

Advance SoC

Lab 2-2 – Catapult HLS – Edge Detect on FSIC

110590022 陳冠晰

GitHub Link: https://github.com/vic9112/Advance_SOC

How we design our work (5 **modifications** and “**additional optimization**”)

- Process four pixels per clock cycle. (1)

Inside the C testbench, we send 4 pixels simultaneously:

```
pixelType4x pix4;
for (unsigned int y = 0; y < height; y++) {
    for (unsigned int x = 0; x < width; x+=4) {
        pix4.set_slc( 0, (pixelType) yarray[cnt+0]);
        pix4.set_slc( 8, (pixelType) yarray[cnt+1]);
        pix4.set_slc(16, (pixelType) yarray[cnt+2]);
        pix4.set_slc(24, (pixelType) yarray[cnt+3]);
    }
}
```

Here I pack four 8-bit pixels into one 32-bit data.

```
typedef uint32 pixelType4x;
```

In [hls_bluebook.pdf](#) page 102, we can see the following example which use conditional tasks to merge IO:

Example 51 Simplifying Conditional IO to Help Merging

```
1
2 void accumulate (int din[4],
3                 int &dout,
4                 bool flag[4])
5 {
6     int acc=0;
7     int din_int[4];
8     bool flag_int;
9     FLAG:for (int i=0; i<4; i++)
10    { flag_int |= flag[i]; }
11    DIN:for (int i=0; i<4; i++)
12        if (flag_int)
13            { din_int[i] = din[i]; }
14    ACCUM:for (int i=0; i<4; i++) {
15        if (flag[i])
16            { acc += din_int[i]; }
17        else
18            { acc += 0; }
19    }
20    dout = acc;
21 }
```

PATH: \$MGC_HOME/shared/examples/docs/bluebook/schedule/throughput_limit_cond.cpp

In our design, we use the function `set_slc()`, which is a function that replaces slice of bits. For example, we calculate the derivative of y and write into `pix4`, which is a 32-bit unsigned integer type that contains four 8-bit pixel values.

```

for(int i = 0; i < 4; i++)
{
    gradType pix_tmp;
    pix_tmp = pix2.slc<8>(i * 8) * hw_kernel[0] +
              pix1.slc<8>(i * 8) * hw_kernel[1] +
              pix0.slc<8>(i * 8) * hw_kernel[2];
    pix4.set_slc(i * 9, pix_tmp);
}

```

- Use sum of absolute difference (SAD) for edge magnitude calculation. (2)

Originally, we use square root with piecewise linear implementation:

```

ac_math::ac_sqrt_pwl(sum,sq_rt);
magn.write(sq_rt.to_uint());

```

Now we use SAD for edge magnitude calculation.

```

pixelType abs_dx, abs_dy;

#pragma hls_unroll yes
for(int i = 0; i < 4; i++)
{
    ac_math::ac_abs(dx.slc<9>(i * 9), abs_dx);
    ac_math::ac_abs(dy.slc<9>(i * 9), abs_dy);
    uint9 abs_sum = abs_dx + abs_dy;
    ac_fixed<8, 8, false, AC_TRN, AC_SAT> abs_sum_clip = abs_sum;
    magType tmp = (magType) abs_sum_clip.to_uint();
    magn.set_slc(i * 8, tmp);
}

if (!sw_in)
    magn_out = pix;
else
    magn_out = magn;

```

The function calculating absolute value needs to be included from **ac_math**

library. After getting the absolute value of **dx** and **dy**, we can add them together to get SAD.

- Add two crc32 calculation on image input/output. (3)

Using “crc32” to calculate input and output:

```

crc32_pix_in_tmp = calc_crc32<32>(crc32_pix_in_tmp, pix);
crc32_dat_out_tmp = calc_crc32<32>(crc32_dat_out_tmp, magn_out);

```

Which the “crc32” function is provided:

Link:< https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/catapult_hls/crc32.cpp>

```
private:
template <int len>
uint32 calc_crc32(uint32 crc_in, ac_int<len, false> dat_in)
{
    const uint32 CRC_POLY = 0xEDB88320;
    uint32 crc_tmp = crc_in;

    #pragma hls_unroll yes
    for(int i=0; i<len; i++)
    {
        uint1 tmp_bit = crc_tmp[0] ^ dat_in[i];

        uint31 mask;

        #pragma hls_unroll yes
        for(int i=0; i<31; i++)
            mask[i] = tmp_bit & CRC_POLY[i];

        uint31 crc_tmp_h31 = crc_tmp.slc<31>(1);

        crc_tmp_h31 ^= mask;

        crc_tmp.set_slc(31,tmp_bit);
        crc_tmp.set_slc(0,crc_tmp_h31);
    }
    return crc_tmp;
}
```

- Select the output source from input image or the calculated magnitude. (4)

Using `sw_in` defined on top function to select the output source:

`pix`: input image.

`magn`: calculated magnitude.

```
if (!sw_in)
    magn_out = pix;
else
    magn_out = magn;
```

- Remove the angle calculation. (5)

Originally, we output our calculated angle and magnitude, and testbench will generate image file after received them.

Top function defined in the original design:

```
void CCS_BLOCK(run)(ac_channel<pixelType> &dat_in,
                  maxWType &widthIn,
                  maxHType &heightIn,
                  ac_channel<magType> &magn,
                  ac_channel<angType> &angle)
{
    VerDer_inst.run(dat_in, widthIn, heightIn, dat, dy);
    HorDer_inst.run(dat, widthIn, heightIn, dx);
    MagAng_inst.run(dx, dy, widthIn, heightIn, magn, angle);
}
```



The new top function:

```
void CCS_BLOCK(run)(maxWType      widthIn,
                   maxHType      heightIn,
                   bool           sw_in,
                   uint32         &crc32_pix_in,
                   uint32         &crc32_dat_out,
                   ac_channel<Stream_t> &dat_in,
                   ac_channel<Stream_t> &dat_out)
{
    // #pragma busifc widthIn      WordOffset=0 Slave=slv0
    // #pragma busifc heightIn     WordOffset=1 Slave=slv0
    // #pragma busifc sw_in        WordOffset=2 Slave=slv0
    // #pragma busifc crc32_pix_in WordOffset=3 Slave=slv0
    // #pragma busifc crc32_dat_out WordOffset=4 Slave=slv0
    VerDer_inst.run(dat_in, widthIn, heightIn, pix_chan1, dy_chan);
    HorDer_inst.run(pix_chan1, widthIn, heightIn, pix_chan2, dx_chan);
    MagAng_inst.run(dx_chan, dy_chan, pix_chan2, widthIn, heightIn, sw_in, crc32_pix_in, crc32_dat_out, dat_out);
}
```

Notice that `dat_out` only select the `pixel_in` or `magn_out`, the angle output is not existed anymore since we don't need to calculate the angle. Instead, we add `crc32` input and output.

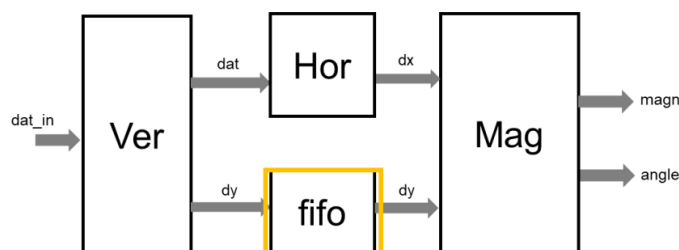
Additional Optimization (which is not the version we run on FSIC!!)

- Our original design has the following throughput and area:

Solution /	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
 solution.v1 (new)						
 EdgeDetect_Top.v1 (extract)	7	70.00	6	60.00	7.30	10399.01

The reason why throughput is “6” is described as below:

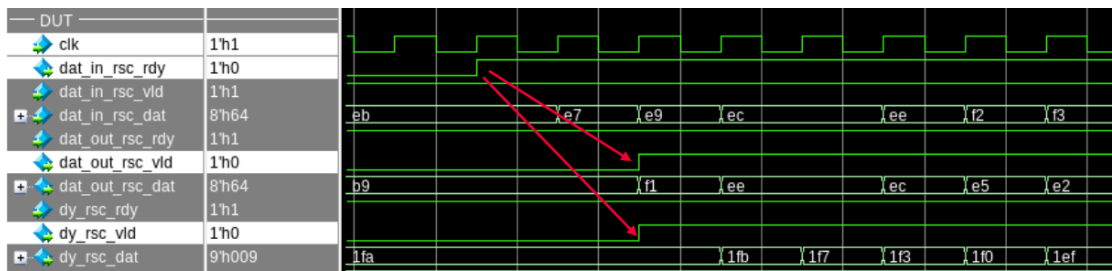
Let's look at the block diagram of our design:



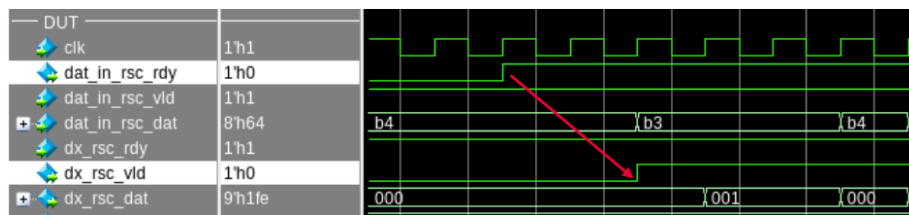
We need to add a “FIFO” between module **Ver** and **Mag**, which delay the arrival time of `dy`, make sure that `dx` and `dy` arrive at the same clock cycle.

In the original architecture, we can see that both module **Hor** and **Ver** takes 2 clock periods to generate one output (latency = 2)

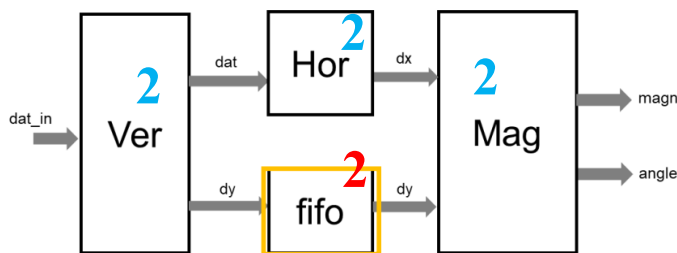
Ver:



Hor:



Thus, we add a FIFO with fifo length=2:



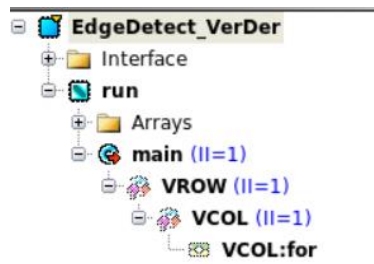
$$2 + 2 + 2 = 6$$

- Now, I try to improve the throughput of original design from 6 to 1, the result is shown below:

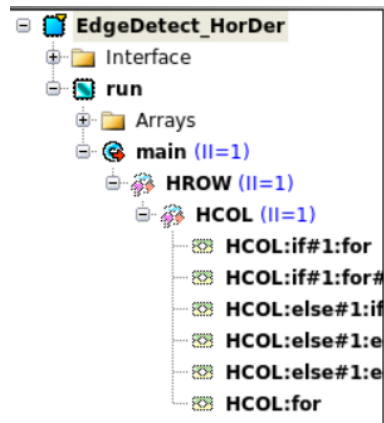
Solution /	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
solution.v1 (new)						
EdgeDetect_Top.v1 (extract)	7	70.00	6	60.00	7.30	10399.01
EdgeDetect_Top.v2 (extract)	7	70.00	6	60.00	7.30	10399.01
EdgeDetect_Top.v3 (extract)	6	60.00	1	10.00	7.79	10637.66

What I do is unrolling the loops and pipelining the function of each step:

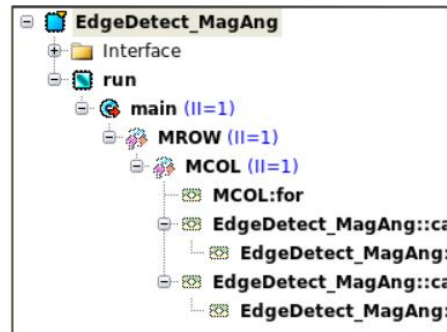
■ Ver:



■ Hor:

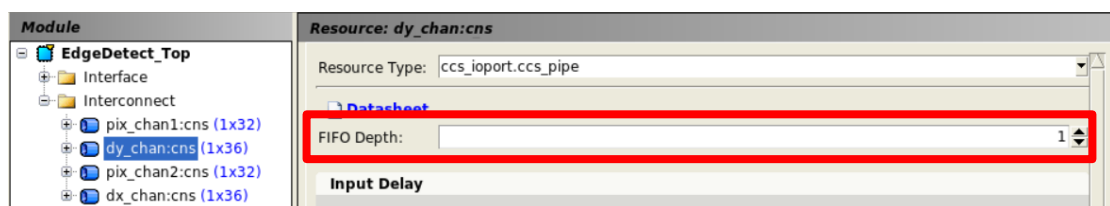


■ MagAng:



We can see that the area only increases a little, the tradeoff between area and throughput is acceptable.

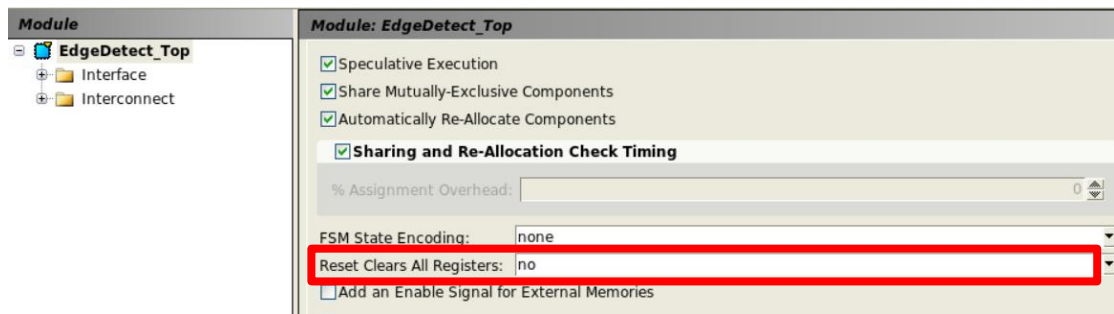
- Next step is to change the length of FIFO because of the new latency of each module.



The new FIFO depth is set to 1.

- Turn off resets on all registers for area reduction.

- Only registers that need to be reset for correct simulation result are reset.

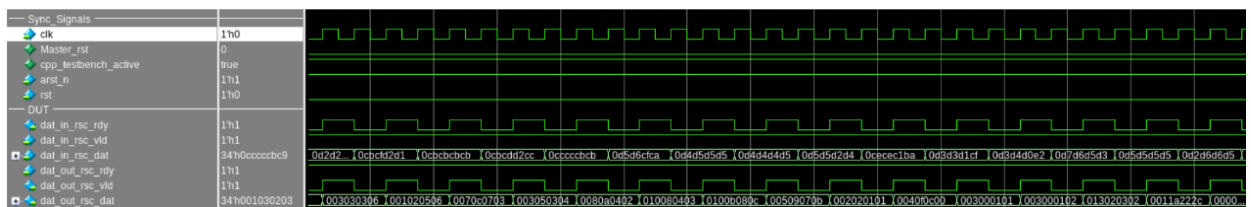


- Result:

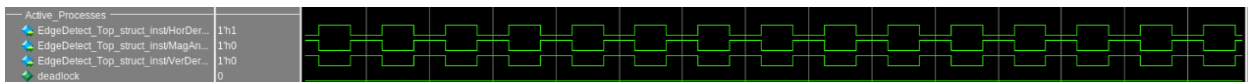
Solution /	Latency...	Latency...	Throug...	Throug...	Slack	Total Area
solution.v1 (new)						
EdgeDetect_Top.v1 (extract)	7	70.00	6	60.00	7.30	10399.01
EdgeDetect_Top.v2 (extract)	7	70.00	6	60.00	7.30	10399.01
EdgeDetect_Top.v3 (extract)	6	60.00	1	10.00	7.79	10637.66
EdgeDetect_Top.v4 (extract)	6	60.00	1	10.00	7.79	8391.70

- RTL simulation:

We get **dout** every two clock cycles:



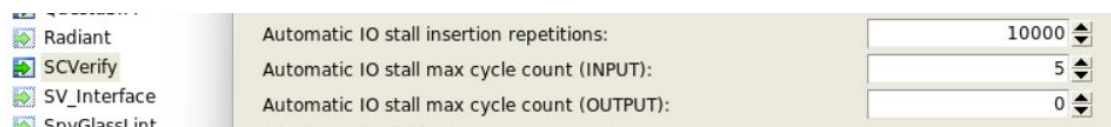
Active time of three modules:



We can see that **Hor** active right after **Ver**, and since we add a FIFO with **fifo_length=1** to make sure that **Mag** receive **dx** and **dy** at the same clock cycle, **Mag** can do the calculation after getting the **dx** from **Hor**.

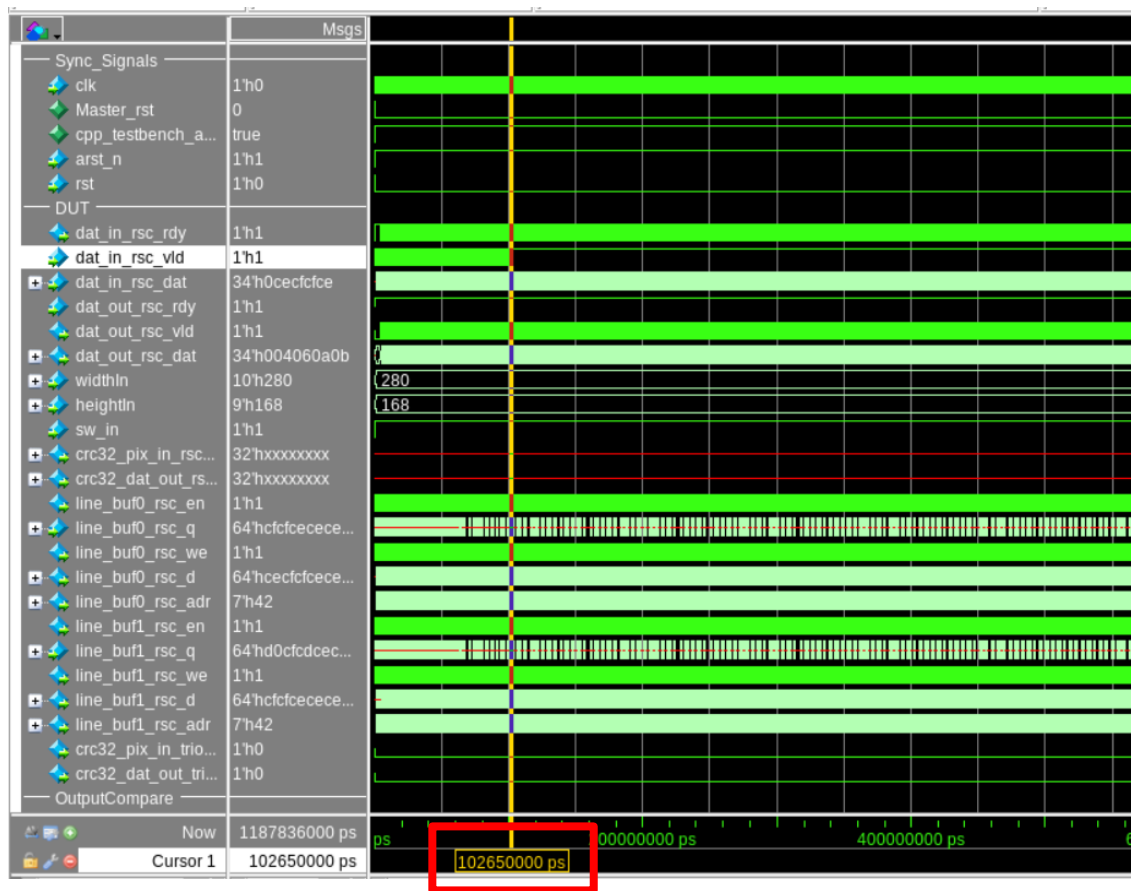
- Controlling stalling behavior in the testbench:

- Stall **dat_in**, totally 10000 samples, for random cycles between 0 and 5:

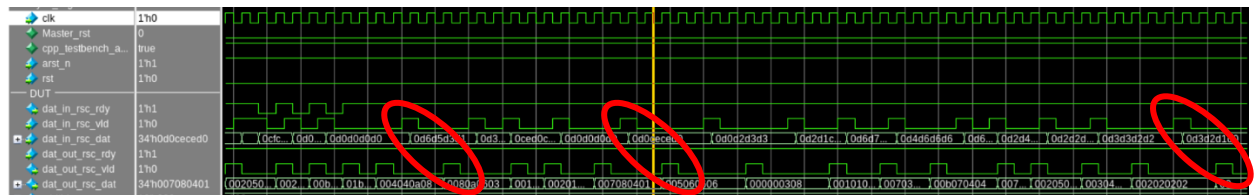


■ Waveform:

Only the first 10000 samples (100000ns=100000000ps) have the random cycles:



Zoom in:



Input data valid → Output data valid: 2 clock cycles.

We can see that despite randomly stalling the input signal, the EdgeDetect module still reacts correctly.

Test Result of Catapult design (C design checker, testbench)

- C design checker

Design checker raise the **UMR error** which is “uninitialized memory read”.

```
ERROR
UMR - Uninitialized Memory Read - 3
EdgeDetect_VerDer.h:49:26
| 47         if ( (x4 & 1) == 0 ) {
| 48             // vertical window of pixels
> 49             rdbuf1_pix = line_buf1[x4 / 2];
| 50             rdbuf0_pix = line_buf0[x4 / 2];
| 51         } else { // Write line buffer caches on odd iterations of COL loop

EdgeDetect_VerDer.h:50:26
| 48             // vertical window of pixels
| 49             rdbuf1_pix = line_buf1[x4 / 2];
> 50             rdbuf0_pix = line_buf0[x4 / 2];
| 51         } else { // Write line buffer caches on odd iterations of COL loop
| 52             line_buf1[x4 / 2] = rdbuf0_pix; // copy previous line

EdgeDetect_HorDer.h:62:24
| 60         #pragma hls_unroll yes
| 61         for (int i = 0; i < 5; i++) {
> 62             win[i] = win[i + 4];
| 63         }
| 64         #pragma hls_unroll yes
```

Since `rdbuf1_pix`, `rdbuf0_pix` and `win[9]` are not initialized when we defined them.

Although we've encountered uninitialized issues, these problems will not affect our ability to generate the correct magnitude.

Following shows that we've also got the warning about **FVI**, which is “For Loop with Variable Iterations”. The reason why the warning was generated is that boundaries in the following ‘for loops’ are not given.

```
WARNING
FVI - For Loop with Variable Iterations - 6
EdgeDetect_VerDer.h:34:15
| 32         VROW: for (maxHType y = 0; ; y++) { // VROW has one extra iteration to ramp-up window
| 33             #pragma hls_pipeline_init_interval 1
> 34             VCOL: for (maxWType x = 0; ; x += 4) {
| 35                 x4 = x / 4;
| 36                 if (y <= heightIn - 1) {

EdgeDetect_VerDer.h:32:13
| 30         // Remove loop upperbounds for RTL code coverage
| 31         // Use bit accurate data types on loop iterator
> 32         VROW: for (maxHType y = 0; ; y++) { // VROW has one extra iteration to ramp-up window
| 33             #pragma hls_pipeline_init_interval 1
| 34             VCOL: for (maxWType x = 0; ; x += 4) {

EdgeDetect_HorDer.h:32:15
| 30         HROW: for (maxHType y = 0; ; y++) {
| 31             #pragma hls_pipeline_init_interval 1
> 32         HCOL: for (maxWType x = 0; ; x += 4) { // Add 4
| 33             x4 = x / 4; // Four pixels, s.t. x divided by 4
| 34

EdgeDetect_HorDer.h:30:13
| 28         // Since we've got 4 pixel at one time,
| 29         // all "widthIn" in origin code must divided by 4
> 30         HROW: for (maxHType y = 0; ; y++) {
| 31             #pragma hls_pipeline_init_interval 1
```

- *Testbench*

Original C testbench:

```

severity_EdgeDetect.exe -image run.sh
[ui110590022@ws40 bin]$ source run.sh
Loading Input File
Input file:      ./image/people640x360_rgb.bmp
Mode:           1
Output file (alg): out_algorithm.bmp
Output file (hw): out_hw.bmp
Image width: 640
Image height: 360
##### FRAME NO. 0 #####
Running
Magnitude: Manhattan norm per pixel 5.357491
Writing algorithmic bitmap output to: out_algorithm.bmp
Writing bit-accurate bitmap output to: out_hw.bmp
sofErr: 0 eofErr: 0
crc32_alg_pix_in = ebb44e76  crc32_hw_pix_in = ebb44e76
crc32_alg_dat_out = 398625ad  crc32_hw_dat_out = 49e564fe
Finished

```

How to Integrate our Design in FSIC

- After finish generating RTL on Catapult, we will get `concat_EdgeDetect_Top.v` under the synthesis folder.

Copy `concat_EdgeDetect_Top.v` as `concat_EdgeDetect_Top_fsic.v` in `03_fsic_prj/dsn/rtl`:

- Instantiate EdgeDetect in user project 0, which has provided before:

https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/fsic-sim/fsic_fpga/rtl/user/user_subsys/user_prj0/rtl/user_prj0.v

Above also add `ctrl_regs` for read and write, and two SRAMS (`spram.v`)

- File list:

```

1 | ../fpga.v
2 | ../rtl/axi_ctrl_logic.v
3 | ../rtl/axil_axis.v
4 | ../rtl/axilite_master.v
5 | ../rtl/axilite_slave.v
6 | ../rtl/axis_master.v
7 | ../rtl/axis_slave.v
8 | // ../rtl/axis_switch.v
9 | ../rtl/sw_caravel.v
10 | ../rtl/config_ctrl.v
11 | ../rtl/fsic_clkrst.v
12 | ../rtl/fsic_clock.v
13 | ../rtl/fsic_coreclk_phase_cnt.v
14 | ../rtl/fsic_io_serdes_rx.v
15 | ../rtl/fsic.v
16 | ../rtl/io_serdes.v
17 | ../rtl/logic_anlz.dummy_io.v
18 | ../rtl/logic_anlz.dummy_io.vd
19 | // ../rtl/mprj_io.dummy_io.v
20 | // ../rtl/mprj_io.dummy_io.vd
21 | ../rtl/mprj_io.v
22 | ../rtl/user_subsys.all.v
23 | ../rtl/user_prj0.v
24 | ../rtl/user_prj1.v
25 | ../rtl/user_prj2.v
26 | ../rtl/user_prj3.v
27 | ../rtl/spram.v
28 | ../rtl/concat_EdgeDetect_Top_fsic.v

```

Simulation result of FSIC

- Simulation result:

```
# 4753425=> soc_wtsdone_read_data_result : send_soc_cfg_read_event
# 4753425=> soc_aa_cfg_read : got_soc_cfg_read_event
# 4753425=>Read soc_mb_interrupt_status [PASS] cfg_read_data_expect_value[0]=0, cfg_read_data_captured[0]=0
# =====
#
# 4753925=> Final result [PASS], check_cnt = 115301, error_cnt = 0
# =====
#
# ** Note: $finish : ../tb_fsic.v(437)
# Time: 4753925 ns Iteration: 0 Instance: /tb_fsic
# qwavedb_dumpvars : Simulation ending at [1 458957704] 0
# End time: 00:52:12 on Apr 04,2024, Elapsed time: 0:02:38
# Errors: 0, Warnings: 3
[ui10590022@ws42 vsim]$
```

Above is the result which we run the first version of EdgeDetect(throughput=6).

Because we've already completed this lab during the winter break, we're well-versed in the entire process. Consequently, our approach in this lab is geared towards optimizing the overall design. Although we can run the correct simulation result on QuestaSim when we check our design on Catapult GUI, the version with throughput=1 fails to accurately simulate results on FSIC. The main issue lies in the structure of the "AA module," which requires additional time during the conversion process, resulting in the data transfer of version with throughput=1 not being fully captured.