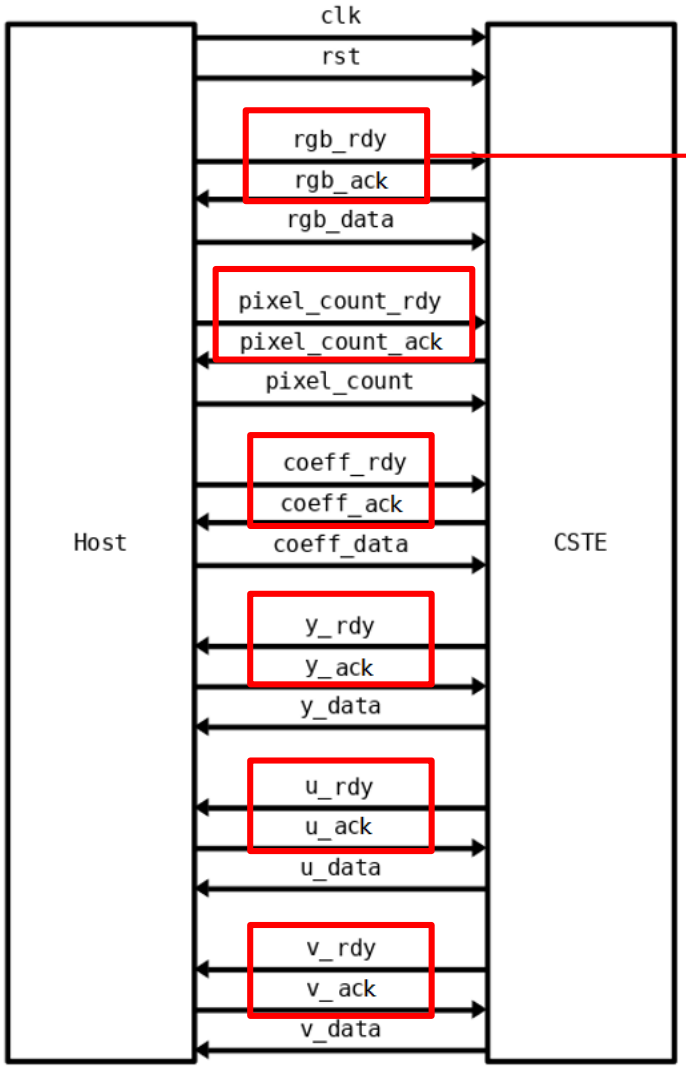
VLSI Lab
Color Space Transform Engine

2020/07/22

問題描述

請各位同學完成 Color Space Transform Engine (CSTE) 的電路設計。此電路輸入二維彩色影像訊號 (RGB)、以及其轉換係數，經電路計算後，得到 YUV 格式的輸出訊號，並降低影像水平大小。有關 CSTE 詳細規格將描述於後。

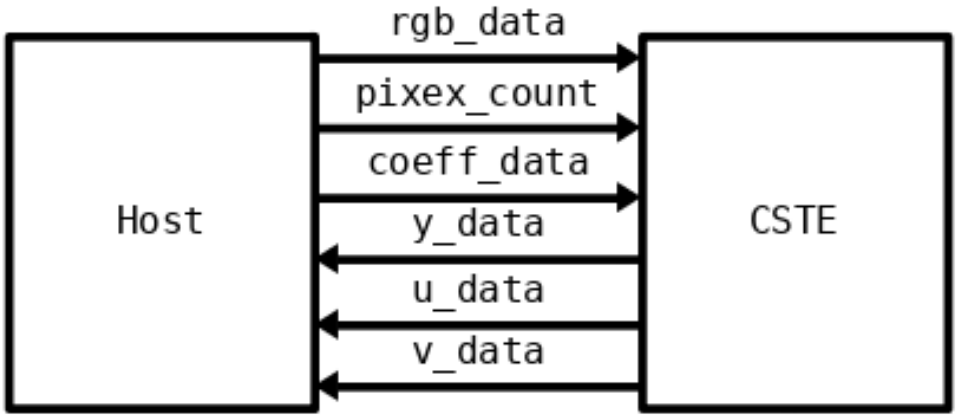
輸入輸出介面



每個資料訊號都有兩個控制訊號 (XX_rdy, XX_ack)

輸入輸出介面

- 移除控制訊號後



系統描述

- YUV Color Space
  - Y: 亮度 (Luminance)
  - U: 色度 (Chrominance)
  - V: 濃度 (Chroma)

$$
\begin{aligned}
Y &= 0.299 * R + 0.587 * G + 0.114 * B \\
U &= -0.169 * R - 0.331 * G + 0.5 * B + 128 \\
V &= 0.5 * R - 0.419 * G - 0.081 * B + 128
\end{aligned}
$$

From Wikipedia

```
Y = (77*R+150*G+29*B+128) >> 8
U = ((-43*R-84*G+127*B+128) >> 8) + 128
V = ((127*R-106*G-21*B+128) >> 8) + 128
```

