Universitatea Tehnică din Cluj-Napoca

Unitatea Aritmetică de Tip MMX

Structura Sistemelor de Calcul

Student:
Pântea Marius Nicușor

CUPRINS

1.INTRODUCERE

Context

Una dintre tehnologiile inovative introduse de Intel în anii 1990 este arhitectura MMX (MultiMedia eXtensions). Aceasta reprezintă un set de instrucțiuni SIMD (Single Instruction, Multiple Data), care permite procesarea simultană a mai multor date cu o singură instrucțiune, îmbunătățind astfel viteza de execuție a operațiilor aritmetice și logice. Inițial concepută pentru a îmbunătăți performanța aplicațiilor multimedia, arhitectura MMX și-a dovedit utilitatea și în alte domenii care necesită procesare intensă de date, cum ar fi codarea video, grafica 3D și recunoașterea imaginii.

Obiective

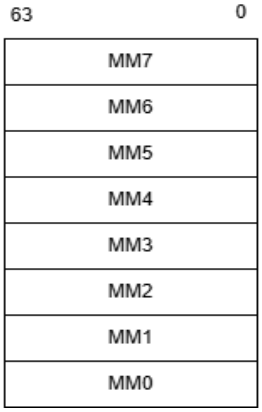
Acest proiect își propune să exploreze și să implementeze o unitate aritmetică de tip MMX, capabilă să execute șase operații aritmetice esențiale. Unitatea va fi proiectată în VHDL și inclusă într-un proiect Xilinx Vivado. Un banc de testare este, de asemenea, furnizat în scopuri de simulare , pentru a demonstra funcționalitatea corectă a unității.

2.STUDIU BIBLIOGRAFIC

Ce este Tehnologia MMX?

Tehnologia Intel MMX™ cuprinde un set de extensii ale arhitecturii Intel (IA) care sunt concepute pentru a spori considerabil performanța aplicațiilor media și de comunicații avansate. Aceste extensii (care includ registre noi, tipuri de date și instrucțiuni) sunt combinat cu un model de execuție cu o singură instrucțiune, date multiple (SIMD) pentru a accelera performanța aplicațiilor precum video în mișcare, grafică combinată cu video, imagine procesare, sinteza audio, sinteza si compresia vorbirii, telefonie, videoconferința, și grafică 2D și 3D, care utilizează de obicei algoritmi de calcul intensiv pentru a efectua operații repetitive pe rețele mari de elemente de date simple, native.

MMX oferă programatorului 8 registre de uz general pe 64 de biți. Aceste registre, numite MM0 - MM7, pot fi utilizate în mai multe moduri. Ele pot fi utilizate ca valori unice pe 64 de biți cantități duble de 32 de biți, 4 cantități de 16 biți sau 8 cantități de 8 biți. Când se întreprinde orice acțiune asupra unui registru MMX, aceasta se aplică tuturor elementelor registrului în același timp. Acest lucru permite software-ului să funcționeze de până la 8 ori mai rapid.



Figură 1

Ce aduce nou MMX?

Tehnologia MMX™ suportă o nouă capacitate aritmetică cunoscută sub numele de aritmetică cu saturație. Saturația este cel mai bine definită prin contrast cu modul de înfășurare (wraparound). În modul de înfășurare, rezultatele care depășesc limitele sau sunt sub pragul minim sunt trunchiate, iar doar biții inferiori (cei mai puțin semnificativi) ai rezultatului sunt returnați; cu alte cuvinte, transportul (carry) este ignorat. Dacă există un overflow (depășire a limitei maxime) sau un underflow (depășire a limitei minime) după o operație de adunare sau scădere, rezultatul ar trebui să fie saturat la valoarea maximă sau minimă posibilă pentru tipul de date.

