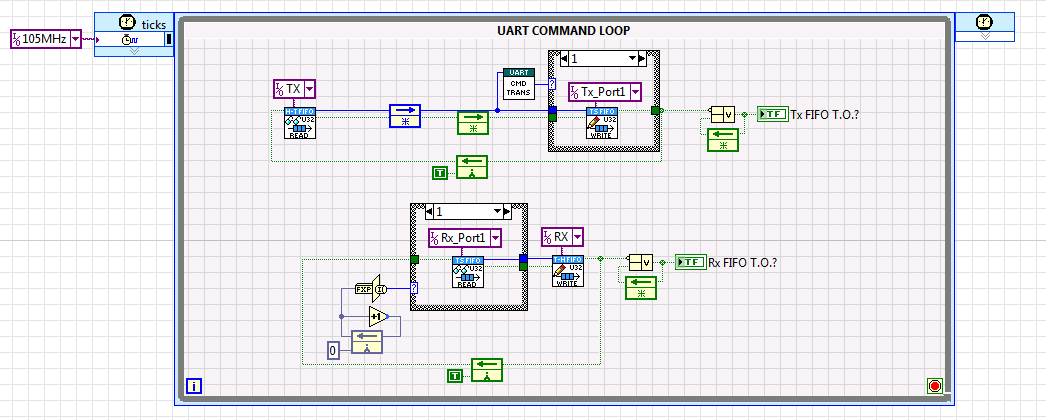
UART FPGA Implementation

## Purpose

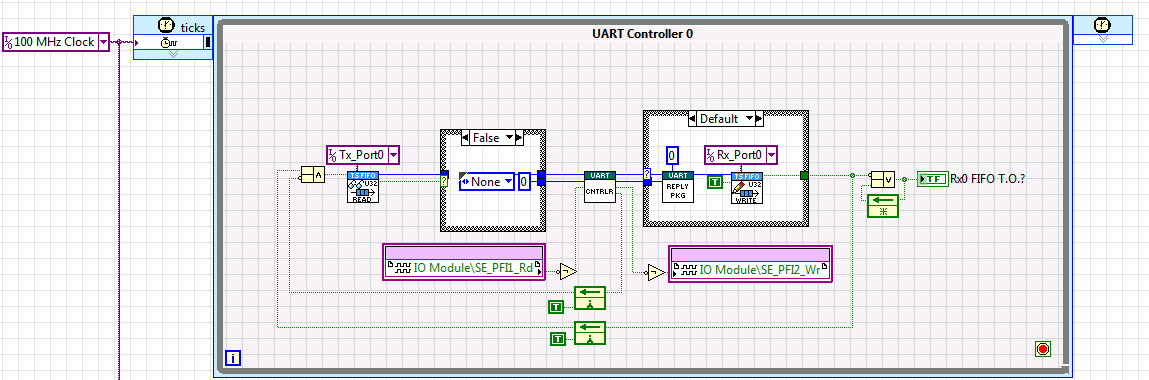
This document outlines how the Field Programmable Gate Array (FPGA) Universal Asynchronous Receiver/Transmitter (UART) based implementation is intended to work.

## Key Components

* UART Command Loop
  + Responsible for both receiving and transmitting messages from the host.
  + Responsible for transmitting messages to each UART Controller
  + SCTL executes 105MHz – because we need to ensure that both incoming and outgoing messages are processed and don’t result in a FIFO Timeout. This value needs to be greater than the controller loop rate (in this case 100MHz).



