CYPRESS SEMICONDUCTOR CORPORATION

Internal Correspondence

Date:	Monday 1/21/2013	**WW:**	**1304**
To:	**PSoC Apps**		
Author:	Chris Keeser (KEES)		
Author File#:	KEES#188		
Subject:	**UDB Based SAR DSI Output Offset Removal and 4 Input Mixer Component**		
Distribution:	PSoC Apps		

Summary:

This memo documents and distributes a UDB based offset correction and mixer component for use with the DSI output enabled SAR ADC (See KEES# 187), specifically for FSK demodulation (but not limited to FSK demodulation applications). This component corrects the offset binary output of the SAR ADC and converts the result into 2's compliment. The results are then multiplied by +1 or -1 depending on the 4 mixing inputs. These offset corrected and mixed results are then loaded into the FIFO in the following order: F0_IP, F0_QP, F1_IP, F1_QP. DMA or the CPU can then unload these values for further processing. The entire offset correction and mixing process requires 8 clock cycles, and DMA can unload the entire FIFO (all 4 results of 16 bit data) in 15 bus clock cycles with an 8 byte burst. The component takes advantage of the Dynamic Parallel input on the UDBs, and so this component will only work with Panther LP (or production Leopard if another data source is used), since Panther TM does not have the Dynamic Parallel input feature.

Attached File Summary:

File # 2 is the component exported with Creator 2.2's component export feature. To use the component, download the file and remove the .PDF extension.

File # 3 is an example project bundle using the SAR ADC with the parallel output and the OffsetMixer component displaying the data on an LCD. To use the file, download it and remove the .PDF extension.

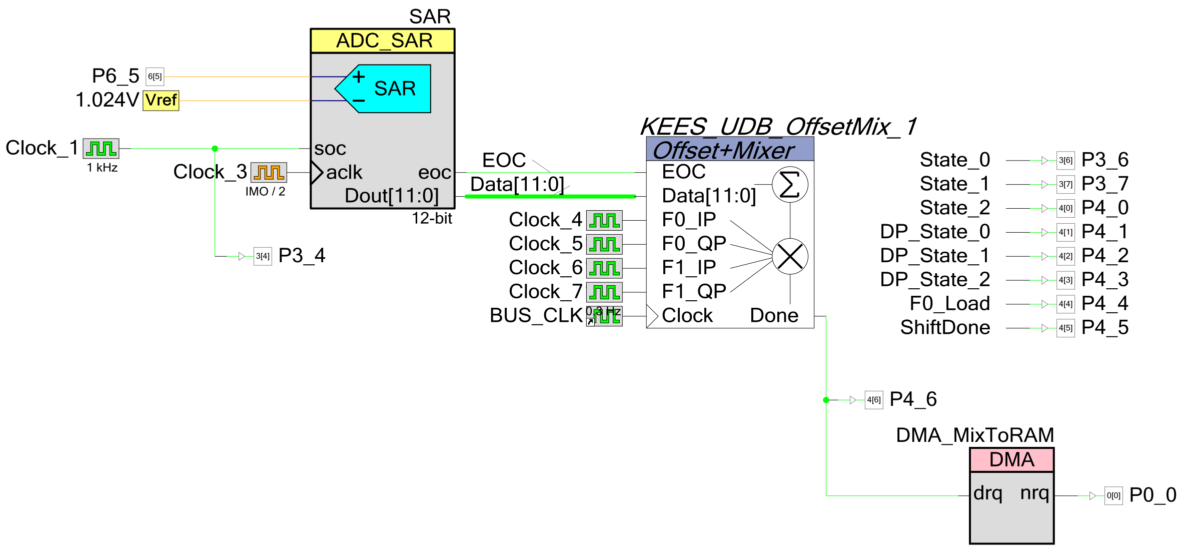
File # 4 is the scan of the UDB instructions used to generate the component.

