1. Features

SERIAL_INTF is a companion core that can be used for MVD modulator cores initialization. It allows to set parameters and to read status.

External input interface can be :

- I2C 400KHz slave interface for 8-bit registers access
- UART interface with configurable Baud rates (from 9 600 to 921 600 Bauds), 8-bit, NO parity, 1 STOP bit

2. Applications

SERIAL_INTF core can be used with any MVD modulator core when local CPU is not available.

3. SERIAL_INTF overview

This core can directly drive the 32-bit CPU interface of the MVD modulator cores. It also delivers connection for master SPI, master I2C and GPIO interfaces.

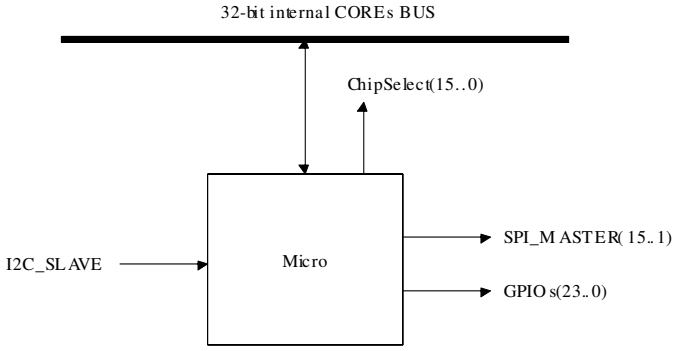


Figure 1- Overview diagram of the SERIAL_INTF Core with I2C input

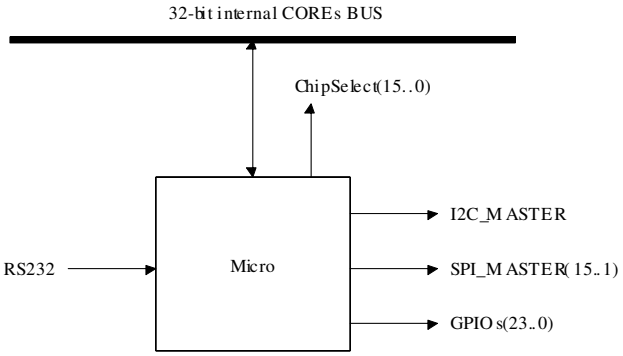


Figure 2- Overview diagram of the SERIAL_INTF Core with UART input

SUMMARY

4. I2C slave interface

I2C slave interface is made of S_SCL input only signal for clock with 400 kHz maximum frequency, and S_SDA signal for bidirectional data bit. The S_SDA output is open collector. The default I2C chip address is 0x18 (WriteAddress) for write and 0x19 (ReadAddress) for read. This interface can drive the 32-bit internal bus, a SPI master output interface and a 24-bit GPIO port.

The I2C component address can be selected over 8 possibilities, a 3-bit address vector S_ADR can be used to define this address.

Bit(7)							Bit(0)
S_ADR(2)	0	0	1	1	S_ADR(1)	S_ADR(0)	0

I2C compliant 8-bit registers read and write are supported.

Start	WriteAddress	Ack	SubAddress	Ack	ByteWrite	Ack	Stop

Figure 3 – Byte write cycle

Start	WriteAddress	Ack	SubAddress	Ack	Start	ReadAddress	Ack	ByteRead	Nack	Stop

Figure 4 – Byte read cycle

SubAddress is used for internal registers addressing. SubAddress from 0x00 to 0x7F are directed to MVD_DVBx core registers (look at MVD_DVBx specific core documentation for registers mapping). SubAddress from 0x80 to 0x8F are used for additional external IOs addressing.

Exemple of programming a MVD CORE register :

Programming 32 bits register with address 0x40 and values 0x12345678, following commands will be sent :

START / 0x18 / ACK / 0x40 / ACK / 0x78 / ACK / STOP
START / 0x18 / ACK / 0x41 / ACK / 0x56 / ACK / STOP
START / 0x18 / ACK / 0x42 / ACK / 0x34 / ACK / STOP
START / 0x18 / ACK / 0x43 / ACK / 0x12 / ACK / STOP

To read this register following dialog will be done :

START / 0x18 / ACK / 0x40 / ACK / START / 0x19 / ACK / 0x78 / NACK / STOP
START / 0x18 / ACK / 0x41 / ACK / START / 0x19 / ACK / 0x56 / NACK / STOP
START / 0x18 / ACK / 0x42 / ACK / START / 0x19 / ACK / 0x34 / NACK / STOP
START / 0x18 / ACK / 0x43 / ACK / START / 0x19 / ACK / 0x12 / NACK / STOP

4.1. Internal bus Core interface

SubAddress from 0x00 to 0x7F are directed to MVD_DVBx core registers. Address 0xX0 is for lower data byte bit(7..0), address 0xX1 is for bit(15..8), address 0xX2 is for bit(23..16) and address 0xX3 is for upper data byte bit(31..24).

Chip select Write/Read : SubAddress 0x80

The MVD_DVBx 32-bit core bus can drive 15 chip selects. This SubAddress is used to set and reset a particular chip select. The chip select must be set active prior to start register access.

NOTA : A write operation must be done before reading operation in order to activate the right CS.
The write operation validates the programmed CS for next read operations.

The ByteWrite and ByteRead format for chip select is :

Bit(7..4) Unused
Bit(3..0) Chip select number from 0 to 15 (0x0 to 0xF)

By default chip select 0 is set active.

4.2. SPI master interface

The highest bit is sent first.

Chip select write only : SubAdress 0x84

SPI master serial interface can drive 15 chip selects. One chip select must be set active before starting a data transfert. Only write accesses are supported.

The ByteWrite format is :

Bit(7..4) Unused
Bit(3..0) Chip select number from 1 to 15 (0x1 to 0xF, 0x0 all chip select are inactive)

By default all chip select are in inactive state.

