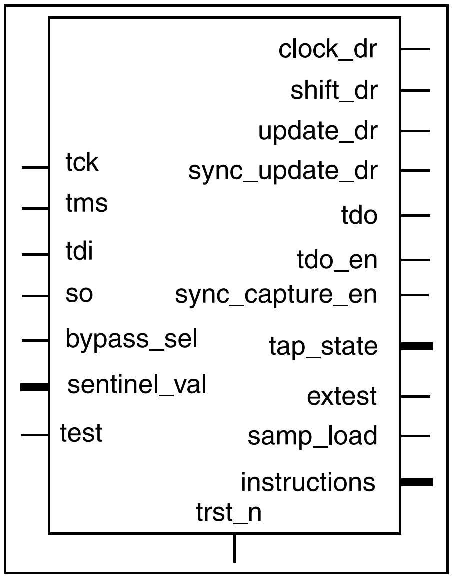
DW_tap

TAP Controller

Version, STAR and Download Information: IP Directory

Features and Benefits

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous registers with respect to `tck`
- Supports the standard instructions EXTEST, SAMPLE/PRELOAD, and BYPASS
- Supports the optional instructions IDCODE, INTEST, RUNBIST, CLAMP, and HIGHZ
- Optional use of device identification register and IDCODE instruction
- Parameterized instruction register width



Description

DW_tap provides access to on-chip boundary scan logic.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
tck	1 bit	Input	Test clock
trst_n	1 bit	Input	Test reset, active low
tms	1 bit	Input	Test mode select
tdi	1 bit	Input	Test data in
so	1 bit	Input	Serial data from boundary scan register and data registers
bypass_sel	1 bit	Input	Selects the bypass register, active high
sentinel_val	*width* – 1 bit(s)	Input	User-defined status bits
clock_dr	1 bit	Output	Clocks in data in asynchronous mode
shift_dr	1 bit	Output	Enables shifting of data in both synchronous and asynchronous mode, active high
update_dr	1 bit	Output	Enables updating data in asynchronous mode, active high
tdo	1 bit	Output	Test data out

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
tdo_en	1 bit	Output	Enable for tdo output buffer, active high
tap_state	16 bits	Output	Current state of the TAP finite state machine
extest	1 bit	Output	EXTEST decoded instruction
samp_load	1 bit	Output	SAMPLE/PRELOAD decoded instruction
instructions	*width* bit(s)	Output	Instruction register output
sync_capture_en	1 bit	Output	Enable for synchronous capture, active low
sync_update_dr	1 bit	Output	Enables updating new data in *sync_mode*, active high
test	1 bit	Input	For scannable designs, the `test` pin is held active (HIGH) during testing. For normal operation, it is held inactive (LOW).

Table 1-2 Parameter Description

Parameter	Values	Description
width	2 to 32 Default: None	Width of instruction register
id	0 or 1 Default: 0	Determines whether the device identification register is present 0: Not present 1: Present
version	0 to 15 Default: 0	4-bit version number
part	0 to 65535 Default: 0	16-bit part number
man_num	0 to 2047, man_num ≠ 127 Default: 0	11-bit JEDEC manufacturer identity code
sync_mode	0 or 1 Default: 0	Determines whether the bypass, device identification, and instruction registers are synchronous with respect to tck 0: Asynchronous, 1: Synchronous
tst_mode	0 or 1 Default: 1	Controls whether the `test` input is used 1: The `test` input is used; (backward compatible with older versions) 0: The `test` input is not used, which ensures that unused clock gating logic will not be included

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
str	Synthesis model	DesignWare or Test-IEEE-STD-1149-1

Table 1-4 Simulation Models

Model	Function
DW04.DW_TAP_CFG_SIM	Design unit name for VHDL simulation
dw/dw04/src/DW_tap_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_tap.v	Verilog simulation model source code

Table 1-5 Decoded Instructions

Instruction	Decoded Value
BYPASS	all ones
EXTEST	all zeros
IDCODE	000...01
SAMPLE/PRELOAD	000...10

Control of DW_tap is through the pins `tck`, `tms`, `tdi`, `tdo`, and `trst_n`. `tck`, `tms`, and `trst_n` control the states of the boundary scan test logic. `tdi` and `tdo` provide serial access to the instruction and data registers.

