Full Source Code Design (VERILOG)

Contents

Top module:

```
module top(
   //system
   input i_clk50,
   //----------------------------
   input i_pclk,
   input i_vsync,
   input i_href,
   input [7:0] i_data,
   input i_reset,
   output o_clkCam,
   output o_camRes,
   output o_camPowerDown,
   output o_camSioc,
   output o_camSiod,
   //----------------------------
   output [12:0]  sdram_addr,
   output [1:0]   sdram_ba,
   output       sdram_cas_n,
   output       sdram_ras_n,
   output       sdram_we_n,
   output       sdram_cke,
   output       sdram_cs_n,
   output [15:0]  sdram_dq,
   output       sdram_clk,
   output       sdram_maskLow,
   output       sdram_maskHigh,
   //----------------------------
   output led
);
//-------------------Pll conponent---------------
wire i_clk100, i_clk24;
mypll PllBlock(
   .areset(~i_reset),
   .inclk0(i_clk50),
   .c0(i_clk24),
   .c1(i_clk100)
);
//----------------Config camera------------------
wire startConfigCam;
cameraStartConfig controlStartConfigCameraBlock(
   .i_clk(i_clk24),
```

```
   .i_reset(i_reset),
   .o_startConf(startConfigCam)
);
camera_configure #(.CLK_FREQ ( 24000000 )) camera_configure_0
(
          .clk   (i_clk24),
          .start (startConfigCam),
          .sioc  (o_camSioc),
          .siod  (o_camSiod),
          .done  () //dont need to use
);
//--------------------read-cam-------------------
wire [15:0] dataPixelFromCam;
wire sdramBusy;
wire sdramReady;
assign sdramReady = ~sdramBusy;
wire pixelValidCam;
wire [9:0] o_xIndex, o_yIndex;
wire pixelClock;
wire enableWriteFifoMaster;
wire i_get; //signal to start capature frame
wire completeCaptureFrame; //signal notice complete frame
wire processCaptureFrame; //signal notice processing frame
wire [9:0] remainWordFifo;
wire enableReadFifo;
wire [15:0] dataFromFifo;
wire [15:0] dataToSdram;
wire [18:0] addressToSdram;
wire rdClkFifo;
wire enableWrSdram;
wire flagFinish_StCam;
wire startStCam;
assign o_clkCam = i_clk24; //supply clock for camera 24Mhz
assign o_camRes = i_reset;  //reset pin camera /active low
assign o_camPowerDown = 1'd0; //disable power down mode
cameraRead cameraReadBlock(
   .i_pclk(i_pclk),
   .i_vsync(i_vsync),
   .i_href(i_href),
   .i_data(i_data),
   .i_reset(i_reset),
   .o_pixelOut(dataPixelFromCam),
   .o_pixelValid(pixelValidCam),
   .o_xIndex(o_xIndex),
```

```
   .o_yIndex(o_yIndex),
   .o_pixelClk(pixelClock)
);
controlWriteToFifo controlWriteFifoBlock(
   .i_xIndex(o_xIndex),
   .i_yIndex(o_yIndex),
   .i_clk(pixelClock),
   .i_reset(i_reset),
   .i_get(i_get),
   .o_eWriteFifo(enableWriteFifoMaster),
   .o_complete(completeCaptureFrame),
   .o_process(processCaptureFrame)
);
storeCamControl controlStoreToSdramBlock(
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_start(startStCam),
   .i_remainOnFifo(remainWordFifo),
   .i_process(processCaptureFrame),
   .i_sdramReady(sdramReady),
   .i_complete(completeCaptureFrame),
   .i_dataFifo(dataFromFifo),
   .o_get(i_get),
   .o_EnReadFifo(enableReadFifo),
   .o_RdClkFifo(rdClkFifo),
   .o_dataSdram(dataToSdram),
   .o_addressToSdram(addressToSdram),
   .o_wrSdram(enableWrSdram),
   .o_finish(flagFinish_StCam)
);
fifoMem fifoBlock(
   .aclr(~i_reset),
   .data(dataPixelFromCam),
   .rdclk(rdClkFifo),
   .rdreq(enableReadFifo),
   .wrclk(pixelClock),
   .wrreq(enableWriteFifoMaster & pixelValidCam),
   .q(dataFromFifo),
   .rdempty(), //unuse
   .rdusedw(remainWordFifo),
   .wrfull(), //unuse
   .wrusedw() //unuse
);
//------------------SDRAM control----------------------
```

```
wire [15:0] dataWriteToSdram;
wire [18:0] addrWrireToSdram;
wire [18:0] addrFromWriteBackToSdram;
wire [15:0] dataFromWriteBack;
wire wrActiveCam;
wire enableWriteSdram;
wire enableReadSdram;
wire enableWrSdramFromWriteback;
wire [18:0] addrReadToSdram;
wire [15:0] dataReadFromSdram;
wire [23:0] addrWriteToSdram24;
wire [23:0] addrReadToSdram24;
assign addrWriteToSdram24 = {5'd0,addrWrireToSdram};
assign addrReadToSdram24 = {5'd0, addrReadToSdram};
dataSteamingSdramWrite dataStreamingWriteSdram(
   .i_dataA(dataFromWriteBack), //write back //0
   .i_dataB(dataToSdram), //camera // 1
   .i_addrA(addrFromWriteBackToSdram),
   .i_addrB(addressToSdram),
   .i_enableWriteA(enableWrSdramFromWriteback),
   .i_enableWriteB(enableWrSdram),
   .i_sel(wrActiveCam),
   .o_data(dataWriteToSdram),
   .o_addr(addrWrireToSdram),
   .o_enableWrite(enableWriteSdram)
);
sdram_controller sdramControlBlock(
   .wr_addr(addrWriteToSdram24),
   .wr_data(dataWriteToSdram),
   .wr_enable(enableWriteSdram),
   .rd_addr(addrReadToSdram24),
   .rd_data(dataReadFromSdram),
   .rd_ready(), //unuse use busy instant
   .rd_enable(enableReadSdram),
   .busy(sdramBusy),
   .rst_n(i_reset),
   .clk(i_clk100),
   //----------sdram out------------
   .addr(sdram_addr),
   .bank_addr(sdram_ba),
   .data(sdram_dq),
   .clock_enable(sdram_cke),
   .cs_n(sdram_cs_n),
   .ras_n(sdram_ras_n),
```

```
   .cas_n(sdram_cas_n),
   .we_n(sdram_we_n),
   .data_mask_low(sdram_maskLow),
   .data_mask_high(sdram_maskHigh)
);
assign sdram_clk = i_clk100;  //clock supply for SDRAM synchronous with control
module
//----------------------Memory fectching---------------------
wire startFeching;
wire [5:0] opcode;
wire [18:0] baseAddr0, baseAddr1, baseAddr2;
wire [11:0] addrToSourceRam;
wire enableWrSourceRam0, enableWrSourceRam1, enableWrSourceRam2;
wire flagFinish_Fet;

baseAddrFetDecode baseAddressFetchingDescodeBlock(
   .i_opcode(opcode),
   .o_baseAddr0(baseAddr0),
   .o_baseAddr1(baseAddr1),
   .o_baseAddr2(baseAddr2)
);

fetchingControl fetchingControlBlock(
   .i_baseAddr0(baseAddr0),
   .i_baseAddr1(baseAddr1),
   .i_baseAddr2(baseAddr2),
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_sdramReady(sdramReady),
   .i_start(startFeching),
   .o_rdSdram(enableReadSdram),
   .o_addrToSdram(addrReadToSdram),
   .o_finish(flagFinish_Fet),
   .o_wrRam0(enableWrSourceRam0),
   .o_wrRam1(enableWrSourceRam1),
   .o_wrRam2(enableWrSourceRam2),
   .o_addrToRam(addrToSourceRam)
);

//-------------------Source Ram--------------
wire startReadSourceRam;
wire startReadSourceFromAve;
wire startReadSource;
wire rdActiveConvolution;
```

```
wire [89:0] dataForConvFromSourceRam0, dataForConvFromSourceRam1,
dataForConvFromSourceRam2;
wire [107:0] addrReadDataForConvFromSourceRam0,
addrReadDataForConvFromSourceRam1, addrReadDataForConvFromSourceRam2;
wire dataForConvFromSourceRamValid;
wire [107:0] addrReadDataForAveFromSourceRam0,
addrReadDataForAveFromSourceRam1, addrReadDataForAveFromSourceRam2;
wire [107:0] addrReadSource0, addrReadSource1, addrReadSource2;
selAddrSourceRam selAddrSource0(
   .i_sel(rdActiveConvolution),
   .i_addrConv(addrReadDataForConvFromSourceRam0),
   .i_addrAve(addrReadDataForAveFromSourceRam0),
   .i_startReadFromConv(startReadSourceRam),
   .i_startReadFromAve(startReadSourceFromAve),
   .o_addr(addrReadSource0),
   .o_startRead(startReadSource)
);
selAddrSourceRam selAddrSource1(
   .i_sel(rdActiveConvolution),
   .i_addrConv(addrReadDataForConvFromSourceRam1),
   .i_addrAve(addrReadDataForAveFromSourceRam1),
   .i_startReadFromConv(startReadSourceRam),
   .i_startReadFromAve(startReadSourceFromAve),
   .o_addr(addrReadSource1),
   .o_startRead()
);
selAddrSourceRam selAddrSource2(
   .i_sel(rdActiveConvolution),
   .i_addrConv(addrReadDataForConvFromSourceRam2),
   .i_addrAve(addrReadDataForAveFromSourceRam2),
   .i_startReadFromConv(startReadSourceRam),
   .i_startReadFromAve(startReadSourceFromAve),
   .o_addr(addrReadSource2),
   .o_startRead()
);
ramControl sourceRam0Block(
   .i_addrOut(addrReadSource0), //addr read data
   .i_addrIn(addrToSourceRam),
   .i_data(dataReadFromSdram),    // Đầu vào dữ liệu
   .i_wrEnable(enableWrSourceRam0),
   .i_quickGet(1'd0), //dont need to read mode quick
   .i_addrOutQuick(12'd0), //dont need to use quick mode
   .i_clk(i_clk100),
   .i_reset(i_reset),
```

```
    .i_start(startReadSource),
    .o_data(dataForConvFromSourceRam0),    // Đầu ra dữ liệu
    .o_quickData(), //dont use read quick mode
    .o_valid(dataForConvFromSourceRamValid),
    .o_ready() //dont use
);
ramControl sourceRam1Block(
    .i_addrOut(addrReadSource1), //addr read data
    .i_addrIn(addrToSourceRam),
    .i_data(dataReadFromSdram),    // Đầu vào dữ liệu
    .i_wrEnable(enableWrSourceRam1),
    .i_quickGet(1'd0), //dont need to read mode quick
    .i_addrOutQuick(12'd0), //dont need to use quick mode
    .i_clk(i_clk100),
    .i_reset(i_reset),
    .i_start(startReadSource),
    .o_data(dataForConvFromSourceRam1),    // Đầu ra dữ liệu
    .o_quickData(), //dont use read quick mode
    .o_valid(), //only use at source ram 0
    .o_ready() //dont use
);
ramControl sourceRam2Block(
    .i_addrOut(addrReadSource2), //addr read data
    .i_addrIn(addrToSourceRam),
    .i_data(dataReadFromSdram),    // Đầu vào dữ liệu
    .i_wrEnable(enableWrSourceRam2),
    .i_quickGet(1'd0), //dont need to read mode quick
    .i_addrOutQuick(12'd0), //dont need to use quick mode
    .i_clk(i_clk100),
    .i_reset(i_reset),
    .i_start(startReadSource),
    .o_data(dataForConvFromSourceRam2),    // Đầu ra dữ liệu
    .o_quickData(), //dont use read quick mode
    .o_valid(), //only use at source ram 0
    .o_ready() //dont use
);
//---------------------Convolution control--------------
wire startConvolution;
wire opcodeConv;
wire opcodeSelConv;
wire enableWriteDestinationRam;
wire flagFinish_Conv;
wire [11:0] locationAddrConv;
```

```
wire [11:0] addrDestinationRamForConv0, addrDestinationRamForConv1,
addrDestinationRamForConv2;
convolutionControl convControlBlock(
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_start(startConvolution),
   .i_opcode(opcodeConv),
   .i_validRam(dataForConvFromSourceRamValid),
   .o_addrRead0(addrReadDataForConvFromSourceRam0),
   .o_addrRead1(addrReadDataForConvFromSourceRam1),
   .o_addrRead2(addrReadDataForConvFromSourceRam2),
   .o_startRam(startReadSourceRam),
   .o_selRamD0(opcodeSelConv),
   .o_addrWrite0(addrDestinationRamForConv0),
   .o_addrWrite1(addrDestinationRamForConv1),
   .o_addrWrite2(addrDestinationRamForConv2),
   .o_wrEnable(enableWriteDestinationRam),
   .o_finish(flagFinish_Conv),
   .o_localAddr(locationAddrConv)
);
//----------------------Convolution-----------------------------
wire [89:0] weight0, weight1, weight2;
wire [9:0] dataOutConv0, dataOutConv1, dataOutConv2;
wire paddingSel0, paddingSel1, paddingSel2, paddingSel3, paddingSel4, paddingSel5,
paddingSel6, paddingSel7, paddingSel8;
wire [8:0] selPadding;
assign selPadding = {paddingSel8, paddingSel7, paddingSel6, paddingSel5,
paddingSel4, paddingSel3, paddingSel2, paddingSel1, paddingSel0};
wire [89:0] dataToConv0, dataToConv1, dataToConv2;
zeroPadding zeropaddingBlock0(
   .i_sel(selPadding),
   .i_data(dataForConvFromSourceRam0),
   .o_data(dataToConv0)
);
zeroPadding zeropaddingBlock1(
   .i_sel(selPadding),
   .i_data(dataForConvFromSourceRam1),
   .o_data(dataToConv1)
);
zeroPadding zeropaddingBlock2(
   .i_sel(selPadding),
   .i_data(dataForConvFromSourceRam2),
   .o_data(dataToConv2)
);
```

```
conv ConvolutionBlock(
   .i_busData0(dataToConv0),
   .i_busData1(dataToConv1),
   .i_busData2(dataToConv2),
   .i_busWeight0(weight0),
   .i_busWeight1(weight1),
   .i_busWeight2(weight2),
   .i_opcode(opcodeSelConv),
   .o_data0(dataOutConv0),
   .o_data1(dataOutConv1),
   .o_data2(dataOutConv2)
);
paddingControl paddingControlBlock(
   .i_localAddr(locationAddrConv),
   .o_sel0(paddingSel0),
   .o_sel1(paddingSel1),
   .o_sel2(paddingSel2),
   .o_sel3(paddingSel3),
   .o_sel4(paddingSel4),
   .o_sel5(paddingSel5),
   .o_sel6(paddingSel6),
   .o_sel7(paddingSel7),
   .o_sel8(paddingSel8)
);
//----------------------Weight Genneration-------------------------
weightGen weightGennerationBlock(
   .i_opcode(opcode),
   .o_weight0(weight0),
   .o_weight1(weight1),
   .o_weight2(weight2)
);
//-----------------------Average Control--------------------------
wire enableWriteAdd;
wire startAve;
wire resetValueAve;
wire maskAve;
wire flagFinish_Average;
averageControl averageControlBlock(
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_start(startAve),
   .i_validRam(dataForConvFromSourceRamValid),
   .o_addrRead0(addrReadDataForAveFromSourceRam0),
   .o_addrRead1(addrReadDataForAveFromSourceRam1),
```

```
   .o_addrRead2(addrReadDataForAveFromSourceRam2),
   .o_writeEnable(enableWriteAdd),
   .o_resetAverage(resetValueAve),
   .o_mask(maskAve),
   .o_startRam(startReadSourceFromAve),
   .o_finish(flagFinish_Average)
);
//----------------------Average------------------
wire [89:0] dataToAve0, dataToAve1, dataToAve2;
wire [9:0] dataOutAve0, dataOutAve1, dataOutAve2;
mask MaseBlock0(
   .i_mask(maskAve),
   .i_data(dataForConvFromSourceRam0),
   .o_data(dataToAve0)
);
mask MaseBlock1(
   .i_mask(maskAve),
   .i_data(dataForConvFromSourceRam1),
   .o_data(dataToAve1)
);
mask MaseBlock2(
   .i_mask(maskAve),
   .i_data(dataForConvFromSourceRam2),
   .o_data(dataToAve2)
);
average AverageBlock0(
   .i_data(dataToAve0),
   .i_clk(i_clk100),
   .i_reset(resetValueAve),
   .i_writeAdd(enableWriteAdd),
   .o_data(dataOutAve0)
);
average AverageBlock1(
   .i_data(dataToAve1),
   .i_clk(i_clk100),
   .i_reset(resetValueAve),
   .i_writeAdd(enableWriteAdd),
   .o_data(dataOutAve1)
);
average AverageBlock2(
   .i_data(dataToAve2),
   .i_clk(i_clk100),
   .i_reset(resetValueAve),
   .i_writeAdd(enableWriteAdd),
```

```
   .o_data(dataOutAve2)
);
//--------------------Average control write register---------------
wire [15:0] selWriteRegister;
averageWriteControl controlWriteToRegister(
   .i_opcode(opcode),
   .o_selWrite(selWriteRegister)
);
//-------------------------Register Ave----------------------------
wire [159:0] dataOutFromRegisterAve;
registerAverage RegisterStoreAverageBlock(
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_enableWrite(flagFinish_Average),
   .i_selWrite(selWriteRegister),
   .i_data0(dataOutAve0),
   .i_data1(dataOutAve1),
   .i_data2(dataOutAve2),
   .o_ave(dataOutFromRegisterAve)
);
//---------------------Fully Connected-----------------------------
wire enableFullyConnected;
wire [19:0] resultFullyConnected;
fullyConnected fullyConnectedBlock(
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_enable(enableFullyConnected),
   .i_data(dataOutFromRegisterAve),
   .o_result(resultFullyConnected)
);
assign led = resultFullyConnected[19]; //activation function
//----------------------Destination RAM--------------------------
wire quickModeEnable;
wire [18:0] quickModeAddr0, quickModeAddr1, quickModeAddr2;
wire [11:0] quickModeAddr0_12, quickModeAddr1_12, quickModeAddr2_12;
assign quickModeAddr0_12 = quickModeAddr0[11:0];
assign quickModeAddr1_12 = quickModeAddr1[11:0];
assign quickModeAddr2_12 = quickModeAddr2[11:0];
wire [9:0] dataOutQuickMode0, dataOutQuickMode1, dataOutQuickMode2;
ramControl destinationRam0Block(
   .i_addrOut(108'd0), //Dont use this read mode
   .i_addrIn(addrDestinationRamForConv0),
   .i_data(dataOutConv0),    // Đầu vào dữ liệu
   .i_wrEnable(enableWriteDestinationRam),
```

```
    .i_quickGet(quickModeEnable),
    .i_addrOutQuick(quickModeAddr0_12),
    .i_clk(i_clk100),
    .i_reset(i_reset),
    .i_start(1'd0), //dont use this read mode
    .o_data(),    // dont use this read mode
    .o_quickData(dataOutQuickMode0), //dont use read quick mode
    .o_valid(), //dont use this mode
    .o_ready() //dont use
);
ramControl destinationRam1Block(
    .i_addrOut(108'd0), //Dont use this read mode
    .i_addrIn(addrDestinationRamForConv1),
    .i_data(dataOutConv1),    // Đầu vào dữ liệu
    .i_wrEnable(enableWriteDestinationRam),
    .i_quickGet(quickModeEnable),
    .i_addrOutQuick(quickModeAddr1_12),
    .i_clk(i_clk100),
    .i_reset(i_reset),
    .i_start(1'd0), //dont use this read mode
    .o_data(),    // dont use this read mode
    .o_quickData(dataOutQuickMode1), //dont use read quick mode
    .o_valid(), //dont use this mode
    .o_ready() //dont use
);
ramControl destinationRam2Block(
    .i_addrOut(108'd0), //Dont use this read mode
    .i_addrIn(addrDestinationRamForConv2),
    .i_data(dataOutConv2),    // Đầu vào dữ liệu
    .i_wrEnable(enableWriteDestinationRam),
    .i_quickGet(quickModeEnable),
    .i_addrOutQuick(quickModeAddr2_12),
    .i_clk(i_clk100),
    .i_reset(i_reset),
    .i_start(1'd0), //dont use this read mode
    .o_data(),    // dont use this read mode
    .o_quickData(dataOutQuickMode2), //dont use read quick mode
    .o_valid(), //dont use this mode
    .o_ready() //dont use
);
//------------------------Write back-------------------------------
wire [18:0] baseAddrWriteBack0, baseAddrWriteBack1, baseAddrWriteBack2;
wire startWriteBack;
wire [1:0] selDataForWriteBack;
```

```
wire flagFinish_WriteBack;
baseAddrWriteBackDecode writeBackDecodeAddrBlock(
   .i_opcode(opcode),
   .o_baseAddr0(baseAddrWriteBack0),
   .o_baseAddr1(baseAddrWriteBack1),
   .o_baseAddr2(baseAddrWriteBack2)
);
writeBackControl writeBackControlBlock(
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_sdramReady(sdramReady),
   .i_baseAddr0(baseAddrWriteBack0),
   .i_baseAddr1(baseAddrWriteBack1),
   .i_baseAddr2(baseAddrWriteBack2),
   .i_start(startWriteBack),
   .o_addrToRam0(quickModeAddr0),
   .o_addrToRam1(quickModeAddr1),
   .o_addrToRam2(quickModeAddr2),
   .o_quickRam(quickModeEnable),
   .o_addrToSdram(addrFromWriteBackToSdram),
   .o_wrSdram(enableWrSdramFromWriteback),
   .o_selData(selDataForWriteBack),
   .o_finish(flagFinish_WriteBack)
);
selDataForWriteBack selDataForWriteBackBlock(
   .i_sel(selDataForWriteBack),
   .i_data0({5'd0, dataOutQuickMode0}),
   .i_data1({5'd0, dataOutQuickMode1}),
   .i_data2({5'd0, dataOutQuickMode2}),
   .o_data(dataFromWriteBack)
);
//------------------------Master control------------------------------
masterControl masterControlBlock(
   .i_clk(i_clk100),
   .i_reset(i_reset),
   .i_finish_cam(flagFinish_StCam),
   .i_finish_fet(flagFinish_Fet),
   .i_finish_conv(flagFinish_Conv),
   .i_finish_writeBack(flagFinish_WriteBack),
   .i_finish_ave(flagFinish_Average),
   .o_startCam(startStCam),
   .o_startFet(startFeching),
   .o_startConvolution(startConvolution),
   .o_startAve(startAve),
```

```
    .o_startWriteBack(startWriteBack),
    .o_wrActiveCam(wrActiveCam),
    .o_rdActiveConvolution(rdActiveConvolution),
    .o_opConv(opcodeConv),
    .o_opcode(opcode),
    .o_enableFullyConnected(enableFullyConnected)
);
endmodule
```


