Universitatea
Transilvania
din Brașov
FACULTATEA DE INGINERIE ELECTRICĂ
ȘI ȘTIINȚA CALCULATOARELOR

Multiplicator de numere complexe

Coordonator:
Prof. dr. ing. Nicula Dan

Student:
Feldioreanu George-Aurelian

BRAȘOV, 2022

1 MULTIPLICATORUL DE NUMERE COMPLEXE (COMP_MULT)

1.1 FUNCȚIONARE

Circuitul realizează înmulțirea a două numere complexe reprezentate sub formă algebrică **(z = x + i·y).** Părțile reale și imaginare ale operanzilor sunt numere întregi reprezentate pe număr parametrizabil de biți, în complement față de 2.

Dacă se notează:

$z1 = x1 + i \cdot y1$
$z2 = x2 + i \cdot y2$

atunci rezultatul este:

$z1 \cdot z2 = r = xr + i \cdot yr$

unde:

$xr = x1 \cdot x2 - y1 \cdot y2$
$yr = x1 \cdot y2 + x2 \cdot y1$

Circuitul este sincron și are:

- un semnal de reset asincron activ în 0
- semnal de reset sincron activ în 1

Interfețele cu operanzii și cu rezultatul sunt de tip "valid-ready".

1.2 SIMBOL

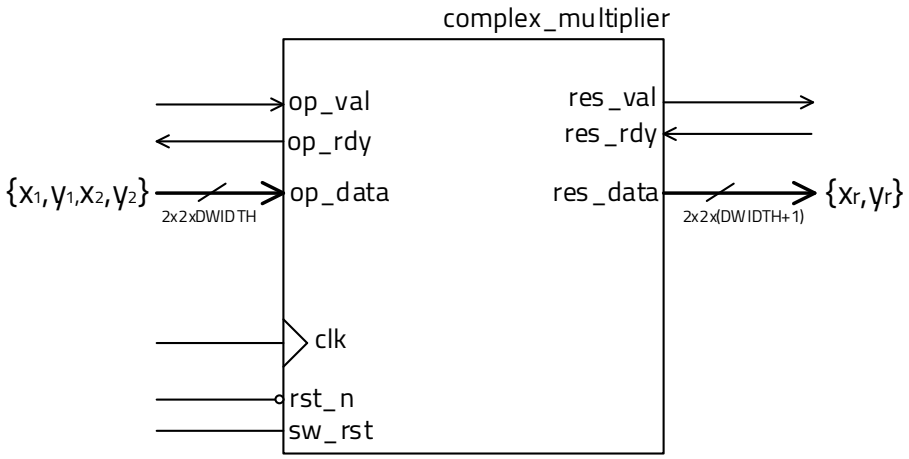

Fig. 1 – Simbolul bloc al multiplicatorului

1.3 PARAMETRI

DWIDTH – Lățimea în biți de reprezentare a componentelor unui număr complex (partea reală, partea imaginară). Determină lățimile valorilor x_1, y_1, x_2, y_2.

NO_MULT – Numărul de multiplicatoare folosit în calcularea rezultatului.

1.4 INTERFEȚE

Interfața	Port	Descriere	Lățime [biți]	Direcție
Sistem	clk	Semnal de ceas	1	I
	rst_n	Reset asincron activ în 0	1	I
	sw_rst	Reset sincron activ în 1	1	I
Operanzi	op_val	Operanzi valizi.	1	I
	op_rdy	Este permisă primirea operanzilor.	1	O
	op_data	Operanzi {x_1, y_1, x_2, y_2}	2 * 2 * DWIDTH	I
Rezultat	res_val	Rezultat valid.	1	O
	res_rdy	Este acceptat rezultatul.	1	I
	res_data	Rezultatul {xr, yr}	2 * (DWIDTH + 1)	O

