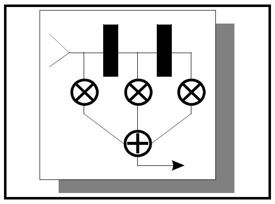


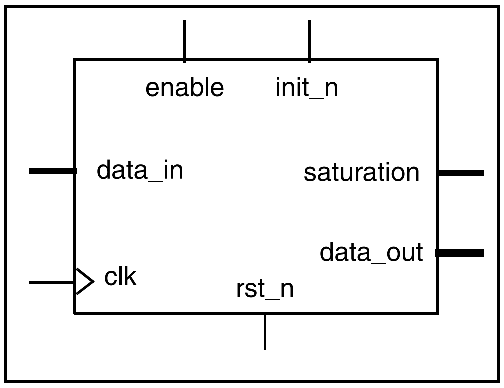
DW_iir_sc

High-Speed Digital IIR Filter with Static Coefficients

Version, STAR and Download Information: IP Directory

Features

- High-speed direct-form vector sum architecture
- High-speed transposed-form multiplier architecture
- Parameterized input, output, and feedback data widths
- Parameterized coefficient values and widths
- Parameterized fraction widths and saturation mode
- DesignWare datapath generator is employed for better timing and area



Applications

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

Description

DW_iir_sc is a high-speed digital IIR (Infinite Impulse Response) filter designed for Digital Signal Processing applications employing very high sampling rates.

The coefficient values, coefficient widths, and data widths are parameterized.

Table 1-1 Signal Description

Name	Width	I/O	Description
clk	1 bit	Input	Clock signal. All internal registers are sensitive on the positive edge of `clk` and all setup and hold times are with respect to this edge of `clk`.
rst_n	1 bit	Input	Synchronous reset, active-low. Clears all registers
init_n	1 bit	Input	Synchronous, active-low signal to clear all registers
enable	1 bit	Input	Active-high signal to enable all registers

Table 1-1 Signal Description

Name	Width	I/O	Description
data_in	*data_in_width* bit(s)	Input	Input data.
data_out	*data_out_width* bit(s)	Output	Accumulated sum of products of the IIR filter.
saturation	1 bit	Output	Used to indicate the output data or feedback data is in saturation.

Table 1-2 Parameter Description

Parameter	Values	Description
data_in_width	≥ 2 Default = 4	Input data word length
data_out_width	≥ 2 Default = 6	Width of output data. This parameter should also satisfy the following equation: *data_out_width* ≤ maximum(*feedback_width*, *data_in_width* + *frac_data_out_width*) + *max_coef_width* + 3 - *frac_coef_width* This upper bound comes from the internal datapath widths of the architectures shown in Figure 1 and Figure 2.
frac_data_out_width	0 to *data_out_width* –1 Default = 0	Width of fraction portion of data_out
feedback_width	≥ 2 Default = 8	Width of feedback_data (feedback_data is internal to the DW_iir_sc)
max_coef_width	≥ 2 to 31 Default = 4	Maximum coefficient word length
frac_coef_width	0 to *max_coef_width* –1 Default = 0	Width of the fraction portion of the coefficients
saturation_mode	0 or 1 Default = 1	Controls the mode of operation of the saturation output
out_reg	0 or 1 Default = 1	Controls whether data_out and saturation are registered
A1_coef	*range* Default = -2	Constant coefficient value A1 $range = -2^{max\_coef\_width-1}$ to $2^{max\_coef\_width-1} - 1$
A2_coef	*range* Default = 3	Constant coefficient value A2 $range = -2^{max\_coef\_width-1}$ to $2^{max\_coef\_width-1} - 1$
B0_coef	*range* Default = 5	Constant coefficient value B0 $range = -2^{max\_coef\_width-1}$ to $2^{max\_coef\_width-1} - 1$
B1_coef	*range* Default = -6	Constant coefficient value B1 $range = -2^{max\_coef\_width-1}$ to $2^{max\_coef\_width-1} - 1$

Table 1-2 Parameter Description

Parameter	Values	Description
B2_coef	*range* Default = -2	Constant coefficient value B2 *range* = $-2^{max\_coef\_width-1}$ to $2^{max\_coef\_width-1} - 1$

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Required
mult	Multiplier synthesis model	DesignWare
vsum	Vector sum synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW03.DW_IIR_SC_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/dw_iir_sc_sim.vhd	VHDL simulation model source code
dw/sim_ver/dw_iir_sc.v	Verilog simulation model source code

