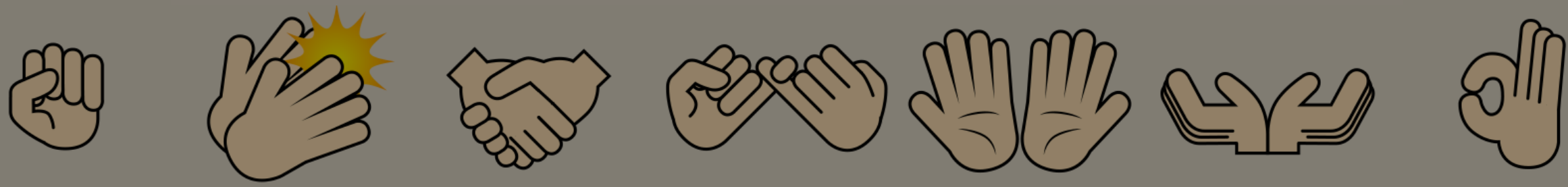
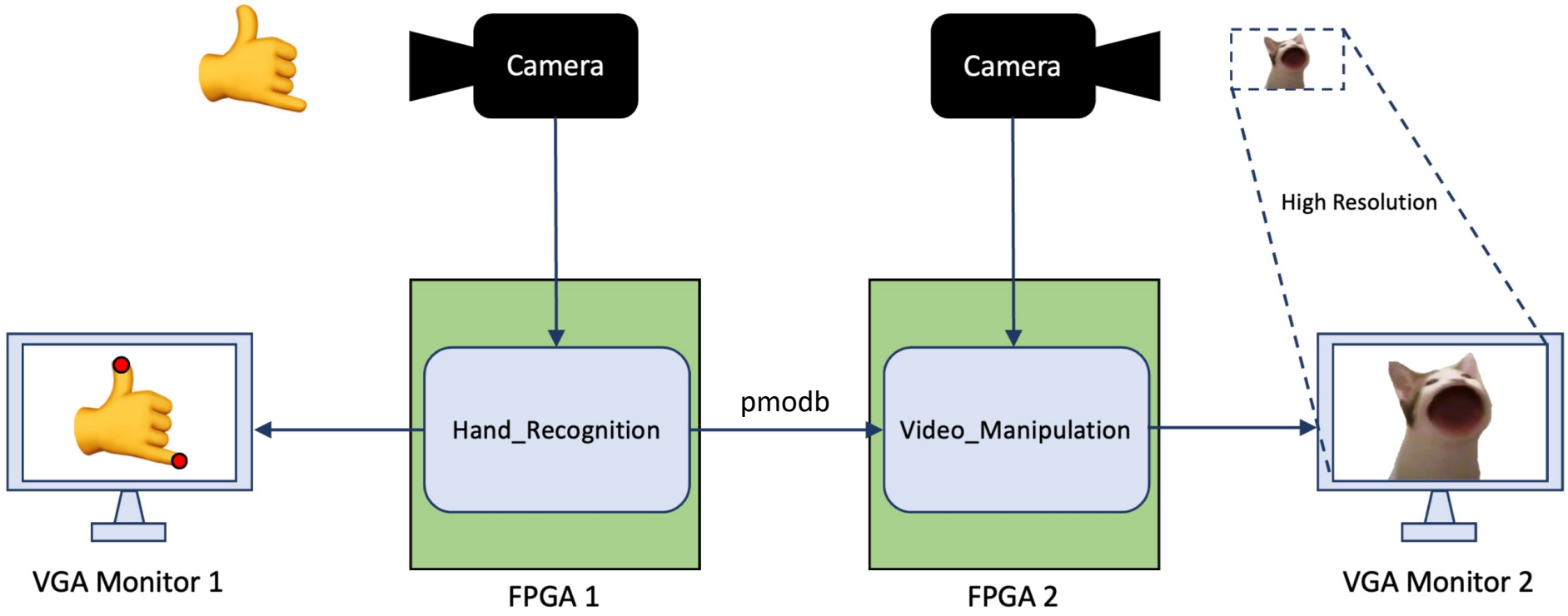
FPGesture Controlled Video Enhancement