1.5 ARHITECTURA INTERNĂ

1.5.1 Implementarea cu 4 multiplicatoare (NO_MULT = 4)

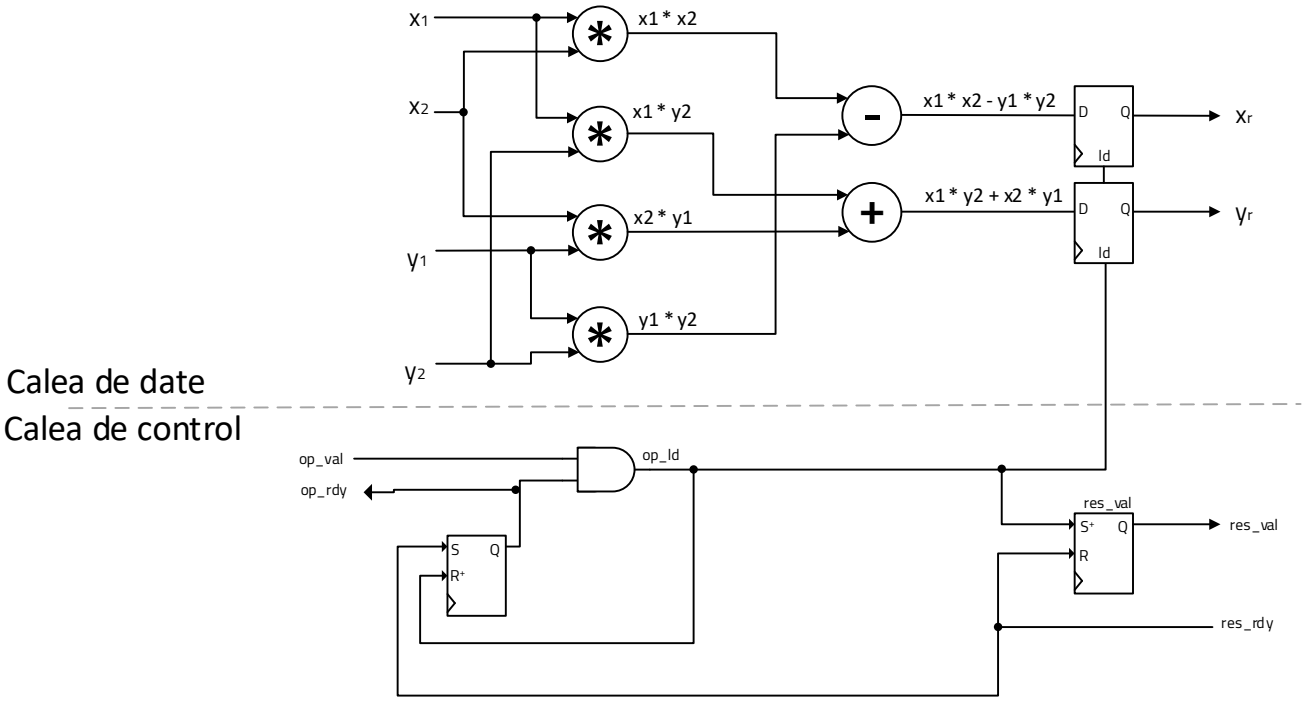

Fig. 2 - Arhitectura multiplicatorului cu 4 multiplicatoare întregi

1.5.2 Implementarea cu 2 multiplicatoare (NO_MULT = 2)

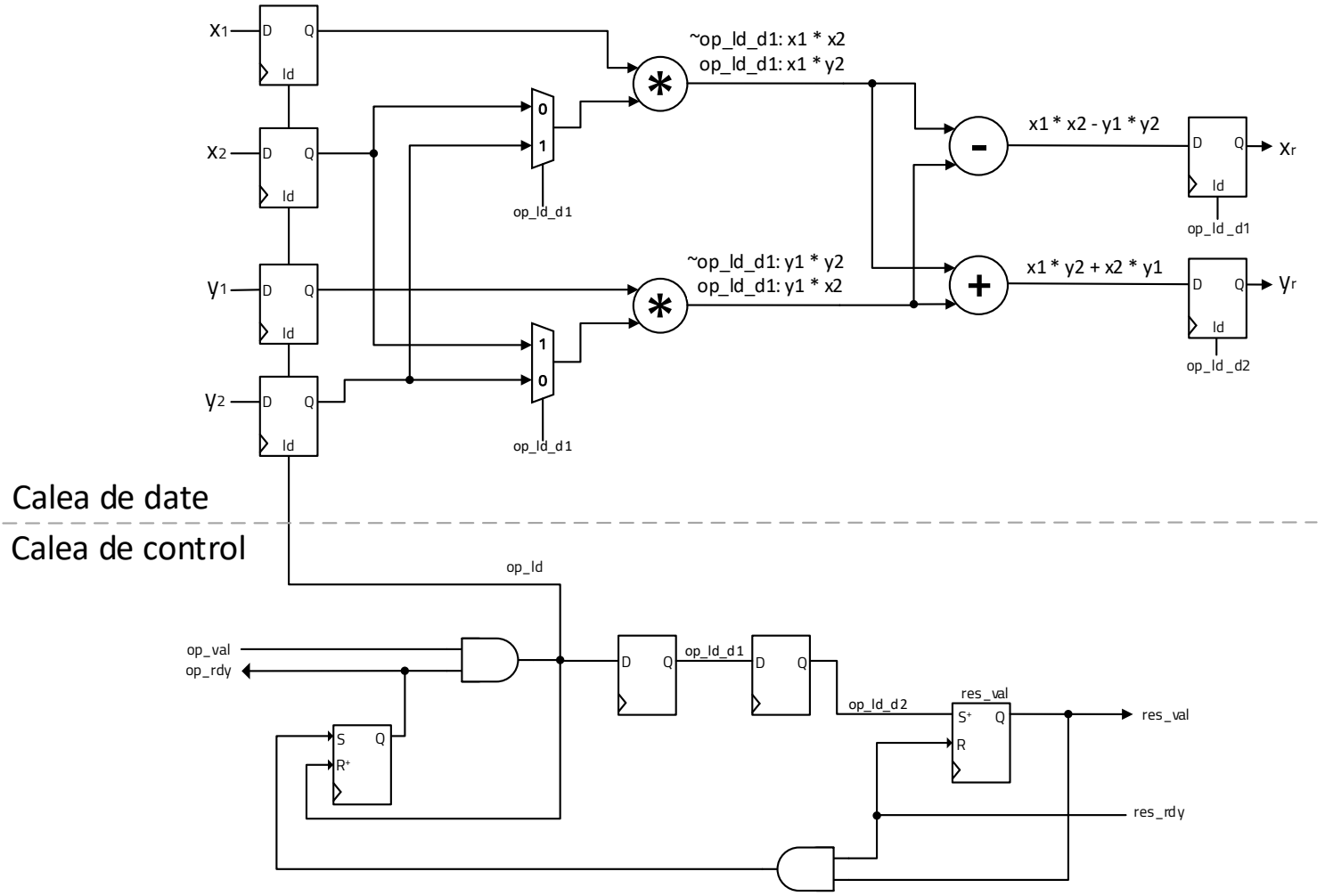

Fig. 3 - Arhitectura internă a multiplicatorului cu 2 multiplicatoare întregi

1.5.3 Implementarea cu 1 multiplicator (NO_MULT = 1)

