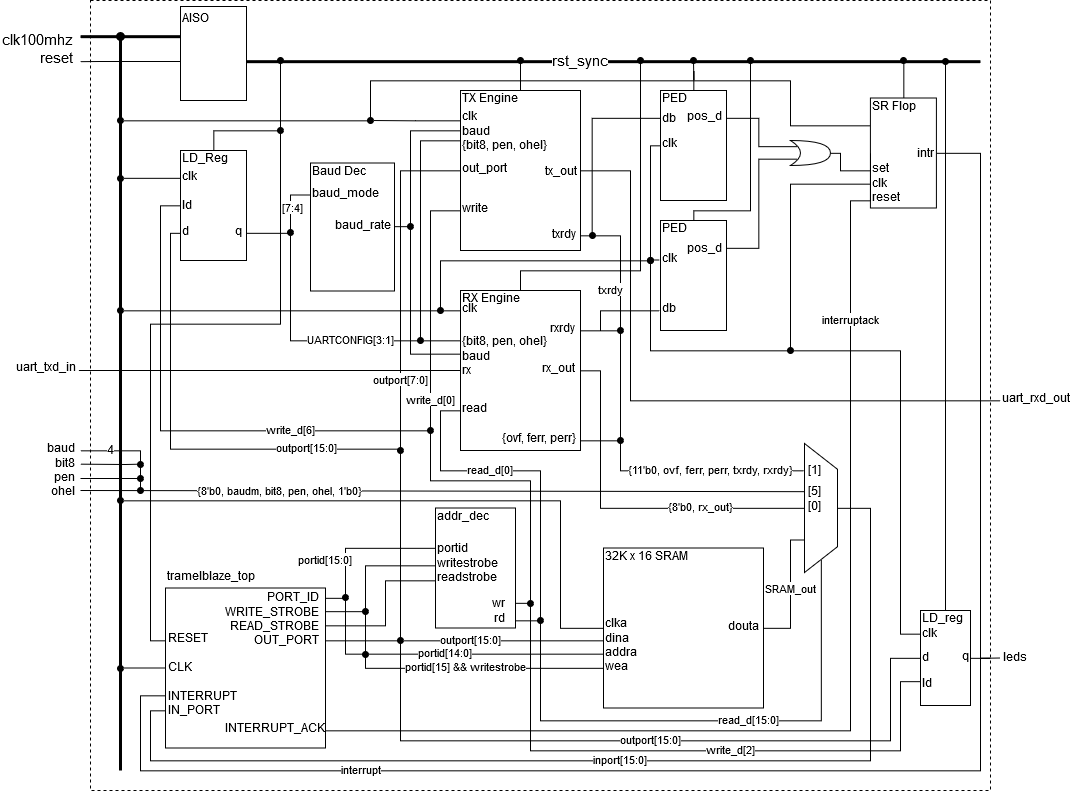
LEDs – The LED register is assigned the output of the contents of specific register of the program depending on a specific write address.

SR\_flop – The set and reset flip flop waits for the pulse signal from the posedge\_detect to acknowledge the interrupt request and turn off the interrupt.

LD\_reg – This register receives a load signal from a specific write address of the program from TramelBlaze and assigns the UARTCONFIG signal the state of the switches from the FPGA.

### 6.2.2: BLOCK DIAGRAM

Figure : SOPC\_CORE Detailed Block Design



### 6.2.3: INPUT/OUTPUT

Table : SOPC\_CORE Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk100mhz | uart\_rxd\_out |
| reset | [15:0] leds |
| bit8 |  |
| pen |  |
| ohel |  |
| [3:0] baudm |  |
| uart\_txd\_in |  |

### 6.2.4: VERIFICATION

See section [7.1: SOPC CORE VERIFICATION](#_Toc8425619)

### 6.2.5: SOURCE CODE

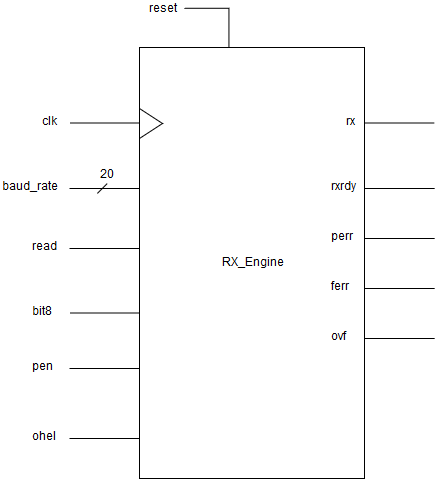
See [A.3: SOPC CORE](#_Toc8444576)

## 6.3: RECEIVE ENGINE

### 6.3.1: DESCRIPTION

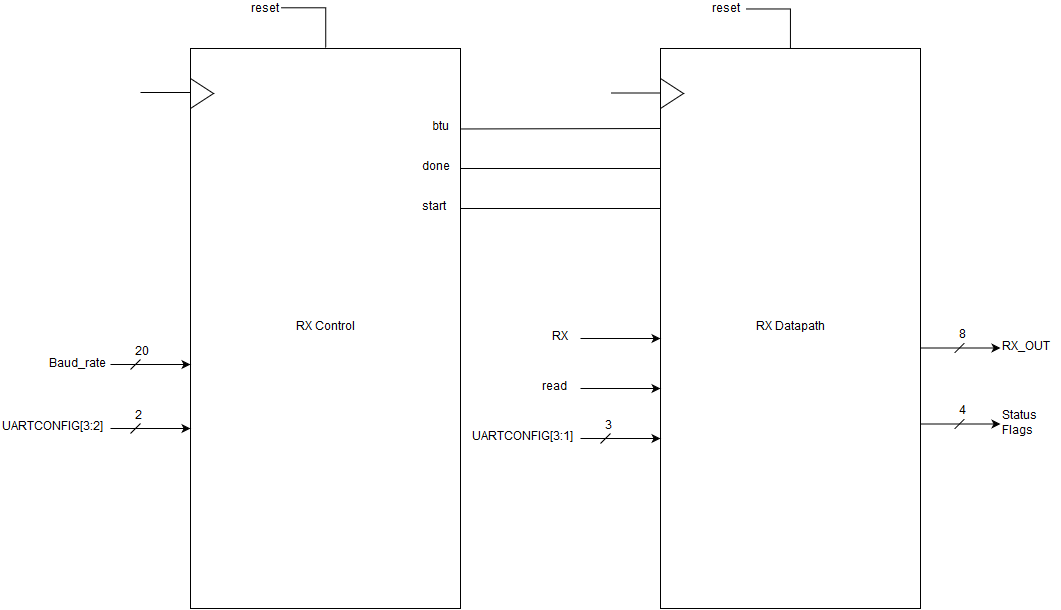
The purpose of this module is to provide half of the Universal Asynchronous Receiver Transmitter (UART). The focus will be the receive. The receive engine consists of two module components that make it up. It has the RX Control Unit and RX Datapath. The RX Control comprises the counter values to make sure the speed at which bits are received are the same as the baud rate. The RX Datapath uses the counter to make sure the RX data in are received at the right time and checks for when it is ready to receive the next set of data, parity error, framing error and overflow error.

Figure : RX\_Engine Block



### 6.3.2: BLOCK DIAGRAM

Figure : Receive Engine Detailed Block Design



### 6.3.3: INPUT/OUTPUT

Table : RX Engine Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | rx |
| reset | rxrdy |
| baud\_rate [19:0] | perr |
| read | ferr |
| bit8 | ovf |
| pen |  |
| ohel |  |

### 6.3.4: VERIFICATION

See section [7.2: RECEIVE ENGINE VERIFICATION](#_Toc8425620)

### 6.3.5: SOURCE CODE

See [A.11: RECEIVE ENGINE](#_Toc8444584)

### 6.3.6: RECEIVE ENGINE CONTROL UNIT

#### 6.3.6.1: Definition

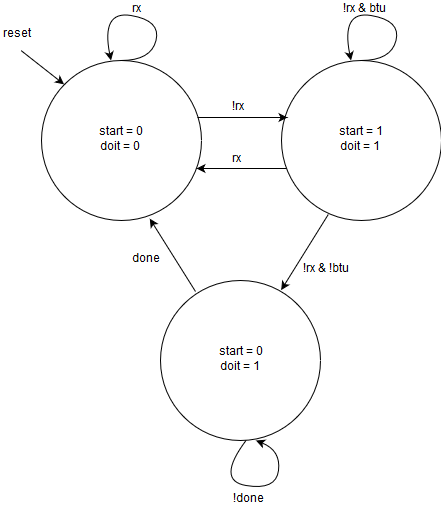
The receive engine control unit generates the control signals to the receive engine datapath. The control unit has a bit time counter that counts to the full or half bit time value of the baud rate according to which state it is in the Finite State Machine (FSM). In state 1 or the start state, the time is half because the state machine is checking to see if the start bit remains for half a bit time.

There is a bit counter, which counts the frame of the data selected by eight bit and parity enable bit.

The FSM consists of 3 states. For the first state, it is consistently checking for a transition from high to low of the RX before going to state 1 in order to find when to get the start bit. Start then remains high for half a bit time before moving to the last state. It remains in state 2 until done is set and starts over from state 0.

#### 6.3.6.2: Finite State Machine

Figure : RX\_control FSM



#### 6.3.6.3: Input/Output

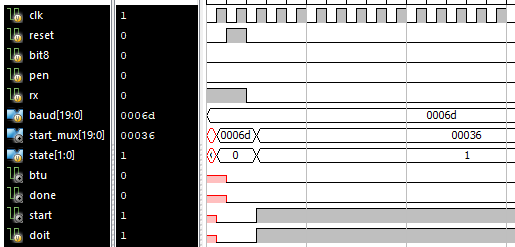
Table : RX\_control Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | btu |
| reset | done |
| Rx | start |
| [19:0] baud |  |
| bit8 |  |
| pen |  |

#### 6.3.6.4: Verification

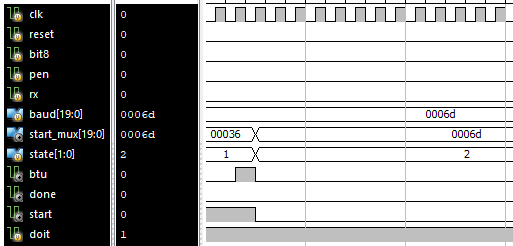
The module starts in state 0 before going into state 1 as soon as rx is low. Start and doit are set as soon as they reach state 1.

Figure : RX\_control verification 1



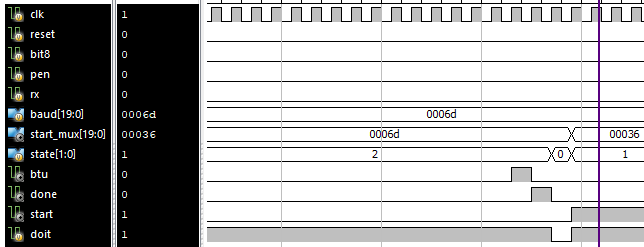
It remains in state 1 until btu is high and rx remaining low.

