Cogging Test Switch Module

Plus BDOT Interval Time Measurement

# Craig Drennan

# June 23, 2016

The Cogging Test Switch Module currently has two functions. The first is used for testing an alternate Cogging controller during Booster operation. The module supports switching between Primary and Alternate sets of 4 digital output signals and a Primary and Alternate analog signal (+/- 5 Volts). ***This first function is active when a 50 Ohm terminator is installed at input SW2.***

When a Booster event trigger pulse is seen at the “Alternate I/O Select” input of the module (SW1), the 4 alternate digital signals and the alternate analog signal are switched onto the outputs of the module for 40 milli-seconds. Otherwise, the primary 4 digital signals and the primary analog signal appear at the outputs. Figure 1 indicates the connections on the front and rear of the module.

The module is a modified Dual Phase Detector Module Version 2 (phasedet2). Figures 2 to 5 show the circuit modifications. Beams-doc-3922-v2 can be referred to for the other details of the module.

The analog signals through the module can be calibrated with gain and offset adjustment set using the USB – PC interface (DLP). A screenshot of this interface is shown in Figure 6.

The second function is to provide an analog output that represents the deviation from the nominal time interval between the BDOT trigger pulses from the GMPS system. This function is active when the SW2 input is left open or set to a logic High voltage ( >2.8 V ). The input SW2 is pulled High internally for convenience. This second function only requires a BDOT trigger pulse on input T4. The analog output can be monitored on the rear of the module at AOUT 1.

The BDOT interval measurement is made by counting the number of 10 MHz clocks that occur between BDOT pulses. This count is subtracted from the nominal count (666,666) and the difference is scale by 32 times and output onto a 16 bit +/- 10V DAC.

The FPGA logic that controls these functions is shown in Figure 7 and Listing 1.

“A pictures worth a thousand words”, Arthur Brisbane.

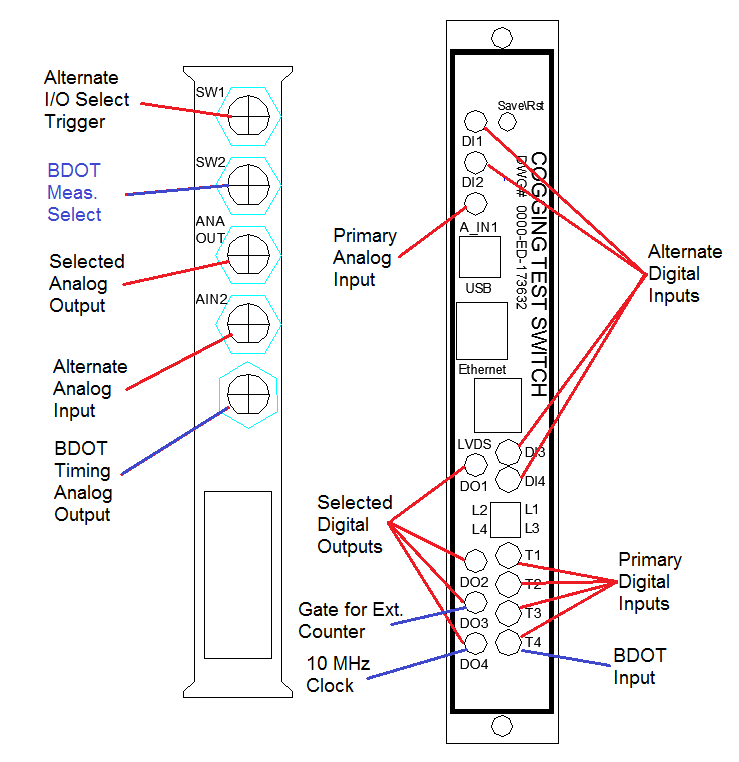


Figure 1.

