

Digital Circuit Lab Final Project Report–It Counts!

Team 11 B09901011 杜昀蓉 B09901027 賀崇恩 B09901029 楊珩

1. 動機

每次去買電子零件的時候店員都要花很多時間點材料的數量，因此想實作出可以快速點好大量物品的相機。

2. 功能

- a. 燒錄到板子後會auto start，當作一般相機串流，將目前影像顯示在螢幕上，而七段顯示器會顯示目前是第幾個frame(HEX 4&5)
- b. 按下Capture按鈕 (key[2])後擷取影像並進行處理，轉成灰階後數數並顯示在DE2-115的七段顯示器上(HEX 6&7)，同時螢幕會顯示Capture的畫面，按下key[3]會重新進行串流，count歸零，等待下一次數數。
- c. 切換Switch可以分別數兩種不同大小的圖案並個別顯示在HEX (HEX 0&1/HEX 2&3; SW12可以切換模式)

3. 運作方式

- a. 燒錄Code到板子之後，相機會開始串流，串流的方式為FPGA從相機得到資料後，會寫到SDRAM的Write FIFO當中，SDRAM一次存取一個frame的資料 (800*600 pixels)，VGA再從SDRAM的Read FIFO讀取資料並輸出到螢幕。
- b. 按下KEY[2]，相機會停止串流，SDRAM中的資料不會再更動，VGA讀取數字後進行YCbCr的轉換，並判斷Y與Threshold的大小關係將圖片轉成黑白，並將Y傳給Blob_final.sv
- c. Blob_final.sv一個一個的接收經過灰階處理的pixels串流並處理，並非random access，最後輸出數量
- d. 按下KEY[3]會恢復串流模式
- e. 按下KEY[0]可以reset

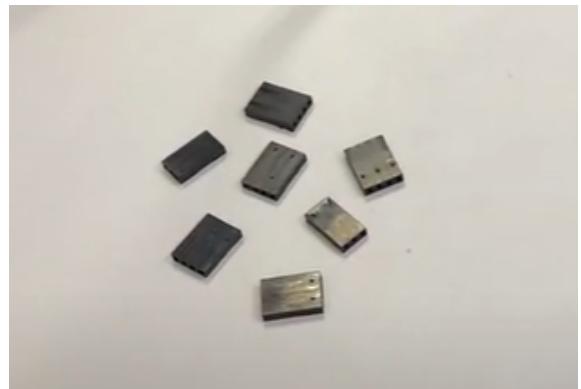


Fig 1. 拍攝電子零件 (3,4pin排母)

