Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Tecnologías Industriales

ANEXOS: DESARROLLO DE UN MODELO DE SIMULACIÓN EN TIEMPO REAL PARA UN GENERADOR FOTOVOLTAICO BASADO EN UN CONVERTIDOR MODULAR MULTINIVEL

Autora: Herrero Herrero, Rocío

Tutor:

De Pablo Gómez, Santiago
Departamento de Tecnología Electrónica

Valladolid, abril 2021

ÍNDICE

1. CÓDIGO DE MATLAB

1.1. CÓDIGO COMPLETO DEL CONTROLADOR

```
function [control3u, control2u, control1u, rampasu,
sync, Ipv, rampasl, control1l, control2l, control3l] =
controller(Vcondensador_3u, iu3, Vcondensador_2u, iu2,
Vcondensador_1u, iu1, V, il1, Vcondensador_1l, il2,
Vcondensador_2l, il3, Vcondensador_3l)

persistent Ts myTime w desfase2 desfase3 pSync
tension_base constDiv pS_1l pS_1u pS_2u pS_2l pS_3u
pS_3l rampas1u rampas2u rampas3u rampas1l rampas2l
rampas3l;

if ( isempty(Ts) )
    % El controlador se ejecuta cada 250 us
    Ts           = 250e-6;
    myTime       = 0;
    w            = 2 * pi * 50;            % 50 Hz
    desfase2     = 2 * pi / 3;             % -120°
    desfase3     = 4 * pi / 3;             % -240°
    pSync        = 0;
    tension_base = 2 * 807.4;
    constDiv     = 1/807.4;
    pS_1u        = [0 1 0 1];
    pS_1l        = [0 1 0 1];
    pS_2u        = [0 1 0 1];
    pS_2l        = [0 1 0 1];
    pS_3u        = [0 1 0 1];
    pS_3l        = [0 1 0 1];
    rampas1u     = 1;
    rampas2u     = 1;
    rampas3u     = 1;
    rampas1l     = 0;
    rampas2l     = 0;
    rampas3l     = 0;
end

% Generación de la señal de sincronización
if ( pSync ~= 0 )
    pSync = 0;
```

```
else
    pSync = 1;
end
sync = pSync;

% Actualización del tiempo
myTime = myTime + Ts;

% Generación de senoides
sinF1 = sin(w * myTime);
sinF2 = sin(w * myTime - desfase2);
sinF3 = sin(w * myTime - desfase3);

% Generación de la señal moduladora FASE1
modulatoru_F1 = tension_base - V * sinF1;
modulatorl_F1 = tension_base + V * sinF1;

% Generación de la señal moduladora FASE2
modulatoru_F2 = tension_base - V * sinF2;
modulatorl_F2 = tension_base + V * sinF2;

% Generación de la señal moduladora FASE3
modulatoru_F3 = tension_base - V * sinF3;
modulatorl_F3 = tension_base + V * sinF3;


celdas_insertadas = 1;
celdas_bypass     = 0;




% ************** FASE 1U ****************

norm_1u          = modulatoru_F1 * constDiv;
nivel_1u         = floor(norm_1u);
porcentaje_1u    = norm_1u - nivel_1u;
estado_actual_1u = pS_1u(1) + pS_1u(2) + pS_1u(3) +
pS_1u(4);
control1u        = [pS_1u(1) pS_1u(2) pS_1u(3)
pS_1u(4)];
estado_previo    = estado_actual_1u;

while (norm_1u > estado_actual_1u)
```

```
    if (iu1 > 0)
          pos_1u = posVmin(celdas_bypass, pS_1u,
          Vcondensador_1u);
    else
          pos_1u = posVmax(celdas_bypass, pS_1u,
          Vcondensador_1u);
    end
    if (pos_1u == 0 )
        break;
    end
    control1u(pos_1u) = porcentaje_1u;
    pS_1u(pos_1u)     = 1;
    rampas1u = 0;
    estado_actual_1u = estado_actual_1u + 1;
    porcentaje_1u = 1;
end

estado_actual_1u = estado_previo;

while (norm_1u < estado_actual_1u)
    if (iu1 > 0)
          pos_1u = posVmax(celdas_insertadas, pS_1u,
          Vcondensador_1u);
    else
          pos_1u = posVmin(celdas_insertadas, pS_1u,
          Vcondensador_1u);
    end
    if (pos_1u == 0 )
        break;
    end
    control1u(pos_1u) = porcentaje_1u;
    pS_1u(pos_1u)     = 0;
    rampas1u          = 1;
    estado_actual_1u  = estado_actual_1u - 1;
    porcentaje_1u     = 0;
end



% ************** FASE 1L ****************

norm_1l          = modulatorl_F1 * constDiv;
nivel_1l         = floor(norm_1l);
```

```
porcentaje_1l     = norm_1l - nivel_1l;
estado_actual_1l = pS_1l(1) + pS_1l(2) + pS_1l(3) +
pS_1l(4);
control1l         = [pS_1l(1) pS_1l(2) pS_1l(3)
pS_1l(4)];
estado_previo     = estado_actual_1l;

while (norm_1l > estado_actual_1l)
    if (il1 > 0)
          pos_1l = posVmin(celdas_bypass, pS_1l,
          Vcondensador_1l);
    else
          pos_1l = posVmax(celdas_bypass, pS_1l,
          Vcondensador_1l);
    end
    if (pos_1l == 0 )
        break;
    end
    control1l(pos_1l) = porcentaje_1l;
    pS_1l(pos_1l)      = 1;
    rampas1l           = 0;
    estado_actual_1l   = estado_actual_1l + 1;
    porcentaje_1l      = 1;
end

estado_actual_1l = estado_previo;

while (norm_1l < estado_actual_1l)
    if (il1 > 0)
          pos_1l = posVmax(celdas_insertadas, pS_1l,
          Vcondensador_1l);
    else
          pos_1l = posVmin(celdas_insertadas, pS_1l,
          Vcondensador_1l);
    end
    if (pos_1l == 0 )
        break;
    end
    control1l(pos_1l) = porcentaje_1l;
    pS_1l(pos_1l)      = 0;
    rampas1l           = 1;
    estado_actual_1l   = estado_actual_1l - 1;
    porcentaje_1l      = 0;
```

```
end



% ************** FASE 2U ****************

norm_2u          = modulatoru_F2 * constDiv;
nivel_2u         = floor(norm_2u);
porcentaje_2u    = norm_2u - nivel_2u;
estado_actual_2u = pS_2u(1) + pS_2u(2) + pS_2u(3) + 
pS_2u(4);
control2u        = [pS_2u(1) pS_2u(2) pS_2u(3) 
pS_2u(4)];
estado_previo    = estado_actual_2u;

while (norm_2u > estado_actual_2u)
    if (iu2 > 0)
          pos_2u = posVmin(celdas_bypass, pS_2u,
          Vcondensador_2u);
    else
          pos_2u = posVmax(celdas_bypass, pS_2u,
          Vcondensador_2u);
    end
    if (pos_2u == 0 )
        break;
    end
    control2u(pos_2u)  = porcentaje_2u;
    pS_2u(pos_2u)      = 1;
    rampas2u           = 0;
    estado_actual_2u   = estado_actual_2u + 1;
    porcentaje_2u      = 1;
end

estado_actual_2u = estado_previo;

while (norm_2u < estado_actual_2u)
    if (iu2 > 0)
          pos_2u = posVmax(celdas_insertadas, pS_2u,
          Vcondensador_2u);
    else
          pos_2u = posVmin(celdas_insertadas, pS_2u,
          Vcondensador_2u);
    end
    if (pos_2u == 0 )
        break;
    end
    control2u(pos_2u) = porcentaje_2u;
    pS_2u(pos_2u)     = 0;
    rampas2u          = 1;
    estado_actual_2u  = estado_actual_2u - 1;
    porcentaje_2u     = 0;
end



% ************** FASE 2L ****************

norm_2l          = modulatorl_F2 * constDiv;
nivel_2l         = floor(norm_2l);
porcentaje_2l    = norm_2l - nivel_2l;
estado_actual_2l = pS_2l(1) + pS_2l(2) + pS_2l(3) +
pS_2l(4);
control2l        = [pS_2l(1) pS_2l(2) pS_2l(3)
pS_2l(4)];
estado_previo    = estado_actual_2l;

while (norm_2l > estado_actual_2l)
    if (il2 > 0)
          pos_2l = posVmin(celdas_bypass, pS_2l,
          Vcondensador_2l);
    else
          pos_2l = posVmax(celdas_bypass, pS_2l,
          Vcondensador_2l);
    end
    if (pos_2l == 0 )
        break;
    end
    control2l(pos_2l) = porcentaje_2l;
    pS_2l(pos_2l)     = 1;
    rampas2l          = 0;
    estado_actual_2l  = estado_actual_2l + 1;
    porcentaje_2l     = 1;
end

estado_actual_2l = estado_previo;
```

```
while (norm_2l < estado_actual_2l)
    if (il2 > 0)
          pos_2l = posVmax(celdas_insertadas, pS_2l,
          Vcondensador_2l);
    else
          pos_2l = posVmin(celdas_insertadas, pS_2l,
          Vcondensador_2l);
    end
    if (pos_2l == 0 )
        break;
    end
    control2l(pos_2l) = porcentaje_2l;
    pS_2l(pos_2l)     = 0;
    rampas2l          = 1;
    estado_actual_2l  = estado_actual_2l - 1;
    porcentaje_2l     = 0;
end



% ************** FASE 3U ****************

norm_3u          = modulatoru_F3 * constDiv;
nivel_3u         = floor(norm_3u);
porcentaje_3u    = norm_3u - nivel_3u;
estado_actual_3u = pS_3u(1) + pS_3u(2) + pS_3u(3) +
pS_3u(4);
control3u        = [pS_3u(1) pS_3u(2) pS_3u(3)
pS_3u(4)];
estado_previo    = estado_actual_3u;

while (norm_3u > estado_actual_3u)
    if (iu3 > 0)
          pos_3u = posVmin(celdas_bypass, pS_3u,
          Vcondensador_3u);
    else
          pos_3u = posVmax(celdas_bypass, pS_3u,
          Vcondensador_3u);
    end
    if (pos_3u == 0 )
        break;
    end
    control3u(pos_3u) = porcentaje_3u;
    pS_3u(pos_3u)     = 1;
    rampas3u          = 0;
    estado_actual_3u  = estado_actual_3u + 1;
    porcentaje_3u     = 1;
end

estado_actual_3u = estado_previo;

while  (norm_3u < estado_actual_3u)
    if  (iu3 > 0)
          pos_3u = posVmax(celdas_insertadas, pS_3u,
          Vcondensador_3u);
    else
          pos_3u = posVmin(celdas_insertadas, pS_3u,
          Vcondensador_3u);
    end
    if  (pos_3u == 0 )
        break;
    end
    control3u(pos_3u) = porcentaje_3u;
    pS_3u(pos_3u)     = 0;
    rampas3u          = 1;
    estado_actual_3u  = estado_actual_3u - 1;
    porcentaje_3u     = 0;
end



% ************** FASE 3L ****************

norm_3l          = modulatorl_F3 * constDiv;
nivel_3l         = floor(norm_3l);
porcentaje_3l    = norm_3l - nivel_3l;
estado_actual_3l = pS_3l(1) + pS_3l(2) + pS_3l(3) +
pS_3l(4);
control3l        = [pS_3l(1) pS_3l(2) pS_3l(3)
pS_3l(4)];
estado_previo    = estado_actual_3l;

while  (norm_3l > estado_actual_3l)
    if  (il3 > 0)
          pos_3l = posVmin(celdas_bypass, pS_3l,
          Vcondensador_3l);
    else
          pos_3l = posVmax(celdas_bypass, pS_3l,
          Vcondensador_3l);
    end
    if (pos_3l == 0 )
        break;
    end
    control3l(pos_3l) = porcentaje_3l;
    pS_3l(pos_3l)     = 1;
    rampas3l          = 0;
    estado_actual_3l  = estado_actual_3l + 1;
    porcentaje_3l     = 1;
end

estado_actual_3l = estado_previo;

while (norm_3l < estado_actual_3l)
    if (il3 > 0)
          pos_3l = posVmax(celdas_insertadas, pS_3l,
          Vcondensador_3l);
    else
          pos_3l = posVmin(celdas_insertadas, pS_3l,
          Vcondensador_3l);
    end
    if (pos_3l == 0 )
        break;
    end
    control3l(pos_3l) = porcentaje_3l;
    pS_3l(pos_3l)     = 0;
    rampas3l          = 1;
    estado_actual_3l  = estado_actual_3l - 1;
    porcentaje_3l     = 0;
end



% *********** PANELES FOTOVOLTAICOS *************

Ipv_1up1 = PanelPV(Vcondensador_1u(1));
Ipv_1up2 = PanelPV(Vcondensador_1u(2));
Ipv_1up3 = PanelPV(Vcondensador_1u(3));
Ipv_1up4 = PanelPV(Vcondensador_1u(4));
```

```
Ipv_1lw1 = PanelPV(Vcondensador_1l(1));
Ipv_1lw2 = PanelPV(Vcondensador_1l(2));
Ipv_1lw3 = PanelPV(Vcondensador_1l(3));
Ipv_1lw4 = PanelPV(Vcondensador_1l(4));

Ipv_2up1 = PanelPV(Vcondensador_2u(1));
Ipv_2up2 = PanelPV(Vcondensador_2u(2));
Ipv_2up3 = PanelPV(Vcondensador_2u(3));
Ipv_2up4 = PanelPV(Vcondensador_2u(4));

Ipv_2lw1 = PanelPV(Vcondensador_2l(1));
Ipv_2lw2 = PanelPV(Vcondensador_2l(2));
Ipv_2lw3 = PanelPV(Vcondensador_2l(3));
Ipv_2lw4 = PanelPV(Vcondensador_2l(4));

Ipv_3up1 = PanelPV(Vcondensador_3u(1));
Ipv_3up2 = PanelPV(Vcondensador_3u(2));
Ipv_3up3 = PanelPV(Vcondensador_3u(3));
Ipv_3up4 = PanelPV(Vcondensador_3u(4));

Ipv_3lw1 = PanelPV(Vcondensador_3l(1));
Ipv_3lw2 = PanelPV(Vcondensador_3l(2));
Ipv_3lw3 = PanelPV(Vcondensador_3l(3));
Ipv_3lw4 = PanelPV(Vcondensador_3l(4));

Ipv = [Ipv_1up1 Ipv_1up2 Ipv_1up3 Ipv_1up4 Ipv_1lw1
Ipv_1lw2 Ipv_1lw3 Ipv_1lw4 ...
    Ipv_2up1 Ipv_2up2 Ipv_2up3 Ipv_2up4 Ipv_2lw1
Ipv_2lw2 Ipv_2lw3 Ipv_2lw4 ...
    Ipv_3up1 Ipv_3up2 Ipv_3up3 Ipv_3up4 Ipv_3lw1
Ipv_3lw2 Ipv_3lw3 Ipv_3lw4];



% ************** SEÑALES DE RAMPAS ****************

rampasu = [rampas1u rampas2u rampas3u];
rampasl = [rampas1l rampas2l rampas3l];

end %function
```

```
% FUNCIONES PARA CALCULAR LAS POSICIONES DE DE TENSIÓN
MIN Y MAX

% Vmin: tensión minima de las celdas buscadas = el más
descargado de los buscados
function pos = posVmin(estado_buscado, pS, Vcond)
    Vmin = +10000;
    pos   = 0;
    for i=1:4
        if (pS(i) == estado_buscado)
            if (Vcond(i) < Vmin)
                Vmin = Vcond(i);
                pos   = i;
            end
        end
    end
end % function

% Vmax: tensión máxima de las celdas buscadas = el más
cargado de los buscados
function pos = posVmax(estado_buscado, pS, Vcond)
    Vmax = -10000;
    pos   = 0;
    for i=1:4
        if (pS(i) == estado_buscado)
            if (Vcond(i) > Vmax)
                Vmax = Vcond(i);
                pos   = i;
            end
        end
    end
end %function



% FUNCIÓN PARA CALCULAR LA CORRIENTE DE LOS PANELES
FOTOVOLTAICOS:

function Ipv = Fcorriente(Vpv)
    % Constantes ecuación de 2° orden:
    cte_a = -0.001322673;
    cte_b = 10.0523151;
    %cte_c = 0;
    % Constantes ecuación de 6° orden:
    cte_d = 0.00000000000357988;
    cte_e = -0.0000000142497;
    cte_f = 0.0000222097;
    cte_g = -0.017016857;
    cte_h = 6.426997334;
    cte_i = -949.7475378;
    %cte_j = 0;

    if (Vpv < 700)
          Ipv = (cte_a * Vpv) + cte_b;
    else
          Ipv = ((((cte_d * Vpv + cte_e) * Vpv + cte_f)
          * Vpv + cte_g) * Vpv + cte_h) * Vpv + cte_i;
    end

end %function
```

