

**Multiplicator de numere complexe**

**Coordonator:**

**Prof. dr. ing. Nicula Dan**

**Student:**

**Feldioreanu George-Aurelian**

**BRAȘOV, 2022**

# Multiplicatorul de numere complexe (comp\_mult)

## 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 · y1

z2= x2 + i · y2

atunci rezultatul este:

z1 · z2 = r = xr + i·yr

unde:

xr = x1 · x2 - y1 · y2

yr = x1 · y2 + x2 · 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”.

## Simbol



Fig. 1 – Simbolul bloc al multiplicatorului

## Parametri

DWIDTH – Lățimea în biți de reprezentare a componentelor unui număr complex (partea reală, partea imaginară). Determină lățimile valorilor x1, y1, x2, y2.

NO\_MULT – Numărul de multiplicatoare folosit în calcularea rezultatului.

## 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 {x1, y1, x2, y2} | 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 |

## Arhitectura internă

### Implementarea cu 4 multiplicatoare (NO\_MULT = 4)



Fig. 2 - Arhitectura multiplicatorului cu 4 multiplicatoare întregi

### Implementarea cu 2 multiplicatoare (NO\_MULT = 2)



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

### Implementarea cu 1 multiplicator (NO\_MULT = 1)



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

## 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

# Integrarea într-un sistem cu procesor și interfațarea cu memorie dual-port

## 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.

## Schema bloc



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

## 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.

## 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 |

## Regiștrii APB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

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](https://developer.arm.com/documentation/ihi0024/c).

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**

## 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

## Livrabile

Este necesar fișierul comp\_mult\_top.v, alături de fișierele menționate la secțiunea 1.6.

# Testarea sistemului

## 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)

## 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\_rs**t,
* 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