IDC

IPDB_SPMI2, Implementation Specification

Revision 2.3, 15-Oct-2018

Document Number: N/A

Revision History

Revision Number	Description	Revision Date
0.8	Initial release.	18-Apr-2017
0.9	Several updates	10-May-2017
1.0	A0 RTL2.8 update	16-June-2017
1.1	A0 RTL2.9 update	23-June-2017
1.2	A0 RTL3 update	13-July-2017
1.21	Added wave diagram	25-July-2017
1.4	Updates	7-Nov-2017
1.5	B0 RTL0.5 update	28-Dec-2017
1.6	B0 RTL0.8 update	24-Feb-2018
1.7	B0 RTL1.0 update	28-Mar-2018
1.8	B0 RTL1.0 simulation and coverage updates	10-Apr-2018
1.11.0	Polaris A0 RTL0.8	10-Aug-.2018
2.0	Polaris A0 RTL0.8 SPMI-2 (auto response registers)	11-Aug-.2018
2.2	Polaris A0 RTL0.9 SPMI-2	5-Oct-.2018
2.3	Polaris A0 RTL0.95 SPMI-2	15-Oct-2018

Contents

Figures

Contents

Tables

1 Overview

1.1 Design Hierarchy

1.2 Basic structure description

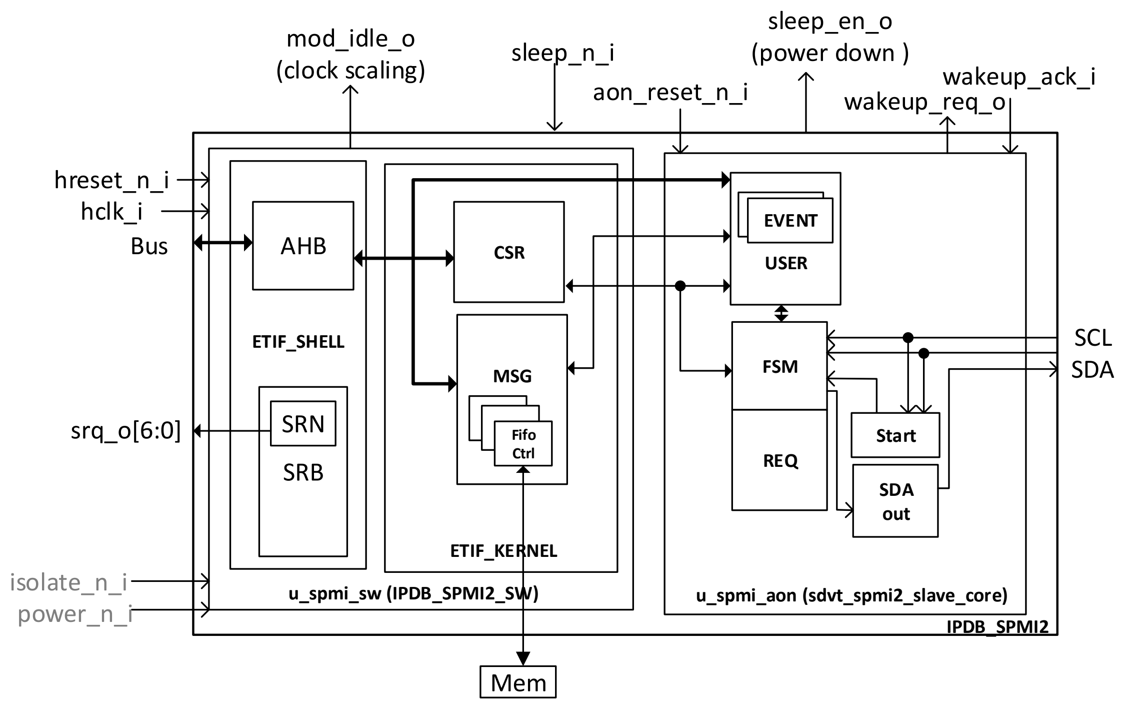


Figure 1: Blockdiagram

Toplevel

The SPMI2 top entity contains a switchable and an always on block. Beside of the AHB slave interface there are seven interrupts, the SPMI interface and some power and clocking related signals. There are three clock domains, the bus clock domain (hclk), the spmi clock domain (scl) and the reference clock domain (ref_clk). There are two reset signals, one for the switchable part (hreset) and one for the always on part (aon_reset). Both resets are in the hclk domain. The scl domain in the switchable part have a reset synchronization, the one in the always on part has no reset synchronization. SCL should not toggle while the always on reset is being active. The ref clock domain has a reset synchronization.

Power down

Isolate (isolate_n_i) and power switch (power_n_i) input signals are available for isolating and switching the switchable part. If the power switching of the switchable part is not being used then the two signals can be tied to any dummy value. Powering down of the memories is an alternative (signaled by sleep_n_i=0). The sleep enable signal (sleep_n_o) signals that the switchable part or the memories can be powered off (no ingoing or outgoing sequences are active and no messages in the buffers). The wakeup request signals that the switchable part or message buffer memories need to be powered (a message or a wake-able event has been received). The wakeup acknowledge signal handshakes this request. Both signals are level signals in the scl clock domain. If the powerdown (power_n_i=1, isolate_n_i=0, hreset_n_i=0) or memory power saving (sleep_n_i=0) is active then a NAK is returned in case of SPMI writes the rx message buffers. An active reset (hreset_n_i=0) has the same effect as the power down. The powerdown (sleep_n_o=0) can be prevented by the panic mode register.

Clock Scaling

The module idle signal signals that the bus clock can be scaled down (e.g. from 105.6 MHz to 26 MHz).

Clock Gating

It is possible to gate hclk_i externally (in case of mod_idle=1 or sleep_en=1), but this has the following implications: The signals mod_idle_o, sleep_en_o, all interrupts signals and the timeout are frozen. It is possible to read the TX message buffers from SPMI bus side, but it is not possible to write the RX buffers. The wakeup handshake is available (wakeup if wake-able event or message has been received). The interrupts are updated after few hclk cycles. It is possible to gate the kernel internally by using the register SPMI_CLC.

