Rapid Prototyping of Application Specific Signal Processors

RASSP

Process - Architecture - Framework

Rapid Prototyping of Application—Specific Signal Processors (RASSP)

RASSP Model Year Architecture Specification Volume IV

Standard Virtual Interface Specification Appendices

Version 1.0

November 25, 1996

ADVANCED TECHNOLOGY LABORATORIES

LOCKHEED MARTIN

Appendix I. svi_types_pkg.vhd

This VHDL file should be used by all SVI encapsulations. It will keep SVI port definitions compatible and will also keep all encapsulations current with evolving SVI commands, aborts, and errors.

```
--------------------------------------------------------------------
-- Program:              RASSP
-- Project:              Model Year Architecture
-- Title:                SVI Types Package
-- Description:          This file contains constant and type defs
--                       which should be used by all encapsulations.
-- Library:              WORK
-- Company:              Lockheed Martin – Advanced Technology Laboratories
-- Author Info:          Greg Buchanan (from David Mazik)
--                       email: gbuchana@atl.lmco.com
--                       phone: (609) 338-4250
--                       address:  A&E-2W
--                                 1 Federal St.
--                                 Camden,  NJ 08102
-- Filename:             svi_types_pkg_.vhd
-- Revision History:     Date       Author       Purpose
--                       ----------------------------------------------
-- Known bugs:
------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.util_1164.all;

PACKAGE svi_types_pkg IS

   -- WIDTH of all multi-bit SVI Signals(except data signal)
   ------------------------------------------------

   -- width of channel id signal
   CONSTANT channel_id_width : INTEGER := 4;

   -- width of all abort signals
   CONSTANT abort_width : INTEGER := 4;

   -- width of all error signals
   CONSTANT error_width : INTEGER := 4;
```

```
 –– constants for SVI Commands
 ––––––––––––––––––––––––––––––––––––––––––––––
 –– 1 = External Write
 –– 2 = Internal Write Request
 –– 3 = External Read Request
 –– 4 = Internal Read Request
 –– 5 = External Read Response
 –– 6 = Internal Read Response

 –– width of command on data bus
 CONSTANT cmd_width : INTEGER := 8;

 CONSTANT ext_write_cmd_constant : std_logic_vector
                                 (cmd_width–1 DOWNTO 0)
                                 := to_vector(1,cmd_width);
 CONSTANT int_write_cmd_constant : std_logic_vector
                                 (cmd_width–1 DOWNTO 0)
                                 := to_vector(2,cmd_width);
 CONSTANT ext_read_req_cmd_constant : std_logic_vector
                                 (cmd_width–1 DOWNTO 0)
                                 := to_vector(3,cmd_width);
 CONSTANT int_read_req_cmd_constant : std_logic_vector
                                 (cmd_width–1 DOWNTO 0)
                                 := to_vector(4,cmd_width);
 CONSTANT ext_read_resp_cmd_constant : std_logic_vector
                                 (cmd_width–1 DOWNTO 0)
                                 := to_vector(5,cmd_width);
 CONSTANT int_read_resp_cmd_constant : std_logic_vector
                                 (cmd_width–1 DOWNTO 0)
                                 := to_vector(6,cmd_width);

–– constants for Abort Signals
–––––––––––––––––––––––––––––––––––––––––––––––––––
––  The meanings of these abort signals are left to the
––  encapsulation dsgnr.
––  He/she may chose which code to use under different conditions and also
––  may chose how the encapsulation will react to different abort codes.
––  0 = No abort.
––  1 = Abort due to System Error Detected.
––  2 = Abort due to Data Error Detected.
––  3 = Abort due to Refused Request
––  4 = Abort due to Destination Lost
––  5 = Abort due to Source Lost
––  6 = Abort due to Synchronization Error
```

```
  --  (others tbd)
  CONSTANT no_abort_constant : std_logic_vector
                                  (abort_width-1 DOWNTO 0)
                                   := to_vector(0, abort_width);
  CONSTANT abort_sys_err_constant : std_logic_vector
                                  (abort_width-1 DOWNTO 0)
                                  := to_vector(1, abort_width);
  CONSTANT abort_data_err_constant : std_logic_vector
                                  (abort_width-1 DOWNTO 0)
                                  := to_vector(2, abort_width);
  CONSTANT abort_ref_constant : std_logic_vector
                                  (abort_width-1 DOWNTO 0)
                                  := to_vector(3, abort_width);
  CONSTANT abort_dest_lost_constant : std_logic_vector
                                  (abort_width-1 DOWNTO 0)
                                    := to_vector(4, abort_width);
  CONSTANT abort_src_lost_constant : std_logic_vector
                                  (abort_width-1 DOWNTO 0)
                                  := to_vector(5, abort_width);
  CONSTANT abort_sync_err_constant : std_logic_vector
                                  (abort_width-1 DOWNTO 0)
                                   := to_vector(6, abort_width);

  -- constants for Error Signals
  ----------------------------------------------------
  --  The meanings of these error signals are left to the encapsulation
  --  dsgnr.
  --  He/she may chose which code to use under different conditions and also
  --  may chose how the encapsulation will react to different error codes.
  --  0 = No error.
  --  1 = System Error.
  --  2 = Data Error.(Parity, edac, crc,  etc)
  --  3 = Synchronization Error. (Overrun or underrun)
  --  (others tbd)
  CONSTANT no_error_constant : std_logic_vector(error_width-1 DOWNTO 0) :=
                                to_vector(0, error_width);
  CONSTANT sys_err_constant : std_logic_vector(error_width-1 DOWNTO 0) :=
                               to_vector(1, error_width);
  CONSTANT data_err_constant : std_logic_vector(error_width-1 DOWNTO 0) :=
                                to_vector(2, error_width);
  CONSTANT sync_err_constant : std_logic_vector(error_width-1 DOWNTO 0) :=
                                to_vector(3, error_width);
END svi_types_pkg;
```

Appendix II. SVI Encapsulation Template

This SVI encapsulation template is provided to aid in the design of the structure of a new SVI encapsulation. All of the logic for the encapsulation must still be designed. All VHDL entities are described by two VHDL files. The entity description is contained in the file ending with the text ”...ent.vhd.” The architecture description is contained in the file ending with the text ”...arch.vhd.” This allows for easier debugging and modification. If the behavior of an entity changes, only that entity’s architecture must be recompiled, unless the entity’s port definition changes as well. This template is designed for an encapsulation of a PE or an interface without using any COTS interface controller. An encapsulation using a COTS interface controller can easily be developed by referring to section 4.1.3 in Volume I. and this template. All places which need to be changed by the encapsulation designer are marked with ’x’, ’X’, or ’#### MODIFY ####.’ The string ”xxx” in the filenames and entities should be replaced with the actual name of the item being encapsulated to establish a common naming convention. NO TAG shows how all of the entities contained in the template are hierarchically arranged. The names of the entities included in the template appear in italics. The ’xxx’ contained in the entity names should be replaced with the name of the PE or interconnect fabric being encapsulated.