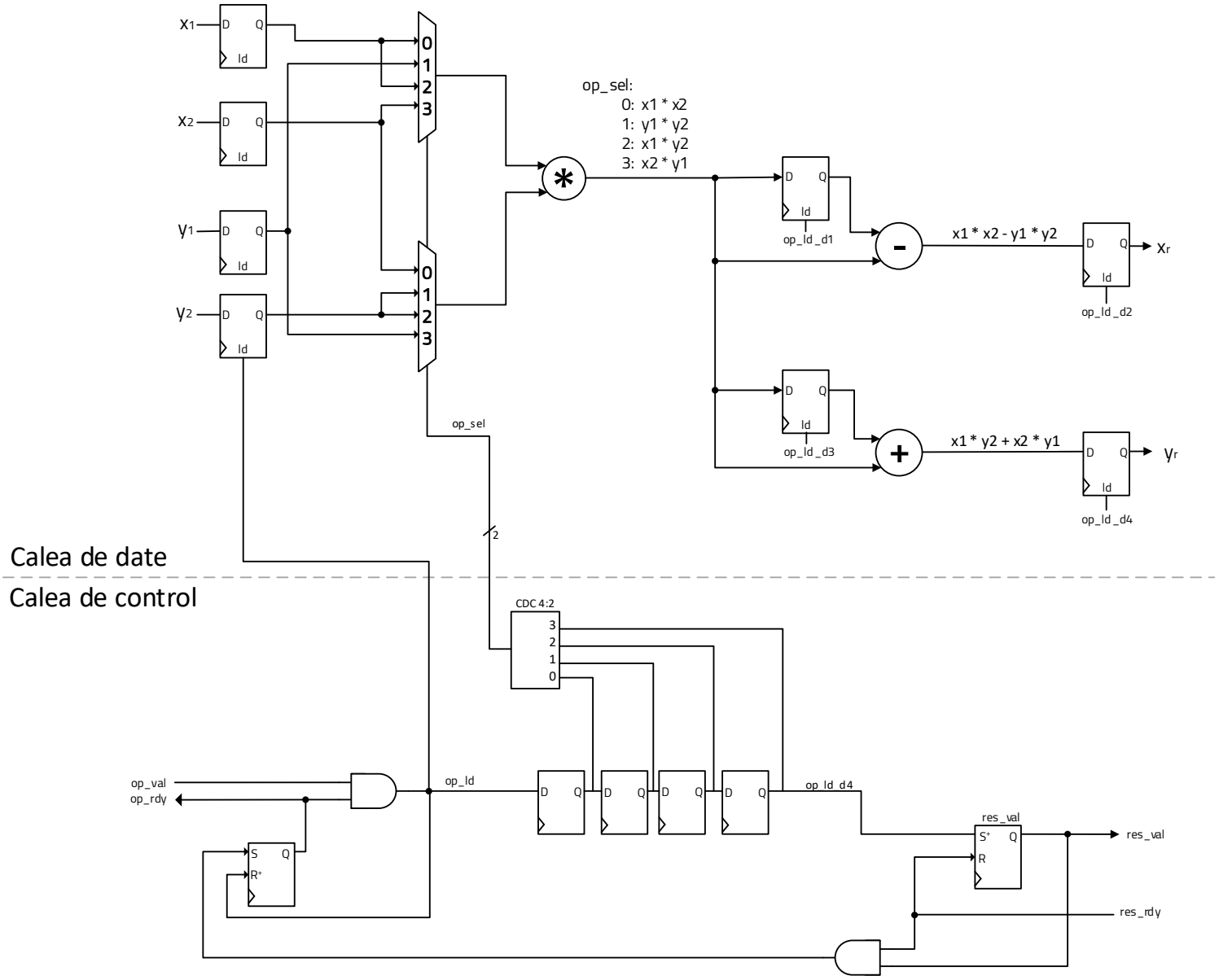

Fig. 4 - Arhitectura internă a multiplicatorului cu 1 multiplicator întreg

1.6 LIVRABILE

Multiplicatorul va fi folosit instațiind fișierul Verilog **`comp_mult_wrapper.v`** și vor fi necesare cele modulele celor 3 implementări, alături de multiplicatoarele de numere naturale și numere întregi:

- `comp_mult_4.v`
- `comp_mult_2.v`
- `comp_mult_1.v`
- `signed_mult.v`
- `unsigned_mult.v`

2 INTEGRAREA ÎNTR-UN SISTEM CU PROCESOR ȘI INTERFAȚAREA CU MEMORIE DUAL-PORT

2.1 FUNCȚIONARE

Sistemul descris în secțiunea 1 este încorporat într-un bloc care realizează înmulțirea a două șiruri de numere complexe stocate într-o memorie dual port organizată pe byte și stocarea rezultatelor în această memorie. Adresele și numărul de operații sunt comandate de un CPU cu ajutorul unor regiștrii de configurare, accesați prin interfața AMBA APB.