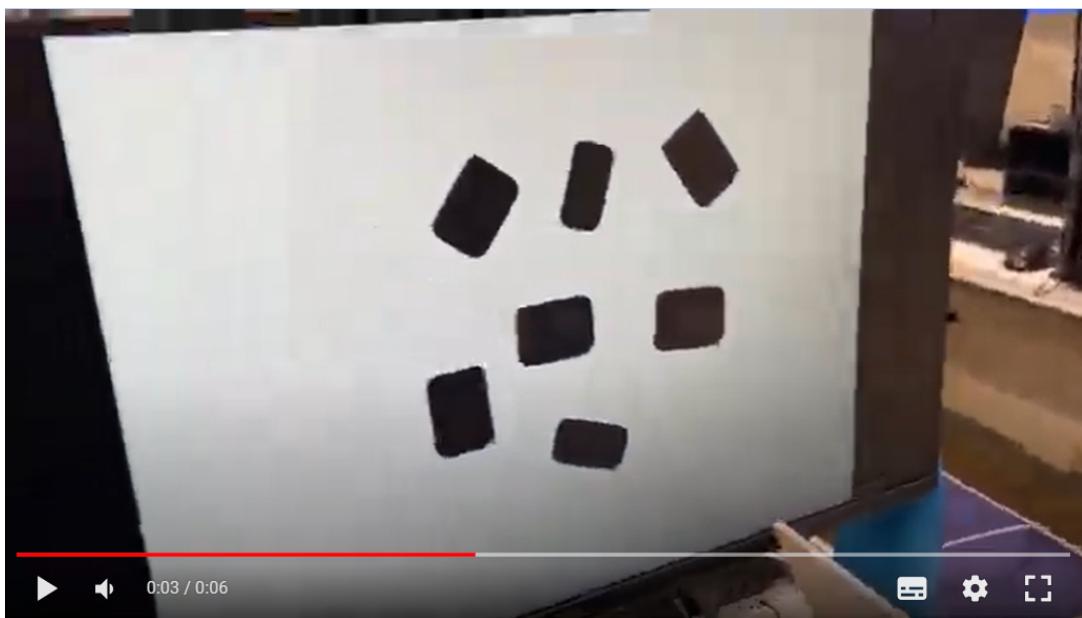


Fig. 2. 螢幕顯示相機拍攝畫面，經過FPGA的灰階和**binarize**處理

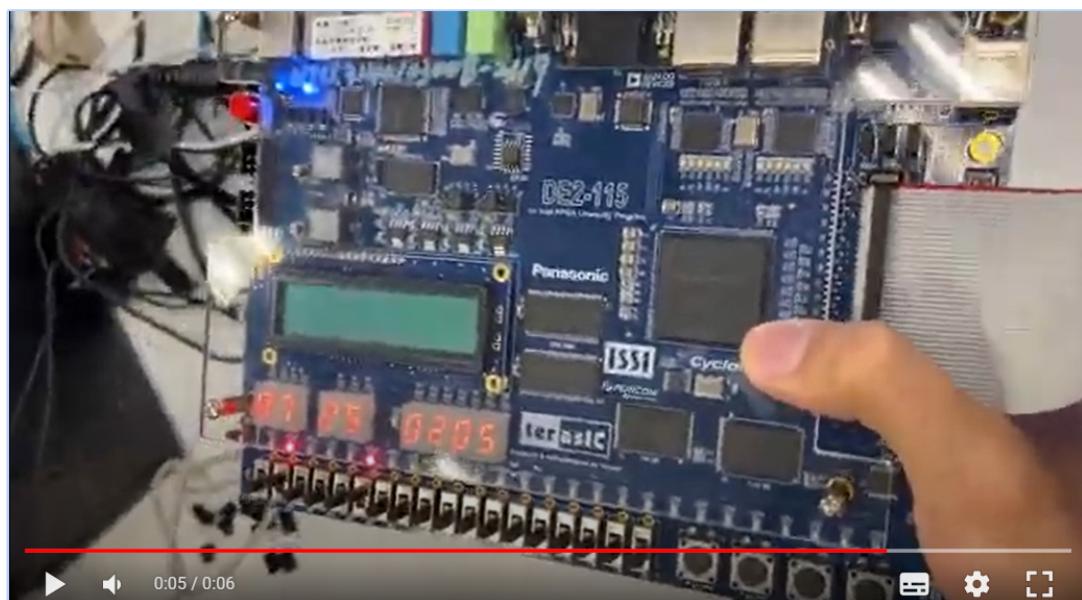


Fig. 3. 按下Capture後，FPGA的HEX顯示Total count和Small count /Big count(Total count: HEX7/HEX6, Small count: HEX3/HEX2, Big count: HEX1/HEX0, 圖中數字為07 02 05)

4. 使用模組

- a. Altera Cyclone IV FPGA DE2-115 * 1
- b. VGA傳輸線 * 1
- c. 螢幕 * 1
- d. 5 Mega Pixel Digital Camera Package * 1
- e. Jtag UART傳輸線 *1

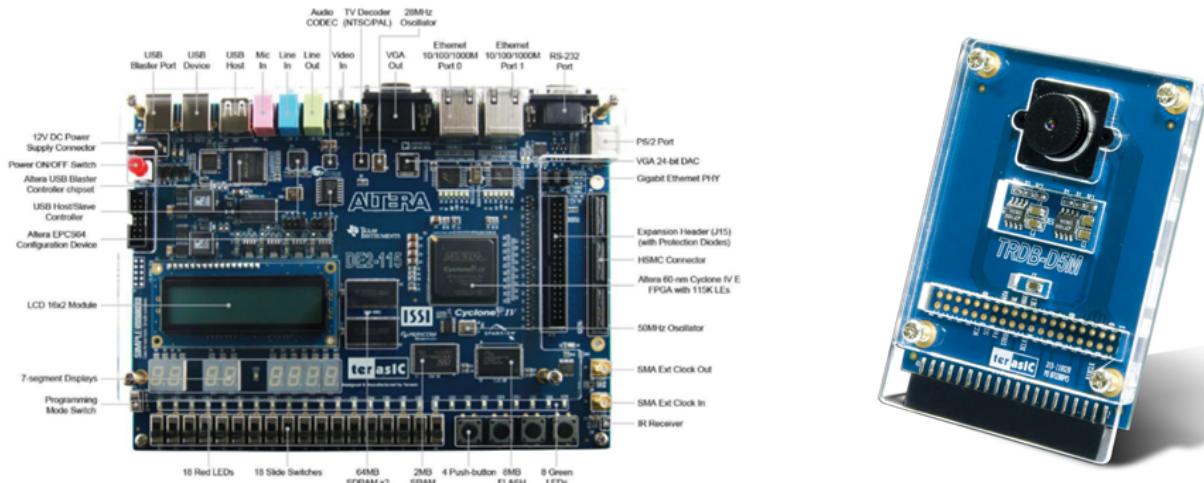


Fig. 4. DE2-115 FPGA

Fig. 5. 5 Mega Pixel Digital Camera Package (TRDB-D5M)

5. 演算法介紹

a. RGB to YCbCr

YCbCr不同於RGB的色彩空間，三個Channel分別表示的是Y (luma, 流明) 以及Cb Cr兩個blue-difference, red-difference chroma, 常用於連續的影像處理。在我們的系統中，要取得YCbCr的資料，首先要從SDRAM得到每一個pixel的RGB值並進行weighted sum，算出深淺 $o_color=0.297*R + 0.586*G + 0.145*B$ 。小數點乘法的做法是算出小數點的二進位後shift成整數，乘完再shift回去，因此會造成一些quantize誤差。算出深淺後再透過判斷是否大於Threshold而決定把該pixel調成黑色(10'h000)或白色(10'h3ff)，也就是binarize。這一步在我們整個辨識過程中的目的主要是去除噪音、去除計算複雜程度，以及得到零件更清晰的輪廓。