Structure

The switchable block contains the ETIF shell and the ETIF kernel. The shell contains the AHB interface, the register block and the Interrupt service request block. The kernel contains the SmartDV CSR block (register block) and the Message Buffer block. The message buffer block contains asynchronous fifos (TX) and synchronous fifos (RT RX) and buffers (RT TX). They can be implemented by using RAMs, which need to support a write port in one clock domain and a read port in the other clock domain. If no RAMs are available, then an internal register file can be enabled by generic.

The always on entity is the SmartDV core, which contains the blocks REQ (Slave requests), FSM (Slave state machine), SDA_OUT (output data) an START (start and arbitration detection). The User block was added. It contains the customer registers and the Event block which contains a fifo and the wakeup logic.

2 Detailed Description

2.1 ETIF Shell

The ETIF shell contains the AHB 2.0 interface, the interrupt nodes and a register block (except SmartDV core registers).

2.2 ETIF Kernel

The CSR block of the SmartDV core is connected to the RAM interface of the TOPSPIN block. The RX, TX and RT TX message buffers and the User block have a direct connection to the RAM interface. Five chip select signals distinguish between SmartDV core, RX messages, TX messages, RT TX messages and user block (Ram_no). The following code snippet shows the memory layout. Eg. the SmartDV core has an 8bit wide data bus, this is mapped to the least significant bits of 32bit data on AHB bus. The minimum distance of a core address to the next one is therefore 4. The auto response buffers have a width of 32.

```
--     ram-name       |  start    |   end     |  offset    |Ram| width |secure |prot1 | mode
--                    |  address  |  address  |            |_No|       |       |      |
0 => ( Slave          ,X"00001000",X"0000103B",X"00001000", 0 ,   8   ,   o   ,  o   ,   readwrite   ),
1 => ( Request        ,X"00001100",X"000012FF",X"00001000", 0 ,   8   ,   o   ,  o   ,   readwrite   ),
2 => ( RT_TX_MSG_BUF  ,X"00002000",X"0000217F",X"00002000", 1 ,   32  ,   o   ,  o   ,   readwrite   ),
3 => ( General        ,X"00008000",X"000081FF",X"00008000", 2 ,   8   ,   o   ,  o   ,   readwrite   ),
4 => ( Auto_Response  ,X"00008200",X"0000820F",X"00008000", 2 ,   32  ,   o   ,  o   ,   readwrite   ),
5 => ( Config         ,X"00008400",X"000087FF",X"00008000", 2 ,   8   ,   o   ,  o   ,   readwrite   ),
6 => ( TX_MSG_BUF     ,X"00008280",X"000083FF",X"00008280", 3 ,   8   ,   o   ,  o   ,   readwrite   ),
7 => ( RT_RX_MSG_BUF  ,X"0000C000",X"0000C5FF",X"0000C000", 4 ,   32  ,   o   ,  o   ,   readwrite   ));
```

Figure 2: Memory mapping

The kernel contains a reset synchronization for the message buffers and the CSR block in the SCL clock domain. It also generates a power on signal for the FSM based on this reset synchronization.

2.2.1 Core - Start Block

The start block contains start and arbitration condition detection. **It needs special handling in layout, descripted in chapter 3.10.**

The data transmission is performed in the way: sda (data) shifting on rising edge, sda capturing on falling edge of scl (clock), whereas sda change is expected after scl rising edge. The start condition and the arbitration condition are exceptions.

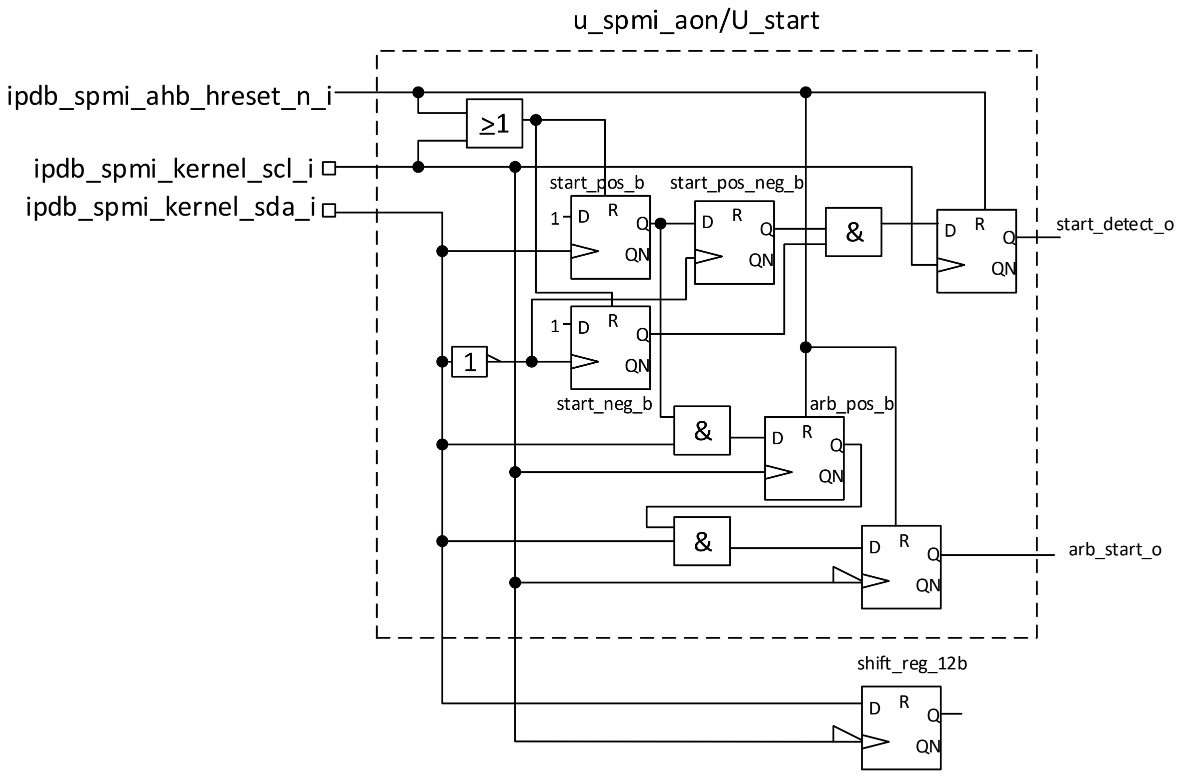


Figure 3: Start block circuit diagram (updated compared to XG766A0)