DW_tap contains the IEEE standard 1149.1 TAP finite state machine, instruction register, bypass register, and the optional device identification register.

The `tck` signal is the clock for the TAP finite state machine. The data on the `tms` and `tdi` signals is loaded on the rising edge of `tck`. Data is available at `tdo` on the falling edge of `tck`. If the parameter *sync_mode* is set to 1, then `tck` is used to clock data into the instruction register, bypass register, and identification register. If *sync_mode* is set to 0, then the signals generated by the TAP finite state machine control the clocking of the registers.

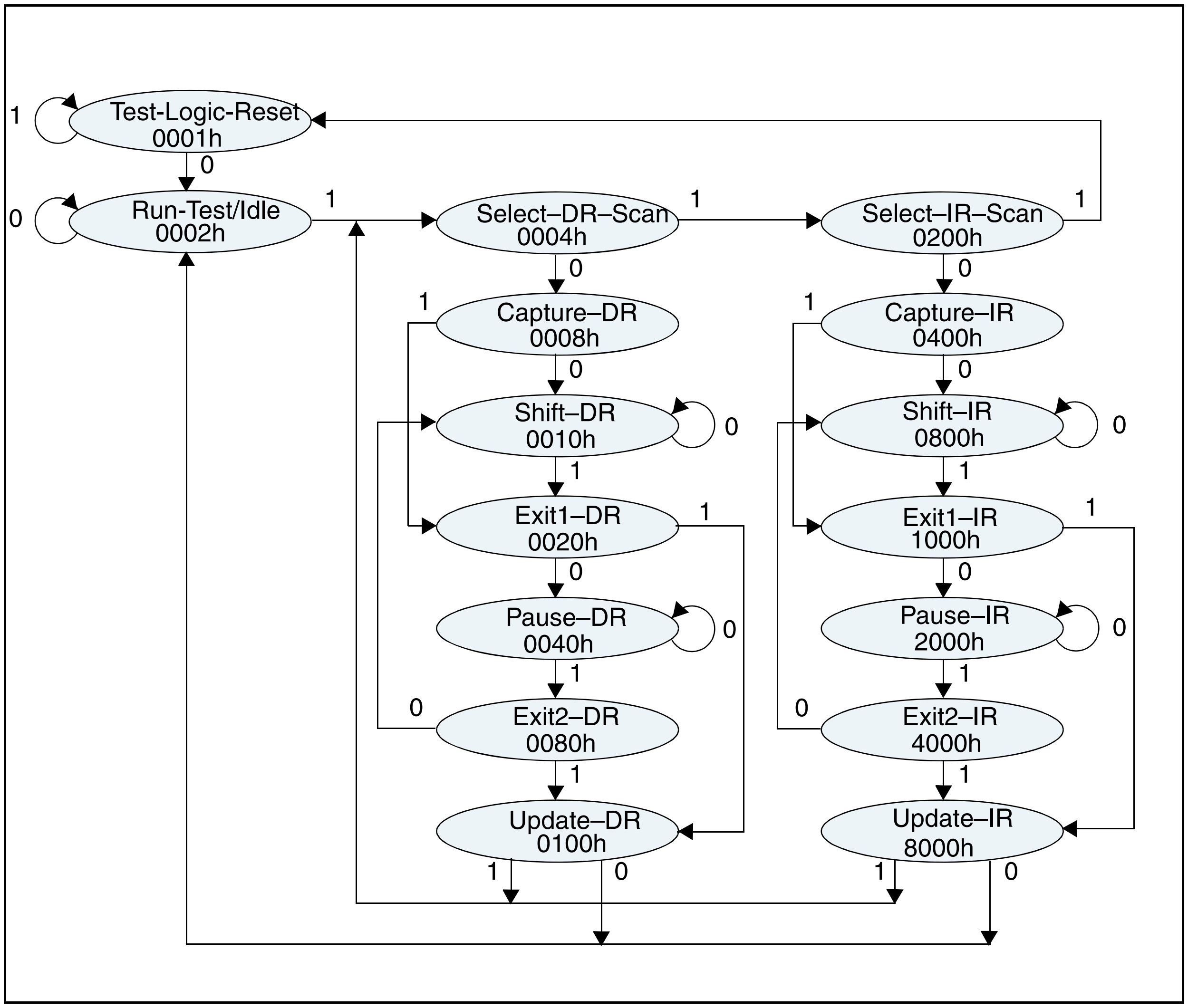
The `tms` signal controls the transitions of the TAP state machine, as illustrated in the state diagram (Figure 1-1 on page 4). The state transitions occur at the rising edge of `tck`. The `tdi` signal is the serial test data input, and the `tdo` signal is the serial test data output.

