UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Eléctrica

ANEXO: "Simulación en tiempo real de un enlace HVDC con convertidores MMC"

Autor:

Gregori Cano, Aitor

Tutor(es):

De Pablo Gómez, Santiago

Departamento de Tecnología Electrónica

Valladolid, febrero 2022.

ÍNDICE

1.- Código del convertidor de Matlab

1.1.-Código del controlador

```
function [modulator_UP1, modulator_UP2, modulator_UP3,
          SyncUP, modulator_LW1, modulator_LW2,
          modulator_LW3, SyncLW] = controller(V_P1, V_P2,
          V_P3, I_P, V_N1, V_N2, V_N3, I_N)


% setup ()

persistent Ts myTime w pSyncUP pSyncLW;
persistent m_UP1 m_UP2 m_UP3 m_LW1 m_LW2 m_LW3 swUP swLW;
persistent Nsm Vsm invVsm Vac_peak Vdc_half;

if (isempty (Ts))
    Ts        = 250e-6; % Periodo controlador 250 us
    myTime    = 0;
    w         = 2 * pi * 50; % 50 Hz

    Nsm       = 4;
    Vsm       = 800;
    Vac_peak  = 1200;
    invVsm    = 1 / Vsm;
    Vdc_half  = Nsm * Vsm / 2;

    pSyncUP   = [0 0 0];
    pSyncLW   = [0 0 0];
    m_UP1     = [1 1 0 0];
    m_UP2     = [1 1 0 0];
    m_UP3     = [1 1 0 0];
    m_LW1     = [1 1 0 0];
    m_LW2  = [1 1 0 0];
    m_LW3  = [1 1 0 0];
    swUP   = [0 0 0]; % 1 es ON, 0 es OFF
    swLW   = [0 0 0]; % 1 es ON, 0 es OFF
end


% loop ()

% Actualización del tiempo
myTime = myTime + Ts;

% Generar señal de sincronizacion
for i = 1:3
    if (swUP(i) == 0)
        if (pSyncUP(i) > 1)
            pSyncUP(i) = 0;
        else
            pSyncUP(i) = pSyncUP(i) -1;
        end
    else
        if (pSyncUP(i) < -1)
            pSyncUP(i) = 0;
        else
            pSyncUP(i) = pSyncUP(i) + 1;
        end
    end

    if (swLW(i) == 0)
        if (pSyncLW(i) > 1)
            pSyncLW(i) = 0;
        else
            pSyncLW(i) = pSyncLW(i) -1;
        end
    else
        if (pSyncLW(i) < -1)
            pSyncLW(i) = 0;
        else
            pSyncLW(i) = pSyncLW(i) + 1;
        end
    end
end % for señal sincronización

SyncUP = pSyncUP;
SyncLW = pSyncLW;


% 1. Consolidación del estado de los polos
% Rama superior
for j = 1:3
    if swUP(j) == 0 % se hizo un off en el periodo anterior
        % el que no sea 1.0 que sea 0.0
        if j == 1
            for i=1:4
                if (m_UP1(i) < 1.0)
                    m_UP1(i) = 0.0;
                end
            end
        elseif j == 2
            for i=1:4
                if (m_UP2(i) < 1.0)
                    m_UP2(i) = 0.0;
                end
            end
        else % j = 3
            for i=1:4
                if (m_UP3(i) < 1.0)
                    m_UP3(i) = 0.0;
                end
            end
        end
    else % el que no sea 0.0 que sea 1.0 (hubo un ON)
        if j == 1
            for i=1:4
                if (m_UP1(i) > 0.0)
                    m_UP1(i) = 1.0;
                end
            end
        elseif j == 2
            for i=1:4
                if (m_UP2(i) > 0.0)
                    m_UP2(i) = 1.0;
                end
            end
        else % j = 3
            for i=1:4
                if (m_UP3(i) > 0.0)
                    m_UP3(i) = 1.0;
                end
            end
        end
    end
end
% Rama inferior
for j = 1:3
    if swLW(j) == 0 % se hizo un off en el periodo anterior
        % el que no sea 1.0 que sea 0.0
        if j == 1
            for i=1:4
                if (m_LW1(i) < 1.0)
                    m_LW1(i) = 0.0;
                end
            end
        elseif j == 2
            for i=1:4
                if (m_LW2(i) < 1.0)
                    m_LW2(i) = 0.0;
                end
            end
        else % j = 3
            for i=1:4
                if (m_LW3(i) < 1.0)
                    m_LW3(i) = 0.0;
                end
            end
        end
    else % hubo un ON % el que no sea 0.0 que sea 1.0
        if j == 1
            for i=1:4
```

```
                if (m_LW1(i) > 0.0)
                    m_LW1(i) = 1.0;
                end
            end
        elseif j == 2
            for i=1:4
                if (m_LW2(i) > 0.0)
                    m_LW2(i) = 1.0;
                end
            end
        else % j = 3
            for i=1:4
                if (m_LW3(i) > 0.0)
                    m_LW3(i) = 1.0;
                end
            end
        end
    end
end

% 2. Cálculo del estado final del periodo de muestreo
anterior
% Rama superior
N_activasUP1 = m_UP1(1) + m_UP1(2) + m_UP1(3) + m_UP1(4);
N_activasUP2 = m_UP2(1) + m_UP2(2) + m_UP2(3) + m_UP2(4);
N_activasUP3 = m_UP3(1) + m_UP3(2) + m_UP3(3) + m_UP3(4);

% Rama inferior
N_activasLW1 = m_LW1(1) + m_LW1(2) + m_LW1(3) + m_LW1(4);
N_activasLW2 = m_LW2(1) + m_LW2(2) + m_LW2(3) + m_LW2(4);
N_activasLW3 = m_LW3(1) + m_LW3(2) + m_LW3(3) + m_LW3(4);


% 3. Generación de las señales moduladoras (valor de 0.0 a
4.0)
% Rama superior
m_P1 = invVsm * (Vdc_half - Vac_peak * sin (w * myTime));

m_P2 = invVsm * (Vdc_half - Vac_peak * sin (w * myTime -
2/3*pi));
m_P3 = invVsm * (Vdc_half - Vac_peak * sin (w * myTime -
4/3*pi));

% Rama inferior
m_N1 = invVsm * (Vdc_half + Vac_peak * sin (w * myTime));

m_N2 = invVsm * (Vdc_half + Vac_peak * sin (w * myTime -
2/3*pi));
m_N3 = invVsm * (Vdc_half + Vac_peak * sin (w * myTime -
4/3*pi));
```

```
% 4. Cálculo de moduladoras para el periodo de muestreo
actual
% Rama superior
residuo_P1 = m_P1 - floor(m_P1);
residuo_P2 = m_P2 - floor(m_P2);
residuo_P3 = m_P3 - floor(m_P3);

% Rama inferior
residuo_N1 = m_N1 - floor(m_N1);
residuo_N2 = m_N2 - floor(m_N2);
residuo_N3 = m_N3 - floor(m_N3);


% 5. Selección de las celdas que deben ser insertadas o
retiradas
% FASE 1 Rama superior
if (m_P1 < N_activasUP1)    % debe haber OFF (o doble OFF)

    while (m_P1 < N_activasUP1)%hace OFF una o varias veces

        swUP(1) = 0;

        if (I_P(1) > 0)
            pos_P1 = maxV(1, V_P1, m_UP1);
        else % I_P < 0
            pos_P1 = minV(1, V_P1, m_UP1);
        end

        if (pos_P1 > 0)
            m_UP1(pos_P1) = residuo_P1;
        end
        N_activasUP1 = N_activasUP1 - 1;
        residuo_P1 = 0;
    end
else                             % debe haber ON (o doble ON)

    while (m_P1 > N_activasUP1)% hace ON una o varias veces

        swUP(1) = 1;

        if (I_P(1) > 0)
            pos_P1 = minV(0, V_P1, m_UP1);
        else % I_P < 0
            pos_P1 = maxV(0, V_P1, m_UP1);
        end
        if (pos_P1 > 0)
            m_UP1(pos_P1) = residuo_P1;
        end
        N_activasUP1 = N_activasUP1 + 1;
        residuo_P1 = 1;
    end
```

```
end

% FASE 1 Rama inferior
if (m_N1 < N_activasLW1)      % debe haber OFF (o doble OFF)

    while (m_N1 < N_activasLW1)%hace OFF una o varias veces

        swLW(1) = 0;

        if (I_N(1) > 0)
            pos_N1 = maxV(1, V_N1, m_LW1);
        else % I_N < 0
            pos_N1 = minV(1, V_N1, m_LW1);
        end

        if (pos_N1 > 0)
            m_LW1(pos_N1) = residuo_N1;
        end
        N_activasLW1 = N_activasLW1 - 1;
        residuo_N1 = 0;
    end

else                             % debe haber ON (o doble ON)

    while (m_N1 > N_activasLW1)% hace ON una o varias veces

        swLW(1) = 1;

        if (I_N(1) > 0)
            pos_N1 = minV(0, V_N1, m_LW1);
        else % I_N < 0
            pos_N1 = maxV(0, V_N1, m_LW1);
        end

        if (pos_N1 > 0)
            m_LW1(pos_N1) = residuo_N1;
        end
        N_activasLW1 = N_activasLW1 + 1;
        residuo_N1 = 1;

    end
end

% FASE 2 Rama superior
if (m_P2 < N_activasUP2)      % debe haber OFF (o doble OFF)

    while (m_P2 < N_activasUP2)%hace OFF una o varias veces

        swUP(2) = 0;
        if (I_P(2) > 0)
            pos_P2 = maxV(1, V_P2, m_UP2);
        else % I_P < 0
            pos_P2 = minV(1, V_P2, m_UP2);
        end

        if (pos_P2 > 0)
            m_UP2(pos_P2) = residuo_P2;
        end
        N_activasUP2 = N_activasUP2 - 1;
        residuo_P2 = 0;
    end

else                           % debe haber ON (o doble ON)

    while (m_P2 > N_activasUP2)% hace ON una o varias veces

        swUP(2) = 1;

        if (I_P(2) > 0)
            pos_P2 = minV(0, V_P2, m_UP2);
        else % I_P < 0
            pos_P2 = maxV(0, V_P2, m_UP2);
        end

        if (pos_P2 > 0)
            m_UP2(pos_P2) = residuo_P2;
        end
        N_activasUP2 = N_activasUP2 + 1;
        residuo_P2 = 1;
    end
end

% FASE 2 Rama inferior
if (m_N2 < N_activasLW2)     % debe haber OFF (o doble OFF)

    while (m_N2 < N_activasLW2)%hace OFF una o varias veces

        swLW(2) = 0;

        if (I_N(2) > 0)
            pos_N2 = maxV(1, V_N2, m_LW2);
        else % I_N < 0
            pos_N2 = minV(1, V_N2, m_LW2);
        end

        if (pos_N2 > 0)
            m_LW2(pos_N2) = residuo_N2;
        end
        N_activasLW2 = N_activasLW2 - 1;
        residuo_N2 = 0;
    end
else                             % debe haber ON (o doble ON)

    while (m_N2 > N_activasLW2)% hace ON una o varias veces

        swLW(2) = 1;

        if (I_N(2) > 0)
            pos_N2 = minV(0, V_N2, m_LW2);
        else % I_N < 0
            pos_N2 = maxV(0, V_N2, m_LW2);
        end

        if (pos_N2 > 0)
            m_LW2(pos_N2) = residuo_N2;
        end

        N_activasLW2 = N_activasLW2 + 1;
        residuo_N2 = 1;

    end
end

% FASE 3 Rama superior
if (m_P3 < N_activasUP3)       % debe haber OFF (o doble OFF)

    while (m_P3 < N_activasUP3)%hace OFF una o varias veces

        swUP(3) = 0;

        if (I_P(3) > 0)
            pos_P3 = maxV(1, V_P3, m_UP3);
        else % I_P < 0
            pos_P3 = minV(1, V_P3, m_UP3);
        end

        if (pos_P3 > 0)
            m_UP3(pos_P3) = residuo_P3;
        end
        N_activasUP3 = N_activasUP3 - 1;
        residuo_P3 = 0;
    end
else                             % debe haber ON (o doble ON)

    while (m_P3 > N_activasUP3)% hace ON una o varias veces

        swUP(3) = 1;

        if (I_P(3) > 0)
            pos_P3 = minV(0, V_P3, m_UP3);
        else % I_P < 0
            pos_P3 = maxV(0, V_P3, m_UP3);
        end
        if (pos_P3 > 0)
            m_UP3(pos_P3) = residuo_P3;
        end
        N_activasUP3 = N_activasUP3 + 1;
        residuo_P3 = 1;

    end
end

% FASE 3 Rama inferior
if (m_N3 < N_activasLW3)      % debe haber OFF (o doble OFF)

    while (m_N3 < N_activasLW3)%hace OFF una o varias veces

        swLW(3) = 0;

        if (I_N(3) > 0)
            pos_N3 = maxV(1, V_N3, m_LW3);
        else % I_N < 0
            pos_N3 = minV(1, V_N3, m_LW3);
        end

        if (pos_N3 > 0)
            m_LW3(pos_N3) = residuo_N3;
        end

        N_activasLW3 = N_activasLW3 - 1;
        residuo_N3 = 0;

    end

else                            % debe haber ON (o doble ON)

    while (m_N3 > N_activasLW3)% hace ON una o varias veces

        swLW(3) = 1;

        if (I_N(3) > 0)
            pos_N3 = minV(0, V_N3, m_LW3);
        else % I_N < 0
            pos_N3 = maxV(0, V_N3, m_LW3);
        end

        if (pos_N3 > 0)
            m_LW3(pos_N3) = residuo_N3;
        end
        N_activasLW3 = N_activasLW3 + 1;
        residuo_N3 = 1;
    end
```

```
end

% 6. Generación de salidas
% Rama superior
modulator_UP1 = [m_UP1(1) m_UP1(2) m_UP1(3) m_UP1(4)];
modulator_UP2 = [m_UP2(1) m_UP2(2) m_UP2(3) m_UP2(4)];
modulator_UP3 = [m_UP3(1) m_UP3(2) m_UP3(3) m_UP3(4)];

% Rama inferior
modulator_LW1 = [m_LW1(1) m_LW1(2) m_LW1(3) m_LW1(4)];
modulator_LW2 = [m_LW2(1) m_LW2(2) m_LW2(3) m_LW2(4)];
modulator_LW3 = [m_LW3(1) m_LW3(2) m_LW3(3) m_LW3(4)];



end %function

function pos = maxV(insertadas, V, S)

pos=0;
maxValue = -10000;
if S(1) == insertadas
    if V(1) > maxValue
        maxValue = V(1);
        pos = 1;
    end
end
if S(2) == insertadas
    if V(2) > maxValue
        maxValue = V(2);
        pos = 2;
    end
end
if S(3) == insertadas
    if V(3) > maxValue
        maxValue = V(3);
        pos = 3;
    end
end
if S(4) == insertadas
    if V(4) > maxValue
        % maxValue = V4;
        pos = 4;
    end
end

end % maxV

function pos = minV(insertadas, V, S)

pos=0;
```

```
minValue = 10000;
if S(1) == insertadas
    if V(1) < minValue
        minValue = V(1);
        pos = 1;
    end
end
if S(2) == insertadas
    if V(2) < minValue
        minValue = V(2);
        pos = 2;
    end
end
if S(3) == insertadas
    if V(3) < minValue
        minValue = V(3);
        pos = 3;
    end
end
if S(4) == insertadas
    if V(4) < minValue
        pos = 4;
    end
end

end % minV
```

