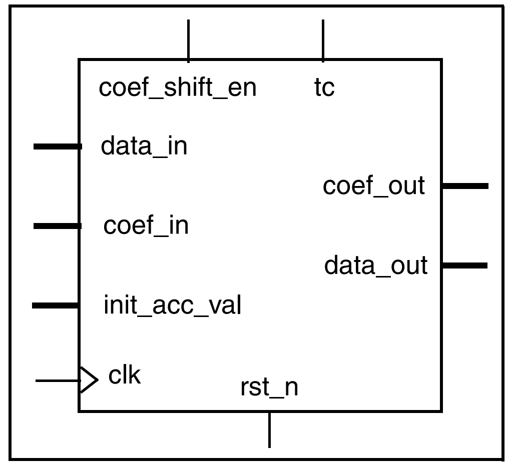
DW_fir

High-Speed Digital FIR Filter

Version, STAR and Download Information: IP Directory

Features

- High-speed transposed canonical FIR filter architecture
- Parameterized coefficient, data, and accumulator word lengths
- Parameterized filter order
- Serially loadable coefficients
- Cascadable architecture for easy partitioning
- DesignWare datapath generator is employed for better timing and area



Applications

- 1-D FIR filtering
- Matched filtering
- Correlation
- Pulse shaping
- Adaptive filtering
- Equalization

Description

DW_fir is a high-speed digital FIR filter designed for Digital Signal Processing applications employing very high sampling rates.

The number of coefficients in the filter as well as the coefficient, data, and accumulator word lengths are parameterized.

The device has a cascadable design enabling easy partitioning of a large order filter over several ASIC devices.

A serial scan chain is used for loading all of the coefficients.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock. All internal registers are sensitive on the positive edge of `clk` and all setup/hold times are with respect to this edge of `clk`.
rst_n	1 bit	Input	Asynchronous reset, active low. Clears all coefficient and data values.
coef_shift_en	1 bit	Input	Enable coefficient shift loading at `coef_in`, active high.
tc	1 bit	Input	Defines `data_in` and `coef_in` values as two's complement or unsigned. If low, the `data_in` and `coef_in` values are unsigned; if high, they are two's complement.
data_in	*data_in_width* bit(s)	Input	Input data.
coef_in	*coef_width* bit(s)	Input	Serial coefficient `coef_shift_en` port. This port is enabled when the `coef_shift_en` pin is set high. A rising edge of `clk` loads the coefficient data at `coef_in` into the first internal coefficient register and shifts all other coefficients in the internal registers one location to the right.
init_acc_val	*data_out_width* bit(s)	Input	Initial accumulated sum value. If unused, this pin is tied to low ("000...000"), that is, when the FIR filter is implemented with a single DW_fir component. When several DW_fir components are cascaded, the `data_out` of the previous stage is connected to the `init_acc_val` port of the next.
data_out	*data_out_width* bit(s)	Output	Accumulated sum of products of the FIR filter.
coef_out	*coef_width* bit(s)	Output	Serial coefficient output port. When the `coef_shift_en` pin is high and coefficients are being loaded serially, the coefficient data in the last internal coefficient register is output through the `coef_out` port.

Table 1-2 Parameter Description

Parameter	Values	Description
data_in_width	≥ 1	Input data word length
coef_width	≥ 1	Coefficient word length
data_out_width[a]	≥ 1	Accumulator word length
order	2 to 256	FIR filter order

a. The parameter *data_out_width* is normally set to a value of *coef_width* + *data_in_width* + *margin*. The value *coef_width* + *data_in_width* accounts for the internal coefficient multiplications. An appropriate margin must be included if the filter coefficients have a gain or are cascaded. The value *margin* ≤ log2(*order*).

Table 1-3 - Synthesis Implementations

Implementation Name	Function	License Required
str	Structural synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW03.DW_FIR_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_fir_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fir.v	Verilog simulation model source code

Table 1-5 Modes of Operation

rst_n	coef_shift_en	Mode	Operation
1	1	Coefficient load	Serially load coefficients into the filter starting with coef(0). See Table 1-6 on page 4.
1	0	Filter	$\text{data\_out}(n) = \text{init\_acc\_val}(n-1) + \sum_{i=0}^{order-1} \text{data\_in}(n-i-1)\ \text{coef}(i)$
0	X	Reset	Asynchronously clear all internal registers to zero state

Functional Description

A block diagram of the DW_fir filter is given in Figure 1-1. The DW_fir is clocked with the `clk` pin and is sensitive to the rising edge of `clk`. An asynchronous active-low reset pin, `rst_n`, clears all internal registers to the zero state.

