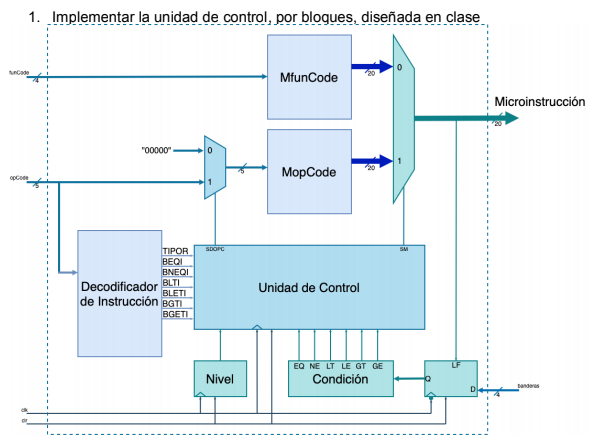
# **Práctica 14 Unidad de Control**





## MfunCode

### Código de implementación

*----------------------------------------------------------------------------------*

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_arith.ALL;

use IEEE.STD\_LOGIC\_unsigned.ALL;

entity CodFunROM is

    generic

    (

*--Tamanio del Bus de Microinstrucciones*

        constant tamWord: integer:=20;

*--Tamanio del Bus de Direcciones*

        constant tamDir: integer:=4

    );

    port

    (

*--Bus de Direcciones*

        BusDir: in std\_logic\_vector(tamDir-1 downto 0);

*--Bus de Microinstrucciones*

        MicroinstruccionFuncion: out std\_logic\_vector(tamWord-1 downto 0)

    );

end CodFunROM;

architecture Behavioral of CodFunROM is

TYPE ROM IS ARRAY (0 to ((2\*\*tamDir)-1)) OF std\_logic\_vector(tamWord-1 downto 0);

*--SDMP,UP,DW,WPC,SR2,SWD,SEXT,SHE,DIR,WR,SOP1,SOP2,ALUOP[3],ALUOP[2],ALUOP[1],ALUOP[0],SDMD,WD,SR,LF*

    constant ADD:    std\_logic\_vector:="00000100010000110010";

    constant SUB:    std\_logic\_vector:="00000100010001110010";

    constant AND\_M:  std\_logic\_vector:="00000100010000000010";

    constant OR\_M:   std\_logic\_vector:="00000100010000010010";

    constant XOR\_M:  std\_logic\_vector:="00000100010000100010";

    constant NAND\_M: std\_logic\_vector:="00000100010011010010";

    constant NOR\_M:  std\_logic\_vector:="00000100010011000010";

    constant XNOR\_M: std\_logic\_vector:="00000100010010100010";

    constant NOT\_M:  std\_logic\_vector:="00000100010011010010";

    constant SLL\_M:  std\_logic\_vector:="00000001110000000000";

    constant SRL\_M:  std\_logic\_vector:="00000001010000000000";

    constant Memoria: ROM:=(

        ADD,            *--00*

        SUB,            *--01*

        AND\_M,      *--02*

        OR\_M,           *--03*

        XOR\_M,      *--04*

        NAND\_M,     *--05*

        NOR\_M,      *--06*

        XNOR\_M,     *--07*

        NOT\_M,      *--08*

        SLL\_M,      *--09*

        SRL\_M,      *--10*

        others=>(others=>'0')

        );

begin

    MicroinstruccionFuncion<=Memoria(conv\_integer(BusDir));

end Behavioral;

### Código de simulación.

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_CodFunROM is

end tb\_CodFunROM;

architecture tb of tb\_CodFunROM is

    component CodFunROM

        port (BusDir                  : in std\_logic\_vector (3 downto 0);

              MicroinstruccionFuncion : out std\_logic\_vector (19 downto 0));

    end component;

    signal BusDir                  : std\_logic\_vector (3 downto 0);

    signal MicroinstruccionFuncion : std\_logic\_vector (19 downto 0);

begin

    dut : CodFunROM

    port map (BusDir                  => BusDir,

             MicroinstruccionFuncion => MicroinstruccionFuncion);

    stimuli : process

    begin

        BusDir <= "1010";

        wait for 100ns;

        BusDir <= "0101";

        wait for 100ns;

*-- EDIT Add stimuli here*

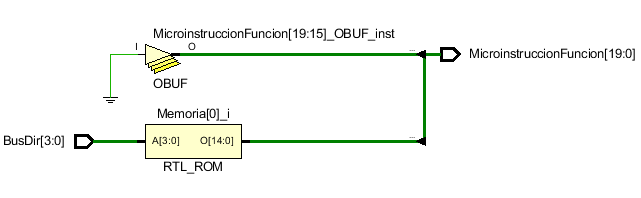
    end process;

end tb;

### Forma de onda de la simulación

### 

### Diagrama RTL



## MopCode

### Código de implementación

*----------------------------------------------------------------------------------*

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_arith.ALL;

use IEEE.STD\_LOGIC\_unsigned.ALL;

entity CodOperROM is

generic

    (

*--Tamanio del Bus de Microinstrucciones*

        constant tamWord: integer:=20;

*--Tamanio del Bus de Direcciones*

        constant tamDir: integer:=5

    );

    port

    (

*--Bus de Direcciones*

        BusDir: in std\_logic\_vector(tamDir-1 downto 0);

*--Bus de Instrucciones*

        MicroInstruccionOP: out std\_logic\_vector(tamWord-1 downto 0)

    );

end CodOperROM;

architecture Behavioral of CodOperROM is

type ROM is array(0 to ((2\*\*tamDir)-1)) of std\_logic\_vector(tamWord-1 downto 0);