Camera config

```
module camera_configure
    #(
    parameter CLK_FREQ=25000000
    )
    (
    input wire clk,
    input wire start,
    output wire sioc,
    output wire siod,
    output wire done
    );

    wire [7:0] rom_addr;
    wire [15:0] rom_dout;
    wire [7:0] SCCB_addr;
    wire [7:0] SCCB_data;
    wire SCCB_start;
    wire SCCB_ready;
    wire SCCB_SIOC_oe;
    wire SCCB_SIOD_oe;

    assign sioc = SCCB_SIOC_oe ? 1'b0 : 1'bZ;
    assign siod = SCCB_SIOD_oe ? 1'b0 : 1'bZ;

    OV7670_config_rom rom1(
        .clk(clk),
        .addr(rom_addr),
        .dout(rom_dout)
        );

    OV7670_config #(.CLK_FREQ(CLK_FREQ)) config_1(
```

```
    .clk(clk),
    .SCCB_interface_ready(SCCB_ready),
    .rom_data(rom_dout),
    .start(start),
    .rom_addr(rom_addr),
    .done(done),
    .SCCB_interface_addr(SCCB_addr),
    .SCCB_interface_data(SCCB_data),
    .SCCB_interface_start(SCCB_start)
    );

  SCCB_interface #( .CLK_FREQ(CLK_FREQ)) SCCB1(
    .clk(clk),
    .start(SCCB_start),
    .address(SCCB_addr),
    .data(SCCB_data),
    .ready(SCCB_ready),
    .SIOC_oe(SCCB_SIOC_oe),
    .SIOD_oe(SCCB_SIOD_oe)
    );

endmodule
```


Camera config ROM:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/06/2015 02:41:55 PM
// Design Name:
// Module Name: OV7670_config_rom
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module OV7670_config_rom(
  input wire clk,
  input wire [7:0] addr,
  output reg [15:0] dout
  );
  //FFFF is end of rom, FFF0 is delay
  /*always @(posedge clk) begin
  case(addr)
  0:  dout <= 16'h12_80; //reset
  1:  dout <= 16'hFF_F0; //delay
  2:  dout <= 16'h12_04;//16'h12_08;// COM7,    set RGB color output
  //3: dout <= 16'h4F_B3; //MTX1      all of these are magical matrix coefficients
  //4: dout <= 16'h50_B3; //MTX2
  //5: dout <= 16'h51_00; //MTX3
  //6: dout <= 16'h52_3d; //MTX4
  //7: dout <= 16'h53_A7; //MTX5
  //8: dout <= 16'h54_E4; //MTX6
  //9: dout <= 16'h58_9E; //MTXS
  //10: dout <= 16'h69_00; //GFIX      fix gain control
  //11: dout <= 16'hB0_84;
  3:  dout <= 16'hFF_FF;

  //4:  dout <= 16'h0C_00; // COM3,    default settings
  //5:  dout <= 16'h3E_00; // COM14,   no scaling, normal pclock
  //6:  dout <= 16'h04_00; // COM1,    disable CCIR656
  //7:  dout <= 16'h40_d0; //COM15,    RGB565, full output range
  //8:  dout <= 16'h3a_04; //TSLB      set correct output data sequence (magic)
  //9:  dout <= 16'h14_18; //COM9      MAX AGC value x4
  //10: dout <= 16'h4F_B3; //MTX1      all of these are magical matrix coefficients
  //11: dout <= 16'h50_B3; //MTX2
  //12: dout <= 16'h51_00; //MTX3
  //13: dout <= 16'h52_3d; //MTX4
  //14: dout <= 16'h53_A7; //MTX5
  //15: dout <= 16'h54_E4; //MTX6
  //16: dout <= 16'h58_9E; //MTXS
  //17: dout <= 16'h3D_C0; //COM13     sets gamma enable, does not preserve reserved
bits, may be wrong?
  //18: dout <= 16'h17_14; //HSTART    start high 8 bits
  //19: dout <= 16'h18_02; //HSTOP     stop high 8 bits //these kill the odd colored line
```

```
   //20: dout <= 16'h32_80; //HREF      edge offset
   //21: dout <= 16'h19_03; //VSTART    start high 8 bits
   //22: dout <= 16'h1A_7B; //VSTOP     stop high 8 bits
   //23: dout <= 16'h03_0A; //VREF      vsync edge offset
   //24: dout <= 16'h0F_41; //COM6      reset timings
   //25: dout <= 16'h1E_00; //MVFP      disable mirror / flip //might have magic value of 03
   //26: dout <= 16'h33_0B; //CHLF      //magic value from the internet
   //27: dout <= 16'h3C_78; //COM12     no HREF when VSYNC low
   //28: dout <= 16'h69_00; //GFIX      fix gain control
   //29: dout <= 16'h74_00; //REG74     Digital gain control
   //30: dout <= 16'hB0_84; //RSVD      magic value from the internet *required* for good
color
   //31: dout <= 16'hB1_0c; //ABLC1
   //32: dout <= 16'hB2_0e; //RSVD      more magic internet values
   //33: dout <= 16'hB3_80; //THL_ST
   default: dout <= 16'hFF_FF;       //mark end of ROM
   endcase*/
    always @(posedge clk) begin
   case(addr)
    /*
   0:  dout <= 16'h12_80; //reset
   1:  dout <= 16'hFF_F0; //delay
   2:  dout <= 16'h12_04; // COM7,    set RGB color output
   3:  dout <= 16'h11_80; // CLKRC    internal PLL matches input clock
   4:  dout <= 16'h0C_00; // COM3,    default settings
   5:  dout <= 16'h3E_00; // COM14,   no scaling, normal pclock
   6:  dout <= 16'h04_00; // COM1,    disable CCIR656
   7:  dout <= 16'h40_C0; //COM15,    RGB565, full output range
   8:  dout <= 16'h3a_04; //TSLB      set correct output data sequence (magic)
   9:  dout <= 16'h14_18; //COM9      MAX AGC value x4
   10: dout <= 16'h4F_B3; //MTX1      all of these are magical matrix coefficients
   11: dout <= 16'h50_B3; //MTX2
   12: dout <= 16'h51_00; //MTX3
   13: dout <= 16'h52_3d; //MTX4
   14: dout <= 16'h53_A7; //MTX5
   15: dout <= 16'h54_E4; //MTX6
   16: dout <= 16'h58_9E; //MTXS
   17: dout <= 16'h3D_C0; //COM13     sets gamma enable, does not preserve reserved
bits, may be wrong?
   18: dout <= 16'h17_14; //HSTART    start high 8 bits
   19: dout <= 16'h18_02; //HSTOP     stop high 8 bits //these kill the odd colored line
   //20: dout <= 16'h32_80; //HREF      edge offset
    20: dout <= 16'h8C_02;
   21: dout <= 16'h19_03; //VSTART    start high 8 bits
```