6.111 Final Project

Soojung Bae, Jorge Tomaylla Eme



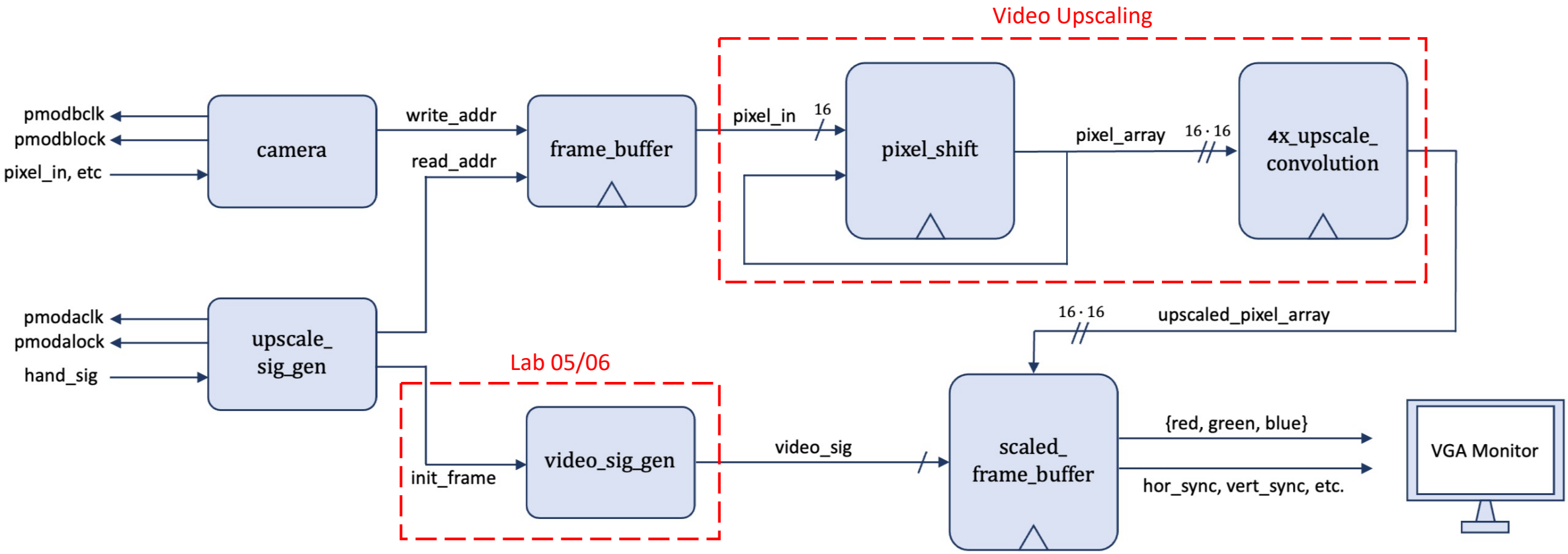
Project Design (from 5,000 ft up)



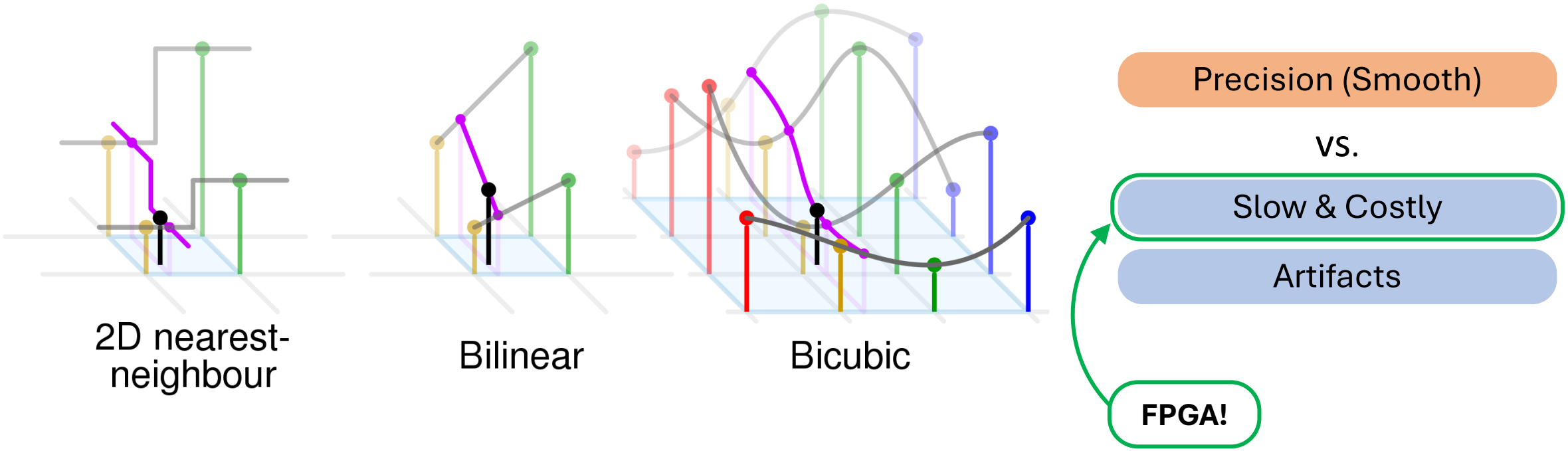
Q) Why use two cameras, two boards?

