SPI to 4-channel UART converter

SLG47921

Abstract

This application shows a Serial Peripheral Interface (SPI) to four channel Universal Asynchronous Receiver-Transmitter (UART) converter implementation using the SLG47920/21 FPGA chip. The functionality was verified by creating a Testbench and observing the output waveforms on the inbuilt GTKWave Software.

This document is supported by design files listed in Reference section.

Contents

1. Introduction

The SPI to 4-channel UART converter described in this application note can be implemented on the SLG47921 chip. A controller device is connected to the converter via SPI and up to four target devices are connected to the converter via four UART channels. The application circuit is shown below:

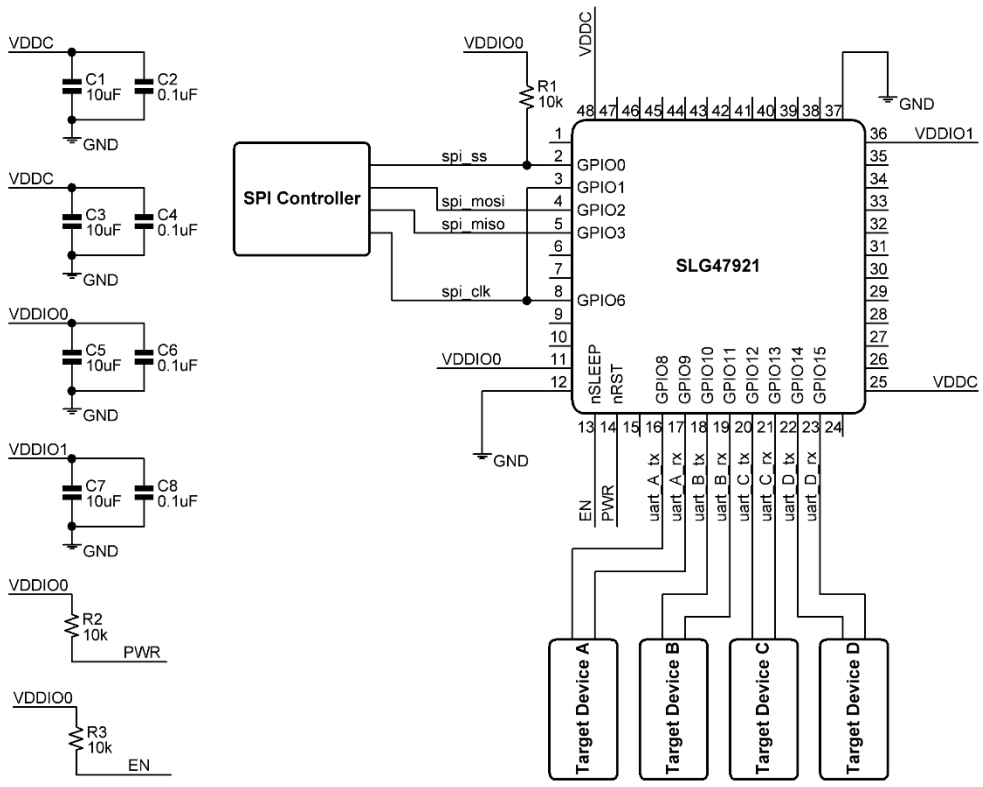

Figure 1. Application circuit

The 10µF and 0.1µF decoupling capacitors should be connected as close as possible to each power pin of the chip. The external pull-up resistors are used to ensure proper configuration of the chip after power-up, which can be configured from an external device (SPI Controller) or internal OTP memory. Once the chip is configured, it goes into functional mode and can operate as an SPI to 4-channel UART converter.

The SPI controller device can receive and send data from/to the required target device by sending the converter an appropriate command byte. The converter supports up to 25MHz clock speed SPI with defined protocol. The SPI protocol is following:

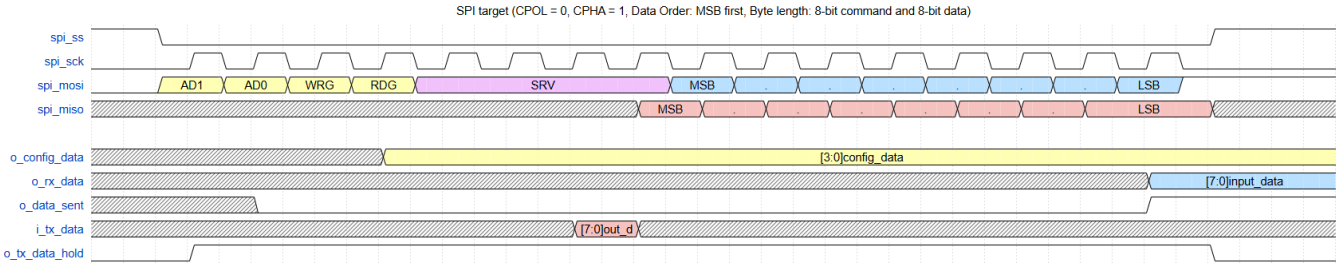

Figure 2. SPI Protocol

The SPI Protocol parameters are next: Clock Polarity (CPOL) = 0, Clock Phase (CPHA) = 0 and the bit transmission starts with MSB first. One SPI message consists of two parts: command byte and data bytes. The command byte contains: 2-bit UART device address definition (AD1, AD0), 2-bit data direction (WRG, RDG) and 4-bit (SRV) which are all required to synchronize command data between different clock domains within the design. A detailed description of the command data bits is shown below:

- AD1, AD0 – 2-bit UART address: ‘00’ – channel A, ‘01’ – channel B, ‘10’ – channel C and ‘11’ – channel D
- WRG, RDG – 2-bit data direction: ‘10’ – only transmit data from SPI to UART, '01' – only receive data from UART to SPI, '11' – full duplex communication and '00' – reserved combination.

Depending on the command byte value, the SPI message can also contain a byte of data sent from the controller, a byte of data received from the appropriate target device via the UART interface or both.

Each of four UART channels have a transmitter and receiver parts which are configured to support 115200 bits/s baud rate. There is a possibility to reconfigure the baud rate to support from 4800 bits/s up to 115200 bits/s by design change. Each UART transmitter and receiver has its own synchronous FIFO (8-sync. FIFOs in total) with size 64 x 8 bit. The data is buffered in this manner, hence preventing data loss caused by the differing transfer rates between SPI and UART. The UART message is simple and the same for transmitting and receiving parts. It consists of the next parts: 1-bit Start Bit, 8-bit Data Frame (the bit transmission starts with MSB first) and 1-bit Stop Bit. The UART Message is shown below:

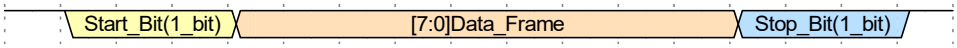

Figure 3. UART Message

The devices connected to the converter via the UART interface can transmit any data to the converter. This data will be stored in the corresponding FIFO memory, which can hold up to 64 bytes of data simultaneously. The controller device connected to the converter via SPI can read data received from any UART channel and previously stored in RX FIFOs by sending the appropriate command byte. This device can also send data to devices connected to the UART channels. At first the data will be stored in the corresponding TX FIFO memory until it is transmitted via UART. Up to 64 bytes of data can be stored at once.

2. Ingredients

- ForgeFPGA SLG47921V IC
- Latest Revision of ForgeFPGA Workshop software
- ForgeFPGA Development Board with Socket Adaptor Board, USB cable and power supply

3. Design Overview

Design structure

Following is the design block diagram of the SPI to 4-channel UART converter:

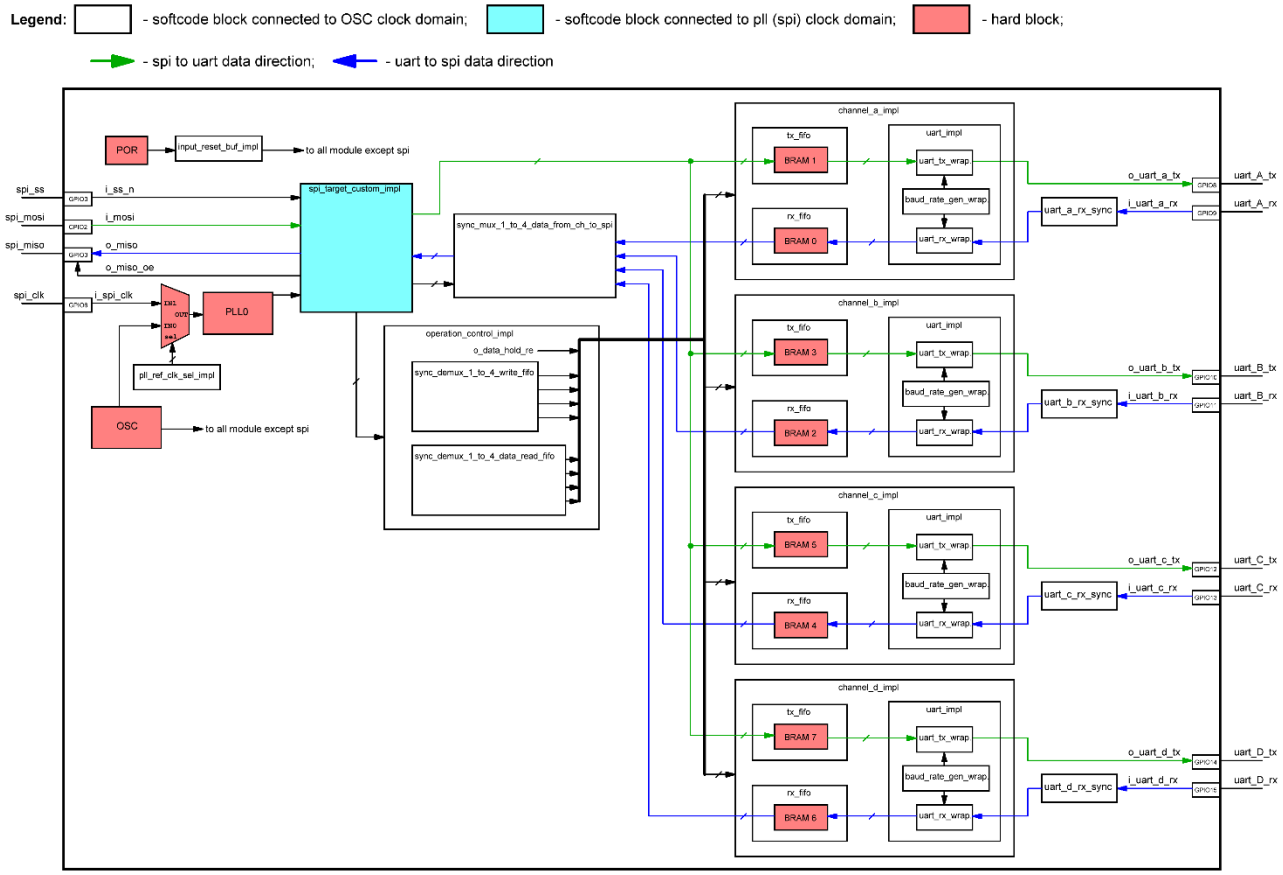

Figure 4. Design Block Diagram

The design of the converter consists of soft-code blocks and hard blocks. The soft-code blocks use FPGA Core logic resources and written using the Verilog language. The hard blocks are physical and are not parts of the FPGA Core. They are connected to the FPGA Core inside the chip by hard-wired connections. The design block diagram is not so detailed and does not include all used blocks and connections. Only the minimum required blocks to understand logic of operation of the design is shown.

The used hard blocks are listed below:

- POR – Power-on reset. Used as global reset for almost all soft-code blocks (except SPI)
- OSC – On-chip Oscillator. Generates 50MHz clock
- PLL1 – Phase-Locked Loop. Operates in bypass mode. Provides ‘spi_clk’ signal to SPI block
- BRAM0...7 – Embedded Block RAM. Used in sync. FIFOs for storing data

The Verilog code of the design consists of the next following modules:

- baud_rate_generator – generates ‘tick’ pulses for UART transmitter and receiver modules.

- channel – contains RX and TX FIFOs and UART transmitter and receiver modules for one channel
- input_reset_buf – input reset bufer
- one_dff – provides PLL input clock source control
- operation_control – generates control signals
- spi_target_custom – SPI target module
- spi_to_4x_uart – top module of the design
- sync_demux_1_to_4 – synchronous 1 to 4 demultiplexer
- sync_fifo_w_bram_custom – synchronous FIFO with using BRAM
- sync_mux_4_to_1 – synchronous 4 to 1 multiplexer
- sync_pin – input signal synchronization
- uart – contains Baud rate generator, UART transmitter and receiver modules
- uart_rx – UART receiver
- uart_tx – UART transmitter

In this design, four independent UART channels have been implemented. In Figure 4, they are named as **channel_a_impl**, **channel_b_impl**, **channel_c_impl** and **channel_d_impl**. Each channel has its own UART transmitter (**uart_tx_wrap**), UART receiver (**uart_rx_wrap**). Also, it has TX FIFO (**tx_fifo**) and RX FIFO (**rx_fifo**) memory blocks. Embedded Block RAM is used to implement 8 independent FIFO memory blocks. So, in the design it divided by 8 parts (**BRAM0…7**).

The **sync_pin** module is used for synchronizing input signals to internal clock. There are four blocks in the design that are used for this purpose: **uart_a_rx_sync**, **uart_b_rx_sync**, **uart_c_rx_sync**, **uart_d_rx_sync**.

In this design two different clock sources are used, the SPI block (named **spi_target_custom_impl** in Figure 4) uses clock signal that comes from the controller device through ‘spi_clk’ (**GPIO6**) pin and PLL0 block (which is in bypass mode). Other modules use the 50MHz clock signal that comes from the **OSC.** The **input_reset_buf_impl** block provides reset signal for modules that use the OSC clock. The **pll_ref_clk_sel_impl** is used for selecting external clock source for PLL.

The SPI target block (**spi_target_custom_impl** in Figure 4) parses the command byte received from the host via the SPI interface. Also, it can receive or transmit data from/to the controller device. According to Figure 2, the SPI starts receiving the data when ‘spi_ss’ (**GPIO0**) goes LOW. The data on the ‘spi_mosi’ (**GPIO2**) pin is shifted out on the falling edge of the ‘spi_clk’ signal and sampled on the rising edge. The output data on the ‘spi_miso’ (**GPIO3**) pin is shifted out on the rising edge of the ‘spi_clk’. Since the ‘spi_clk’ is provided by the controller device and the ‘spi_miso’ – by the converter, there is some propagation delay on the converter SoC (about 25 - 28 nanoseconds) between the rising edge of the ‘spi_clk’ and shifting out the ‘spi_miso’ signal. This propagation time should be enough to safe sample the data bits on the controller device side.

