# Trabajo 16

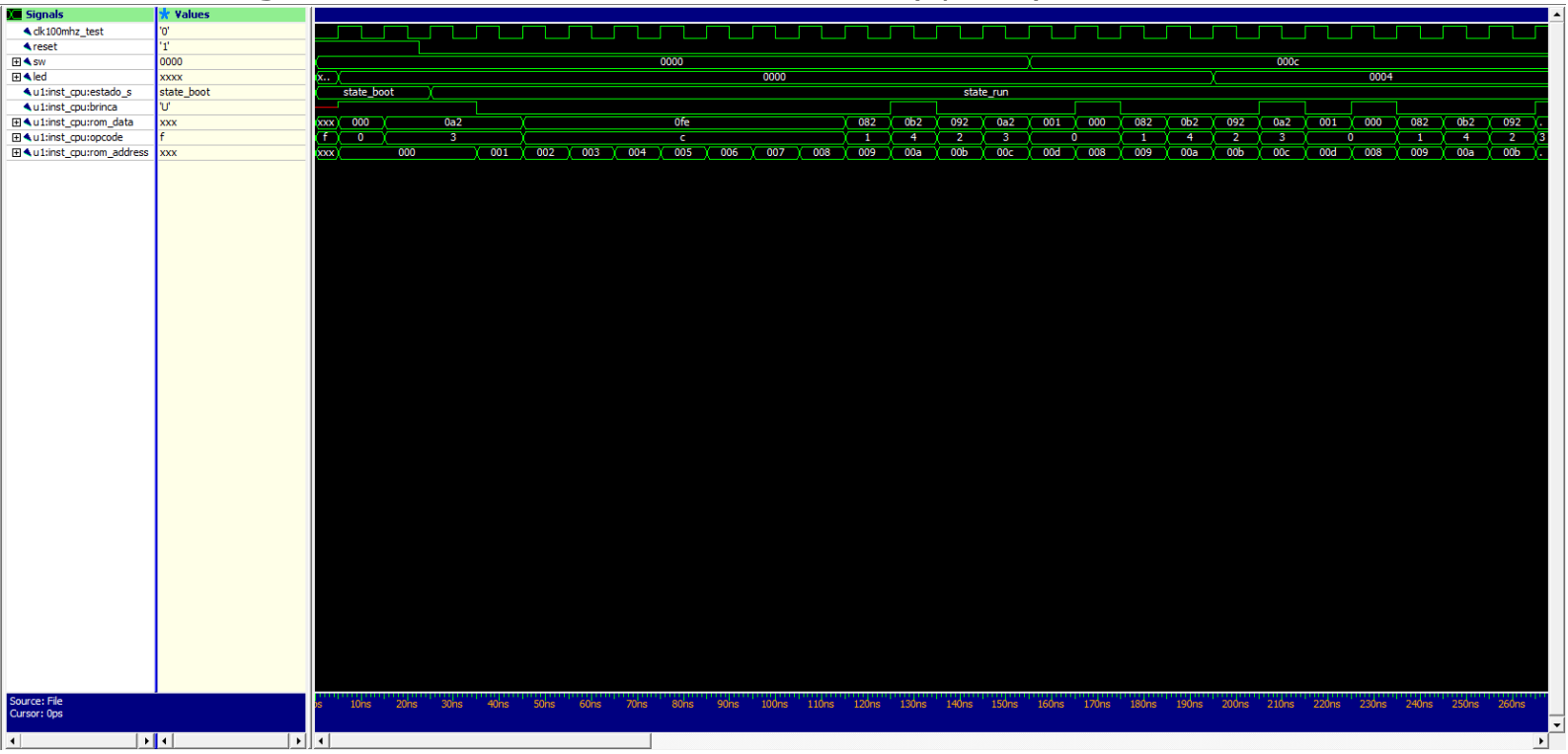
## Diseño e implementación de un I2C maestro

## Miembros del grupo:

- Martínez Castaño, Carlos carlos.martinez.castano18@gmail.com
- Pérez de León de los Santos, Ibrahim ibraaps@gmail.com