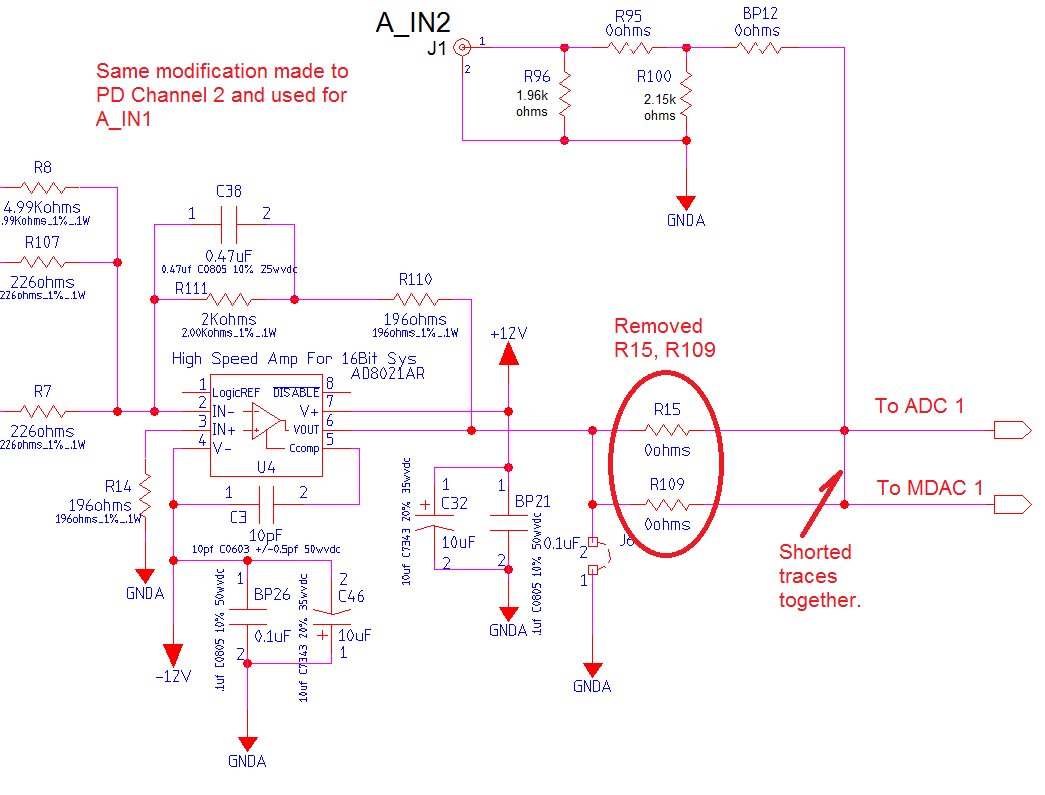


Figure 2

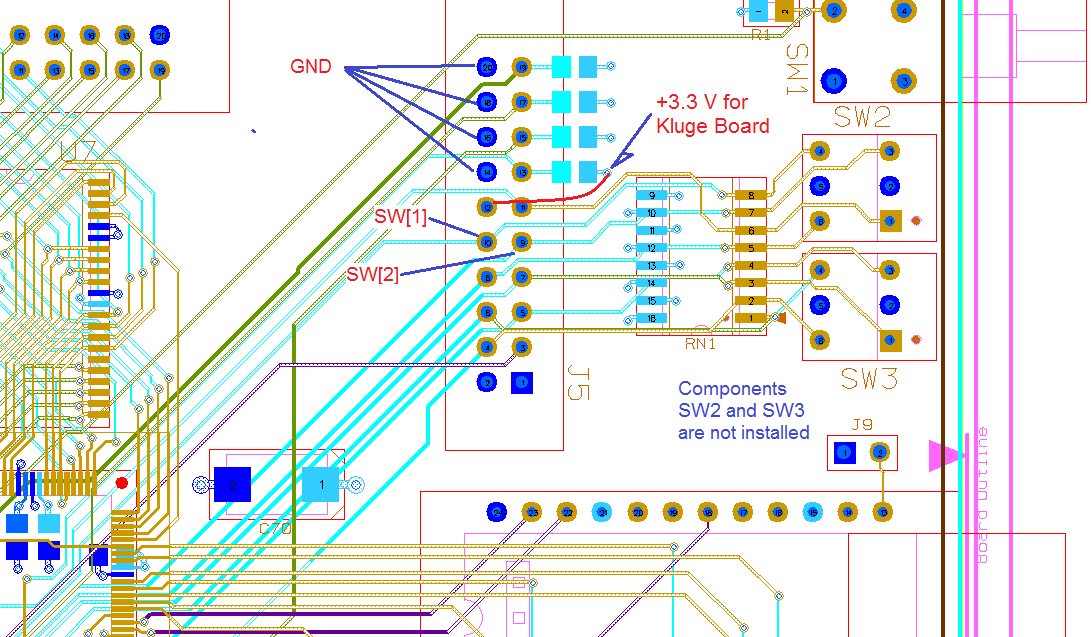


