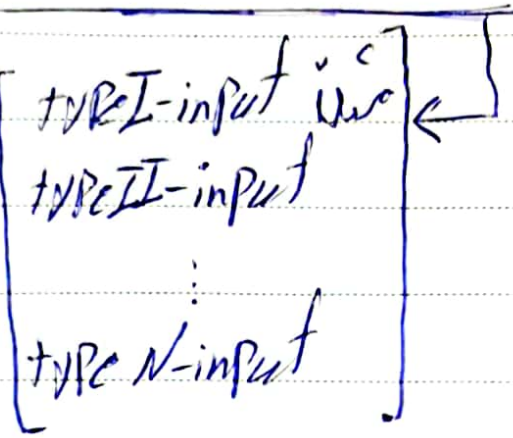
Subject.
Date.

NOC (پرطرفدارهای سابله)

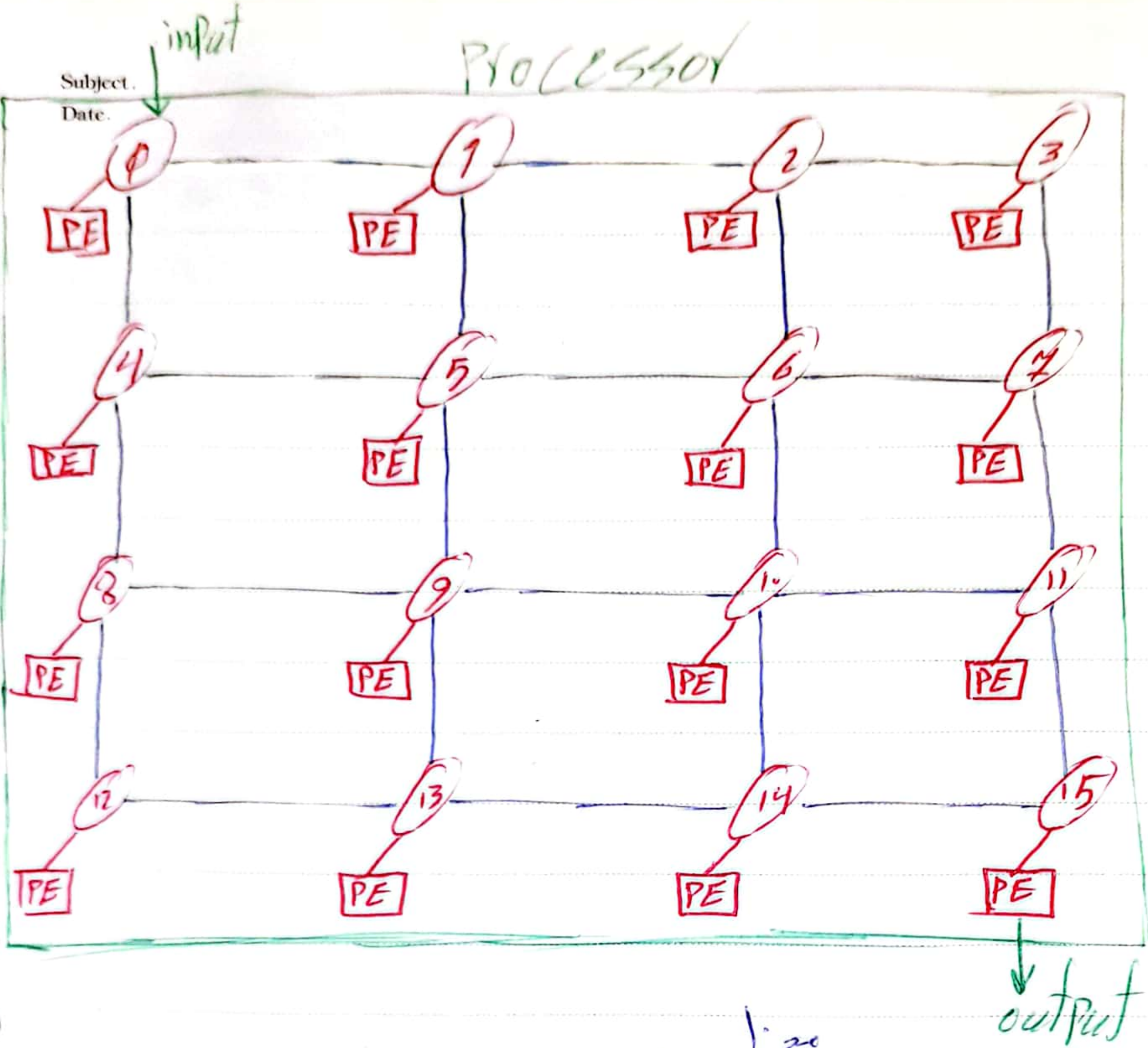
اینده NOC به این صورت است که مثلاً فرض کنید ما یک پرطرفدار به داده‌های ورودی آن بدهیم که نوع ورودی تقسیم‌بندی کنید.



حال فرض کنید ورودی تایپ X، از بین ورودی‌های X_1 تا X_5، با این عملیات‌های X_3 و X_2 و X_4 انجام شود. حال سوال اینجاست چگونه می‌توان یک پرطرفدار طراحی کرد که این تشابه‌ها را مدیریت کند و جلوی آن را بگیرد.

برای این مورد می‌توان از کلاسیر پرطرفدارهای سابله (یا NOC) استفاده کرد.
برای مثال پرطرفدار زیر را در نظر بگیرید:

Subject.
Date.



این پردازنده ۱۶ تا unit دارد هر unit منابع مختص (مجزا) به خودش را دارد

مثل PE (processing Element)، Memory و .... همچنین

این unit ها بطوری به هم متصل شده اند که بین هر دو Unit

حداقل یک مسیر وجود دارد.

حال فرض کنید ما ۳ نوع ورودی داریم که ورودی هر نوع نیاز است که دیگری عملیات انجام شود مثلاً

+ روی ورودی TYPE I باید عملیات های 14 ، 9 ، 5 ، 0 ،

+ روی ورودی TYPE II باید عملیات های 3 ، 2 ، 1

+ و روی ورودی TYPE III باید همه عملیات ها انجام شوند.

در این حالت زمانی که یک ورودی می‌آید [crossed out] سناریو زیر اتفاق می‌افتد:

۱- داده وارد Unit 0 می‌شود

۲- اگر نیاز به عملیات 0 بود، وارد PE 0 می‌شود در غیر این صورت وارد Unit 1 می‌شود.

۳- مشابه عملیات ۲ تکرار می‌شود تا داده وارد Unit 15 شود.

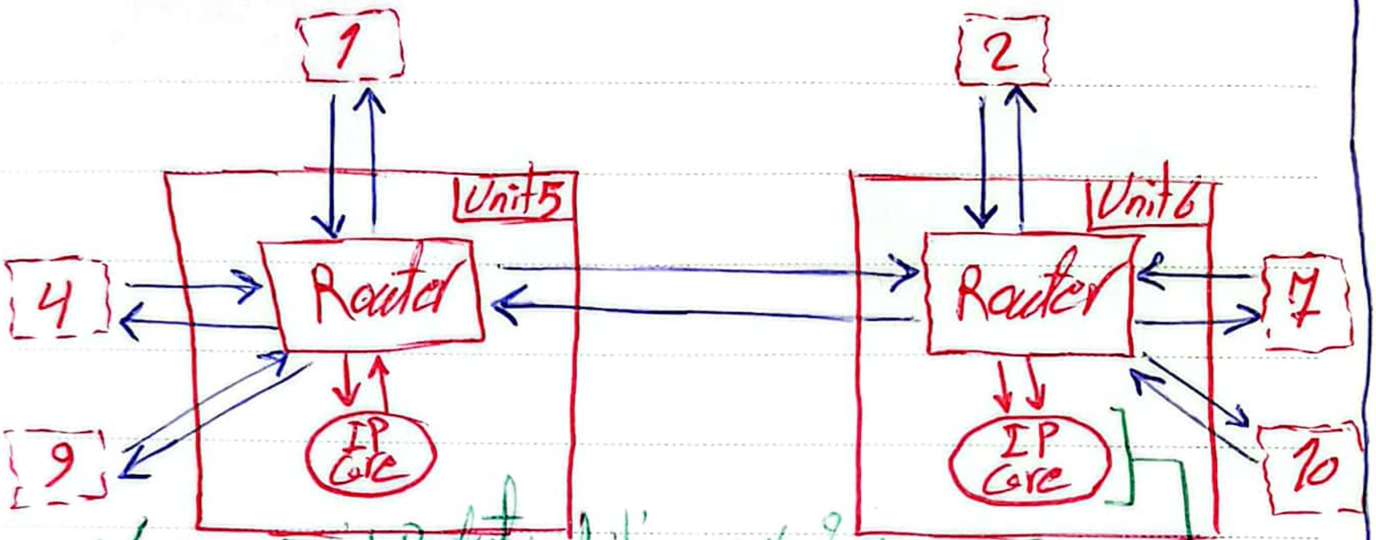
۴- در نهایت داده از Unit 15 خارج و [crossed out] روی خروجی قرار می‌گیرد.

* همانطور که گفته شد نیاز است که سیستم بصورت بهینه کار کند، در نتیجه نیاز به طراحی الگوریتم برای routing پکت ها داریم که تا Packet ها بصورت بهینه بین Unit ها منتقل شوند.

* تا اینجا با ساختار کلی یک NOC آشنا شدیم. حال سوالی که ممکن است پیش بیاید این است که عملیات routing بین Unit ها به چه صورت اتفاق می‌افتد؟

Subject.
Date.