From Lab Github

系統描述

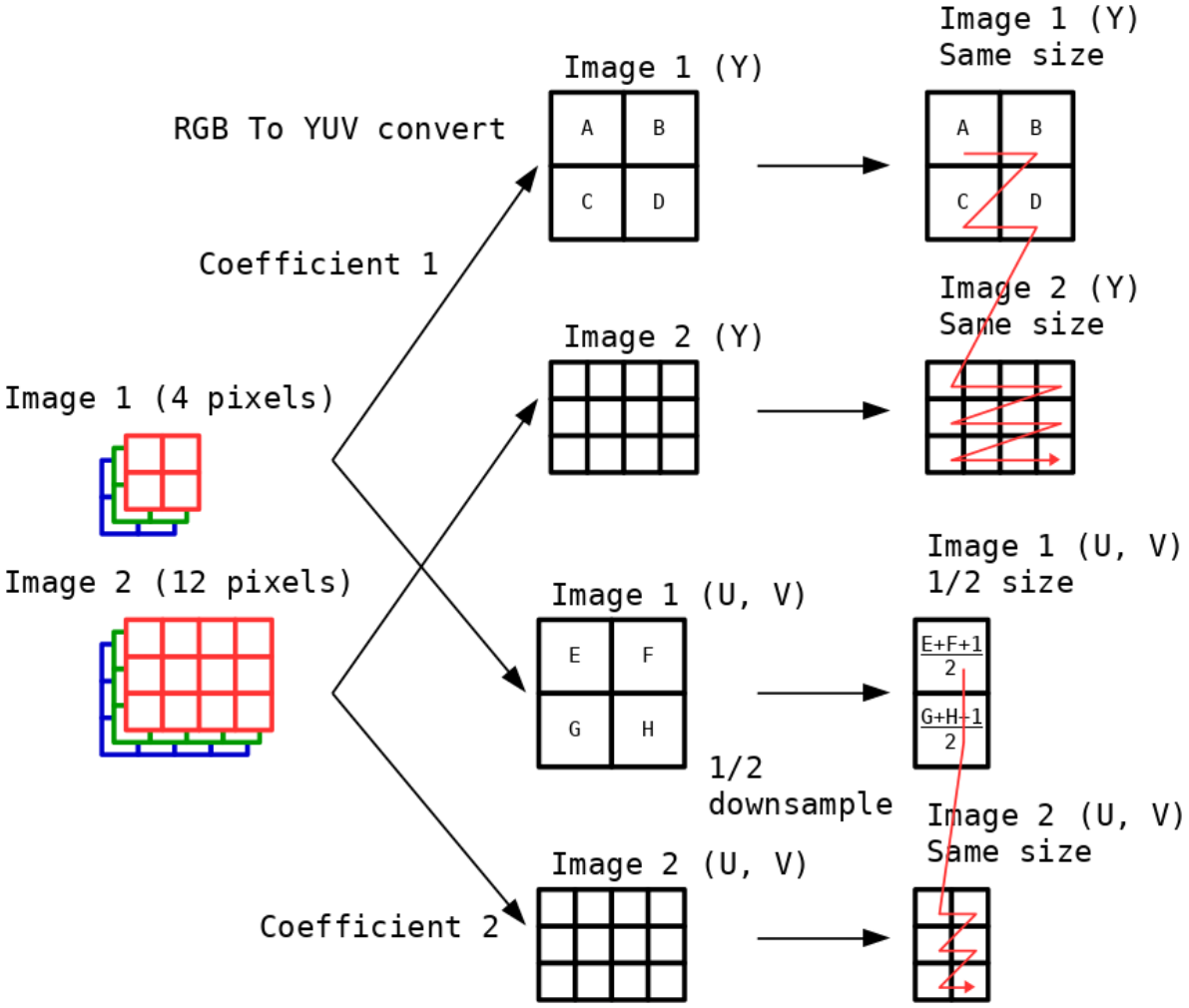
