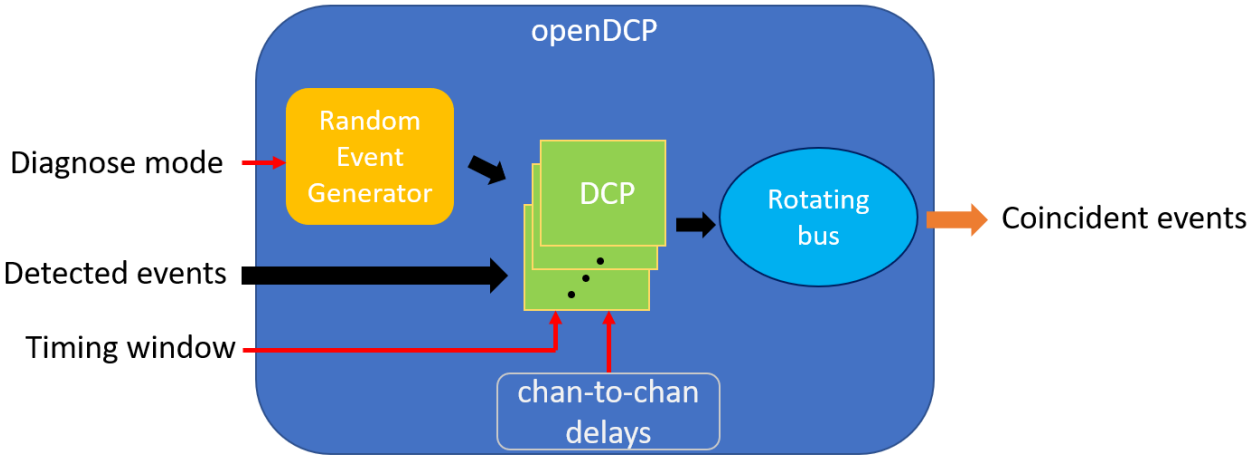
DCP user guide

This guided is for someone to apply DCP with the background in PET instrumentation and FPGA based data acquisition and signal processing experience.

1. Structure diagram of DCP firmware:



- DCP: Distributed Coincidence Processors.
- Detected events: Input detected singles event to DCP from PET detectors or formatted digital signals (which may generated from a seperaed FPGA board) with the same singles event data format as the output of PET singles event.
- Diagnose mode: A FPGA setting to select PET acquisition mode with input data to DCP from PET detectors or Diagnose mode with input data to DCP generated from internal Random Event Generator for diagnosis purpose.
- Timing window: The timing window to be applied to the DCP.
- chan-to-chan delays: The timing delays (or differences) among different DCP input channels. These timing delays among different channels will be compensated in the coincidence event selection by “aligning” the arrival timings among input channels.
- The outputs of DCP are the selected coincidence events.
- Other noes:

  DCP:
  A single Distributed Coincidence Processor (DCP) is responsible for comparing the timing difference between two events and deciding if they are coincident based on

the coincidence timing window. The number of CPs is mainly determined by detector channel numbers.

Rotating bus:
Each DCP will output a coincident event pair (doubled data length) if the two events are coincident. Considering in extreme condition (in testing mode or wth extremely high radioactive activity) the total coincidence event pairs could be close to the number of total coincidence processors, the data transfer and acquisition could be saturated. The Rotating bus is utilized to cache every event pair to FIFOs and output the event pairs in serial so that data loss can be prevented or minimized. In this specific implementation, Rotating bus was implemented and used to output the coincidence events. Other data transfer methods can certainly be implemented and output as users' choice.

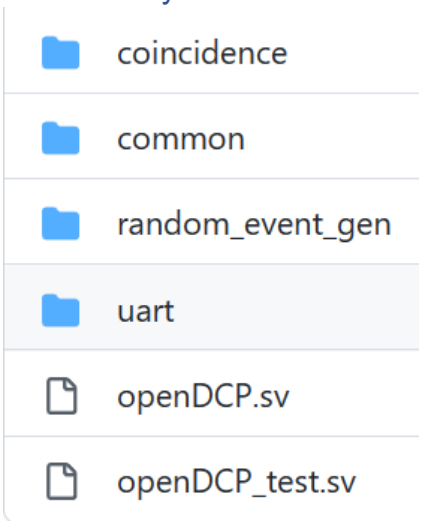
Random Event Generator:
This module generates 8-bit random numbers by means of Linear Feedback Shift Register (LFSR). It has the same number of channels as the number of input detectors. Each channel has a uniquely different seed to ensure different random number sequences among all channels.

