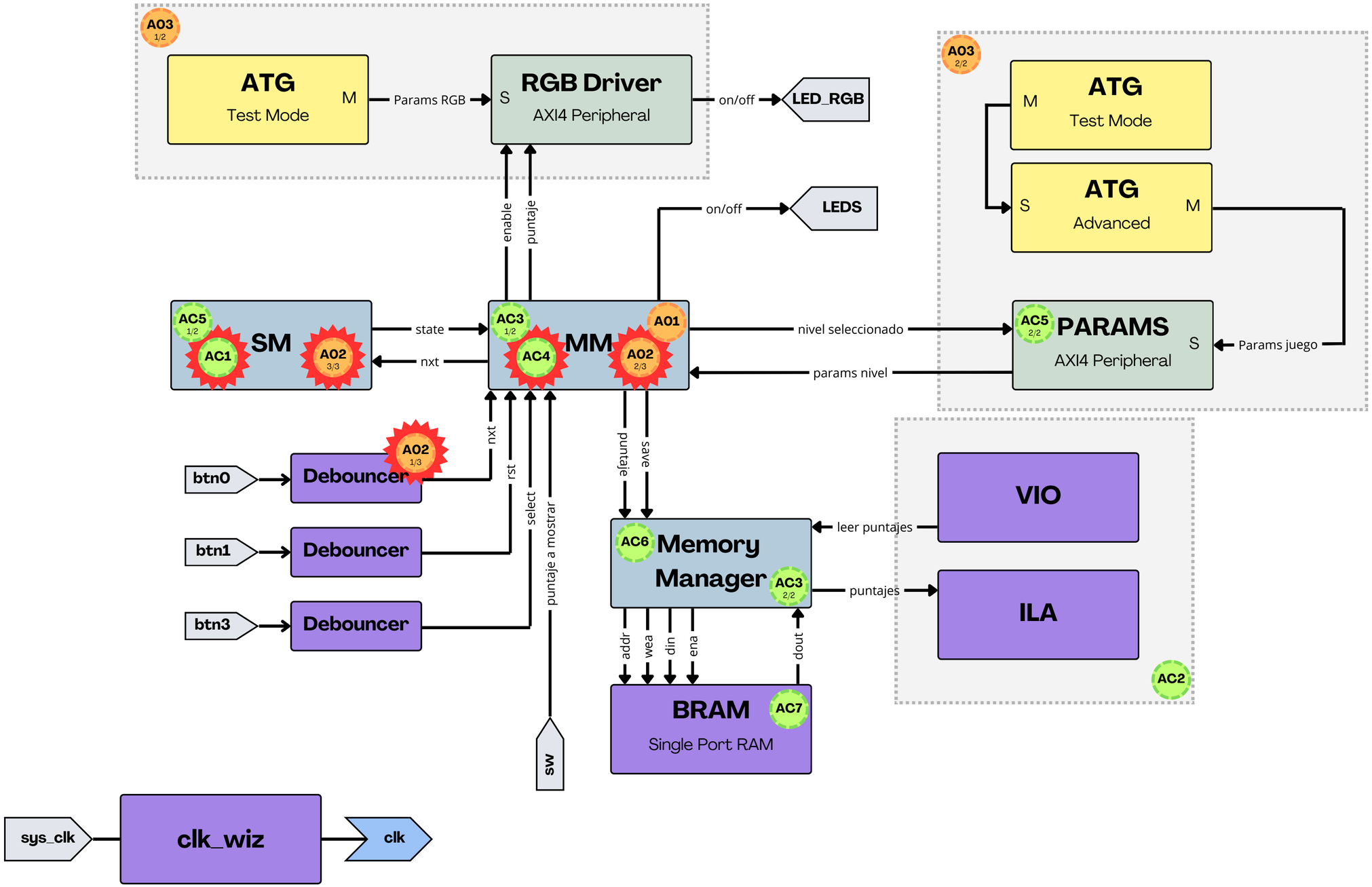
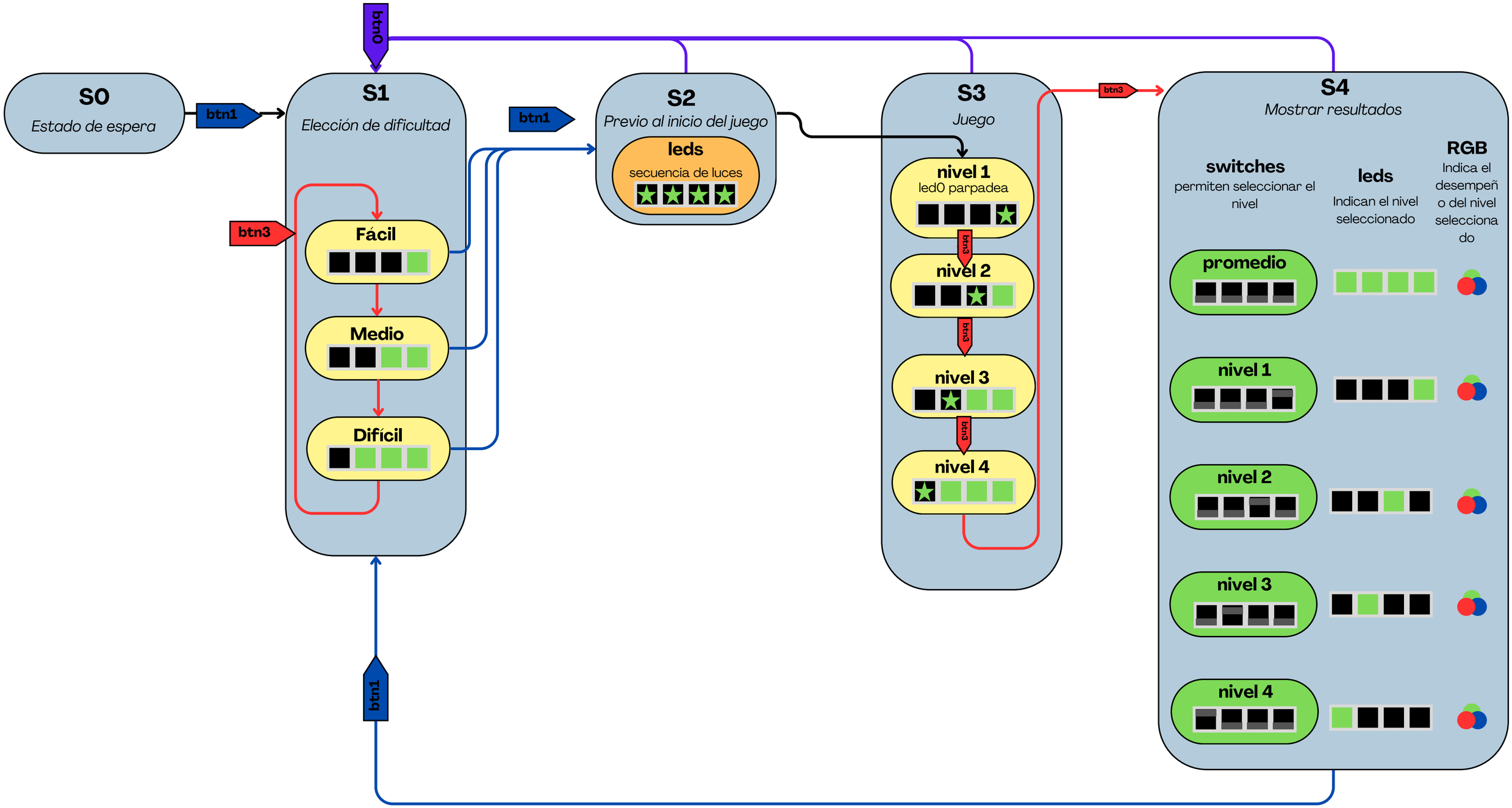
