

UART Tile

High-level Arch Spec

Rev 1.0

Chris Shakkour, chrisshakkour@gmail.com

Shahar Dror, shdrth1@gmail.com

Supervisor:

Amichai Ben-David, amichaibd@gmail.com

GitHub:

<https://github.com/amichai-bd/riscv-multi-core-lotr>

Revision History:

<i>Author</i>	<i>Date</i>	<i>Rev</i>	<i>Description</i>
<i>Chris Shakkour</i>	<i>18-Nov-22</i>	<i>0.1</i>	<i>Adds initial seed and content structure</i>
<i>Chris Shakkour</i>	<i>19-Nov-22</i>	<i>0.2</i>	<i>Adds wishbone interface and UART config sequence</i>
<i>Chris Shakkour</i>	<i>21-Nov-22</i>	<i>0.3</i>	<i>Adds read/write transfer sequences and references</i>
<i>Chris Shakkour</i>	<i>30-Nov-22</i>	<i>0.4</i>	<i>Adds introduction and block level diagrams</i>
<i>Chris Shakkour</i>	<i>04-Dec-22</i>	<i>0.5</i>	<i>Adds FPGA integration section</i>
<i>Chris Shakkour</i>	<i>06-Dec-22</i>	<i>0.6</i>	<i>Adds UART and USB protocol</i>
<i>Chris Shakkour</i>	<i>09-Dec-22</i>	<i>0.7</i>	<i>Adds RC interface and Ring Controller sections</i>
<i>Shahar Dror</i>	<i>22-Dec-22</i>	<i>0.8</i>	<i>Adds Transfer Handler uArch design spec</i>
<i>Shahar Dror</i>	<i>06-Jan-23</i>	<i>0.9</i>	<i>Adds Transfer Handler FSM tables</i>
<i>Chris Shakkour</i>	<i>10-Jan-23</i>	<i>1.0</i>	<i>Adds final touches</i>

Contents

TERMEONOLOGY.....	3
ACRONYMS.....	3
REQUIRMENTS.....	3
INTRODUCTION	4
SYSTEM LEVEL DIAGRAMS.....	5
Host to device connection	5
Communication components.....	5
SW-HW interface	6
SOFTWARE INTERFACE	7
Py-Terminal.....	7
Host to Device Transfers	8
Write transfer	8
Read transfer	8
Write burst transfer	9
Read burst transfer	9
USB serial transfer handler	9
HARDWARE INTERFACE.....	10
Block-level diagram.....	10
Low-level communication diagram	10
Communication interfaces.....	11
Wishbone interface.....	11
RC interface.....	13
Communication protocols.....	14
UART protocol.....	14
USB protocol	15
UART wrapper (IP)	16
UART config	17
Gateway.....	18
Transfer handler engine.....	18
Ring Controller (IP).....	23
VERIFICATION & VALIDATION	24
Block-level RTL simulations.....	24
System-level RTL simulations.....	24
Functional validation.....	25
FPGA INTEGRATION	26

FPGA AREA REPORTS	28
REFERENCES	28

TERMINOLOGY

The following terminology is being used interchangeably during this document along with many useful acronyms.

- The term “Host” refers to any user computer (laptop, desktop, SBC, Etc.)
- The term “Device” refers to the device receiving transfers from the Host. the FPGA target in this project.

ACRONYMS

Acronym	Definition
UART	<i>Universal asynchronous receiver-transmitter</i>
LOTR	<i>Lord of the ring</i>
FPGA	<i>Field programmable gate array</i>
RC	<i>Ring controller</i>
IP	<i>Intellectual property</i>
ASCII	<i>American standard code for information interchange</i>
USB	<i>Universal serial bus</i>
TTL	<i>Transistor-transistor logic</i>
HW	<i>Hardware</i>
SW	<i>Software</i>
PC	<i>Personal computer</i>

Table 1: acronyms

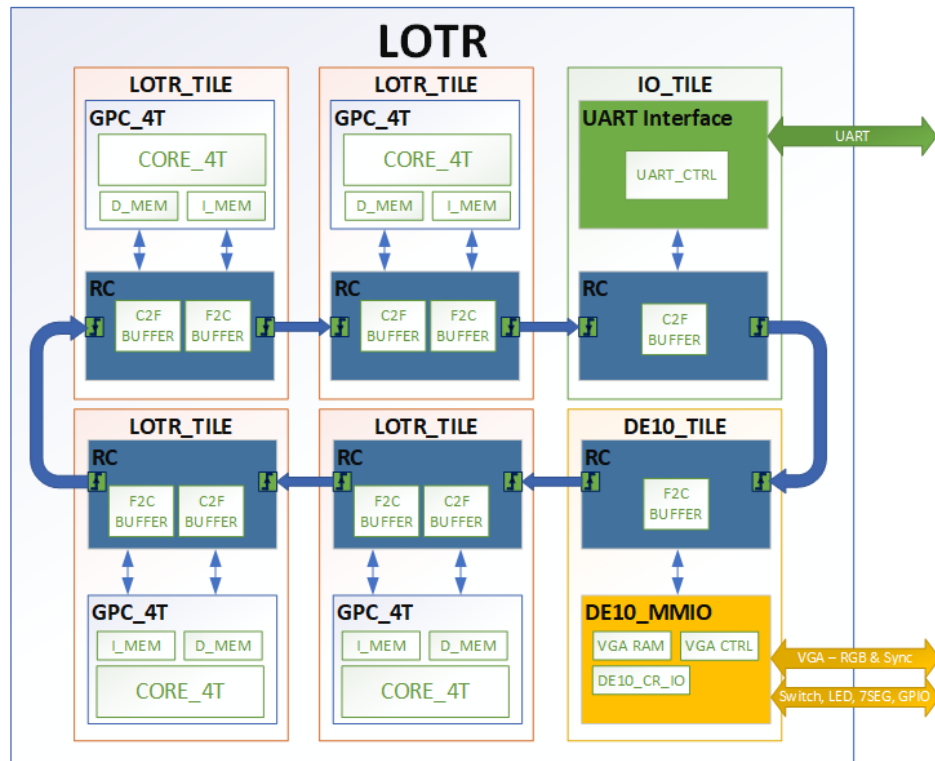
REQUIREMENTS

To run this project successfully you will need a few SW and HW tools, for SW you will need a PC with python installed on the PowerShell to run the Py-terminal, make sure your PC has a USB 2.0+ ports.

1. Host Device (PC) with USB ports.
2. Python3 and higher installed.
3. PySerial library installed (see [references](#) section).
4. USB to TTL serial converter (see [references](#) section).
5. DE10-Lite FPGA dev kit (see [references](#) section).

INTRODUCTION

The UART tile project is a part of a bigger project called the LOTR. The LOTR project is a multi-core design project featuring a ring-like architecture to connect all the cores/tiles in a ring, this LOTR project runs on a FPGA target the DE10-NANO development kit, the UART tile is here to provide communication from host devices to the LOTR target to be able to transfer data between a host and the device(FPGA) more precisely to be able to have access to the physical memory address space from an external Host unit.



SYSTEM LEVEL DIAGRAMS

This section will present a few system-level diagrams to give the reader more visual on the project and its different components in high-level.

Host to device connection

The following diagram shows the high-level connection diagram between the Host(pc) and Device (LOTR target on the FPGA)

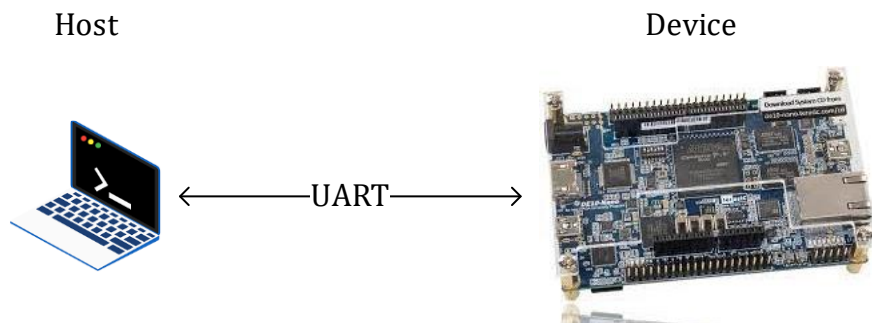


Figure 1: host to device connection

Communication components

The following diagram dives deeper to the actual components that define the communication between the Host and Device. the Host (PC) communicates with a USB to UART converter then the UART transfers are handled by the UART tile to communicate the transfers to the LOTR target. The HW interface of the project serves as the blue part in this diagram and the SW interface serves as the green part.

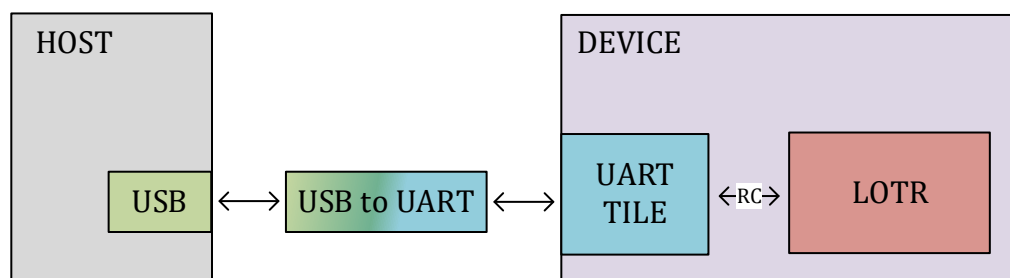


Figure 2: host to device Datapath

SW-HW interface

The following diagram shows the different abstraction levels within each interface, both interfaces SW, and HW have multiple abstraction levels as seen below.