Chan-to-Chan delays:
This table defines the timing delays among different detector channels. It provides a convenience to compensate the timing differences among difference detector channels.

This open source version of DCP is also called openDCP and used exchangably with DCP.

2. Directories and files under Code:

src directory:



Discriptions of files:

openDCP.sv is the top level file containing all modules and functions described above.

openDCP_test.sv is a simple example of DCP that uses internal Dignose Mode.

Coincidence

coincidence.v will let a paird event pass or reject when the timing difference between them is smaller or bigger than predefined timing window.

Coincidence_buffer.v is a FIFO wapper for caching singles event data before feeding it into a DCP.

dcp_pairs.v contains the DCP pairs for the PET system with specific number of detectors

delay_pairs.v contains the chan-to-chan delays

rotating_bus.v will cache all of the coincident events from every DCP and transfer them in serial so that the data transfer won't crash.

Common:

function_log2.v is to calculate the base-2 logarithm of a number

random_event_gen:

random_event_gen.sv will generate 8-bit random numbers.

lfsr_8bits.v is the algorithm for random event generator

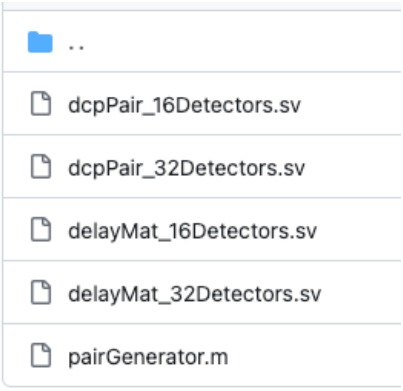
uart:

uart_rx.v is a receiver module of Universal Asynchronous Receiver/Transmitter (UART) communication.

uart_tx.v is a transmitter module of UART conmunication.

uart_tx_warp.sv is a wrapper of uart_rx for easy use.

matlab directory:



dcpPair_16Detectors.sv is an example of generated DCP pair matrix with 16 detector channels.

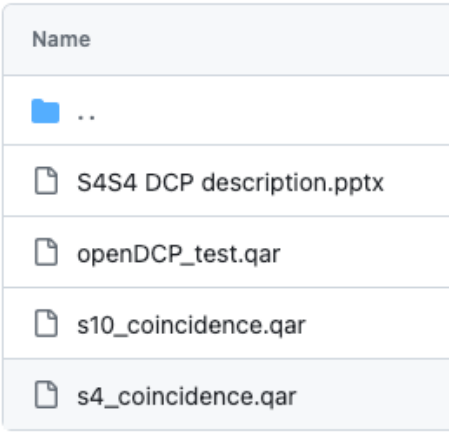
dcpPair_32Detectors.sv is an example of generated DCP pair matrix with 32 detector channels.

delayMat_16Detectors.sv is an example of generated chan-to-chan delays with 16 detector channels.

delayMat_32Detectors.sv is an example of generated chan-to-chan delays with 32 detector channels.

pairGenerator.m is MATLAB script that will generate DCP pair and chan-to-chan delay matrixs based on detector channels.

demo_project directory:



openDCP_test.qar is an archived FPGA project for DCP verification. It can be restored by Quartus Prime 19.1.

s10_coincidence.qar is an archived FPGA project for DCP testing with a lage volume Stratix 10 GX development board. Detailed information can be found with the following link:

https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=248&No=1068#contents

s4_coincidence.qar is an archived FPGA project for cascaded DCP testing with two Stratix 4 development board. Detailed information can be found with the following link:

https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=138&No=683#contents

S4S4 DCP description.pptx describes how the firmware workd between two FPGA boards.

