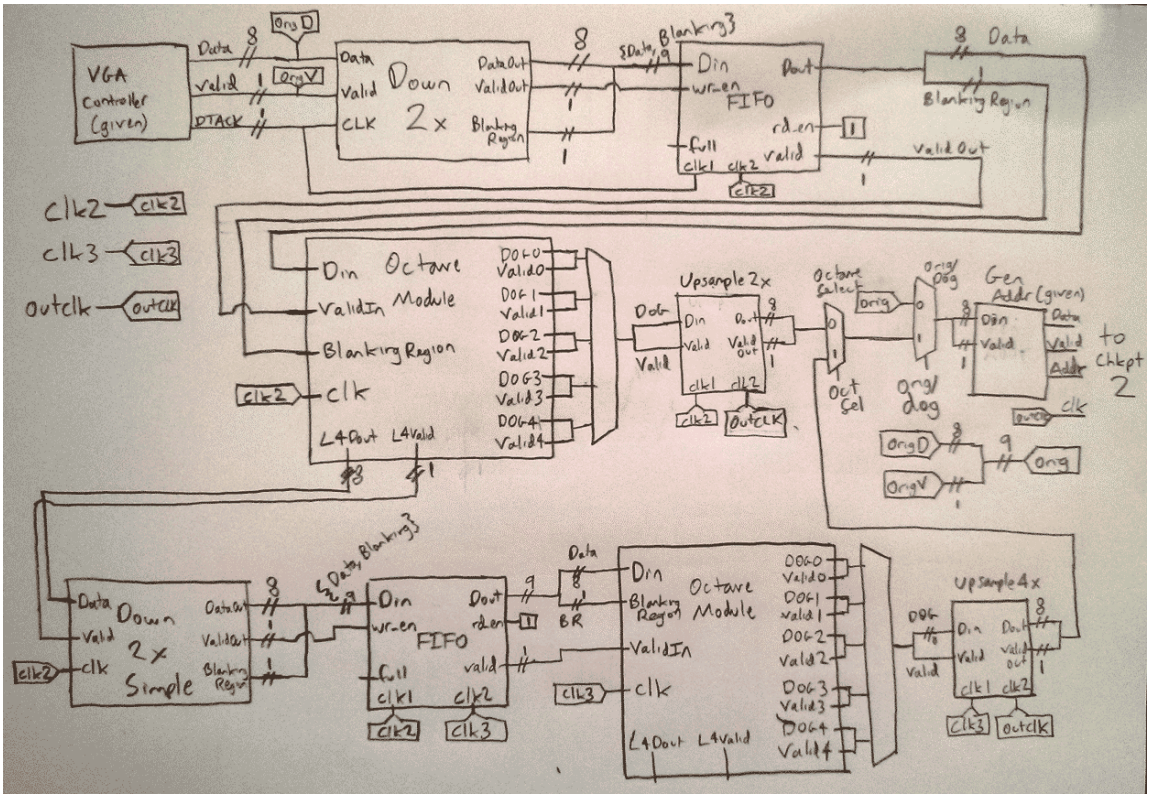
CS150 Checkpoint 3 Proposal, Team 07

Sahar Mesri, Sagar Karandikar, cs150-bw, cs150-bn

November 14, 2013

1. Block Diagrams

