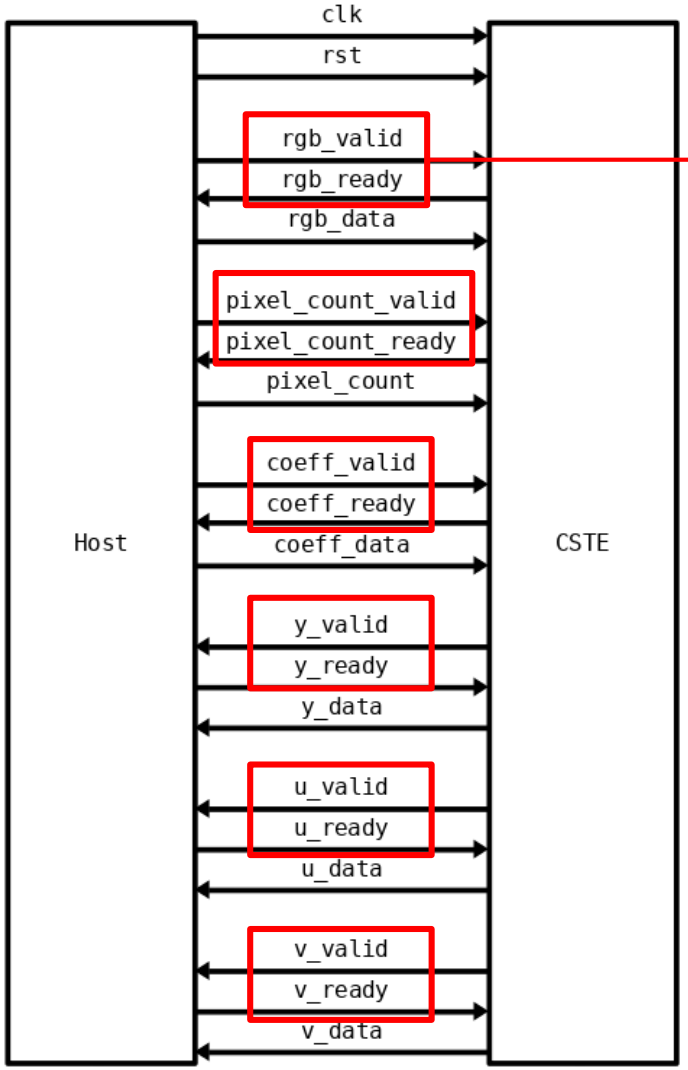
VLSI_Lab

輸入輸出介面

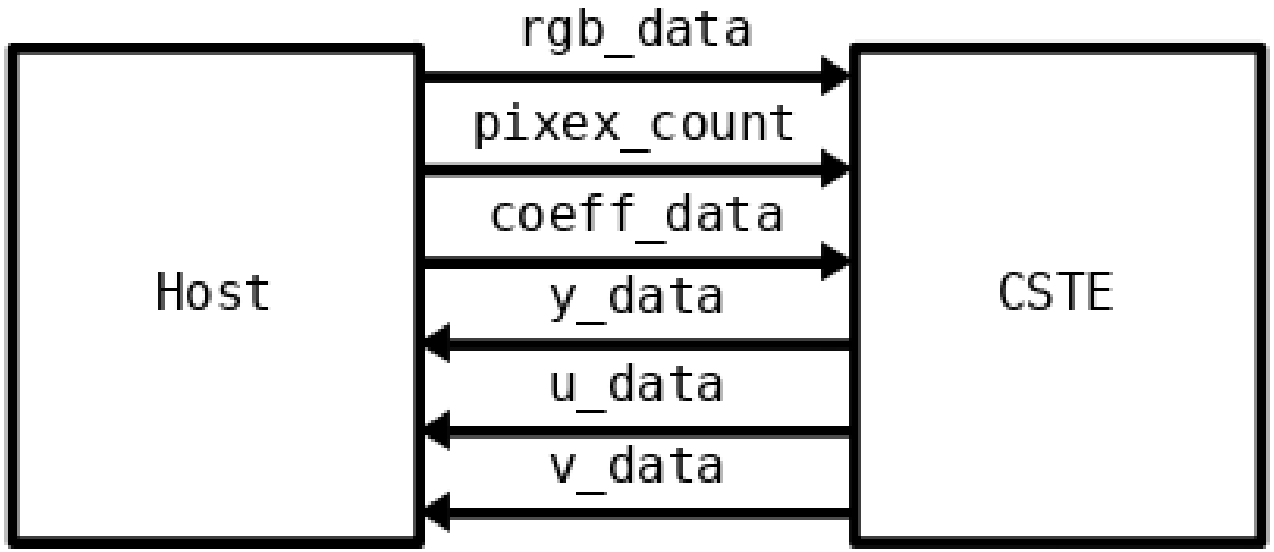


RGB to YUV422

每個資料訊號都有兩個控制訊號 (XX_valid, XX_ready)