Data register write only : SubAdress 0x85

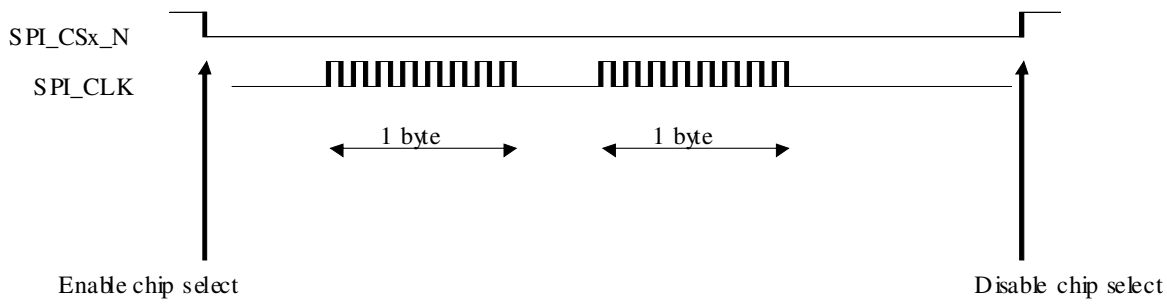


Figure 5 – SPI master example

4.3. GPIO interface

GPIO is a 24-bit general purpose parallel port. Each bit can be configured as output or input with a direction register. During write access only outputs are driven, during read access data is coming from output data register for output ports and from chip pad for input ports.

Direction registers Write/Read :

SubAddress 0x88	direction for GPIO(7..0)	1= output 0=input
SubAddress 0x89	direction for GPIO(15..8)	1= output 0=input
SubAddress 0x8A	direction for GPIO(23..16)	1= output 0=input

By default GPIO(7..0) are outputs , others are inputs.

Data registers Write/Read :

SubAddress 0x8C data for GPIO(7..0)
SubAddress 0x8D data for GPIO(15..8)
SubAddress 0x8E data for GPIO(23..16)

By default GPIO(7..0) are set to 0 level.

5. UART interface

UART interface can be configured with any Baud rates from 9 600 to 921 600 (*for higher values please contact us*) 8-bit data, NO parity and 1 STOP bit. No hardware flow control is available. Transfers are defined as 6 bytes frame for write and 4 bytes frame for read. This interface can drive 32-bit internal bus, SPI master output interface, I2C master 8/16-bit output interface, and 24-bit GPIO port.

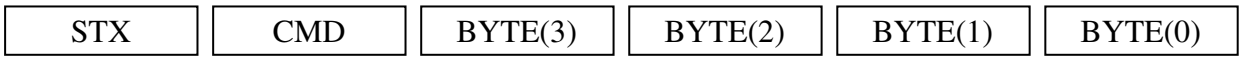


Figure 6 – UART write frame

STX	start of frame code 0x55
CMD	command field
BYTE(3)	upper data byte bit(31..24)
BYTE(2)	data byte bit(23..16)
BYTE(1)	data byte bit(15..8)
BYTE(0)	lower data byte bit(7..0)

Any read command returns a four byte frame.

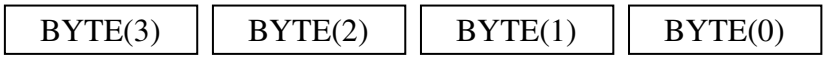


Figure 7 – UART read frame

5.1. Command field :

Bit(7..6) Two bits command
- '00' Chip select number on 32-bit MVD_DVBx core bus
- '01' IOs commands
- '10' Write 32-bit MVD_DVBx core bus
- '11' Read 32-bit MVD_DVBx core bus

Bit(5..0) For '00' command
- Bit(5..4) unused
- Bit(3..0) Chip select number from 0 to 15 (By default chip select 0 is active)

Bit(5..0) For '1x' commands
- Bit(5) must be 0
- Bit(4..0) 32 registers address (Address is for 32-bit register size) shifted on the right of 2 bits (example : **0x04** register address of MVD_DVBx core is **0x01** register address of UART)

Bit(5..0) For '01' command
- Bit(5..4) '00' I2C master
  - '01' SPI master
  - '10' GPIO
  - '11' Reserved
- Bit(3..1) IO parameters (Look at specific IO mode I2C, SPI, GPIO)
- Bit(0) '1' for read access, '0' for write access

5.2. SPI master interface :

The command field header must be : CMD(7..4) = '0101' (0x5)

Notice that only write accesses are supported (CMD(0)='0').

The highest bit is sent first.

Chip select write only :

SPI master serial interface can drive 15 chip selects. One chip select must be set active before starting a data transfer.

The IO parameters field CMD(3..1) format is :

'000' Enable/Disable chip select number (Number is coded in data Byte(0))

The Byte(0) format is :

Bit(7..4) Unused
Bit(3..0) Chip select number from 1 to 15 (0x1 to 0xF, 0x0 all chip select are inactive)

By default all chip select are in inactive state.

Data register write only :

SPI master serial interface support 1, 2, 3 or 4 bytes transfer in a single command.

The IO parameters field CMD(3..1) format is :

'001'	One byte transfer	(Byte(0))
'010'	Two bytes transfer	(Byte(1) & Byte(0))
'011'	Three bytes transfer	(Byte(2) & Byte(1) & Byte(0))
'100'	Four bytes transfer	(Byte(3) & Byte(2) & Byte(1) & Byte(0))

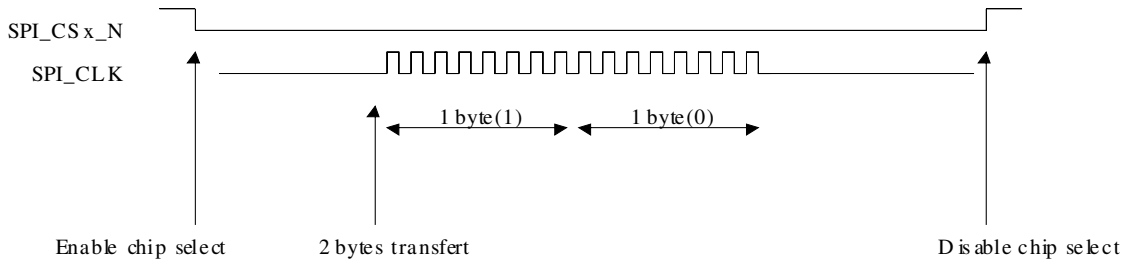


Figure 8 – SPI master example

5.3. GPIO interface :

The command field header must be : CMD(7..4) = '0110' (0x6)

GPIO is a 24-bit general purpose parallel port. Each bit can be configured as output or input with a direction register. During write access only outputs are driven, during read access data is coming from output data register for output ports and from chip pin for input ports.