Figure 3

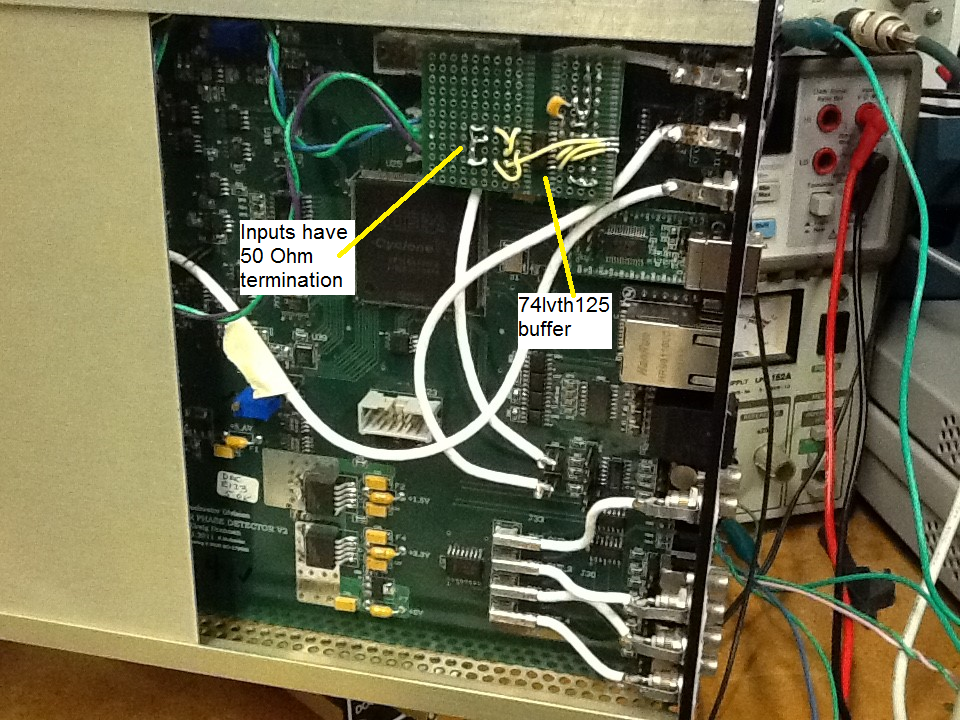


Figure 4