Details:

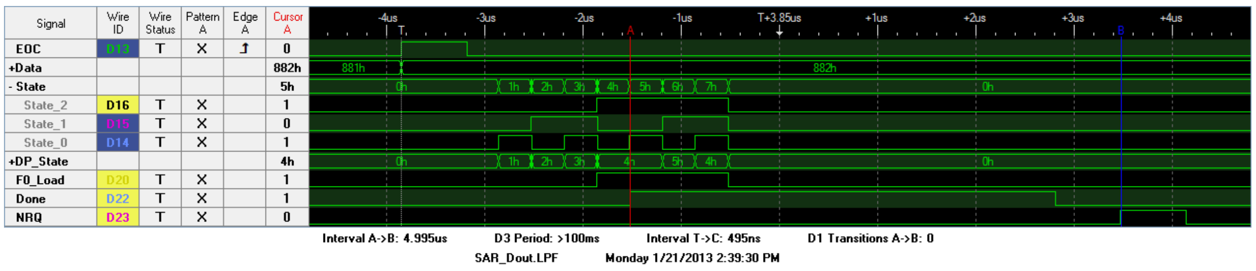
This component will be used to reduce the DMA requirements from our SAR+DMA+DFB FSK demodulator application from 12 + channels down to 1. The previous design made creative use of DMA, Status and Control registers and lookup tables in RAM to perform the offset correction and mixing. This implementation takes the digital data directly from the SAR using the DSI parallel output and performs the offset correction and 4 channel mixing, all within a 16 bit datapath. The 4 results are loaded into a FIFO in order, which allows the DMA to take advantage of bursting to reduce the bus clocks required to move the data. The modified SAR component and the UDB OffsetMixer component will be used in conjunction with a custom DFB program to perform FSK demodulation in hardware. The ‘Done’ signal is the FIFO “not empty” signal, allowing for flow control if necessary. The DFB may not be able to take all 4 samples as fast as the DMA can provide them.

The component has no API (none are required). The component includes a DMA capability file that greatly simplifies setting up DMA for the component.

Below is a schematic capture from the example project. The screen cap is showing the optional debug feature enable, that exposes internal signals from the component.



Here is a logic analyzer shot showing the state machine progression;

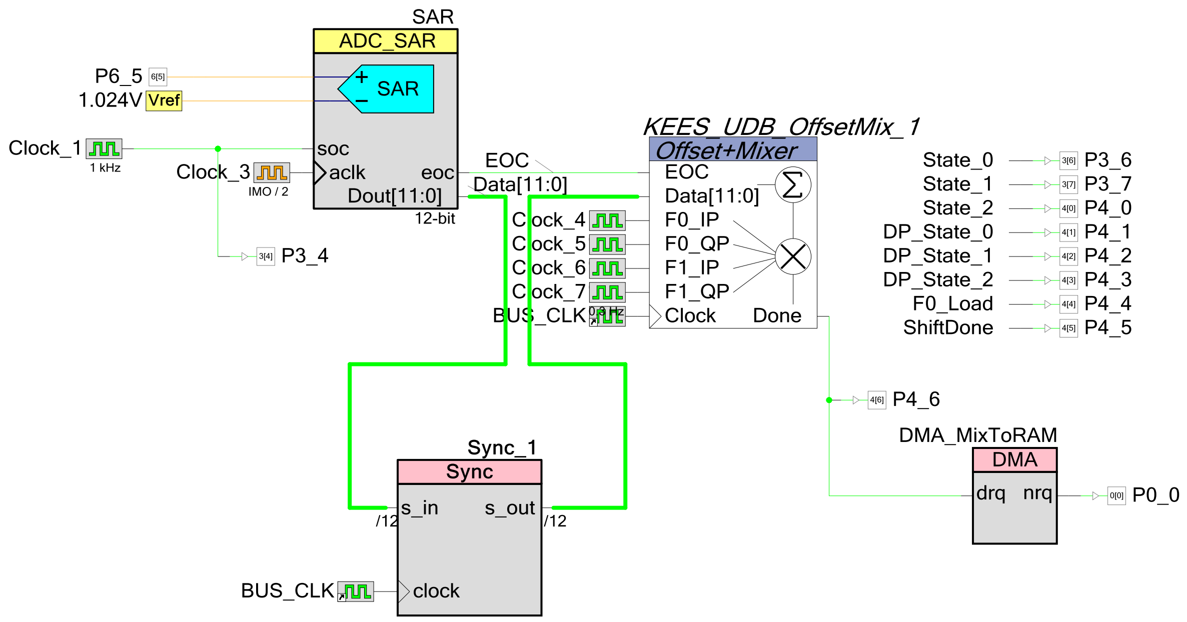


Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
EOC	D13	T	X	↑	0
+Data					882h
- State					5h
State_2	D16	T	X		1
State_1	D15	T	X		0
State_0	D14	T	X		1
+DP_State					4h
F0_Load	D20	T	X		1
Done	D22	T	X		1
NRQ	D23	T	X		0

Interval A->B: 4.995us   D3 Period: >100ms   Interval T->C: 495ns   D1 Transitions A->B: 0

