Rubén Comerón Galán

Sesión 2

Modelo de puerta NAND:

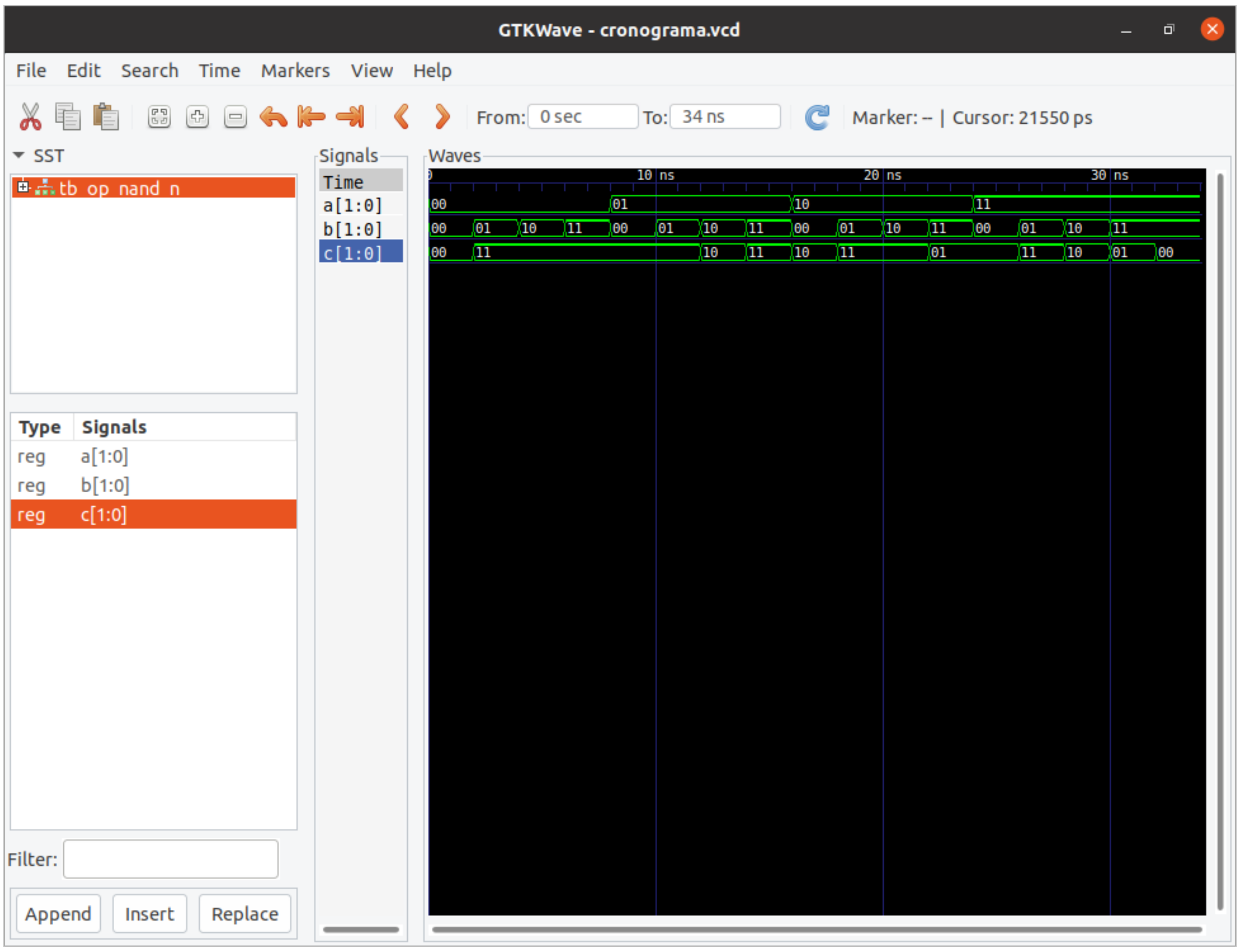
```
ENTITY op_nand_n IS
        GENERIC (n: INTEGER:=8);
        PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r:OUT BIT_VECTOR(n-1
DOWNTO 0));
END op_nand_n;

ARCHITECTURE estructural OF op_nand_n IS
        COMPONENT nand2
                PORT (e1, e2: IN BIT; sal: OUT BIT);
        END COMPONENT;
BEGIN
        bateria: FOR i IN 0 TO n-1 GENERATE
                puertas: nand2 PORT MAP (x(i), y(i), r(i));
        END GENERATE;
END estructural;
```