Figure : RX\_control verification 2



To go back to state 0, the signal done must be set. Doit remains high and start has already turned off at this point. As soon as done is set, the next state becomes 0 and the cycle is repeated.

Figure : RX\_control verification 3



#### 6.3.6.5: Source Code

See [A.12: RECEIVE ENGINE CONTROL UNIT](#_Toc8444585)

### 6.3.7: RECEIVE ENGINE DATAPATH

#### 6.3.7.1: Description

The receive engine datapath receives the frame of bits from the UART RX to the shift register within and continues until it receives the stop bit. The bits inside the shift register are then remapped to 10 bits. 8 bits are transferred as data output. The UART status flags are also set in this module.

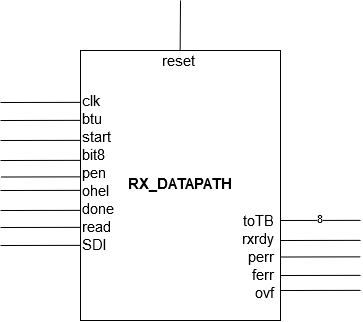
The RXRDY flag waits for when done is set before going high.

The PERR, or parity-bit error, flag is set when the data frame is mismatched with the parity bit set by the parity control.

The FERR, or framing error, flag is set when the stop bit is mismatched with the settings of parity control.

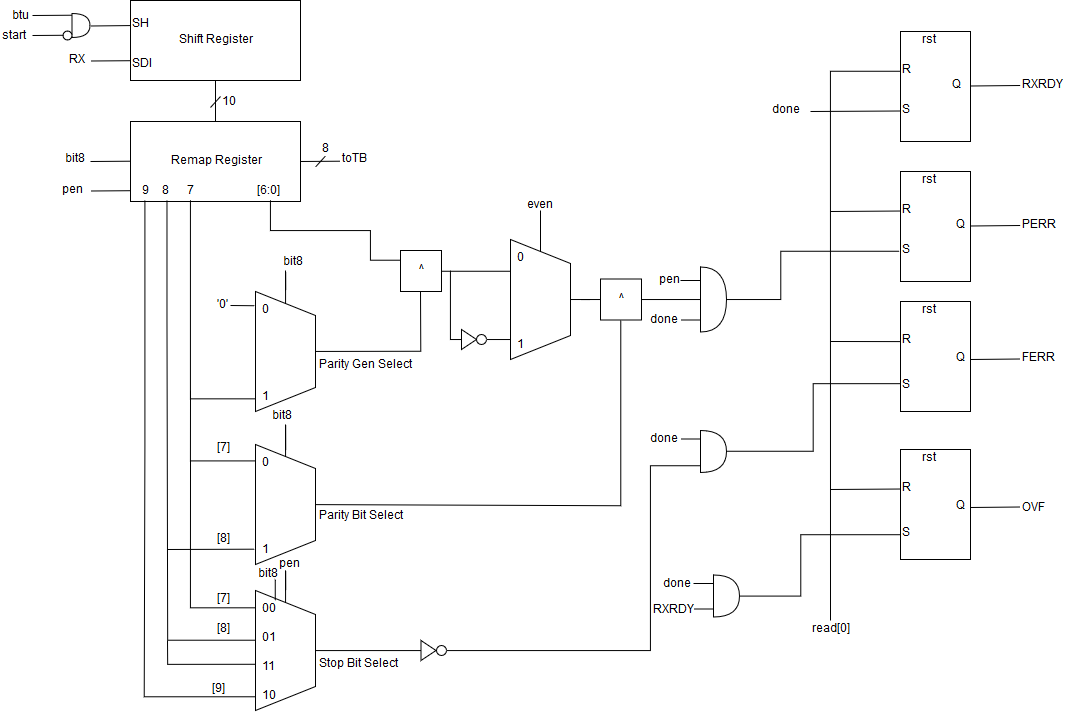
The OVF, or overflow error, flag is set when more data is received when the frame is already finished.

Figure : RX\_DATAPATH Block Diagram



#### 6.3.7.2: Block Diagram

Figure : RX\_Datapath Detailed Block Design



#### 6.3.7.3: Input/Output

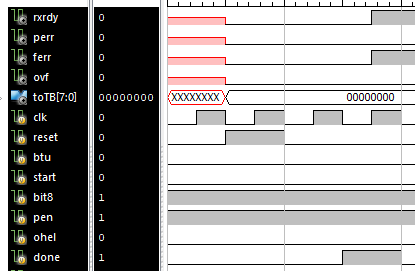
Table : RX\_Datapath Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| Clk | [7:0] toTB |
| Reset | rxrdy |
| Btu | perr |
| Start | ferr |
| bit8 | ovf |
| Pen |  |
| Ohel |  |
| Done |  |
| Read |  |
| SDI |  |

#### 6.3.7.4: Verification

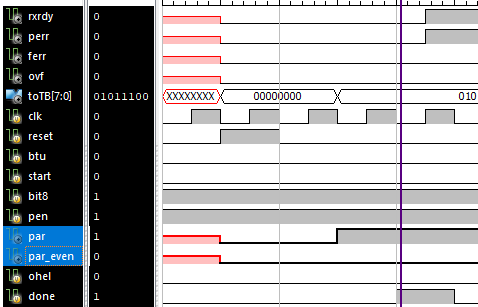
This verification confirms that RXRDY is set when done is high. It also shows that FERR will set because done and the stop bit are high as well.

Figure : RX\_Datapath Verification 1



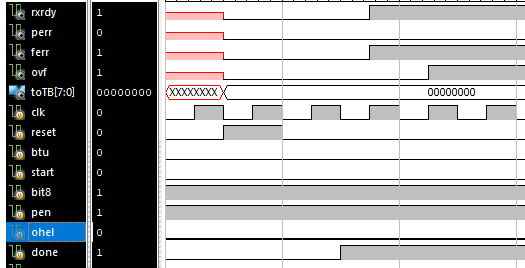
PERR is set when pen, done and the par OR par\_even are all high.

Figure : RX\_Datapath Verification 2



OVF is set when rxrdy and done are active at the same time.

Figure : RX\_Datapath Verification 3



#### 6.3.7.5: Source Code

See [A.13: RECEIVE ENGINE DATAPATH](#_Toc8444586)

#### 6.3.7.6: Receive Engine Shift Register

##### 6.3.7.6.1: Definition

The RX shift register takes in incoming data bits from the UART RX. When bit time up is set, the bits start shifting until start is set. The first bit in is the first out.

##### 6.3.7.6.2: Block Diagram

Figure : RX Shift Register Detailed Block Diagram



##### 6.3.7.6.3: Input/Output

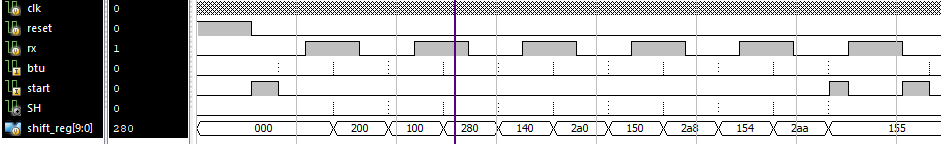
Table : RX Shift Register Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | [9:0] shift\_reg |
| rst |  |
| SH |  |
| SDI |  |

##### 6.3.7.6.4: Verification

The simulation shows that the bits have been shifting in from MSB to LSB whenever SH is high in the figure below.

Figure : RX Shift Register Verification



##### 6.3.7.6.5: Source Code

See [A.13: RECEIVE ENGINE DATAPATH](#_Toc8444586)

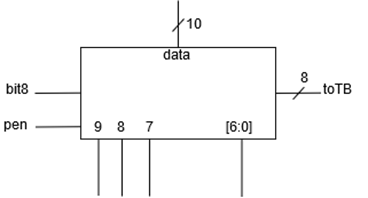
#### 6.3.7.7: Receive Engine Remap

##### 6.3.7.7.1: Definition

The remap is a combinational block used to re-position the data from the RX shift register depending on the parity control bits. This is to ensure the data frame is can be used to detect framing, parity and overflow error.

##### 6.3.7.7.2: Block Diagram

Figure : Remap Register Detailed Block Diagram



##### 6.3.7.7.3: Input/Output

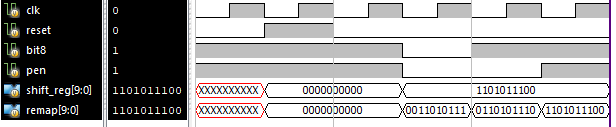
Table : Remap Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| bit8 | [9:0] remap |
| pen |  |
| [9:0] data |  |

##### 6.3.7.7.4: Verification

While bit8 and pen are inactive, the data from the shift register is re-position 2 bits to the right because there are only 8 bits. When bit8 is set, it is re-position 1 bit to the right because now there is 9 bits. With both bit8 and pen there is 10 bits so no re-position.

Figure : Remap Verification



##### 6.3.7.7.5: Source Code

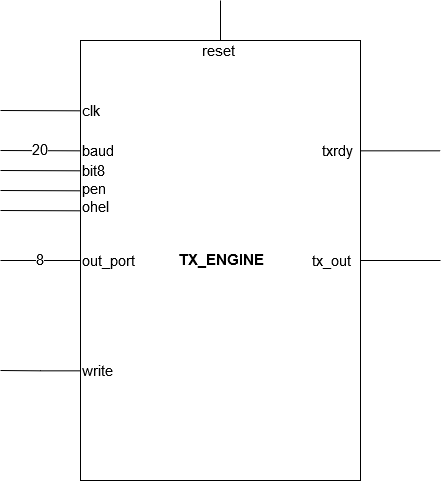
See [A.13: RECEIVE ENGINE DATAPATH](#_Toc8444586)

## 6.4: TRANSMIT ENGINE

### 6.4.1: DESCRIPTION

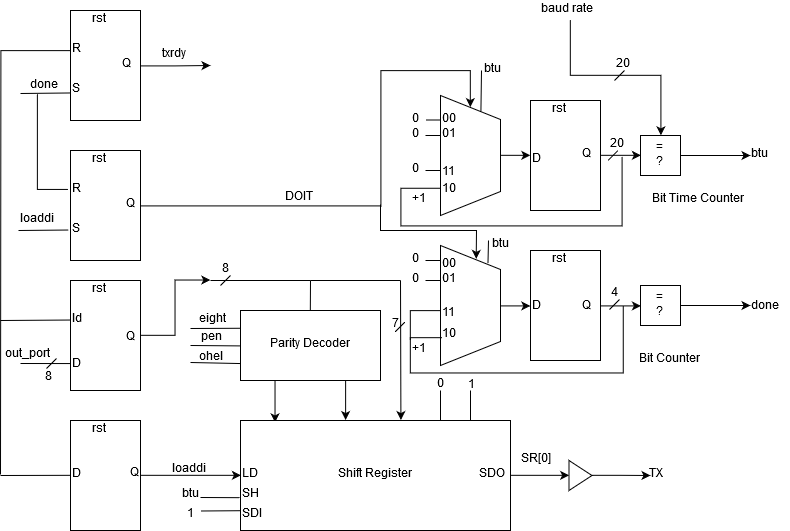
The purpose of this module is to provide half of the Universal Asynchronous Receiver Transmitter (UART). The focus will be the transmit. The transmit engine consists of a shift register that shifts out one bit of data at a time to be transmitted to a terminal. There will be a baud rate controller to adjust the speed in which the data is transmitted. The TX engine uses a bit time counter to count to the baud rate in order to change the transfer speed. The bit counter lets the engine know when it is done. When this happens RS flip flops output bit 0 and TXRDY is asserted to prepare for the next TX. The parity decoder gives the two most significant bits to the SR if the SR wants it. It depends on the input of eight, pen and ohel.