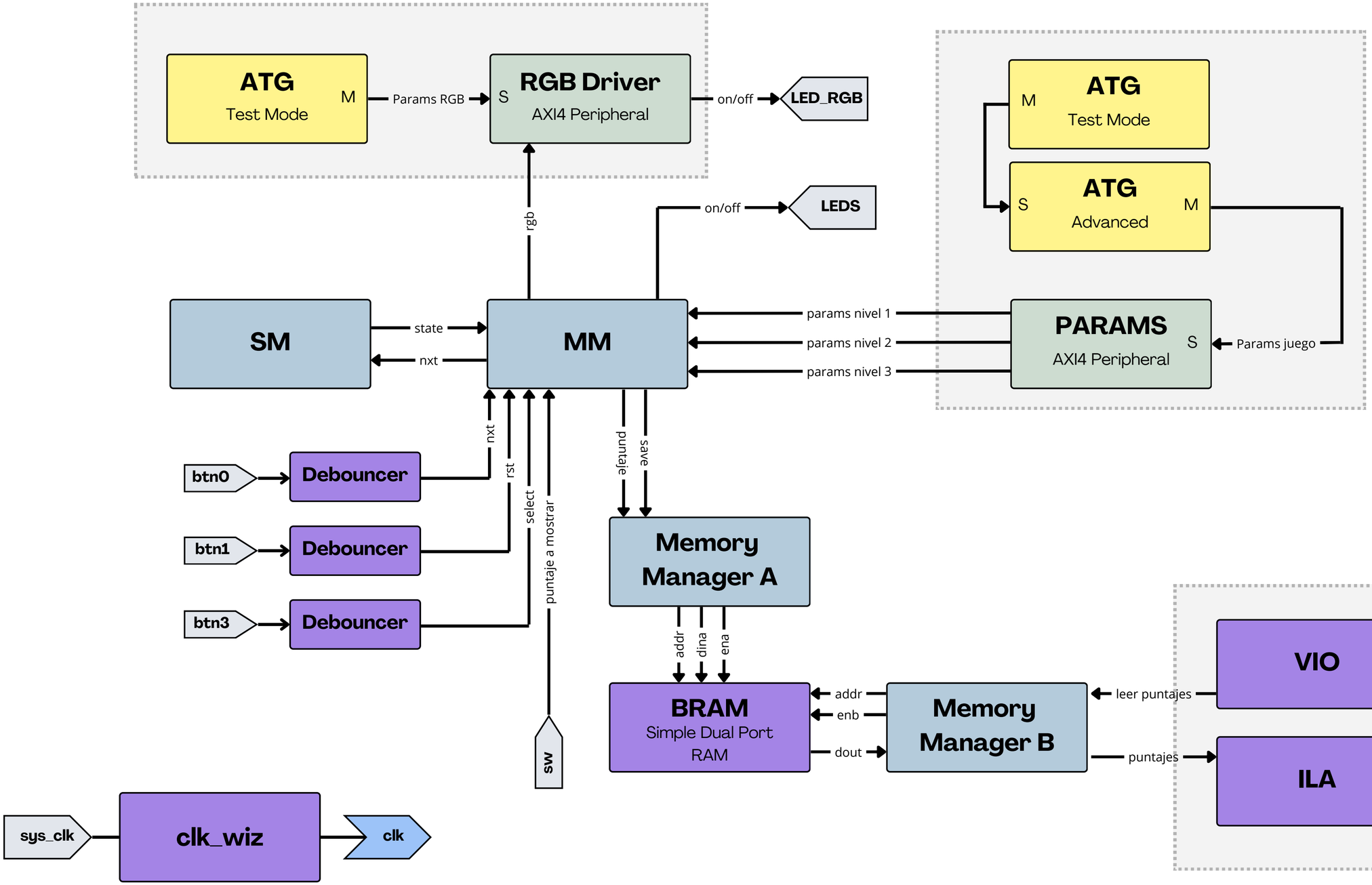
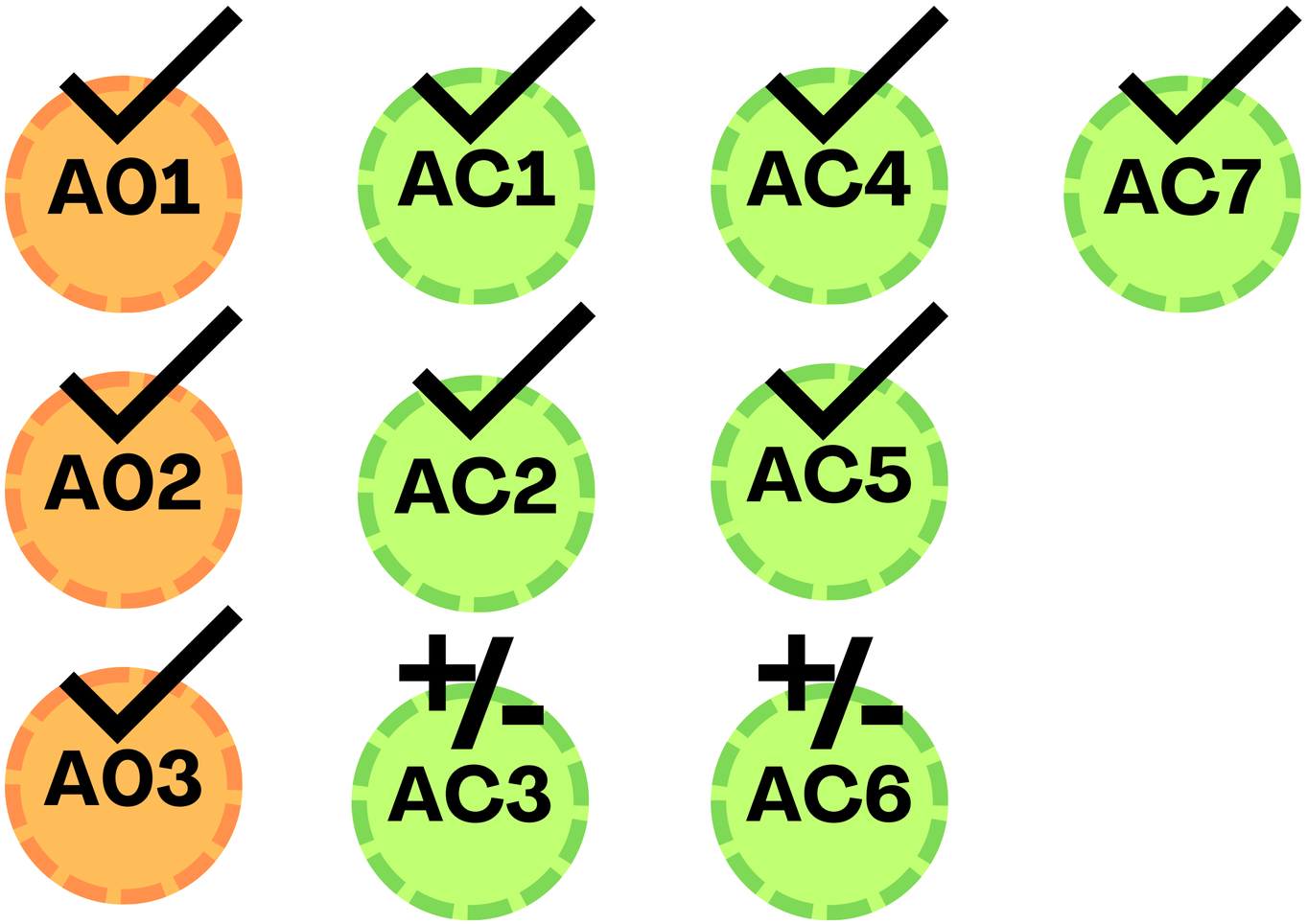

