UART TO SPI   SPECIFICATION

Author: Dinesh Annayya

dinesha@opencores.org

Table of Contents

PREFACE

SCOPE

This document describes the UART to SPI operation, architecture and interfaces.

REVISION HISTORY

Rev	*Date*	*Author*	*Description*
1.0	*27/09/2012*	*Dinesh Annayya*	*First Draft*

ABBREVIATIONS

SPI - Serial Peripheral Interface

UART- Universal Asynchronous Receiver / Transmitter

INTRODUCTION

The UART to SPI IP Core include a simple command parser that can be used to access an internal bus of SPI via a UART interface. This IP can be used understand the SPI transaction protocol. The internal bus is designed with address bus of 16 bits and data bus of 8 bits. The core implements a very basic UART transmit & receive blocks which share a common baud rate generator and a command parser. The parser supports text mode of command parsing. Text mode commands are designed to be used with hyper terminal software and enable easy access to the internal bus.

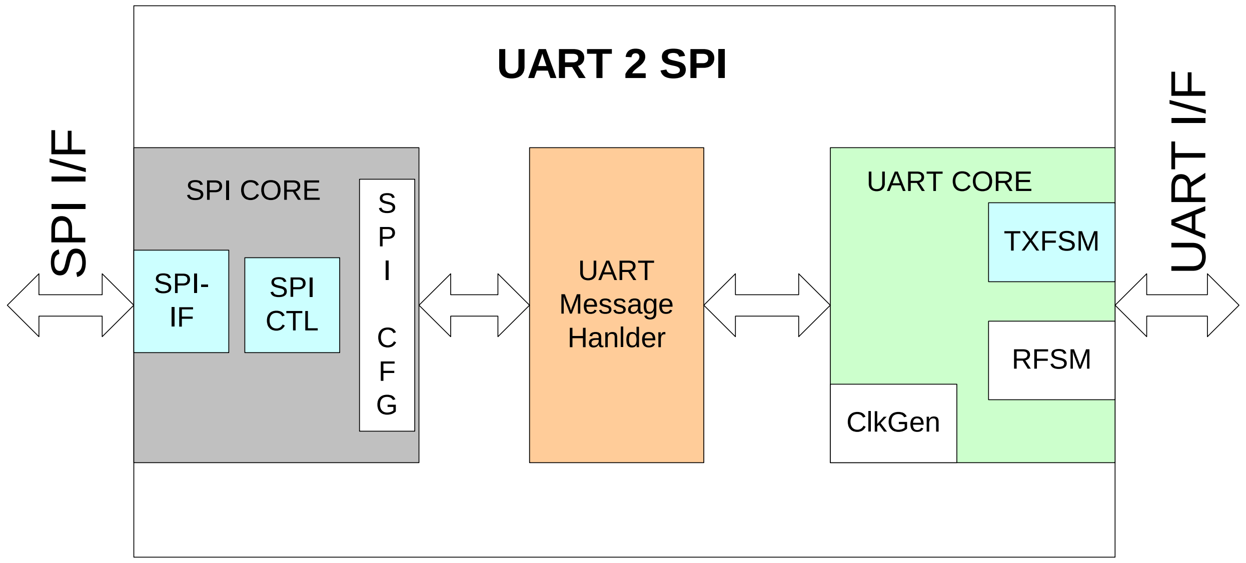
The core is verified with mentor graphics model simulator and cadence nc-simulator.

The following table summarizes the synthesis results of the core for different FPGA families.

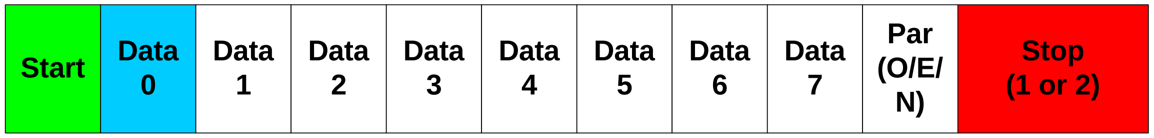
Manufacture	Family	Device Utilization	Fmax
Xilinux	Spartan 3E		

ARCHITECTURE

The core is includes a UART interface module, Message handler and a SPI core. The following figure depicts a block diagram of the core.