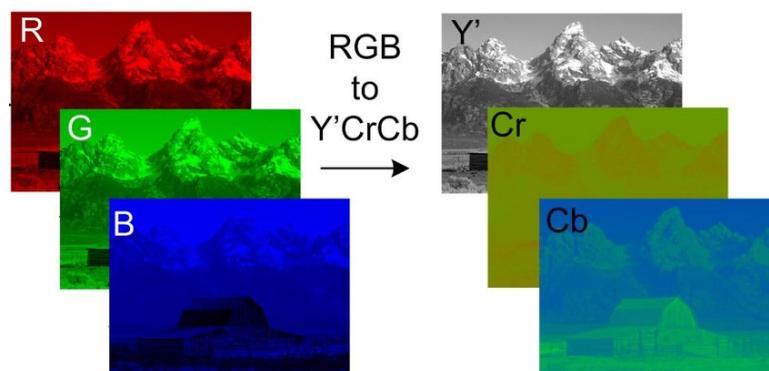


Fig. 6. RGB to YCbCr transform

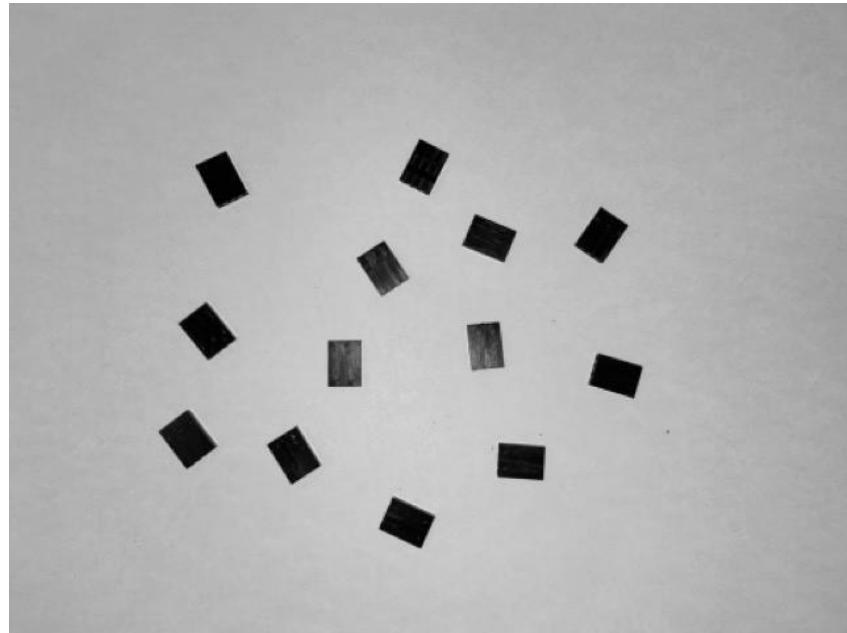


Fig. 7. 拍攝零件照片

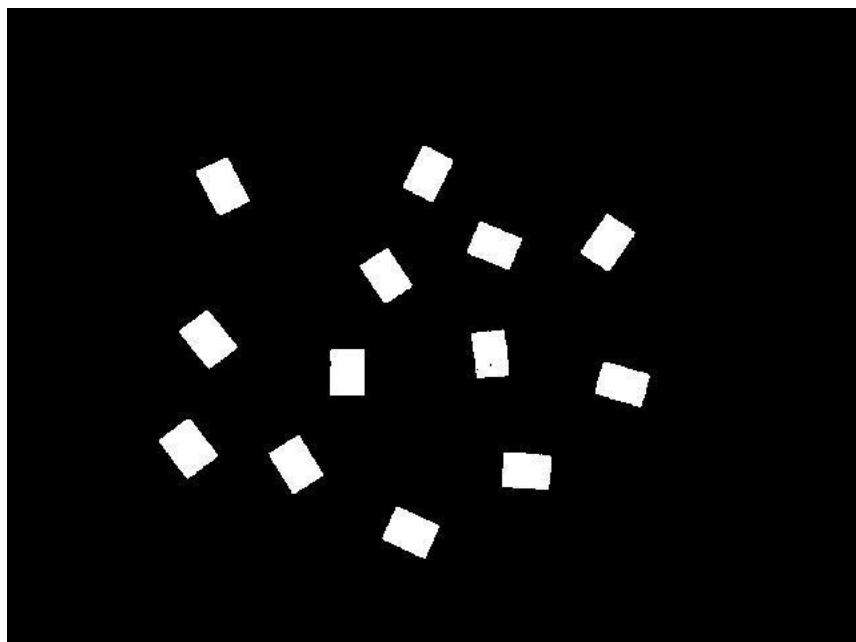


Fig. 8. 在Python通過YCbCr + Binarize + 補色處理後圖片

b. BLOB

Blob detection是做物件辨識的方法之一，方法是偵測圖像中與周圍相比有不同屬性的區域（顏色或亮度）。一個Blob的所有點應該有類似的屬性。我們的Blob detection是基於connected-component analysis演算法（CCA）。軟體的connected-component analysis是基於graph theory的資料結構、需要two pass圖片或需要random access像素內容，但這些在數位系統都比較難做到。