The **sync_mux_1_to_4_data_from_ch_to_spi** block is a four-to-one multiplexer that provides 8-bit parallel data from the channels to the SPI module. It has 2-bit select input. The data on this input is 2-bit UART address definition (AD1, AD0) and updates during each SPI transition.

The **operation_control_impl** block uses four MSB of the command byte (AD1, AD0, WRG, RDG) for communicating (sending or/and receiving the data) with required one of four channels.

4. Verilog Code

The design file associated with this document is available for download (AN-FG-024 SPI to 4-channel UART Converter.ffpga). It contains all the submodules and testbench to successfully run this application note.

Shown below is the UART module of the design. The user can notice is instantiation of both submodules in it

```
module uart #(
  parameter IN_CLK_HZ         = 50_000_000, // input operating frequency (Hz)
  parameter DATA_FRAME        = 8,          // number of data bits (5 ~ 9 bits long)
  parameter BAUD_RATE         = 115_200,    // transmitting speed 4800 - 115200
  parameter OVERSAMPLING_MODE = 16,         // bit offset or overlap
  parameter STOP_BIT          = 1,          // length of stop bit
  parameter LSB               = 1           // determine serial data transfer format
("0" = LSB to MSB, "1" = MSB to LSB)
) (
  input                       i_clk,        // input clock signal
  input                       i_rst,        // input reset signal
  // tx
  input      [DATA_FRAME-1:0] i_tx_data,    // transmit data inputs
  input                       i_tx_start,   // input (rising edge detect) which
enable transmit data
  output                      o_tx,         // tx output carries the output serial
data
  output                      o_tx_done,    // done transmit signal
  // rx
  input                       i_rx,         // rx input carries the input serial data
  output     [DATA_FRAME-1:0] o_rx_data,    // receive data outputs
  output                      o_rx_done     // done signal
);

  wire w_tick;

// FSM module transmit data to UART
  uart_tx #(
    .DATA_FRAME         (DATA_FRAME),
    .BAUD_RATE          (BAUD_RATE),
    .OVERSAMPLING_MODE  (OVERSAMPLING_MODE),
    .STOP_BIT           (STOP_BIT),
    .LSB                (LSB)
  ) uart_tx_wrapper (
    .i_clk              (i_clk),
    .i_rst              (i_rst),
    .i_tx_data          (i_tx_data),
    .i_tx_start         (i_tx_start),
    .i_tick             (w_tick),
    .o_tx               (o_tx),
    .o_tx_done          (o_tx_done)
  );

// FSM module receive data to UART
  uart_rx #(
    .DATA_FRAME         (DATA_FRAME),
    .BAUD_RATE          (BAUD_RATE),
    .OVERSAMPLING_MODE  (OVERSAMPLING_MODE),
```

```
    .STOP_BIT           (STOP_BIT),
    .LSB                (LSB)
  ) uart_rx_wrapper (
    .i_clk              (i_clk),
    .i_rst              (i_rst),
    .i_rx               (i_rx),
    .i_tick             (w_tick),
    .o_rx_data          (o_rx_data),
    .o_rx_done          (o_rx_done)
  );

// Module generate one bit period
  baud_rate_generator #(
    .BAUD_RATE          (BAUD_RATE),
    .OVERSAMPLING_MODE  (OVERSAMPLING_MODE),
    .IN_CLK_HZ          (IN_CLK_HZ)
  ) baud_rate_gen_wrapper (
    .i_clk              (i_clk),
    .i_rst              (i_rst),
    .o_tick             (w_tick)
  );

endmodule
```

5. Simulation Results

The design functionality can be verified by using the ‘spi_to_4x_uart_tb’ test bench file found under the design file (AN-FG-024 SPI to 4-UART Converter). The simulation waveforms are produced by inbuilt GTKWave software. Below are the testbench simulation results:

Figure 5 showcases the external communication input and output signals.

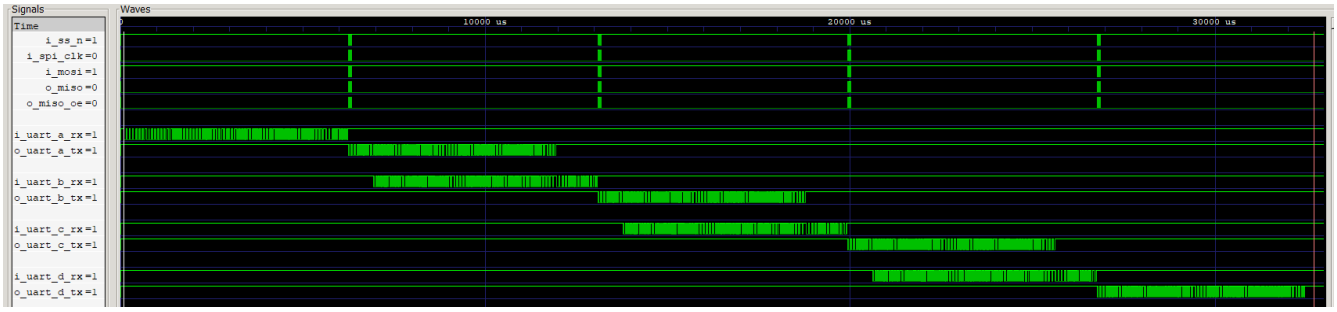

Figure 5. General View of Simulation Waveforms

There are SPI signals:

- i_ss_n – input target select signal
- i_spi_clk – input clock signal
- i_mosi – input controller output target input signal
- o_miso – output controller input target output signal
- o_miso_oe – output o_miso output enable signal

and UART signals:

- i_uart_a_rx – input channel A receive serial data

- o_uart_a_tx – output channel A transmit serial data
- i_uart_b_rx – input channel B receive serial data
- o_uart_b_tx – output channel B transmit serial data
- i_uart_c_rx – input channel C receive serial data
- o_uart_c_tx – output channel C transmit serial data
- i_uart_d_rx – input channel D receive serial data
- o_uart_d_tx – output channel D transmit serial data

The Figure 5 shows the simulation waveforms for all four UART channels. Test sequence for verification of each channel functionality consists of three phases: sending data to the converter through UART (Figure 7), communication with the converter via SPI (Figure 8) and transmitting data from the converter via UART (Figure 11).

The initial phase begins by sending a data byte with the value ‘h01’ at ‘i_uart_a_rx’ input (Figure 6).

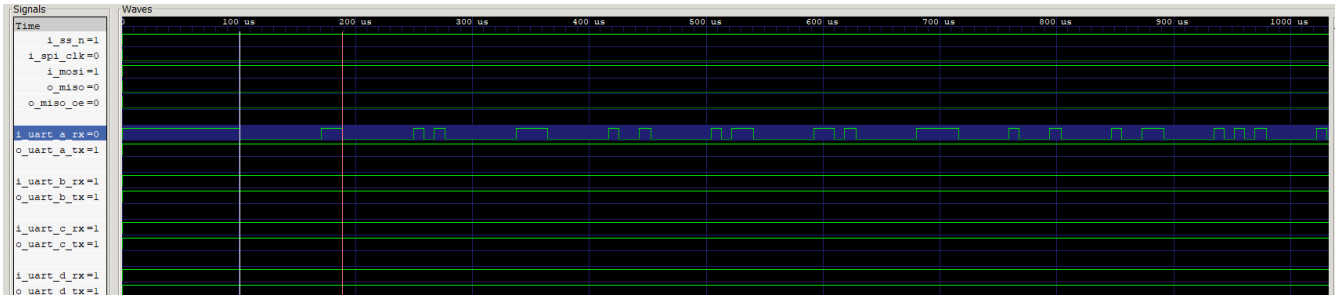

Figure 6. Sending first byte with value ‘h01

Following this, an additional 69 bytes are sent, with each byte’s data being incremented by 1 sequentially. Since the size of the rx sync. FIFO is only 64 bytes, not all data sent via UART in the first phase can be stored in it, but only the first 64 bytes.

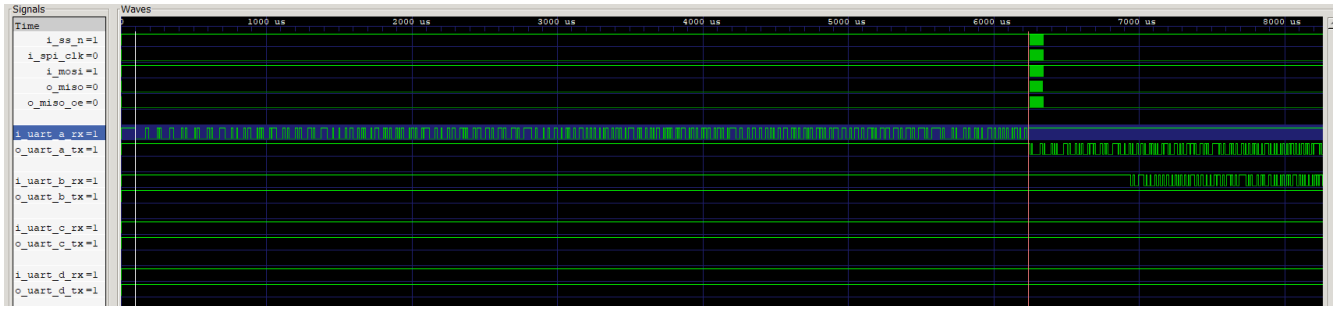

Figure 7. Sending 70 Bytes of Data to the Converter via UART Channel A

The second phase of channel verification is communication with the converter via SPI.