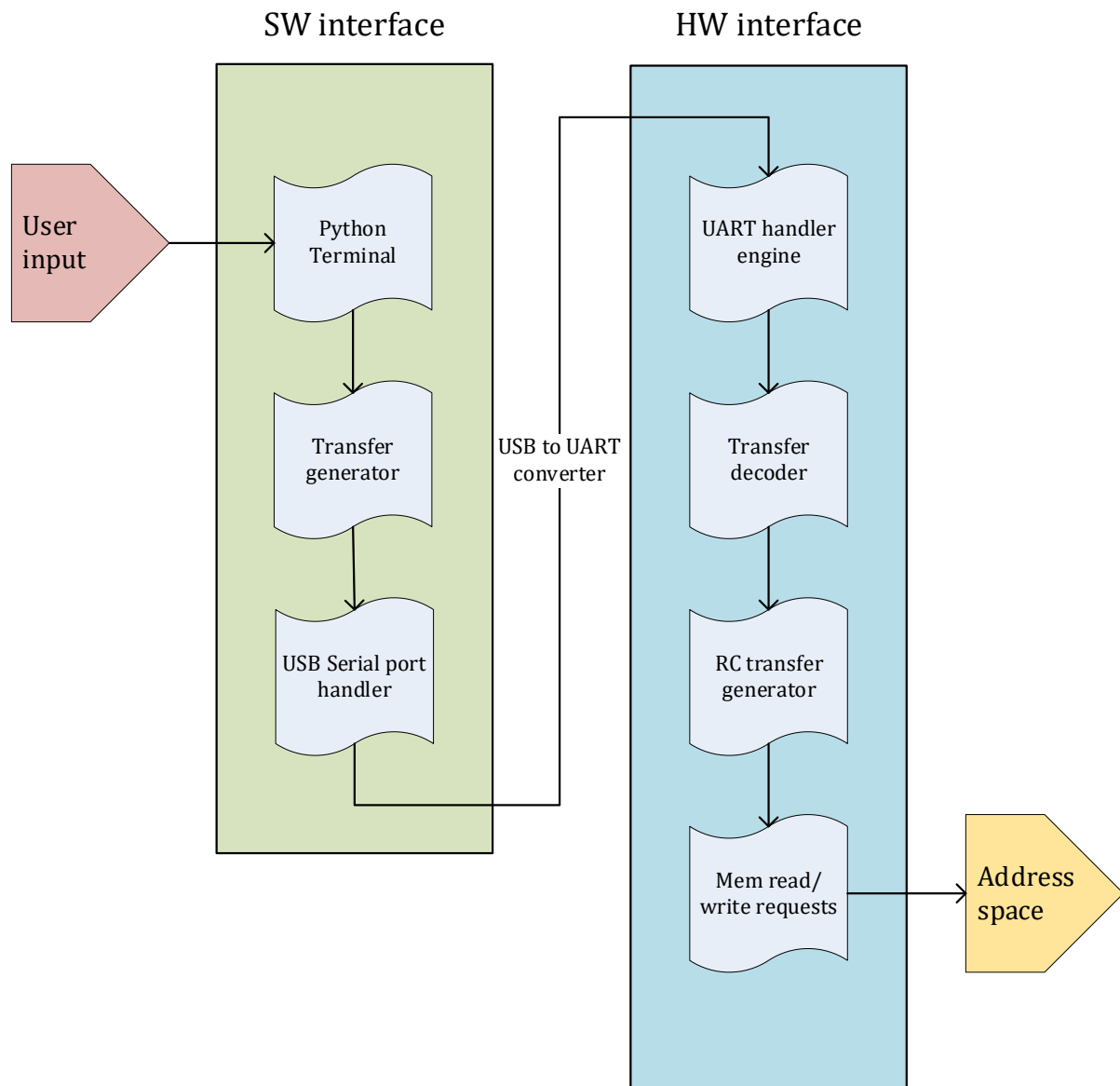


Figure 3: HW-SW interface block diagram

SOFTWARE INTERFACE

The software interface is just a piece of code running on a Host device. The interface on one side interacts with a user by a friendly Py-Terminal and communicates with the Device via USB serial transfers. Under the hood the SW interface does two main functions, 1. Decoding user input, 2. Generating “Host to Device” transfers. Following detailed explanation of the 3 parts of the SW interface, 1. Py-Terminal, 2. Host to Device transfer generator, and 3. USB serial transfers handler.

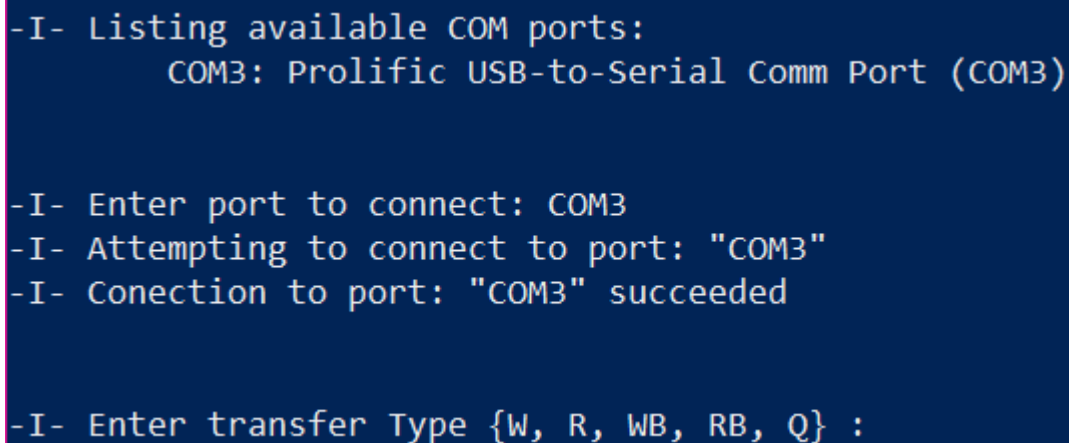
Py-Terminal

The Py-Terminal is a python program that mimics an interactive terminal with a simple UI. The terminal prints instructions to the user and receives inputs like shown below. these inputs are checked for validity first then sent to the transfer generator to form a correct transfer before transferred to the USB serial port. more details on the transfers structure are documented down below this section.

You might need to download these packages before running the script:

```
pip install serial
pip install pySerial
```

To run the terminal run the python script in windows PowerShell>> python3 uart_term.py



```
-I- Listing available COM ports:
      COM3: Prolific USB-to-Serial Comm Port (COM3)

-I- Enter port to connect: COM3
-I- Attempting to connect to port: "COM3"
-I- Conection to port: "COM3" succeeded

-I- Enter transfer Type {W, R, WB, RB, Q} :
```

Figure 4: Terminal example figure

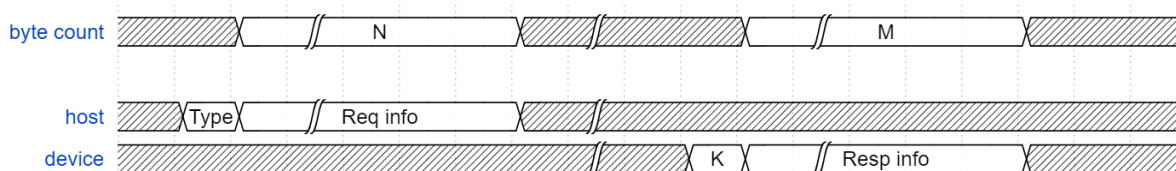
When the terminal is run it checks and displays any available COM ports connected to the PC to make sure the UART to TTL converter is plugged to the PC before running the script, then the user must plug the desired comport to connect to. After this is done, the terminal will switch to transfer mode to receive transfers from the user.

The user can Write a single word from the Host to the Device using the “W” command, read a word from the Device to the Host using the “R” command, write multiple words from a file on the Host to the Device using the write burst “WB” command, and read multiple words from the Device into a file on the Host using the read burst “RB” command. Following command requirement table. Next section will explain the structure of each transfer.

Host to Device Transfers

All transfers are initiated by the Host only, the transfers are non-posted, the Host starts the transfer and waits for a response from the Device or expire upon timeout. the granularity of data transfers on the UART line is one Byte per transfer meaning to send a 32-bit word the Host must send 4 bytes.

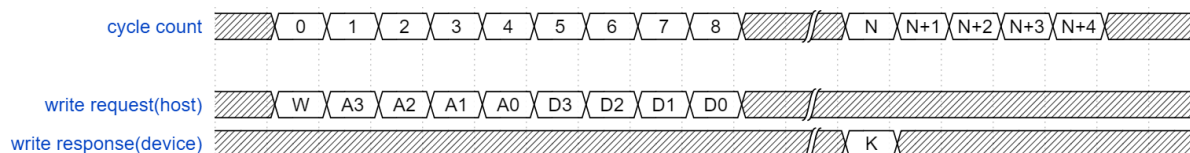
Each transfer starts with one byte indicating the transfer type {R, W, WB, RB} followed by a sequence of bytes 'N' with required information to complete the transfer like transfer address and data from Host to Device. For each transfer a series of bytes is responded from the Device starting with 'K' followed by "M" bytes of response information.



Waveform 1: host-device transfers generic structure

Write transfer

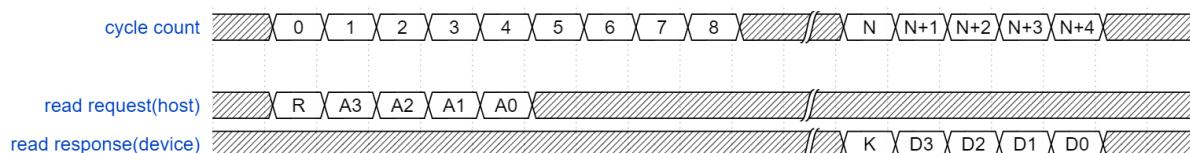
The write transfer is initiated by the Host with a starting byte 'W' in ascii 8'd87 in Hex followed by 4 bytes representing the transfer Address sent in Big-Endian format (A3, A2, A1, A0), followed by 4 bytes of transfer data also Big-Endian format (D3, D2, D1, D0). Once the transfer is completed the Device responds with a byte of "K" in ascii indicating transfer completion.