*--Microinstruccion*

*--SDMP,UP,DW,WPC,SR2,SWD,SEXT,SHE,DIR,WR,SOP1,SOP2,ALUOP[3],ALUOP[2],ALUOP[1],ALUOP[0],SDMD,WD,SR,LF*

    constant verificacion: std\_logic\_vector:="00001000000001110001";

    constant LI:  std\_logic\_vector:="00000000010000000000";

    constant LWI: std\_logic\_vector:="00000100010000001000";

    constant LW:     std\_logic\_vector:="00000110010100110000";

    constant SWI: std\_logic\_vector:="00001000000000001100";

    constant SW:  std\_logic\_vector:="00001010000100110100";

    constant ADDI:  std\_logic\_vector:="00000100010100110010";

    constant SUBI:  std\_logic\_vector:="00000100010101110010";

    constant ANDI:  std\_logic\_vector:="00000100010100000010";

    constant ORI:   std\_logic\_vector:="00000100010100010010";

    constant XORI:  std\_logic\_vector:="00000100010100100010";

    constant NANDI: std\_logic\_vector:="00000100010111010010";

    constant NORI:  std\_logic\_vector:="00000100010111000010";

    constant XNORI: std\_logic\_vector:="00000100010110100010";

    constant salto: std\_logic\_vector:="10010000001100110010";

    constant B:      std\_logic\_vector:="00010000000000000000";

    constant CALL:  std\_logic\_vector:="01010000000000000000";

    constant RET:    std\_logic\_vector:="00100000000000000000";

    constant NOP:    std\_logic\_vector(tamWord-1 downto 0):=(others=>'0');

    constant Memoria: ROM:=(

            verificacion,   *--00*

            LI,             *--01*

            LWI,                *--02*

            SWI,                *--03*

            SW,             *--04*

            ADDI,               *--05*

            SUBI,               *--06*

            ANDI,               *--07*

            ORI,                *--08*

            XORI,               *--09*

            NANDI,          *--10*

            NORI,               *--11*

            XNORI,          *--12*

            salto,          *--13*

            salto,          *--14*

            salto,          *--15*

            salto,          *--16*

            salto,          *--17*

            salto,          *--18*

            B,                  *--19*

            CALL,               *--20*

            RET,                *--21*

            NOP,                *--22*

            LW,             *--23*

            others=>(others=>'0')

            );

begin

MicroInstruccionOP<=Memoria(conv\_integer(BusDir));

end Behavioral;

### Código de simulación

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_CodOperROM is

end tb\_CodOperROM;

architecture tb of tb\_CodOperROM is

    component CodOperROM

        port (BusDir             : in std\_logic\_vector (4 downto 0);

              MicroInstruccionOP : out std\_logic\_vector (19 downto 0));

    end component;

    signal BusDir             : std\_logic\_vector (4 downto 0);

    signal MicroInstruccionOP : std\_logic\_vector (19 downto 0);

begin

    dut : CodOperROM

    port map (BusDir             => BusDir,

              MicroInstruccionOP => MicroInstruccionOP);

    stimuli : process

    begin

       BusDir <= "00011";

       wait for 100ns;

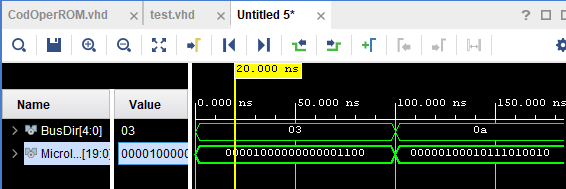
       BusDir <= "01010";

       wait for 100ns;

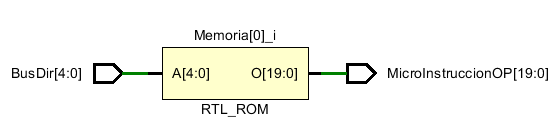
    end process;

end tb;

### Forma de onda de la simulación



### Diagrama RTL



## Decodificador de instrucción

### Código de implementación

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity DecoIns is

port

    (

        OP\_CODE: in std\_logic\_vector(4 downto 0);

        TIPOR:  out std\_logic;

        BEQI:   out std\_logic;

        BNEI:   out std\_logic;

        BLTI:   out std\_logic;

        BLETI:  out std\_logic;

        BGTI:   out std\_logic;

        BGETI:  out std\_logic

    );

end DecoIns;

architecture Behavioral of DecoIns is

begin

    TIPOR   <= '1' when OP\_CODE="00000" else '0';*--00*

    BEQI    <= '1' when OP\_CODE="01101" else '0';*--13*

    BNEI    <= '1' when OP\_CODE="01110" else '0';*--14*

    BLTI    <= '1' when OP\_CODE="01111" else '0';*--15*

    BLETI   <= '1' when OP\_CODE="10000" else '0';*--16*

    BGTI    <= '1' when OP\_CODE="10001" else '0';*--17*

    BGETI   <= '1' when OP\_CODE="10010" else '0';*--18*

end Behavioral;

### Código de simulación

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_DecoIns is

end tb\_DecoIns;

architecture tb of tb\_DecoIns is

    component DecoIns

        port (OP\_CODE : in std\_logic\_vector (4 downto 0);

              TIPOR   : out std\_logic;

              BEQI    : out std\_logic;

              BNEI    : out std\_logic;

              BLTI    : out std\_logic;

              BLETI   : out std\_logic;

              BGTI    : out std\_logic;

              BGETI   : out std\_logic);

    end component;

    signal OP\_CODE : std\_logic\_vector (4 downto 0);

    signal TIPOR   : std\_logic;

    signal BEQI    : std\_logic;

    signal BNEI    : std\_logic;

    signal BLTI    : std\_logic;

    signal BLETI   : std\_logic;

    signal BGTI    : std\_logic;

    signal BGETI   : std\_logic;

begin

    dut : DecoIns

    port map (OP\_CODE => OP\_CODE,

              TIPOR   => TIPOR,

              BEQI    => BEQI,

              BNEI    => BNEI,

              BLTI    => BLTI,

              BLETI   => BLETI,

              BGTI    => BGTI,

              BGETI   => BGETI);

    stimuli : process

    begin

        OP\_CODE <= "00000";

        wait for 100ns;

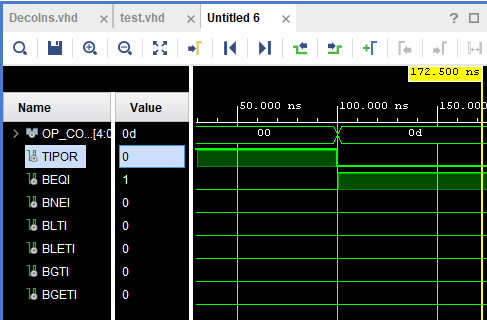
        OP\_CODE <= "01101";

        wait for 100ns;

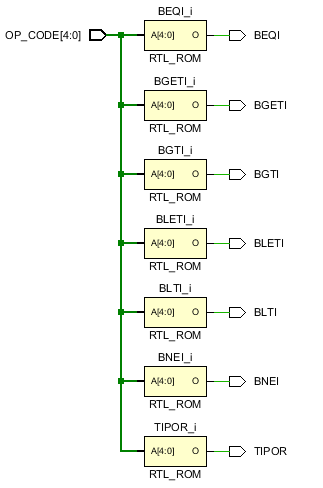
end process;

end tb;

### Forma de onda de la simulación



### Diagrama RTL



## Nivel

### Código de implementación

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_arith.ALL;

use IEEE.STD\_LOGIC\_unsigned.ALL;

entity DetectorNivel is

    port

    (

        clk: in std\_logic;

        clr: in std\_logic;

        NA: out std\_logic

    );

end DetectorNivel;

architecture Behavioral of DetectorNivel is

    signal pclk1,nclk1: std\_logic;

begin

    process(clk,clr)

    begin

        if (clr='1') then

            pclk1<='1';

        elsif (rising\_edge(clk)) then

            pclk1<=(not pclk1);

        end if;

    end process;

    process(clk,clr)

    begin

        if (clr='1') then

            nclk1<='0';

        elsif (falling\_edge(clk)) then

            nclk1<=(not nclk1);

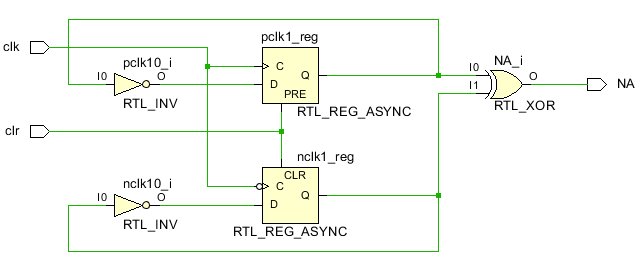
        end if;

    end process;

    NA <= (pclk1 xor nclk1);

end Behavioral;

### Diagrama RTL



## Verificación de condición

### Código de implementación

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity VerifCond is

port

    (

*--Banderas de la ALU*

        OV: in std\_logic;

        Z: in std\_logic;

        N: in std\_logic;

        C: in std\_logic;

        EQ: out std\_logic;

        NEQ: out std\_logic;

        LT: out std\_logic;

        LET: out std\_logic;

        GT: out std\_logic;

        GET: out std\_logic

    );

end VerifCond;

architecture Behavioral of VerifCond is

begin

*--Igualdad*

        EQ<=Z;

*--Desigualdad*

        NEQ<=(not Z);

*--Menor*

        LT<=(not C);

*--Menor o igual*

        LET<=(Z or (not C));

*--Mayor*

        GT<=(C and (not Z));

*--Mayor o igual*

        GET<=C;

end Behavioral;

### Código de simulación

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_VerifCond is

end tb\_VerifCond;

architecture tb of tb\_VerifCond is

    component VerifCond

        port (OV  : in std\_logic;

              Z   : in std\_logic;

              N   : in std\_logic;

              C   : in std\_logic;

              EQ  : out std\_logic;

              NEQ : out std\_logic;

              LT  : out std\_logic;

              LET : out std\_logic;

              GT  : out std\_logic;

              GET : out std\_logic);

    end component;

    signal OV  : std\_logic;

    signal Z   : std\_logic;

    signal N   : std\_logic;

    signal C   : std\_logic;

    signal EQ  : std\_logic;

    signal NEQ : std\_logic;

    signal LT  : std\_logic;

    signal LET : std\_logic;

    signal GT  : std\_logic;

    signal GET : std\_logic;

begin

    dut : VerifCond

    port map (OV  => OV,

              Z   => Z,

              N   => N,

              C   => C,

              EQ  => EQ,

              NEQ => NEQ,

              LT  => LT,

              LET => LET,

              GT  => GT,

              GET => GET);

    stimuli : process

    begin

*-- EDIT Adapt initialization as needed*

        OV <= '0';

        Z <= '1';

        N <= '0';

        C <= '0';

        wait for 100ns;

        OV <= '1';

        Z <= '0';

        N <= '1';

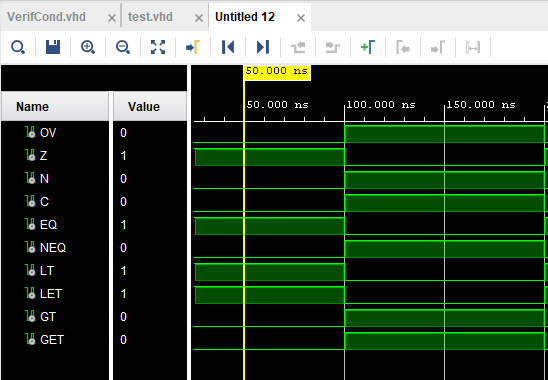
        C <= '1';

        wait for 100ns;

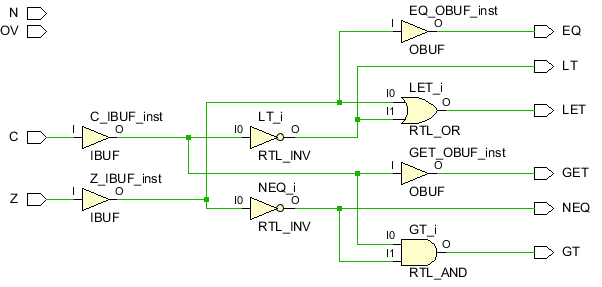
    end process;

end tb;

### Forma de onda de la señal



### Diagrama RTL



## Registro de Banderas

### Código de implementación

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity RegFlags is

port

    (

*--Banderas de la ALU*

        OV: in std\_logic;

        N:  in std\_logic;

        Z:  in std\_logic;

        C:  in std\_logic;

        LF:  in std\_logic;

        clk: in std\_logic;

        clr: in std\_logic;

        OV\_out: inout std\_logic;

        N\_out:  inout std\_logic;

        Z\_out:  inout std\_logic;

        C\_out:  inout std\_logic

    );

end RegFlags;

architecture Behavioral of RegFlags is

begin

process (clk,clr)

    begin

        if (clr='1') then

            OV\_out<='0';

            N\_out<='0';

            Z\_out<='0';

            C\_out<='0';

        elsif (falling\_edge(clk)) then

            if (LF='1') then

                OV\_out<=OV;

                N\_out<=N;

                Z\_out<=Z;

                C\_out<=C;

            else

                OV\_out<=OV\_out;

                N\_out<=N\_out;

                Z\_out<=Z\_out;

                C\_out<=C\_out;

            end if;

        end if;

    end process;

end Behavioral;

Código de simulación

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_RegFlags is

end tb\_RegFlags;

architecture tb of tb\_RegFlags is

    component RegFlags

        port (OV     : in std\_logic;

              N      : in std\_logic;

              Z      : in std\_logic;

              C      : in std\_logic;

              LF     : in std\_logic;

              clk    : in std\_logic;

              clr    : in std\_logic;

              OV\_out : inout std\_logic;

              N\_out  : inout std\_logic;

              Z\_out  : inout std\_logic;

              C\_out  : inout std\_logic);

    end component;

    signal OV     : std\_logic;

    signal N      : std\_logic;

    signal Z      : std\_logic;

    signal C      : std\_logic;

    signal LF     : std\_logic;

    signal clk    : std\_logic;

    signal clr    : std\_logic;

    signal OV\_out : std\_logic;

    signal N\_out  : std\_logic;

    signal Z\_out  : std\_logic;

    signal C\_out  : std\_logic;

    constant TbPeriod : time := 1000 ns; *-- EDIT Put right period here*

    signal TbClock : std\_logic := '0';

    signal TbSimEnded : std\_logic := '0';

begin

    dut : RegFlags

    port map (OV     => OV,

              N      => N,

              Z      => Z,

              C      => C,

              LF     => LF,

              clk    => clk,

              clr    => clr,

              OV\_out => OV\_out,

              N\_out  => N\_out,

              Z\_out  => Z\_out,

              C\_out  => C\_out);

*-- Clock generation*

    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';

    clk <= TbClock;

    stimuli : process

    begin

        OV <= '0';

        N <= '0';

        Z <= '1';

        C <= '1';

        LF <= '1';

        clr <= '0';

        wait for 100 \* TbPeriod;

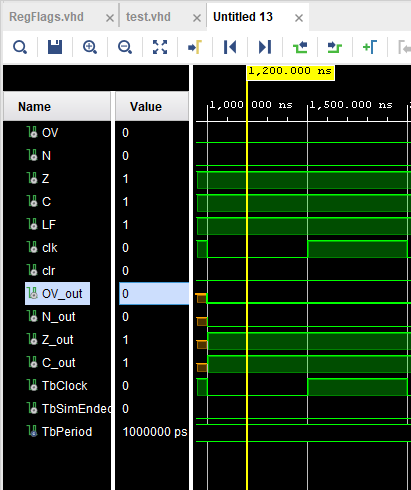
        TbSimEnded <= '1';

        wait;

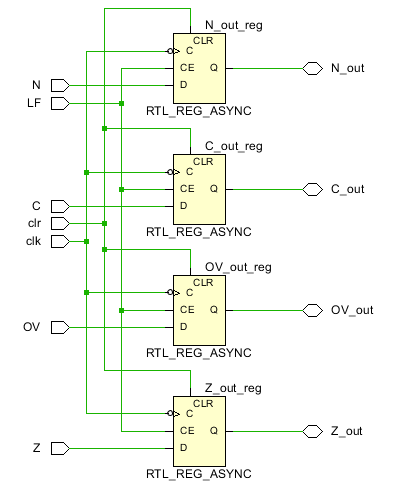
    end process;

end tb;

### Forma de onda de la señal



### Diagrama RTL



## Unidad de Control

### Código de implementación

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity UC is

port

    (

        TIPOR: in std\_logic;

        BEQI: in std\_logic;

        BNEI: in std\_logic;

        BLTI: in std\_logic;

        BLETI: in std\_logic;

        BGTI: in std\_logic;

        BGETI: in std\_logic;

        NA: in std\_logic;

        EQ: in std\_logic;

        NEQ: in std\_logic;

        LT: in std\_logic;

        LET: in std\_logic;

        GT: in std\_logic;

        GET: in std\_logic;

        SDOPC: out std\_logic;

        SM: out std\_logic

    );

end UC;

architecture Behavioral of UC is

begin

    SDOPC<=(

                (

                    (not NA) and

                        (

                            (BEQI  and EQ ) or

                            (BNEI  and NEQ) or

                            (BLTI  and LT ) or

                            (BLETI and LET) or

                            (BGTI  and GT ) or

                            (BGETI and GET)

                        )

                ) or (not (TIPOR or BEQI or BNEI or BLTI or BLETI or BGTI or BGETI))

            );

    SM<=(not TIPOR);

end Behavioral;

### Código de simulación

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_UC is

end tb\_UC;

architecture tb of tb\_UC is

    component UC

        port (TIPOR : in std\_logic;

              BEQI  : in std\_logic;

              BNEI  : in std\_logic;

              BLTI  : in std\_logic;

              BLETI : in std\_logic;

              BGTI  : in std\_logic;

              BGETI : in std\_logic;

              NA    : in std\_logic;

              EQ    : in std\_logic;

              NEQ   : in std\_logic;

              LT    : in std\_logic;

              LET   : in std\_logic;

              GT    : in std\_logic;

              GET   : in std\_logic;

              SDOPC : out std\_logic;

              SM    : out std\_logic);

    end component;

    signal TIPOR : std\_logic;

    signal BEQI  : std\_logic;

    signal BNEI  : std\_logic;

    signal BLTI  : std\_logic;

    signal BLETI : std\_logic;

    signal BGTI  : std\_logic;

    signal BGETI : std\_logic;

    signal NA    : std\_logic;

    signal EQ    : std\_logic;

    signal NEQ   : std\_logic;

    signal LT    : std\_logic;

    signal LET   : std\_logic;

    signal GT    : std\_logic;

    signal GET   : std\_logic;

    signal SDOPC : std\_logic;

    signal SM    : std\_logic;

    constant TbPeriod : time := 1000 ns; *-- EDIT Put right period here*

    signal TbClock : std\_logic := '0';

    signal TbSimEnded : std\_logic := '0';

begin

    dut : UC

    port map (TIPOR => TIPOR,

              BEQI  => BEQI,

              BNEI  => BNEI,

              BLTI  => BLTI,

              BLETI => BLETI,

              BGTI  => BGTI,

              BGETI => BGETI,

              NA    => NA,

              EQ    => EQ,

              NEQ   => NEQ,

              LT    => LT,

              LET   => LET,

              GT    => GT,

              GET   => GET,

              SDOPC => SDOPC,

              SM    => SM);

*-- Clock generation*

    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';

*--  EDIT: Replace YOURCLOCKSIGNAL below by the name of your clock as I haven't guessed it*

*--  YOURCLOCKSIGNAL <= TbClock;*

    stimuli : process

    begin

*-- EDIT Adapt initialization as needed*

        TIPOR <= '0';

        BEQI <= '0';

        BNEI <= '0';

        BLTI <= '0';

        BLETI <= '0';

        BGTI <= '0';

        BGETI <= '0';

        NA <= '0';

        EQ <= '1';

        NEQ <= '0';

        LT <= '0';

        LET <= '0';

        GT <= '0';

        GET <= '0';

*-- EDIT Add stimuli here*

        wait for 100 \* TbPeriod;

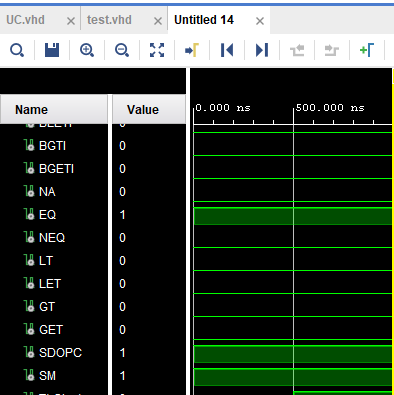
*-- Stop the clock and hence terminate the simulation*

        TbSimEnded <= '1';

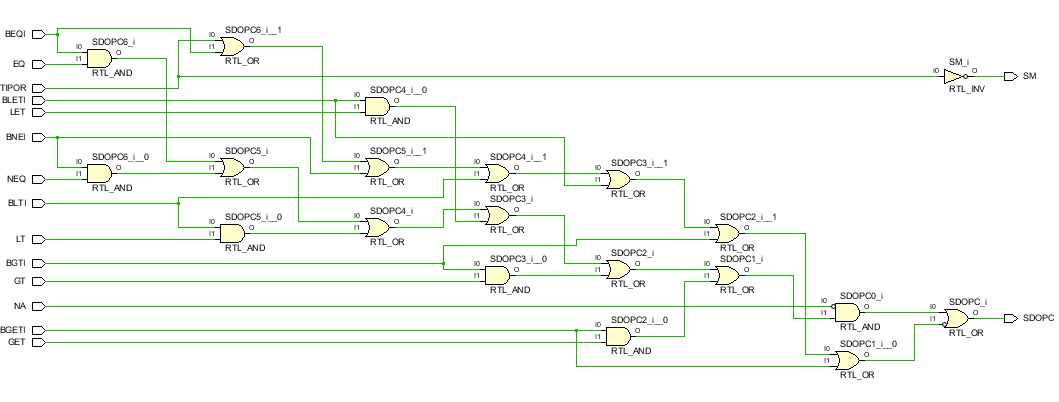
        wait;

    end process;

### Forma de onda de la señal



### Diagrama RTL



## Código Unidad de Control Main

### Código de implementación

library IEEE;

library WORK;

use IEEE.STD\_LOGIC\_1164.ALL;

use WORK.PackUP.all;

entity Main is

    generic

    (

        constant tam\_Microinstruccion:integer:=20;

        constant tam\_OPCODE: integer:=5;

        constant tam\_FUNCIONCODE: integer:=4

    );

    port

    (

*--Banderas de la ALU*

        OV:  in std\_logic;

        N:   in std\_logic;

        Z:   in std\_logic;

        C:   in std\_logic;

        LF:  in std\_logic;

        clk: in std\_logic;

        clr: in std\_logic;

        OP\_CODE:    in std\_logic\_vector(tam\_OPCODE-1 downto 0);

        FUN\_CODE:   in std\_logic\_vector(tam\_FUNCIONCODE-1 downto 0);

        NA\_out: out std\_logic;

        SDOPC\_out: out std\_logic;

        SM\_out: out std\_logic;

        MicroInstruccion: out std\_logic\_vector(tam\_Microinstruccion-1 downto 0)

    );

end Main;

architecture Behavioral of Main is

constant ceros:                         std\_logic\_vector(tam\_OPCODE-1 downto 0):=(others=>'0');

    signal OPCODE\_SDOPC:                std\_logic\_vector(tam\_OPCODE-1 downto 0);

    signal MicroInstruccionOP:      std\_logic\_vector(tam\_Microinstruccion-1 downto 0);

    signal MicroInstruccionFuncion:     std\_logic\_vector(tam\_Microinstruccion-1 downto 0);

    signal SDOPC:   std\_logic;

    signal SM:      std\_logic;

    signal TIPOR:   std\_logic;

    signal BEQI:    std\_logic;

    signal BNEI:    std\_logic;

    signal BLTI:    std\_logic;

    signal BLETI:   std\_logic;

    signal BGTI:    std\_logic;

    signal BGETI:   std\_logic;

    signal NA:      std\_logic;

    signal EQ:      std\_logic;

    signal NEQ:     std\_logic;

    signal LT:      std\_logic;

    signal LET:     std\_logic;

    signal GT:      std\_logic;

    signal GET:     std\_logic;

    signal OV1: std\_logic;

    signal Z1:  std\_logic;

    signal C1:  std\_logic;

    signal N1:  std\_logic;

begin

    Banderas: RegFlags

        port map

        (

            OV=>OV,

            N=>N,

            Z=>Z,

            C=>C,

            LF=>LF,

            clk=>clk,

            clr=>clr,

            OV\_out=>OV1,

            N\_out=>N1,

            Z\_out=>Z1,

            C\_out=>C1

        );

    Condicion: VerifCond

        port map

        (

            OV=>OV1,

            Z=>Z1,

            N=>N1,

            C=>C1,

            EQ=>EQ,

            NEQ=>NEQ,

            LT=>LT,

            LET=>LET,

            GT=>GT,

            GET=>GET

        );

    Nivel: DetectorNivel

        port map

        (

            clk=>clk,

            clr=>clr,

            NA=>NA

        );

        Deco: DecoIns

        port map

        (

            OP\_CODE=>OP\_CODE,

            TIPOR=>TIPOR,

            BEQI=>BEQI,

            BNEI=>BNEI,

            BLTI=>BLTI,

            BLETI=>BLETI,

            BGTI=>BGTI,

            BGETI=>BGETI

        );

        UnidadControl: UC

        port map

        (

            TIPOR=>TIPOR,

            BEQI=>BEQI,

            BNEI=>BNEI,

            BLTI=>BLTI,

            BLETI=>BLETI,

            BGTI=>BGTI,

            BGETI=>BGETI,

            NA=>NA,

            EQ=>EQ,

            NEQ=>NEQ,

            LT=>LT,

            LET=>LET,

            GT=>GT,

            GET=>GET,

            SDOPC=>SDOPC,

            SM=>SM

        );

        OP\_CODE\_ROM: CodOperROM

        port map

        (

            BusDir=>OPCODE\_SDOPC,

            MicroInstruccionOP=>MicroInstruccionOP

        );

        FUNCION\_CODE\_ROM: CodFunROM

        port map

        (

            BusDir=>FUN\_CODE,

            MicroinstruccionFuncion=>MicroinstruccionFuncion

        );

        OPCODE\_SDOPC<=OP\_CODE when SDOPC='1' else ceros;

        Microinstruccion<=MicroinstruccionOP when SM='1' else MicroinstruccionFuncion;

        NA\_out<=NA;

        SDOPC\_out<=SDOPC;

        SM\_out<=SM;

end Behavioral;

### Código de simulación

LIBRARY ieee;

LIBRARY STD;

USE IEEE.std\_logic\_1164.ALL;

USE IEEE.std\_logic\_textio.ALL;

USE STD.textio.ALL;

ENTITY TBUC IS

END TBUC;

ARCHITECTURE behavior OF TBUC IS

*-- Component Declaration for the Unit Under Test (UUT)*

    COMPONENT Main

    PORT(

         OV : IN  std\_logic;

         N : IN  std\_logic;

         Z : IN  std\_logic;

         C : IN  std\_logic;

         LF : IN  std\_logic;

         clk : IN  std\_logic;

         clr : IN  std\_logic;

         OP\_CODE : IN  std\_logic\_vector(4 downto 0);

         FUN\_CODE : IN  std\_logic\_vector(3 downto 0);

         NA\_out : OUT  std\_logic;

         SDOPC\_out : OUT  std\_logic;

         SM\_out : OUT  std\_logic;

         MicroInstruccion : OUT  std\_logic\_vector(19 downto 0)

        );

    END COMPONENT;

*--Inputs*

   signal OV : std\_logic := '0';

   signal N : std\_logic := '0';

   signal Z : std\_logic := '0';

   signal C : std\_logic := '0';

   signal LF : std\_logic := '0';

   signal clk : std\_logic := '0';

   signal clr : std\_logic := '0';

   signal OP\_CODE : std\_logic\_vector(4 downto 0) := (others => '0');

   signal FUN\_CODE : std\_logic\_vector(3 downto 0) := (others => '0');

*--Outputs*

   signal NA\_out : std\_logic;

   signal SDOPC\_out : std\_logic;

   signal SM\_out : std\_logic;

   signal MicroInstruccion : std\_logic\_vector(19 downto 0);

*-- Clock period definitions*

   constant clk\_period : time := 10 ns;

BEGIN

*-- Instantiate the Unit Under Test (UUT)*

   uut: Main PORT MAP (

          OV => OV,

          N => N,

          Z => Z,

          C => C,

          LF => LF,

          clk => clk,

          clr => clr,

          OP\_CODE => OP\_CODE,

          FUN\_CODE => FUN\_CODE,

          NA\_out => NA\_out,

          SDOPC\_out => SDOPC\_out,

          SM\_out => SM\_out,

          MicroInstruccion => MicroInstruccion

        );

*-- Clock process definitions*

   clk\_process :process

   begin

        clk <= '0';

        wait for clk\_period/2;

        clk <= '1';

        wait for clk\_period/2;

   end process;

*-- Stimulus process*

   stim\_proc: process

*--Entradas*

        FILE archivoEntrada      : TEXT;

        variable lineaEntrada : LINE;

        variable var\_flags  : std\_logic\_vector(3 downto 0);

        variable var\_LF         : std\_logic;

        variable var\_clr        : std\_logic;

        variable var\_OP\_CODE        : std\_logic\_vector(4 downto 0);

        variable var\_FUN\_CODE   : std\_logic\_vector(3 downto 0);

*--Salidas*

        FILE archivoSalida  : TEXT;

        variable lineaSalida : LINE;

        variable var\_MicroInstruccion : std\_logic\_vector(19 downto 0);

        variable var\_NA: std\_logic;

        variable var\_SDOPC: std\_logic;

        variable var\_SM: std\_logic;

        constant E\_OP\_CODE      : STRING:="Op\_Code";

        constant E\_FUN\_CODE     : STRING:="Fun\_Code";

        constant E\_FLAGS        : STRING:="Banderas";

        constant E\_CLR      : STRING:="CLR";

        constant E\_LF           : STRING:="LF";

        constant E\_NIVEL        : STRING:="Nivel";

        constant E\_ALTO         : STRING:="ALTO";

        constant E\_BAJO         : STRING:="BAJO";

        constant E\_NA           : STRING:="NA";

        constant E\_SDOPC        : STRING:="SDOPC";

        constant E\_SM           : STRING:="SM";

        constant E\_Microinstruccion : STRING:="Microinstruccion";

    begin

        FILE\_OPEN (archivoEntrada, "entradas.txt", READ\_MODE);

        FILE\_OPEN (archivoSalida,"salidas.txt",WRITE\_MODE);

        WRITE (lineaSalida, E\_OP\_CODE, RIGHT, E\_OP\_CODE'LENGTH+1);

        WRITE (lineaSalida, E\_FUN\_CODE, RIGHT, E\_FUN\_CODE'LENGTH+1);

        WRITE (lineaSalida, E\_FLAGS, RIGHT, E\_FLAGS'LENGTH+1);

        WRITE (lineaSalida, E\_CLR, RIGHT, E\_CLR'LENGTH+1);

        WRITE (lineaSalida, E\_LF, RIGHT, E\_LF'LENGTH+1);

        WRITE (lineaSalida, E\_Microinstruccion, RIGHT, var\_Microinstruccion'LENGTH+1);

        WRITE (lineaSalida, E\_NIVEL, RIGHT, E\_NIVEL'LENGTH+1);

        WRITELINE (archivoSalida, lineaSalida);

        clr<='1';*--este daba lata y los clears del archivo*

        wait for 12 ns;

        clr<='0';

        WHILE NOT ENDFILE (archivoEntrada) LOOP

            WAIT UNTIL RISING\_EDGE ( clk );

            READLINE (archivoEntrada, lineaEntrada);

            READ(lineaEntrada, var\_OP\_CODE);

            OP\_CODE <= var\_OP\_CODE;

            READ(lineaEntrada, var\_FUN\_CODE);

            FUN\_CODE <= var\_FUN\_CODE;

            READ(lineaEntrada, var\_FLAGS);

            OV <= var\_FLAGS(3);

            N  <= var\_FLAGS(2);