Figure : TX\_ENGINE Block Diagram



### 6.4.2: BLOCK DIAGRAM

Figure : TX\_ENGINE Detailed Block Diagram



### 6.4.3: INPUT/OUTPUT

Table : TX\_ENGINE Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | tx |
| reset | txrdy |
| [19:0] baud\_rate |  |
| [7:0] out\_port |  |
| bit8 |  |
| pen |  |
| ohel |  |

### 6.4.4: VERIFICATION

See section [7.3: TRANSMIT ENGINE VERIFICATION](#_Toc8425621)

### 6.4.5: SOURCE CODE

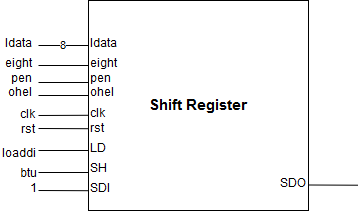
See [A.10: TRANSMIT ENGINE](#_Toc8444583)

### 6.4.6: SHIFT REGISTER

#### 6.4.6.1: Definition

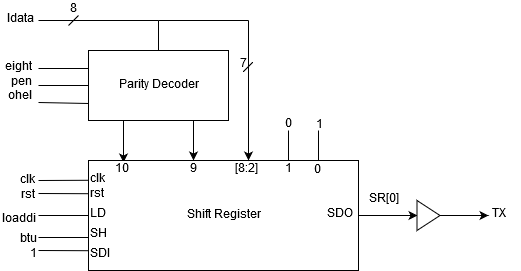
The shift register serially shifts 11-bit encoded data depending on the parity-bit controls from the parity-bit encoder. After shifting data internally, the shift register transmits LSB to MSB at the frequency of the baud rate. Transmit bit remains high until the start bit is sent.

Figure : Shift Register Block Diagram



#### 6.4.6.2: Block Diagram

Figure : Shift Register Detailed Block Diagram



#### 6.4.6.3: Input/Output

Table : Shift Register Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | SDO |
| reset |  |
| LD |  |
| SH |  |
| SDI |  |
| pen |  |
| ohel |  |
| eight |  |
| [7:0] ldata |  |

#### 6.4.6.3: Verification

See section [7.3: TRANSMIT ENGINE VERIFICATION](#_Toc8425621)

#### 6.4.6.4: Source Code

See [A.10: TRANSMIT ENGINE](#_Toc8444583)

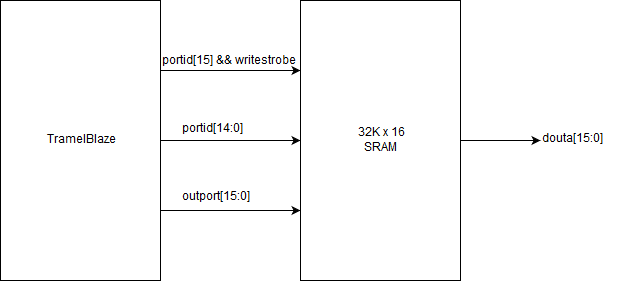
## 6.5: EMBEDDED STATIC RAM

### 6.5.1: DESCRIPTION

The 32K x 16 Embedded Static RAM takes up half of the memory space of the microprocessor. The SRAM stores every received byte from the UART and is never cleared. Immediately after the program is ran, a memory test is done and the memory goes through all the addresses from 0x8000 to 0xFFFF and tests it with repeating A’s, then 5’s, then the address and the address values. After that, the program runs normally. To do a memory dump, the program waits for a hex 25 or the % character to dump the memory. The contents in memory continue to increase every time a new byte is received.

### 6.5.2: BLOCK DIAGRAM

Figure : SRAM Block Diagram



### 6.5.3: INPUT/OUTPUT

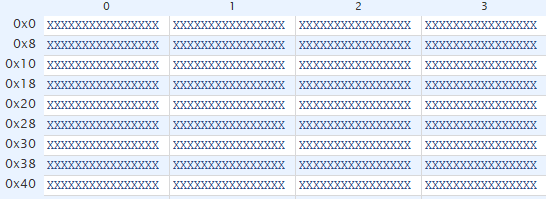
Table : SRAM Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clka | douta |
| wea |  |
| dina |  |
| addra |  |

### 6.5.4: VERIFICATION

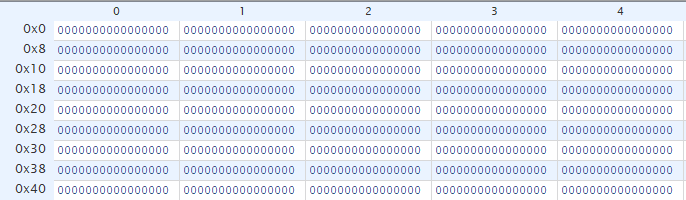
A verification is done to see if the memory takes in values to check for bit errors in the memory. The memory before reset is in an unknown state is shown in the figure below.

Figure : SRAM Verification 1



Memory is then filled with zeros after reset is shown in the figure below.

Figure : SRAM Verification 2

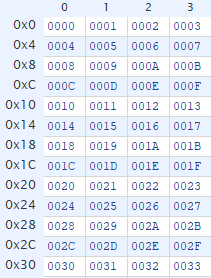


In the figures below, the memory is being filled with repeating A’s first in Figure 35 and followed by repeating 5’s in Figure 36 to test if any bits are stuck.

|  |  |
| --- | --- |
| Figure : SRAM Verification 3 | Figure : SRAM Verification 4 |

Lastly, the memory locations in the SRAM are being filled with their corresponding address value in Figure 37 below. This shows that the memory test is a success and the SRAM works.

Figure : SRAM Verification 5



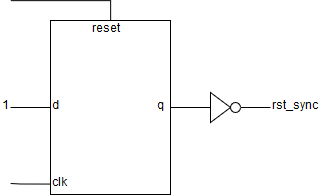
## 6.6: ASYNCHRONOUS-IN SYNCHRONOUS-OUT

### 6.6.1: DEFINITION

This module is used to bring the entire system into a known state at the same edge of the clock. This prevents unexpected results and metastability. This is done incase system reset de-asserts itself in the middle of the clock pulse so the synchronized reset signal will de-assert itself on the next active edge of the clock.

### 6.6.2: BLOCK DIAGRAM

Figure : AISO Block Detailed Diagram



### 6.6.3: INPUT/OUTPUT

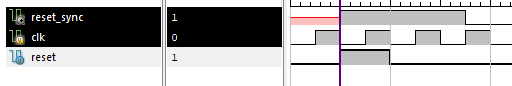
Table : AISO Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | reset\_sync |
| reset |  |

### 6.6.4: VERIFICATION

The figure below shows that when system reset is pressed, the synchronized reset will de-assert itself at the active edge of the clock.

Figure : AISO Verification



### 6.6.5: SOURCE CODE

See [A.4: ASYNCHRONOUS-IN SYNCHRONOUS-OUT REGISTER](#_Toc8444577)

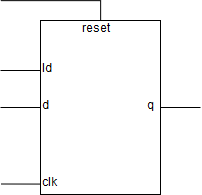
## 6.7: LOAD REGISTER

### 6.7.1: DEFINITION

This register takes in the out port signal from the TramelBlaze when the program is updating the switch configurations. It will only update output when that happens.

### 6.7.2: BLOCK DIAGRAM

Figure : Load Register Block Detailed Diagram



### 6.7.3: INPUT/OUTPUT

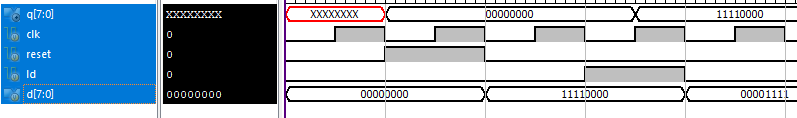
Table : Load Register Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | [7:0] UARTCONFIG |
| reset |  |
| Ld |  |
| [7:0] out\_port |  |

### 6.7.4: VERIFICATION

The figure below shows that the output will only change when the load is asserted.

Figure : Load Register Verification



### 6.7.5: SOURCE CODE

See [A.8: LOAD REGISTER](#_Toc8444581)

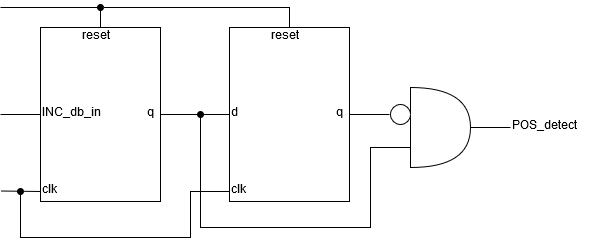
## 6.8: POSEDGE DETECT

### 6.8.1: DEFINITION

This module is used to capture a signal for one clock cycle. It is necessary in case a signal stays on too long and causes a specific hardware to do more than it is supposed to because of a long signal.

### 6.8.2: BLOCK DIAGRAM

Figure : Posedge Detect Detailed Block Diagram



### 6.8.3: INPUT/OUTPUT

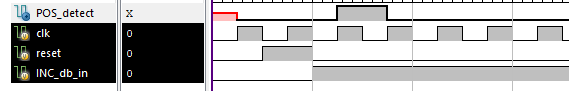
Table : Posedge Detect Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | POS\_detect |
| reset |  |
| INC\_db\_in |  |

### 6.8.4: VERIFICATION

The figure below shows that the PED captures a signal for one clock cycle even though signal remains on.

