Lab2-2 Report (Team 6)

陳揚哲, 電機系, 111061545
葉承泓, 半導體研究學院碩士班(設計部), 112501538
張育碩, 電機系, 110030039

✓ Due date: 2024/4/7 23:59

Answers to the questions in the workbook

1. How you design your work (5 modifications)

(1) Process four pixels per clock cycle.

To process 4 pixels at a time, we need to modify the blocks in the datapath. First, in EdgeDetect.h, we add types " pixelType4x, " " pixelType8x, " and " gradType4x " to store enough information, which corresponds to " 4-pixel values packed, " " 8-pixel values packed, " and " 4 gradient values packed, " respectively.

```
// Define some bit-accurate types to use in this model
typedef uint8                pixelType;   // input pixel is 0-255
typedef uint16               pixelType2x; // two pixels packed
typedef uint32               pixelType4x; // four pixels packed // added
//typedef uint64               pixelType8x;  // eight pixels packed // added
typedef ac_int<64,false>             pixelType8x;
typedef int9                 gradType;    // Derivative is max range -255 to 255
typedef int36                gradType4x;
```

In EdgeDetect.h, the top block, we write it as the workbook suggested. The unspecified things in workbook, such as the ac channel types, are written by ourselves:

```
#pragma hls_design top
class EdgeDetect_Top
{
  //instances
  EdgeDetect_VerDer VerDer_inst;
  EdgeDetect_HorDer HorDer_inst;
  EdgeDetect_MagAng MagAng_inst;

  // Static interconnect channels (FIFOs) between blocks
  ac_channel<gradType4x>       dy_chan;
  ac_channel<gradType4x>       dx_chan;
  ac_channel<Stream_t>      pix_cahn1; // channel for passing input pixels to horizontalDerivative
  ac_channel<Stream_t>      pix_cahn2;

public:
  EdgeDetect_Top() {}

  //--------------------------------------------------------------------------
  // Function: run
  //   Top interface for data in/out of class. Combines vertical and
  //   horizontal derivative and magnitude/angle computation.
  #pragma hls_design interface
  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)
  {
    VerDer_inst.run(dat_in, widthIn, heightIn, pix_cahn1, dy_chan);
    HorDer_inst.run(pix_cahn1, widthIn, heightIn, pix_cahn2, dx_chan);
    MagAng_inst.run(dx_chan, dy_chan, pix_cahn2, widthIn, heightIn, sw_in, crc32_pix_in, crc32_dat_out, dat_out);
  }
};
```

For the first block, EdgeDetect_VerDer.h, the I/O channels are designed to match the top block, and the internal registers are defined as follows:

```
// Line buffers store pixel line history - Mapped to RAM
pixelType8x line_buf0[maxImageWidth/8];
pixelType8x line_buf1[maxImageWidth/8];
pixelType8x rdbuf0_pix, rdbuf1_pix;
pixelType8x wrbuf0_pix, wrbuf1_pix;
Stream_t data_input_buffer;
pixelType4x pixel_input;
pixelType pix2a, pix2b, pix2c, pix2d; // pixel 0 --> now; pixel 1 --> pixel of last row ; pixel 2 --> pixel of last last row
pixelType pix1a, pix1b, pix1c, pix1d;
pixelType pix0a, pix0b, pix0c, pix0d;

gradType pixel_derivative_a, pixel_derivative_b, pixel_derivative_c, pixel_derivative_d;

gradType4x dy_value;
```

Some of them are designed to be pixelType8x type because we need a ping-pong buffer such that we can do READ and WRITE “(as if) the same” buffer at the same time. And because we have 4 pixels input at a time, we need to have 4-gradient packed as output at a time, to prevent data congestion.