The SVI_xxx_master is responsible for sending all outgoing SVI messages from the encapsulated entity. The SVI_xxx_slave is responsible for routing all incoming SVI messages through the fifo to the xxx_master which interfaces to the encapsulated entity. The fifo and xxx_master are not italicized because they are not contained in this template. This is due to the fact that they will be completely determined by which COTS fifo may be used and what entity is being encapsulated. This storage may also be implemented as part of the SVI_xxx_wrapper as a macro–cell if the encapsulation is targeted for a FPGA or ASIC. The SVI_xxx_wrapper serves to partition all logic which must be synthesized from COTS components which are already realized, such as the fifo. This will help the encapsulation end–user easily separate out this logic which must be synthesized from the encapsulation. Note that the block labeled encapsulated entity may or may not be an actual physical block contained within the encapsulation. If the encapsulation is for a PE, then the encapsulated entity will physically exist in the encapsulation, containing one or more processors along with any DMA and memory system necessary. However, in the case of an interconnect fabric, there would be no physical ”encapsulated entity” in the encapsulation, even though the interface is said to be encapsulated. The interconnect is implemented by the encapsulation. The files contained in the template are listed below.

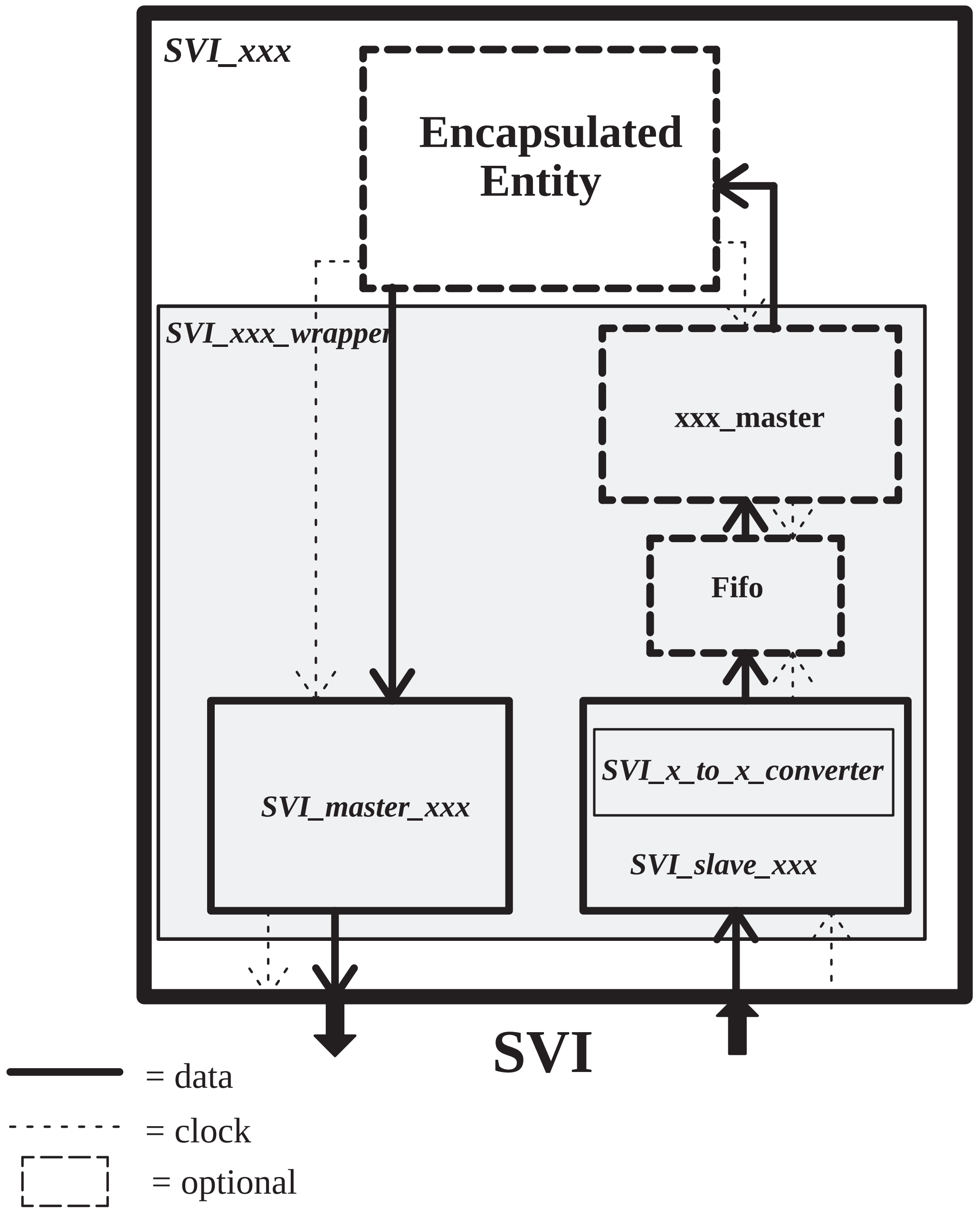

Figure A–1 Encapsulation Template Structure

II.i. VHDL Source Files

The source files are listed below for the SVI encapsulation template. These files will be available in softcopy in the near feature via electronic distribution.