1.2. CÓDIGO COMPLETO DE LOS GENERADORES DE PWM DE UNA RAMA DE UNA FASE

A continuación, se muestra el código desarrollado para los generadores de PWM de la rama superior y fase 1.

El código implementado en las demás ramas y fases es idéntico, a excepción de dos apuntes.

El primero es la inicialización de las variables ‘contador’ y ‘paso’ en el isempty(). Dado que la señal de rampas que se envía del controlador a los generadores de PWM se ha inicializado en 1 para los valores de la rama superior y en 0 para la rama inferior, las variables ‘contador’ y ‘paso’ se inicializan a -0.5 y +1 en los generadores PWM de las ramas superiores respectivamente y a 250.5 y -1 en los generadores de PWM de las ramas inferiores respectivamente por mantener cierta coherencia.

Y el segundo es la posición del vector ‘rampas’ que se selecciona. En función de la fase en la que se esté generando el código, es decir, fase 1, 2 o 3, se seleccionara la posición con el mismo número en el vector ‘rampas’, para ello se ha creado la variable persistente ‘fase’, la única que varía con respecto a otras fases.

1.2.1. FASE 1, RAMA SUPERIOR, CELDA A

```
function salidaPWM = pwm_gen(sync, control, rampas)

     persistent oldSync contador paso periodo fase
     celda;

     if ( isempty(oldSync) )
         oldSync  = 0;
         contador = -0.5;
         paso     = +1;
         periodo  = 250;
         fase     = 1;
         celda    = 1;
     end

     porcentajePWM = control(celda);
     moduladora    = porcentajePWM * periodo;
     rampaControl  = rampas(fase);
```

```
    contador      = contador + paso;

    if ( sync ~= oldSync )
        if ( rampaControl == 1 )  % Rampa ascendente
            contador = 0.5;
            paso     = 1;
        else                      % Rampa descendente
            contador = 249.5;
            paso     = -1;
        end
    end

    oldSync = sync;

    if ( moduladora > contador )
        salidaPWM = 1;
    else
        salidaPWM = 0;
    end

end %function
```

1.2.2. FASE 1, RAMA SUPERIOR, CELDA B

```
function salidaPWM = pwm_gen(sync, control, rampas)

    persistent oldSync contador paso periodo fase
    celda;

    if ( isempty(oldSync) )
        oldSync  = 0;
        contador = -0.5;
        paso     = +1;
        periodo  = 250;
        fase     = 1;
        celda    = 2;
    end

    porcentajePWM = control(celda);
    moduladora    = porcentajePWM * periodo;
    rampaControl  = rampas(fase);
    contador      = contador + paso;

    if ( sync ~= oldSync )
        if ( rampaControl == 1 )   % Rampa ascendente
            contador = 0.5;
            paso     = 1;
        else                       % Rampa descendente
            contador = 249.5;
            paso     = -1;
        end
    end

    oldSync = sync;

    if ( moduladora > contador )
        salidaPWM = 1;
    else
        salidaPWM = 0;
    end

end %function
```

1.2.3. FASE 1, RAMA SUPERIOR, CELDA C

```
function salidaPWM = pwm_gen(sync, control, rampas)

    persistent oldSync contador paso periodo fase
    celda;

    if ( isempty(oldSync) )
        oldSync   = 0;
        contador  = -0.5;
        paso      = +1;
        periodo   = 250;
        fase      = 1;
        celda     = 3;
    end

    porcentajePWM = control(celda);
    moduladora     = porcentajePWM * periodo;
    rampaControl   = rampas(fase);
    contador       = contador + paso;

    if ( sync ~= oldSync )
        if ( rampaControl == 1 )  % Rampa ascendente
            contador = 0.5;
            paso     = 1;
        else                      % Rampa descendente
            contador = 249.5;
            paso     = -1;
        end
    end

    oldSync = sync;

    if ( moduladora > contador )
        salidaPWM = 1;
    else
        salidaPWM = 0;
    end

end %function
```

1.2.4. FASE 1, RAMA SUPERIOR, CELDA D

```
function salidaPWM = pwm_gen(sync, control, rampas)

    persistent oldSync contador paso periodo fase
    celda;

    if ( isempty(oldSync) )
        oldSync  = 0;
        contador = -0.5;
        paso     = +1;
        periodo  = 250;
        fase     = 1;
        celda    = 4;
    end
    porcentajePWM = control(celda);
    moduladora    = porcentajePWM * periodo;
    rampaControl  = rampas(fase);
    contador      = contador + paso;

    if ( sync ~= oldSync )
        if ( rampaControl == 1 )   % Rampa ascendente
            contador = 0.5;
            paso     = 1;
        else                       % Rampa descendente
            contador = 249.5;
            paso     = -1;
        end
    end

    oldSync = sync;

    if ( moduladora > contador )
        salidaPWM = 1;
    else
        salidaPWM = 0;
    end

end %function
```

1.3. FUNCIÓN EMBEBIDA “IPV”

En este apartado se indica el código correspondiente a la rama superior y la fase 1. Para el resto de ramas y fases el código es idéntico a excepción de las posiciones seleccionadas del vector ‘Ipv’, las cuales se corresponden con las del vector declarado en la función embebida que implementa el controlador.

```
function [Ipv1, Ipv2, Ipv3, Ipv4] = corriente(Ipv)

Ipv1 = Ipv(1);
Ipv2 = Ipv(2);
Ipv3 = Ipv(3);
Ipv4 = Ipv(4);

end %function
```

1.4. FUNCIÓN EMBEBIDA “SALIDA_1U”

Se muestra la función de la rama superior de la fase 1. No obstante, las funciones correspondientes a las demás ramas y fases integran exactamente el mismo código.

```
function V_cond  = vcond(V1, V2, V3, V4)

    V_cond = [V1 V2 V3 V4];

end %function
```

2. CÓDIGO DE LA SIMULACIÓN EN TIEMPO REAL

2.1. PESTAÑA GLOBAL

```
// Global

#define TPWM 500

#pragma GETTIME OFF

#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 A_UP 0
#define B_UP 1
```

```
#define C_UP 2
#define A_LW 3
#define B_LW 4
#define C_LW 5

#define V_DC_HALF 1614.8

#define I_UP_A readAO(1)
#define I_LW_A readAO(2)
#define I_UP_B readAO(3)
#define I_LW_B readAO(4)
#define I_UP_C readAO(5)
#define I_LW_C readAO(6)

#define INSERTED 1
#define BYPASS   0

//
// 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];


double mmc01_phA_Iup[4];
double mmc01_phA_Ilw[4];

double mmc01_phB_Iup[4];
double mmc01_phB_Ilw[4];

double mmc01_phC_Iup[4];
double mmc01_phC_Ilw[4];


//
// Global variables
```

```
//

int    syncP, syncN;

double phase, incPhase;
double mySin, myCos;

double Valpha_ref, Vbeta_ref;
double Va_ref, Vb_ref, Vc_ref;

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

int   ESTADO_ACTUAL_A_UP, ESTADO_ACTUAL_B_UP,
      ESTADO_ACTUAL_C_UP, ESTADO_ACTUAL_A_LW,
      ESTADO_ACTUAL_B_LW, ESTADO_ACTUAL_C_LW;

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;

//
// Variables vectoriales (double anotherOne[2040])
//

Int    pS[25];
double Iph[6];
double dutyCycle [25];

//
```