1.2.- Código de un generador de PWM

```
function [S1, S2, S3, S4] = pwm_gen(modulator, Sync)

persistent carrier step oldSync;
if ( isempty(carrier) )
    carrier   = 249.5; % 0.5;
    step      = -1;    % +1;
    oldSync   = 0;
end

% Generación de la portadora
carrier = carrier + step;

if (Sync < oldSync)      % 250us subiendo
    carrier = 0.5;
    step = +1;
end
if (Sync > oldSync)      % 250us bajando
    carrier = 249.5;
    step = -1;
end

oldSync = Sync;

% Generación de la salida PWM

if (modulator(1) * 250 < carrier)
    S1 = 0;
else
    S1 = 1;
end

if (modulator(2) * 250 < carrier)
    S2 = 0;
else
    S2 = 1;
end

if (modulator(3) * 250 < carrier)
    S3 = 0;
else
    S3 = 1;
end

if (modulator(4) * 250 < carrier)
    S4 = 0;
else
    S4 = 1;
end
```

```
end %function
```

2.- Código del bloque RCPcore

2.1.-Global

```
// Global

#define TPWM 400

#pragma GETTIME OFF

#define MMC_PROMEDIADO
#define MMC_REALISTA

#define _MMC01_SLAVE
#define MMC01_MASTER

#define UP_A_1 1
#define UP_A_2 2
#define UP_A_3 3
#define UP_A_4 4

#define LW_A_1 5
#define LW_A_2 6
#define LW_A_3 7
#define LW_A_4 8

#define UP_B_1 9
#define UP_B_2 10
#define UP_B_3 11
#define UP_B_4 12

#define LW_B_1 13
#define LW_B_2 14
#define LW_B_3 15
#define LW_B_4 16

#define UP_C_1 17
#define UP_C_2 18
#define UP_C_3 19
#define UP_C_4 20

#define LW_C_1 21
#define LW_C_2 22
#define LW_C_3 23
#define LW_C_4 24

#define DUTYCYCLE_1 temp_1
#define DUTYCYCLE_2 temp_2
#define DUTYCYCLE_3 temp_3
#define DUTYCYCLE_4 temp_4
```

```
#define V_A_REF temp_3
#define V_B_REF temp_4
#define V_C_REF temp_5

#define ESTADO_ACTUAL_A_UP temp_1
#define ESTADO_ACTUAL_B_UP temp_2
#define ESTADO_ACTUAL_C_UP temp_3
#define ESTADO_ACTUAL_A_LW temp_4
#define ESTADO_ACTUAL_B_LW temp_5
#define ESTADO_ACTUAL_C_LW temp_6

// Para el calculo de la tensión de cada rama del convertidor
promediado
#define V_A_UP_PROMED temp_1
#define V_B_UP_PROMED temp_2
#define V_C_UP_PROMED temp_3
#define V_A_LW_PROMED temp_4
#define V_B_LW_PROMED temp_5
#define V_C_LW_PROMED temp_6

// Variables corrientes circulantes
#define IZ_A temp_1
#define IZ_B temp_2
#define IZ_C temp_3
#define IZ temp_4


#ifdef MMC_REALISTA
#define I_UP_A_MMC01   readAO(1)
#define I_LW_A_MMC01   readAO(2)
#define I_UP_B_MMC01   readAO(3)
#define I_LW_B_MMC01   readAO(4)
#define I_UP_C_MMC01   readAO(5)
#define I_LW_C_MMC01   readAO(6)

#define V_A_GRID_MMC01 readAO(7)
#define V_B_GRID_MMC01 readAO(8)
#define V_C_GRID_MMC01 readAO(9)

#define VDC_MMC01      readAO(10)
#endif

#ifdef MMC_PROMEDIADO
#define I_UP_A_MMC02   readAO(11)
#define I_LW_A_MMC02   readAO(12)
#define I_UP_B_MMC02   readAO(13)
#define I_LW_B_MMC02   readAO(14)
#define I_UP_C_MMC02   readAO(15)
#define I_LW_C_MMC02   readAO(21)
```

```
#define V_A_GRID_MMC02 readAO(17)
#define V_B_GRID_MMC02 readAO(18)
#define V_C_GRID_MMC02 readAO(19)

#define VDC_MMC02      readAO(20)
#endif

#define V_DC_HALF    1600

#define INSERTED     1
#define BYPASS       0

#define INV_PERIOD   5000 // 1/200us
#define FREC_REF     314.1592654 // 2 x pi x 50

#define WXL_EQ_MMC01 0.39269908 // W = 2*pi*50; Leq = Lo +
0.5 * Larm
#define WXL_EQ_MMC02 0.39269908 // W = 2*pi*50; Leq = Lo +
0.5 * Larm

#define K_TS_CSM     0.0016666666666
// cte Ts/(n°celdas*Csm); n°celdas=4 SOLO EN PROMEDIADO

#ifdef MMC_REALISTA
#define KP_SYNC_MMC01    1
#define TS_KI_SYNC_MMC01 0.02

#define KP_IODQ_MMC01    0.2
#define TSXKI_IODQ_MMC01 0.002 // Ts * Ki_iod // 200us

#define KPV_DC_MMC01     1
#define TSKIV_DC_MMC01   0.0012
#endif

#ifdef MMC_PROMEDIADO
#define KP_SYNC_MMC02    1
#define TS_KI_SYNC_MMC02 0.02

#define KP_IODQ_MMC02    0.2
#define TSXKI_IODQ_MMC02 0.002 // Ts * Ki_iod // 200us

#define KPV_DC_MMC02     0.6
#define TSKIV_DC_MMC02   0.0025
#endif

#define PI_LIMIT_A       666 // 1000kW / 200V / 1.5 +20%

#define PERIODO          200e-6;

#define P_REF           -1000000
```

```
//
// Para la lectura de variables
//

double mmc01_phA_Vup[4];
double mmc01_phA_Vlw[4];

double mmc01_phB_Vup[4];
double mmc01_phB_Vlw[4];

double mmc01_phC_Vup[4];
double mmc01_phC_Vlw[4];



// Variables temporales
double temp_1, temp_2, temp_3, temp_4, temp_5, temp_6;

//
// Global variables
//

int syncP, syncN;

double Va_up_ref, Vb_up_ref, Vc_up_ref;
double Va_lw_ref, Vb_lw_ref, Vc_lw_ref;

double Vdc_ref;
double integralVdc;

double Vcond_UP_A_1, Vcond_UP_A_2, Vcond_UP_A_3,
       Vcond_UP_A_4, Vcond_LW_A_1, Vcond_LW_A_2,
       Vcond_LW_A_3, Vcond_LW_A_4;

double Vcond_UP_B_1, Vcond_UP_B_2, Vcond_UP_B_3,
       Vcond_UP_B_4, Vcond_LW_B_1, Vcond_LW_B_2,
       Vcond_LW_B_3, Vcond_LW_B_4;

double Vcond_UP_C_1, Vcond_UP_C_2, Vcond_UP_C_3,
       Vcond_UP_C_4, Vcond_LW_C_1, Vcond_LW_C_2,
       Vcond_LW_C_3, Vcond_LW_C_4;

double porcentaje_A_UP, porcentaje_B_UP, porcentaje_C_UP,
       porcentaje_A_LW, porcentaje_B_LW, porcentaje_C_LW;

double V_A_UP_ave, V_B_UP_ave, V_C_UP_ave, V_A_LW_ave,
       V_B_LW_ave, V_C_LW_ave;

double IupA, IupB, IupC;
double IlwA, IlwB, IlwC;
```

```
//
// Variables de sincronización con la RED
//

double V_d_ref, V_q_ref, integral_Vq;
double theta;
double seno_mmc01, coseno_mmc01;
double seno_mmc02, coseno_mmc02;
double w;

double Io_d, Io_q;
double Io_d_ref, Io_q_ref;
double integral_Iod, integral_Ioq;