Video Manipulation

Block Diagram: Video Manipulation

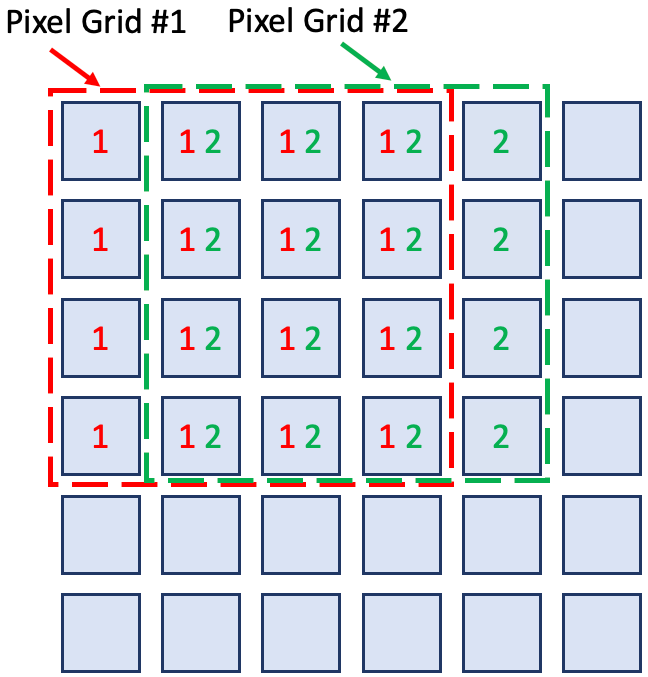


Upscaling Algorithms



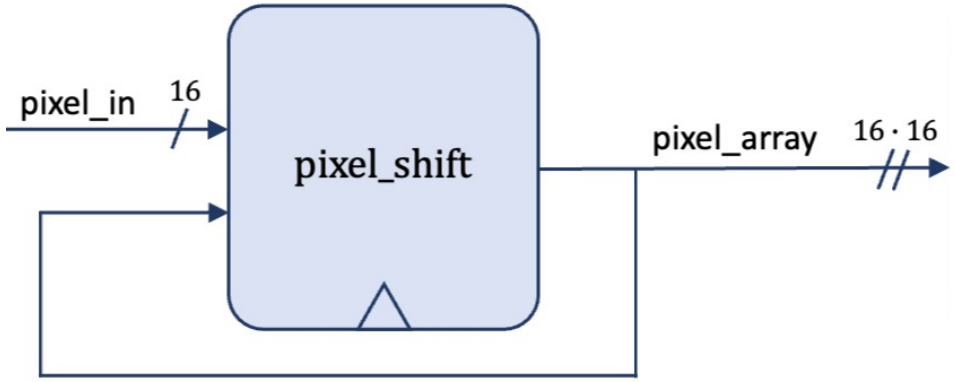
Reference) Comparison of nearest-neighbour, bilinear and bicubic interpolation methods by CMG Lee [1]

4x Upscaling: Pixel Shift



- Upscaling is convolution of 4-by-4 *continuous* image patches

→ *Pixel-level* pipelining maximizes throughput



4x Convolution Module (Bicubic Interpolation)

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j \qquad (\, a_{ij} = f(p_{00}, p_{01}, \dots, p_{44}))$$

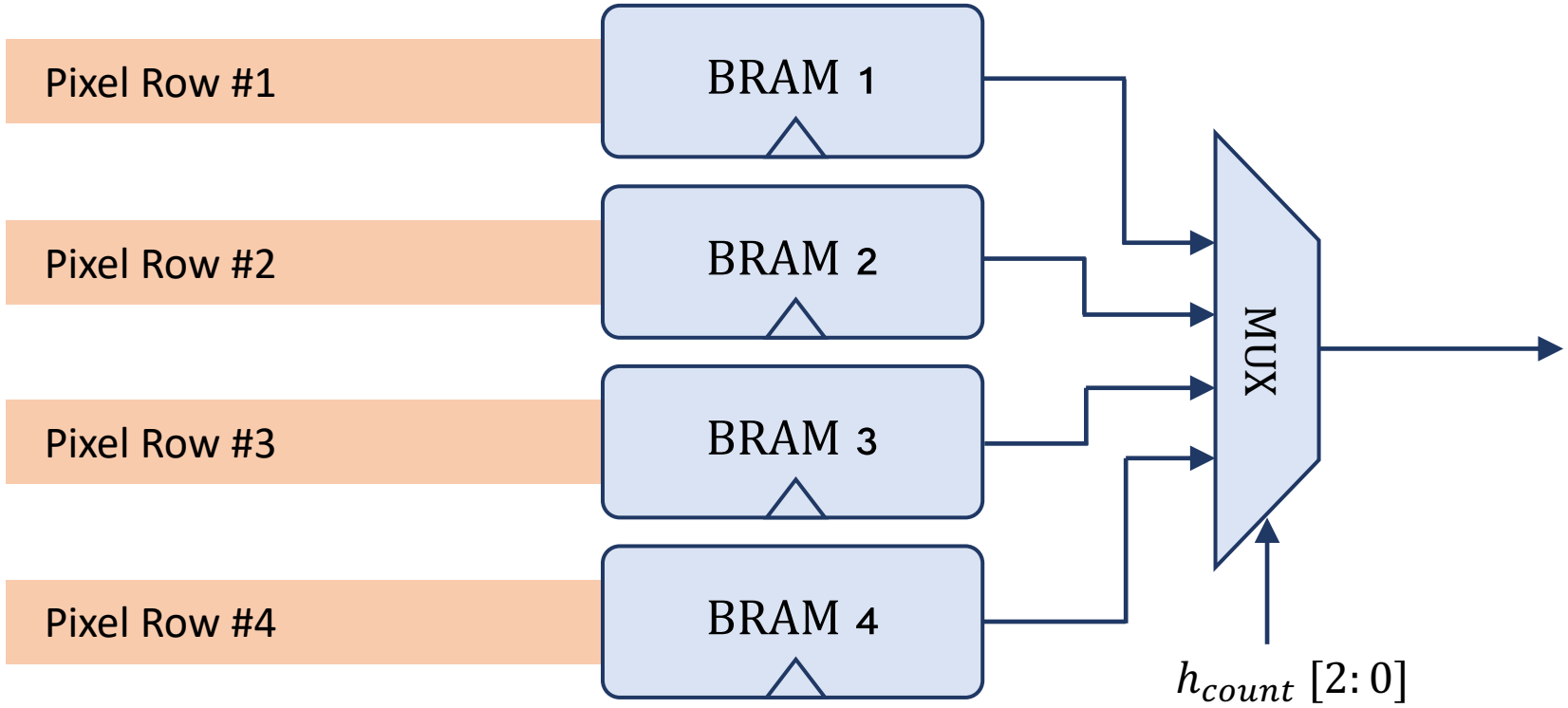
For each upscaled pixel (16 in total),

We have $(16\ multiply) + (16\ add) \sim 32\ operations$

$\therefore$ Theoretically $\sim 512\ Ops$ (*highly efficient* owning to pipelining)