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

int    nivel;
int    pos;
int    ESTADO_PREVIO;
double pos_vector;
double Vpv;

double dutyCycle_1, dutyCycle_2, dutyCycle_3,
       dutyCycle_4;

//
// 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 = 0;

     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 = 0;

	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;
}

//
// Función para calcular la corriente de los paneles
//

#define CONSTANTE_A  -0.001322673
#define CONSTANTE_B  10.0523151
#define CONSTANTE_D  0.00000000000357988
#define CONSTANTE_E  -0.0000000142497
#define CONSTANTE_F  0.0000222097
#define CONSTANTE_G  -0.017016857
#define CONSTANTE_H  6.426997334
#define CONSTANTE_I  -949.7475378

double panelPV(double Vpv)
{

     double Ipv;

     if (Vpv < 700) {
          Ipv = (CONSTANTE_A * Vpv) + CONSTANTE_B;
     }
     else {
          Ipv = ((((CONSTANTE_D * Vpv + CONSTANTE_E) *
          Vpv + CONSTANTE_F) * Vpv + CONSTANTE_G) * Vpv
          + CONSTANTE_H) * Vpv + CONSTANTE_I;
     }
     return Ipv;
}
```

2.2. PESTAÑA SETUP

```
// Setup

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

syncP = 1;
syncN = 0;

phase = 0.0;

dutyCycle_1 = 0;
dutyCycle_2 = 1;
dutyCycle_3 = 0;
dutyCycle_4 = 1;

dutyCycle[UP_A_1] = 0;
dutyCycle[UP_A_2] = 1;
dutyCycle[UP_A_3] = 0;
dutyCycle[UP_A_4] = 1;

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

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

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

dutyCycle[UP_C_1] = 0;
dutyCycle[UP_C_2] = 1;
dutyCycle[UP_C_3] = 0;
```

```
dutyCycle[UP_C_4] = 1;

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

ESTADO_ACTUAL_A_UP = 0;
ESTADO_ACTUAL_B_UP = 0;
ESTADO_ACTUAL_C_UP = 0;
ESTADO_ACTUAL_A_LW = 0;
ESTADO_ACTUAL_B_LW = 0;
ESTADO_ACTUAL_C_LW = 0;

ESTADO_PREVIO = 0;

pS[UP_A_1] = 0;
pS[UP_A_2] = 1;
pS[UP_A_3] = 0;
pS[UP_A_4] = 1;

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

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

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

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

pS[LW_C_1] = 0;
pS[LW_C_2] = 1;
```

```
pS[LW_C_3] = 0;
pS[LW_C_4] = 1;

porcentaje_A_UP = 0;
porcentaje_B_UP = 0;
porcentaje_C_UP = 0;
porcentaje_A_LW = 0;
porcentaje_B_LW = 0;
porcentaje_C_LW = 0;
```

2.3. PESTAÑA LOOP

```
// Loop

// Generar salidas de PWM sincronizadas

dutyCycle_1 = dutyCycle[UP_A_1];
dutyCycle_2 = dutyCycle[UP_A_2];
dutyCycle_3 = dutyCycle[UP_A_3];
dutyCycle_4 = dutyCycle[UP_A_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);


dutyCycle_1 = dutyCycle[LW_A_1];
dutyCycle_2 = dutyCycle[LW_A_2];
dutyCycle_3 = dutyCycle[LW_A_3];
dutyCycle_4 = dutyCycle[LW_A_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);


dutyCycle_1 = dutyCycle[UP_B_1];
dutyCycle_2 = dutyCycle[UP_B_2];
dutyCycle_3 = dutyCycle[UP_B_3];
dutyCycle_4 = dutyCycle[UP_B_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);


dutyCycle_1 = dutyCycle[LW_B_1];
dutyCycle_2 = dutyCycle[LW_B_2];
dutyCycle_3 = dutyCycle[LW_B_3];
dutyCycle_4 = dutyCycle[LW_B_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);


dutyCycle_1 = dutyCycle[UP_C_1];
dutyCycle_2 = dutyCycle[UP_C_2];
dutyCycle_3 = dutyCycle[UP_C_3];
dutyCycle_4 = dutyCycle[UP_C_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);


dutyCycle_1 = dutyCycle[LW_C_1];
dutyCycle_2 = dutyCycle[LW_C_2];
dutyCycle_3 = dutyCycle[LW_C_3];
dutyCycle_4 = dutyCycle[LW_C_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);


// Lectura tensiones de los condensadores

Vcond_UP_A_1 = mmc01_phA_Vup[0];
Vcond_UP_A_2 = mmc01_phA_Vup[1];
Vcond_UP_A_3 = mmc01_phA_Vup[2];
Vcond_UP_A_4 = mmc01_phA_Vup[3];

Vcond_LW_A_1 = mmc01_phA_Vlw[0];
Vcond_LW_A_2 = mmc01_phA_Vlw[1];
Vcond_LW_A_3 = mmc01_phA_Vlw[2];
Vcond_LW_A_4 = mmc01_phA_Vlw[3];

Vcond_UP_B_1 = mmc01_phB_Vup[0];
Vcond_UP_B_2 = mmc01_phB_Vup[1];
Vcond_UP_B_3 = mmc01_phB_Vup[2];
Vcond_UP_B_4 = mmc01_phB_Vup[3];

Vcond_LW_B_1 = mmc01_phB_Vlw[0];
Vcond_LW_B_2 = mmc01_phB_Vlw[1];
```

```
Vcond_LW_B_3 = mmc01_phB_Vlw[2];
Vcond_LW_B_4 = mmc01_phB_Vlw[3];

Vcond_UP_C_1 = mmc01_phC_Vup[0];
Vcond_UP_C_2 = mmc01_phC_Vup[1];
Vcond_UP_C_3 = mmc01_phC_Vup[2];
Vcond_UP_C_4 = mmc01_phC_Vup[3];

Vcond_LW_C_1 = mmc01_phC_Vlw[0];
Vcond_LW_C_2 = mmc01_phC_Vlw[1];
Vcond_LW_C_3 = mmc01_phC_Vlw[2];
Vcond_LW_C_4 = mmc01_phC_Vlw[3];


// Corriente Paneles Fotovoltaicos

mmc01_phA_Iup[0] = panelPV(Vcond_UP_A_1);
mmc01_phA_Iup[1] = panelPV(Vcond_UP_A_2);
mmc01_phA_Iup[2] = panelPV(Vcond_UP_A_3);
mmc01_phA_Iup[3] = panelPV(Vcond_UP_A_4);

mmc01_phA_Ilw[0] = panelPV(Vcond_LW_A_1);
mmc01_phA_Ilw[1] = panelPV(Vcond_LW_A_2);
mmc01_phA_Ilw[2] = panelPV(Vcond_LW_A_3);
mmc01_phA_Ilw[3] = panelPV(Vcond_LW_A_4);

mmc01_phB_Iup[0] = panelPV(Vcond_UP_B_1);
mmc01_phB_Iup[1] = panelPV(Vcond_UP_B_2);
mmc01_phB_Iup[2] = panelPV(Vcond_UP_B_3);
mmc01_phB_Iup[3] = panelPV(Vcond_UP_B_4);

mmc01_phB_Ilw[0] = panelPV(Vcond_LW_B_1);
mmc01_phB_Ilw[1] = panelPV(Vcond_LW_B_2);
mmc01_phB_Ilw[2] = panelPV(Vcond_LW_B_3);
mmc01_phB_Ilw[3] = panelPV(Vcond_LW_B_4);

mmc01_phC_Iup[0] = panelPV(Vcond_UP_C_1);
mmc01_phC_Iup[1] = panelPV(Vcond_UP_C_2);
mmc01_phC_Iup[2] = panelPV(Vcond_UP_C_3);
mmc01_phC_Iup[3] = panelPV(Vcond_UP_C_4);

mmc01_phC_Ilw[0] = panelPV(Vcond_LW_C_1);
mmc01_phC_Ilw[1] = panelPV(Vcond_LW_C_2);
```

```
mmc01_phC_Ilw[2] = panelPV(Vcond_LW_C_3);
mmc01_phC_Ilw[3] = panelPV(Vcond_LW_C_4);


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


// Actualización de la fase (de -pi a +pi)
// 2pi x 50Hz x Ts
incPhase = 314.1592654 * loopPeriod();
phase    = wrapToPI(phase + incPhase);

// Obtención de las componentes alpha y beta
Valpha_ref = 1200.0 * cos(phase);
Vbeta_ref  = 1200.0 * sin(phase);

// Obtención de la tensión fase (a-b-c) neutro
ab2uvw(Valpha_ref, Vbeta_ref, Va_ref, Vb_ref, Vc_ref);

// Obtención tensiones de referencia superior e
inferior para las celdas del MMC con 807.4 V
Va_up_ref = V_DC_HALF * 0.00125 - Va_ref * 0.00125;
Vb_up_ref = V_DC_HALF * 0.00125 - Vb_ref * 0.00125;
Vc_up_ref = V_DC_HALF * 0.00125 - Vc_ref * 0.00125;
Va_lw_ref = V_DC_HALF * 0.00125 + Va_ref * 0.00125;
Vb_lw_ref = V_DC_HALF * 0.00125 + Vb_ref * 0.00125;
Vc_lw_ref = V_DC_HALF * 0.00125 + Vc_ref * 0.00125;


// Obtención de los ciclos de trabajo de cada rama
nivel = floor(Va_up_ref);
porcentaje_A_UP = Va_up_ref - nivel;

nivel = floor(Vb_up_ref);
porcentaje_B_UP = Vb_up_ref - nivel;

nivel = floor(Vc_up_ref);
porcentaje_C_UP = Vc_up_ref - nivel;

nivel = floor(Va_lw_ref);
porcentaje_A_LW = Va_lw_ref - nivel;
```

```
nivel = floor(Vb_lw_ref);
porcentaje_B_LW = Vb_lw_ref - nivel;

nivel = floor(Vc_lw_ref);
porcentaje_C_LW = Vc_lw_ref - nivel;


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

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

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

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

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

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


// Elección del que cambia y asignación de porcentaje

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

pos = 0;

ESTADO_PREVIO = ESTADO_ACTUAL_A_UP;

dutyCycle[UP_A_1] = pS[UP_A_1];
dutyCycle[UP_A_2] = pS[UP_A_2];
dutyCycle[UP_A_3] = pS[UP_A_3];
dutyCycle[UP_A_4] = pS[UP_A_4];

while (Va_up_ref > ESTADO_ACTUAL_A_UP) {
if (I_UP_A > 0) {
```

```
          pos = posVmin (BYPASS, Vcond_UP_A_1, Vcond_UP_A_2,
          Vcond_UP_A_3, Vcond_UP_A_4, pS[UP_A_1],
          pS[UP_A_2], pS[UP_A_3], pS[UP_A_4]);
          } else {
               pos = posVmax (BYPASS, Vcond_UP_A_1,
               Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4,
               pS[UP_A_1], pS[UP_A_2], pS[UP_A_3],
               pS[UP_A_4]);
          }
          ESTADO_ACTUAL_A_UP     = ESTADO_ACTUAL_A_UP + 1;
          pos_vector             = UP_A_1 + pos;
          dutyCycle[pos_vector] = porcentaje_A_UP;
          pS[pos_vector]         = 1;
          porcentaje_A_UP        = 1;
}

ESTADO_ACTUAL_A_UP = ESTADO_PREVIO;

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



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

pos = 0;
```

```
ESTADO_PREVIO = ESTADO_ACTUAL_B_UP;

dutyCycle[UP_B_1] = pS[UP_B_1];
dutyCycle[UP_B_2] = pS[UP_B_2];
dutyCycle[UP_B_3] = pS[UP_B_3];
dutyCycle[UP_B_4] = pS[UP_B_4];