```
   22: dout <= 16'h1A_7B; //VSTOP     stop high 8 bits
   23: dout <= 16'h03_0A; //VREF      vsync edge offset
   24: dout <= 16'h0F_41; //COM6      reset timings
   25: dout <= 16'h1E_00; //MVFP      disable mirror / flip //might have magic value of 03
   26: dout <= 16'h33_0B; //CHLF      //magic value from the internet
   27: dout <= 16'h3C_78; //COM12     no HREF when VSYNC low
   28: dout <= 16'h69_00; //GFIX      fix gain control
   29: dout <= 16'h74_00; //REG74     Digital gain control
   30: dout <= 16'hB0_84; //RSVD      magic value from the internet *required* for good
color
   31: dout <= 16'hB1_0c; //ABLC1
   32: dout <= 16'hB2_0e; //RSVD      more magic internet values
   33: dout <= 16'hB3_80; //THL_ST*/
    0:  dout <= 16'h12_80; //reset
   1:  dout <= 16'hFF_F0; //delay
   2:  dout <= 16'h12_14;//04; // COM7,     set RGB color output
   3:  dout <= 16'h11_80; // CLKRC     internal PLL matches input clock
   4:  dout <= 16'h0C_00; // COM3,     default settings
   5:  dout <= 16'h3E_00; // COM14,    no scaling, normal pclock
   6:  dout <= 16'h04_00; // COM1,     disable CCIR656
   7:  dout <= 16'h40_d0; //COM15,     RGB565, full output range
   8:  dout <= 16'h3a_04; //TSLB      set correct output data sequence (magic)
   9:  dout <= 16'h14_18; //COM9      MAX AGC value x4
   10: dout <= 16'h4F_B3; //MTX1      all of these are magical matrix coefficients
   11: dout <= 16'h50_B3; //MTX2
   12: dout <= 16'h51_00; //MTX3
   13: dout <= 16'h52_3d; //MTX4
   14: dout <= 16'h53_A7; //MTX5
   15: dout <= 16'h54_E4; //MTX6
   16: dout <= 16'h58_9E; //MTXS
   17: dout <= 16'h3D_C0; //COM13     sets gamma enable, does not preserve reserved
bits, may be wrong?
   18: dout <= 16'h17_14; //HSTART    start high 8 bits
   19: dout <= 16'h18_02; //HSTOP     stop high 8 bits //these kill the odd colored line
   20: dout <= 16'h32_80; //HREF      edge offset
   21: dout <= 16'h19_03; //VSTART    start high 8 bits
   22: dout <= 16'h1A_7B; //VSTOP     stop high 8 bits
   23: dout <= 16'h03_0A; //VREF      vsync edge offset
   24: dout <= 16'h0F_41; //COM6      reset timings
   25: dout <= 16'h1E_30; //MVFP      disable mirror / flip //might have magic value of 03
   26: dout <= 16'h33_0B; //CHLF      //magic value from the internet
   27: dout <= 16'h3C_78; //COM12     no HREF when VSYNC low
   28: dout <= 16'h69_00; //GFIX      fix gain control
   29: dout <= 16'h74_00; //REG74     Digital gain control
```

```
   30: dout <= 16'hB0_84; //RSVD      magic value from the internet *required* for good
color
   31: dout <= 16'hB1_0c; //ABLC1
   32: dout <= 16'hB2_0e; //RSVD      more magic internet values
   33: dout <= 16'hB3_80; //THL_ST
   //begin mystery scaling numbers
   34: dout <= 16'h70_3a;
   35: dout <= 16'h71_35;
   36: dout <= 16'h72_11;
   37: dout <= 16'h73_f0;
   38: dout <= 16'ha2_02;
   //gamma curve values
   39: dout <= 16'h7a_20;
   40: dout <= 16'h7b_10;
   41: dout <= 16'h7c_1e;
   42: dout <= 16'h7d_35;
   43: dout <= 16'h7e_5a;
   44: dout <= 16'h7f_69;
   45: dout <= 16'h80_76;
   46: dout <= 16'h81_80;
   47: dout <= 16'h82_88;
   48: dout <= 16'h83_8f;
   49: dout <= 16'h84_96;
   50: dout <= 16'h85_a3;
   51: dout <= 16'h86_af;
   52: dout <= 16'h87_c4;
   53: dout <= 16'h88_d7;
   54: dout <= 16'h89_e8;
   //AGC and AEC
   55: dout <= 16'h13_e0; //COM8, disable AGC / AEC
   56: dout <= 16'h00_00; //set gain reg to 0 for AGC
   57: dout <= 16'h10_00; //set ARCJ reg to 0
   58: dout <= 16'h0d_40; //magic reserved bit for COM4
   59: dout <= 16'h14_18; //COM9, 4x gain + magic bit
   60: dout <= 16'ha5_05; // BD50MAX
   61: dout <= 16'hab_07; //DB60MAX
   62: dout <= 16'h24_95; //AGC upper limit
   63: dout <= 16'h25_33; //AGC lower limit
   64: dout <= 16'h26_e3; //AGC/AEC fast mode op region
   65: dout <= 16'h9f_78; //HAECC1
   66: dout <= 16'ha0_68; //HAECC2
   67: dout <= 16'ha1_03; //magic
   68: dout <= 16'ha6_d8; //HAECC3
   69: dout <= 16'ha7_d8; //HAECC4
```

```
   70: dout <= 16'ha8_f0; //HAECC5
   71: dout <= 16'ha9_90; //HAECC6
   72: dout <= 16'haa_94; //HAECC7
    //73: dout <= 16'h13_e5; //COM8, enable AGC / AEC
   73: dout <= 16'h13_e7; //COM8, enable AGC / AEC
    //
    //74: dout <= 16'h01_96;
    //75: dout <= 16'h02_40;
    //
   default: dout <= 16'hFF_FF;       //mark end of ROM
   endcase
   end
endmodule
```


Camera config control:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/06/2015 02:09:28 PM
// Design Name:
// Module Name: OV7670_config
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module OV7670_config
#(
   parameter CLK_FREQ = 25000000
)
(
```

```
input wire clk,
input wire SCCB_interface_ready,
input wire [15:0] rom_data,
input wire start,
output reg [7:0] rom_addr,
output reg done,
output reg [7:0] SCCB_interface_addr,
output reg [7:0] SCCB_interface_data,
output reg SCCB_interface_start
);

initial begin
  rom_addr = 0;
  done = 0;
  SCCB_interface_addr = 0;
  SCCB_interface_data = 0;
  SCCB_interface_start = 0;
end

localparam FSM_IDLE = 0;
localparam FSM_SEND_CMD = 1;
localparam FSM_DONE = 2;
localparam FSM_TIMER = 3;

reg [2:0] FSM_state = FSM_IDLE;
reg [2:0] FSM_return_state;
reg [31:0] timer = 0;

always@(posedge clk) begin

  case(FSM_state)

    FSM_IDLE: begin
      FSM_state <= start ? FSM_SEND_CMD : FSM_IDLE;
      rom_addr <= 1'h0;
      done <= start ? 1'h0 : done;
    end

    FSM_SEND_CMD: begin
      case(rom_data)
        16'hFFFF: begin //end of ROM
          FSM_state <= FSM_DONE;
        end
```

```
            16'hFFF0: begin //delay state
              timer <= (CLK_FREQ/100); //10 ms delay
              FSM_state <= FSM_TIMER;
              FSM_return_state <= FSM_SEND_CMD;
              rom_addr <= rom_addr + 1'h1;
            end

            default: begin //normal rom commands
              if (SCCB_interface_ready) begin
                FSM_state <= FSM_TIMER;
                FSM_return_state <= FSM_SEND_CMD;
                timer <= 1'h0; //one cycle delay gives ready chance to deassert
                rom_addr <= rom_addr + 1'h1;
                SCCB_interface_addr <= rom_data[15:8];
                SCCB_interface_data <= rom_data[7:0];
                SCCB_interface_start <= 1;
              end
            end
          endcase
        end

        FSM_DONE: begin //signal done
          FSM_state <= FSM_IDLE;
          done <= 1'h1;
        end


        FSM_TIMER: begin //count down and jump to next state
          FSM_state <= (timer == 0) ? FSM_return_state : FSM_TIMER;
          timer <= (timer==0) ? 0 : ( timer - 1'h1 );
          SCCB_interface_start <= 1'h0;
        end
      endcase
    end
endmodule
```

SCCB camera config interface:

```
module SCCB_interface
#(
  parameter CLK_FREQ = 25000000,
  parameter SCCB_FREQ = 100000
)
```

```
(
  input wire clk,
  input wire start,
  input wire [7:0] address,
  input wire [7:0] data,
  output reg ready,
  output reg SIOC_oe,
  output reg SIOD_oe
  );

  localparam CAMERA_ADDR = 8'h42;//8'h84;
  localparam FSM_IDLE = 0;
  localparam FSM_START_SIGNAL = 1;
  localparam FSM_LOAD_BYTE = 2;
  localparam FSM_TX_BYTE_1 = 3;
  localparam FSM_TX_BYTE_2 = 4;
  localparam FSM_TX_BYTE_3 = 5;
  localparam FSM_TX_BYTE_4 = 6;
  localparam FSM_END_SIGNAL_1 = 7;
  localparam FSM_END_SIGNAL_2 = 8;
  localparam FSM_END_SIGNAL_3 = 9;
  localparam FSM_END_SIGNAL_4 = 10;
  localparam FSM_DONE = 11;
  localparam FSM_TIMER = 12;


  initial begin
    SIOC_oe = 0;
    SIOD_oe = 0;
    ready = 1;
  end

  reg [3:0] FSM_state = 0;
  reg [3:0] FSM_return_state = 0;
  reg [31:0] timer = 0;
  reg [7:0] latched_address;
  reg [7:0] latched_data;
  reg [1:0] byte_counter = 0;
  reg [7:0] tx_byte = 0;
  reg [3:0] byte_index = 0;


  always@(posedge clk) begin
```

```
case(FSM_state)

  FSM_IDLE: begin
    byte_index <= 0;
    byte_counter <= 0;
    if (start) begin
      FSM_state <= FSM_START_SIGNAL;
      latched_address <= address;
      latched_data <= data;
      ready <= 0;
    end
    else begin
      ready <= 1;
    end
  end

  FSM_START_SIGNAL: begin //comunication interface start signal, bring SIOD low
    FSM_state <= FSM_TIMER;
    FSM_return_state <= FSM_LOAD_BYTE;
    timer <= (CLK_FREQ/(4*SCCB_FREQ));
    SIOC_oe <= 0;
    SIOD_oe <= 1;
  end

  FSM_LOAD_BYTE: begin //load next byte to be transmitted
    FSM_state <= (byte_counter == 3) ? FSM_END_SIGNAL_1 : FSM_TX_BYTE_1;
    byte_counter <= byte_counter + 1'h1;
    byte_index <= 0; //clear byte index
    case(byte_counter)
      0: tx_byte <= CAMERA_ADDR;
      1: tx_byte <= latched_address;
      2: tx_byte <= latched_data;
      default: tx_byte <= latched_data;
    endcase
  end

  FSM_TX_BYTE_1: begin //bring SIOC low and and delay for next state
    FSM_state <= FSM_TIMER;
    FSM_return_state <= FSM_TX_BYTE_2;
    timer <= (CLK_FREQ/(4*SCCB_FREQ));
    SIOC_oe <= 1;
  end

  FSM_TX_BYTE_2: begin //assign output data,
```

```
            FSM_state <= FSM_TIMER;
            FSM_return_state <= FSM_TX_BYTE_3;
            timer <= (CLK_FREQ/(4*SCCB_FREQ)); //delay for SIOD to stabilize
            SIOD_oe <= (byte_index == 8) ? 1'h0 : ~tx_byte[7]; //allow for 9 cycle ack, output
enable signal is inverting
        end

        FSM_TX_BYTE_3: begin // bring SIOC high
            FSM_state <= FSM_TIMER;
            FSM_return_state <= FSM_TX_BYTE_4;
            timer <= (CLK_FREQ/(2*SCCB_FREQ));
            SIOC_oe <= 0; //output enable is an inverting pulldown
        end

        FSM_TX_BYTE_4: begin //check for end of byte, incriment counter
            FSM_state <= (byte_index == 8) ? FSM_LOAD_BYTE : FSM_TX_BYTE_1;
            tx_byte <= tx_byte<<1; //shift in next data bit
            byte_index <= byte_index + 1'h1;
        end

        FSM_END_SIGNAL_1: begin //state is entered with SIOC high, SIOD high. Start by
bringing SIOC low
            FSM_state <= FSM_TIMER;
            FSM_return_state <= FSM_END_SIGNAL_2;
            timer <= (CLK_FREQ/(4*SCCB_FREQ));
            SIOC_oe <= 1;
        end

        FSM_END_SIGNAL_2: begin // while SIOC is low, bring SIOD low
            FSM_state <= FSM_TIMER;
            FSM_return_state <= FSM_END_SIGNAL_3;
            timer <= (CLK_FREQ/(4*SCCB_FREQ));
            SIOD_oe <= 1;
        end

        FSM_END_SIGNAL_3: begin // bring SIOC high
            FSM_state <= FSM_TIMER;
            FSM_return_state <= FSM_END_SIGNAL_4;
            timer <= (CLK_FREQ/(4*SCCB_FREQ));
            SIOC_oe <= 0;
        end

        FSM_END_SIGNAL_4: begin // bring SIOD high when SIOC is high
            FSM_state <= FSM_TIMER;
```

```
            FSM_return_state <= FSM_DONE;
            timer <= (CLK_FREQ/(4*SCCB_FREQ));
            SIOD_oe <= 0;
          end

          FSM_DONE: begin //add delay between transactions
            FSM_state <= FSM_TIMER;
            FSM_return_state <= FSM_IDLE;
            timer <= (2*CLK_FREQ/(SCCB_FREQ));
            byte_counter <= 0;
          end

          FSM_TIMER: begin //count down and jump to next state
            FSM_state <= (timer == 0) ? FSM_return_state : FSM_TIMER;
            timer <= (timer==0) ? 0 : timer - 1;
          end
        endcase
      end
endmodule
```