Figure : Posedge Detect Verification



### 6.8.5: SOURCE CODE

See [A.7: POSITIVE-EDGE DETECT](#_Toc8444580)

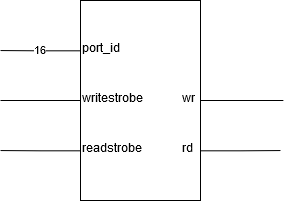
## 6.9: ADDRESS DECODER

### 6.9.1: DEFINITION

This combination block decodes the port ID from the TramelBlaze. It uses a low active port ID [15] as an enable. When read or write strobe is high, the bit position is the current write or read output index. If port ID is hex 0, then output[0] is used.

### 6.9.2: BLOCK DIAGRAM

Figure : Address Decoder Block Diagram



### 6.9.3: INPUT/OUTPUT

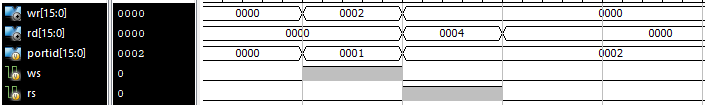
Table : Address Decoder Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| [15:0] port\_id | wr |
| writestrobe | rd |
| readstrobe |  |

### 6.9.4: VERIFICATION

When the port ID is hex 1 and write strobe is high, the write output reads hex 2. When the port ID is hex 2 and read strobe is high, the read output reads hex 4.

Figure : Address Decoder Verification



### 6.9.5: SOURCE CODE

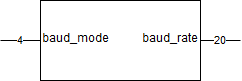
See [A.5: ADDRESS DECODER](#_Toc8444578)

## 6.10: BAUD DECODER

### 6.10.1: DEFINITION

This combination block determines the baud rate frequency for the UART protocol to allow communication with external devices. Frequencies must match in order to receive or transmit data bits correctly.

Figure : Baud Decoder Block Diagram



|  |  |  |  |
| --- | --- | --- | --- |
| **baud\_mode [7:4]** | **Bit Time** | **Baud Rate** | **Bit Time Count** |
| 0000 | 3.3333 ms | 300 | 333,333 |
| 0001 | 833.33 us | 1200 | 83,333 |
| 0010 | 416.66 us | 2400 | 41,667 |
| 0011 | 208.33 us | 4800 | 20,833 |
| 0100 | 104.16 us | 9600 | 10,417 |
| 0101 | 52.083 us | 19200 | 5,208 |
| 0110 | 26.041 us | 38400 | 2,604 |
| 0111 | 17.361 us | 57600 | 1,736 |
| 1000 | 8.6806 us | 115200 | 868 |
| 1001 | 4.3403 us | 230400 | 434 |
| 1010 | 2.1701 us | 460800 | 217 |
| 1011 | 1.0851 us | 921600 | 109 |

### 6.10.2: INPUT/OUTPUT

Table : Baud Decoder Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| [4:0] baud\_mode | [19:0] baud\_rate |

### 6.10.3: VERIFICATION

The baud mode switches do match up with the bit time count value for the baud rates in the figure below.

Figure : Baud Decoder Verification



### 6.10.4: SOURCE CODE

See [A.9: BAUD DECODER](#_Toc8444582)

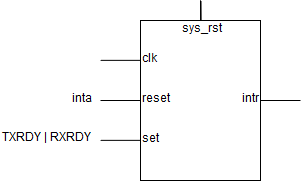
## 6.9: SR FLOP

### 6.11.1: DEFINITION

The set reset flip-flop’s purpose is to send an interrupt request to the TramelBlaze when set is high. Set gets its signal from TXRDY and RXRDY. Reset is active when the TramelBlaze acknowledges the interrupt.

### 6.11.2: BLOCK DIAGRAM

Figure : SR Flop Block Diagram



### 6.11.3: INPUT/OUTPUT

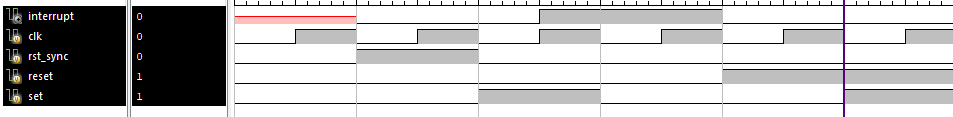
Table : SR Flop Signal Names

|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |
| clk | intr |
| sys\_rst |  |
| reset |  |
| set |  |

### 6.11.4: VERIFICATION

The figure below shows that interrupt only turns on when set turns high. It remains high until reset is asserted. It won’t turn on even if set is active while reset is high.

Figure : SR Flop Verification



### 6.11.5: SOURCE CODE

See [A.6: SET RESET FLOP](#_Toc8447379)

# SECTION 7: CHIP VERIFICATION PLAN

## 7.1: SOPC CORE VERIFICATION

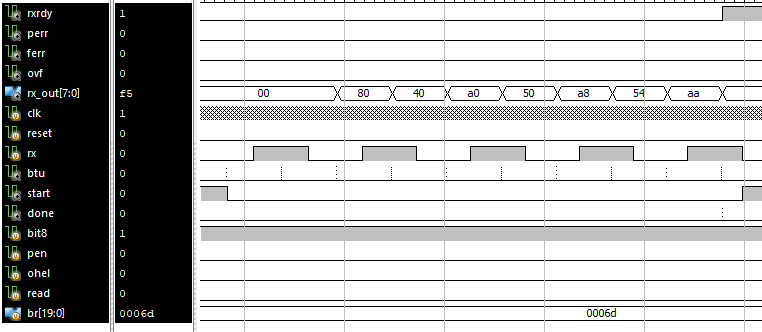
To prove that the SOPC\_CORE works with the TramelBlaze, this simulation log file shows that the banner and prompt of the program is being received. It receives a data bits and transmits them. In this case CSULB 460 is the banner and thomas:~$ is the prompt.

|  |
| --- |
| Isim log file  Isim P.20131013 (signature 0x7708f090)  This is a Full version of Isim.  Time resolution is 1 ps  # onerror resume  # wave add /  # run 1000 ns  Simulator is doing circuit initialization process.  Finished circuit initialization process.  Serial data transmitted=000000000000  data transmitted=00 Initial  leds=0000000000000000  New Data Byte Incoming for i= 0  {EIGHT,PEN,OHEL}=000  Incoming bit = # run all  PTR is 0000  leds=0000000000000001  1Serial data transmitted=011100000000  data transmitted=00 PTR is 0000  0Serial data transmitted=011100000000  data transmitted=00 PTR is 0001  0Serial data transmitted=011110000110  data transmitted=43 C  PTR is 0002  0Serial data transmitted=011110100110  data transmitted=53 S  PTR is 0003  00Serial data transmitted=011110101010  data transmitted=55 U  PTR is 0004  0Serial data transmitted=011110011000  data transmitted=4c L  PTR is 0005  0Serial data transmitted=011110000100  data transmitted=42 B  PTR is 0006  0Serial data transmitted=011101000000  data transmitted=20  PTR is 0007  1Serial data transmitted=011110000110  data transmitted=43 C  PTR is 0008  11Serial data transmitted=011110001010  data transmitted=45 E  PTR is 0009  1Serial data transmitted=011110000110  data transmitted=43 C  PTR is 000a  1Serial data transmitted=011110100110  data transmitted=53 S  PTR is 000b  1Serial data transmitted=011101000000  data transmitted=20  PTR is 000c  1Serial data transmitted=011101101000  data transmitted=34 4  PTR is 000d  1Serial data transmitted=011101101100  data transmitted=36 6  1PTR is 000e  1Serial data transmitted=011101100000  data transmitted=30 0  PTR is 000f  1Serial data transmitted=011100011010  data transmitted=0d  PTR is 0010  1Serial data transmitted=011100010100  data transmitted=0a  1PTR is 0000  1Serial data transmitted=011111101000  data transmitted=74 t  PTR is 0001  1  New Data Byte Incoming for i= 1  {EIGHT,PEN,OHEL}=001  Incoming bit = Serial data transmitted=011111010000  data transmitted=68 h  PTR is 0002  1Serial data transmitted=011111011110  data transmitted=6f o  0PTR is 0003  1Serial data transmitted=011111011010  data transmitted=6d m  PTR is 0004  0Serial data transmitted=011111000010  data transmitted=61 a  PTR is 0005  0Serial data transmitted=011111100110  data transmitted=73 s  PTR is 0006  0Serial data transmitted=011101110100  data transmitted=3a :  PTR is 0007  0Serial data transmitted=011111111100  data transmitted=7e ~  0PTR is 0008  0Serial data transmitted=011101001000  data transmitted=24 $  PTR is 0009  1Serial data transmitted=011101000000  data transmitted=20  leds=0000000000011000    DATA RECEIVED=1000000101  11111111111111 |

## 7.2: RECEIVE ENGINE VERIFICATION

When parity is set to 8N1 and the value is 01010101, the receive bit reads in bit in this order: 10101010. The stop bit as the last high signal of rx where done is high. Btu marks the k/2 of the parity bits. Start happens as soon as the mark goes low for k/2.

Figure : RX\_Engine Verification



## 7.3: TRANSMIT ENGINE VERIFICATION

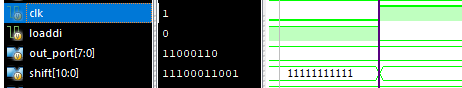
When using and observing the test bench file for the TX engine. We can find that for every time there is an out\_port data, the shift register will then take in the first two bits of 1 and 0 and then the outport and an MSB bit. The shift register is a 11-bit register while the out\_port is an 8-bit register. When pen is enabled, that is when parity bits matter.

Because the TX only triggers every time a bit is shifted out of the shift register, this simulation log is used to compare the values it takes from the shift register. They happen to be the same.

|  |
| --- |
| ISim log file  Running: C:\Users\vshoot\Documents\CECS\cecs460\Lab2\Lab2\tx\_engine\_tb\_isim\_beh.exe -intstyle ise -gui -tclbatch isim.cmd -wdb C:/Users/vshoot/Documents/CECS/cecs460/Lab2/Lab2/tx\_engine\_tb\_isim\_beh.wdb  ISim P.20131013 (signature 0x7708f090)  This is a Full version of ISim.  Time resolution is 1 ps  # onerror resume  # wave add /  # run 1000 ns  Simulator is doing circuit initialization process.  -----------------------------------------------------------------  CECS 460 Lab 2: Transmit Engine Test Bench  -----------------------------------------------------------------    Finished circuit initialization process.  {bit8, pen, ohel} = 000 --- Out\_Port = 10101010  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 11010101001  TX = # run all  10010101011  NP bit  {bit8, pen, ohel} = 001 --- Out\_Port = 10101011  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 11010101101  TX = 10110101011  NP bit  {bit8, pen, ohel} = 010 --- Out\_Port = 10101101  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 10010110101  TX = 10101101001  EP bit = 0  {bit8, pen, ohel} = 011 --- Out\_Port = 10110000  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 11011000001  TX = 10000011011  OP bit = 0  {bit8, pen, ohel} = 100 --- Out\_Port = 10110100  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 11011010001  TX = 10001011011  NP bit  {bit8, pen, ohel} = 101 --- Out\_Port = 10111001  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 11011100101  TX = 10100111011  NP bit  {bit8, pen, ohel} = 110 --- Out\_Port = 10111111  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 11011111101  TX = 10111111011  EP bit = 1  {bit8, pen, ohel} = 111 --- Out\_Port = 11000110  TRANSMIT (LSB to MSB) and SHIFT Comparison:  Shift = 11100011001  TX = 10011000111  OP bit = 1  Stopped at time : 96912 ns : File "C:/Users/vshoot/Documents/CECS/cecs460/Lab2/Lab2/tx\_engine\_tb.v" Line 79 |

You can see here, as soon as loaddi is active on the active edge of the clock, the shift register takes in the first two bits of 1 and 0 and then the rest of the out\_port plus another 1 bit.