UART Frame format



UART CORE translates data between parallel and serial forms. This core include

BAUD-RATE GENERATOR (CLKGEN):

This block includes a clock divider circuit to generate required baud-clock required to sample/drive the UART interface signal. 16x Baud clock formual is = System-Clock (in Hz) / (2 + Configured Baud Register value).

Example: to generate 19200 Baud clock from 50Mhz System clock

50 * 1000 * 1000 / (2 + cfg_baud_16x) = 19200 * 16

Configured Baud Register Value = 0xA0 (160)

1. **RXFSM:** This block monitor the UART receive port (uart_rxd) and decode it into 8 bit data format
2. **TXFSM:** This block translate 8 Bit Data into serial UART bit frame format and drive it into UART Transmit Port (uart_txd).

UART MESSAGE HANDLER:

This block include a state machine and decodes

1. wm <addr> <data> <newline> command into register write towards SPI core with <Addr> <Data>
2. rm <addr> <newline> command into a register read access towards SPI core and transmit back the read data received from SPI interface.

SERIAL PERIPHERAL INTERFACE (SPI)

Introduction:

Serial to Peripheral Interface (SPI) is a hardware/firmware communications protocol developed by Motorola and later adopted by others in the industry. Microwire of National Semiconductor is same as SPI. Sometimes SPI is also called a "four wire" serial bus.

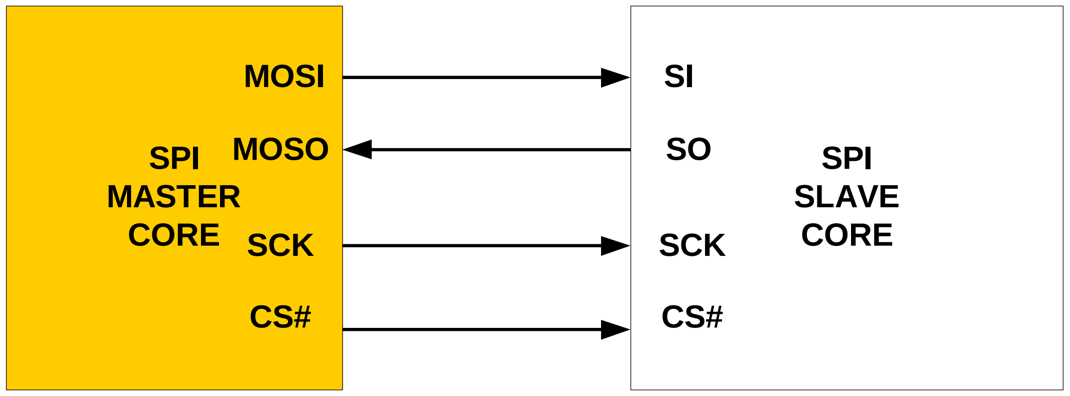
The Serial Peripheral Interface or SPI-bus is a simple 4-wire serial communications interface used by many microprocessor/microcontroller peripheral chips that enables the controllers and peripheral devices to communicate each other. Even though it is developed primarily for the communication between host processor and peripherals, a connection of two processors via SPI is just as well possible.

The SPI bus, which operates at full duplex (means, signals carrying data can go in both directions simultaneously), is a synchronous type data link setup with a Master / Slave interface and can support up to 1 megabaud or 10Mbps of speed.

The peripherals can be a Real Time Clocks, converters like ADC and DAC, memory modules like EEPROM and FLASH, sensors like temperature sensors and pressure sensors, or some other devices like signal-mixer, potentiometer, LCD controller, UART, CAN controller, USB controller and amplifier.

An SPI Bus consists of 4 signal wires. See Figure.

