111064503吳紹齊 HW4

https://github.com/cowboy35927/ESL/tree/main/Hw4

Cross-compile Median and Weighted Mean Filter to RISC-V VP platform:

```
void write_data_to_ACC(char* ADDR, unsigned char* buffer, int len){
  if(_is_using_dma){
    // Using DMA
    *DMA_SRC_ADDR = (uint32_t)(buffer);
    *DMA_DST_ADDR = (uint32_t)(ADDR);
    *DMA_LEN_ADDR = len;
    *DMA_OP_ADDR  = DMA_OP_MEMCPY;
  }else{
    // Directly Send
    memcpy(ADDR, buffer, sizeof(unsigned char)*len);
  }
}
void read_data_from_ACC(char* ADDR, unsigned char* buffer, int len){
  if(_is_using_dma){
    // Using DMA
    *DMA_SRC_ADDR = (uint32_t)(ADDR);
    *DMA_DST_ADDR = (uint32_t)(buffer);
    *DMA_LEN_ADDR = len;
    *DMA_OP_ADDR  = DMA_OP_MEMCPY;
  }else{
    // Directly Read
    memcpy(buffer, ADDR, sizeof(unsigned char)*len);
  }
}

int main(int argc, char *argv[]) {
```

一個使用 DMA 執行傳輸，另一個使用處理器移動數據 (memcpy)。

```
  read_data_from_ACC(SOBELFILTER_READ_ADDR, buffer, 4);

  memcpy(data.uc, buffer, 4);
  total = (data).sint;
  *(target_bitmap + bytes_per_pixel * (width * i + j - 2) + 2) =data.uc[0];
  *(target_bitmap + bytes_per_pixel * (width * i + j - 2) + 1) =data.uc[1];
  *(target_bitmap + bytes_per_pixel * (width * i + j - 2) + 0) =data.uc[2];
```

使用write_data_to_ACC寫入pixel，他是使用DMA來進行操作。

```
read_data_from_ACC(SOBELFILTER_READ_ADDR, buffer, 4);

memcpy(data.uc, buffer, 4);
total = (data).sint;
*(target_bitmap + bytes_per_pixel * (width * i + j - 2) + 2) =data.uc[0];
*(target_bitmap + bytes_per_pixel * (width * i + j - 2) + 1) =data.uc[1];
*(target_bitmap + bytes_per_pixel * (width * i + j - 2) + 0) =data.uc[2];
```

使用read_data_to_ACC讀pixel，是使用DMA來進行操作，再用memcpy複製資料。

Filter.h

```
sc_dt::sc_uint<12> mean_r;
sc_dt::sc_uint<12> mean_g;
sc_dt::sc_uint<12> mean_b;
sc_dt::sc_uint<32> total ;
sc_dt::sc_uint<8> o_r;
sc_dt::sc_uint<8> o_g;
sc_dt::sc_uint<8> o_b;
```

上圖是定義參數，我都使用sc_uint

```
for (unsigned int v = 0; v < MASK_Y; ++v)
{
  for (unsigned int u = 0; u < MASK_X; ++u)
  {
    val[0][v * 3 + u] = i_r.read();
    val[1][v * 3 + u] = i_g.read();
    val[2][v * 3 + u] = i_b.read();
    //cout << "Now at " << sc_time_stamp() << "
    //wait(1 * CLOCK_PERIOD, SC_NS);
  }
}
```

讀取pixel點

```
for(int id=0;id<3;id++)
{
  for(int i = 8; i > 0; i--)
  {
      for(int j = 0; j <= i-1; j++)
      {
          if( val[id][j] > val[id][j+1])
          {
              swap(val[id][j], val[id][j+1]);
              //temp = val[id][j];
              //val[id][j] = val[id][j+1];
              //val[id][j+1] = temp;
          }
      }
  }
}
if (i < 5)
{
  buffer[0][i * 3] = val[0][4];
  buffer[1][i * 3] = val[1][4];
  buffer[2][i * 3] = val[2][4];
}
else if (i >= 5 && i < 10)
{
  buffer[0][(i - 5) * 3 + 1] = val[0][4];
  buffer[1][(i - 5) * 3 + 1] = val[1][4];
  buffer[2][(i - 5) * 3 + 1] = val[2][4];
}
else
{
  buffer[0][(i - 10) * 3 + 2] = val[0][4];
  buffer[1][(i - 10) * 3 + 2] = val[1][4];
  buffer[2][(i - 10) * 3 + 2] = val[2][4];
}
```