The ways of designing this block is referred to the original design in 01_edgedetect, but extended to 4-pixel in order to match the requirement. First, for the inner loop, because it changes to input 4 pixel values at a time, the loop index “x” needs to be +4 every time.

```
// Remove loop upperbounds for RTL code coverage
// Use bit accurate data types on loop iterator
VROW: for (maxHType y = 0; ; y++) { // VROW has one extra iteration to ramp-up window
  #pragma hls_pipeline_init_interval 1
  VCOL: for (maxWType x = 0; ; x=x+4) {
```

Next, read one pack of data, which includes pix, sof, and eol elements. Store pix element into " pixel_input, " and store it into buffer, ready to store into line buffer. However, we cannot read and write the same address of line buffer at the same time unless we use lots of resources to build a special line buffer, so we need to do it in a ping-pong manner, as the following shows:

```
if (y <= heightIn-1) {
  data_input_buffer = dat_in.read(); // Read streaming interface
  pixel_input=data_input_buffer.pix;
}
// Write data cache, write lower 8 on even iterations of COL loop, upper 8 on odd
if ( ((x>>2)&1) == 0 ) {
  wrbuf0_pix.set_slc(0,pixel_input);
} else {
  wrbuf0_pix.set_slc(32,pixel_input);
}
```

Here, we store the values into a temporary buffer for line buffer in a ping-pong manner. Then we store two packs of values at once when x is counted odd, and retrieve the values stored in the line buffers when x is counted even. In this way, we can split the time to read/ write the line buffer, but it has an advantage that it needs double space for an element for storage, resulting in the need for pixelType8x type.

```
// Read line buffers into read buffer caches on even iterations of COL loop
if ( ((x>>2)&1) == 0 ) {
  // vertical window of pixels
  rdbuf1_pix = line_buf1[x/8];
  rdbuf0_pix = line_buf0[x/8];
} else { // Write line buffer caches on odd iterations of COL loop
  line_buf1[x/8] = rdbuf0_pix; // copy previous line
  line_buf0[x/8] = wrbuf0_pix; // store current line
}
```

Because we get 4 pixel values at a time, we should calculate 4 dy’s at one output. Thus we first get the pixel values of current (0), past (1), and past-past (2), of the four consecutive pixels a~d.

```
// Get 8-bit data from read buffer caches, lower 8 on even iterations of COL loop
pix2a = (((x>>2)&1)==0) ? rdbuf1_pix.slc<8>(0) : rdbuf1_pix.slc<8>(32);
pix2b = (((x>>2)&1)==0) ? rdbuf1_pix.slc<8>(8) : rdbuf1_pix.slc<8>(40);
pix2c = (((x>>2)&1)==0) ? rdbuf1_pix.slc<8>(16) : rdbuf1_pix.slc<8>(48);
pix2d = (((x>>2)&1)==0) ? rdbuf1_pix.slc<8>(24) : rdbuf1_pix.slc<8>(56);

pix1a = (((x>>2)&1)==0) ? rdbuf0_pix.slc<8>(0) : rdbuf0_pix.slc<8>(32);
pix1b = (((x>>2)&1)==0) ? rdbuf0_pix.slc<8>(8) : rdbuf0_pix.slc<8>(40);
pix1c = (((x>>2)&1)==0) ? rdbuf0_pix.slc<8>(16) : rdbuf0_pix.slc<8>(48);
pix1d = (((x>>2)&1)==0) ? rdbuf0_pix.slc<8>(24) : rdbuf0_pix.slc<8>(56);

pix0a = pixel_input.slc<8>(0);
pix0b = pixel_input.slc<8>(8);
pix0c = pixel_input.slc<8>(16);
pix0d = pixel_input.slc<8>(24);
```

And we need to set the boundary condition:

```
// Boundary condition processing
if (y == 1) {
  pix2a = pix1a; // top boundary (replicate pix1 up to pix2)
  pix2b = pix1b;
  pix2c = pix1c;
  pix2d = pix1d;
}
if (y == heightIn) {
  pix0a = pix1a; // bottom boundary (replicate pix1 down to pix0)
  pix0b = pix1b;
  pix0c = pix1c;
  pix0d = pix1d;
}
```

The method used in the above figure is referred to the description in Step_by_step_lab2_EdgeDetect.pdf page 11:

Boundary pixels are replicated.