1. Master Out Slave In (MOSI) — MOSI signal is generated by Master, recipient is the Slave.
2. Master In Slave Out (MISO) — MISO signals are generated by Slaves, recipient is the Master.
3. Serial Clock (SCK) — SCLK signal is generated by the Master to synchronize data transfers.
4. Chip Select (CS#) — CS# signal is generated by Master to select individual Slave devices.



How do they communicate?

The communication is initiated by the master all the time. The master first configures the clock, using a frequency, which is less than or equal to the maximum frequency that the slave device supports. The master then select the desired slave for communication by pulling the chip select (SS) line of that particular slave-peripheral to "low" state. If a waiting period is required (such as for analog-to-digital conversion) then the master must wait for at least that period of time before starting to issue clock cycles.

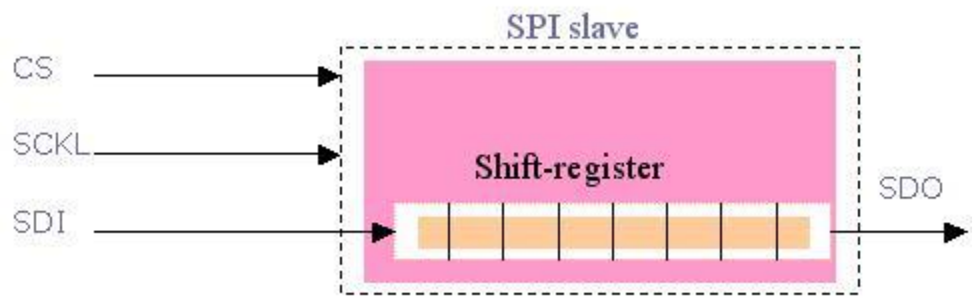
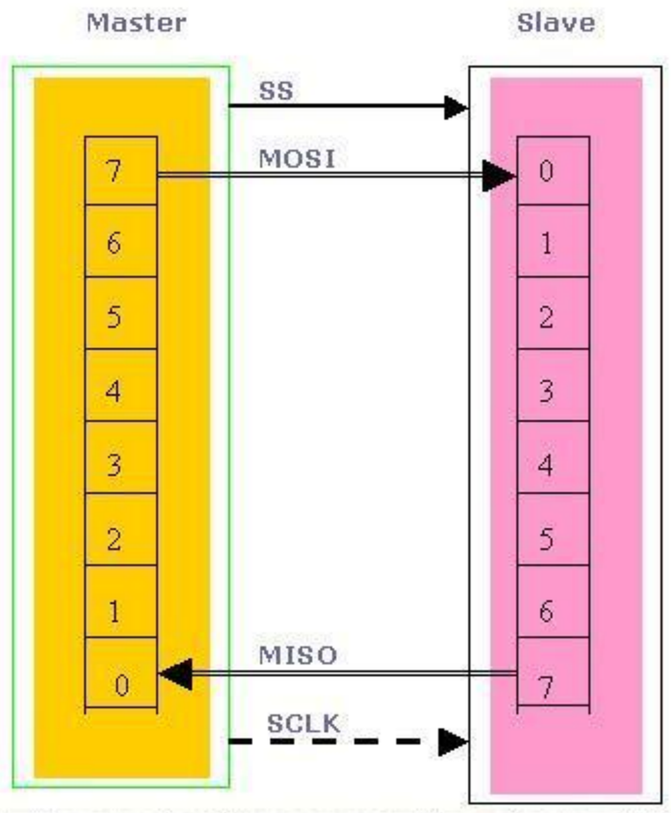


Fig-2 (The shift register used)

The slaves on the bus that has not been activated by the master using its slave select signal will disregard the input clock and MOSI signals from the master, and must not drive MISO. That means the master selects only one slave at a time.

Most devices/peripherals have tri-state outputs, which goes to high impedance state (disconnected) when the device is not selected. Devices without this tri-state outputs cannot share SPI bus with other devices, because such slave's chip-select may not get activated.

A full duplex data transmission can occur during each clock cycle. That means the master sends a bit on the MOSI line; the slave reads it from that same line and the slave sends a bit on the MISO line; the master reads it from that same line.