* UART Controller Loop
  + Responsible for receiving messages from the UART Command Loop and processing the data accordingly



* Controller
  + Core Component of the UART FPGA solution
    - Responsible for decoding messages from the host
    - Responsible for receiving messages
    - Responsible for transmitting messages
    - Responsible for send a complete packet back to UART Command Loop to be transmitted to the host



## Features

* Supports:
  + Variable baud rates up to 25MHz
  + Variable character length
  + Stop Bits (1.0, 1.5, and 2.0)
  + Parity (None, Even, and Odd)
  + Character Spacing

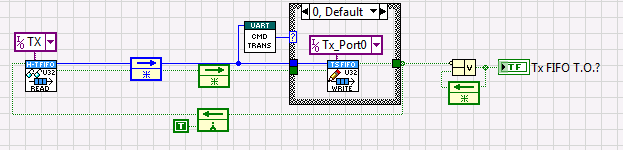
## UART Tx Data Operation

### Execution Order

1. A command is received from the host application in the UART Command Loop.

2. The command is decoded to determine which UART controller is the intended target of the command.

3. The command is then transmitted to the correct controller.



4. The UART controller decodes the command message and populates the data payload for the respective field.

* Command Messages:
  + None
  + Tx
  + Set Baud\_High
  + Set Baud\_Low
  + SetCharLength
  + SetParity
  + SetStopBits
  + SetSpacing\_High
  + SetSpacing\_Low

5. New Commands are processed until a Tx command is received.

* Once a Tx command is received no new commands will be processed until the Tx data payload is fully transmitted.

6. The Core of the Tx data transmission is located in the Tx\_logic.vi. This is a state machine based design.

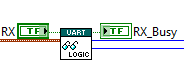
Macintosh HD:Users:aryan:Desktop:Screen Shot 2013-05-23 at 1.46.13 PM.png

* States
  + Idle
    - 1st state of execution when a Tx command is received
    - Populates
      * *CharCount*
      * *Character*
      * *Parity*
  + Start
    - 2nd state of execution
      * Counts up to *N ticks* based upon the SCTL rate (100MHz)/Baud rate = *N ticks*
      * This state allows us to align the data bit transmission with when the data can be safety acquired
  + Character
    - 3rd state of execution
      * The *Character* data payload is transmitted bit by bit based upon *N ticks* count until all the bits have been transmitted
      * Once all of the *Character* bits have been transmitted the next state depends upon the value of *Parity*.
  + Parity
    - 4th state of execution?
      * Transmit *Parity* bit for *N ticks*
      * Proceed to Stop State once complete
  + Stop
    - 4th state of execution?
      * Transmit Stop Bit based upon the number of value of StopBits set by Command Message for…
        + 1.0(Default) = *N ticks*
        + 1.5 = 1.5\**N ticks*
        + 2.0 = 2\**N ticks*

## UART Rx Data Operation

### Execution Order

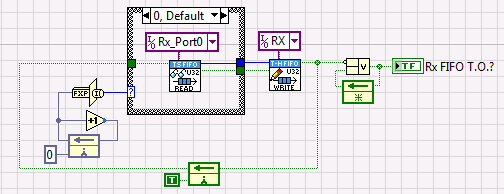
1. The Core of the Rx data transmission is located in the Rx\_logic.vi. This is a state machine based design.



* States
  + Idle
    - 1st state of execution when a Rx command is received
    - Populates
      * *CharCount*
      * *Character*
      * *Parity*
      * *Rx Reply Message­ –* Used by host application to decode message
  + FindCenterStart
    - 2nd state of execution
      * Counts up to *N ticks* based upon the SCTL rate ((100MHz)/Baud rate)/2 = *M ticks*
      * This state calculates have the pulse period *N ticks* and shifts when the data is sampled
        + By halving the period we ensure a safely shift the acquisition later in time such that a pulse can be sampled in the middle and mitigate the threat of a metastable event.
  + Character
    - 3rd state of execution
      * The *Character* data payload is received bit by bit based upon *N ticks* count until all the bits have been received.
      * Once all of the *Character* bits have been received the next state depends upon the value of *Parity*.
  + Parity
    - 4th state of execution?
      * Wait on *Parity* bit for *N ticks*
        + If *Parity* hasn’t occurred with *N ticks* generate a ParityError *Reply Rx Message*
      * Proceed to Stop State once complete
  + Stop
    - 4th state of execution?
      * Wait on Stop Bit based upon the number of value of
        + If *Parity* hasn’t occurred with *N ticks* generate a FrameError *Reply Rx Message*
      * Once Stop is proceed to Idle State.
  + Default
    - Should never execute

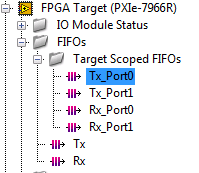
2. Once the Stop state has finished, the UART\_Reply and Reply\_Data (combined data called *Reply Package*) are transferred to UART Command Loop.