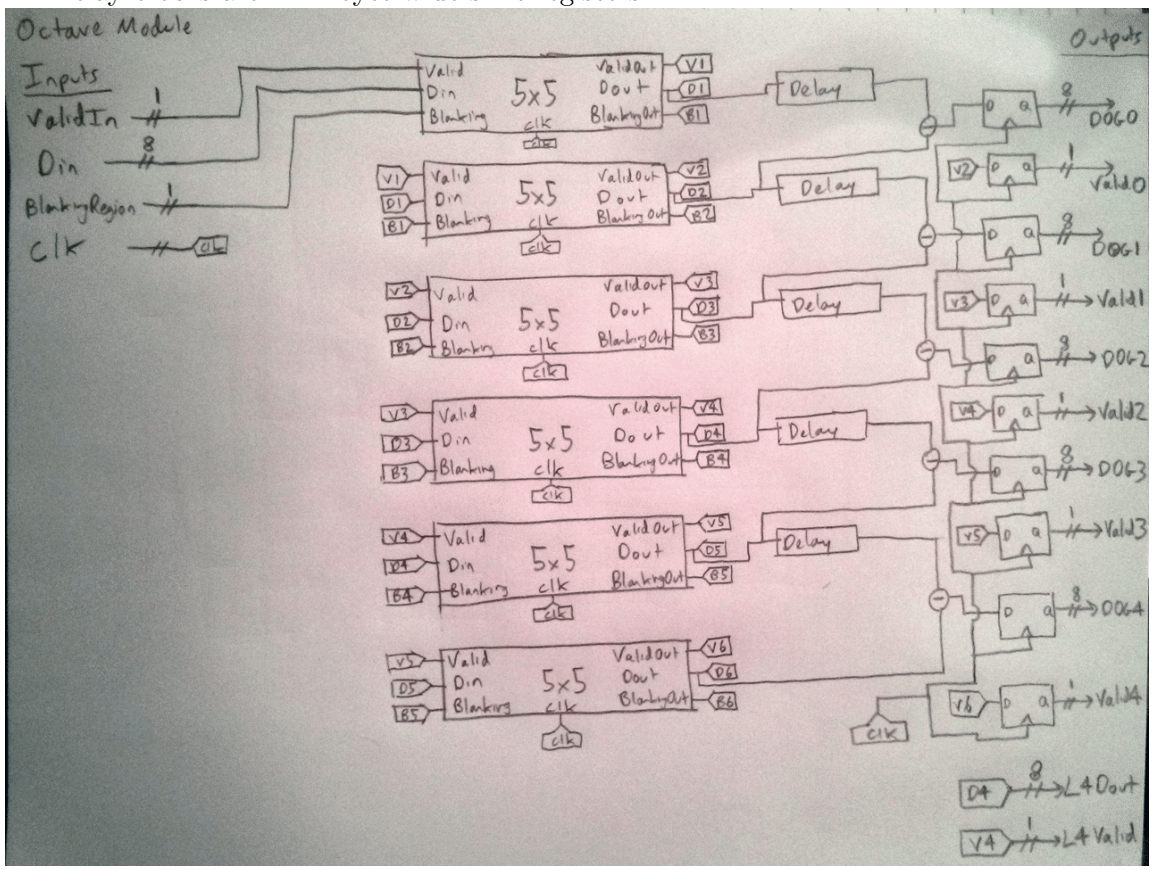
Overall System



a. Gaussian Filter Banks

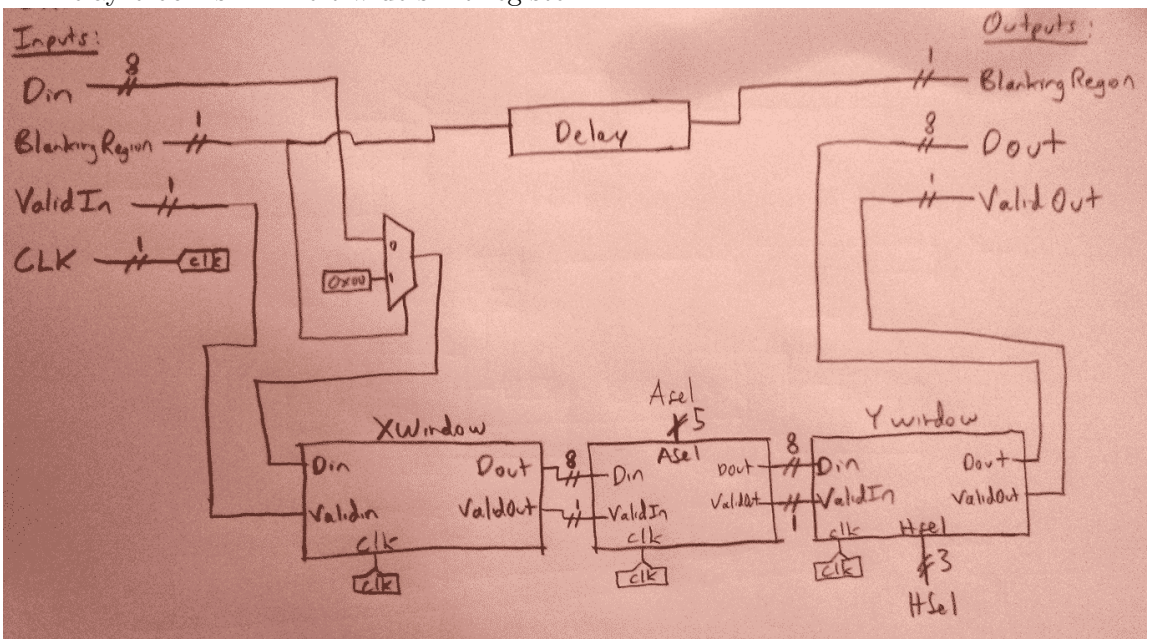
Octave Module:

Delay blocks are 2112 byte wide shift registers.