Frame Buffer (for scaled pixels)

$$\texttt{(number\_of\_rows)} \times \texttt{(upscaled\_frame\_width)} \times \texttt{(bits\_per\_pixel)}$$
$$= 4 \times (4 \times 320) \times 16\,\texttt{bit} = 81.92\,\texttt{kBits}.$$



Python Demo

Hard-coded filter vs. CV2 Python package methods

```
X = {{f00, f01, f02, f03}, {f10, f11, f12, f13}, {f20, f21, f22, f2
   {f30, f31, f32, f33}};



CX = {{f11, f12}, {f21, f22}};
FX = {{f12 - f10, f13 - f11}, {f22 - f20, f23 - f21}}/2;
FY = {{f21 - f01, f22 - f02}, {f31 - f11, f32 - f12}}/2;
FXY =
  {{f22 - f02 - f20 + f00, f23 - f03 - f21 + f01},
    {f32 - f12 - f30 + f10, f33 - f13 - f31 + f11}}/4;

F = Join[Join[CX, FX, 2], Join[FY, FXY, 2]];

Convert = {{1, 0, 0, 0}, {0, 0, 1, 0}, {-3 , 3, -2, -1}, {2, -2, 1,
A = Convert.F.Transpose[Convert] ;
(A) // MatrixForm
```

Mathematica to acquire filter coefficients

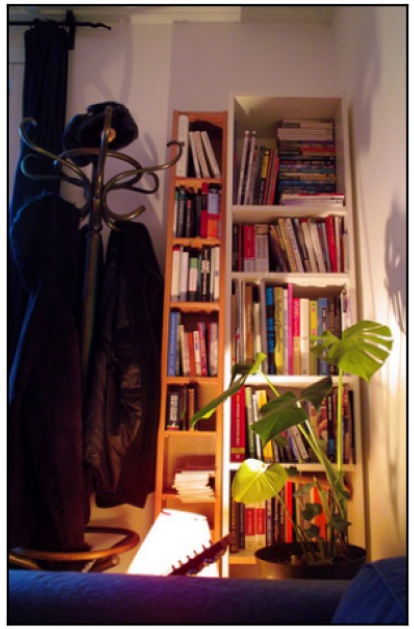
```
35  x30 = np.flip(x10, 0)
36
37  x01 = np.transpose(x10)
38  x11 = np.array(
39        [[81, -999, -261, 27],
40         [-999, 12321, 3219, -333],
41         [-261, 3219, 841, -87],
42         [27, -333, -87, 9]])/2**14
43  x21 = np.array(
44        [[9, -111, -29, 3],
45         [-81, 999, 261, -27],
46         [-81, 999, 261, -27],
47         [9, -111, -29, 3]])/2**11
48  x31 = np.flip(x11, 0)
```

Python follows the exact FPGA algorithm

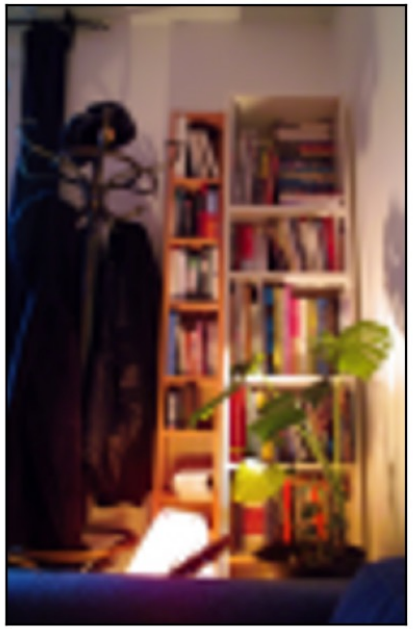
Python Demo

Hard-coded filter vs. CV2 Python package methods

GT



Bicubic Spline

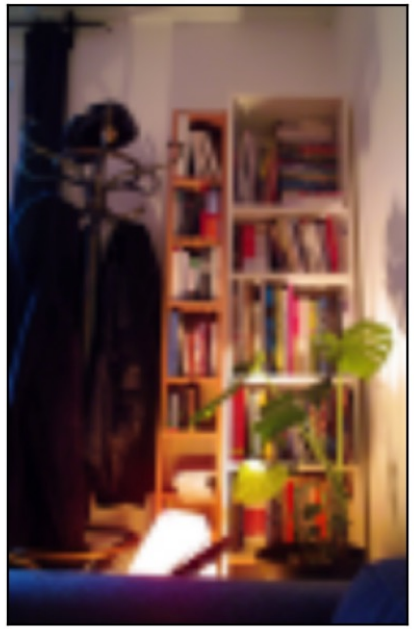


(23.1 dB)
(Ours)

Bicubic Conv.