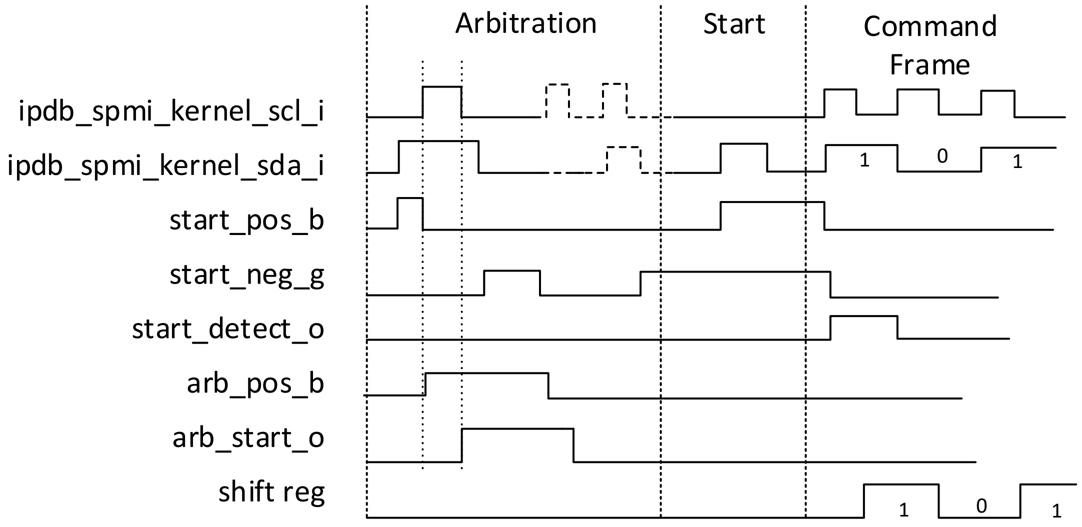


Figure 4: Start block wave diagram

Detailed Description

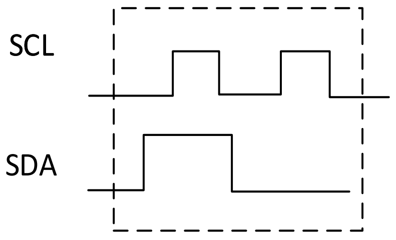


Figure 5: Arbitration Condition

The arbitration condition is an sda rising edge followed by a scl rising edge. The second part is a falling edge on scl followed by a falling edge on sda. If the scl has a higher signal delay than sda, then normal data transmission can cause a wrong arbitration detection.

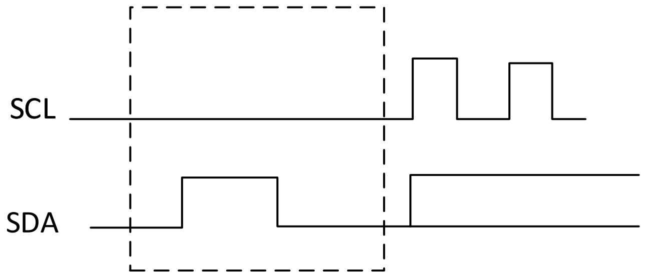


Figure 6: Start Condition

The start condition is a pulse on sda while scl stays low.

2.2.2 Core - FSM and REQ Block

The FSM and Req blocks contain the state machines for the MIPI SPMI Slave and MIPI SPMI Response-capable slave. The signal rd_busy_i of the FSM can be used to create no response frames. The FSM was modified to return ACK/NACK based on User block decision.

```
`ifdef INTEL_USER_V04
  input     wire                                i_clk_off_ack      , // ack due to standby register
  input     wire                                i_clk_on_nack      , // nack due to buffer full
`endif
```

Figure 7: FSM port modifications

2.2.3 CSR block

The CSR contains registers. The CSR block contains a sub design ipdb_spmi2_sync with contains a synchronizer, a flipflop and a reset synchronization. It is used for setting a flipflop in the SPMI clock domain and reading and clearing it from bus clock domain.

2.2.4 User Block

The user block contains the error/status registers at SPMI address 0x000 and configuration registers at 0x100, a timeout counter, an event fifo, wakeup logic and multiplexer logic for the message buffer accesses. It is connected on the local interface between the CSR block and the FSM block. A SPMI bus write to a RX message buffer will cause a NAK if the switchable part is turned off (in reset or isolated/powered down). A SPMI bus read to a TX message buffer will cause a no response frame in the same situation.

```
input     wire [15:0]                          slv_addr_i        , // SPMI FSM address
input     wire [7:0]                           slv_data_i        , // SPMI FSM write data
output    wire [7:0]                           slv_data_o        , // SPMI FSM read data
input     wire                                 slv_wr_i          , // SPMI FSM write
input     wire                                 slv_rd_i          , // SPMI FSM read
input     wire [4:0]                           slv_cmd_type_i    , // SPMI FSM command type
output    wire                                 slv_rd_busy_o     , // SPMI FSM no response
output    reg                                  slv_nack_o        , // SPMI nack in active mode
output    reg                                  slv_ack_o         , // SPMI ack in standby mode
```

Figure 8: FSM from/to User block signals

Most registers in the user block can be written from one side and read from both sides. The Event related registers can be accessed from AHB bus side only. The SPMI slave core gets USID, GSID from the user block. At the beginning the SPMI Core (register SPMI_CONTROL and SPMI_GSID_CONTROL) values are being used. If the SPMI_CTRL register or SPMI_GROUP_ID are written from the SPMI bus then these values are being used. The SPMI core registers REQ_FREE can be programmed from the bus side or the the SPMI side (ARB_MON register).

```
input     wire                                 bus_en_i                , // bus select signal
input     wire [8:0]                           bus_addr_i              , // bus address
output    reg  [31:0]                          bus_data_o              , // bus read data
input     wire [31:0]                          bus_data_i              , // bus write data
input     wire                                 bus_rd_i                , // bus read
input     wire                                 bus_wr_i                , // bus write
output    wire                                 bus_busy_o              , // bus delay
```

Figure 9: Bus to/from User block signals

Access conflicts caused by SPMI and AHB bus accesses cause a short delay on the AHB bus access (AHB write to EVT_MASK registers while an event is being received, AHB read from REGx or configuration registers while a SPMI write access is being received, AHB read on status register, AHB write on auto response registers). The data has to cross clock domains, the delay is needed to provide a stable data signal for capturing. This delay is implemented by using the bus_busy_o signal, which inserts busy clock cycles in the two clock cycle bus reads.