Reprezentare pe 8 biți fără semn (unsigned):

- Adunare normală (wraparound): 80h + 90h = 10h (eroare, overflow)
- Cu saturație: 80h + 90h = FFh (valoarea maximă)
- Scădere normală (wraparound): 80h - 90h = F0h (underflow)
- Cu saturație: 80h - 90h = 00h (valoarea minimă)

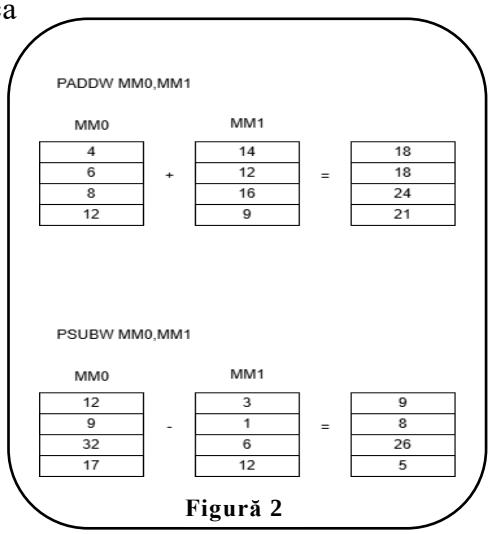
Reprezentare pe 8 biți cu semn (complement pe 2):

- Adunare normală (wraparound): 70h + 20h = 90h (eroare, rezultat negativ)
- Cu saturație: 70h + 20h = 7Fh (valoarea maximă)
- Scădere normală (wraparound): 80h - 20h = 60h (eroare, rezultat pozitiv)
- Cu saturație: 80h - 20h = 80h (valoarea minimă, -128)

3.ANALIZA

Specificațiile Unității Aritmetice MMX

Operațiile care vor fi implementate in aceasta unitate aritmetica sunt:



Figură 2

- Adunare/Scăderea– În hardware, instrucțiunea PADD/PSUB funcționează prin realizarea de adunări/scăderi paralele pe segmente de date, cum ar fi octeți sau cuvinte, într-o arhitectură SIMD (Single Instruction, Multiple Data). Această operație permite efectuarea mai multor adunări/scăderi simultan într-o singură instrucțiune, procesând fiecare segment separat.

- Se vor implementa:

  PADD/PSUB (B, W, D)-wraparound

  PADDS/PSUBS (B, W)- saturatie cu semn

  PADDUS/PSUBUS (B, W)-saturatie fara semn

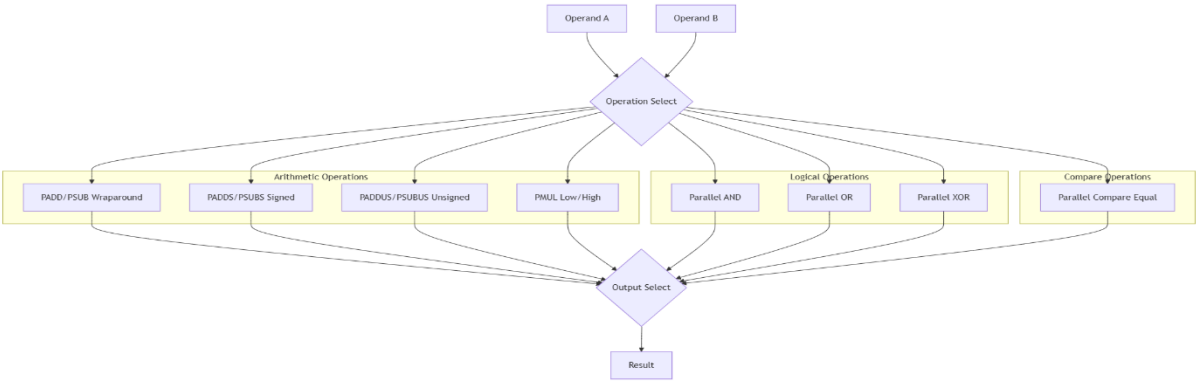
- Comparare (PCMPEQW) - compară segmente de 16 biți pentru a verifica dacă sunt egale și returnează un indicator de egalitate per segment. Dacă segmentele sunt egale, valoarea 0xFFFF este setată pe segmentul de ieșire corespunzător; altfel, se setează 0x0000.

- Operații Logice (PAND,POR,PXOR) - Aceste instrucțiuni permit manipularea bit cu bit, aplicând operațiile logice AND, OR sau XOR între segmentele de date. Fiecare segment din primul registru este combinat cu segmentul corespunzător din al doilea registru.