2.2 SCHEMA BLOC

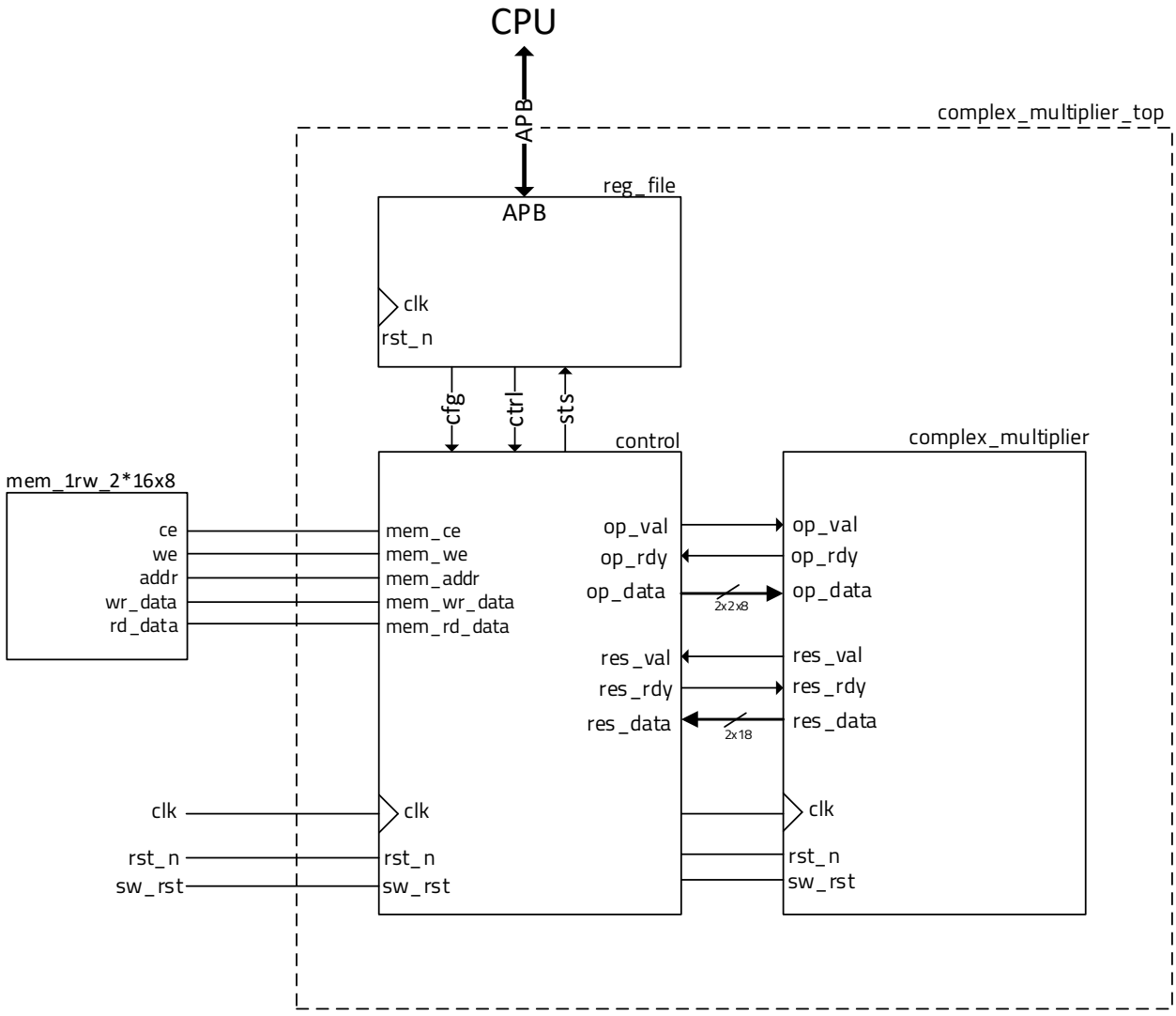

Fig. 5 - Schema bloc a sistemului de interfață cu memorie și CPU

2.3 PARAMETRI

NO_MULT - Numărul de circuite multiplicatoare ai multiplicatorului complex folosit în calcularea rezultatelor.

APB_BADDR – Adresa de bază în spațiul de adresare al sistemului la care se găsesc regiștrii de configurare

SYS_AW - Lățimea în biți a adresei de sistem.

REG_DW - Lățimea în biți a regiștrilor de configurare/stare.

Parametrul DWIDTH al multiplicatorului intern este setat pe valoarea implicită 8 deoarece memoria este organizată pe bytes.

2.4 INTERFEȚE

Interfața	Port	Descriere	Lățime [biți]	Direcție
Sistem	clk	Semnal de ceas	1	I
	rst_n	Reset asincron activ în 0	1	I
	sw_rst	Reset sincron activ în 1	1	I
APB	apb_paddr	Adresa registrului accesat	SYS_AW	I
	apb_psel	Selecția perifericului	1	I
	apb_penable	Semnalizare transfer	1	I
	apb_pwrite	Semnalizare operație de scriere	1	I
	apb_pwdata	Datele scrise	REG_DW	I
	apb_pready	Semnalizare acceptare transfer	1	O
	apb_prdata	Datele citite	REG_DW	O
	apb_pslverr	Semnalizare eroare acces invalid	1	O
Memorie	mem_ce	Selecție memorie	1	O
	mem_we	Semnalizare operație de scriere	1	O
	mem_addr	Adresa accesului	SYS_AW	O
	mem_wr_data	Datele scrise	DWIDTH	O
	mem_rd_data	Datele citite	DWIDTH	I

Offset adresă	Nume	Semnificație	Tip acces	Lățime [biți]
0x0000	op1_addr	Adresa de bază a operanzilor 1	W	REG_DW
0x0001	op2_addr	Adresa de bază a operanzilor 2	W	REG_DW
0x0002	res_addr	Adresa de bază a rezultatelor	W	REG_DW
0x0003	no_op	Numărul de operații efectuate	W	REG_DW
0x0004	cfg_start	Registrul de configurare {soft reset, start}	W, AR	2
0x0005	sts_stop	Registrul de semnalizare al finalizării procesării	R, W	1
0x0006	sts_cstate	Registrul de interogare al stării curente	R	2

2.5 Regiștrii APB

W – write
R – read
AR – auto-reset

Toți regiștrii au lățimea parametrizabilă cu ajutorul parametrului REG_DW. În tabel sunt listate lățimile funcționale. Accesul acestora este realizat conform specificației oficiale AMBA APB.