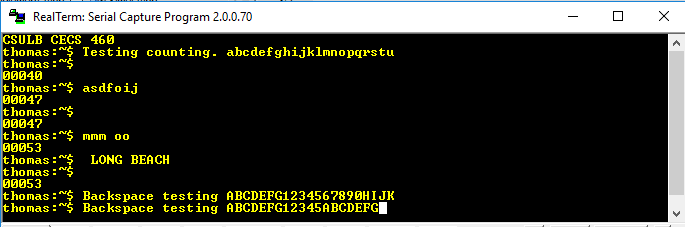
Figure : Transmit Engine Verification



# SECTION 8: CHIP LEVEL TEST

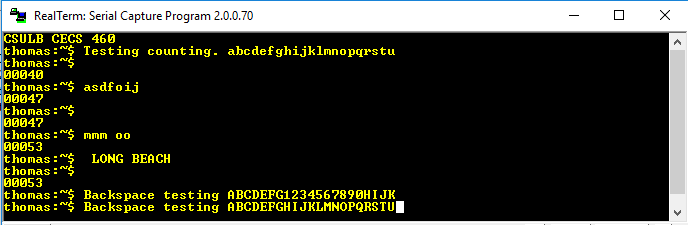
In the terminal images below, the banner and the prompt are shown. The prompt repeats itself after character limits hits 40 and it returns automatically, any time the return key is pressed, and any time the key @ or \* is pressed. When the @ key is pressed, the program will do a carriage return and linefeed before displaying the number of characters received since reset. Any time the \* key is pressed, “LONG BEACH” will be displayed. Verification 2 just shows that the backspace key can be erased any characters entered and also modify the inputs.

Figure : Program Verification 1



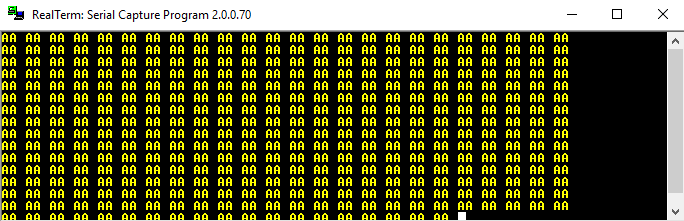
The last line of the two figures show that the backspace is working properly.

Figure : Program Verification 2



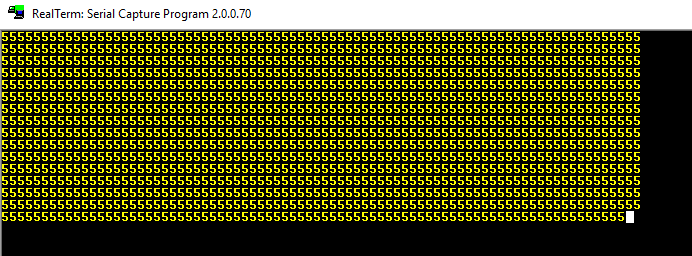
Here, the memory is tested. First repeated A’s are displayed, followed by repeated 5’s to show no bits are stuck and then the SRAM addresses are displayed after that. After the memory test, the banner and prompt are displayed normally. Then after every received input, the data is stored into the SRAM. To dump the SRAM, the ‘%’ character is inputted.

Figure : Memory Test 1



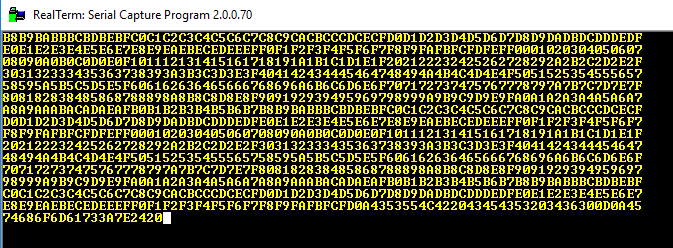
Repeating 5’s are done here and has proven that no bits have gotten stuck, therefore the SRAM is working properly.

Figure : Memory Test 2



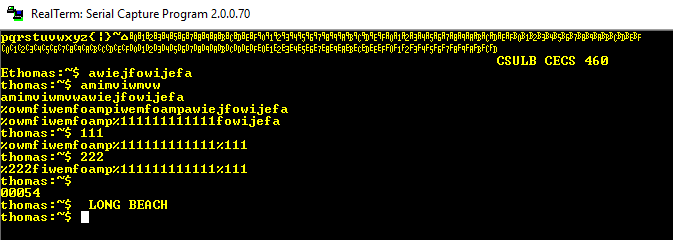
The figure below shows the last memory test. This dumps all the address locations in the SRAM.

Figure : Memory Test 3



A memory dump test with % key. The % key is stored into the SRAM and dumped as well. A carriage return and linefeed are also stored in memory and affects the prompt by overwriting it sometimes. However, if no carriage returns and linefeed are done, everything will display correctly.

Figure : Dump Memory Test



# APPENDIX:

This section includes all the Verilog code for every module and the assembly program code for the chip design, except for external sources.

## A.1: TOP LEVEL

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: top\_level.v //

// //

// Created by Thomas Nguyen on 5/01/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module top\_level(SYS\_CLK, SYS\_RST, i\_RX, i\_SW, o\_TX, o\_LED);

input SYS\_CLK, SYS\_RST, i\_RX;

input [7:0] i\_SW;

output o\_TX;

output [15:0] o\_LED;

wire w\_CLK, w\_RST, w\_TX, w\_RX;

wire [7:0] w\_SW;

wire [15:0] w\_LED;

TSI tsi(.SYS\_CLK(SYS\_CLK), .SYS\_RST(SYS\_RST), .i\_RX(i\_RX),

.i\_SW(i\_SW), .o\_TX(o\_TX), .o\_LED(o\_LED),

.i\_TX(w\_TX), .i\_LED(w\_LED), .o\_CLK(w\_CLK),

.o\_RST(w\_RST), .o\_RX(w\_RX), .o\_SW(w\_SW));

SOPC\_CORE cor(.clk100mhz(w\_CLK), .reset(w\_RST), .baudm(w\_SW[7:4]),

.bit8(w\_SW[3]), .pen(w\_SW[2]), .ohel(w\_SW[1]),

.uart\_txd\_in(w\_RX),

.uart\_rxd\_out(w\_TX), .leds(w\_LED));

endmodule

## A.2: TECHNOLOGY SPECIFIC INSTATIATIONS

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: top\_level.v //

// //

// Created by Thomas Nguyen on 5/01/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module TSI(SYS\_CLK, SYS\_RST, i\_RX, i\_SW, i\_TX, i\_LED,

o\_CLK, o\_RST, o\_RX, o\_SW, o\_TX, o\_LED);

input SYS\_CLK, SYS\_RST, i\_RX;

input [7:0] i\_SW;

input i\_TX;

input [15:0] i\_LED;

output o\_CLK, o\_RST, o\_RX;

output [7:0] o\_SW;

output o\_TX;

output [15:0] o\_LED;

//Input Buffer

IBUFG #(.IOSTANDARD("DEFAULT"))

SYSTEM\_CLK(.O(o\_CLK), .I(SYS\_CLK));

IBUF #(.IOSTANDARD("DEFAULT"))

SYSTEM\_RESET(.O(o\_RST), .I(SYS\_RST));

IBUF #(.IOSTANDARD("DEFAULT"))

RX(.O(o\_RX), .I(i\_RX));

IBUF #(.IOSTANDARD("DEFAULT"))

SWITCHES[7:1](.O(o\_SW[7:1]), .I(i\_SW[7:1]));

//Output Buffer

OBUF #(.IOSTANDARD("DEFAULT"))

TX(.O(o\_TX), .I(i\_TX));

OBUF #(.IOSTANDARD("DEFAULT"))

LED[15:0](.O(o\_LED), .I(i\_LED));

endmodule

## A.3: SOPC CORE

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: SOPC\_CORE.v //

// //

// Created by Thomas Nguyen on 5/01/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module SOPC\_CORE(clk100mhz, reset, baudm, bit8, pen, ohel,

uart\_txd\_in, uart\_rxd\_out, leds);

input clk100mhz, reset;

//Baud rate mode

input [3:0] baudm;

//Status Flags

input bit8, pen, ohel;

//Receive signal

input uart\_txd\_in;

//Transmit signal

output uart\_rxd\_out;

//16 LEDs

output [15:0] leds;

reg [15:0] leds;

//Tramelblaze

wire [15:0] inport, outport, portid;

wire rst\_sync;

wire ped\_d;

wire [7:0] rx\_out;

wire [7:0] UARTCONFIG;

wire [15:0] write\_d, read\_d, SRAM\_out;

wire [19:0] br;

// -------------------- INSTANTIATE MODULES -------------------------------

AISO\_register AISO(.clk(clk100mhz), .reset(reset), .reset\_sync(rst\_sync));

LD\_reg LR(.clk(clk100mhz), .reset(rst\_sync), .ld(write\_d[6]),

.d(outport[15:0]), .q(UARTCONFIG));

baud\_dec BAUD(.baud\_mode(UARTCONFIG[7:4]), .baud\_rate(br));

tx\_engine TX(.clk(clk100mhz), .reset(rst\_sync), .ld(write\_d[0]),

.bit8(UARTCONFIG[3]), .pen(UARTCONFIG[2]),

.ohel(UARTCONFIG[1]),

.baud\_rate(br), .out\_port(outport[7:0]),

.tx(uart\_rxd\_out), .txrdy(txrdy));

rx\_engine RX(.clk(clk100mhz), .reset(rst\_sync), .rx(uart\_txd\_in),

.bit8(UARTCONFIG[3]), .pen(UARTCONFIG[2]),

.ohel(UARTCONFIG[1]), .br(br),

.rxrdy(rxrdy), .perr(perr), .ferr(ferr), .ovf(ovf),

.rx\_out(rx\_out), .read(read\_d[0]));

posedge\_detect PEDTX(.clk(clk100mhz), .reset(rst\_sync), .INC\_db\_in(txrdy),

.POS\_detect(ped\_tx));

posedge\_detect PEDRX(.clk(clk100mhz), .reset(rst\_sync), .INC\_db\_in(rxrdy),

.POS\_detect(ped\_rx));