در اصل، هر Unit شامل یک router است که وظیفه انتقال Packet ها از current unit به next unit را (در مسیر src به dest) بر عهده دارد.



در این قسمت پردازش های مورد نیاز را روی Packet انجام می‌دهد و سپس یک Packet جدید می‌سازد و آن را به NOC تزریق می‌کند (برای انتقال به واحد های دیگر)

به Packet های که بین router ها، در بستر NOC منتقل می‌شوند اصطلاحاً Flit (= Flow Control digit) گویند. در بستر NOC، انواع مختلفی از Flit ها می‌توانند وجود داشته باشند، مثل:

Header Flit, Tail Flit, Payload Flit, Credit Flit, Routing Flit

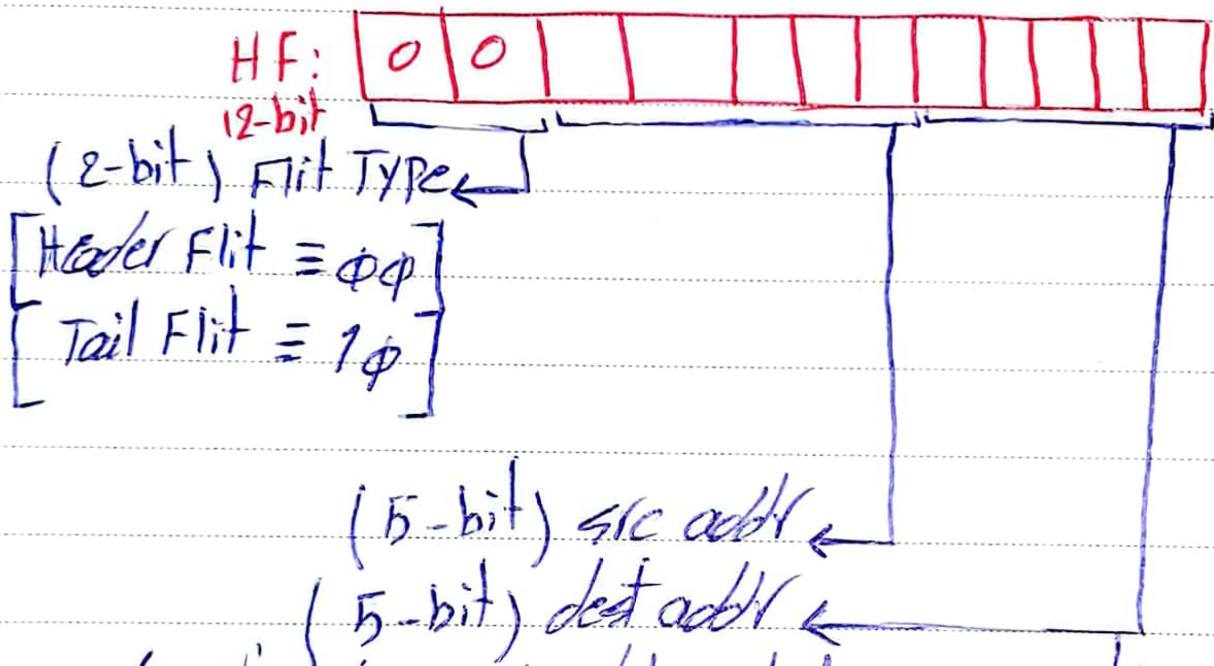
Virtual channel Flit, Error Flit

۲ نوع خاص از Flit ها را اصطلاحاً { Header Flit + و Tail Flit + } گویند.

Subject.
Date.

Header Flit → The Header Flit is the initial flit of a packet and contains control information such as

+ src & dest addresses,
+ routing information,
+ packet length
+ and etc.



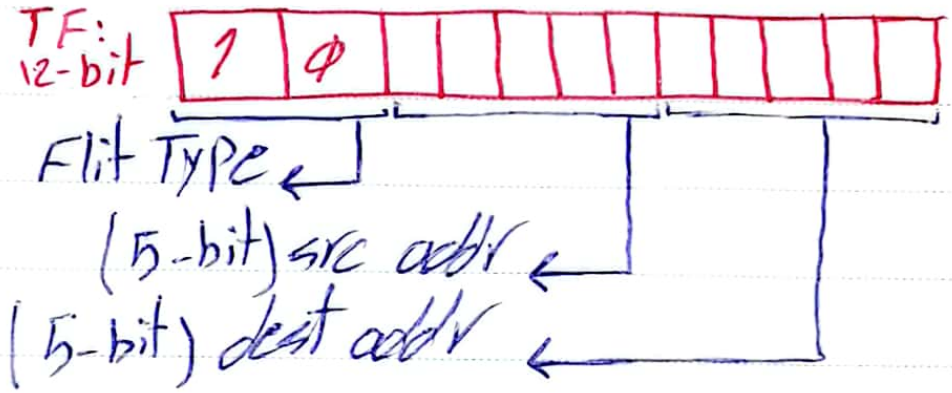
(* ازآنجایی که در این پروژه data packet نداریم در نتیجه نیاز به routing information ذکر شده داخل HF نداریم)

Subject.

Date.

Tail Flit → The Tail Flit is the final flit of a packet and typically includes an end-of-packet marker or other indicators to signal the completion of packet transmission.

* the tail flit informs the receiving router or processing element that the entire packet has been received successfully.



تا اینجا با ساختار کلی یک Unit و مبحث Packet در NOC آشنا شدیم.
حال تصمیم داریم صورت جزئی تر ساختار داخلی Router موجود داخل Unit ها مورد بررسی قرار دهیم.

Subject.
Date.

همانطور که می دانیم، از مبدأ 5 به مقصد 7 چندین مسیر وجود دارد:

$5 \rightarrow 6 \rightarrow 7$

$5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 7$

⋮

حال سوال این است که router 5 از کجا بفهمد به این پکت باید روی کدام خروجی خود قرار دهد.

برای مدیریت این مسئله به سه المان دیگر داخل router نیاز داریم:

1) routing unit ← این المان آدرس src, dest یا از Arbiter می گیرد و شماره پورت (PortA) ای که پکت باید روی آن قرار گیرد را به Arbiter برمی گرداند.

2) Arbiter ← بر اساس PortA و اینکه Full یا PortA مناظر با input buffer آن است یا نه

تصمیمات کنترلی مورد نیاز برای FIFO buffer، crossbar switch و ... توسط کنترل جاری تولید می شوند.

به منظور خدمات ورودی به خروجی مناسب

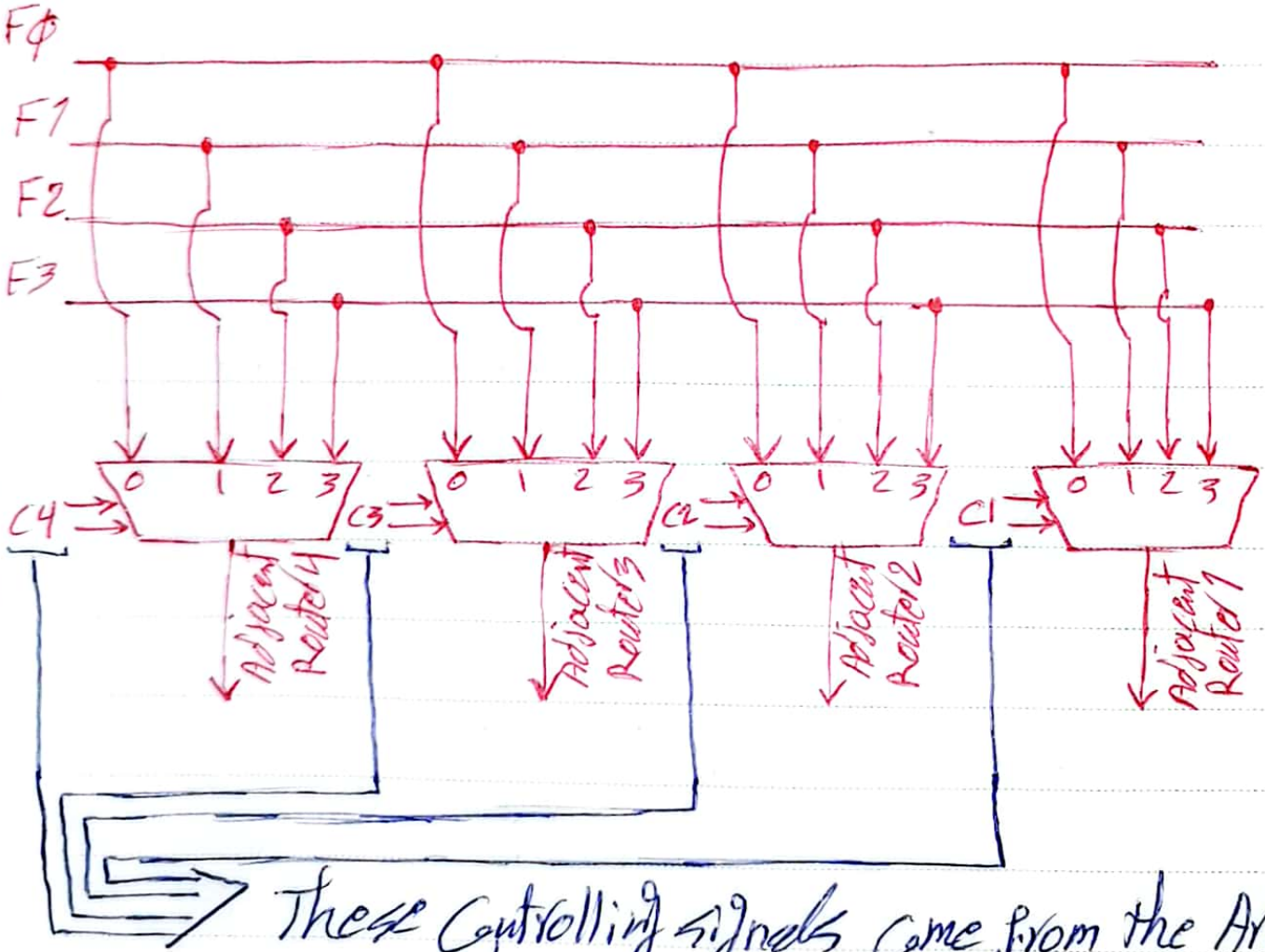
مشخص می کند آیا پکت از بافر POP شود یا نه.
(اگر بافر مقصد Full باشد نباید POP شود.)