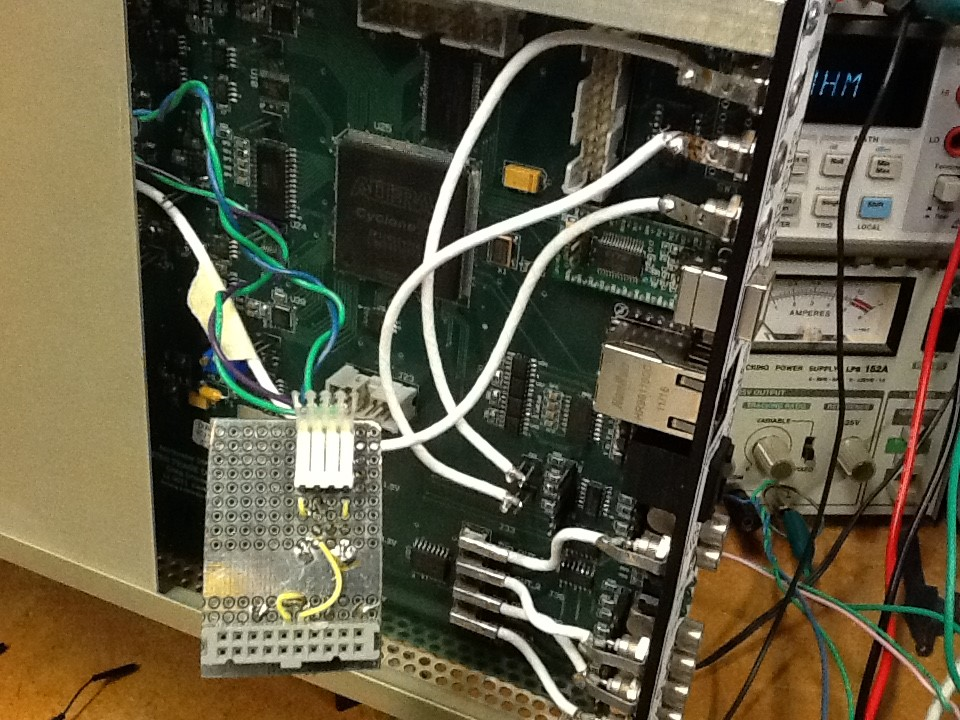


Figure 5

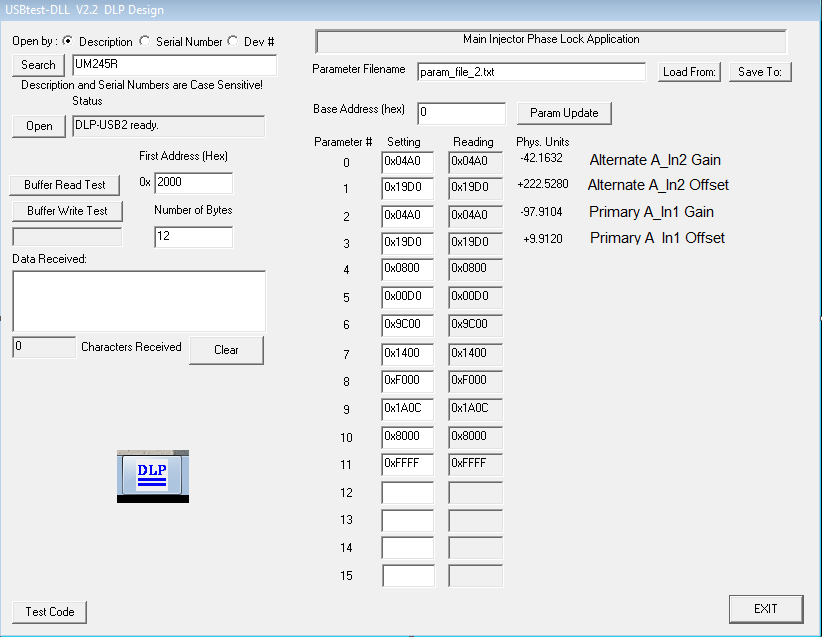


Figure 6.