Direction registers Write/Read :

The IO parameters field CMD(3..1) format is : '000'

Byte(0) direction for GPIO(7..0) 1= output 0=input
Byte(1) direction for GPIO(15..8) 1= output 0=input
Byte(2) direction for GPIO(23..16) 1= output 0=input

By default GPIO(7..0) are outputs , others are inputs.

Data registers Write/Read :

The IO parameters field CMD(3..1) format is : '001'

Byte(0) data for GPIO(7..0)
Byte(1) data for GPIO(15..8)
Byte(2) data for GPIO(23..16)

By default GPIO(7..0) are set to 0 level.

5.4. I2C master interface :

The command field header must be : CMD(7..4) = '0100' (0x4)

The I2C master serial port can generate 8-bit or 16-bit read and write transfers.

The IO parameters field CMD(3..1) format is :

'000' for 1 byte transfer 8-bit
'001' for 2 bytes transfer 16-bit

CMD(0)'1' for read access, '0' for write access

The data format is :

BYTE(3) Write Address of slave component on I2C bus
BYTE(2) Register SubAddress of slave component on I2C bus
BYTE(1) Upper data byte in 16-bit transfer (unused in 1 byte transfer)
BYTE(0) Lower data byte in 16-bit transfer or data byte in 8-bit transfer

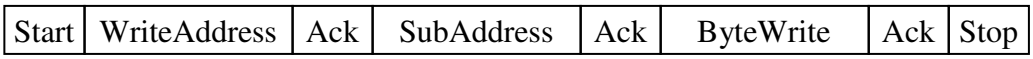


Figure 9 – I2C master 8-bit write transfer

Ex : 0x55 0x40 0x18 0x40 0x00 0x34 (UART COMMAND)

Figure 10 – I2C master 8-bit read transfer

Ex : 0x55 0x41 0x18 0x40 0x00 0x00 (UART COMMAND)

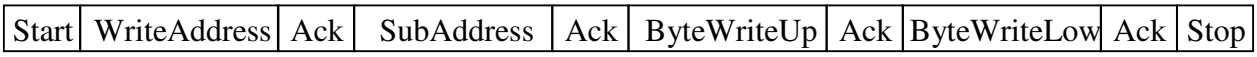


Figure 11 – I2C master 16-bit write transfer

Ex : 0x55 0x42 0x18 0x40 0x12 0x34 (UART COMMAND)

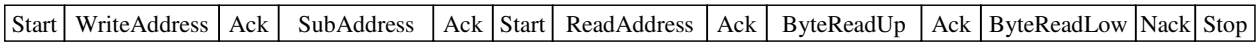
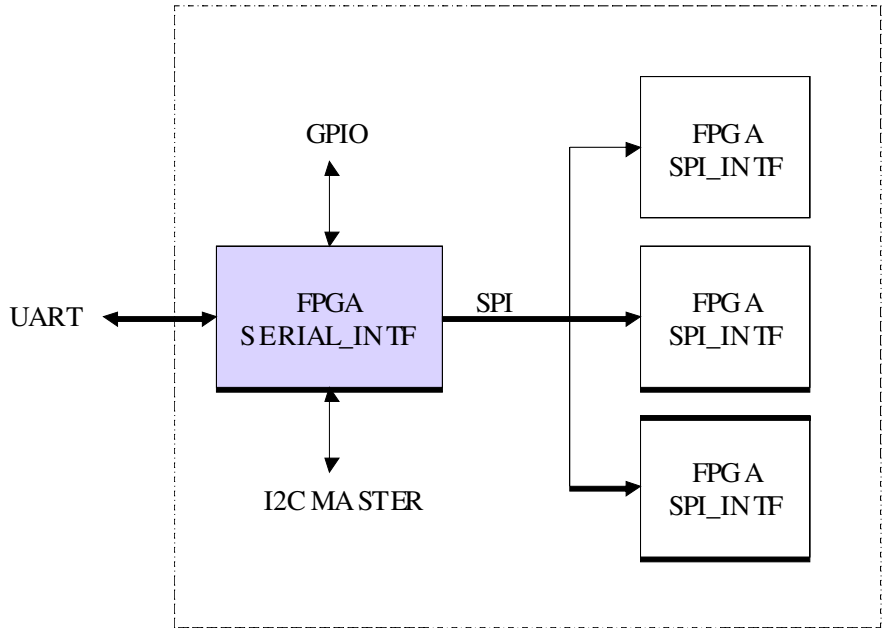


Figure 11 – I2C master 16-bit read transfer

Ex : 0x55 0x43 0x18 0x40 0x00 0x00 (UART COMMAND)

6. Multi FPGA SPI configuration



The SPI master interface of SERIAL_INTF can be used to read and write 32-bit registers inside SPI slave FPGAs containing MVD SPI_INTF core.

Only the UART serial line source is supported (No transfer from I2C slave interface are available).

The SPI frame format is the following (With high bit sent first) :

SPI write register	0	ADDR	BYTE_3	BYTE_2	BYTE_1	BYTE_0	DUMMY
SPI read register	1	ADDR	DUMMY	BYTE_3	BYTE_2	BYTE_1	BYTE_0

ADDR field : 7-bit

Bit(6..5)	'00'	Internal 32-bit bus
	'01'	Chip select register (Byte_0 for chip select number from1 to 15)
	'1x'	Reserved

Bit(4..0) 32-bit register address shifted on the right of 2 bits (example : **0x04** register address of MVD_DVBx core is **0x01** register address of UART)
(Look at specific core documentation for registers mapping)

DUMMY :

8-bit clock time data is unused, the master must send the clock signal.