Subject.
Date.

تنها Crossbar switch ⇐ عملکرد این کامپوننت به این صورت است که

- چند پورت ورودی } دارد
- و
- چند پورت خروجی }

و به الگوی سیگنال های کنترلی ورودی، هر پورت ورودی را به پورت خروجی مناسب متصل می کند.



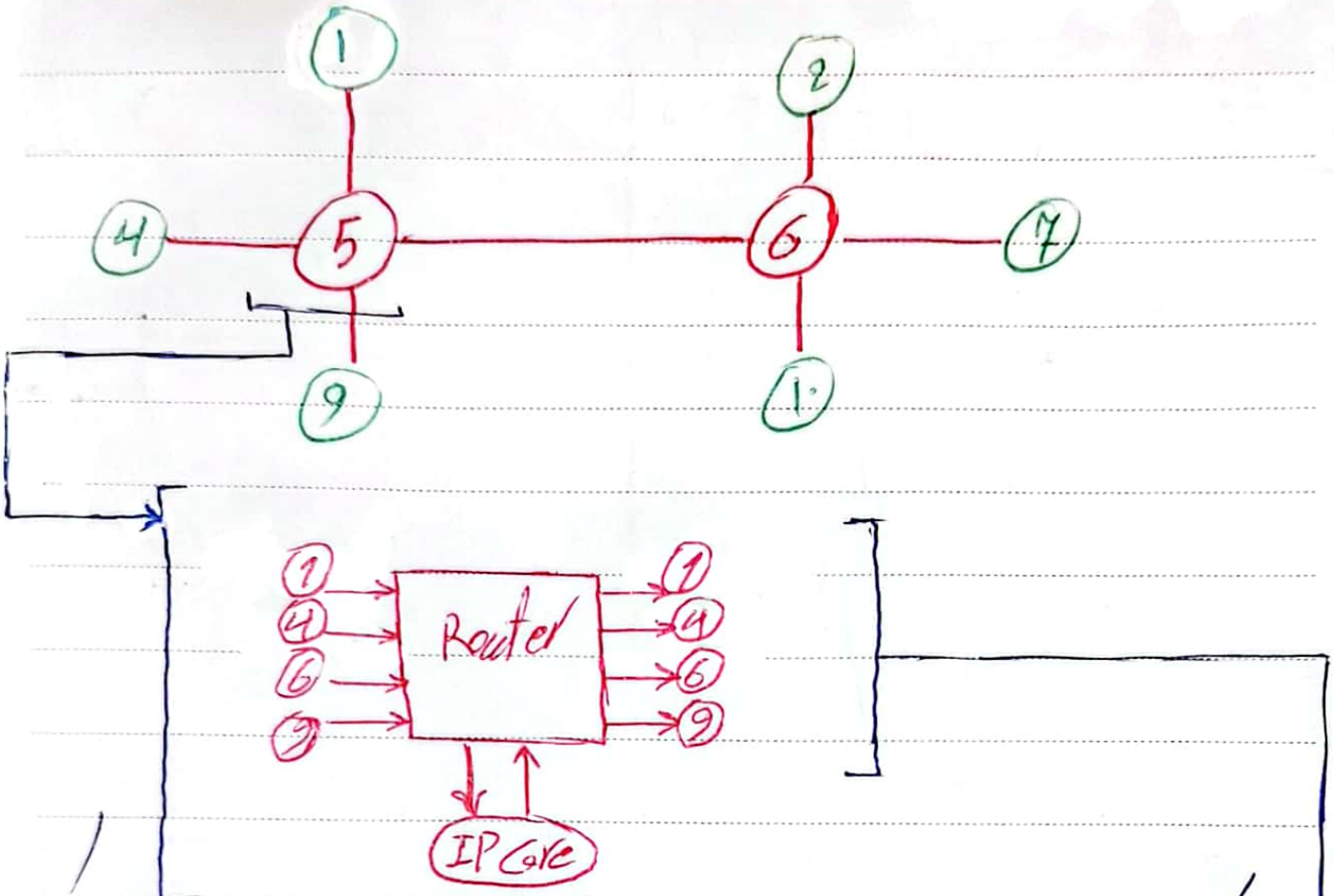
These Controlling signals come from the Arbiter.

C_1 ← مشخص می کند که Packet موجود در ابتدای کدام صف وارد Adjacent Router 1 (①) شود.

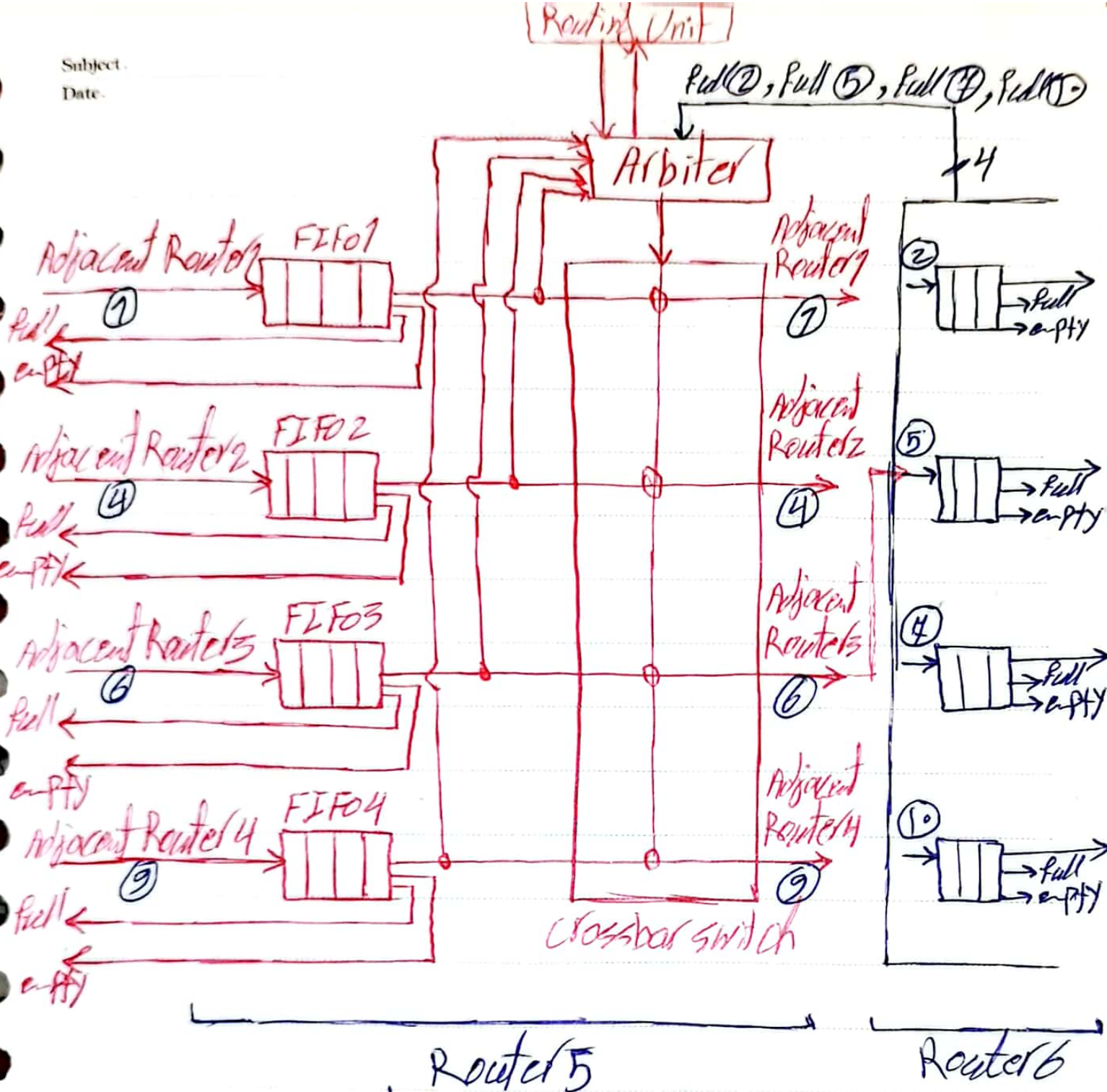
C2 ← مشخص می کند که Packet موجود در ابتدای کدام صف وارد Adjacent Router 2 (②) شود.