3. The UART Command Loop then transmits the *Reply Package* back to the host application.

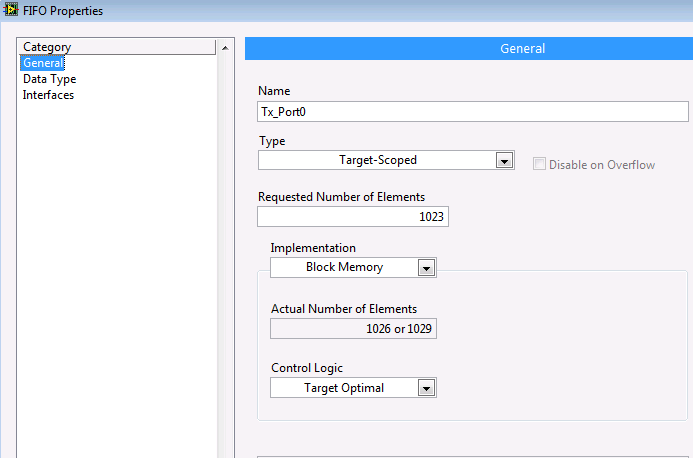


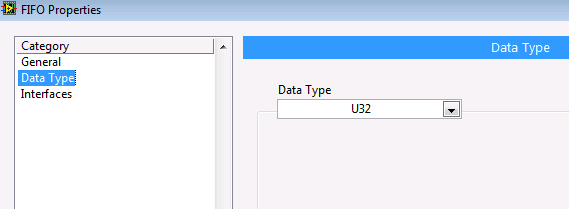
## Extending the number of UART Controllers

1. Create 2 new Target Scoped FIFOs

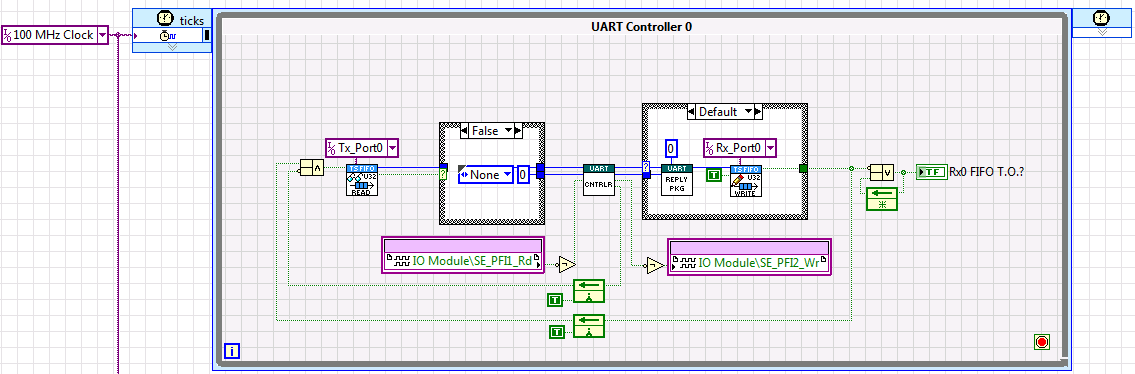


* One for Tx and one for Rx
* Name the FIFOs accordingly
* Configuration the FIFOs accordingly