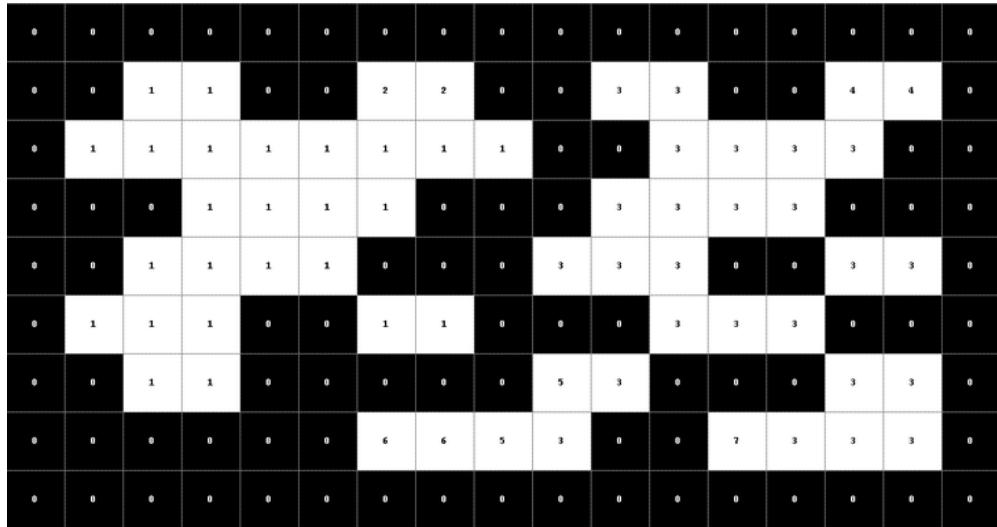


Fig. 9. CCA標示label的過程

因此我們希望可以採用pixel streaming based的演算法，加上heuristic來歸納出所有需要處理的情形，以計算出新pixel的正確的label，如下：

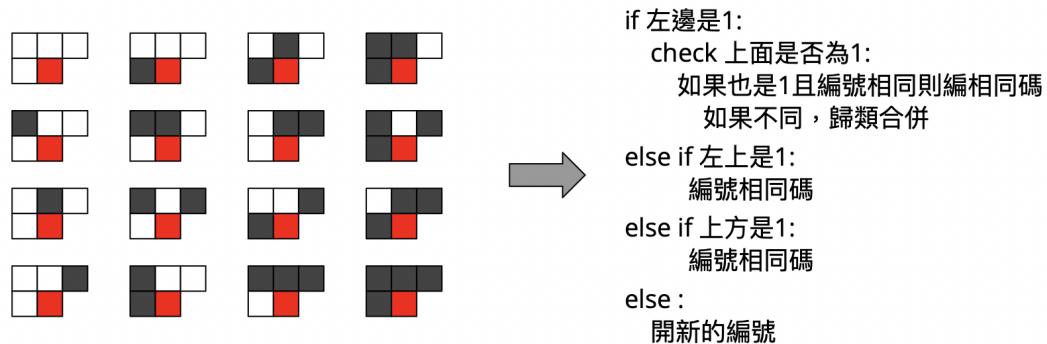


Fig. 10. CCA algorithm with heuristic solutions

實際處理時，我們會有三個陣列：

- 1. buffer:** 儲存一整排，加上一格的像素所對應的label為何 (黑色的label為0)。這是因為Blob演算法會需要圖片上方以及左上方pixel的資訊。每個cycle buffer的尾會pop掉，所有element shift一格，然後buffer的頭放入當前收到的pixel計算出來的label。
- 2. pixels:** 儲存每個label的區域擁有幾個像素。這個陣列大小是可以自定義的 (例如：定義畫面中最多有100個label的區域)。每次有一個新的pixel加入，它對應的類別的pixel array內容就會+1，除了黑色的pixel，label為0的pixel count總是為0。

3. **table**: 儲存該編號合併的指向(大小即label總數)。在上面CCA演算法中，如果左邊和上面的方塊都是黑色而且label不同就表示要merge，我們用table的內容紀錄有哪些label的區域merge過。例如：讀取pixel stream的時候發現label 為3的區域要跟label為2的區域合併，那我們以取最小label為原則(先產生的label不要回溯為原則)，所以table的第3個register就會記錄2表示merge到label 2。爾後label 3的pixel array可以繼續記錄label 3的個數，但最後紀錄的pixel數都會加到label 2裡面。

buffer : 存一排圖片像素的編號

1	1	2	2	3	3	4	4	5	
1 1									

pixel : 存該編號有幾個像素

2	2	2	2	1	0	0
4 0						

table : 存該編號合併的指向

0	1	2	3	4	5	0
0						

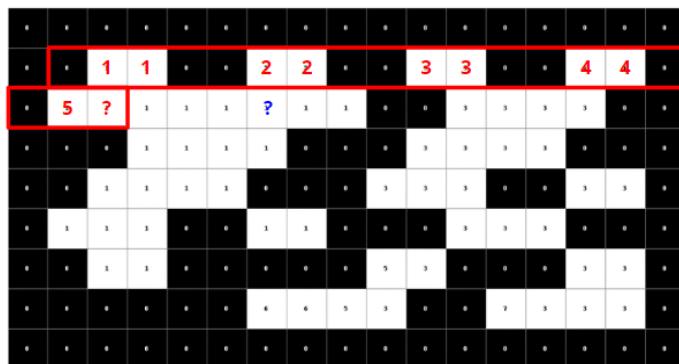


Fig. 11. CCA演算法在pixel streaming時所需的資料結構

在pixel 的stream完成後，table和pixel的內容也已經完成，buffer的內容可以捨棄。接下來進行merge，也就是處理table內紀錄merge的事件，我們根據table裡記錄指向較小的label，將該label的pixel加到指向的label的pixel數，並把自己的pixel count歸0，以免接下來重複計算。接下來就是進入count，我們利用在pixel array剩餘的元素最大者除8 (shift 三位)作為threshold，count即為紀錄pixel 的array中剩餘比該threshold高的個數。