double Vo_d_ref_conv, Vo_q_ref_conv;

//
// Temporales para operar por fallos del
compilador/optimizador
//

int pos;
int ESTADO_PREVIO;
double pos_vector;


//
// Variables vectoriales
//

double pS_mmc01[25];
double dutyCycle_mmc01[25];
double valores_mmc01[12];

double pS_mmc02[25];
double dutyCycle_mmc02[25];
double valores_mmc02[12];

double fuente_externa_UP[3];
double fuente_externa_LW[3];


//
// Funciones para calcular posiciones de Vmin, Vmax
//

int posVmin (int estado_buscado, double Vcond_1,
             double Vcond_2, double Vcond_3,
             double Vcond_4, double pS_1, double pS_2,
             double pS_3, double pS_4)
{
double Vmin;
```

```
Vmin = 1600;

int pos;
pos = -1;

if ((pS_1 == estado_buscado) && (Vcond_1 < Vmin)) {
pos = 0;
Vmin = Vcond_1;
}

if ((pS_2 == estado_buscado) && (Vcond_2 < Vmin)) {
pos = 1;
Vmin = Vcond_2;
}

if ((pS_3 == estado_buscado) && (Vcond_3 < Vmin)) {
pos = 2;
Vmin = Vcond_3;
}

if ((pS_4 == estado_buscado) && (Vcond_4 < Vmin)) {
pos = 3;
//Vmin = Vcond_4;
}

return pos;
}

int posVmax (int estado_buscado, double Vcond_1,
             double Vcond_2, double Vcond_3,
             double Vcond_4, double pS_1, double pS_2,
             double pS_3, double pS_4)
{
double Vmax;
Vmax = -1600;

int pos;
pos = -1;

if ((pS_1 == estado_buscado) && (Vcond_1 > Vmax)) {
pos = 0;
Vmax = Vcond_1;
}

if ((pS_2 == estado_buscado) && (Vcond_2 > Vmax)) {
pos = 1;
Vmax = Vcond_2;
}

if ((pS_3 == estado_buscado) && (Vcond_3 > Vmax)) {
pos = 2;
Vmax = Vcond_3;
}

if ((pS_4 == estado_buscado) && (Vcond_4 > Vmax)) {
pos = 3;
//Vmax = Vcond_4;
}

return pos;
}
```

2.2.-Setup

```
// Setup

Vuz = 0;
Vvz = 0;
Vwz = 0;
Vxz = 0;

syncP = 1;
syncN = 0;

// setup() ciclo de servicio mmc01
dutyCycle_mmc01[UP_A_1] = 0;
dutyCycle_mmc01[UP_A_2] = 1;
dutyCycle_mmc01[UP_A_3] = 0;
dutyCycle_mmc01[UP_A_4] = 1;

dutyCycle_mmc01[LW_A_1] = 0;
dutyCycle_mmc01[LW_A_2] = 1;
dutyCycle_mmc01[LW_A_3] = 0;
dutyCycle_mmc01[LW_A_4] = 1;

dutyCycle_mmc01[UP_B_1] = 0;
dutyCycle_mmc01[UP_B_2] = 1;
dutyCycle_mmc01[UP_B_3] = 0;
dutyCycle_mmc01[UP_B_4] = 1;

dutyCycle_mmc01[LW_B_1] = 0;
dutyCycle_mmc01[LW_B_2] = 1;
dutyCycle_mmc01[LW_B_3] = 0;
dutyCycle_mmc01[LW_B_4] = 1;

dutyCycle_mmc01[UP_C_1] = 0;
dutyCycle_mmc01[UP_C_2] = 1;
dutyCycle_mmc01[UP_C_3] = 0;
dutyCycle_mmc01[UP_C_4] = 1;

dutyCycle_mmc01[LW_C_1] = 0;
dutyCycle_mmc01[LW_C_2] = 1;
dutyCycle_mmc01[LW_C_3] = 0;
dutyCycle_mmc01[LW_C_4] = 1;

// setup() ciclo de servicio mmc02
dutyCycle_mmc02[UP_A_1] = 0;
dutyCycle_mmc02[UP_A_2] = 1;
dutyCycle_mmc02[UP_A_3] = 0;
dutyCycle_mmc02[UP_A_4] = 1;

dutyCycle_mmc02[LW_A_1] = 0;
```

```
dutyCycle_mmc02[LW_A_2] = 1;
dutyCycle_mmc02[LW_A_3] = 0;
dutyCycle_mmc02[LW_A_4] = 1;

dutyCycle_mmc02[UP_B_1] = 0;
dutyCycle_mmc02[UP_B_2] = 1;
dutyCycle_mmc02[UP_B_3] = 0;
dutyCycle_mmc02[UP_B_4] = 1;

dutyCycle_mmc02[LW_B_1] = 0;
dutyCycle_mmc02[LW_B_2] = 1;
dutyCycle_mmc02[LW_B_3] = 0;
dutyCycle_mmc02[LW_B_4] = 1;

dutyCycle_mmc02[UP_C_1] = 0;
dutyCycle_mmc02[UP_C_2] = 1;
dutyCycle_mmc02[UP_C_3] = 0;
dutyCycle_mmc02[UP_C_4] = 1;

dutyCycle_mmc02[LW_C_1] = 0;
dutyCycle_mmc02[LW_C_2] = 1;
dutyCycle_mmc02[LW_C_3] = 0;
dutyCycle_mmc02[LW_C_4] = 1;

// setup() estado polos mmc01
pS_mmc01[UP_A_1] = 0;
pS_mmc01[UP_A_2] = 1;
pS_mmc01[UP_A_3] = 0;
pS_mmc01[UP_A_4] = 1;

pS_mmc01[LW_A_1] = 0;
pS_mmc01[LW_A_2] = 1;
pS_mmc01[LW_A_3] = 0;
pS_mmc01[LW_A_4] = 1;

pS_mmc01[UP_B_1] = 0;
pS_mmc01[UP_B_2] = 1;
pS_mmc01[UP_B_3] = 0;
pS_mmc01[UP_B_4] = 1;

pS_mmc01[LW_B_1] = 0;
pS_mmc01[LW_B_2] = 1;
pS_mmc01[LW_B_3] = 0;
pS_mmc01[LW_B_4] = 1;

pS_mmc01[UP_C_1] = 0;
pS_mmc01[UP_C_2] = 1;
pS_mmc01[UP_C_3] = 0;
pS_mmc01[UP_C_4] = 1;

pS_mmc01[LW_C_1] = 0;
```

```
pS_mmc01[LW_C_2] = 1;
pS_mmc01[LW_C_3] = 0;
pS_mmc01[LW_C_4] = 1;

// setup() estado polos mmc02
pS_mmc02[UP_A_1] = 0;
pS_mmc02[UP_A_2] = 1;
pS_mmc02[UP_A_3] = 0;
pS_mmc02[UP_A_4] = 1;

pS_mmc02[LW_A_1] = 0;
pS_mmc02[LW_A_2] = 1;
pS_mmc02[LW_A_3] = 0;
pS_mmc02[LW_A_4] = 1;

pS_mmc02[UP_B_1] = 0;
pS_mmc02[UP_B_2] = 1;
pS_mmc02[UP_B_3] = 0;
pS_mmc02[UP_B_4] = 1;

pS_mmc02[LW_B_1] = 0;
pS_mmc02[LW_B_2] = 1;
pS_mmc02[LW_B_3] = 0;
pS_mmc02[LW_B_4] = 1;

pS_mmc02[UP_C_1] = 0;
pS_mmc02[UP_C_2] = 1;
pS_mmc02[UP_C_3] = 0;
pS_mmc02[UP_C_4] = 1;

pS_mmc02[LW_C_1] = 0;
pS_mmc02[LW_C_2] = 1;
pS_mmc02[LW_C_3] = 0;
pS_mmc02[LW_C_4] = 1;

V_A_UP_ave = 800;
V_B_UP_ave = 800;
V_C_UP_ave = 800;
V_A_LW_ave = 800;
V_B_LW_ave = 800;
V_C_LW_ave = 800;

ESTADO_PREVIO = 0;

theta = 0;
seno_mmc01      = sin(theta);
coseno_mmc01 = cos(theta);
seno_mmc02      = sin(theta);
coseno_mmc02 = cos(theta);

integral_Vq = 0;
```

```
w = 0;

Vdc_ref = 3200;

Io_d_ref = 0;
Io_q_ref = 0;

integralVdc = 0;

integral_Iod = 0;
integral_Ioq = 0;


valores_mmc01[7]  = 0; // integralVdc
valores_mmc01[8]  = 0; // theta
valores_mmc01[9]  = 0; // integralVq
valores_mmc01[10] = 0; // integral_Iod
valores_mmc01[11] = 0; // integral_Ioq


valores_mmc02[7]  = 0; // integralVdc
valores_mmc02[8]  = 0; // theta
valores_mmc02[9]  = 0; // integralVq
valores_mmc02[10] = 0; // integral_Iod
valores_mmc02[11] = 0; // integral_Ioq


fuente_externa_UP[0] = 1600;
fuente_externa_UP[1] = 1600;
fuente_externa_UP[2] = 1600;

fuente_externa_LW[0] = 1600;
fuente_externa_LW[1] = 1600;
fuente_externa_LW[2] = 1600;
```