Store camera control:

```
module storeCamControl(
    input           i_clk,
    input           i_reset,
    input           i_start,
    input [9:0]       i_remainOnFifo,
    input           i_process,
    input           i_sdramReady,
    input           i_complete, //complete signal of controlWriteFifo
    input [15:0]      i_dataFifo,
    output reg        o_get,
    output reg        o_EnReadFifo,
    output reg        o_RdClkFifo,
    output    [15:0]  o_dataSdram,
    output reg [18:0]  o_addressToSdram,
    output reg        o_wrSdram,
    output reg        o_finish
);

localparam [3:0]    idle = 4'd0,
                start = 4'd1,
```

```
            suspend = 4'd2,
            readFifo0 = 4'd3,
            readFifo1 = 4'd4,
            setRed = 4'd5,
            wait0 = 4'd6,
            setGreen = 4'd7,
            wait1 = 4'd8,
            setBlue = 4'd9,
            wait2 = 4'd10,
            update = 4'd11,
            finish = 4'd12;

reg [3:0] state_q, state_d;
reg [18:0] addrLocal_q, addrLocal_d;
reg [7:0] dataToSdram8;

always @(posedge i_clk or negedge i_reset) begin    //sequential circuit update state,
addr local
  if(!i_reset) begin
    state_q <= idle;
    addrLocal_q <= 19'd0;
  end else begin
    state_q <= state_d;
    addrLocal_q <= addrLocal_d;
  end
end

always @(*) begin
  case (state_q)
    idle: begin
      if(i_start & i_reset) begin
        state_d = start;
      end else begin
        state_d = idle;
      end
      o_get = 0;
      o_EnReadFifo = 0;
      o_RdClkFifo = 0;
      dataToSdram8 = 8'd0;
      o_addressToSdram = 19'd0;
      o_wrSdram = 0;
      o_finish = 0;
      addrLocal_d = 19'd0;
    end
```

```
start: begin
  if(i_process) begin
    state_d = suspend;
  end else begin
    state_d = start;
  end
  o_get = 1;
  o_EnReadFifo = 0;
  o_RdClkFifo = 0;
  dataToSdram8 = 8'd0;
  o_addressToSdram = 19'd0;
  o_wrSdram = 0;
  o_finish = 0;
  addrLocal_d = 19'd0;
end
suspend: begin
  if((i_remainOnFifo > 10'd16) | (i_complete)) begin
    state_d = readFifo0;
  end else begin
    state_d = suspend;
  end
  o_get = 0;
  o_EnReadFifo = 1;
  o_RdClkFifo = 0;
  dataToSdram8 = 8'd0;
  o_addressToSdram = 19'd0;
  o_wrSdram = 0;
  o_finish = 0;
  addrLocal_d = addrLocal_q;
end
readFifo0: begin
  state_d = readFifo1;
  o_get = 0;
  o_EnReadFifo = 1;
  o_RdClkFifo = 1;
  dataToSdram8 = 8'd0;
  o_addressToSdram = 19'd0;
  o_wrSdram = 0;
  o_finish = 0;
  addrLocal_d = addrLocal_q;
end
readFifo1: begin
  state_d = setRed;
  o_get = 0;
```

```
        o_EnReadFifo = 1;
        o_RdClkFifo = 0;
        dataToSdram8 = 8'd0;
        o_addressToSdram = 19'd0;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
      end
      setRed: begin
        state_d = wait0;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[15:11], 1'd0};
        o_addressToSdram = addrLocal_q;
        o_wrSdram = 1;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
      end
      wait0: begin
        if(i_sdramReady) begin
          state_d = setGreen;
        end else begin
          state_d = wait0;
        end
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[15:11], 1'd0};
        o_addressToSdram = addrLocal_q;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
      end
      setGreen: begin
        state_d = wait1;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[10:5]};
        o_addressToSdram = addrLocal_q + 19'd4096;    //sdram region 2 store feature
Green
        o_wrSdram = 1;
        o_finish = 0;
```

```
        addrLocal_d = addrLocal_q;
      end
      wait1: begin
        if(i_sdramReady) begin
          state_d = setBlue;
        end else begin
          state_d = wait1;
        end
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[10:5]};
        o_addressToSdram = addrLocal_q + 19'd4096;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
      end
      setBlue: begin
        state_d = wait2;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0, i_dataFifo[4:0], 1'd0};
        o_addressToSdram = addrLocal_q + 19'd8192;    //sdram region 2 store feature
Green
        o_wrSdram = 1;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
      end
      wait2: begin
        if(i_sdramReady) begin
          state_d = update;
        end else begin
          state_d = wait2;
        end
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0, i_dataFifo[4:0], 1'd0};
        o_addressToSdram = addrLocal_q + 19'd8192;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
      end
```

```
    update: begin
      if((~i_complete) | (i_remainOnFifo > 10'd0)) begin
        state_d = suspend;
      end else begin
        state_d = finish;
      end
      o_get = 0;
      o_EnReadFifo = 0;
      o_RdClkFifo = 0;
      dataToSdram8 = 8'd0;
      o_addressToSdram = 19'd0;
      o_wrSdram = 0;
      o_finish = 0;
      addrLocal_d = addrLocal_q + 19'd1; //update local address
    end
    finish: begin
      state_d = idle;
      o_get = 0;
      o_EnReadFifo = 0;
      o_RdClkFifo = 0;
      dataToSdram8 = 8'd0;
      o_addressToSdram = 19'd0;
      o_wrSdram = 0;
      o_finish = 1;
      addrLocal_d = 19'd0;
    end
    default: begin
      state_d = idle;
      o_get = 0;
      o_EnReadFifo = 0;
      o_RdClkFifo = 0;
      dataToSdram8 = 8'd0;
      o_addressToSdram = 19'd0;
      o_wrSdram = 0;
      o_finish = 0;
      addrLocal_d = 19'd0;
    end
  endcase
end
assign o_dataSdram = {8'd0, dataToSdram8}; //assign to 16 bit for SDRAM (Extend
unsigned)
endmodule
```

Camera read:

```
module cameraRead(
  input  wire       i_pclk,     // Camera pixel clock
  input  wire       i_vsync,    // Vertical sync from camera (start/end of frame)
  input  wire       i_href,     // Horizontal reference (indicates valid pixel i_data in a row)
  input  wire [7:0]  i_data,     // 8-bit pixel i_data from camera
  input  wire       i_reset,    // Active-high synchronous reset

  output reg  [15:0]  o_pixelOut,  // 16-bit pixel i_data in RGB565 format
  output reg        o_pixelValid,// Indicates when o_pixelOut is valid (i.e. after both bytes
are received)
  output reg  [9:0]   o_xIndex,   // Horizontal pixel index (adjust bit-width as needed)
  output reg  [9:0]   o_yIndex,   // Vertical pixel index (adjust bit-width as needed)
  output wire       o_pixelClk   // Clock for downstream pixel processing (can be derived
from i_pclk)
);

  // Internal state to manage byte pairing from the camera
  // 0 indicates waiting for the high byte, 1 for the low byte
  reg byte_state;

  // Generate a simple pixel clock output.
  // For this example, we simply use the same i_pclk.
  // In a real system you might need to generate or divide clocks differently.
  assign o_pixelClk = i_pclk;

  // Main process: driven by the rising edge of i_pclk
  always @(posedge i_pclk or negedge i_reset) begin
    if (!i_reset) begin
      // Reset all signals
      o_pixelOut  <= 16'd0;
      o_pixelValid <= 1'b0;
      o_xIndex    <= 10'd0;
      o_yIndex    <= 10'd0;
      byte_state  <= 1'b0;
    end else begin
      // Start-of-frame detected: reset row and column counters
      if (i_vsync) begin
        o_xIndex    <= 10'd0;
        o_yIndex    <= 10'd0;
        byte_state  <= 1'b0;
        o_pixelValid <= 1'b0;
```

```
        end else if (i_href) begin
          // When i_href is high, the pixel i_data is valid for the current row.
          if (byte_state == 1'b0) begin
            // Fii_reset byte: store in the high 8 bits of o_pixelOut
            o_pixelOut[15:8] <= i_data;
            // Next cycle will capture the low byte.
            byte_state <= 1'b1;
            // Do not assert o_pixelValid until the full pixel is available
            o_pixelValid <= 1'b0;
          end else begin
            // Second byte: store in the low 8 bits of o_pixelOut
            o_pixelOut[7:0] <= i_data;
            // Now the pixel is complete – signal that the pixel is ready
            o_pixelValid <= 1'b1;
            // Increment the horizontal index after a full pixel is captured
            o_xIndex <= o_xIndex + 10'd1;
            // Reset state for next pixel
            byte_state <= 1'b0;
          end
        end else begin
          // When i_href is low, we are not receiving valid pixel i_data.
          // If we were in the middle of a pixel (odd byte received), discard it.
          byte_state  <= 1'b0;
          o_pixelValid <= 1'b0;

          // End of a line: if o_xIndex is not zero then
          // the transition of i_href to low indicates end-of-line.
          // Update the vertical index and reset horizontal index.
          if (o_xIndex != 0) begin
            o_xIndex <= 10'd0;
            o_yIndex <= o_yIndex + 10'd1;
          end
        end
      end
    end
endmodule
```

Control write to fifo:

```
module controlWriteToFifo(
  input wire [9:0]   i_xIndex,
  input wire [9:0]   i_yIndex,
```

```
  input wire        i_clk,
  input wire        i_reset,
  input wire        i_get,
  output reg        o_eWriteFifo,
  output reg        o_complete,
  output reg        o_process
);

localparam [1:0]   idle = 2'd0,
              process = 2'd1,
              finish = 2'd2;
reg [1:0] state_q, state_d;
reg masterE;

always @(posedge i_clk or negedge i_reset) begin    //sequential logic circuit update state
  if(!i_reset) begin
    state_q <= idle;
  end else begin
    state_q <= state_d;
  end
end

always @(*) begin    //combitional logic circuit control next state and output of state
machine
  case (state_q)
    idle: begin
      if((i_xIndex == 10'd0) & (i_yIndex == 10'd0) & (i_get) & (i_reset)) begin
        state_d = process;
      end else begin
        state_d = idle;
      end
      masterE = 0;
      o_complete = 1;
      o_process = 0;
    end
    process: begin
      if((i_xIndex == 10'd64) & (i_yIndex == 10'd63)) begin
        state_d = finish;
      end else begin
        state_d = process;
      end
      masterE = 1;
      o_complete = 0;
      o_process = 1;
```

```
      end
      finish: begin
        state_d = idle;
        masterE = 0;
        o_complete = 1;
        o_process = 0;
      end
      default: begin
        state_d = idle;
        masterE = 0;
        o_complete = 0;
        o_process = 0;
      end
    endcase
end

always @(*) begin //combitional logic circuit control o_eWriteFifo signal.
    if((i_xIndex <= 10'd64) & (i_yIndex < 10'd64) & (masterE)) begin  //only allow write to fifo
x < 64 and y < 64 and when sate machine
      o_eWriteFifo = 1;
    end else begin
      o_eWriteFifo = 0;
    end
end
endmodule
```


Fetching control:

```
module fetchingControl(
    input [18:0]   i_baseAddr0,
    input [18:0]   i_baseAddr1,
    input [18:0]   i_baseAddr2,
    input          i_clk,
    input          i_reset,
    input          i_sdramReady,
    input          i_start,
    output reg     o_rdSdram,
    output reg [18:0] o_addrToSdram,
    output reg     o_finish,
    output reg     o_wrRam0,
    output reg     o_wrRam1,
    output reg     o_wrRam2,
```

```
  output  [11:0]  o_addrToRam
);
localparam [3:0]    idle = 4'd0,
                    setSdram0 = 4'd1,
                    waitSdram0 = 4'd2,
                    setRam0 = 4'd3,
                    setSdram1 = 4'd4,
                    waitSdram1 = 4'd5,
                    setRam1 = 4'd6,
                    setSdram2 = 4'd7,
                    waitSdram2 = 4'd8,
                    setRam2 = 4'd9,
                    update = 4'd10,
                    finish = 4'd11;

reg [3:0] state_q, state_d;
reg [18:0] addrLocal_q, addrLocal_d;

always @(posedge i_clk or negedge i_reset) begin   //update state and addr counter
  if(!i_reset) begin
    state_q <= idle;
    addrLocal_q <= 19'd0;
  end else begin
    state_q <= state_d;
    addrLocal_q <= addrLocal_d;
  end
end

always @(*) begin
  case (state_q)
    idle: begin
      if(i_start) begin
        state_d = setSdram0;
      end else begin
        state_d = idle;
      end
      o_rdSdram = 0;
      o_addrToSdram = 19'd0;
      addrLocal_d = 0;
      o_wrRam0 = 0;
      o_wrRam1 = 0;
      o_wrRam2 = 0;
      o_finish = 0;
    end
```

```
    setSdram0: begin
      state_d = waitSdram0;
      o_rdSdram = 1;
      o_addrToSdram = addrLocal_q + i_baseAddr0;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 0;
      o_wrRam1 = 0;
      o_wrRam2 = 0;
      o_finish = 0;
    end
    waitSdram0: begin
      if(i_sdramReady) begin
        state_d = setRam0;
      end else begin
        state_d = waitSdram0;
      end
      o_rdSdram = 0;
      o_addrToSdram = addrLocal_q + i_baseAddr0;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 0;
      o_wrRam1 = 0;
      o_wrRam2 = 0;
      o_finish = 0;
    end
    setRam0: begin
      state_d = setSdram1;
      o_rdSdram = 0;
      o_addrToSdram = addrLocal_q + i_baseAddr1;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 1;
      o_wrRam1 = 0;
      o_wrRam2 = 0;
      o_finish = 0;
    end
    setSdram1: begin
      state_d = waitSdram1;
      o_rdSdram = 1;
      o_addrToSdram = addrLocal_q + i_baseAddr1;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 0;
      o_wrRam1 = 0;
      o_wrRam2 = 0;
      o_finish = 0;
    end
```

```
    waitSdram1: begin
      if(i_sdramReady) begin
        state_d = setRam1;
      end else begin
        state_d = waitSdram1;
      end
      o_rdSdram = 0;
      o_addrToSdram = addrLocal_q + i_baseAddr1;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 0;
      o_wrRam1 = 0;
      o_wrRam2 = 0;
      o_finish = 0;
    end
    setRam1: begin
      state_d = setSdram2;
      o_rdSdram = 0;
      o_addrToSdram = addrLocal_q + i_baseAddr1;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 0;
      o_wrRam1 = 1;
      o_wrRam2 = 0;
      o_finish = 0;
    end
    setSdram2: begin
      state_d = waitSdram2;
      o_rdSdram = 1;
      o_addrToSdram = addrLocal_q + i_baseAddr2;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 0;
      o_wrRam1 = 0;
      o_wrRam2 = 0;
      o_finish = 0;
    end
    waitSdram2: begin
      if(i_sdramReady) begin
        state_d = setRam2;
      end else begin
        state_d = waitSdram2;
      end
      o_rdSdram = 0;
      o_addrToSdram = addrLocal_q + i_baseAddr2;
      addrLocal_d = addrLocal_q;
      o_wrRam0 = 0;
```

```
        o_wrRam1 = 0;
        o_wrRam2 = 0;
        o_finish = 0;
      end
      setRam2: begin
        state_d = update;
        o_rdSdram = 0;
        o_addrToSdram = addrLocal_q + i_baseAddr2;
        addrLocal_d = addrLocal_q;
        o_wrRam0 = 0;
        o_wrRam1 = 0;
        o_wrRam2 = 1;
        o_finish = 0;
      end
      update: begin
        if(addrLocal_q == 19'd4095) begin //final data address
          state_d = finish;
        end else begin
          state_d = setSdram0;
        end
        o_rdSdram = 0;
        o_addrToSdram = addrLocal_q + i_baseAddr0;
        addrLocal_d = addrLocal_q + 19'd1;
        o_wrRam0 = 0;
        o_wrRam1 = 0;
        o_wrRam2 = 0;
        o_finish = 0;
      end
      finish: begin
        state_d = idle;
        o_rdSdram = 0;
        o_addrToSdram = 19'd0;
        addrLocal_d = 0;
        o_wrRam0 = 0;
        o_wrRam1 = 0;
        o_wrRam2 = 0;
        o_finish = 1;
      end
      default: begin
        state_d = idle;
        o_rdSdram = 0;
        o_addrToSdram = 19'd0;
        addrLocal_d = 0;
        o_wrRam0 = 0;
```

```
        o_wrRam1 = 0;
        o_wrRam2 = 0;
        o_finish = 0;
      end
   endcase
end
assign o_addrToRam = addrLocal_q[11:0]; //just need 12 bit lower
endmodule
```