## Índice

## 1. Programación de instrucciones básicas (1punto)




En este ejercicio se han realizado las siguientes modificaciones en el archivo cpu.vhd:

```
--                  IMPORTANTE     !!!!!!!!!!!!!!!!
-- Hay que a adir l neas por cada instrucci n nueva que se a ada al dise o
opcode <= OPCODE_SALTA           when ROM_data(8 downto 7) = "00"        else
          OPCODE_BRINCA_SI_1     when ROM_data(8 downto 4) = "01001"     else
          OPCODE_PIN_1           when ROM_data(8 downto 4) = "01011"     else
          OPCODE_BRINCA_SI_0     when ROM_data(8 downto 4) = "01000"     else
          OPCODE_PIN_0           when ROM_data(8 downto 4) = "01010"     else
          OPCODE_NOP             when ROM_data(8 downto 0) = "011111110" else
          OPCODE_UNKNOWN;
```

```
when OPCODE_BRINCA_SI_0 =>
   -- Salta una instruccion si el pin de entrada n es '0' -> sw(n)= '0'
   brinca <='0';
   if sw( to_integer(unsigned(ROM_data(3 downto 0))) )='0' then
      brinca <='1';
   end if;
   pcnext <= pcnext+1;
```

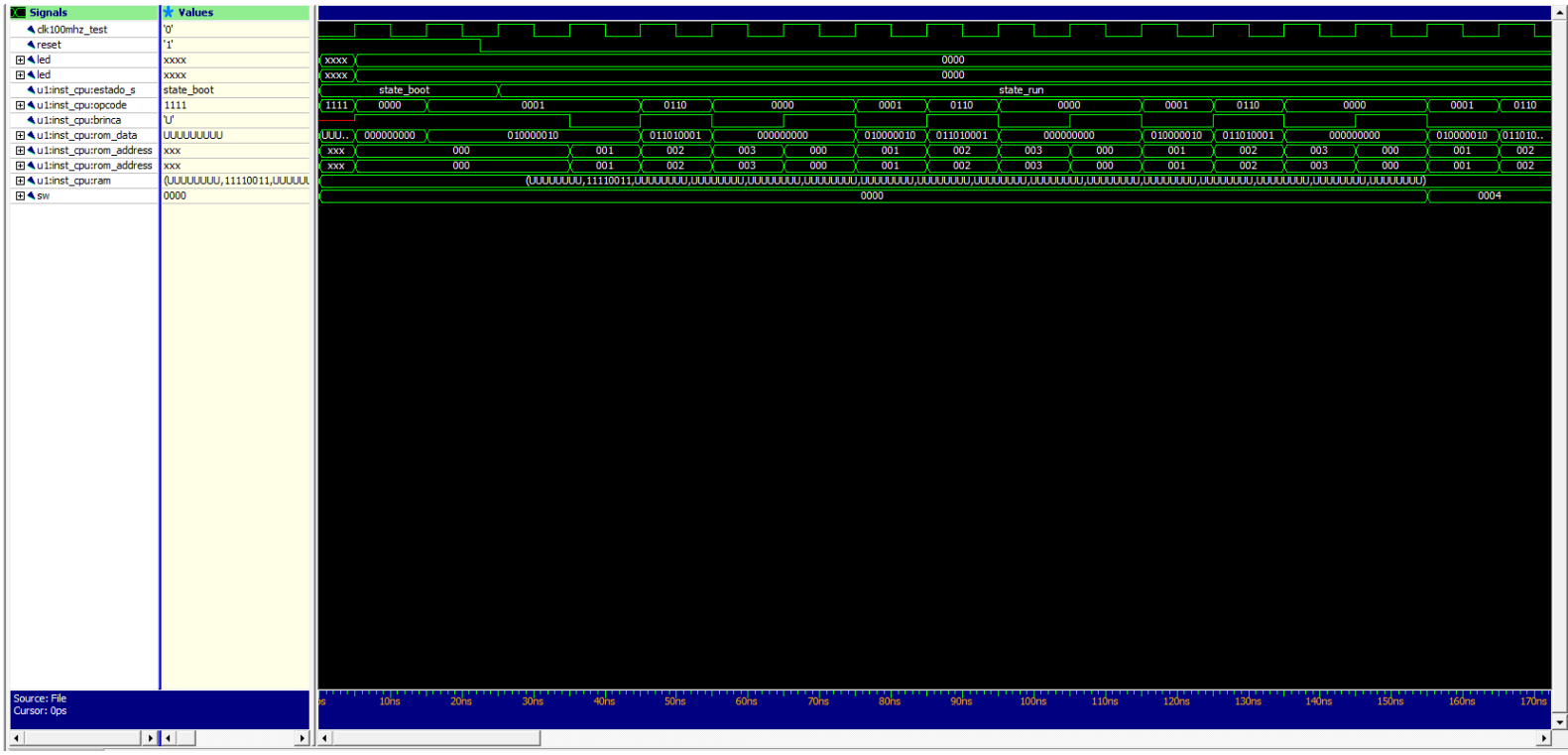
```
when OPCODE_PIN_0 =>
   -- La salida con la posici n de 0-15 [ROM_data(3 downto 0)] la pones a 0 (ROM_data(4)
   -- Ej.: led(5) <= '0'
   led(to_integer(unsigned(ROM_data(3 downto 0)))) <= ROM_data(4);
   pcnext <= pcnext+1;
```

```
when OPCODE_NOP =>
   -- Esta instruccion no hace nada. Deja pasar un ciclo de reloj e incrementa el
   -- contador de programa para que apunte a la siguiente instruccion
   pcnext <= pcnext+1;
```