3. FPGA and Parameter setting

openDCP module:

openDCP.sv

You, last week | 1 author (You)

```
`include "coincidence/dcp_pairs.sv"
`include "coincidence/delay_pairs.sv"
`include "coincidence/coincidence.v"
`include "coincidence/rotating_bus.sv"
`include "random_event_gen/random_event_gen.sv"

module openDCP(
    clk_200M,
    rst_n,


    test_mode,
    timing_window,

    event_data,
    event_data_en,

    coincidence_data,
    coincidence_data_en
);
```

Ports:

```
//=============================================================
//  Port declarations
//=============================================================
input                                clk_200M;
input                                rst_n;

input                       [1:0]    test_mode;
input                       [7:0]    timing_window;

input [DETECTOR_DATA_WIDTH-1:0]      event_data             [LVDS_CHAN_NUM-1:0];
input       [LVDS_CHAN_NUM-1:0]      event_data_en;

output    [PAIR_DATA_WIDTH-1:0]      coincidence_data;
output                               coincidence_data_en;
```

clk_200M is the *working clock* for the module and its frequency is 200 MHz.

rst_n is a negative *active reset signal*.

test_mode == 2'b01 or 2'b10 enters Diagnose Mode; 2'b00 exits Diagnose Mode and enters Acquisition Mode)

When test_mode = 2'b01, internal Random Event Generator puts 8-bit random numbers in Fine Stamp (see below) so that all events are coincident. When test_mode = 2'b10, the random numbers are placed in Coarse Stamp[7:0], those events only with their timing differences (also the difference of Coarse Stamp[7:0]) are less than Timing Window are considered as coincident.

timing_window (8-bit) is used to set the coincidence timing window. It ranges from 0-255 with a unit of the event's TDC coarse timing. For example, if the TDC's coarse and fine timing resolution is 5ns and 19.5ps respectively, then a timing_window of 3 equals to 15ns coincidence window.

event_data receives *detector's singles data*. It is a singles-event-data-width vector with a depth of number of detector channels.

event_data_en is the *singles data valid signal*. It has the same bit width of detector channels.

coincidence_data is an *output signal* representing coincident event pair.

coincidence_data_en is the *coincident event valid signal*.

User-defined parameters:

```
//=====================================================
//  PARAMETER declarations
//=====================================================
parameter LVDS_CHAN_NUM = 8'd32;
parameter CHANNEL_TIMING_WIDTH = 8'd32 + 8'd16 + 8'd8;
parameter TIMING_START_POS = 16*4, TIMING_COARSE_POS = 16*4+8, TIMING_COARSE_WIDTH = 24;
parameter P_WIDTH = 8'd16, E_WIDTH = 8'd16, BOARDID_WIDTH = 8'd8;
parameter DETECTOR_DATA_WIDTH = P_WIDTH*2 + E_WIDTH*2 + CHANNEL_TIMING_WIDTH + BOARDID_WIDTH;
parameter PAIR_WIDTH = 8'd16; parameter PAIR_DATA_WIDTH = DETECTOR_DATA_WIDTH*2 + PAIR_WIDTH;
```

LVDS_CHAN_NUM	Number of detector channels
CHANNEL_TIMING_WIDTH	Bit width of timing information
TIMING_START_POS	The position where timing information starts
TIMING_COARSE_POS	The position where coarse timing information starts
TIMING_COARSE_WIDTH	Bit width of coarse timing information
P_WIDTH	Bit width of event position information
E_WIDTH	Bit width of event energy information
BOARDID_WIDTH	Bit width of detector ID
DETECTOR_DATA_WIDTH	Bit width of detector total data
PAIR_WIDTH	Bit width of number of DCP pairs
PAIR_DATA_WIDTH	Bit width of coincident event data

The default values of each parameter have been assigned when they are created. They should not exceed the maximal value defined by the data bit width. Here we define the number of detector channels:

LVDS_CHAN_NUM is 32.

CHANNEL_TIMING_WIDTH is 56 defined as the bit width of the timing information of an event.

The TIMING_START_POS, TIMING_COARSE_POS, and TMING_COARSE_WIDTH together determin the details of TDC timing information.

P_WIDTH and E_WIDTH define the widths of position and energy respectively.

BOARDID_WIDTH set the bit width of the ID of each detector output electronics.

With the parameters mentioned above, DETECTOR_DATA_WIDTH is the bit width of one detector's singles event data and is equal to the summed bit width of the above defined parameters.

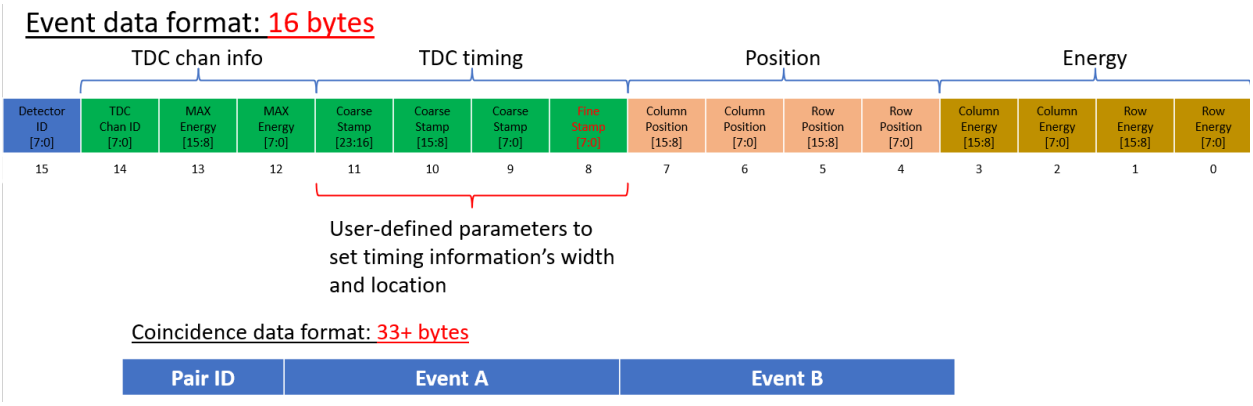
PAIR_WIDTH is the DCP pair width which is set to 16, meaning that the maximal DCP pair number is 65536 that should be enough for a practical brain or preclinical PET system.

The PAIR_DATA_WIDTH is simply two times of singles data width plus the PAIR_WIDTH.

Check the data fromat in section 4 for more information.

4. Data format

Singles interaction event and coincidence event data format:



Brief descriptions of each items in the data format:

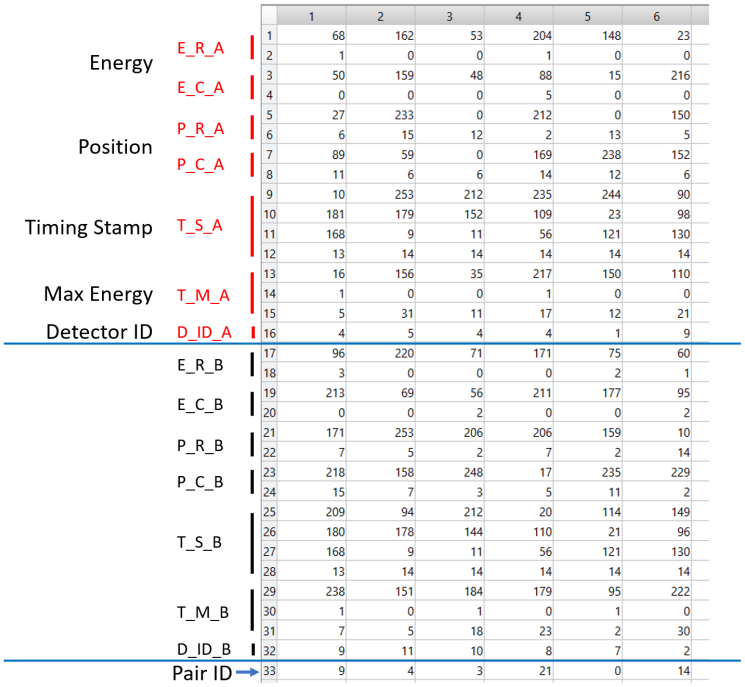
1. Singles interaction event data format (16 Bytes):
    a. TDC channel information includes TDC ID and Max energy of a single interaction when there are multiple interactions.
    b. TDC timing (3 Bytes coarse timing stamp and 1 Byte fine timing stamp). In project implementation, coarse timing stamp range and fine timing stamp range are 83.9 ms and 5 ns, respectively)
    c. Representative event position (column and row) within detector interaction map. Depth of interaction was not included but can be rearranged within these 4 Bytes space.
    d. Representative (summed) event energy along the event position's column and row.