SAR_Dout.LPF   Monday 1/21/2013 2:39:30 PM

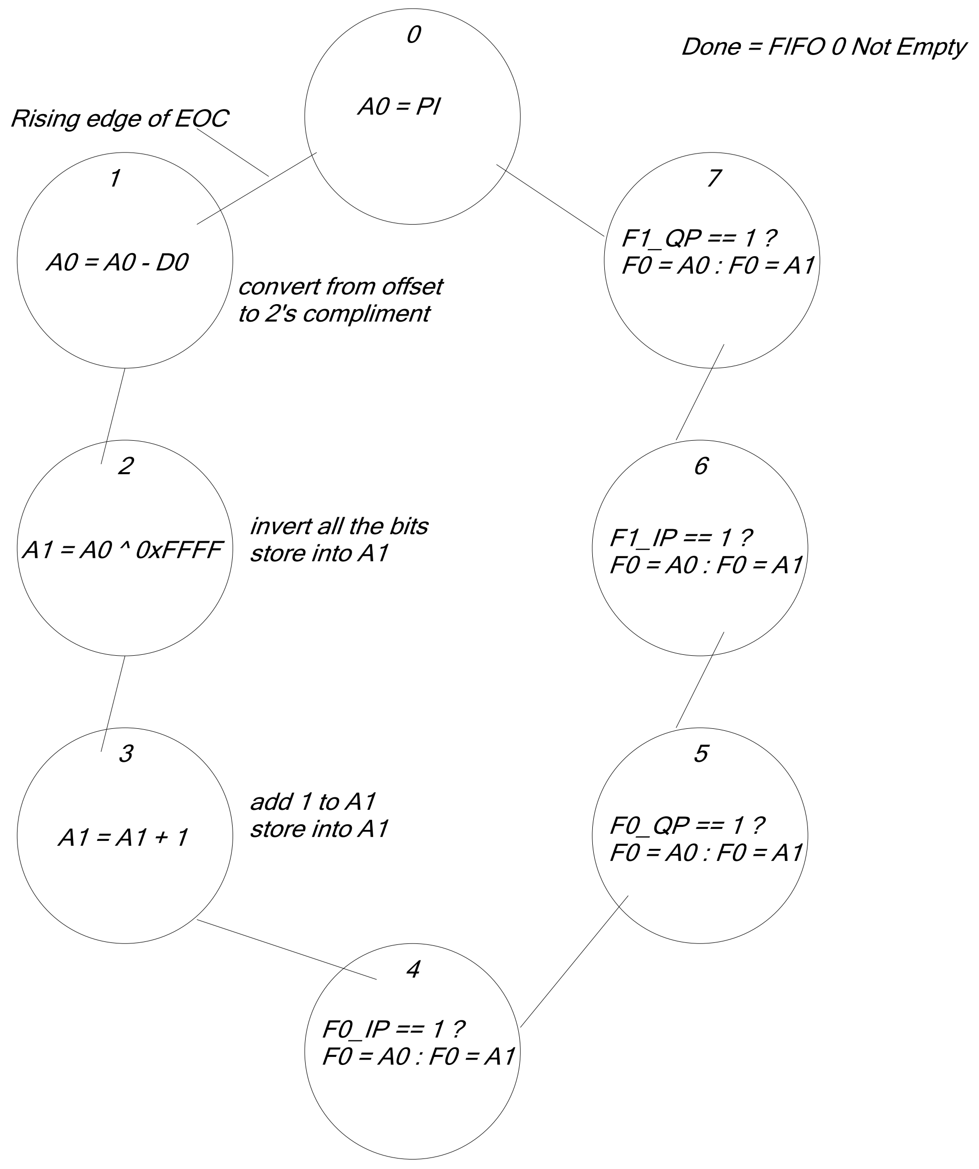
STA is not a problem in our application with low clock speeds (Bus clock @ 3 Mhz) but to eliminate STA warnings, a sync component can be placed in series with the data signals. The downside to this is the use of 3+ status registers since each status register can only synchronize 4 signals, and the DSI output of the SAR is 12 signals.



Datapath state machine (use in conjunction with the datapath instruction file attachment and the logic trace):

Side note, I had to construct my own 16 bit chained datapath because the 16 bit datapath in the datapath config tool does not enable parallel in or out. The template files for the 16, 24 and 32 bit datapaths are stored in : C:\Program Files (x86)\Cypress\PSoC Creator\x.x\PSoC Creator\warp\lib\sim\presynth\vlg\cy_psoc3.v

I had to use this file as guide when connecting the datapaths. The “a” datapath is LSB and the “b” datapath is the MSB (16 bit).