Primero añadimos el código que identifica cada operación, en éste caso los códigos correspondientes a las operaciones brincaSi0, pin0 y nop. Posteriormente hemos programado cada operación de la siguiente manera:

- brincaSi0: compara si el pin de entrada es igual a '0' y si cumple está condición asigna a 1 la variable brinca, lo que significa que la siguiente instrucción se ignorará. Por último incrementa el contador de programa.
- pin0: se asigna a '0' a la salida con la posición que se le indique en el ensamblador. Incrementa el contador de programa.
- nop: solo incrementa el contador de programa, esta operación no realiza nada más.

## 2. Programación de la instrucción 'lee'. (1punto)




```
OPCODE_LEE              when ROM_data(8 downto 4) = "01101"     else
```

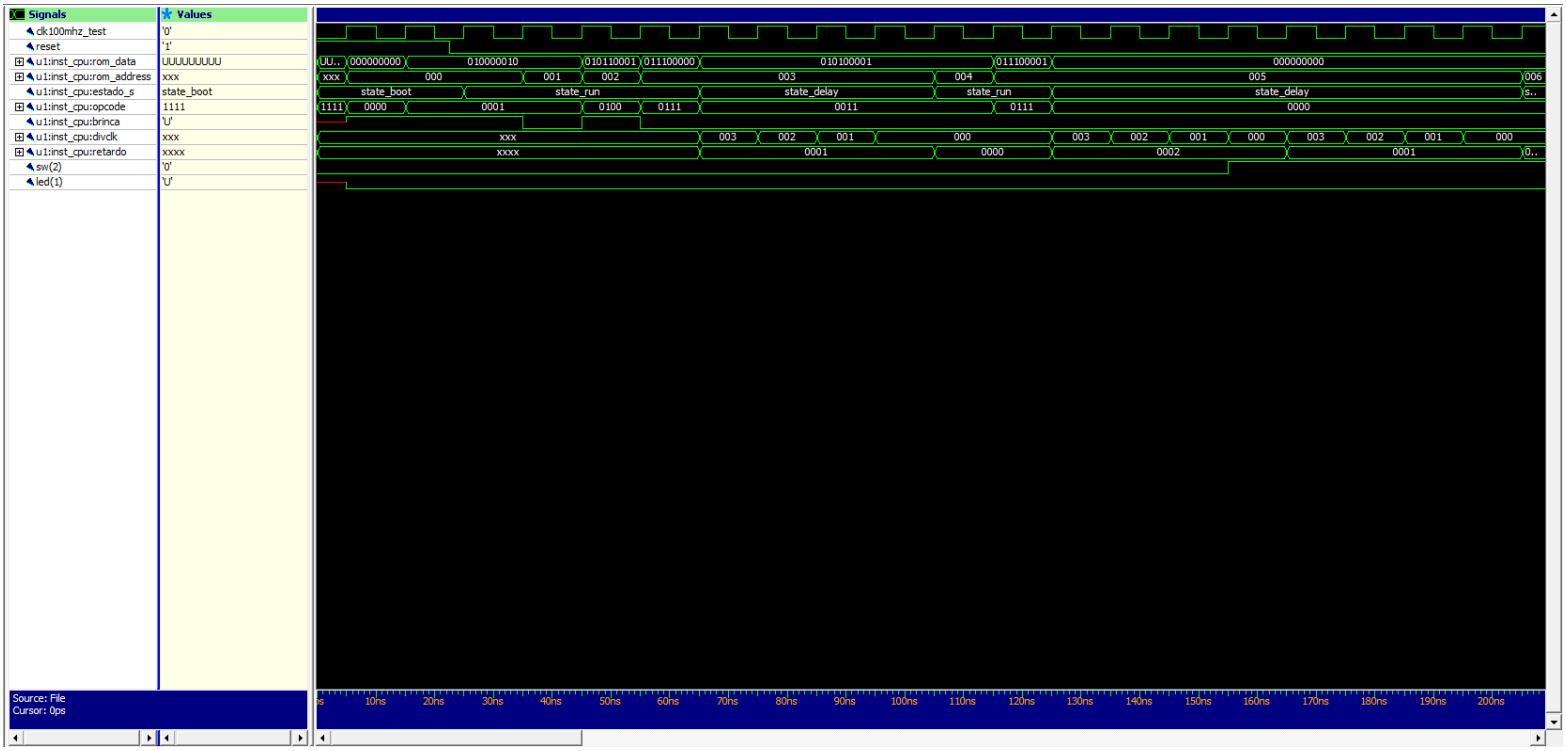
```
when OPCODE_LEE =>
   -- Esta instruccion no hace nada. Deja pasar un ciclo de reloj e incrementa el
   -- contador de programa para que apunte a la siguiente instruccion
   led(15 downto 8) <= RAM(to_integer(unsigned(ROM_data(3 downto 0))));
   pcnext <= pcnext+1;
```