Timeout

The timeout counter is loaded in case of an arbitration (ARB_EN=1) initiated by the CSR block (SDA set to one) or if a SCL clock transition is visible (SEQ_EN=1). It is stopped if a SEQ has ended or a clock transition is seen (ARB_EN=1 and SEQ_EN=0). It causes an unsynchronized reset if it expires. This reset does not affect the message buffers.

Error registers

The error bits are in the scl clock domain. Timeout related errors need scl clock transitions to be reflected. The error bits can only be cleared from the SPMI bus side. Any error bit being set causes a low priority interrupt.

Detailed Description

Legal commands are reads and writes. Other standard commands can processed correctly by the SmartDV core, but they are not supported by definition. Nonstandard command words are ignored (NAK on bus) and they cause a command parity error.

Parity errors are gathered from incoming sequences based on the slave state machine transitions. If a write has a parity error in one data frame then the next data frames are also flagged as having an error.

A sequence error is flagged if an outgoing sequence is being interrupted by a timeout. A slv_reset is flagged if a timeout (on an outgoing or incoming sequence) has occurred.

The retry counter needed for the status bit RETRY_TO counts a series of received NAKs. It does not analyze if the software did perform a retry of the same sequence or a new sequence.

Auto response registers

The 4 auto response registers are located at offset 0x8200.. 0x820F. They are accessible from SPMI side at 0x80..0x8F (even in powerdown state).

2.2.5 Event Block

The event block contains an asynchronous fifos for the events and a flipflop for the wakeup request signal.

The fifo has an EVT_RXD register, an EVT_STATUS register (filllevel) and an EVT_CLEAR (if all events have to be cleared at once. An event interrupt will be issued if an event is being received. If the according EVT_MASK bit being set then the wakeup handshake will be issued in addition. The event is being put in the event fifo.

The wakeup_req is a level signal. It is activated if a message buffer is written on SPMI bus, a wakeup command is being received, or a wake-capable event (EVT_MASK register bit is 1) is being received. It is cleared by the wakeup_ack signal. A new wakeup request is only set if the wakeup_ack signal is begin cleared (level handshake).

2.2.6 Message Buffers

The TX message buffers are implemented as asynchronous fifos. The storage of this fifos can be optionally 2-Port RAMs (RAMS with two clocks, writing on one side, reading on the other). The TX message buffers have reset synchronizations for clearing the content. Each message buffer can issue an interrupt based on a threshold. An RX buffer will issue if it contains more words than the threshold. A TX buffer will issue if it contains more or equal events than the threshold. Each message buffer has a STATUS (filllevel), a CTRL (threshold) and a CLEAR register. An address area is being used as RXD and TXD register for each buffer. The first TX message buffer has its memory area at 0x8280-0x8FF, the second is at from 0x8300-0x837F, the third is at 0x8380-0x83FF. The first RX message buffer has its area at 0xC000-0xC0FF, the second at 0xC100, ..., the sixth at 0xC500.

2.2.7 Asynchronous Fifo

The TX message buffers and the event block contain an asynchronous fifo design. It uses gray code secured pointers. It can be used with flipflops or a synchronous two port ram.

2.2.8 Synchronous Fifo

The RX message buffers contain an synchronous fifo design. It uses pointers. It can be used with flipflops or a synchronous two port ram.

2.2.9 Register file

The Realtime TX message buffers can be used with flipflops or a synchronous two port ram.

2.3 Packages and Include files

2.3.1 ipdb_spmi2_etif_comp_pack-p.vhd

The package contains a component declaration for ipdb_spmi2.

2.3.2 ipdb_spmi2-p.vhd

The package contains all important constants and the component declarations.

2.3.3 ipdb_spmi2_etif_pack-p.vhd

The package contains all important constants for TOPSPIN (e.g. register definitions).

2.3.4 ipdb_spmi2_defines.v

The package contains all important constants for Core modifications.

```
`define INTEL_ADAPTION                      clock/reset adaptions
`define INTEL_USER                          structural modification in RTL
`define INTEL_USER_V04                      customer 0.4 spec features
`define INTEL_OPTIMIZE                      area optimization


`define EVENT_WIDTH           5             32 Event-types
`define EVENT_DEPTH           3             8  Fifo stages
`define EVENT_FIFOS           1             1  Event Fifos
`define CNT_WIDTH            14             Counter width for timeouts (max. 96us)
```

Figure 10: Core modification constants

3 Design Integration

3.1 Design ports

Table 1. Design ports

Port name	direction	width	clock	Short description
ahb_hreset_n_i	in	1	hclk	Asynchronous active low reset of the switchable part
ahb_hclk_i	in	1	hclk	Bus clock (AHB clock)
ahb_hsel_i	in	1	hclk	AHB peripheral select signal
ahb_haddr_i	in	32	hclk	AHB peripheral address line
ahb_hwrite_i	in	1	hclk	AHB peripheral transfer direction
ahb_htrans_i	in	2	hclk	AHB peripheral transfer type
ahb_hsize_i	in	3	hclk	AHB peripheral transfer size
ahb_hburst_i	in	3	hclk	AHB peripheral burst type
ahb_hprot_i	in	4	hclk	AHB protection
ahb_hready_i	in	1	hclk	AHB peripheral ready input
ahb_hwdata_i	in	32	hclk	AHB peripheral write data line
ahb_hready_o	out	1	hclk	AHB peripheral ready output
ahb_hresp_o	out	2	hclk	AHB peripheral transfer response
ahb_hrdata_o	out	32	hclk	AHB peripheral read data line
ahb_sb_bigendian_i	in	1	hclk	Little or big endian
ahb_sb_hprot1_i	in	1	hclk	
ahb_sb_trustzone_en_i	in	1	hclk	trustzone enable
ahb_sb_tz_secure_i	in	1	hclk	trustzone mode
bus_disable_n_i	in	1	hclk	Kernel clock disable
srq_o	out	7	hclk	Interrupt vector (0..5 processors, 6 general)

