INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

Mult8bit

Practica 10

Materia:

Arquitectura de computadoras

Profesor:

Castillo Cabrera Gelacio

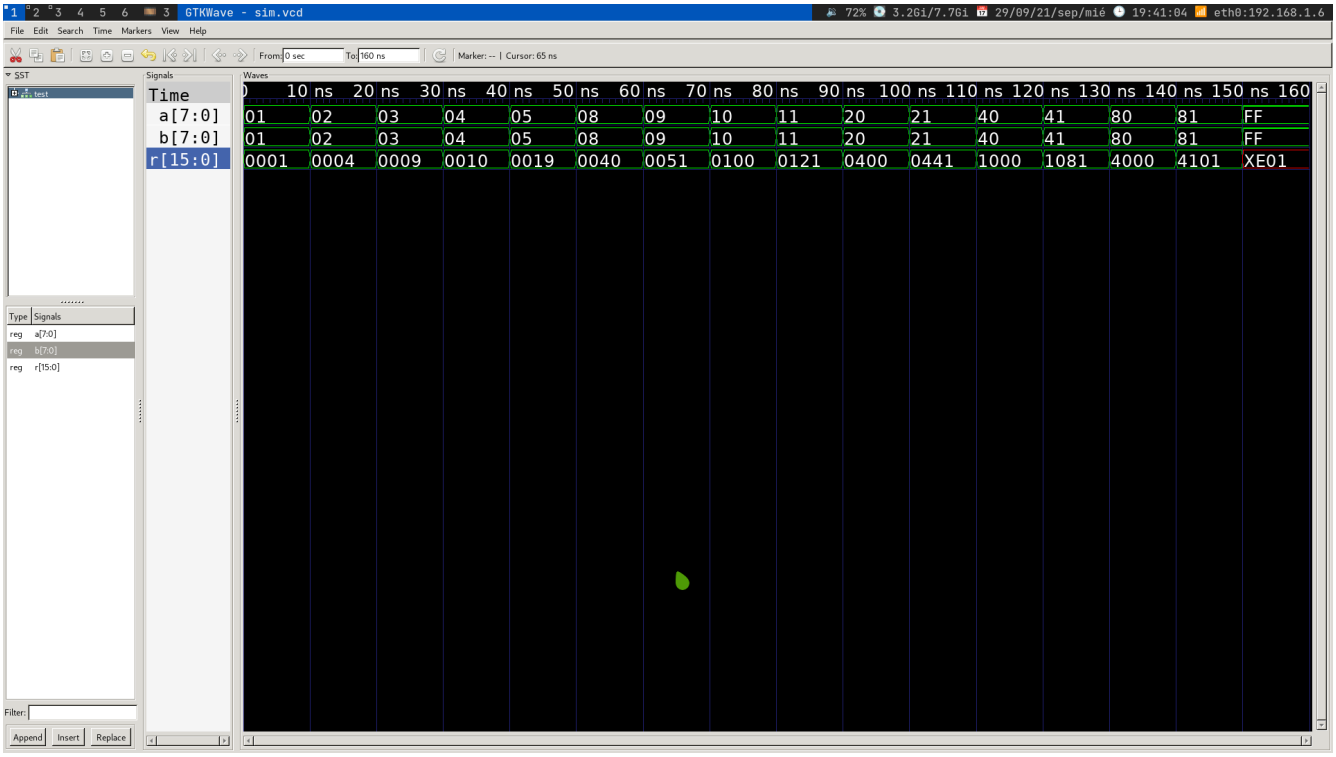
Alumno:

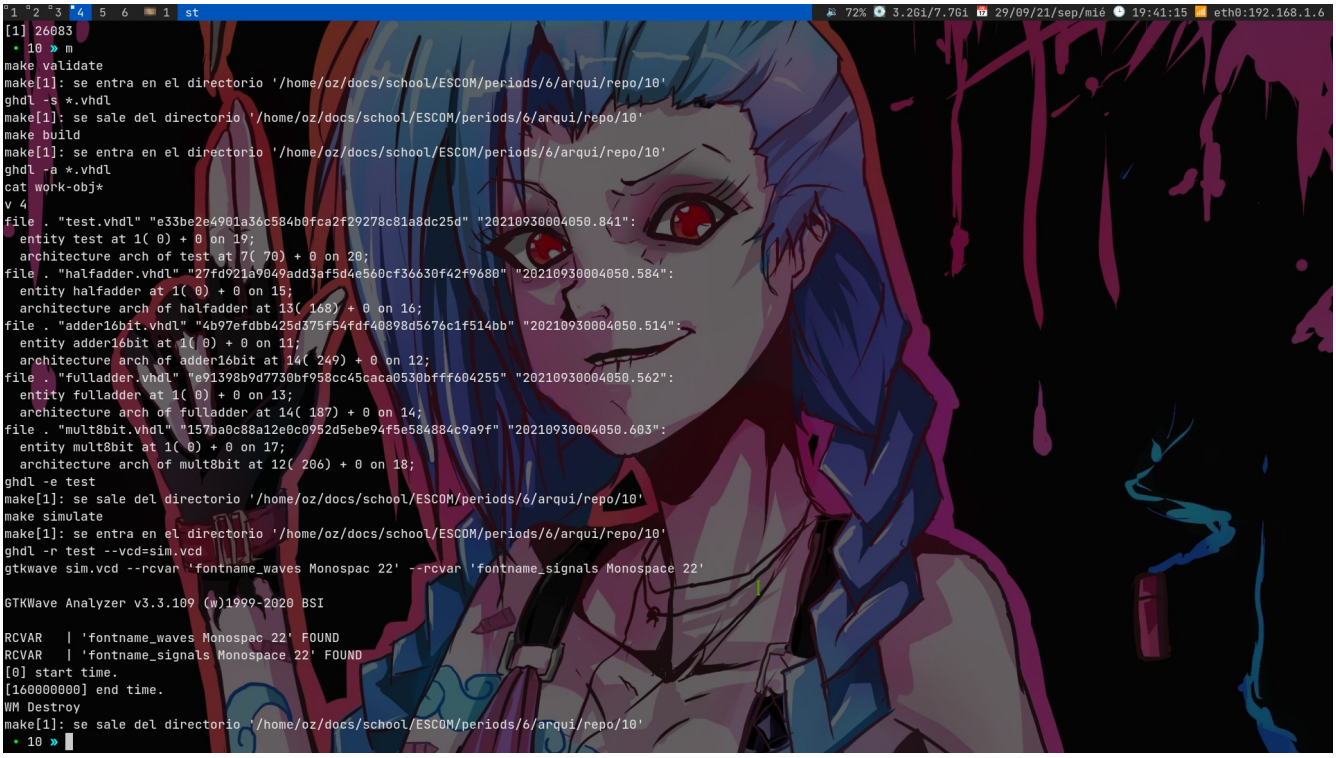
Cortés Piña Oziel

Grupo:

3CM12

Simulación en GHDL y GTLWAVE





Código VHDL

Halfadder

```
library ieee;
use ieee.std_logic_1164.all;

entity halfadder is
      port(
            a:    in    std_logic;
            b:    in    std_logic;
            s:    out   std_logic;
            c:    out   std_logic
      );
end halfadder;

architecture arch of halfadder is
begin
      s <= a xor b;
      c <= a and b;
end arch;
```

Fulladder

```
library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
      port(
            a:    in    std_logic;
            b:    in    std_logic;
            e:    in    std_logic;
            s:    out   std_logic;
            c:    out   std_logic
      );
end fulladder;

architecture arch of fulladder is
      component halfadder is
            port(
                  a:    in    std_logic;
                  b:    in    std_logic;
                  s:    out   std_logic;
                  c:    out   std_logic
            );
      end component;
      signal sum, carryA, carryB:     std_logic;
begin
      first: halfadder    port map( a => a, b => b, s => sum,    c => carryA );
      second:      halfadder    port map( a => e, b => sum,     s => s,            c =>
carryB );
      c <= carryA or carryB;
end arch;
```