Table 1-5 Modes of Operation

saturation_mode	Operation
0	$-2^{data\_out\_width-1} \leq data\_out \leq 2^{data\_out\_width-1} - 1$ and $-2^{feedback\_width-1} \leq feedback\_data \leq 2^{feedback\_width-1} - 1$
1	$-2^{data\_out\_width-1} + 1 \leq data\_out \leq 2^{data\_out\_width-1} - 1$ and $-2^{feedback\_width-1} + 1 \leq feedback\_data \leq 2^{feedback\_width-1} - 1$

Functional Description

The data-flow diagram for the DW_iir_sc multiplier architecture is shown in Figure 1-1. The vector sum architecture data-flow diagram is shown in Figure 1-2 on page 5.

Figure 1-1 DW_iir_sc Multiplier Architecture

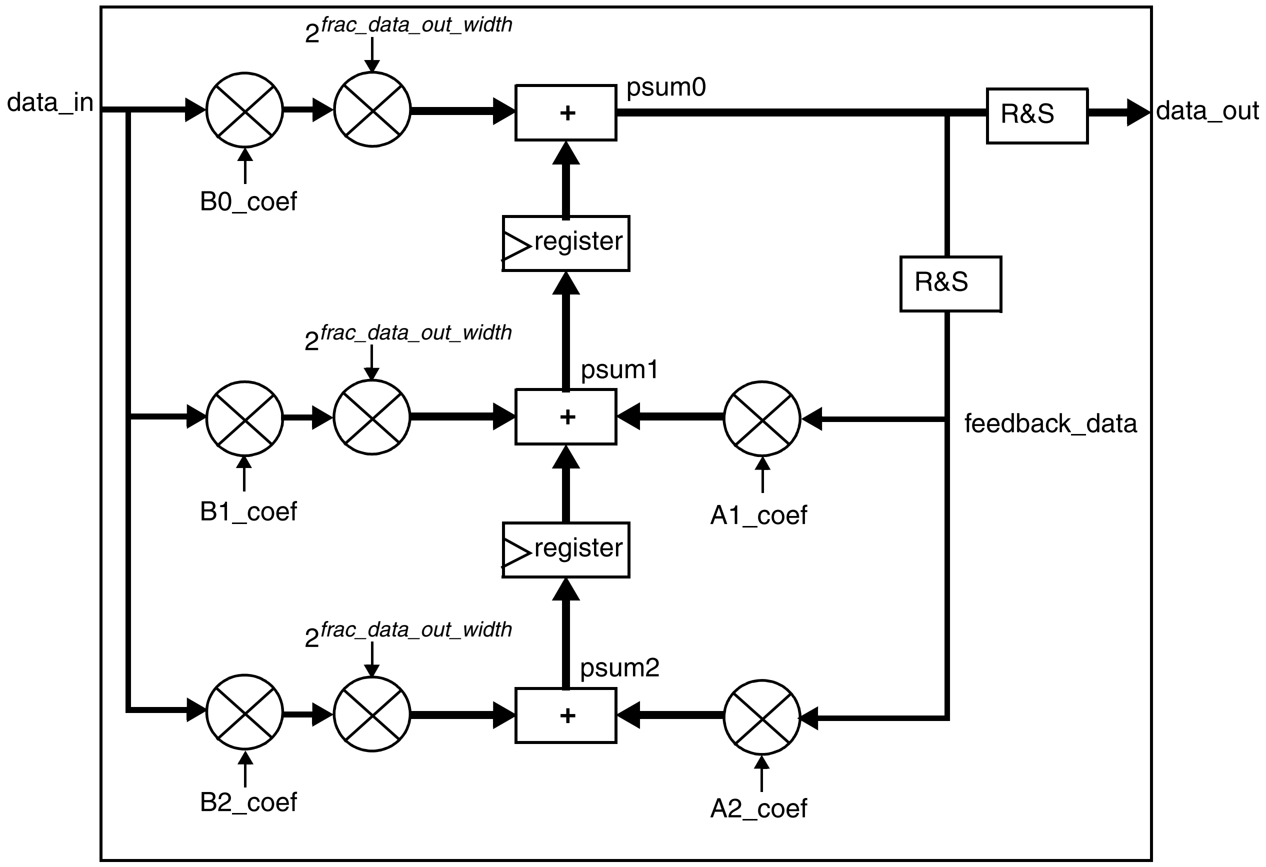
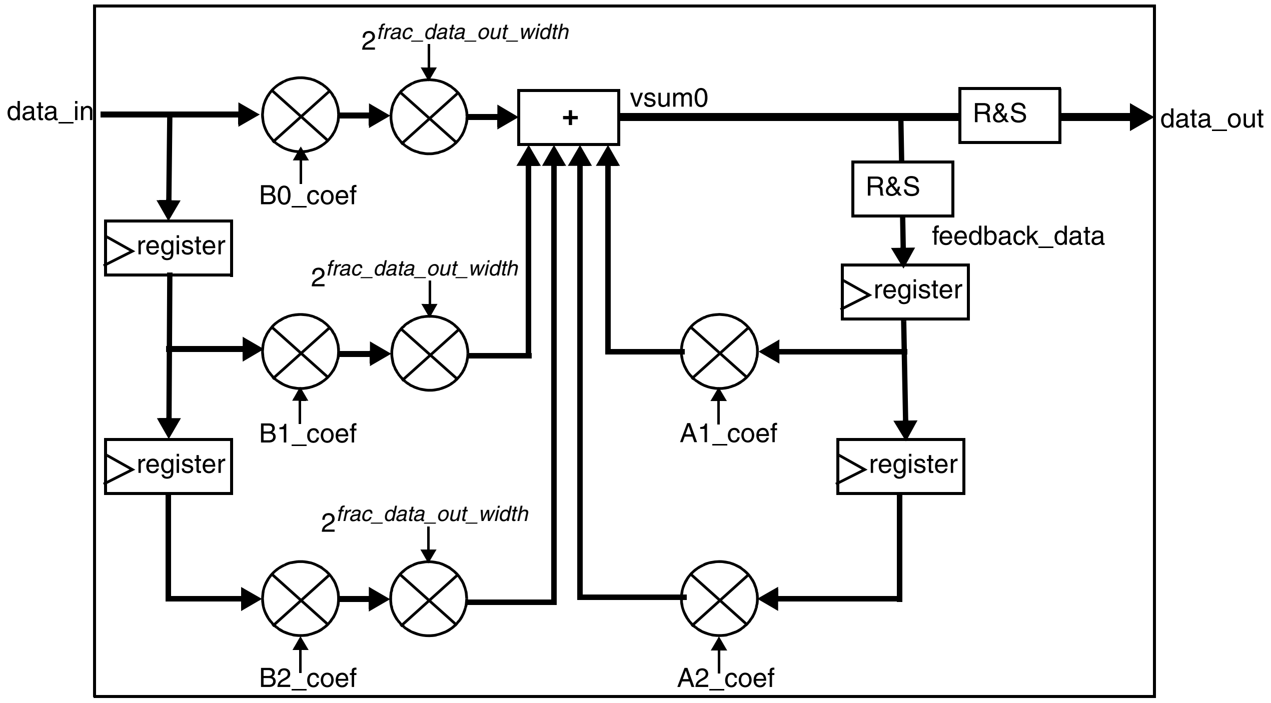