Waveform 2: write transfer structure

Read transfer

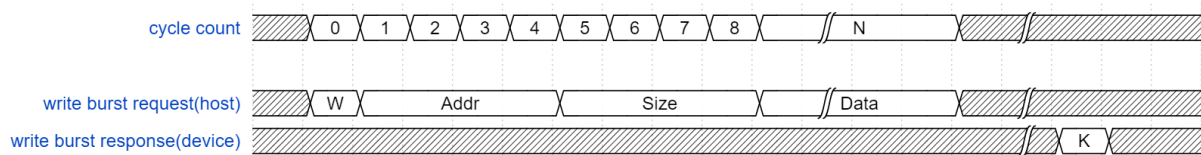
The read transfer is initiated by the Host with a starting byte 'R' in ascii 8'd82 in Hex followed by 4 bytes representing the transfer Address sent in Big-Endian format (A3, A2, A1, A0). Once the transfer is completed the Device responds with a byte of "K" in ascii indicating transfer completion followed by 4 bytes of response data also Big-Endian format (D3, D2, D1, D0).



Waveform 3: read transfer structure

Write burst transfer

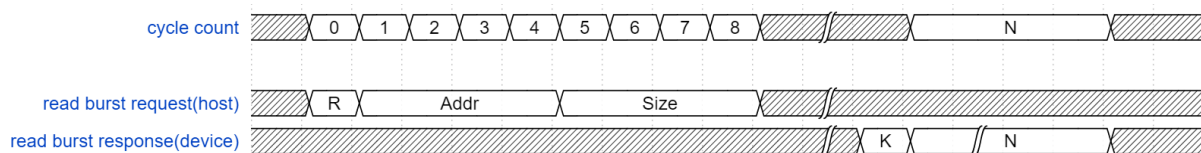
The write burst transfer is initiated by the Host with a starting byte 'W' in ascii 8'd87 in Hex followed by 4 bytes representing the transfer starting Address sent in Big-Endian format (A3, A2, A1, A0), followed by 4 bytes of transfer Size also Big-Endian format (S3, S2, S1, S0), followed with all bytes of data where the number of bytes equates to 4*SIZE. Once the transfer is completed the Device responds with a byte of "K" in ascii indicating transfer completion.



Waveform 4: write burst transfer structure

Read burst transfer

The read transfer is initiated by the Host with a starting byte 'R' in ascii 8'd82 in Hex followed by 4 bytes representing the transfer Address sent in Big-Endian format (A3, A2, A1, A0), followed by 4 bytes of transfer SIZE also Big-Endian format (S3, S2, S1, S0). Once the transfer is completed the Device responds with a byte of "K" in ascii indicating transfer completion followed by 4*SIZE bytes of response data also Big-Endian format (D3, D2, D1, D0).



Waveform 5: read burst transfer structure

USB serial transfer handler

The PySerial library in python was used to send USB transfers from the Host (PC) though one of the USB ports that has the USB to TTL converter connected following short example on how to use it.

```
# step-1 import the library
import serial

# step-2 open port connection. Port COM7 was used here for example.
port = serial.Serial("COM7")

# step-3 configure the UART communication protocol.
port.baudrate = 9600 # set Baud rate to 9600 from available list [9600,
19200, 38400, 57600, 115200]
port.bytesize = 8 # Number of data bits = 8
port.parity = 'N' # No parity
port.stopbits = 1 # Number of Stop bits = 1
port.timeout = 5 # Read timeout is 5 seconds

port.write("hello")
```

HARDWARE INTERFACE

This section will explain the components related to the HW interface, introduction to the interfaces used in the low-level implementation, as well as protocols and HW state machines.

Block-level diagram

The UART tile connects to the Ring using the RC block, the RC block communicates with the Gateway module that acts as a conversion bridge between the UART transfers and the RC transfers. Detailed explanation of each element is presented below.

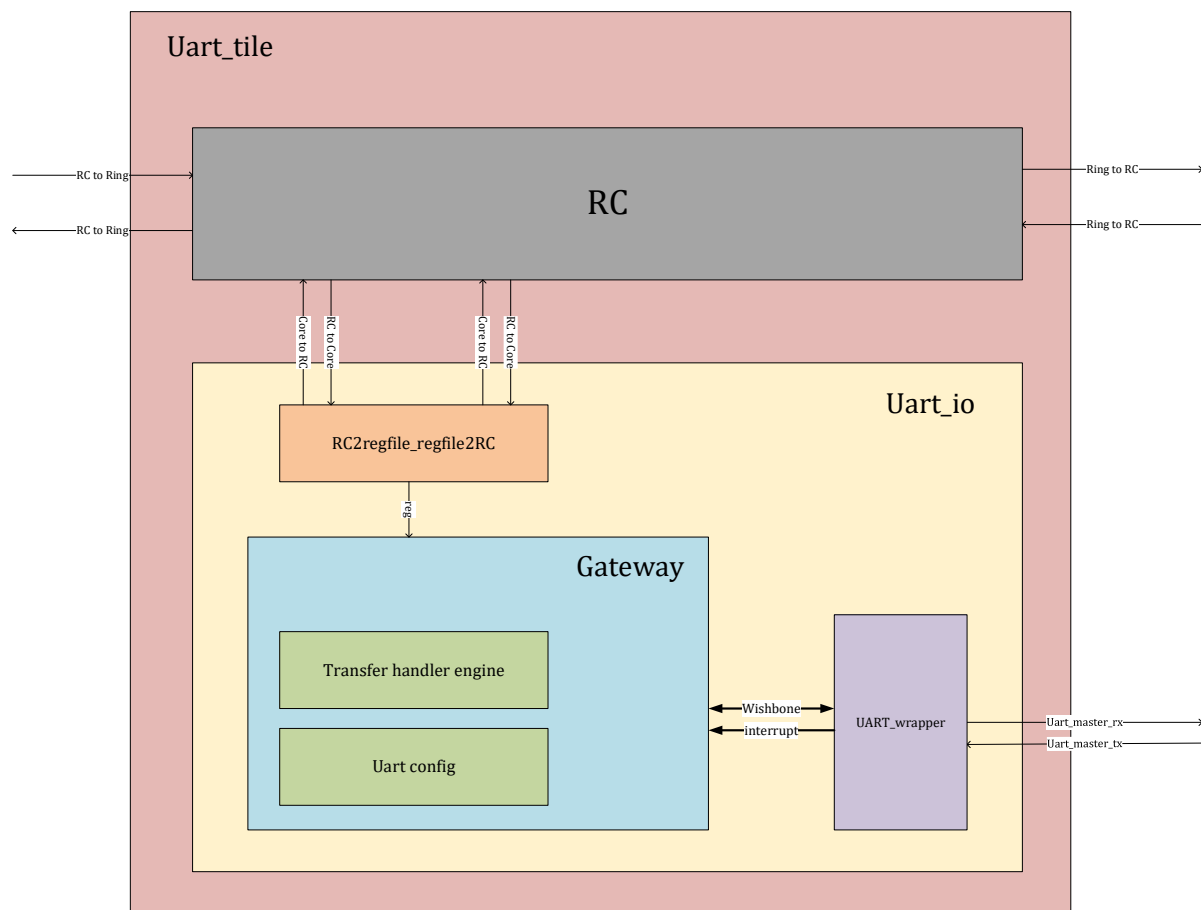


Figure 5: UART tile block-level diagram

Low-level communication diagram

Following drawing shows the full communication flow between the Host and the Device, the communication flow is symmetric for both transfers from the Host to Device and vice versa. The transfers are sent in asynchronous serial communication protocols from the host via the USB line and the UART line, then translated to reg access transfers passing through the wishbone, and RC interfaces and all the way back to the HOST in the opposite direction.

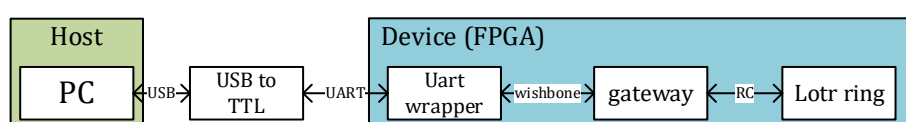


Figure 6: communication flow block diagram

Communication interfaces

In this section you'll find a short introduction to the wishbone and RC interfaces used in this project before we dive deeper to the actual logic. further information can be found in the [references](#) section below.

Wishbone interface

The WISHBONE interface is a standard open-source interface for SOC interconnect communication. In this project the wishbone interface is used for communicating with the UART IP block since this block only supports the wishbone interface. Following details and basic transfers, you can find the complete spec in the [references](#) section below.

Interface signals

Port	Width	Direction	Description
CLK	1	Input	Block's clock input
WB_RST_I	1	Input	Asynchronous Reset
WB_ADDR_I	5 or 3	Input	Used for register selection
WB_SEL_I	4	Input	Select signal
WB_DAT_I	32 or 8	Input	Data input
WB_DAT_O	32 or 8	Output	Data output
WB_WE_I	1	Input	Write or read cycle selection
WB_STB_I	1	Input	Specifies transfer cycle
WB_CYC_I	1	Input	A bus cycle is in progress
WB_ACK_O	1	Output	Acknowledge of a transfer

Table 2: wishbone interface signal information

Master-Slave interconnect

Follow the connection scheme below for a successful integration, this shows a direct slave to master connection, when working with multiple slaves/masters follow the guidelines in the complete spec.

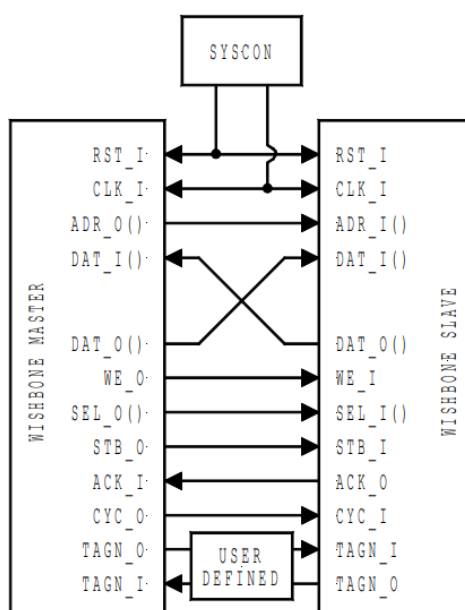


Figure 7: wishbone interface connection diagram

Read/Write transfers

To initiate a successful read/write request the STB, CYC, and SEL must be asserted with a valid address on the ADDR signal. In addition, for a write request the WE (write enable) signal must be asserted along with the data that needs to be written via the DATA_O signal, else when WE is deserted a read transfer is requested instead.

once a valid transfer is requested an acknowledge response is expected on the ACK signal, on read transfers only a data is also expected on the DATA_I signal.