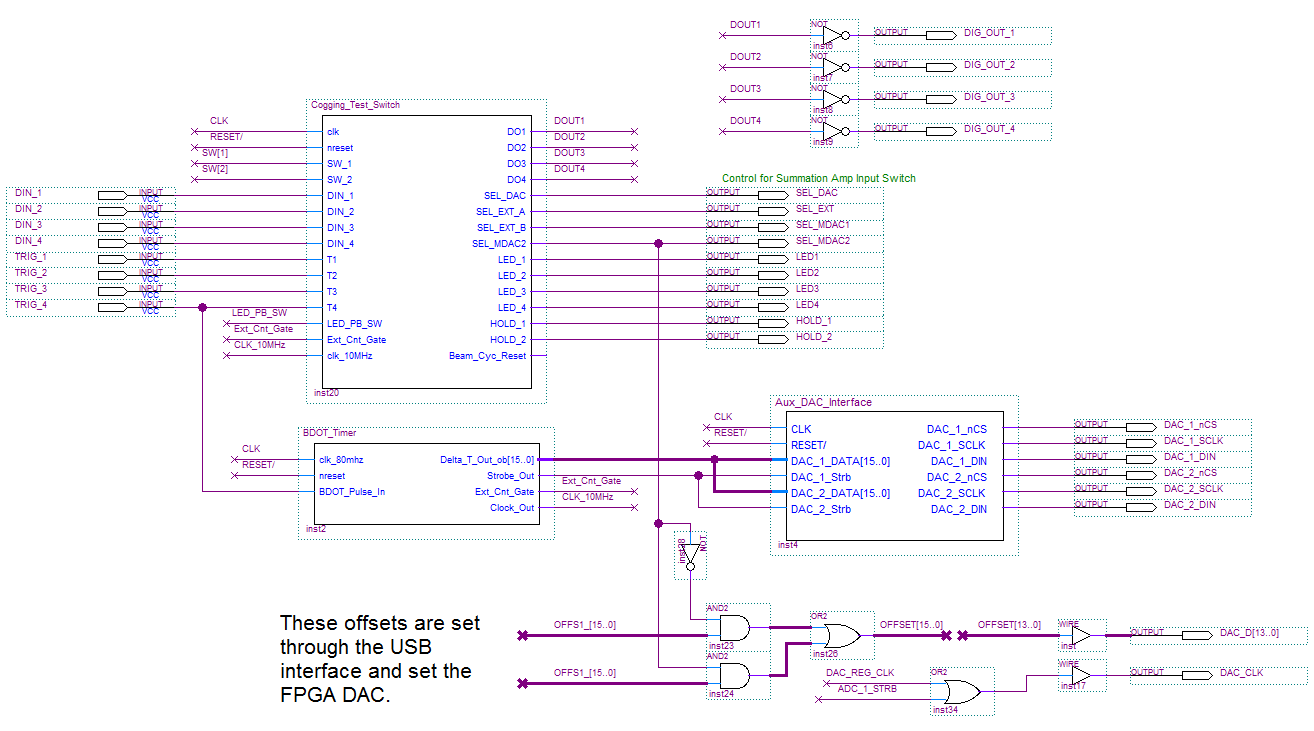


Figure 7

# Listing 1

Library IEEE;

USE IEEE.std\_logic\_1164.all;

USE IEEE.std\_logic\_arith.all;

USE ieee.std\_logic\_unsigned.all;

-- My Definitions --------------------------------------------------

USE work.pkg\_081006.all; -- my definitions and entities

--Version 4 breaks the EXTERNAL / MADC1 Switch logic out from the

-- Accel\_PL\_Ena and Start / Stop logic.

entity Cogging\_Test\_Switch is

port(clk : in std\_logic;

nreset : in std\_logic;

SW\_1 : in std\_logic; -- momentary trigger runs mode for 40ms

SW\_2 : in std\_logic; -- A High logic level sets BDOT timer mode

-- until returned to a Low logic level

DIN\_1 : in std\_logic;

DIN\_2 : in std\_logic;

DIN\_3 : in std\_logic;

DIN\_4 : in std\_logic;

T1 : in std\_logic;

T2 : in std\_logic;

T3 : in std\_logic;

T4 : in std\_logic;

LED\_PB\_SW : in std\_logic;

Ext\_Cnt\_Gate : in std\_logic;

clk\_10MHz : in std\_logic;

DO1 : out std\_logic;

DO2 : out std\_logic;

DO3 : out std\_logic;

DO4 : out std\_logic;

SEL\_DAC : out std\_logic;

SEL\_EXT\_A : out std\_logic;

SEL\_EXT\_B : out std\_logic;

SEL\_MDAC2 : out std\_logic;

LED\_1 : out std\_logic;

LED\_2 : out std\_logic;

LED\_3 : out std\_logic;

LED\_4 : out std\_logic;

HOLD\_1 : out std\_logic;

HOLD\_2 : out std\_logic;

Beam\_Cyc\_Reset : out std\_logic

);

end Cogging\_Test\_Switch;

architecture main of Cogging\_Test\_Switch is

signal sel\_alternate : std\_logic;

signal sel\_bdot\_timer\_mode : std\_logic;

signal timer : std\_logic\_vector(23 downto 0);

signal timer\_ena : std\_logic;

signal alt\_IO\_mode : std\_logic;

signal cycle\_reset : std\_logic;

-- State Machine declarations ------------------------------------

type STATE\_TYPE is

(idle, switched, send\_reset, bdot\_timer\_mode);

signal CS, NS : STATE\_TYPE;

-- ## MAIN ##############################################################

begin

EDGE\_ON\_TEST\_SELECT\_TRIGGER: led

port map(sig\_in => SW\_1, clk => clk, clrn => nreset, pls\_out => sel\_alternate);

SELECT\_BDOT\_TIMER\_MODE: process(clk, nreset, SW\_2)

begin

if (nreset = '0') then sel\_bdot\_timer\_mode <= '0';

elsif rising\_edge(clk) then

if (SW\_2 = '1') then sel\_bdot\_timer\_mode <= '1';

else sel\_bdot\_timer\_mode <= '0';

end if;

end if;

end process;

PROCESS\_TIMER: process(clk, nreset, timer\_ena, timer)

begin

if (nreset = '0') then timer <= (others => '0');

elsif rising\_edge(clk) then

if (timer\_ena = '0') then timer <= (others => '0');

else timer <= timer + 1;

end if;

end if;

end process;