    Note: Data format shown here is implemented with project. Please note that within the singles event data format, only the byte number 8, 9, 10, 11, 14, and 15 are DCP specific and should be strictly kept for storing intended event timing and id information. Other event Position and Energy information (byte number 0-7 and 12-13) are PET system specific and won't be used in DCP processing.

2. Coincidence event data format (33+ Bytes):
    a. Paired ID is usually 1-2 Byte.
    b. Event A and B are paired events.

Example:

Below shows 6 coincident events (33x6), each coincident event has 33 bytes:



			1	2	3	4	5	6
Energy	E_R_A	1	68	162	53	204	148	23
		2	1	0	0	1	0	0
	E_C_A	3	50	159	48	88	15	216
		4	0	0	0	5	0	0
Position	P_R_A	5	27	233	0	212	0	150
		6	6	15	12	2	13	5
	P_C_A	7	89	59	0	169	238	152
		8	11	6	6	14	12	6
Timing Stamp	T_S_A	9	10	253	212	235	244	90
		10	181	179	152	109	23	98
		11	168	9	11	56	121	130
		12	13	14	14	14	14	14
Max Energy	T_M_A	13	16	156	35	217	150	110
		14	1	0	0	1	0	0
Detector ID	D_ID_A	15	5	31	11	17	12	21
		16	4	5	4	4	1	9
	E_R_B	17	96	220	71	171	75	60
		18	3	0	0	0	2	1
	E_C_B	19	213	69	56	211	177	95
		20	0	0	2	0	0	2
	P_R_B	21	171	253	206	206	159	10
		22	7	5	2	7	2	14
	P_C_B	23	218	158	248	17	235	229
		24	15	7	3	5	11	2
	T_S_B	25	209	94	212	20	114	149
		26	180	178	144	110	21	96
		27	168	9	11	56	121	130
		28	13	14	14	14	14	14
	T_M_B	29	238	151	184	179	95	222
		30	1	0	1	0	1	0
	D_ID_B	31	7	5	18	23	2	30
		32	9	11	10	8	7	2
Pair ID →		33	9	4	3	21	0	14

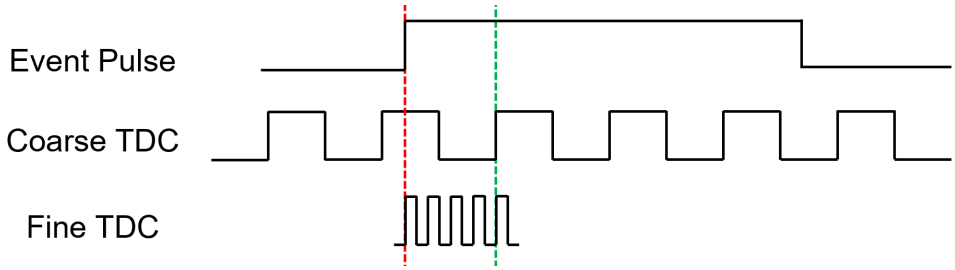
Coarse and fine timing:

To achieve both high resolution and a wide measurement range, many TDCs use a combination of coarse and fine timing mechanisms.

Coarse Timing: This mechanism measures the coarse time intervals and provides the basic time measurement resolution. It typically uses a lower-frequency clock or a counter that increments with each cycle of a reference clock. Coarse timing gives the TDC its range but with limited resolution.

Fine Timing: Fine timing often uses techniques such as time interpolation, where the time between two reference clock edges is divided into smaller intervals using a faster clock or an analog-to-digital conversion of a time-dependent voltage. This mechanism allows the TDC to measure time intervals with higher precision. In FPGA, delay line is often used as the time interpolation technology.