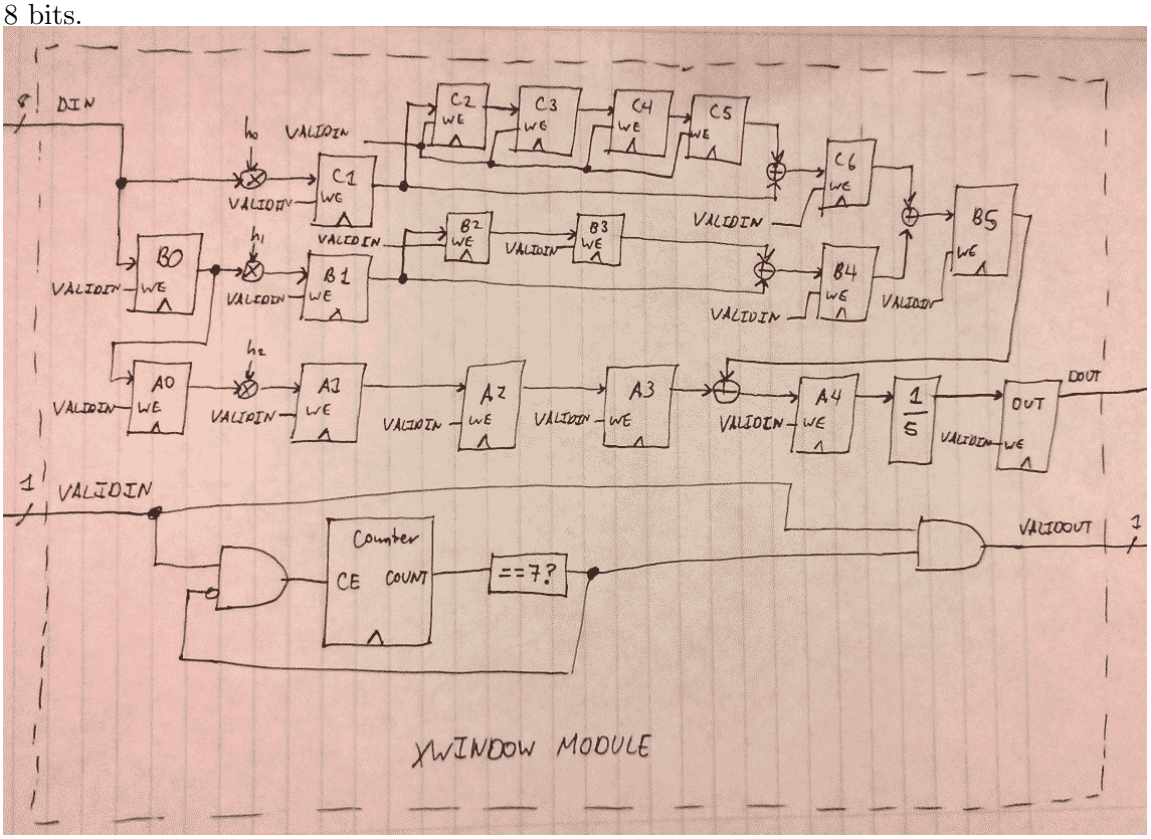
5x5 Window Module:

Delay block is 2112 bit wide shift register.



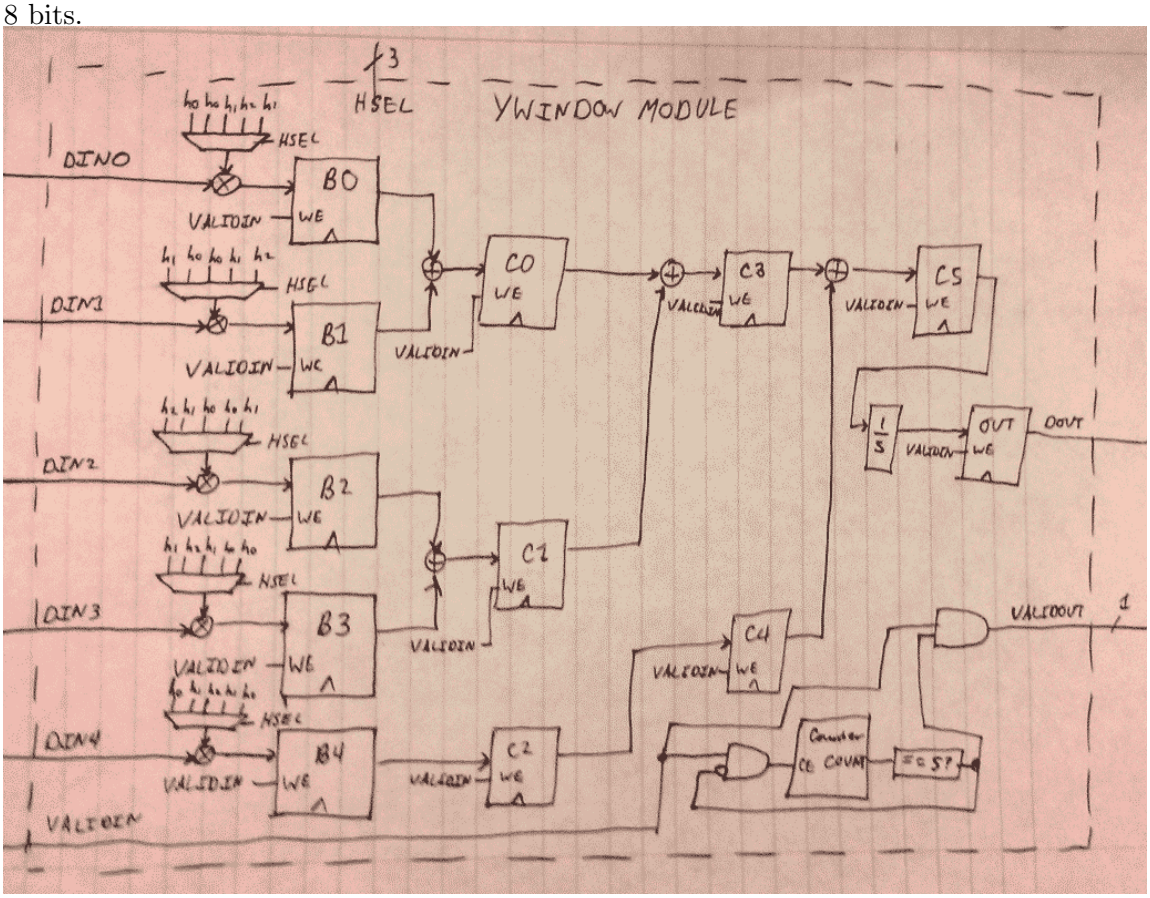
X Window Module:

Data lines after multiplication blocks are all 16 bits wide, DOUT is cut back down to 8 bits.

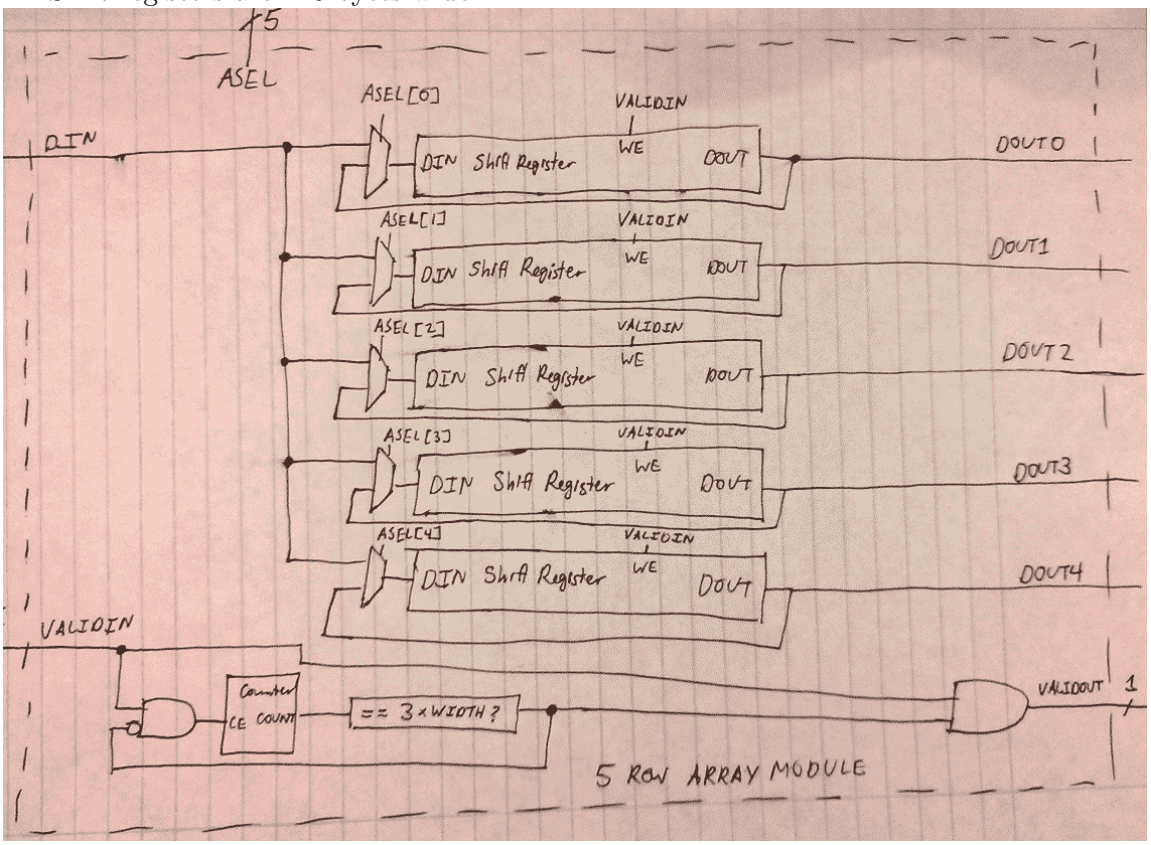


Y Window Module:

Data lines after multiplication blocks are all 16 bits wide, DOUT is cut back down to 8 bits.