AO3
1/2
ATG
Test Mode
M
Params RGB
S
RGB Driver
AXI4 Peripheral
on/off
LED_RGB
enable
puntaje
on/off
LEDS
AO3
2/2
M
ATG
Test Mode
S
ATG
Advanced
M
AC5
1/2
AC1
SM
AO2
3/3
state
nxt
AC3
1/2
AC4
MM
AO1
AO2
2/3
nivel seleccionado
params nivel
AC5
2/2
PARAMS
AXI4 Peripheral
S
Params juego
AO2
1/3
btn0
Debouncer
btn1
Debouncer
btn3
Debouncer
nxt
rst
select
puntaje a mostrar
sw
puntaje
save
AC6
Memory
Manager
AC3
2/2
leer puntajes
puntajes
addr
wea
din
ena
dout
BRAM
Single Port RAM
AC7
VIO
ILA
AC2
sys_clk
clk_wiz
clk

# El Para Cronómetro Inador

## GRUPO 19


S0
Estado de espera
btn1
S1
Elección de dificultad
btn3
Fácil
Medio
Difícil
btn0
btn1
S2
Previo al inicio del juego
leds
secuencia de luces
S3
Juego
nivel 1
led0 parpadea
btn3
nivel 2
btn3
nivel 3
btn3
nivel 4
btn3
S4
Mostrar resultados
switches
permiten seleccionar el nivel
leds
Indican el nivel seleccionado
RGB
Indica el desempeño del nivel seleccionado
promedio
nivel 1
nivel 2
nivel 3
nivel 4
btn1