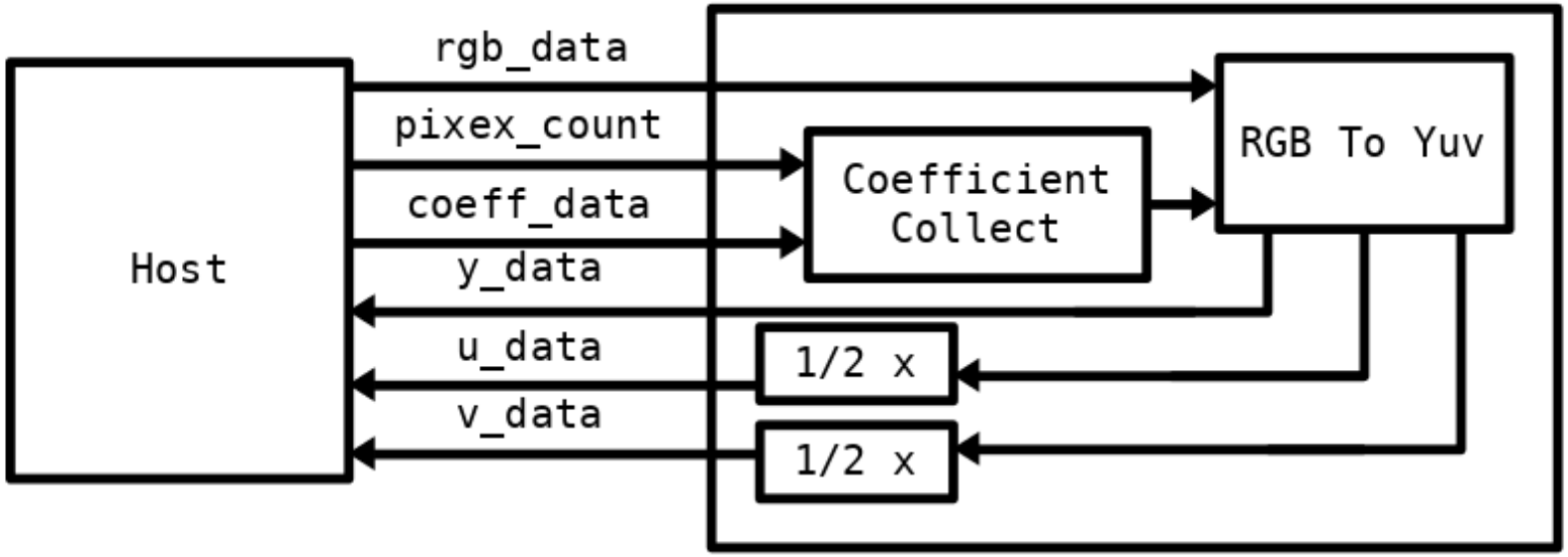
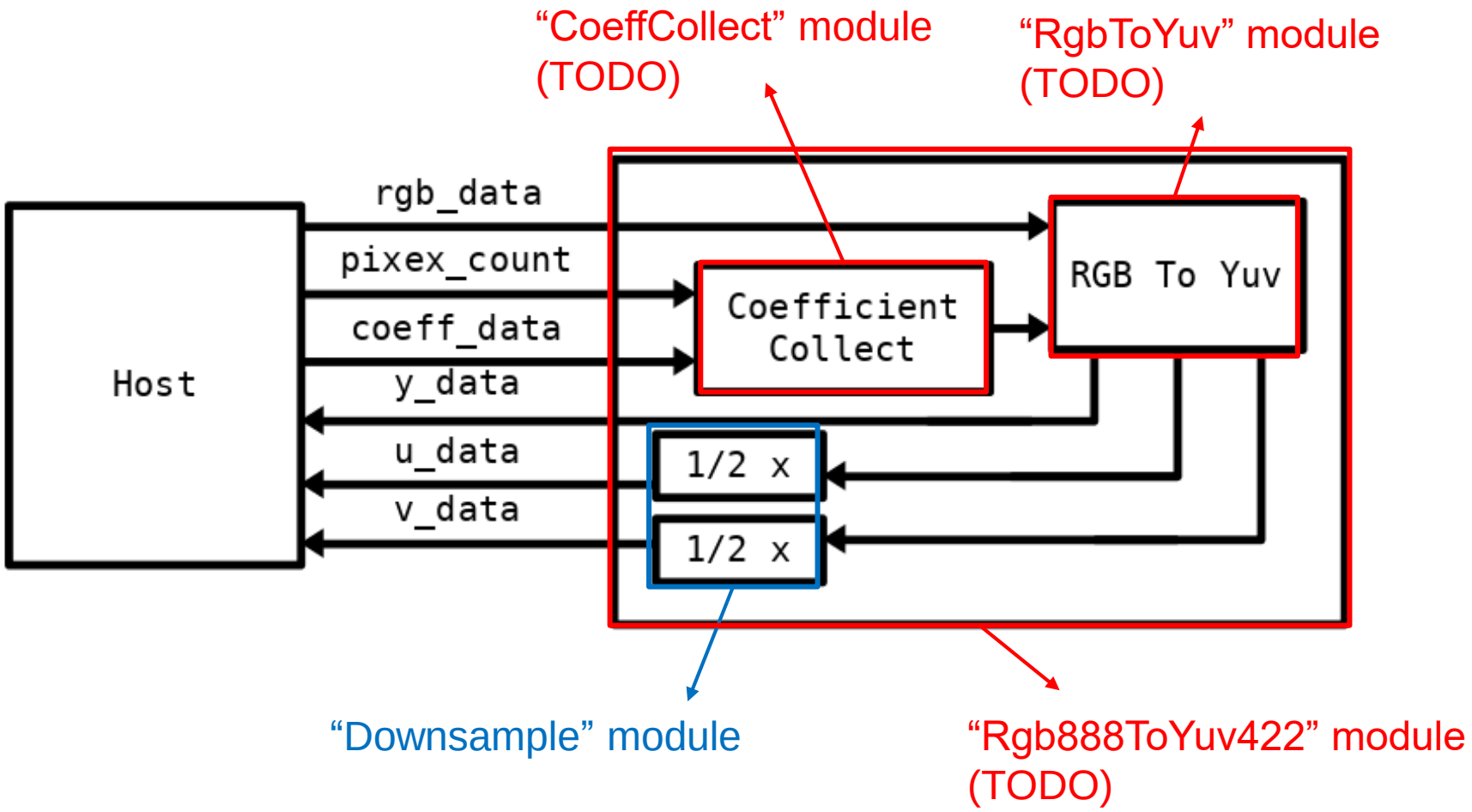