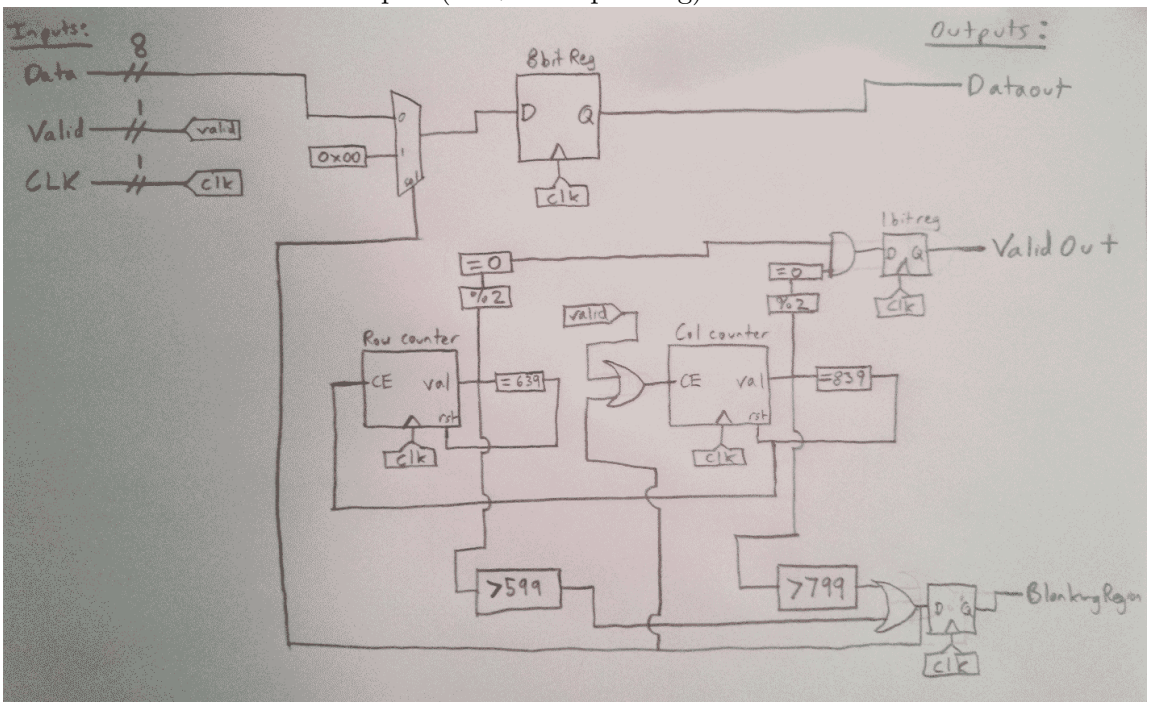


5 Row Array Module:
Shift registers are 420 bytes wide.

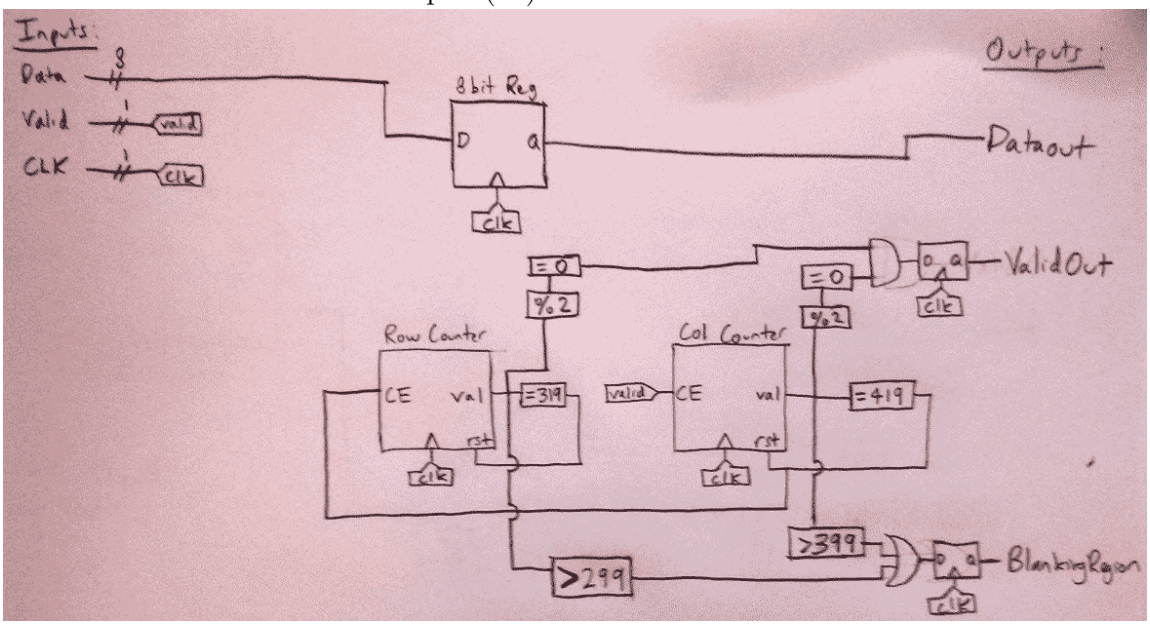


b. Downsampling

800x600 to 420x320 downsampler (2x + adds padding):