II.i.i. svi_xxx_types_pkg.vhd

This file is used for defining all constants, typedefs, procedures, and functions which are common to files throughout the encapsulation here.

```
-----------------------------------------------------------------
-- Program:             RASSP
-- Project:             Model Year Architecture
-- Title:               SVI XXX Types Package
-- Description:         This file contains constants ,type defs
--                      and procedures/functions
--                      which are used with
--                      SVI encapsulation.
-- Library:
-- Filename:            svi_xxx_pkg.vhd
-- Company:
-- Author Info:         name, email address , phone number,
--                      mail address
-- Revision History:    Date       Author       Purpose
--                      -----------------------------------------
-- Known bugs:
-------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.util_1164.all;

PACKAGE svi_xxx_types_pkg IS

##################### MODIFY #################
-- put type defs, constants, and common procs
-- and functions which will be used throughout
-- the encapsulation but not outside of this
-- encapsulation.
##################### MODIFY #################
   CONSTANT xxx_width : INTEGER := XXX; -- width of svi_data in and out
                                        -- for this encapsulation

END svi_xxx_types_pkg;
```

II.i.ii. svi_xxx_ent.vhd

This file contains the VHDL entity description for the encapsulation. It needs the encapsulated side of the port description added.

```
-----------------------------------------------------------------
-- Program:             RASSP
-- Project:             Model Year Architecture
-- Title:               SVI XXX encapsulation – Entity
-- Description:         This file contains entity description for
--                      encapsulation for xxx.
-- Library:
-- Filename:            svi_xxx_ent.vhd
-- Company:
-- Author Info:         name, email address , phone number,
--                      mail address
-- Revision History:    Date       Author       Purpose
--                      -----------------------------------------
-- Known bugs:
-------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ENTITY svi_xxx is
   PORT (

         -------------------------------------------
         -- SVI SIDE of ENCAPSULATION
         -------------------------------------------

         -- DATA INPUT INTERFACE(Relative to PE) (SVI SLAVE)
         -------------------------------------------------------------
         ########################## MODIFY ########################
         -- specify expected incoming data width here with
         -- constant xxx_width defined in svi_xxx_types_pkg_.vhd
         svi_data_in    :           IN     std_logic_vector(
                                           xxx_width-1 DOWNTO 0);
         ########################## MODIFY ########################
         svi_last_word_in :         IN     std_logic;
         svi_channel_id_in :        IN     std_logic_vector(
                                           channel_id_width-1
                                           DOWNTO 0);
         svi_clock_in :             IN     std_logic;
         svi_xfer_request_in :      IN     std_logic;
         svi_ready_out :               OUT std_logic;
         svi_data_valid_in :        IN     std_logic;
```

```
svi_slave_abort_in :       IN     std_logic_vector(
                                  abort_width–1
                                  DOWNTO 0);
svi_slave_abort_out:         OUT std_logic_vector(
                                  abort_width–1
                                  DOWNTO 0);
svi_slave_error_in :       IN     std_logic_vector(
                                  error_width–1
                                  DOWNTO 0);
svi_slave_error_out :         OUT std_logic_vector(
                                  error_width–1
                                  DOWNTO 0);

–– DATA OUTPUT INTERFACE(Relative to PE) (SVI MASTER)
–––––––––––––––––––––––––––––––––––––––––––––––––
########################## MODIFY #######################
–– specify expected incoming data width here with
–– constant xxx_width defined in svi_xxx_types_pkg_.vhd
svi_data_out    :              OUT std_logic_vector(
                                   xxx_width–1 DOWNTO 0);
########################## MODIFY #######################
svi_last_word_out :            OUT std_logic;
svi_channel_id_out :           OUT std_logic_vector(
                                   channel_id_width–1
                                   DOWNTO 0);
svi_clock_out :                OUT std_logic;
svi_xfer_request_out :         OUT std_logic;
svi_ready_in :              IN     std_logic;
svi_data_valid_out :           OUT std_logic;

svi_master_abort_in :       IN     std_logic_vector(
                                   abort_width–1
                                   DOWNTO 0);
svi_master_abort_out :         OUT std_logic_vector(
                                   abort_width–1
                                   DOWNTO 0);
svi_master_error_in :       IN     std_logic_vector(
                                   error_width–1
                                   DOWNTO 0);
svi_master_error_out :         OUT std_logic_vector(
                                   error_width–1
                                   DOWNTO 0);

–– INTERRUPTS, SYSTEM CLOCKS, RESETS
––––––––––––––––––––––––––––––––––––––––
svi_interrupt_in :  IN     std_logic;
svi_interrupt_out :    OUT std_logic;
```

```
         svi_sclock_in  : IN std_logic;
         svi_sclock_out : IN std_logic;

         svi_sreset_in :  IN     std_logic;
         svi_sreset_out :    OUT std_logic;

         ################### MODIFY #######################
         -- put interface or pe signals here
         ################### MODIFY #######################

       );

END svi_xxx;
```