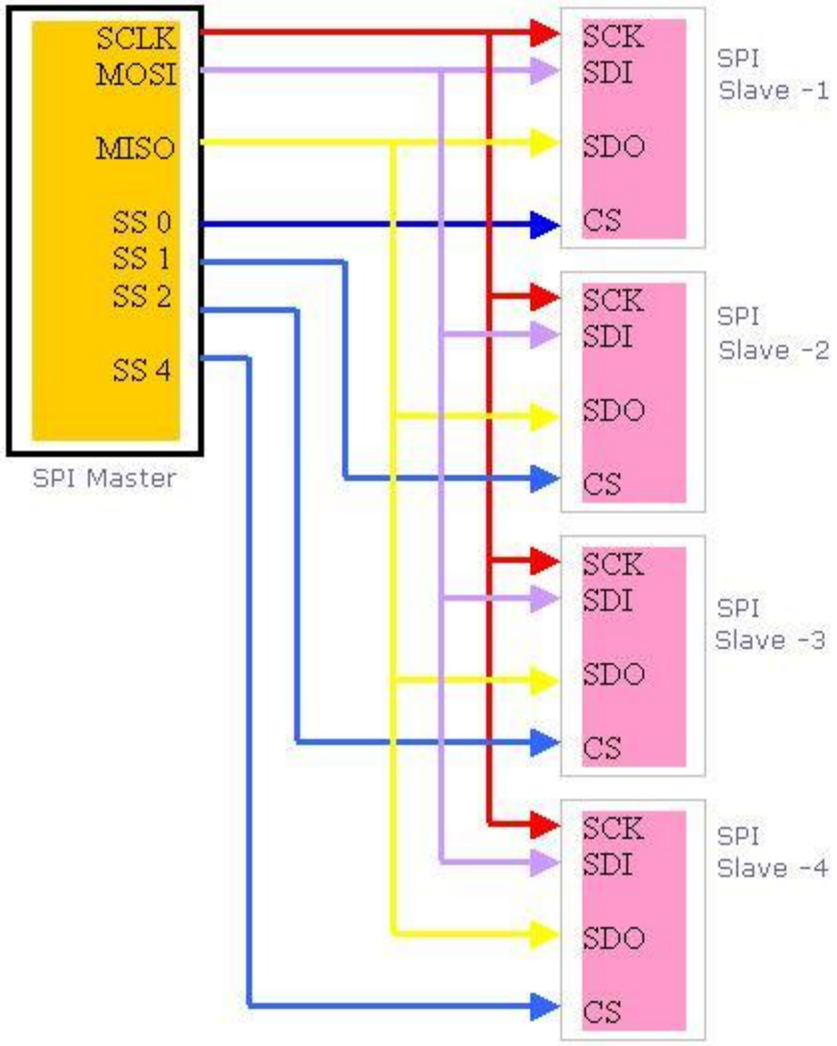


hardware setup for communication using two shift registers

Data are usually shifted out with the MSB first, while shifting a new LSB into the same register. After that register has been shifted out, the master and slave have exchanged their register values. Then each device takes that value and does the necessary operation with it (for example, writing it to memory). If there are more data to be exchanged, the shift registers are loaded with new data and the process is repeated. When there are no more data to be transmitted, the master stops its clock. Normally, it then rejects the slave.

There is a "multiple byte stream mode" available with SPI bus interface. In this mode the master can shift bytes continuously. In this case, the slave select (SS) is kept low until all stream process gets finished.