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

ESTADO_ACTUAL_B_UP = ESTADO_PREVIO;

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

```
        pS[pos_vector]          = 0;
        porcentaje_B_UP         = 0;
}



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

pos = 0;

ESTADO_PREVIO = ESTADO_ACTUAL_C_UP;

dutyCycle[UP_C_1] = pS[UP_C_1];
dutyCycle[UP_C_2] = pS[UP_C_2];
dutyCycle[UP_C_3] = pS[UP_C_3];
dutyCycle[UP_C_4] = pS[UP_C_4];

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

ESTADO_ACTUAL_C_UP = ESTADO_PREVIO;

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



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

pos = 0;

ESTADO_PREVIO = ESTADO_ACTUAL_A_LW;

dutyCycle[LW_A_1] = pS[LW_A_1];
dutyCycle[LW_A_2] = pS[LW_A_2];
dutyCycle[LW_A_3] = pS[LW_A_3];
dutyCycle[LW_A_4] = pS[LW_A_4];

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

ESTADO_ACTUAL_A_LW = ESTADO_PREVIO;

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



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

pos = 0;

ESTADO_PREVIO = ESTADO_ACTUAL_B_LW;

dutyCycle[LW_B_1] = pS[LW_B_1];
dutyCycle[LW_B_2] = pS[LW_B_2];
dutyCycle[LW_B_3] = pS[LW_B_3];
dutyCycle[LW_B_4] = pS[LW_B_4];

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

ESTADO_ACTUAL_B_LW = ESTADO_PREVIO;

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



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

pos = 0;

ESTADO_PREVIO = ESTADO_ACTUAL_C_LW;
```

```
dutyCycle[LW_C_1] = pS[LW_C_1];
dutyCycle[LW_C_2] = pS[LW_C_2];
dutyCycle[LW_C_3] = pS[LW_C_3];
dutyCycle[LW_C_4] = pS[LW_C_4];

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

ESTADO_ACTUAL_C_LW = ESTADO_PREVIO;

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

```
    porcentaje_C_LW       = 0;
}

Vuz = loopMicroseconds();
```

2.4. PESTAÑA ASM

```
init:
$k0 = 1600
$k1 = 0
$k2 = 1
$k3 = 2
$k4 = 3
$k5 = -1600
$k6 = 700
$k7 = -0.001322673
$k8 = 10.0523151
$k9 = 3.57988e-12
$k10 = -1.42497e-08
$k11 = 2.22097e-05
$k12 = -0.017016857
$k13 = 6.426997334
$k14 = -949.7475378
$k15 = 4
$k16 = 5
$k17 = 6
$k18 = 7
$k19 = 8
$k20 = 9
$k21 = 10
$k22 = 11
$k23 = 12
$k24 = 13
$k25 = 14
$k26 = 15
$k27 = 16
$k28 = 17
$k29 = 18
$k30 = 19
$k31 = 20
$k32 = 21
$k33 = 22
$k34 = 23
$k35 = 24
$k36 = 500
$k37 = 314.1592654
$k38 = 1200
```

```
$k39 = 1614.8
$k40 = 0.00125
__SHAREDK1__ = 0
$Vcond_4 = __SHAREDK1__
$Vcond_5 = __SHAREDK1__
$Vcond_6 = __SHAREDK1__
$Vcond_7 = __SHAREDK1__
$Vcond_8 = __SHAREDK1__
$Vcond_9 = __SHAREDK1__
$Vb_up_ref = __SHAREDK1__
$pos_vector = __SHAREDK1__
$Vpv = __SHAREDK1__
$nivel = __SHAREDK1__
$Vc_ref = __SHAREDK1__
$Vc_lw_ref = __SHAREDK1__
$pos = __SHAREDK1__
$Valpha_ref = __SHAREDK1__
$phase = __SHAREDK1__
$mySin = __SHAREDK1__
$Va_lw_ref = __SHAREDK1__
$syncN = __SHAREDK1__
$syncP = __SHAREDK1__
$ESTADO_PREVIO = __SHAREDK1__
$Vcond_10 = __SHAREDK1__
$Vc_up_ref = __SHAREDK1__
$Vcond_11 = __SHAREDK1__
$Vb_lw_ref = __SHAREDK1__
$Vcond_12 = __SHAREDK1__
$Vcond_13 = __SHAREDK1__
$dutyCycle_1 = __SHAREDK1__
$Vcond_14 = __SHAREDK1__
$dutyCycle_2 = __SHAREDK1__
$Vcond_15 = __SHAREDK1__
$dutyCycle_3 = __SHAREDK1__
$ESTADO_ACTUAL_0 = __SHAREDK1__
$Vcond_16 = __SHAREDK1__
$dutyCycle_4 = __SHAREDK1__
$ESTADO_ACTUAL_1 = __SHAREDK1__
$Vbeta_ref = __SHAREDK1__
$Vcond_17 = __SHAREDK1__
$ESTADO_ACTUAL_2 = __SHAREDK1__
$Vcond_18 = __SHAREDK1__
$myCos = __SHAREDK1__
```

```
$ESTADO_ACTUAL_3 = __SHAREDK1__
$Vcond_19 = __SHAREDK1__
$ESTADO_ACTUAL_4 = __SHAREDK1__
$ESTADO_ACTUAL_5 = __SHAREDK1__
$Va_up_ref = __SHAREDK1__
$Va_ref = __SHAREDK1__
$Vcond_20 = __SHAREDK1__
$incPhase = __SHAREDK1__
$Vcond_21 = __SHAREDK1__
$Vcond_22 = __SHAREDK1__
$porcentaje_0 = __SHAREDK1__
$Vcond_23 = __SHAREDK1__
$porcentaje_1 = __SHAREDK1__
$Vcond_24 = __SHAREDK1__
$porcentaje_2 = __SHAREDK1__
$porcentaje_3 = __SHAREDK1__
$porcentaje_4 = __SHAREDK1__
$Vcond_1 = __SHAREDK1__
$porcentaje_5 = __SHAREDK1__
$Vcond_2 = __SHAREDK1__
$Vcond_3 = __SHAREDK1__
$Vb_ref = __SHAREDK1__
noop * 2

setup:
// #234: Vuz = 0;
$Vuz = $k1
// #235: Vvz = 0;
$Vvz = $k1
// #236: Vwz = 0;
$Vwz = $k1
// #237: Vxz = 0;
$Vxz = $k1
// #239: syncP = 1;
$syncP = $k2
// #240: syncN = 0;
$syncN = $k1
// #242: phase = 0.0;
$phase = $k1
// #244: dutyCycle_1 = 0;
$dutyCycle_1 = $k1
// #245: dutyCycle_2 = 1;
$dutyCycle_2 = $k2
```

```
// #246: dutyCycle_3 = 0;
$dutyCycle_3 = $k1
// #247: dutyCycle_4 = 1;
$dutyCycle_4 = $k2
// #249: dutyCycle[UP_A_1] = 0;
rwp = $k2 + 79
mem(rwp++) = $k1
nop * 1
// #250: dutyCycle[UP_A_2] = 1;
rwp = $k3 + 79
mem(rwp++) = $k2
nop * 1
// #251: dutyCycle[UP_A_3] = 0;
rwp = $k4 + 79
mem(rwp++) = $k1
nop * 1
// #252: dutyCycle[UP_A_4] = 1;
rwp = $k15 + 79
mem(rwp++) = $k2
nop * 1
// #254: dutyCycle[LW_A_1] = 0;
rwp = $k16 + 79
mem(rwp++) = $k1
nop * 1
// #255: dutyCycle[LW_A_2] = 1;
rwp = $k17 + 79
mem(rwp++) = $k2
nop * 1
// #256: dutyCycle[LW_A_3] = 0;
rwp = $k18 + 79
mem(rwp++) = $k1
nop * 1
// #257: dutyCycle[LW_A_4] = 1;
rwp = $k19 + 79
mem(rwp++) = $k2
nop * 1
// #259: dutyCycle[UP_B_1] = 0;
rwp = $k20 + 79
mem(rwp++) = $k1
nop * 1
// #260: dutyCycle[UP_B_2] = 1;
rwp = $k21 + 79
mem(rwp++) = $k2
```

```
nop * 1
// #261: dutyCycle[UP_B_3] = 0;
rwp = $k22 + 79
mem(rwp++) = $k1
nop * 1
// #262: dutyCycle[UP_B_4] = 1;
rwp = $k23 + 79
mem(rwp++) = $k2
nop * 1
// #264: dutyCycle[LW_B_1] = 0;
rwp = $k24 + 79
mem(rwp++) = $k1
nop * 1
// #265: dutyCycle[LW_B_2] = 1;
rwp = $k25 + 79
mem(rwp++) = $k2
nop * 1
// #266: dutyCycle[LW_B_3] = 0;
rwp = $k26 + 79
mem(rwp++) = $k1
nop * 1
// #267: dutyCycle[LW_B_4] = 1;
rwp = $k27 + 79
mem(rwp++) = $k2
nop * 1
// #269: dutyCycle[UP_C_1] = 0;
rwp = $k28 + 79
mem(rwp++) = $k1
nop * 1
// #270: dutyCycle[UP_C_2] = 1;
rwp = $k29 + 79
mem(rwp++) = $k2
nop * 1
// #271: dutyCycle[UP_C_3] = 0;
rwp = $k30 + 79
mem(rwp++) = $k1
nop * 1
// #272: dutyCycle[UP_C_4] = 1;
rwp = $k31 + 79
mem(rwp++) = $k2
nop * 1
// #274: dutyCycle[LW_C_1] = 0;
rwp = $k32 + 79
```

```
mem(rwp++) = $k1
nop * 1
// #275: dutyCycle[LW_C_2] = 1;
rwp = $k33 + 79
mem(rwp++) = $k2
nop * 1
// #276: dutyCycle[LW_C_3] = 0;
rwp = $k34 + 79
mem(rwp++) = $k1
nop * 1
// #277: dutyCycle[LW_C_4] = 1;
rwp = $k35 + 79
mem(rwp++) = $k2
// #279: ESTADO_ACTUAL_A_UP = 0;
$ESTADO_ACTUAL_0 = $k1
// #280: ESTADO_ACTUAL_B_UP = 0;
$ESTADO_ACTUAL_1 = $k1
// #281: ESTADO_ACTUAL_C_UP = 0;
$ESTADO_ACTUAL_2 = $k1
// #282: ESTADO_ACTUAL_A_LW = 0;
$ESTADO_ACTUAL_3 = $k1
// #283: ESTADO_ACTUAL_B_LW = 0;
$ESTADO_ACTUAL_4 = $k1
// #284: ESTADO_ACTUAL_C_LW = 0;
$ESTADO_ACTUAL_5 = $k1
// #286: ESTADO_PREVIO = 0;
$ESTADO_PREVIO = $k1
// #288: pS[UP_A_1] = 0;
rwp = $k2 + 48
mem(rwp++) = $k1
nop * 1
// #289: pS[UP_A_2] = 1;
rwp = $k3 + 48
mem(rwp++) = $k2
nop * 1
// #290: pS[UP_A_3] = 0;
rwp = $k4 + 48
mem(rwp++) = $k1
nop * 1
// #291: pS[UP_A_4] = 1;
rwp = $k15 + 48
mem(rwp++) = $k2
nop * 1
```

```
// #293: pS[LW_A_1] = 0;
rwp = $k16 + 48
mem(rwp++) = $k1
nop * 1
// #294: pS[LW_A_2] = 1;
rwp = $k17 + 48
mem(rwp++) = $k2
nop * 1
// #295: pS[LW_A_3] = 0;
rwp = $k18 + 48
mem(rwp++) = $k1
nop * 1
// #296: pS[LW_A_4] = 1;
rwp = $k19 + 48
mem(rwp++) = $k2
nop * 1
// #298: pS[UP_B_1] = 0;
rwp = $k20 + 48
mem(rwp++) = $k1
nop * 1
// #299: pS[UP_B_2] = 1;
rwp = $k21 + 48
mem(rwp++) = $k2
nop * 1
// #300: pS[UP_B_3] = 0;
rwp = $k22 + 48
mem(rwp++) = $k1
nop * 1
// #301: pS[UP_B_4] = 1;
rwp = $k23 + 48
mem(rwp++) = $k2
nop * 1
// #303: pS[LW_B_1] = 0;
rwp = $k24 + 48
mem(rwp++) = $k1
nop * 1
// #304: pS[LW_B_2] = 1;
rwp = $k25 + 48
mem(rwp++) = $k2
nop * 1
// #305: pS[LW_B_3] = 0;
rwp = $k26 + 48
mem(rwp++) = $k1
```