Core convolution:

```
module coreConv(
   input [89:0] i_data,
   input [89:0] i_weight,
   output [19:0] o_data20,
   output [9:0] o_data
);
//internal wire
wire [19:0] dataTemp;
//i_data de_Bus
wire [9:0] data [8:0];
assign data[0] = i_data[9:0];
assign data[1] = i_data[19:10];
assign data[2] = i_data[29:20];
assign data[3] = i_data[39:30];
assign data[4] = i_data[49:40];
assign data[5] = i_data[59:50];
assign data[6] = i_data[69:60];
assign data[7] = i_data[79:70];
assign data[8] = i_data[89:80];
//i_weight de_bus
wire [9:0] weight [8:0];
assign weight[0] = i_weight[9:0];
assign weight[1] = i_weight[19:10];
assign weight[2] = i_weight[29:20];
assign weight[3] = i_weight[39:30];
assign weight[4] = i_weight[49:40];
assign weight[5] = i_weight[59:50];
assign weight[6] = i_weight[69:60];
assign weight[7] = i_weight[79:70];
assign weight[8] = i_weight[89:80];
//internal wire result of multi 20 bit
```

```
wire [19:0] resultMulti [8:0];
//mutiplication block
multi multiBlock_0(.i_data(data[0]), .i_weight(weight[0]), .o_data(resultMulti[0]));
multi multiBlock_1(.i_data(data[1]), .i_weight(weight[1]), .o_data(resultMulti[1]));
multi multiBlock_2(.i_data(data[2]), .i_weight(weight[2]), .o_data(resultMulti[2]));
multi multiBlock_3(.i_data(data[3]), .i_weight(weight[3]), .o_data(resultMulti[3]));
multi multiBlock_4(.i_data(data[4]), .i_weight(weight[4]), .o_data(resultMulti[4]));
multi multiBlock_5(.i_data(data[5]), .i_weight(weight[5]), .o_data(resultMulti[5]));
multi multiBlock_6(.i_data(data[6]), .i_weight(weight[6]), .o_data(resultMulti[6]));
multi multiBlock_7(.i_data(data[7]), .i_weight(weight[7]), .o_data(resultMulti[7]));
multi multiBlock_8(.i_data(data[8]), .i_weight(weight[8]), .o_data(resultMulti[8]));
//Plus block
plus9Para adder9paraBlock(  .i_data0(resultMulti[0]),
                            .i_data1(resultMulti[1]),
                            .i_data2(resultMulti[2]),
                            .i_data3(resultMulti[3]),
                            .i_data4(resultMulti[4]),
                            .i_data5(resultMulti[5]),
                            .i_data6(resultMulti[6]),
                            .i_data7(resultMulti[7]),
                            .i_data8(resultMulti[8]),
                            .o_data(dataTemp)
                          );
assign o_data = dataTemp[19] ? 10'd0 : dataTemp[18:9];
assign o_data20 = dataTemp;
endmodule
```


Convolution:

```
module conv(
   input [89:0] i_busData0,
   input [89:0] i_busData1,
   input [89:0] i_busData2,
   input [89:0] i_busWeight0,
   input [89:0] i_busWeight1,
   input [89:0] i_busWeight2,
   input      i_opcode,
   output [9:0] o_data0,
   output [9:0] o_data1,
   output [9:0] o_data2
);
//internal wire connect result of coreConv block
```

```
wire [9:0] result [2:0];
wire [19:0] data20_0, data20_1, data20_2;
//architechture
coreConv coreBlock0(.i_data(i_busData0), .i_weight(i_busWeight0),
.o_data20(data20_0), .o_data(result[0]));
coreConv coreBlock1(.i_data(i_busData1), .i_weight(i_busWeight1),
.o_data20(data20_1), .o_data(result[1]));
coreConv coreBlock2(.i_data(i_busData2), .i_weight(i_busWeight2),
.o_data20(data20_2), .o_data(result[2]));
//plus to handel CONV operation
wire [9:0] resultConv;
plus3para plus3paraBlock(.i_data0(data20_0), .i_data1(data20_1), .i_data2(data20_2),
.o_data(resultConv));
//assign output
mux2to1 mux2to1Block(.i_dataA(resultConv), .i_dataB(result[0]), .i_sel(i_opcode),
.o_data(o_data0));
assign o_data1 = result[1];
assign o_data2 = result[2];
endmodule

module plus3para(
   input [19:0] i_data0,
   input [19:0] i_data1,
   input [19:0] i_data2,
   output [9:0] o_data
);
//for high performance
//state 0
wire [19:0] data01;
wire [19:0] dataTemp;
assign data01 = i_data1 + i_data0;
//final state
assign dataTemp = data01 + i_data2;
assign o_data = dataTemp[19] ? 10'd0 : dataTemp[18:9];
endmodule
```

Convolution control:

```
module convolutionControl(
   input          i_clk,
   input          i_reset,
   input          i_start,
```

```
  input          i_opcode,
  input          i_validRam,
  output [107:0]    o_addrRead0,
  output [107:0]    o_addrRead1,
  output [107:0]    o_addrRead2,
  output reg       o_startRam,
  output reg       o_selRamD0,
  output reg [11:0]  o_addrWrite0,
  output reg [11:0]  o_addrWrite1,
  output reg [11:0]  o_addrWrite2,
  output reg       o_wrEnable,
  output reg       o_finish,
  output [11:0]     o_localAddr
);
localparam [2:0]   idle = 3'd0,
            readRam = 3'd1,
            waitDone = 3'd2,
            setWrite = 3'd3,
            update = 3'd4,
            finish = 3'd5;

reg [2:0] state_q, state_d;
reg [11:0] addrLocal_q, addrLocal_d;
//output of state
//address read port for ram source
reg [11:0] addrRead0;
reg [11:0] addrRead1;
reg [11:0] addrRead2;
reg [11:0] addrRead3;
reg [11:0] addrRead4;
reg [11:0] addrRead5;
reg [11:0] addrRead6;
reg [11:0] addrRead7;
reg [11:0] addrRead8;
//-----
always @(posedge i_clk or negedge i_reset) begin
  if(!i_reset) begin
    state_q <= idle;
    addrLocal_q <= 12'd0;
  end else begin
    state_q <= state_d;
    addrLocal_q <= addrLocal_d;
  end
end
```

```
always @(*) begin
   case (state_q)
      idle: begin
         if(i_start & i_reset) begin
            state_d = readRam;
         end else begin
            state_d = idle;
         end
         o_startRam = 0;
         o_selRamD0 = i_opcode;
         o_addrWrite0 = 12'd0;
         o_addrWrite1 = 12'd0;
         o_addrWrite2 = 12'd0;
         o_wrEnable = 0;
         o_finish = 0;
         addrRead0 = 12'd0;
         addrRead1 = 12'd0;
         addrRead2 = 12'd0;
         addrRead3 = 12'd0;
         addrRead4 = 12'd0;
         addrRead5 = 12'd0;
         addrRead6 = 12'd0;
         addrRead7 = 12'd0;
         addrRead8 = 12'd0;
         addrLocal_d = 12'd0;
      end
      readRam: begin
         state_d = waitDone;
         o_startRam = 1;
         o_selRamD0 = i_opcode;
         o_addrWrite0 = addrLocal_q;
         o_addrWrite1 = addrLocal_q;
         o_addrWrite2 = addrLocal_q;
         o_wrEnable = 0;
         o_finish = 0;
         addrRead0 = addrLocal_q - 12'd65;
         addrRead1 = addrLocal_q - 12'd64;
         addrRead2 = addrLocal_q - 12'd63;
         addrRead3 = addrLocal_q - 12'd1;
         addrRead4 = addrLocal_q;  //center address
         addrRead5 = addrLocal_q + 12'd1;
         addrRead6 = addrLocal_q + 12'd63;
         addrRead7 = addrLocal_q + 12'd64;
```

```
      addrRead8 = addrLocal_q + 12'd65;
      addrLocal_d = addrLocal_q;
    end
    waitDone: begin
      if(i_validRam) begin
        state_d = setWrite;
      end else begin
        state_d = waitDone;
      end
      o_startRam = 0;
      o_selRamD0 = i_opcode;
      o_addrWrite0 = addrLocal_q;
      o_addrWrite1 = addrLocal_q;
      o_addrWrite2 = addrLocal_q;
      o_wrEnable = 0;
      o_finish = 0;
      addrRead0 = addrLocal_q - 12'd65;
      addrRead1 = addrLocal_q - 12'd64;
      addrRead2 = addrLocal_q - 12'd63;
      addrRead3 = addrLocal_q - 12'd1;
      addrRead4 = addrLocal_q;  //center address
      addrRead5 = addrLocal_q + 12'd1;
      addrRead6 = addrLocal_q + 12'd63;
      addrRead7 = addrLocal_q + 12'd64;
      addrRead8 = addrLocal_q + 12'd65;
      addrLocal_d = addrLocal_q;
    end
    setWrite: begin
      state_d = update;
      o_startRam = 0;
      o_selRamD0 = i_opcode;
      o_addrWrite0 = addrLocal_q;
      o_addrWrite1 = addrLocal_q;
      o_addrWrite2 = addrLocal_q;
      o_wrEnable = 1;
      o_finish = 0;
      addrRead0 = addrLocal_q - 12'd65;
      addrRead1 = addrLocal_q - 12'd64;
      addrRead2 = addrLocal_q - 12'd63;
      addrRead3 = addrLocal_q - 12'd1;
      addrRead4 = addrLocal_q;  //center address
      addrRead5 = addrLocal_q + 12'd1;
      addrRead6 = addrLocal_q + 12'd63;
      addrRead7 = addrLocal_q + 12'd64;
```

```
        addrRead8 = addrLocal_q + 12'd65;
        addrLocal_d = addrLocal_q;
    end
    update: begin
        if(addrLocal_q == 12'd4095) begin
            state_d = finish;
        end else begin
            state_d = readRam;
        end
        o_startRam = 0;
        o_selRamD0 = i_opcode;
        o_addrWrite0 = addrLocal_q;
        o_addrWrite1 = addrLocal_q;
        o_addrWrite2 = addrLocal_q;
        o_wrEnable = 0;
        o_finish = 0;
        addrRead0 = addrLocal_q - 12'd65;
        addrRead1 = addrLocal_q - 12'd64;
        addrRead2 = addrLocal_q - 12'd63;
        addrRead3 = addrLocal_q - 12'd1;
        addrRead4 = addrLocal_q;  //center address
        addrRead5 = addrLocal_q + 12'd1;
        addrRead6 = addrLocal_q + 12'd63;
        addrRead7 = addrLocal_q + 12'd64;
        addrRead8 = addrLocal_q + 12'd65;
        addrLocal_d = addrLocal_q + 12'd1;
    end
    finish: begin
        state_d = idle;
        o_startRam = 0;
        o_selRamD0 = i_opcode;
        o_addrWrite0 = 12'd0;
        o_addrWrite1 = 12'd0;
        o_addrWrite2 = 12'd0;
        o_wrEnable = 0;
        o_finish = 1;
        addrRead0 = 12'd0;
        addrRead1 = 12'd0;
        addrRead2 = 12'd0;
        addrRead3 = 12'd0;
        addrRead4 = 12'd0;
        addrRead5 = 12'd0;
        addrRead6 = 12'd0;
        addrRead7 = 12'd0;
```

```
        addrRead8 = 12'd0;
        addrLocal_d = 12'd0;
      end
      default: begin
        state_d = idle;
        o_startRam = 0;
        o_selRamD0 = i_opcode;
        o_addrWrite0 = 12'd0;
        o_addrWrite1 = 12'd0;
        o_addrWrite2 = 12'd0;
        o_wrEnable = 0;
        o_finish = 0;
        addrRead0 = 12'd0;
        addrRead1 = 12'd0;
        addrRead2 = 12'd0;
        addrRead3 = 12'd0;
        addrRead4 = 12'd0;
        addrRead5 = 12'd0;
        addrRead6 = 12'd0;
        addrRead7 = 12'd0;
        addrRead8 = 12'd0;
        addrLocal_d = 12'd0;
      end
    endcase
end
assign o_addrRead0 = {addrRead8, addrRead7, addrRead6, addrRead5, addrRead4,
addrRead3, addrRead2, addrRead1, addrRead0};
assign o_addrRead1 = {addrRead8, addrRead7, addrRead6, addrRead5, addrRead4,
addrRead3, addrRead2, addrRead1, addrRead0};
assign o_addrRead2 = {addrRead8, addrRead7, addrRead6, addrRead5, addrRead4,
addrRead3, addrRead2, addrRead1, addrRead0};
assign o_localAddr = addrLocal_q;
endmodule
```

Padding control:

```
module paddingControl(
  input [11:0] i_localAddr,
  output o_sel0,
  output o_sel1,
  output o_sel2,
  output o_sel3,
```

```
  output o_sel4,
  output o_sel5,
  output o_sel6,
  output o_sel7,
  output o_sel8
);
wire hHigh, hLow, lHigh, lLow;
assign hHigh = &(i_localAddr[11:6]);
assign hLow = ~(|(i_localAddr[11:6]));
assign lHigh = &(i_localAddr[5:0]);
assign lLow = ~(|(i_localAddr[5:0]));

assign o_sel0 = hLow | lLow;
assign o_sel1 = hLow;
assign o_sel2 = hLow | lHigh;
assign o_sel3 = lLow;
assign o_sel4 = 1'd0;
assign o_sel5 = lHigh;
assign o_sel6 = lLow | hHigh;
assign o_sel7 = hHigh;
assign o_sel8 = hHigh | lLow;
endmodule
```

Zero padding:

```
module zeroPadding(
  input [8:0] i_sel,
  input [89:0] i_data,
  output [89:0] o_data
);

assign o_data[9:0] = {10{i_sel[0]}} & i_data[9:0];
assign o_data[19:10] = {10{i_sel[1]}} & i_data[19:10];
assign o_data[29:20] = {10{i_sel[2]}} & i_data[29:20];
assign o_data[39:30] = {10{i_sel[3]}} & i_data[39:30];
assign o_data[49:40] = {10{i_sel[4]}} & i_data[49:40];
assign o_data[59:50] = {10{i_sel[5]}} & i_data[59:50];
assign o_data[69:60] = {10{i_sel[6]}} & i_data[69:60];
assign o_data[79:70] = {10{i_sel[7]}} & i_data[79:70];
assign o_data[89:80] = {10{i_sel[8]}} & i_data[89:80];

endmodule
```

Weight Gen:

```
module weightGen(
   input [5:0] i_opcode,
   output reg [89:0] o_weight0,
   output reg [89:0] o_weight1,
   output reg [89:0] o_weight2
);

//expected to generate ROM
reg [89:0] weight0 [0:31];
reg [89:0] weight1 [0:31];
reg [89:0] weight2 [0:31];

initial begin
   $readmemh("weight0.hex", weight0);
   $readmemh("weight1.hex", weight1);
   $readmemh("weight2.hex", weight2);
end

always @(*) begin
   o_weight0 = weight0[i_opcode];
   o_weight1 = weight1[i_opcode];
   o_weight2 = weight2[i_opcode];
end
endmodule
```