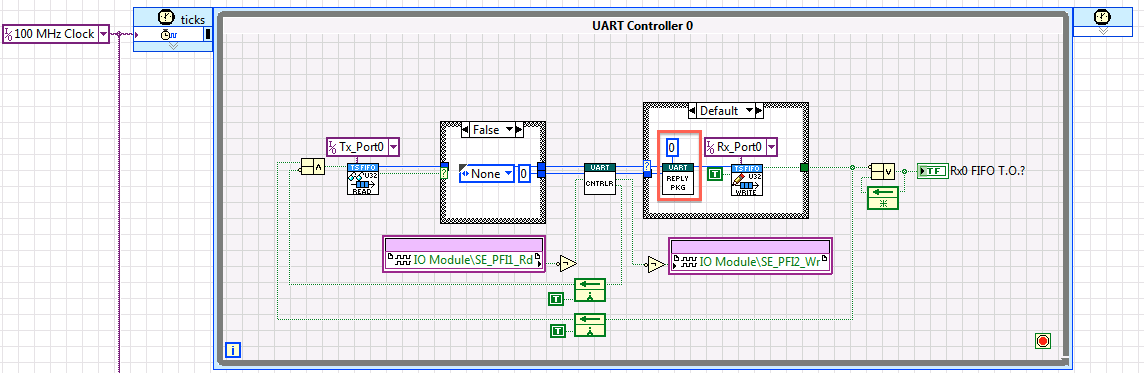




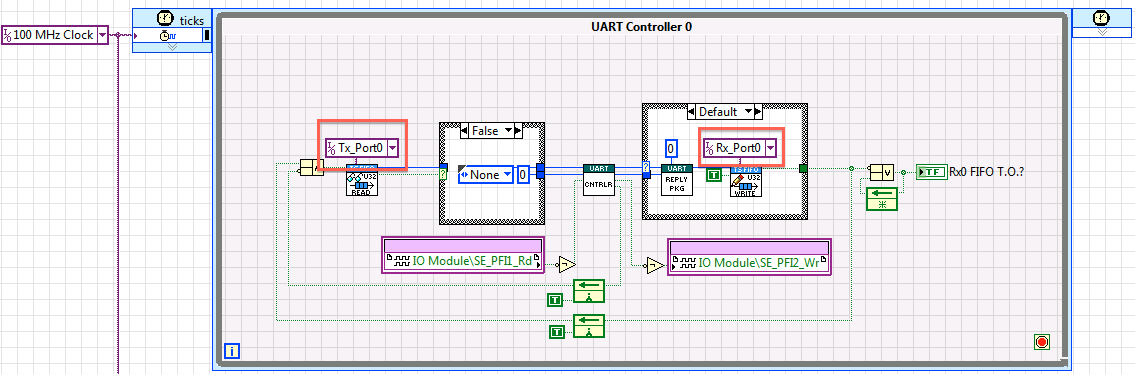
2. Copy one of the UART Controller Loops



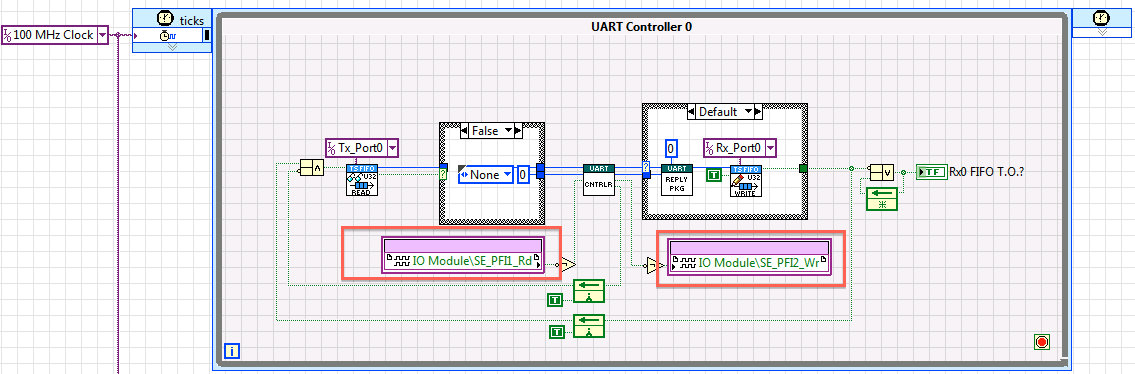
3. Change the Reply Package Channel control to match UART Controller Number