```
nop * 1
// #306: pS[LW_B_4] = 1;
rwp = $k27 + 48
mem(rwp++) = $k2
nop * 1
// #308: pS[UP_C_1] = 0;
rwp = $k28 + 48
mem(rwp++) = $k1
nop * 1
// #309: pS[UP_C_2] = 1;
rwp = $k29 + 48
mem(rwp++) = $k2
nop * 1
// #310: pS[UP_C_3] = 0;
rwp = $k30 + 48
mem(rwp++) = $k1
nop * 1
// #311: pS[UP_C_4] = 1;
rwp = $k31 + 48
mem(rwp++) = $k2
nop * 1
// #313: pS[LW_C_1] = 0;
rwp = $k32 + 48
mem(rwp++) = $k1
nop * 1
// #314: pS[LW_C_2] = 1;
rwp = $k33 + 48
mem(rwp++) = $k2
nop * 1
// #315: pS[LW_C_3] = 0;
rwp = $k34 + 48
mem(rwp++) = $k1
nop * 1
// #316: pS[LW_C_4] = 1;
rwp = $k35 + 48
mem(rwp++) = $k2
// #318: porcentaje_A_UP = 0;
$porcentaje_0 = $k1
// #319: porcentaje_B_UP = 0;
$porcentaje_1 = $k1
// #320: porcentaje_C_UP = 0;
$porcentaje_2 = $k1
// #321: porcentaje_A_LW = 0;
```

```
$porcentaje_3 = $k1
// #322: porcentaje_B_LW = 0;
$porcentaje_4 = $k1
// #323: porcentaje_C_LW = 0;
$porcentaje_5 = $k1
nop * 1

loop:
waitT
// #333: dutyCycle_1 = dutyCycle[UP_A_1];
rwp = $k2 + 79
nop * 1
$dutyCycle_1 = mem(rwp)
// #334: dutyCycle_2 = dutyCycle[UP_A_2];
rwp = $k3 + 79
nop * 1
$dutyCycle_2 = mem(rwp)
// #335: dutyCycle_3 = dutyCycle[UP_A_3];
rwp = $k4 + 79
nop * 1
$dutyCycle_3 = mem(rwp)
// #336: dutyCycle_4 = dutyCycle[UP_A_4];
rwp = $k15 + 79
nop * 1
$dutyCycle_4 = mem(rwp)
// #337: pwmConfig(UP_A_1, TPWM, syncP, dutyCycle_1);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_1__ = $dutyCycle_1
// #338: pwmConfig(UP_A_2, TPWM, syncP, dutyCycle_2);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_2__ = $dutyCycle_2
// #339: pwmConfig(UP_A_3, TPWM, syncP, dutyCycle_3);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_3__ = $dutyCycle_3
// #340: pwmConfig(UP_A_4, TPWM, syncP, dutyCycle_4);
port(__PWM_CONFIG__ + $syncP) = $k36
__PWM_DUTY_4__ = $dutyCycle_4
// #343: dutyCycle_1 = dutyCycle[LW_A_1];
rwp = $k16 + 79
nop * 1
```

```
$dutyCycle_1 = mem(rwp)
// #344: dutyCycle_2 = dutyCycle[LW_A_2];
rwp = $k17 + 79
nop * 1
$dutyCycle_2 = mem(rwp)
// #345: dutyCycle_3 = dutyCycle[LW_A_3];
rwp = $k18 + 79
nop * 1
$dutyCycle_3 = mem(rwp)
// #346: dutyCycle_4 = dutyCycle[LW_A_4];
rwp = $k19 + 79
nop * 1
$dutyCycle_4 = mem(rwp)
// #347: pwmConfig(LW_A_1, TPWM, syncN, dutyCycle_1);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_5__ = $dutyCycle_1
// #348: pwmConfig(LW_A_2, TPWM, syncN, dutyCycle_2);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_6__ = $dutyCycle_2
// #349: pwmConfig(LW_A_3, TPWM, syncN, dutyCycle_3);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_7__ = $dutyCycle_3
// #350: pwmConfig(LW_A_4, TPWM, syncN, dutyCycle_4);
port(__PWM_CONFIG__ + $syncN) = $k36
__PWM_DUTY_8__ = $dutyCycle_4
// #353: dutyCycle_1 = dutyCycle[UP_B_1];
rwp = $k20 + 79
nop * 1
$dutyCycle_1 = mem(rwp)
// #354: dutyCycle_2 = dutyCycle[UP_B_2];
rwp = $k21 + 79
nop * 1
$dutyCycle_2 = mem(rwp)
// #355: dutyCycle_3 = dutyCycle[UP_B_3];
rwp = $k22 + 79
nop * 1
$dutyCycle_3 = mem(rwp)
// #356: dutyCycle_4 = dutyCycle[UP_B_4];
rwp = $k23 + 79
nop * 1
```

```
$dutyCycle_4 = mem(rwp)
// #357: pwmConfig(UP_B_1, TPWM, syncP, dutyCycle_1);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_9__ = $dutyCycle_1
// #358: pwmConfig(UP_B_2, TPWM, syncP, dutyCycle_2);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_10__ = $dutyCycle_2
// #359: pwmConfig(UP_B_3, TPWM, syncP, dutyCycle_3);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_11__ = $dutyCycle_3
// #360: pwmConfig(UP_B_4, TPWM, syncP, dutyCycle_4);
port(__PWM_CONFIG__ + $syncP) = $k36
__PWM_DUTY_12__ = $dutyCycle_4
// #363: dutyCycle_1 = dutyCycle[LW_B_1];
rwp = $k24 + 79
nop * 1
$dutyCycle_1 = mem(rwp)
// #364: dutyCycle_2 = dutyCycle[LW_B_2];
rwp = $k25 + 79
nop * 1
$dutyCycle_2 = mem(rwp)
// #365: dutyCycle_3 = dutyCycle[LW_B_3];
rwp = $k26 + 79
nop * 1
$dutyCycle_3 = mem(rwp)
// #366: dutyCycle_4 = dutyCycle[LW_B_4];
rwp = $k27 + 79
nop * 1
$dutyCycle_4 = mem(rwp)
// #367: pwmConfig(LW_B_1, TPWM, syncN, dutyCycle_1);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_13__ = $dutyCycle_1
// #368: pwmConfig(LW_B_2, TPWM, syncN, dutyCycle_2);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_14__ = $dutyCycle_2
// #369: pwmConfig(LW_B_3, TPWM, syncN, dutyCycle_3);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
```

```
__PWM_DUTY_15__ = $dutyCycle_3
// #370: pwmConfig(LW_B_4, TPWM, syncN, dutyCycle_4);
port(__PWM_CONFIG__ + $syncN) = $k36
__PWM_DUTY_16__ = $dutyCycle_4
// #373: dutyCycle_1 = dutyCycle[UP_C_1];
rwp = $k28 + 79
nop * 1
$dutyCycle_1 = mem(rwp)
// #374: dutyCycle_2 = dutyCycle[UP_C_2];
rwp = $k29 + 79
nop * 1
$dutyCycle_2 = mem(rwp)
// #375: dutyCycle_3 = dutyCycle[UP_C_3];
rwp = $k30 + 79
nop * 1
$dutyCycle_3 = mem(rwp)
// #376: dutyCycle_4 = dutyCycle[UP_C_4];
rwp = $k31 + 79
nop * 1
$dutyCycle_4 = mem(rwp)
// #377: pwmConfig(UP_C_1, TPWM, syncP, dutyCycle_1);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_17__ = $dutyCycle_1
// #378: pwmConfig(UP_C_2, TPWM, syncP, dutyCycle_2);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_18__ = $dutyCycle_2
// #379: pwmConfig(UP_C_3, TPWM, syncP, dutyCycle_3);
port(__PWM_CONFIG__ + $syncP) = $k36
nop * 1
__PWM_DUTY_19__ = $dutyCycle_3
// #380: pwmConfig(UP_C_4, TPWM, syncP, dutyCycle_4);
port(__PWM_CONFIG__ + $syncP) = $k36
__PWM_DUTY_20__ = $dutyCycle_4
// #383: dutyCycle_1 = dutyCycle[LW_C_1];
rwp = $k32 + 79
nop * 1
$dutyCycle_1 = mem(rwp)
// #384: dutyCycle_2 = dutyCycle[LW_C_2];
rwp = $k33 + 79
nop * 1
$dutyCycle_2 = mem(rwp)
```

```
// #385: dutyCycle_3 = dutyCycle[LW_C_3];
rwp = $k34 + 79
nop * 1
$dutyCycle_3 = mem(rwp)
// #386: dutyCycle_4 = dutyCycle[LW_C_4];
rwp = $k35 + 79
nop * 1
$dutyCycle_4 = mem(rwp)
// #387: pwmConfig(LW_C_1, TPWM, syncN, dutyCycle_1);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_21__ = $dutyCycle_1
// #388: pwmConfig(LW_C_2, TPWM, syncN, dutyCycle_2);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_22__ = $dutyCycle_2
// #389: pwmConfig(LW_C_3, TPWM, syncN, dutyCycle_3);
port(__PWM_CONFIG__ + $syncN) = $k36
nop * 1
__PWM_DUTY_23__ = $dutyCycle_3
// #390: pwmConfig(LW_C_4, TPWM, syncN, dutyCycle_4);
port(__PWM_CONFIG__ + $syncN) = $k36
__PWM_DUTY_24__ = $dutyCycle_4
// #395: Vcond_UP_A_1 = mmc01_phA_Vup[0];
rwp = $k1 + 0
nop * 1
$Vcond_1 = mem(rwp)
// #396: Vcond_UP_A_2 = mmc01_phA_Vup[1];
rwp = $k2 + 0
nop * 1
$Vcond_2 = mem(rwp)
// #397: Vcond_UP_A_3 = mmc01_phA_Vup[2];
rwp = $k3 + 0
nop * 1
$Vcond_3 = mem(rwp)
// #398: Vcond_UP_A_4 = mmc01_phA_Vup[3];
rwp = $k4 + 0
nop * 1
$Vcond_4 = mem(rwp)
// #400: Vcond_LW_A_1 = mmc01_phA_Vlw[0];
rwp = $k1 + 4
nop * 1
$Vcond_5 = mem(rwp)
```

```
// #401: Vcond_LW_A_2 = mmc01_phA_Vlw[1];
rwp = $k2 + 4
nop * 1
$Vcond_6 = mem(rwp)
// #402: Vcond_LW_A_3 = mmc01_phA_Vlw[2];
rwp = $k3 + 4
nop * 1
$Vcond_7 = mem(rwp)
// #403: Vcond_LW_A_4 = mmc01_phA_Vlw[3];
rwp = $k4 + 4
nop * 1
$Vcond_8 = mem(rwp)
// #405: Vcond_UP_B_1 = mmc01_phB_Vup[0];
rwp = $k1 + 8
nop * 1
$Vcond_9 = mem(rwp)
// #406: Vcond_UP_B_2 = mmc01_phB_Vup[1];
rwp = $k2 + 8
nop * 1
$Vcond_10 = mem(rwp)
// #407: Vcond_UP_B_3 = mmc01_phB_Vup[2];
rwp = $k3 + 8
nop * 1
$Vcond_11 = mem(rwp)
// #408: Vcond_UP_B_4 = mmc01_phB_Vup[3];
rwp = $k4 + 8
nop * 1
$Vcond_12 = mem(rwp)
// #410: Vcond_LW_B_1 = mmc01_phB_Vlw[0];
rwp = $k1 + 12
nop * 1
$Vcond_13 = mem(rwp)
// #411: Vcond_LW_B_2 = mmc01_phB_Vlw[1];
rwp = $k2 + 12
nop * 1
$Vcond_14 = mem(rwp)
// #412: Vcond_LW_B_3 = mmc01_phB_Vlw[2];
rwp = $k3 + 12
nop * 1
$Vcond_15 = mem(rwp)
// #413: Vcond_LW_B_4 = mmc01_phB_Vlw[3];
rwp = $k4 + 12
nop * 1
```