Image 1 (Y)
Same size
Image 1 (Y)
RGB To YUV convert
A
B
C
D
A
B
C
D
Coefficient 1
Image 2 (Y)
Same size
Image 2 (Y)
Image 1 (4 pixels)
Image 1 (U, V)
1/2 size
Image 2 (12 pixels)
Image 1 (U, V)
E
F
G
H
E+F+1
2
G+H+1
2
1/2
downsample
Image 2 (U, V)
Same size
Image 2 (U, V)
Coefficient 2

CSTE 架構

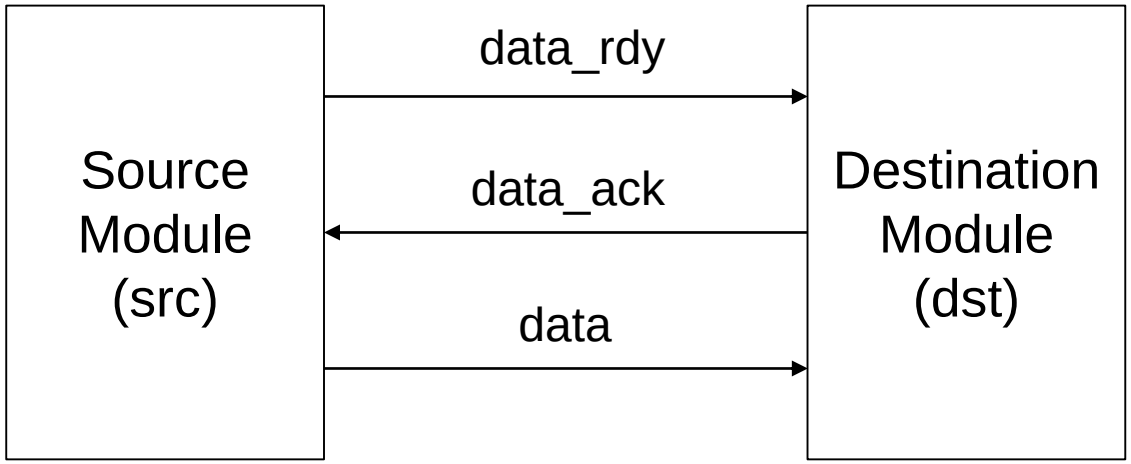

Host
rgb_data
pixex_count
coeff_data
y_data
u_data
v_data
Coefficient Collect
RGB To Yuv
1/2 x
1/2 x

CSTE 架構

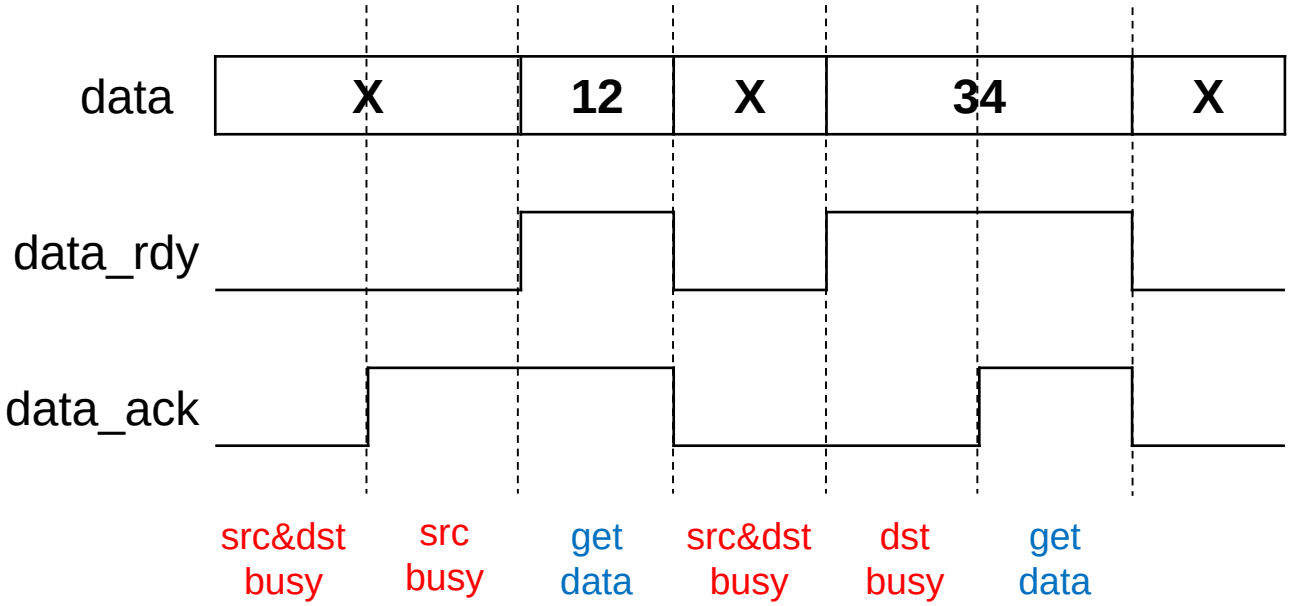

“CoeffCollect” module
(TODO)
“RgbToYuv” module
(TODO)
rgb_data
pixex_count
coeff_data
y_data
u_data
v_data
Host
Coefficient
Collect
RGB To Yuv
1/2 x
1/2 x
“Downsample” module
“Rgb888ToYuv422” module
(TODO)