Done = FIFO 0 Not Empty
Rising edge of EOC
0
A0 = PI
1
A0 = A0 - D0
convert from offset
to 2's compliment
2
A1 = A0 ^ 0xFFFF
invert all the bits
store into A1
3
A1 = A1 + 1
add 1 to A1
store into A1
4
F0_IP == 1 ?
F0 = A0 : F0 = A1
5
F0_QP == 1 ?
F0 = A0 : F0 = A1
6
F1_IP == 1 ?
F0 = A0 : F0 = A1
7
F1_QP == 1 ?
F0 = A0 : F0 = A1

Appendix: Verilog code

```
//`#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 01/17/2013 at 15:33
// Component: KEES_UDB_OffsetMix_v1_00
module KEES_UDB_OffsetMix_v1_00 (
    output  Done,
    output  reg F0_Load,
    output  ShiftDone,
    output  State_0,
    output  State_1,
    output  State_2,
    output  DP_State_0,
    output  DP_State_1,
    output  DP_State_2,
    input   Clock,
    input  [11:0] Data,
    input   EOC,
    input   F0_IP,
    input   F0_QP,
    input   F1_IP,
    input   F1_QP
);

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

//parameter LeftJustify = 1'b0;

// chaining signals
wire    carry, sh_right, sh_left, msb, cfb;
wire    [01:00] cmp_eq, cmp_lt, cmp_zero, cmp_ff, cap;
// end chaning signals

localparam  STATE_0 = 3'h0;
localparam  STATE_1 = 3'h1;
localparam  STATE_2 = 3'h2;
localparam  STATE_3 = 3'h3;
localparam  STATE_4 = 3'h4;
localparam  STATE_5 = 3'h5;
localparam  STATE_6 = 3'h6;
localparam  STATE_7 = 3'h7;

reg [2:0] DP_State;
reg [2:0] State;
reg EOC_delayed;
reg F0_ip_reg;
reg F0_qp_reg;
reg F1_ip_reg;
reg F1_qp_reg;

wire EOC_edgedet;
wire EOC_synced;


assign State_0 = State[0];
assign State_1 = State[1];
assign State_2 = State[2];
assign DP_State_0 = DP_State[0];
assign DP_State_1 = DP_State[1];
assign DP_State_2 = DP_State[2];

assign EOC_edgedet = ((EOC_synced == 1) && (EOC_delayed == 0)) ? 1'b1 : 1'b0;
assign ShiftDone = 1'b0;

always @(posedge Clock)
begin

    EOC_delayed <= EOC_synced;
    DP_State <= STATE_0;
    State <= STATE_0;
    F0_Load <= 0;
    F0_ip_reg <= F0_ip_reg;
    F0_qp_reg <= F0_qp_reg;
    F1_ip_reg <= F1_ip_reg;
    F1_qp_reg <= F1_qp_reg;

    case(State)

        STATE_0:
        begin
            F0_ip_reg <= F0_IP;
            F0_qp_reg <= F0_QP;
            F1_ip_reg <= F1_IP;
            F1_qp_reg <= F1_QP;

            if(EOC_edgedet == 1)
            begin
                DP_State <= STATE_1;
                State <= STATE_1;
            end
            else
            begin
                DP_State <= STATE_0;
                State <= STATE_0;
            end
        end

        STATE_1:  // possibly a shift state, for now its just a skipped state
        begin
            DP_State <= STATE_2;
            State <= STATE_2;
        end

        STATE_2:
        begin
            DP_State <= STATE_3;
            State <= STATE_3;
        end

        STATE_3:
        begin
            if(F0_ip_reg == 1)
            begin
                DP_State <= STATE_4;
            end
            else
            begin
                DP_State <= STATE_5;
            end
            F0_Load <= 1;

            State <= STATE_4;
        end

        STATE_4:
        begin
            if(F0_qp_reg == 1)
            begin
                DP_State <= STATE_4;
            end
            else
            begin
                DP_State <= STATE_5;
            end
            F0_Load <= 1;

            State <= STATE_5;
        end

        STATE_5:
        begin
            if(F1_ip_reg == 1)
            begin
                DP_State <= STATE_4;
            end
            else
            begin
                DP_State <= STATE_5;
            end
            F0_Load <= 1;

            State <= STATE_6;
        end

        STATE_6:
        begin
```

```
            if(F1_qp_reg == 1)
            begin
                DP_State <= STATE_4;
            end
            else
            begin
                DP_State <= STATE_5;
            end
            F0_Load <= 1;

            State <= STATE_7;
        end

        STATE_7:
        begin
            DP_State <= STATE_0;
            State <= STATE_0;
        end

    endcase
end

// EOC sync component
cy_psoc3_sync EOC_Sync (
.clock (Clock),
.sc_in (EOC),
.sc_out (EOC_synced));

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

cy_psoc3_dp #(.d0_init(8'h00), .d1_init(8'hFF),
.cy_dpconfig(
{
    `CS_ALU_OP_PASS, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_ENBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM0:  A0=PI*/