Modelo del test-bench de la puerta NAND:

```
ENTITY tb_op_nand_n IS
END tb_op_nand_n;

ARCHITECTURE estructural OF tb_op_nand_n IS
        COMPONENT op_nand_n
                GENERIC (n: INTEGER:=8);
                PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r: OUT BIT_VECTOR(n-
1 DOWNTO 0));
```

```
        END  COMPONENT;

        FOR  operador:  op_nand_n  USE  ENTITY  WORK.op_nand_n(estructural);

        CONSTANT  TAMANO_PALABRA:  integer  :=  2;

        SIGNAL  A,  B,  C:  BIT_VECTOR(TAMANO_PALABRA-1  DOWNTO  0);

BEGIN
        operador:  op_nand_n  GENERIC  MAP  (TAMANO_PALABRA)  PORT  MAP(A,  B,
C);

        PROCESS
        VARIABLE  valor,  limite:  INTEGER;
        BEGIN
                limite:=  2**TAMANO_PALABRA-1;

                FOR  i  IN  0  TO  limite  LOOP
                        valor:=i;

                        FOR  k  IN  A'REVERSE_RANGE  LOOP
                                A(k)<=BIT'VAL(valor  REM  2);
                                valor:=valor/2;
                        END  LOOP;

                        FOR  j  IN  0  TO  limite  LOOP
                                valor:=j;

                                FOR  k  IN  B'REVERSE_RANGE  LOOP
                                        B(k)<=BIT'VAL(valor  REM  2);
                                        valor:=valor/2;
                                END  LOOP;
                                WAIT  FOR  2  ns;
                END LOOP;
            END LOOP;

            WAIT FOR 2 ns;
            WAIT;
        END PROCESS;
END estructural;
```

Resultado de la prueba del test-bench de la puerta NAND:



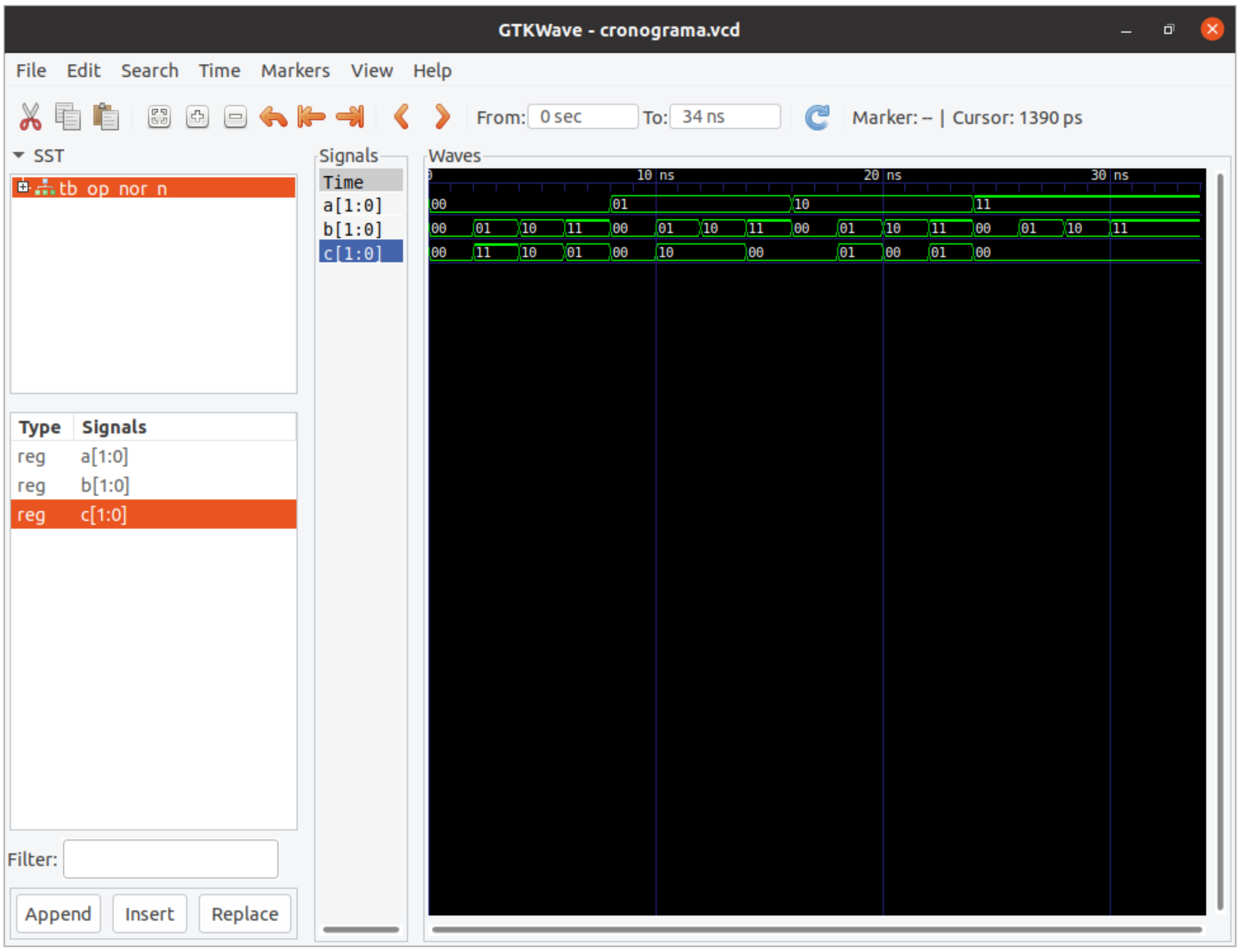
Modelo de puerta NOR:

```
ENTITY op_nor_n IS
        GENERIC (n: INTEGER:=8);
        PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r:OUT BIT_VECTOR(n-1
DOWNTO 0));
END op_nor_n;

ARCHITECTURE estructural OF op_nor_n IS
        COMPONENT nor2
                PORT (e1, e2: IN BIT; sal: OUT BIT);
        END COMPONENT;
BEGIN
        bateria: FOR i IN 0 TO n-1 GENERATE
                puertas: nor2 PORT MAP (x(i), y(i), r(i));
        END GENERATE;
END estructural;
```