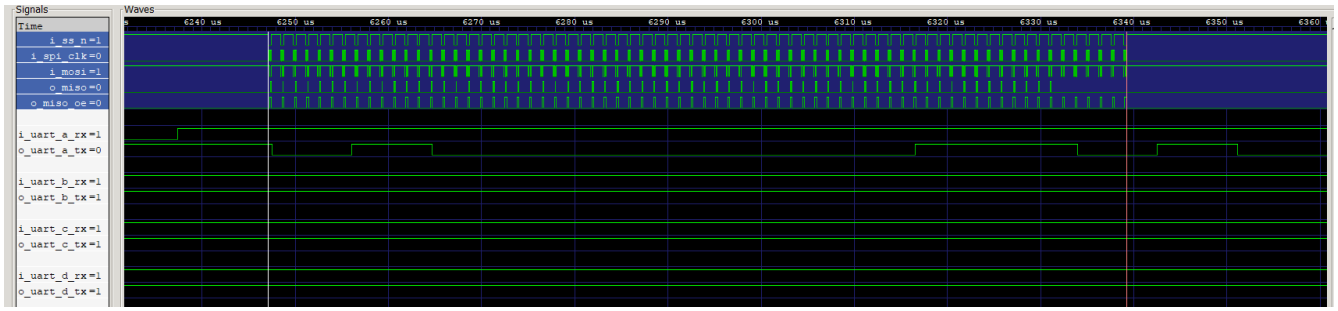

Figure 8. Communication with the Converter via SPI

The first SPI frame contains command and data byte which is sent to the converter on ‘i_mosi’ line and requested data on ‘o_miso’ line (see Figure 8). When the ‘o_miso_oe’ is HIGH, the ‘spi_miso’ (**GPIO3**) is operating as Push-Pull output, otherwise it is at high impedance state.

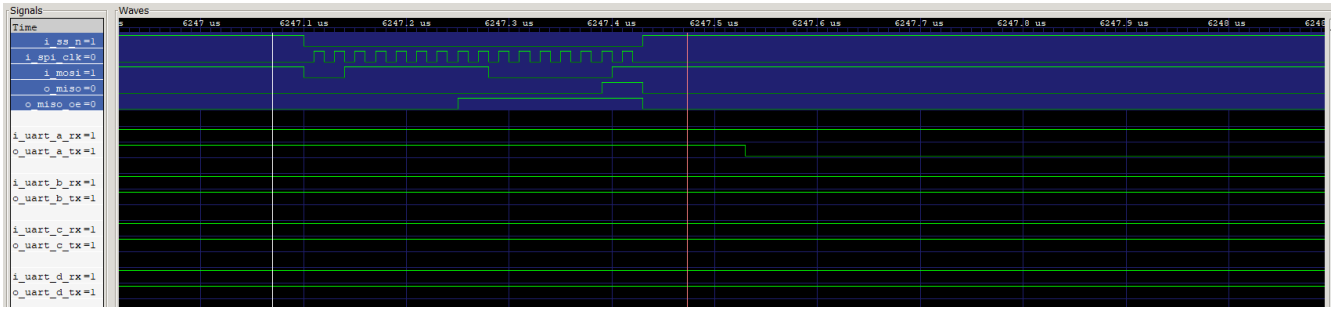

Figure 9. The First Frame Communication Through SPI

The command byte value is ‘h3F or ‘b00111111. This means, that in this frame the controller device requests to write and read data from the device connected to the UART channel A. In the second byte, it sends data ‘h81 and at the same time receives ‘h01 (this data was sent to the converter via UART in the first phase) at ‘o_miso’ line. Subsequently, 69 additional SPI frames are transmitted, with the data being sent and read incrementing by one sequentially. Since the size of the RX sync. FIFO is only 64 bytes, not all 70 bytes of data sent via UART in the first phase was stored in it, but only the first 64 bytes. If the FIFO is empty, the converter will transmit bytes with data equal to ‘h00. This way the RX sync. FIFO size is checked.

The third phase is transmitting data from the converter via UART Channel A. When the converter gets the data from the SPI controller device it starts transmitting this data to the device connected to UART channel A pins. The first byte data is ‘h81 (Figure 10).