II.i.iii. svi_xxx_arch.vhd

This files contains the VHDL architecture description for the encapsulation. The architecture instantiates the svi_xxx_wrapper and any other components necessary to implement the encapsulation.

```
--------------------------------------------------------------------
-- Program:               RASSP
-- Project:               Model Year Architecture
-- Title:                 SVI xxx encapsulation - Architecture
-- Description:           This file contains arch. description for
--                        encapsulation for xxx.
-- Library:
-- Filename:              svi_xxx_arch.vhd
-- Company:
-- Author Info:           name, email address , phone number,
--                        mail address
-- Revision History:      Date       Author       Purpose
--                        ----------------------------------------------
-- Known bugs:
------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ARCHITECTURE structural OF svi_xxx IS

COMPONENT svi_xxx_wrapper
   PORT (

         ------------------------------------------
         -- SVI SIDE of ENCAPSULATION
         ------------------------------------------
```

```
–– DATA INPUT INTERFACE(Relative to PE) (SVI SLAVE)
––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
###################### MODIFY ###########################
–– specify expected incoming data width here with
–– constant xxx_width defined in svi_xxx_types_pkg
svi_data_in    :             IN   std_logic_vector(
                                  xxx_width–1 DOWNTO 0);
###################### MODIFY ###########################
svi_last_word_in :         IN     std_logic;
svi_channel_id_in :        IN     std_logic_vector(
                                  channel_id_width–1
                                  DOWNTO 0);
svi_clock_in :             IN     std_logic;
svi_xfer_request_in :      IN     std_logic;
svi_ready_out :               OUT std_logic;
svi_data_valid_in :        IN     std_logic;
svi_slave_abort_in :       IN     std_logic_vector(
                                  abort_width–1
                                  DOWNTO 0);
svi_slave_abort_out :         OUT std_logic_vector(
                                  abort_width–1
                                  DOWNTO 0);
svi_slave_error_in :       IN     std_logic_vector(
                                  error_width–1
                                  DOWNTO 0);
svi_slave_error_out :         OUT std_logic_vector(
                                  error_width–1
                                  DOWNTO 0);
–– DATA OUTPUT INTERFACE(Relative to PE) (SVI MASTER)
––––––––––––––––––––––––––––––––––––––––––––––––
####################### MODIFY ###########################
–– specify expected outgoing data width here
–– with constant xxx_width defined in svi_xxx_types_pkg_.vhd
svi_data_out    :             OUT std_logic_vector(
                                  xxx_width–1 DOWNTO 0);
####################### MODIFY ###########################
svi_last_word_out :           OUT std_logic;
svi_channel_id_out :          OUT std_logic_vector(
                                  channel_id_width–1
                                  DOWNTO 0);
svi_clock_out :               OUT std_logic;
svi_xfer_request_out :        OUT std_logic;
svi_ready_in :             IN     std_logic;
svi_data_valid_out :          OUT std_logic;

svi_master_abort_in :      IN     std_logic_vector(
                                  abort_width–1
```

```
                                        DOWNTO 0);
         svi_master_abort_out :         OUT std_logic_vector(
                                        abort_width-1
                                        DOWNTO 0);
         svi_master_error_in :       IN     std_logic_vector(
                                        error_width-1
                                        DOWNTO 0);
         svi_master_error_out :         OUT std_logic_vector(
                                        error_width-1
                                        DOWNTO 0);

         -- INTERRUPTS, SYSTEM CLOCKS, RESETS
         ----------------------------------------
         svi_interrupt_in :  IN     std_logic;
         svi_interrupt_out :    OUT std_logic;

         svi_sclock_in  : IN std_logic;
         svi_sclock_out : IN std_logic;

         svi_sreset_in :  IN     std_logic;
         svi_sreset_out :    OUT std_logic;

         ####################### MODIFY ###########################
         -- put interface/encapsulation signals here
         ####################### MODIFY ###########################
       );

END COMPONENT;
FOR ALL : svi_xxx_wrapper USE ENTITY WORK.svi_xxx_wrapper(mixed);

######################### MODIFY ###########################
-- OPTIONAL:  put other encapsulated entity and any
-- non-svi encapsulation components here.  Should
-- be already-existing actual hw models.(i.e. - fifo's,
-- COTs ifc chips, PE's etc)
######################### MODIFY ###########################

######################### MODIFY ###########################
--  define interconnecting signals here
######################### MODIFY ###########################

BEGIN -- ARCHITECTURE structural

inst_svi_xxx_wrapper : svi_xxx_wrapper
   PORT MAP (

         -- DATA INPUT INTERFACE
         -------------------------------------------
         svi_data_in => svi_data_in,
         svi_last_word_in  => svi_last_word_in,
         svi_channel_id_in  => svi_channel_id_in,
         svi_clock_in  => svi_clock_in,
         svi_xfer_request_in  => svi_xfer_request_in,
         svi_ready_out  => svi_ready_out,
         svi_data_valid_in  => svi_data_valid_in,
         svi_slave_abort_in  => svi_slave_abort_in,
         svi_slave_abort_out  => svi_slave_abort_out,
         svi_slave_error_in  => svi_slave_error_in,
         svi_slave_error_out  => svi_slave_error_out,

         -- DATA OUTPUT INTERFACE(Relative to PE) (SVI MASTER)
         -------------------------------------------------
         svi_data_out => svi_data_out,
         svi_last_word_out  => svi_last_word_out,
         svi_channel_id_out  => svi_channel_id_out,
         svi_clock_out  => svi_clock_out,
         svi_xfer_request_out  => svi_xfer_request_out,
         svi_ready_in  => svi_ready_in,
         svi_data_valid_out  => svi_data_valid_out,
         svi_master_abort_in  => svi_master_abort_in,
         svi_master_abort_out  => svi_master_abort_out,
         svi_master_error_in  => svi_master_error_in,
         svi_master_error_out  => svi_master_error_out,

         -- INTERRUPTS, SYSTEM CLOCKS, RESETS
         ----------------------------------------
         svi_interrupt_in  => svi_interrupt_in,
         svi_interrupt_out  => svi_interrupt_out,
         svi_sclock_in  => svi_sclock_in,
         svi_sclock_out  => svi_sclock_out,
         svi_sreset_in  => svi_sreset_in,
         svi_sreset_out  => svi_sreset_out,

         ####################### MODIFY ###########################
         --  put other interconnect signals here for encapsulation,
         --  ifc
         ####################### MODIFY ###########################

        );

   ######################### MODIFY ###########################
   --  OPTIONAL: put other component instantions here – Fifo?, PE?, DMA?
   ######################### MODIFY ###########################

END structural;
```