ATG
Test Mode
M
Params RGB
S
RGB Driver
AXI4 Peripheral
on/off
LED_RGB
rgb
on/off
LEDS
M
ATG
Test Mode
S
ATG
Advanced
M
SM
state
nxt
MM
params nivel 1
params nivel 2
params nivel 3
PARAMS
AXI4 Peripheral
S
Params juego
nxt
rst
select
puntaje a mostrar
puntaje
save
btn0
Debouncer
btn1
Debouncer
btn3
Debouncer
Memory
Manager A
addr
dina
ena
BRAM
Simple Dual Port
RAM
addr
enb
dout
Memory
Manager B
leer puntajes
puntajes
VIO
ILA
sw
sys_clk
clk_wiz
clk


AO1
AC1
AC4
AC7
AO2
AC2
AC5
AO3
+/-
AC3
+/-
AC6

# A01

Utilice components para integrar al menos tres componentes dentro de otra entity

**MM**
*MM.vhd*

```
COMPONENT Luces IS
    Port ( strt : in STD_LOGIC;
           rst : in STD_LOGIC;
           clock : in STD_LOGIC;
           luces : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           fin : out STD_LOGIC := '0'
           );
END COMPONENT;
```

```
COMPONENT Leds_Juego IS
    Port ( encender : in STD_LOGIC;
           nxt : in STD_LOGIC;
           apagar : in STD_LOGIC;
           clock : in STD_LOGIC;
           luces : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           fin : out STD_LOGIC := '0'
           );
END COMPONENT;
```

```
COMPONENT Leds_Admin IS
    Port (leds_show, leds_game, leds_dificultad, leds_ptje: in STD_LOGIC_VECTOR(3 downto 0);
          estado : in STD_LOGIC_VECTOR (2 downto 0);
          leds : out STD_LOGIC_VECTOR (3 downto 0)
          );
END COMPONENT;
```

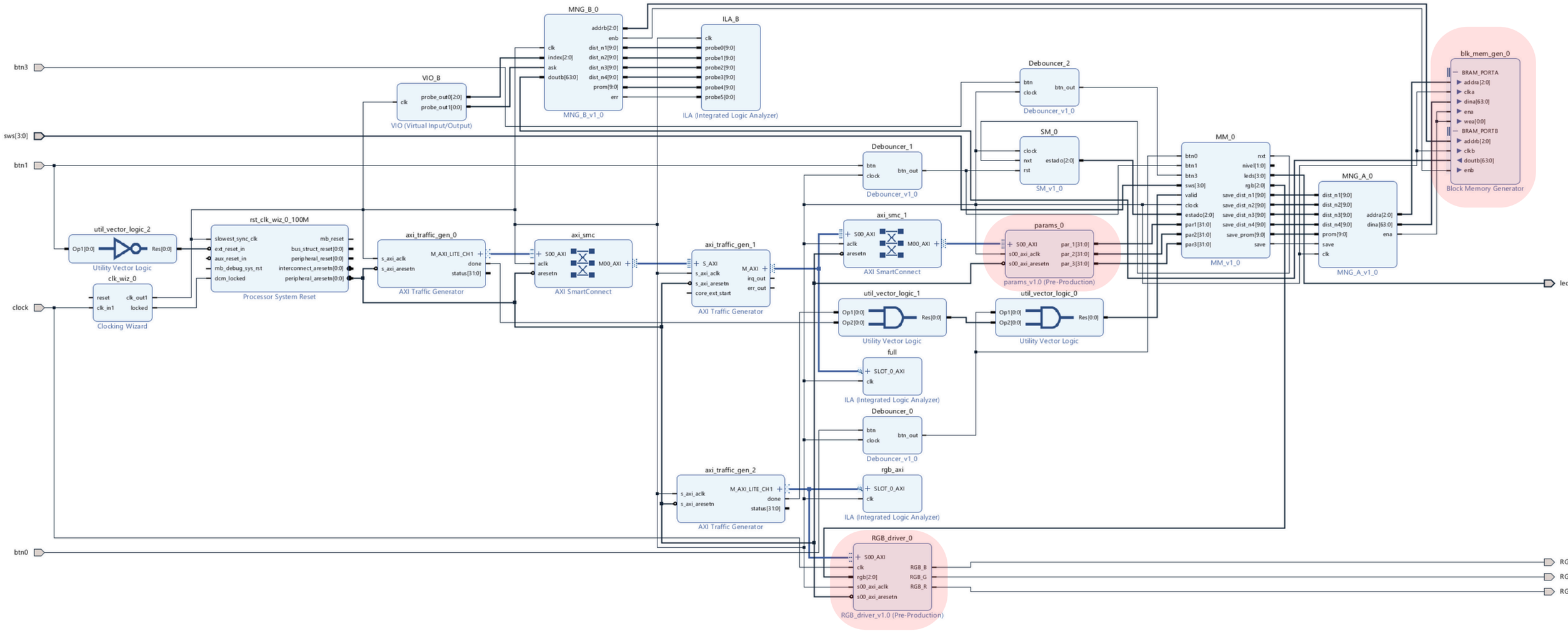
```
lights : Luces PORT MAP ( strt => iniciar_show,
                          rst => btn1,
                          clock => clock,
                          luces => leds_show,
                          fin => fin_show
                          );
```

```
game_lights : Leds_Juego PORT MAP ( encender => game_leds_on,
                                    nxt => game_leds_timeout,
                                    apagar => btn1,
                                    clock => clock,
                                    luces => leds_game,
                                    fin => game_leds_over
                                    );
```

```
leds_ctrler : Leds_Admin PORT MAP ( leds_show => leds_show,
                                    leds_game => leds_game,
                                    leds_dificultad => leds_dificultad,
                                    leds_ptje => leds_ptje,
                                    estado => estado,
                                    leds => leds
                                    );
```