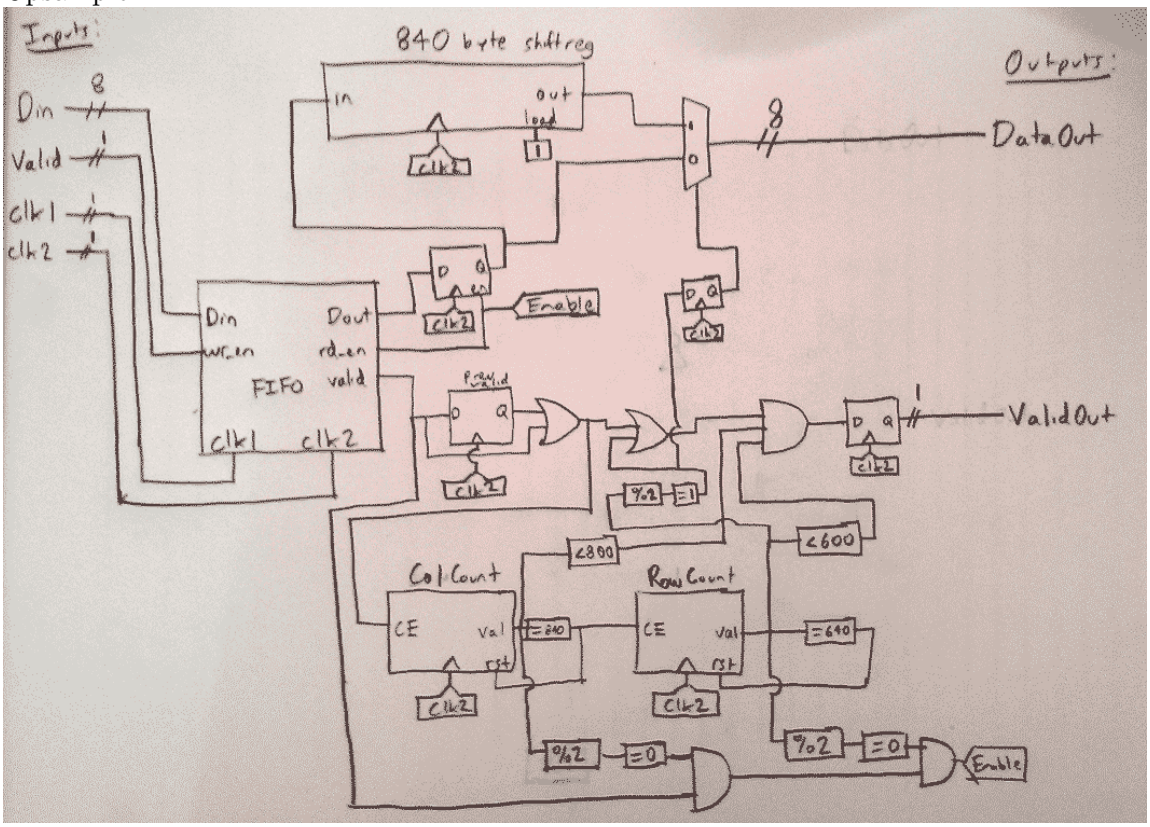


420x320 to 210x160 downsampler (2x)