II.i.iv. svi_xxx_wrapper_ent.vhd

This files contains the VHDL entity description for the SVI wrapper. The SVI wrapper distinguishes all logic which needs to be synthesized to implement SVI from all components necessary to implement the encapsulation.

```
--------------------------------------------------------------------
-- Program:             RASSP
-- Project:             Model Year Architecture
-- Title:               SVI XXX Wrapper - entity
-- Description:         This file defines entity for logic needed to
--                      encapsulate element xxx.
-- Filename:            svi_xxx_wrapper_ent.vhd
-- Company:
-- Author Info:         name, email address , phone number,
--                      mail address
-- Revision History:    Date       Author       Purpose
--                      ----------------------------------------------
-- Known bugs:
-------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ENTITY svi_xxx_wrapper is
   PORT (

         -- DATA INPUT INTERFACE(SVI SLAVE)
         ------------------------------------------------------------------
         ##################### MODIFY ########################
         -- Specify incoming data_width here with
         -- constant xxx_width defined in svi_xxx_types_pkg
         svi_data_in    :           IN     std_logic_vector(
                                           xxx_width-1 DOWNTO 0);
         ##################### MODIFY ########################
         svi_last_word_in :         IN     std_logic;
         svi_channel_id_in :        IN     std_logic_vector(
                                           channel_id_width-1
                                           DOWNTO 0);
         svi_clock_in :             IN     std_logic;
         svi_xfer_request_in :      IN     std_logic;
         svi_ready_out :               OUT std_logic;
         svi_data_valid_in :        IN     std_logic;

         svi_slave_abort_in :       IN     std_logic_vector(
                                           abort_width-1
                                           DOWNTO 0);
```

```
svi_slave_abort_out :         OUT std_logic_vector(
                                  abort_width–1
                                  DOWNTO 0);
svi_slave_error_in :       IN     std_logic_vector(
                                  error_width–1
                                  DOWNTO 0);
svi_slave_error_out :         OUT std_logic_vector(
                                  error_width–1
                                  DOWNTO 0);

–– DATA OUTPUT INTERFACE(SVI MASTER)
-------------------------------------------------
##################### MODIFY ########################
–– Specify outgoing data width here
–– with constant xxx_width defined in svi_xxx_types_pkg
svi_data_out    :              OUT std_logic_vector(
                                  xxx_width–1 DOWNTO 0);
##################### MODIFY ########################
svi_last_word_out :            OUT std_logic;
svi_channel_id_out :           OUT std_logic_vector(
                                   channel_id_width–1
                                   DOWNTO 0);
svi_clock_out :                OUT std_logic;
svi_xfer_request_out :         OUT std_logic;
svi_ready_in :              IN     std_logic;
svi_data_valid_out :           OUT std_logic;

svi_master_abort_in :       IN     std_logic_vector(
                                   abort_width–1
                                   DOWNTO 0);
svi_master_abort_out :         OUT std_logic_vector(
                                   abort_width–1
                                   DOWNTO 0);
svi_master_error_in :       IN     std_logic_vector(
                                   error_width–1
                                   DOWNTO 0);
svi_master_error_out :         OUT std_logic_vector(
                                   error_width–1
                                   DOWNTO 0);

–– INTERRUPTS, SYSTEM CLOCKS, RESETS
----------------------------------------
svi_interrupt_in :  IN     std_logic;
svi_interrupt_out :    OUT std_logic;

svi_sclock_in  : IN std_logic;
svi_sclock_out : IN std_logic;
```

```
          svi_sreset_in :  IN     std_logic;
          svi_sreset_out :    OUT std_logic;

          ##################### MODIFY #######################
          -- Specify signals going to ifc or pe side of wrapper
          ##################### MODIFY #######################


        );

END svi_xxx_wrapper;
```

II.i.v. svi_xxx_wrapper_arch.vhd

This file contains the architecture description for the SVI wrapper. It instantiates the SVI master and SVI slave.

```
------------------------------------------------------------
-- Program:               RASSP
-- Project:               Model Year Architecture
-- Title:                 SVI xxx wrapper architecture
-- Description:           This file contains arch definition for
--                        svi_wrapper
--                        needed to impelment encapsulation for xxx.
-- Library:
-- Filename:              svi_xxx_wrapper_arch.vhd
-- Company:
-- Author Info:           name, email address , phone number,
--                        mail address
-- Revision History:      Date       Author       Purpose
--                        -------------------------------------------
-- Known bugs:
----------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ARCHITECTURE mixed OF svi_xxx_wrapper IS

COMPONENT xxx_ifc
   PORT (
         ####################### MODIFY ##########################
         -- put signals for logic block implementing master of interconnect
         --  or pe
         -- (or the block that takes data from svi_slave fifo and
         -- outputs
         -- it to pe or interconnect)
         ####################### MODIFY ##########################
        );
END COMPONENT;

COMPONENT svi_master_xxx
   PORT (
         ############## MODIFY #################
         -- put incoming xxx_ifc, other encapsulation
         -- signals here
         ############## MODIFY #################

         ---------------------------
         -- SIGNALS GOING OUT TO SVI
         ---------------------------

         -- DATA OUTPUT INTERFACE(Relative to PE)
         ----------------------------------------
         ############## MODIFY #################
         -- put expected outgoing data_width here
         svi_data_out    :              OUT std_logic_vector(
                                            xxx_width-1 DOWNTO 0);
         ############## MODIFY #################
         svi_last_word_out :            OUT std_logic;
         svi_channel_id_out :           OUT std_logic_vector(
                                            channel_id_width-1
                                            DOWNTO 0);
         svi_clock_out :                OUT std_logic;
         svi_xfer_request_out :         OUT std_logic;
         svi_ready_in :              IN     std_logic;
         svi_data_valid_out :           OUT std_logic;

         svi_master_abort_in :       IN     std_logic_vector(
                                            abort_width-1
                                            DOWNTO 0);
         svi_master_abort_out :         OUT std_logic_vector(
                                            abort_width-1
                                            DOWNTO 0);
         svi_master_error_in :       IN     std_logic_vector(
                                            error_width-1
                                            DOWNTO 0);
         svi_master_error_out :         OUT std_logic_vector(
                                            error_width-1
                                            DOWNTO 0);


         -- INTERRUPTS, SYSTEM CLOCKS, RESETS
         ----------------------------------------
         svi_interrupt_in :  IN     std_logic;
         svi_interrupt_out :    OUT std_logic;
        );

END COMPONENT;

COMPONENT svi_slave_xxx
   PORT (

         -------------------------------
         -- SIGNALS COMING IN FROM SVI
         -------------------------------

         -- DATA INPUT INTERFACE(Relative to PE)
         ----------------------------------------
         ################### MODIFY #########################
         -- specify incoming data_width here
         svi_data_in    :           IN     std_logic_vector(
                                           xxx_width-1 DOWNTO 0);
         ################### MODIFY #########################
         svi_last_word_in :         IN     std_logic;
         svi_channel_id_in :        IN     std_logic_vector(
                                           channel_id_width-1
                                           DOWNTO 0);
         svi_clock_in :             IN     std_logic;
         svi_xfer_request_in :      IN     std_logic;
         svi_ready_out :               OUT std_logic;
         svi_data_valid_in :        IN     std_logic;
         svi_slave_abort_in :       IN     std_logic_vector(
                                           abort_width-1
                                           DOWNTO 0);
         svi_slave_abort_out :         OUT std_logic_vector(
                                           abort_width-1
                                           DOWNTO 0);
         svi_slave_error_in :       IN     std_logic_vector(
                                           error_width-1
                                           DOWNTO 0);
         svi_slave_error_out :         OUT std_logic_vector(
                                           error_width-1
                                           DOWNTO 0);

         -- INTERRUPTS, SYSTEM CLOCKS, RESETS
         ----------------------------------------
         svi_interrupt_in :  IN     std_logic;

         ------------------------------------------------------
         -- END OF  SIGNALS GOING TO SVI
         ------------------------------------------------------

         ################### MODIFY #########################
         -- put outgoing encapsulation, ifc, or pe signals here
         ################### MODIFY #########################
        );

END COMPONENT;

################### MODIFY #########################
––– Put signals used to interconnect components here
################### MODIFY #########################

BEGIN –– architecture mixed of svi_xxx_wrapper

################### MODIFY #########################
–– Specify xxx_ifc component signals
inst_xxx_ifc :  xxx_ifc
   PORT MAP(
        );
################### MODIFY #########################

inst_svi_master_xxx : svi_master_xxx
   PORT MAP(
         ############## MODIFY #################
         –– put incoming xxx_ifc, other encapsulation
         –– signals here
         ############## MODIFY #################

         –– DATA OUTPUT INTERFACE(Relative to PE)
         ––––––––––––––––––––––––––––––––––––––––
         svi_data_out              => svi_data_out,
         svi_last_word_out         => svi_last_word_out,
         svi_channel_id_out        => svi_channel_id_out,
         svi_clock_out             => svi_clock_out,
         svi_xfer_request_out      => svi_xfer_request_out,
         svi_ready_in              => svi_ready_in,
         svi_data_valid_out        => svi_data_valid_out,
         svi_master_abort_in       => svi_master_abort_in,
         svi_master_abort_out      => svi_master_abort_out,
         svi_master_error_in       => svi_master_error_in,
         svi_master_error_out      => svi_master_error_out,

         –– INTERRUPTS, SYSTEM CLOCKS, RESETS
         ––––––––––––––––––––––––––––––––––––––––
         svi_interrupt_in          => svi_interrupt_in,
         svi_interrupt_out         => svi_interrupt_out,

        );

inst_svi_slave_xxx : svi_slave_xxx
   PORT  MAP(
         –– SIGNALS COMING IN FROM SVI
         ––––––––––––––––––––––––––––––––
         -- DATA INPUT INTERFACE(Relative to PE)
         ----------------------------------------
         svi_data_in                     => svi_data_in,
         svi_last_word_in                => svi_last_word_in,
         svi_channel_id_in               => svi_channel_id_in,
         svi_clock_in                    => svi_clock_in,
         svi_xfer_request_in             => svi_xfer_request_in,
         svi_ready_out                   => svi_ready_out,
         svi_data_valid_in               => svi_data_valid_in,
         svi_slave_abort_in              => svi_slave_abort_in,
         svi_slave_abort_out             => svi_slave_abort_out,
         svi_slave_error_in              => svi_slave_error_in,
         svi_slave_error_out             => svi_slave_error_out,

         -- INTERRUPTS, SYSTEM CLOCKS, RESETS
         ----------------------------------------
         svi_interrupt_in                => svi_interrupt_in,

         ################### MODIFY #########################
         -- put outgoing encapsulation, ifc, or pe signals here
         ################### MODIFY #########################

        );


################### MODIFY #########################
-- put any misc. logic here
################### MODIFY #########################

END mixed;
```