輸入輸出介面

Finish Color Space Transform Engine (CSTE) circuit design



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 Lab1 Github

定點數(Fix Point)

8 bits for whole number part

8 bits for decimal part

0.299 --> 77

77_{10} --> 01001101_2

$0.299 \times 2^8 = 76.544$

$2^{-2}+2^{-5}+2^{-6}+2^{-8}=0.30078125$

Fix Point 運算

Add / Subtraction

Align the decimal point

Multiplication

Extend bits

Input: 8 bits for whole number part, 8 bits for decimal part

Output: 15 bits for whole number part, 16 bits for decimal part

Keep 16bits

Behavior

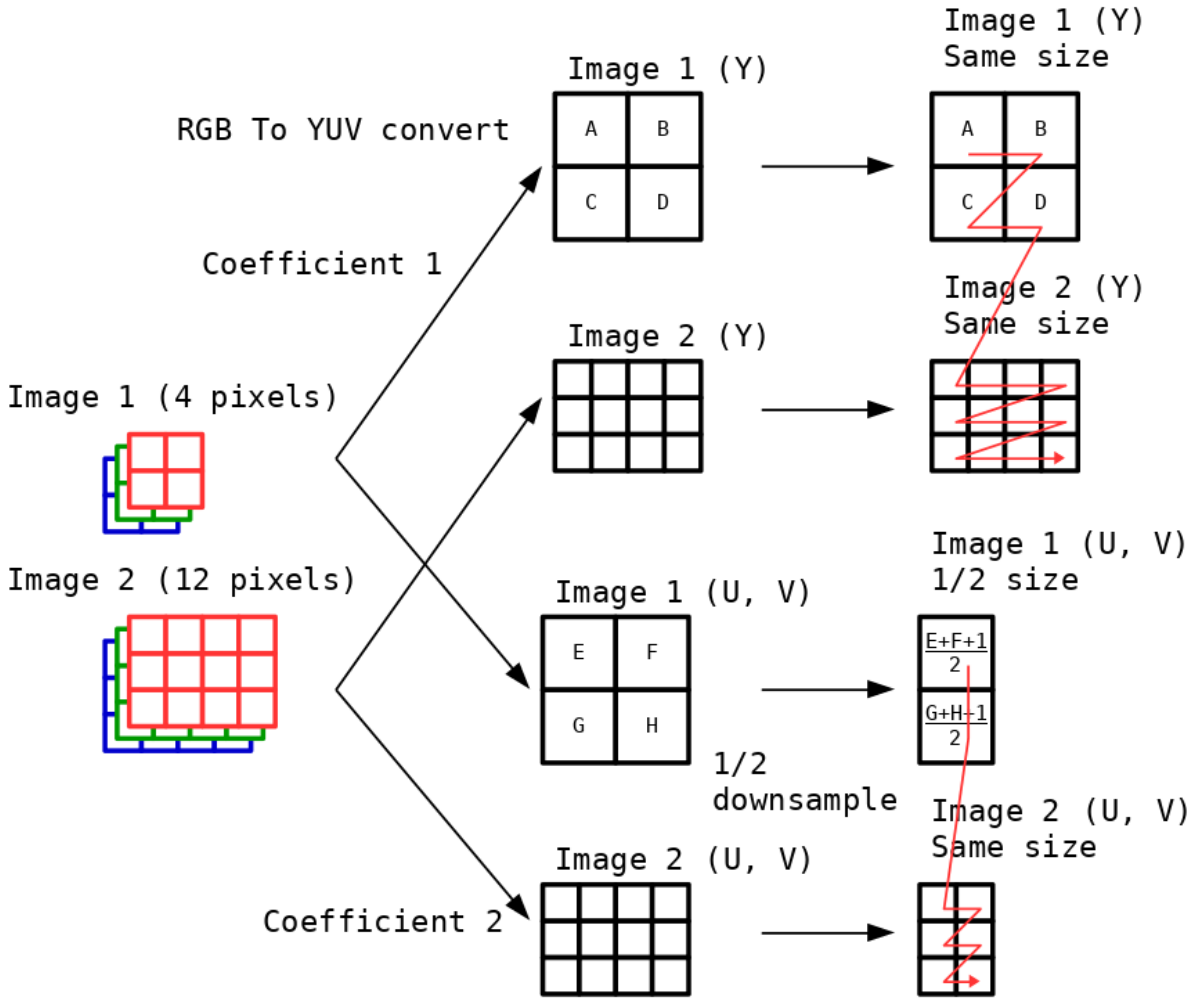
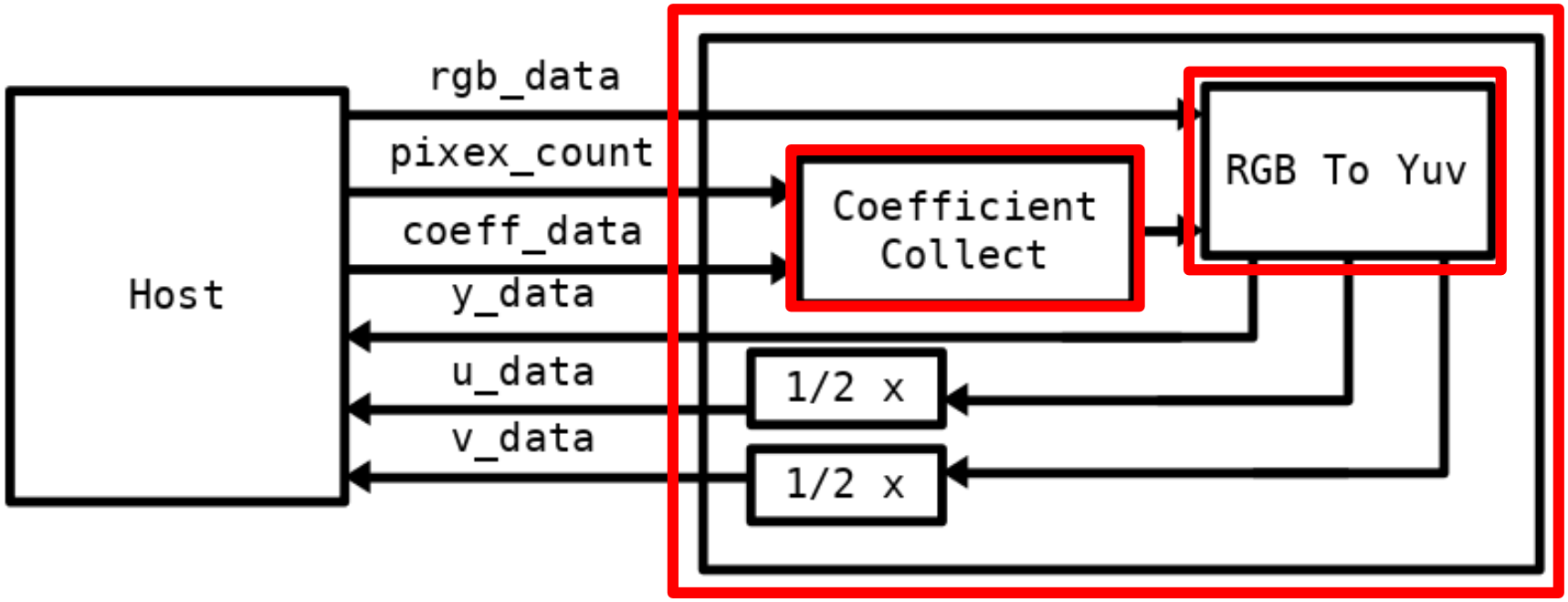
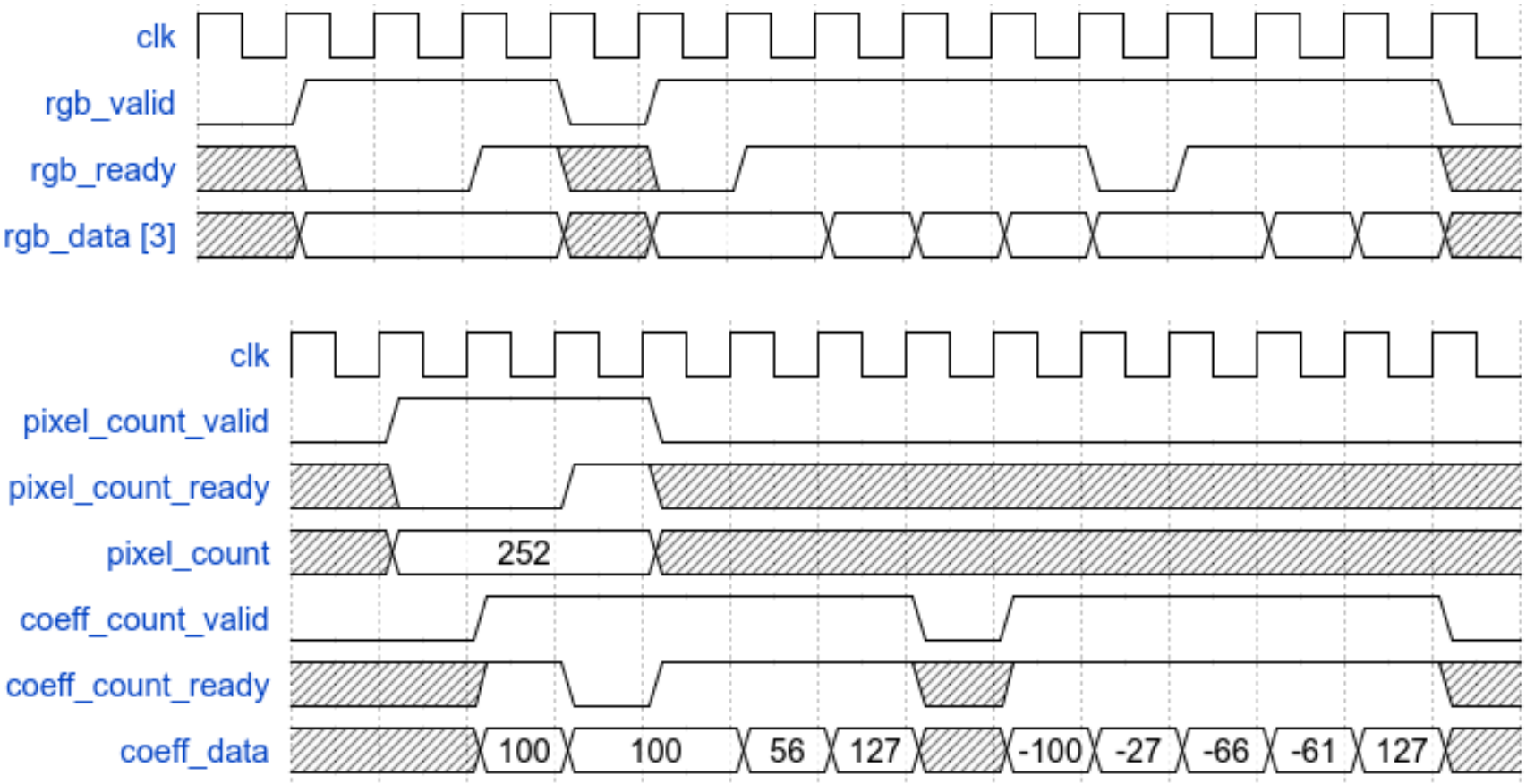