2.3.-Loop

```
// Loop

// Generar salidas de PWM sincronizadas
#ifdef    MMC_REALISTA

readVector(dutyCycle_mmc01, 1, DUTYCYCLE_1, DUTYCYCLE_2,
           DUTYCYCLE_3, DUTYCYCLE_4);

pwmConfig(UP_A_1, TPWM, syncP, DUTYCYCLE_1);
pwmConfig(UP_A_2, TPWM, syncP, DUTYCYCLE_2);
pwmConfig(UP_A_3, TPWM, syncP, DUTYCYCLE_3);
pwmConfig(UP_A_4, TPWM, syncP, DUTYCYCLE_4);


readVector(dutyCycle_mmc01, 5, DUTYCYCLE_1, DUTYCYCLE_2,
           DUTYCYCLE_3, DUTYCYCLE_4);

pwmConfig(LW_A_1, TPWM, syncN, DUTYCYCLE_1);
pwmConfig(LW_A_2, TPWM, syncN, DUTYCYCLE_2);
pwmConfig(LW_A_3, TPWM, syncN, DUTYCYCLE_3);
pwmConfig(LW_A_4, TPWM, syncN, DUTYCYCLE_4);


readVector(dutyCycle_mmc01, 9, DUTYCYCLE_1, DUTYCYCLE_2,
           DUTYCYCLE_3, DUTYCYCLE_4);

pwmConfig(UP_B_1, TPWM, syncP, DUTYCYCLE_1);
pwmConfig(UP_B_2, TPWM, syncP, DUTYCYCLE_2);
pwmConfig(UP_B_3, TPWM, syncP, DUTYCYCLE_3);
pwmConfig(UP_B_4, TPWM, syncP, DUTYCYCLE_4);


readVector(dutyCycle_mmc01, 13, DUTYCYCLE_1, DUTYCYCLE_2,
           DUTYCYCLE_3, DUTYCYCLE_4);

pwmConfig(LW_B_1, TPWM, syncN, DUTYCYCLE_1);
pwmConfig(LW_B_2, TPWM, syncN, DUTYCYCLE_2);
pwmConfig(LW_B_3, TPWM, syncN, DUTYCYCLE_3);
pwmConfig(LW_B_4, TPWM, syncN, DUTYCYCLE_4);


readVector(dutyCycle_mmc01, 17, DUTYCYCLE_1, DUTYCYCLE_2,
           DUTYCYCLE_3, DUTYCYCLE_4);

pwmConfig(UP_C_1, TPWM, syncP, DUTYCYCLE_1);
pwmConfig(UP_C_2, TPWM, syncP, DUTYCYCLE_2);
pwmConfig(UP_C_3, TPWM, syncP, DUTYCYCLE_3);
pwmConfig(UP_C_4, TPWM, syncP, DUTYCYCLE_4);
```

```
readVector(dutyCycle_mmc01, 21, DUTYCYCLE_1, DUTYCYCLE_2,
           DUTYCYCLE_3, DUTYCYCLE_4);

pwmConfig(LW_C_1, TPWM, syncN, DUTYCYCLE_1);
pwmConfig(LW_C_2, TPWM, syncN, DUTYCYCLE_2);
pwmConfig(LW_C_3, TPWM, syncN, DUTYCYCLE_3);
pwmConfig(LW_C_4, TPWM, syncN, DUTYCYCLE_4);

#endif // MMC_REALISTA

// Definición de variables locales

double Io_a, Io_b, Io_c;

double Va_grid_mmc01, Vb_grid_mmc01, Vc_grid_mmc01;
double Va_grid_mmc02, Vb_grid_mmc02, Vc_grid_mmc02;

double V_dc;


#ifdef    MMC_PROMEDIADO
// PRIMERO: se leen las corrientes del 2° mmc
// Lectura de amperimetros del mmc02
IupA = I_UP_A_MMC02;
IlwA = I_LW_A_MMC02;
IupB = I_UP_B_MMC02;
IlwB = I_LW_B_MMC02;
IupC = I_UP_C_MMC02;
IlwC = I_LW_C_MMC02;

// Lectura de la tensión en continua
V_dc = VDC_MMC02;

// Lectura tensiones red
Va_grid_mmc02 = V_A_GRID_MMC02;
Vb_grid_mmc02 = V_B_GRID_MMC02;
Vc_grid_mmc02 = V_C_GRID_MMC02;

// meter valores en vector.

writeVector(valores_mmc02, 0, IupA, IlwA, IupB, IlwB, IupC,
            IlwC, V_dc);
#endif

#ifdef    MMC_REALISTA
// DESPUES: leer corrientes del primer mmc
IupA = I_UP_A_MMC01;
IlwA = I_LW_A_MMC01;
IupB = I_UP_B_MMC01;
IlwB = I_LW_B_MMC01;
```

```
IupC = I_UP_C_MMC01;
IlwC = I_LW_C_MMC01;

// Lectura de la tensión en continua
V_dc = VDC_MMC01;

// Lectura tensiones red
Va_grid_mmc01 = V_A_GRID_MMC01;
Vb_grid_mmc01 = V_B_GRID_MMC01;
Vc_grid_mmc01 = V_C_GRID_MMC01;

//#endif

// Lectura tensiones de los condensadores

//#ifdef  MMC_REALISTA
     readVector(mmc01_phA_Vup, 0, Vcond_UP_A_1,
                Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4);
     readVector(mmc01_phA_Vlw, 0, Vcond_LW_A_1,
                Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4);

     readVector(mmc01_phB_Vup, 0, Vcond_UP_B_1,
                Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4);
     readVector(mmc01_phB_Vlw, 0, Vcond_LW_B_1,
                Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4);

     readVector(mmc01_phC_Vup, 0, Vcond_UP_C_1,
                Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4);
     readVector(mmc01_phC_Vlw, 0, Vcond_LW_C_1,
                Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4);
#endif // MMC_REALISTA

/*
// Calculo de corrientes circulantes:
IZ_A = 0.5 * IupA + 0.5 * IlwA;
IZ_B = 0.5 * IupB + 0.5 * IlwB;
IZ_C = 0.5 * IupC + 0.5 * IlwC;

IZ = IZ_A + IZ_B + IZ_C;
*/

// Generación señal de sincronización
syncP = syncN;
syncN = (syncN == 0);


#ifdef    MMC_REALISTA

// RECUPERAR VALORES DE INTEGRACIÓN
readVector(valores_mmc01, 7, integralVdc, theta,
          integral_Vq, integral_Iod, integral_Ioq);
```

```
// Calculo referencia para mantener tensión de los
condensadores
#ifdef    MMC01_SLAVE
// error = VDC - Vdc_ref;
integralVdc = integralVdc + V_dc - Vdc_ref;
Io_d_ref = KPV_DC_MMC01 * V_dc - KPV_DC_MMC01 * Vdc_ref +
           TSKIV_DC_MMC01 * integralVdc;
Io_d_ref = constrain(Io_d_ref, -PI_LIMIT_A, PI_LIMIT_A);
#endif

// Sincronizacion con la RED, DQ_PLL
// ABC to DQ
uvw2dq    (Va_grid_mmc01, Vb_grid_mmc01, Vc_grid_mmc01,
           seno_mmc01, coseno_mmc01, V_d_ref, V_q_ref);


#ifdef    MMC01_MASTER
Io_d_ref = P_REF * 0.6666666667 / V_d_ref;
#endif


integral_Vq = integral_Vq + V_q_ref;
w = KP_SYNC_MMC01 * V_q_ref + TS_KI_SYNC_MMC01 *
    integral_Vq + FREC_REF;
theta = theta + loopPeriod() * w;
theta = wrapToPI(theta);

// Obtener referencias para sincronización
seno_mmc01 = sin(theta);
coseno_mmc01 = cos(theta);


// Calculos de referencias a partir de Io_d Io_q

Io_a = IupA - IlwA;
Io_b = IupB - IlwB;
Io_c = IupC - IlwC;

uvw2dq (Io_a, Io_b, Io_c, seno_mmc01, coseno_mmc01, Io_d,
        Io_q);

// errorIoD = Io_d_ref - Io_d;
integral_Iod = integral_Iod + Io_d_ref - Io_d;
Vo_d_ref_conv = KP_IODQ_MMC01 * Io_d_ref - KP_IODQ_MMC01 *
                Io_d + TSXKI_IODQ_MMC01 * integral_Iod –
                WXL_EQ_MMC01 * Io_q + V_d_ref;

// errorIoQ = Io_q_ref - Io_q;
integral_Ioq = integral_Ioq + Io_q_ref - Io_q;
Vo_q_ref_conv = KP_IODQ_MMC01 * Io_q_ref - KP_IODQ_MMC01 *
```

```
                Io_q + TSXKI_IODQ_MMC01 * integral_Ioq +
                WXL_EQ_MMC01 * Io_d + V_q_ref;


dq2uvw(Vo_d_ref_conv, Vo_q_ref_conv, seno_mmc01,
       coseno_mmc01, V_A_REF, V_B_REF, V_C_REF);

// Obtención tensiones de referencia (sincronizadas con la
red) superior e inferior para las celdas del MMC con 800 V
Va_up_ref = V_DC_HALF * 0.00125 - V_A_REF * 0.00125;
Vb_up_ref = V_DC_HALF * 0.00125 - V_B_REF * 0.00125;
Vc_up_ref = V_DC_HALF * 0.00125 - V_C_REF * 0.00125;

Va_lw_ref = V_DC_HALF * 0.00125 + V_A_REF * 0.00125;
Vb_lw_ref = V_DC_HALF * 0.00125 + V_B_REF * 0.00125;
Vc_lw_ref = V_DC_HALF * 0.00125 + V_C_REF * 0.00125;

// Obtención de los ciclos de trabajo de cada rama
porcentaje_A_UP = Va_up_ref - floor (Va_up_ref);
porcentaje_B_UP = Vb_up_ref - floor (Vb_up_ref);
porcentaje_C_UP = Vc_up_ref - floor (Vc_up_ref);

porcentaje_A_LW = Va_lw_ref - floor(Va_lw_ref);
porcentaje_B_LW = Vb_lw_ref - floor(Vb_lw_ref);
porcentaje_C_LW = Vc_lw_ref - floor(Vc_lw_ref);

// Estado actual
ESTADO_ACTUAL_A_UP = pS_mmc01[UP_A_1] + pS_mmc01[UP_A_2] +
                     pS_mmc01[UP_A_3] + pS_mmc01[UP_A_4];

ESTADO_ACTUAL_B_UP = pS_mmc01[UP_B_1] + pS_mmc01[UP_B_2] +
                     pS_mmc01[UP_B_3] + pS_mmc01[UP_B_4];

ESTADO_ACTUAL_C_UP = pS_mmc01[UP_C_1] + pS_mmc01[UP_C_2] +
                     pS_mmc01[UP_C_3] + pS_mmc01[UP_C_4];