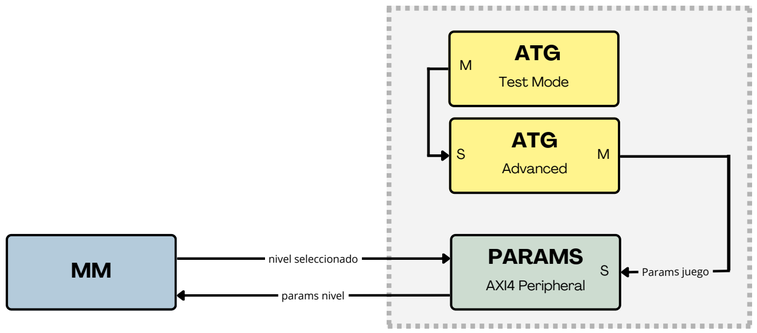
# A02

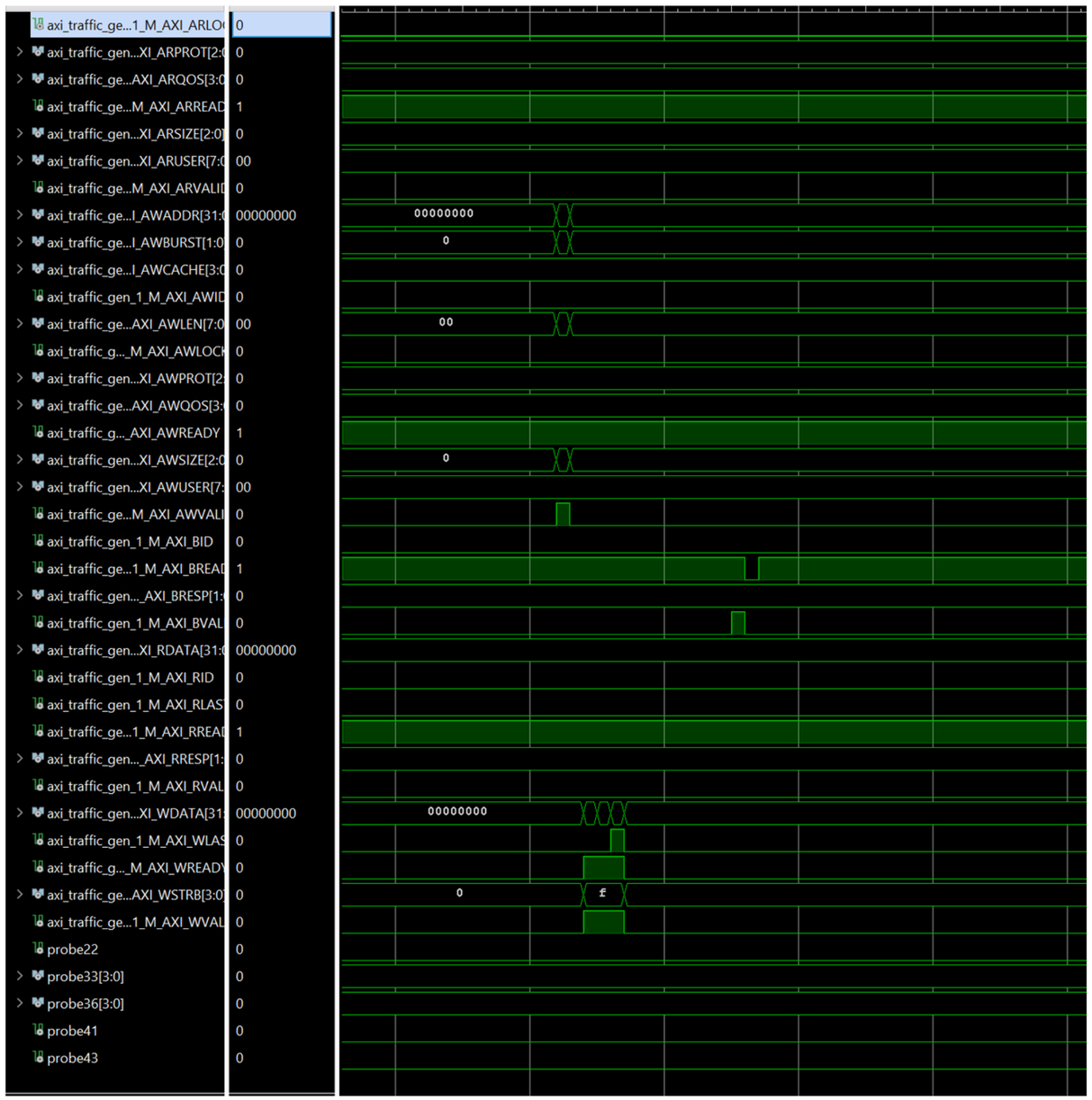
Genere al menos 3 diferentes packages (IP-Cores) en vivado block design para incorporar sus códigos. Utilice parámetos genéricos para la configuración de sus packages.



# A03

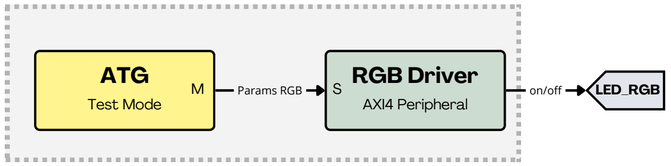
Utilice comunicación AXI para comunicar 1. al menos un IP core esclavo creado por ud con un ATG maestro en Test Mode y 2. al menos un IP-core esclavo creado por ud con un ATG maestro en Advance Mode.