The signals in the waveform below belong to the master. The signals located in the top part of the waveform are actual signals on the wishbone interface, denoted in uppercase. and the signals on the bottom are just indication/helper signals to understand what's happening they are not a part of the interface.

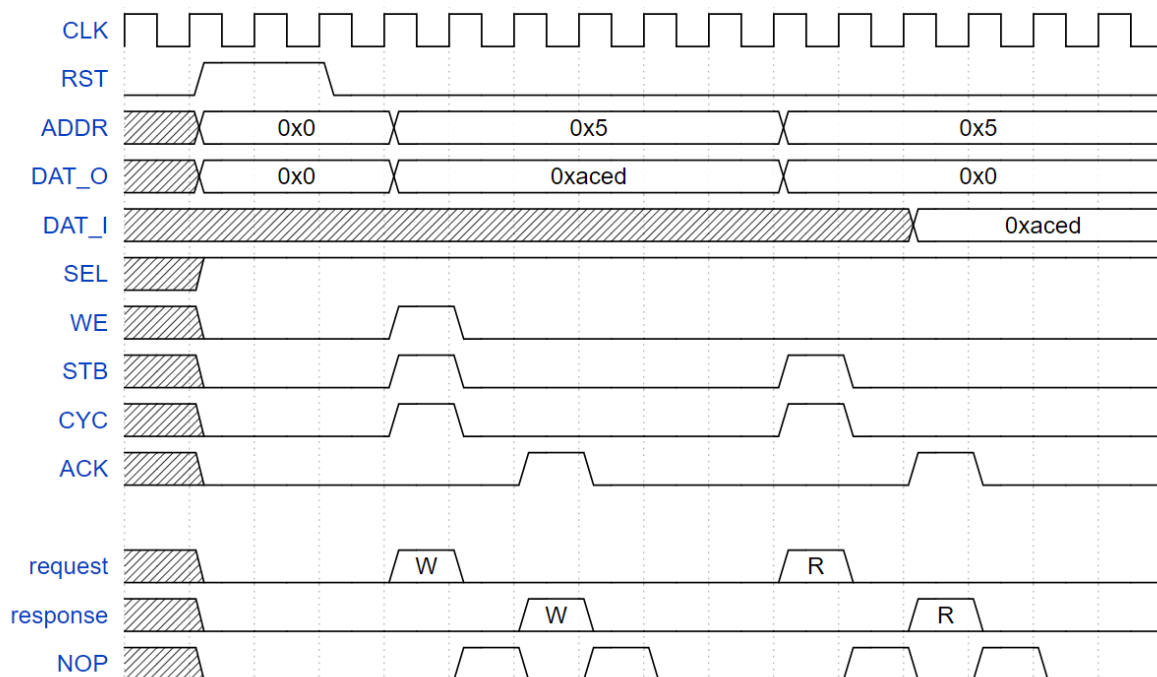


Figure 8: wishbone interface waveform

Constraints

- One transfer at a time, a transfer can't be requested unless the previous transfer is completed with an acknowledge signal see NOP's above.
- One must wait one full clock cycle after the acknowledge is asserted before requesting any further transfers see NOP's above.

RC interface

The RC interface is made up of 2 identical channels one for the requestor and one for the responder. each channel consists of 5 signals each plays a role in the communication interface. Following high-level integration guides, see [references](#) below for more information.

Agent to Agent interconnect:

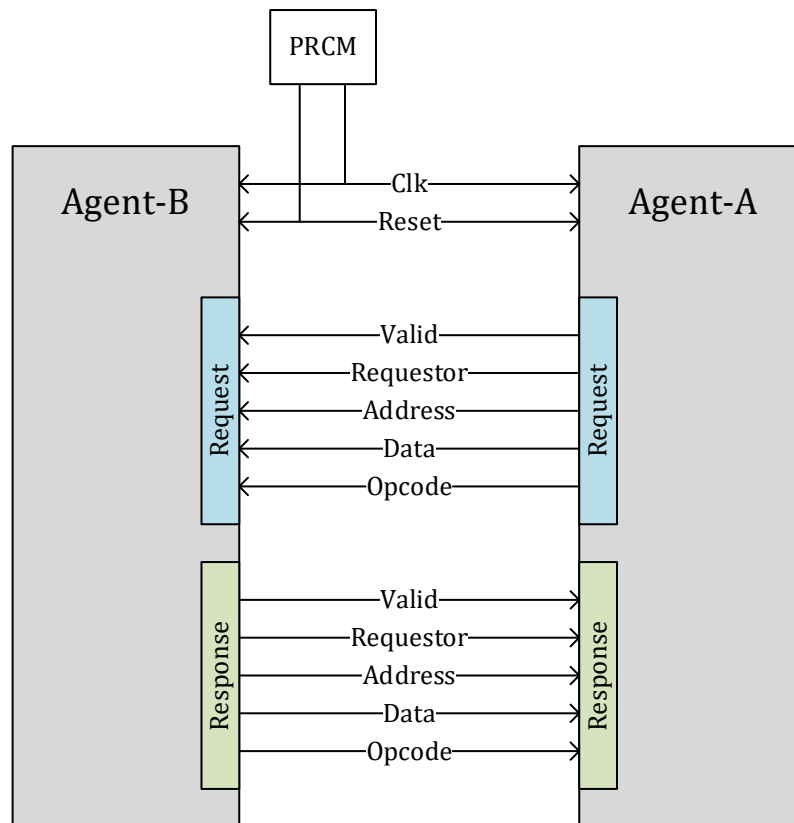


Figure 9: RC interface interconnect

Request channel Signal list:

signals	Type	Width [bits]	Description
Valid	Output	1	Indicate valid transaction
Requestor	Output	10	Requestor Identification
Address	Output	32	Target transfer address
Data	Output	32	Target transfer Data
Opcode	Output	2	Type of transaction {RD, RD_RSP, WR, WR_BCAST}

Table 3: request channel signal list

Response channel signal list:

Signals	Type	Width [bits]	Description
Valid	Input	1	Indicate valid transaction
Requestor	Input	10	Requestor Identification
Address	Input	32	Target transfer address
Data	Input	32	Target transfer Data
Opcode	Input	2	Type of transaction {RD, RD_RSP, WR, WR_BCAST}

Table 4: response channel signal list

Communication protocols

In this section you'll find a short introduction to the UART, and USB protocols used in this project before we dive deeper to the actual logic. further information can be found in the [references](#) section below.

UART protocol

UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end.

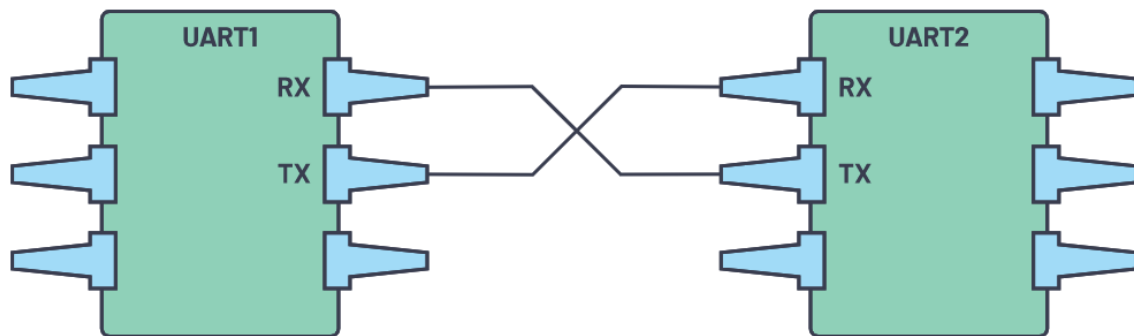


Figure 10: UART device to device connection

Each transmission is characterized with 4 elements, start bit, data frame, parity bit, and stop bits. The start bit is just constant value usually low (0) to indicate a transmission start, the data bits are followed one after the other between 5 to 9 bits, then an optional parity bit is concatenated followed by one or two stop bits usually high (1) to indicate transmission end.

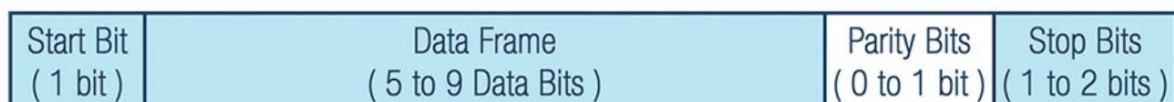


Figure 11: UART Packet

Following example with 8 data bits (d [7:0]), parity (p) and one stop bit (high), the packet is defined by 11 serial bits, one start and stop bits, one party, and 8 data bits.

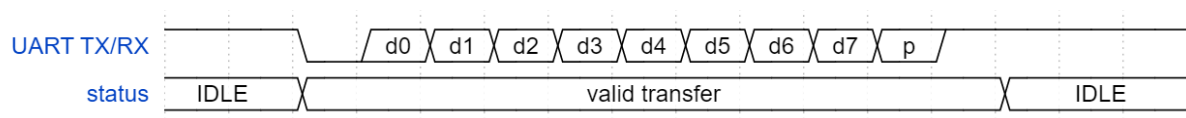


Figure 12: UART packet example waveform

USB protocol

The USB protocol is highly used in device-to-device communication, the protocol is characterized with many advanced features, here you will find high-level information, see [references](#) below.

The USB connector consists of 4 wires, positive (VBUS), negative (GROUND), and an inverted data pair (D-, D+) as shown below, all transactions are initiated by the master device only, the slave device can only respond.

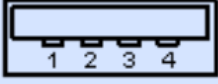
 Type A USB Connector	Pin Number	Cable Colour	Function
	1	Red	V _{BUS} (5 volts)
	2	White	D-
	3	Green	D+
	4	Black	Ground

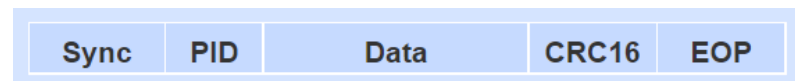
Figure 13

The protocol is characterized by 4 types of packets, each packet has its own set of fields, following packet structures and short explanation of each field.