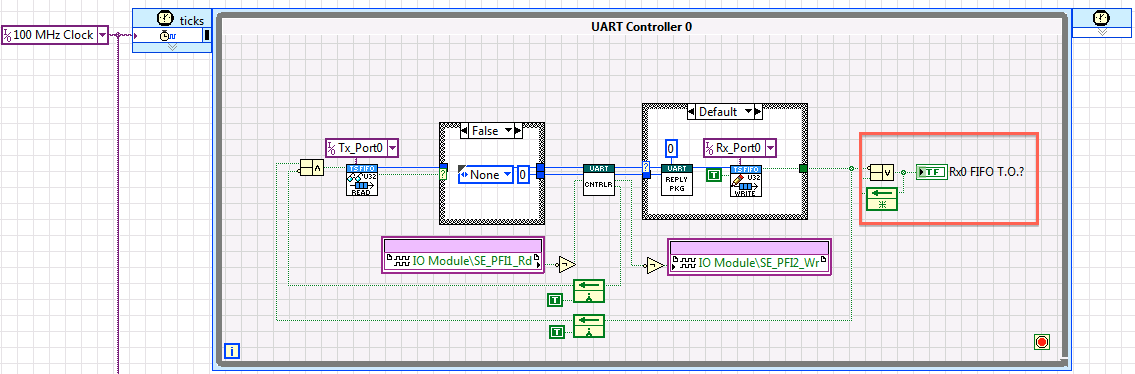
4. Select the correct drop down for the Target Scoped FIFOs