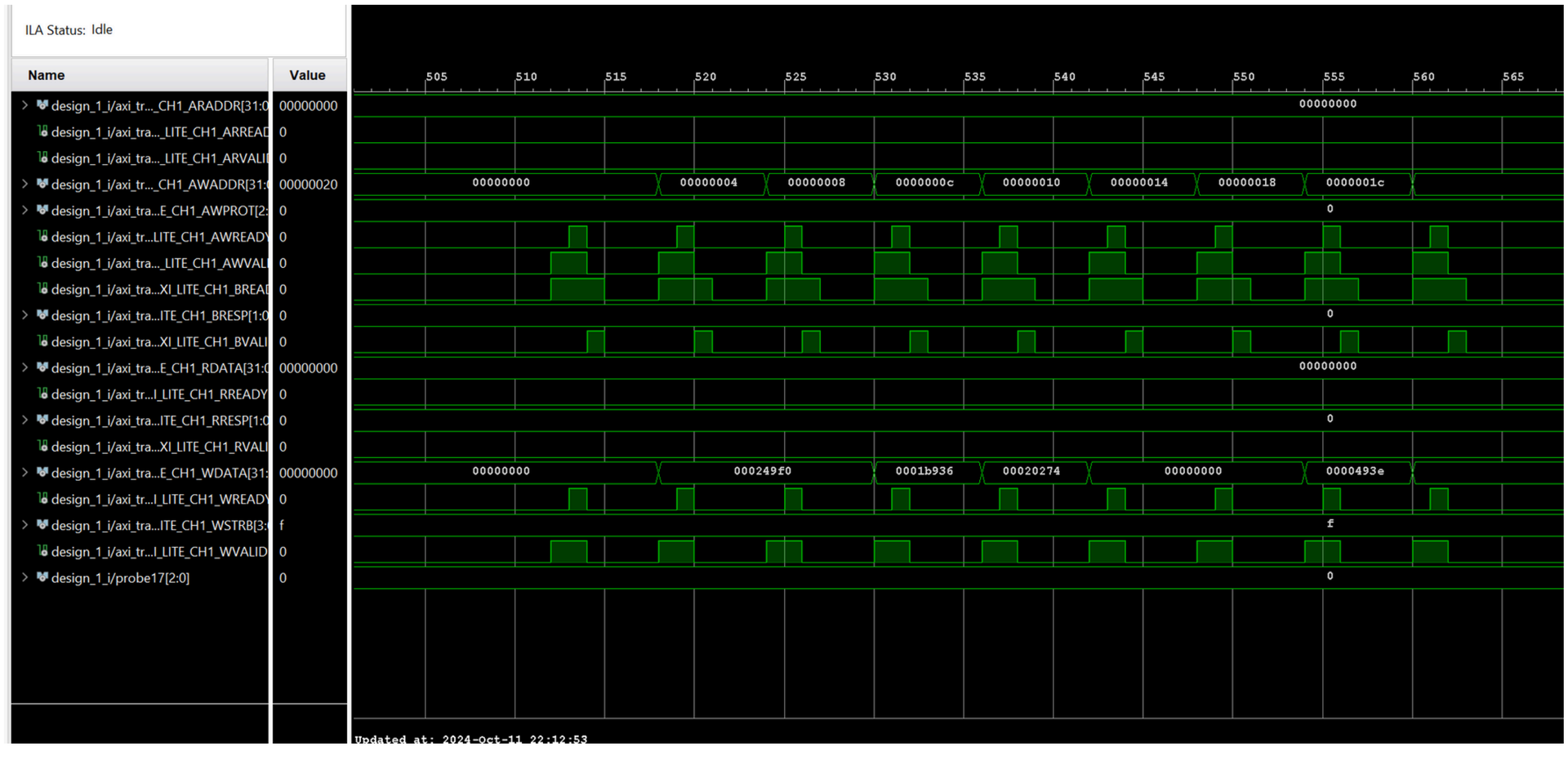


# AO3

Utilice comunicación AXI para comunicar 1. al menos un IP core esclavo creado por ud con un ATG maestro en Test Mode y 2. al menos un IP-core esclavo creado por ud con un ATG maestro en Advance Mode.

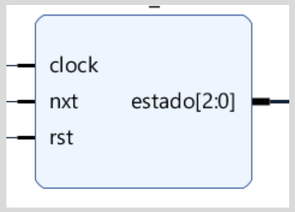


# AC1

Genere una máquina de estados que esté asociada a parte o a el total de su proyecto.