    `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, /*CFGRAM1:  A0=A0-D0*/
    `CS_ALU_OP__XOR, `CS_SRCA_A0, `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, /*CFGRAM2:  A1=A0^D1*/
    `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, /*CFGRAM3:  A1=A1+1*/
    `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, /*CFGRAM4:  F0=A0*/
    `CS_ALU_OP_PASS, `CS_SRCA_A1, `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, /*CFGRAM5:  F0=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, /*CFGRAM6:         */
    `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:         */
    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, `SC_PI_DYN_EN,
    1'h0, `SC_FIFO1_ALU, `SC_FIFO0_ALU,
    `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_LEVEL,`SC_FIFO__SYNC,`SC_EXTCRC_DSBL,
    `SC_WRK16CAT_DSBL /*CFG17-16:         */
}
)) OffsetMixer_LSB(
        /*  input                   */  .reset(1'b0),
        /*  input                   */  .clk(Clock),
        /*  input   [02:00]         */  .cs_addr(DP_State),
        /*  input                   */  .route_si(1'b0),
        /*  input                   */  .route_ci(1'b0),
        /*  input                   */  .f0_load(F0_Load),
        /*  input                   */  .f1_load(1'b0),
        /*  input                   */  .d0_load(1'b0),
        /*  input                   */  .d1_load(1'b0),
        /*  output                  */  .ce0(),
        /*  output                  */  .cl0(),
        /*  output                  */  .z0(),
        /*  output                  */  .ff0(),
        /*  output                  */  .ce1(),
        /*  output                  */  .cl1(),
        /*  output                  */  .z1(),
        /*  output                  */  .ff1(),
        /*  output                  */  .ov_msb(),
        /*  output                  */  .co_msb(),
        /*  output                  */  .cmsb(),
        /*  output                  */  .so(),
        /*  output                  */  .f0_bus_stat(Done),
        /*  output                  */  .f0_blk_stat(),
        /*  output                  */  .f1_bus_stat(),
        /*  output                  */  .f1_blk_stat(),

        /* input                    */  .ci(1'b0),     // Carry in from previous stage
        /* output                   */  .co(carry),         // Carry out to next stage
        /* input                    */  .sir(1'b0),    // Shift in from right side
        /* output                   */  .sor(),        // Shift out to right side
        /* input                    */  .sil(sh_right),    // Shift in from left side
        /* output                   */  .sol(sh_left),        // Shift out to left side
        /* input                    */  .msbi(msb),   // MSB chain in
        /* output                   */  .msbo(),       // MSB chain out
        /* input [01:00]            */  .cei(2'b0),    // Compare equal in from prev stage
        /* output [01:00]           */  .ceo(cmp_eq),        // Compare equal out to next stage
        /* input [01:00]            */  .cli(2'b0),    // Compare less than in from prv stage
        /* output [01:00]           */  .clo(cmp_lt),        // Compare less than out to next stage
        /* input [01:00]            */  .zi(2'b0),     // Zero detect in from previous stage
        /* output [01:00]           */  .zo(cmp_zero),         // Zero detect out to next stage
        /* input [01:00]            */  .fi(2'b0),     // 0xFF detect in from previous stage
        /* output [01:00]           */  .fo(cmp_ff),        // 0xFF detect out to next stage
        /* input [01:00]            */  .capi(2'b0),   // Software capture from previous stage
        /* output [01:00]           */  .capo(cap),       // Software capture to next stage
        /* input                    */  .cfbi(1'b0),   // CRC Feedback in from previous stage
        /* output                   */  .cfbo(cfb),       // CRC Feedback out to next stage
        /* input [07:00]            */  .pi(Data[7:0]),     // Parallel data port  LeftJustify == 1 ?
{Data[3:0], 4'h0} : {Data[7:0]}
        /* output [07:00]           */  .po()          // Parallel data port
);
cy_psoc3_dp #(.d0_init(8'h08), .d1_init(8'hFF),
//LeftJustify == 1 ? {8'h80} : {8'h08}
.cy_dpconfig(
{
    `CS_ALU_OP_PASS, `CS_SRCA_A0, `CS_SRCB_D0,
    `CS_SHFT_OP_PASS, `CS_A0_SRC__ALU, `CS_A1_SRC_NONE,
    `CS_FEEDBACK_ENBL, `CS_CI_SEL_CFGA, `CS_SI_SEL_CFGA,