This is the typical SPI-bus configuration with one SPI-master and multiple slaves/peripherals.



typical SPI-bus configuration with one SPI-master and multiple slaves/peripherals

SPI core includes

1. SPI_CFG: This blocks host a 3 register to manage the SPI control state-machine

Register Address: 0x00

Bits	
[31]	This bit will be set by control unit to initiate the SPI transaction After completion of the SPI transaction, this bit will be reset by the SPI state
[24:23]	target chip select 2'b00 – Chip select- 0 2'b01 – Chip Select-1 2'b10 – Chip Select-2 2'b11 – Chip Select-3
[22:21]	SPI operation type 2'b00 - Write Access 2'b01 - Read Access
[20:19]	SPI transfer size: 2'b00 – 1 Byte Access 2'b01 – 2 Byte Access

	2’b10 – 3 Byte Access 2’b11 – 4 Byte Access
[18:13]	SPI clock period ; Spi-clock = System Clock/ Configured Value
[12:8]	Chip-Select Setup and Hold period
[7:0]	Chip select state during each byte

Register Address : 0x04

Bits	
[31:0]	SPI Write Data

Register Address : 0x08

Bits	
[31:0]	SPI Read Data

2. SPI_CTL: This block manages the state-machine and control the transaction at SPI line interface.
3. SPI_IF : - This block does the serial to parallel 8 bit translation at receive interface and parallel 8 bit to serial translation at transmit interface.

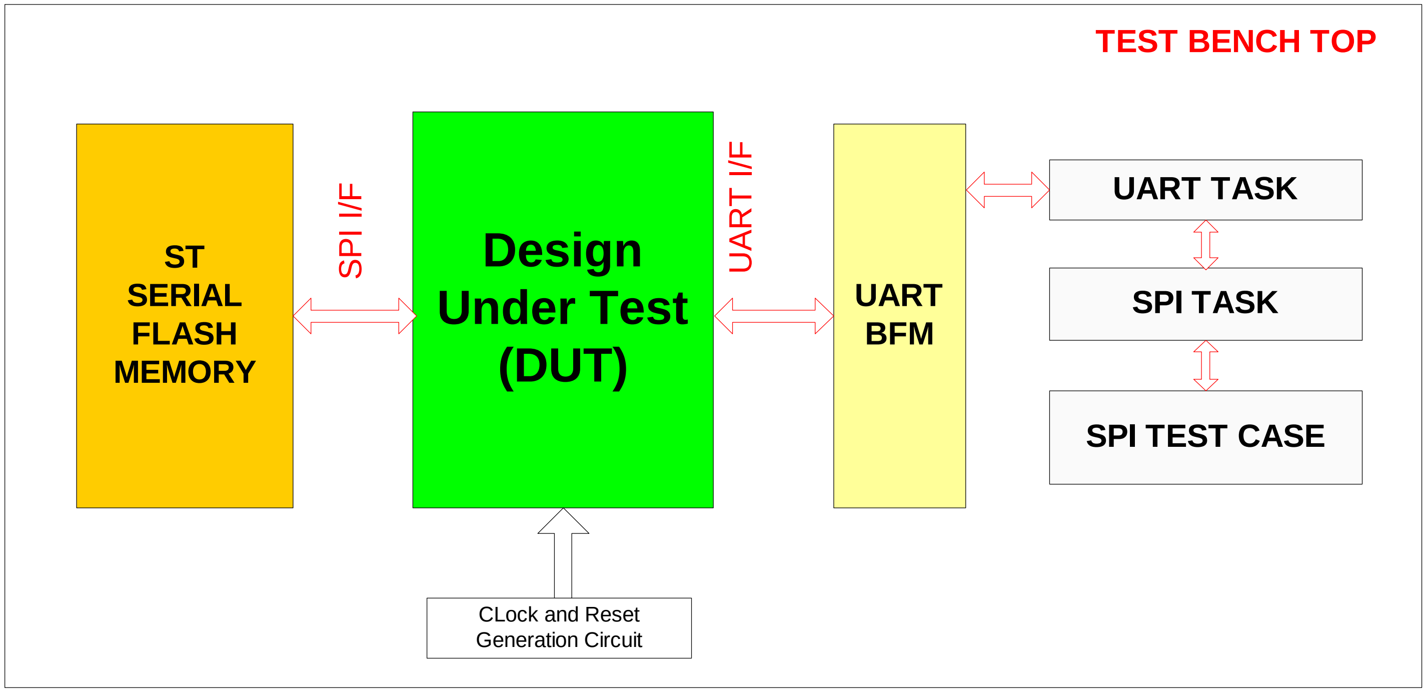
Advantages of SPI

1. Full duplex communication
2. Higher throughput than I^{2}C protocol
3. Not limited to 8-bit words in the case of bit-transferring
4. Arbitrary choice of message size, contents, and purpose
5. Simple hardware interfacing
6. Typically lower power requirements than I^{2}C due to less circuitry.
7. No arbitration or associated failure modes.
8. Slaves use the master's clock, and don't need precision oscillators.
9. Transceivers are not needed.
10. At most one "unique" bus signal per device (CS); all others are shared