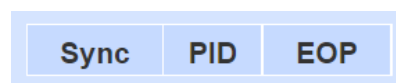
Token packets:



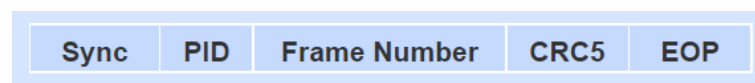
Data packets:



Handshake packets



Start of frame packets



Sync: All packets must start with a sync field. The sync field is 8 or 32 bits long.

PID: stands for Packet ID, this field is used to identify the type of packet that is being sent.

ADDR: The address field specifies which device the packet is designated for.

DATA: The data field specifies the transmitted data.

ENDP: The endpoint field is made up of 4 bits, allowing 16 possible endpoints.

CRC: Cyclic redundancy checks are performed in the data within the packet payload, all token packets have a 5-bit CRC while data packets have 16-bit CRC.

EOP: End of packet. Signaled by a single ended zero.

UART wrapper (IP)

The UART Wrapper unit holds a UART IP engine that can send and receive UART transfers. On one side this block communicates with any other UART engine via UART transfers and on the other side in a custom interface called the “wishbone”.

To make this block functional when out of reset a sequence of reads and writes via the wishbone interface must be done to configure this block, see wishbone and UART config sections down below.

Once configured the receiving/sending of data can be done via the wishbone interface this is how the gateway communicates with the UART wrapper.

UART config

This block is responsible for configuring the UART IP via a series of wishbone reads and writes to set registers with certain values like working baud rate, word length, and other settings, this section will go over the whole configuration flow. A hardware FSM machine is used to configure the UART IP, however one can chose to configure this IP with a piece of SW code if you have a CPU with access to the wishbone interface of this block.

The UART IP has many configuration registers, here you will find the list of relevant and basic configurations that will get the UART IP block up and running.

Following available configuration register list below these registers will be used in the configuration flow, further info regarding the functionality in the [references](#) below.

Name	Address	Width	Access	Description
Receiver Buffer	0	8	R	Receiver FIFO output
Transmitter Holding Register (THR)	0	8	W	Transmit FIFO input
Interrupt Enable	1	8	RW	Enable/Mask interrupts generated by the UART
Interrupt Identification	2	8	R	Get interrupt information
FIFO Control	2	8	W	Control FIFO options
Line Control Register	3	8	RW	Control connection
Modem Control	4	8	W	Controls modem
Line Status	5	8	R	Status information
Modem Status	6	8	R	Modem Status

Table 5: configuration registers

Configuration flow

Before you start configuration make sure the UART IP is out of reset and has an active clock.

- Set bit 7 to 1'b1 of the line control register to allow access to the devisor register.
- Set devisor latches values (see equation below), MSB then LSB.
- Set bit 7 of the line control register back to 1'b0 to disable the access to the devisor latches.
- Set the FIFO trigger level to 0x01 to set the amount of UART writes that will generate interrupt in our case of 0x01 an interrupt is triggered for each word transferred.
- Enable desired interrupts by setting bits [7:6] to low in the interrupt enable register.

To calculate the devisor value, use the following equation:

$$Devisor_{value}[16bit] = \frac{clock_{frequency}}{16 * baudrate} \quad Ex. Devisor = \frac{20MHz}{16 * 9600}$$

The uart_config block is an FSM that handles the upper configuration flow and goes to sleep after config flow is done, when out of reset this block waits for config start trigger.

Gateway

The gateway block is located under the UART io block as seen in previous figures above, the gateway communicates in wishbone from one side with the UART wrapper, and in RC on the other side towards the LOTR ring. This block handles the back pressure created from moving transfers from a fast interface the RC to a significantly slower interface the UART and vice versa.

Transfer handler engine

This unit is responsible for decoding the transfers sent by the Host as explained in the SOFTWARE INTERFACE section above using an FSM to convert them to direct access register transfers where address and data information are sent in parallel in a single cycle to the Reg2rc conversion unit that then moves this transfer forward to the RC controller.

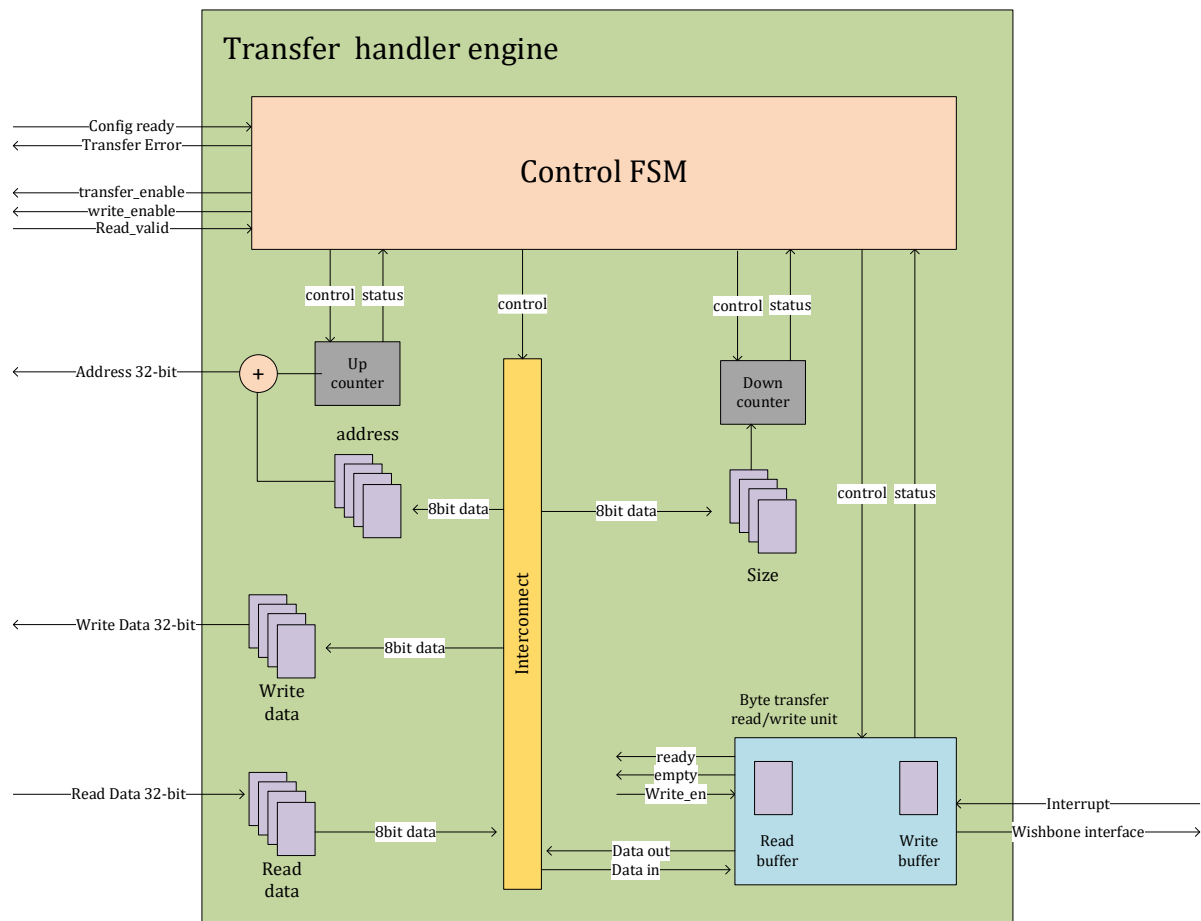
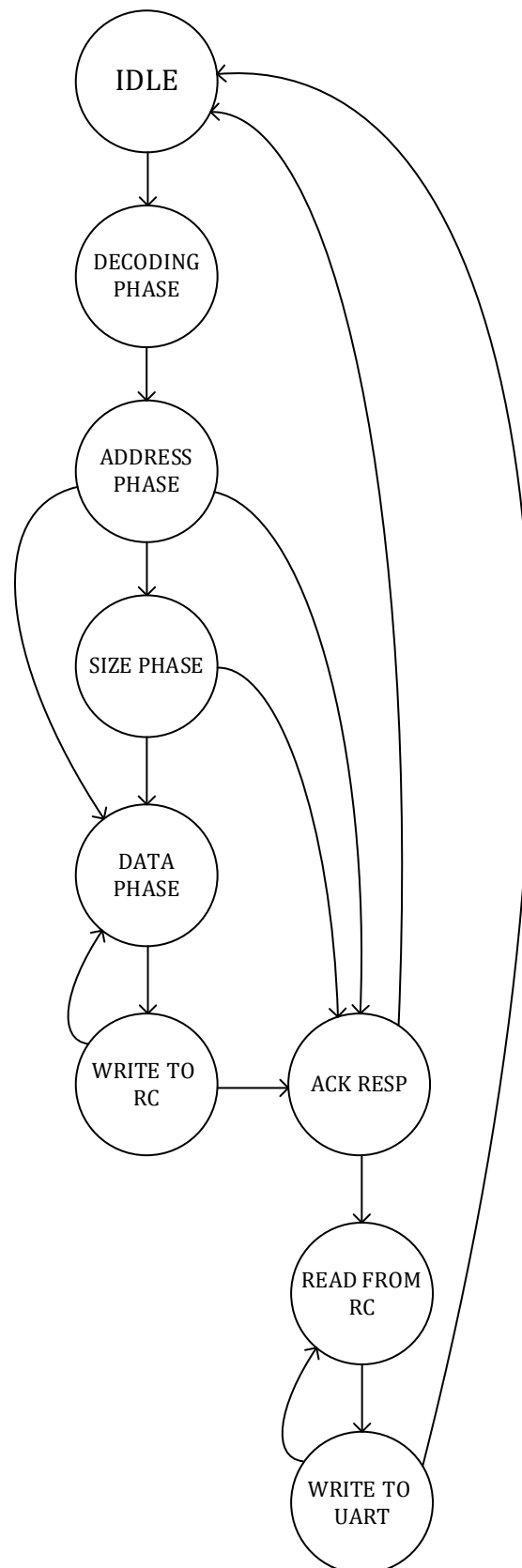


Figure 14: transfer handler engine block diagram

The byte transfer read/write unit (in blue) is a small state machine that is capable of reading and writing from the UART wrapper block using the wishbone interface. this block is used by a higher order state machine that manages the reads and writes to the UART wrapper block according to the transfer structure.