ESTADO_ACTUAL_A_LW = pS_mmc01[LW_A_1] + pS_mmc01[LW_A_2] +
                     pS_mmc01[LW_A_3] + pS_mmc01[LW_A_4];

ESTADO_ACTUAL_B_LW = pS_mmc01[LW_B_1] + pS_mmc01[LW_B_2] +
                     pS_mmc01[LW_B_3] + pS_mmc01[LW_B_4];

ESTADO_ACTUAL_C_LW = pS_mmc01[LW_C_1] + pS_mmc01[LW_C_2] +
                     pS_mmc01[LW_C_3] + pS_mmc01[LW_C_4];


// VOLCADO DE VALORES AL VECTOR PARA CONSERVARLOS PARA LA
PRÓXIMA ITERACIÓN
writeVector(valores_mmc01, 7, integralVdc, theta,
            integral_Vq, integral_Iod, integral_Ioq);
```

```
// Elección del que cambia y asignación de porcentaje

// **************** A_UP ****************

ESTADO_PREVIO = ESTADO_ACTUAL_A_UP;

dutyCycle_mmc01[UP_A_1] = pS_mmc01[UP_A_1];
dutyCycle_mmc01[UP_A_2] = pS_mmc01[UP_A_2];
dutyCycle_mmc01[UP_A_3] = pS_mmc01[UP_A_3];
dutyCycle_mmc01[UP_A_4] = pS_mmc01[UP_A_4];

while (Va_up_ref > ESTADO_ACTUAL_A_UP) {
     if (IupA > 0) {
          pos = posVmin (BYPASS, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc01[UP_A_1], pS_mmc01[UP_A_2],
          pS_mmc01[UP_A_3], pS_mmc01[UP_A_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc01[UP_A_1], pS_mmc01[UP_A_2],
          pS_mmc01[UP_A_3], pS_mmc01[UP_A_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_A_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_A_UP;
          pS_mmc01[pos_vector]          = 1;
          porcentaje_A_UP               = 1;
     }
     ESTADO_ACTUAL_A_UP = ESTADO_ACTUAL_A_UP + 1;
}

ESTADO_ACTUAL_A_UP = ESTADO_PREVIO;

while (Va_up_ref < ESTADO_ACTUAL_A_UP) {
     if (IupA > 0) {
          pos = posVmax (INSERTED, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc01[UP_A_1], pS_mmc01[UP_A_2],
          pS_mmc01[UP_A_3], pS_mmc01[UP_A_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc01[UP_A_1], pS_mmc01[UP_A_2],
          pS_mmc01[UP_A_3], pS_mmc01[UP_A_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_A_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_A_UP;
          pS_mmc01[pos_vector]          = 0;
          porcentaje_A_UP               = 0;
     }
     ESTADO_ACTUAL_A_UP = ESTADO_ACTUAL_A_UP - 1;
}

// **************** B_UP ****************

ESTADO_PREVIO = ESTADO_ACTUAL_B_UP;

dutyCycle_mmc01[UP_B_1] = pS_mmc01[UP_B_1];
dutyCycle_mmc01[UP_B_2] = pS_mmc01[UP_B_2];
dutyCycle_mmc01[UP_B_3] = pS_mmc01[UP_B_3];
dutyCycle_mmc01[UP_B_4] = pS_mmc01[UP_B_4];

while (Vb_up_ref > ESTADO_ACTUAL_B_UP) {
     if (IupB > 0) {
          pos = posVmin (BYPASS, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc01[UP_B_1], pS_mmc01[UP_B_2],
          pS_mmc01[UP_B_3], pS_mmc01[UP_B_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc01[UP_B_1], pS_mmc01[UP_B_2],
          pS_mmc01[UP_B_3], pS_mmc01[UP_B_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_B_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_B_UP;
          pS_mmc01[pos_vector]          = 1;
          porcentaje_B_UP               = 1;
     }
     ESTADO_ACTUAL_B_UP = ESTADO_ACTUAL_B_UP + 1;
}

ESTADO_ACTUAL_B_UP = ESTADO_PREVIO;

while (Vb_up_ref < ESTADO_ACTUAL_B_UP) {
     if (IupB > 0) {
          pos = posVmax (INSERTED, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc01[UP_B_1], pS_mmc01[UP_B_2],
          pS_mmc01[UP_B_3], pS_mmc01[UP_B_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc01[UP_B_1], pS_mmc01[UP_B_2],
          pS_mmc01[UP_B_3], pS_mmc01[UP_B_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_B_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_B_UP;
          pS_mmc01[pos_vector]          = 0;
          porcentaje_B_UP               = 0;
     }
     ESTADO_ACTUAL_B_UP = ESTADO_ACTUAL_B_UP - 1;
}

// **************** C_UP ****************

ESTADO_PREVIO = ESTADO_ACTUAL_C_UP;

dutyCycle_mmc01[UP_C_1] = pS_mmc01[UP_C_1];
dutyCycle_mmc01[UP_C_2] = pS_mmc01[UP_C_2];
dutyCycle_mmc01[UP_C_3] = pS_mmc01[UP_C_3];
dutyCycle_mmc01[UP_C_4] = pS_mmc01[UP_C_4];

while (Vc_up_ref > ESTADO_ACTUAL_C_UP) {
     if (IupC > 0) {
          pos = posVmin (BYPASS, Vcond_UP_C_1,
          Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4,
          pS_mmc01[UP_C_1], pS_mmc01[UP_C_2],
          pS_mmc01[UP_C_3], pS_mmc01[UP_C_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_UP_C_1,
          Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4,
          pS_mmc01[UP_C_1], pS_mmc01[UP_C_2],
          pS_mmc01[UP_C_3], pS_mmc01[UP_C_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_C_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_C_UP;
          pS_mmc01[pos_vector]          = 1;
          porcentaje_C_UP               = 1;
     }
     ESTADO_ACTUAL_C_UP = ESTADO_ACTUAL_C_UP + 1;
}

ESTADO_ACTUAL_C_UP = ESTADO_PREVIO;

while (Vc_up_ref < ESTADO_ACTUAL_C_UP) {
     if (IupC > 0) {
          pos = posVmax (INSERTED, Vcond_UP_C_1,
          Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4,
          pS_mmc01[UP_C_1], pS_mmc01[UP_C_2],
          pS_mmc01[UP_C_3], pS_mmc01[UP_C_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_UP_C_1,
          Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4,
          pS_mmc01[UP_C_1], pS_mmc01[UP_C_2],
          pS_mmc01[UP_C_3], pS_mmc01[UP_C_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_C_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_C_UP;
          pS_mmc01[pos_vector]          = 0;
          porcentaje_C_UP               = 0;
     }
     ESTADO_ACTUAL_C_UP = ESTADO_ACTUAL_C_UP - 1;
}

// ***************** A_LW *****************

ESTADO_PREVIO = ESTADO_ACTUAL_A_LW;

dutyCycle_mmc01[LW_A_1] = pS_mmc01[LW_A_1];
dutyCycle_mmc01[LW_A_2] = pS_mmc01[LW_A_2];
dutyCycle_mmc01[LW_A_3] = pS_mmc01[LW_A_3];
dutyCycle_mmc01[LW_A_4] = pS_mmc01[LW_A_4];

while (Va_lw_ref > ESTADO_ACTUAL_A_LW) {
     if (IlwA > 0) {
          pos = posVmin (BYPASS, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc01[LW_A_1], pS_mmc01[LW_A_2],
          pS_mmc01[LW_A_3], pS_mmc01[LW_A_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc01[LW_A_1], pS_mmc01[LW_A_2],
          pS_mmc01[LW_A_3], pS_mmc01[LW_A_4]);
     }
     if (pos >= 0){
          pos_vector                    = LW_A_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_A_LW;
          pS_mmc01[pos_vector]          = 1;
          porcentaje_A_LW               = 1;
     }
     ESTADO_ACTUAL_A_LW = ESTADO_ACTUAL_A_LW + 1;
}

ESTADO_ACTUAL_A_LW = ESTADO_PREVIO;

while (Va_lw_ref < ESTADO_ACTUAL_A_LW) {
     if (IlwA > 0) {
          pos = posVmax (INSERTED, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc01[LW_A_1], pS_mmc01[LW_A_2],
          pS_mmc01[LW_A_3], pS_mmc01[LW_A_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc01[LW_A_1], pS_mmc01[LW_A_2],
          pS_mmc01[LW_A_3], pS_mmc01[LW_A_4]);
     }
     if (pos >= 0){
          pos_vector                    = LW_A_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_A_LW;