sleep_n_i	In	1	Hclk	0 means that the memories are in powersave mode
ref_time_i	In	32	Ref	Reference timer value
rf_clk_i	In	1	Ref	Reference clock
kernel_sda_en_n_o	Out	1	Scl	Low active output enable for SDA pad
kernel_sda_o	Out	1	Scl	SDA pad output
kernel_sda_i	In	1	Scl	SDA pad input
kernel_scl_i	in	1	Scl	SCL clock pad input
sleep_en_o	out	1	hclk	Switchable power domain or memories can be put in power save state
isolate_n_i	in	1	hclk	Enable isolation of switchable power domain (optional)
aon_reset_n_i	in	1	hclk	Asynchronous active low reset reset of aon part
wakeup_req_o	out	1	Scl	Wakeup request signal (level, 1 = request)
wakeup_ack_i	in	1	Scl	Wakeup acknowledge (level, 1 = acknowledged)
mod_idle_o	out	1	hclk	Idle signal (hclk can be reduced to 26 MHz if one)
ocds_suspend_i	in	1	hclk	On chip debug suspend
pdft_scan_mode_i	in	1		Scan mode (post reset sync bypass)
pdft_scen_i	in	1		Scan shift (also enables test input of clock gates)
signal_monitor_o	out	30		Monitor signals
ram1_clka_o	out	1	hclk	TX message buffer memory 96x8 TP_HSTPRF
ram1_clkb_o	out	1	Scl	TX message buffer memory
ram1_csba_o	out	1	hclk	TX message buffer memory
ram1_csbb_o	out	1	Scl	TX message buffer memory
ram1_aa_o	out	7	hclk	TX message buffer memory

ram1_ab_o	out	7	Scl	TX message buffer memory
ram1_dia_o	Out	8	hclk	TX message buffer memory
ram1_dob_i	In	8	Scl	TX message buffer memory
Ram2_clka_o	out	1	Hclk	RX message buffer memory 432x32 TP_HSTPRF
Ram2_clkb_o	out	1	Hclk	RX message buffer memory
Ram2_csba_o	out	1	Hclk	RX message buffer memory
Ram2_csbb_o	out	1	Hclk	RX message buffer memory
Ram2_aa_o	out	9	hclk	RX message buffer memory
Ram2_ab_o	out	9	Hclk	RX message buffer memory
Ram2_dia_o	Out	32	hclk	RX message buffer memory
Ram2_dob_i	In	32	Hclk	RX message buffer memory
pdft_rst_external_sel_i	In	1		Select external reset for soft reset logic
pdft_rst_clk_sel_i	In	1		Select scan related clock
pdft_switch_on_all_clk_i	In	1		Functional enable of all clock gates (e.g. for mbist)
Ram3_clka_o	out	1	Hclk	RT TX message buffer memory 128x32 TP_HSTPRF
Ram3_clkb_o	out	1	Hclk	RT TX message buffer memory
Ram3_csba_o	out	1	Hclk	RT TX message buffer memory
Ram3_csbb_o	out	1	Hclk	RT TX message buffer memory
Ram3_aa_o	out	7	hclk	RT TX message buffer memory
Ram3_ab_o	out	7	Hclk	RT TX message buffer memory
Ram3_dia_o	Out	32	hclk	RT TX message buffer memory
Ram3_dob_i	In	32	Hclk	RX TX message buffer memory

3.2 Design generics

The toplevel supports following generics:

sim_jitter_g	Use sim_jitter simulation mechanism for synchronizers. Setting is not relevant for synthesis.
ram1 _g	Implementation of RAM1 (0=internal flipflops , 1 = RAM)
ram2_g	Implementation of RAM2 (0=internal flipflops , 1 = RAM)
usid_g	Reset value of USID value (0xF= Baseband IC recommendation)
freq_g	Kernel frequency (1056 = 105.6 MHz)
Ram3_g	Implementation of RAM (0=internal flipflops, 1 = RAM)

Table 2. Design generics

3.3 Libraries

The libraries ipdb_cclib and ipdb_csblib are being used:

Ipdb_cclib_sync	2-stages synchronizer
Ipdb_cclib_sync_rst	Reset synchronizer
Ipdb_cclib_mux	Mux used in reset trees
Ipdb_cclib_clk_mux	Mux in clock tree
Ipdb_cclib_clk_inv	Inverter in clock tree
ipdb_csblib_pulse_sync_v2_0_0	Pulse 2 pulse synchronizer
ipdb_csblib_vecsync_v2_0_0_rtl_conf	Vector synchronisation

3.4 Vendor documents

The directory **delivery** contains the vendor release (e.g. linting, cdc, synthesis scripts) as reference.

3.5 Design clocking

The bus clock and the SPMI interface related clocks are asynchronous to each other.

3.5.1 Busclock ipdb_spmi2_ahb_hclk_i (105.6 MHz)

This is the AHB-Busclock used for AHB Slave, Registers, Buffers and Interrupts.

3.5.2 SPMI-Clock ipdb_spmi2_ kernel_scl_i (26 MHz)

The SPMI interface clock. It is used as flipflop clock. Both edges are being used. The signal is being used as reset for the two flipflops of the sda clock domain.

3.5.3 SPMI special clock ipdb_spmi2_ kernel_sda_i (26/2 MHz)

The SPMI interface data signal. It is used as clock for two flipflops and as data for several flipflops in the scl clock domain. Both edges are being used. There is a clock multiplexer in case of scan mode (which replaces by ipdb_spmi2_kernel_scl_i).