BYTE :

Byte_3 is for register bit(31..24)
Byte_2 is for register bit(23..16)
Byte_1 is for register bit(15..8)
Byte_0 is for register bit(7..0)

The UART master can use the following sequence to generate the transfers.

Write sequence example :

0x55, 0x50, 0x00, 0x00, 0x00, 0x01	to activate SPI chip select 1
0x55, 0x52, 0x00, 0x00, 0x00, 0x05	one byte transfer for write address
0x55, 0x58, 0x12, 0x34, 0x56, 0x78	four bytes transfer for write data
0x55, 0x52, 0x00, 0x00, 0x00, 0x00	one byte transfer for dummy byte
0x55, 0x50, 0x00, 0x00, 0x00, 0x00	to deactivate SPI all chip select

Read sequence example :

0x55, 0x50, 0x00, 0x00, 0x00, 0x02	to activate SPI chip select 2
0x55, 0x54, 0x00, 0x00, 0x85, 0x00	two bytes transfer for read address and dummy byte
0x55, 0x58, 0x00, 0x00, 0x00, 0x00	four bytes transfer for read data
0x55, 0x50, 0x00, 0x00, 0x00, 0x00	to deactivate SPI all chip select

Notice that the SPI chip select number is coded in Byte_0 on bit(3..0) and can be from 1 to 15. The 0 value deselect all SPI slaves.

The address byte contains the transfer direction on bit 7 (0 for a write or 1 for a read).

7. Default initialization provided as option

The SERIAL_INTF core can be upgraded to implement static I2C or SPI configuration on external components.

8. Port definition

```
entity SERIAL_INTF is port (
   RST          : in  std_logic;  -- reset
   RST_CPU      : in  std_logic;  -- Reset pin for CPU
   CLK          : in  std_logic;  -- CPU clock
   -- CPU clock frequency in KHz
   FREQUENCY_i : in  std_logic_vector(19 downto 0) := (others=>'0');
   -- Baud Rate value
   BAUDRATE_i  : in  std_logic_vector(19 downto 0) := (others=>'0');
   -- External Bus
   CPU_ADR_o    : out std_logic_vector(6 downto 0);
   CPU_DATAW_i : in  std_logic_vector(31 downto 0);
   CPU_DATAR_o : out std_logic_vector(31 downto 0);
   CPU_RD_o     : out std_logic;
   CPU_WR_o     : out std_logic_vector(3 downto 0);
   CPU_CS_o     : out std_logic_vector(15 downto 0);
   -- GPIOs
   GPIO_IN_i    : in  std_logic_vector(23 downto 0) := (others=>'0');
   GPIO_OUT_o   : out std_logic_vector(23 downto 0) := (others=>'0');
   GPIO_EN_o    : out std_logic_vector(23 downto 0) := (others=>'0');
   -- SPI master serial bus
   I_CLK_o      : out std_logic;
   I_MOSI_o     : out std_logic;
   I_MISO_i     : in  std_logic := '0';
   I_CS_N_o     : out std_logic_vector(15 downto 1);
   -- I2C master serial bus
   M_SCL_o      : out std_logic;
   M_SDA_IN_i   : in  std_logic;
   M_SDA_OUT_o : out std_logic;
   M_SDA_EN_o   : out std_logic;
   -- I2C slave serial bus
   S_ADR_i      : in  std_logic_vector(2 downto 0) := (others=>'0');
   S_SCL_i      : in  std_logic := '0';
   S_SDA_i      : in  std_logic;
   S_SDA_o      : out std_logic;
   S_SDA_EN_o   : out std_logic;
   -- UART
   RXD_i        : in  std_logic := '0';
   TXD_o        : out std_logic);
end SERIAL_INTF;
```

RST and RST_CPU are asynchronous reset inputs. They can be connected together.

The FREQUENCY_i input vector must be fixed at : FREQUENCY=Integer value(CLK_frequency(in KHz))

Example : For a clock frequency of 27 MHz the value should be defined as the following:

```
FREQUENCY_i <=  conv_std_logic_vector(27000,20);
```

The BaudRate_i input vector must be defined in Bauds as the following:

```
BAUDRATE_i  <=  conv_std_logic_vector(115200,20);
```

or

```
BAUDRATE_i  <=  conv_std_logic_vector(921600,20);
```

9. SERIAL_INTF core files and deliverables

9.1. Core file

SERIAL_INTF.ngc : Core netlist for ISE
SERIAL_INTF.edif : Core netlist for Vivado