Subject.
Date

* بطور خیلی ساده، یک Router می‌تواند ساختاری مشابه زیر داشته باشد:



این بلوک جزئی از یک Router یا Unit از NOC یا بردی است
با ساختاری مشابه زیر در نظر بگیرید:

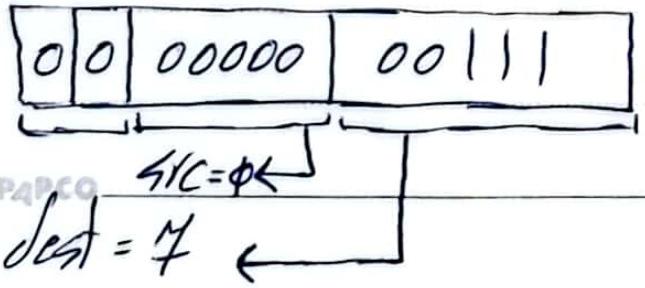


طبق این ساختار، ورودی هر router به یکی از FIFO buffer ها متصل می‌شود.

به پکت های ورودی داخل آن ترازی گیرند.

(≡ پکت های فرستی از Unit مجاور)

حال فرض کنید Packet زیر در ابتدای FIFO1 قرار دارد.



از Unit 1 خارج شده و
وارد Unit 5 خواهد شد

Subject.
Date.

نکات مربوط به شبیه سازی با ابزار Quartus II ⇐

Device : cyclone IV E_EP4CE6E22A7

کامپوننت MUX 4:1 ⇐

RTL simulation → { Analysis & Elaboration + RTL simulation } فقط شامل

است.

در این شبیه سازی هیچ تأخیری وجود ندارد. ولی ورودی testbench باید وجود داشته باشد.

Gatelevel simulation (synthesis) → { Analysis & synthesis + Filter (place and route) + Timing Analysis + EDA Netlist Writer + } شامل

است.

* در این شبیه سازی ورودی testbench شامل تأخیر است. (Timing model: slow - 1.2V ... باید)

$t_{pdr\_out} \simeq 8ns$

$t_{pdf\_out} \simeq 9ns$

Subject.
Date.

~~ ... generic ... ۲۴~~

کامپوننت register file ⇐

* مثلاً به mux در gate level simulation دارای یک تأخیر حدوداً 17ns

و همزمان تغییر خروجی است.

↳ اگر در خروجی شبیه سازی همزمان عملیات read،

دچار مشکل، همزمان تغییر خروجی یکسان glitch روی

بیت های مختلف به وجود می آید تا خروجی به حالت پایدار برسد.

(در صورت وجود)

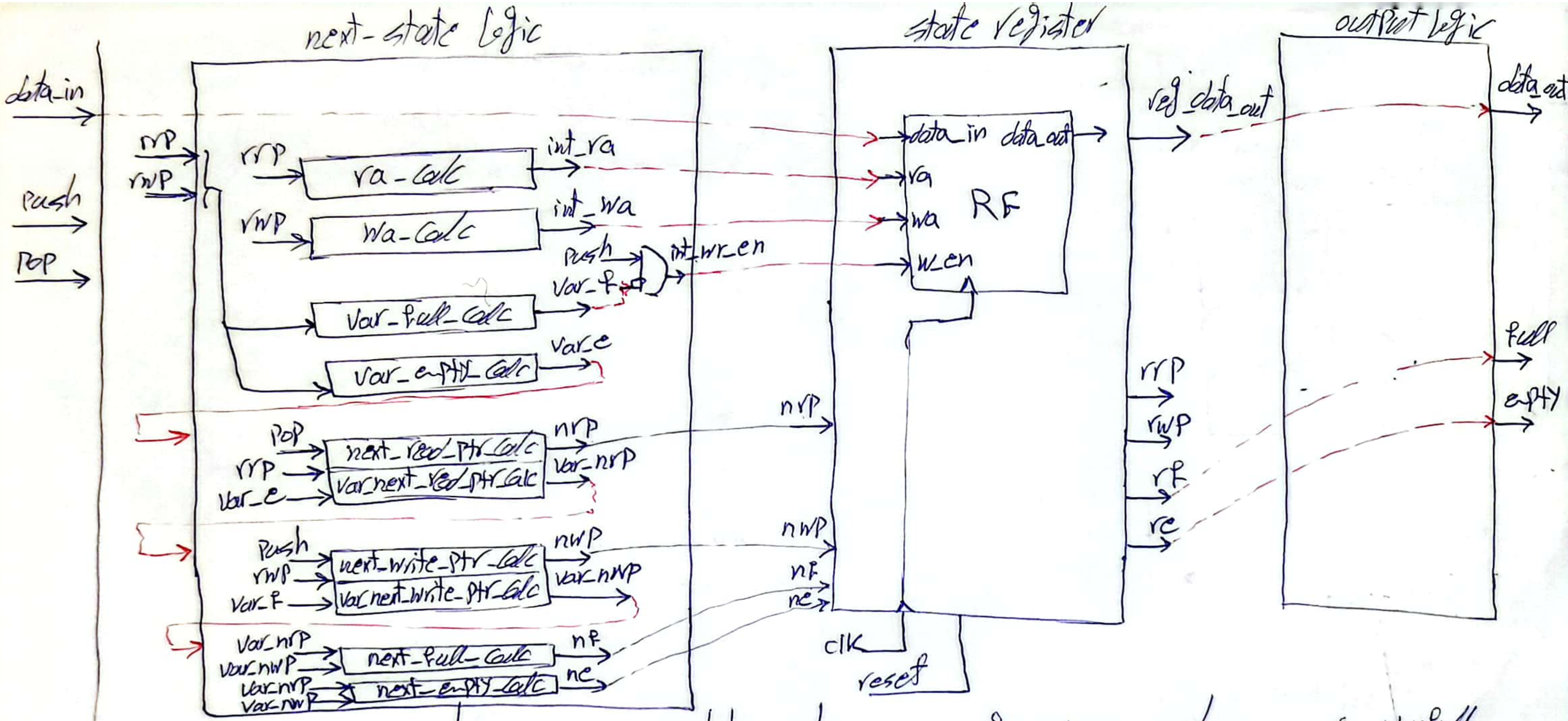
* دچار مشکل در کل به جای مقدار X عموماً 0 در خروجی ظاهر می کند.

* دچار مشکل در زمان نمونه گیری از generic component،

مقادیر موجود در generic ... { component + entity (default value) + }

باید یکسان باشند

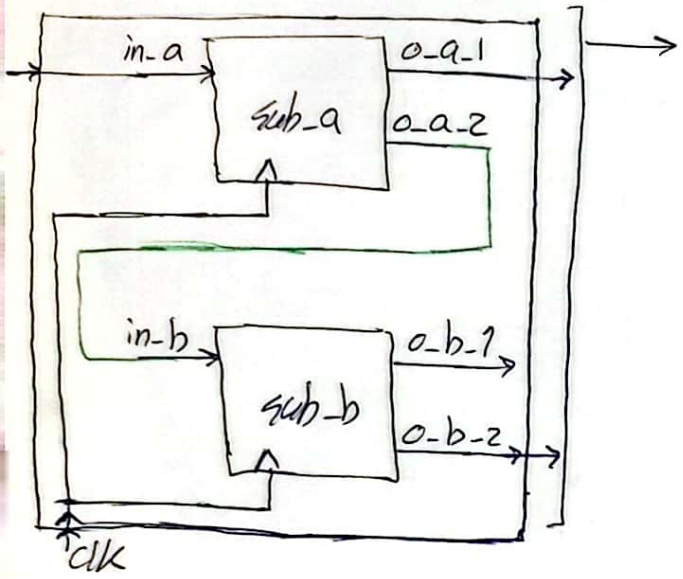
وگرنه EDA tool همزمان شبیه سازی به ما خطا می دهد.



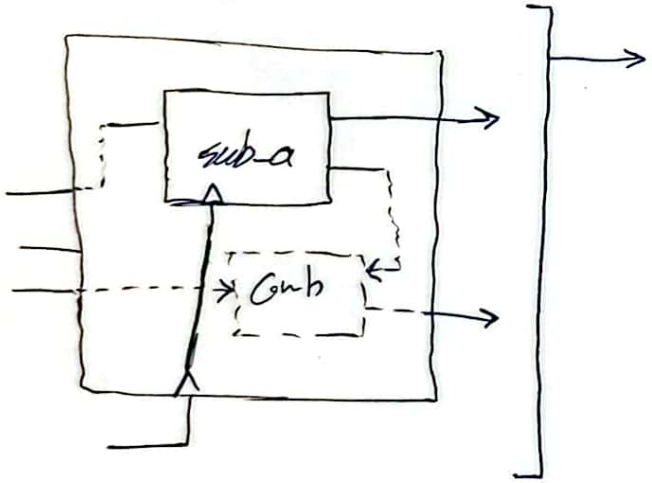
* {var_full و var_empty} براساس سیگنال‌ها {reg_read_ptr و reg_write_ptr} و برای تصمیم گیری در کلاک جاری مورد استفاده قرار می‌گیرند.

اما {next_full و next_empty} براساس {next_read_ptr و next_write_ptr} محاسبه می‌شوند و برای نمایش داخل کلاک بعدی مورد استفاده قرار می‌گیرند که در شبیه‌سازی نیاز است.

با مقداردهی های {var_next_read_ptr := next_read_ptr و var_next_write_ptr := next_write_ptr} (را نیز اینجا) همین تا بتوان در کلاک جاری مقادیر {next_full و next_empty} را براساس {next_read_ptr و next_write_ptr} محاسبه کرد



در حالتی که sub_a و sub_b هر دو یک زیرسیستم مستقل هستند، هرکدام را بصورت جدا با روش two-segment Coding پیاده‌سازی کرده و سپس با Port mapping آنها را با هم متصل می‌کنیم.