Embeded RAM conv:

```
module ramConv (
   input [11:0] i_addrIn,   // 12-bit address (4096 locations)
   input [11:0] i_addrOut,   // 12-bit address (4096 locations)
   input [9:0]  i_data,   // 10-bit data input
   input      i_wrEnable,
   input      i_clk,
   output reg [9:0] o_data  // 10-bit data output
);

   // Declare BRAM and force Quartus to use M10K blocks
```

```
  reg [9:0] RAM [0:4095];

  always @(posedge i_clk) begin
    if (i_wrEnable) RAM[i_addrIn] <= i_data; // Write operation
    o_data <= RAM[i_addrOut];
  end

endmodule
```


RAM control:

```
module ramControl(
  input [107:0]     i_addrOut, //addr read data
  input [11:0]      i_addrIn,
  input [9:0]       i_data,    // Đầu vào dữ liệu
  input             i_wrEnable,
  input             i_quickGet,
  input [11:0]      i_addrOutQuick,
  input             i_clk,
  input             i_reset,
  input             i_start,
  output reg [89:0]  o_data,    // Đầu ra dữ liệu
  output [9:0]      o_quickData,
  output reg        o_valid,
  output reg        o_ready
);
wire [11:0] addrOut [9:0];
assign addrOut[0] = i_addrOut[11:0]; //data0
assign addrOut[1] = i_addrOut[23:12]; //data1,
assign addrOut[2] = i_addrOut[35:24]; //data2,
assign addrOut[3] = i_addrOut[47:36];
assign addrOut[4] = i_addrOut[59:48];
assign addrOut[5] = i_addrOut[71:60];
assign addrOut[6] = i_addrOut[83:72];
assign addrOut[7] = i_addrOut[95:84];
assign addrOut[8] = i_addrOut[107:96]; //data8
assign addrOut[9] = i_addrOutQuick;
reg [11:0] activeAddr;
localparam [3:0]   idle  = 4'd0,
              load0 = 4'd1,
              load1 = 4'd2,
              load2 = 4'd3,
```

```
            load3 = 4'd4,
            load4 = 4'd5,
            load5 = 4'd6,
            load6 = 4'd7,
            load7 = 4'd8,
            load8 = 4'd9,
            buffer = 4'd10;

reg [3:0] state_q, state_d;
reg [3:0] selAddr;
reg [8:0] enableWriteBuffer;
wire [9:0] dataFromRamBlock;

always @(*) begin //Mux select address
  activeAddr = addrOut[selAddr];
end

always @(posedge i_clk or negedge i_reset) begin
  if(!i_reset) begin
    state_q <= idle;
  end else begin
    state_q <= state_d;
  end
end

always @(*) begin
  case (state_q)
    idle: begin
      if(i_reset) begin
        if(i_start) begin
          state_d = load0;
          selAddr = 4'd0;
        end else if (i_quickGet) begin
          state_d = idle;
          selAddr = 4'd9;
        end else begin
          state_d = idle;
          selAddr = 4'd0;
        end
      end else begin
        state_d = idle;
        selAddr = 4'd0;
      end
      o_ready = 1;
```

```
        o_valid = 0;
        enableWriteBuffer = 9'b0000_0000_0;
    end
    load0: begin
        state_d = load1;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0000_0000_1;
        selAddr = 4'd1;
    end
    load1: begin
        state_d = load2;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0000_0001_0;
        selAddr = 4'd2;
    end
    load2: begin
        state_d = load3;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0000_0010_0;
        selAddr = 4'd3;
    end
    load3: begin
        state_d = load4;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0000_0100_0;
        selAddr = 4'd4;
    end
    load4: begin
        state_d = load5;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0000_1000_0;
        selAddr = 4'd5;
    end
    load5: begin
        state_d = load6;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0001_0000_0;
        selAddr = 4'd6;
```

```
      end
      load6: begin
        state_d = load7;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0010_0000_0;
        selAddr = 4'd7;
      end
      load7: begin
        state_d = load8;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0100_0000_0;
        selAddr = 4'd8;
      end
      load8: begin
        state_d = buffer;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b1000_0000_0;
        selAddr = 4'd0;
      end
      buffer: begin
        state_d = idle;
        o_ready = 0;
        o_valid = 1;
        enableWriteBuffer = 9'b0000_0000_0;
        selAddr = 4'd0;
      end
      default: begin
        state_d = idle;
        o_ready = 0;
        o_valid = 0;
        enableWriteBuffer = 9'b0000_0000_0;
        selAddr = 4'd0;
      end
    endcase
end

ramConv RAMBLOCK(
    .i_addrIn(i_addrIn),   // 12-bit address (4096 locations)
    .i_addrOut(activeAddr),   // 12-bit address (4096 locations)
    .i_data(i_data),   // 10-bit data input
    .i_wrEnable(i_wrEnable),
```

```
  .i_clk(i_clk),
  .o_data(dataFromRamBlock)  // 10-bit data output
);

always @(posedge i_clk or negedge i_reset) begin
  if(!i_reset) begin
    o_data <= 90'd0;
  end else begin
    case (enableWriteBuffer)
      9'b0000_0000_1: o_data[9:0] <= dataFromRamBlock;
      9'b0000_0001_0: o_data[19:10] <= dataFromRamBlock;
      9'b0000_0010_0: o_data[29:20] <= dataFromRamBlock;
      9'b0000_0100_0: o_data[39:30] <= dataFromRamBlock;
      9'b0000_1000_0: o_data[49:40] <= dataFromRamBlock;
      9'b0001_0000_0: o_data[59:50] <= dataFromRamBlock;
      9'b0010_0000_0: o_data[69:60] <= dataFromRamBlock;
      9'b0100_0000_0: o_data[79:70] <= dataFromRamBlock;
      9'b1000_0000_0: o_data[89:80] <= dataFromRamBlock;
      default: o_data <= o_data;
    endcase
  end
end
assign o_quickData = dataFromRamBlock;
endmodule
```


Base address decoder fetching

```
module baseAddrFetDecode(
  input [5:0] i_opcode,
  output [18:0] o_baseAddr0,
  output [18:0] o_baseAddr1,
  output [18:0] o_baseAddr2
);
reg [6:0] featureIndex0;
reg [6:0] featureIndex1;
reg [6:0] featureIndex2;
always @(*) begin
  case (i_opcode)
    6'd0: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
```

```
    end
    6'd1: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd2: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd3: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd4: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd5: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd6: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd7: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd8: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
      featureIndex2 = 7'd2;
    end
    6'd9: begin
      featureIndex0 = 7'd0;
      featureIndex1 = 7'd1;
```

```
    featureIndex2 = 7'd2;
  end
  6'd10: begin
    featureIndex0 = 7'd0;
    featureIndex1 = 7'd1;
    featureIndex2 = 7'd2;
  end
  6'd11: begin
    featureIndex0 = 7'd0;
    featureIndex1 = 7'd1;
    featureIndex2 = 7'd2;
  end
  6'd12: begin
    featureIndex0 = 7'd0;
    featureIndex1 = 7'd1;
    featureIndex2 = 7'd2;
  end
  6'd13: begin
    featureIndex0 = 7'd0;
    featureIndex1 = 7'd1;
    featureIndex2 = 7'd2;
  end
  6'd14: begin
    featureIndex0 = 7'd0;
    featureIndex1 = 7'd1;
    featureIndex2 = 7'd2;
  end
  6'd15: begin
    featureIndex0 = 7'd0;
    featureIndex1 = 7'd1;
    featureIndex2 = 7'd2;
  end
  6'd16: begin
    featureIndex0 = 7'd3;
    featureIndex1 = 7'd4;
    featureIndex2 = 7'd5;
  end
  6'd17: begin
    featureIndex0 = 7'd6;
    featureIndex1 = 7'd7;
    featureIndex2 = 7'd8;
  end
  6'd18: begin
    featureIndex0 = 7'd9;
```

```
    featureIndex1 = 7'd10;
    featureIndex2 = 7'd11;
  end
  6'd19: begin
    featureIndex0 = 7'd12;
    featureIndex1 = 7'd13;
    featureIndex2 = 7'd14;
  end
  6'd20: begin
    featureIndex0 = 7'd15;
    featureIndex1 = 7'd16;
    featureIndex2 = 7'd17;
  end
  6'd21: begin
    featureIndex0 = 7'd18;
    featureIndex1 = 7'd19;
    featureIndex2 = 7'd20;
  end
  6'd22: begin
    featureIndex0 = 7'd21;
    featureIndex1 = 7'd22;
    featureIndex2 = 7'd23;
  end
  6'd23: begin
    featureIndex0 = 7'd24;
    featureIndex1 = 7'd25;
    featureIndex2 = 7'd26;
  end
  6'd24: begin
    featureIndex0 = 7'd27;
    featureIndex1 = 7'd28;
    featureIndex2 = 7'd29;
  end
  6'd25: begin
    featureIndex0 = 7'd30;
    featureIndex1 = 7'd31;
    featureIndex2 = 7'd32;
  end
  6'd26: begin
    featureIndex0 = 7'd33;
    featureIndex1 = 7'd34;
    featureIndex2 = 7'd35;
  end
  6'd27: begin
```

```
      featureIndex0 = 7'd36;
      featureIndex1 = 7'd37;
      featureIndex2 = 7'd38;
    end
    6'd28: begin
      featureIndex0 = 7'd39;
      featureIndex1 = 7'd40;
      featureIndex2 = 7'd41;
    end
    6'd29: begin
      featureIndex0 = 7'd42;
      featureIndex1 = 7'd43;
      featureIndex2 = 7'd44;
    end
    6'd30: begin
      featureIndex0 = 7'd45;
      featureIndex1 = 7'd46;
      featureIndex2 = 7'd47;
    end
    6'd31: begin
      featureIndex0 = 7'd48;
      featureIndex1 = 7'd49;
      featureIndex2 = 7'd50;
    end
    6'd32: begin
      featureIndex0 = 7'd51;
      featureIndex1 = 7'd52;
      featureIndex2 = 7'd53;
    end
    6'd33: begin
      featureIndex0 = 7'd54;
      featureIndex1 = 7'd55;
      featureIndex2 = 7'd56;
    end
    6'd34: begin
      featureIndex0 = 7'd57;
      featureIndex1 = 7'd58;
      featureIndex2 = 7'd59;
    end
    6'd35: begin
      featureIndex0 = 7'd60;
      featureIndex1 = 7'd61;
      featureIndex2 = 7'd62;
    end
```

```
      6'd36: begin
        featureIndex0 = 7'd63;
        featureIndex1 = 7'd64;
        featureIndex2 = 7'd65;
      end
      6'd37: begin
        featureIndex0 = 7'd66;
        featureIndex1 = 7'd0;
        featureIndex2 = 7'd0;
      end
      default: begin
        featureIndex0 = 7'd0;
        featureIndex1 = 7'd0;
        featureIndex2 = 7'd0;
      end
    endcase
end
assign o_baseAddr0 = featureIndex0 *19'd64*19'd64;
assign o_baseAddr1 = featureIndex1 *19'd64*19'd64;
assign o_baseAddr2 = featureIndex2 *19'd64*19'd64;

endmodule
```

Write back control;

```
module writeBackControl (
    input i_clk,
    input i_reset,
    input i_sdramReady,
    input [18:0] i_baseAddr0,
    input [18:0] i_baseAddr1,
    input [18:0] i_baseAddr2,
    input i_start,
    output [18:0] o_addrToRam0,
    output [18:0] o_addrToRam1,
    output [18:0] o_addrToRam2,
    output reg  o_quickRam,
    output reg [18:0] o_addrToSdram,
    output reg o_wrSdram,
    output reg [1:0] o_selData,
    output reg o_finish
);
```

```
localparam [3:0]    idle = 4'd0,
                    setRam = 4'd1,
                    setSdram0 = 4'd2,
                    waitDone0 = 4'd3,
                    setSdram1 = 4'd4,
                    waitDone1 = 4'd5,
                    setSdram2 = 4'd6,
                    waitDone2 = 4'd7,
                    update = 4'd8,
                    finish = 4'd9;

reg [18:0] addrLocal_q, addrLocal_d;
reg [3:0] state_q, state_d;

always @(posedge i_clk or negedge i_reset) begin  //update next state, local address and
handle reset.
   if(!i_reset) begin
      state_q <= idle;
      addrLocal_q <= 19'd0;
   end else begin
      state_q <= state_d;
      addrLocal_q <= addrLocal_d;
   end
end

always @(*) begin  //combitional ciucirt control state machine.
   case (state_q)
      idle: begin
         if(i_reset & i_start) begin
            state_d = setRam;
         end else begin
            state_d = idle;
         end
         o_finish = 0;
         o_addrToSdram = 19'd0;
         addrLocal_d = 19'd0;
         o_wrSdram = 0;
         o_quickRam = 0;
         o_selData = 2'd0;
      end
      setRam: begin
         state_d = setSdram0;
         o_finish = 0;
         o_addrToSdram = 0;
```

```
        addrLocal_d = addrLocal_q;
        o_wrSdram = 0;
        o_quickRam = 1;
        o_selData = 2'd0;
      end
      setSdram0: begin
        state_d = waitDone0;
        o_finish = 0;
        o_addrToSdram = i_baseAddr0 + addrLocal_q;
        addrLocal_d = addrLocal_q;
        o_wrSdram = 1;
        o_quickRam = 0;
        o_selData = 2'd0;
      end
      waitDone0: begin
        if(i_sdramReady) begin
          state_d = setSdram1;
        end else begin
          state_d = waitDone0;
        end
        o_finish = 0;
        o_addrToSdram = i_baseAddr0 + addrLocal_q;
        addrLocal_d = addrLocal_q;
        o_wrSdram = 0;
        o_quickRam = 0;
        o_selData = 2'd0;
      end
      setSdram1: begin
        state_d = waitDone1;
        o_finish = 0;
        o_addrToSdram = i_baseAddr1 + addrLocal_q;
        addrLocal_d = addrLocal_q;
        o_wrSdram = 1;
        o_quickRam = 0;
        o_selData = 2'd1;
      end
      waitDone1: begin
        if(i_sdramReady) begin
          state_d = setSdram2;
        end else begin
          state_d = waitDone1;
        end
        o_finish = 0;
        o_addrToSdram = i_baseAddr1 + addrLocal_q;
```

```
        addrLocal_d = addrLocal_q;
        o_wrSdram = 0;
        o_quickRam = 0;
        o_selData = 2'd1;
      end
      setSdram2: begin
        state_d = waitDone2;
        o_finish = 0;
        o_addrToSdram = i_baseAddr2 + addrLocal_q;
        addrLocal_d = addrLocal_q;
        o_wrSdram = 1;
        o_quickRam = 0;
        o_selData = 2'd2;
      end
      waitDone2: begin
        if(i_sdramReady) begin
          state_d = update;
        end else begin
          state_d = waitDone2;
        end
        o_finish = 0;
        o_addrToSdram = i_baseAddr2 + addrLocal_q;
        addrLocal_d = addrLocal_q;
        o_wrSdram = 0;
        o_quickRam = 0;
        o_selData = 2'd2;
      end
      update: begin
        if(addrLocal_q == 19'd4095) begin
          state_d = finish;
        end else begin
          state_d = setRam;
        end
        o_finish = 0;
        o_addrToSdram = 19'd0;
        addrLocal_d = addrLocal_q + 19'd1;
        o_wrSdram = 0;
        o_quickRam = 0;
        o_selData = 2'd0;
      end
      finish: begin
        state_d = idle;
        o_finish = 1;
        o_addrToSdram = 19'd0;
```

```
        o_wrSdram = 0;
        addrLocal_d = 19'd0;
        o_quickRam = 0;
        o_selData = 2'd0;
      end
      default: begin
        state_d = idle;
        o_finish = 0;
        o_addrToSdram = 19'd0;
        o_wrSdram = 0;
        addrLocal_d = 19'd0;
        o_quickRam = 0;
        o_selData = 2'd0;
      end
    endcase
end

assign o_addrToRam0 = addrLocal_q[11:0];
assign o_addrToRam1 = addrLocal_q[11:0];
assign o_addrToRam2 = addrLocal_q[11:0];

endmodule
```

