CYPRESS SEMICONDUCTOR CORPORATION

Internal Correspondence

Date:	Tuesday 1/8/2013	**WW: 1302**
To:	**PSoC Apps**	
Author:	Chris Keeser (KEES)	
Author File#:	KEES#182	
Subject:	**UDB IIR filter component**	
Distribution:	PSoC Apps	

Summary:

This memo introduces and distributes an IIR filter implemented in a UDB. This component implements the first order IIR low pass filter discussed in KEES#84. It features 8 cutoff frequency choices (as a function of the sample clock rate, i.e. ~1/10*Fsample, to ~1/2000*Fsample), configurable output choices of: 2’s Compliment, Sign / Magnitude, Offset mode (0 – 254) and Absolute value. The maximum output can be configured to saturate at +/- 127 all the way down to +/- 1. The component has a configurable decimation rate to arbitrarily throw away 0 to 127 samples between outputs. The filter outputs can also be staggered to provide ways for multiple filters to run in parallel without all the filters finishing at the same time. The DMA wizard is aware of the output of this filter to facilitate easily configuring DMA to move data from the filter to some other location when the calculation is complete. The filter is extremely fast, and completes 1 filter calculation in a minimum of 5 clocks (BETA of 0 and 2’s compliment output mode) and a maximum of 14 clocks (BETA of 7 and Sign/Mag, Offset or Absolute value output mode)

This filter can be used as a ‘decimator’ for an SC/CT based modulator when you desire an output sample rate equal to the modulator clock (not possible with a traditional CIC decimator) or if you wish to convert any single bit bitstream into a digital number from -127 to +127 through low pass filtering. This could be used as a duty cycle measurement tool, or for any other creative use. As an example, multiple instances of this component were used in conjunction with the SC/CT block modulator, LUT state machines, XOR gates and another special purpose datapath block to produce a 100% hardware based FSK demodulator.

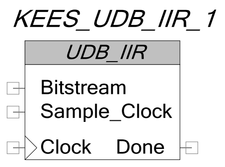
<u>Attached File Summary:</u>

File #2 is the component, exported using the component export feature of Creator 2.2. To use the file, download it and remove the .PDF extension.

File #3 is the an example project bundle, including the SC/CT modulator, IIR filter and configure with DMA moving the data to a VDAC. To use the file, download it and remove the .PDF extension.

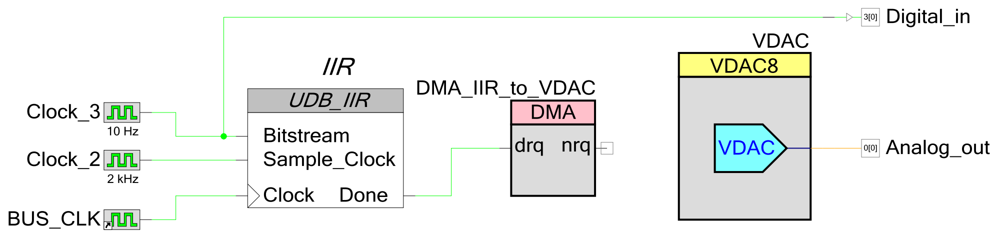
Details:

This is what the component looks like:

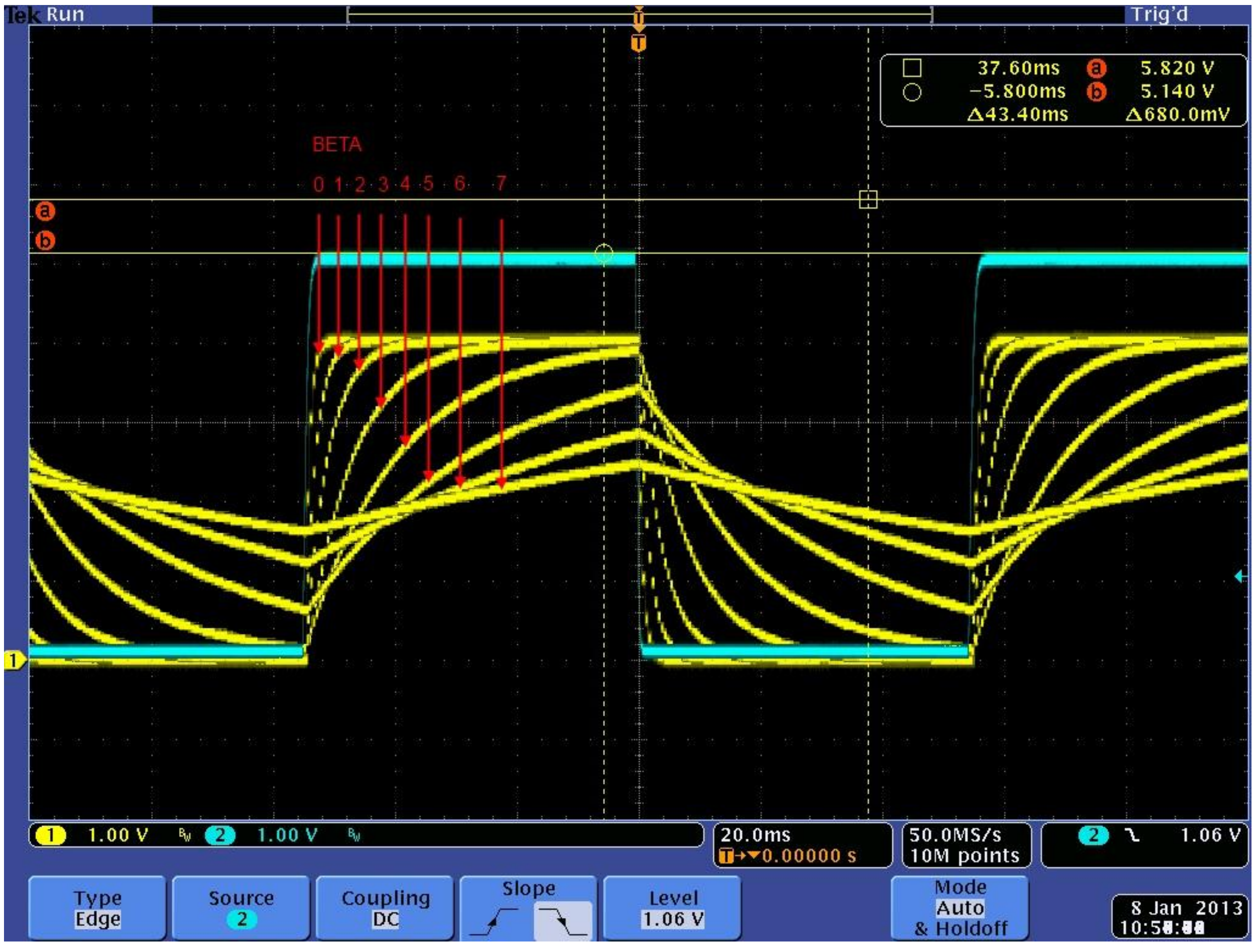


- You only need to call KEES_UDB_IIR_Start();
- The bitstream input signal is connected to any digital source with information that you want to filter.
- The sample clock is asserted at the sample rate that either controls the generation of the bitstream or is related to the desired cutoff frequency.
- The clock input is the clock that drives the UDB / Datapath, and must be at least 5x to 14x the sample clock (depending on configuration options).
- The Done signal is asserted when the filter calculation is complete. The completed result is loaded into a FIFO so up to 4 results can be stored before the output must be read. Usually, the Done signal is connected to the DRQ terminal of a DMA channel reading the output of the filter.

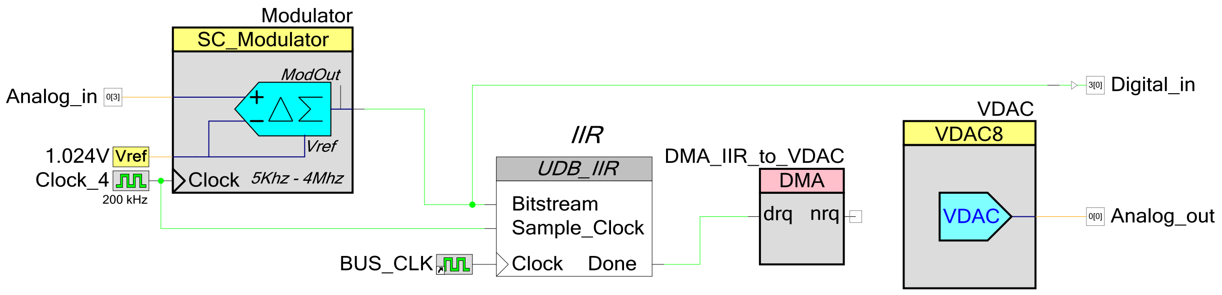
As an example, a square wave of 10 Hz was fed into the IIR filter with a sample clock of 2 Khz. The result was configured in 'offset' mode, which produces a result centered around 127 and is ideal for using DMA to send the result directly to a VDAC. The 'BETA' parameter was configured from 0 to 7, changing the cutoff frequency of the filter.



This is the result. Digital In is the signal in blue, and Analog Out is the signal in yellow. The results were layered on top of each other to illustrate the familiar shape produced by varying the cutoff frequency:

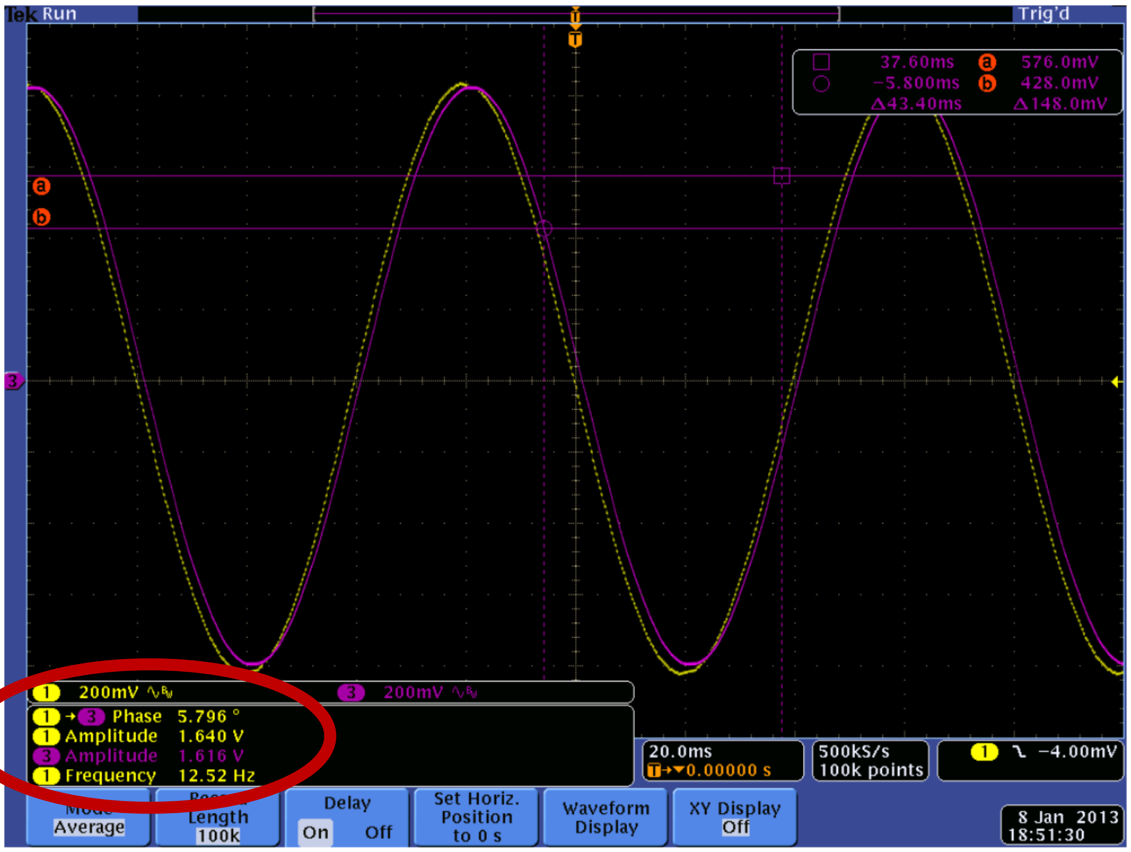


When used in conjunction with the SC/CT delta sigma modulator component, these two blocks forms a rudimentary ADC. Adding the VDAC creates an analog low pass filter (similar to the ADC –> DFB -> VDAC project, but without any of the complexity):



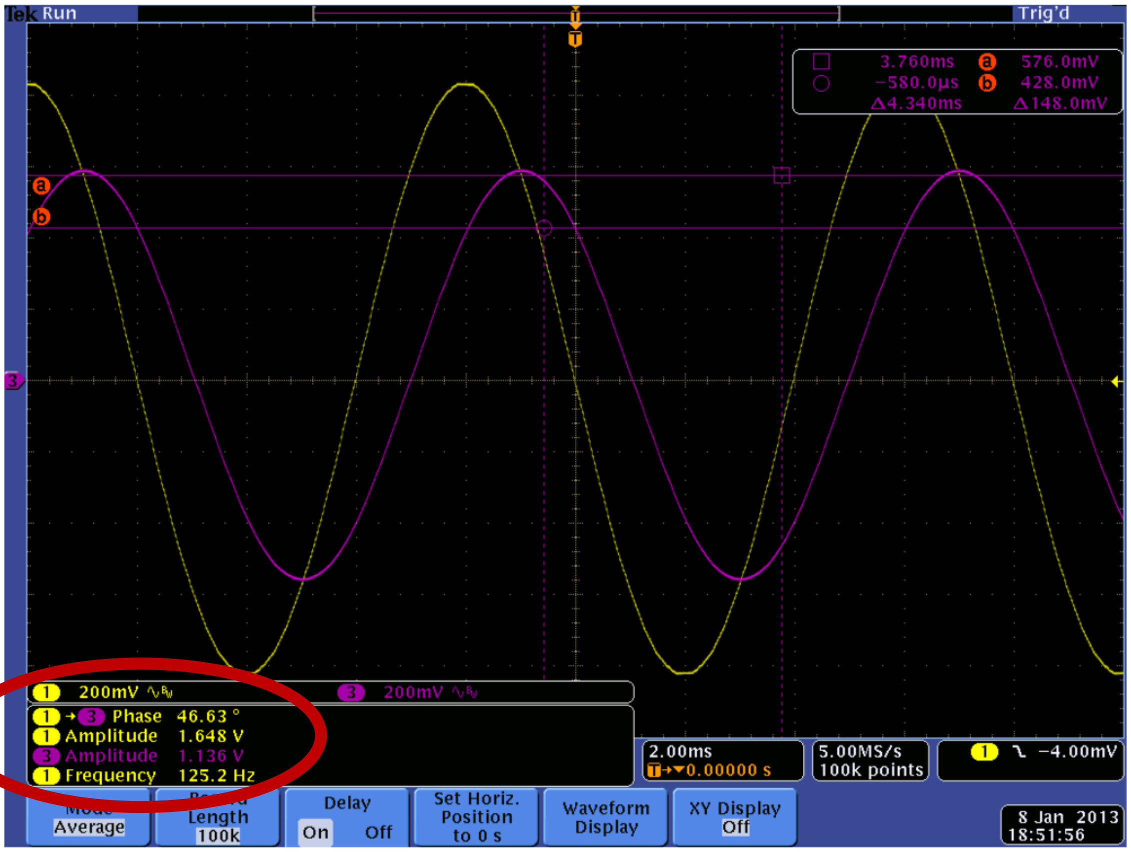
The IIR filter has a BETA set to 7, resulting in a cutoff frequency of 0.000623*Fsample = 0.000623*200 Khz = 124.58 Hz. The input signal is shown as the Yellow trace, and the output signal (pulled directly from the VDAC output) is shown in the purple trace. Note that the filter converts *a single bit output from*

*the modulator at 200 Khz into a low pass filtered **digital word** at the same rate!*
There is no code running to perform these functions.