9.2. Implementation examples

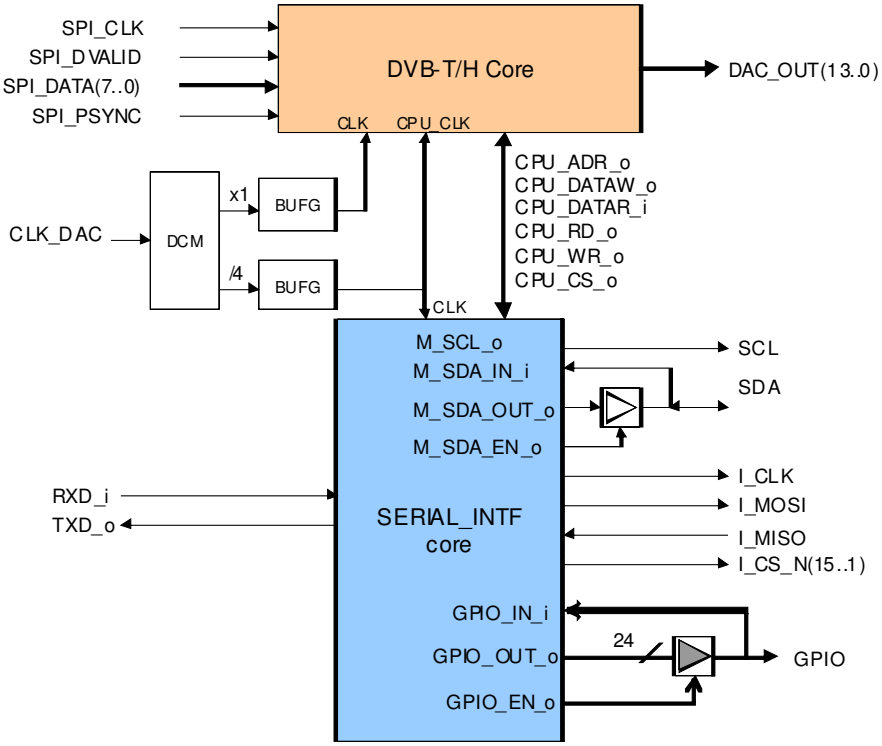


Figure 12 – SERIAL_INTF example with MVD DVB-T core configured via UART

In this case, SDA implementation is as follow :

```
SDA <= SDA_OUT_o when SDA_EN_o = '1' else 'Z';
SDA_IN_i <= SDA;
```

Where SDA is the FPGA output pin.

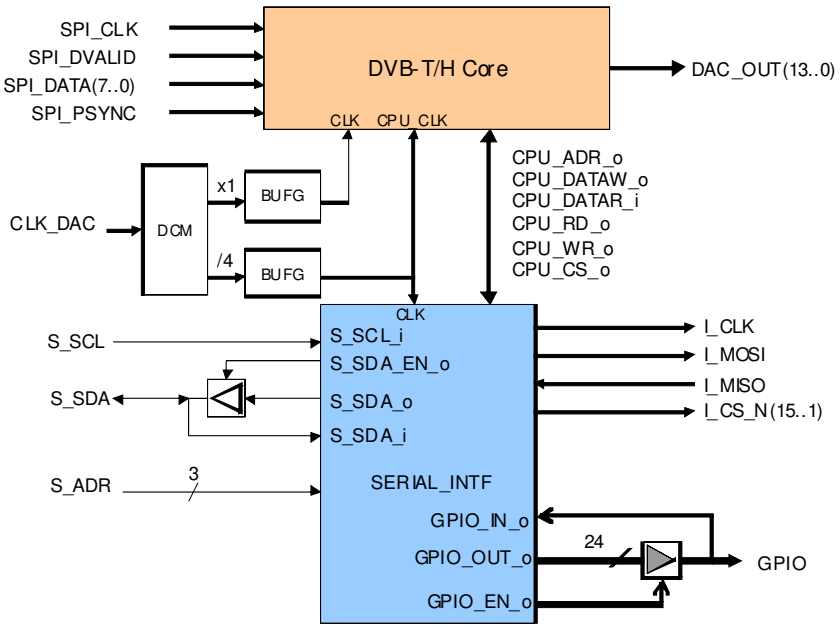


Figure 13 – SERIAL_INTF example with MVD DVB-T core configured via I2C

In this case, SDA implementation is as follow :

```
S_SDA    <= S_SDA_o when S_SDA_EN_o = '1' else 'Z';
S_SDA_i <= S_SDA;
```

Where S_SDA is the FPGA output pin.

10. Resource Utilization

	Slices	BRAMs	MULT or DSP48	BUFG
Spartan-6 TM	**280**	**1**	**0**	**1**
7 Series	**260**	**1**	**0**	**1**

11. Ordering information and related cores

Designation
MVD_SERIAL_INTF_NET

VHDL source code : can be delivered as an option under NDA and other specific clauses.

Related cores : Cable Modulator J83B, DVB-C, DVB-S, DVB-T/H, IP-TV cores

12. Recommanded tools

The MVD Serial Interface is available for implementation on ISE 14.7 and Vivado® Xilinx tools.

13. Tcl Software package

The SERIAL_INTF core can be delivered with a Tcl library package. This TCL library allows to control the SERIAL_INTF core by the way of its RS232 interface thanks to a computer serial COM interface.
A complete set of script examples is provided to give program examples for controlling companion CORES. This paragraph details required software and existing commands for the serial interface Tcl package.

13.1. Tcl Software components

The MvdXilinxTclLib is a complete package of Libraries that can be used with xilinx software tools, such as PlanAhead™ or Vivado™, or in a standalone mode with ActiveState Tcl software which can be easily downloaded on internet.

The TCL Serial Interface library is a part of the MvdXilinxTclLib and needs the installation of the following software if Xilinx tools are not available :

"ActiveState ActiveTcl 8.5.11.0 (64-bit or 32 bit)" Software or later

The MvdXilinxTclLib folder must be copied into the lib folder of the Tcl software installation directory:

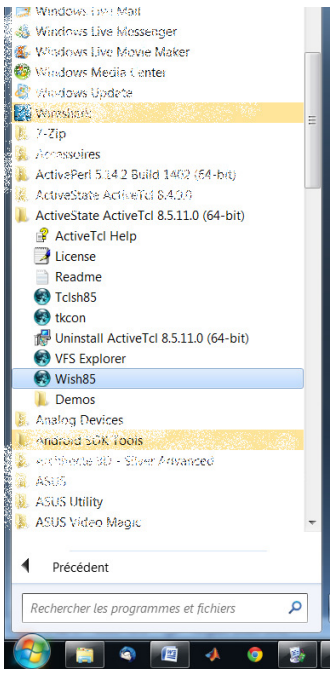
installation_dir/Tcl/Lib/MvdXilinxTclLib