In the project, the coarse timing of TDC is made of a 24-bit counter working at a 200MHz clock, thus its resolution is 5 ns and has a maximal measurement range of (2^24)x5ns = 83.9 ms. It will always count from 0 to 0xFFFFFF each time after power up or reset. The fine timing is a tapped-delay-line TDC which further interpolates the 5 ns into 255 intervals (8-bit), so the fine timing resolution is 5 ns / 255 = 19.6 ps. It will start to work after an event triggers, see the figure below:



When an event signal arrives (red dashed line), the coarse TDC will record the arrival time with the closest rising clock edge (green dashed line). The coarse timing stamp will be coarse TDC's counter number multiplies 5 ns. Meanwhile, the fine TDC also counts how many small intervals that take to propagate to the same rising edge. The fine timing stamp is then calculated by the interval number multiplies 19.5 ps. Thus the actual event's timing stamp is calculated by subtracting the fine timing stamp from the coarse timing stamp.

Thus, the event (arrival) timing stamp = [Coarse timing – Fine timing].

Take the first column of the above table as an example:

The first (1-16) and second (17-32) 16 rows contain the singles event data acquired from Detector A and B, respectively.

The timing stamp of each singles event is calculated as the following (for the first singles event A and B):

Timging stamp A: T_S_A = [-event(9) + event(10)x256 + event(11)x256^2 + event(12)x256^3]x(TDC timing resolution) = [-10 + 181x256 + 168x256^2 + 13x256^3]x 19.5 ps = 4475784.8 ns

Timging stamp B: T_S_B = [-event(25) + event(26)x256 + event(27)x256^2 + event(28)x256^3]x(TDC timing resolution) = [-209 + 180x256 + 168x256^2 + 13x256^3]x19.5 ps = 4475775.9 ns

The TDC resolution used in the project was 19.5 ps which was the same for both detector readout electronics.

The timing difference between the first two singles event A and B = T_S_A – T_S_B = 4475784.8 - 4475775.9 = 8.9 ns.

If the timing window is set to □10 ns, the first two singles event A and B (referring to the first column) would be accepted as a coincident event.

5. Generation of DCP pairs and Chan-to-chan delays

Use the MATLAB script under directory matlab to generate DCP pairs and delay matrix; Simply change the “detectors” and “det_gap” to the numbers as your PET scanner and run the code as shown in the following:

```
%% Scanner parameters
detectors = 32; % Detector number
det_gap = 2; % Avoid generating coincidence pairs for two adjecant detectors on both sides
total_pairs = detectors * (detectors-1-2*det_gap) / 2
%% Pair Generation
fid = fopen('dcpPairs.sv','w');
fwrite(fid, ['parameter int PAIR_NUM = ' num2str(total_pairs) ';' newline 'parameter int COINCIDENCE_PAIR_MAT[PAIR_NUM][2] = ''{'
%%
pairs_num = [];
for ii = 0:detectors-1
    pairs_str = [];
    for jj = 0:detectors-1
        if abs(ii-jj) <= det_gap || abs(ii-jj) >= detectors - det_gap || jj < ii % avoid adjecant 2 detectors and repeated pairs
            continue
        end
        pairs_num = [pairs_num [ii; jj]];
        pairs_str = [pairs_str [char(39), '{', num2str(ii), ',', num2str(jj), '},']];
    end
    fwrite(fid,[pairs_str newline]);
end
fwrite(fid, '};');
fclose all;
%% Channel Delay matrix gen
%chan_delay = []; % Relative physical channel delays. Rounded. Substract the smallest channel delay
chan_delay = round(rand(1,detectors)*5) % Random delay for test
%%
fid = fopen('delayMat.sv','w');
fwrite(fid, ['parameter int DELAY_PAIR = ' num2str(total_pairs) ';' newline 'parameter int CHAN_DELAY_MAT[PAIR_NUM][2] = ''{']);
%%
delay_pair = [];
for ii = 1:detectors
    pairs_str = [];
    for jj = 1:detectors
        if abs(ii-jj) <= det_gap || abs(ii-jj) >= detectors - det_gap || jj < ii % avoid adjecant 2 detectors and repeated pairs
            continue
        end
        delay_pair = [delay_pair [chan_delay(ii); chan_delay(jj)]];
        pairs_str = [pairs_str [char(39), '{', num2str(chan_delay(ii)), ',', num2str(chan_delay(jj)), '},']];
    end
    fwrite(fid,[pairs_str newline]);
end
fwrite(fid, '};');
fclose all;
```