----------------------------------------------------------

sm\_reg: process (clk, nreset)

begin

if(nreset='0') then CS <= idle;

elsif(clk'event and clk='1')then CS <= NS;

end if;

end process; -- sm\_reg

sm\_combinatorial:

process (CS, timer, sel\_alternate, DIN\_1, DIN\_2, DIN\_3, DIN\_4,

T1, T2, T3, T4,

sel\_bdot\_timer\_mode, Ext\_Cnt\_Gate, clk\_10MHz)

begin

-- default outputs --

timer\_ena <= '1'; -- not enabled only when in idle state

cycle\_reset <= '0';

DO1 <= T1; -- Normal Digital Inputs to Outputs

DO2 <= T2;

DO3 <= T3;

DO4 <= T4;

case CS is

when idle =>

timer\_ena <= '0';

if (sel\_alternate = '1') then NS <= switched;

elsif(sel\_bdot\_timer\_mode = '1') then NS <= bdot\_timer\_mode;

else NS <= idle;

end if;

when switched =>

if (timer = X"30D400") then NS <= send\_reset; -- time out at 40 ms

else NS <= switched;

end if;

DO1 <= DIN\_1;

DO2 <= DIN\_2;

DO3 <= DIN\_3;

DO4 <= DIN\_4;

when send\_reset => NS <= idle;

cycle\_reset <= '1';

when bdot\_timer\_mode =>

timer\_ena <= '0';

DO3 <= Ext\_Cnt\_Gate;

DO4 <= clk\_10MHz;

if (sel\_bdot\_timer\_mode = '1') then NS <= bdot\_timer\_mode;

else NS <= idle;

end if;

when others => NS <= idle;

end case;

end process;

----------------------------------------------------------

SWITCH\_CONTROL: process(clk, nreset, sel\_alternate, cycle\_reset)

begin -- THIS ACTS AS AN SR-FLIP FLOP

if (nreset = '0') then alt\_IO\_mode <= '0';

elsif rising\_edge(clk) then

if (sel\_alternate = '1') then alt\_IO\_mode <= '1';

elsif(cycle\_reset = '1') then alt\_IO\_mode <= '0';

end if;

end if;

end process;

Beam\_Cyc\_Reset <= cycle\_reset;

SEL\_DAC <= '1';

SEL\_EXT\_A <= '0';

SEL\_EXT\_B <= alt\_IO\_mode;

SEL\_MDAC2 <= not alt\_IO\_mode;

LED\_1 <= SW\_1;

LED\_2 <= LED\_PB\_SW;

LED\_3 <= not alt\_IO\_mode;

LED\_4 <= SW\_2;

HOLD\_1 <= '0';

HOLD\_2 <= '0';

end main;

Library IEEE;

USE IEEE.std\_logic\_1164.all;

USE IEEE.std\_logic\_arith.all;

USE ieee.std\_logic\_unsigned.all;

-- My Definitions --------------------------------------------------

USE work.pkg\_081006.all; -- my definitions and entities

--This Module counts the number 10 MHz clocks between BDOT inputs.

-- It then subtracts the count from the nominal count (666,666) and

-- puts the difference out as the top ten bits of a +/- 15 bit DAC

entity BDOT\_Timer is

port(

clk\_80mhz : in std\_logic;

nreset : in std\_logic;

BDOT\_Pulse\_In : in std\_logic;

Delta\_T\_Out\_ob : out std\_logic\_vector(15 downto 0);

Strobe\_Out : out std\_logic;

Ext\_Cnt\_Gate : out std\_logic;

Clock\_Out : out std\_logic

);

end BDOT\_Timer;

architecture main of BDOT\_Timer is

signal cnt\_A : std\_logic\_vector(31 downto 0);

signal cnt\_A\_ena : std\_logic;

signal cnt\_A\_rst : std\_logic;

signal cnt\_B : std\_logic\_vector(31 downto 0);

signal cnt\_B\_ena : std\_logic;

signal cnt\_B\_rst : std\_logic;

signal bdot : std\_logic;

signal get\_cnt\_A : std\_logic;

signal get\_cnt\_B : std\_logic;

signal latched\_cnt : std\_logic\_vector(31 downto 0);

signal delta\_cnt\_2s : std\_logic\_vector(31 downto 0);

signal delta\_cnt\_ob : std\_logic\_vector(31 downto 0);

signal delta\_out\_ob : std\_logic\_vector(31 downto 0);

signal clk\_div : std\_logic\_vector(2 downto 0);

signal clk : std\_logic;

component addsub32signed

PORT

(

add\_sub : IN STD\_LOGIC ;

dataa : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0);

datab : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0);

result : OUT STD\_LOGIC\_VECTOR (31 DOWNTO 0)