Realizamos los mismos pasos que en el apartado anterior para programar la instrucción lee. Esta instrucción consiste en leer datos de la memoria RAM y asignarlos a la señal "led", y luego incrementa el contador de programa para que apunte a la siguiente instrucción en la secuencia.

```
begin
   RAM(1) <= "11110011";
```

Adicionalmente, tal y como se detalla en el enunciado del proyecto, después del begin asignamos el valor '11110011' en la posición 1 de la memoria RAM.

## 3. Programación de la instrucción ‘retardo’. (1.5 puntos)




Identificamos el código de operación correspondiente a la operación ‘retardo’ y añadimos las dos transiciones entre estados: del estado STATE_RUN al estado STATE_DELAY cuando la instrucción sea de retardo; y del estado STATE_DELAY al estado STATE_RUN cuando se haya completado el retardo:

```
OPCODE_RETARDO          when ROM_data(8 downto 4) = "01110"     else
```

```
when OPCODE_RETARDO =>
   -- Se asigna la entrada estado_c para el cambio de estado
    estado_c    <= STATE_DELAY;
```

```
when STATE_DELAY =>
  if (divclk=x"0000" and retardo=x"0001") then
      estado_c    <= STATE_RUN;
  end if;
```

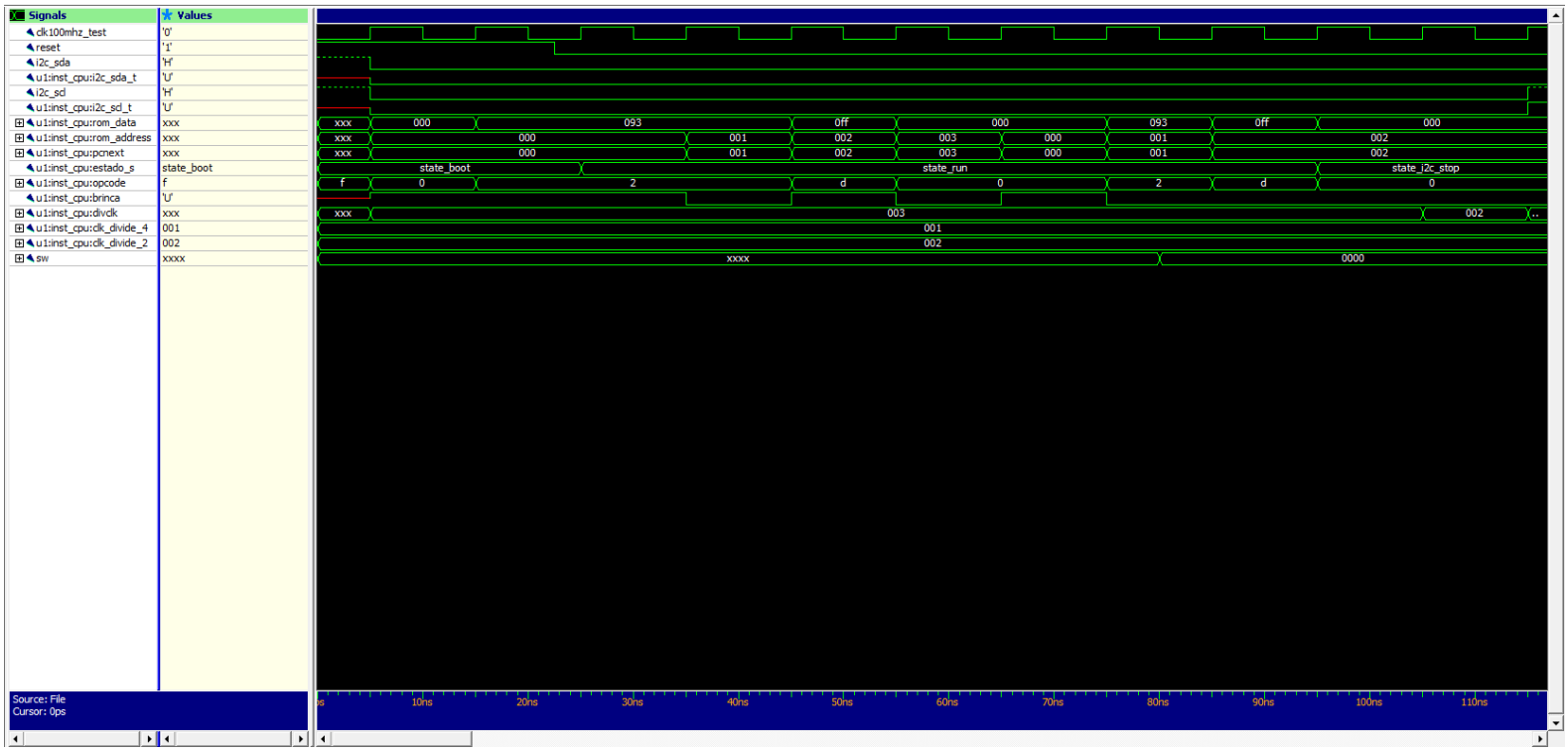
Siguiendo con las instrucciones del enunciado, lo siguiente a realizar es iniciar los contadores divclk y retardo, que se realiza en el estado STATE_RUN. La peculiaridad de éste apartado consiste en asignar un valor específico de retardo, ya que el operando de la instrucción retardo no puede ser un número cualquiera, tiene que ser potencia de 2 y esto se realiza con un case de la siguiente forma:

```
when OPCODE_RETARDO =>
   -- Valores iniciales de divlck y retardo para realizar la cuenta de retardo*clk_divide
   divclk <= unsigned(clk_divide) - 1;
   case ROM_data(3 downto 0) is
       when "0000" => retardo <= x"0001";
       when "0001" => retardo <= x"0002";
       when "0010" => retardo <= x"0004";
       when "0011" => retardo <= x"0008";
       when "0100" => retardo <= x"0010";
       when "0101" => retardo <= x"0020";
       when "0110" => retardo <= x"0040";
       when "0111" => retardo <= x"0080";
       when "1000" => retardo <= x"0100";
       when "1001" => retardo <= x"0200";
       when "1010" => retardo <= x"0400";
       when "1011" => retardo <= x"0800";
       when "1100" => retardo <= x"1000";
       when "1101" => retardo <= x"2000";
       when "1110" => retardo <= x"4000";
       when "1111" => retardo <= x"8000";
       when others => retardo <= x"0001";
   end case;
```

Por último tenemos que decrementar los contadores en el estado STATE_DELAY. Primero se asegura de que el contador divclk sea mayor que cero. Si es así, se decrementa divclk en 1. Una vez que divclk llega a cero, el contador de retardo se decrementa y cuando retardo llega a cero, el estado de retardo termina.

```
when STATE_DELAY =>
   -- divclk permite obtener un ciclo del reloj i2c_scl
   -- deja pasar retardo*clk_divide ciclos del reloj de 100Mhz
    if divclk > x"0000" then
        divclk <= divclk - 1; -- Decrementa divclk
        elsif retardo > x"0000" then
            retardo <= retardo - 1; -- Decrementa retardo
        end if;
```