          pS_mmc01[pos_vector]          = 0;
          porcentaje_A_LW               = 0;
     }
     ESTADO_ACTUAL_A_LW = ESTADO_ACTUAL_A_LW - 1;
}

// **************** B_LW ****************

ESTADO_PREVIO = ESTADO_ACTUAL_B_LW;

dutyCycle_mmc01[LW_B_1] = pS_mmc01[LW_B_1];
dutyCycle_mmc01[LW_B_2] = pS_mmc01[LW_B_2];
dutyCycle_mmc01[LW_B_3] = pS_mmc01[LW_B_3];
dutyCycle_mmc01[LW_B_4] = pS_mmc01[LW_B_4];

while (Vb_lw_ref > ESTADO_ACTUAL_B_LW) {
     if (IlwB > 0) {
          pos = posVmin (BYPASS, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc01[LW_B_1], pS_mmc01[LW_B_2],
          pS_mmc01[LW_B_3], pS_mmc01[LW_B_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc01[LW_B_1], pS_mmc01[LW_B_2],
          pS_mmc01[LW_B_3], pS_mmc01[LW_B_4]);
     }
     if (pos >= 0){
          pos_vector                    = LW_B_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_B_LW;
          pS_mmc01[pos_vector]          = 1;
          porcentaje_B_LW               = 1;
     }
     ESTADO_ACTUAL_B_LW = ESTADO_ACTUAL_B_LW + 1;
}

ESTADO_ACTUAL_B_LW = ESTADO_PREVIO;

while (Vb_lw_ref < ESTADO_ACTUAL_B_LW) {
     if (IlwB > 0) {
          pos = posVmax (INSERTED, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc01[LW_B_1], pS_mmc01[LW_B_2],
          pS_mmc01[LW_B_3], pS_mmc01[LW_B_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc01[LW_B_1], pS_mmc01[LW_B_2],
          pS_mmc01[LW_B_3], pS_mmc01[LW_B_4]);
     }
     if (pos >= 0){
          pos_vector                      = LW_B_1 + pos;
          dutyCycle_mmc01[pos_vector]     = porcentaje_B_LW;
          pS_mmc01[pos_vector]            = 0;
          porcentaje_B_LW                 = 0;
     }
     ESTADO_ACTUAL_B_LW = ESTADO_ACTUAL_B_LW - 1;
}

// **************** C_LW ****************

ESTADO_PREVIO = ESTADO_ACTUAL_C_LW;

dutyCycle_mmc01[LW_C_1] = pS_mmc01[LW_C_1];
dutyCycle_mmc01[LW_C_2] = pS_mmc01[LW_C_2];
dutyCycle_mmc01[LW_C_3] = pS_mmc01[LW_C_3];
dutyCycle_mmc01[LW_C_4] = pS_mmc01[LW_C_4];

while (Vc_lw_ref > ESTADO_ACTUAL_C_LW) {
     if (IlwC > 0) {
          pos = posVmin (BYPASS, Vcond_LW_C_1,
          Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
          pS_mmc01[LW_C_1], pS_mmc01[LW_C_2],
          pS_mmc01[LW_C_3], pS_mmc01[LW_C_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_LW_C_1,
          Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
          pS_mmc01[LW_C_1], pS_mmc01[LW_C_2],
          pS_mmc01[LW_C_3], pS_mmc01[LW_C_4]);
     }
     if (pos >= 0){
          pos_vector                      = LW_C_1 + pos;
          dutyCycle_mmc01[pos_vector]     = porcentaje_C_LW;
          pS_mmc01[pos_vector]            = 1;
          porcentaje_C_LW                 = 1;
     }
     ESTADO_ACTUAL_C_LW = ESTADO_ACTUAL_C_LW + 1;
}

ESTADO_ACTUAL_C_LW = ESTADO_PREVIO;

while (Vc_lw_ref < ESTADO_ACTUAL_C_LW) {
     if (IlwC > 0) {
          pos = posVmax (INSERTED, Vcond_LW_C_1,
          Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
          pS_mmc01[LW_C_1], pS_mmc01[LW_C_2],
          pS_mmc01[LW_C_3], pS_mmc01[LW_C_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_LW_C_1,
          Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
          pS_mmc01[LW_C_1], pS_mmc01[LW_C_2],
          pS_mmc01[LW_C_3], pS_mmc01[LW_C_4]);
     }
     if (pos >= 0){
          pos_vector                    = LW_C_1 + pos;
          dutyCycle_mmc01[pos_vector]   = porcentaje_C_LW;
          pS_mmc01[pos_vector]          = 0;
          porcentaje_C_LW               = 0;
     }
     ESTADO_ACTUAL_C_LW = ESTADO_ACTUAL_C_LW - 1;
}

#endif

//
// SEGUNDO CONVERTIDOR
//

#ifdef    MMC_PROMEDIADO

// RECUPERAR VALORES MEDIDOS AL INICIO Y RECORDAR VALOR DE
VARIABLES INTEGRADAS

readVector(valores_mmc02, 0, IupA, IlwA, IupB, IlwB, IupC,
           IlwC, V_dc, integralVdc, theta, integral_Vq,
           integral_Iod, integral_Ioq);

/*
// Calculo de corrientes circulantes:
IZ_A = 0.5 * IupA + 0.5 * IlwA;
IZ_B = 0.5 * IupB + 0.5 * IlwB;
IZ_C = 0.5 * IupC + 0.5 * IlwC;

IZ = IZ_A + IZ_B + IZ_C;

*/

V_A_UP_ave = V_A_UP_ave + (dutyCycle_mmc02[UP_A_1] +
dutyCycle_mmc02[UP_A_2] + dutyCycle_mmc02[UP_A_3] +
dutyCycle_mmc02[UP_A_4]) * K_TS_CSM * IupA;

V_B_UP_ave = V_B_UP_ave + (dutyCycle_mmc02[UP_B_1] +
dutyCycle_mmc02[UP_B_2] + dutyCycle_mmc02[UP_B_3] +
dutyCycle_mmc02[UP_B_4]) * K_TS_CSM * IupB;
```

```
V_C_UP_ave = V_C_UP_ave + (dutyCycle_mmc02[UP_C_1] +
dutyCycle_mmc02[UP_C_2] + dutyCycle_mmc02[UP_C_3] +
dutyCycle_mmc02[UP_C_4]) * K_TS_CSM * IupC;

V_A_LW_ave = V_A_LW_ave + (dutyCycle_mmc02[LW_A_1] +
dutyCycle_mmc02[LW_A_2] + dutyCycle_mmc02[LW_A_3] +
dutyCycle_mmc02[LW_A_4]) * K_TS_CSM * IlwA;

V_B_LW_ave = V_B_LW_ave + (dutyCycle_mmc02[LW_B_1] +
dutyCycle_mmc02[LW_B_2] + dutyCycle_mmc02[LW_B_3] +
dutyCycle_mmc02[LW_B_4]) * K_TS_CSM * IlwB;

V_C_LW_ave = V_C_LW_ave + (dutyCycle_mmc02[LW_C_1] +
dutyCycle_mmc02[LW_C_2] + dutyCycle_mmc02[LW_C_3] +
dutyCycle_mmc02[LW_C_4]) * K_TS_CSM * IlwC;


Vcond_UP_A_1 = V_A_UP_ave;
Vcond_UP_A_2 = V_A_UP_ave;
Vcond_UP_A_3 = V_A_UP_ave;
Vcond_UP_A_4 = V_A_UP_ave;

Vcond_LW_A_1 = V_A_LW_ave;
Vcond_LW_A_2 = V_A_LW_ave;
Vcond_LW_A_3 = V_A_LW_ave;
Vcond_LW_A_4 = V_A_LW_ave;

Vcond_UP_B_1 = V_B_UP_ave;
Vcond_UP_B_2 = V_B_UP_ave;
Vcond_UP_B_3 = V_B_UP_ave;
Vcond_UP_B_4 = V_B_UP_ave;

Vcond_LW_B_1 = V_B_LW_ave;
Vcond_LW_B_2 = V_B_LW_ave;
Vcond_LW_B_3 = V_B_LW_ave;
Vcond_LW_B_4 = V_B_LW_ave;

Vcond_UP_C_1 = V_C_UP_ave;
Vcond_UP_C_2 = V_C_UP_ave;
Vcond_UP_C_3 = V_C_UP_ave;
Vcond_UP_C_4 = V_C_UP_ave;

Vcond_LW_C_1 = V_C_LW_ave;
Vcond_LW_C_2 = V_C_LW_ave;
Vcond_LW_C_3 = V_C_LW_ave;
Vcond_LW_C_4 = V_C_LW_ave;

// Calculo referencia para mantener tensión de los
condensadores
// Si el MMC01 esta en master, el 02 esta en slave
#ifdef    MMC01_MASTER
```

```
// error = VDC_MMC02 - Vdc_ref;
integralVdc = integralVdc + V_dc - Vdc_ref;
Io_d_ref = KPV_DC_MMC02 * V_dc - KPV_DC_MMC02 * Vdc_ref +
           TSKIV_DC_MMC02 * integralVdc;
Io_d_ref = constrain(Io_d_ref, -PI_LIMIT_A, PI_LIMIT_A);
#endif

// Sincronizacion con la RED, DQ_PLL
uvw2dq    (Va_grid_mmc02, Vb_grid_mmc02, Vc_grid_mmc02,
           seno_mmc02, coseno_mmc02, V_d_ref, V_q_ref);

// Si el MMC01 esta en slave, el MMC02 esta en master
#ifdef    MMC01_SLAVE
Io_d_ref = P_REF * 0.6666666667 / V_d_ref;
#endif

integral_Vq = integral_Vq + V_q_ref;
w = KP_SYNC_MMC02 * V_q_ref + TS_KI_SYNC_MMC02 *
    integral_Vq + FREC_REF;
theta = theta + loopPeriod() * w;
theta = wrapToPI(theta);

// Obtener referencias para sincronización
seno_mmc02 = sin(theta);
coseno_mmc02 = cos(theta);

// Calculos de referencias a partir de Io_d Io_q

Io_a = IupA - IlwA;
Io_b = IupB - IlwB;
Io_c = IupC - IlwC;

uvw2dq (Io_a, Io_b, Io_c, seno_mmc02, coseno_mmc02, Io_d,
        Io_q);

// errorIoD = Io_d_ref - Io_d;
integral_Iod = integral_Iod + Io_d_ref - Io_d;
Vo_d_ref_conv = KP_IODQ_MMC02 * Io_d_ref - KP_IODQ_MMC02 *