Adder16bit

```
library ieee;
use ieee.std_logic_1164.all;

entity adder16bit is
	port(
		a:	in	std_logic_vector(15 downto 0);
		b:	in	std_logic_vector(15 downto 0);
		e:	in	std_logic;
		s:	out	std_logic_vector(15 downto 0);
		c:	out	std_logic
	);
end adder16bit;

architecture arch of adder16bit is
	component fulladder  is
		port(
			a:	in	std_logic;
			b:	in	std_logic;
			e:	in	std_logic;
			s:	out	std_logic;
			c:	out	std_logic
		);
	end component;
	signal carry: std_logic_vector(14 downto 0);
begin
	s00: fulladder port map( a => a(0),  b => b(0),  e => e,           s => s(0),  c => carry(0) );
	s01: fulladder port map( a => a(1),  b => b(1),  e => carry(0),  s => s(1),  c => carry(1) );
	s02: fulladder port map( a => a(2),  b => b(2),  e => carry(1),  s => s(2),  c => carry(2) );
	s03: fulladder port map( a => a(3),  b => b(3),  e => carry(2),  s => s(3),  c => carry(3) );
	s04: fulladder port map( a => a(4),  b => b(4),  e => carry(3),  s => s(4),  c => carry(4) );
	s05: fulladder port map( a => a(5),  b => b(5),  e => carry(4),  s => s(5),  c => carry(5) );
	s06: fulladder port map( a => a(6),  b => b(6),  e => carry(5),  s => s(6),  c => carry(6) );
	s07: fulladder port map( a => a(7),  b => b(7),  e => carry(6),  s => s(7),  c => carry(7) );
	s08: fulladder port map( a => a(8),  b => b(8),  e => carry(7),  s => s(8),  c => carry(8) );
	s09: fulladder port map( a => a(9),  b => b(9),  e => carry(8),  s => s(9),  c => carry(9) );
	s10: fulladder port map( a => a(10), b => b(10), e => carry(9),  s => s(10), c => carry(10) );
	s11: fulladder port map( a => a(11), b => b(11), e => carry(10), s => s(11), c => carry(11) );
	s12: fulladder port map( a => a(12), b => b(12), e => carry(11), s => s(12), c => carry(12) );
	s13: fulladder port map( a => a(13), b => b(13), e => carry(12), s => s(13), c => carry(13) );
	s14: fulladder port map( a => a(14), b => b(14), e => carry(13), s => s(14), c => carry(14) );
	s15: fulladder port map( a => a(15), b => b(15), e => carry(14), s => s(15), c => c );
end arch;
```

