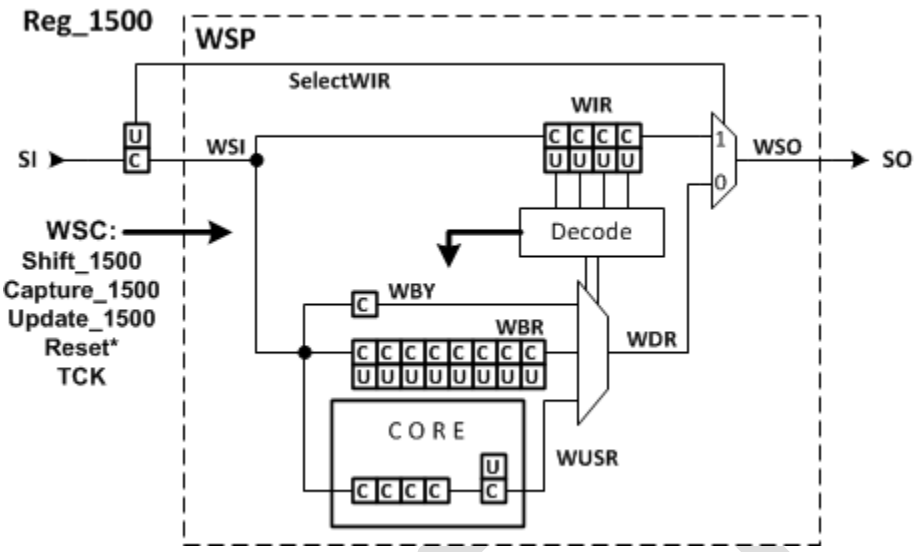
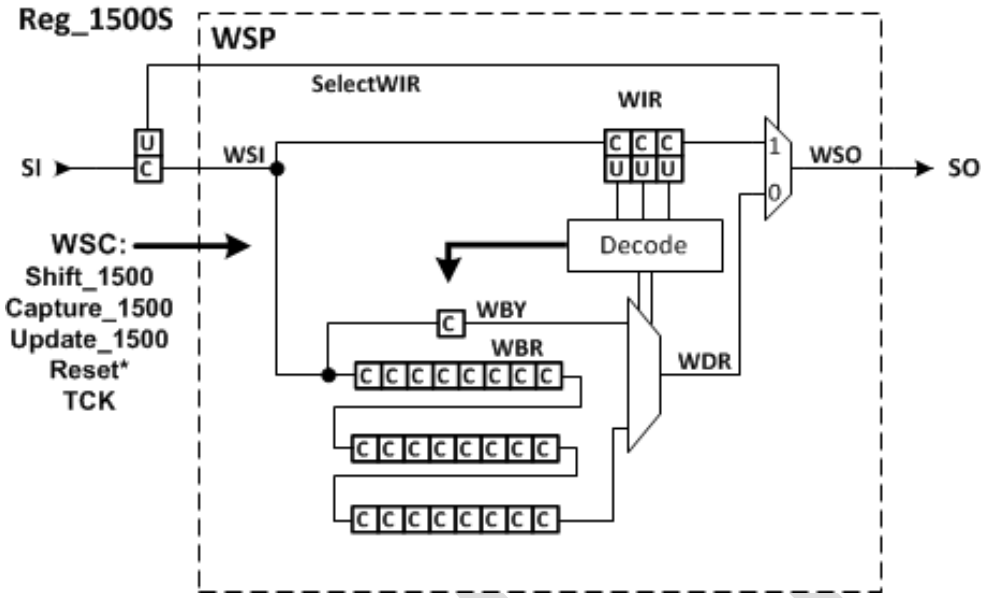