Image 1 (Y)
Same size
Image 1 (Y)
RGB To YUV convert
A
B
C
D
Coefficient 1
Image 2 (Y)
Same size
Image 2 (Y)
Image 1 (4 pixels)
Image 1 (U, V)
1/2 size
Image 2 (12 pixels)
Image 1 (U, V)
E
F
G
H
E+F+1
2
G+H+1
2
1/2
downsample
Image 2 (U, V)
Same size
Image 2 (U, V)
Coefficient 2

CSTE Over View



Wave Form


clk
rgb_valid
rgb_ready
rgb_data [3]
clk
pixel_count_valid
pixel_count_ready
pixel_count
252
coeff_count_valid
coeff_count_ready
coeff_data
100
100
56
127
-100
-27
-66
-61
127

Folder

design
- Your design

sim
- Test bench, model

syn
- Gate level synthesis files

Helper.sv

Forward Module

Merge Module

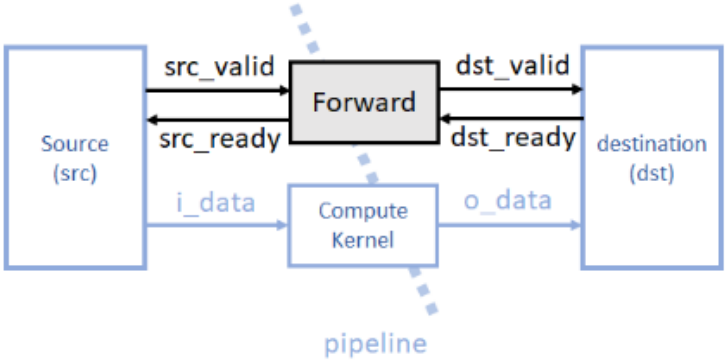
Broadcast Module

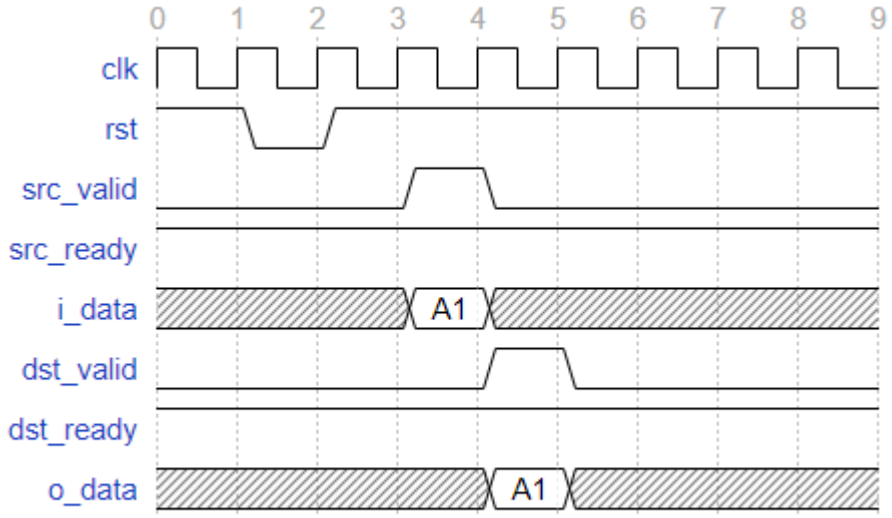
AcceptIf Module

IgnoreIf Module

Forward Module

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

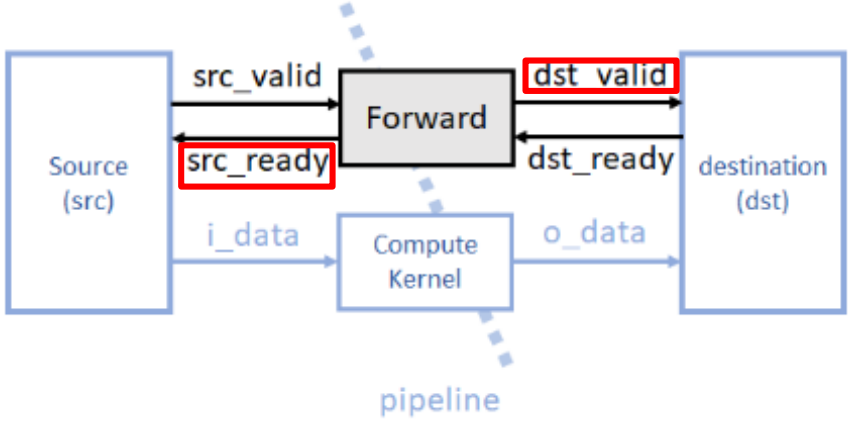




Forward-2wire

Forward Module

```
module Forward(
        input  logic clk,
        input  logic rst,
        input  logic src_valid,
        output logic src_ready,
        output logic dst_valid,
        input  logic dst_ready
);
        logic dst_valid_w;
        assign dst_valid_w = src_valid || (dst_valid && !dst_ready);
        assign src_ready = !dst_valid || dst_ready;
        always_ff @(posedge clk or negedge rst) begin
                if (!rst) dst_valid <= 1'b0;
                else dst_valid <= dst_valid_w;
        end
endmodule
```