## 4. Programación de la instrucción ‘stop’. (1.5 puntos)



Primero se definen las señales clk_divide_2 y clk_divide_4, que corresponden a la mitad y la cuarta parte de ciclos del reloj GENERIC clk_divide.

```
signal clk_divide_2 : unsigned(clk_divide'high downto 0);
signal clk_divide_4 : unsigned(clk_divide'high downto 0);
```

```
begin

   RAM(1) <= "11110011";
   clk_divide_2 <= unsigned(clk_divide)/2;
   clk_divide_4 <= unsigned(clk_divide)/4;


   i2c_sda_o <= '0';
   i2c_scl_o <= '0';     -- si i2c_scl_t='0' (buffer de salida)  -> saldr  un  '0'
                         -- si i2c_scl_t='1' (buffer de entrada) -> entrar  un '1' por el pullup interno
```

Identificamos el código de operación que corresponde a la operación stop:

```
OPCODE_I2C_STOP         when ROM_data(8 downto 0) = "011111111" else
```

Codificamos las dos transiciones de estados: del estado STATE_RUN al estado STATE_I2C_STOP cuando la instrucción sea ‘stop’; y del estado STATE_I2C_STOP al estado STATE_RUN cuando se haya completado la secuencia de stop, es decir, cuando el contador divclk haya contado decrecientemente hasta 0.

```
when OPCODE_I2C_STOP =>
   estado_c <= STATE_I2C_STOP;
```

```
when STATE_I2C_STOP =>
   if(divclk = x"0000") then
      estado_c <= STATE_RUN;
   end if;
```

Excepcionalmente se ha puesto a ‘0’ la señal bidireccional del reloj I2C_scl, como se indica en el enunciado:

```
i2c_scl_t <= '0';   -- pin I2C_scl de entrada (se ver  un '1' por el pull-up interno del pin)
i2c_sda_t <= '0';   -- pin I2C_sda de entrada (se ver  un '1' por el pull-up interno del pin)
```

```
when OPCODE_I2C_STOP =>
   divclk <= unsigned(clk_divide) - 1;
```

En el estado STATE_RUN del último proceso iniciamos el contador divclk.