REG_1500.pac :

```
package REG_1500 is
   use STD_1149_1_2012.all;

Attribute REGISTER_MNEMONICS of REG_1500  : entity is
  "WIR_decode  ( "&
    "WS_BYPASS  (0B0000) <Wrapper Bypass Instruction>, "&
    "WS_EXTEST  (0B0001) <Wrapper Serial External Boundary Instruction>, "&
    "WS_INTEST  (0B0010) <Wrapper Serial Internal Boundary Instruction>, "&
    "WS_BIST    (0B0100) <BIST Instruction>, "&
    "WP_ALL     (0B1xxx) <Wrapper Parallel instructions> "&
    " )," &
  "BISTGROUP ( "&
    "Disable  (0B0) < BIST has not been enabled >, "&
    "Enable   (0B1) < BIST enabled > "&
    " ),"&
  "STATGROUP ( "&
    "PASS     (0B1001),  "&
    "FAIL     (0B0111)   "&
    " )," &
  "MODEGROUP ( "&
    "MODE0    (0X0),  "&
    "MODE3    (0X3)   "&
    " )";

Attribute REGISTER_ASSEMBLY of REG_1500  : entity IS
  "REG_1500 ( " &  -- The Select WIR bit and the Wrapper Serial Port
    -- Reset to WBY
    "(SELWIR [1] DelayPO ResetVal(0b0) TAPReset ),  "&
    "(WSP IS WSP_MUX) "&
  " ), "&
```

```
  "WSP_MUX ( "&   -- The outer selectable segments: WIR and WDR
    "(SelectMUX "&
      -- Reset to WBY
      "(WIR IS WIR_Seg), "&
      "(WDR IS WDR_MUX) "&
      "SelectField (SELWIR) "&
      "SelectValues ((WIR : 0b1) (WDR : 0b0)) "&
    " ) "&
  " ), "&
  "WIR_Seg ((WIR_field [4] DelayPO "&
          "ResetVal(WIR_decode(WS_BYPASS)) TAPReset ), "&
  "WDR_MUX ( "&  -- The inner selectable segments: WBY, WBR, and Wusr
    "(SelectMUX "&
      "(WBY  IS Reg_WBY CAPTURES(0) ), "&
      "(WBR  IS Reg_WBR), "&
      "(WUSR IS Reg_WUSER), "&
      "SelectField (WIR) "&
      "SelectValues ("&
        "(WBY  : WS_BYPASS, WP_ALL) "&
        "(WBR  : WS_EXTEST, WS_INTEST) "&
        "(WUSR : WS_BIST) "&
        " ) "&
    " ) "&
  " ), "&
  "REG_WBY   ( (WBY[1] NOPO)), " &
  "REG_WBR   ( (WBR[8] )), " &
  "REG_WUSER ( (CSR[4] CAPTURES(STATGROUP(-)) DEFAULT(MODEGROUP(MODE0)) NOUPD )," &
  "           ( GO [1] ResetVal(BISTGROUP(Disable)) ) " ;

end REG_1500;
package body REG_1500 is
   use STD_1149_1_2012.all;
end REG_1500;
```