- Multiplicare (PMULLW/PMULHW) - realizează multiplicarea segmentată, aplicând operația pe câte două segmente de 16 biți. Rezultatul fiecărei multiplicări este stocat într-un registru de ieșire. Pentru a evita overflow-ul, unitatea de control gestionează overflow-ul pe fiecare segment

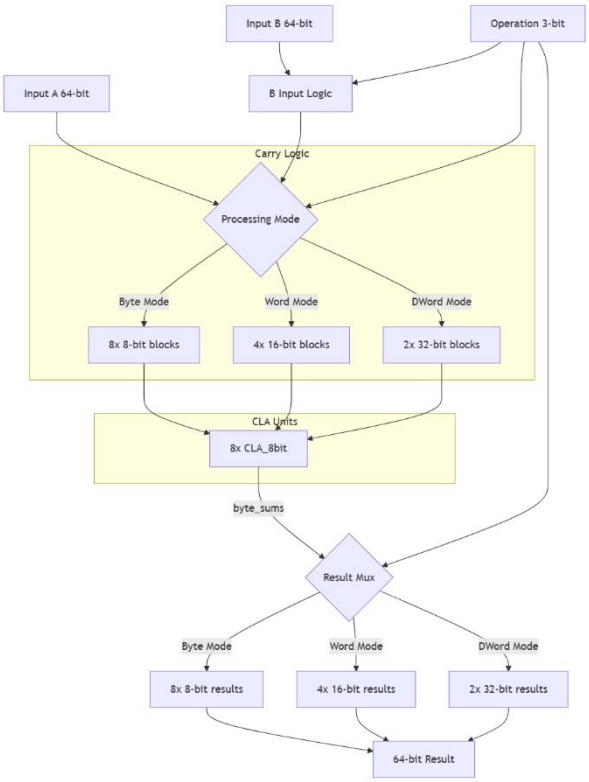
4.DESIGN

Arhitectura generală



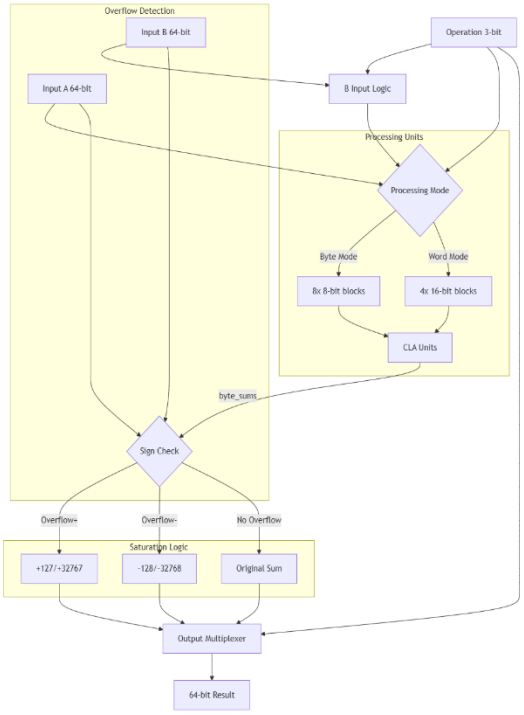
Figură 3

PADD/PSUB (B, W, D)-wraparound



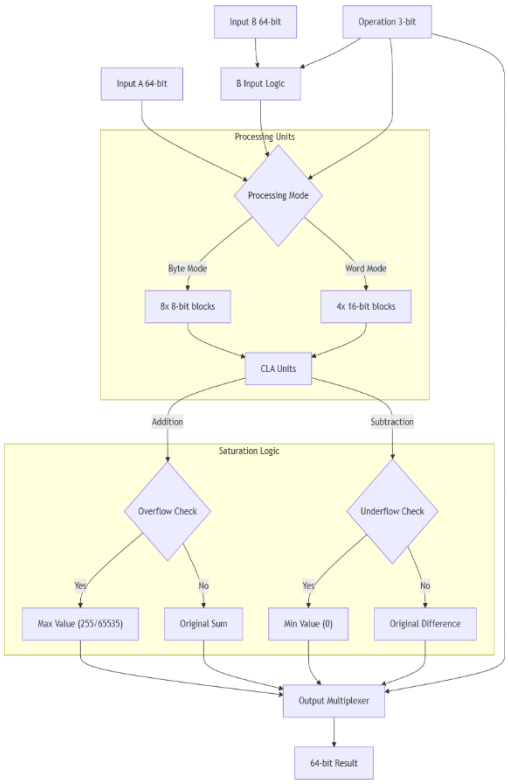
Figură 4

PADDS/PSUBS (B, W)- saturatie cu semn



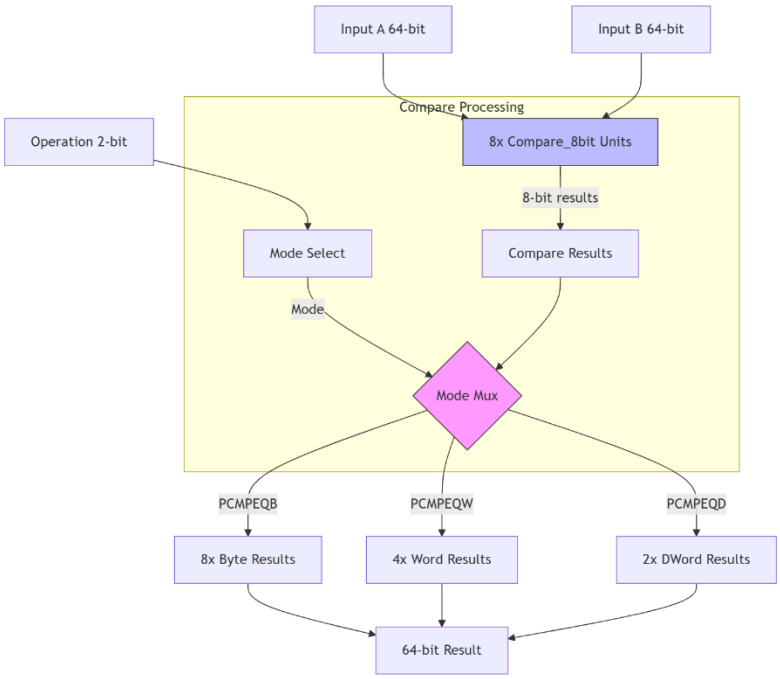
Figură 5

PADDUS/PSUBUS (B, W)-saturatie fara semn



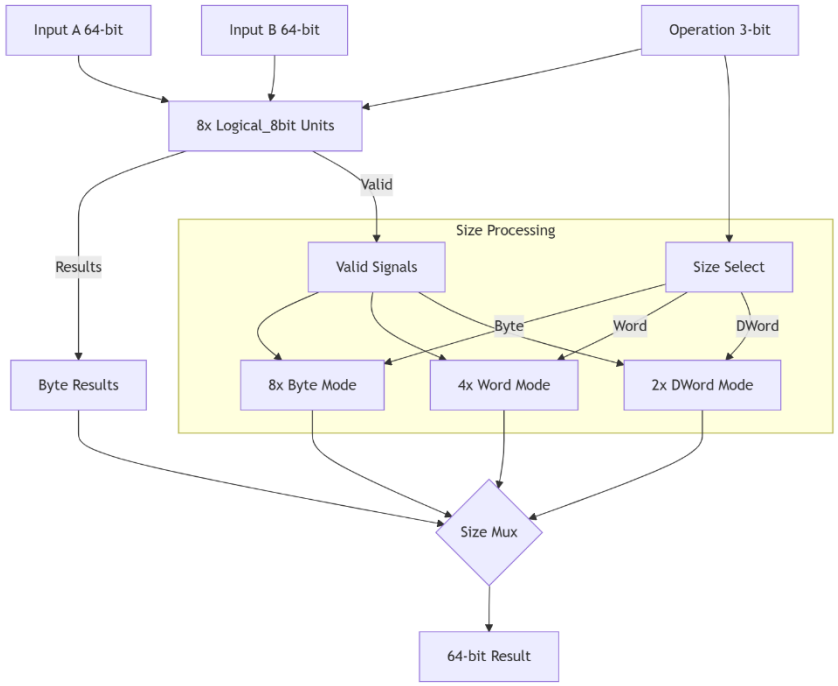
Figură 6

PCMPEQW



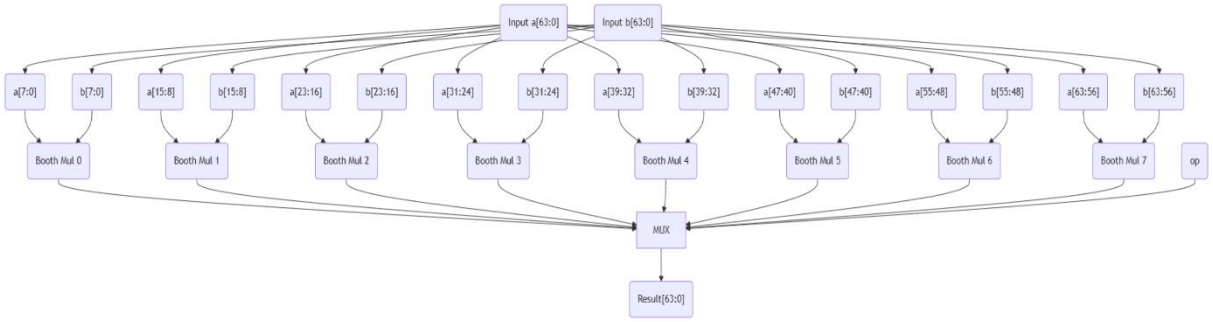
Figură 7

PAND/PXOR/POR



Figură 8

PMULLW/PMULHW



Figură 9

5.IMPLEMENTARE

Unitatea MMX este implementată printr-o arhitectură modulară care conține următoarele componente principale. Pentru fiecare componentă, vom prezenta atât descrierea funcțională cât și implementarea în cod VHDL:

Unitatea de Adunare/Scădere (unified_padd_psub)

- Gestionează operații de adunare și scădere paralelă
- Suportă trei moduri de operare:
  - Wraparound (depășire circulară)
  - Saturație cu semn
  - Saturație fără semn
- Operează pe bytes (8 biți), words (16 biți) și double words (32 biți)

Implementarea include un sistem complex de gestionare a operațiilor prin intermediul unui semnal de control cu 5 biți:

```
entity unified_padd_psub is
    Port (
        a, b      : in  std_logic_vector(63 downto 0);
        operation: in  std_logic_vector(4 downto 0);
        -- operation(4 downto 3) = operation type:
        --   00: wraparound
        --   01: signed saturation
        --   10: unsigned saturation
        --   11: --
        -- operation(2) = subtract ('1') / add ('0')
        -- operation(1:0) = operand size (00=byte, 01=word, 10=dword)
        result    : out std_logic_vector(63 downto 0)
    );
end unified_padd_psub;
```

Figură 10

Pentru gestionarea saturației, se folosesc constante pentru valorile maxime și minime:

```
constant MAX_POS_BYTE : std_logic_vector(7 downto 0) := "01111111";  -- +127
constant MIN_NEG_BYTE : std_logic_vector(7 downto 0) := "10000000";  -- -128
constant MAX_POS_WORD : std_logic_vector(15 downto 0) := "0111111111111111";  -- +32767
constant MIN_NEG_WORD : std_logic_vector(15 downto 0) := "1000000000000000";  -- -32768
constant MAX_UNSIGNED_BYTE : std_logic_vector(7 downto 0) := "11111111";  -- 255
constant MAX_UNSIGNED_WORD : std_logic_vector(15 downto 0) := "1111111111111111";  -- 65535
```

Figură 11

Logica pentru detectarea overflow-ului în modul cu saturație cu semn:

```
for i in 0 to 7 loop
    if operation(2) = '0' then
        -- Addition
        -- Positive overflow