```
$Vcond_16 = mem(rwp)
// #415: Vcond_UP_C_1 = mmc01_phC_Vup[0];
rwp = $k1 + 16
nop * 1
$Vcond_17 = mem(rwp)
// #416: Vcond_UP_C_2 = mmc01_phC_Vup[1];
rwp = $k2 + 16
nop * 1
$Vcond_18 = mem(rwp)
// #417: Vcond_UP_C_3 = mmc01_phC_Vup[2];
rwp = $k3 + 16
nop * 1
$Vcond_19 = mem(rwp)
// #418: Vcond_UP_C_4 = mmc01_phC_Vup[3];
rwp = $k4 + 16
nop * 1
$Vcond_20 = mem(rwp)
// #420: Vcond_LW_C_1 = mmc01_phC_Vlw[0];
rwp = $k1 + 20
nop * 1
$Vcond_21 = mem(rwp)
// #421: Vcond_LW_C_2 = mmc01_phC_Vlw[1];
rwp = $k2 + 20
nop * 1
$Vcond_22 = mem(rwp)
// #422: Vcond_LW_C_3 = mmc01_phC_Vlw[2];
rwp = $k3 + 20
nop * 1
$Vcond_23 = mem(rwp)
// #423: Vcond_LW_C_4 = mmc01_phC_Vlw[3];
rwp = $k4 + 20
nop * 1
$Vcond_24 = mem(rwp)
// #428: mmc01_phA_Iup[0] = panelPV (Vcond_UP_A_1);
push $Vcond_1
noop * 1
call _panelPV_
rwp = $k1 + 24
mem(rwp++) = r0
// #429: mmc01_phA_Iup[1] = panelPV (Vcond_UP_A_2);
push $Vcond_2
noop * 1
call _panelPV_
```

```
rwp = $k2 + 24
mem(rwp++) = r0
// #430: mmc01_phA_Iup[2] = panelPV (Vcond_UP_A_3);
push $Vcond_3
noop * 1
call _panelPV_
rwp = $k3 + 24
mem(rwp++) = r0
// #431: mmc01_phA_Iup[3] = panelPV (Vcond_UP_A_4);
push $Vcond_4
noop * 1
call _panelPV_
rwp = $k4 + 24
mem(rwp++) = r0
// #433: mmc01_phA_Ilw[0] = panelPV (Vcond_LW_A_1);
push $Vcond_5
noop * 1
call _panelPV_
rwp = $k1 + 28
mem(rwp++) = r0
// #434: mmc01_phA_Ilw[1] = panelPV (Vcond_LW_A_2);
push $Vcond_6
noop * 1
call _panelPV_
rwp = $k2 + 28
mem(rwp++) = r0
// #435: mmc01_phA_Ilw[2] = panelPV (Vcond_LW_A_3);
push $Vcond_7
noop * 1
call _panelPV_
rwp = $k3 + 28
mem(rwp++) = r0
// #436: mmc01_phA_Ilw[3] = panelPV (Vcond_LW_A_4);
push $Vcond_8
noop * 1
call _panelPV_
rwp = $k4 + 28
mem(rwp++) = r0
// #438: mmc01_phB_Iup[0] = panelPV (Vcond_UP_B_1);
push $Vcond_9
noop * 1
call _panelPV_
rwp = $k1 + 32
```

```
mem(rwp++) = r0
// #439: mmc01_phB_Iup[1] = panelPV (Vcond_UP_B_2);
push $Vcond_10
noop * 1
call _panelPV_
rwp = $k2 + 32
mem(rwp++) = r0
// #440: mmc01_phB_Iup[2] = panelPV (Vcond_UP_B_3);
push $Vcond_11
noop * 1
call _panelPV_
rwp = $k3 + 32
mem(rwp++) = r0
// #441: mmc01_phB_Iup[3] = panelPV (Vcond_UP_B_4);
push $Vcond_12
noop * 1
call _panelPV_
rwp = $k4 + 32
mem(rwp++) = r0
// #443: mmc01_phB_Ilw[0] = panelPV (Vcond_LW_B_1);
push $Vcond_13
noop * 1
call _panelPV_
rwp = $k1 + 36
mem(rwp++) = r0
// #444: mmc01_phB_Ilw[1] = panelPV (Vcond_LW_B_2);
push $Vcond_14
noop * 1
call _panelPV_
rwp = $k2 + 36
mem(rwp++) = r0
// #445: mmc01_phB_Ilw[2] = panelPV (Vcond_LW_B_3);
push $Vcond_15
noop * 1
call _panelPV_
rwp = $k3 + 36
mem(rwp++) = r0
// #446: mmc01_phB_Ilw[3] = panelPV (Vcond_LW_B_4);
push $Vcond_16
noop * 1
call _panelPV_
rwp = $k4 + 36
mem(rwp++) = r0
```

```
// #448: mmc01_phC_Iup[0] = panelPV (Vcond_UP_C_1);
push $Vcond_17
noop * 1
call _panelPV_
rwp = $k1 + 40
mem(rwp++) = r0
// #449: mmc01_phC_Iup[1] = panelPV (Vcond_UP_C_2);
push $Vcond_18
noop * 1
call _panelPV_
rwp = $k2 + 40
mem(rwp++) = r0
// #450: mmc01_phC_Iup[2] = panelPV (Vcond_UP_C_3);
push $Vcond_19
noop * 1
call _panelPV_
rwp = $k3 + 40
mem(rwp++) = r0
// #451: mmc01_phC_Iup[3] = panelPV (Vcond_UP_C_4);
push $Vcond_20
noop * 1
call _panelPV_
rwp = $k4 + 40
mem(rwp++) = r0
// #453: mmc01_phC_Ilw[0] = panelPV (Vcond_LW_C_1);
push $Vcond_21
noop * 1
call _panelPV_
rwp = $k1 + 44
mem(rwp++) = r0
// #454: mmc01_phC_Ilw[1] = panelPV (Vcond_LW_C_2);
push $Vcond_22
noop * 1
call _panelPV_
rwp = $k2 + 44
mem(rwp++) = r0
// #455: mmc01_phC_Ilw[2] = panelPV (Vcond_LW_C_3);
push $Vcond_23
noop * 1
call _panelPV_
rwp = $k3 + 44
mem(rwp++) = r0
// #456: mmc01_phC_Ilw[3] = panelPV (Vcond_LW_C_4);
```

```
push $Vcond_24
noop * 1
call _panelPV_
rwp = $k4 + 44
mem(rwp++) = r0
// #460: syncP = syncN;
$syncP = $syncN
// #461: syncN = (syncN == 0);
r0 = $syncN - $k1
nop * 1
$syncN = EQ ? __ONE__ : __ZERO__
// #466: incPhase = 314.1592654 * loopPeriod();
__SHAREDK1__ = 0.00025
nop * 2
p0 = $k37 * __SHAREDK1__
nop * 11
$incPhase = p0
nop * 2
// #467: phase = wrapToPI(phase + incPhase);
r0 = $phase + $incPhase
nop * 2
p0 = r0 * __INV_PI__
noop * 11
r0 = wrap(p0)
noop * 2
p0 = r0 * __PI__
noop * 11
$phase = p0
nop * 2
// #470: Valpha_ref = 1200.0 * cos(phase);
r0 = $phase
nop * 1
call iridium_cos
p0 = $k38 * r0
nop * 11
$Valpha_ref = p0
// #471: Vbeta_ref = 1200.0 * sin(phase);
r0 = $phase
nop * 1
call iridium_sin
p0 = $k38 * r0
nop * 11
$Vbeta_ref = p0
```

```
nop * 1
// #474: ab2uvw(Valpha_ref, Vbeta_ref, Va_ref, Vb_ref, Vc_ref);
p0 = - $Valpha_ref * __HALF__
p0 = p0 + $Vbeta_ref * __SQRT3BY2__
p1 = - $Valpha_ref * __HALF__
p1 = p1 - $Vbeta_ref * __SQRT3BY2__
nop * 8
$Va_ref = $Valpha_ref
$Vb_ref = p0
nop * 1
$Vc_ref = p1
// #477: Va_up_ref = V_DC_HALF * 0.00125 - Va_ref * 0.00125;
p0 = $k39 * $k40
p0 = p0 - $Va_ref * $k40
nop * 11
$Va_up_ref = p0
// #478: Vb_up_ref = V_DC_HALF * 0.00125 - Vb_ref * 0.00125;
p0 = $k39 * $k40
p0 = p0 - $Vb_ref * $k40
nop * 11
$Vb_up_ref = p0
// #479: Vc_up_ref = V_DC_HALF * 0.00125 - Vc_ref * 0.00125;
p0 = $k39 * $k40
p0 = p0 - $Vc_ref * $k40
nop * 11
$Vc_up_ref = p0
// #480: Va_lw_ref = V_DC_HALF * 0.00125 + Va_ref * 0.00125;
p0 = $k39 * $k40
p0 = p0 + $Va_ref * $k40
nop * 11
$Va_lw_ref = p0
// #481: Vb_lw_ref = V_DC_HALF * 0.00125 + Vb_ref * 0.00125;
p0 = $k39 * $k40
p0 = p0 + $Vb_ref * $k40
nop * 11
$Vb_lw_ref = p0
// #482: Vc_lw_ref = V_DC_HALF * 0.00125 + Vc_ref * 0.00125;
p0 = $k39 * $k40
p0 = p0 + $Vc_ref * $k40
nop * 11
$Vc_lw_ref = p0
// #486: nivel = floor(Va_up_ref);
r0 = $Va_up_ref & __INTEGPART__
```

```
nop * 2
$nivel = r0 & __INTEGPART__
nop * 2
// #487: porcentaje_A_UP = Va_up_ref - nivel;
$porcentaje_0 = $Va_up_ref - $nivel
// #489: nivel = floor(Vb_up_ref);
r0 = $Vb_up_ref & __INTEGPART__
nop * 2
$nivel = r0 & __INTEGPART__
nop * 2
// #490: porcentaje_B_UP = Vb_up_ref - nivel;
$porcentaje_1 = $Vb_up_ref - $nivel
// #492: nivel = floor(Vc_up_ref);
r0 = $Vc_up_ref & __INTEGPART__
nop * 2
$nivel = r0 & __INTEGPART__
nop * 2
// #493: porcentaje_C_UP = Vc_up_ref - nivel;
$porcentaje_2 = $Vc_up_ref - $nivel
// #495: nivel = floor(Va_lw_ref);
r0 = $Va_lw_ref & __INTEGPART__
nop * 2
$nivel = r0 & __INTEGPART__
nop * 2
// #496: porcentaje_A_LW = Va_lw_ref - nivel;
$porcentaje_3 = $Va_lw_ref - $nivel
// #498: nivel = floor(Vb_lw_ref);
r0 = $Vb_lw_ref & __INTEGPART__
nop * 2
$nivel = r0 & __INTEGPART__
nop * 2
// #499: porcentaje_B_LW = Vb_lw_ref - nivel;
$porcentaje_4 = $Vb_lw_ref - $nivel
// #501: nivel = floor(Vc_lw_ref);
r0 = $Vc_lw_ref & __INTEGPART__
nop * 2
$nivel = r0 & __INTEGPART__
nop * 2
// #502: porcentaje_C_LW = Vc_lw_ref - nivel;
$porcentaje_5 = $Vc_lw_ref - $nivel
// #506: ESTADO_ACTUAL_A_UP = pS[UP_A_1] + pS[UP_A_2] + pS[UP_A_3] + pS[UP_A_4];
rwp = $k2 + 48
```