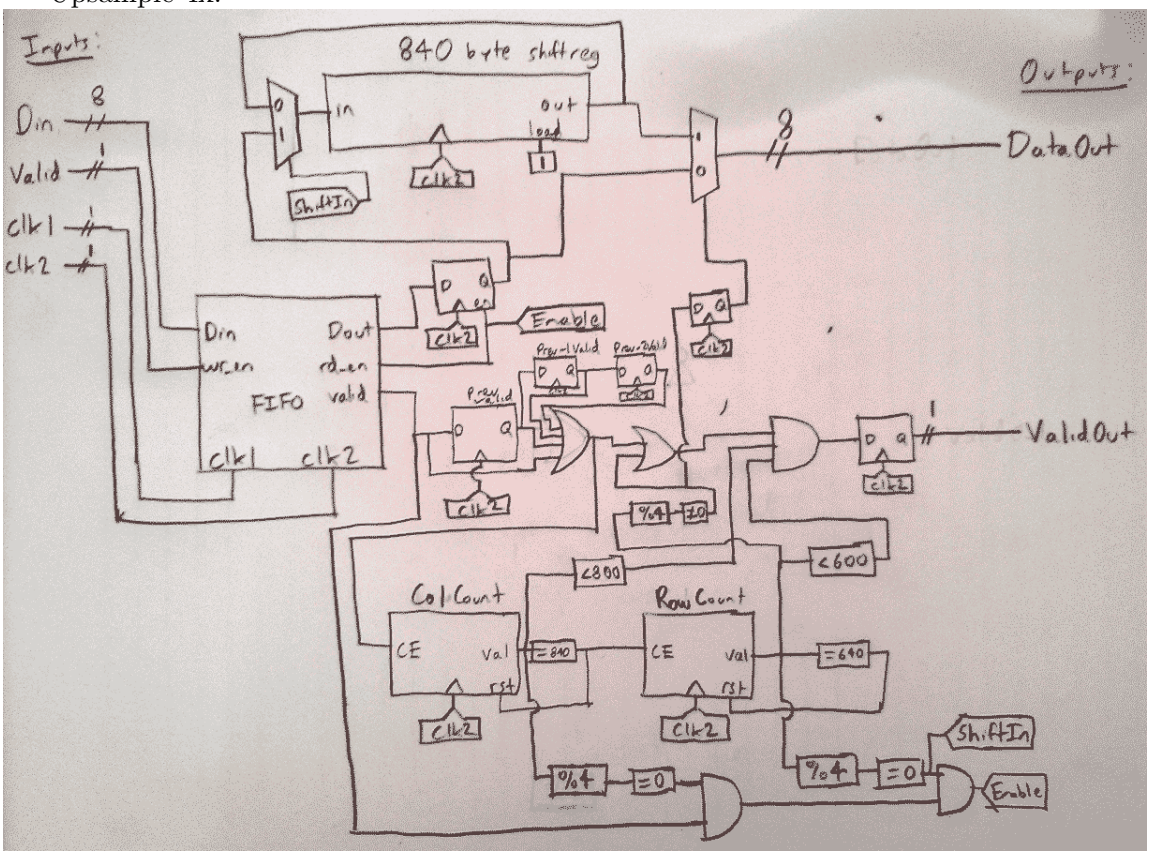


c. Upsampling

Upsample 2x:



Upsample 4x:



d. Interface Connections

(See overall diagram)

2. List of Signals

Datapath Inputs

8 bit Data In (from VGA controller)

1 bit Valid (from VGA controller)

1 bit DTACK (from VGA controller)

3 external clocks (not necessarily different clocks, but can be to reduce stalls)

1 bit out_sel: selects which octave to output if orig/dog = dog

1 bit orig/dog: 0 indicates output VGA input, 1 indicates output dog octave indicated by out_sel

5 bit ASEL into 5 Row Array Module

3 bit HSEL into Y Window Module

Datapath Outputs

8 bit Data Out
1 bit Valid Out
18 bit Address Out (from Address Generator block)
1 bit output clock (for checkpoint 2 write clock)

3. RT Language Description

Octave Module

```
while(true){
    Dog0ShiftReg <- (D1 << 2111*8) | (Dog0ShiftReg >> 8),
    Dog1ShiftReg <- (D2 << 2111*8) | (Dog1ShiftReg >> 8),
    Dog2ShiftReg <- (D3 << 2111*8) | (Dog2ShiftReg >> 8),
    Dog3ShiftReg <- (D4 << 2111*8) | (Dog3ShiftReg >> 8),
    Dog4ShiftReg <- (D5 << 2111*8) | (Dog4ShiftReg >> 8),
    Dog0Reg <- Dog0ShiftReg - D2,
    Dog1Reg <- Dog1ShiftReg - D3,
    Dog2Reg <- Dog2ShiftReg - D4,
    Dog3Reg <- Dog3ShiftReg - D5,
    Dog4Reg <- Dog4ShiftReg - D6,
    Valid0Reg <- V2,
    Valid1Reg <- V3,
    Valid2Reg <- V4,
    Valid3Reg <- V5,
    Valid4Reg <- V6;
}
```

5x5 Window Module

```
while(true){
    DelayShiftReg <- (BlankingRegion << 2111) | (DelayShiftReg >>
        1),
    XWindow Ops, 5x5 Array Ops, YWindow Ops;
}
```

X Window Module

```
while(true){
    if(ValidIn) {
        B0 <- Din, A0 <- B0, C1 <- h0*Din, B1 <- h1*B0,
        A1 <- h2*A0, C2 <- C1, C3 <- C2, C4 <- C3,
        C5 <- C4, C6 <- C5 + C1, B2 <- B1, B3 <- B2,
        B4 <- B3 + B1, B5 <- C6 + B4, A2 <- A1,
        A3 <- A2, A4 <- B5 + A3, Out <- 1/5 * A4,
        Counter <- Counter + 1 if Counter != 7;
    }
}
```

Y Window Module

```
while(true){
    if(ValidIn){
        B0 <- coeff*DIN0,
        B1 <- coeff*DIN1,
        B2 <- coeff*DIN2,
        B3 <- coeff*DIN3,
        B4 <- coeff*DIN4,
        C0 <- B0 + B1, C1 <- B2 + B3,
        C2 <- B4, C3 <- C0 + C1, C4 <- C2,
        C5 <- C3 + C4, Out <- 1/5*C5,
        Counter = Counter + 1 if Counter != 5;
    }
}
```

5 Row Array Module

```
while(true){
    if(ValidIn){
        ShiftReg0 <- {(DIN if ASEL[0] else (ShiftReg0 & 0xFF)),
            (ShiftReg0 >> 8)},
        ShiftReg1 <- {(DIN if ASEL[1] else (ShiftReg1 & 0xFF)),
            (ShiftReg1 >> 8)},
        ShiftReg2 <- {(DIN if ASEL[2] else (ShiftReg2 & 0xFF)),
            (ShiftReg2 >> 8)},
        ShiftReg3 <- {(DIN if ASEL[3] else (ShiftReg3 & 0xFF)),
            (ShiftReg3 >> 8)},
        ShiftReg4 <- {(DIN if ASEL[4] else (ShiftReg4 & 0xFF)),
            (ShiftReg4 >> 8)},
```

```
        Counter = Counter + 1 if Counter != 3*WIDTH;
    }
}
```