The filter is programmed by serially loading coefficients into the device through the `coef_in` port when the `coef_shift_en` pin is high. The loading sequence is clocked off the rising edge of `clk`. In the loading mode, coefficients are loaded serially starting at `coef(0)` progressing up to `coef(order-1)` for a single filter. When multiple filters are cascaded together, each filter's `coef_in` port is connected to the preceding filter's `coef_out` port to create a single, serial chain for loading the coefficients. See the Application section at the end of this datasheet for an example of cascaded filters. Table 1-6 shows the coefficient register loading sequence for an 8-tap filter.

The `tc` pin identifies the type of data entering the `data_in` port and the type of coefficients. The data and coefficient types must be the same. When `tc` is high, the data and coefficient type is two's complement. When `tc` is low, the type is unsigned.

When multiple filters are cascaded together, each filter's `init_acc_val` port is connected to the preceding filter's `data_out` port. The critical path is always from `data_in` through the multiplier and adder to the accumulator register. The critical path timing is not affected by filter order because of the filter's transposed canonical form.

Figure 1-1 Block Diagram

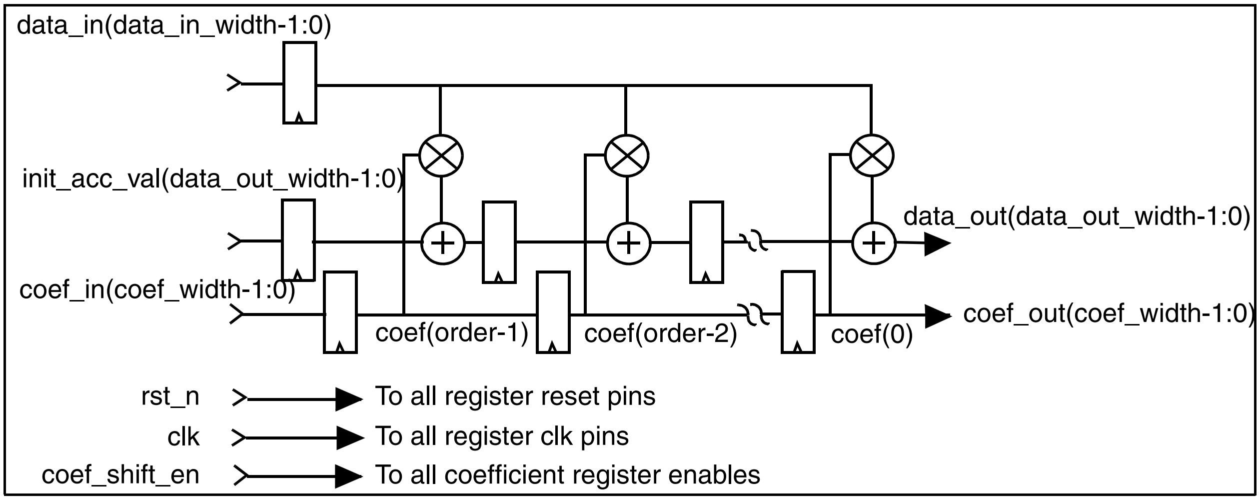
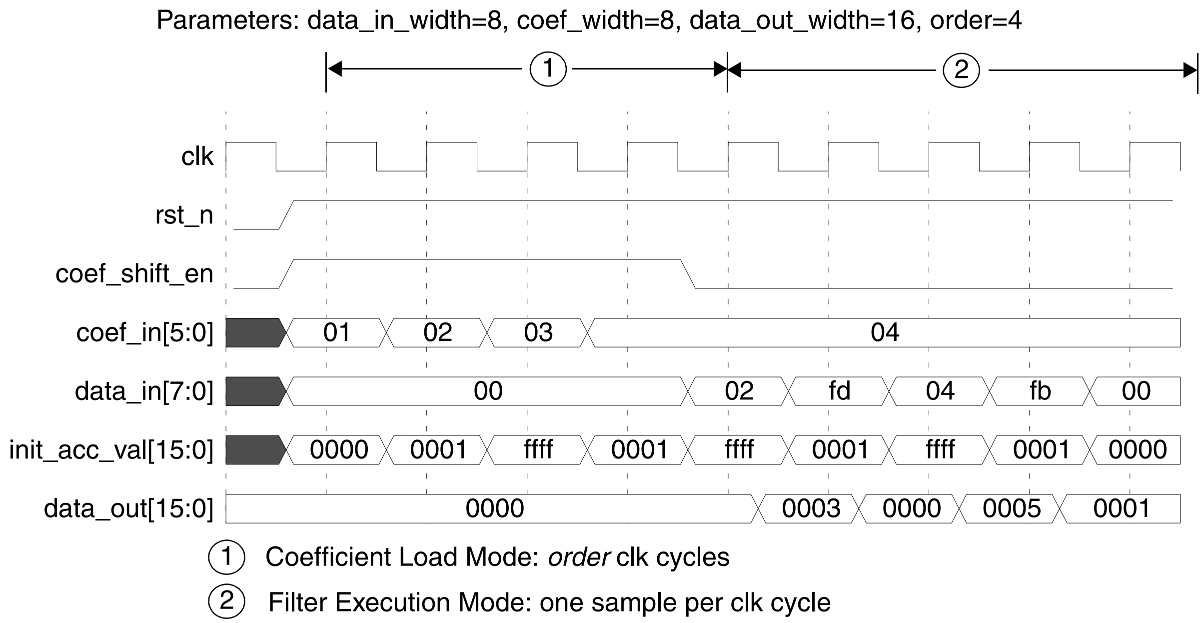