Base address decoder write back:

```
module baseAddrWriteBackDecode(
    input [5:0] i_opcode,
    output [18:0] o_baseAddr0,
    output [18:0] o_baseAddr1,
    output [18:0] o_baseAddr2
);
reg [6:0] featureIndex0;
reg [6:0] featureIndex1;
reg [6:0] featureIndex2;
always @(*) begin
    case (i_opcode)
      6'd0: begin
        featureIndex0 = 7'd3;
        featureIndex1 = 7'd67;
        featureIndex2 = 7'd67;
      end
      6'd1: begin
```

```
    featureIndex0 = 7'd4;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd2: begin
    featureIndex0 = 7'd5;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd3: begin
    featureIndex0 = 7'd6;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd4: begin
    featureIndex0 = 7'd7;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd5: begin
    featureIndex0 = 7'd8;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd6: begin
    featureIndex0 = 7'd9;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd7: begin
    featureIndex0 = 7'd10;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd8: begin
    featureIndex0 = 7'd11;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
  6'd9: begin
    featureIndex0 = 7'd12;
    featureIndex1 = 7'd67;
    featureIndex2 = 7'd67;
  end
```

```
    6'd10: begin
      featureIndex0 = 7'd13;
      featureIndex1 = 7'd67;
      featureIndex2 = 7'd67;
    end
    6'd11: begin
      featureIndex0 = 7'd14;
      featureIndex1 = 7'd67;
      featureIndex2 = 7'd67;
    end
    6'd12: begin
      featureIndex0 = 7'd15;
      featureIndex1 = 7'd67;
      featureIndex2 = 7'd67;
    end
    6'd13: begin
      featureIndex0 = 7'd16;
      featureIndex1 = 7'd67;
      featureIndex2 = 7'd67;
    end
    6'd14: begin
      featureIndex0 = 7'd17;
      featureIndex1 = 7'd67;
      featureIndex2 = 7'd67;
    end
    6'd15: begin
      featureIndex0 = 7'd18;
      featureIndex1 = 7'd67;
      featureIndex2 = 7'd67;
    end
    6'd16: begin
      featureIndex0 = 7'd19;
      featureIndex1 = 7'd20;
      featureIndex2 = 7'd21;
    end
    6'd17: begin
      featureIndex0 = 7'd22;
      featureIndex1 = 7'd23;
      featureIndex2 = 7'd24;
    end
    6'd18: begin
      featureIndex0 = 7'd25;
      featureIndex1 = 7'd26;
      featureIndex2 = 7'd27;
```

```
    end
    6'd19: begin
      featureIndex0 = 7'd28;
      featureIndex1 = 7'd29;
      featureIndex2 = 7'd30;
    end
    6'd20: begin
      featureIndex0 = 7'd31;
      featureIndex1 = 7'd32;
      featureIndex2 = 7'd33;
    end
    6'd21: begin
      featureIndex0 = 7'd34;
      featureIndex1 = 7'd35;
      featureIndex2 = 7'd36;
    end
    6'd22: begin
      featureIndex0 = 7'd37;
      featureIndex1 = 7'd38;
      featureIndex2 = 7'd39;
    end
    6'd23: begin
      featureIndex0 = 7'd40;
      featureIndex1 = 7'd41;
      featureIndex2 = 7'd42;
    end
    6'd24: begin
      featureIndex0 = 7'd43;
      featureIndex1 = 7'd44;
      featureIndex2 = 7'd45;
    end
    6'd25: begin
      featureIndex0 = 7'd46;
      featureIndex1 = 7'd47;
      featureIndex2 = 7'd48;
    end
    6'd26: begin
      featureIndex0 = 7'd49;
      featureIndex1 = 7'd50;
      featureIndex2 = 7'd51;
    end
    6'd27: begin
      featureIndex0 = 7'd52;
      featureIndex1 = 7'd53;
```

```
        featureIndex2 = 7'd54;
      end
      6'd28: begin
        featureIndex0 = 7'd55;
        featureIndex1 = 7'd56;
        featureIndex2 = 7'd57;
      end
      6'd29: begin
        featureIndex0 = 7'd58;
        featureIndex1 = 7'd59;
        featureIndex2 = 7'd60;
      end
      6'd30: begin
        featureIndex0 = 7'd61;
        featureIndex1 = 7'd62;
        featureIndex2 = 7'd63;
      end
      6'd31: begin
        featureIndex0 = 7'd64;
        featureIndex1 = 7'd65;
        featureIndex2 = 7'd66;
      end
      default: begin
        featureIndex0 = 7'd67;
        featureIndex1 = 7'd67;
        featureIndex2 = 7'd67;
      end
    endcase
end
assign o_baseAddr0 = featureIndex0 *19'd64*19'd64;
assign o_baseAddr1 = featureIndex1 *19'd64*19'd64;
assign o_baseAddr2 = featureIndex2 *19'd64*19'd64;
endmodule
```


Average:

```
module average(
  input [89:0] i_data,
  input      i_clk,
  input      i_reset,
  input      i_writeAdd,
  output [9:0] o_data
```

```
);
reg [21:0] data;
always @(posedge i_clk or negedge i_reset) begin
  if(!i_reset) begin
    data <= 22'd0;
  end else begin
    if(i_writeAdd) begin
      data <= data + i_data[9:0] + i_data[19:10] + i_data[29:20] + i_data[39:30] +
i_data[49:40] + i_data[59:50] + i_data[69:60] + i_data[79:70] + i_data[89:80];
    end else begin
      data <= data;
    end
  end
end
assign o_data = data[21:12];
endmodule
```


Average control:

```
module averageControl(
  input i_clk,
  input i_reset,
  input i_start,
  input i_validRam,
  output [107:0] o_addrRead0,
  output [107:0] o_addrRead1,
  output [107:0] o_addrRead2,
  output reg o_writeEnable,
  output reg o_resetAverage,
  output reg o_mask,
  output reg o_startRam,
  output reg o_finish
);
localparam [2:0]   idle = 3'd0,
                   reset = 3'd1,
                   buffer = 3'd2,
                   setAddr = 3'd3,
                   waitDone = 3'd4,
                   writeAdd = 3'd5,
                   update = 3'd6,
                   finish = 3'd7;
```

```
reg [2:0] state_q, state_d;
reg [11:0] addrLocal_q, addrLocal_d;
reg [11:0] addrRead0;
reg [11:0] addrRead1;
reg [11:0] addrRead2;
reg [11:0] addrRead3;
reg [11:0] addrRead4;
reg [11:0] addrRead5;
reg [11:0] addrRead6;
reg [11:0] addrRead7;
reg [11:0] addrRead8;

always @(posedge i_clk or negedge i_reset) begin
  if(!i_reset) begin
    state_q <= idle;
    addrLocal_q <= 12'd0;
  end else begin
    state_q <= state_d;
    addrLocal_q <= addrLocal_d;
  end
end

always @(*) begin
  case (state_q)
    idle: begin
      if(i_start & i_reset) begin
        state_d = reset;
      end else begin
        state_d = idle;
      end
      addrRead0 = 12'd0;
      addrRead1 = 12'd0;
      addrRead2 = 12'd0;
      addrRead3 = 12'd0;
      addrRead4 = 12'd0;
      addrRead5 = 12'd0;
      addrRead6 = 12'd0;
      addrRead7 = 12'd0;
      addrRead8 = 12'd0;
      o_writeEnable = 0;
      o_mask = 0;
      o_startRam = 0;
      o_finish = 0;
      o_resetAverage = 1;
```

```
        addrLocal_d = 12'd0;
    end
    reset: begin
        state_d = buffer;
        addrRead0 = 12'd0;
        addrRead1 = 12'd0;
        addrRead2 = 12'd0;
        addrRead3 = 12'd0;
        addrRead4 = 12'd0;
        addrRead5 = 12'd0;
        addrRead6 = 12'd0;
        addrRead7 = 12'd0;
        addrRead8 = 12'd0;
        o_writeEnable = 0;
        o_mask = 0;
        o_startRam = 0;
        o_finish = 0;
        o_resetAverage = 0;
        addrLocal_d = 12'd0;
    end
    buffer: begin
        state_d = setAddr;
        addrRead0 = 12'd0;
        addrRead1 = 12'd0;
        addrRead2 = 12'd0;
        addrRead3 = 12'd0;
        addrRead4 = 12'd0;
        addrRead5 = 12'd0;
        addrRead6 = 12'd0;
        addrRead7 = 12'd0;
        addrRead8 = 12'd0;
        o_writeEnable = 0;
        o_mask = 0;
        o_startRam = 0;
        o_finish = 0;
        o_resetAverage = 1;
        addrLocal_d = 12'd0;
    end
    setAddr: begin
        state_d = waitDone;
        addrRead0 = addrLocal_q;
        addrRead1 = addrLocal_q + 12'd1;
        addrRead2 = addrLocal_q + 12'd2;
        addrRead3 = addrLocal_q + 12'd3;
```

```
      addrRead4 = addrLocal_q + 12'd4;
      addrRead5 = addrLocal_q + 12'd5;
      addrRead6 = addrLocal_q + 12'd6;
      addrRead7 = addrLocal_q + 12'd7;
      addrRead8 = addrLocal_q + 12'd8;
      o_writeEnable = 0;
      if(addrLocal_q == 12'd4095) begin
        o_mask = 0;
      end else begin
        o_mask = 1;
      end
      o_startRam = 1;
      o_finish = 0;
      o_resetAverage = 1;
      addrLocal_d = addrLocal_q;
    end
    waitDone: begin
      if(i_validRam) begin
        state_d = writeAdd;
      end else begin
        state_d = waitDone;
      end
      addrRead0 = addrLocal_q;
      addrRead1 = addrLocal_q + 12'd1;
      addrRead2 = addrLocal_q + 12'd2;
      addrRead3 = addrLocal_q + 12'd3;
      addrRead4 = addrLocal_q + 12'd4;
      addrRead5 = addrLocal_q + 12'd5;
      addrRead6 = addrLocal_q + 12'd6;
      addrRead7 = addrLocal_q + 12'd7;
      addrRead8 = addrLocal_q + 12'd8;
      o_writeEnable = 0;
      if(addrLocal_q == 12'd4095) begin
        o_mask = 0;
      end else begin
        o_mask = 1;
      end
      o_startRam = 0;
      o_finish = 0;
      o_resetAverage = 1;
      addrLocal_d = addrLocal_q;
    end
    writeAdd: begin
      state_d = update;
```

```
        addrRead0 = addrLocal_q;
        addrRead1 = addrLocal_q + 12'd1;
        addrRead2 = addrLocal_q + 12'd2;
        addrRead3 = addrLocal_q + 12'd3;
        addrRead4 = addrLocal_q + 12'd4;
        addrRead5 = addrLocal_q + 12'd5;
        addrRead6 = addrLocal_q + 12'd6;
        addrRead7 = addrLocal_q + 12'd7;
        addrRead8 = addrLocal_q + 12'd8;
        o_writeEnable = 1;
        if(addrLocal_q == 12'd4095) begin
          o_mask = 0;
        end else begin
          o_mask = 1;
        end
        o_startRam = 0;
        o_finish = 0;
        o_resetAverage = 1;
        addrLocal_d = addrLocal_q;
      end
      update: begin
        addrRead0 = addrLocal_q;
        addrRead1 = addrLocal_q + 12'd1;
        addrRead2 = addrLocal_q + 12'd2;
        addrRead3 = addrLocal_q + 12'd3;
        addrRead4 = addrLocal_q + 12'd4;
        addrRead5 = addrLocal_q + 12'd5;
        addrRead6 = addrLocal_q + 12'd6;
        addrRead7 = addrLocal_q + 12'd7;
        addrRead8 = addrLocal_q + 12'd8;
        o_writeEnable = 0;
        if(addrLocal_q == 12'd4095) begin
          state_d = finish;
          o_mask = 0;
        end else begin
          state_d = setAddr;
          o_mask = 1;
        end
        o_startRam = 0;
        o_finish = 0;
        o_resetAverage = 1;
        addrLocal_d = addrLocal_q + 12'd9;
      end
      finish: begin
```

```
        state_d = idle;
        addrRead0 = 12'd0;
        addrRead1 = 12'd0;
        addrRead2 = 12'd0;
        addrRead3 = 12'd0;
        addrRead4 = 12'd0;
        addrRead5 = 12'd0;
        addrRead6 = 12'd0;
        addrRead7 = 12'd0;
        addrRead8 = 12'd0;
        o_writeEnable = 0;
        o_mask = 0;
        o_startRam = 0;
        o_finish = 1;
        o_resetAverage = 1;
        addrLocal_d = 12'd0;
      end
      default: begin
        state_d = idle;
        addrRead0 = 12'd0;
        addrRead1 = 12'd0;
        addrRead2 = 12'd0;
        addrRead3 = 12'd0;
        addrRead4 = 12'd0;
        addrRead5 = 12'd0;
        addrRead6 = 12'd0;
        addrRead7 = 12'd0;
        addrRead8 = 12'd0;
        o_writeEnable = 0;
        o_mask = 0;
        o_startRam = 0;
        o_finish = 0;
        o_resetAverage = 1;
        addrLocal_d = 12'd0;
      end
    endcase
end

assign o_addrRead0 = {addrRead8, addrRead7, addrRead6, addrRead5, addrRead4,
addrRead3, addrRead2, addrRead1, addrRead0};
assign o_addrRead1 = {addrRead8, addrRead7, addrRead6, addrRead5, addrRead4,
addrRead3, addrRead2, addrRead1, addrRead0};
assign o_addrRead2 = {addrRead8, addrRead7, addrRead6, addrRead5, addrRead4,
addrRead3, addrRead2, addrRead1, addrRead0};
```

```
endmodule
```