        if (a((i+1)*8-1) = '0' and b((i+1)*8-1) = '0' and byte_sums(i)(7) = '1') then
            result((i+1)*8-1 downto i*8) <= MAX_POS_BYTE;
        -- Negative overflow
        elsif (a((i+1)*8-1) = '1' and b((i+1)*8-1) = '1' and byte_sums(i)(7) = '0') then
            result((i+1)*8-1 downto i*8) <= MIN_NEG_BYTE;
        else
            result((i+1)*8-1 downto i*8) <= byte_sums(i);
        end if;
```

Figură 12

Unitatea Logică Paralelă (parallel_logical)

- Implementează operații logice paralele: AND, OR XOR
- Suportă operații pe diferite dimensiuni de date:
    - Byte (8 biți)
    - Word (16 biți)
    - Double word (32 biți)

Implementarea folosește o abordare modulară cu componente de 8 biți:

```
entity parallel_logical is
    Port (
        a, b : in std_logic_vector(63 downto 0);
        operation : in std_logic_vector(3 downto 0);
        -- operation(3 downto 2) = operation type
        -- 00-AND 10-OR 01-XOR
        -- operation(1 downto 0) = size(00:byte, 01:word, 10:dword)
        result : out std_logic_vector(63 downto 0)
    );
end parallel_logical;
```

Figură 13

Componenta de bază pentru operații logice pe 8 biți:

```
    component logical_8bit is
        Port (
            a, b : in std_logic_vector(7 downto 0);
            op_type : in std_logic_vector(1 downto 0);
            result : out std_logic_vector(7 downto 0);
            result_valid : out std_logic
        );
    end component;
```