We can then calculate the dy values now:

```
// Calculate derivative
pixel_derivative_a = pix2a*kernel[0] + pix1a*kernel[1] + pix0a*kernel[2];
pixel_derivative_b = pix2b*kernel[0] + pix1b*kernel[1] + pix0b*kernel[2];
pixel_derivative_c = pix2c*kernel[0] + pix1c*kernel[1] + pix0c*kernel[2];
pixel_derivative_d = pix2d*kernel[0] + pix1d*kernel[1] + pix0d*kernel[2];

dy_value.set_slc(0,pixel_derivative_a);
dy_value.set_slc(9,pixel_derivative_b);
dy_value.set_slc(18,pixel_derivative_c);
dy_value.set_slc(27,pixel_derivative_d);
```

Here we calculate dy for four pixels, because we have all we need for deriving them, and in order to prevent dataflow congestion.

Finally, we can output pixel data and dy data to output channel:

```
if (y < heightIn) {
  dat_out.write(data_input_buffer); // Pass thru original data
}
if (y != 0) { // Write streaming interfaces
  dy.write(dy_value); // derivative output
}
```

Because we did not set the loop endpoint condition in for loop, we have to use "break" condition:

```
    // programmable width exit condition
    if (x == maxWType(widthIn-4)) { // cast to maxWType for RTL code coverage
      break;
    }
  }
  // programmable height exit condition
  if (y == heightIn) {
    break;
  }
```

For EdgeDetect_HorDer.h block, the main idea is mostly like that in EdgeDetect_VerDer.h. First, we declare some internal buffer for storage:

```
// pixel buffers store pixel history
pixelType pix_buf5 = 0;
pixelType pix_buf4 = 0;
pixelType pix_buf3 = 0;
pixelType pix_buf2 = 0;
pixelType pix_buf1 = 0;

pixelType pix5 = 0;
pixelType pix4 = 0;
pixelType pix3 = 0;
pixelType pix2 = 0;
pixelType pix1 = 0;
pixelType pix0 = 0;

Stream_t data_input_buffer;
pixelType4x pixel_input;

gradType  pixel_derivative_a, pixel_derivative_b, pixel_derivative_c, pixel_derivative_d;

gradType4x dx_value;
```

Then we loop over the overall framewith a step of 4 in x direction, and get the pixel values either from internal buffers or from input channel:

```
HROW: for (maxHType y = 0; ; y++) {
  #pragma hls_pipeline_init_interval 1
  HCOL: for (maxWType x = 0; ; x=x+4) { // HCOL has one extra iteration to ramp-up window
    pix5 = pix_buf5;
    pix4 = pix_buf4;
    pix3 = pix_buf3;
    pix2 = pix_buf2;
    pix1 = pix_buf1;
    if (x <= widthIn-1) {
      data_input_buffer = dat_in.read(); // Read streaming interface
      pixel_input=data_input_buffer.pix;
    }
    if (x == widthIn) {
      pix0=pix1;
    } else {
      pix0=pixel_input.slc<8>(0);
    }
```

Write some boundary condition as Step_by_step_lab2_EdgeDetect.pdf indicates:

```
if (x == 0) {
  pix_buf5 = pix0;
} else {
  pix_buf5 = pix1;
}
pix_buf4 = pix0;
pix_buf3 = pixel_input.slc<8>(8);
pix_buf2 = pixel_input.slc<8>(16);
pix_buf1 = pixel_input.slc<8>(24);
```

Calculate dx:

```
// Calculate derivative
pixel_derivative_a = pix5*kernel[0] + pix4*kernel[1] + pix3*kernel[2];
pixel_derivative_b = pix4*kernel[0] + pix3*kernel[1] + pix2*kernel[2];
pixel_derivative_c = pix3*kernel[0] + pix2*kernel[1] + pix1*kernel[2];
pixel_derivative_d = pix2*kernel[0] + pix1*kernel[1] + pix0*kernel[2];

dx_value.set_slc(0,pixel_derivative_a);
dx_value.set_slc(9,pixel_derivative_b);
dx_value.set_slc(18,pixel_derivative_c);
dx_value.set_slc(27,pixel_derivative_d);
```