                Io_d + TSXKI_IODQ_MMC02 * integral_Iod –
                WXL_EQ_MMC02 * Io_q + V_d_ref;

// errorIoQ = Io_q_ref - Io_q;
integral_Ioq = integral_Ioq + Io_q_ref - Io_q;
Vo_q_ref_conv = KP_IODQ_MMC02 * Io_q_ref - KP_IODQ_MMC02 *
                Io_q + TSXKI_IODQ_MMC02 * integral_Ioq +
                WXL_EQ_MMC02 * Io_d + V_q_ref;

dq2uvw(Vo_d_ref_conv, Vo_q_ref_conv, seno_mmc02,
       coseno_mmc02, V_A_REF, V_B_REF, V_C_REF);
```

```
// Obtención tensiones de referencia (sincronizadas con la
red) superior e inferior para las celdas del MMC con 800 V
Va_up_ref = V_DC_HALF * 0.00125 - V_A_REF * 0.00125;
Vb_up_ref = V_DC_HALF * 0.00125 - V_B_REF * 0.00125;
Vc_up_ref = V_DC_HALF * 0.00125 - V_C_REF * 0.00125;
Va_lw_ref = V_DC_HALF * 0.00125 + V_A_REF * 0.00125;
Vb_lw_ref = V_DC_HALF * 0.00125 + V_B_REF * 0.00125;
Vc_lw_ref = V_DC_HALF * 0.00125 + V_C_REF * 0.00125;

// Obtención de los ciclos de trabajo de cada rama
porcentaje_A_UP = Va_up_ref - floor (Va_up_ref);
porcentaje_B_UP = Vb_up_ref - floor (Vb_up_ref);
porcentaje_C_UP = Vc_up_ref - floor (Vc_up_ref);

porcentaje_A_LW = Va_lw_ref - floor(Va_lw_ref);
porcentaje_B_LW = Vb_lw_ref - floor(Vb_lw_ref);
porcentaje_C_LW = Vc_lw_ref - floor(Vc_lw_ref);

// Estado actual
ESTADO_ACTUAL_A_UP = pS_mmc02[UP_A_1] + pS_mmc02[UP_A_2] +
                     pS_mmc02[UP_A_3] + pS_mmc02[UP_A_4];

ESTADO_ACTUAL_B_UP = pS_mmc02[UP_B_1] + pS_mmc02[UP_B_2] +
                     pS_mmc02[UP_B_3] + pS_mmc02[UP_B_4];

ESTADO_ACTUAL_C_UP = pS_mmc02[UP_C_1] + pS_mmc02[UP_C_2] +
                     pS_mmc02[UP_C_3] + pS_mmc02[UP_C_4];


ESTADO_ACTUAL_A_LW = pS_mmc02[LW_A_1] + pS_mmc02[LW_A_2] +
                     pS_mmc02[LW_A_3] + pS_mmc02[LW_A_4];

ESTADO_ACTUAL_B_LW = pS_mmc02[LW_B_1] + pS_mmc02[LW_B_2] +
                     pS_mmc02[LW_B_3] + pS_mmc02[LW_B_4];

ESTADO_ACTUAL_C_LW = pS_mmc02[LW_C_1] + pS_mmc02[LW_C_2] +
                     pS_mmc02[LW_C_3] + pS_mmc02[LW_C_4];


// VOLCADO DE VALORES AL VECTOR PARA CONSERVARLOS PARA LA
PRÓXIMA ITERACIÓN
writeVector(valores_mmc02, 7, integralVdc, theta,
            integral_Vq, integral_Iod, integral_Ioq);


// Elección del que cambia y asignación de porcentaje

// ***************** A_UP *****************

ESTADO_PREVIO = ESTADO_ACTUAL_A_UP;
```

```
dutyCycle_mmc02[UP_A_1] = pS_mmc02[UP_A_1];
dutyCycle_mmc02[UP_A_2] = pS_mmc02[UP_A_2];
dutyCycle_mmc02[UP_A_3] = pS_mmc02[UP_A_3];
dutyCycle_mmc02[UP_A_4] = pS_mmc02[UP_A_4];

while (Va_up_ref > ESTADO_ACTUAL_A_UP) {
     if (IupA > 0) {
          pos = posVmin (BYPASS, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc02[UP_A_1], pS_mmc02[UP_A_2],
          pS_mmc02[UP_A_3], pS_mmc02[UP_A_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc02[UP_A_1], pS_mmc02[UP_A_2],
          pS_mmc02[UP_A_3], pS_mmc02[UP_A_4]);
     }

     if (pos >= 0){
          pos_vector                    = UP_A_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_A_UP;
          pS_mmc02[pos_vector]          = 1;
          porcentaje_A_UP               = 1;
     }
     ESTADO_ACTUAL_A_UP = ESTADO_ACTUAL_A_UP + 1;
}

ESTADO_ACTUAL_A_UP = ESTADO_PREVIO;

while (Va_up_ref < ESTADO_ACTUAL_A_UP) {
     if (IupA > 0) {
          pos = posVmax (INSERTED, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc02[UP_A_1], pS_mmc02[UP_A_2],
          pS_mmc02[UP_A_3], pS_mmc02[UP_A_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_UP_A_1,
          Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
          pS_mmc02[UP_A_1], pS_mmc02[UP_A_2],
          pS_mmc02[UP_A_3], pS_mmc02[UP_A_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_A_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_A_UP;
          pS_mmc02[pos_vector]          = 0;
          porcentaje_A_UP               = 0;
     }
     ESTADO_ACTUAL_A_UP = ESTADO_ACTUAL_A_UP - 1;
}

// **************** B_UP ****************
```

```
ESTADO_PREVIO = ESTADO_ACTUAL_B_UP;

dutyCycle_mmc02[UP_B_1] = pS_mmc02[UP_B_1];
dutyCycle_mmc02[UP_B_2] = pS_mmc02[UP_B_2];
dutyCycle_mmc02[UP_B_3] = pS_mmc02[UP_B_3];
dutyCycle_mmc02[UP_B_4] = pS_mmc02[UP_B_4];

while (Vb_up_ref > ESTADO_ACTUAL_B_UP) {
     if (IupB > 0) {
          pos = posVmin (BYPASS, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc02[UP_B_1], pS_mmc02[UP_B_2],
          pS_mmc02[UP_B_3], pS_mmc02[UP_B_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc02[UP_B_1], pS_mmc02[UP_B_2],
          pS_mmc02[UP_B_3], pS_mmc02[UP_B_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_B_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_B_UP;
          pS_mmc02[pos_vector]          = 1;
          porcentaje_B_UP               = 1;
     }
     ESTADO_ACTUAL_B_UP = ESTADO_ACTUAL_B_UP + 1;
}

ESTADO_ACTUAL_B_UP = ESTADO_PREVIO;

while (Vb_up_ref < ESTADO_ACTUAL_B_UP) {
     if (IupB > 0) {
          pos = posVmax (INSERTED, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc02[UP_B_1], pS_mmc02[UP_B_2],
          pS_mmc02[UP_B_3], pS_mmc02[UP_B_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_UP_B_1,
          Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4,
          pS_mmc02[UP_B_1], pS_mmc02[UP_B_2],
          pS_mmc02[UP_B_3], pS_mmc02[UP_B_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_B_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_B_UP;
          pS_mmc02[pos_vector]          = 0;
          porcentaje_B_UP               = 0;
     }
     ESTADO_ACTUAL_B_UP = ESTADO_ACTUAL_B_UP - 1;
}
```

```
// **************** C_UP ****************

ESTADO_PREVIO = ESTADO_ACTUAL_C_UP;

dutyCycle_mmc02[UP_C_1] = pS_mmc02[UP_C_1];
dutyCycle_mmc02[UP_C_2] = pS_mmc02[UP_C_2];
dutyCycle_mmc02[UP_C_3] = pS_mmc02[UP_C_3];
dutyCycle_mmc02[UP_C_4] = pS_mmc02[UP_C_4];

while (Vc_up_ref > ESTADO_ACTUAL_C_UP) {
     if (IupC > 0) {
          pos = posVmin (BYPASS, Vcond_UP_C_1,
          Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4,
          pS_mmc02[UP_C_1], pS_mmc02[UP_C_2],
          pS_mmc02[UP_C_3], pS_mmc02[UP_C_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_UP_C_1,
Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4, pS_mmc02[UP_C_1],
pS_mmc02[UP_C_2], pS_mmc02[UP_C_3], pS_mmc02[UP_C_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_C_1 + pos;
          dutyCycle_mmc02[pos_vector] = porcentaje_C_UP;
          pS_mmc02[pos_vector]          = 1;
          porcentaje_C_UP               = 1;
     }
     ESTADO_ACTUAL_C_UP            = ESTADO_ACTUAL_C_UP +
1;
}

ESTADO_ACTUAL_C_UP = ESTADO_PREVIO;

while (Vc_up_ref < ESTADO_ACTUAL_C_UP) {
     if (IupC > 0) {
          pos = posVmax (INSERTED, Vcond_UP_C_1,
          Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4,
          pS_mmc02[UP_C_1], pS_mmc02[UP_C_2],
          pS_mmc02[UP_C_3], pS_mmc02[UP_C_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_UP_C_1,
          Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4,