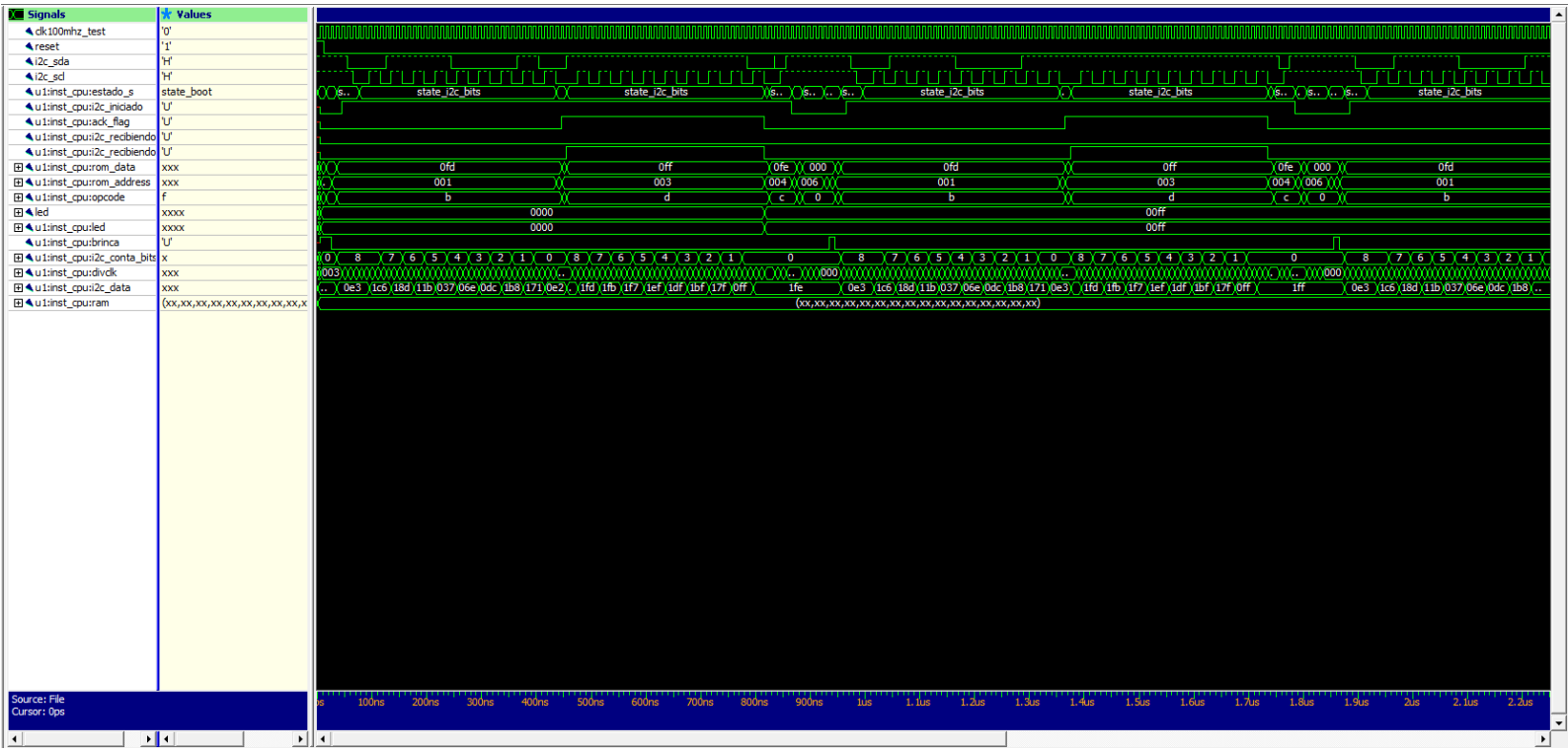
Por último realizamos las siguientes modificaciones en el estado STATE_I2C_STOP: poner la señal I2C_sda a ‘0’ cuando ha pasado un cuarto de la cuenta anterior, poner I2C_scl a ‘1’ cuando la cuenta de divclk vaya por la mitad, poner I2C_sda a ‘1’ cuando la cuenta de divclk vaya por ¾ del total e incrementar el contador de programa cuando el contador divclk llegue a cero:

```
when STATE_I2C_STOP =>
    if (divclk = "0") then
            pcnext <= pcnext + 1;
    end if;
    if (divclk > "0") then
            divclk <= divclk-1;
            if (divclk = (unsigned(clk_divide) - unsigned(clk_divide_4))) then
                    --i2c_sda_t <= '0' del STATE BOOT
                    --i2c_sda_t <=0 | i2c_sda_i <= 0 | i2c_sda_o = 0 ==> i2c_sda <= 0
                    --      T       |       O       |      I
                    i2c_sda_t <= '0';
            end if;

            if (divclk = (unsigned(clk_divide) - unsigned(clk_divide_2))) then
                    --i2c_scl_t <= '0' del STATE BOOT
                    --i2c_scl_t <=0 | i2c_scl_i <= 0 | i2c_scl_o = 1 ==> i2c_sda <= 1
                    --      T       |       O       |      I
                    i2c_scl_t <= '1';
            end if;

            if (divclk = unsigned(clk_divide_4)) then
                    --i2c_sda_t <= '1' del STATE BOOT
                    --i2c_sda_t <=0 | i2c_sda_i <= 0 | i2c_sda_o = 0 ==> i2c_sda <= 0
                    --      T       |       O       |      I
                    i2c_sda_t <= '1';
            end if;
    end if;
```