Figură 14

Exemplu de implementare pentru operațiile logice:

```
case op_type is
    when "00" =>    -- PAND
        result <= a and b;
    when "10" =>    -- POR
        result <= a or b;
    when "01" =>    -- PXOR
        result <= a xor b;
    when others =>
        result <= (others => '0');
```

Figură 15

Unitatea de Comparare Paralelă (parallel_compare_equality)

- Realizează comparații de egalitate între operanzi
- Suportă trei tipuri de comparații:
    - PCMPEQB (comparare bytes)
    - PCMPEQW (comparare words)
    - PCMPEQD (comparare double words)
    -

Implementarea folosește comparatoare de 8 biți și combină rezultatele în funcție de dimensiunea operației:

```
entity parallel_compare_equality is
    Port (
        a, b : in std_logic_vector(63 downto 0);
        operation : in std_logic_vector(1 downto 0);
        -- "00":PCMPEQB, "01":PCMPEQW, "10":PCMPEQD
        result : out std_logic_vector(63 downto 0)
    );
end parallel_compare_equality;
```

Figură 16

Ex: Implementare pentru Dword

```
when "01" =>  -- PCMPEQW
    for i in 0 to 3 loop
        if word_select(i) = '1' then
            if (compare_results(i*2) = x"FF" and compare_results(i*2+1) = x"FF") then
                final_results((i+1)*16-1 downto i*16) <= (others => '1');
            else
                final_results((i+1)*16-1 downto i*16) <= (others => '0');
            end if;
        end if;
    end loop;
```

Figură 17

Unitatea de Înmulțire Paralelă (pmul)

- Implementează două tipuri de înmulțiri:
  - PMULLW (partea inferioară a rezultatului)
  - PMULHW (partea superioară a rezultatului)
- Utilizează algoritmul Booth pentru înmulțire

```
entity pmul is
    port (
        a, b : in std_logic_vector(63 downto 0);
        op : in std_logic; -- '0' for pmullw, '1' for pmulhw
        result : out std_logic_vector(63 downto 0)
    );
end pmul;
```

Figură 18

Implementarea folosește multiplicatorul Booth pentru operațiile de înmulțire:

```
component booth_multiplier is
    port(
        m : in std_logic_vector(7 downto 0);
        r : in std_logic_vector(7 downto 0);
        result : out std_logic_vector(15 downto 0)
    );
end component;
```

Figură 19

```
process(mul_results, op)
begin
    for i in 0 to 7 loop
        if op = '0' then
            -- PMULLW: lower byte
            result((i*8+7) downto (i*8)) <= mul_results(i)(7 downto 0);
        else
            -- PMULHW: upper byte
            result((i*8+7) downto (i*8)) <= mul_results(i)(15 downto 8);
        end if;
    end loop;
end process;
```

Figură 20

6.TESTARE

Testarea unității MMX a fost realizată folosind un testbench comprehensiv (mmx_unit_tb.vhd) care verifică toate operațiile principale. Mai jos sunt prezentate cazurile de test importante și rezultatele obținute:

Test 1: Adunare Byte (Wraparound)

- Opcode: 0000000
- Operanzi:
  - a = 0x0102030405060708
  - b = 0x0807060504030201
- Rezultat așteptat: 0x0909090909090909
- Rezultat obținut: 0x0909090909090909
- Status: PASS

Test 2: Adunare Word (Wraparound)

- Opcode: 0000001
- Operanzi:
  - a = 0x0001000200030004
  - b = 0x0004000300020001
- Rezultat așteptat: 0x0005000500050005
- Rezultat obținut: 0x0005000500050005
- Status: PASS

Test 3: Scădere Byte (Saturație cu Semn)

- Opcode: 0001100
- Operanzi:
  - a = 0x7F7F7F7F7F7F7F7F
  - b = 0x8080808080808080
- Rezultat așteptat: 0x7F7F7F7F7F7F7F7F
- Rezultat obținut: 0x7F7F7F7F7F7F7F7F
- Status: PASS

Test 4: Adunare Word (Saturație fără Semn)

- Opcode: 0010001
- Operanzi:
  - a = 0xFFFF000100020003
  - b = 0x0001FFFF00030004
- Rezultat așteptat: 0xFFFFFFFF00050007
- Rezultat obținut: 0xFFFFFFFF00050007
- Status: PASS