Output and break conditions:

```
  if (x < widthIn) {
    dat_out.write(data_input_buffer); // Pass through original data
  }
  if (x != 0) { // Write streaming interface
    dx.write(dx_value); // derivative out
  }
  // programmable width exit condition
  if ( x == widthIn) {
    break;
  }
}
// programmable height exit condition
if (y == maxHType(heightIn-1)) { // cast to maxHType for RTL code coverage
  break;
}
```

For EdgeDetect_MagAng.h, we split dx and dy of each pixel:

```
dx = dx_in.read();
dx1=dx.slc<9>(0);
dx2=dx.slc<9>(9);
dx3=dx.slc<9>(18);
dx4=dx.slc<9>(27);
//printf("HLS: %08x\n", (signed int) dx1);
//printf("HLS: %08x\n", (signed int) dx2);
//printf("HLS: %08x\n", (signed int) dx3);
//printf("HLS: %08x\n", (signed int) dx4);
dy = dy_in.read();
dy1=dy.slc<9>(0);
dy2=dy.slc<9>(9);
dy3=dy.slc<9>(18);
dy4=dy.slc<9>(27);
```

Then calculate SAD of the 4 pixels:

```
if (dx1<0) {
  dx1_abs = -dx1; // int9 --> uint8 (pixelType)
} else {
  dx1_abs = dx1;
}
if (dx2<0) {
  dx2_abs = -dx2;
} else {
  dx2_abs = dx2;
}
if (dx3<0) {
  dx3_abs = -dx3;
} else {
  dx3_abs = dx3;
}
if (dx4<0) {
  dx4_abs = -dx4;
} else {
  dx4_abs = dx4;
}
```

```
if (dy1<0) {
  dy1_abs = -dy1;
} else {
  dy1_abs = dy1;
}
if (dy2<0) {
  dy2_abs = -dy2;
} else {
  dy2_abs = dy2;
}
if (dy3<0) {
  dy3_abs = -dy3;
} else {
  dy3_abs = dy3;
}
if (dy4<0) {
  dy4_abs = -dy4;
} else {
  dy4_abs = dy4;
}

sum1 = dx1_abs + dy1_abs;
sum2 = dx2_abs + dy2_abs;
sum3 = dx3_abs + dy3_abs;
sum4 = dx4_abs + dy4_abs;
```

Output magnitude of 4 pixels at a time:

```
magn_value.pix.set_slc(0,sum1);
magn_value.pix.set_slc(8,sum2);
magn_value.pix.set_slc(16,sum3);
magn_value.pix.set_slc(24,sum4);

if ((x==0)&&(y==0)) {
  magn_value.sof=1;
} else {
  magn_value.sof=0;
}

if (x == maxWType(widthIn-4)) {
  magn_value.eol=1;
} else {
  magn_value.eol=0;
}

data_input_buffer = dat_in.read(); // Read streaming interface
pixel_input=data_input_buffer.pix;
```

Calculate crc32 for pixel and magnitude, and break conditions:

```
  //crc32 for pix_in
  //printf("HLS: %08x\n", (unsigned int)  crc32_pix_in);
  //printf("HLS: %08x\n", (unsigned int)  pixel_input);
  crc32_pix_in = calc_crc32<32>(crc32_pix_in, pixel_input);


  //crc32 for magn
  //printf("HLS: %08x\n", (unsigned int)  crc32_dat_out);
  //printf("HLS: %08x\n", (unsigned int)  magn_value.pix);
  crc32_dat_out = calc_crc32<32>(crc32_dat_out, magn_value.pix);


  if (sw_in==0) {
    magn.write(data_input_buffer);
  } else {
    magn.write(magn_value);
  }


  if (x == maxWType(widthIn-4)) { // cast to maxWType for RTL code coverage
    //printf("!!!!!\n");
    break;
  }
}
// programmable height exit condition
if (y == maxHType(heightIn-1)) { // cast to maxHType for RTL code coverage
  //printf("!!!!!\n");
  //printf("!! %08x\n", (unsigned int)  crc32_pix_in);
  crc32_pix_in = ~crc32_pix_in;
  crc32_dat_out = ~crc32_dat_out;
  //printf("!! %08x\n", (unsigned int)  crc32_pix_in);
  break;
}
```