控制訊號

- 2 wire control



控制訊號

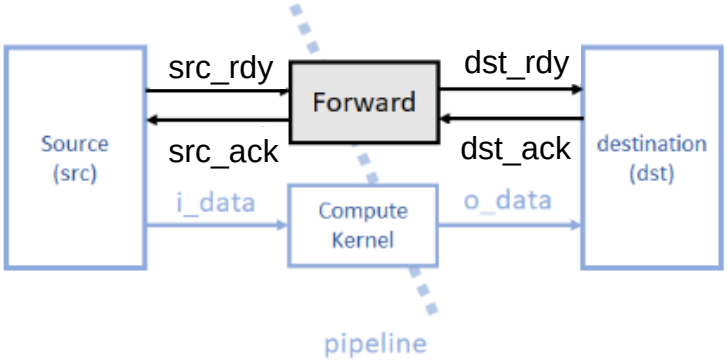

data
X
12
X
34
X
data_rdy
data_ack
src&dst busy
src busy
get data
src&dst busy
dst busy
get data

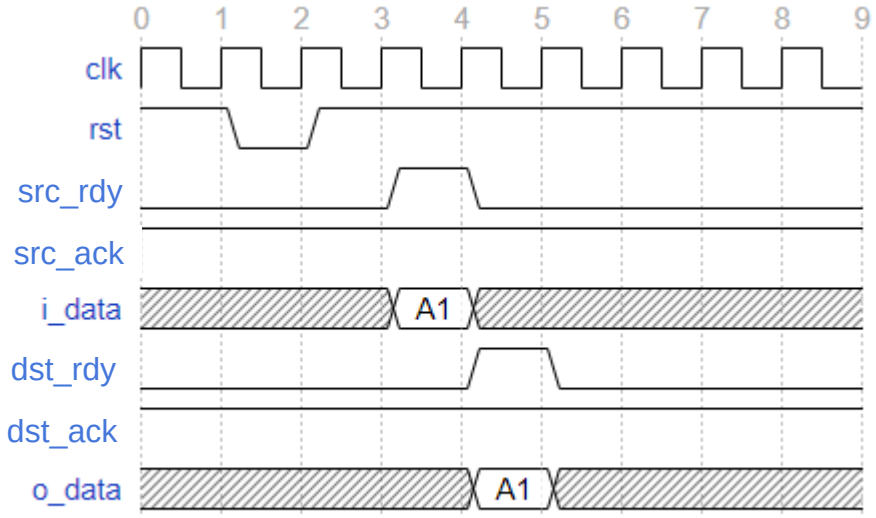
Helper.sv

- Forward Module
- Merge Module
- Broadcast Module
- AcceptIf Module
- IgnoreIf Module

Forward Module

- Forward the handshake protocol signal to the next pipeline stage. This module has to go with one cycle latency computation.

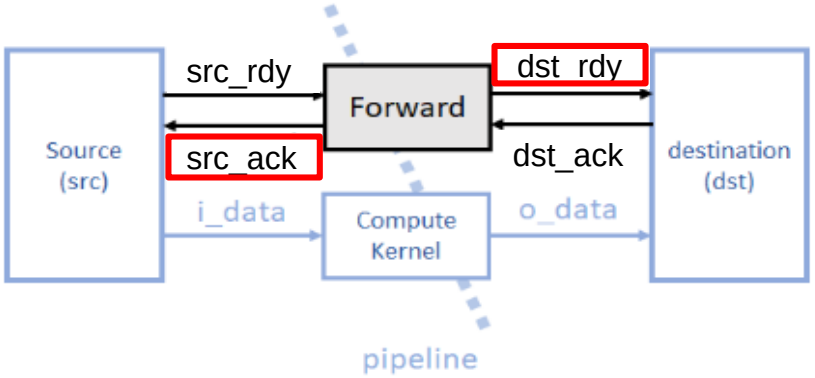




Forward-2wire

Forward Module

```
module Forward(
    input  logic clk,
    input  logic rst,
    input  logic src_rdy,
    output logic src_ack,
    output logic dst_rdy,
    input  logic dst_ack
);
    logic dst_rdy_w;
    assign dst_rdy_w = src_rdy || (dst_rdy && !dst_ack);
    assign src_ack = !dst_rdy || dst_ack;
    always_ff @(posedge clk or negedge rst) begin
        if (!rst) dst_rdy <= 1'b0;
        else dst_rdy <= dst_rdy_w;
    end
endmodule
```