這邊是median filter

```
unsigned char arr_g1 [9];
unsigned char arr_b1 [9];
for(int i=0;i<3;i++){
  for(int j=0;j<9;j++){
    if(i==0){
      arr_r1[j]=buffer[i][j];
    }else if(i==1){
      arr_g1[j]=buffer[i][j];
    }else{
      arr_b1[j]=buffer[i][j];
    }
  }
}

//mid_r = 0, mid_g = 0, mid_b = 0;
for (int i = 0; i < MASK_Y * MASK_X; i++)
{
  if (i == 4)
  {
    mean_r = mean_r + (arr_r1[i]<<1) / 10;
    mean_g = mean_g + (arr_g1[i]<<1) / 10;
    mean_b = mean_b + (arr_b1[i]<<1) / 10;
  }
  else
  {
    mean_r = mean_r + arr_r1[i] / 10;
    mean_g = mean_g + arr_g1[i] / 10;
    mean_b = mean_b + arr_b1[i] / 10;
  }
}

o_r=mean_r;
o_g=mean_g;
o_b=mean_b;

//int total = 0;
//total = (mid_r<<16) + (mid_g<<8) + mid_b ;
total=(0, o_b, o_g, o_r);
o_result.write(total);
```

這裡做mean filter這裡做mean filter

```
switch (cmd) {
  case tlm::TLM_READ_COMMAND:
    // cout << "READ" << endl;
    switch (addr) {
      case SOBEL_FILTER_RESULT_ADDR:
        buffer.uint = o_result.read();
        break;
      default:
        std::cerr << "READ Error! SobelFilter::blocking_transport: address 0x"
                  << std::setfill('0') << std::setw(8) << std::hex << addr
                  << std::dec << " is not valid" << std::endl;
      }
    data_ptr[0] = buffer.uc[0];
    data_ptr[1] = buffer.uc[1];
    data_ptr[2] = buffer.uc[2];
    data_ptr[3] = buffer.uc[3];
    break;
  case tlm::TLM_WRITE_COMMAND:
    // cout << "WRITE" << endl;
    switch (addr) {
      case SOBEL_FILTER_R_ADDR:
        i_r.write(data_ptr[0]);
        i_g.write(data_ptr[1]);
        i_b.write(data_ptr[2]);
        break;
      default:
        std::cerr << "WRITE Error! SobelFilter::blocking_transport: address 0x"
                  << std::setfill('0') << std::setw(8) << std::hex << addr
                  << std::dec << " is not valid" << std::endl;
    }
    break;
  case tlm::TLM_IGNORE_COMMAND:
    payload.set_response_status(tlm::TLM_GENERIC_ERROR_RESPONSE);
    return;
  default:
    payload.set_response_status(tlm::TLM_GENERIC_ERROR_RESPONSE);
    return;
  }
  payload.set_response_status(tlm::TLM_OK_RESPONSE); // Always OK
```

上圖就是當 a ddress 是 SOBEL_FILTER_R_ADDR 會執行
tlm::TLM_WRITE_COMMAND 的指令，會將 i _r 、 i_g 、 i_b 分別寫入資料。
read_from_socket 函式會 我們設置了 payload 。 寫入
SOBEL_FILTER_R_ADDR 地址。

main.cpp

```
addr_t mem_size = 1024 * 1024 * 32;  // 32 MB ram, to place it before the CLINT and run the base examples (assume
                                     // memory start at zero) without modifications
addr_t mem_start_addr = 0x00000000;
addr_t mem_end_addr = mem_start_addr + mem_size - 1;
addr_t clint_start_addr = 0x02000000;
addr_t clint_end_addr = 0x0200ffff;
addr_t sys_start_addr = 0x02010000;
addr_t sys_end_addr = 0x020103ff;
addr_t term_start_addr = 0x20000000;
addr_t term_end_addr = term_start_addr + 16;
addr_t ethernet_start_addr = 0x30000000;
addr_t ethernet_end_addr = ethernet_start_addr + 1500;
addr_t plic_start_addr = 0x40000000;
addr_t plic_end_addr = 0x41000000;
addr_t sensor_start_addr = 0x50000000;
addr_t sensor_end_addr = 0x50001000;
addr_t sensor2_start_addr = 0x50002000;
addr_t sensor2_end_addr = 0x50004000;
addr_t mram_start_addr = 0x60000000;
addr_t mram_size = 0x10000000;
addr_t mram_end_addr = mram_start_addr + mram_size - 1;
addr_t dma_start_addr = 0x70000000;
addr_t dma_end_addr = 0x70001000;
addr_t flash_start_addr = 0x71000000;
addr_t flash_end_addr = flash_start_addr + Flashcontroller::ADDR_SPACE;  // Usually 528 Byte
addr_t display_start_addr = 0x72000000;
addr_t display_end_addr = display_start_addr + Display::addressRange;
addr_t sobelFilter_start_addr = 0x73000000;
addr_t sobelFilter_size = 0x01000000;
addr_t sobelFilter_end_addr = sobelFilter_start_addr + sobelFilter_size - 1;
```