The `trst_n` signal is an active low signal that asynchronously resets the TAP state machine to the Test-Logic-Reset state.

The DW_tap output signals `clock_dr`, `shift_dr`, `update_dr`, `sync_update_dr`, and `sync_capture_en` control the boundary scan cells. In synchronous mode, boundary scan cells are updated on the rising edge of `tck`. In asynchronous mode, the boundary scan cells are updated by the rising edge of the `clock_dr` signal, asynchronous to `tck`.

The `tap_state` port provides access to the one-hot encoded TAP state machine, enabling advanced users to construct add-on circuits.

Figure 1-1 State Diagram



DW_tap Registers

The bypass register is a mandatory IEEE 1149.1 standard 1-bit register that provides a minimum length path through the IC. The instruction value of all ones selects the bypass register as the serial connection between `tdi` and `tdo`. Other decoded instructions can select the bypass register through the `bypass_sel` input signal.

The device identification register is an optional IEEE 1149.1 standard register. If the parameter `id` is set to 1, then synthesis builds the 32-bit device identification register. The parameters `part`, `version`, and `man_num` determine the 32-bit number that is loaded into the device identification register when the instruction:

```
000...01 (IDCODE instruction)
```

is selected. This allows the manufacturer, part number, and variant for the component to be read in a serial binary form.

The instruction register is used to serially shift in the instructions that operate the boundary scan circuitry. The instruction register must be at least two bits wide to hold the instruction codes for the three IEEE 1149.1 standard required boundary scan instructions: BYPASS, EXTEST, SAMPLE/PRELOAD. The parameter *width* determines the size of the instruction register.

The `sentinel_val` bus provides extra status bits in the IC design. You can connect the `sentinel_val` bus to any signal you want to view when accessing the instruction register. If you do not want to use this feature, you must tie the `sentinel_val` bus to logic zero. The most significant bit of `sentinel_val` is always tied to zero.