```
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k3 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k4 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k15 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
$ESTADO_ACTUAL_0 = r1 + r0
// #508: ESTADO_ACTUAL_B_UP = pS[UP_B_1] + pS[UP_B_2] + pS[UP_B_3]
+ pS[UP_B_4];
rwp = $k20 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k21 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
```

```
nop * 2
push r0
nop * 2
rwp = $k22 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k23 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
$ESTADO_ACTUAL_1 = r1 + r0
// #510: ESTADO_ACTUAL_C_UP = pS[UP_C_1] + pS[UP_C_2] + pS[UP_C_3] +
pS[UP_C_4];
rwp = $k28 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k29 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k30 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
```

```
nop * 2
rwp = $k31 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
$ESTADO_ACTUAL_2 = r1 + r0
// #512: ESTADO_ACTUAL_A_LW = pS[LW_A_1] + pS[LW_A_2] + pS[LW_A_3]
+ pS[LW_A_4];
rwp = $k16 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k17 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k18 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k19 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
$ESTADO_ACTUAL_3 = r1 + r0
// #514: ESTADO_ACTUAL_B_LW = pS[LW_B_1] + pS[LW_B_2] + pS[LW_B_3]
+ pS[LW_B_4];
rwp = $k24 + 48
nop * 1
```

```
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k25 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k26 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k27 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
$ESTADO_ACTUAL_4 = r1 + r0
// #516: ESTADO_ACTUAL_C_LW = pS[LW_C_1] + pS[LW_C_2] + pS[LW_C_3]
+ pS[LW_C_4];
rwp = $k32 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k33 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
```

```
push r0
nop * 2
rwp = $k34 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
r0 = r1 + r0
nop * 2
push r0
nop * 2
rwp = $k35 + 48
nop * 1
r0 = mem(rwp)
pop r1
nop * 2
$ESTADO_ACTUAL_5 = r1 + r0
// #523: pos = 0;
$pos = $k1
// #525: ESTADO_PREVIO = ESTADO_ACTUAL_A_UP;
$ESTADO_PREVIO = $ESTADO_ACTUAL_0
// #527: dutyCycle[UP_A_1] = pS[UP_A_1];
rwp = $k2 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k2 + 79
mem(rwp++) = r0
nop * 1
// #528: dutyCycle[UP_A_2] = pS[UP_A_2];
rwp = $k3 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k3 + 79
mem(rwp++) = r0
nop * 1
// #529: dutyCycle[UP_A_3] = pS[UP_A_3];
rwp = $k4 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k4 + 79
```

```
mem(rwp++) = r0
nop * 1
// #530: dutyCycle[UP_A_4] = pS[UP_A_4];
rwp = $k15 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k15 + 79
mem(rwp++) = r0
nop * 2
// #532: while (Va_up_ref > ESTADO_ACTUAL_A_UP) {
_cpp_loop_10_:
r0 = $Va_up_ref - $ESTADO_ACTUAL_0
r0 =__ZERO__
nop * 1
jpLE _cpp_continue_10_
// #533: if (I_UP_A > 0) {
r1 = __ANALOG_OUTPUT_1__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_11a_
// #534: pos = posVmin (BYPASS, Vcond_UP_A_1, Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4, pS[UP_A_1], pS[UP_A_2], pS[UP_A_3], pS[UP_A_4]);
push $k1
nop * 2
push $Vcond_1
nop * 2
push $Vcond_2
nop * 2
push $Vcond_3
nop * 2
push $Vcond_4
nop * 2
rwp = $k2 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k3 + 48
```

```
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k4 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k15 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 1
// #535: else {
goto _cpp_if_11b_
_cpp_if_11a_:
// #536: pos = posVmax (BYPASS, Vcond_UP_A_1, Vcond_UP_A_2,
Vcond_UP_A_3, Vcond_UP_A_4, pS[UP_A_1], pS[UP_A_2], pS[UP_A_3],
pS[UP_A_4]);
push $k1
nop * 2
push $Vcond_1
nop * 2
push $Vcond_2
nop * 2
push $Vcond_3
nop * 2
push $Vcond_4
nop * 2
rwp = $k2 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k3 + 48
```

```
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k4 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k15 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 2
_cpp_if_11b_:
// #538: ESTADO_ACTUAL_A_UP = ESTADO_ACTUAL_A_UP + 1;
$ESTADO_ACTUAL_0 = $ESTADO_ACTUAL_0 + $k2
// #539: pos_vector = UP_A_1 + pos;
$pos_vector = $k2 + $pos
nop * 2
// #540: dutyCycle[pos_vector] = porcentaje_A_UP;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_0
nop * 1
// #541: pS[pos_vector] = 1;
rwp = $pos_vector + 48
mem(rwp++) = $k2
// #542: porcentaje_A_UP = 1;
$porcentaje_0 = $k2
nop * 1
goto _cpp_loop_10_
_cpp_continue_10_:
// #545: ESTADO_ACTUAL_A_UP = ESTADO_PREVIO;
$ESTADO_ACTUAL_0 = $ESTADO_PREVIO
nop * 2
// #547: while (Va_up_ref < ESTADO_ACTUAL_A_UP) {
_cpp_loop_12_:
```

```
r0 = $Va_up_ref - $ESTADO_ACTUAL_0
r0 =__ZERO__
nop * 1
jpGE _cpp_continue_12_
// #548: if (I_UP_A > 0) {
r1 = __ANALOG_OUTPUT_1__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_13a_
// #549: pos = posVmax (INSERTED, Vcond_UP_A_1, Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4, pS[UP_A_1], pS[UP_A_2], pS[UP_A_3], pS[UP_A_4]);
push $k2
nop * 2
push $Vcond_1
nop * 2
push $Vcond_2
nop * 2
push $Vcond_3
nop * 2
push $Vcond_4
nop * 2
rwp = $k2 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k3 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k4 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k15 + 48
```

```
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 1
// #550: else {
goto _cpp_if_13b_
_cpp_if_13a_:
// #551: pos = posVmin (INSERTED, Vcond_UP_A_1, Vcond_UP_A_2, Vcond_UP_A_3, Vcond_UP_A_4, pS[UP_A_1], pS[UP_A_2], pS[UP_A_3], pS[UP_A_4]);
push $k2
nop * 2
push $Vcond_1
nop * 2
push $Vcond_2
nop * 2
push $Vcond_3
nop * 2
push $Vcond_4
nop * 2
rwp = $k2 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k3 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k4 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k15 + 48
```

```
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 2
_cpp_if_13b_:
// #553: ESTADO_ACTUAL_A_UP = ESTADO_ACTUAL_A_UP - 1;
$ESTADO_ACTUAL_0 = $ESTADO_ACTUAL_0 - $k2
// #554: pos_vector = UP_A_1 + pos;
$pos_vector = $k2 + $pos
nop * 2
// #555: dutyCycle[pos_vector] = porcentaje_A_UP;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_0
nop * 1
// #556: pS[pos_vector] = 0;
rwp = $pos_vector + 48
mem(rwp++) = $k1
// #557: porcentaje_A_UP = 0;
$porcentaje_0 = $k1
nop * 1
goto _cpp_loop_12_
_cpp_continue_12_:
// #564: pos = 0;
$pos = $k1
// #566: ESTADO_PREVIO = ESTADO_ACTUAL_B_UP;
$ESTADO_PREVIO = $ESTADO_ACTUAL_1
// #568: dutyCycle[UP_B_1] = pS[UP_B_1];
rwp = $k20 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k20 + 79
mem(rwp++) = r0
nop * 1
// #569: dutyCycle[UP_B_2] = pS[UP_B_2];
rwp = $k21 + 48
nop * 1
r0 = mem(rwp)
nop * 1
```

```
rwp = $k21 + 79
mem(rwp++) = r0
nop * 1
// #570: dutyCycle[UP_B_3] = pS[UP_B_3];
rwp = $k22 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k22 + 79
mem(rwp++) = r0
nop * 1
// #571: dutyCycle[UP_B_4] = pS[UP_B_4];
rwp = $k23 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k23 + 79
mem(rwp++) = r0
nop * 2
// #573: while (Vb_up_ref > ESTADO_ACTUAL_B_UP) {
_cpp_loop_14_:
r0 = $Vb_up_ref - $ESTADO_ACTUAL_1
r0 =__ZERO__
nop * 1
jpLE _cpp_continue_14_
// #574: if (I_UP_B > 0) {
r1 = __ANALOG_OUTPUT_3__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_15a_
// #575: pos = posVmin (BYPASS, Vcond_UP_B_1, Vcond_UP_B_2,
Vcond_UP_B_3, Vcond_UP_B_4, pS[UP_B_1], pS[UP_B_2], pS[UP_B_3],
pS[UP_B_4]);
push $k1
nop * 2
push $Vcond_9
nop * 2
push $Vcond_10
nop * 2
push $Vcond_11
nop * 2
```

```
push $Vcond_12
nop * 2
rwp = $k20 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k21 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k22 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k23 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 1
// #576: else {
goto _cpp_if_15b_
_cpp_if_15a_:
// #577: pos = posVmax (BYPASS, Vcond_UP_B_1, Vcond_UP_B_2,
Vcond_UP_B_3, Vcond_UP_B_4, pS[UP_B_1], pS[UP_B_2], pS[UP_B_3],
pS[UP_B_4]);
push $k1
nop * 2
push $Vcond_9
nop * 2
push $Vcond_10
nop * 2
push $Vcond_11
nop * 2
```

```
push $Vcond_12
nop * 2
rwp = $k20 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k21 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k22 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k23 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 2
_cpp_if_15b_:
// #579: ESTADO_ACTUAL_B_UP = ESTADO_ACTUAL_B_UP + 1;
$ESTADO_ACTUAL_1 = $ESTADO_ACTUAL_1 + $k2
// #580: pos_vector = UP_B_1 + pos;
$pos_vector = $k20 + $pos
nop * 2
// #581: dutyCycle[pos_vector] = porcentaje_B_UP;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_1
nop * 1
// #582: pS[pos_vector] = 1;
rwp = $pos_vector + 48
mem(rwp++) = $k2
// #583: porcentaje_B_UP = 1;
```

```
$porcentaje_1 = $k2
nop * 1
goto _cpp_loop_14_
_cpp_continue_14_:
// #586: ESTADO_ACTUAL_B_UP = ESTADO_PREVIO;
$ESTADO_ACTUAL_1 = $ESTADO_PREVIO
nop * 2
// #588: while (Vb_up_ref < ESTADO_ACTUAL_B_UP) {
_cpp_loop_16_:
r0 = $Vb_up_ref - $ESTADO_ACTUAL_1
r0 =__ZERO__
nop * 1
jpGE _cpp_continue_16_
// #589: if (I_UP_B > 0) {
r1 = __ANALOG_OUTPUT_3__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_17a_
// #590: pos = posVmax (INSERTED, Vcond_UP_B_1, Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4, pS[UP_B_1], pS[UP_B_2], pS[UP_B_3], pS[UP_B_4]);
push $k2
nop * 2
push $Vcond_9
nop * 2
push $Vcond_10
nop * 2
push $Vcond_11
nop * 2
push $Vcond_12
nop * 2
rwp = $k20 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k21 + 48
nop * 1
r0 = mem(rwp)
nop * 2
```

```
push r0
nop * 2
rwp = $k22 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k23 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 1
// #591: else {
goto _cpp_if_17b_
_cpp_if_17a_:
// #592: pos = posVmin (INSERTED, Vcond_UP_B_1, Vcond_UP_B_2, Vcond_UP_B_3, Vcond_UP_B_4, pS[UP_B_1], pS[UP_B_2], pS[UP_B_3], pS[UP_B_4]);
push $k2
nop * 2
push $Vcond_9
nop * 2
push $Vcond_10
nop * 2
push $Vcond_11
nop * 2
push $Vcond_12
nop * 2
rwp = $k20 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k21 + 48
nop * 1
r0 = mem(rwp)
nop * 2
```