SR\_flop SRF(.clk(clk100mhz), .rst\_sync(rst\_sync),

.reset(interruptack), .set(ped\_d),

.interrupt(interrupt));

tramelblaze\_top TBT(.CLK(clk100mhz), .RESET(rst\_sync),

.IN\_PORT(inport), .INTERRUPT(interrupt),

.OUT\_PORT(outport), .PORT\_ID(portid),

.WRITE\_STROBE(writestrobe),

.READ\_STROBE(readstrobe),

.INTERRUPT\_ACK(interruptack));

SRAM SRAM(.clka(clk100mhz),

.wea(portid[15] && writestrobe), .addra(portid[14:0]),

.dina(outport), .douta(SRAM\_out));

//TXRDY OR RXRDY combo

assign ped\_d = ((ped\_tx | ped\_rx) == 1'b1);

//mux for inport

assign inport = (read\_d[5]) ? {8'b0, baudm, bit8, pen, ohel, 1'b0}:

(read\_d[1]) ? {11'b0, ovf, ferr, perr, txrdy, rxrdy}:

(read\_d[0]) ? {8'b0, rx\_out}:

SRAM\_out;

//Address decoder for write and read

addr\_dec DEC(.portid(portid), .ws(writestrobe), .wr(write\_d),

.rs(readstrobe), .rd(read\_d));

always@(posedge clk100mhz)

if(rst\_sync)

leds <= 16'b0;

else if(write\_d[2])

leds <= outport;

else

leds <= leds;

endmodule

## A.4: ASYNCHRONOUS-IN SYNCHRONOUS-OUT REGISTER

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: AISO\_register.v //

// //

// Created by Thomas Nguyen on 9/13/18 . //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

/\* Description:

\* Asynchronous in and synchronous out for reset.

\*/

module AISO\_register(clk, reset, reset\_sync);

input clk, reset;

output reset\_sync;

reg q, d\_reg;

always@(posedge clk or posedge reset)

if(reset) begin

d\_reg <= 1'b0;

q <= 1'b0;

end else begin

d\_reg <= 1'b1;

q <= d\_reg;

end

assign reset\_sync = ~q;

endmodule

## A.5: ADDRESS DECODER

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: addr\_dec.v //

// //

// Created by Thomas Nguyen on 4/08/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module addr\_dec(portid, ws, rs, wr, rd);

input ws, rs;

input [15:0] portid;

output reg [15:0] wr, rd;

always@(\*) begin

wr = 16'b0;

rd = 16'b0;

if(~portid[15]) begin

wr[portid[2:0]] = ws;

rd[portid[2:0]] = rs;

end

else begin

wr = wr;

rd = rd;

end

end

endmodule

## A.6: SET RESET FLOP

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: top\_level.v //

// //

// Created by Thomas Nguyen on 1/29/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module SR\_flop(clk, rst\_sync, reset, set, interrupt);

input clk, rst\_sync, reset, set;

output interrupt;

reg intr;

always@(posedge clk, posedge rst\_sync)

if(rst\_sync)

intr <= 1'b0;

else

case({reset, set})

{1'b0, 1'b1}: intr <= 1'b1;

{1'b1, 1'b0}: intr <= 1'b0;

default: intr <= intr;

endcase

assign interrupt = reset ? 1'b0 : intr;

endmodule

## A.7: POSITIVE-EDGE DETECT

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: posedge\_detect.v //

// //

// Created by Thomas Nguyen on 09/20/2018. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Description:

// A pulse maker that detects the first rising edge and generates

// a one clock pulse

////////////////////////////////////////////////////////////////////

module posedge\_detect(clk, reset, INC\_db\_in, POS\_detect);

input clk, reset, INC\_db\_in;

output POS\_detect;

reg q, d;

always@(posedge clk or posedge reset)

if(reset) begin

d <= 1'b0;

q <= 1'b0;

end else begin

d <= INC\_db\_in;

q <= d;

end

assign POS\_detect = ~q & d;

endmodule

## A.8: LOAD REGISTER

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: LD\_reg.v //

// //

// Created by Thomas Nguyen on 3/26/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module LD\_reg(clk, reset, ld, d, q);

input clk, reset, ld;

input [7:0] d;

output reg [7:0] q;

//Sequential Block. If load is set, output gets input else stays the same

always@(posedge clk, posedge reset)

if(reset) q <= 8'b0;

else if(ld) q <= d;

else q <= q;

endmodule

## A.9: BAUD DECODER

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: baud\_dec.v //

// //

// Created by Thomas Nguyen on 3/17/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module baud\_dec(baud\_mode, baud\_rate);

input [3:0] baud\_mode;

output reg [19:0] baud\_rate;

// ------------ Get Baud Rate --------- //

always@(\*)

case(baud\_mode)

// ------------------ Count No. ------ Rate

4'b0000 : baud\_rate = 20'd333\_333; // 300

4'b0001 : baud\_rate = 20'd083\_333; // 1200

4'b0010 : baud\_rate = 20'd041\_667; // 2400

4'b0011 : baud\_rate = 20'd020\_833; // 4800

4'b0100 : baud\_rate = 20'd010\_417; // 9600

4'b0101 : baud\_rate = 20'd005\_208; // 19200

4'b0110 : baud\_rate = 20'd002\_604; // 38400

4'b0111 : baud\_rate = 20'd001\_736; // 57600

4'b1000 : baud\_rate = 20'd000\_868; //115200

4'b1001 : baud\_rate = 20'd000\_434; //230400

4'b1010 : baud\_rate = 20'd000\_217; //460800

default : baud\_rate = 20'd000\_109; //921600

endcase

endmodule

## A.10: TRANSMIT ENGINE

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: tx\_engine.v //

// //

// Created by Thomas Nguyen on 2/18/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module tx\_engine( clk, reset, ld, bit8, pen, ohel, baud\_rate,

out\_port, tx, txrdy);

input clk, reset, ld, bit8, pen, ohel;

input [7:0] out\_port;

input [19:0] baud\_rate; //Baud count value

output tx; //Transmit

//flip flops

output reg txrdy; //Transmit Ready

reg doit, loaddi;

//registers

reg [7:0] ldata; //Ldata

reg [10:0] shift; //Shift Register

//counters

reg [3:0] bc, bc\_i; //Bit Count Value

reg [19:0] btc, btc\_i; //Bit Time Count Value

wire done, btu;

reg [1:0] parity; //2 MSB for Shift Register

////////////COUNTERS//////////////////

//bit counter

assign done = (bc\_i == 4'b1011);

always@(\*)

if(done) bc = 4'b0;

else if({doit, btu} == 2'b11)

bc = bc\_i + 4'b1;

else if({doit, btu} == 2'b10)

bc = bc\_i;

else bc = 4'b0;

always@(posedge clk, posedge reset)

if(reset) bc\_i <= 4'b0;

else bc\_i <= bc;

//bit time counter

assign btu = (btc\_i == baud\_rate);

always@(\*)

if(btu) btc = 20'b0;

else if({doit, btu} == 2'b10)

btc = btc\_i + 20'b1;

else btc = 20'b0;

always@(posedge clk, posedge reset)

if(reset) btc\_i <= 20'b0;

else btc\_i <= btc;

// ------------ FLIP FLOPS ------------ //

//loaddi flop

always@(posedge clk, posedge reset)

if(reset) loaddi <= 1'b0;

else loaddi <= ld;

//RS flop called DOIT

always@(posedge clk, posedge reset)

if(reset) doit <= 1'b0;

else if(done) doit <= 1'b0;

else if(loaddi) doit <= 1'b1;

else doit <= doit;

//TX bit

assign tx = shift[0];

//TXRDY RS flop (LOW ACTIVE)

always@(posedge clk, posedge reset)

if(reset) txrdy <= 1'b1;

else if(ld) txrdy <= 1'b0;

else if(done) txrdy <= 1'b1;

else txrdy <= txrdy;

// ------------ REGISTERS ------------ //

//ldata register

always@(posedge clk, posedge reset)

if(reset) ldata <= 8'b0;

else if(ld) ldata <= out\_port;

else ldata <= ldata;

//shift register

always@(posedge clk, posedge reset)

if(reset) shift <= 11'h7ff;

else if(loaddi) shift <= {parity, ldata[6:0], 1'b0, 1'b1};

else if(btu) shift <= {1'b1, shift[10:1]};

else shift <= shift;

//2-bit data for parity

always@(\*)

case({bit8, pen, ohel})

3'h2 : parity = { 1'b1, (^ldata[6:0])};

3'h3 : parity = { 1'b1, ~(^ldata[6:0])};

3'h4 : parity = { 1'b1, ldata[7]};

3'h5 : parity = { 1'b1, ldata[7]};

3'h6 : parity = { (^ldata), ldata[7]};

3'h7 : parity = {~(^ldata), ldata[7]};

default: parity = 2'b11;

endcase

endmodule

## A.11: RECEIVE ENGINE

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: rx\_engine.v //

// //

// Created by Thomas Nguyen on 3/26/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module rx\_engine(clk, reset, rx, bit8, pen, ohel, br, rxrdy, perr,

ferr, ovf, rx\_out, read);

input clk, reset;

input rx, bit8, pen, ohel, read;

input [19:0] br;

output rxrdy, perr, ferr, ovf;

output [7:0] rx\_out;

wire btu, done, start;

rx\_control RXC(.clk(clk), .reset(reset), .rx(rx),

.baud(br), .start(start),

.bit8(bit8), .pen(pen), .done(done), .btu(btu));

rx\_datapath RXD(.clk(clk), .reset(reset), .btu(btu),

.start(start), .bit8(bit8), .pen(pen),

.ohel(ohel), .done(done), .SDI(rx),

.rxrdy(rxrdy), .perr(perr), .ferr(ferr),

.ovf(ovf), .toTB(rx\_out), .read(read));

endmodule

## A.12: RECEIVE ENGINE CONTROL UNIT

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: rx\_control.v //

// //

// Created by Thomas Nguyen on 3/18/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module rx\_control(clk, reset, rx, baud, start, bit8, pen, done, btu);

input clk, reset;

input rx;

input bit8, pen;

input [19:0] baud;

output btu;

output done;

//Present Output

output start;

reg start;

reg doit;

reg [1:0] state;

//Next Outputs

reg nstart, ndoit;

reg [1:0] nstate;

//Counters

reg [3:0] bc, bc\_i; //Bit Count Value

reg [19:0] btc, btc\_i; //Bit Time Count Value

reg [3:0] num;

wire [19:0] start\_mux;

////////////COUNTERS//////////////////

//Bit Time Counter

assign btu = (btc\_i == start\_mux);

always@(\*)

if(btu) btc = 20'b0;

else if({doit, btu} == 2'b10)

btc = btc\_i + 20'b1;

else btc = 20'b0;

always@(posedge clk, posedge reset)

if(reset) btc\_i <= 20'b0;

else btc\_i <= btc;

//Bit Counter

assign done = (bc\_i == num);

always@(\*)

if(done) bc = 4'b0;

else if({doit, btu} == 2'b11)

bc = bc\_i + 4'b1;

else if({doit, btu} == 2'b10)

bc = bc\_i;

else bc = 4'b0;

always@(posedge clk, posedge reset)

if(reset) bc\_i <= 4'b0;

else bc\_i <= bc;

//Combination logic for Bit Counter to increment to

always@(\*)

case({bit8, pen})

2'b00 : num = 4'h9;

2'b11 : num = 4'hB;

default: num = 4'hA;

endcase

//Start Select Mux for K and K/2

assign start\_mux = (start) ? (baud >> 1) : baud;

//FINITE STATE MACHINE (FSM)

//Modified Moore to ensure that the start bit remains

//active for half a bit time.

always@(posedge clk, posedge reset)

if(reset) {state, start, doit} <= 4'b00\_0\_0;

else {state, start, doit} <= {nstate, nstart, ndoit};

always@(\*)

case(state)

2'b00 : {nstate, nstart, ndoit} = (rx) ? 4'b00\_0\_0 :

4'b01\_1\_1 ;

2'b01 : {nstate, nstart, ndoit} = (rx) ? 4'b00\_0\_0 :

(!btu) ? 4'b01\_1\_1 :

4'b10\_0\_1 ;

2'b10 : {nstate, nstart, ndoit} = (done) ? 4'b00\_0\_0 :

4'b10\_0\_1 ;

default: {nstate, nstart, ndoit} = 4'b00\_0\_0 ;

endcase

endmodule

## A.13: RECEIVE ENGINE DATAPATH

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// File name: rx\_datapath.v //

// //

// Created by Thomas Nguyen on 3/18/19. //

// Copyright c 2019 Thomas Nguyen. All rights reserved. //

// //

// //

// In submitting this file for class work at CSULB //

// I am confirming that this is my work and the work //

// of no one else. In submitting this code I acknowledge that //

// plagiarism in student project work is subject to dismissal. //

// from the class //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module rx\_datapath(clk, reset, btu, start, bit8, pen, ohel, done,

SDI, rxrdy, perr, ferr, ovf, read, toTB);

input clk, reset;

input btu, start, bit8, pen, ohel, done, read;

//Serial Data Input

input SDI;

//Output of remap to tramel blaze

output [7:0] toTB;

//Status for recieve, parity error, framing error and overflow

output reg rxrdy, perr, ferr, ovf;

reg [9:0] shift\_reg, remap;

//Mux Signals that are used to create load signals to Status Regs

wire par, par\_even, gen, SH;

reg stop;

/////FOR SHIFT REGISTER//////

//Shift signal

assign SH = btu & ~start;

always@(posedge clk, posedge reset)

if(reset) shift\_reg <= 10'b0;

else if(SH) shift\_reg <= {SDI, shift\_reg[9:1]};

else shift\_reg <= shift\_reg;

/////FOR REMAP COMBO////////

assign toTB = (bit8) ? remap[7:0] : {1'b0, remap[6:0]};

always@(\*)

case({bit8, pen})

2'b00 : remap = {2'b0, shift\_reg[9:2]};

2'b01 : remap = {1'b0, shift\_reg[9:1]};

2'b10 : remap = {1'b0, shift\_reg[9:1]};

default: remap = shift\_reg;

endcase

//Parity Gen Select

assign gen = (bit8) ? remap[7] : 1'b0;

//Even Parity Mux

assign par\_even = (~ohel) ? (remap[6:0] ^ gen) :

~(remap[6:0] ^ gen) ;

//Parity Bit Select

assign par = (bit8) ? remap[8] : remap[7];

//Stop Bit Select

always@(\*)

case({bit8, pen})

2'b00 : stop = ~remap[7];

2'b11 : stop = ~remap[9];

default: stop = ~remap[8];

endcase

/////STATUS REGISTERS///////

//RXRDY REG

always@(posedge clk, posedge reset)

if(reset) rxrdy <= 1'b0;

else if(read) rxrdy <= 1'b0;

else if(done) rxrdy <= 1'b1;

else rxrdy <= rxrdy;

//PERR REG

always@(posedge clk, posedge reset)

if(reset) perr <= 1'b0;

else if(read) perr <= 1'b0;

else if(done & (par ^ par\_even) & pen)

perr <= 1'b1;

else perr <= perr;

//FERRR REG

always@(posedge clk, posedge reset)

if(reset) ferr <= 1'b0;

else if(read) ferr <= 1'b0;

else if(done & stop) ferr <= 1'b1;

else ferr <= ferr;

//OVF REG

always@(posedge clk, posedge reset)

if(reset) ovf <= 1'b0;

else if(read) ovf <= 1'b0;

else if(done & rxrdy)

ovf <= 1'b1;

else ovf <= ovf;

endmodule

## A.14: ASSEMBLY PROGRAM

; Thomas Nguyen

; 016238935

; CECS 460

; SOPC TSI External Memory w/ UART

;

; Assembly code for UART communicating with an

; external memory outside of the MCU. The memory

; will be written to on every receive from UART.

; It can be accessed and get a memory dump by the use

; of the "%" character.

; ================================================

; Declare EQU constants

; ================================================

ZEROS EQU 0000

ONE EQU 0001

; ================================================

; Register EQU

; ================================================

SCRATCH EQU R0 ; Scratch RAM

SP EQU R1 ; Stack Pointer

SRAM EQU R2 ; SRAM pointer

TEMP EQU R3 ; Temp reg that takes Scratch mem

LEDS EQU R4 ;

STATUS EQU R5 ; Status register

COUNT EQU R6 ; Char Counter

DISPLAY EQU RA ; Keeps track of what is being displayed

RX EQU RB ; Received data bits

CONFIG EQU RC ; The config switch register

S\_INDEX EQU RD ; Keep track of inputted strings

DELAY EQU RE ; Delays leds

DELAY2 EQU RF ;

;Temp Reg for Bin to ASCII

RE EQU R7 ; Temp Reg for current count

RD EQU R8 ; Temp Reg for max count

RB EQU R9 ; Temp Reg for Adding

; ================================================

; Initialization

; ================================================

START INPUT CONFIG, 0005

OUTPUT CONFIG, 0006

LOAD COUNT, ZEROS

LOAD SP, ZEROS

LOAD CONFIG, ZEROS

LOAD LEDS, ONE

LOAD TEMP, ZEROS

LOAD STATUS, ZEROS

LOAD DISPLAY, 0008

LOAD S\_INDEX, ZEROS

LOAD DELAY, ZEROS

LOAD DELAY2, ZEROS

LOAD SRAM, 8000

; ================================================

; Initialize Scratch Memory with ASCII Hex values of characters from message

; ================================================

LOAD SCRATCH, 0043 ; C

CALL STORER

LOAD SCRATCH, 0053 ; S

CALL STORER

LOAD SCRATCH, 0055 ; U

CALL STORER

LOAD SCRATCH, 004C ; L

CALL STORER

LOAD SCRATCH, 0042 ; B

CALL STORER

LOAD SCRATCH, 0020 ; Space

CALL STORER

LOAD SCRATCH, 0043 ; C

CALL STORER

LOAD SCRATCH, 0045 ; E

CALL STORER

LOAD SCRATCH, 0043 ; C

CALL STORER

LOAD SCRATCH, 0053 ; S

CALL STORER

LOAD SCRATCH, 0020 ; Space

CALL STORER

LOAD SCRATCH, 0034 ; 4

CALL STORER

LOAD SCRATCH, 0036 ; 6

CALL STORER

LOAD SCRATCH, 0030 ; 0

CALL STORER

LOAD SCRATCH, 000D ; <CR>

CALL STORER

LOAD SCRATCH, 000A ; <LF>

CALL STORER ; Stored @ 0x0F, SP @ 0x10

LOAD SP, 00F0 ;

LOAD SCRATCH, 000D

CALL STORER

LOAD SCRATCH, 000A ; <LF>

CALL STORER

LOAD SP, 8000 ; Set to Addr 8000 to prepare for mem\_dump

ENINT

; ================================================

; MAIN LOOP for LEDs

; ================================================

MAIN OUTPUT LEDS, 0002

COMP LEDS, 9000

JUMPNZ LEDDELAY

INPUT CONFIG, 0005

OUTPUT CONFIG, 0006

LOAD LEDS, ONE

JUMP MAIN

LEDDELAY RL LEDS

LOAD DELAY, FFFF

LOAD DELAY2, 0004

COUNT\_DWN SUB DELAY, ONE

COMP, DELAY, ZEROS

JUMPNZ COUNT\_DWN

LOAD DELAY, FFFF

SUB DELAY2, ONE

COMP DELAY2, ZEROS

JUMPNZ COUNT\_DWN

JUMP MAIN

; ================================================

; Bin to ASCII

; ================================================

ADDRESS 0200

BIN2ASCII ADD COUNT, S\_INDEX

LOAD S\_INDEX, ZEROS

LOAD RE, COUNT

LOAD RD, 2710 ; 10,000

CALL FINDIT

ADD RB, 0030

STORE RB, 00F2

LOAD RD, 03E8 ; 1,000

CALL FINDIT

ADD RB, 0030

STORE RB, 00F3

LOAD RD, 0064 ; 100

CALL FINDIT

ADD RB, 0030

STORE RB, 00F4

LOAD RD, 000A ; 10

CALL FINDIT

ADD RB, 0030

STORE RB, 00F5

LOAD RD, ONE

CALL FINDIT

ADD RB, 0030 ; 1

STORE RB, 00F6

RETURN ; return to ISR

; ================================================

; Subroutines for Bin to ASCII

; ================================================

ADDRESS 0300

FINDIT LOAD RB, ZEROS

FNDLOOP SUB RE, RD ; RE <- RE - RD

JUMPC RESTORE ; if carry jump

ADD RB, ONE

JUMP FNDLOOP

RESTORE ADD RE, RD ; RE <- RE + RD

RETURN ; return to procedure

; ================================================

; Store to scratch ram and increment pointer

; ================================================

STORER STORE SCRATCH, (SP)