src有data傳過來

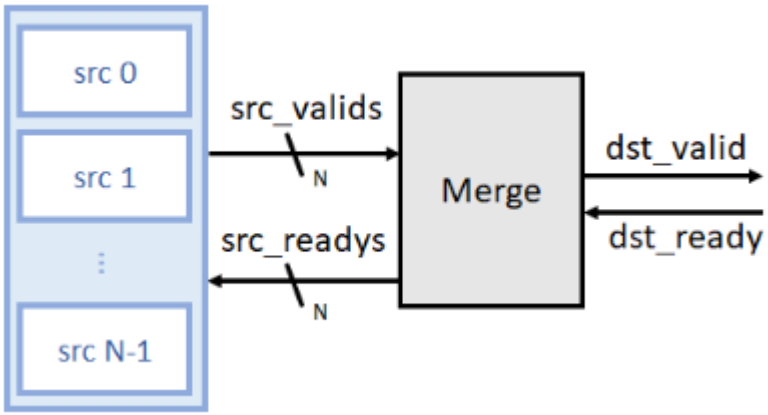
data還留在register沒被dst拿走

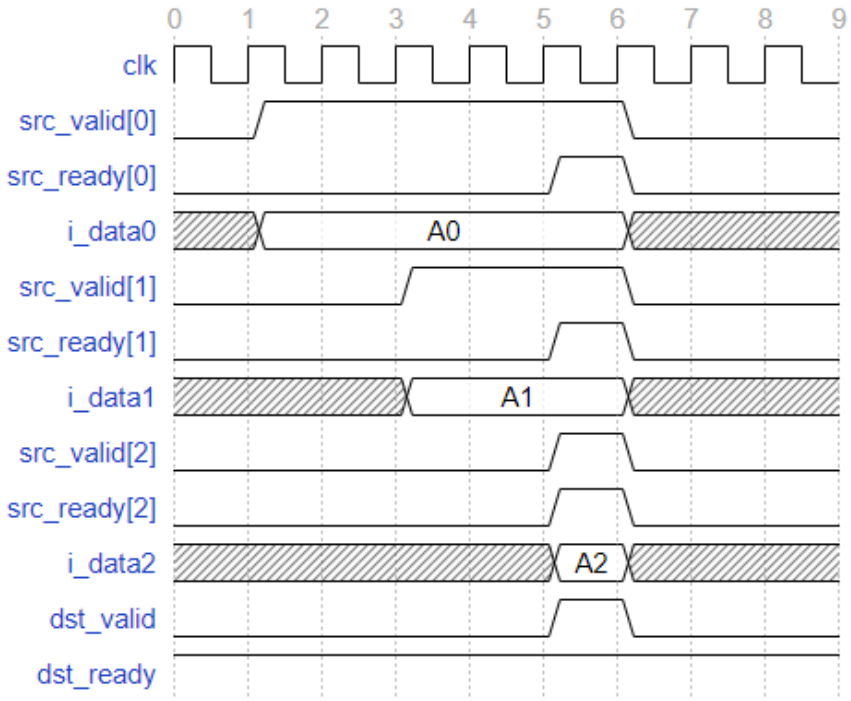
dst 正在拿走 data register 的 data

data register是空的

Merge Module

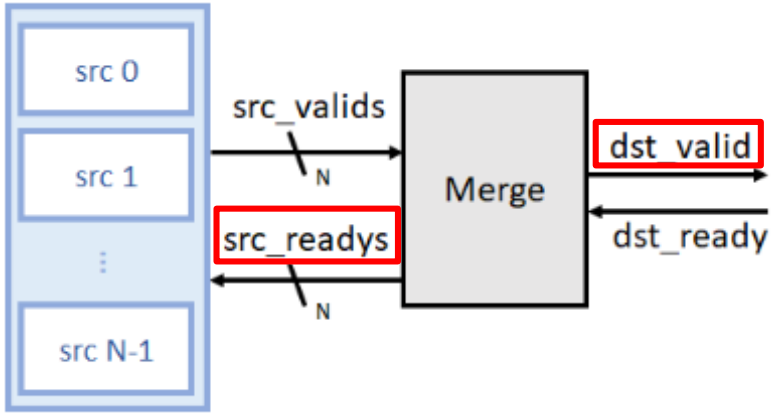
Handling the control signal between multiple sources and ensuring all the sources sending next input data until all the data from sources are collected.





Merge-2wire, N=3

Merge Module



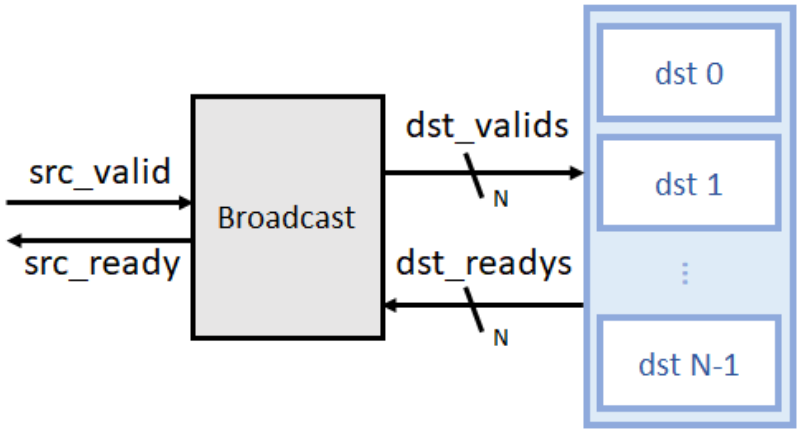

```
module Merge(
        src_valids,
        src_readys,
        dst_valid,
        dst_ready
);
        parameter N = 2;
        input  logic [N-1:0] src_valids;
        output logic [N-1:0] src_readys;
        output logic dst_valid;
        input  logic dst_ready;
        assign dst_valid = &src_valids;
        assign src_readys = {N{dst_valid && dst_ready}};
endmodule
```