FSM states:



IDLE:

As the name suggests this state keeps the block in idle until any activity is found on the UART block.

DECODONG PHASE:

This state decodes the transfer type, the first byte that is sent over the UART is the transfer type.

ADDRESS PHASE:

This state reads 4 bytes from the UART engine and puts them in the address register.

SIZE PHASE:

This state reads 4 bytes from the UART engine and puts them in the transfer size register.

DATA PHASE:

This state reads 4 bytes from the UART engine and puts them in the data register.

WRITE TO RC:

This state puts a write request on the RC interface using the address and data registers.

ACK RESP:

this state writes to the UART engine a single byte to reply to the Host with an acknowledge.

READ FROM RC:

This state puts a read request on the RC interface using the address register.

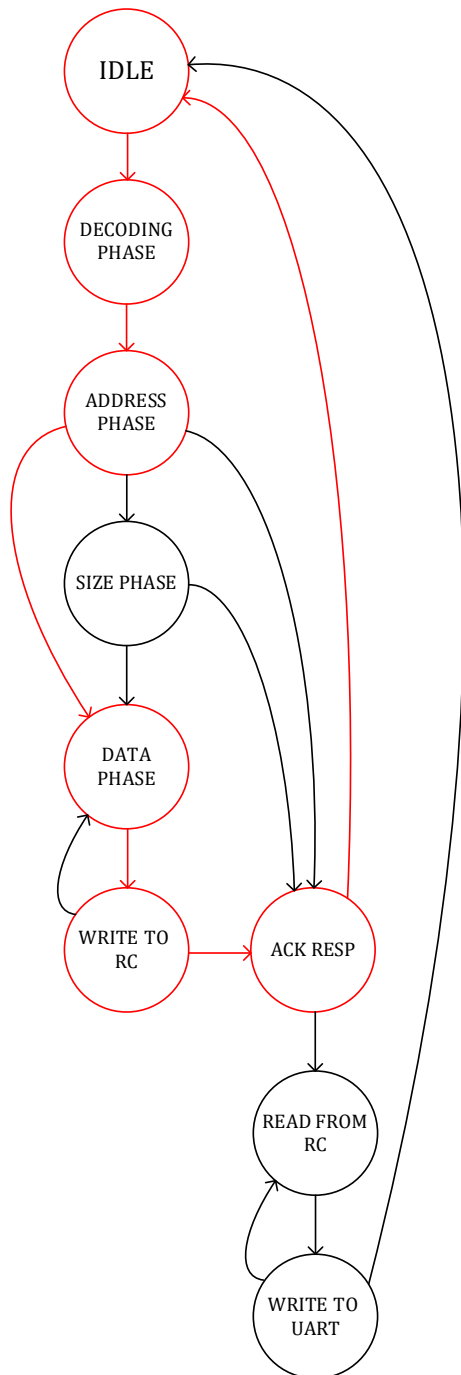
WRITE TO UART:

This state writes the data that is read from the RC to the UART engine, this state creates back pressure towards the RC since the speed at which we can read from the RC is much higher than the speed at which we can write data to the UART.

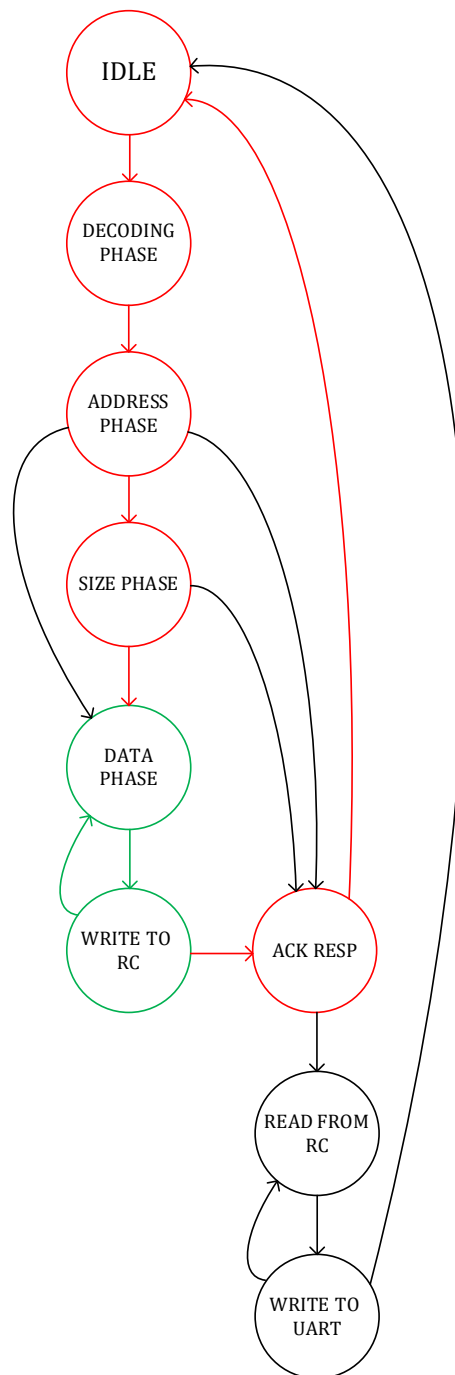
Write and Write burst transfers FSM flow:

In red are states and transitions that occur only once per transfer. In green are the states and transitions that might occur more than once for example in the write burst the FSM reads 4 bytes of data from UART in the DATA_PHASE state and writes them on the RC then goes back to read 4 more bytes.

Write transfer



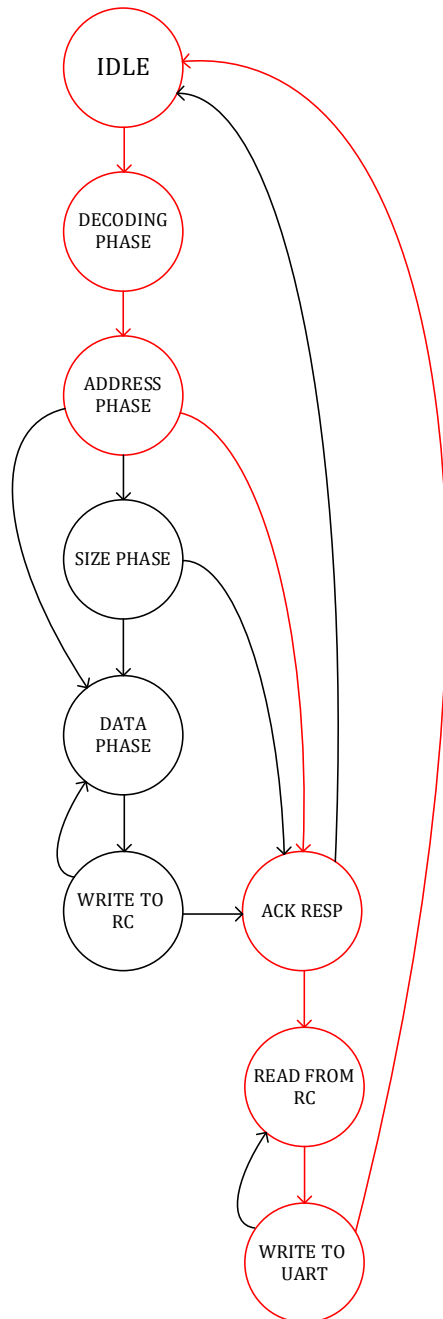
Write burst transfer



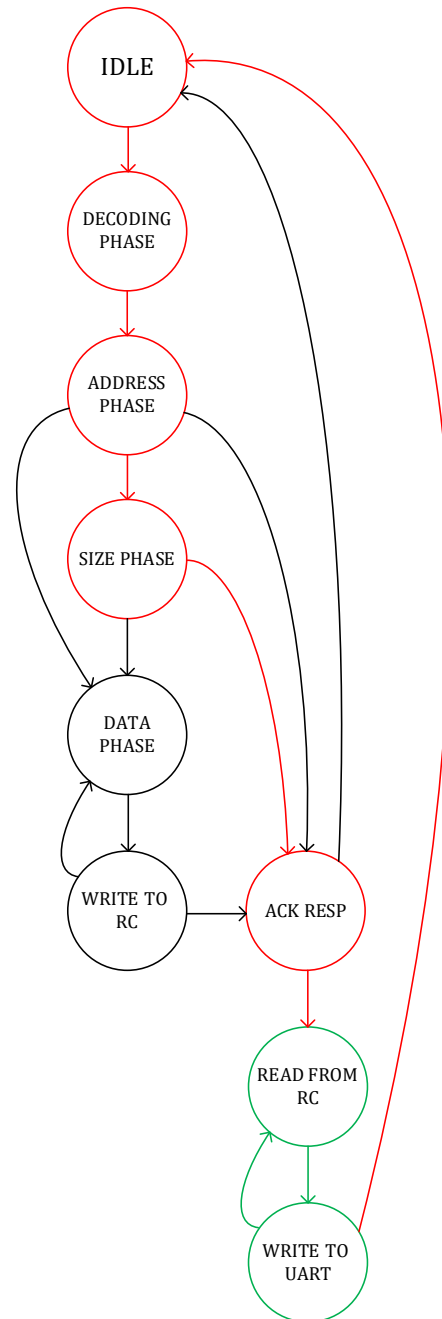
Read and Read burst transfers FSM flow:

In red are states and transitions that occur only once per transfer. In green are the states and transitions that might occur more than once for example in the read burst the FSM reads 4 bytes of data from the RC in the READ_FROM_RC state and writes them on the UART then goes back to read 4 more bytes. In black are states and transitions that are not used.

Read transfer



Read burst transfer



Ring Controller (IP)

The Ring controller is a ring network topology in which each agent connects to exactly two other agents, forming a single continuous pathway for transactions through each agent in the ring.

