Key Design Features

- Synthesizable, technology independent IP Core for FPGA, ASIC and SoC
- Supplied as human-readable VHDL (or Verilog) RTL source
- UART (Master) connects to a standard I2C bus and operates as a fully 'transparent' bridge between the two buses
- Support for standard UART data rates between 9600 and 921600 baud
- Support for all common I2C bit rates such as 100kbps, 400kbps, 1Mbps and 4Mbps
- Simple command interface allows the programming of I2C peripherals using a serial terminal program (e.g.TeraTerm, HyperTerminal, PuTTY, YAT etc.)

Applications

- Convenient method for translating commands between UART and I2C devices and peripherals
- Essential tool for the remote debug of I2C devices using an external PC, micro-processor or mico-controller

Block Diagram

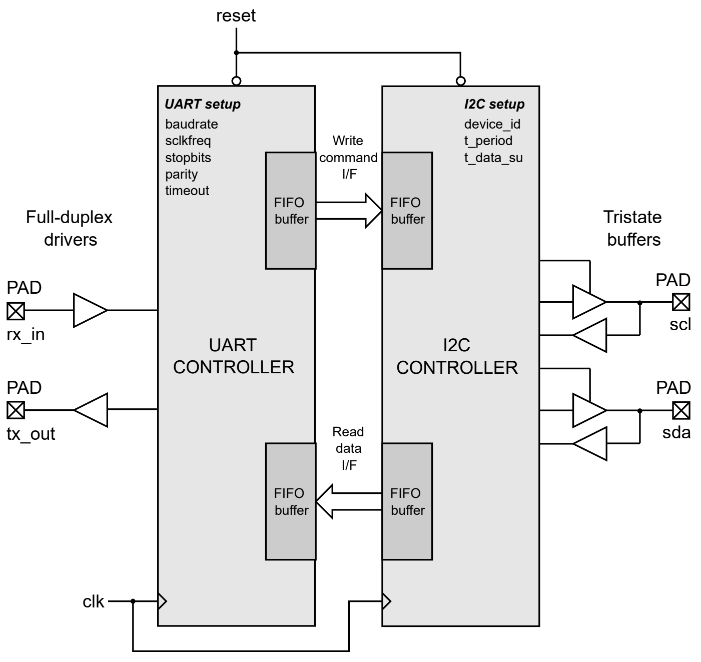


Figure 1: UART-to-I2C bridge architecture

Generic Parameters

Generic name	*Description*	*Type*	*Valid range*
baudrate	UART baud rate in bits per second	integer	9600 to 921600 (custom rates also supported)
sclkfreq	System clock frequency in Hz	integer	Ratio (sclkfreq / baudrate) < 65536
databits	Number of data bits	integer	Set to 8 only
stopbits	Number of stop bits	integer	1,2
parity	Enable parity bit after data payload in bitstream	integer	0: none 1: even 2: odd 3: mark 4: space
timeout	UART timeout in system clock cycles (must be greater than the time taken to send 32 bits at the current baud rate)	integer	≥ 2 (but much larger in practice)
device_id	I2C slave device ID	integer	≥ 2
t_period	SCL clock period (as number of system clock cycles)	integer	≥ 2
t_data_su	SDA setup/hold time (as number of system clock cycles)	integer	≥ 4

Pin-out Description

Pin name	*I/O*	*Description*	*Active state*
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
rx_in	in	Serial bits in	UART serial data
tx_out	out	Serial bits out	UART serial data
scl	i/o	I2C bi-directional SCL clock pin	As per I2C specification
sda	i/o	I2C bi-directional SDA data pin	As per I2C specification

General Description

The BRIDGE_UART_I2C IP Core (Figure 1) provides a simple and convenient way to interface a standard UART bus to a standard I2C bus. The circuit operates as a completely transparent 'bridge' between the two buses and allows I2C peripherals to be programmed using a set of basic commands over a (UART) serial interface.

Both the UART and I2C transceivers may be configured individually to support a wide range of standard and custom settings. The bridge circuit is technology independent and may be implemented as a custom ASIC, or using a standard FPGA or SoC.