<EOF>

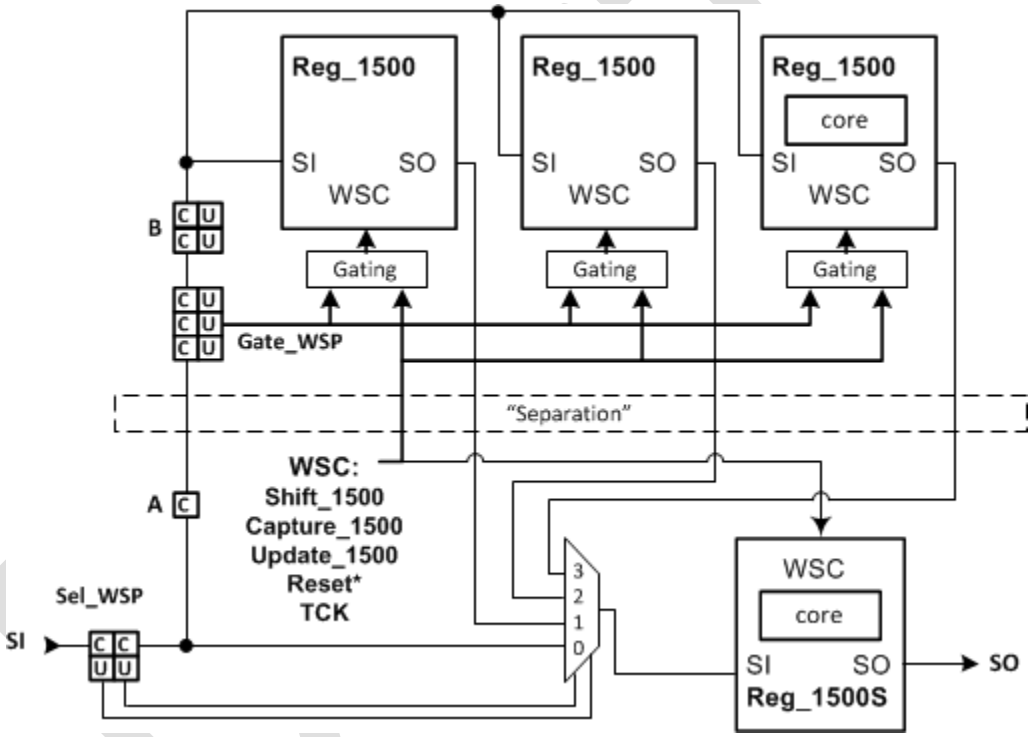


REG_1500S.pac :

```
package REG_1500S is
   use STD_1149_1_2012.all;

Attribute REGISTER_MNEMONICS of REG_1500S  : entity is
  "WIR_decode  ( "&
    "WS_BYPASS  (0B000) <Wrapper Bypass Instruction>, "&
    "WS_EXTEST  (0B001) <Wrapper Serial External Boundary Instruction>, "&
    "WS_INTEST  (0B010) <Wrapper Serial Internal Boundary Instruction>  "&
      " )";

Attribute REGISTER_ASSEMBLY of REG_1500S  : entity IS
  "REG_1500S ( " &  -- The Select WIR bit and the Wrapper Serial Port
    -- Reset to WBY
    "(SELWIR [1] DelayPO ResetVal(0b0) TAPReset ),  "&
    "(WSP IS WSP_MUX) "&
  " ), "&
  "WSP_MUX ( "&   -- The outer selectable segments: WIR and WDR
    "(SelectMUX "&
      -- Reset to WBY
      "(WIR IS WIR_Seg), "&
      "(WDR IS WDR_MUX) "&
      "SelectField (SELWIR) "&
      "SelectValues ((WIR : 0b1) (WDR : 0b0)) "&
    " ) "&
  " ), "&
  "WIR_Seg ((WIR_field [3] DelayPO "&
          "ResetVal(WIR_decode(WS_BYPASS)) TAPReset ), "&
  "WDR_MUX ( "&  -- The inner selectable segments: WBY, WBR, and Wusr
    "(SelectMUX "&
      "(WBY  IS Reg_WBY), "&
      "(WBR  IS Reg_WBR), "&
      "SelectField (WIR) "&
      "SelectValues ("&
        "(WBY  : WS_BYPASS ) "&
```

```
        "(WBR  : WS_EXTEST, WS_INTEST) "&
        " ) "&
    " ) "&
  " ), "&
  "REG_WBY    (( WBY[1]  NOPO )), " &
  "REG_WBR    (( WBR[24] NOPO )) ";

end REG_1500S;
package body REG_1500S is
   use STD_1149_1_2012.all;
end REG_1500S;
```