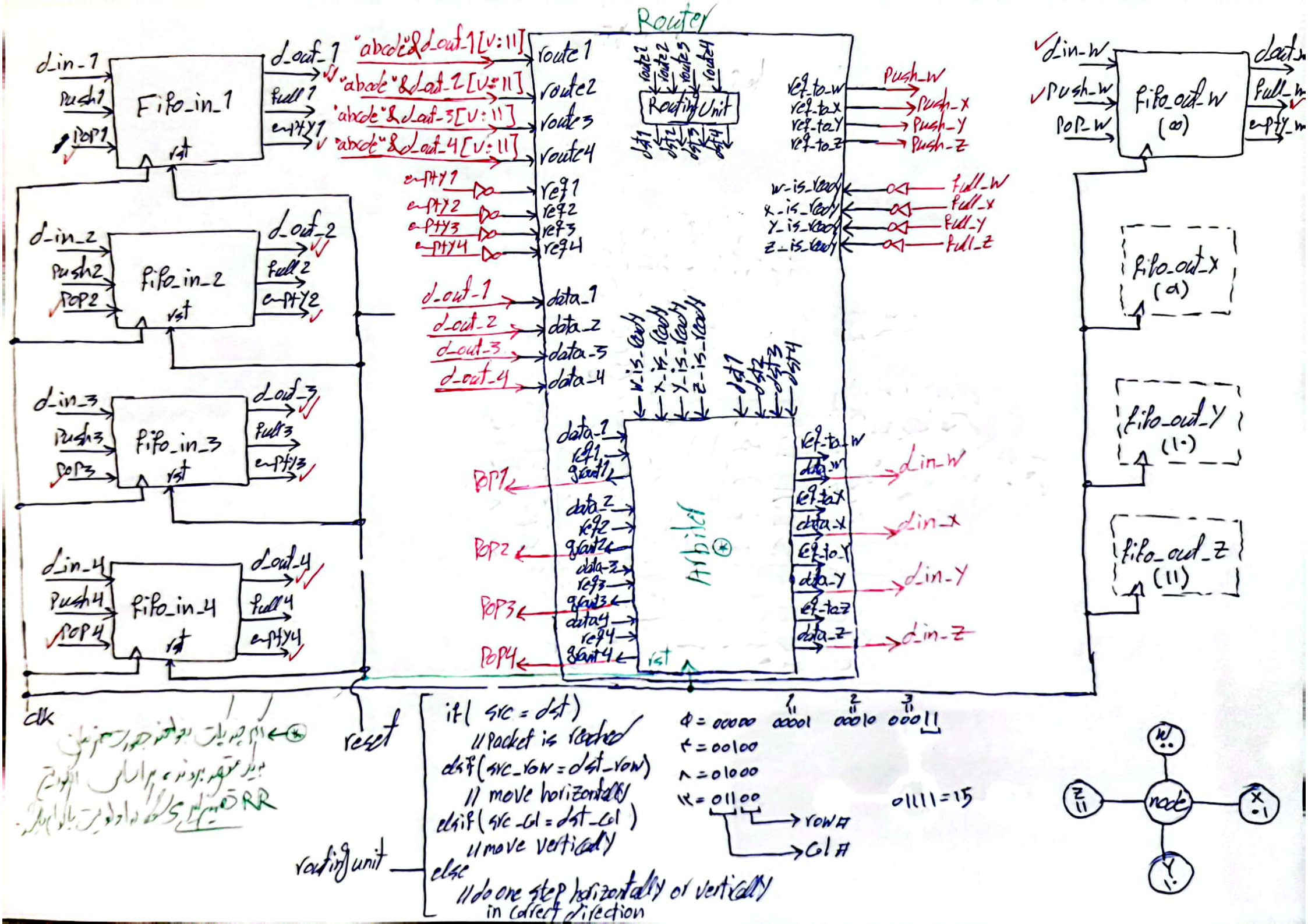


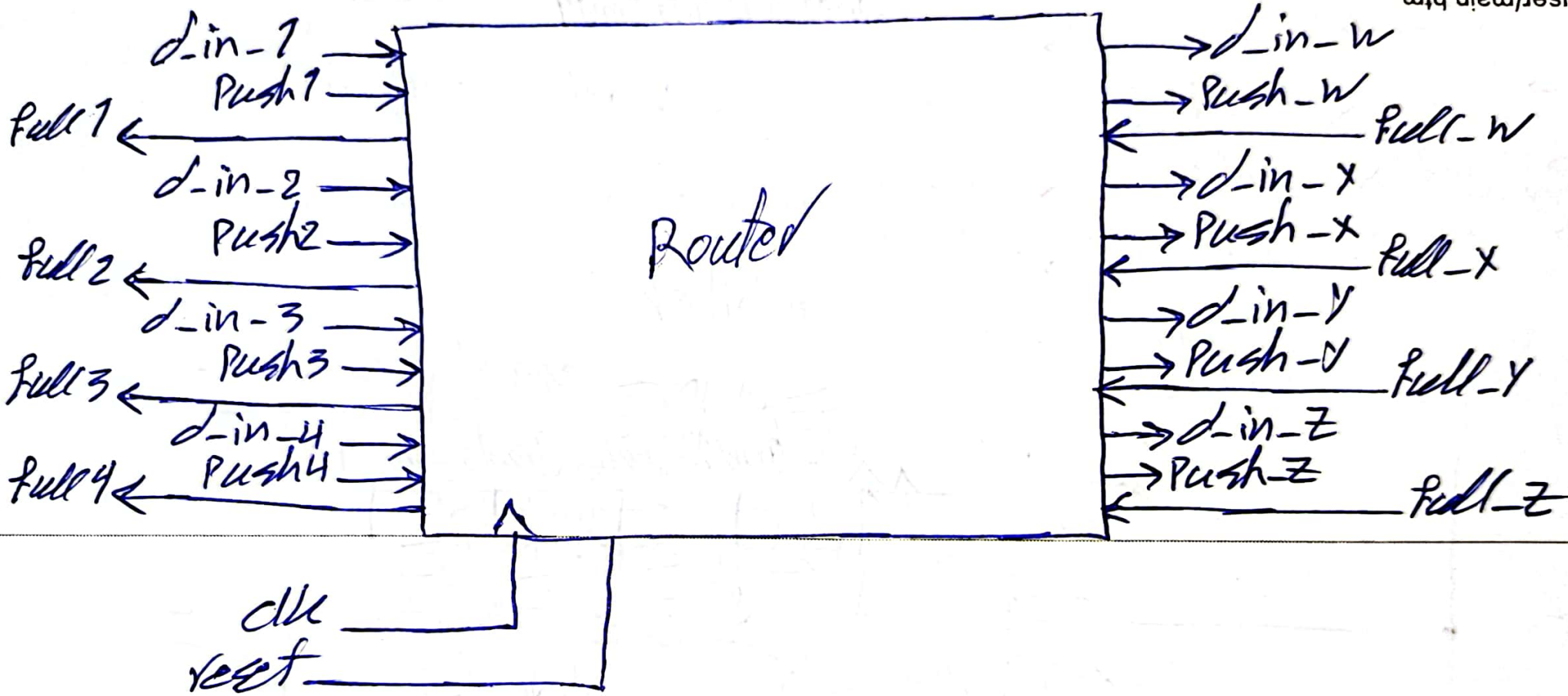
اما زمانی که sub_a جزئی از یک سیستم دیگر باشد، ابتدا sub_a را با روش two-segment Coding پیاده می‌کنیم و سپس دوباره system اصلی را که حالا sub_a یک Component از آن است، با روش two-segment Coding پیاده می‌کنیم.