In particular, the IP Core is ideal for the remote programming of I2C peripherals using a remote PC or micro-controller. This is especially useful during the debug stages of a project when the I2C registers of a device may be written and read with the help of a simple terminal program.

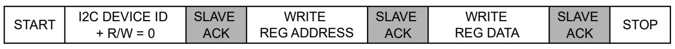
UART Command Interface

The UART command interface consists of 4 consecutive bytes that are used to initiate a read or a write on the I2C bus. Typically, this is used to read or write a register accessed via I2C.

An I2C write is initiated by the following command:

0xFF 0x55 0xADDR 0xDATA

(Where 0xADDR is the register address to be accessed and 0xDATA is the byte of data to be written). Figure 2 shows the corresponding command that is generated on the I2C bus:



Example: With the I2C 'device_id' generic set to 0x40, the UART command: 0xFF 0x55 0x3A 0x72 will generate the following sequence on the I2C bus:

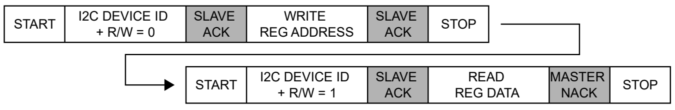
<START> <0x40> <ACK> <0x3A> <ACK> <0x72> <ACK> <STOP>

Figure 2: I2C write command sequence

Similarly, an I2C read is initiated by the following command:

0xFF 0xAA 0xADDR 0xXX

(Where 0xADDR is the register address to be accessed and the value of byte 0xXX is don't care). Figure 3 below shows the command that is generated on the I2C bus:



Example: With the I2C 'device_id' generic set to 0x40, the UART command: 0xFF 0xAA 0x3A 0x00 will generate the following sequence on the I2C bus:

<START> <0x40> <ACK> <0x3A> <ACK> <STOP>
<START> <0x41> <ACK> <0xREAD DATA> <NACK> <STOP>

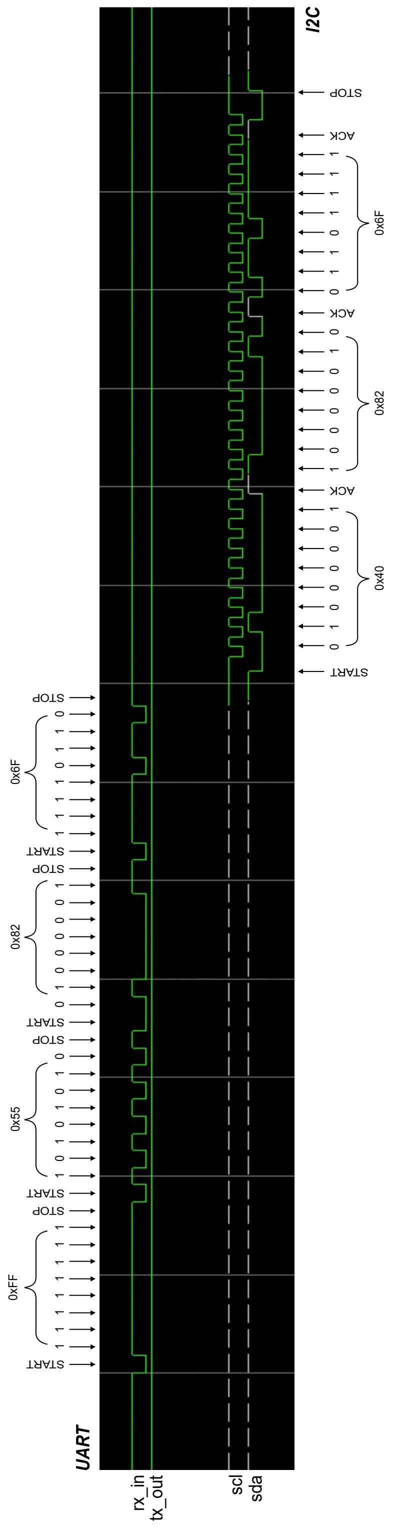
Figure 3: I2C read command sequence

Note that the I2C slave address *device_id* must be set correctly in the generic parameters so that the correct I2C slave device is accessed on the bus.