Figure 1-2 dw_iir_sc Vector Sum Architecture



The DW_iir_sc is clocked with the `clk` signal and is sensitive to the rising edge of `clk`. An active-low, asynchronous reset signal, `rst_n`, clears all registers to the zero state. An active-low, synchronous initialization signal, `init_n`, clears all registers to the zero state on the rising edge of `clk`. Signal `init_n` is also used to asynchronously gate `data_in` so that internally generated signal `gated_data_in` becomes zero if `init_n` is zero. The filter is set by choosing the parameters for the coefficients.

Because the output width and the feedback width are parameters, it is possible for the filter to try to generate a value that exceeds the two's complement range of the parameter of either output width or feedback width, or both. When this case occurs, the output signal `saturation` is asserted. Because the two's complement of a given width can represent a maximum negative number that is one larger than the maximum positive number, a parameter called *saturation_mode* is provided.

The operations controlled by *saturation_mode* is shown in Table 1-5. When *saturation_mode=0*, the full range of numbers is employed. When *saturation_mode=1*, the range of numbers is symmetrically limited.

If *frac_data_out_width>0*, the products of `data_in` and coefficients are scaled up by $2^{frac\_data\_out\_width}$ in order to align the fractional parts of addition operands. If *frac_coef_width>0*, the right *frac_coef_width* bits of `psum0` in Figure 1 and `vsum0` in Figure 2 are truncated and rounded to the nearest for `feedback_data` and `data_out`. In Figure 1-1 on page 4 and Figure 1-2 on page 5, block "R&S" implements the operation of rounding and saturation.