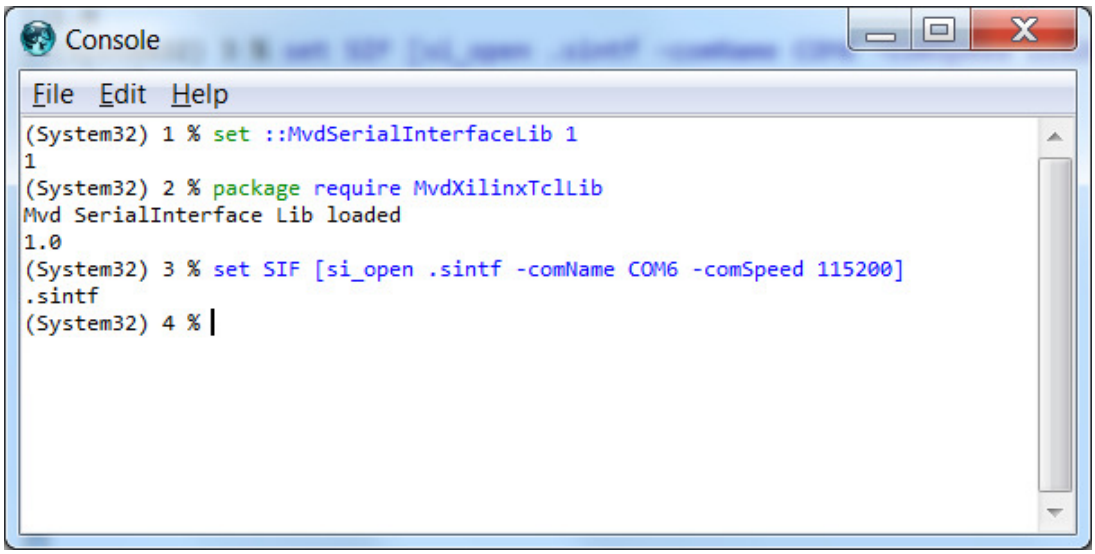
This folder must contains at least 3 files for Serial Interface functionalities :

pkgIndex.tcl
MvdSerialInterfaceLib.dll
TclSerialInterfaceAPI_V1.tcl

13.2. Tcl Application

TCL application to use is wish85.exe :





13.3. Loading

Following lines allow to load the communication package and to open the uart link with the serial interface.

```
 (System32) 1 % set ::MvdSerialInterfaceLib 1
1
(System32) 2 % package require MvdXilinxTclLib
Mvd SerialInterface Lib loaded
1.0
(System32) 3 % set SIF [si_open .sintf -comName COM6 -comSpeed 115200]
.sintf
```

```
where si_open is the command to create a serial interface widget.
parameters are as follow :
comName       : COM port ID connected to the serial interface
comSpeed      : Baudrate of the link (can be 9600, 57600 or 115200 according to your
configuration)
```

Once Serial Interface widget created, all API functionalities can be managed by a set of commands. Two ways can be used :

By the SIF variable: - $SIF.getType (SIF is an example and can be other name !)
By the id of the widget : - .sintf.getType

Both call to the API functions will have the same results.

13.4. TCL low level Functions

13.4.1. .getType

This function returns 0 and allows to identify the serial link type which corresponds to Serial Interface.

```
(Notepad++) 3 % .sintf.getType
0
```

13.4.2. .getCom

This function returns 1 if status link is ok, 0 otherwise.

```
(Notepad++) 2 % .sintf.getCom
1
```

13.4.3. .setVerbose

When verbosity is enable, the command sent to the UART is displayed into the console window.

```
(Notepad++) 4 % .sintf.setVerbose
wrong args: should be ".sintf.setVerbose 0|1"
```

13.4.4. .getVerbose

Returns the current verbosity configuration.

```
(Notepad++) 5 % .sintf.getVerbose
0
```

13.4.5. .setSPICS

Allows to set the next addressed SPI component.

```
(Notepad++) 6 % .sintf.setSPICS
wrong # args: should be ".sintf.setSPICS CS"

CS     : CS of the next SPI component to address

 (Notepad++) 14 % .sintf.setSPICS 0
Command Sent : 0x55  0x50 0x00 0x00 0x00 0x00
```

13.4.6. .writeSPI

Allows to send SPI Bytes commands (4 bytes mode only)

```
wrong # args: should be ".sintf.writeSPI WORD0 WORD1 WORD2 WORD3"
(Notepad++) 12 % .sintf.writeSPI 0 1 2 3
(Notepad++) 19 % .sintf.writeSPI 0 1 2 3
Command Sent : 0x55  0x58 0x03 0x02 0x01 0x00
```

13.4.7. .setGPIODirection

Allows to set each 24 bits direction of the GPIO bus

```
(Notepad++) 21 % .sintf.setGPIODirection
wrong # args: should be ".sintf.setGPIODirection DIR0 DIR1 DIR2"
(Notepad++) 23 % .sintf.setGPIODirection [expr 0xAA] [expr 0x55] [expr 0xCC]
Command Sent : 0x55  0x60 0x00 0xCC 0x55 0xAA
```

13.4.1. .getGPIODirection

Allows to read each 24 bits direction status of the GPIO bus

```
(Notepad++) 21 % .sintf.getGPIODirection
 (Notepad++) 23 % .sintf.getGPIODirection
Command Sent : 0x55  0x61 0x00 0x00 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.4.2. .writeGPIO

This command sends write information on the GPIO bus

```
(Notepad++) 24 % .sintf.writeGPIO
wrong # args: should be ".sintf.writeGPIO DIR0 DIR1 DIR2"
(Notepad++) 25 % .sintf.writeGPIO 1 2 3
Command Sent : 0x55  0x62 0x00 0x03 0x02 0x01
```

13.4.3. .readGPIO

This command reads GPIO values on the 3 GPIO bytes (take care to the GPIO direction settings)

```
(Notepad++) 24 % .sintf.readGPIO
Command Sent : 0x55  0x63 0x00 0x00 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.4.4. .writeI2C

This command sends write information on the I2C bus

```
(Notepad++) 27 % .sintf.writeI2C
wrong # args: should be ".sintf.writeI2C COMP_ADDR REG_ADDR WORD0 WORD1"

COMP_ADDR     : Address of the I2C component
REG_ADDR      : Address of the register to write in.
WORD0  : First byte to write
WORD1  : Second byte to write

(Notepad++) 28 % .sintf.writeI2C 0 10 255 128
Command Sent : 0x55  0x42 0x00 0x0A 0x80 0xFF
```

13.4.5. .readI2C

Returns value read on the I2C interface.

```
(Notepad++) 29 % .sintf.readI2C
wrong # args: should be ".sintf.readI2C COMP_ADDR REG_ADDR BYTE_LENGTH"
COMP_ADDR     : Address of the I2C component
REG_ADDR      : Address of the register to write in.
BYTE_LENGTH   : Number of byte to read

(Notepad++) 30 % .sintf.readI2C 0 10 2
Command Sent : 0x55  0x43 0x00 0x0A 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.4.6. .setCPUCS

Set the component CS of the next CPU interface component to address.