Test 5: Operație AND pe Bytes

- Opcode: 0100000
- Operanzi:
  - a = 0xFF00FF00FF00FF00
  - b = 0x0F0F0F0F0F0F0F0F
- Rezultat așteptat: 0x0F000F000F000F00
- Rezultat obținut: 0x0F000F000F000F00
- Status: PASS

Figură 21

Test 6: Operație OR pe Words

- Opcode: 0101001
- Operanzi:
  - a = 0xF0F00000F0F00000
  - b = 0x0F0F0F0F0F0F0F0F
- Rezultat așteptat: 0xFFFF0F0FFFFF0F0F
- Rezultat obținut: 0xFFFF0F0FFFFF0F0F
- Status: PASS

Test 7: Operație XOR pe Double Words

- Opcode: 0100110
- Operanzi:
  - a = 0xAAAAAAAAAAAAAAAA
  - b = 0x5555555555555555
- Rezultat așteptat: 0xFFFFFFFFFFFFFFFF
- Rezultat obținut: 0xFFFFFFFFFFFFFFFF
- Status: PASS

Test 8: Comparare Bytes (Egale)

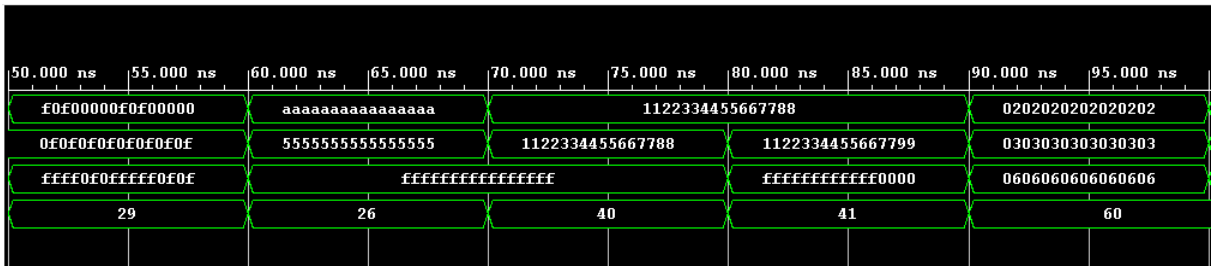
- Opcode: 1000000
- Operanzi:
  - a = 0x1122334455667788
  - b = 0x1122334455667788
- Rezultat așteptat: 0xFFFFFFFFFFFFFFFF
- Rezultat obținut: 0xFFFFFFFFFFFFFFFF
- Status: PASS

Test 9: Comparare Words (Parțial Egale)

- Opcode: 1000001
- Operanzi:
  - a = 0x1122334455667788
  - b = 0x1122334455667799
- Rezultat așteptat: 0xFFFFFFFFFFFF0000
- Rezultat obținut: 0xFFFFFFFFFFFF0000
- Status: PASS
-

Test 10: PMULLW (Înmulțire Pozitiv cu Pozitiv)

- Opcode: 1100000
- Operanzi:
  - a = 0x0202020202020202
  - b = 0x0303030303030303
- Rezultat așteptat: 0x0606060606060606
- Rezultat obținut: 0x0606060606060606
- Status: PASS



Figură 22

Test 11: PMULLW (Înmulțire Word - negativ * pozitiv)

- Opcode: 1100000 (PMULLW)
- Input A: 0x8080808080808080
- Input B: 0x0202020202020202
- Expected: 0x0000000000000000
- Status: ✓ PASS

Test 12: Test operanzi zero

- Opcode: 0000000
- Input A: 0x0000000000000000
- Input B: 0x0000000000000000
- Expected: 0x0000000000000000
- Status: ✓ PASS