因為table和pixel array大小是有限的，也就是說label總數是有限的，我們並不希望拍攝畫面的黑色區域過於零碎(即會產生很多label)，否則數位系統的行為將會是undefined。這可以從拍攝的方式或相機參數調整來解決，也就是相機的曝光時間，以及拍攝的背景反光程度，或是前面轉YCbCr的threshold都可以解決這個問題。在我們的實驗裡，透過這些調整，相機拍攝的畫面，在YCbCr + binarize後可以做到辨識完整的邊緣。

c. Classification

我們希望可以做到拍攝電子零件時簡單分類的功能，我們採用的是用物體的像素數量來做分類。前面已經提到，我們的CCA演算法在看完pixel streaming之後可以知道每個label區域的pixel數。我們的做法是從一個max的pixel數往下分成N個區間，每個區間是400個pixel，每個label歸入一個區間，並記錄相鄰區間的label個數差值，最大差值之處即為兩類物品的分界。

這個方法也可以套用到多種物體，但物體的area要有區別、物體的種類也要事先決定。

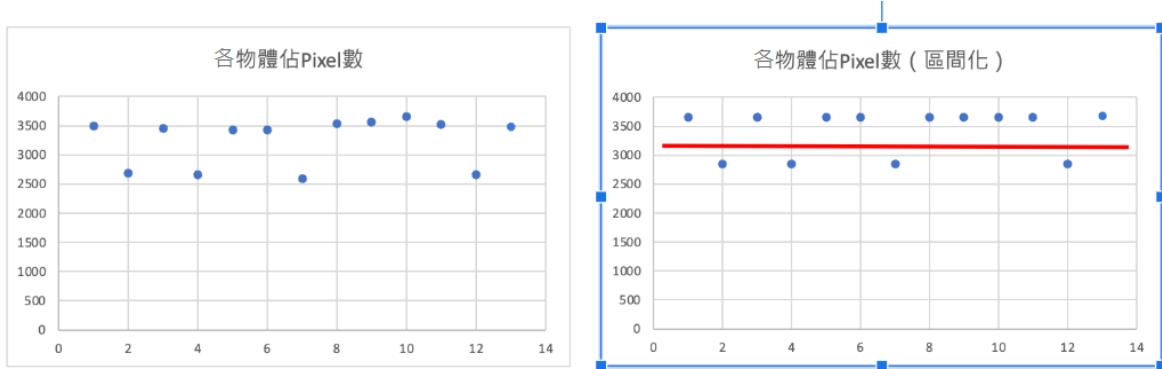


Fig. 12. 我們拍攝的sample圖片，經過ncverilog simulation後的13個類別的pixel count(橫軸為label，縱軸為pixel count)，可以發現在pixels = 3200的時候恰可以把兩個類別分開。因此大的圖形有9個，小的圖形有4個。

6. 層級架構

- A. **DE2_115_CAMERA.v**: Top Module, 包含Reset_Delay module (Reset_Delay.v)、CCD_CAPTURE module (CCD_Capture.v)、RAW2RGB (RAW2RGB.v)、SEG7LUT8 module (SEG7_LUT_8.v)、sdram_pll module (sdram_pll.v)、Sdram_Control module (Sdram_Control.v)、I2C_CCD_Config module (I2C_CCD_Config.v)、Blob_final module (Blob_final.sv)、VGA_Controller module (VGA_Controller.v)
- a. **Reset_Delay.v**: 輸入是原本的reset訊號，輸出是有不同delay的reset訊號分別給不同module。
 - b. **CCD_Capture.v**: 從Camera讀取資料，切換stream與capture模式，傳來的資料是Bayer Pattern，即一個pixel是由一個R，兩個G和一個B組成，如下圖所示。
-
- The diagram shows a camera sensor layout and its corresponding pixel data. On the left, a square 'Active Image' area is labeled '2592x1944 Pixels'. This area is surrounded by a yellow 'Active boundary (10)' and a black 'Dark (134)' region. The overall sensor size is indicated as '(0,0)' at the top right. On the right, a grid of pixels is shown with a 'column readout direction' arrow pointing right and a 'row readout direction' arrow pointing down. The grid contains alternating colored pixels: Green (G), Red (R), Green (G), Blue (B), Green (G), Red (R), Green (G), Green (G). A specific pixel at position (10, 50) is labeled 'First Clear Pixel (10,50)'. Some pixels are also labeled as 'black pixels'.
- c. **RAW2RGB.v**: 要把Bayer Pattern轉成VGA可以讀的RGB資料。
- d. **SEG7LUT8.v**: 控制七段顯示器的module，會顯示的資料有frame數量、物品總數、兩類別分別的物品個數，是以16進位顯示。
- e. **sdram_pll.v**: 生成不同Clock, 如SDRAM需要的100M, VGA需要的25k(frame size 640*480)或40k(frame size 800*600)。
- f. **Sdram_Control.v**: 有兩個Read FIFO和兩個Write FIFO, 分別讀取來自Camera資料的輸入與輸出給VGA, 底下有FIFO的module。
- g. **I2C_CCD_Config.v**: 負責使用I2C傳exposure time與是否zoom等Camera相關設定給Camera。
- h. **Blob_final.sv**: 負責執行Blob演算法，讀入Pixel後輸出物品總數與分類數目。
- i. **VGA_Controller.v**: Request並接收來自SDRAM的資料，將資料進行YCbCr的演算法處理得到灰階後，藉由VGA顯示在螢幕上並傳給 Blob_final.sv。