```
(Notepad++) 31 % .sintf.setCPUCS
wrong # args: should be ".sintf.setCPUCS CS"
CS     : CS of the next SPI component to address

(Notepad++) 32 % .sintf.setCPUCS 4
Command Sent : 0x55  0x04 0x00 0x00 0x00 0x04
```

13.4.7. .writeCPU

Writes 4 bytes on the 32 bit CPU interface.

```
(Notepad++) 33 % .sintf.writeCPU
wrong # args: should be ".sintf.writeCPU ADDRESS WORD0 WORD1 WORD2 WORD3"
ADDRESS       : Address of the register in the component
WORD0         : First byte to write
WORD1         : Second byte to write
WORD2         : Third byte to write
WORD3         : Fourth byte to write

(Notepad++) 34 % .sintf.writeCPU 4 1 2 3 4
Command Sent : 0x55  0x84 0x04 0x03 0x02 0x01
```

13.4.8. .readCPU

Returns the 32 bits read on the CPU interface.

```
(Notepad++) 35 % .sintf.readCPU
wrong # args: should be ".sintf.readCPU ADDRESS"
ADDRESS       : Address of the register in the component

(Notepad++) 36 % .sintf.readCPU 4
Command Sent : 0x55  0xC4 0x00 0x00 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.5. Tcl API functions

These functions are higher level function that can be used whatever is the communications link id. This allows to have more than one communication link in a script by using the same functions ...

13.5.1. OPEN_SI

Open the link and return the handle on the communication link.

```
(pkg_mvd) 6 % OPEN_SI
wrong # args: should be "OPEN_SI PATH NAME SPEED"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
NAME   : Com Port Name: COMx
SPEED  : Speed of the link

 (pkg_mvd) 2 % set mylink [OPEN_SI .mylink COM6 115200]
.mylink
```

13.5.2. enableVerbosity

Enables the verbosity (print command sent/received into the console)

```
(pkg_mvd) 7 % enableVerbosity
wrong # args: should be "enableVerbosity PATH"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)

(pkg_mvd) 3 % enableVerbosity $mylink
Verbosity ON
```

13.5.3. disableVerbosity

Disables the verbosity (print command sent/received will not be displayed any more into the console)

```
(pkg_mvd) 8 % disableVerbosity
wrong # args: should be "disableVerbosity PATH"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)

(pkg_mvd) 4 % disableVerbosity $mylink
Verbosity OFF
```

13.5.4. setVerbosity

Forces verbosity into the expected state.

```
(pkg_mvd) 9 % setVerbosity
wrong # args: should be "setVerbosity PATH VERBOSITY"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
VERBOSITY     : Expected state (0|1)

(pkg_mvd) 10 % setVerbosity $mylink 0
Verbosity OFF
```

13.5.5. getVerbosity

Returns the current verbosity state.

```
(pkg_mvd) 11 % getVerbosity
wrong # args: should be "getVerbosity PATH"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)

(pkg_mvd) 12 % getVerbosity $mylink
0
```

13.5.6. setSpiCS

Set the next SPI component CS to address

```
(pkg_mvd) 13 % setSpiCS
wrong # args: should be "setSpiCS PATH CS"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
CS     : CS of the component to address

(pkg_mvd) 14 % setSpiCS $mylink 0
(pkg_mvd) 15 %
```

13.5.7. writeSpi

Writes to SPI bus defined Bytes(4 bytes mode only)

```
(pkg_mvd) 18 % writeSpi
wrong # args: should be "writeSpi PATH WORD0 ?WORD1? ?WORD2? ?WORD3?"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
WORD0  : Byte 0 of the command, this value is mandatory

WORD1, WORD2, WORD3 = BYTES 1, 2 and 3 (optional parameters)

(pkg_mvd) 20 % writeSpi $mylink 0
Command Sent : 0x55  0x52 0x00 0x00 0x00 0x00
```

13.5.8. setGPIODirection

Set each 24 bits direction of the GPIO bus

```
(pkg_mvd) 21 % setGPIODirection
wrong # args: should be "setGPIODirection PATH WORD0 WORD1 WORD2"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
WORD0  : Byte 0 direction configuration
WORD1  : Byte 1 direction configuration
WORD2  : Byte 2 direction configuration

(pkg_mvd) 22 % setGPIODirection $mylink 0 1 2
Command Sent : 0x55  0x60 0x00 0x02 0x01 0x00
```

13.5.9. getGPIODirection

Reads each 24 bits direction status of the GPIO bus

```
(pkg_mvd) 23 % getGPIODirection
wrong # args: should be "getGPIODirection PATH"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)

(pkg_mvd) 24 % getGPIODirection $mylink
Command Sent : 0x55  0x61 0x00 0x00 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.5.10. writeGPIO

This command sends write information on the GPIO bus

```
(pkg_mvd) 25 % writeGPIO
wrong # args: should be "writeGPIO PATH WORD0 WORD1 WORD2"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
WORD0  : Byte 0 value
WORD1  : Byte 1 value
WORD2  : Byte 2 value

(pkg_mvd) 26 % writeGPIO $mylink 0 1 2
Command Sent : 0x55  0x62 0x00 0x02 0x01 0x00
```

13.5.11. readGPIO

This command reads GPIO values on the 3 GPIO bytes (take care to the GPIO direction settings)