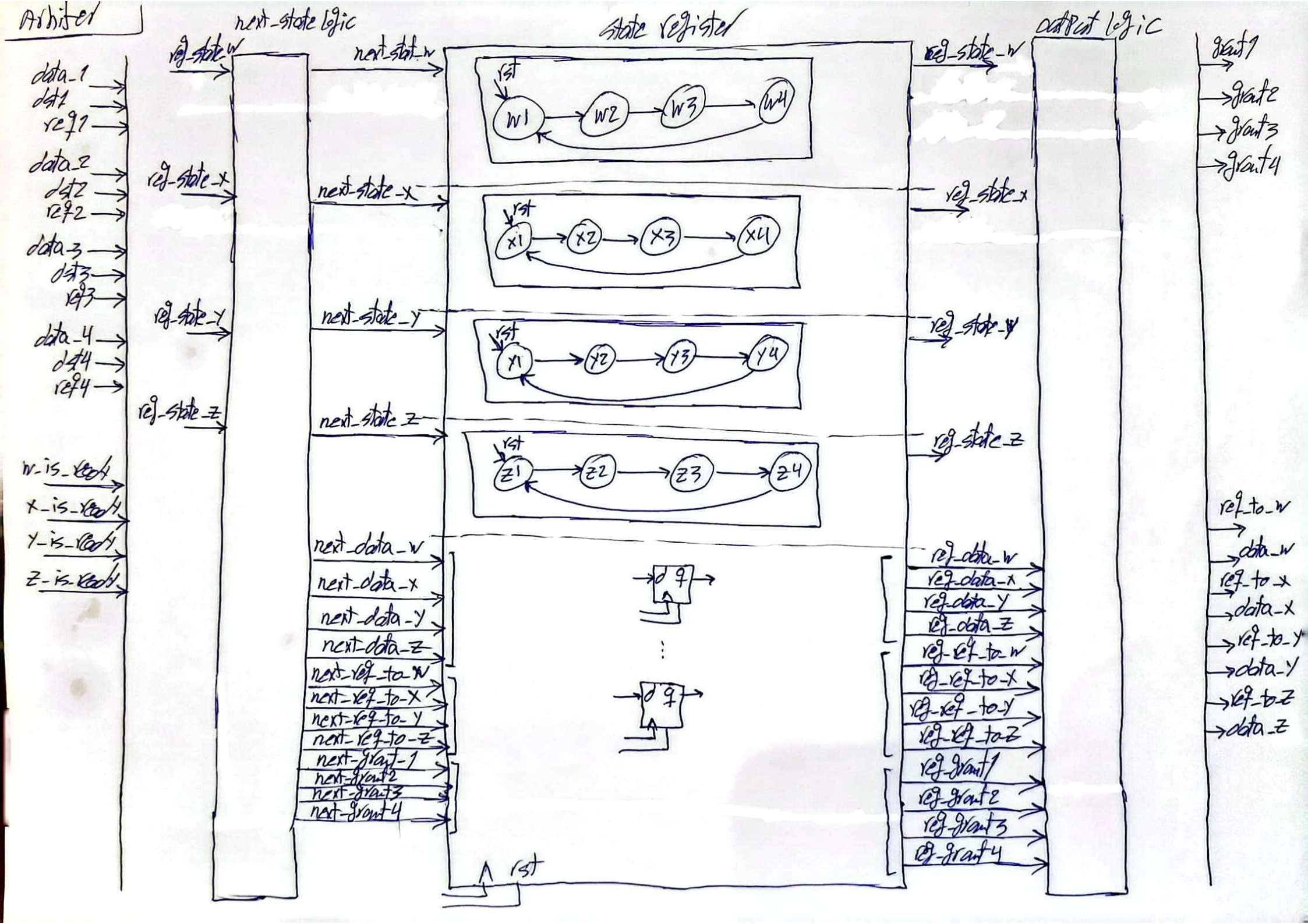
(مثل Fifo_buffer)

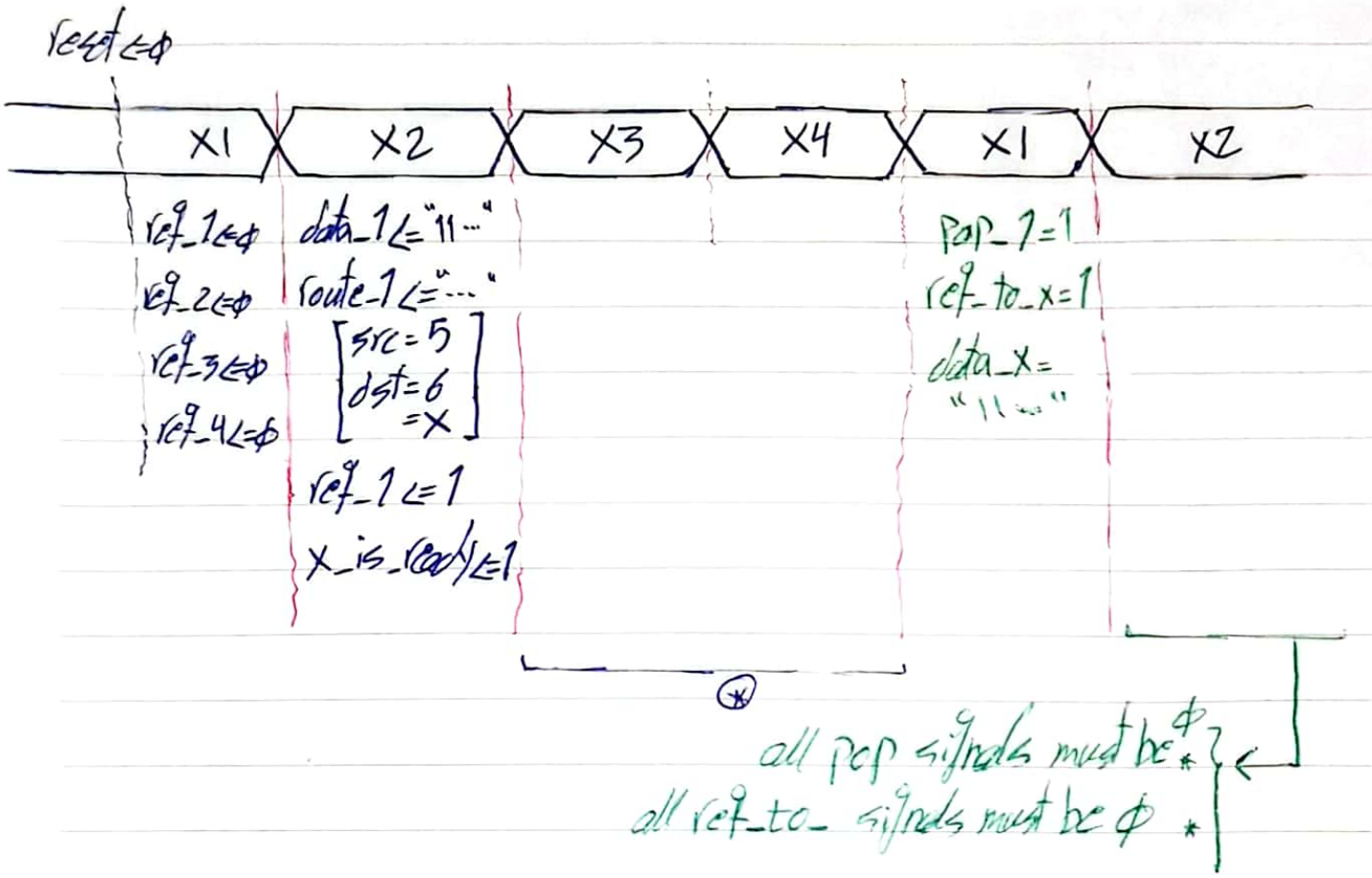
* Variable زمانی به کار می‌آید که خواهیم داخل یک process، بعد از یک Variable assignment از مقادیر آن استفاده کنیم.

(این کار با Signal assignment قابل انجام نیست زیرا مقدار Signal حداقل انتهای process به آنی assign می‌شود)


Router
Fifo_in_1
Fifo_in_2
Fifo_in_3
Fifo_in_4
Routing Unit
Arbiter
Fifo_out_w (∞)
Fifo_out_x (a)
Fifo_out_y (1·)
Fifo_out_z (11)
clk
reset
routing unit
if(src = dst)
// Packet is reached
elsif(src_row = dst_row)
// move horizontally
elsif(src_col = dst_col)
// move vertically
else
// do one step horizontally or vertically in correct direction
Φ = 00000
↑ = 00100
^ = 01000
« = 01100
row #
Col #
00001
00010
00011
01111 = 15
node