);

end component;

-- State Machine declarations ------------------------------------

type STATE\_TYPE is

(count\_on\_a, latch\_cnt\_a, reset\_cnt\_a,

count\_on\_b, latch\_cnt\_b, reset\_cnt\_b );

signal CS, NS : STATE\_TYPE;

-- ## MAIN ##############################################################

begin

EDGE\_ON\_BDOT\_PULSE\_IN: led

port map(sig\_in => BDOT\_Pulse\_In, clk => clk, clrn => nreset, pls\_out => bdot);

DIVIDE\_CLOCK: process(clk\_80mhz, nreset, clk\_div)

begin

if (nreset = '0') then clk\_div <= (others => '0');

elsif rising\_edge(clk\_80mhz) then clk\_div <= clk\_div + 1;

end if;

end process;

-----------------------------------------------------

clk <= clk\_div(2); -- 80 MHz divided by 8 = 10 MHz

Clock\_Out <= clk;

-----------------------------------------------------

PROCESS\_TIMER\_A: process(clk, nreset, cnt\_A\_ena, cnt\_A\_rst, cnt\_A)

begin

if (nreset = '0') then cnt\_A <= (others => '0');

elsif rising\_edge(clk) then

if (cnt\_A\_ena = '1') then cnt\_A <= cnt\_A + 1;

elsif(cnt\_A\_rst = '1') then cnt\_A <= (others => '0');

end if;

end if;

end process;

PROCESS\_TIMER\_B: process(clk, nreset, cnt\_B\_ena, cnt\_B\_rst, cnt\_B)

begin

if (nreset = '0') then cnt\_B <= (others => '0');

elsif rising\_edge(clk) then

if (cnt\_B\_ena = '1') then cnt\_B <= cnt\_B + 1;

elsif(cnt\_B\_rst = '1') then cnt\_B <= (others => '0');

end if;

end if;

end process;

DAC\_COUNT\_LATCH: process(clk, nreset, get\_cnt\_A, get\_cnt\_B, cnt\_A, cnt\_B)

begin

if (nreset = '0') then latched\_cnt <= (others => '0');

elsif rising\_edge(clk) then

if (get\_cnt\_A = '1') then latched\_cnt <= cnt\_A;

elsif(get\_cnt\_B = '1') then latched\_cnt <= cnt\_B;

end if;

end if;

end process;

----------------------------------------------------------

COMPUTE\_DELTA: addsub32signed

PORT MAP (

add\_sub => '0',

dataa => X"000A2C2A", -- 666666

datab => latched\_cnt,

result => delta\_cnt\_2s

);

delta\_cnt\_ob(31 downto 0) <= (not delta\_cnt\_2s(31)) & delta\_cnt\_2s(30 downto 0);

LIMIT\_DELTA\_TO\_10\_BITS:

process(delta\_cnt\_ob)

begin

if (delta\_cnt\_ob > X"800003FF") then delta\_out\_ob <= X"800003FF";

elsif(delta\_cnt\_ob < X"7FFFFC01") then delta\_out\_ob <= X"7FFFFC01";

else delta\_out\_ob <= delta\_cnt\_ob;

end if;

end process;

----------------------------------------------------------------------------------------------

Delta\_T\_Out\_ob(15 downto 0) <= (not delta\_out\_ob(10)) & delta\_out\_ob(9 downto 0) & "00000";

Ext\_Cnt\_Gate <= cnt\_A\_ena;

----------------------------------------------------------

sm\_reg: process (clk, nreset)

begin

if(nreset='0') then CS <= count\_on\_a;

elsif(clk'event and clk='1')then CS <= NS;

end if;

end process; -- sm\_reg

sm\_combinatorial:

process (CS, bdot)

begin

-- default outputs --

cnt\_A\_ena <= '0';

cnt\_A\_rst <= '0';

cnt\_B\_ena <= '0';

cnt\_B\_rst <= '0';

get\_cnt\_A <= '0';

get\_cnt\_B <= '0';