```
 (pkg_mvd) 27 % readGPIO
wrong # args: should be "readGPIO PATH"

PATH   : Widget path name (must start with a point and a lower case letter : .myLink)

(pkg_mvd) 28 % readGPIO $mylink
Command Sent : 0x55  0x63 0x00 0x00 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.5.12. writeI2C

This command sends write information on the I2C bus

```
(pkg_mvd) 29 % writeI2C
wrong # args: should be "writeI2C PATH ADDRESS SUB WORD0 ?WORD1?"
PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
ADDRESS       : Address of the I2C component
SUB    : Address of the register to write in.
WORD0  : Byte 0 value
WORD1  : Byte 1 value is optionnal
(pkg_mvd) 30 % writeI2C $mylink 0 10 0
Command Sent : 0x55  0x40 0x00 0x0A 0x00 0x00
```

13.5.13. readI2C

Returns value read on the I2C interface.

```
(pkg_mvd) 31 % readI2C
wrong # args: should be "readI2C PATH ADDRESS SUB BYTE"
PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
ADDRESS       : Address of the I2C component
SUB    : Address of the register to write in.
BYTE   : Number of byte to read (1 or 2)
(pkg_mvd) 32 % readI2C $mylink 0 12 1
Command Sent : 0x55  0x41 0x00 0x0C 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.5.14. setCpuCS

Set the component CS of the next CPU interface component to address.

```
(pkg_mvd) 35 % setCpuCS
wrong # args: should be "setCpuCS PATH CS"
PATH   : Widget path name (must start with a point and a lower case letter : .myLink)
CS     : CS of the component to address
(pkg_mvd) 36 % setCpuCS $mylink 1
Command Sent : 0x55  0x01 0x00 0x00 0x00 0x01
```

13.5.15. writeCPU

Writes 4 bytes on the 32 bit CPU interface.

```
(pkg_mvd) 37 % writeCPU
wrong # args: should be "writeCPU PATH ADDRESS WORD0 WORD1 WORD2 WORD3"
PATH          : Widget path name (must start with a point and a lower case letter : .myLink)
ADDRESS       : ADDRESS of the register to address
WORD0         : First byte to write
WORD1         : Second byte to write
WORD2         : Third byte to write
WORD3         : Fourth byte to write
(pkg_mvd) 38 % writeCPU $mylink 4 0 1 2 3
Command Sent : 0x55  0x84 0x03 0x02 0x01 0x00
```

13.5.16. readCPU

Returns the 32 bits read on the CPU interface.

```
(pkg_mvd) 39 % readCPU
wrong # args: should be "readCPU PATH ADDRESS"

PATH          : Widget path name (must start with a point and a lower case letter : .myLink)
ADDRESS       : ADDRESS of the register to address

(pkg_mvd) 40 % readCPU $mylink 4
Command Sent : 0x55  0xC4 0x00 0x00 0x00 0x00
Response Received :  0xCC 0xCC 0xCC 0xCC
```

13.6. Script example

```
set ::linkA [OPEN_SI .link1 COM6 115200]

proc write32bCPU { hCom cCs cAddr cData } {
        # Serial Interface
        # Activate Chip Select
        setCpuCS $hCom $cCs
        # programming register cAddr
        set BYTE0 [expr ($cData &0x000000FF)>>0]
        set BYTE1 [expr ($cData &0x0000FF00)>>8]
        set BYTE2 [expr ($cData &0x00FF0000)>>16]
        set BYTE3 [expr ($cData &0xFF000000)>>24]
        puts [writeCPU $hCom [expr $cAddr >>2] $BYTE0 $BYTE1 $BYTE2 $BYTE3]
}


proc read32bCPU { hCom cCs cAddr } {
  # Serial Interface
  set VERBOSITY [getVerbosity $hCom]
  disableVerbosity $hCom
  # Activate DVBT Chip Select
  setCpuCS $hCom $CS
  # programming register 0
  set VALUE [readCPU $hCom [expr $ADDR>>2]]
  set RESULT [expr [expr [lindex $VALUE 0]]*2**24 + [expr [lindex $VALUE 1]]*2**16  + [expr
[lindex $VALUE 2]]*2**8 + [expr [lindex $VALUE 3]]]
  setVerbosity $hCom $VERBOSITY
  return $RESULT
}

# read 10 first register value..
for {  set i 0 } { $i <10 } { incr i } {
        puts "Register [$i*4] : [format %08X [read32bCPU $::link1 1 [expr $i*4]]]"
}
```

14. Ordering information and related cores

Xilinx Programmable Logic

For information on Xilinx programmable logic or development system software, contact your local Xilinx sales office, or :

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
URL : www.xilinx.com

15. Revision History

Revision	Date	Modified Pages	Observations
2.0 A	12/2008	All	
2.0 B	06/2009	All 10 15	Changes of MVD Logo Add 2 bits shifting address requirement for MVD Core Address field Add virtex 5 ressources information Add example of MVD's companion core
2.0 C	03/2011	3 13 14 15	Add example of programming commands modify I²C interface for netlist compatibility Show I²C interface implementation example for master mode Show I²C interface implementation example for slave mode
2.0 D	11/2011	All 16	Removing of support_cores@mvd-fpga.com address Add this history revision paragraph
2.0 E	01/2013	NA	Add Tcl packages documentation
2.0 F	09/2013	13,14,15 1, 7, 13	Add GPIO Tristate compliant buses Add B57K input signal
2.0 G	08/2014	5	Add information about I2C slave operation and Chip Select programming protocol
2.1 A	09/2016	13,14,15	Add spartan 6 and 7 Series resources Modify entity declaration to conform MVD's VHDL coding rules
2.2 A	12/2016	13,14,15	Renaming signal External bus to CPU_X_x signals Add _i on signal b115 which was missing
3.0A	03/2017	1,7,15	Modified functionality (baud rate support) Modified port definition according to new baud rates supported Removed Spartan-3 ™, Virtex-4 ™ and Virtex-5 ™ ressource