src有data傳過來

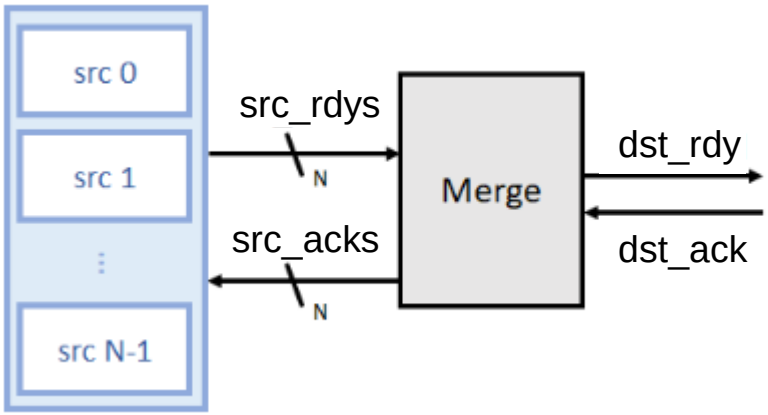
data還留在register沒被dst拿走

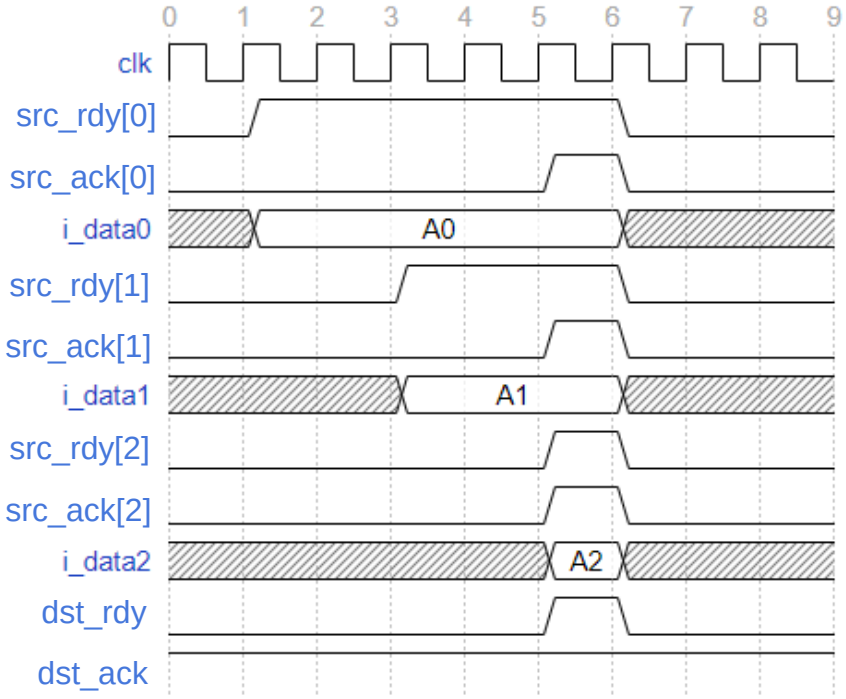
dst 正在拿走 data register 的 data

data register是空的

Merge Module

- Handling the control signal between multiple sources and ensuring all the sources sending next input data until all the data from sources are collected.

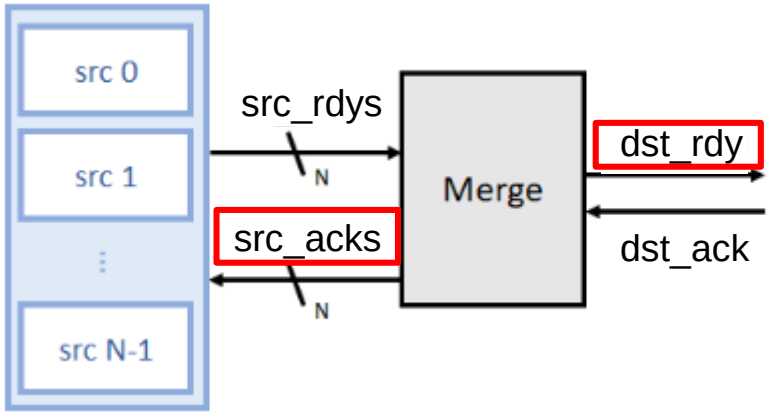




Merge-2wire, N=3

Merge Module

```
module Merge(
    src_rdys,
    src_acks,
    dst_rdy,
    dst_ack
);
    parameter N = 2;
    input  logic [N-1:0] src_rdys;
    output logic [N-1:0] src_acks;
    output logic dst_rdy;
    input  logic dst_ack;
    assign dst_rdy = &src_rdys;
    assign src_acks = {N{dst_rdy && dst_ack}};
endmodule
```