Example:

Generated DCP pairs and skew pairs below with 16 detector channels:

```
parameter int PAIR_NUM = 104;
parameter int COINCIDENCE_PAIR_MAT[PAIR_NUM][2] = '{'{0,2},'{0,3},'{0,4},'{0,5},'{0,6},'{0,7},'{0,8},'{0,9},'{0,10},'{0,11},'{0,12},'{0,13},'{0,14},
                                                   '{1,3},'{1,4},'{1,5},'{1,6},'{1,7},'{1,8},'{1,9},'{1,10},'{1,11},'{1,12},'{1,13},'{1,14},'{1,15},
                                                   '{2,4},'{2,5},'{2,6},'{2,7},'{2,8},'{2,9},'{2,10},'{2,11},'{2,12},'{2,13},'{2,14},'{2,15},
                                                   '{3,5},'{3,6},'{3,7},'{3,8},'{3,9},'{3,10},'{3,11},'{3,12},'{3,13},'{3,14},'{3,15},
                                                   '{4,6},'{4,7},'{4,8},'{4,9},'{4,10},'{4,11},'{4,12},'{4,13},'{4,14},'{4,15},
                                                   '{5,7},'{5,8},'{5,9},'{5,10},'{5,11},'{5,12},'{5,13},'{5,14},'{5,15},
                                                   '{6,8},'{6,9},'{6,10},'{6,11},'{6,12},'{6,13},'{6,14},'{6,15},
                                                   '{7,9},'{7,10},'{7,11},'{7,12},'{7,13},'{7,14},'{7,15},
                                                   '{8,10},'{8,11},'{8,12},'{8,13},'{8,14},'{8,15},
                                                   '{9,11},'{9,12},'{9,13},'{9,14},'{9,15},
                                                   '{10,12},'{10,13},'{10,14},'{10,15},
                                                   '{11,13},'{11,14},'{11,15},
                                                   '{12,14},'{12,15},
                                                   '{13,15}
                                                   };
```

chan_delay = [4 2 1 2 0 1 5 5 3 0 1 2 4 0 0 1]

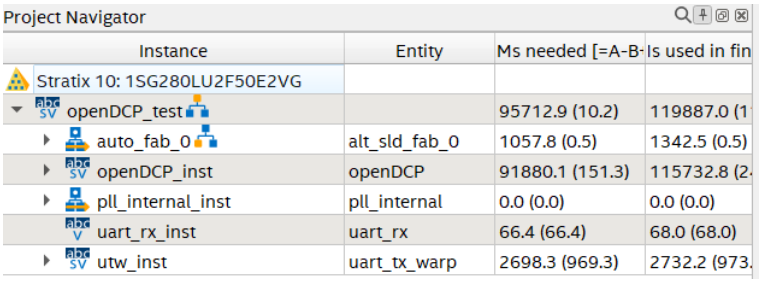
```
parameter int DELAY_PAIR = 104;
parameter int CHAN_DELAY_MAT[PAIR_NUM][2] = '{'{4,1},'{4,2},'{4,0},'{4,1},'{4,5},'{4,5},'{4,3},'{4,0},'{4,1},'{4,2},'{4,4},'{4,0},'{4,0},
                                              '{2,2},'{2,0},'{2,1},'{2,5},'{2,5},'{2,3},'{2,0},'{2,1},'{2,2},'{2,4},'{2,0},'{2,0},'{2,1},
                                              '{1,0},'{1,1},'{1,5},'{1,5},'{1,3},'{1,0},'{1,1},'{1,2},'{1,4},'{1,0},'{1,0},'{1,1},
                                              '{2,1},'{2,5},'{2,5},'{2,3},'{2,0},'{2,1},'{2,2},'{2,4},'{2,0},'{2,0},'{2,1},
                                              '{0,5},'{0,5},'{0,3},'{0,0},'{0,1},'{0,2},'{0,4},'{0,0},'{0,0},'{0,1},
                                              '{1,5},'{1,3},'{1,0},'{1,1},'{1,2},'{1,4},'{1,0},'{1,0},'{1,1},
                                              '{5,3},'{5,0},'{5,1},'{5,2},'{5,4},'{5,0},'{5,0},'{5,1},
                                              '{5,0},'{5,1},'{5,2},'{5,4},'{5,0},'{5,0},'{5,1},
                                              '{3,1},'{3,2},'{3,4},'{3,0},'{3,0},'{3,1},
                                              '{0,2},'{0,4},'{0,0},'{0,0},'{0,1},
                                              '{1,4},'{1,0},'{1,0},'{1,1},
                                              '{2,0},'{2,0},'{2,1},
                                              '{4,0},'{4,1},
                                              '{0,1}};
```