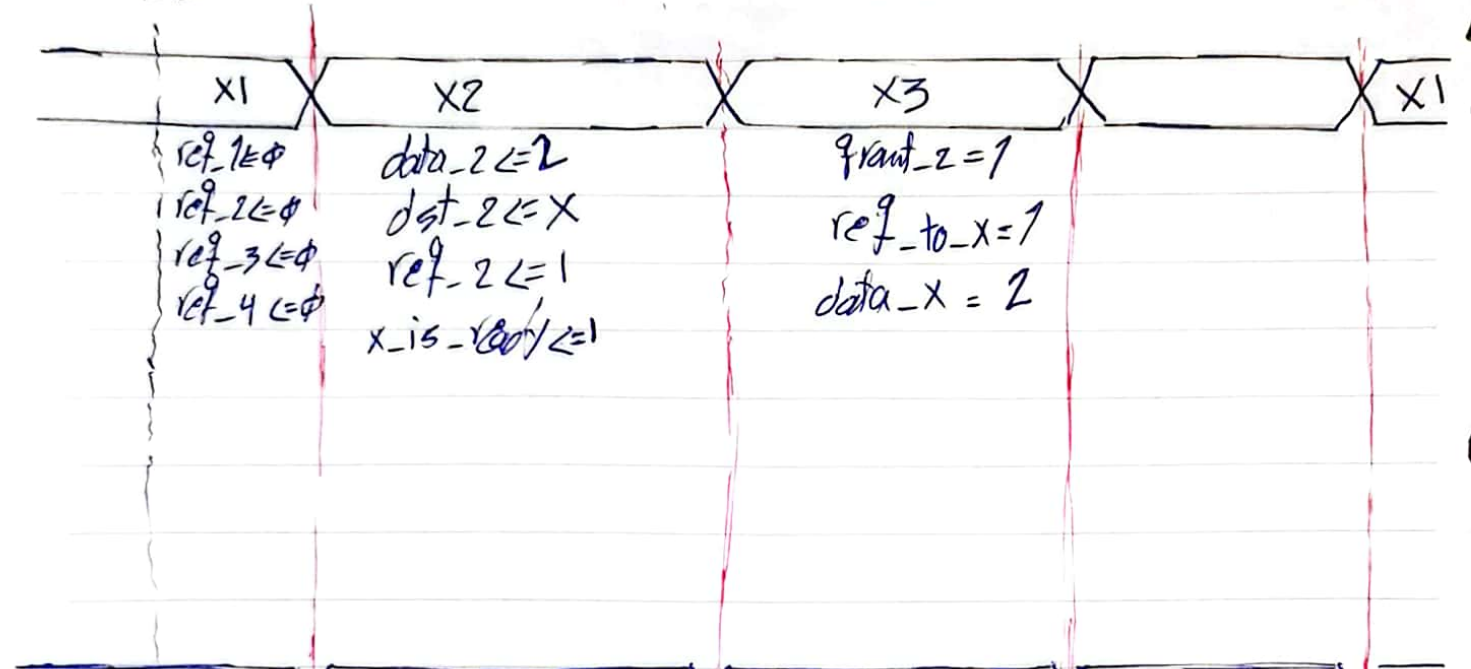

Arbiter
next_state logic
state register
output logic
data_1
dst1
req1
data_2
dst2
req2
data_3
dst3
req3
data_4
dst4
req4
w_is_ready
x_is_ready
y_is_ready
z_is_ready
req_state_w
req_state_x
req_state_y
req_state_z
next_state_w
next_state_x
next_state_y
next_state_z
rst
w1
w2
w3
w4
x1
x2
x3
x4
y1
y2
y3
y4
z1
z2
z3
z4
next_data_w
next_data_x
next_data_y
next_data_z
next_req_to_w
next_req_to_x
next_req_to_y
next_req_to_z
next_grant_1
next_grant2
next_grant3
next_grant4
d
q
req_state_w
req_state_x
req_state_y
req_state_z
req_data_w
req_data_x
req_data_y
req_data_z
req_req_to_w
req_req_to_x
req_req_to_y
req_req_to_z
req_grant1
req_grant2
req_grant3
req_grant4
grant1
grant2
grant3
grant4
req_to_w
data_w
req_to_x
data_x
req_to_y
data_y
req_to_z
data_z

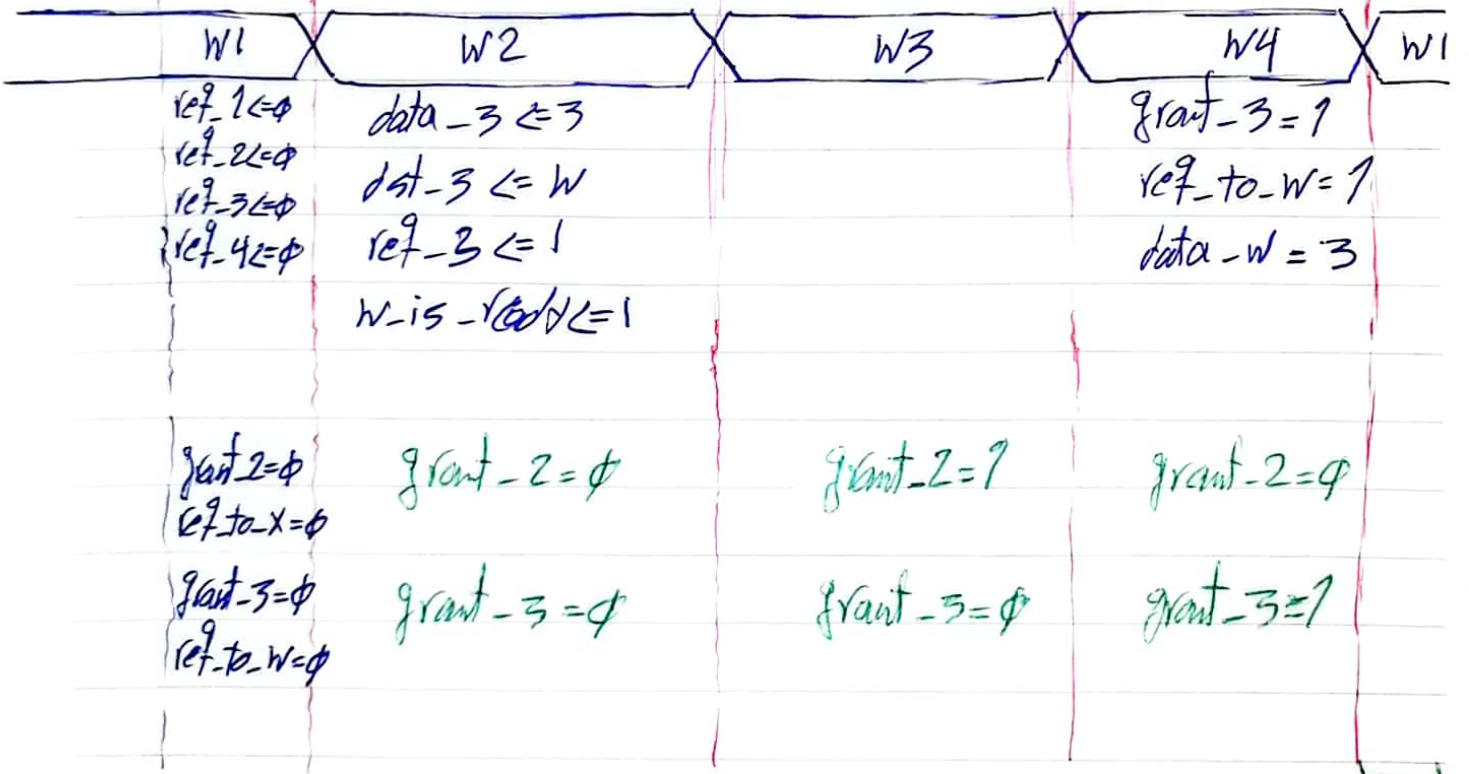


نکته 1 ← فقط چون route_1 از routing_unit پیاده سازی شده است.

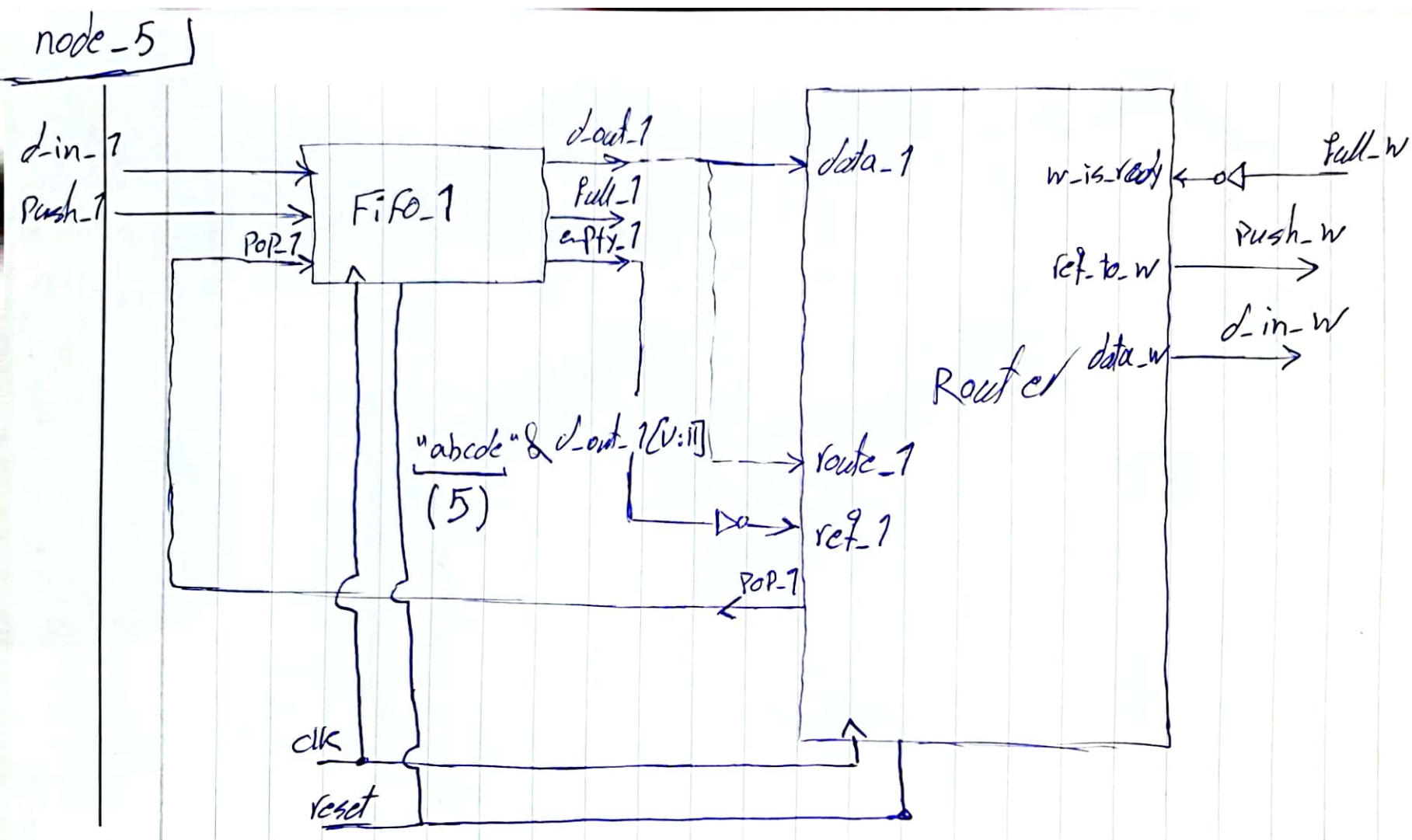
* نکته 2 ← دقت شود اولویت هر کدام (در arbiter) براساس الگوریتم Round Robin است.
یعنی ابتدا { Fifo_1 / Fifo_2 } بخواهند خروج بفرستند، خروج X (خود شماره 6 یا مبدأ 5) اولیت اول را دارد، با فرض اینکه arbiter در state X2 باشد، با این کار دیتای Fifo_2 به مقصد X اول می‌رود.