Modelo del test-bench de la puerta NOR:

```
ENTITY tb_op_nor_n IS
END tb_op_nor_n;

ARCHITECTURE estructural OF tb_op_nor_n IS
        COMPONENT op_nor_n
                GENERIC (n: INTEGER:=8);
                PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r: OUT BIT_VECTOR(n-
1 DOWNTO 0));
        END COMPONENT;
        FOR operador: op_nor_n USE ENTITY WORK.op_nor_n(estructural);

        CONSTANT TAMANO_PALABRA: integer := 2;

        SIGNAL A, B, C: BIT_VECTOR(TAMANO_PALABRA-1 DOWNTO 0);

BEGIN
        operador: op_nor_n GENERIC MAP (TAMANO_PALABRA) PORT MAP(A, B,
C);

        PROCESS
        VARIABLE valor, limite: INTEGER;
        BEGIN
                limite:= 2**TAMANO_PALABRA-1;

                FOR i IN 0 TO limite LOOP
                        valor:=i;

                        FOR k IN A'REVERSE_RANGE LOOP
                                A(k)<=BIT'VAL(valor REM 2);
                                valor:=valor/2;
                        END LOOP;

                        FOR j IN 0 TO limite LOOP
                                valor:=j;

                                FOR k IN B'REVERSE_RANGE LOOP
                                        B(k)<=BIT'VAL(valor REM 2);
                                        valor:=valor/2;
                                END LOOP;
                                WAIT FOR 2 ns;

                        END LOOP;
                END LOOP;


                WAIT FOR 2 ns;
                WAIT;
        END PROCESS;
END estructural;
```

Resultado de la prueba del test-bench de la puerta NOR:



Modelo de puerta NOT:

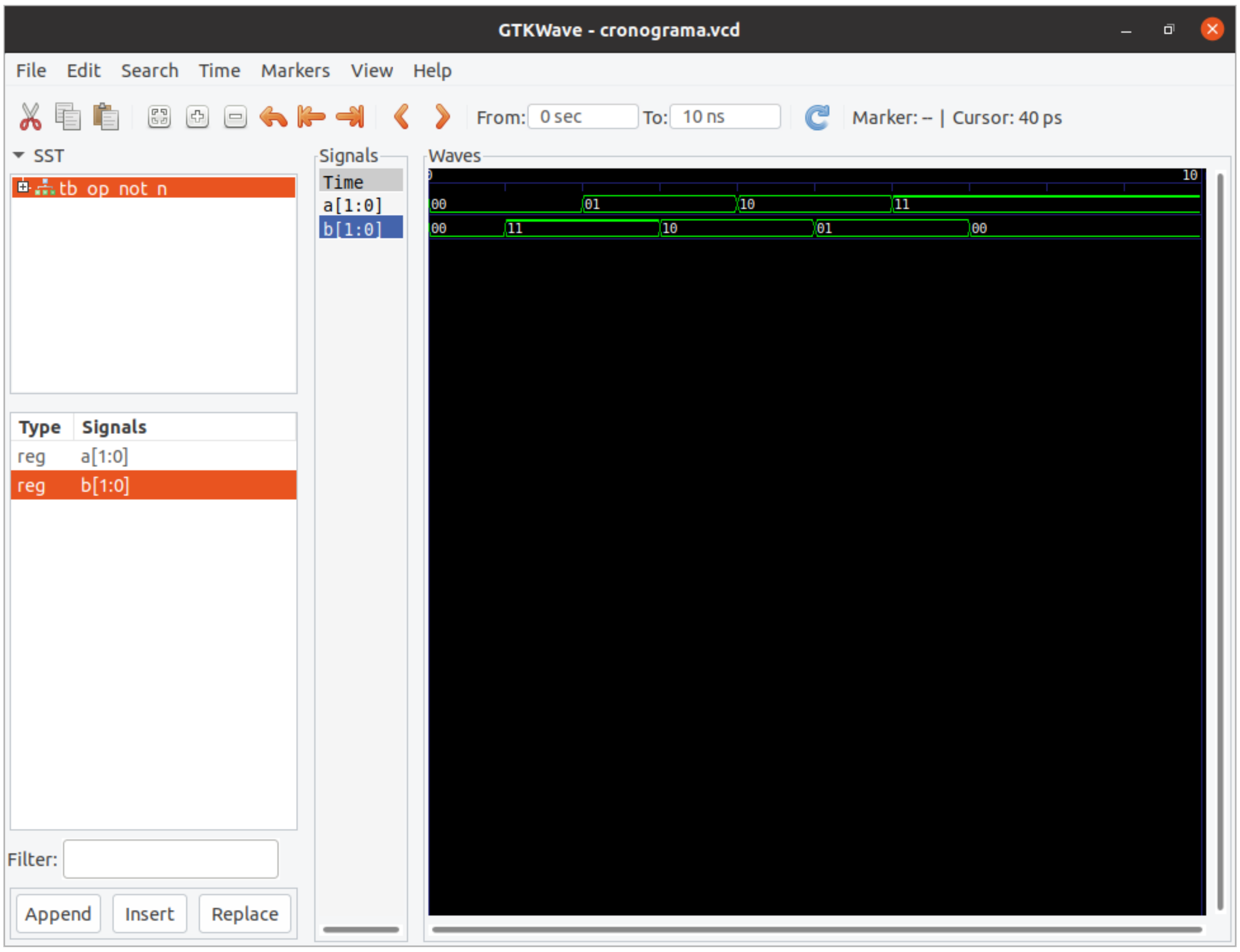
```
ENTITY op_not_n IS
        GENERIC (n: INTEGER:=8);
        PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r, s:OUT BIT_VECTOR(n-1
DOWNTO 0));
END op_not_n;

ARCHITECTURE estructural OF op_not_n IS
        COMPONENT not2
                PORT (e1, e2: IN BIT; sal1, sal2: OUT BIT);
        END COMPONENT;
BEGIN
        bateria: FOR i IN 0 TO n-1 GENERATE
                puertas: not2 PORT MAP (x(i), y(i), r(i), s(i));
        END GENERATE;
END estructural;
```

Modelo del test-bench de la puerta NOT:

```
ENTITY tb_op_not_n IS
END tb_op_not_n;

ARCHITECTURE estructural OF tb_op_not_n IS
        COMPONENT op_not_n
                GENERIC (n: INTEGER:=8);
                PORT (x: IN BIT_VECTOR(n-1 DOWNTO 0); s: OUT BIT_VECTOR(n-1
DOWNTO 0));
        END COMPONENT;
```

```
    FOR operador: op_not_n USE ENTITY WORK.op_not_n(estructural);

    CONSTANT TAMANO_PALABRA: integer := 2;

    SIGNAL A, B: BIT_VECTOR(TAMANO_PALABRA-1 DOWNTO 0);

BEGIN
    operador: op_not_n GENERIC MAP (TAMANO_PALABRA) PORT MAP(A, B);

    PROCESS
    VARIABLE valor, limite: INTEGER;
    BEGIN
        limite:= 2**TAMANO_PALABRA-1;

        FOR i IN 0 TO limite LOOP
            valor:=i;

            FOR k IN A'REVERSE_RANGE LOOP
                A(k)<=BIT'VAL(valor REM 2);
                valor:=valor/2;
            END LOOP;
                WAIT FOR 2 ns;

        END LOOP;

        WAIT FOR 2 ns;
        WAIT;
    END PROCESS;
END estructural;
```

Resultado de la prueba del test-bench de la puerta NOT:



```
a[1:0]  00 01 10 11
b[1:0]  00 11 10 01 00
```