6. Compile DCP

Compile code

After setting the user-defined parameters and generating the required DCP pairs and chan-to-chan delays, a full compilation can be performed. An openDCP_test.sv file is also included as an example

After compilation, all modules used will be displayed as shown below:



The openDCP_inst contains all the DCPs:

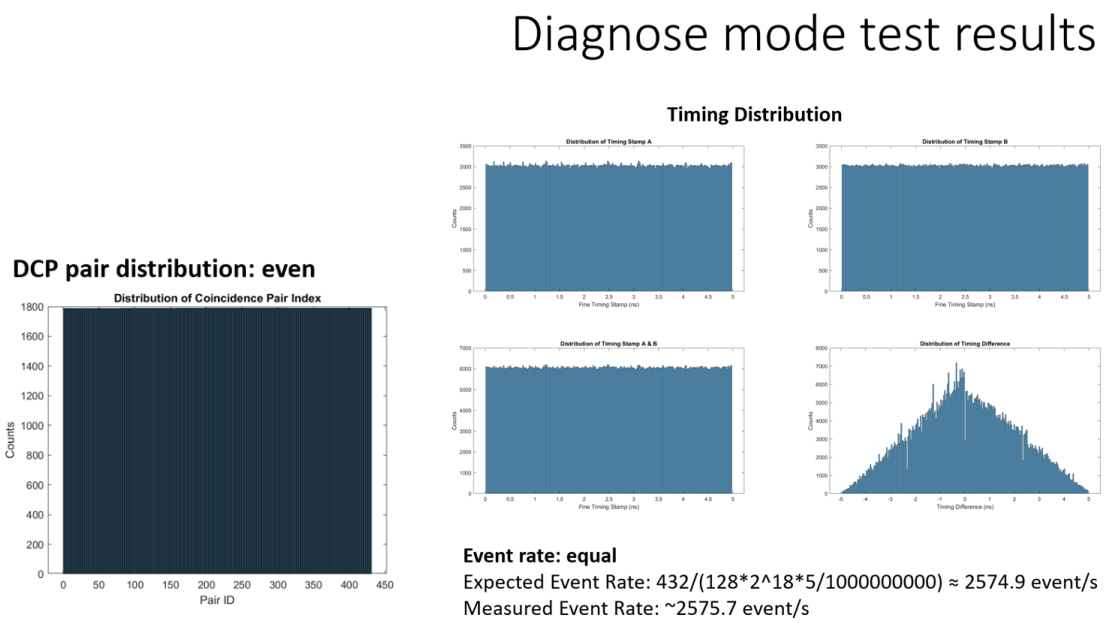
Note: The figure only shows part of the 432 CPs corresponding to 32 Detectors.

7. Diagnose or Acquisiiton mode

Diagnose Mode

Entering Diagnose Mode by sending 2'b01 or 2'b10 to Port "test_mode" to FPGA board via UART.

Example: In the development project, singles data from Random Event Generator were acquired and analyzed. As shown below, the results indicate Random Event Generator were running without errors.



Acquisition Mode

Entering Acquisition Mode by sending 2'b00 to FPGA board via UART.