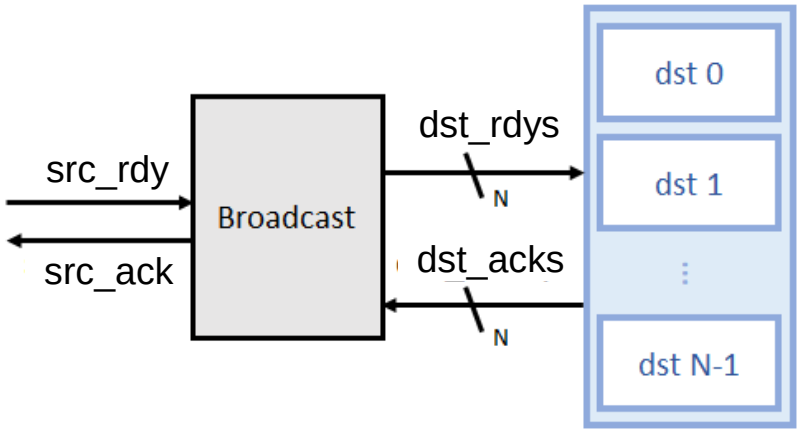


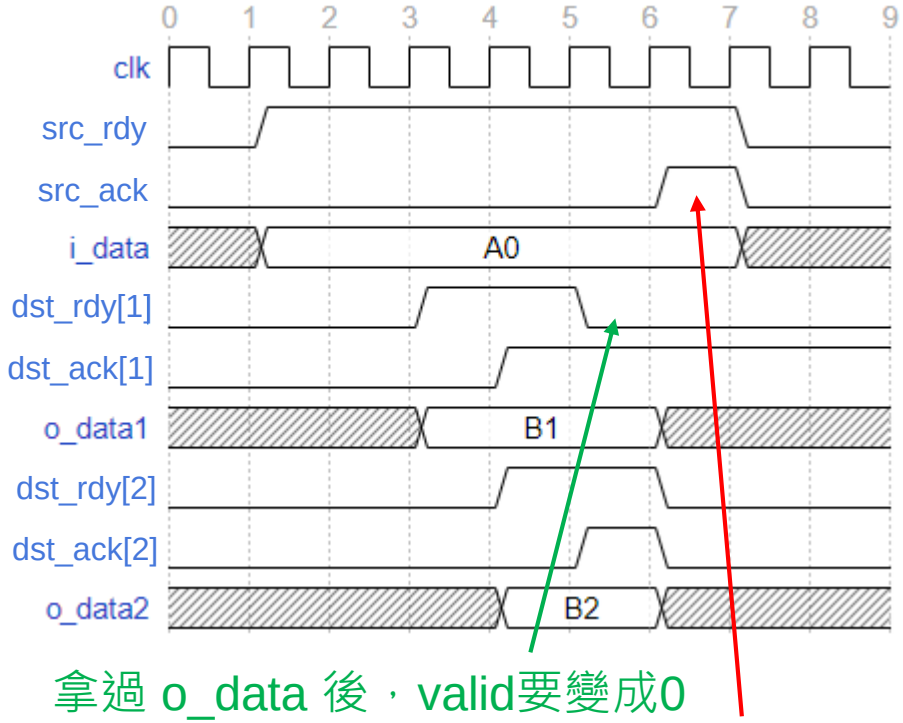
所有 src 都準備好

dst 可以拿 data 同時 所有 src 也準備好

Broadcast Module

- Handling the control signal between multiple destinations and ensuring input data remaining the same until all the destinations receiving the valid output data.



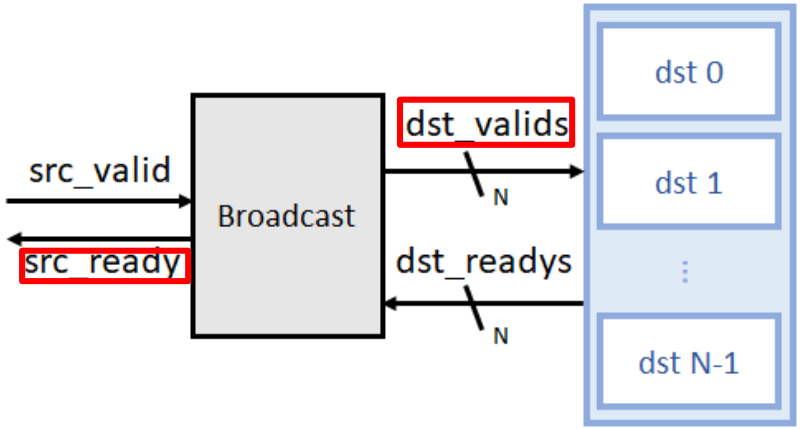


拿過 o_data 後，valid要變成0

所有 dst 都拿過 o_data 後，才能拿下一筆 i_data

Broadcast Module

```
module Broadcast(
    clk,
    rst,
    src_rdy,
    src_ack,
    dst_rdys,
    dst_acks
);
    parameter N = 2;
    input  logic clk;
    input  logic rst;
    input  logic src_rdy;
    output logic src_ack;
    output logic [N-1:0] dst_rdys;
    input  logic [N-1:0] dst_acks;
    logic [N-1:0] got, got_test, got_w;
    assign dst_rdys = {N{src_rdy}} & ~got;
    assign got_test = got | dst_acks;
    assign src_ack = &got_test;
    assign got_w = (src_rdy && !src_ack) ? got_test : '0;
    always_ff @(posedge clk or negedge rst) begin
        if (!rst) got <= '0;
        else got <= got_w;
    end
endmodule
```



src 準備好，並扣掉拿過的 dst

加上準備要拿 o_data 的 dst

所有 dst 都拿過 o_data 後，才能拿下一筆 i_data

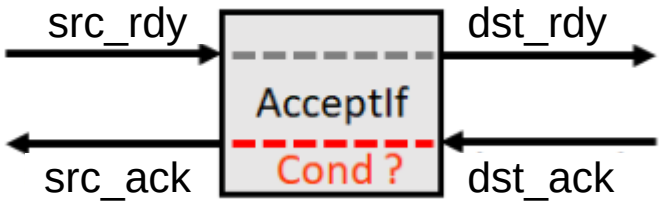
如果還有 dst 還沒拿 o_data → 更新 got
反之 dst 都拿過了或 src 還沒準備好 → got 歸0

記錄哪些 dst 已經拿過 o_data

AcceptIf Module

- Pass the dst_ready to src_ready when condition is high

```
module AcceptIf(
    input  logic cond,
    input  logic src_rdy,
    output logic src_ack,
    output logic dst_rdy,
    input  logic dst_ack,
    output logic accept
);
    parameter bit COND = 1;
    assign accept = cond == COND;
    assign dst_rdy = src_rdy;
    assign src_ack = dst_ack && accept;
endmodule
```