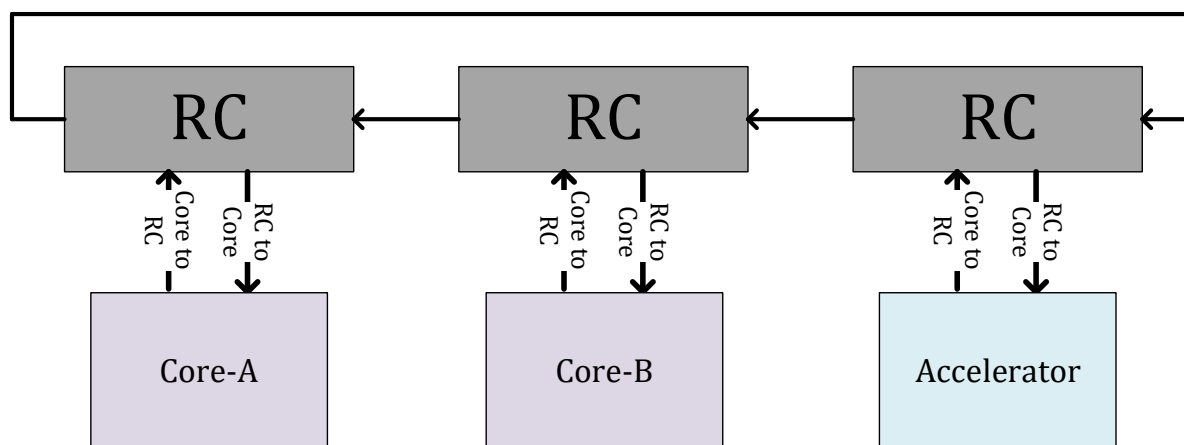


Figure 15: Ring network topology example

The Ring controller is an arbitration logic that manages the transactions from any core unit compatible with the RC interface mentioned [above](#) to the ring. the Ring controller has 4 interfaces that follow the RC interface rules each with two channels, request, and response.

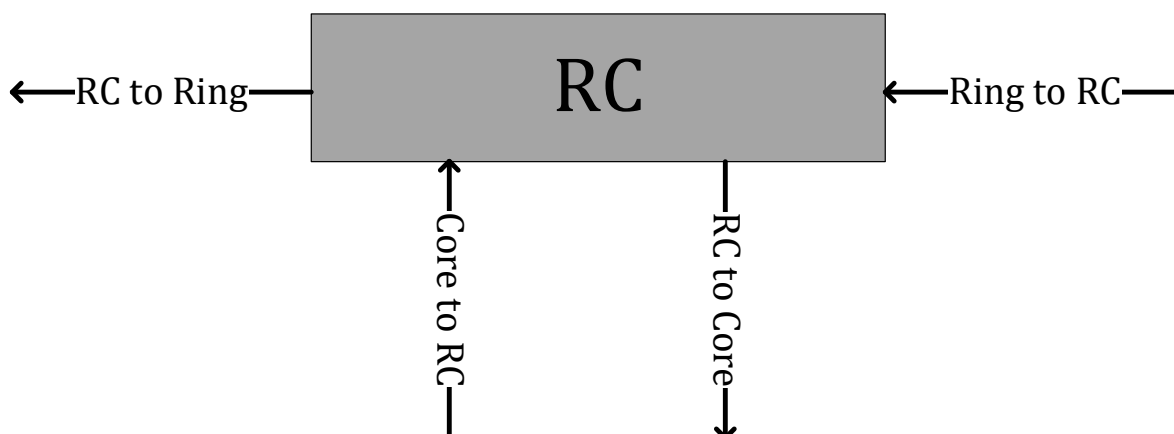


Table 6: Ring controller interface exploration

Interface exploration:

Interface	Source and destination
Ring to RC	Ring transaction requests towards the RC and the RC's response lines
RC to Ring	RC transaction requests towards the Ring and the Ring's response lines
Core to RC	Core transaction requests towards the RC and the RC's response lines
RC to Core	RC transaction requests towards the Core and the Core's response lines

Table 7: interface exploration

Detailed explanation can be found in the [references](#) below.

VERIFICATION & VALIDATION

Block-level RTL simulations

The simulation includes the uart_io module, the UART signal behavior has been imitated using systemverilog tasks, on top of that other tasks are used to imitate the transfers

Compilation command:

```
vlog.exe -f ./uart_io/uart_io_list.f -work ./uart_io/work -skipsynthoffregion -lint -source -warning [] -fatal [] -note [] -error [] -suppress 2275
```

simulation command:

```
vsim.exe -Ldir "uart_io" -t 1ns work.uart_io_tb -work ./uart_io/work -gui -do "run -all" -checkvifacedrivers 1 -no_autoacc -keeploaded -suppress 8315,3584,8233,3408,3999 +UART_SIMULATION
```

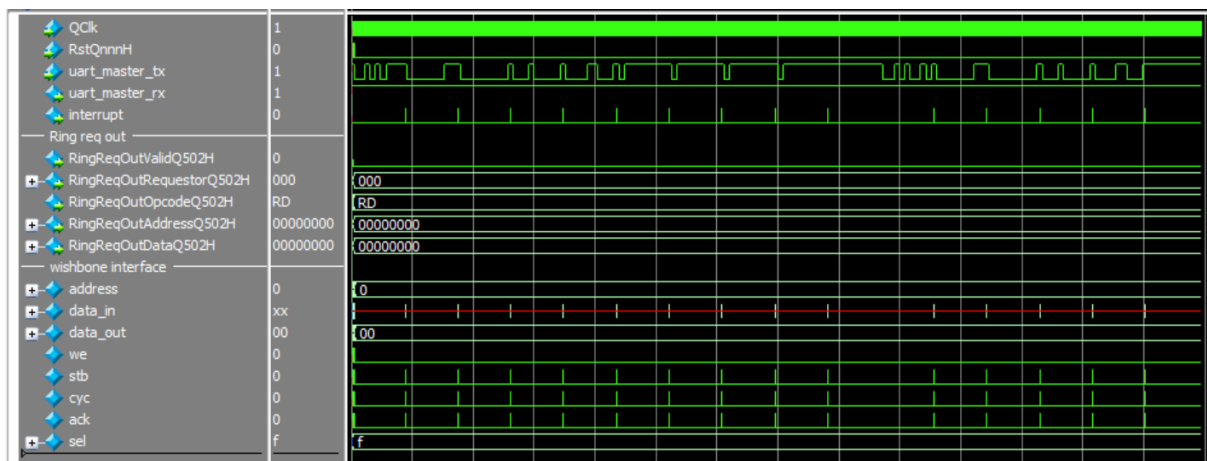


Figure 16: block-level simulation waveform

System-level RTL simulations

Compilation command:

```
vlog.exe -f ./uart_io/uart_io_list.f -work ./uart_io/work -skipsynthoffregion -lint -source -warning [] -fatal [] -note [] -error [] -suppress 2275
```

Simulation command:

```
vsim.exe -Ldir "uart_io" -t 1ns work.uart_io_tb -work ./uart_io/work -gui -do "run -all" -checkvifacedrivers 1 -no_autoacc -keeploaded -suppress 8315,3584,8233,3408,3999 +UART_SIMULATION
```

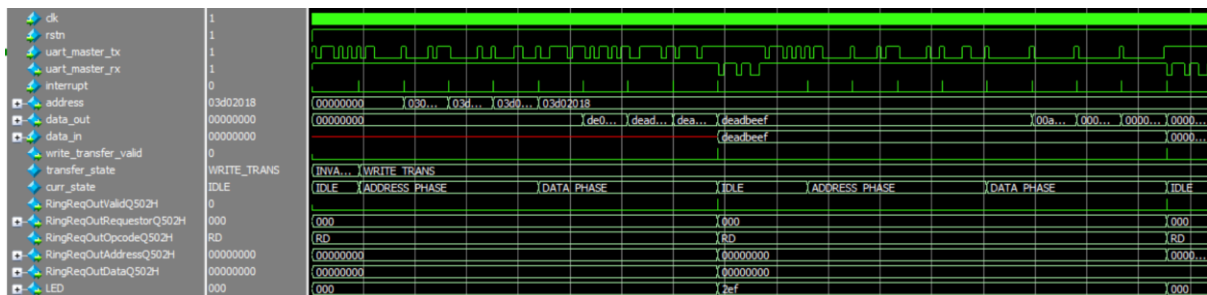
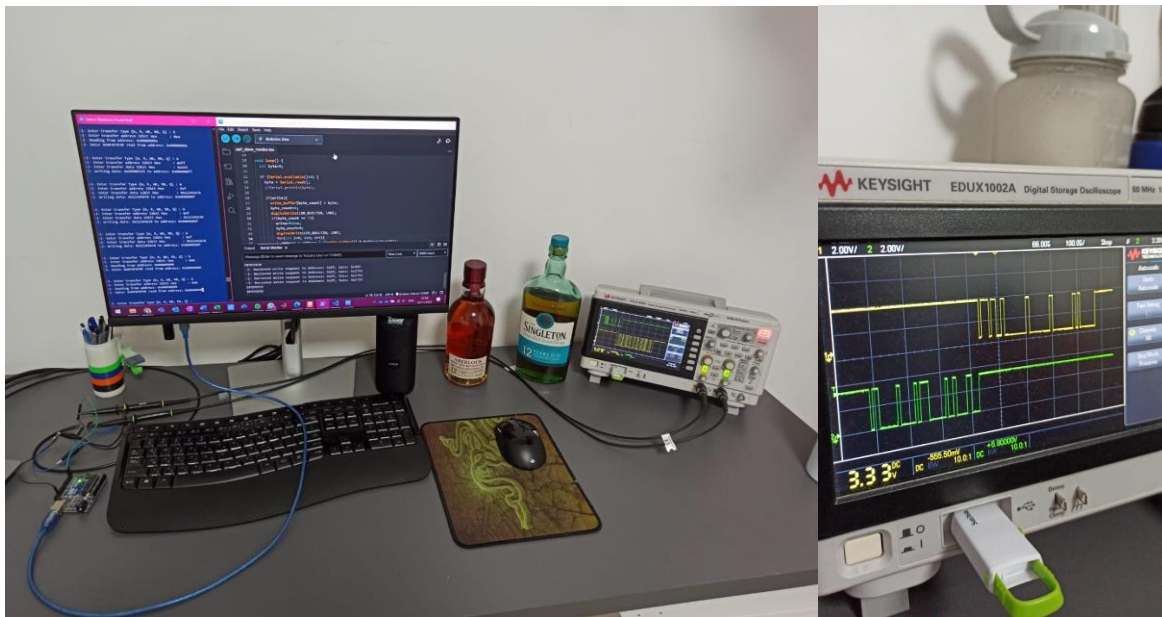


Figure 17: system-level simulation waveform

Functional validation

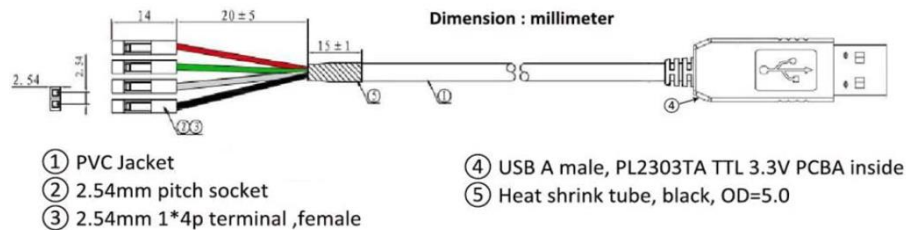
The functional validation has been done using an Oscilloscope to monitor the UART RX and TX lines, using these two lines one can observe the transactions being sent to and from the device.