Utilizare:

- Regiștrii **op1_addr, op2_addr, no_op** trebuie setați conform valorilor cunoscute de utilizator. Registrul **res_addr** trebuie configurat de utilizator a.î.:

```
max(res_addr + 6 · no_op, op1_addr + 2 · no_op) − min(res_addr, op1_addr) > 0 &
max(res_addr + 6 · no_op, op2_addr + 2 · no_op) − min(res_addr, op2_addr) > 0 &
max(op2_addr + 2 · no_op, op2_addr + 2 · no_op) − min(op1_addr, op2_addr) > 0
```

- După ce sunt setate cele 4 valori, pentru activarea procesării trebuie setat bitul de indice 0 al registrului de configurare astfel încât: `cfg_start = 'hxx..x01`. Valoarea va fi resetată când ciclul a început.
- Finalizarea procesării întregului tabel de valori este semnalizată cu ajutorul registrului de stare dacă `sts_stop = 'hxx..xx1.` Aceasta trebuie resetată prin acces de scriere la registrul respectiv.
- Pt. o funcționare corectă, nu este permisă modificarea valorilor regiștrilor **op1_addr, op2_addr, res_addr** și **no_op** dacă ciclul nu a fost finalizat.
- Funcționalitatea de software reset sincron, activ 1 (**sw_rst**) a multiplicatorului de numere complexe poate fi activată prin setarea bitului de indice 1 al registrului de

configurare cu `cfg_start = 'hxx..x1x`; Ciclul de procesare nu poate începe decât dacă acest bit este 0.

- Se poate interoga starea de control internă cu un acces de citire la registrul **sts_cstate**

2.6 FUNCȚIONARE

Valorile operanzilor {**x, y**} pe 2 bytes trebuie organizate în memorie după principiul Little Endian, a.î. dacă un operand se află la adresa A:

```
A + 0   <- y
A + 1   <- x
```

Valorile rezultatului vor avea cele două rezultate **xr, yr** pe 18 biți fiecare. Ambele sunt umplute cu valori de 0 a.î. vor avea lățime de 24 biți. Rezultatele vor fi scrise pe 6 bytes în memorie la adresa A drept { **xr**_extins**, yr_**extins } organizate tot Little Endian:

```
A + 0   <- yr[7:0]
A + 1   <- yr[15:8]
A + 2   <- {6'd0,yr[17:16]}
A + 3   <- xr[7:0]
A + 4   <- xr[15:8]
A + 5   <- {6'd0,xr[17:16]}
```

Sistemul are 4 stări de funcționare:

IDLE – sistemul este inactiv, așteaptă semnalul de start
RD_OPS – citirea operanzilor din memorie
WORK – calcularea unui rezultat
WR_RES – scrierea rezultatului în memorie

Funcționarea acestora este descrisă cu ajutorul următorului graf de tranziții:

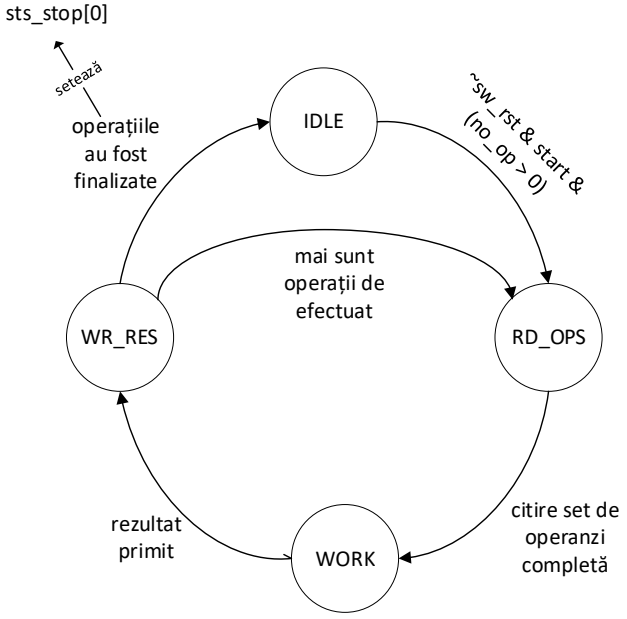

Fig. 6 - Graful de tranziții al căii de control

2.7 LIVRABILE

Este necesar fișierul **`comp_mult_top.v`**, alături de fișierele menționate la secțiunea 1.6.