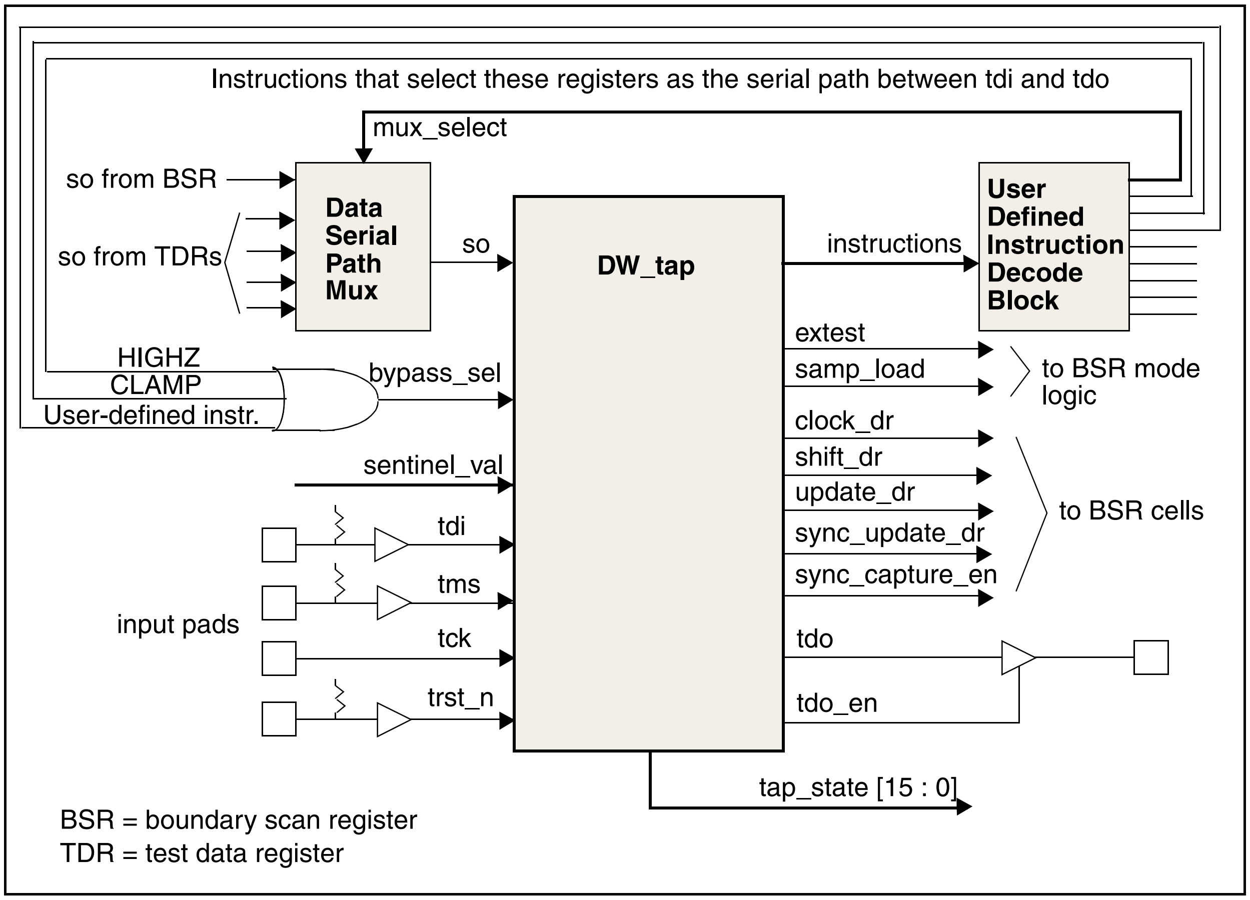
DW_tap decodes the mandatory instructions BYPASS, EXTEST, SAMPLE/PRELOAD, and the optional instruction IDCODE, as listed in Table 1-5 on page 3. Decoding of other optional and user-defined instructions must be implemented in user-defined instruction decode logic.

DW_tap supports user-defined test data registers. The test data registers are connected serially between `tdi` and `tdo` by a user-defined multiplexer. The multiplexer must select either the boundary scan register serial path or any of the user-defined test data registers during the appropriate instruction. The output of the multiplexer is connected to the *so* input of DW_tap.

Figure 1-2 shows the user-defined interface logic needed to implement DW_tap in an integrated circuit.