II.i.vi. svi_master_xxx_ent.vhd

This file contains the VHDL entity description for the SVI master.

```
--------------------------------------------------------------------
-- Program:             RASSP
-- Project:             Model Year Architecture
-- Title:               SVI Master xxx  Entity Definition
-- Description:         This file contains the entity def. for
--                      outgoing SVI state machine to encapsulate
--                      xxx.
-- Library:             xxx put vhdl library here
-- Filename:            svi_master_xxx_ent.vhd
-- Company:
-- Author Info:         name, email address, phone number,
--                      mail address
-- Revision History:    Date       Author       Purpose
--                      ---------------------------------------------
-- Known bugs:
-----------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ENTITY svi_master_xxx IS
   PORT (
         ############## MODIFY #################
         -- put incoming xxx_ifc, other encapsulation
         -- signals here
         ############## MODIFY #################

         ---------------------------
         -- SIGNALS GOING OUT TO SVI
         ---------------------------

         -- DATA OUTPUT INTERFACE(Relative to PE)
         ----------------------------------------
         ############## MODIFY #################
         -- put data width out constant here
         svi_data_out    :              OUT std_logic_vector(
                                            xxx-1 DOWNTO 0);
         ############## MODIFY #################

         svi_last_word_out :            OUT std_logic;
         svi_channel_id_out :           OUT std_logic_vector(
                                            channel_id_width-1
                                            DOWNTO 0);
         svi_clock_out :                OUT std_logic;
         svi_xfer_request_out :         OUT std_logic;
         svi_ready_in :              IN     std_logic;
         svi_data_valid_out :           OUT std_logic;

         svi_master_abort_in :       IN     std_logic_vector(
                                            abort_width-1
                                            DOWNTO 0);
         svi_master_abort_out :         OUT std_logic_vector(
                                            abort_width-1
                                            DOWNTO 0);
         svi_master_error_in :       IN     std_logic_vector(
                                         error_width–1
                                         DOWNTO 0);
         svi_master_error_out :         OUT std_logic_vector(
                                         error_width–1
                                         DOWNTO 0);


         –– INTERRUPTS, SYSTEM CLOCKS, RESETS
         ––––––––––––––––––––––––––––––––––––––––
         svi_interrupt_in :  IN     std_logic;
         svi_interrupt_out :    OUT std_logic;

        );
END svi_master_xxx;
```