3 TESTAREA SISTEMULUI

3.1 TESTAREA MULTIPLICATORULUI

Toate variantele ale multiplicatorului de numere complexe sunt testate cu DWIDTH = 8 într-un mediu ce generează vectori de test cu diferite valori ale operanzilor pe interfețele de intrare (op) și compară rezultatele produse cu un model de referință nesintetizabil descris în Verilog, verificând, de asemenea, funcționarea corectă a protocolului valid-ready pe toate interfețele din mediu.

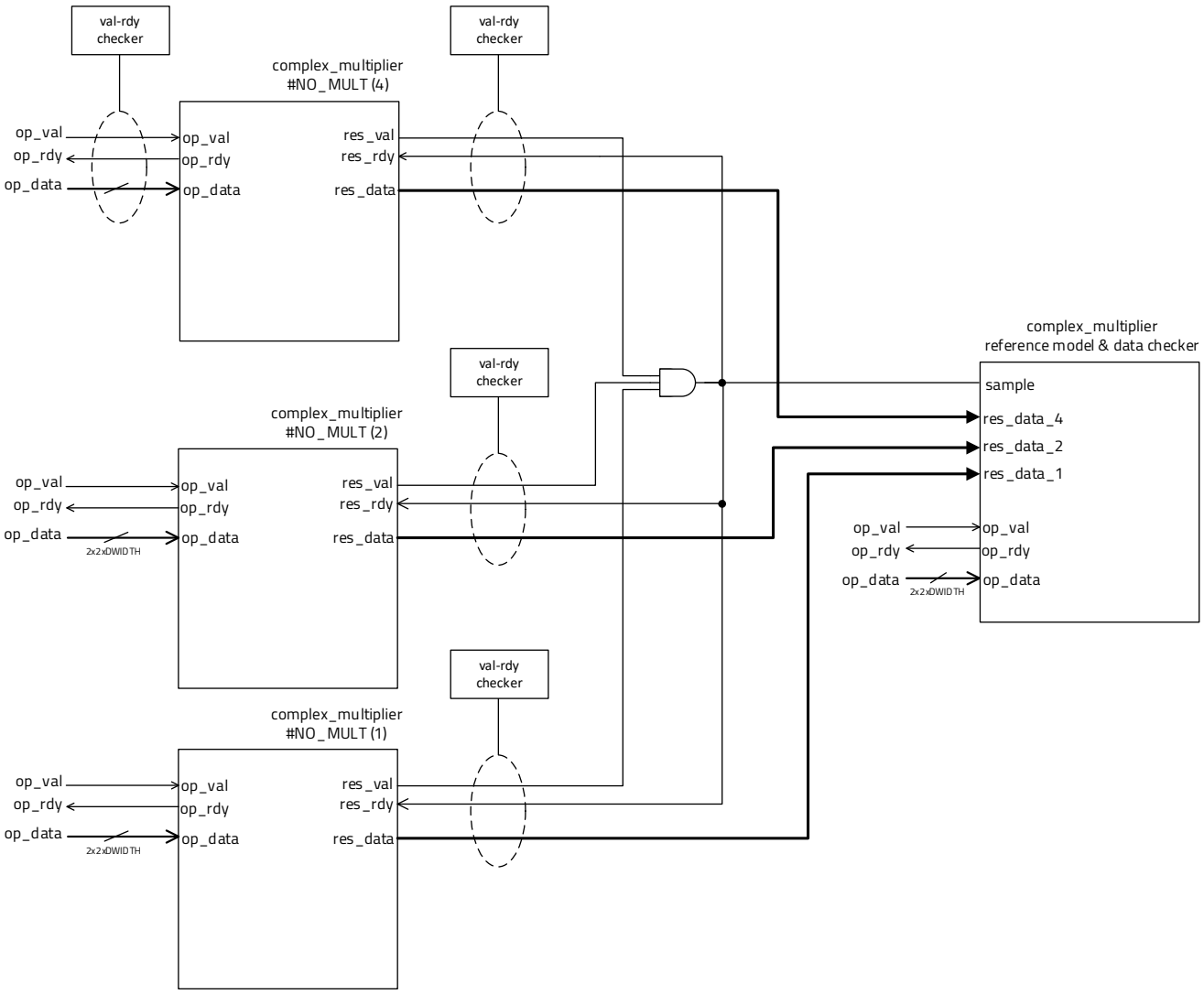

Fig. 7 - Mediul de testare pentru multiplicatorul de numere complexe

Sunt generați operanzi cu valori aleatorii și câteva operații cu valori specifice:

- (2 + i·3) · (4 + i·2)
- (3 + 3i) · (4 + 2i)
- (0 + 0i) · (0 + 0i)
- (−1 + −1i) · (−1 + −1i)
- (127 + 127i) · (127 + 127i)
- (−127 + −127i) · (−127 + −127i)
- (100 + 100i) · (100 + 100i)
- (100 + 101i) · (102 + 103i)

3.2 TESTAREA ÎNTREGULUI SISTEM

Sistemul complet este instanțiat într-un mediu de testare alături de un modul de memorie single port (1RW) care este inițializată cu valori aleatoare de operanzi generate de un script Python. De asemenea, script-ul Python calculează tabela rezultate și sunt scrise la o altă adresă din memorie pentru a fi comparate cu tabela de rezultate obținută de DUT.

Configurarea APB este realizată cu vectori de test a.î.:

- se testează funcționalitatea **sw_rst**,
- sunt configurate pe rând regiștrii de configurare conform parametrilor stabiliți în script-ul Python,
- este citit registrul de status până la activarea bitului de stop,
- se interoghează periodic starea internă.
- La final este dezactivat indicatorul de stop și este verificată și funcționarea semnalului **apb_pslverr** prin accesul la o adresă invalidă

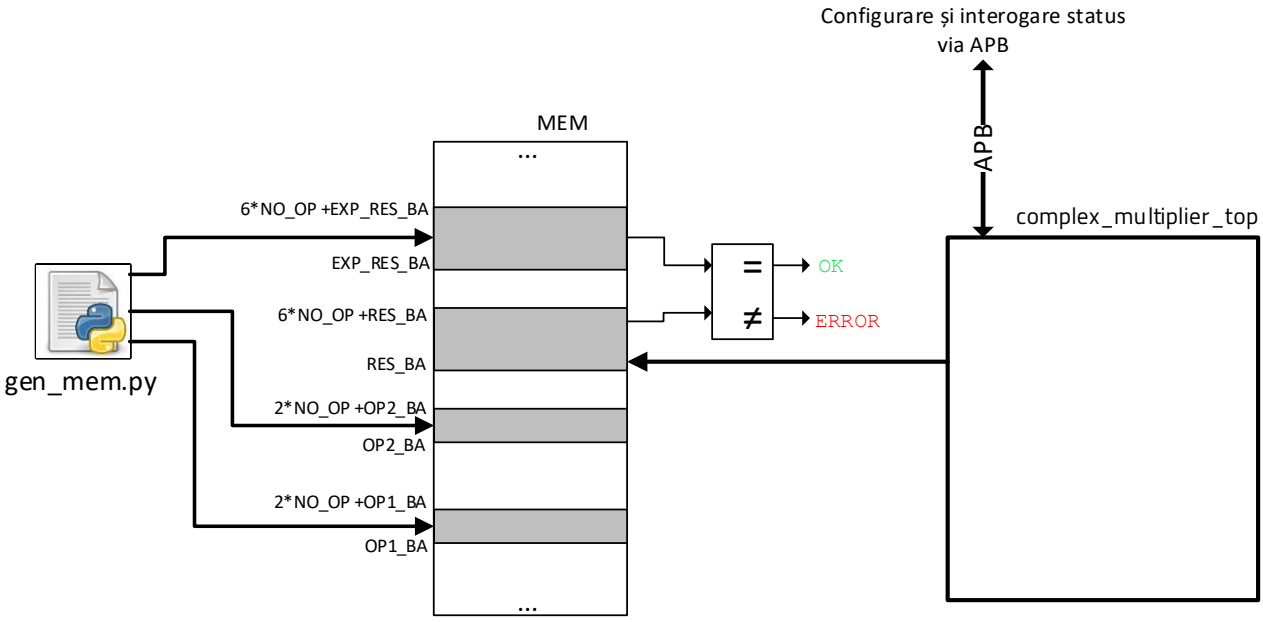

Fig. 8 - Testarea sistemului complet