    `CS_CMP_SEL_CFGA, /*CFGRAM0:  A0=PI*/
    `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, /*CFGRAM1:  A0=A0-D0*/
    `CS_ALU_OP__XOR, `CS_SRCA_A0, `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, /*CFGRAM2:  A1=A0^D1*/
    `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, /*CFGRAM3:  A1=A1+1*/
    `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, /*CFGRAM4:  F0=A0*/
    `CS_ALU_OP_PASS, `CS_SRCA_A1, `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, /*CFGRAM5:  F1=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, /*CFGRAM6:        */
    `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:        */
    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, `SC_PI_DYN_EN,
    `SC_SR_SRC_MSB, `SC_FIFO1_ALU, `SC_FIFO0_ALU,
    `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_LEVEL,`SC_FIFO__SYNC,`SC_EXTCRC_DSBL,
    `SC_WRK16CAT_DSBL /*CFG17-16:          */
}
)) OffsetMixer_MSB(
        /*  input                   */  .reset(1'b0),
        /*  input                   */  .clk(Clock),
        /*  input   [02:00]         */  .cs_addr(DP_State),
        /*  input                   */  .route_si(1'b0),
        /*  input                   */  .route_ci(1'b0),
        /*  input                   */  .f0_load(F0_Load),
        /*  input                   */  .f1_load(1'b0),
        /*  input                   */  .d0_load(1'b0),
        /*  input                   */  .d1_load(1'b0),
        /*  output                  */  .ce0(),
        /*  output                  */  .cl0(),
        /*  output                  */  .z0(),
        /*  output                  */  .ff0(),
        /*  output                  */  .ce1(),
        /*  output                  */  .cl1(),
        /*  output                  */  .z1(),
        /*  output                  */  .ff1(),
        /*  output                  */  .ov_msb(),
        /*  output                  */  .co_msb(),
        /*  output                  */  .cmsb(),
        /*  output                  */  .so(),
        /*  output                  */  .f0_bus_stat(),
        /*  output                  */  .f0_blk_stat(),
        /*  output                  */  .f1_bus_stat(),
        /*  output                  */  .f1_blk_stat(),

        /* input                    */  .ci(carry),      // Carry in from previous stage
        /* output                   */  .co(),          // Carry out to next stage
        /* input                    */  .sir(sh_left),    // Shift in from right side
        /* output                   */  .sor(sh_right),        // Shift out to right side
        /* input                    */  .sil(1'b0),     // Shift in from left side
        /* output                   */  .sol(),         // Shift out to left side
        /* input                    */  .msbi(1'b0),    // MSB chain in
        /* output                   */  .msbo(msb),        // MSB chain out
        /* input [01:00]            */  .cei(cmp_eq),     // Compare equal in from prev stage
        /* output [01:00]           */  .ceo(),         // Compare equal out to next stage
        /* input [01:00]            */  .cli(cmp_lt),     // Compare less than in from prv stage
        /* output [01:00]           */  .clo(),         // Compare less than out to next stage
        /* input [01:00]            */  .zi(cmp_zero),      // Zero detect in from previous stage
        /* output [01:00]           */  .zo(),          // Zero detect out to next stage
        /* input [01:00]            */  .fi(cmp_ff),      // 0xFF detect in from previous stage
        /* output [01:00]           */  .fo(),          // 0xFF detect out to next stage
        /* input [01:00]            */  .capi(cap),    // Software capture from previous stage
        /* output [01:00]           */  .capo(),        // Software capture to next stage
        /* input                    */  .cfbi(cfb),    // CRC Feedback in from previous stage
        /* output                   */  .cfbo(),        // CRC Feedback out to next stage
        /* input [07:00]            */  .pi({4'h0,Data[11:8]}),     // Parallel data port  LeftJustify
== 1 ? {Data[11:4]} : {4'h0,Data[11:8]}
        /* output [07:00]           */  .po()           // Parallel data port
);
endmodule
//`#start footer` -- edit after this line, do not edit this line
//`#end` -- edit above this line, do not edit this line
```