Disadvantages of SPI

1. Requires more pins on IC packages than I^{2}C
2. No in-band addressing. Out-of-band chip select signals are required on shared busses.
3. No hardware flow control
4. No slave acknowledgment
5. Multi-master busses are rare and awkward, and are usually limited to a single slave.
6. Without a formal standard, validating conformance is not possible
7. Only handles short distances compared to RS-232, RS-485, or CAN.

VERIFICATION FLOW:



UART Bus Functional Model (BFM): This block manages the UART protocol format translation from 8bit data to serial format and vice-versa.

Configuration parameter:

a. data_bit: transmit and receive data width. 2'b00

2'b00	5 bit data

2’b01	6 bit data
2’b10	7 bit data
2’b11	8 bit data

b. stop_bits: 1’b0 : One bit Stop bit; 1’b1: two bit Stop bit

c. parity_enable: 1’b0: Disable parity; 1’b1: Enable parity.

d. Even_odd_parity: 1’b0: ODD Parity; 1’b1: Even Parity.

e. Time_out: Receive Time out time in terms of baud clock.

UART Receive Interface: This block translate 8 bit data into serial UART bit frame format and drive it into UART receive port(uart_rxd)

UART Transmit Interface: This block monitor the UART transmit interface (uart_txd) and decode it into 8 bit data format.

This block also does the following UART protocol validation and generated corresponding error message.

- Parity Error – Received byte parity does not match with configured parity mode.
- One Stop Bit Error – Missing STOP bit information.
- Two Stop Bit Error – Missing two STOP bit information

UART Task: This module include two task

Reg_write: This task takes address(16 bit) and write data(32 bit) as a input information and drive the UART BFM with “wm <addr> <data> “ command

Reg_read: This task takes address (16 bit) as input information and drive the UART BFM with “rm <addr>” and return the read data received from UART BFM.

SPI Task: This module include various task to configure the on-chip SPI module

spi_write_byte: This task generates necessary sequence of command to transmit send one byte of information serially at SPI interface.

spi_write_dword: This task generates necessary sequence of command to transmit send two byte of information serially at SPI interface.

spi_read_dword: This task generates necessary sequence of command to receive two byte of information serially at SPI interface.

spi_sector_errase: This task generates necessary sequence of command to erase a user defined serial flash memory sector.

spi_page_write: This task generates necessary sequence of command to write 256 location of user defined page of serial flash memory.

spi_page_read_verify: This task generates necessary sequence of command to read and verify the 256 location of user defined page of serial flash memory.

SIMULATION

Run Directory: **trunk/verif/run**

All the compile and simulation are compatible with cadence and model simulator.