            Z  <= var\_FLAGS(1);

            C  <= var\_FLAGS(0);

            READ(lineaEntrada, var\_CLR);

            clr<=var\_CLR;

            READ(lineaEntrada, var\_LF);

            LF<=var\_LF;

            WAIT FOR 4 NS;

            var\_Microinstruccion := Microinstruccion;

*--Se escriben entradas*

            WRITE(lineaSalida, var\_OP\_CODE, RIGHT, E\_OP\_CODE'LENGTH+1);

            WRITE(lineaSalida, var\_FUN\_CODE, RIGHT, E\_FUN\_CODE'LENGTH+1);

            WRITE(lineaSalida, var\_FLAGS, RIGHT, E\_FLAGS'LENGTH+1);

            WRITE(lineaSalida, var\_CLR, RIGHT, E\_CLR'LENGTH+1);

            WRITE(lineaSalida, var\_LF, RIGHT, E\_LF'LENGTH+1);

*--Se escriben salidas*

            WRITE(lineaSalida, var\_Microinstruccion, RIGHT, Microinstruccion'LENGTH+1);

            WRITE(lineaSalida, E\_ALTO, RIGHT, E\_NIVEL'LENGTH+1);

            WRITELINE(archivoSalida, lineaSalida);

            WAIT UNTIL FALLING\_EDGE ( clk );

            wait for 4 ns;

            var\_Microinstruccion := Microinstruccion;

*--Se escriben entradas*

            WRITE(lineaSalida, var\_OP\_CODE, RIGHT, E\_OP\_CODE'LENGTH+1);

            WRITE(lineaSalida, var\_FUN\_CODE, RIGHT, E\_FUN\_CODE'LENGTH+1);

            WRITE(lineaSalida, var\_FLAGS, RIGHT, E\_FLAGS'LENGTH+1);

            WRITE(lineaSalida, var\_CLR, RIGHT, E\_CLR'LENGTH+1);

            WRITE(lineaSalida, var\_LF, RIGHT, E\_LF'LENGTH+1);

*--Se escriben salidas*

            WRITE(lineaSalida, var\_Microinstruccion, RIGHT, Microinstruccion'LENGTH+1);

            WRITE(lineaSalida, E\_BAJO, RIGHT, E\_NIVEL'LENGTH+1);

            WRITELINE(archivoSalida, lineaSalida);

        END LOOP;

*--Se cierran archivos de texto*

        FILE\_CLOSE(archivoEntrada);

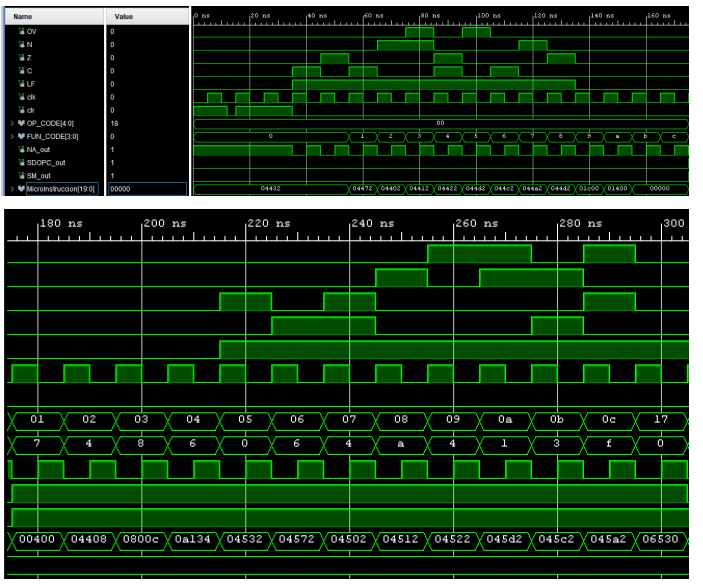
        FILE\_CLOSE(archivoSalida);

      wait;

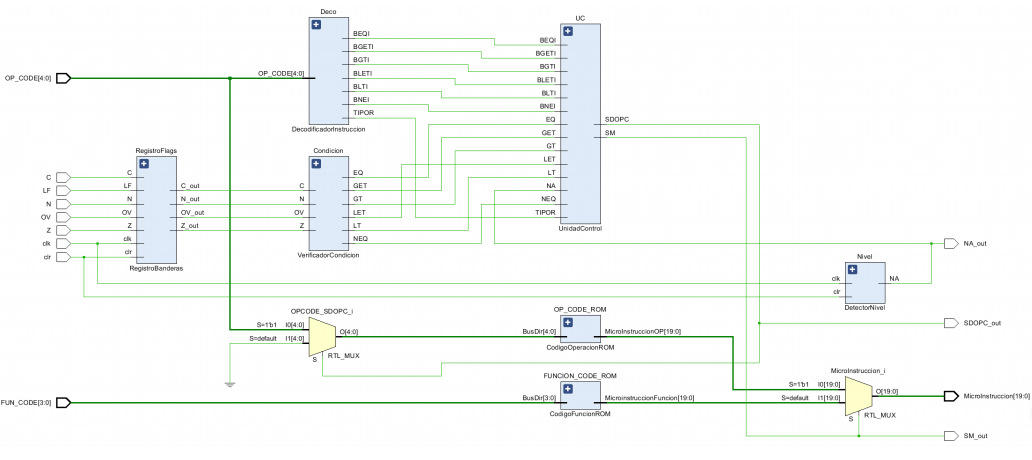
   end process;

END;

### Forma de señal de salida



### Diagrama RTL



### Salidas