Mult8bit

```
library ieee;
use ieee.std_logic_1164.all;

entity mult8bit is
	port(
		a:	in	std_logic_vector(7  downto 0);
		b:	in	std_logic_vector(7  downto 0);
		r:	out	std_logic_vector(15 downto 0)
	);
end mult8bit;

architecture arch of mult8bit is
	component adder16bit is
		port(
			a:	in	std_logic_vector(15 downto 0);
			b:	in	std_logic_vector(15 downto 0);
			e:	in	std_logic;
			s:	out	std_logic_vector(15 downto 0);
			c:	out	std_logic
		);
	end component;
	signal p0, p1, p2, p3, p4, p5, p6, p7:	std_logic_vector(15 downto 0);
	signal s0, s1, s2, s3, s4, s5:			std_logic_vector(15 downto 0);
```

```
        signal carry:                std_logic_vector(5 downto 0);
begin

        p0(00) <= b(0) and a(0);
        p0(01) <= b(1) and a(0);
        p0(02) <= b(2) and a(0);
        p0(03) <= b(3) and a(0);
        p0(04) <= b(4) and a(0);
        p0(05) <= b(5) and a(0);
        p0(06) <= b(6) and a(0);
        p0(07) <= b(7) and a(0);
        p0(08) <= '0';
        p0(09) <= '0';
        p0(10) <= '0';
        p0(11) <= '0';
        p0(12) <= '0';
        p0(13) <= '0';
        p0(14) <= '0';
        p0(15) <= '0';

        p1(00) <= '0';
        p1(01) <= b(0) and a(1);
        p1(02) <= b(1) and a(1);
        p1(03) <= b(2) and a(1);
        p1(04) <= b(3) and a(1);
        p1(05) <= b(4) and a(1);
        p1(06) <= b(5) and a(1);
        p1(07) <= b(6) and a(1);
        p1(08) <= b(7) and a(1);
        p1(09) <= '0';
        p1(10) <= '0';
        p1(11) <= '0';
        p1(12) <= '0';
        p1(13) <= '0';
        p1(14) <= '0';
        p1(15) <= '0';

        p2(00) <= '0';
        p2(01) <= '0';
        p2(02) <= b(0) and a(2);
        p2(03) <= b(1) and a(2);
        p2(04) <= b(2) and a(2);
        p2(05) <= b(3) and a(2);
        p2(06) <= b(4) and a(2);
        p2(07) <= b(5) and a(2);
        p2(08) <= b(6) and a(2);
        p2(09) <= b(7) and a(2);
        p2(10) <= '0';
        p2(11) <= '0';
        p2(12) <= '0';
        p2(13) <= '0';
        p2(14) <= '0';
        p2(15) <= '0';

        p3(00) <= '0';
        p3(01) <= '0';
        p3(02) <= '0';
        p3(03) <= b(0) and a(3);
        p3(04) <= b(1) and a(3);
        p3(05) <= b(2) and a(3);
        p3(06) <= b(3) and a(3);
        p3(07) <= b(4) and a(3);
        p3(08) <= b(5) and a(3);
        p3(09) <= b(6) and a(3);
        p3(10) <= b(7) and a(3);
        p3(11) <= '0';
        p3(12) <= '0';
        p3(13) <= '0';
        p3(14) <= '0';
        p3(15) <= '0';

        p4(00) <= '0';
```

```
p4(01) <= '0';
p4(02) <= '0';
p4(03) <= '0';
p4(04) <= b(0) and a(4);
p4(05) <= b(1) and a(4);
p4(06) <= b(2) and a(4);
p4(07) <= b(3) and a(4);
p4(08) <= b(4) and a(4);
p4(09) <= b(5) and a(4);
p4(10) <= b(6) and a(4);
p4(11) <= b(7) and a(4);
p4(12) <= '0';
p4(13) <= '0';
p4(14) <= '0';
p4(15) <= '0';

p5(00) <= '0';
p5(01) <= '0';
p5(02) <= '0';
p5(03) <= '0';
p5(04) <= '0';
p5(05) <= b(0) and a(5);
p5(06) <= b(1) and a(5);
p5(07) <= b(2) and a(5);
p5(08) <= b(3) and a(5);
p5(09) <= b(4) and a(5);
p5(10) <= b(5) and a(5);
p5(11) <= b(6) and a(5);
p5(12) <= b(7) and a(5);
p5(13) <= '0';
p5(14) <= '0';
p5(15) <= '0';

p6(00) <= '0';
p6(01) <= '0';
p6(02) <= '0';
p6(03) <= '0';
p6(04) <= '0';
p6(05) <= '0';
p6(06) <= b(0) and a(6);
p6(07) <= b(1) and a(6);
p6(08) <= b(2) and a(6);
p6(09) <= b(3) and a(6);
p6(10) <= b(4) and a(6);
p6(11) <= b(5) and a(6);
p6(12) <= b(6) and a(6);
p6(13) <= b(7) and a(6);
p6(14) <= '0';
p6(15) <= '0';

p7(00) <= '0';
p7(01) <= '0';
p7(02) <= '0';
p7(03) <= '0';
p7(04) <= '0';
p7(05) <= '0';
p7(06) <= '0';
p7(07) <= b(0) and a(7);
p7(08) <= b(1) and a(7);
p7(09) <= b(2) and a(7);
p7(10) <= b(3) and a(7);
p7(11) <= b(4) and a(7);
p7(12) <= b(5) and a(7);
p7(13) <= b(6) and a(7);
p7(14) <= b(7) and a(7);
p7(15) <= '0';

r0: adder16bit port map( a => p0, b => p1, e => '0', s => s0, c => carry(0) );
r1: adder16bit port map( a => s0, b => p2, e => carry(0), s => s1,c => carry(1) );
r2: adder16bit port map( a => s1, b => p3, e => carry(1), s => s2,c => carry(2) );
r3: adder16bit port map( a => s2, b => p4, e => carry(2), s => s3,c => carry(3) );
r4: adder16bit port map( a => s3, b => p5, e => carry(3), s => s4,c => carry(4) );
```

```
      r5: adder16bit port map( a => s4, b => p6, e => carry(4), s => s5,c => carry(5) );
      r6: adder16bit port map( a => s5, b => p7, e => carry(5), s => r, c => r(15) );
end arch;
```