Table 1-6 Coefficient Register Loading Sequence for 8-Tap FIR Filter (order = 8)

Clock Cycle	Mode of Operation	Internal Coefficient Register State							
		7	6	5	4	3	2	1	0
0	Reset	0	0	0	0	0	0	0	0
1	Coefficient Load	coef(0)	0	0	0	0	0	0	0
2	Coefficient Load	coef(1)	coef(0)	0	0	0	0	0	0
3	Coefficient Load	coef(2)	coef(1)	coef(0)	0	0	0	0	0
4	Coefficient Load	coef(3)	coef(2)	coef(1)	coef(0)	0	0	0	0
5	Coefficient Load	coef(4)	coef(3)	coef(2)	coef(1)	coef(0)	0	0	0
6	Coefficient Load	coef(5)	coef(4)	coef(3)	coef(2)	coef(1)	coef(0)	0	0
7	Coefficient Load	coef(6)	coef(5)	coef(4)	coef(3)	coef(2)	coef(1)	coef(0)	0
8	Coefficient Load	coef(7)	coef(6)	coef(5)	coef(4)	coef(3)	coef(2)	coef(1)	coef(0)
9	Filter	coef(7)	coef(6)	coef(5)	coef(4)	coef(3)	coef(2)	coef(1)	coef(0)

Timing Waveforms

Figure 1-2 DW_fir Timing Diagram



Theory of Operation

An FIR filter implements the difference equation:

$$output(n) = \sum_{i=0}^{order-1} data\_in(n-i-1)coef(i)$$

A hardware architecture called the canonical direct-form can be directly derived from this equation by implementing multiplication operators with hardware multipliers and the summation operator with a series of adders; see Figure 1-3. The disadvantage of this architecture is poor performance due to the `order-1` additions in its critical timing path. This can be improved by structuring the additions into a tree, but there is a better approach.

From linear systems theory, we can apply the transpose operation to the canonical direct form architecture and arrive at a more speed-efficient structure. DW_fir has this structure. The taps are topologically in the reverse order of the canonical form; see Figure 1-4.

The primary advantage of the transpose architecture is that the addition operators are automatically pipelined without introducing extra latency. The pipelined addition allows very large order filters to be designed in which the clock rate is independent of filter order.