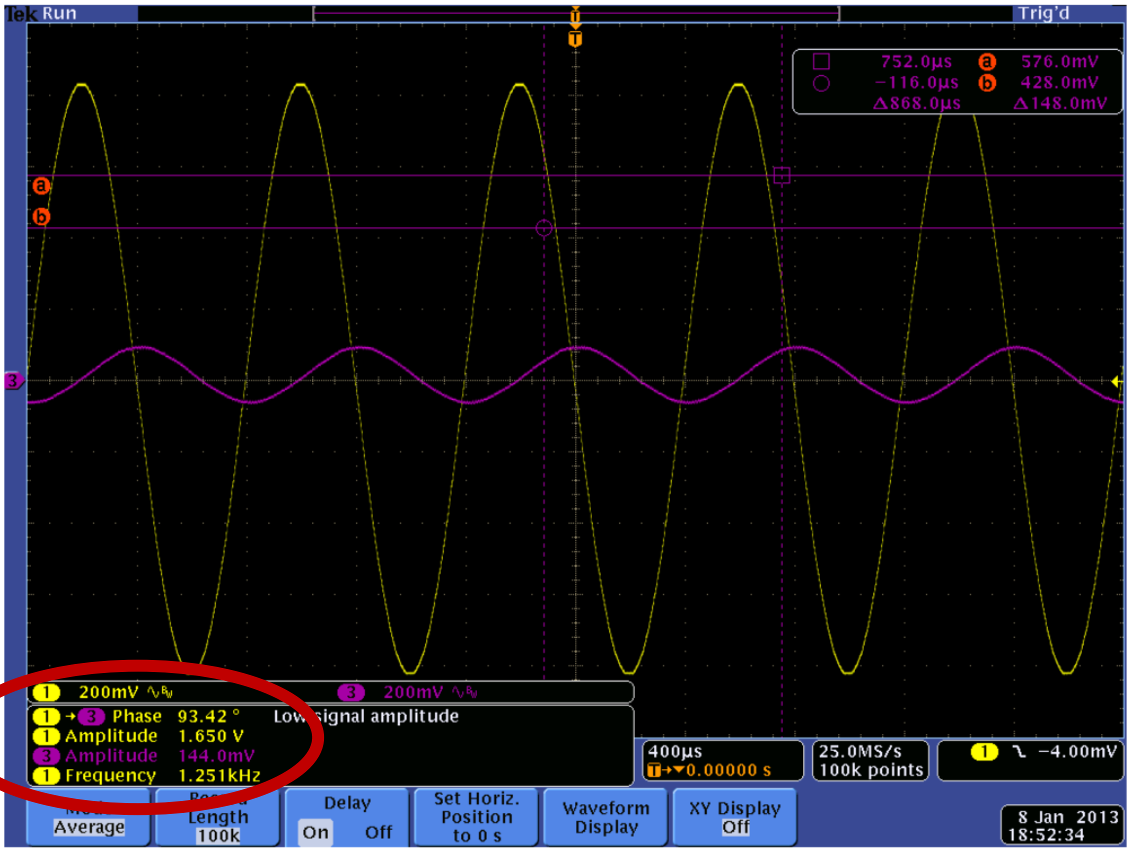
```
Tek Run                                              Trig'd
□  37.60ms    a  576.0mV
○  −5.800ms   b  428.0mV
   Δ43.40ms      Δ148.0mV

1  200mV                 3  200mV
1 → 3 Phase      5.796 °
1 Amplitude      1.640 V
3 Amplitude      1.616 V
1 Frequency      12.52 Hz

20.0ms          500kS/s        1  ⤵  −4.00mV
T→▼0.00000 s    100k points

Mode      Record    Delay     Set Horiz.   Waveform   XY Display   8 Jan  2013
Average   Length    On  Off   Position     Display    Off          18:51:30
          100k                to 0 s
```

DPO4104 - 4:57:21 PM   Tuesday 1/8/2013



```
1 → 3 Phase   46.63 °
1 Amplitude   1.648 V
3 Amplitude   1.136 V
1 Frequency   125.2 Hz
```

DPO4104 - 4:57:48 PM   Tuesday 1/8/2013

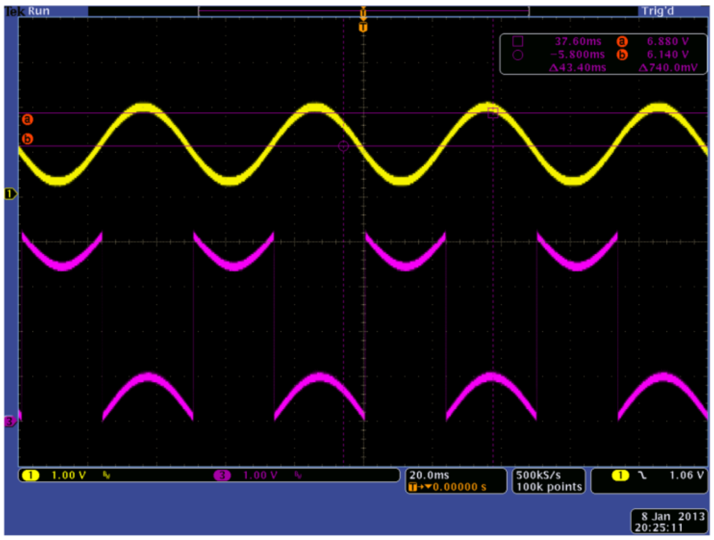


```
752.0µs   a 576.0mV
−116.0µs  b 428.0mV
Δ868.0µs    Δ148.0mV

1 200mV        3 200mV
1→3 Phase      93.42 °    Low signal amplitude
1 Amplitude    1.650 V
3 Amplitude    144.0mV
1 Frequency    1.251kHz

400µs            25.0MS/s      1 ⌐ −4.00mV
T→▼0.00000 s     100k points
```