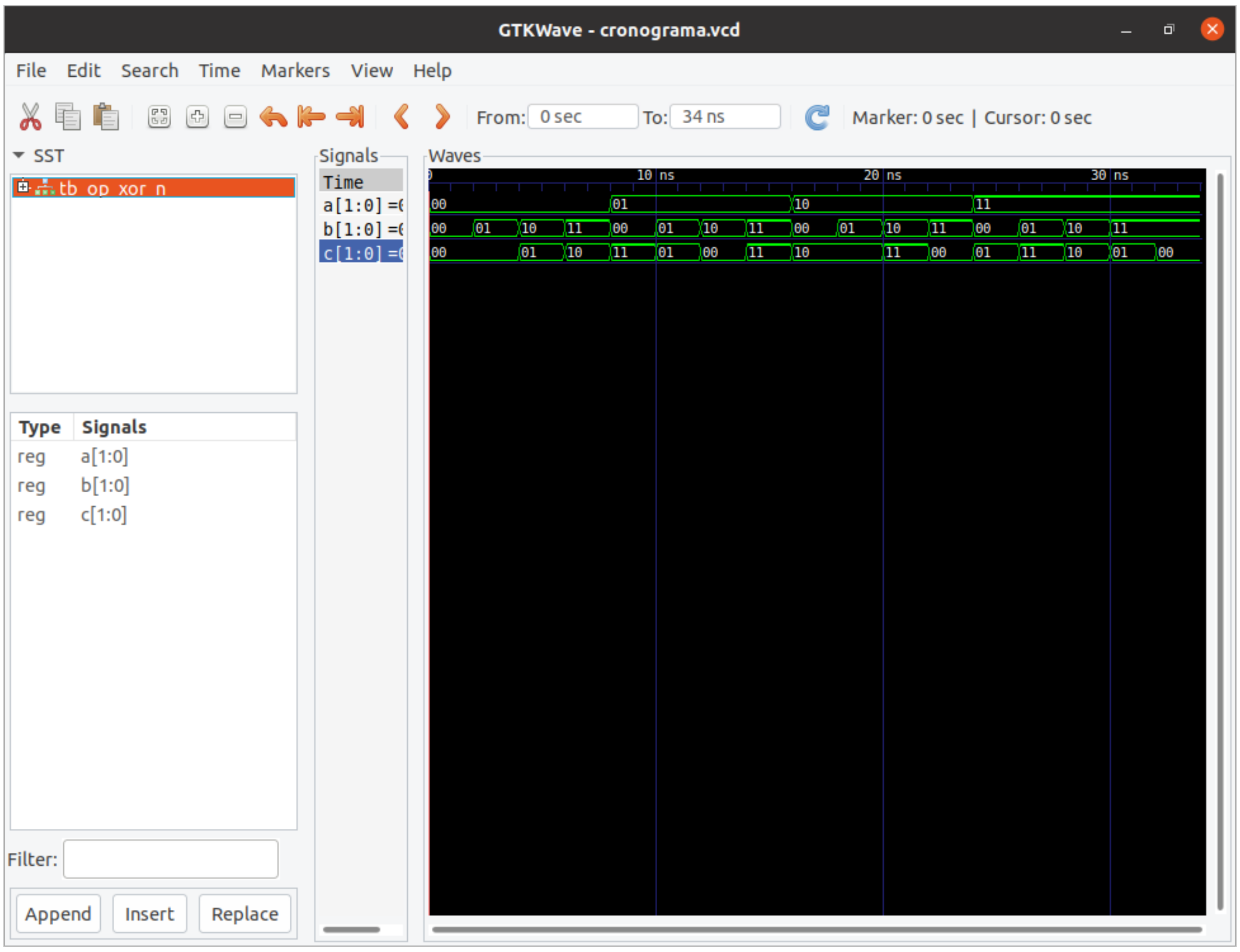
Modelo de puerta XOR:

```
ENTITY op_xor_n IS
        GENERIC (n: INTEGER:=8);
        PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r:OUT BIT_VECTOR(n-1
DOWNTO 0));
END op_xor_n;

ARCHITECTURE estructural OF op_xor_n IS
        COMPONENT xor2
                PORT (e1, e2: IN BIT; sal: OUT BIT);
        END COMPONENT;
BEGIN
        bateria: FOR i IN 0 TO n-1 GENERATE
                puertas: xor2 PORT MAP (x(i), y(i), r(i));
        END GENERATE;
END estructural;
```

Modelo del test-bench de la puerta XOR:

```
ENTITY tb_op_xor_n IS
END tb_op_xor_n;

ARCHITECTURE estructural OF tb_op_xor_n IS
        COMPONENT op_xor_n
                GENERIC (n: INTEGER:=8);
                PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r: OUT BIT_VECTOR(n-
1 DOWNTO 0));
        END COMPONENT;

        FOR operador: op_xor_n USE ENTITY WORK.op_xor_n(estructural);

        CONSTANT TAMANO_PALABRA: integer := 2;

        SIGNAL A, B, C: BIT_VECTOR(TAMANO_PALABRA-1 DOWNTO 0);

BEGIN
        operador: op_xor_n GENERIC MAP (TAMANO_PALABRA) PORT MAP(A, B,
C);

        PROCESS
        VARIABLE valor, limite: INTEGER;
        BEGIN
                limite:= 2**TAMANO_PALABRA-1;

                FOR i IN 0 TO limite LOOP
                        valor:=i;

                        FOR k IN A'REVERSE_RANGE LOOP
                                A(k)<=BIT'VAL(valor REM 2);
                                valor:=valor/2;
                        END LOOP;

                        FOR j IN 0 TO limite LOOP
                                valor:=j;

                                FOR k IN B'REVERSE_RANGE LOOP
                                        B(k)<=BIT'VAL(valor REM 2);
                                        valor:=valor/2;
                                END LOOP;
                                WAIT FOR 2 ns;

                        END LOOP;
                END LOOP;

                WAIT FOR 2 ns;
                WAIT;
        END PROCESS;
END estructural;
```