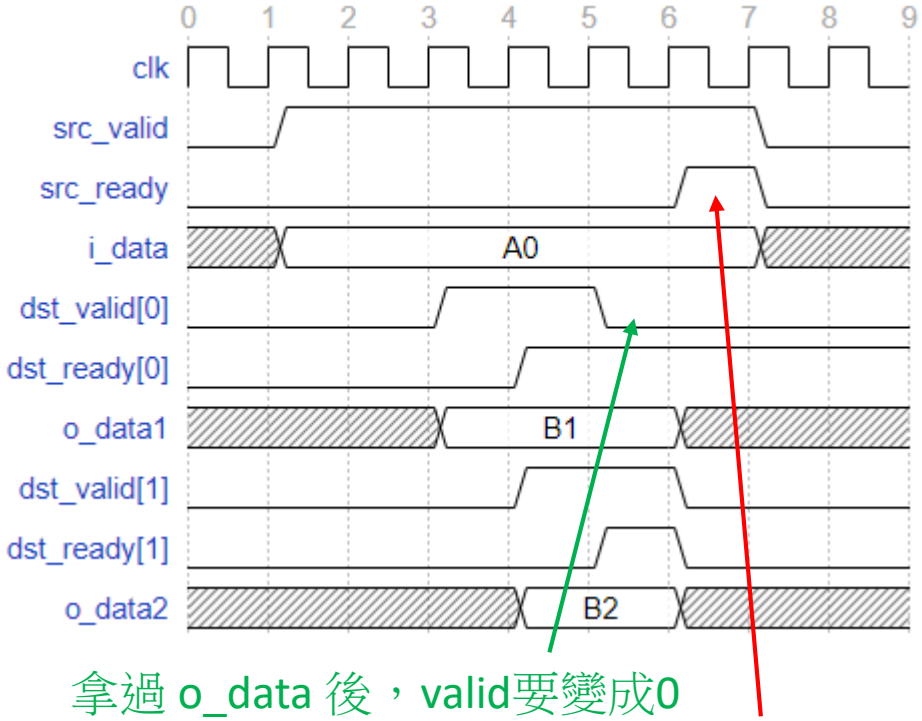
所有 src 都準備好

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.



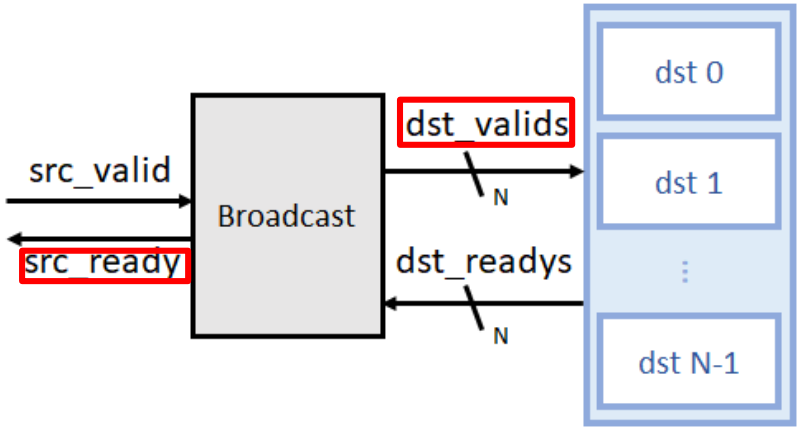


拿過 o_data 後，valid要變成0

所有 dst 都拿過 o_data 後，才能拿下一筆 i_data

Broadcast Modul

```
module Broadcast(
        clk,
        rst,
        src_valid,
        src_ready,
        dst_valids,
        dst_readys
);
        parameter N = 2;
        input  logic clk;
        input  logic rst;
        input  logic src_valid;
        output logic src_ready;
        output logic [N-1:0] dst_valids;
        input  logic [N-1:0] dst_readys;
        logic [N-1:0] got, got_test, got_w;
        assign dst_valids = {N{src_valid}} & ~got;
        assign got_test = got | dst_readys;
        assign src_ready = &got_test;
        assign got_w = (src_valid && !src_ready) ? 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

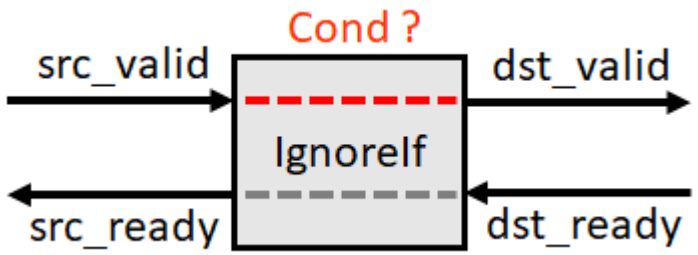
如果還有 dst 還沒拿 o_data → 更新 got
反之 dst 都拿過了或 src 還沒準備好 → got 歸0

記錄哪些 dst 已經拿過 o_data

IgnoreIf Module

Skip the src_valid signal to dst_valid when cond is high.