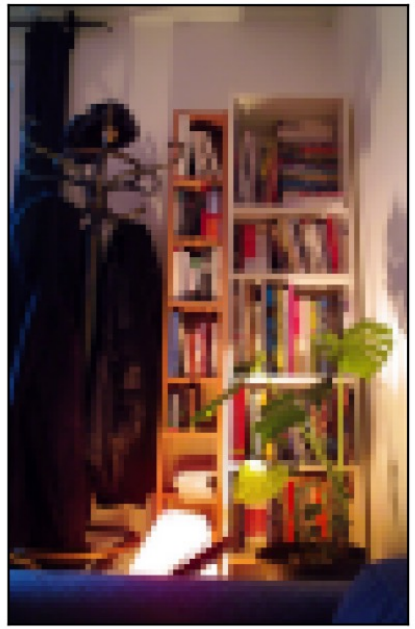
(23.1 dB)

Bilinear



(22.6 dB)

Nearest Neighbor



(22.1 dB)

Design Evaluation: Video Manipulation

Speed

Latency

Throughput

- Limited by HDMI (60 fps)
- Compare w/ Software Implementations

Performance

Accuracy

- Bit Arithmetic (errors)
- Filter's performance

Resource Utilization

Memory

Arithmetic Units

- How effective is our pipelining strategy?

Project Overview – Video Manipulation

(Commitment) With pixel-level pipelining, *Highly optimized* bicubic upscaling algorithm of video stream implemented purely on FPGA

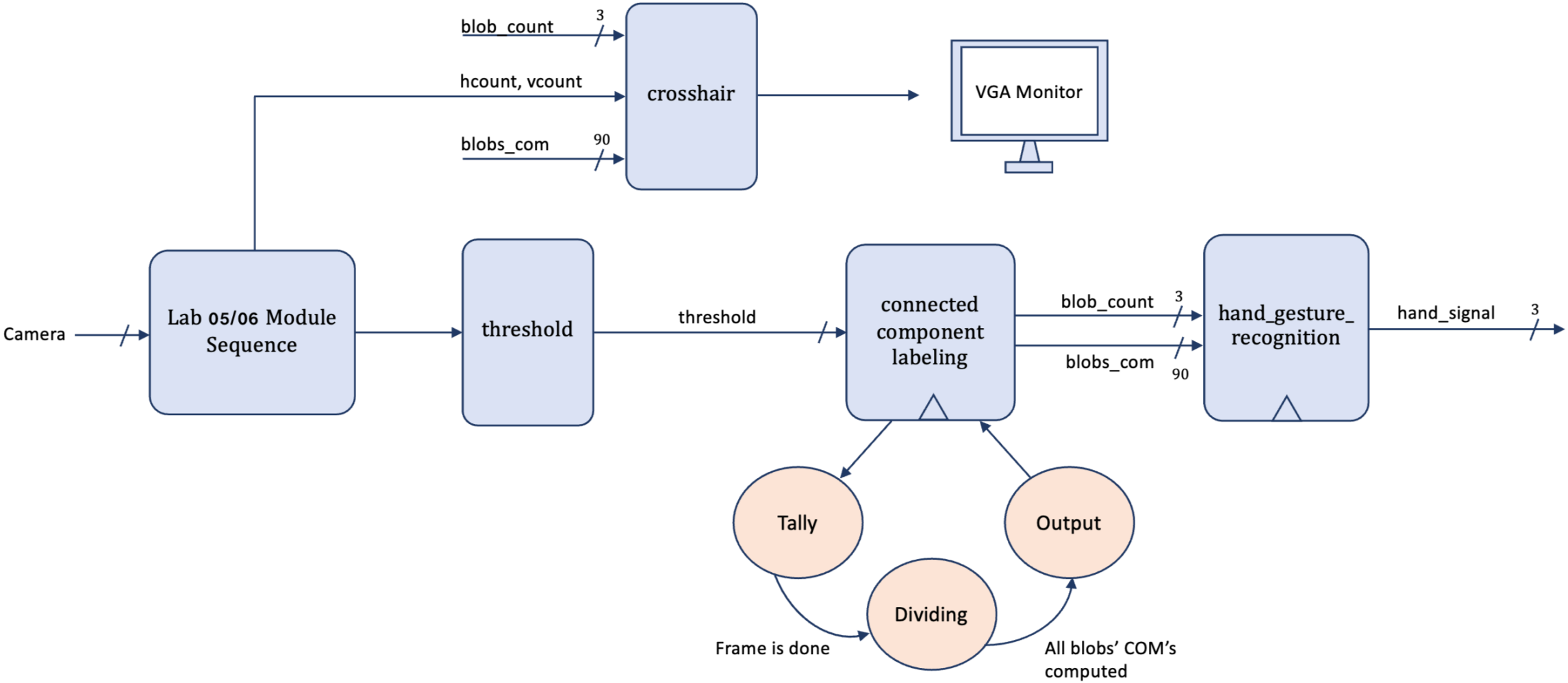
(Goal) 4x video enhancement pipeline, integrated with hand recognition

(Stretch) Add an additional convolution layer—*arbitrary* image filters

(e.g. Edge Detection, Sharp, Blur, …)