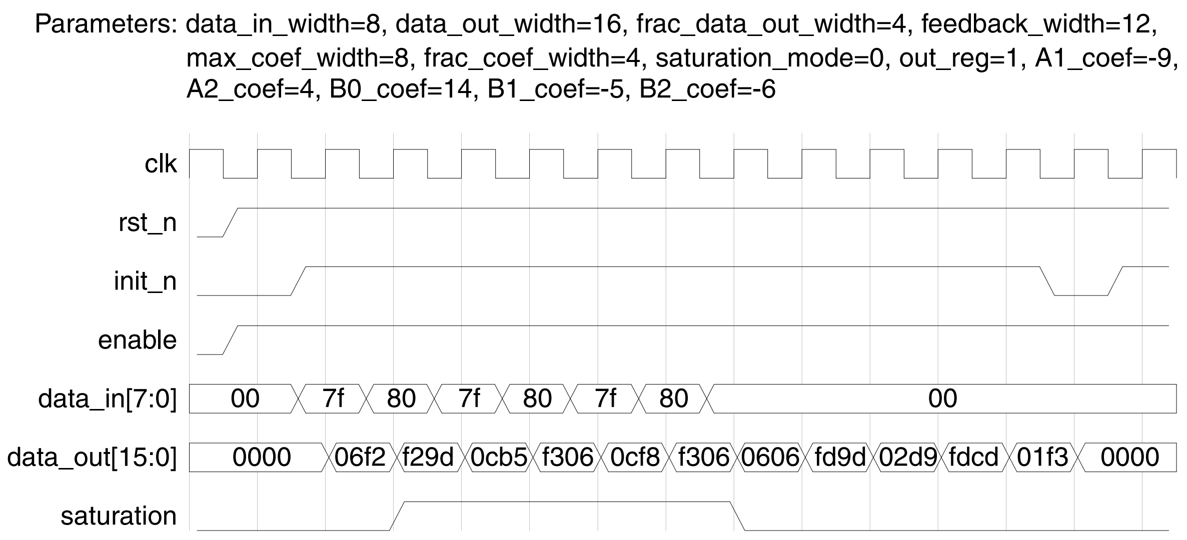
If *feeback_width=data_out_width*, the rounding and saturation circuitry for `feedback_data` and `data_out` is shared.

The mult architecture (Figure 1-1 on page 4) is a transposed-form implementation of an IIR filter with the delay elements repositioned. It has the benefit of breaking up the critical path on `clk` to `data_out`, making it faster. In some cases with certain parameter settings, the total number of flip-flops is reduced in transposed-form implementation. The mult architecture finds every possible case of coefficients equal to one, coefficients equal to zero, and two coefficients being equal, and then simplifies the logic accordingly.

The vector sum (vsum) architecture (Figure 1-2 on page 5) is a direct-form implementation of an IIR filter. It is similar to the multiplier architecture, except that all of the multipliers have been absorbed into a massive vector sum of all the Booth coded partial products of the constant coefficients. Furthermore, three or more adjacent 1 bits in any group of coefficients are reduced to a positive 1 and a negative 1, which results in just two partial products for each such pattern of 1s. For example, the binary number “0111100” can be replaced with a positive “1000000” and a negative ““0000100”, reducing the ones count from four to two, thus reducing the number of partial products from four to two. This is done using signed binary arithmetic and, each partial product is either a positive input or a negative input to the final vector sum.

Timing Waveforms

Figure 1-3 dw_iir_sc Timing Diagram


Parameters: data_in_width=8, data_out_width=16, frac_data_out_width=4, feedback_width=12, max_coef_width=8, frac_coef_width=4, saturation_mode=0, out_reg=1, A1_coef=-9, A2_coef=4, B0_coef=14, B1_coef=-5, B2_coef=-6
clk
rst_n
init_n
enable
data_in[7:0]
00
7f
80
7f
80
7f
80
00
data_out[15:0]
0000
06f2
f29d
0cb5
f306
0cf8
f306
0606
fd9d
02d9
fdcd
01f3
0000
saturation