DPO4104 - 4:58:26 PM   Tuesday 1/8/2013

<u>Examples of the output options:</u>

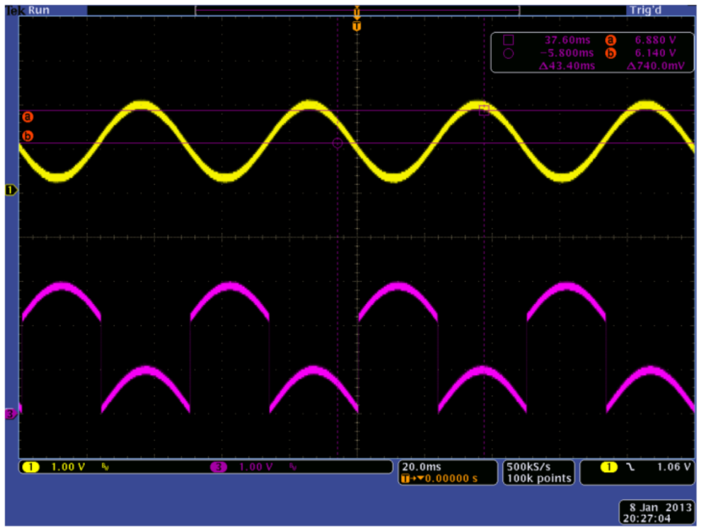
2’s compliment



```
37.60ms   a 6.880 V
−5.800ms  b 6.140 V
Δ43.40ms    Δ740.0mV

1 1.00 V     3 1.00 V
20.0ms           500kS/s       1 ⌐ 1.06 V
T→▼0.00000 s     100k points
```

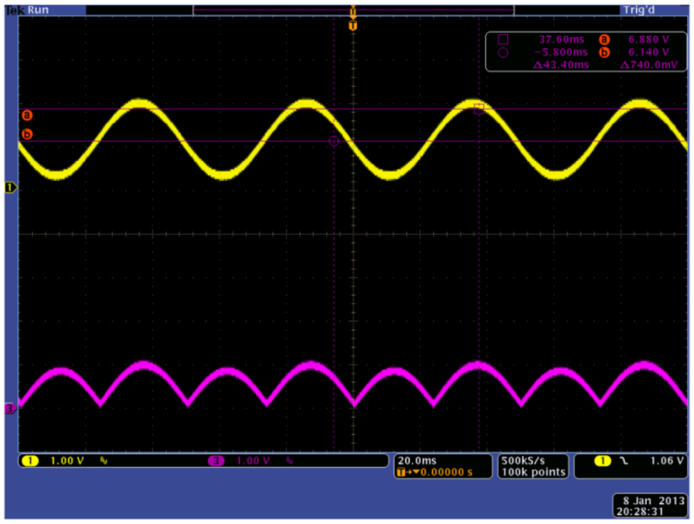
DPO4104 - 6:31:05 PM   Tuesday 1/8/2013

Sign / Magnitude



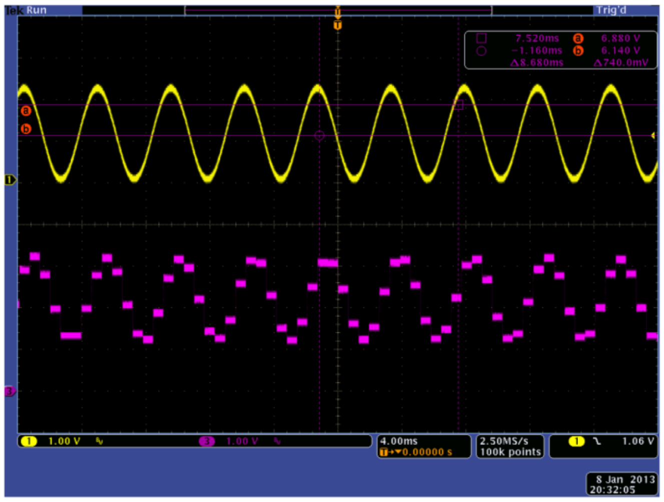
DPO4104 - 6:32:58 PM Tuesday 1/8/2013

Absolute Value:



DPO4104 - 6:34:24 PM Tuesday 1/8/2013

Offset with decimation:

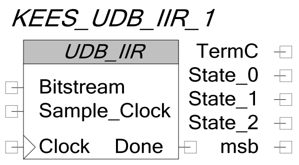


DPO4104 - 6:37:58 PM   Tuesday 1/8/2013

Obviously you wouldn’t always use these modes with the VDAC, you would use them when the data is meant to be consumed by something else in the PSoC. For instance, I also used the Absolute Value mode in my FSK demodulator application. 2’s Compliment is great for firmware, and I’ll leave it up to you to come up with neat uses for the other features.

Some technical details:

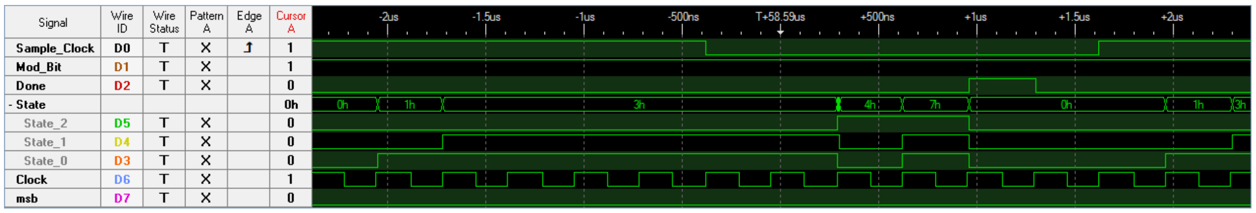
The UDB IIR component uses 2 datapaths (16 bits wide) to produce an 8 bit result. The extra width ensures that there are sufficient bits to perform the calculations of the filter. There is a debug mode option that brings out internal signals to visualize the filter calculation process. When the debug option is enabled, the component will have the following extra terminals:



A sample logic trace showing the state progressions for a 2’s compliment output as well as a sign / magnitude conversion (2 extra clock cycles)