<EOF>



1500_ASSEMBLY.pac :

```
package REG_1500_ASSM is

   use STD_1149_1_2012.all;
   use REG_1500.all;
   use REG_1500S.all;

Attribute REGISTER_ASSEMBLY of REG_1500_ASSM  : entity IS


attribute REGISTER_MNEMONICS of REG_1500_ASSM  : entity is
      "WSP  ( "&
```

```
        "   None    (0B00) <Bypass all WSPs>, "&
        "   WSP1    (0B01) <Observe WSP(1)>, "&
        "   WSP2    (0B10) <Observe WSP(2)>, "&
        "   WSP3    (0B11) <Observe WSP(3)>  "&
        " )," &
      "BRDCST ( "&
        "   None      (0B000) <All WSP held>, "&
        "   WSP1      (0B001) <Scan WSP(1) only>, "&
        "   WSP2      (0B010) <Scan WSP(2) only>, "&
        "   WSP3      (0B011) <Scan WSP(3) only>, "&
        "   1AND2     (0B110) <Scan just WSP(1) and WSP(2)>, "&
        "   ALLWSP    (0B111) <Scan all WSPs >  "&
        "  )";

Attribute REGISTER_ASSEMBLY of REG_1500_ASSM : entity IS
  "Reg_1500_MUX ( " &
    "(Sel_WSP[2]  ResetVal(WSP(None))    TAPReset  ) ,"&
    "(SELECTMUX " &
      "(WIRE1 is WIRE)," &
      "(ARRAY WSP(1 TO 3) IS WSP_inst) " &
      "SELECTFIELD (Sel_WSP) "&   -- 4:1 selection
      "SELECTVALUES ( "& --   Decode logic for connecting a WSP to Scan-Out
        "(WIRE1:None) (WSP(1):WSP1) (WSP(2):WSP2) (WSP(3):WSP3) )"&
      "BROADCASTFIELD (Gate_WSP) "&  -- Could use WSP_common.Gate_WSP
      "BROADCASTVALUES ( "&    --  Decode  logic for gating WSC
        "(WSP(1),WSP(2),WSP(3) : ALLWSP) "&
        "(WSP(1),WSP(2)        : 1AND2 ) "&
        "(WSP(1)               : WSP1) "&
        "(WSP(2)               : WSP2) "&
        "(WSP(3)               : WSP3) "&
                    ")"&
    ")"&
    "( WSP_1500S is Reg_1500S)," &    -- Reg_1500S comes after MUX
              "), " &                 -- end REG_1500_MUX
  "WIRE ( ( WIRE[0] ) ), "&
  "WSP_inst ( "&
      "(WSP_common), "&
      "(WSP_1500 IS Reg_1500) " &
  "), "&
  "common_seg ( (WSP_common IS common) ), "&
  "common ("&
    "(A [1] NOUPD), "&
    "(Gate_WSP[3] ResetVal(BRDCST(None)) TAPReset ), "&
    "(B [2] ) "&
  ") " ;



attribute REGISTER_CONSTRAINTS of REG_1500_ASSM : entity is
  "REG_1500_MUX (" &

       "( Gate_WSP ==  BRDCST{1AND2}   &&   Sel_WSP ==  WSP{WSP3} ) "&
       "ERROR < Sel_WSP of WSP3 not possible with Gate_WSP of 1AND2>, "&

       "(( (Gate_WSP == BRDCST{WSP2} ) || (Gate_WSP == BRDCST{WSP3}) ) "&
       "     && (Sel_WSP == WSP{WSP1}) ) "&
```

```
       "ERROR < Sel_WSP of WSP1 not possible with Gate_WSP of WSP2 or 3 >, "&

       "(( (Gate_WSP == BRDCST{WSP1} ) || (Gate_WSP == BRDCST{WSP3}) ) "&
       "     && (Sel_WSP == WSP{WSP2}) ) "&
       "ERROR < Sel_WSP of WSP2 not possible with Gate_WSP of WSP1 or 3 >, "&

       "(( (Gate_WSP == BRDCST{WSP1} ) || (Gate_WSP == BRDCST{WSP2}) ) "&
       "     && (Sel_WSP == WSP{WSP3}) ) "&
     "ERROR < Sel_WSP of WSP3 not possible with Gate_WSP of WSP1 or 2 > "&
   ")";

end REG_1500_ASSM ;


package body REG_1500_ASSM is
   use STD_1149_1_2012.all;
end REG_1500_ASSM;

<EOF>
```

Reg_1500.pdl:

```
# Supplied by MyCorp for REG_1500 version 1.0


iPDLLevel 0 -version STD_1149_1_2012
iProcGroup REG_1500

#  check that bypass register can be scanned
iProc check_bypass { } {
  iWrite WIR WS_BYPASS;  # Use WS_BYPASS and not WP_ALL
  iRead WBY 0
  iApply
  }

#
iProc start_bist { mode } {
  # CSR is documented to be a c/s register only.  GO has c/s and update.
  # Setting up mode and executing BIST can be done in 1 scan operation
  iWrite CSR $mode
  iWrite GO  Enable
  iApply
  iRunLoop  100000
  }

# shame there is not a PDL command or predefined variable $Curr_inst to use
iProc check_bist { instance mode } {
  iRead  CSR PASS
  iApply -nofail
  ifFalse
     iSetFail "$instance REG_1500 BIST test with mode = $mode failed\n"
  ifEnd
  }
```

```
<EOF>
```

Reg_1500S.pdl:

```
# Supplied by MyCorp for 1500S version 1.0

iPDLLevel 0 -version STD_1149_1_2012
iProcGroup REG_1500S ;

#  check that bypass register can be scanned
iProc check_bypass { } {
  iRead WBY 0
  }

#

<EOF>
```