Mask:

```
module mask(
   input i_mask,
   input [89:0] i_data,
   output [89:0] o_data
);
assign o_data[9:0] = i_data[9:0];
assign o_data[19:10] = {10{i_mask}} & i_data[19:10];
assign o_data[29:20] = {10{i_mask}} & i_data[29:20];
assign o_data[39:30] = {10{i_mask}} & i_data[39:30];
assign o_data[49:40] = {10{i_mask}} & i_data[49:40];
assign o_data[59:50] = {10{i_mask}} & i_data[59:50];
assign o_data[69:60] = {10{i_mask}} & i_data[69:60];
assign o_data[79:70] = {10{i_mask}} & i_data[79:70];
assign o_data[89:80] = {10{i_mask}} & i_data[89:80];

endmodule
```

Register average:

```
//incomplete
module registerAverage(
   input i_clk,
   input i_reset,
   input i_enableWrite,
   input [15:0] i_selWrite,
   input [9:0] i_data0,
   input [9:0] i_data1,
   input [9:0] i_data2,
   output [159:0] o_ave
);
reg [9:0] register [0:15];
always @(posedge i_clk or negedge i_reset) begin
   if(!i_reset) begin
      register[0] <= 10'd0;
      register[1] <= 10'd0;
      register[2] <= 10'd0;
```

```
    register[3] <= 10'd0;
    register[4] <= 10'd0;
    register[5] <= 10'd0;
    register[6] <= 10'd0;
    register[7] <= 10'd0;
    register[8] <= 10'd0;
    register[9] <= 10'd0;
    register[10] <= 10'd0;
    register[11] <= 10'd0;
    register[12] <= 10'd0;
    register[13] <= 10'd0;
    register[14] <= 10'd0;
    register[15] <= 10'd0;
  end else begin
    if(i_enableWrite) begin
      case (i_selWrite)
        16'b0000_0000_0000_0111: begin
          register[0] <= i_data0;
          register[1] <= i_data1;
          register[2] <= i_data2;
        end
        16'b0000_0000_0011_1000: begin
          register[3] <= i_data0;
          register[4] <= i_data1;
          register[5] <= i_data2;
        end
        16'b0000_0001_1100_0000: begin
          register[6] <= i_data0;
          register[7] <= i_data1;
          register[8] <= i_data2;
        end
        16'b0000_1110_0000_0000: begin
          register[9] <= i_data0;
          register[10] <= i_data1;
          register[11] <= i_data2;
        end
        16'b0111_0000_0000_0000: begin
          register[12] <= i_data0;
          register[13] <= i_data1;
          register[14] <= i_data2;
        end
        16'b1000_0000_0000_0000: begin
          register[15] <= i_data0;
        end
```

```
            default: ;
          endcase
        end
    end
end
assign o_ave = {register[15],
            register[14],
            register[13],
            register[12],
            register[11],
            register[10],
            register[9],
            register[8],
            register[7],
            register[6],
            register[5],
            register[4],
            register[3],
            register[2],
            register[1],
            register[0]};
endmodule
```

Fully connected:

```
module fullyConnected (
    input           i_clk,
    input           i_reset,
    input           i_enable,
    input  [159:0]    i_data,    // 16 x 10-bit packed inputs Q3.6
    output reg [19:0]  o_result
);

    // Input unpacking
    wire signed [9:0] in_data0 ;
    assign in_data0 = i_data[9:0];
    wire signed [9:0] in_data1 ;
    assign in_data1 = i_data[19:10];
    wire signed [9:0] in_data2 ;
    assign in_data2 = i_data[29:20];
    wire signed [9:0] in_data3 ;
    assign in_data3 = i_data[39:30];
```

```
  wire signed [9:0] in_data4 ;
  assign in_data4 = i_data[49:40];
  wire signed [9:0] in_data5 ;
  assign in_data5 = i_data[59:50];
  wire signed [9:0] in_data6 ;
  assign in_data6 = i_data[69:60];
  wire signed [9:0] in_data7 ;
  assign in_data7 = i_data[79:70];
  wire signed [9:0] in_data8 ;
  assign in_data8 = i_data[89:80];
  wire signed [9:0] in_data9 ;
  assign in_data9 = i_data[99:90];
  wire signed [9:0] in_data10;
  assign in_data10 = i_data[109:100];
  wire signed [9:0] in_data11;
  assign in_data11 = i_data[119:110];
  wire signed [9:0] in_data12;
  assign in_data12 = i_data[129:120];
  wire signed [9:0] in_data13;
  assign in_data13 = i_data[139:130];
  wire signed [9:0] in_data14;
  assign in_data14 = i_data[149:140];
  wire signed [9:0] in_data15;
  assign in_data15 = i_data[159:150];

  // Weights
  reg signed [9:0] w0 = -10'd257, w1 = -10'd292, w2 = -10'd115, w3 = 10'd143, w4 =
10'd269, w5 = -10'd25, w6 = 10'd179, w7 = 10'd44;
  reg signed [9:0] w8 = -10'd175, w9 = 10'd300, w10 = 10'd168, w11 = 10'd241, w12 = -
10'd10, w13 = -10'd132, w14 = -10'd139, w15 = 10'd86;
  reg signed [9:0] w_bias = -10'd107;

  // Sequential multiply-accumulate using pairing
  reg signed [19:0] sum;

  always @(posedge i_clk or posedge i_reset) begin
    if (i_reset) begin
      o_result <= 0;
    end else if (i_enable) begin
      sum = 0;
      sum = sum +
        $signed(in_data0)  * w0  +
        $signed(in_data1)  * w1;
      sum = sum +
```

```
            $signed(in_data2)  * w2  +
            $signed(in_data3)  * w3;
        sum = sum +
            $signed(in_data4)  * w4  +
            $signed(in_data5)  * w5;
        sum = sum +
            $signed(in_data6)  * w6  +
            $signed(in_data7)  * w7;
        sum = sum +
            $signed(in_data8)  * w8  +
            $signed(in_data9)  * w9;
        sum = sum +
            $signed(in_data10) * w10 +
            $signed(in_data11) * w11;
        sum = sum +
            $signed(in_data12) * w12 +
            $signed(in_data13) * w13;
        sum = sum +
            $signed(in_data14) * w14 +
            $signed(in_data15) * w15;
        sum = sum + w_bias;
        o_result <= sum;
      end
    end

endmodule
```


Master control:

```
module masterControl(
    input i_clk,
    input i_reset,
    input i_finish_cam,
    input i_finish_fet,
    input i_finish_conv,
    input i_finish_writeBack,
    input i_finish_ave,
    output reg o_startCam,
    output reg o_startFet,
    output reg o_startConvolution,
    output reg o_startAve,
    output reg o_startWriteBack,
    output reg o_wrActiveCam,
```

```
  output reg o_rdActiveConvolution,
  output reg o_opConv,
  output [5:0] o_opcode,
  output reg o_enableFullyConnected
);
localparam [4:0]   idle = 5'd0,
                   startCam = 5'd1,
                   waitDoneCam = 5'd2,
                   fet1 = 5'd3,
                   waitDoneFet1 = 5'd4,
                   convolution0 = 5'd5,
                   waitDoneConv0 = 5'd6,
                   writeBack1 = 5'd7,
                   waitDoneWriteBack1 = 5'd8,
                   update1 = 5'd9,
                   fet2 = 5'd10,
                   waitDoneFet2 = 5'd11,
                   convolution1 = 5'd12,
                   waitDoneConv1 = 5'd13,
                   writeBack2 = 5'd14,
                   waitDoneWriteBack2 = 5'd15,
                   update2 = 5'd16,
                   fet3 = 5'd17,
                   waitDoneFet3 = 5'd18,
                   average = 5'd19,
                   waitDoneAve = 5'd20,
                   update3 = 5'd21;
reg [4:0] state_q, state_d;
reg [5:0] opcode_q, opcode_d;

always @(posedge i_clk or negedge i_reset) begin
  if(!i_reset) begin
    state_q <= idle;
    opcode_q <= 6'd0;
  end else begin
    state_q <= state_d;
    opcode_q <= opcode_d;
  end
end

always @(*) begin
  case (state_q)
    idle: begin
      state_d = startCam;
```

```
        o_startCam = 0;
        o_startFet = 0;
        o_startConvolution = 0;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = 6'd0;
        o_rdActiveConvolution = 0;
        o_enableFullyConnected = 1;
      end
      startCam: begin
        state_d = waitDoneCam;
        o_startCam = 1;
        o_startFet = 0;
        o_startConvolution = 0;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = opcode_q;
        o_rdActiveConvolution = 1;
        o_enableFullyConnected = 1;
      end
      waitDoneCam: begin
        if(i_finish_cam) begin
          state_d = fet1;
        end else begin
          state_d = waitDoneCam;
        end
        o_startCam = 0;
        o_startFet = 0;
        o_startConvolution = 0;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = opcode_q;
        o_rdActiveConvolution = 1;
        o_enableFullyConnected = 1;
      end
      fet1: begin
        state_d = waitDoneFet1;
        o_startCam = 0;
```

```
        o_startFet = 1;
        o_startConvolution = 0;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = opcode_q;
        o_rdActiveConvolution = 1;
        o_enableFullyConnected = 1;
      end
      waitDoneFet1: begin
        if(i_finish_fet) begin
          state_d = convolution0;
        end else begin
          state_d = waitDoneFet1;
        end
        o_startCam = 0;
        o_startFet = 0;
        o_startConvolution = 0;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = opcode_q;
        o_rdActiveConvolution = 1;
        o_enableFullyConnected = 1;
      end
      convolution0: begin
        state_d = waitDoneConv0;
        o_startCam = 0;
        o_startFet = 0;
        o_startConvolution = 1;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = opcode_q;
        o_rdActiveConvolution = 1;
        o_enableFullyConnected = 1;
      end
      waitDoneConv0: begin
        if(i_finish_conv) begin
          state_d = writeBack1;
        end else begin
```

```
        state_d = waitDoneConv0;
      end
      o_startCam = 0;
      o_startFet = 0;
      o_startConvolution = 0;
      o_startAve = 0;
      o_startWriteBack = 0;
      o_wrActiveCam = 0;
      o_opConv = 0;
      opcode_d = opcode_q;
      o_rdActiveConvolution = 1;
      o_enableFullyConnected = 1;
    end
    writeBack1: begin
      state_d = waitDoneWriteBack1;
      o_startCam = 0;
      o_startFet = 0;
      o_startConvolution = 0;
      o_startAve = 0;
      o_startWriteBack = 1;
      o_wrActiveCam = 0;
      o_opConv = 0;
      opcode_d = opcode_q;
      o_rdActiveConvolution = 1;
      o_enableFullyConnected = 1;
    end
    waitDoneWriteBack1: begin
      if(i_finish_writeBack) begin
        state_d = update1;
      end else begin
        state_d = waitDoneWriteBack1;
      end
      o_startCam = 0;
      o_startFet = 0;
      o_startConvolution = 0;
      o_startAve = 0;
      o_startWriteBack = 0;
      o_wrActiveCam = 0;
      o_opConv = 0;
      opcode_d = opcode_q;
      o_rdActiveConvolution = 1;
      o_enableFullyConnected = 1;
    end
    update1: begin
```

```
    if(opcode_q == 6'd15) begin
      state_d = fet2;
    end else begin
      state_d = convolution0;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q + 6'd1;
    o_rdActiveConvolution = 1;
    o_enableFullyConnected = 1;
  end
  fet2: begin
    state_d = waitDoneFet2;
    o_startCam = 0;
    o_startFet = 1;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
    o_rdActiveConvolution = 1;
    o_enableFullyConnected = 1;
  end
  waitDoneFet2: begin
    if(i_finish_fet) begin
      state_d = convolution1;
    end else begin
      state_d = waitDoneFet2;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
    o_rdActiveConvolution = 1;
```

```
    o_enableFullyConnected = 1;
  end
  convolution1: begin
    state_d = waitDoneConv1;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 1;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
    o_rdActiveConvolution = 1;
    o_enableFullyConnected = 1;
  end
  waitDoneConv1: begin
    if(i_finish_conv) begin
      state_d = writeBack2;
    end else begin
      state_d = waitDoneConv1;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
    o_rdActiveConvolution = 1;
    o_enableFullyConnected = 1;
  end
  writeBack2: begin
    state_d = waitDoneWriteBack2;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 1;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
    o_rdActiveConvolution = 1;
    o_enableFullyConnected = 1;
```

```
    end
    waitDoneWriteBack2: begin
      if(i_finish_writeBack) begin
        state_d = update2;
      end else begin
        state_d = waitDoneWriteBack2;
      end
      o_startCam = 0;
      o_startFet = 0;
      o_startConvolution = 0;
      o_startAve = 0;
      o_startWriteBack = 0;
      o_wrActiveCam = 0;
      o_opConv = 1;
      opcode_d = opcode_q;
      o_rdActiveConvolution = 1;
      o_enableFullyConnected = 1;
    end
    update2: begin
      if(opcode_q == 6'd31) begin
        state_d = fet3;
      end else begin
        state_d = fet2;
      end
      o_startCam = 0;
      o_startFet = 0;
      o_startConvolution = 0;
      o_startAve = 0;
      o_startWriteBack = 0;
      o_wrActiveCam = 0;
      o_opConv = 1;
      opcode_d = opcode_q + 6'd1;
      o_rdActiveConvolution = 1;
      o_enableFullyConnected = 1;
    end
    fet3: begin
      state_d = waitDoneFet3;
      o_startCam = 0;
      o_startFet = 1;
      o_startConvolution = 0;
      o_startAve = 0;
      o_startWriteBack = 0;
      o_wrActiveCam = 0;
      o_opConv = 0;
```

```
    opcode_d = opcode_q;
    o_rdActiveConvolution = 0;
    o_enableFullyConnected = 0;
  end
  waitDoneFet3: begin
    if(i_finish_fet) begin
      state_d = average;
    end else begin
      state_d = waitDoneFet3;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
    o_rdActiveConvolution = 0;
    o_enableFullyConnected = 0;
  end
  average: begin
    state_d = waitDoneAve;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 1;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
    o_rdActiveConvolution = 0;
    o_enableFullyConnected = 0;
  end
  waitDoneAve: begin
    if(i_finish_ave) begin
      state_d = update3;
    end else begin
      state_d = waitDoneAve;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
```

```
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = opcode_q;
        o_rdActiveConvolution = 0;
        o_enableFullyConnected = 0;
      end
      update3: begin
        if(opcode_q == 6'd37) begin
          state_d = idle;
        end else begin
          state_d = fet3;
        end
        o_startCam = 0;
        o_startFet = 0;
        o_startConvolution = 0;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = opcode_q + 6'd1;
        o_rdActiveConvolution = 0;
        o_enableFullyConnected = 0;
      end
      default: begin
        state_d = idle;
        o_startCam = 0;
        o_startFet = 0;
        o_startConvolution = 0;
        o_startAve = 0;
        o_startWriteBack = 0;
        o_wrActiveCam = 0;
        o_opConv = 0;
        opcode_d = 6'd0;
        o_rdActiveConvolution = 0;
        o_enableFullyConnected = 0;
      end
    endcase
end
assign o_opcode = opcode_q;
endmodule
```