```
module IgnoreIf(
        input  logic cond,
        input  logic src_valid,
        output logic src_ready,
        output logic dst_valid,
        input  logic dst_ready,
        output logic ignore
);
        parameter bit COND = 1;
        assign ignore = cond == COND;
        assign dst_valid = !ignore && src_valid;
        assign src_ready = ignore || dst_ready;
endmodule
```



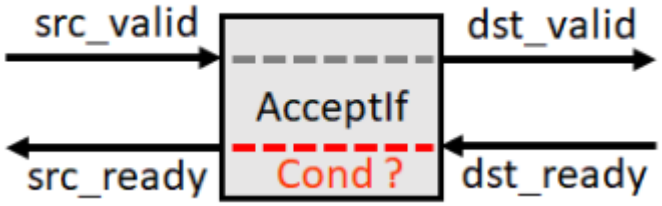
ignore=1時，dst_valid必為0
ignore=0時，將src_valid傳給dst_valid

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

AcceptIf Module

Pass the dst_ready to src_ready when condition is high

```
module AcceptIf(
        input  logic cond,
        input  logic src_valid,
        output logic src_ready,
        output logic dst_valid,
        input  logic dst_ready,
        output logic accept
);
        parameter bit COND = 1;
        assign accept = cond == COND;
        assign dst_valid = src_valid;
        assign src_ready = dst_ready && accept;
endmodule
```



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

Nicotb

- Calculate answer
- Connect to Verilog
- Initialization
- start simulation

Calculate answer

Generate pattern and golden

```
# Calculate answer
coeff_test = M.coeff_test
N_ANS = sum(x[0] for x in coeff_test)
ans = np.vstack(
    np.repeat(np.reshape(c[1], (1,-1)), c[0], axis=0)
    for c in coeff_test
)
```

Connect to Verilog

```
# Connect to Verilog
(
    ivalid, iready,
    ovalid, oready,
    idata, odata,
) = CreateBuses([
    (("dut", "i_valid"),),
    (("dut", "i_ready"),),
    (("dut", "o_valid"),),
    (("dut", "o_ready"),),
    (("dut", "i_data"),),
    (("dut", "o_data"),),
])
```

```
module Downsample(
    input clk,
    input rst,
    // input pixel stream
    input  logic       i_valid,
    output logic       i_ready,
    input  logic [7:0] i_data,
    // output pixel stream
    output logic       o_valid,
    input  logic       o_ready,
    output logic [7:0] o_data
);
```

Initialization

- Define clk and rst even
- Create score board
- Define master and slave
- Extract the data bus of master
- Check the data at slave.

```
rst_out_ev, ck_ev = CreateEvents(["rst_out", "ck_ev"])
scb = Scoreboard("Counter")
test = scb.GetTest("Counter")
st = Stacker(N_ANS, callbacks=[test.Get])
bg = BusGetter(callbacks=[st.Get])
master = TwoWire.Master(ivalid, iready, idata, ck_ev, A=1, B=2)
slave = TwoWire.Slave(ovalid, oready, odata, ck_ev, A=1, B=2, callbacks=[bg.Get])
mdata = master.values
test.Expect((gold_out[:,np.newaxis],))
```

Start simulation

```
yield rst_out_ev
yield ck_ev
```

```
def Iter():
    for i in gold_in:
        mdata.i_data[0] = i
        yield mdata
yield from master.SendIter(Iter())
```

```
def IterP():
    for i in coeff_test:
        mdatap.pixel_count[0] = i[0]
        yield mdatap

def IterC():
    for i in coeff_test:
        for j in i[1].flat:
            mdatac.coeff_data[0] = j
            yield mdatac
th_1 = JoinableFork(masterp.SendIter(IterP()))
th_2 = JoinableFork(masterc.SendIter(IterC()))
yield from th_1.Join()
yield from th_2.Join()
th_1.Destroy()
th_2.Destroy()
```

```
for i in range(100):
    yield ck_ev
assert st.is_clean
FinishSim()
```

TODO

design

1. CoeffCollect
2. RgbToYuv
3. Rgb888ToYuv422

sim

MyModel.py

RgbToYuv_test.py

Rgb888ToYuv422_test.py

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

如果使用 Verilog 語法，輸入以下命令

- make USE_VERILOG=true top
- make USE_VERILOG=true coeff_col
- make USE_VERILOG=true rgb2yuv
- make USE_VERILOG=true downsample