FPGA INTEGRATION

This section will show the setup and connections to the FPGA, the UART tile has 2 external signals, meaning 2 signals that connect with the outer world like switches and LED's. these two signals are the UART RX and TX pins that are driven by the USB-to-TTL converter cable, these signals are the UART TX and RX signals you saw earlier on the UART protocol section [above](#). This is how the Host communicates with the Device.

Following pin description of USB-to-TTL converter



1*4P Female Socket	Name	Colour	Description
Pin 1	TXD	White	Transmit Asynchronous Data
Pin 2	RXD	Green	Receive Asynchronous Data
Pin 3	GND	Black	Device ground supply
Pin 4	VCC	Red	+5V

Figure 18: USB-to-TTL

Three terminals should be connected, GND as a reference ground to any GND pin on the FPGA, the green terminal as the RX of the master, and the white terminal as the TX of the master.

On the FPGA there are many available pins on the GPIO port. On this project PIN_AA15 has been chosen to be the TX and PIN AA14 has been chosen to be the RX pin.

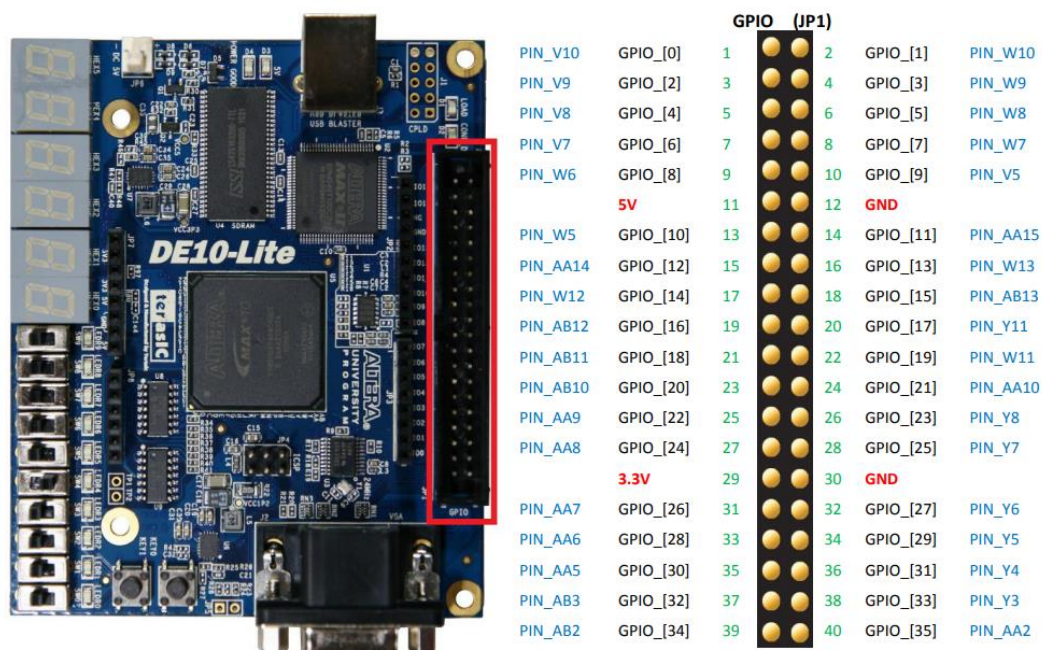


Figure 19: DE10-Lite GPIO pins

USB-to-TTL connection to the FPGA:

Source	Destination	Comments
White (TXD)	PIN_AA15	
Green (RXD)	PIN_AA14	
Black (GND)	GND	
Red (VCC)	-	Floating

Table 8: USB-to-TTL connection

Following reference photo:

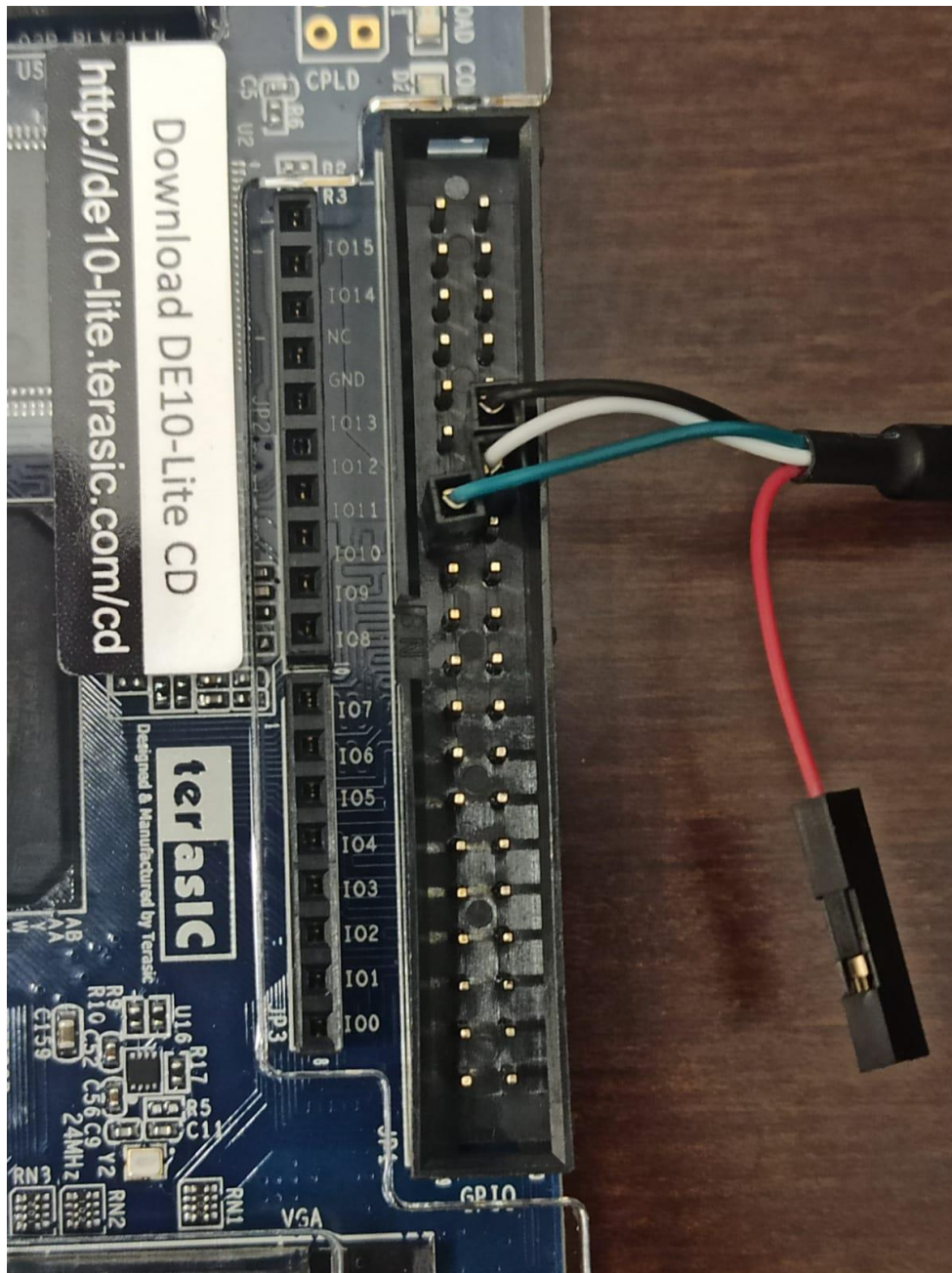


Figure 20: connecting the terminals

FPGA AREA REPORTS

The area reports have been done on a 50Mhz clock while using the INTEL Quartus prime software.

hierarchy	unit	Logic cells	Registers
0	lotr	25785	15675
1	gpc4t_tile	11047	6683
1	fpga_tile	1536	966
1	uart_tile	2202	1343
2	rc	989	563
2	uart_io	1273	780
3	uart_wrapper	870	525
3	gateway	416	255
4	transfer handler engine	360	221
4	uart_config	37	34

Table 9: fpga synthesis area reports

REFERENCES

Any further information can be found in the references, this project relies tightly on the following Links, feel free to further educate yourself.

Reference	Link
<i>Ascci table</i>	Link
<i>Little/Big endianness</i>	Link
<i>UART16550 IP spec</i>	Link
<i>Wishbone interface</i>	Link
<i>DE10-Lite FPGA</i>	Link
<i>USB-to-TTL converter</i>	Link
<i>PySerial library</i>	Link
<i>UART protocol</i>	Link
<i>Ring Controller</i>	Link
<i>RC interface</i>	Link
<i>USB protocol</i>	Link , Link

Table 10: references table