**5. Programación parcial de la instrucción 'envia'.** (2 puntos)

**6. Programación completa de la instrucción 'envia' y 'enviaBajo'.** (2 puntos)

## 7. Programación de las instrucciones ‘masterAck’, ‘recibe’ y ‘recibeBajo’. (2puntos)




En este apartado ya tenemos la identificación de todos los códigos de operaciones disponible:

```
opcode <= OPCODE_SALTA           when ROM_data(8 downto 7) = "00"        else
          OPCODE_BRINCA_SI_0     when ROM_data(8 downto 4) = "01000"     else
          OPCODE_BRINCA_SI_1     when ROM_data(8 downto 4) = "01001"     else
          OPCODE_PIN_1           when ROM_data(8 downto 4) = "01011"     else
          OPCODE_PIN_0           when ROM_data(8 downto 4) = "01010"     else
          OPCODE_LEE             when ROM_data(8 downto 4) = "01101"     else
          OPCODE_RETARDO         when ROM_data(8 downto 4) = "01110"     else
          OPCODE_NOP             when ROM_data(8 downto 0) = "011111110" else
          OPCODE_I2C_STOP        when ROM_data(8 downto 0) = "011111111" else
          OPCODE_I2C_ENVIA       when ROM_data(8 downto 8) = "1"         else
          OPCODE_I2C_ENVIA_BAJO  when ROM_data(8 downto 0) = "011110010" else
          OPCODE_MASTER_ACK      when ROM_data(8 downto 0) = "011111101" else
          OPCODE_I2C_RECIBE      when ROM_data(8 downto 4) = "01100"     else
          OPCODE_I2C_RECIBE_BAJO when ROM_data(8 downto 0) = "011110001" else
          OPCODE_UNKNOWN;
```

El siguiente paso es definir las señales ack_flag, i2c_recibiendo e i2c_recibiendoBajo:

```
signal ack_flag  : std_logic;
```

```
-- I2C estado
signal i2c_iniciado    : std_logic;
signal i2c_recibiendo  : std_logic;
signal i2c_recibiendoBajo : std_logic;
signal i2c_conta_bits  : unsigned(3 downto 0);   -- cuenta bits a enviar/recibir de forma decreciente de 8 a 0 (incluye el ack)
signal i2c_data        : std_logic_vector(8 downto 0);
```

Inicializamos la señal ack_flag a ‘0’ en el siguiente proceso:

```
IF (rising_edge(clk)) THEN
  case estado_s is

     when STATE_BOOT =>
         led       <= (others => '0');
         i2c_scl_t <= '1';   -- pin I2C_scl de entrada (se ver  un '1' por el pull-up interno del pin)
         i2c_sda_t <= '1';   -- pin I2C_sda de entrada (se ver  un '1' por el pull-up interno del pin)

         ack_flag  <= '0';
         brinca    <= '1';   -- La ROM es sincrona y tarde un ciclo en poner la instrucci n
                             -- desde que se pone la direccion de la misma. La CPU no hace
                             -- nada en el primer ciclo de reloj despu s del reset hasta que
                             -- disponible la primera instruccion
```

Codificamos las operaciones master_ACK, i2c_recibe e i2c_recibeBajo. La operación master_ACK se encarga de poner a '1' la señal ack_flag e incrementa el contador de programa. Las operacines recibe y recibeBajo primero inician el contador divclk y se asigna a la dirección RAM los datos correspondientes de ROM_data. Lo siguiente es preparar el registro de desplazamiento con el valor: I2C_data <= 0xFF & (not ack_flag) y pasa al estado STATE_I2C_BITS.

```
when OPCODE_MASTER_ACK =>
   ack_flag <= '1';
   pcnext <= pcnext+1;

when OPCODE_I2C_RECIBE =>
   divclk <= unsigned(clk_divide) - 1;
   ram_addr <= ROM_data(3 downto 0);
   i2c_data <= x"FF" & (not ack_flag);
   i2c_recibiendo <= '1';
   i2c_conta_bits <= "1000";

when OPCODE_I2C_RECIBE_BAJO =>
   divclk <= unsigned(clk_divide) - 1;
   i2c_data <= x"FF" & (not ack_flag);
   i2c_recibiendoBajo <= '1';
   i2c_conta_bits <= "1000";
```

Si la operación que se está realizando es recibiendo, se guarda en la RAM los datos de i2c_data (los primeros 8 bits son los que contienen los datos). Una vez guardado pone a '0' la señal i2c_recibiendo. Por el contrario, si la operación es recibiendoBajo, los datos de i2c_data los pasa directamente a la señal 'led' y cuando acaba pone a '0' la señal i2c_recibiendoBajo. Al final se incrementa el contador de programa.

```
if (i2c_conta_bits = "0" and divclk = "0") then


        --ack_flag <= i2c_sda_i;
        ack_flag <= '0';

        if i2c_recibiendo = '1' then
                RAM(to_integer(unsigned(ram_addr))) <= i2c_data(8 downto 1);
                i2c_recibiendo <= '0';
        end if;

        if i2c_recibiendoBajo = '1' then
                led(7 downto 0) <= i2c_data(8 downto 1);
                i2c_recibiendoBajo <= '0';
        end if;

        pcnext <= pcnext + 1;
end if;
```