Another important consideration is to make sure that the UART and I2C generics are set correctly for the desired data rates. In addition, the UART *timeout* setting should be set appropriately for the given baud rate. For instance, if the baud rate is set to 9600 with 1 stop bit and no parity, then a four byte UART command should take: 10 x 4 / 9600 = approx 4.2 ms. As such, the timeout should be set to greater than 4.2 ms with some margin to compensate for a slow terminal.

Functional Timing

Both the UART and I2C buses implement standard timing waveforms as described in the UART and I2C specifications. The timing diagram below gives an example for the command: 0xFF 0x55 0x82 0x6F with the baud rate set to 115200 and the I2C clock set to 100kHz.



Source File Description

All source files are provided as text files coded in VHDL (or Verilog on request). The following table gives a brief description of each file.

Source file	***Description***
uart_fifo.vhd	UART transmit and receive FIFOs
uart_tx.vhd	UART Transmitter
uart_rx.vhd	UART Receiver
uart_cont.vhd	UART controller FSM
uart_packet_decode.vhd	UART data packet decoder
i2c_iobuf.vhd	Bi-directional tristate buffer
i2c_delay.vhd	Adds setup delay to the SDA line
i2c_fifo.vhd	I2C transmit and receive FIFOs
i2c_master_cont.vhd	I2C controller FSM
i2c_master.vhd	I2C controller top-level
i2c_packet_decode.vhd	I2C data packet decoder
i2c_slave_dummy.vhd	I2C dummy slave device for testing
bridge_uart_i2c.vhd	Top-level bridge component
bridge_uart_i2c_bench.vhd	Top-level bridge test bench

Functional Testing

An example VHDL test bench is provided for use in a suitable hardware simulator such as Modelsim® from Mentor Graphics. The test bench instantiates the top-level BRIDGE_UART_I2C IP Core and drives the UART serial inputs with a sequence of 4 x write commands and 4 x read commands.

In the example provided, the generic settings have been set such that the baud rate is 115200 and the I2C clock rate is 100 kHz. Data bits are fixed at 8 with 1 stop bit and no parity.

For the purposes of simulation, the bridge is connected to a dummy I2C slave device. The slave device drives random bits back to the master.

In the default setup, the simulation should be run for around 100 ms. During the simulation, the UART transmit bytes and UART receive bytes are captured in the text files: *uart_tx_out.txt* and *uart_rx_out.txt.*

Development Board Testing

The UART-to-I2C bridge controller was tested using a Xilinx® Artix-7 AC701 development board running at a system clock frequency of 100 MHz. The AC701 features a single 6-pin PMOD™ header which was used to connect the external UART and I2C buses.

The UART Rx/Tx lines were connected to the host PC serial port via an RS-232 adapter. This was to ensure that the UART lines were at the correct 3.3V levels. The I2C bus was connected to the PICkit Serial I2C demo board from Microchip®. Figure 4 shows the basic bench setup in more detail.