Theory of Operation

In a sampled linear system, the inputs and outputs are coupled by finite difference equations. These equations can be written as follows:

$$\sum_{r=0}^{N} -A_r y(n-r) = \sum_{k=0}^{M} B_k x(n-k)$$

Because changes in the output cannot precede changes in the input, the output can be computed from the current input, previous inputs, and previous outputs as follows:

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) + \sum_{r=1}^{N} a_r y(n-r)$$

In the FIR filter, the outputs are not dependent upon the previous states of the outputs (the a_r coefficients are all zero). Thus, the system's response will be of only finite duration (finite impulse response). The IIR filter contains feedback from the previous outputs. Some a_r coefficients are non-zero. Thus, a filter set in motion may continue to respond forever (infinite impulse response), though usually at diminished amplitudes (damping). Limiting it to second order, the following simpler biquad equation is derived:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + a_1 y(n-1) + a_2 y(n-2)$$

HDL Usage Through Component Instantiation - VHDL

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

entity DW_iir_sc_inst is
  generic (inst_data_in_width   : POSITIVE := 4;
           inst_data_out_width  : POSITIVE := 6;
           inst_frac_data_out_width : NATURAL := 0;
           inst_feedback_width  : POSITIVE := 8;
           inst_max_coef_width  : POSITIVE := 4;
           inst_frac_coef_width : NATURAL := 0;
           inst_saturation_mode : NATURAL := 1;
           inst_out_reg : NATURAL := 1;
           inst_A1_coef : INTEGER := -2;
           inst_A2_coef : INTEGER :=  3;
           inst_B0_coef : INTEGER :=  5;
           inst_B1_coef : INTEGER := -6;
           inst_B2_coef : INTEGER := -2 );
  port (inst_clk      : in std_logic;   inst_rst_n   : in std_logic;
        inst_init_n   : in std_logic;   inst_enable : in std_logic;
        inst_data_in  : in std_logic_vector(inst_data_in_width-1 downto 0);
        data_out_inst : out std_logic_vector(inst_data_out_width-1 downto 0);
        saturation_inst : out std_logic  );
end DW_iir_sc_inst;

architecture inst of DW_iir_sc_inst is
begin
  -- Instance of DW_iir_sc
  U1 : DW_iir_sc
    generic map (data_in_width => inst_data_in_width,
                 data_out_width => inst_data_out_width,
                 frac_data_out_width => inst_frac_data_out_width,
                 feedback_width => inst_feedback_width,
                 max_coef_width => inst_max_coef_width,
                 frac_coef_width => inst_frac_coef_width,
                 saturation_mode => inst_saturation_mode,
                 out_reg => inst_out_reg,   A1_coef => inst_A1_coef,
                 A2_coef => inst_A2_coef,   B0_coef => inst_B0_coef,
                 B1_coef => inst_B1_coef,   B2_coef => inst_B2_coef )
    port map (clk => inst_clk,   rst_n => inst_rst_n,
              init_n => inst_init_n,   enable => inst_enable,
              data_in => inst_data_in,   data_out => data_out_inst,
              saturation => saturation_inst );
end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_iir_sc_inst(inst_clk, inst_rst_n, inst_init_n, inst_enable,
                      inst_data_in, data_out_inst, saturation_inst );

  parameter data_in_width = 4;
  parameter data_out_width = 6;
  parameter frac_data_out_width = 0;
  parameter feedback_width = 8;
  parameter max_coef_width = 4;
  parameter frac_coef_width = 0;
  parameter saturation_mode = 1;
  parameter out_reg = 1;
  parameter A1_coef = -2;
  parameter A2_coef =  3;
  parameter B0_coef =  5;
  parameter B1_coef = -6;
  parameter B2_coef = -2;

  input inst_clk;
  input inst_rst_n;
  input inst_init_n;
  input inst_enable;
  input [data_in_width-1 : 0] inst_data_in;
  output [data_out_width-1 : 0] data_out_inst;
  output saturation_inst;

  // Instance of DW_iir_sc
  DW_iir_sc #(data_in_width, data_out_width, frac_data_out_width,
            feedback_width, max_coef_width, frac_coef_width,
            saturation_mode, out_reg, A1_coef, A2_coef, B0_coef,
            B1_coef, B2_coef)
    U1 (.clk(inst_clk),   .rst_n(inst_rst_n),   .init_n(inst_init_n),
        .enable(inst_enable),   .data_in(inst_data_in),
        .data_out(data_out_inst),   .saturation(saturation_inst) );
endmodule
```