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

In example code, we will use the square root for our magnitude calculation in Edgedetect_MagAng.h file. The code is shown below.

```
MROW: for (maxHType y = 0; ; y++) {
  #pragma hls_pipeline_init_interval 1
  MCOL: for (maxWType x = 0; ; x++) {
    dx = dx_in.read();
    dy = dy_in.read();
    dx_sq = dx * dx;
    dy_sq = dy * dy;
    sum = dx_sq + dy_sq;
    // Catapult's math library piecewise linear implementation of sqrt and atan2
    ac_math::ac_sqrt_pwl(sum,sq_rt);
    magn.write(sq_rt.to_uint());
    ac_math::ac_atan2_cordic((ac_fixed<9,9>)dy, (ac_fixed<9,9>) dx, at);
    angle.write(at);
    // programmable width exit condition
    if (x == maxWType(widthIn-1)) { // cast to maxWType for RTL code coverage
      break;
    }
  }
```

The code shows the magnitude is calculate by counting out the square sum (dx^2 + dy^2) first, then calculate the root value.

To change into SAD, we need exchange the dx^2 and dy^2 by the absolute value of dx and dy. dx and dy are both 9 bits parameter (from –255 to 255), if we use SAD, the effective bit number is only 8. Therefore, we define abs_sum_clip which is 8 bits and unsigned. Besides, the dx and dy throughput are 4 times of the example code, so we need add the for loop and let catapult to perform unroll modification.

Note: the AC_TRN and AC_SAT are quantization mode and overflow mode, because we have made the abs_dx and abs_dy be the unsigned value. To avoid the overflow, we use as the overflow mode. If data is overflow, it would be viewed as closet of MIN and MAX.

```
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);
}
```

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

We use the sample code of crc32, from the function definition, we can understand how to input the data. Since crc32 is a cyclic calculation, we add a parameter crc32_pix_in_tmp and crc32_dat_out_tmp, which is used in input image and output magnitude. The input pix will be used in counting out the crc32_pix_in, and the output magnitude will be used in counting out the crc32_dat_out_tmp.

```
uint32 crc32_pix_in_tmp = 0XFFFFFFFF;
uint32 crc32_dat_out_tmp = 0XFFFFFFFF;
```

```
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;
  }
```

```
      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);

      // programmable width exit condition
      if (x4 == maxWType(widthIn/4-1)) { // cast to maxWType for RTL code coverage
        break;
      }
    }
    // programmable height exit condition
    if (y == maxHType(heightIn-1)) { // cast to maxHType for RTL code coverage
      break;
    }
  }

  crc32_pix_in  = ~crc32_pix_in_tmp;
  crc32_dat_out = ~crc32_dat_out_tmp;
```

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

To select the output source from input image or calculated magnitude, we add the input signal (sw_in) as the mux. First, we should add the pix_channel to transmit the input image to magnitude module. Then we will decide what the data should be written into output channel. Finally, we use the sw_in to choose whether the output source should be input image or calculated magnitude.

```
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);
void CCS_BLOCK(run)(ac_channel<gradType4x>  &dx_in,
                    ac_channel<gradType4x>  &dy_in,
                    ac_channel<pixelType4x> &pix_in,
                    maxWType                &widthIn,
                    maxHType                &heightIn,
                    bool                    &sw_in,
                    uint32                  &crc32_pix_in,
                    uint32                  &crc32_dat_out,
                    ac_channel<Stream_t>    &dat_out)
{
  dx = dx_in.read();
  dy = dy_in.read();
  pix = pix_in.read();
 if (!sw_in)
   magn_out = pix;
 else
   magn_out = magn;

 dat.pix = magn_out;
 dat.sof = (x4==0 && y==0);
 dat.eol = (x4== maxWType(widthIn/4-1));

 dat_out.write(dat);
```

(5) Remove the angle calculation.