II.i.vii. svi_master_xxx_arch.vhd

This file contains the behavioral architecture description for the SVI master.

```
––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
–– Program:             RASSP
–– Project:             Model Year Architecture
–– Title:               SVI Master xxx Architecture
–– Description:         This file contains the arch. def. for
––                      outgoing SVI state machine necessary to
––                      encapsulate xxx.
–– Library:             xxx put vhdl library here xxx
–– Filename:            svi_master_xxx_arch.vhd
–– Company:
–– Author Info:         name, email address , phone number,
––                      mail address
–– Revision History:    Date       Author       Purpose
––                      ––––––––––––––––––––––––––––––––––––––––––––––
–– Known bugs:
–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ARCHITECTURE behave OF svi_master_xxx IS

   TYPE state_type  IS (
                        idle,
                        ########## MODIFY ############
                        –– add states here
                        ########## MODIFY ############
                        );
   -- curr_state will be assigned nxt_state
   -- on rising clock edge
   SIGNAL nxt_state : state_type := idle;
   SIGNAL curr_state : state_type := idle;


BEGIN
   -- Asynch SIGNAL Assignments
   ----------------------------
   ########## MODIFY ###############################
   -- put any asynchronous signal assignments here
   ########## MODIFY ###############################

-- this process handles reset, updates state and also
-- controls all synchronous data signals
########## MODIFY #######################################
-- put clock name here for xxx_clock, reset for xxx_reset
########## MODIFY #######################################
clock : PROCESS(xxx_clock, xxx_reset)

BEGIN

   IF xxx_reset = '1' THEN
      curr_state <= idle;

   -- clock edge
   ELSIF xxx_clock'EVENT AND xxx_clock = '1' THEN

      ##################### MODIFY ######################
      -- based on curr_state and inputs,  drive data/synchronous
      -- signals here
      ##################### MODIFY ######################


      -- update state on rising clock edge
      curr_state <= nxt_state;

   END IF; -- if rising clock edge or reset

END PROCESS clock;

-- This process updates nxt_state based on changes on sensitivity
-- list
state_machine : PROCESS(curr_state, svi_master_error_in,
                        svi_master_abort_in, svi_ready_in, xxx);
      ##################### MODIFY ######################
      -- Add ifc xxx control signals to sensitivity list
      ##################### MODIFY ######################
```

```
BEGIN –– state_machine process

   –– Make sure all outputs get default assignments
   –– to avoid creating latches instead of combinational
   –– logic, will be overridden with different values determined
   –– by curr_state with same prop. delay
   ––––––––––––––––––––––––––––––––––––––––––––––––––––––––

   –– SVI signals
   svi_xfer_request_out <= ’0’;
   svi_master_abort_out <= no_abort_constant;
   svi_master_error_out <= no_error_constant;
   ##################### MODIFY #######################
   –– Add ifc xxx output signals here
   ##################### MODIFY #######################


   –– prevent pulse on control signal from incorrectly
   –– updating state
   nxt_state <= curr_state;

   CASE curr_state IS

      WHEN idle =>
      ––––––––––––––––––––––––––––––––––––––––––––––
      ##################### MODIFY #######################
      –– Add states and logic here
      ##################### MODIFY #######################

   END CASE;

END PROCESS state_machine;

END PROCESS test_state_machine;

interrupt_process : PROCESS(svi_interrupt_in)
BEGIN
   –– TBD TBD TBD TBD
   ASSERT (FALSE)
      REPORT ” Interrupt interface not implemented.”
      SEVERITY note;
END PROCESS interrupt_process;

END behave;
```

II.i.viii. svi_slave_xxx_ent.vhd

This file contains the VHDL entity description for the SVI slave.

```
--------------------------------------------------------------------
-- Program:             RASSP
-- Project:             Model Year Architecture
-- Title:               SVI Slave XXX Entity Definition
-- Description:         This file contains the entity description
--                      for
--                      svi slave for xxx interface/pe.
--                      Takes incoming info from svi and routes to
--                      xxx.
-- Library:
-- Filename:            svi_slave_xxx_ent.vhd
-- Company:
-- Author Info:         name, email address , phone number,
--                      mail address
-- Revision History:    Date       Author       Purpose
--                      ----------------------------------------------
-- Known bugs:
------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ENTITY svi_slave_xxx is
   PORT (

         -------------------------------
         -- SIGNALS COMING IN FROM SVI
         -------------------------------

         -- DATA INPUT INTERFACE(Relative to PE)
         ----------------------------------------
         ####################### MODIFY #######################
         -- put expected data input width here
         svi_data_in    :          IN     std_logic_vector(
                                          xxx_width-1 DOWNTO 0);
         ####################### MODIFY #######################
         svi_last_word_in :        IN     std_logic;
         -- used width of address/data bus, not maximum
         -- remains constant once instantiated
         -- specifies which channel in interconnect message came
         -- from
         svi_channel_id_in :       IN     std_logic_vector(
                                          channel_id_width-1
                                          DOWNTO 0);
         -- data clock, not system clock
         svi_clock_in :            IN     std_logic;
         -- svi ifc wishes to send data
         svi_xfer_request_in :     IN     std_logic;
         -- tell svi ifc when ready to accept data
         svi_ready_out :              OUT std_logic;
         -- svi ifc has valid data on adbus
         svi_data_valid_in :        IN     std_logic;
         --  used by master or slave to abort
         svi_slave_abort_in :       IN     std_logic_vector(
                                           abort_width-1 DOWNTO 0);
         svi_slave_abort_out :         OUT std_logic_vector(
                                           abort_width-1
                                           DOWNTO 0);
         svi_slave_error_in :       IN     std_logic_vector(
                                           error_width-1
                                           DOWNTO 0);
         svi_slave_error_out :         OUT std_logic_vector(
                                           error_width-1
                                           DOWNTO 0);

         -- INTERRUPTS, SYSTEM CLOCKS, RESETS
         ----------------------------------------
         svi_interrupt_in :  IN     std_logic;

         ------------------------------------------------------
         -- END OF  SIGNALS GOING TO SVI
         ------------------------------------------------------

         ####################### MODIFY ########################
         -- put encapsulated entity and fifo signals here
         ####################### MODIFY ########################

        );
END svi_slave_xxx;
```