Compiling and simulating with Cadence simulator

1. File.f → Includes RTL and TB files list
2. run_nc → run scripts include compile and simulation command.

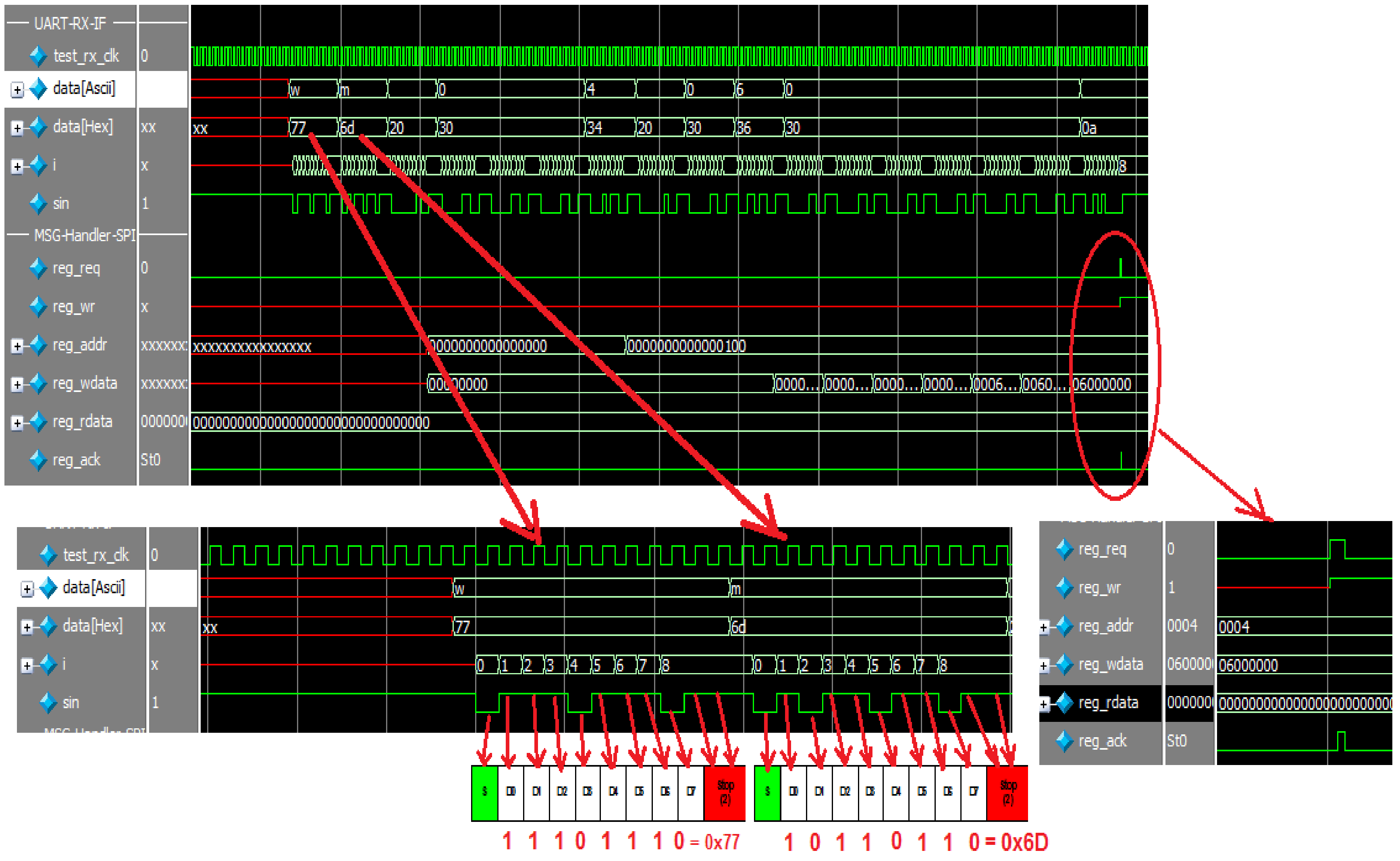
Compiling and simulating with Model Simulator

1. run_modelsim → Includes compile and simulation command.