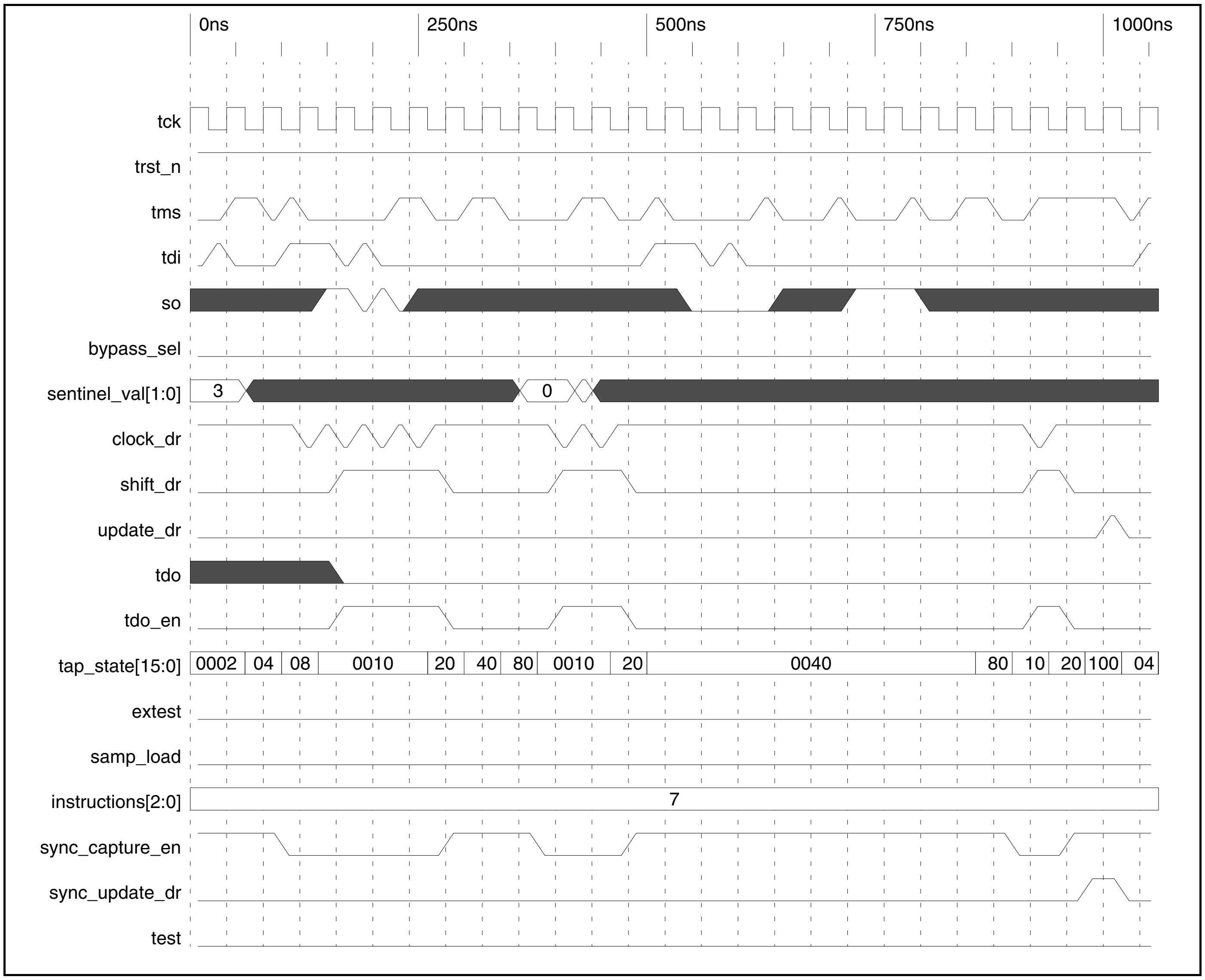
For more information about how to implement DesignWare Building Blocks boundary scan components in your design, refer to Application Note "AN 96-004".

Figure 1-2 Interface Diagram



Timing Diagram

Figure 1-3 Timing Diagram



Related Topics

- Application Specific – JTAG Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW_tap_inst is
      generic (
        inst_width : INTEGER := 8;
        inst_id : INTEGER := 0;
        inst_version : INTEGER := 0;
        inst_part : INTEGER := 0;
        inst_man_num : INTEGER := 0;
        inst_sync_mode : INTEGER := 0;
        inst_tst_mode : INTEGER := 1
        );
      port (
        inst_tck : in std_logic;
        inst_trst_n : in std_logic;
        inst_tms : in std_logic;
        inst_tdi : in std_logic;
        inst_so : in std_logic;
        inst_bypass_sel : in std_logic;
        inst_sentinel_val : in std_logic_vector(inst_width-2 downto 0);
        clock_dr_inst : out std_logic;
        shift_dr_inst : out std_logic;
        update_dr_inst : out std_logic;
        tdo_inst : out std_logic;
        tdo_en_inst : out std_logic;
        tap_state_inst : out std_logic_vector(15 downto 0);
        extest_inst : out std_logic;
        samp_load_inst : out std_logic;
        instructions_inst : out std_logic_vector(inst_width-1 downto 0);
        sync_capture_en_inst : out std_logic;
        sync_update_dr_inst : out std_logic;
        inst_test : in std_logic
        );
    end DW_tap_inst;