Figure 1-3 Canonical Direct-Form FIR Filter Architecture

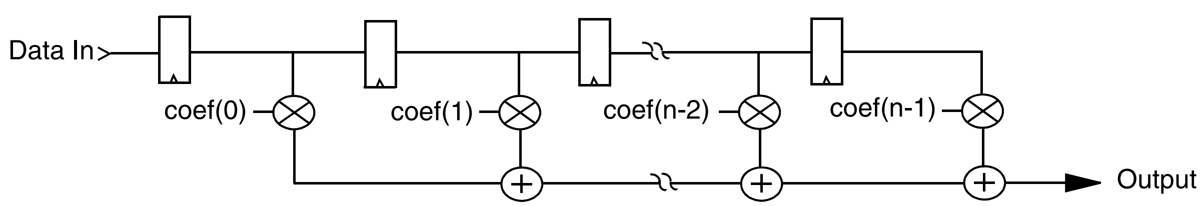
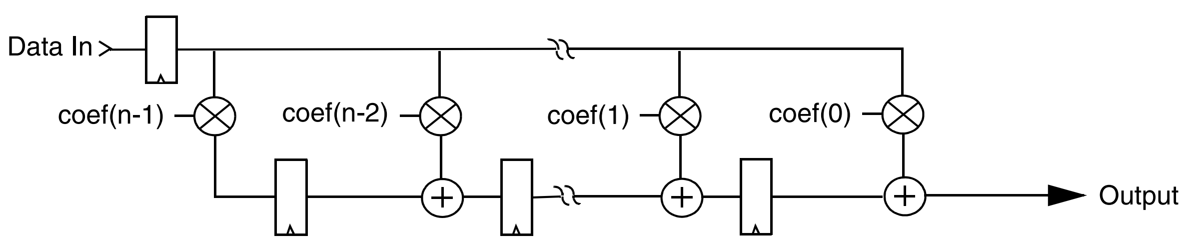


Figure 1-4 DW_fir Equivalent Transposed Canonical Architecture

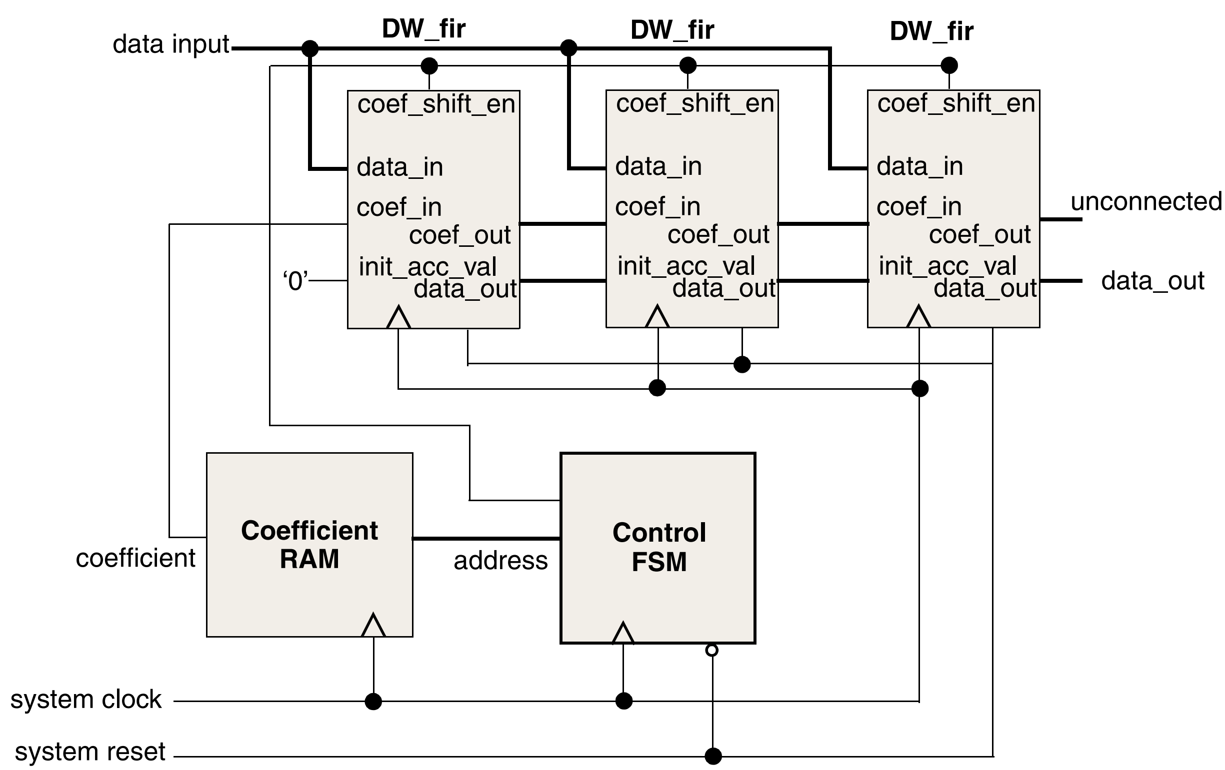


Application Example

The DW_fir high-speed FIR filter can be implemented as a small filter on a single ASIC or as a large filter spread across several ASICs. Figure 1-5 shows a block diagram of a possible design using 3 filters cascaded together to implement a large filter spanning several ASICs. The order of the filter, and data, coefficient, and accumulator word widths are specified through parameters during elaboration. The actual filter type is determined by the values of the coefficients serially scanned into the filter at run-time.