7. Block Diagram

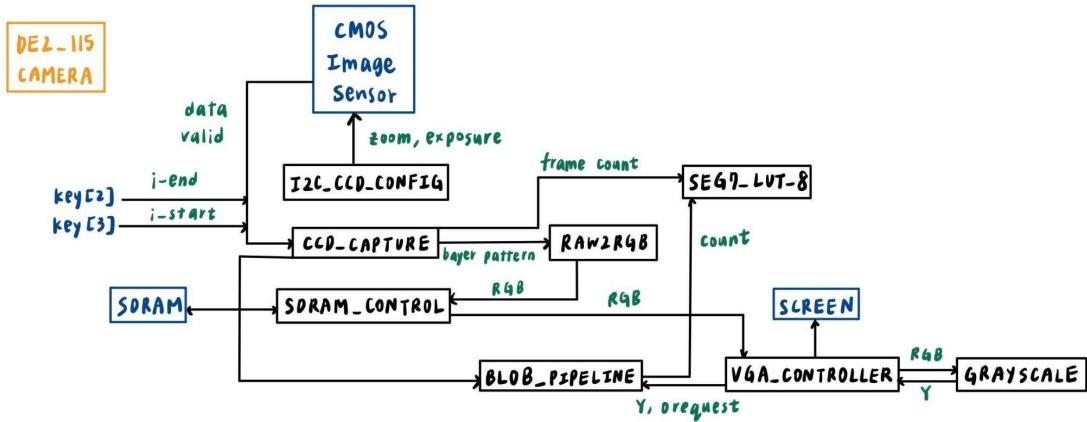


Fig. 14. Block diagram of the design

我們的design裡的block diagram如上圖所示。相機的部分，I2C_CCD_Config在開機的時候傳送給CMOS image sensor zoom in 和曝光等參數。CMOS image sensor的data會通過RAW2RGB將bayer pattern轉換成RGB data, 然後寫進Sdram Controller的Read FIFO。VGA_Controller則會根據VGA的sync週期向Sdram controller request data , 寫到VGA, 同時VGA_controller內部會由counter產生VGA需要的HSYNC和VSYNC訊號。

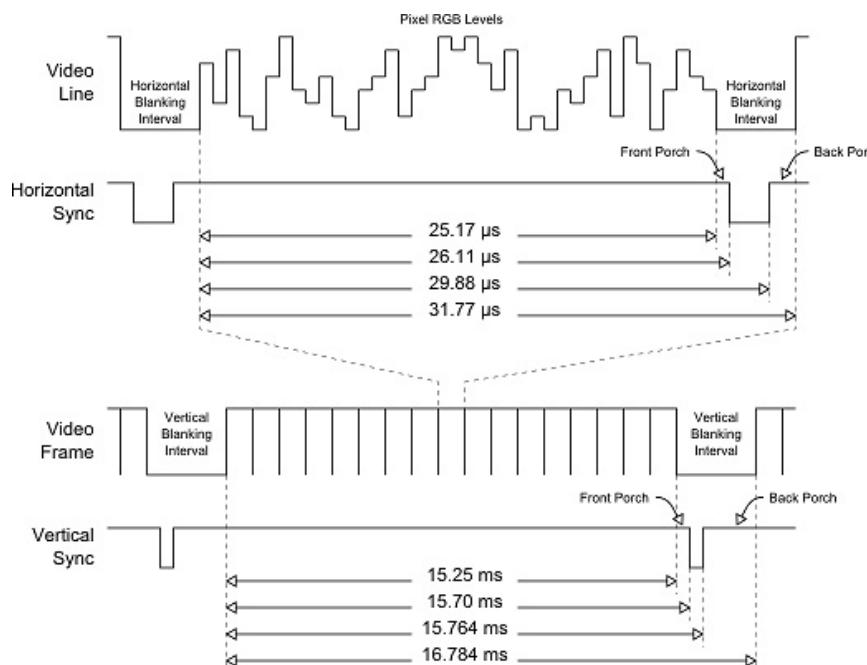


Fig. 15. VGA的sync週期, VGA所需的訊號包含Horizontal sync (表示要換行), Vertical sync (表示要換frame), 在horizontal sync 和vertical sync都activate時傳送RGB的訊號。

在CCD_Capture偵測到key的事件的時候，會發生以下變化：(1) CMOS image sensor的資料不寫進SDRAM write FIFO, (2) Blob計算的 module 會準備開始要資料。在偵測到key的時候，因為key的事件跟VGA不同步，如果從這時候開始Blob就向VGA (或SDRAM)要資料，圖片就是從錯誤的地方而不是左上角開始，就像整張圖被拼接了一樣。因此接收到key的訊號之後，Blob還要等VGA_Controller的vertical sync訊號才能開始計算。計算時由於還有horizontal sync，基本上VGA_Controller有request SDRAM的時候Blob module才吃進一個pixel做計算，否則就要hold住所有data。由此可知，Blob 的計算module使用的是VGA的clock。我們也試過在vertical sync之後讓Blob 連續request整個frame的pixel，結束後等待vertical sync再讓VGA去request，但我們這樣做後VGA顯示畫面總是錯的，因此才改為Blob隨VGA request data才做計算的模式。