Test 13: Cod de operație invalid

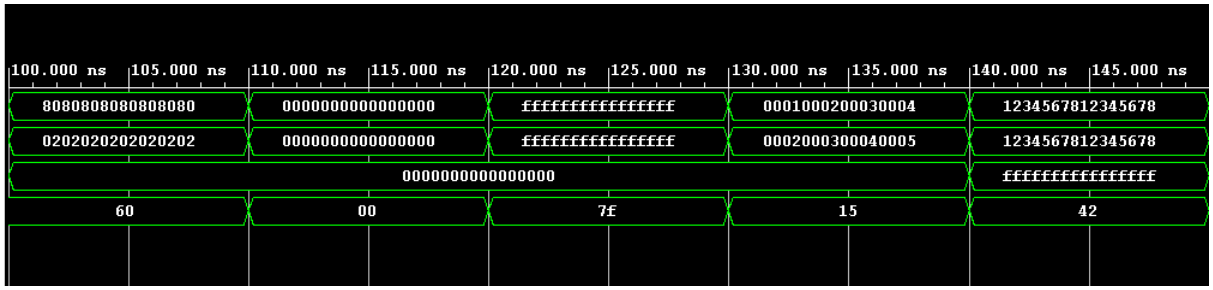
- Opcode: 1111111
- Input A: 0xFFFFFFFFFFFFFFFF
- Input B: 0xFFFFFFFFFFFFFFFF
- Expected: 0x0000000000000000
- Status: ✓ PASS

Test 14: Scădere Word cu saturație fără semn (test underflow)

- Opcode: 0010101
- Input A: 0x0001000200030004
- Input B: 0x0002000300040005
- Expected: 0x0000000000000000
- Status: ✓ PASS

Test 15: Comparare egalitate DWord

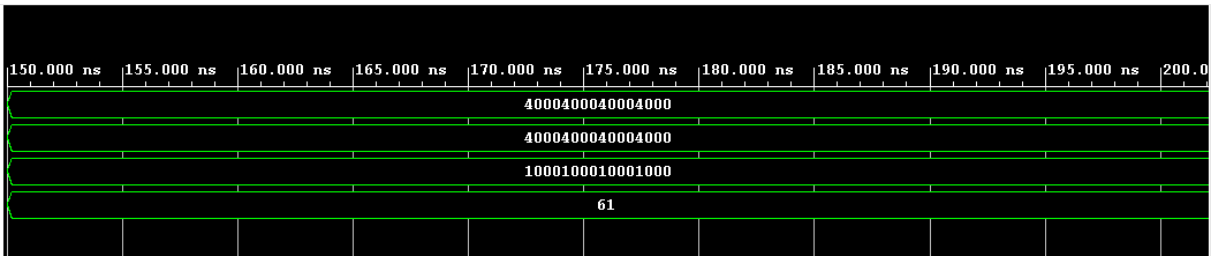
- Opcode: 1000010 (PCMPEQD)
- Input A: 0x1234567812345678
- Input B: 0x1234567812345678
- Expected: 0xFFFFFFFFFFFFFFFF
- Status: ✓ PASS
- Descriere: Verifică compararea de egalitate pentru operanzi double word



Figură 23

Test 16: PMULHW (Înmulțire cu Semn - Partea Superioară)

- Opcode: 1100001
- Operanzi:
  - a = 0x4000400040004000
  - b = 0x4000400040004000
- Rezultat așteptat: 0x1000100010001000
- Rezultat obținut: 0x1000100010001000
- Status: PASS



Figură 24

7.CONCLUZII

Această implementare în VHDL demonstrează cu succes un procesor de tip MMX care suportă operații multimedia esențiale. Arhitectura modulară a sistemului permite o separare clară între diferitele unități funcționale, folosind componente reutilizabile precum adderul cu anticiparea transportului (CLA) și multiplicatorul Booth.

Procesorul suportă o gamă largă de operații, incluzând aritmetică împachetată cu trei moduri (wraparound, saturație cu semn și fără semn), operații logice (AND, OR, XOR), operații de comparare și multiplicare. Un aspect important este flexibilitatea în ceea ce privește dimensiunea operanzilor, sistemul putând procesa date pe 8 biți (byte), 16 biți (word) și 32 biți (dword)..

Arhitectura permite extinderi viitoare pentru instrucțiuni suplimentare MMX și optimizări de performanță. Sistemul de testare poate fi de asemenea extins pentru a include mai multe cazuri de test și metode de verificare formală.

În concluzie, această implementare oferă o bază solidă pentru procesarea instrucțiunilor multimedia, demonstrând conceptele SIMD (Single Instruction, Multiple Data) într-un context hardware real.

Bibliografie:

- x86 Assembly Language Reference Manual-Oracle
- Intel® 64 and IA-32 Architectures Software Developer's Manual ,Volume 2 (2A, 2B, 2C, & 2D):Instruction Set Reference, A-Z
- Intel ArchitectureSoftware Developer's , Manual Volume 1:Basic Architecture