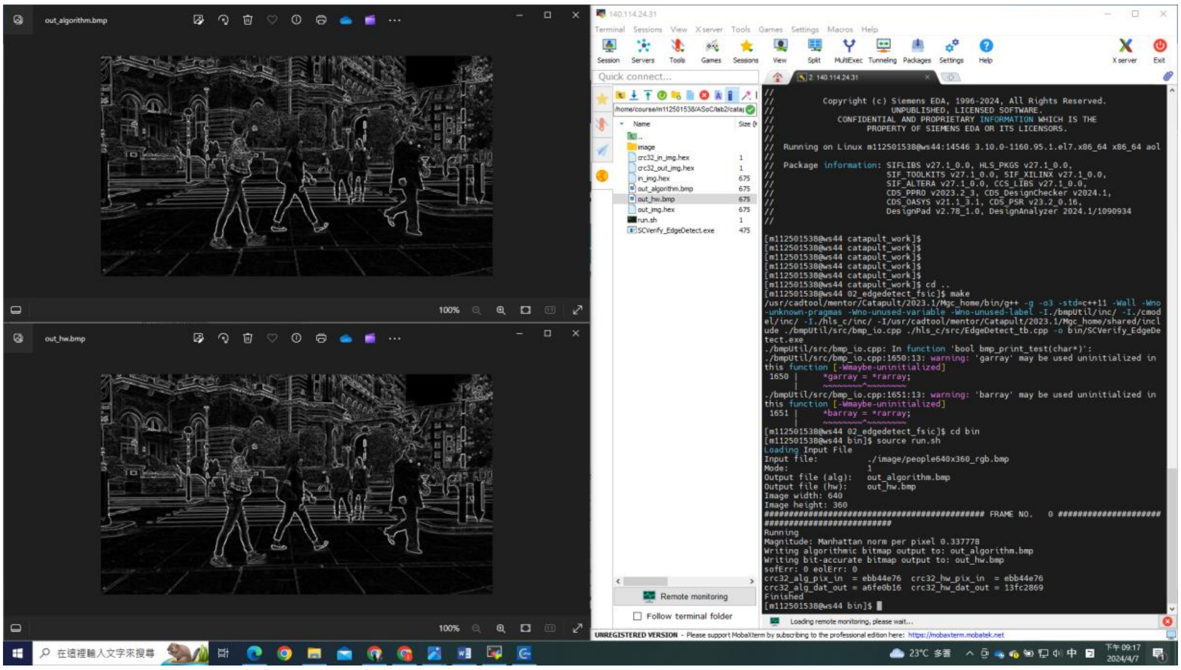
To remove the angle calculation, we should remove the atan calculation in the sample code. We directly remove this calculation and output parameter (angle).

```
ac_math::ac_sqrt_pwl(sum,sq_rt);
magn.write(sq_rt.to_uint());
ac_math::ac_atan2_cordic((ac_fixed<9,9>)dy, (ac_fixed<9,9>) dx, at);
angle.write(at);
```

2. What's the test result of catapult design(C design checker, testbench)

(1) Testbench

We use " make → cd bin → source run.sh " to do pre-HLS simulation in command line. By doing so, it will run the HLS as well as the reference C/C++ model, and compare their output data, printing the results on screen. Besides, it will generate two .bmp pictures: one for HLS result and the other for reference C/C++ result. The final result shown on screen and the output .bmp files are screenshotted together as follows:



Looking at the pictures, we almost cannot differentiate them. It looks like the function is correct! From another point of view, the result printed on command line shows that the sofError is 0, the eolError is

0, and the crc32 result for pixel datapath is the same between HLS and reference model, which means that the pixel is passing through the blocks without any error! As for the crc32 values for data_out （calculated magnitude), the value is different from reference model. The possible reason is because the different bit number we chose in HLS code, in order to make the output channel as Stream_t. (This results in the fact that in C/C++ model, we use “int” to record the calculated number; but in HLS code, we need to record it within 8 bits, in order to match pix as pixelType in Stream_t struct.) The reported " Magnitude: Manhattan norm per pixel " value is 0.337778, a small -enough number to show that our method to deal with the problem about 4 input pixel data at the same read cycle works!