reset = 0





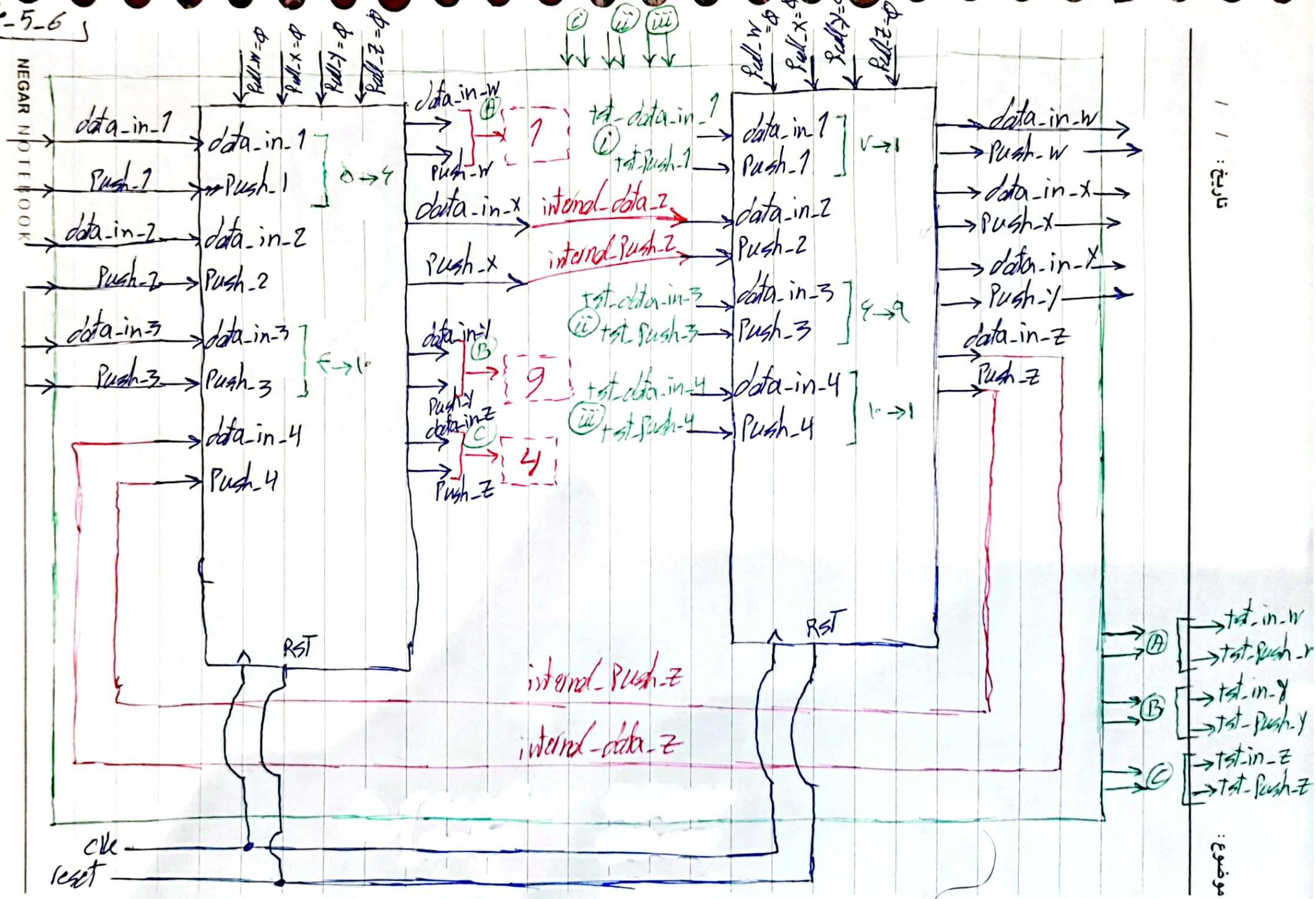
grant_2 = 0
req_to_x = 0
grant_3 = 0
req_to_w = 0



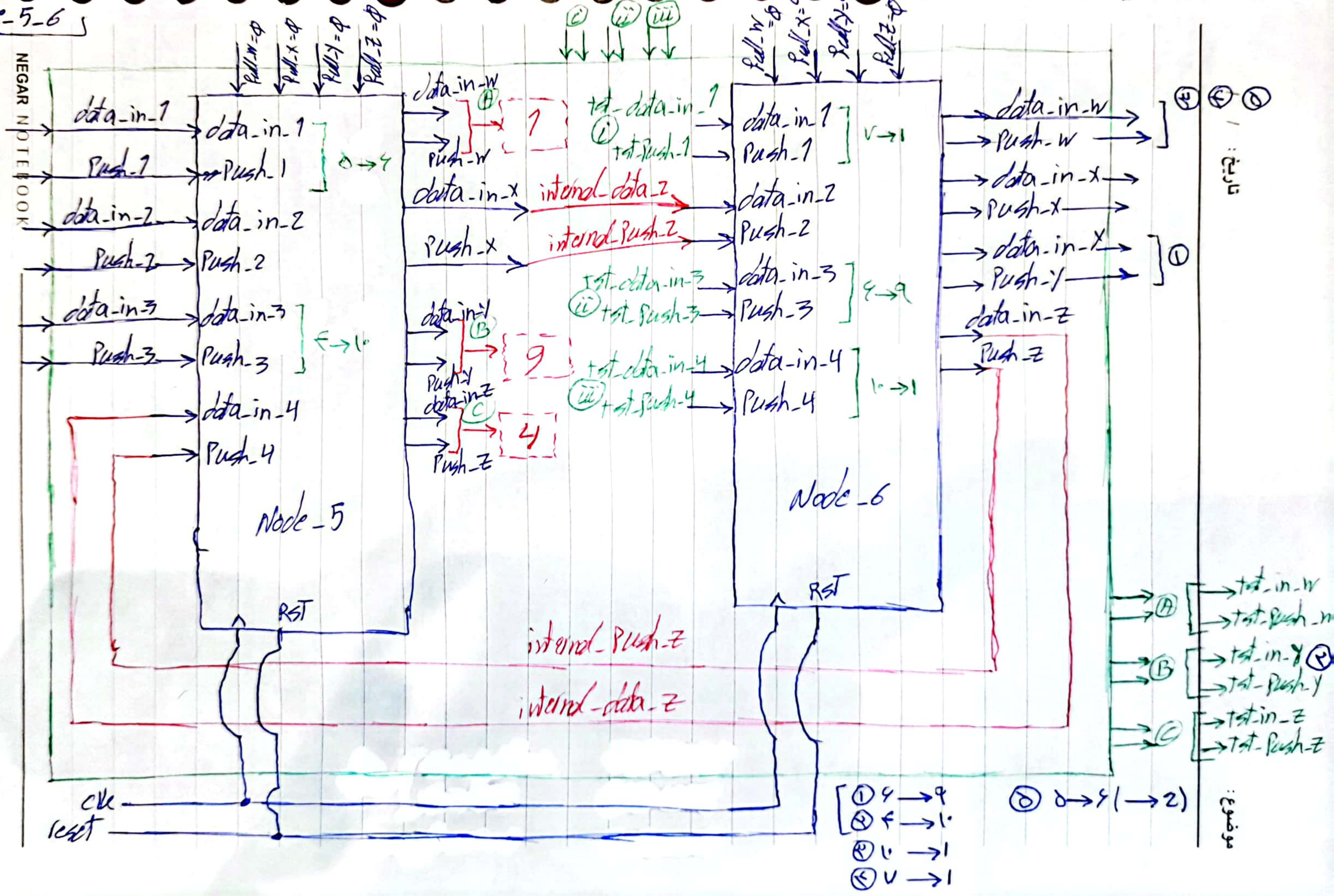
تاریخ : / / /

موضوع :

node_5_6


NEGAR NOTEBOOK
Pull_w=0
Pull_x=0
Pull_y=0
Pull_z=0
data_in_1
Push_1
data_in_2
Push_2
data_in_3
Push_3
data_in_4
Push_4
RST
data_in_w
Push_w
data_in_x
Push_x
data_in_y
Push_y
data_in_z
Push_z
internal_data_z
internal_Push_z
tst_data_in_1
tst_Push_1
tst_data_in_3
tst_Push_3
tst_data_in_4
tst_Push_4
clk
reset
tst_in_w
tst_Push_w
tst_in_y
tst_Push_y
tst_in_z
tst_Push_z
تاریخ:
موضوع:

node_5_6



Node_5
Node_6
data_in_1
Push_1
data_in_2
Push_2
data_in_3
Push_3
data_in_4
Push_4
full_W=Φ
full_X=Φ
full_Y=Φ
full_Z=Φ
data_in_W
Push_W
data_in_X
Push_X
data_in_Y
Push_Y
data_in_Z
Push_Z
internal_data_Z
internal_Push_Z
tst_data_in_1
tst_Push_1
tst_data_in_3
tst_Push_3
tst_data_in_4
tst_Push_4
tst_in_W
tst_Push_W
tst_in_Y
tst_Push_Y
tst_in_Z
tst_Push_Z
RST
clk
reset
NEGAR NOTEBOOK