Strobe\_Out <= '0';

case CS is

when count\_on\_a =>

cnt\_A\_ena <= '1';

if (bdot = '1') then NS <= latch\_cnt\_a;

else NS <= count\_on\_a;

end if;

when latch\_cnt\_a => NS <= reset\_cnt\_a;

cnt\_B\_ena <= '1';

get\_cnt\_A <= '1';

when reset\_cnt\_a => NS <= count\_on\_b;

cnt\_B\_ena <= '1';

cnt\_A\_rst <= '1';

Strobe\_Out <= '1';

when count\_on\_b =>

cnt\_B\_ena <= '1';

if (bdot = '1') then NS <= latch\_cnt\_b;

else NS <= count\_on\_b;

end if;

when latch\_cnt\_b => NS <= reset\_cnt\_b;

cnt\_A\_ena <= '1';

get\_cnt\_B <= '1';

when reset\_cnt\_b => NS <= count\_on\_a;

cnt\_A\_ena <= '1';

cnt\_B\_rst <= '1';

Strobe\_Out <= '1';

when others => NS <= count\_on\_a;

end case;

end process;

end main;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Listing 2BDOT Interval Measurement | | |  |  |  |  |  |  |  |
| 23-Jun-16 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Signal Generator, ms | Delta ms | External  Counter | Delta  Count | DAC  AOUT 1, V |  |  |  |  |  |
| **Column1** | **Column2** | **Column3** | **Column4** | **Column5** |  |  |  |  |  |
| 66.66667 | 0.00117 | 666655 | 11 | 0.161 |  | Linest | delta\_cnt = m Volts + b | | |
| 66.66767 | 0.00017 | 666665 | 1 | 0.0635 |  | m | b |  |  |
| 66.66867 | -0.00083 | 666675 | -9 | -0.034 |  | 102.3428 | -5.74425 |  |  |
| 66.66967 | -0.00183 | 666685 | -19 | -0.132 |  |  |  |  |  |
| 66.67167 | -0.00383 | 666705 | -39 | -0.327 |  | Linest | Volts = m delta\_cnt + b | | |
| 66.67267 | -0.00483 | 666715 | -49 | -0.424 |  | m | b |  |  |
| 66.67667 | -0.00883 | 666755 | -89 | -0.814 |  | 0.009771 | 0.056128 |  |  |
| 66.68267 | -0.01483 | 666815 | -149 | -1.401 |  |  |  |  |  |
| 66.69067 | -0.02283 | 666895 | -229 | -2.1828 |  | Linest | Delta\_ms=m Volts + b | | |
| 66.70667 | -0.03283 | 666995 | -329 | -3.159 |  | m | b |  |  |
| 66.72067 | -0.05283 | 667195 | -529 | -5.112 |  | 0.010234 | -0.0005 |  |  |
| 66.74067 | -0.07283 | 667395 | -729 | -7.065 |  |  |  |  |  |
| 66.76067 | -0.09283 | 667595 | -929 | -9.018 |  | Linest | Volts = m delta\_ms + b | | |
| 66.66567 | 0.00217 | 666645 | 21 | 0.259 |  | m | b |  |  |
| 66.66467 | 0.00317 | 666635 | 31 | 0.3605 |  | 97.71083 | 0.049288 |  |  |
| 66.66267 | 0.00517 | 666615 | 51 | 0.556 |  |  |  |  |  |
| 66.66067 | 0.00717 | 666595 | 71 | 0.751 |  |  |  |  |  |
| 66.65667 | 0.01117 | 666555 | 111 | 1.141 |  |  |  |  |  |
| 66.65067 | 0.01717 | 666495 | 171 | 1.728 |  |  |  |  |  |
| 66.64067 | 0.02717 | 666395 | 271 | 2.705 |  |  |  |  |  |
| 66.63067 | 0.03717 | 666295 | 371 | 3.683 |  |  |  |  |  |
| 66.62067 | 0.04717 | 666195 | 471 | 4.66 |  |  |  |  |  |
| 66.61067 | 0.05717 | 666095 | 571 | 5.637 |  |  |  |  |  |
| 66.59067 | 0.07717 | 665895 | 771 | 7.592 |  |  |  |  |  |
| 66.57067 | 0.09717 | 665695 | 971 | 9.543 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |