

Chip Specification for CECS 460 SoPC

**Table of Contents**

**1.Introduction4**

1.1 Purpose4

**2.Applicable Documents5**

2.1 References5

2.1.1 MicroSemi5

2.1.2 RS-2325

2.1.3 PicoBlaze5

2.1.4 Artix-7 Library5

**3.Requirements6**

3.1 Performance Requirements6

3.2 Interface Requirements6

3.3 Physical Requirements6

3.4 Power Requirements6

**4.Implementation7**

4.1 External Documents7

4.2 Block Diagram8

4.3 Data Flow Description8

4.4 Embedded Memories9

4.5 Input / Output Interface Description10

4.6 Reset11

4.6.1 Reset Type11

**5.Externally Acquired Blocks12**

5.1 TramelBlaze12

5.1.1 Description12

5.1.2 Performance Requirements12

5.1.3 Block Diagram13

5.1.4 I / O Definition14

5.1.5 Verification Plan14

5.2 TSI Blocks15

5.2.1 Description15

5.2.2 Performance Requirements15

5.2.3 Block Diagram16

**6.Internally Developed Blocks (Digital)17**

6.1 UART17

6.1.1 Description17

6.1.2 Performance Requirements17

6.1.3 Block Diagram18

6.1.4 I / O Definition19

6.1.5 Verification Plan20

**7.Chip Verification Plan21**

7.1 Simulation Test Environment21

7.1.1 Testbench Description21

7.2 Formal Verification21

**A. Glossary / Definitions22**

**B. Revision History23**

**C. Appendix**

C.1 Verilog Source Code

C.2 Assembly Source Code

**1. Introduction**

This Chip Specification is for the MD UART IP, designed in CECS 460 lead by Professor Tramel. Using this SoPC Chip Specification will allow one to transmit and receive data through a Nexys 4 board.

**1.1 Purpose**

The Purpose of this document is to specify the requirements of our Transmit Engine, and the issues related to its development

**2. Applicable Documents**

**2.1 References**

**2.1.1 MicroSemi**

This Chip Specification references the CoreUART Chip Specification by MicroSemi found at:

<http://soc.microsemi.com/ipdocs/CoreUART_HB.pdf>

**2.1.2 RS-232**

This design follows RS-232 qualifications for full-duplex communication.

ARC electronics explains this common standard in detail here:

<https://arcelect.com/rs232.htm>

**2.1.3 PicoBlaze**

This design utilizes a microcontroller that emulates Xilinx’s PicoBlaze.

**2.1.4 Artix-7 Library**

This design will use TSI blocks which are referenced by this library that can be found on Xilinx’s website.

**3. Requirements**

**3.1 Performance Requirements**

This design must be able to transmit data serially through the Nexys 4 board. It must be able to output CSULB CECS 460 <Line Number> continuously at the baud rate specified.

**3.2 Interface Requirements**

For this design, we will be able to control the baud rate, data size, and parity control.

**3.3 Physical Requirements**

This design is made for the Nexys 4 FPGA by Digilent. In order to test the data being transmitted, we need to have Realterm installed. The download for this software is available for free online.

**3.4 Power Requirements**

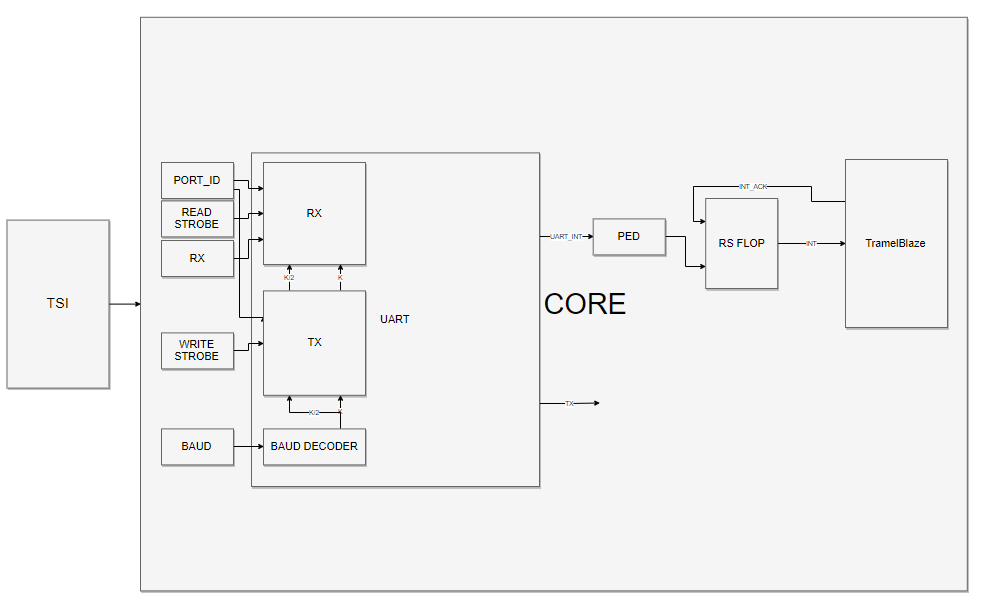
The only power requirements for this design are the requirements to power the Nexys 4 FPGA.

**4. Implementation**

**4.1 Design Description**

With this design, we will be able to transmit and receive ASCII codes serially through our Nexys 4 FPGA at a frequency specified by the baud rate. This data will be able to handle parity as well to ensure data is transferred correctly.

**4.2 Block Diagram**



**4.3 Data Flow Description**

Initially, we must write assembly code for the Tramelblaze. Depending on which ASCII codes one wishes to write, it must be loaded sequentially into the scratch ram using assembly. When the Tramelblaze interrupt is enabled, it checks to see if we are receiving data or transferring data. If we wish to transmit data, a counter in our assembly will be loaded with the first value where the first ASCII code is stored, then step through until the word is finished. If we wish to receive data, we will take in the input and compare it to decisive values. From there, the counter will be loaded with specific values then output. Currently, the ACSII code for “\*” will output the users hometown. The ACSII code for “@” will output the number of inputs the user has entered. In order to display it correctly, we must have a decimal to hex converter in our assembly to output each decimal place of the number.

**4.4 Embedded Memories**

In this design, the only memory modules we have are from the externally acquired Tramelblaze. The Tramelblaze consists of a 4Kx16 PROM, 512X16 Scratch Ram, and 128x16 Stack Ram.

**4.5 Input / Output Interface Description**

For this design, we have control of the baud rate, data size, and parity control. Our design is capable of inputting or outputting data serially.

INPUTS:

BAUD: Baud rate is the rate at which our data is transmitted or received. The translation from this 4-bit input value to frequency can be with at this table:

|  |  |
| --- | --- |
| BAUD [3:0] | RATE |
| 0000 | 300 |
| 0001 | 1200 |
| 0010 | 2400 |
| 0011 | 4800 |
| 0100 | 9600 |
| 0101 | 19200 |
| 0110 | 38400 |
| 0111 | 57600 |
| 1000 | 115200 |
| 1001 | 230400 |
| 1010 | 460800 |
| 1011 | 921600 |

EIGHT: This 1-bit input will determine whether our input data is 8 bits or 7 bits.

PEN: This 1-bit input is our parity enable.

OHEL: (Up High, Even Low) This 1-bit input will only be effective when parity enable is high, and will determine if the parity is odd or even.

RX: 1-bit serial data input

OUTPUTS:

TX: 1-bit serial data output.

**4.6 Reset**

For this design, we will be implementing a synchronizing reset module. Our AISO (asynchronous in, synchronous out) will make sure our whole design is reset synchronously. (Module can be seen in appendix)

**4.6.1 Reset Type**

We will be using a high-active reset with this design.

**5. Externally Acquired Blocks**

**5.1 TramelBlaze**

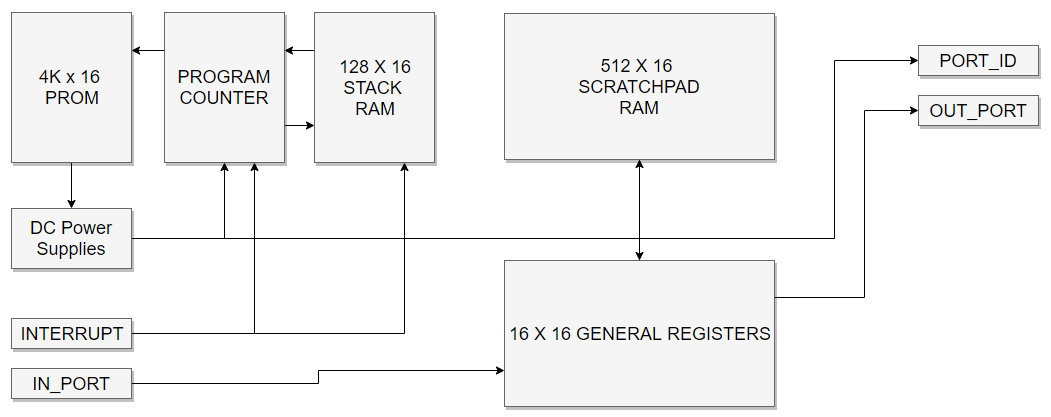
**5.1.1 Description**

The Tramelblaze is a 16-bit processor designed to emulate the 8-bit Picoblaze by Xilinx.

**5.1.2 Performance Requirements**

The Tramelblaze must have assembly written to it and assembled by the Trambler. The Trambler is the assembler for the Tramelblaze ISA and is written in Python. This will generate the memory module required for building a Xilinx Single-Port ROM.

**5.1.3 Block Diagram**



**5.1.4 I / O Definition**

INPUTS:

IN\_PORT – This 16-bit input is for incoming data that is to be written to memory in the Tramelblaze.

INTERRUPT – This 1-bit input will trigger the interrupt and jump to the ISR which is at Address 0xFFE

OUTPUTS:

OUT\_PORT – This 16-bit output is for outgoing data of the Tramelblaze.

PORT\_ID – This 16-bit output is the address at which the data is being output.

READ\_STROBE – This 1-bit output will notify if the Tramelblaze is reading out data.

WRITE\_STROBE – this 1-bit output will notify if the Tramelblaze is writing data to its memory.

INTERRUPT\_ACK – This 1-bit output is our interrupt acknowledge and will turn off our interrupt flag.

**5.1.5 Verification Plan**

In order to verify that the Tramelblaze is working and that our assembly code is correct, we must trigger the interrupt and allow time for the Tramelblaze to work through its memory.

(Full Testbench in Appendix)

**5.2 TSI Blocks**

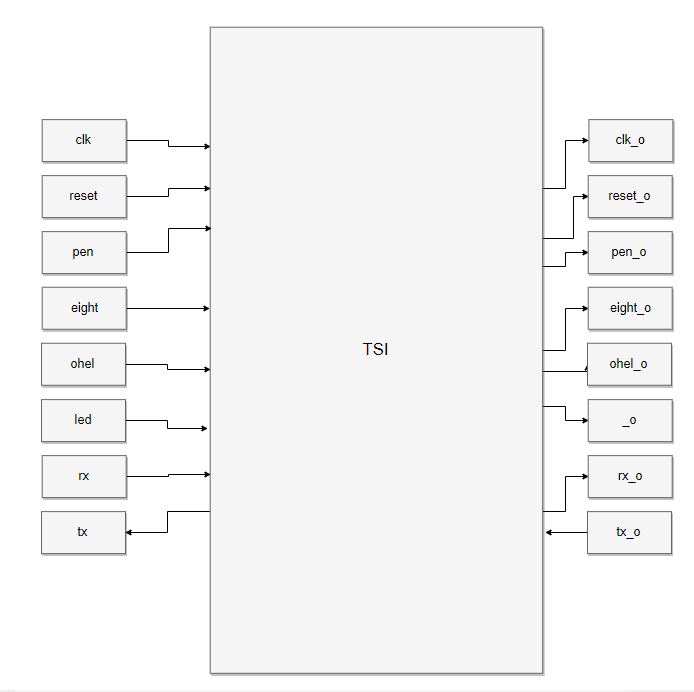
**5.2.1 Description**

The TSI block is used to pass all references from the outside world into our core block.

**5.2.2 Performance Requirements**

Our TSI block must contain all references to the target technology library.