(2) C design checker

在 Catapult 軟體中執行 CDesignChecker 中的 Check Design，結果如下：

```
Goto line...
  STF                          STF - Funcs with statics called multiple times      CHECKED
  SUD                          SUD - Suboptimal Use of Divide and Modulus Operator CHECKED
  UMR                          UMR - Uninitialized Memory Read                     CHECKED

FATAL                                                Violated     Waived  Undecided
------------------------------------------------------ ---------- ---------- ----------

ERROR                                                Violated     Waived  Undecided
------------------------------------------------------ ---------- ---------- ----------
ABR - Array Bounds Read                                     2          0          0
ABW - Array Bounds Write                                    2          0          0
AOB - Arithmetic Operator with Boolean                      0          0          0
CAS - Incomplete Switch-Case                                0          0          0
DBZ - Divide By Zero                                        0          0          0
ISE - Illegal Shift Error                                   0          0          0
OVL - Overflow/Underflow                                   15          0          2
RRT - Reset referenced in thread                            0          0          0
UMR - Uninitialized Memory Read                             4          0          0

WARNING                                              Violated     Waived  Undecided
------------------------------------------------------ ---------- ---------- ----------
ACC - Accumulator of native C type                          0          0          0
ACS - Accumulator of saturated type                         0          0          0
AIC - Assignment used Instead of Comparison                 0          0          0
ALS - Ac_int Left Shift check                               0          0          0
AWE - Assignments Without Effect                            0          0          0
CBU - Conditional break in Unrolled Loop                    0          0          0
CCC - Static constant comparison                            0          0          0
CGR - Conditional Guard in Rolled Loop                      0          0          0
CIA - Comparison Instead of Assignment                      0          0          0
CNS - Constant condition of if/switch                       0          0          0
CWB - Case Without Break                                    0          0          0
DIU - Dynamic Index in Unrolled Loop                        0          0          0
FVI - For Loop with Variable Iterations                     6          0          0
FXD - Mixed fixed and non-fixed datatypes                   0          0          0
MDB - Missing Default Branch                                0          0          0
NCO - No Contribution to Output                             2          0          0
OSA - Optimal Size Accumulator                              0          0          0
PDD - Platform dependent datatype (long)                    0          0          0
RIU - Rolled loop Inside Unrolled loop                      0          0          0
SAT - Sub-optimal Adder Tree                                0          0          0
SUD - Suboptimal Use of Divide and Modulus Operator         0          0          0
```

由上圖可看出此 design 並無 fatal 的問題！而 Error 的部分主要來源應該是在計算 magnitude 時有進行加的動作，再加上為了使 output 能維持在 Stream_t type 必須使 sum 維持在 8 bits，而會導致 overflow 的發生。在拿去 HLS 合成成 RTL 的最終的 version 中，我們透過第 2.(2)點所說明的 " AC_SAT " 的 mode，來修正這個 error。

3. How to integrate your design in FSIC

a. Generate the RTL *concat_EdgeDetect_Top.v* IP under the systhesis folder in the Catapult.
b. Copy the content of the RTL file *concat_EdgeDetect_Top.v* as *concat_EdgeDetect_Top_fsic.v* we uesd in the previous lab
c. Put it in *03_fsic_prj/dsn/rtl* this folder
d. Copy the user_prj0.v from the previous lab since the user_prj0.v in lab_1 is for edge-detect
e. Add 2 sparm in *user_prj0.v*