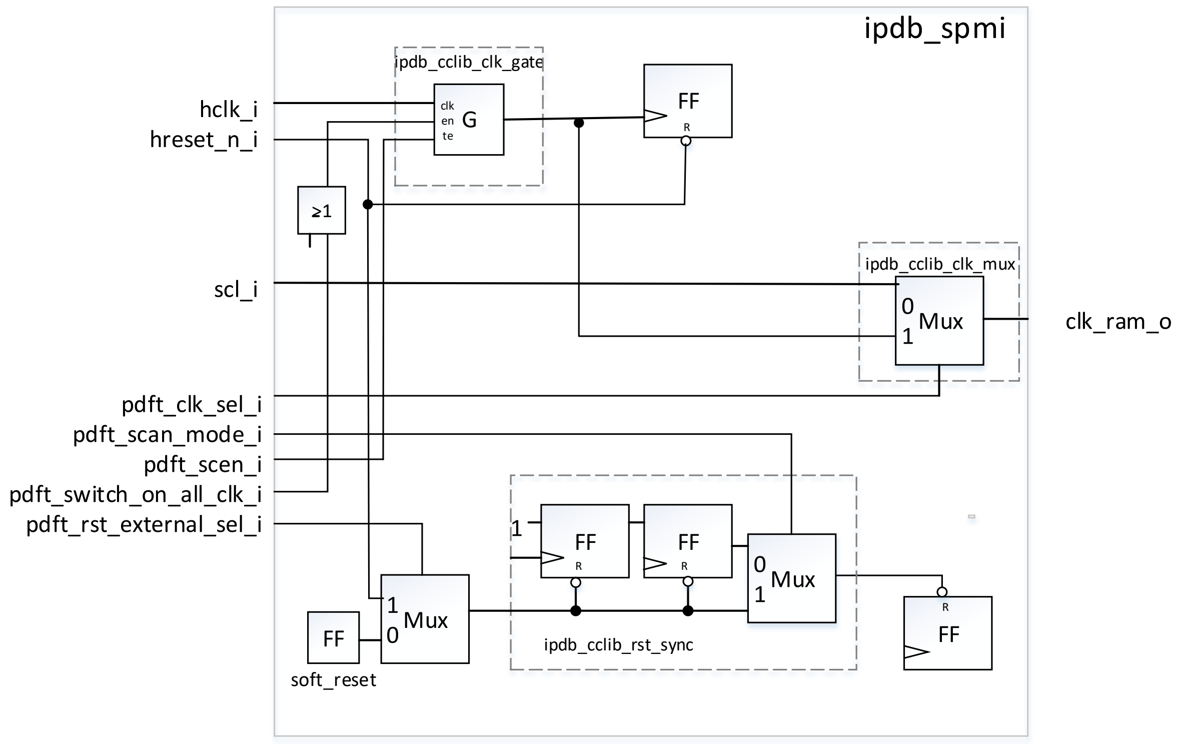
3.5.4 Reference clock (38.4MHz)

The reference clock signal. It is used for a vector synchronization of the reference time value.

3.6 Reset tree

There are the reset signal ipdb_spmi2_ahb_hreset_h_i and ipdb_spmi2_aon_reset_n_i (both in hclk domain). There are reset synchronization cells ipdb_cclib_reset_sync cells being used. The reset for the SCL clock domain is not synchronized for the always on part, because it needs to work without clock. The SmartDV core has an unsynchronized software/timeout reset, a synchronizer prevents issues, if the scl clock is being active. The reference clock domain has a reset synchronizer.

3.7 DFT signals



3.8 Monitor Signals

Table 3. Monitor signals

Port name	direction	width	clock	Short desription
signal_monitor_o[15:0]	Out	16	scl	SmartDV core debug signals
signal_monitor_o[16]	Out	1	scl	SmartDV core busy
signal_monitor_o[29:17]	Out	13	hclk	TOPSPIN monitor signals [31:19]

The SmartDV core allows to select several internal signals for monitoring. The default setting is SDVT_SPMI_SLAVE_DEBUG_FSM_STATE, e.g. the 6 bit FSM state slv_state_6w is at signal_monitor_o[5:0].

```
assign o_debug_data =
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_FSM_STATE) ?
{5'b0,disable_sda_driver_o,silent_o,sda_en_o,arb_start_o,start_detect_w,slv_state_6w}  :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_FSM_ADDR)  ? addr_o              :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_FSM_CMD)   ? {11'b0,cmd_type_o}   :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_FSM_BC)    ? {12'b0,bc_o}         :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_RCS_STATE) ? {11'b0,req_state_o}  :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_RCS_ADDR)  ? req_reg_addr_i       :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_RCS_CMD)   ? {12'b0,req_cmd_i}    :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_RCS_BC)    ? {12'b0,req_bc_i}     :
  (o_debug_sel == `SDVT_SPMI_SLAVE_DEBUG_RCS_ASTATE)? {12'b0,csr_state_4w}  : 16'hDEAD;
```

The TOPSPIN monitor signals:

```
  constant regsync_rdbusy_mon_c         : integer := 19; -- signal_monitor_o[17]
  constant regsync_wrbusy_mon_c         : integer := 20; -- signal_monitor_o[18]
  constant regsync_kernelread_mon_c     : integer := 21; -- signal_monitor_o[19]
  constant regsync_kernelwrite_mon_c    : integer := 22; -- signal_monitor_o[20]
  constant ahb_read_err_resp_mon_c      : integer := 23; -- signal_monitor_o[21]
  constant ahb_write_err_resp_mon_c     : integer := 24; -- signal_monitor_o[22]
  constant ahb_hready_mon_c             : integer := 25; -- signal_monitor_o[23]
  constant ahb_cmdfifo_full_mon_c       : integer := 26; -- signal_monitor_o[24]
  constant bpi_clken_mon_c              : integer := 27; -- signal_monitor_o[25]
  constant ts_dis_n_mon_c               : integer := 28; -- signal_monitor_o[26]
  constant kernel_clken_mon_c           : integer := 29; -- signal_monitor_o[27]
  constant kernel_dis_mon_c             : integer := 30; -- signal_monitor_o[28]
  constant kernel_busy_mon_c            : integer := 31; -- signal_monitor_o[29]
```

3.9 Power isolation

The switchable block can have a power isolation around it.

3.10 Layout recommendations

The inputs ipdb_spmi2_kernel_sda_i (bidirectional pad SDA) and ipdb_spmi2_kernel_scl_i (pad SCL) need to be treated special in case of layout.

They are used as clock (both edges), reset and data of flip-flops. The paths needs to be balanced (for instance U_SPMI/U_SPMI_AON/U_start). The SDA related paths can be slightly slower or equal than the SCL paths.