Reg_1500_Assm.pdl :

```
# Supplied by MyCorp for 1500_ASSM version 1.0
iSource REG_1500.pdl
iSource REG_1500S.pdl
iPDLLevel 0 -version STD_1149_1_2012

iProcGroup REG_1500_ASSM ;


#  check that bypass registers can be scanned
iProc check_bypass { } {
iCall WSP_1500S.check_bypass    ;# make sure WSP_1500S is in bypass mode
                                # scan occurs in next line and checked three times
                                # during bypass check of WSP_1500
iCall WSP(1).WSP_1500.check_bypass
iCall WSP(2).WSP_1500.check_bypass
iCall WSP(3).WSP_1500.check_bypass

}

# start and check BIST for each WSP_1500

iProc bist_test { } {

# enable broadcast to save wait time.  Two modes of broadcast exist
# ALLWSP and 1AND2.  Without specifying which broadcast mode, it is ambiguous

iWrite WSP(1).WSP_common.Gate_WSP ALLWSP  ;# tool selects path to set to broadcast
# Gate_WSP is unique within REG_1500_ASSM package file hence
# iWrite Gate_WSP ALLWSP is unambiguous

iApply                                    ;# need mux set and gate decode prior to test
iCall WSP(1).WSP_1500.start_bist MODE0    ;# writing to just 1 WSP, however in broadcast
                                           # mode all WSPs are getting BIST setup
```

```
iWrite WSP(1).WSP_common.Gate_WSP WSP1    ;# set Gate_WSP back to singular mode
iApply                                     ;# need mux set and gate decode prior

# iWrite Sel_WSP  WSP1                     ;# tool would not need to have Sel_WSP set
# need to pass in instance name shows lack of PDL command to retrieve current instance
iCall WSP(1).WSP_1500.check_bist WSP(1) MODE0
iCall WSP(2).WSP_1500.check_bist WSP(2) MODE0
iCall WSP(3).WSP_1500.check_bist WSP(3) MODE0


iWrite WSP(1).WSP_common.Gate_WSP ALLWSP  ;# tool selects path to set to broadcast
iApply                                     ;# need mux set and gate decode prior to test
iCall  WSP(1).WSP_1500.start_bist MODE1    ;# writing to just 1 WSP, however in broadcast
                                            # mode all WSPs are getting BIST setup

iWrite WSP(1).WSP_common.Gate_WSP WSP1    ;# set Gate_WSP back to singular mode
iApply                                     ;# need mux set and gate decode

# iWrite Sel_WSP  WSP1                     ;# tool would not need to have Sel_WSP set
iCall WSP(1).WSP_1500.check_bist WSP(1) MODE1
iCall WSP(2).WSP_1500.check_bist WSP(2) MODE1
iCall WSP(3).WSP_1500.check_bist WSP(3) MODE1

}


iPDLLevel 1 -version STD_1149_1_2012


iProc interconnect { } {

# Connections exist 1:1 between WSP(3:1) and WSP_1500S

iWrite Gate_WSP ALLWSP                     ;# using short form for illustration
iApply
iWrite WSP(1).WSP_1500.WIR  WS_EXTEST      ;# The WBR access is ambiguous, there are
# two paths for accessing the WBR, WS_EXTEST and WS_INTEST
# all three WSPs get WS_EXTEST in the WIR
iWrite WSP_1500S.WIR  WS_EXTEST
iApply                                     ;# 4 WSPs in WS_EXTEST mode

iWrite WSP(1).WSP_1500.WBR    0
iApply
iRead  WSP_1500S.WBR  0
iWrite WSP(1).WSP_1500.WBR(0) 0b1
iApply

set i 1
while {$i < 8} {
iRead  WSP_1500S.WBR  0
set pos [expr {$i - 1}]
iRead  WSP_1500S.WBR($pos) 1
set pos [expr {$pos +8}]
iRead  WSP_1500S.WBR($pos) 1
set pos [expr {$pos +8}]
iRead  WSP_1500S.WBR($pos) 1
```

```
iWrite WSP(1).WSP_1500.WBR($i) 0b1
iApply
set i [expr {$i + 1}]
}

set pos [expr {$i - 1}]
# read last driven values
iRead  WSP_1500S.WBR($pos) 1
set pos [expr {$pos +8}]
iRead  WSP_1500S.WBR($pos) 1
set pos [expr {$pos +8}]
iRead  WSP_1500S.WBR($pos) 1
iApply

}



<EOF>
```