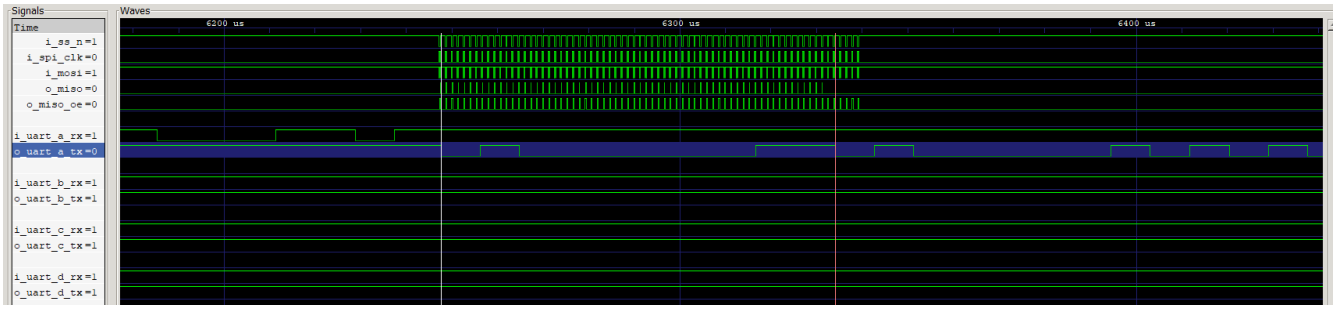

Figure 10. Sending First Byte with value ‘h81

This data is sent to the converter via SPI in the second phase. The converter will continue transmitting the data bytes via UART until the TX FIFO is empty (Figure 11). Since the size of the TX FIFO is only 64 bytes, the converter sends only 65 of 70 bytes of data that the controller sent to the converter during second phase. This way the TX sync. FIFO size is checked.

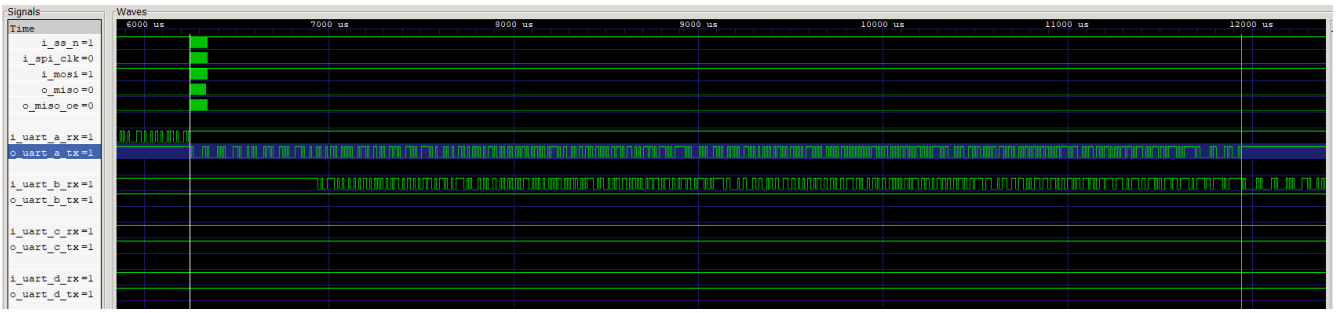

Figure 11. Transmitting Data from the Converter via UART Channel A

The verification of functionality of the other UART channels is the same as channel A.

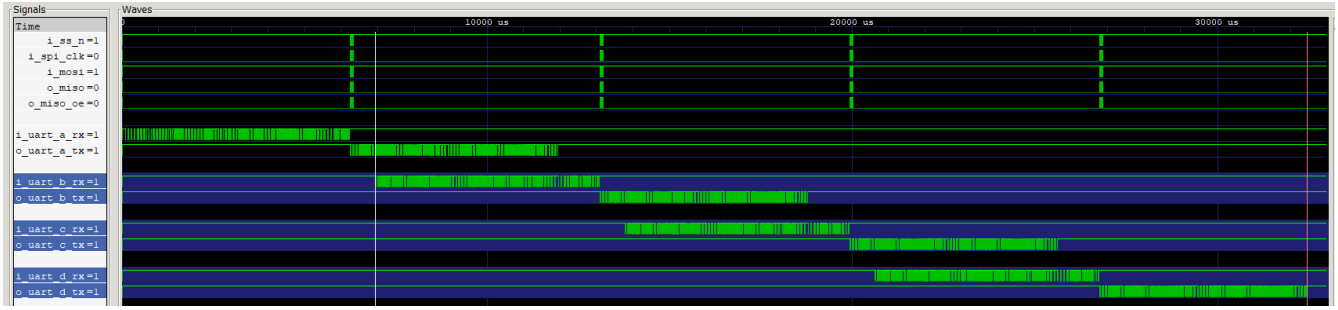

Figure 12. Receiving and Sending Data Through Other UART Channels

6. Logic Analyzer: UART and SPI Data Write and Read Out

The Logic Analyzer in the ForgeFPGA Workshop is used to verify the design functionality on the bench. The UART and SPI communication parts are shown in Figures 13 and 14:

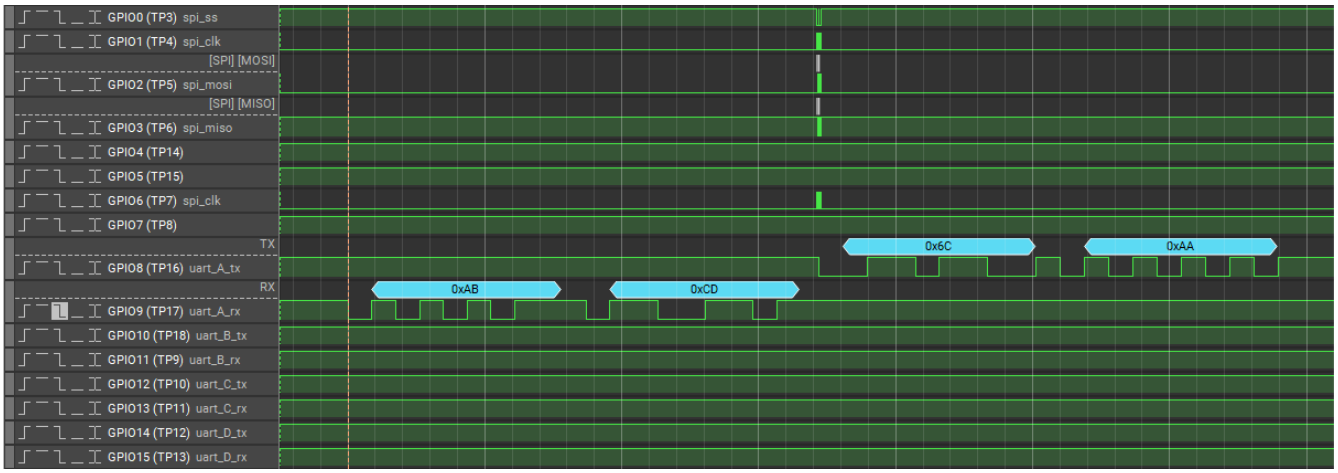

Figure 13. Receiving 0xAB, 0xCD and sending 0x6C, 0xAA data bytes via UART Channel A

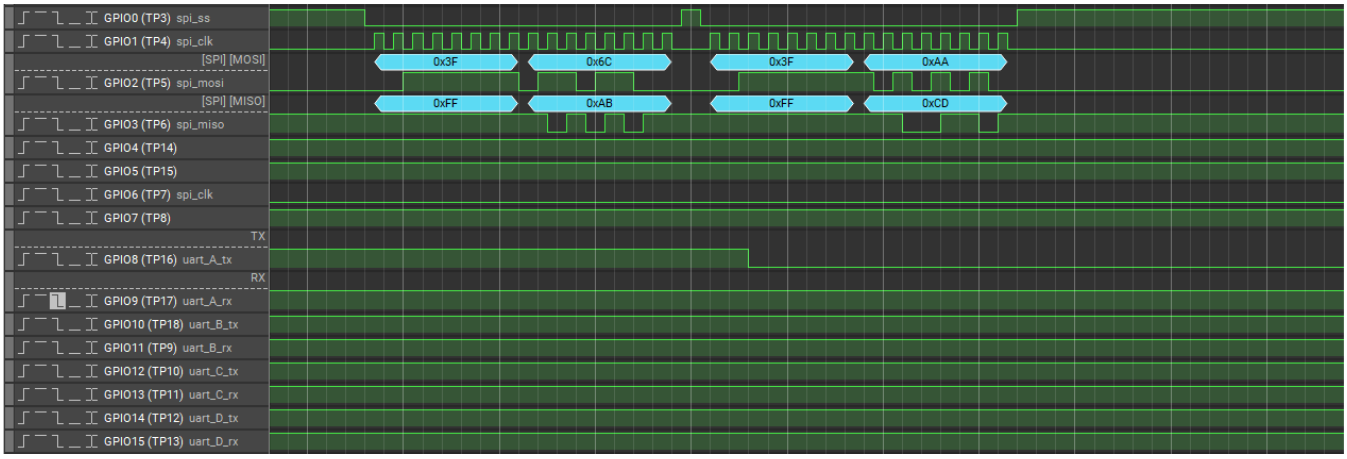

Figure 14. Sending 0x6C, 0xAA and reading 0xAB, 0xCD data bytes to/from UART Channel A via SPI

7. Conclusion

This application note shows how the SLG47921 can be used to implement a Serial Peripheral Interface (SPI) to four channel Universal Asynchronous Receiver-Transmitter (UART) converter.

For more information, contact the ForgeFPGA Business Support Team.

8. Terms and Definitions

CLB	Configuration Logic Block
OTP Memory	One-Time Programmable Memory
FPGA	Field Programmable Gate Array
FPGA Editor	Main FPGA design and simulation window
Go Configure Software Hub	Main window for device selection
ForgeFPGA Workshop	Window Main FPGA project window for debug and IO programming

9. References

For related documents and software, please visit:ForgeFPGA Low-density FPGAs | Renesas.
Download our free ForgeFPGA™ Designer software [1] to open the .ffpga design files [2] and view the proposed circuit design.

[1] Go Configure Software Hub, Software Download and User Guide

[2] AN-FG-024 SPI to 4-channel UART converter.ffpga, ForgeFPGA Design File

[3] ForgeFPGA SLG47920/21 Datasheet, Renesas Electronics

[4] ForgeFPGA Workshop User Guide, Renesas Electronics

10. Revision History

Revision	Date	Description
1.00	Jan 06, 2026	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL

SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Apr 2025)

Corporate Headquarters

TOYOSU FORESIA,3-2-24 Toyosu,

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most

up-to-data version of a document, or your nearest sales

office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas

Electronics Corporation. All trademarks are registered

Trademarks are the property of their respective owners.