The "DW_coef Package" on page 8, defines the values required to implement a 48th order low-pass Kaiser Window filter. The coefficient values defining the Kaiser Window response are specified as a constant array. Because of the symmetry of coefficient values, only half of the values are specified. The filter impulse response is shown in Figure 1-6 on page 9.

Figure 1-5 3-Stage Cascade FIR Filter

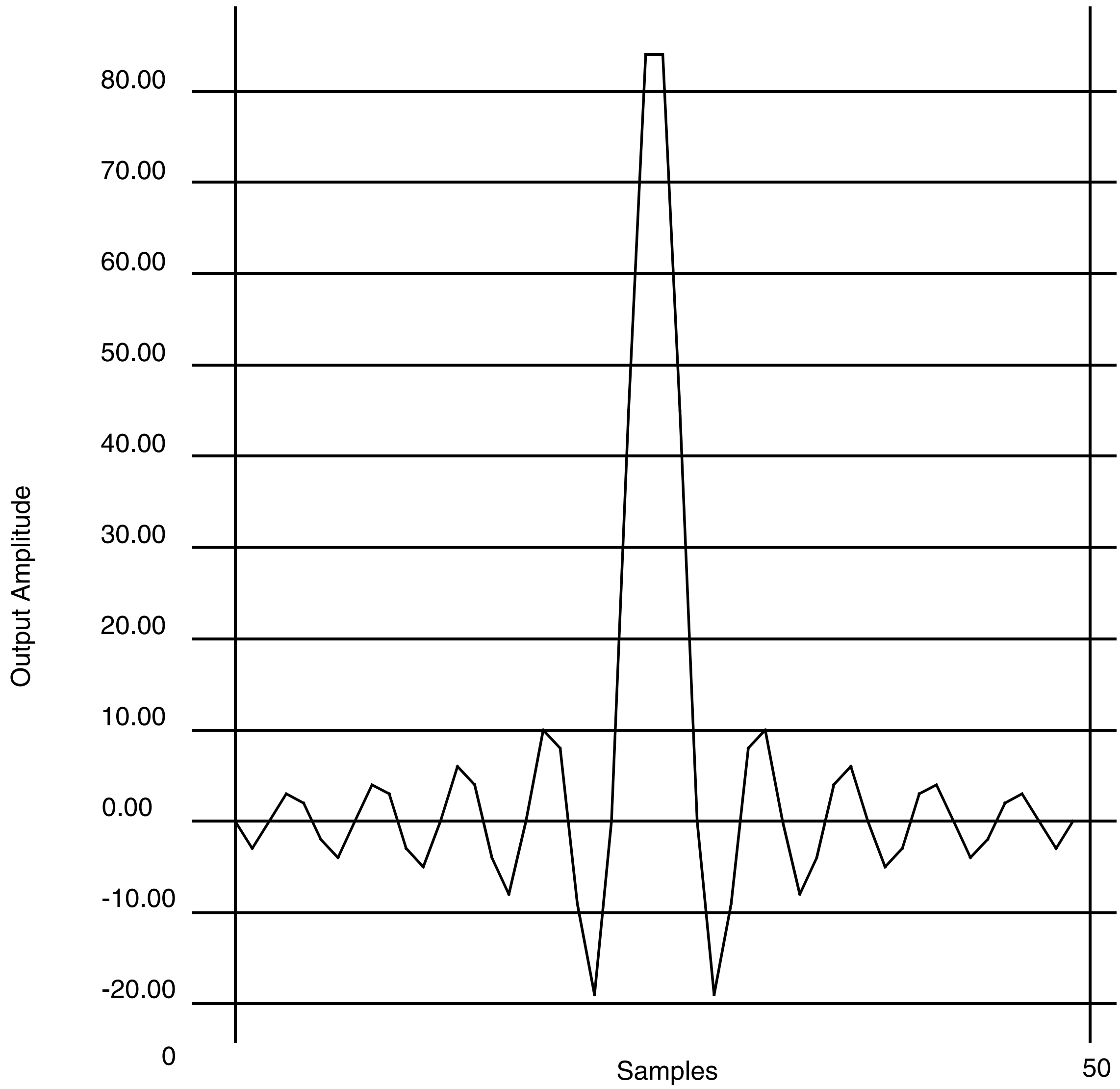


DW_coef Package

```
-- The following VHDL code defines the package that specifies
-- the value of each parameter and the coefficient values
-- defining the Kaiser Window response.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

package DW_coef is
  -- kaiser window LP FIR Filter
  -- only half of the coefficient array is specified because of symmetricity
  constant kaiser_order: INTEGER := 48;
  constant kaiser_coef_width: INTEGER := 12;
  type coef_half_array is array (0 to kaiser_order/2-1) of
                            std_logic_vector_(kaiser_coef_width-1 downto 0);
  constant kaiser_coef_half: coef_half_array :=
    ("111111111101",
     "000000000000",
     "000000000011",
     "000000000010",
     "111111111110",
     "111111111100",
     "000000000000",
     "000000000100",
     "000000000011",
     "111111111101",
     "111111111011",
     "000000000000",
     "000000000110",
     "000000000100",
     "111111111100",
     "111111111000",
     "000000000000",
     "000000001010",
     "000000001000",
     "111111110111",
     "111111101101",
     "000000000000",
     "000000101101",
     "000001010100"
     );
end DW_coef;
```

Figure 1-6 FIR Filter Impulse Response (order = 48, Kaiser Window)



HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp.all;

entity DW_fir_inst is
  generic (inst_data_in_width : POSITIVE := 8;
           inst_coef_width : POSITIVE := 8;
           inst_data_out_width : POSITIVE := 18;
           inst_order : POSITIVE := 6 );
  port (inst_clk      : in std_logic;
        inst_rst_n    : in std_logic;
        inst_coef_shift_en : in std_logic;
        inst_tc       : in std_logic;