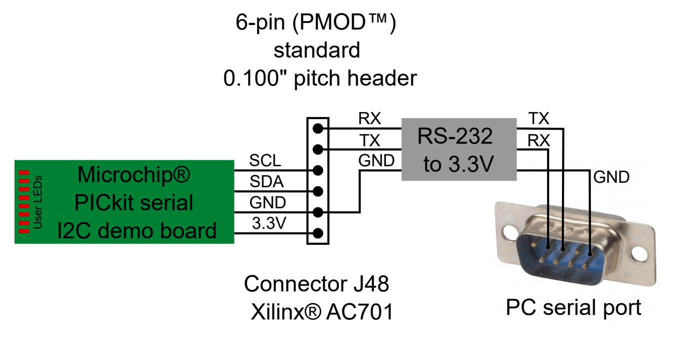
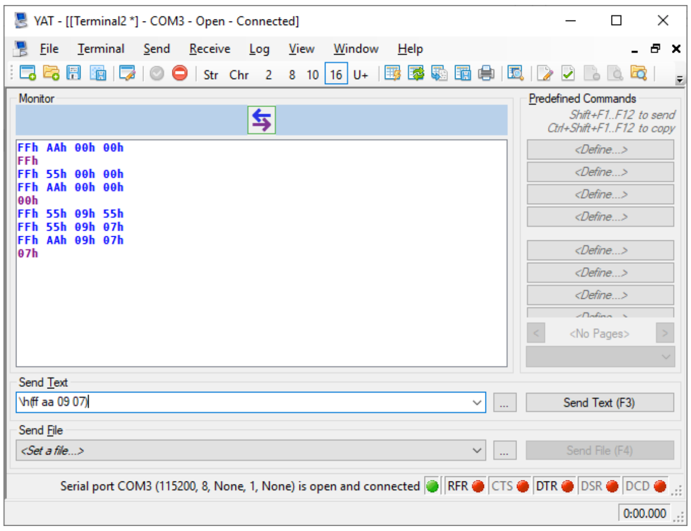

6-pin (PMOD™) standard 0.100" pitch header
RX
TX
GND
RS-232 to 3.3V
TX
RX
GND
SCL
SDA
GND
3.3V
User LEDs
Microchip® PICkit serial I2C demo board
Connector J48 Xilinx® AC701
PC serial port

Figure 4: Bench setup for testing the UART-to-I2C bridge controller on the AC701 development board

Once the kit was set up as described and the bitfile programmed on the AC701 board, the serial terminal application was started on the host PC. In this example the YAT terminal application was chosen for its ease of use and wide range of features.

The command interface was then used to program the Microchip PICkit board over the I2C bus, using it to illuminate various user LEDs. Figure 5 shows the YAT terminal application in action with various reads and writes shown in the terminal window.



```
FFh AAh 00h 00h
FFh
FFh 55h 00h 00h
FFh AAh 00h 00h
00h
FFh 55h 09h 55h
FFh 55h 09h 07h
FFh AAh 09h 07h
07h
```

\h(ff aa 09 07)

Serial port COM3 (115200, 8, None, 1, None) is open and connected

Figure 5: YAT terminal program set up for serial comms

Synthesis and Implementation

The files required for synthesis and the design hierarchy is shown below:

- bridge_uart_i2c.vhd
  - uart_cont.vhd
    - uart_rx.vhd
    - uart_fifo.vhd
    - uart_tx.vhd
  - uart_packet_decode.vhd
  - i2c_packet_decode.vhd
  - i2c_master.vhd
    - i2c_fifo.vhd
    - i2c_master_cont.vhd
    - i2c_iobuf.vhd
    - i2c_delay.vhd

The BRIDGE_UART_I2C IP Core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx® 7-series FPGAs. Synthesis results for other FPGAs and technologies can also be provided on request.

There are no special requirements for implementation. However, it is recommended that the Rx/Tx pins (UART) and the SCL/SDA pins (I2C) are placed directly in the I/O of the device. In this way, it will ensure that bus timing problems are kept to a minimum.

Trial synthesis results are shown with the generic parameters set to: *baudrate* = 115200, *sclkfreq* = 100000000, *databits* = 8, *stopbits* = 1, *parity* = 0, *timeout* = 100000, *device_id* = 64, *t_period* = 1000, *t_data_su* = 100.

Resource usage and timing is specified after Place and Route.

XILINX® 7-SERIES FPGAS

Resource type	***Artix-7***	***Kintex-7***	***Virtex-7***
Slice Register	268	268	268
Slice LUTs	471	470	469
Block RAM	0	0	0
DSP48	0	0	0
Occupied Slices	162	158	147
Clock freq. (approx)	250 MHz	300 MHz	350 MHz

Revision History

Revision	***Change description***	***Date***
1.0	Initial revision	03/06/2019
1.1	First official release	30/07/2019