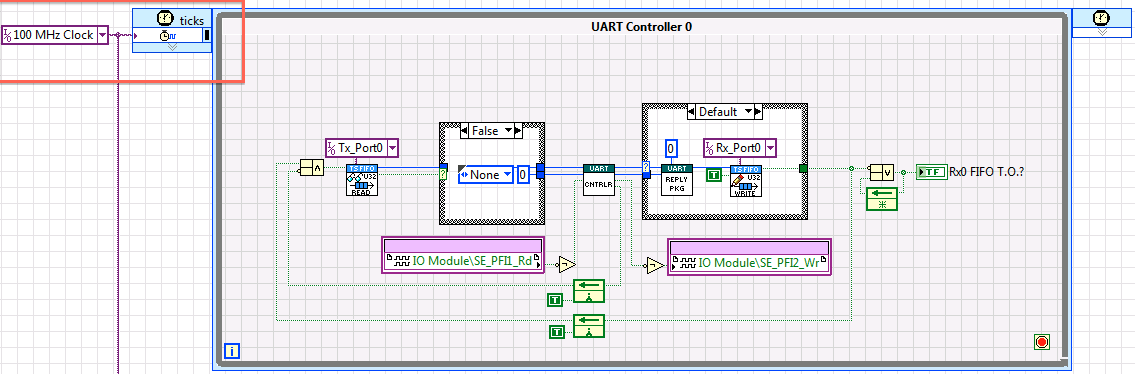
5. Select different IO Channels



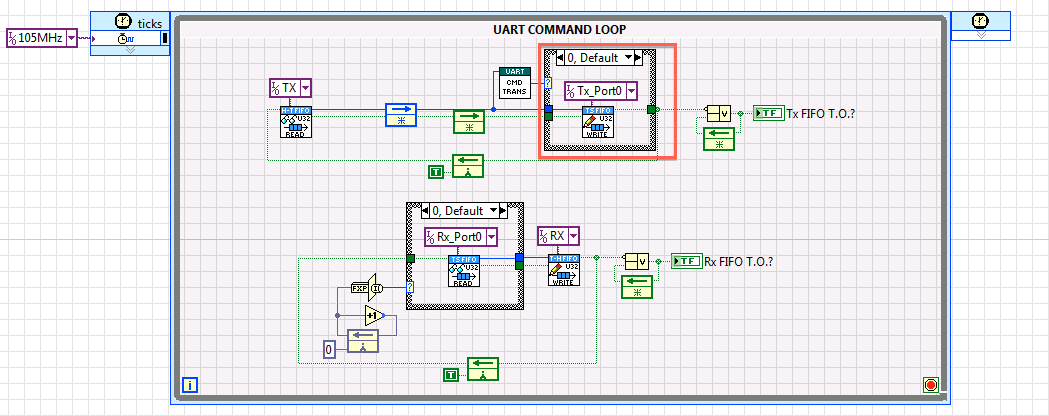
6. Rename Rx\_ FIFO T.O.? to match your new UART Controller number



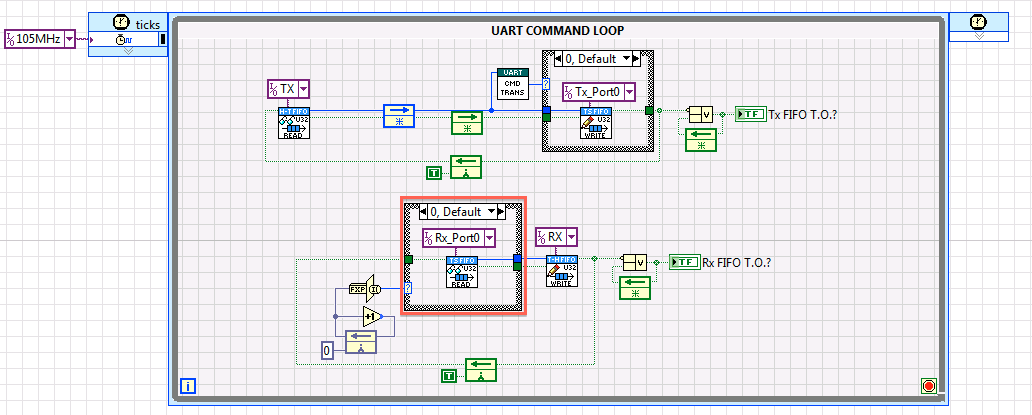
7. Ensure the 100 MHz clock constant is connected to the SCTL input node