2x Downsampler

```
while(true){
    DataoutReg <- 0x00 if PrevBlankingRegion else Data,
    RowCounter <- RowCounter + 1 if ColCounter == 839 else
        RowCounter,
    ColCounter <- ColCounter + 1 if valid & PrevBlankingRegion
        else ColCounter,
    ValidOutReg <- ColCounter % 2 == 0 && RowCounter % 2 == 0;
}
```

2x Downsampler Simple

```
while(true){
    DataoutReg <- Data,
    RowCounter <- RowCounter + 1 if ColCounter == 419 else
        RowCounter,
    ColCounter <- ColCounter + 1 if valid else ColCounter,
    ValidOutReg <- ColCounter % 2 == 0 && RowCounter % 2 == 0;
}
```

2x Upsample

```
while(true){
    DoutReg <- Dout,
    PrevValid <- valid, ColCount <- ColCount + 1 if valid or
        PrevValid else ColCount,
    RowCount <- RowCount + 1 if ColCount == 840 else RowCount,
    ValidOutReg <- (PrevValid | Valid | (RowCount % 2 == 1)) &
        ColCount < 800 & RowCount < 600,
    MuxSelReg <- RowCount % 2 == 1,
    ShiftReg <- (ShiftReg >> 8) | (DoutReg << 839*8);
}
```

4x Upsample

```
while(true){
    DoutReg <- Dout,
    PrevValid <- valid,
```

```
    ColCount <- ColCount + 1 if valid or PrevValid else ColCount,
    RowCount <- RowCount + 1 if ColCount == 840 else RowCount,
    ValidOutReg <- (PrevValid | Valid | (RowCount % 4 != 0) |
        Prev-1Valid | Prev-2Valid) & ColCount < 800 & RowCount <
        600,
    MuxSelReg <- RowCount % 4 != 0, ShiftReg <- (ShiftReg >> 8) |
         (DoutReg << 839*8) if ShiftIn else (ShiftReg >> 8) | (
        ShiftReg << 839*8),
    Prev-1Valid <- PrevValid,
    Prev-2Valid <- Prev-1Valid;
}
```

4. Controller State Diagram

The pipeline does not have a single controller, rather the individual modules inside each contain their own controllers, whose states are maintained using counters (a row and column counter).

5. Testbench Outline

Overall System:

- Feed in 800x600 pixel image, simulating inputs from VGA controller, check that outputs are correct DoG.
- Output correctness can be tested using existing python script.

Octave Module:

- Feed in 5x5 window of data, confirm that 5 DoGs are computed correctly.
- Potential failure modes:
  - Delay length for alignment of DoG computation.
  - Reacting to deassertion of validIn.
  - Ensure correct valid signal propagation for display.

5x5 Window Module:

- Feed in 5x5 window of data, confirm 2D convolution output.
- Potential failure modes:
  - Correct coefficient selection (FSM will go here).

  - Delay length for BlankingRegion.
  - Reacting to deassertion of validIn.
  - Ensure correct handling of BlankingRegion (should force input to zero in the region).

X Window Module:

- Feed in row at a time of video signal and confirm horizontal convolution output.
- Potential failure modes:
  - Correct coefficient selection.

Y Window Module:

- Feed in column at a time of video signal (columns that are 5 pixels tall) and confirm vertical convolution output.
- Potential failure modes:
  - Correct coefficient selection.

5 Row Array Module:

- Feed in first 5 lines of 420x320 AND 210x160 signal, should recieve one "column" out per cycle, consisting of 5 elements. This should continue for 420 cycles for 420x320 and 210 cycles for 210x160.
- Potential failure modes:
  - Reacting to Valid going low (potential for incorrect values to enter shift registers?).

800x600 to 420x320 downsampler (2x + adds padding):

- Feed in 800x600 signal, should recieve 420x320 signal out
- Potential failure modes:
  - Off by one errors in counter logic.
  - Reacting to Valid going low.
  - Case where pipeline consuming data much slower than VGA provides it (FIFO overflow).
  - Potential for errors in adding artifical blanking region.

  - Guaranteed failure if input VGA blanking period is not at least 40px in each direction.

420x320 to 210x160 downsampler (2x):

- Feed in 420x320 signal, should recieve 210x160 signal out
- Potential failure modes:
  - Off by one errors in counter logic.
  - Reacting to Valid going low.
  - Case where pipeline consuming data much slower than VGA provides it (FIFO overflow).

Upsample 2x:

- Feed in 420x320 signal, should recieve 800x400 signal out
- Potential failure modes:
  - Off by one errors in switching between "live" input and shift register values.
  - Reacting to Valid going low.
  - Case where clock 1 faster than clock 2 (FIFO overflow).

Upsample 4x:

- Feed in 210x160 signal, should recieve 800x400 signal out
- Potential failure modes:
  - Off by one errors in switching between "live" input and shift register values.
  - Reacting to Valid going low.
  - Case where clock 1 faster than clock 2 (FIFO overflow).

Interface Connections