architecture inst of DW_tap_inst is

begin

    -- Instance of DW_tap
    U1 : DW_tap
    generic map ( width => inst_width,
                  id => inst_id,
                      version => inst_version,
                      part => inst_part,
                      man_num => inst_man_num,
                      sync_mode => inst_sync_mode,
                      tst_mode => inst_tst_mode )
    port map (  tck => inst_tck,
                trst_n => inst_trst_n,
                tms => inst_tms,
                tdi => inst_tdi,
                so => inst_so,
                bypass_sel => inst_bypass_sel,
                sentinel_val => inst_sentinel_val,
                clock_dr => clock_dr_inst,
                shift_dr => shift_dr_inst,
                update_dr => update_dr_inst,
                tdo => tdo_inst,
                tdo_en => tdo_en_inst,
                tap_state => tap_state_inst,
                extest => extest_inst,
                samp_load => samp_load_inst,
                instructions => instructions_inst,
                sync_capture_en => sync_capture_en_inst,
                sync_update_dr => sync_update_dr_inst,
                test => inst_test );


end inst;

-- pragma translate_off
configuration DW_tap_inst_cfg_inst of DW_tap_inst is
  for inst
  end for; -- inst
end DW_tap_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_tap_inst( inst_tck,
                    inst_trst_n,
                    inst_tms,
                    inst_tdi,
                    inst_so,

              inst_bypass_sel,
                    inst_sentinel_val,
                    clock_dr_inst,
                    shift_dr_inst,
                    update_dr_inst,

              tdo_inst,
                    tdo_en_inst,
                    tap_state_inst,
                    extest_inst,
                    samp_load_inst,

              instructions_inst,
                    sync_capture_en_inst,
                    sync_update_dr_inst,
                    inst_test );

parameter width = 8;
parameter id = 0;
parameter version = 0;
parameter part = 0;
parameter man_num = 0;
parameter sync_mode = 0;
parameter tst_mode = 1;


input inst_tck;
input inst_trst_n;
input inst_tms;
input inst_tdi;
input inst_so;
input inst_bypass_sel;
input [width-2 : 0] inst_sentinel_val;
output clock_dr_inst;
output shift_dr_inst;
output update_dr_inst;
output tdo_inst;
output tdo_en_inst;
output [15 : 0] tap_state_inst;
output extest_inst;
output samp_load_inst;
output [width-1 : 0] instructions_inst;
output sync_capture_en_inst;
output sync_update_dr_inst;
input inst_test;

    // Instance of DW_tap
    DW_tap #(width,
             id,
             version,
             part,
             man_num,
             sync_mode,
         tst_mode)
      U1 (   .tck(inst_tck),
                 .trst_n(inst_trst_n),
                 .tms(inst_tms),
                 .tdi(inst_tdi),
                 .so(inst_so),
                 .bypass_sel(inst_bypass_sel),
                 .sentinel_val(inst_sentinel_val),
                 .clock_dr(clock_dr_inst),
                 .shift_dr(shift_dr_inst),
                 .update_dr(update_dr_inst),
                 .tdo(tdo_inst),
                 .tdo_en(tdo_en_inst),
                 .tap_state(tap_state_inst),
                 .extest(extest_inst),
                 .samp_load(samp_load_inst),
                 .instructions(instructions_inst),
                 .sync_capture_en(sync_capture_en_inst),
                 .sync_update_dr(sync_update_dr_inst),
                 .test(inst_test) );

endmodule
```