**SM**

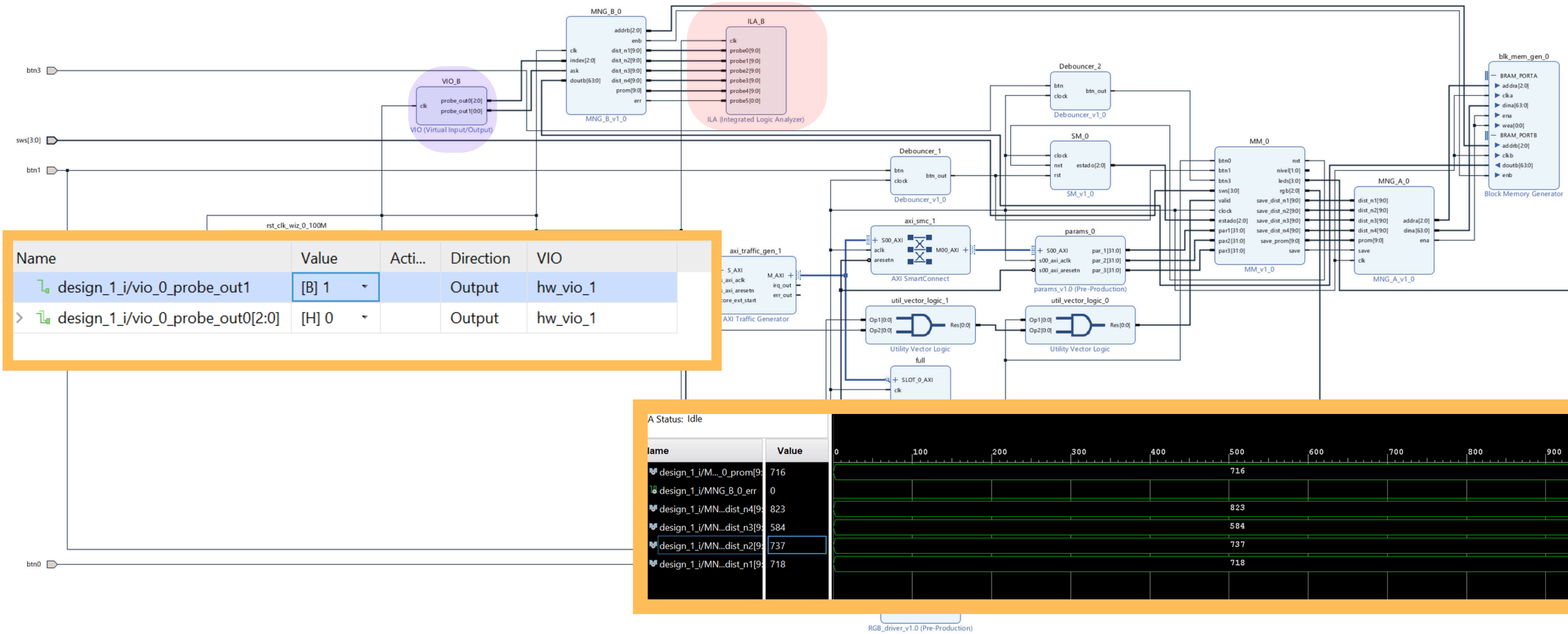



```
rocess(clock, nxt, rst)
   variable flag_nxt, flag_rst : std_logic := '0';
   variable state : INTEGER RANGE 0 TO 4 := 0;
   begin
   if rising_edge(clock) then
       if rst = '1' and flag_rst = '0' and nxt = '0' then
           state := 1;
           estado <= "001";
           flag_rst := '1';
       elsif rst = '0' and flag_rst = '1' then
           flag_rst := '0';
       end if;

       if nxt = '1' and flag_nxt = '0' and rst = '0' then
           if (state = 0 and flag_nxt = '0') then         --"000" -> "001"
               state := 1;
               estado <= "001";
               flag_nxt := '1';
           elsif (state = 1) then      --"001" -> "010"  (Este se define como predeterminado)
               state := 2;
               estado <= "010";
               flag_nxt := '1';
           elsif (state = 2) then      --"010" -> "011"
               state := 3;
               estado <= "011";
               flag_nxt := '1';
           elsif (state = 3) then      --"011" -> "100"
               state := 4;
               estado <= "100";
               flag_nxt := '1';
           elsif (state = 4) then      --"100" -> "000"
               state := 1;
               estado <= "001";
               flag_nxt := '1';
           end if;
       elsif nxt = '0' and flag_nxt = '1' then
           flag_nxt := '0';
       end if;
   end if;
end process;
```

# AC2

Utilice Vio e ILA para monitorear y modificar señales de su proyecto.




| Name | Value | Acti... | Direction | VIO |
|---|---|---|---|---|
| design_1_i/vio_0_probe_out1 | [B] 1 | | Output | hw_vio_1 |
| design_1_i/vio_0_probe_out0[2:0] | [H] 0 | | Output | hw_vio_1 |

**AC3**

Utilice al menos 2 operadores y 2 atributos diferentes.

**MM**

```
FUNCTION promedio (ptj1, ptj2, ptj3, ptj4 : INTEGER RANGE 0 TO 10_000) RETURN INTEGER  IS
    VARIABLE prom : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div1 : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div2 : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div3 : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div4 : INTEGER RANGE 0 TO 10_000 := 0;
BEGIN
    div1 := ptj1/4;
    div2 := ptj2/4;
    div3 := ptj3/4;
    div4 := ptj4/4;

    prom := div1 + div2 + div3 + div4;

    RETURN prom;
END promedio;
```

**AC4**

Utilice variables para actualizar valores instantaneamente en alguna parte de la lógica programada.

**MM**

```
process(clock, btn0, btn1, btn3, sws, valid, estado, par1, par2, par3)

variable flag_btn1 : STD_LOGIC := '0'; -- Se tiene solo para saber si se reinició o no, para resetear info.
variable par1_temp, par2_temp, par3_temp : STD_LOGIC_VECTOR(31 downto 0); -- Para almacenar la info leida.

variable dificultad : INTEGER RANGE 0 TO 2 := 0; -- 0: fácil (verde), 1: media (naranja), 2: difícil (rojo).
variable info_nivel : STD_LOGIC_VECTOR(31 downto 0); -- Para almacenar la data leída de la RAM.
variable flag_btn3 : STD_LOGIC := '0'; -- Variable que evita que se active multiples veces el botón 4.
variable flag_btn0 : STD_LOGIC := '0'; -- Evita muchos cambios de estado.


variable nvl_actl : INTEGER RANGE 0 TO 3 := 0;
variable dist_n0, dist_n1, dist_n2, dist_n3, prom : INTEGER RANGE 0 TO 10_000 := 0;
variable timer1 : INTEGER RANGE 0 TO 100_020 := 0; -- Crearemos un timer que aumenta 1 cada 100_000 ciclos (Baja la frecuencia a 1000 Hz).
variable timer2 : INTEGER RANGE 0 TO 10_020 := 0; -- Este es el que llevará la cuenta total durante el juego.
variable juego_completo : STD_LOGIC := '0'; -- Variable que se vuelve uno al terminar los 4 niveles. Sirve para hacer una pequeña pausa antes de ptje.

-- Nuevo
variable cuenta_regresiva : STD_LOGIC := '0';
variable clk_div_flag : STD_LOGIC := '0'; -- variable que permite lo mismo que la función rising_edge(clk_div).
------------
```

```
        if flag_btn1 = '1' and btn1 = '0' then -- Si se llega a este estado mediante reset
            leds_dificultad <= "0000";
            flag_btn1 := '0';  -- Se desactiva la señal del reset
            timer1 := 0;
            timer2 := 0;
            dificultad := 0;
            nvl_actl := 0;
            juego_completo := '0';
            flag_btn0 := '0';
            -- Nuevo
            iniciar_show <= '0';
            game_leds_on <= '0';
            game_leds_timeout <= '0';
            rgb <= "000" ; -- Apagado
```

# AC5

Utilice código secuencial y código concurrente y evidencie claramente la diferencia de funcionamiento de ambos tipos de códigos.