Resultado del test-bench de la puerta XOR:



Modelo de puerta OR:

ENTITY op_or_n IS

GENERIC (n: INTEGER:=8);

PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r:OUT BIT_VECTOR(n-1 DOWNTO 0));

END op_or_n;

ARCHITECTURE estructural OF op_or_n IS

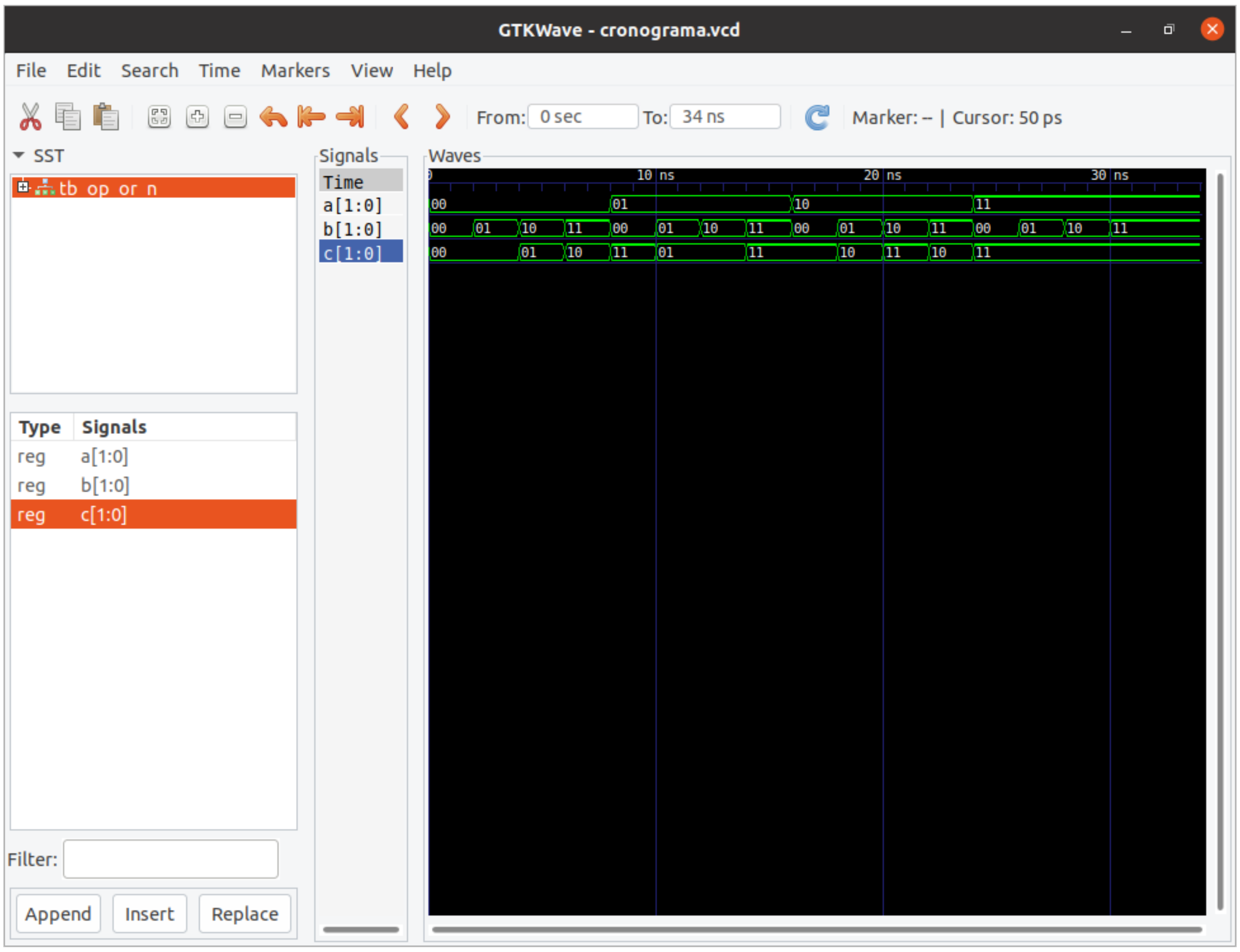
```
        COMPONENT or2
                PORT (e1, e2: IN BIT; sal: OUT BIT);
        END COMPONENT;
BEGIN
        bateria: FOR i IN 0 TO n-1 GENERATE
                puertas: or2 PORT MAP (x(i), y(i), r(i));
        END GENERATE;
END estructural;
```