Test

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
end test;

architecture arch of test is
      component mult8bit is
            port(
                  a:    in    std_logic_vector(7  downto 0);
                  b:    in    std_logic_vector(7  downto 0);
                  r:    out   std_logic_vector(15 downto 0)
            );
      end component;
      signal a: std_logic_vector(7  downto 0);
      signal b: std_logic_vector(7  downto 0);
      signal r: std_logic_vector(15 downto 0);
begin
      conn: mult8bit port map( a => a, b => b, r => r );
      process begin
            a <= "00000001";
            b <= "00000001";
            wait for 10 ns;
            a <= "00000010";
            b <= "00000010";
            wait for 10 ns;
            a <= "00000011";
            b <= "00000011";
            wait for 10 ns;
            a <= "00000100";
            b <= "00000100";
            wait for 10 ns;
            a <= "00000101";
            b <= "00000101";
            wait for 10 ns;
            a <= "00001000";
            b <= "00001000";
            wait for 10 ns;
            a <= "00001001";
            b <= "00001001";
            wait for 10 ns;
            a <= "00010000";
            b <= "00010000";
            wait for 10 ns;
            a <= "00010001";
            b <= "00010001";
            wait for 10 ns;
            a <= "00100000";
            b <= "00100000";
            wait for 10 ns;
            a <= "00100001";
            b <= "00100001";
```

```
            wait for 10 ns;
            a <= "01000000";
            b <= "01000000";
            wait for 10 ns;
            a <= "01000001";
            b <= "01000001";
            wait for 10 ns;
            a <= "10000000";
            b <= "10000000";
            wait for 10 ns;
            a <= "10000001";
            b <= "10000001";
            wait for 10 ns;
            a <= "11111111";
            b <= "11111111";
            wait for 10 ns;
            wait;
      end process;
end arch;
```

Análisis de vectores

Vectores de entrada

A		B	
Binario	Hexadecimal	Binario	Hexadecimal
00000001	01	00000001	01
00000010	02	00000010	02
00000011	03	00000011	03
00000100	04	00000100	04
00000101	05	00000101	05
00001000	08	00001000	08
00001001	09	00001001	09
00010000	10	00010000	10
00010001	11	00010001	11
00100000	20	00100000	20
00100001	21	00100001	21
01000000	40	01000000	40
01000001	41	01000001	41
10000000	80	10000000	80
10000001	81	10000001	81
11111111	FF	11111111	FF

Valores de salida

Binario	Hexadecimal
0000 0000 0000 0001	0001
0000 0000 0000 0100	0004
0000 0000 0000 1001	0009
0000 0000 0001 0000	0010
0000 0000 0001 1001	0019
0000 0000 0100 0000	0040
0000 0000 0101 0001	0051
0000 0001 0000 0000	0100
0000 0001 0010 0001	0121
0000 0100 0000 0000	0400
0000 0100 0100 0001	0441
0001 0000 0000 0000	1000
0001 0000 1000 0001	1081
0100 0000 0000 0000	4000
0100 0001 0000 0001	4101
XXXX 1110 0000 0001	XE01

Conclusión

A pesar de que de cierta manera el diseño del circuito es elegante el inocente tamaño lo vuelve complejo por que usar circuitos auxiliares ayudan pero al usar un sumador positivo corrompe los resultados a escalas grandes por lo que no es conveniente este diseño en el cual me apoyo de circuitos anteriores aunque más rápido de elaborar. Los desbordes al momento de multiplicar números grandes sigue siendo un problema.