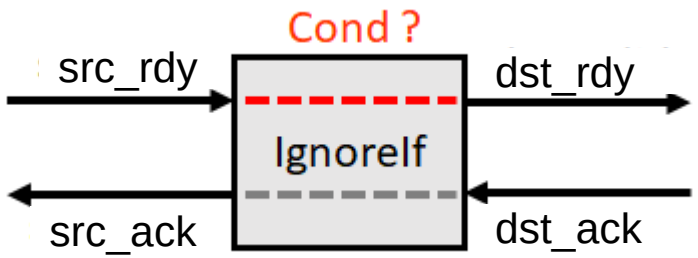


accept=1時，將dst_ready傳給src_ready
accept=0時， src_ready必為0

IgnoreIf Module

- Skip the src_valid signal to dst_valid when cond is high.

```
module IgnoreIf(
    input  logic cond,
    input  logic src_rdy,
    output logic src_ack,
    output logic dst_rdy,
    input  logic dst_ack,
    output logic ignore
);
    parameter bit COND = 1;
    assign ignore = cond == COND;
    assign dst_rdy = !ignore && src_rdy;
    assign src_ack = ignore || dst_ack;
endmodule
```

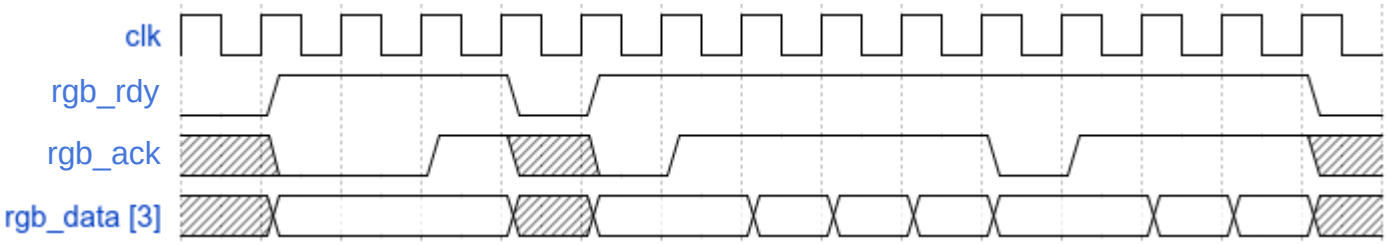


ignore=1時，dst_valid必為0
ignore=0時，將src_valid傳給dst_valid

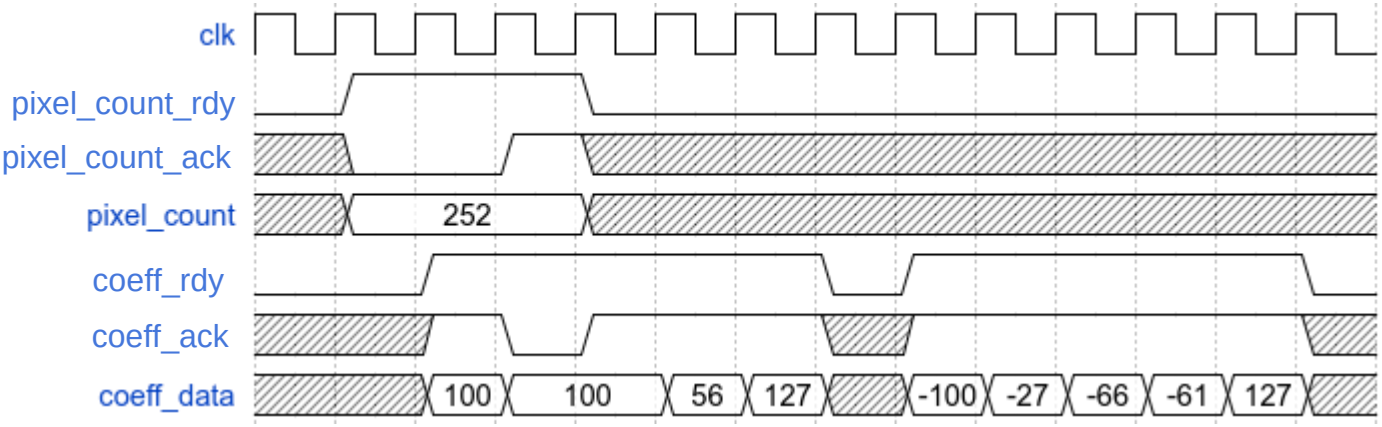
ignore=1時，src_ready必為1，跳過這筆data
ignore=0時，將dst_ready傳給src_ready

CSTE 輸入時序規格

- RGB pixel input



- Coefficient input
  - 每次 pixel_count 對應到 9 次 coeff_data 的輸入



CSTE 輸出時序規格

- 完成電路運算後，欲將資料輸出需將 y_valid設為 high，告知 host 端有轉換過後的 pixel 輸出，本題並未規定需要連續輸出，可自行控制 y_valid 控制訊號。(U, V同理)
- 當所有資料輸出後，系統便會自動結束模擬並檢查。
- 預設模擬時間約 2000 cycles，若屆時未達到足夠筆的資料輸出，系統將輸出 timeout 信號。
- 預設模擬有 10+18 個 pixel，也就是說必須在 y_data 觀察到 28 個輸出，u_data, v_data 看到 14 個。

電路測試

- 本練習電路可以在 sim/ 資料夾下，輸入以下命令分別測試 CSTE 以及 submodule。
  - make top
  - make coeff_col
  - make rgb2yuv
  - make downsample