ADD SP, ONE

RETURN

; ================================================

; Prompt (tomas:~$)

; ================================================

ADDRESS 0400

PROMPT LOAD SCRATCH, 0074 ; t

CALL STORER ; Stored @ 0x00

LOAD SCRATCH, 0068 ; h

CALL STORER

LOAD SCRATCH, 006F ; o

CALL STORER

LOAD SCRATCH, 006D ; m

CALL STORER

LOAD SCRATCH, 0061 ; a

CALL STORER

LOAD SCRATCH, 0073 ; s

CALL STORER

LOAD SCRATCH, 003A ; :

CALL STORER

LOAD SCRATCH, 007E ; ~

CALL STORER

LOAD SCRATCH, 0024 ; $

CALL STORER

LOAD SCRATCH, 0020 ; Space

CALL STORER ; Stored @ 0x09

RETURN

; ================================================

; Home Town and Backspace (LONG BEACH<CR><LF><BS>)

; ================================================

HOMETOWN LOAD SCRATCH, 004C ; L

CALL STORER ; Stored @ 0x0A

LOAD SCRATCH, 004F ; O

CALL STORER

LOAD SCRATCH, 004E ; N

CALL STORER

LOAD SCRATCH, 0047 ; G

CALL STORER

LOAD SCRATCH, 0020 ; Space

CALL STORER

LOAD SCRATCH, 0042 ; B

CALL STORER

LOAD SCRATCH, 0045 ; E

CALL STORER

LOAD SCRATCH, 0041 ; A

CALL STORER

LOAD SCRATCH, 0043 ; C

CALL STORER

LOAD SCRATCH, 0048 ; H

CALL STORER

LOAD SCRATCH, 000D ; <CR>

CALL STORER ; Stored @ 0x14

LOAD SCRATCH, 000A ; <LF>

CALL STORER ; Stored @ 0x15

LOAD SCRATCH, 0008 ; Backspace

CALL STORER ; Stored @ 0x16

LOAD SCRATCH, 0020 ; Space

CALL STORER ; 17

LOAD SCRATCH, 0008 ; Backspace

CALL STORER ; Stored @ 0x18

LOAD SP, 0D00

LOAD SCRATCH, ZEROS

CALL STORER

LOAD SP, ZEROS ; Set to Addr 0

RETURN

; ================================================

; Interrupt Service Routine

; ================================================

ADDRESS 0500

ISR INPUT STATUS, ONE ; read in status flag

AND STATUS, 0002 ; obtain txrdy

COMP STATUS, 0002 ; check if tx is high

CALLZ TXRDY

INPUT STATUS, ONE

ISR2 COMP DISPLAY ZEROS

CALLZ RXRDY

RETEN

; ================================================

; TX ready

; ================================================

TXRDY COMP Display, 0008 ;MemTest Start

JUMPZ MEM\_TEST0

COMP Display, 0009

JUMPZ MEM\_TEST1

COMP Display, 000A

JUMPZ MEM\_TEST2

COMP Display, 000B

JUMPZ MEM\_TEST3

COMP Display, 000C

JUMPZ MEM\_TEST4

COMP Display, 000D ;MemTest End

JUMPZ MEM\_TEST5

COMP DISPLAY, ONE

JUMPNZ PROMPT\_O

; ================================================

; Displays Banner

; ================================================

BANNER\_O COMP SP, 0011

JUMPNZ TX\_OUT

LOAD SP, ZEROS

CALL PROMPT

CALL HOMETOWN

LOAD DISPLAY, 0002

; ================================================

; Displays Prompt

; ================================================

PROMPT\_O COMP DISPLAY, 0002

JUMPNZ HOMETOWN\_O

COMP SP, 000A

JUMPNZ TX\_OUT

LOAD TEMP, RX

LOAD DISPLAY, ZEROS ; finish with prompt message

; ================================================

; Displays Home town

; ================================================

HOMETOWN\_O COMP DISPLAY, 0003

JUMPNZ CHAR\_CNT\_O

COMP SP, 0016

JUMPNZ TX\_OUT

LOAD SP, ZEROS

LOAD DISPLAY, 0002

LOAD TEMP, ZEROS

OUTPUT TEMP, ZEROS

; ================================================

; Displays # of received characters

; ================================================

CHAR\_CNT\_O COMP DISPLAY, 0004

JUMPNZ DUMP\_MEM\_O

COMP SP, 00F7

JUMPNZ TX\_OUT

LOAD SP, 0014

LOAD DISPLAY, 0006

; ================================================

; Dumps contents of SRAM

; ================================================

DUMP\_MEM\_O COMP DISPLAY, 0007

JUMPNZ BACKSPACE\_O

INPUT TEMP, (SP)

OUTPUT TEMP, ZEROS

ADD SP, ONE

COMP SP, SRAM

JUMPC TX\_OUT

; ================================================

; Displays Backspace

; ================================================

BACKSPACE\_O COMP DISPLAY, 0005

JUMPNZ CAR\_RET\_O

COMP SP, 0019

JUMPNZ TX\_OUT

LOAD SP, ZEROS

LOAD TEMP, RX

LOAD DISPLAY, ZEROS

; ================================================

; Displays Carriage Return

; ================================================

CAR\_RET\_O COMP DISPLAY, 0006

RETURNNZ

COMP SP, 0016

JUMPNZ TX\_OUT

LOAD SP, ZEROS

LOAD DISPLAY, 0002

LOAD TEMP, ZEROS

OUTPUT TEMP, ZEROS

JUMP EXIT\_TX

; ================================================

; Transmits Characters through UART

; ================================================

TX\_OUT FETCH TEMP, (SP) ; fetch from memory pointed by sp

OUTPUT TEMP, ZEROS ; output memory content @loc

ADD SP, ONE ; Increment through SP

EXIT\_TX RETURN

; ================================================

; Memory Testing; Display repeating A's, 5's & Address

; ================================================

MEM\_TEST0 LOAD SCRATCH, AAAA

OUTPUT SCRATCH, (SP)

ADD SP, ONE

COMP SP, FFFF

JUMPNZ MEM\_TEST0

LOAD SCRATCH, ZEROS

OUTPUT SCRATCH, ZEROS

LOAD SP, 8000

LOAD DISPLAY, 0009

MEM\_TEST1 INPUT SCRATCH, (SP) ;Display A's

OUTPUT SCRATCH, ZEROS

ADD SP, ONE

COMP SP, FFFF

RETURNC

LOAD SP, 8000

MEM\_TEST2 LOAD SCRATCH, 5555

OUTPUT SCRATCH, (SP)

ADD SP, ONE

COMP SP, FFFF

JUMPNZ MEM\_TEST2

LOAD SP, 8000

LOAD DISPLAY, 000B

MEM\_TEST3 INPUT SCRATCH, (SP) ;Display 5's

OUTPUT SCRATCH, ZEROS

ADD SP, ONE

COMP SP, FFFF

RETURNC

LOAD SP, 8000

LOAD SCRATCH, ZEROS

LOAD DISPLAY, 000C

MEM\_TEST4 OUTPUT SCRATCH, (SP) ;Load Address

ADD SCRATCH, ONE

ADD SP, ONE

COMP SP, FFFF

JUMPNZ MEM\_TEST4

LOAD SP, 8000

OUTPUT SCRATCH, ZEROS

LOAD DISPLAY, 000D

MEM\_TEST5 INPUT SCRATCH, (SP) ;Display Address

OUTPUT SCRATCH, ZEROS

ADD SP, ONE

COMP SP, FFFF

RETURNC

LOAD SCRATCH, ZEROS

LOAD SP, 00F0

FETCH TEMP, (SP)

OUTPUT TEMP, ZEROS

ADD SP, ONE

FETCH TEMP, (SP)

OUTPUT TEMP, ZEROS

LOAD TEMP, ZEROS

LOAD SP, ZEROS

LOAD DISPLAY, ONE

RETURN

; ================================================

; RX ready

; ================================================

RXRDY AND STATUS, 001C

COMP STATUS, 0000

CALLNZ ERRORF

INPUT RX, ZEROS

COMP RX, ZEROS

JUMPZ EXIT\_RX

COMP RX, TEMP

JUMPZ RXFB

COMP RX, 002A ; \*

JUMPZ HOMETOWN\_I

COMP RX, 0040 ; @

JUMPZ CHAR\_CNT\_I

COMP RX, 007F ; Backspace

JUMPZ BACKSPACE\_I

COMP RX, 000D ; CR

JUMPZ CAR\_RET\_I

COMP RX, 0025 ; %

JUMPZ DUMP\_MEM

ADD S\_INDEX, ONE

OUTPUT RX, ZEROS

OUTPUT RX, (SRAM) ; Received Byte to SRAM

ADD SRAM, ONE

LOAD TEMP, RX

COMP S\_INDEX, 0028

JUMPZ CAR\_RET\_I

EXIT\_RX RETEN

RXFB LOAD SP, 0D00

FETCH RX, (SP)

LOAD TEMP, RX

RETEN

; ================================================

; PERR, FERR, or OVF flags set

; ================================================

ERRORF OUTPUT STATUS, 0002 ; Output Error

RETURN

; ================================================

; Receive Home town input

; ================================================

HOMETOWN\_I LOAD DISPLAY, 0003

LOAD SP, 0009

JUMP RESETSTRING

; ================================================

; Receive character count input

; ================================================

CHAR\_CNT\_I CALL BIN2ASCII

LOAD DISPLAY 0004

LOAD SP, 00F0

JUMP RESETSTRING

; ================================================

; Receive Backspace

; ================================================

BACKSPACE\_I COMP S\_INDEX, ZEROS

JUMPZ RXFB

LOAD DISPLAY, 0005

LOAD SP, 0016

SUB S\_INDEX, ONE

SUB SRAM, ONE

LOAD TEMP, ZEROS

OUTPUT TEMP, ZEROS

RETEN

; ================================================

; Receive Carriage Return

; ================================================

CAR\_RET\_I LOAD DISPLAY, 0006

LOAD SP, 0014

JUMP RESETSTRING

; ================================================

; Start Dumping Memory

; ================================================

DUMP\_MEM LOAD DISPLAY, 0007

LOAD SP, 8000

JUMP RESETSTRING

; ================================================

; Reset string index and output nothing

; ================================================

RESETSTRING ADD COUNT, S\_INDEX

OUTPUT RX, (SRAM) ; Received Byte to SRAM

ADD SRAM, ONE

LOAD TEMP, ZEROS

OUTPUT TEMP, ZEROS

LOAD S\_INDEX, ZEROS

RETEN

; ================================================

; ISR vectored through 0FFE

; ================================================

ADDRESS 0FFE

ENDIT JUMP ISR

END

INTENTIONALLY BLANK