8. Finite State Machine

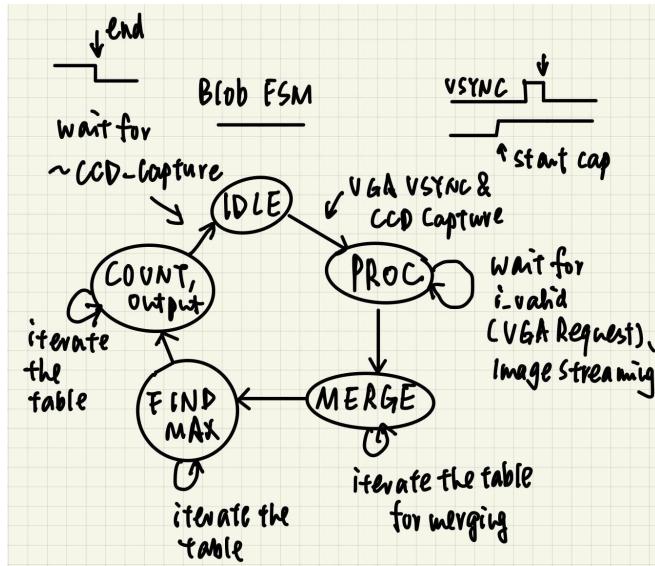


Fig. 16. Blob module FSM

我們的Blob計算單元的finite state machine如上圖所示。如前所述，Blob module要在CCD capture開始之後，等待VGA controller Vertical sync進入process state，在process state如果VGA controller有做request才進行計算，之後進行merge, findmax和count，最後停在output state，等待CCD capture結束（重新回到串流狀態）再回到idle state。

9. Fitter Summary

Compilation Report - DE2_115_CAMERA	
Flow Summary	
Flow Status	Successful - Tue Dec 27 13:09:09 2022
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115_CAMERA
Top-level Entity Name	DE2_115_CAMERA
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	20,667 / 114,480 (18 %)
Total combinational functions	20,242 / 114,480 (18 %)
Dedicated logic registers	4,964 / 114,480 (4 %)
Total registers	4964
Total pins	428 / 529 (81 %)
Total virtual pins	0
Total memory bits	64,658 / 3,981,312 (2 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	1 / 4 (25 %)

Fig. 17. Fitter summary

10. Timing Analyzer

Compilation Report - DE2_115_CAMERA			
Slow 1200mV 85C Model Setup: 'u6 altpll_component auto_generated pll1 clk[4]'			
	Slack	From Node	To Node
1	-3.599	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][6]
2	-3.547	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][8]
3	-3.547	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][13]
4	-3.533	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][10]
5	-3.476	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][6]
6	-3.451	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[11]</u>	Blob_final u10 pixels_r[37][6]
7	-3.442	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][6]
8	-3.424	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][8]
9	-3.424	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][13]
10	-3.410	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][10]
11	-3.399	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[11]</u>	Blob_final u10 pixels_r[37][8]
12	-3.399	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[11]</u>	Blob_final u10 pixels_r[37][13]
13	-3.390	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][8]
14	-3.390	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][13]
15	-3.385	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[11]</u>	Blob_final u10 pixels_r[37][10]
16	-3.376	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[12]</u>	Blob_final u10 pixels_r[37][10]
17	-3.318	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[60][9]
18	-3.308	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][6]
19	-3.306	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][5]
20	-3.306	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][12]
21	-3.303	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[44][7]
22	-3.301	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[11]</u>	Blob_final u10 pixels_r[37][6]
23	-3.292	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][6]
24	-3.268	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[14][8]
25	-3.256	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[14]</u>	Blob_final u10 pixels_r[37][8]
26	-3.256	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[14]</u>	Blob_final u10 pixels_r[37][13]
27	-3.251	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[65][7]
28	-3.249	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[11]</u>	Blob_final u10 pixels_r[37][8]
29	-3.249	Sdram_Control <u>:7 Sdram_RD_FIFO u_read1_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[11]</u>	Blob_final u10 pixels_r[37][13]
30	-3.242	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[14]</u>	Blob_final u10 pixels_r[37][10]
31	-3.240	Sdram_Control <u>:7 Sdram_RD_FIFO u_read2_fifo dc...uto_generated altsyncram_sj31:fifo_ram q_b[10]</u>	Blob_final u10 pixels_r[37][8]

Fig. 18. Timing Analyzer Report

Compilation Report - DE2_115_CAMERA								
	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[793][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.891	2.987
2	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[792][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.891	2.987
3	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[791][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.891	2.987
4	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[790][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.891	2.987
5	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[789][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.891	2.987
6	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[788][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.891	2.987
7	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[785][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.891	2.987
8	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[712][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.900	2.978
9	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[651][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
10	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[650][0]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
11	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[759][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.898	2.980
12	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[758][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.898	2.980
13	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[757][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.898	2.980
14	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[756][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.898	2.980
15	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[755][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.898	2.980
16	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[754][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.898	2.980
17	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[659][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
18	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[658][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
19	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[656][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
20	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[655][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
21	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[654][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
22	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[653][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
23	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[652][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
24	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[651][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
25	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[650][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
26	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[649][1]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
27	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[787][2]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.900	2.978
28	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[785][2]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.900	2.978
29	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[674][2]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.902	2.976
30	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[643][2]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.900	2.978
31	-3.429	Reset_Delayu2 oRST_2	Blob_finalu10 buffer_r[642][2]	CLOCK2_50	u6 atpl_component auto_generated pll1 clk[4]	2,500	-2.900	2.978