4. What's the simulation result of FSIC

```
valid=1, fpga_axi_wready=1
             4781305=> soc_is_cfg_write : wbs_adr=30003000, wbs_sel=0001, wbs_wdata=00000003
             4781305=> soc txen_ctl=1
             4781305=> fpga txen_ctl=1
Enable interrupt, set aa_regs offset 0, bit 0 = 1
             4782025=> soc_aa_cfg_write : wbs_adr=30002100, wbs_sel=1111, wbs_wdata=00000001
             4782625=> soc_aa_cfg_read : wbs_adr=30002100, wbs_sel=1111
             4782625=> soc wishbone read data result : send soc_cfg_read_event
             4782625=> soc_aa_cfg_read : got soc_cfg_read_event
             4782625=> test007_aa_internal_soc_mb_interrupt_en [PASS] cfg_read_data_expect_value=00000001, cfg_read_data_captured=00000001
Read interrupt status, aa_regs offset 4, bit 0 should be 0 by default
             4783225=> soc_aa_cfg_read : wbs_adr=30002104, wbs_sel=1111
             4783225=> soc wishbone read data result : send soc_cfg_read_event
             4783225=> soc_aa_cfg_read : got soc_cfg_read_event
             4783225=> test007_aa_internal_soc_mb_interrupt_en [PASS] cfg_read_data_expect_value[0]=0, cfg_read_data_captured[0]=0
             4783425=> test007_fpga_mail_box_write done
Read mb_regs offset 0
             4784305=> soc_aa_cfg_read : wbs_adr=30002000, wbs_sel=1111
             4784305=> soc wishbone read data result : send soc_cfg_read_event
             4784305=> soc_aa_cfg_read : got soc_cfg_read_event
             4784305=> Result: mb_regs offset 0 [PASS] cfg_read_data_expect_value=11111111, cfg_read_data_captured=11111111
Check interrupt status, read aa_regs offset 4, bit 0
             4784985=> soc_aa_cfg_read : wbs_adr=30002104, wbs_sel=1111
             4784985=> soc wishbone read data result : send soc_cfg_read_event
             4784985=> soc_aa_cfg_read : got soc_cfg_read_event
             4784985=> Read soc_mb_interrupt_status [PASS] cfg_read_data_expect_value[0]=1, cfg_read_data_captured[0]=1
Clear interrupt status, write aa_regs offset 4, bit 0 = 1
             4785305=> soc_aa_cfg_write : wbs_adr=30002104, wbs_sel=1111, wbs_wdata=00000001
             4785905=> soc_aa_cfg_read : wbs_adr=30002104, wbs_sel=1111
             4785905=> soc wishbone read data result : send soc_cfg_read_event
             4785905=> soc_aa_cfg_read : got soc_cfg_read_event
             4785905=>Read soc_mb_interrupt_status [PASS] cfg_read_data_expect_value[0]=0, cfg_read_data_captured[0]=0
===============================================================================================
===============================================================================================
===============================================================================================
             4786405=> Final result [PASS], check_cnt = 115301, error_cnt = 0000
===============================================================================================
===============================================================================================
===============================================================================================
$finish called at time : 4786405 ns : File "/home/ubuntu/SoC_Design/caravel-soc_fpga-lab/fsic-sim/fsic_fpga/rtl/user/testbench/tb_fsic.v" Line 444
run: Time (s): cpu = 00:04:43 ; elapsed = 00:05:14 . Memory (MB): peak = 2851.340 ; gain = 0.000 ; free physical = 145 ; free virtual = 8881
## quit
INFO: xsimkernel Simulation Memory Usage: 126992 KB (Peak: 170836 KB), Simulation CPU Usage: 143850 ms
ubuntu@ubuntu2004:~/SoC_Design/caravel-soc_fpga-lab/fsic-sim/fsic_fpga/rtl/user/testbench/tc$
```

After conducting the process in the 3. How to integrate your design in FSIC, we can run the testbench (the testbench is the sample code from lab1) to check the result.

Then, we pass the testbench in this lab.

- **Github link for our work about this lab**

  https://github.com/ZheChen-Bill/ASoC_catapult

  在上述 Github 連結中有關於這次 lab 中所需繳交的檔案。