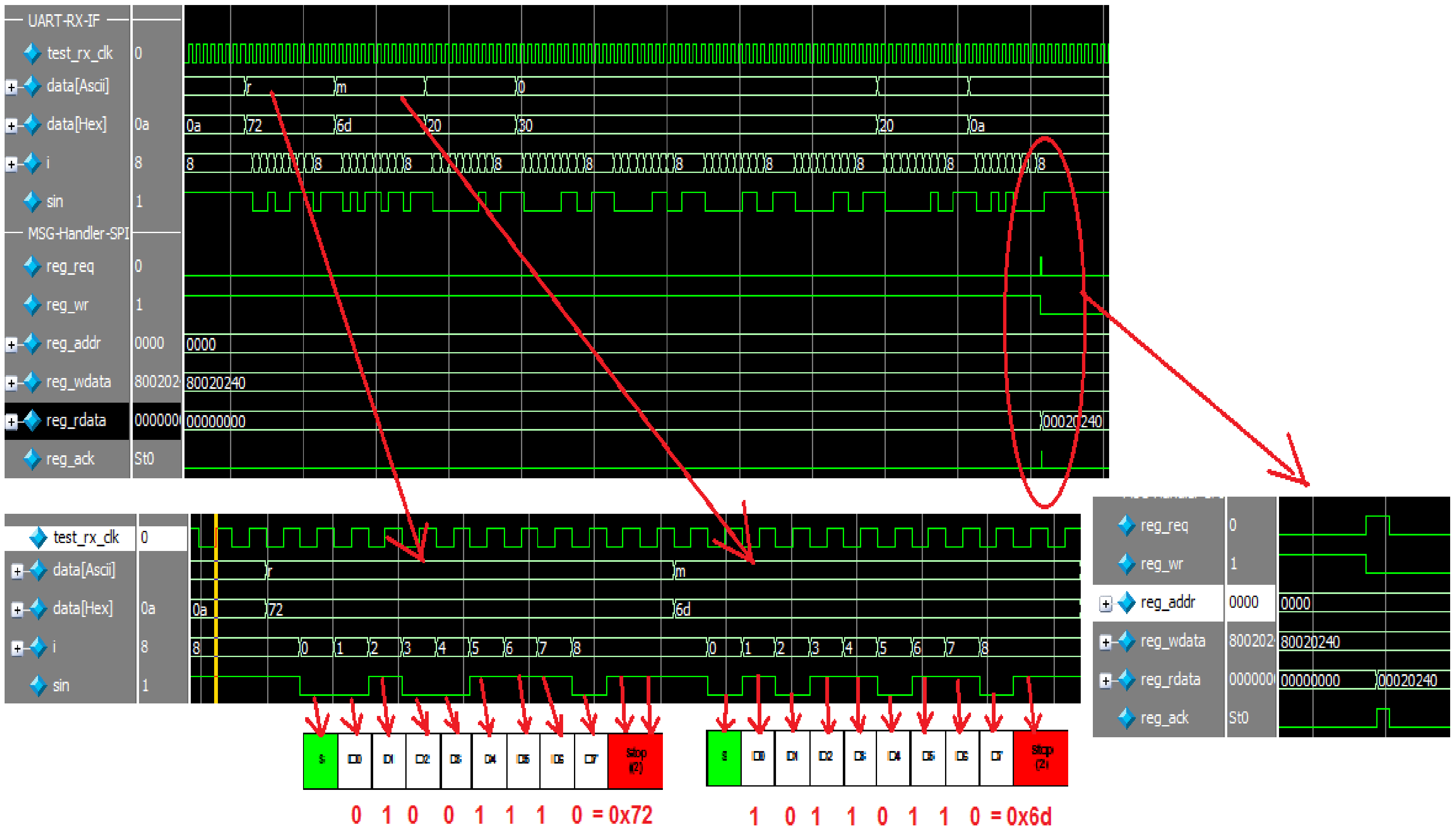
Simulation and compile log file is available under: trunk/verif/log directory

WAVEFORM

UART Receive Simulation for decoding write messages: wm 04 06000000



UART Receive Simulation for decoding read messages: >> rm 0000 # Response: 00020240



UART-RX-IF
test_rx_clk
data[Ascii]
data[Hex]
i
sin
MSG-Handler-SPI
reg_req
reg_wr
reg_addr
reg_wdata
reg_rdata
reg_ack
r
m
0
0a
72
6d
20
30
0000
80020240
00000000
00020240
St0
S
D0
D1
D2
D3
D4
D5
D6
D7
Stop (2)
0 1 0 0 1 1 1 0 = 0x72
1 0 1 1 0 1 1 0 = 0x6d