The paths to the following FFs in this block need to be handled

SCL to start_detect_o_reg/CP
SCL to start_pos_b_reg/R
SCL to start_pos_neg_b_reg/R
SCL to start_neg_b_reg/R
SCL to arb_pos_b_reg/CP
SCL to arb_start_o_reg/CP
SDA to start_pos_b_reg/CP
SDA to start_neg_b_reg/CP
SDA to arb_pos_b_reg/D

4 Miscellaneous Topics

4.1 Simulation notes

4.1.1 RTL-Simulation with SV/UVM

The UVM testbench comprises of several verification IPs to verify DUT functionality:

- SmartDV SPMI Slave verification IP
  - Passive instance for DUT monitoring
  - Active instance
- SmartDV SPMI Master verification IP
  - 2 active instances for generating transactions
- Cadence AHB VIP for writing registers
  - A UVM register model is generated out of register xml description
  - An adapter is used to convert UVM register model transactions to AHB transactions and access the DUT registers

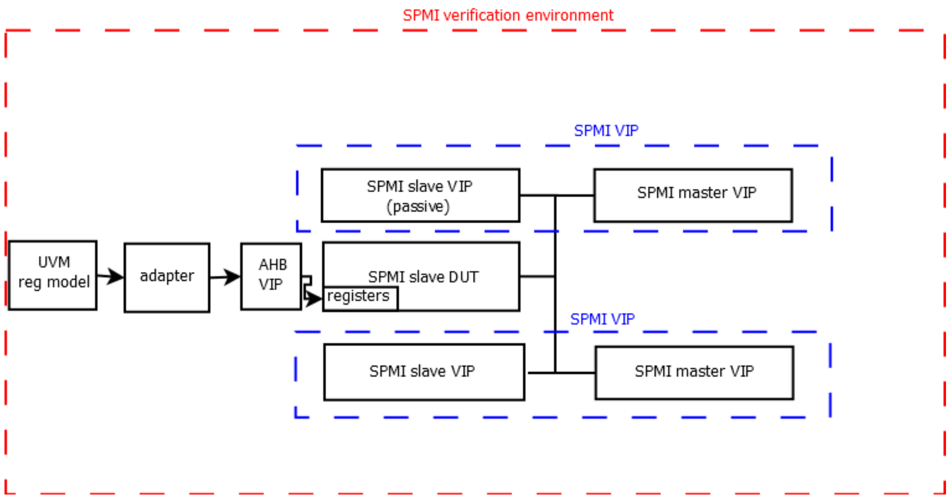


Figure 11: SV/UVM testbench

To start simulation following steps are necessary:

In /vobs/ipdb_es-ipdb_smpi-vob/units/ipdb_spmi2/simulation_vcs/sim run:

gmake clean (delete VCS database from last compilation step)

gmake extract (create dependency tree)

gmake compile (VCS compilation step)

Miscellaneous Topics

gmake elab (VCS elaboration step)

gmake run UVM_TESTNAME=test_vip (test_vip selects SmartDV tests running with SmartDV's test_runner)

Tests are task based and are randomly selected from a list in /vobs/ipdb_es-ipdb_smpi-vob/units/mipi_spmi11/uvm/examples/src/ sdvt_spmi_demo_tb.sv in task test_runner. To run a specific test, test_name_s can be set.

iRunner Regression can be started using the script start_irunner.sh in the simulation directory.

The latest SmartDV SPMI VIP version is located under /vobs/ipdb_es-ipdb_smpi-vob/units/mipi_spmi12. The verification environment is located under /vobs/ipdb_es-ipdb_smpi-vob/units/ipdb_spmi2/source/sv/tb/ipdb_spmi. The top file with instances of DUT and VIP interfaces is located at /vobs/ipdb_es-ipdb_smpi-vob/units/ipdb_spmi2/source/sv/tb/ipdb_spmi_tb

4.1.2 RTL-Simulation with directed testcases

There are directed testcases for additional Features (sfr, special, sema, tx_msg, rx_msg, event, wakeup, gsid, master, power, error, rt_msg) in directory simulation_vcs/pdl. These were used by designers for quick checking.

iwexprt setup_vcs -id directed -uc rtl -unit ipdb_spmi2 -top ipdb_spmi2_tb_struct_conf -force

```
cdunit ipdb_spmi2
cd simulation_vcs/directed
gmake clean
gmake elab
testcase <testcase>
gmake run
```

```
sim> do wave.do
sim> run
sim>exit
```

check

Figure 12: Directed testcase flow

sfr	checks access • SmartDV reg • Message Buffer • Application register • Configuration register

Miscellaneous Topics

	• Spmi access (normal, NAK) • bus rd conflict (app, config, status) • bus wr conflict • bus status conflict
special	checks coverage misses • scan mode • scen enable • other test signals
power	checks access in power states • clocked • unclocked by CLC • power off • rx message during wakeup from poweroff • memory sleep • rx message during wakeup from mem sleep • check sleep_en_o
wakeup	checks wakeup from deep sleep • wakeup cmd • rx message • wakeable event
gsid	checks gsid usage • usid ack • gsid ack • usid nack (power off) • gsid nack (power off) • second gsid • third gsid
tx msg	checks tx message buffers • tx message • tx overflow • tx underflow
master	checks request • master write • master write with NACK • master write with arb timeout • master write with seq timeout • master write with TBD timeout
event	checks events • reg0 • normal write

Miscellaneous Topics

	• event overflow • interrupt • event underflow
error	checks errors • cmd parity • addr parity • data parity • invalid addr • invalid command • reset (sequence timeout)
auto	check auto response buffer • read/write • read during deep sleep
sema	checks semaphore registers • gets semaphore • denies semaphore • retries semaphore
rx msg	checks rt rx message buffers • rx message to CPS • rx message to UPC1/UPC2 • rx broadcast to all • rx underflow • rx overflow (NAK)
rt msg	checks rt tx message buffers • immediate • interrupt generation • best effort • chained • timed • check full state /interrupt • normal master write

Table 4. Directed testcases

4.1.3 Coverage

Figure 13: Coverage SV/UVM + Directed Testcases

4.2 Code check (linting and CDC)

4.2.1 General

Code checking has been done with spyglass®.