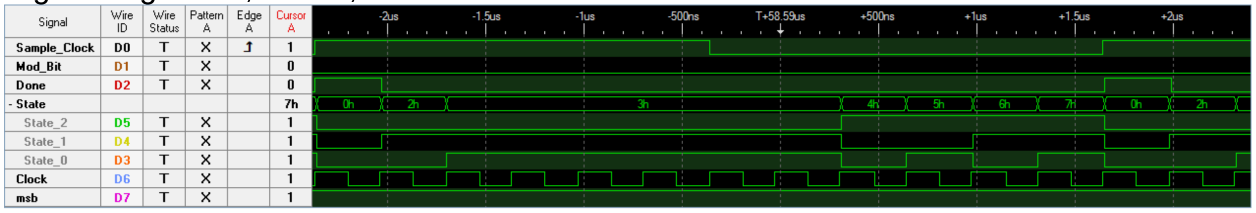
Beta set to 5

2’s compliment calculation:



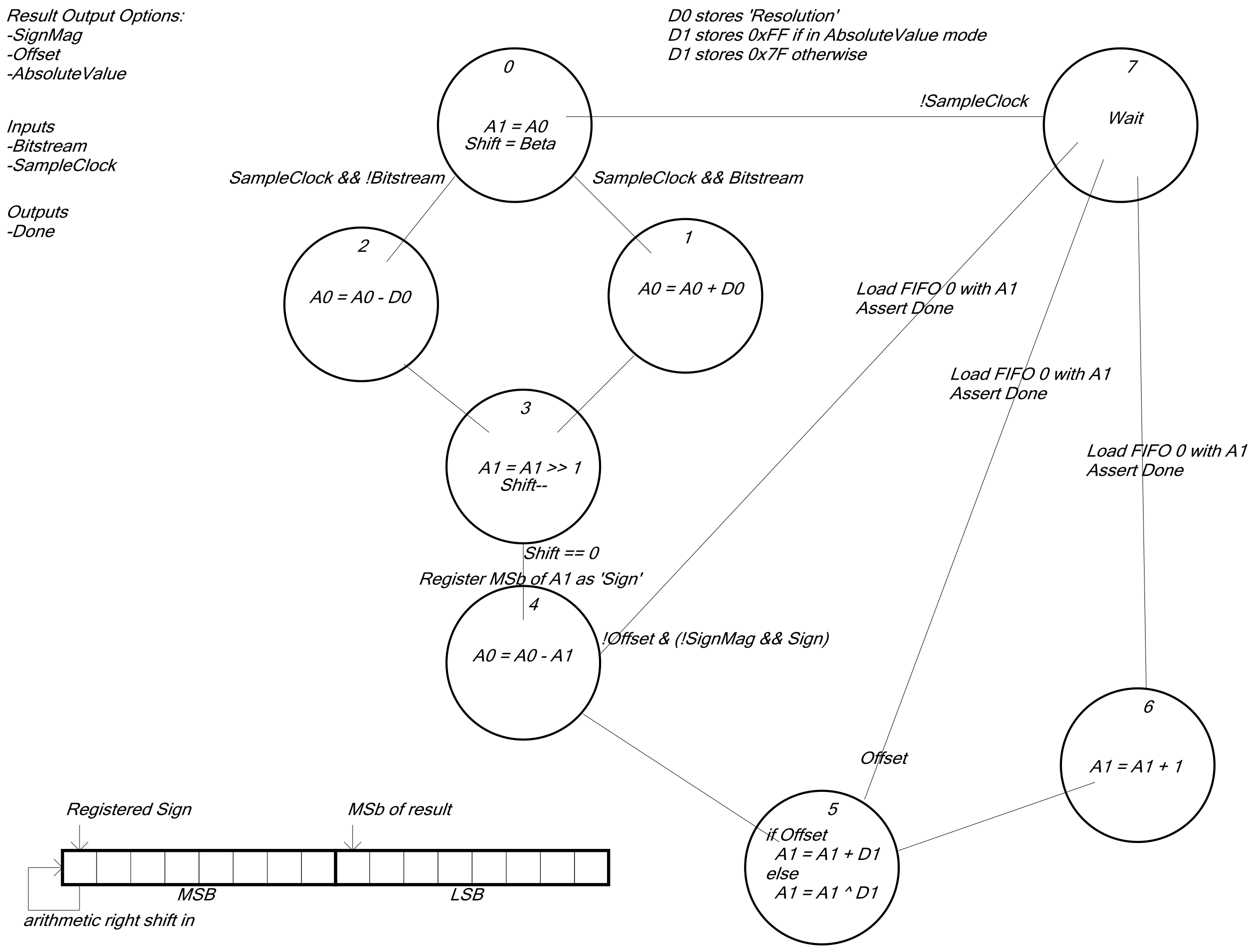
D6 Cycles A->B: 462 D3 Period: 1.60594us Interval T->C: 500ns D1 Transitions A->B: 0
UDB_IIR.LPF Tuesday 11/27/2012 1:35:36 PM

Sign / Magnitude, Offset, Absolute value extra calculation:



D6 Cycles A->B: 453 D3 Period: 1.60671us Interval T->C: 500ns D1 Transitions A->B: 0
UDB_IIR.LPF Tuesday 11/27/2012 1:34:56 PM

State diagram and datapath configuration:



What isn't shown is the Count7 that controls the decimation by asserting the done signal only when the Count7 reaches 0. The Count7 also controls the staggering.