8. Add a new case the UART Command Loop TX Command Pipe with a new Target Scoped FIFO (similar to step 3).

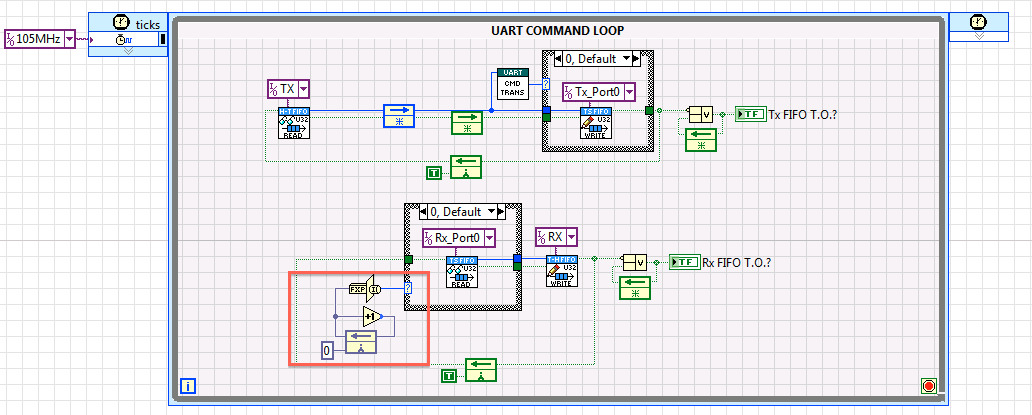


9. Add a new case to the UART Rx Reply Package Pipe with a new Target Scoped FIFO (similar to step 7).

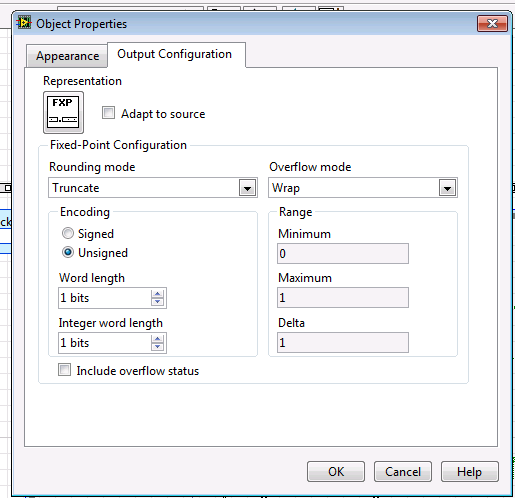


10. Check Rx Mux to ensure you new Rx Target Scope FIFO will be serviced.

* The Increment function is set to wrap, change the FXP value of the output and default value accordingly.



* Increment Output Configuration



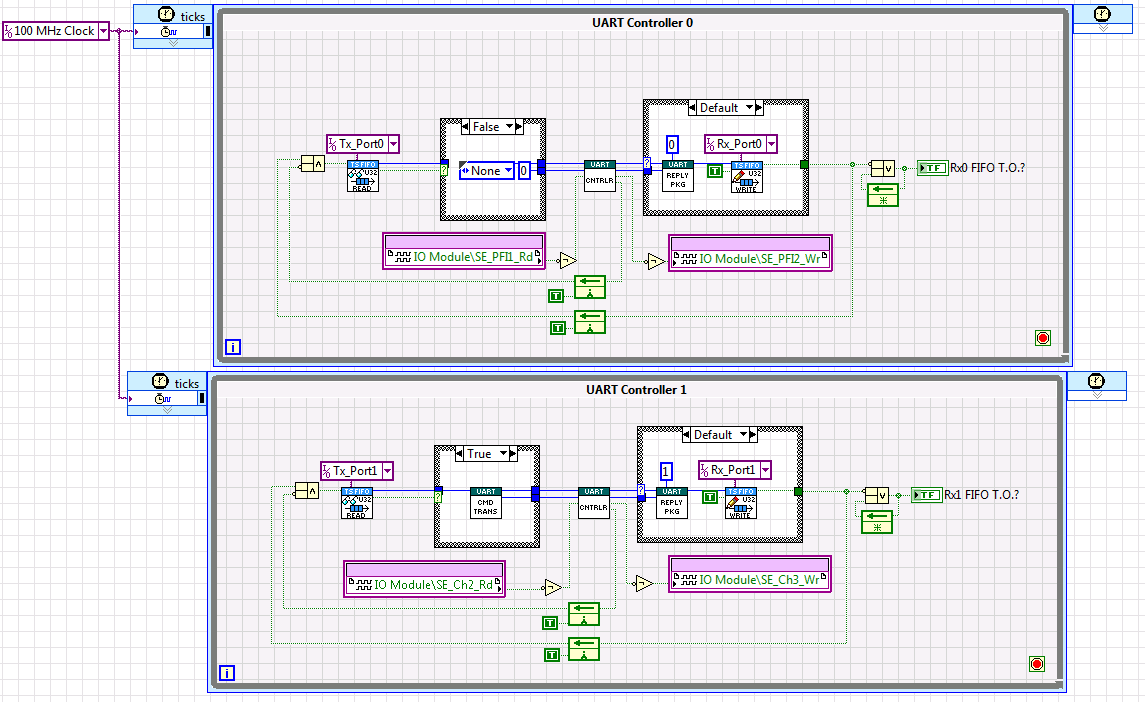


Figure 1: Example of a 2 UART Controller Configuration