Modelo del test-bench de la puerta OR:

```
ENTITY tb_op_or_n IS
END tb_op_or_n;

ARCHITECTURE estructural OF tb_op_or_n IS
        COMPONENT op_or_n
                GENERIC (n: INTEGER:=8);
                PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r: OUT BIT_VECTOR(n-
1 DOWNTO 0));
        END COMPONENT;

        FOR operador: op_or_n USE ENTITY WORK.op_or_n(estructural);

        CONSTANT TAMANO_PALABRA: integer := 2;

        SIGNAL A, B, C: BIT_VECTOR(TAMANO_PALABRA-1 DOWNTO 0);

BEGIN
        operador: op_or_n GENERIC MAP (TAMANO_PALABRA) PORT MAP(A, B, C);
```

```
    PROCESS
    VARIABLE valor, limite: INTEGER;
    BEGIN
        limite:= 2**TAMANO_PALABRA-1;

        FOR i IN 0 TO limite LOOP
            valor:=i;

            FOR k IN A'REVERSE_RANGE LOOP
                A(k)<=BIT'VAL(valor REM 2);
                valor:=valor/2;
            END LOOP;

            FOR j IN 0 TO limite LOOP
                valor:=j;

                FOR k IN B'REVERSE_RANGE LOOP
                    B(k)<=BIT'VAL(valor REM 2);
                    valor:=valor/2;
                END LOOP;
                WAIT FOR 2 ns;

            END LOOP;
        END LOOP;

        WAIT FOR 2 ns;
        WAIT;
    END PROCESS;
END estructural;
```

Resultado del test-bench de la puerta OR:



Modelo de inversor condicional:

```
ENTITY inversor_condicional IS
        GENERIC (n: INTEGER:=8);
        PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r:OUT BIT_VECTOR(n-1
DOWNTO 0));
END inversor_condicional;

ARCHITECTURE estructural OF inversor_condicional IS
```

```
        COMPONENT xor2
                PORT (e1, e2: IN BIT; sal: OUT BIT);
        END COMPONENT;
BEGIN
        bateria: FOR i IN 0 TO n-1 GENERATE
                puertas: xor2 PORT MAP (x(i), y(i), r(i));
        END GENERATE;
END estructural;
```

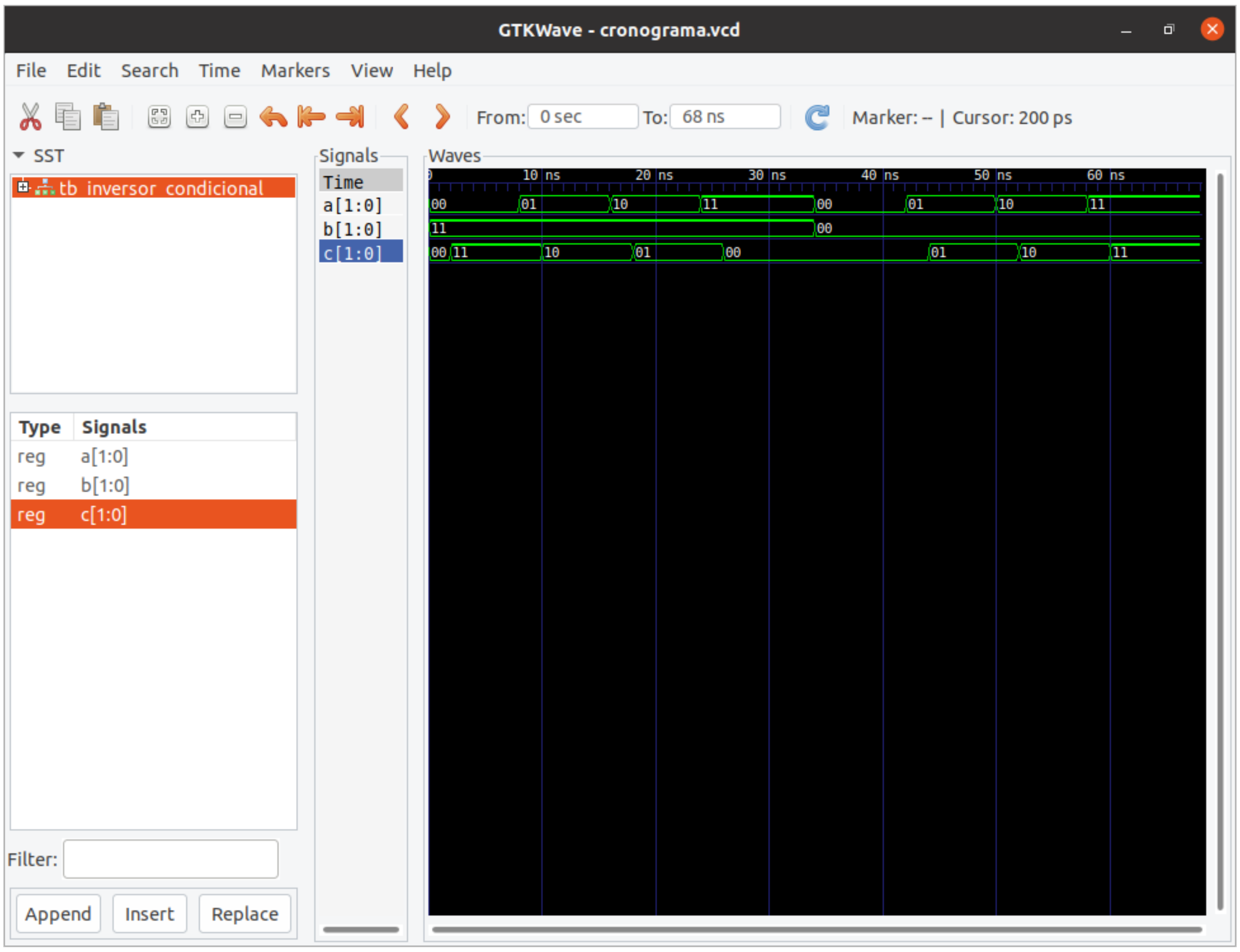
Modelo del test-bench del inversor condicional:

```
ENTITY tb_inversor_condicional IS
END tb_inversor_condicional;

ARCHITECTURE estructural OF tb_inversor_condicional IS
        COMPONENT inversor_condicional
                GENERIC (n: INTEGER:=8);
                PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r: OUT BIT_VECTOR(n-
1 DOWNTO 0));
        END COMPONENT;

        FOR operador: inversor_condicional USE ENTITY
WORK.inversor_condicional(estructural);

        CONSTANT TAMANO_PALABRA: integer := 2;

        SIGNAL A, B, C: BIT_VECTOR(TAMANO_PALABRA-1 DOWNTO 0);

BEGIN
```

```
	operador: inversor_condicional GENERIC MAP (TAMANO_PALABRA) PORT
MAP(A, B, C); --B es la señal ENABLE

	PROCESS
	VARIABLE valor, limite, ENABLE: INTEGER;
	BEGIN
		limite:= 2**TAMANO_PALABRA-1;
		ENABLE:=1;

		FOR i IN 0 TO limite LOOP
			valor:=i;

			FOR k IN A'REVERSE_RANGE LOOP
				A(k)<=BIT'VAL(valor REM 2);
				valor:=valor/2;
			END LOOP;

			FOR j IN 0 TO limite LOOP
				valor:=j;

				FOR k IN B'REVERSE_RANGE LOOP
					B(k)<=BIT'VAL(ENABLE);
					valor:=valor/2;
				END LOOP;
				WAIT FOR 2 ns;

			END LOOP;
		END LOOP;



		WAIT FOR 2 ns;

		ENABLE:=0;
```

```
            FOR i IN 0 TO limite LOOP
                valor:=i;

                FOR k IN A'REVERSE_RANGE LOOP
                    A(k)<=BIT'VAL(valor REM 2);
                    valor:=valor/2;
                END LOOP;

                FOR j IN 0 TO limite LOOP
                    valor:=j;

                    FOR k IN B'REVERSE_RANGE LOOP
                        B(k)<=BIT'VAL(ENABLE);
                        valor:=valor/2;
                    END LOOP;
                    WAIT FOR 2 ns;

                END LOOP;
            END LOOP;



            WAIT FOR 2 ns;
            WAIT;

        END PROCESS;
END estructural;
```

Resultado del test-bench del inversor condicional:


GTKWave - cronograma.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 68 ns
Marker: -- | Cursor: 200 ps
SST
tb inversor condicional
Type Signals
reg a[1:0]
reg b[1:0]
reg c[1:0]
Filter:
Append Insert Replace
Signals
Time
a[1:0]
b[1:0]
c[1:0]
Waves
10 ns 20 ns 30 ns 40 ns 50 ns 60 ns
00 01 10 11 00 01 10 11
11 00
00 11 10 01 00 01 10 11