          pS_mmc02[UP_C_1], pS_mmc02[UP_C_2],
          pS_mmc02[UP_C_3], pS_mmc02[UP_C_4]);
     }
     if (pos >= 0){
          pos_vector                    = UP_C_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_C_UP;
          pS_mmc02[pos_vector]          = 0;
          porcentaje_C_UP               = 0;
     }
     ESTADO_ACTUAL_C_UP = ESTADO_ACTUAL_C_UP - 1;
}

// **************** A_LW ****************

ESTADO_PREVIO = ESTADO_ACTUAL_A_LW;

dutyCycle_mmc02[LW_A_1] = pS_mmc02[LW_A_1];
dutyCycle_mmc02[LW_A_2] = pS_mmc02[LW_A_2];
dutyCycle_mmc02[LW_A_3] = pS_mmc02[LW_A_3];
dutyCycle_mmc02[LW_A_4] = pS_mmc02[LW_A_4];

while (Va_lw_ref > ESTADO_ACTUAL_A_LW) {
     if (IlwA > 0) {
          pos = posVmin (BYPASS, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc02[LW_A_1], pS_mmc02[LW_A_2],
          pS_mmc02[LW_A_3], pS_mmc02[LW_A_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc02[LW_A_1], pS_mmc02[LW_A_2],
          pS_mmc02[LW_A_3], pS_mmc02[LW_A_4]);
     }
     if (pos >= 0){
          pos_vector                    = LW_A_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_A_LW;
          pS_mmc02[pos_vector]          = 1;
          porcentaje_A_LW               = 1;
     }
     ESTADO_ACTUAL_A_LW = ESTADO_ACTUAL_A_LW + 1;
}

ESTADO_ACTUAL_A_LW = ESTADO_PREVIO;

while (Va_lw_ref < ESTADO_ACTUAL_A_LW) {
     if (IlwA > 0) {
          pos = posVmax (INSERTED, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc02[LW_A_1], pS_mmc02[LW_A_2],
          pS_mmc02[LW_A_3], pS_mmc02[LW_A_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_LW_A_1,
          Vcond_LW_A_2, Vcond_LW_A_3, Vcond_LW_A_4,
          pS_mmc02[LW_A_1], pS_mmc02[LW_A_2],
          pS_mmc02[LW_A_3], pS_mmc02[LW_A_4]);
     }
     if (pos >= 0){
          pos_vector                    = LW_A_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_A_LW;
          pS_mmc02[pos_vector]          = 0;
          porcentaje_A_LW                = 0;
     }
     ESTADO_ACTUAL_A_LW = ESTADO_ACTUAL_A_LW - 1;
}

// **************** B_LW ****************

ESTADO_PREVIO = ESTADO_ACTUAL_B_LW;

dutyCycle_mmc02[LW_B_1] = pS_mmc02[LW_B_1];
dutyCycle_mmc02[LW_B_2] = pS_mmc02[LW_B_2];
dutyCycle_mmc02[LW_B_3] = pS_mmc02[LW_B_3];
dutyCycle_mmc02[LW_B_4] = pS_mmc02[LW_B_4];

while (Vb_lw_ref > ESTADO_ACTUAL_B_LW) {
     if (IlwB > 0) {
          pos = posVmin (BYPASS, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc02[LW_B_1], pS_mmc02[LW_B_2],
          pS_mmc02[LW_B_3], pS_mmc02[LW_B_4]);
     } else {
          pos = posVmax (BYPASS, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc02[LW_B_1], pS_mmc02[LW_B_2],
          pS_mmc02[LW_B_3], pS_mmc02[LW_B_4]);
     }
     if (pos >= 0){
          pos_vector                     = LW_B_1 + pos;
          dutyCycle_mmc02[pos_vector]    = porcentaje_B_LW;
          pS_mmc02[pos_vector]           = 1;
          porcentaje_B_LW                = 1;
     }
     ESTADO_ACTUAL_B_LW = ESTADO_ACTUAL_B_LW + 1;
}

ESTADO_ACTUAL_B_LW = ESTADO_PREVIO;

while (Vb_lw_ref < ESTADO_ACTUAL_B_LW) {
     if (IlwB > 0) {
          pos = posVmax (INSERTED, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc02[LW_B_1], pS_mmc02[LW_B_2],
          pS_mmc02[LW_B_3], pS_mmc02[LW_B_4]);
     } else {
          pos = posVmin (INSERTED, Vcond_LW_B_1,
          Vcond_LW_B_2, Vcond_LW_B_3, Vcond_LW_B_4,
          pS_mmc02[LW_B_1], pS_mmc02[LW_B_2],
          pS_mmc02[LW_B_3], pS_mmc02[LW_B_4]);
     }
     if (pos >= 0){
          pos_vector                     = LW_B_1 + pos;
```

```
			dutyCycle_mmc02[pos_vector]    = porcentaje_B_LW;
			pS_mmc02[pos_vector]           = 0;
			porcentaje_B_LW                = 0;
		}
		ESTADO_ACTUAL_B_LW = ESTADO_ACTUAL_B_LW - 1;
	}

	// ***************** C_LW *****************

	ESTADO_PREVIO = ESTADO_ACTUAL_C_LW;

	dutyCycle_mmc02[LW_C_1] = pS_mmc02[LW_C_1];
	dutyCycle_mmc02[LW_C_2] = pS_mmc02[LW_C_2];
	dutyCycle_mmc02[LW_C_3] = pS_mmc02[LW_C_3];
	dutyCycle_mmc02[LW_C_4] = pS_mmc02[LW_C_4];

	while (Vc_lw_ref > ESTADO_ACTUAL_C_LW) {
		if (IlwC > 0) {
			pos = posVmin (BYPASS, Vcond_LW_C_1,
			Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
			pS_mmc02[LW_C_1], pS_mmc02[LW_C_2],
			pS_mmc02[LW_C_3], pS_mmc02[LW_C_4]);
		} else {
			pos = posVmax (BYPASS, Vcond_LW_C_1,
			Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
			pS_mmc02[LW_C_1], pS_mmc02[LW_C_2],
			pS_mmc02[LW_C_3], pS_mmc02[LW_C_4]);
		}
		if (pos >= 0){
			pos_vector                     = LW_C_1 + pos;
			dutyCycle_mmc02[pos_vector]    = porcentaje_C_LW;
			pS_mmc02[pos_vector]           = 1;
			porcentaje_C_LW                = 1;
		}
		ESTADO_ACTUAL_C_LW = ESTADO_ACTUAL_C_LW + 1;
	}

	ESTADO_ACTUAL_C_LW = ESTADO_PREVIO;

	while (Vc_lw_ref < ESTADO_ACTUAL_C_LW) {
		if (IlwC > 0) {
			pos = posVmax (INSERTED, Vcond_LW_C_1,
			Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
			pS_mmc02[LW_C_1], pS_mmc02[LW_C_2],
			pS_mmc02[LW_C_3], pS_mmc02[LW_C_4]);
		} else {
			pos = posVmin (INSERTED, Vcond_LW_C_1,
			Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4,
			pS_mmc02[LW_C_1], pS_mmc02[LW_C_2],
			pS_mmc02[LW_C_3], pS_mmc02[LW_C_4]);
		}
```

```
     if (pos >= 0){
          pos_vector                    = LW_C_1 + pos;
          dutyCycle_mmc02[pos_vector]   = porcentaje_C_LW;
          pS_mmc02[pos_vector]          = 0;
          porcentaje_C_LW               = 0;
     }
     ESTADO_ACTUAL_C_LW = ESTADO_ACTUAL_C_LW - 1;
}

V_A_UP_PROMED = (dutyCycle_mmc02[UP_A_1] +
                 dutyCycle_mmc02[UP_A_2] +
                 dutyCycle_mmc02[UP_A_3] +
                 dutyCycle_mmc02[UP_A_4]) * V_A_UP_ave;

V_B_UP_PROMED = (dutyCycle_mmc02[UP_B_1] +
                 dutyCycle_mmc02[UP_B_2] +
                 dutyCycle_mmc02[UP_B_3] +
                 dutyCycle_mmc02[UP_B_4]) * V_B_UP_ave;

V_C_UP_PROMED = (dutyCycle_mmc02[UP_C_1] +
                 dutyCycle_mmc02[UP_C_2] +
                 dutyCycle_mmc02[UP_C_3] +
                 dutyCycle_mmc02[UP_C_4]) * V_C_UP_ave;


V_A_LW_PROMED = (dutyCycle_mmc02[LW_A_1] +
                 dutyCycle_mmc02[LW_A_2] +
                 dutyCycle_mmc02[LW_A_3] +
                 dutyCycle_mmc02[LW_A_4]) * V_A_LW_ave;

V_B_LW_PROMED = (dutyCycle_mmc02[LW_B_1] +
                 dutyCycle_mmc02[LW_B_2] +
                 dutyCycle_mmc02[LW_B_3] +
                 dutyCycle_mmc02[LW_B_4]) * V_B_LW_ave;

V_C_LW_PROMED = (dutyCycle_mmc02[LW_C_1] +
                 dutyCycle_mmc02[LW_C_2] +
                 dutyCycle_mmc02[LW_C_3] +
                 dutyCycle_mmc02[LW_C_4]) * V_C_LW_ave;


// ENVIAR TENSIONES PROMEDIO A LAS FUENTES DE TENSIÓN
EXTERNAS
writeVector(fuente_externa_UP, 0, V_A_UP_PROMED,
            V_B_UP_PROMED, V_C_UP_PROMED);

writeVector(fuente_externa_LW, 0, V_A_LW_PROMED,
            V_B_LW_PROMED, V_C_LW_PROMED);
#endif

Vuz = loopMicroseconds();
```