Blob Detection

Blob Detection Module Diagram



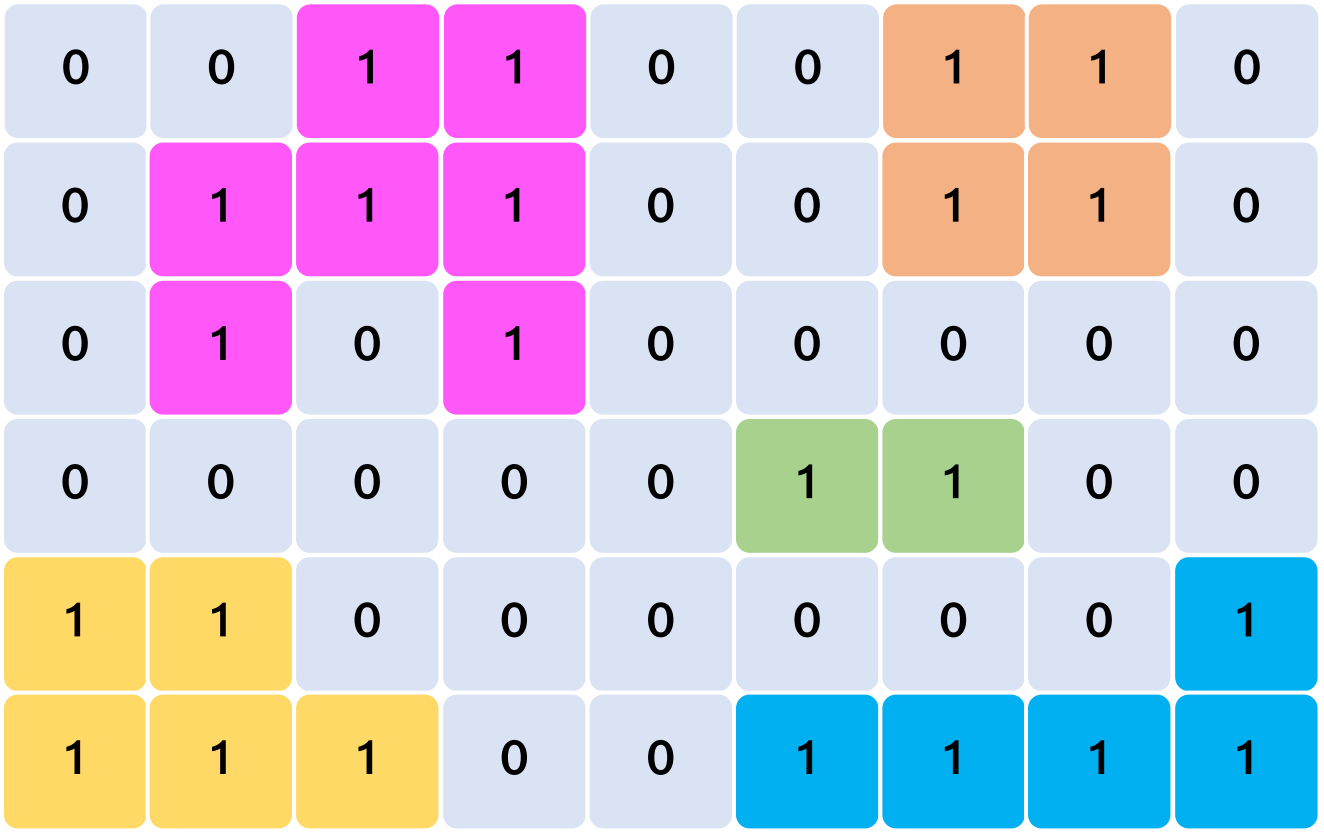
Threshold Input

0	0	1	1	0	0	1	1	0
0	1	1	1	0	0	1	1	0
0	1	0	1	0	0	0	0	0
0	0	0	0	0	1	1	0	0
1	1	0	0	0	0	0	0	1
1	1	1	0	0	1	1	1	1

Threshold:

Pixel threshold is done one-by-one by color values

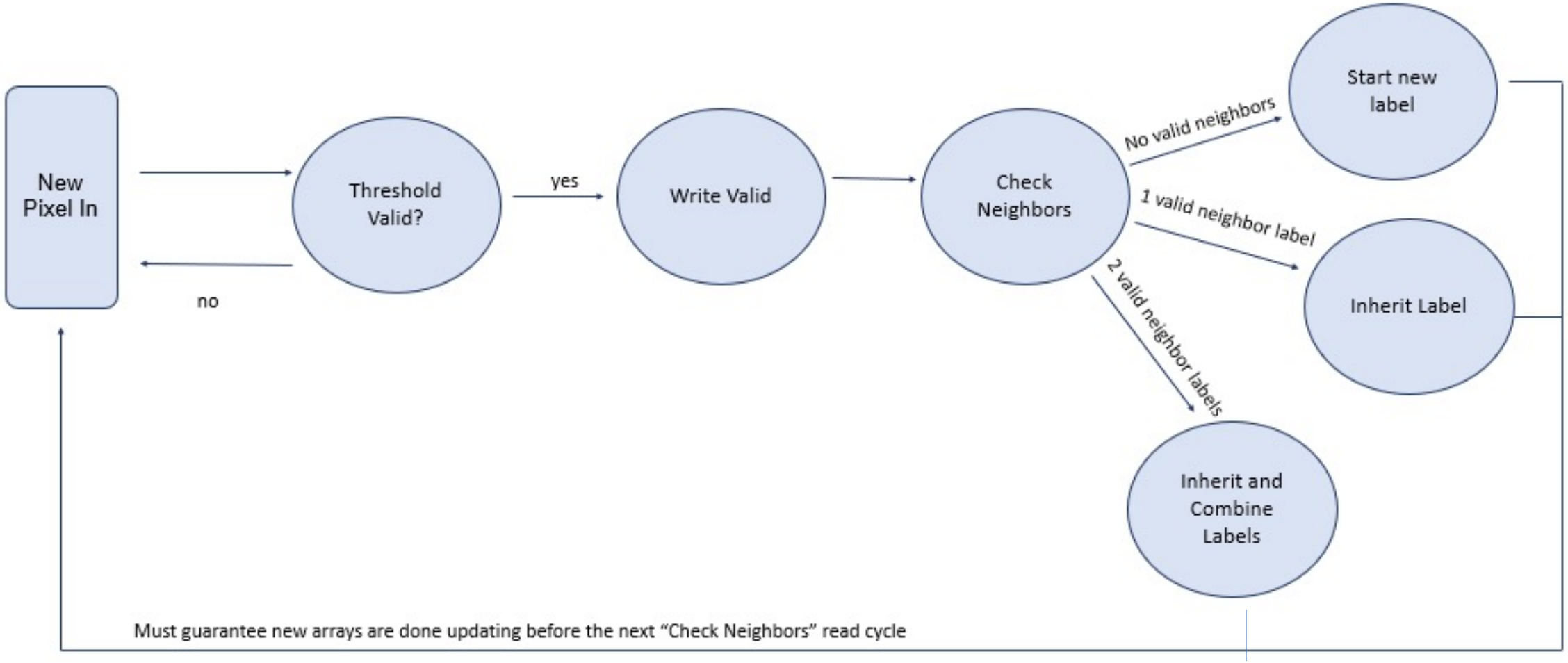
Blob Detection Behavior



Blob:

Continuous group of valid pixels or “strongly connected component”

Tally Flow Diagram



Memory Usage

Height = 38,400

Width = 18 bits

Data = {x [7:0], y [8:0], valid bit}

Address	Valid 1
0	18’b0
1	{8’b1, 9’b0, 1’b0}
2	{8’b10, 9’b0, 1’b0}
…	…
i	{x, y, valid_bit}
…	…
38,399	18’b0

Address	Valid 2
38,400	18’b0
38,401	{8’b10100001, 9’b0, 1’b0}
38,402	{8’b10, 9’b0, 1’b0}
…	…
i	{x, y, valid_bit}
…	…
76,799	18’b0

Mapping each valid pixel to its label for easy access of “read” pixel label

Address	Label 1
0	
1	1
2	1
…	…
i	0 < i < 11
…	…
38,399	2

Address	Label 2
38,400	4
38,401	4
38,402	4
…	…
i	0 < i < 11
…	…
76,799	

- 10 BRAM for 10 blobs (will be finetuned during integration) of height 38,400 because we do not expect blobs of 50% screen size
- Array will keep track of which blob labels are linked (thus avoiding rewrites)

Address	Blob 1
0	1
…	…
38,399	

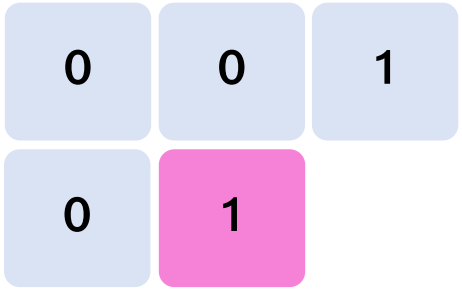
Address	Blob 2
0	54
…	…
38,399	

…

Address	Blob 10
0	
…	…
38,399	

Sample Walk

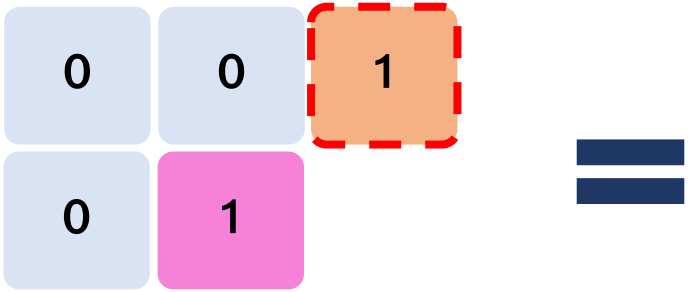
New pixel → Valid? → Yes!



Write Valid Pixel

Address	Valid 1
…	…
4	{8'b1, 9'b1, 1'b1}
…	…

Check Valid Neighbors



Read from Valid BRAM

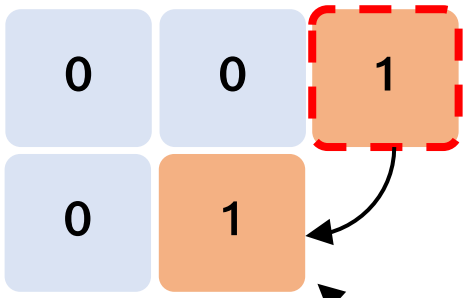
Address	Valid 1
…	…
2	{8'b10, 9'b0, 1'b1}
…	…

Valid?