Appendix: Verilog datapath configuration.

```
//`#start header` -- edit after this line, do not edit this line
// ========================================
//
// Copyright YOUR COMPANY, THE YEAR
// All Rights Reserved
// UNPUBLISHED, LICENSED SOFTWARE.
//
// CONFIDENTIAL AND PROPRIETARY INFORMATION
// WHICH IS THE PROPERTY OF your company.
//
// ========================================
`include "cypress.v"
//`#end` -- edit above this line, do not edit this line
// Generated on 11/26/2012 at 16:34
// Component: KEES_UDB_IIR_v1_00
module KEES_UDB_IIR_v1_00 (
    output  reg Done,
    output  State_0,
    output  State_1,
    output  State_2,
    output  msb,
    output  TermC,
    input   Bitstream,
    input   Clock,
    input   Sample_Clock
);
    parameter [2:0] Beta = "3'b000";
    parameter [7:0] Resolution = "8'b00000000";
    parameter Enable_Sign_Magnitude = "1'b0";
    parameter Decimation = "7'b0000000";
    parameter Enable_ABS = "1'b0";
    parameter Enable_Offset = "1'b0";

//`#start body` -- edit after this line, do not edit this line

//State Machine Variables
    localparam STATE_0  = 4'h0;                 // Wait for rising edge of Sample_Clock
    localparam STATE_1  = 4'h1;                 // increment A0, store result in A0 and A1
    localparam STATE_2  = 4'h2;                 // decrement A0, store result in A0 and A1
    localparam STATE_3  = 4'h3;                 // shift A1 right
    localparam STATE_4  = 4'h4;                 // subtract A1 from A0 and store result in A0
    localparam STATE_5  = 4'h5;                 // invert A0, store result in A0
    localparam STATE_6  = 4'h6;                 // add 1, store result in A0 and A1
    localparam STATE_7  = 4'h7;                 // wait for sample clock to go low

    reg [2:0] State;          // current state, dives config ram for datapath
    reg [2:0] Shift;          // counts the number of times the shift occurs
    reg Sign;                 // stores the sign of the shifted result for sign magnitude conversion

    reg Decrement;
    wire TermCount;
    wire [1:0] temp;
    //wire abs_mode;

    assign msb = Sign;
    assign TermC = TermCount;
    //assign abs_mode = Enable_ABS;

    assign State_0 = State[0];
    assign State_1 = State[1];
    assign State_2 = State[2];

    //assign Decrement = (State == STATE_7);

    always @(posedge Clock)
    begin
        Done <= 0;
        Decrement <= 0;
        case(State)
            STATE_0:  // wait for sample_clock to go high
            begin
                Shift <= Beta;  // shift by this value + 1  beta = 2^(value+1)
                if((Sample_Clock == 1) && (Bitstream == 1))
                begin
                    State <= STATE_1;
                    Decrement <= 1;
                end
                else if((Sample_Clock == 1) && (Bitstream == 0))
                begin
                    State <= STATE_2;
                    Decrement <= 1;
                end
                else
                begin
                    State <= STATE_0;
                end
            end

            STATE_1:  // increment A0, store in A0 and A1, then move to shift state
            begin
                State <= STATE_3;
            end

            STATE_2:  // decrement A0, store in A0 and A1, then move to shift state
            begin
                State <= STATE_3;
            end

            STATE_3:  // shift by BETA
            begin  // ARITHMATIC shift information in the UDB BROS (001-08005) page # 99 under SI SELA
and SI SELB
                Shift <= Shift - 1'b1;
                if(Shift == 0)
                begin
                    Sign <= temp[1];
                    State <= STATE_4;
                end
                else
                begin
                    State <= STATE_3;
                end
            end

            STATE_4:  // subtract A1 from A0 -> A0
            begin
                if(Enable_Offset == 1)
                begin
                    State <= STATE_5;
                end
                else if(Enable_Sign_Magnitude == 1 && Sign == 1)
                begin
                    State <= STATE_5;
                end
                else
                begin
                    State <= STATE_7;
                    Done <= TermCount;
                end
            end

            STATE_5:  // invert all the bits except the MSBs, store result in A1 (or add offset)
            begin
                if(Enable_Offset == 0)
                begin
                    State <= STATE_6;
                end
                else
                begin
                    Done <= TermCount;
                    State <= STATE_7;
                end
            end

            STATE_6:  // add 1, store result in A1
            begin
                Done <= TermCount;
                State <= STATE_7;
            end
            STATE_7:  // wait for sample clock to go low
            begin
                if(Sample_Clock == 0)
                begin
                    State <= STATE_0;
                end
                else
                begin
                    State <= STATE_7;
                end
            end
        endcase
    end

cy_psoc3_count7 #(.cy_period(Decimation), .cy_route_ld(`FALSE), .cy_route_en(`TRUE))
Counter7Name (
/* input */ .clock (Clock), // Clock
/* input */ .reset(1'b0), // Reset
/* input */ .load(1'b0), // Load signal used if cy_route_ld = TRUE
/* input */ .enable(Decrement), // Enable signal used if cy_route_en = TRUE
/* output [6:0] */ .count(), // Counter value output
/* output */ .tc(TermCount) // Terminal Count output
);

//  reg [7:0] Decimator;    // counts the "done" signals, and when it reaches the count value, stores
the result in the FIFO
//                  if(Decimator == 0)
//                  begin