**Memory Manager A**

**RGB Driver**
S
AXI4 Peripheral

```
if(addr = b"111") then
    addr := b"000";
else
    addr := std_logic_vector(conv_unsigned(conv_integer(unsigned( addr )) + 1, 3));
end if;

send_BRAM := '0';
state := b"000";


end if;
```

```
buff_r <= unsigned(slv_reg0) when (rgb = b"100") else
    unsigned(slv_reg1) when (rgb = b"010") else
    unsigned(slv_reg2) when (rgb = b"001");

buff_g <= unsigned(slv_reg3) when (rgb = b"100") else
    unsigned(slv_reg4) when (rgb = b"010") else
    unsigned(slv_reg5) when (rgb = b"001");

buff_b <= unsigned(slv_reg6) when (rgb = b"100") else
    unsigned(slv_reg7) when (rgb = b"010") else
    unsigned(slv_reg8) when (rgb = b"001");
```

**AC6**

Haga uso de functions y procedures, donde el uso de ambas rutinas explicite sus diferencias principales.

```
FUNCTION promedio (ptj1, ptj2, ptj3, ptj4 : INTEGER RANGE 0 TO 10_000) RETURN INTEGER  IS
    VARIABLE prom : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div1 : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div2 : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div3 : INTEGER RANGE 0 TO 10_000 := 0;
    VARIABLE div4 : INTEGER RANGE 0 TO 10_000 := 0;
BEGIN
    div1 := ptj1/4;
    div2 := ptj2/4;
    div3 := ptj3/4;
    div4 := ptj4/4;

    prom := div1 + div2 + div3 + div4;

    RETURN prom;
END promedio;
```

**AC7**

Implemente alguna función que no se haya presentado en el curso.

# Block Memory Generator v8.4

***LogiCORE IP Product Guide***

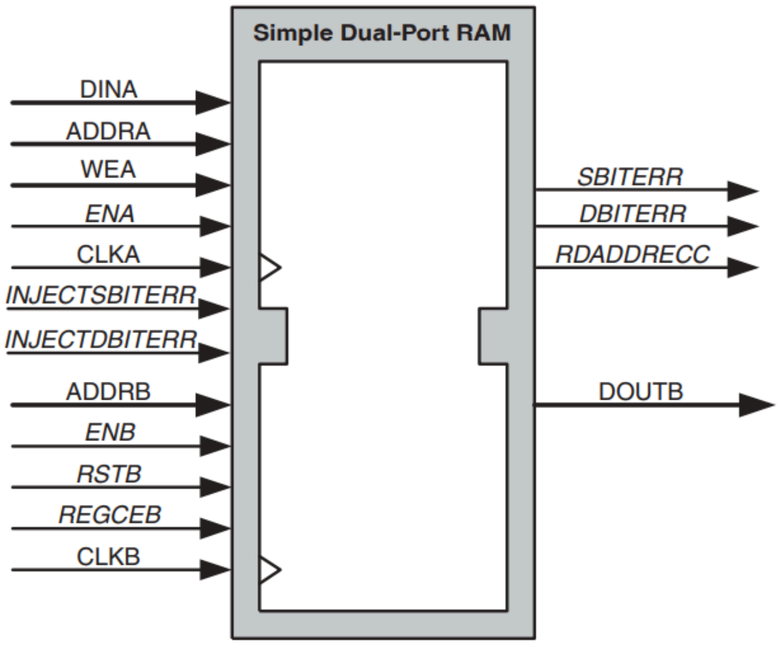



*Figure 3-4:* **Simple Dual-port RAM**

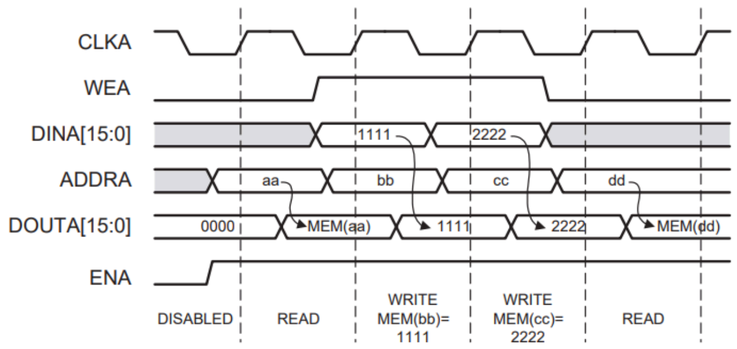



*Figure 3-9:* **Write First Mode Example**

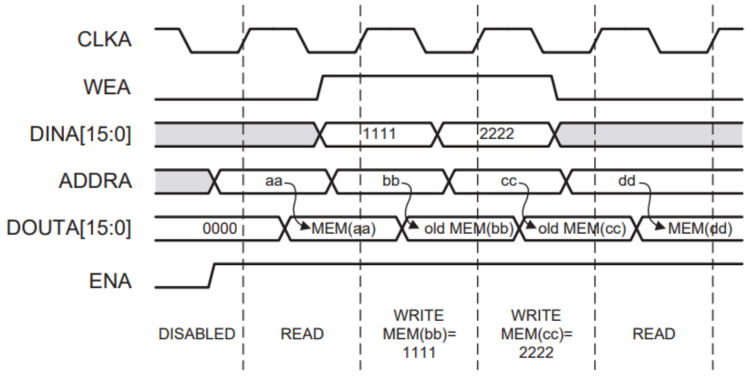



*Figure 3-10:* **Read First Mode Example**