**5.1.3 Block Diagram**



**6. Internally Developed Blocks (Digital)**

**6.1 UART**

**6.1.1 Description**

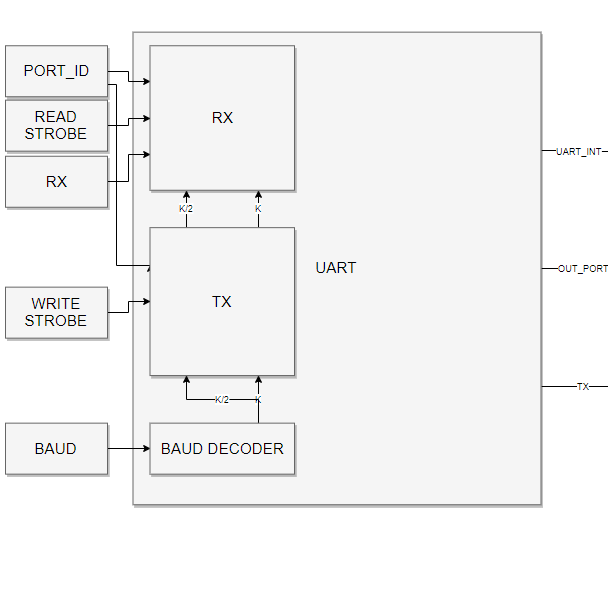
This Universal Asynchronous Receiver Transmitter, consists of the transmit engine and receive engine. With the harmonious working of these two modules we will be able to communicate with an external terminal

**6.1.2 Performance Requirements**

This transmit engine is required to transmit data serially at the specified baud rate. The engine must also decide whether or not parity is enabled and set the parity bit according to the inputs (listed in 6.1.4).

The receive engine is required to intake data serially and output the word when it is fully received. In this design we have error checking bits to check overflow, parity, and framing.

**6.1.3 Block Diagram**



**6.1.4 I / O Definition**

For this design, we have control of the baud rate, data size, and parity control. Our design will output serial data (a single bit) once every clock.

INPUTS:

BAUD: Baud rate is the rate at which our data is transmitted. The translation from this 4-bit input value to frequency can be with at this table:

|  |  |
| --- | --- |
| BAUD [3:0] | RATE |
| 0000 | 300 |
| 0001 | 1200 |
| 0010 | 2400 |
| 0011 | 4800 |
| 0100 | 9600 |
| 0101 | 19200 |
| 0110 | 38400 |
| 0111 | 57600 |
| 1000 | 115200 |
| 1001 | 230400 |
| 1010 | 460800 |
| 1011 | 921600 |

EIGHT: This 1-bit input will determine whether our input data is 8 bits or 7 bits.

PEN: This 1-bit input is our parity enable.

OHEL: (Up High, Even Low) This 1-bit input will only be effective when parity enable is high, and will determine if the parity is odd or even.

LOAD: This 1-bit input enables the engine to load new data to transmit

OUT\_PORT: This 8-bit input coming from our Tramelblaze will send the incoming load data.

RX: 1-bit serial data input.

R\_STATUS - Flag to output status register.

R\_DATA - Flag to output data.

OUTPUTS:

TX: 1-bit serial data output.

UART\_INT - UART interrupt flag

UART\_DS - UART outgoing receive data or status register

**6.1.5 Verification Plan**

In order to verify if our UART is working properly, we will feed the UART data through RX and see if it collects it correctly. Next we will give the engine data to transmit and see if the waveforms output correctly.

(Full Testbench in Appendix)

**7. Chip Verification Plan**

**7.1 Simulation Test Environment**

In order to test the design, we will need to feed the design with the correct stimulus and calculate the correct outputs.

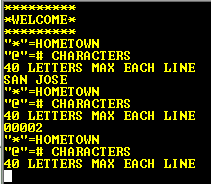
**7.1.1 Testbench Description**

Since there are not many outputs for the design, this testbench will be fairly simple. We plug in a baud rate, data size, and the parity bits. The design will do the rest by itself while we check the outputs.

(Full Testbench in Appendix)

**7.2 Formal Verification**

To test the final design, we will run the board and our Realterm application will receive the data being transmitted. If the results are correct, Realterm should show similar results to these (depending on how your assembly is set-up):

**A. Glossary / Definitions**

SoPC – System on Programmable Chip

UART – Universal Asynchronous Receiver Transmitter

TSI – Technology Specific Instantiation

**B. Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Description | Initials | Date |
| - | Initial Document Creation (Draft Version)  1 - Created template and added Table of Contents | MD | 3/14/18 |
| 1.0 | 1 - Added all descriptions and figures | MD | 3/16/18 |
| 2.0 | 1 – Added receive engine descriptions and figures.  2 – Added TSI description and figures. | MD | 5/8/18 |