//                      Done <= 1;
//                      Decimator <= Decimation;
//                  end
//                  else
//                  begin
//                      Done <= 0;
//                      Decimator <= Decimator - 1'b1;
//                  end

//`#end` -- edit above this line, do not edit this line

//`#start footer` -- edit after this line, do not edit this line
cy_psoc3_dp16 #(.d0_init_a(Resolution), .d1_init_a({Enable_ABS == 0 ? 1'b0 : 1'b1,7'b1111111}), .d0_in
it_b(8'b00000000),
.d1_init_b(8'b00000000),
.cy_dpconfig_a(
{
    `CS_ALU_OP_PASS, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM0:     do nothing wait state, copy A0 into A1*/
    `CS_ALU_OP__ADD, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM1:     mod bit is high, increment A0 -> A0*/
    `CS_ALU_OP__SUB, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM2:     mod bit is low, decrement A0 -> A0*/
    `CS_ALU_OP_PASS, `CS_SRCA_A1, `CS_SRCB_D0,
    `CS_SHFT_OP___SR, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM3:     shift A1 right*/
    `CS_ALU_OP__SUB, `CS_SRCA_A0, `CS_SRCB_A1,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM4:     A0 - A1 ->A0*/
    Enable_Offset == 0 ? `CS_ALU_OP__XOR : `CS_ALU_OP__ADD, `CS_SRCA_A1, `CS_SRCB_D1,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM5:     Invert all bits in A1(but ! Msb) -> A1*/
    `CS_ALU_OP__INC, `CS_SRCA_A1, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM6:     add 1 to A1, store in A1*/
    `CS_ALU_OP_PASS, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM7:     do nothing wait state*/
    8'hFF, 8'h00,  /*CFG9:                        */
    8'hFF, 8'hFF,  /*CFG11-10:                        */
    `SC_CMPB_A1_D1, `SC_CMPA_A1_D1, `SC_CI_B_ARITH,
    `SC_CI_A_ARITH, `SC_C1_MASK_DSBL, `SC_C0_MASK_DSBL,
    `SC_A_MASK_DSBL, `SC_DEF_SI_0, `SC_SI_B_CHAIN,
    `SC_SI_A_CHAIN, /*CFG13-12:                        */
    `SC_A0_SRC_ACC, `SC_SHIFT_SL, 1'h0,
    1'h0, `SC_FIFO1_BUS, `SC_FIFO0__A1,
    `SC_MSB_DSBL, `SC_MSB_BIT0, `SC_MSB_NOCHN,
    `SC_FB_NOCHN, `SC_CMP1_NOCHN,
    `SC_CMP0_NOCHN, /*CFG15-14:                        */
    10'h00, `SC_FIFO_CLK__DP,`SC_FIFO_CAP_AX,
    `SC_FIFO__EDGE,`SC_FIFO__SYNC,`SC_EXTCRC_DSBL,
    `SC_WRK16CAT_DSBL /*CFG17-16:                        */
}
), .cy_dpconfig_b(
{
    `CS_ALU_OP_PASS, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM0:     do nothing wait state, copy A0 into A1*/
    `CS_ALU_OP__ADD, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM1:     mod bit is high, increment A0 -> A0*/
    `CS_ALU_OP__SUB, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM2:     mod bit is low, decrement A0 ->A0*/
    `CS_ALU_OP_PASS, `CS_SRCA_A1, `CS_SRCB_D0,
    `CS_SHFT_OP___SR, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM3:     shift A1 right*/
    `CS_ALU_OP__SUB, `CS_SRCA_A0, `CS_SRCB_A1,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM4:     A0 - A1 ->A0*/
    Enable_Offset == 0 ? `CS_ALU_OP__XOR : `CS_ALU_OP__ADD, `CS_SRCA_A1, `CS_SRCB_D1,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM5:     invert all bits in A1 -> A1*/
    `CS_ALU_OP__INC, `CS_SRCA_A1, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC__ALU,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM6:     increment A1 ->A1*/
    `CS_ALU_OP_PASS, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC_NONE, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_DSBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM7:     do nothing wait state*/
    8'hFF, 8'h00,  /*CFG9:                        */
    8'hFF, 8'hFF,  /*CFG11-10:                        */
    `SC_CMPB_A1_D1, `SC_CMPA_A1_D1, `SC_CI_B_CHAIN,
    `SC_CI_A_CHAIN, `SC_C1_MASK_DSBL, `SC_C0_MASK_DSBL,
    `SC_A_MASK_DSBL, `SC_DEF_SI_0, `SC_SI_B_DEFSI,
    `SC_SI_A_DEFSI, /*CFG13-12:                        */
    `SC_A0_SRC_ACC, `SC_SHIFT_SL, 1'h0,
    `SC_SR_SRC_MSB, `SC_FIFO1_BUS, `SC_FIFO0__A1,
    `SC_MSB_DSBL, `SC_MSB_BIT0, `SC_MSB_NOCHN,
    `SC_FB_NOCHN, `SC_CMP1_CHNED,
    `SC_CMP0_CHNED, /*CFG15-14:                        */
    10'h00, `SC_FIFO_CLK__DP,`SC_FIFO_CAP_AX,
    `SC_FIFO__EDGE,`SC_FIFO__SYNC,`SC_EXTCRC_DSBL,
    `SC_WRK16CAT_DSBL /*CFG17-16:                        */
}
)) UDB_IIR(
        /*  input                   */  .reset(1'b0),
        /*  input                   */  .clk(Clock),
        /*  input   [02:00]         */  .cs_addr(State[2:0]),
        /*  input                   */  .route_si(1'b0),
        /*  input                   */  .route_ci(1'b0),
        /*  input                   */  .f0_load(Done),
        /*  input                   */  .f1_load(1'b0),
        /*  input                   */  .d0_load(1'b0),
        /*  input                   */  .d1_load(1'b0),
        /*  output  [01:00]                  */  .ce0(),
        /*  output  [01:00]                  */  .cl0(),
        /*  output  [01:00]                   */  .z0(),
        /*  output  [01:00]                   */  .ff0(),
        /*  output  [01:00]                   */  .ce1(),
        /*  output  [01:00]                   */  .cl1(),
        /*  output  [01:00]                   */  .z1(),
        /*  output  [01:00]                   */  .ff1(),
        /*  output  [01:00]                   */  .ov_msb(),
        /*  output  [01:00]                   */  .co_msb(),
        /*  output  [01:00]                   */  .cmsb(),
        /*  output  [01:00]                   */  .so(temp),
        /*  output  [01:00]                   */  .f0_bus_stat(),
        /*  output  [01:00]                   */  .f0_blk_stat(),
        /*  output  [01:00]                   */  .f1_bus_stat(),
        /*  output  [01:00]                   */  .f1_blk_stat()
);
//`#end` -- edit above this line, do not edit this line
endmodule
```