Fig. 19. Timing Analyzer Report

11. Demo Video

詳見Drive連結:

- https://drive.google.com/file/d/1m7oEIOGxUc3XEV7_6QprvbQ-uQlla8Qt/view?usp=share_link
- https://drive.google.com/file/d/1cdJJwaxu7jSx2ZjT8dYkPwBhDAWJU4wR/view?usp=share_link
- https://drive.google.com/file/d/1w6uno8K3Y0B7F3B76g4jsuax9J9LO0lb/view?usp=share_link

12. 遇到的問題與解決方法

- a. OV7670 -> 5 Mega Pixel Digital Camera Package: 我們一開始嘗試使用OV7670作為我們的camera, 但因為初始化的protocol以及傳data的模式我們都花了很長一段時間才了解, 後來才選擇跟DE2-115搭配的TRDB-D5M camera。
- b. 影像取得: 我們發現沒有zoom in時會有CCD的黑框。在我們沒有選取zoom in 模式的時候, 該模式所回傳的邊緣CCD data會變成黑色的, 會影響我們對陰影的判斷, 因此採用zoom in (較少CCD, 沒有pooling)的模式。此外, 周圍光線/陰影以及YCbCr binarize的threshold會影響取得的資料, 曝光時間也會影響雜訊的程度。
- c. Grayscale.sv (輸出黑白像素的module) 如果要同時取SDRAM資料和輸出到VGA會有clock的問題。因為SDRAM要在同一個時刻寫入相機的RGB並讓VGA讀取, 因此兩個的clock會相差四倍, 我們嘗試過許多不同的clock餵給Grayscale.sv但無法達成正確讀取與顯示, 最後把Grayscale.sv改成只有combinational logic。
- d. VGA和Blob的SYNC問題
 - i. 一開始希望的訊號傳遞方式是 VGA -> Grayscale -> Blob。後來發現VGA有非Active Area的訊號會影響判斷, 因此串流時傳0也被視為物體的像素。

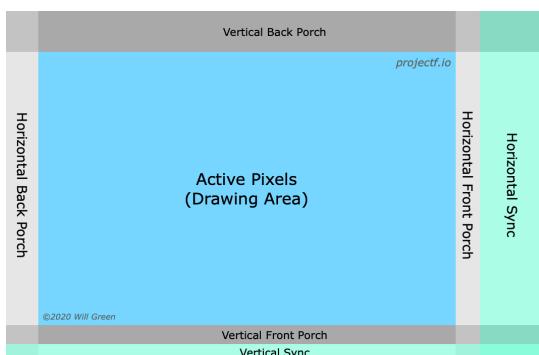


Fig. 20. VGA protocol的active / sync area

- ii. 後來改成 (SDRAM -> Grayscale -> VGA) 和 (SDRAM -> Grayscale -> Blob) 互相配合讀取資料, 以"Capture模式"與"數完"搭配VSYNC作為切換的訊號。但結果是VGA並非從頭讀取SDRAM導致圖片被切割重組。

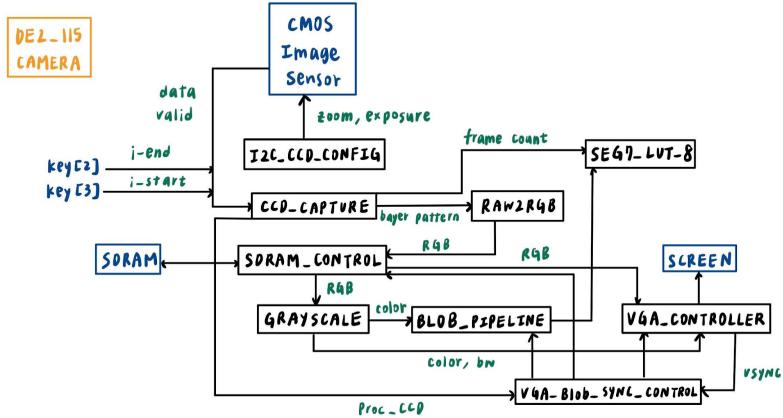


Fig. 21. 先前對Blob / VGA data flow design的嘗試

- iii. 最後改成 VGA + Grayscale -> Blob, VGA同時傳遞是否在Active Area的訊號給Blob判斷需不需要讀input sequence並process。得到的結果是VGA畫面正常, Blob的處理也正常。

e. Synthesizable code

我們的code在一開始很難燒進FPGA, 後來我們發現有兩個問題。

1. 多層的array indexing, 例如 $a[b[c[i]]]$ 會使得logic synthesis卡住。
-> 解決方法: 原本這個語法是CCA演算法的case之一, 後來我們根據我們的圖形的特性、以及更改merge方法的方式對這個演算法做了一些簡化, 去掉這類的語法。
2. 開太大的table size會讓Place and Route卡住。
-> 解決方法: 根據物件可能的大小、個數、邊緣的(因為陰影的)模糊情況做調整, 紿出合理的table size和pixel count bitwidth。

13. Reference

- a. 5 Mega Pixel Digital Camera Package source code, provided by terasic.
- b. M. Bhargava, Chia-Chih Chen, M. S. Ryoo and J. K. Aggarwal, "Detection of abandoned objects in crowded environments," 2007 IEEE Conference on Advanced Video and Signal Based Surveillance, 2007, pp. 271-276, doi: 10.1109/AVSS.2007.4425322.