        inst_data_in : in std_logic_vector(inst_data_in_width-1 downto 0);
        inst_coef_in : in std_logic_vector(inst_coef_width-1 downto 0);
        inst_init_acc_val : in
           std_logic_vector(inst_data_out_width-1 downto 0);
        data_out_inst : out std_logic_vector(inst_data_out_width-1 downto 0);
        coef_out_inst : out std_logic_vector(inst_coef_width-1 downto 0) );
end DW_fir_inst;

architecture inst of DW_fir_inst is
begin
  -- Instance of DW_fir
  U1 : DW_fir
    generic map (data_in_width => inst_data_in_width,
                 coef_width => inst_coef_width,
                 data_out_width => inst_data_out_width, order => inst_order )
    port map (clk => inst_clk,   rst_n => inst_rst_n,
              coef_shift_en => inst_coef_shift_en,   tc => inst_tc,
              data_in => inst_data_in,   coef_in => inst_coef_in,
              init_acc_val => inst_init_acc_val,   data_out => data_out_inst,
              coef_out => coef_out_inst );
end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_fir_inst( inst_clk, inst_rst_n, inst_coef_shift_en, inst_tc,
                    inst_data_in, inst_coef_in, inst_init_acc_val,
                    data_out_inst, coef_out_inst );
  parameter data_in_width = 8;
  parameter coef_width = 8;
  parameter data_out_width = 18;
  parameter order = 6;

  input inst_clk;
  input inst_rst_n;
  input inst_coef_shift_en;
  input inst_tc;
  input [data_in_width-1 : 0] inst_data_in;
  input [coef_width-1 : 0] inst_coef_in;
  input [data_out_width-1 : 0] inst_init_acc_val;
  output [data_out_width-1 : 0] data_out_inst;
  output [coef_width-1 : 0] coef_out_inst;

  // Instance of DW_fir
  DW_fir #(data_in_width, coef_width, data_out_width, order)
    U1 ( .clk(inst_clk),   .rst_n(inst_rst_n),
         .coef_shift_en(inst_coef_shift_en),   .tc(inst_tc),
         .data_in(inst_data_in),   .coef_in(inst_coef_in),
         .init_acc_val(inst_init_acc_val),   .data_out(data_out_inst),
         .coef_out(coef_out_inst) );
endmodule
```

Copyright Notice and Proprietary Information

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com