II.i.ix. svi_slave_xxx_arch.vhd

This file contains the VHDL behavioral architeture description for the SVI slave.

```
--------------------------------------------------------------------
-- Program:             RASSP
-- Project:             Model Year Architecture
-- Title:               SVI Slave XXX Architecture
-- Description:         This file contains behavioral architecture
--                      for
--                      svi slave unit for xxx interface/pe.
-- Library:
-- Filename:            svi_slave_xxx_ent.vhd
-- Company:
-- Author Info:         name, email address , phone number,
--                      mail address
-- Revision History:    Date       Author       Purpose
--                      ----------------------------------------------
-- Known bugs:
------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE WORK.svi_types_pkg.all;
USE WORK.svi_xxx_types_pkg.all;

ARCHITECTURE mixed OF svi_slave_xxx is

   --  an enum type for the state
   TYPE state_type  IS (
                        idle,
                        ############## MODIFY ######################
                        -- put additional states here
                        ############## MODIFY ######################
                        );

   SIGNAL nxt_state : state_type := idle;
   SIGNAL curr_state : state_type := idle;



   ##################### MODIFY ##################################
   -- put necessary width converter Component definition here(or 1
   -- to 1)
   -- which converter is used will be determined by what data width
   -- of SVI is input to this encapsulation
   COMPONENT svi_X_to_X_converter
   ###################### MODIFY ##################################
      GENERIC (
               input_width : INTEGER := X
              );
      PORT (
         -- DATA INPUT INTERFACE ,Coming from svi interface
         ---------------------------------------------------------
   ########################### MODIFY ######################
   -- put expected incoming data width here
         svi_data_in    :           IN     std_logic_vector(
                                           xxx_width-1 DOWNTO 0);
   ########################### MODIFY ######################
         svi_last_word_in :         IN     std_logic;
         svi_channel_id_in :        IN     std_logic_vector(
                                           channel_id_width-1
                                           DOWNTO 0);
         svi_clock_in :             IN     std_logic;
         svi_xfer_request_in :      IN     std_logic;
         svi_ready_out :               OUT std_logic;
         svi_data_valid_in :        IN     std_logic;

         -- DATA OUTPUT INTERFACE , Going to svi_slave
         -----------------------------------------------
   ########################### MODIFY ######################
   -- put expected encapsulation data width here
         svi_data_out    :              OUT std_logic_vector(
                                            xxx_width-1 DOWNTO 0);
   ########################### MODIFY ######################
         svi_last_word_out :            OUT std_logic;
         svi_channel_id_out :           OUT std_logic_vector(
                                            channel_id_width-1
                                            DOWNTO 0);
         svi_xfer_request_out :         OUT std_logic;
         svi_ready_in :              IN     std_logic;
         svi_data_valid_out :           OUT std_logic;


         -- to svi slave, used to abort all processing
         abort :                     IN    std_logic;

         reset :                     IN    std_logic

     );
   END COMPONENT;

   -- SIGNALS used to connect to svi_converter
   --------------------------------------------
   ########################### MODIFY ######################
   -- put expected encapsulation data width here
   SIGNAL conv_svi_data_out : std_logic_vector(xxx_width-1
                               DOWNTO 0);
   ########################### MODIFY ######################

   SIGNAL conv_svi_last_word_out : std_logic;
   SIGNAL conv_svi_channel_id_out : std_logic_vector(
                                    channel_id_width-1
                                    DOWNTO 0);
   SIGNAL conv_svi_xfer_request_out : std_logic;
   SIGNAL conv_svi_ready_in : std_logic;
   SIGNAL conv_svi_data_valid_out : std_logic;
   SIGNAL conv_abort : std_logic;
   SIGNAL conv_reset : std_logic;

BEGIN -- architecture mixed

   -- Used to take X-byte incoming svi data
   -- to go to X-byte svi data
   ########################### MODIFY ######################
   –– put expected encapsulation data width here
   svi_slave_svi_X_to_X_converter : svi_X_to_X_converter
      GENERIC MAP (input_width => X)
   ########################### MODIFY ######################
      PORT MAP (
         svi_data_in,
         svi_last_word_in,
         svi_channel_id_in,
         svi_clock_in,
         svi_xfer_request_in,
         svi_ready_out,
         svi_data_valid_in,
         conv_svi_data_out,
         conv_svi_last_word_out,
         conv_svi_channel_id_out,
         conv_svi_xfer_request_out,
         conv_svi_ready_in,
         conv_svi_data_valid_out,
         conv_abort,
         conv_reset
         );

   –– ASYNCH. sig. assignments
   –– These signals will not change at all
   ––––––––––––––––––––––––––––––––––––––––––––
   ########################### MODIFY ######################
   –– put any asynchronous signal assignments here
   ########################### MODIFY ######################

–– This process handles reset(incoming) and
–– updating the svi_slave’s state on the rising
–– edge of the clock, also responsible for latching/
–– outputting of all synchronous signals
########## MODIFY ########################################
–– put clock name here for xxx_clock, reset for xxx_reset
########## MODIFY ########################################
clock : PROCESS(xxx_clock, xxx_reset)

BEGIN

   IF xxx_reset = ’1’ THEN
      curr_state <= idle;

   –– clock edge
   ELSIF xxx_clock’EVENT AND xxx_clock = ’1’ THEN

      ##################### MODIFY #######################
      –– based on curr_state and inputs,  drive synchrnous
      –– signals here
      ##################### MODIFY #######################
      –– update state
      curr_state <= nxt_state;

   END IF; –– if rising edge of svi_clock_in
END PROCESS clock;

–– This process is responsible for keeping the svi slave
–– in the correct state,  setting outputs correctly
–– based on state and inputs.
state_machine: PROCESS (curr_state, conv_svi_xfer_request_out,
                        svi_slave_abort_in, svi_slave_error_in,
                        conv_svi_last_word_out,
                        conv_svi_data_valid_out,xxx)
      ##################### MODIFY #######################
      –– Add encapsulated entity xxx control signals to
      –– sensitivity list
      ##################### MODIFY #######################

BEGIN

   –– Make sure all outputs get default assignments
   –– to avoid creating latches instead of combinational
   –– logic, will be overridden with different values determined
   –– by curr_state with same prop. delay
   –––––––––––––––––––––––––––––––––––––––––––––––––––––––
   svi_slave_abort_out <= no_abort_constant;
   svi_slave_error_out <= no_error_constant;
   conv_abort <= ’0’;
   conv_svi_ready_in <= ’0’;

   nxt_state <= curr_state; –– default
   ##################### MODIFY #######################
   –– Add ifc xxx output signals here
   ##################### MODIFY #######################

   CASE curr_state IS

      WHEN idle =>
      ––––––––––––––––––––––––––––––––––––––––––––––
      ##################### MODIFY #######################
      –– Add states and logic here
      ##################### MODIFY #######################

   END CASE;

END PROCESS state_machine;

interrupt_process : PROCESS(svi_interrupt_in)
BEGIN
   –– TBD TBD TBD TBD
   ASSERT (FALSE)
      REPORT ” Interrupt interface not implemented.”
      SEVERITY NOTE;
END PROCESS interrupt_process;

END mixed;
```