設置記憶體位置

```
// address mapping
bus.ports[0] = new PortMapping(opt.mem_start_addr, opt.mem_end_addr);
bus.ports[1] = new PortMapping(opt.clint_start_addr, opt.clint_end_addr);
bus.ports[2] = new PortMapping(opt.plic_start_addr, opt.plic_end_addr);
bus.ports[3] = new PortMapping(opt.term_start_addr, opt.term_end_addr);
bus.ports[4] = new PortMapping(opt.sensor_start_addr, opt.sensor_end_addr);
bus.ports[5] = new PortMapping(opt.dma_start_addr, opt.dma_end_addr);
bus.ports[6] = new PortMapping(opt.sensor2_start_addr, opt.sensor2_end_addr);
bus.ports[7] = new PortMapping(opt.mram_start_addr, opt.mram_end_addr);
bus.ports[8] = new PortMapping(opt.flash_start_addr, opt.flash_end_addr);
bus.ports[9] = new PortMapping(opt.ethernet_start_addr, opt.ethernet_end_addr);
bus.ports[10] = new PortMapping(opt.display_start_addr, opt.display_end_addr);
bus.ports[11] = new PortMapping(opt.sys_start_addr, opt.sys_end_addr);
bus.ports[12] = new PortMapping(opt.sobelFilter_start_addr, opt.sobelFilter_end_addr);
```

記憶體連接

```
// connect TLM sockets
iss_mem_if.isock.bind(bus.tsocks[0]);
dbg_if.isock.bind(bus.tsocks[2]);

PeripheralWriteConnector dma_connector("SimpleDMA-Connector");  // to respect ISS bus locking
dma_connector.isock.bind(bus.tsocks[1]);
dma.isock.bind(dma_connector.tsock);
dma_connector.bus_lock = bus_lock;

bus.isocks[0].bind(mem.tsock);
bus.isocks[1].bind(clint.tsock);
bus.isocks[2].bind(plic.tsock);
bus.isocks[3].bind(term.tsock);
bus.isocks[4].bind(sensor.tsock);
bus.isocks[5].bind(dma.tsock);
bus.isocks[6].bind(sensor2.tsock);
bus.isocks[7].bind(mram.tsock);
bus.isocks[8].bind(flashController.tsock);
bus.isocks[9].bind(ethernet.tsock);
bus.isocks[10].bind(display.tsock);
bus.isocks[11].bind(sys.tsock);
bus.isocks[12].bind(sobel_filter.tsock);
```

連接TLM

Reult:

```
=[ core : 0 ]==========================
simulation time: 18976030380 ns
zero (x0) =        0
ra   (x1) =    1037a
sp   (x2) =  1ffffec
gp   (x3) =    354a0
tp   (x4) =        0
t0   (x5) =      3e8
t1   (x6) =        0
t2   (x7) =  24205d4
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =        0
a2  (x12) =       12
a3  (x13) =        0
a4  (x14) =      812
a5  (x15) =        0
a6  (x16) =        1
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =    967ce
t5  (x30) =        0
t6  (x31) =        0
pc = 28afc
num-instr = 472175040
```

```
Info: /OSCI/SystemC: Simulation stopped by user.
=[ core : 0 ]==========================
simulation time: 76058528140 ns
zero (x0) =        0
ra   (x1) =    1037a
sp   (x2) =  1ffffec
gp   (x3) =    354a0
tp   (x4) =        0
t0   (x5) =      3e8
t1   (x6) =        0
t2   (x7) = 1c165b40
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =        0
a2  (x12) =       12
a3  (x13) =        0
a4  (x14) =      812
a5  (x15) =        0
a6  (x16) =        1
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =   1b67ce
t5  (x30) =        0
t6  (x31) =        0
pc = 28afc
num-instr = 1891903759
```

左邊的是256*256*的圖片，右邊則是512*512，simulation time相差4合理，num-instr也相差4倍也很合理。