Analysis as set up with following commands:

```
iwspy prepare -id rtl -top ipdb_spmi2_rtl_conf -fromStep source

iwspy clean -id rtl -top ipdb_spmi2_rtl_conf

iwspy makefiles -id rtl -top ipdb_spmi2_rtl_conf

iwspy compile_libs -id rtl -top ipdb_spmi2_rtl_conf

iwspy create -id rtl -top ipdb_spmi2_rtl_conf

iwspy rungui -id rtl -top ipdb_spmi2_rtl_conf -project standard_rtl.prj &iwspy prepare -id rtl -unit ipdb_spmi2 -force
```

For IPDB_SPMI2 release the latest spyglass project can be found under:
$WORKAREA/units/ipdb_spmi2/linting/rtl/standard_rtl.prj

The used constraint have been put into a common directory and these files are clear case version controlled.

$WORKAREA/units/ipdb_spmi2/linting/common

For the release the gathered information has been released by the iw-scripts for Spyglass.

- iwspy export
- iwspy release

4.3 Trial synthesis

4.3.1 Setup and Synthesis constraints

4.3.1.1 General setup

iwdc prepare -id rtl -top ipdb_spmi2_rtl_conf -local

The root synthesis dir is: `$WORKAREA/units/ipdb_spmi2/rtl2gds`

4.3.1.2 Constraints

The constraints files can be found here:

```
$WORKAREA/units/ipdb_spmi2/source/constraints
```

For details please see the checked in constraints.

4.3.2 Synthesis

Run following command to invoke a synthesis run (be patient a run can take several minutes up to an hour.

```
iwdc compile -id rtl -top ipdb_spmi2_rtl_conf
```

For detailed area reports use area.py and write_hier scripts (start them in the synthesis root directory: `$WORKAREA/units/ipdb_spmi2/rtl2gds/rtl`):

```
$WORKAREA/bin/write_hier_area results/ipdb_spmi2.compile.ddc
ipdb_spmi2
```

```
$WORKAREA/bin/area.py ipdb_spmi2_hier.txt ipdb_spmi2_area.txt
```

Results of these two commands can be found in hier_area.txt or hier_area.csv. (see chapter 4.3.5 for detailed results generated with these scripts.

4.3.3 Scan insertion

The scan insertion will be done automatically when performing an iwdc write_netlist –id rtl. The same for iwdc fix.

Important file for checking DFT is: ipdb_spmi2.dft_post.drc.rpt

Edit following file:

$WORKAREA/units/ipdb_spmi2/rtl2gds/rtl/addons/ipdb_spmi2.user_dft.tcl

```
set_dft_signal -view existing_dft -type Reset -timing {45 55} -port ipdb_spmi2_ahb_hreset_n_i

set_dft_signal -view existing_dft -type Reset -timing {45 55} -port ipdb_spmi2_aon_reset_n_i
```

Miscellaneous Topics

```
set_dft_signal -view existing_dft -type ScanClock -timing {45 55} -port ipdb_spmi2_ahb_hclk_i
set_dft_signal -view existing_dft -type ScanClock -timing {45 55} -port ipdb_spmi2_kernel_ref_clk_i
set_dft_signal -view existing_dft -type ScanClock -timing {45 55} -port ipdb_spmi2_kernel_scl_i
set_dft_signal -view spec         -type ScanEnable              -port ipdb_spmi2_pdft_scen_i
set_dft_signal -view existing_dft -type Constant -active_state 1 -port ipdb_spmi2_pdft_scan_mode_i
set_dft_signal -view existing_dft -type Constant -active_state 1 -port ipdb_spmi2_pdft_clk_sel_i
set_dft_signal -view existing_dft -type Constant -active_state 1 -port ipdb_spmi2_pdft_rst_external_sel_i
```

4.3.4 Write netlist and release

There was an issue with derating setup in the trial synthesis flow. Commenting in the scripts insert_dft.tcl, fix.tcl, write_netlist.tcl solved the issue.

```
#  source -echo -verbose ${R2G_SCRIPTS_DIR}/derating_setup.tcl

#  source -echo -verbose ${R2G_SCRIPTS_DIR}/derating_nonmcmm.tcl
```

```
iwdc write_netlist –id rtl

iwdc release –id rtl
```

4.3.5 Synthesis results

Table 5. Area

Instance name	[#] Instances	[K Nand2E]	[mm^2]
TOPSPIN (AHB, SRB, REG)	1		
CSR	1		
MSG	1		
MSG TX buffer (128x8)	1		
MSG RT TX buffer (128x32)	1		
MSG RT RX buffer(432x32)	1		
SPMI FSM	1		
SPMI REQ	1		
USER	1		
Always on Total			
Switchable Total			

4.4 Equivalence Check (LEC)

4.4.1 Setup

iwdc compile_ec –id rtl

iwdc release –id rtl

The root directory for LEC is: $WORKAREA/units/ipdb_spmi2/equiv_check

- iwlec prepare -force -unit ipdb_spmi2 -id rtlvsgate -top ipdb_spmi2_rtl_conf -revisedTop ipdb_spmi2 -fromStep source -revisedStep synthesis -fromPhase init -revisedPhase init -u2p "ipdb_spmi2 ipdb_topspin" -revisedId rtl
- iwlec extract -force -id rtlvsgate
- iwlec extract_revised -force -id rtlvsgate

4.4.2 LEC run

rtl/user/ lec_user_constraints.tcl

```
add_pin_constraints 0 ipdb_spmi2_pdft_scen_i -both
add_ignored_outputs *test_so* -both
add_ignored_inputs *test_si* -both
```

iwlec run_ec -id rtlvsgate

iwlec run_ec -id rtlvsgate

5 References

No/Name	Title	Source
1	X-GOLD 8x6 A0 SPMI2 Polaris, Hardware Design Document 0.87	Project document storage
2	MIPI® Alliance Specification for System Power Management Interface (SPMI) Version 2.0	www.mipi.com ipdb_spmi2/docs/ mipi_SPMI_specification_v2-0.pdf
3	Register Description	Ipdb_spmi2/docs/INTEL_IPDB_SPMI2.html
4	SmartDV Technologies MIPI SPMI Slave IIP Databook	Ipdb_spmi2/docs/sdvt_mipi_spmi_slave_databook.pdf
5	TOPSPIN 4.7.0 Architecture Specification	Ipdb_topspin/4.7.0/docs/Architecture_Specification
6	Event Bus Specification 0.4	Project document storage