Read Neighbor Label

Address	LABEL 1
…	…
2	1
…	…

4 cycles, 2 reads



Inherit label

Address	Label 1
…	…
4	1
…	…

Add to Blob 1 BRAM

Address	BLOB 1
0	2’b10
1	3’b100
…	…

New pixel inherited the same *label* (color)

Optimizations

- **Locking** : Reserve a register for the most recently added pixel in a blob, keep track of its y location. If we exceed y’ > y + 1, then it is impossible to add a new pixel to this blob or to combine with another blob, so “lock” it and start the x and y count and division for this blob.

- **Fine tuning**: With the assumption that we will only have 5 blobs of pink color in the screen, reduce the number of BRAMs or address entries for faster pipelining

Design Evaluation: Video Manipulation

Robustness

Algorithmic efficiency

- Assumption vs. Real-life Data
- How will it response with worst-case inputs
- Pipelining for optimization

Performance

Accuracy

- Are all connected pixels properly match to connected labels?
- Is there a consistent blob detection output?

Resource Utilization

Memory

- Can we reduce the BRAMS/resources we need?
- Can we accurately account for read and write cycle delays?

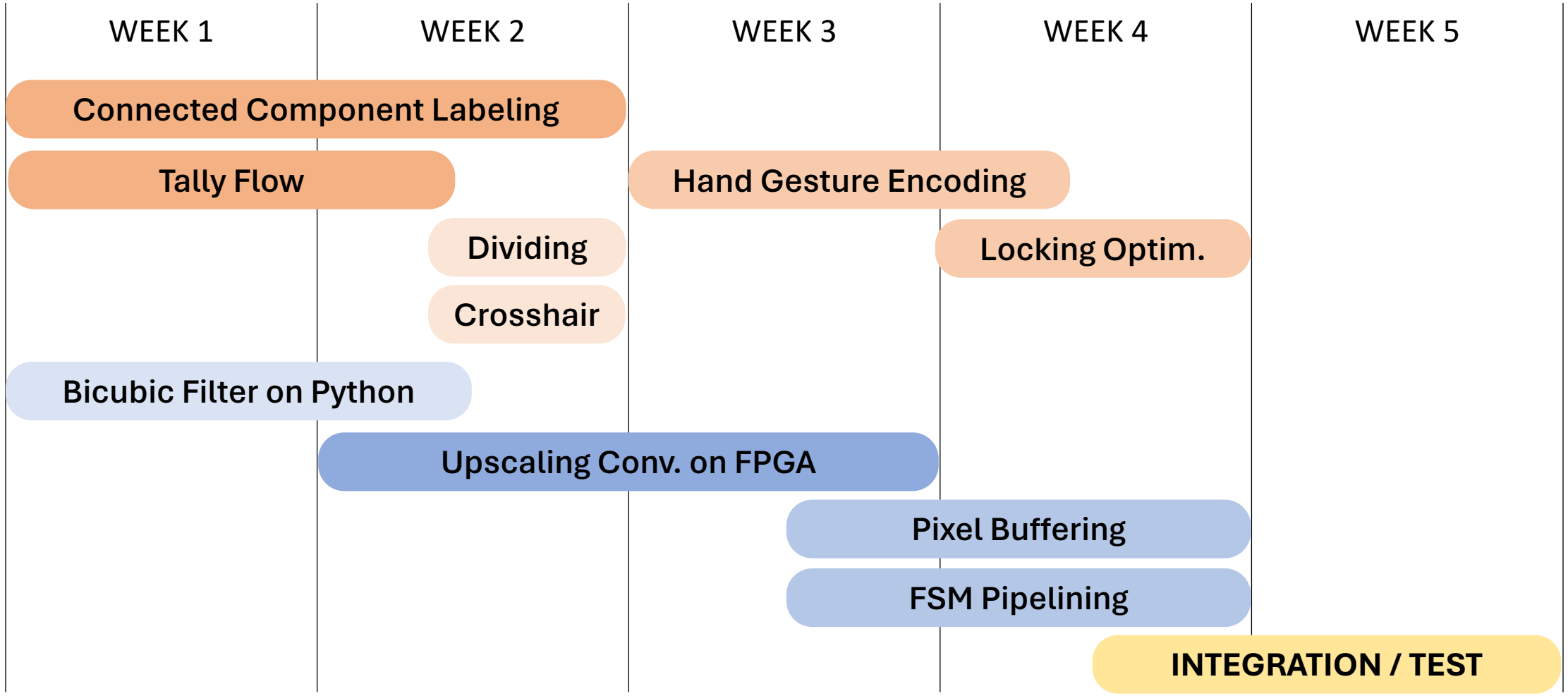
Project Overview – Video Manipulation

(Commitment) Blob detection algorithm correctly labeling the largest 5 color blobs on the screen and displaying centroids with crosshair pattern, allowing some delay

(Goal) Encoding of hand commands and serialization to FPGA2 for video manipulation

(Stretch) Adding optimizations for smaller latency

Timeline


WEEK 1
WEEK 2
WEEK 3
WEEK 4
WEEK 5
Connected Component Labeling
Tally Flow
Hand Gesture Encoding
Dividing
Locking Optim.
Crosshair
Bicubic Filter on Python
Upscaling Conv. on FPGA
Pixel Buffering
FSM Pipelining
INTEGRATION / TEST





Q & A