```
push r0
nop * 2
rwp = $k22 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k23 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 2
_cpp_if_17b_:
// #594: ESTADO_ACTUAL_B_UP = ESTADO_ACTUAL_B_UP - 1;
$ESTADO_ACTUAL_1 = $ESTADO_ACTUAL_1 - $k2
// #595: pos_vector = UP_B_1 + pos;
$pos_vector = $k20 + $pos
nop * 2
// #596: dutyCycle[pos_vector] = porcentaje_B_UP;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_1
nop * 1
// #597: pS[pos_vector] = 0;
rwp = $pos_vector + 48
mem(rwp++) = $k1
// #598: porcentaje_B_UP = 0;
$porcentaje_1 = $k1
nop * 1
goto _cpp_loop_16_
_cpp_continue_16_:
// #605: pos = 0;
$pos = $k1
// #607: ESTADO_PREVIO = ESTADO_ACTUAL_C_UP;
$ESTADO_PREVIO = $ESTADO_ACTUAL_2
// #609: dutyCycle[UP_C_1] = pS[UP_C_1];
rwp = $k28 + 48
nop * 1
r0 = mem(rwp)
```

```
nop * 1
rwp = $k28 + 79
mem(rwp++) = r0
nop * 1
// #610: dutyCycle[UP_C_2] = pS[UP_C_2];
rwp = $k29 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k29 + 79
mem(rwp++) = r0
nop * 1
// #611: dutyCycle[UP_C_3] = pS[UP_C_3];
rwp = $k30 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k30 + 79
mem(rwp++) = r0
nop * 1
// #612: dutyCycle[UP_C_4] = pS[UP_C_4];
rwp = $k31 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k31 + 79
mem(rwp++) = r0
nop * 2
// #614: while (Vc_up_ref > ESTADO_ACTUAL_C_UP) {
_cpp_loop_18_:
r0 = $Vc_up_ref - $ESTADO_ACTUAL_2
r0 =__ZERO__
nop * 1
jpLE _cpp_continue_18_
// #615: if (I_UP_C > 0) {
r1 = __ANALOG_OUTPUT_5__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_19a_
```

```
// #616: pos = posVmin (BYPASS, Vcond_UP_C_1, Vcond_UP_C_2,
Vcond_UP_C_3, Vcond_UP_C_4, pS[UP_C_1], pS[UP_C_2], pS[UP_C_3],
pS[UP_C_4]);
push $k1
nop * 2
push $Vcond_17
nop * 2
push $Vcond_18
nop * 2
push $Vcond_19
nop * 2
push $Vcond_20
nop * 2
rwp = $k28 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k29 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k30 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k31 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 1
// #617: else {
goto _cpp_if_19b_
_cpp_if_19a_:
```

```
// #618: pos = posVmax (BYPASS, Vcond_UP_C_1, Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4, pS[UP_C_1], pS[UP_C_2], pS[UP_C_3], pS[UP_C_4]);
push $k1
nop * 2
push $Vcond_17
nop * 2
push $Vcond_18
nop * 2
push $Vcond_19
nop * 2
push $Vcond_20
nop * 2
rwp = $k28 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k29 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k30 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k31 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 2
_cpp_if_19b_:
// #620: ESTADO_ACTUAL_C_UP = ESTADO_ACTUAL_C_UP + 1;
$ESTADO_ACTUAL_2 = $ESTADO_ACTUAL_2 + $k2
```

```
// #621: pos_vector = UP_C_1 + pos;
$pos_vector = $k28 + $pos
nop * 2
// #622: dutyCycle[pos_vector] = porcentaje_C_UP;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_2
nop * 1
// #623: pS[pos_vector] = 1;
rwp = $pos_vector + 48
mem(rwp++) = $k2
// #624: porcentaje_C_UP = 1;
$porcentaje_2 = $k2
nop * 1
goto _cpp_loop_18_
_cpp_continue_18_:
// #627: ESTADO_ACTUAL_C_UP = ESTADO_PREVIO;
$ESTADO_ACTUAL_2 = $ESTADO_PREVIO
nop * 2
// #629: while (Vc_up_ref < ESTADO_ACTUAL_C_UP) {
_cpp_loop_20_:
r0 = $Vc_up_ref - $ESTADO_ACTUAL_2
r0 =__ZERO__
nop * 1
jpGE _cpp_continue_20_
// #630: if (I_UP_C > 0) {
r1 = __ANALOG_OUTPUT_5__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_21a_
// #631: pos = posVmax (INSERTED, Vcond_UP_C_1, Vcond_UP_C_2, Vcond_UP_C_3, Vcond_UP_C_4, pS[UP_C_1], pS[UP_C_2], pS[UP_C_3], pS[UP_C_4]);
push $k2
nop * 2
push $Vcond_17
nop * 2
push $Vcond_18
nop * 2
push $Vcond_19
nop * 2
push $Vcond_20
```

```
nop * 2
rwp = $k28 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k29 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k30 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k31 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 1
// #632: else {
goto _cpp_if_21b_
_cpp_if_21a_:
// #633: pos = posVmin (INSERTED, Vcond_UP_C_1, Vcond_UP_C_2,
Vcond_UP_C_3, Vcond_UP_C_4, pS[UP_C_1], pS[UP_C_2], pS[UP_C_3],
pS[UP_C_4]);
push $k2
nop * 2
push $Vcond_17
nop * 2
push $Vcond_18
nop * 2
push $Vcond_19
nop * 2
push $Vcond_20
```

```
nop * 2
rwp = $k28 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k29 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k30 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k31 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 2
_cpp_if_21b_:
// #635: ESTADO_ACTUAL_C_UP = ESTADO_ACTUAL_C_UP - 1;
$ESTADO_ACTUAL_2 = $ESTADO_ACTUAL_2 - $k2
// #636: pos_vector = UP_C_1 + pos;
$pos_vector = $k28 + $pos
nop * 2
// #637: dutyCycle[pos_vector] = porcentaje_C_UP;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_2
nop * 1
// #638: pS[pos_vector] = 0;
rwp = $pos_vector + 48
mem(rwp++) = $k1
// #639: porcentaje_C_UP = 0;
$porcentaje_2 = $k1
```

```
nop * 1
goto _cpp_loop_20_
_cpp_continue_20_:
// #646: pos = 0;
$pos = $k1
// #648: ESTADO_PREVIO = ESTADO_ACTUAL_A_LW;
$ESTADO_PREVIO = $ESTADO_ACTUAL_3
// #650: dutyCycle[LW_A_1] = pS[LW_A_1];
rwp = $k16 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k16 + 79
mem(rwp++) = r0
nop * 1
// #651: dutyCycle[LW_A_2] = pS[LW_A_2];
rwp = $k17 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k17 + 79
mem(rwp++) = r0
nop * 1
// #652: dutyCycle[LW_A_3] = pS[LW_A_3];
rwp = $k18 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k18 + 79
mem(rwp++) = r0
nop * 1
// #653: dutyCycle[LW_A_4] = pS[LW_A_4];
rwp = $k19 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k19 + 79
mem(rwp++) = r0
nop * 2
// #655: while (Va_lw_ref > ESTADO_ACTUAL_A_LW) {
_cpp_loop_22_:
r0 = $Va_lw_ref - $ESTADO_ACTUAL_3
r0 =__ZERO__
```

```
nop * 1
jpLE _cpp_continue_22_
// #656: if (I_LW_A > 0) {
r1 = __ANALOG_OUTPUT_2__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_23a_
// #657: pos = posVmin (BYPASS, Vcond_LW_A_1, Vcond_LW_A_2,
Vcond_LW_A_3, Vcond_LW_A_4, pS[LW_A_1], pS[LW_A_2], pS[LW_A_3],
pS[LW_A_4]);
push $k1
nop * 2
push $Vcond_5
nop * 2
push $Vcond_6
nop * 2
push $Vcond_7
nop * 2
push $Vcond_8
nop * 2
rwp = $k16 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k17 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k18 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k19 + 48
nop * 1
r0 = mem(rwp)
```

```
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 1
// #658: else {
goto _cpp_if_23b_
_cpp_if_23a_:
// #659: pos = posVmax (BYPASS, Vcond_LW_A_1, Vcond_LW_A_2,
Vcond_LW_A_3, Vcond_LW_A_4, pS[LW_A_1], pS[LW_A_2], pS[LW_A_3],
pS[LW_A_4]);
push $k1
nop * 2
push $Vcond_5
nop * 2
push $Vcond_6
nop * 2
push $Vcond_7
nop * 2
push $Vcond_8
nop * 2
rwp = $k16 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k17 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k18 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k19 + 48
nop * 1
r0 = mem(rwp)
```

```
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 2
_cpp_if_23b_:
// #661: ESTADO_ACTUAL_A_LW = ESTADO_ACTUAL_A_LW + 1;
$ESTADO_ACTUAL_3 = $ESTADO_ACTUAL_3 + $k2
// #662: pos_vector = LW_A_1 + pos;
$pos_vector = $k16 + $pos
nop * 2
// #663: dutyCycle[pos_vector] = porcentaje_A_LW;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_3
nop * 1
// #664: pS[pos_vector] = 1;
rwp = $pos_vector + 48
mem(rwp++) = $k2
// #665: porcentaje_A_LW = 1;
$porcentaje_3 = $k2
nop * 1
goto _cpp_loop_22_
_cpp_continue_22_:
// #668: ESTADO_ACTUAL_A_LW = ESTADO_PREVIO;
$ESTADO_ACTUAL_3 = $ESTADO_PREVIO
nop * 2
// #670: while (Va_lw_ref < ESTADO_ACTUAL_A_LW) {
_cpp_loop_24_:
r0 = $Va_lw_ref - $ESTADO_ACTUAL_3
r0 =__ZERO__
nop * 1
jpGE _cpp_continue_24_
// #671: if (I_LW_A > 0) {
r1 = __ANALOG_OUTPUT_2__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_25a_
// #672: pos = posVmax (INSERTED, Vcond_LW_A_1, Vcond_LW_A_2,
Vcond_LW_A_3, Vcond_LW_A_4, pS[LW_A_1], pS[LW_A_2], pS[LW_A_3],
pS[LW_A_4]);
```

```
push $k2
nop * 2
push $Vcond_5
nop * 2
push $Vcond_6
nop * 2
push $Vcond_7
nop * 2
push $Vcond_8
nop * 2
rwp = $k16 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k17 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k18 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k19 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 1
// #673: else {
goto _cpp_if_25b_
_cpp_if_25a_:
// #674: pos = posVmin (INSERTED, Vcond_LW_A_1, Vcond_LW_A_2,
Vcond_LW_A_3, Vcond_LW_A_4, pS[LW_A_1], pS[LW_A_2], pS[LW_A_3],
pS[LW_A_4]);
```

```
push $k2
nop * 2
push $Vcond_5
nop * 2
push $Vcond_6
nop * 2
push $Vcond_7
nop * 2
push $Vcond_8
nop * 2
rwp = $k16 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k17 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k18 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k19 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 2
_cpp_if_25b_:
// #676: ESTADO_ACTUAL_A_LW = ESTADO_ACTUAL_A_LW - 1;
$ESTADO_ACTUAL_3 = $ESTADO_ACTUAL_3 - $k2
// #677: pos_vector = LW_A_1 + pos;
$pos_vector = $k16 + $pos
nop * 2
```

```
// #678: dutyCycle[pos_vector] = porcentaje_A_LW;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_3
nop * 1
// #679: pS[pos_vector] = 0;
rwp = $pos_vector + 48
mem(rwp++) = $k1
// #680: porcentaje_A_LW = 0;
$porcentaje_3 = $k1
nop * 1
goto _cpp_loop_24_
_cpp_continue_24_:
// #687: pos = 0;
$pos = $k1
// #689: ESTADO_PREVIO = ESTADO_ACTUAL_B_LW;
$ESTADO_PREVIO = $ESTADO_ACTUAL_4
// #691: dutyCycle[LW_B_1] = pS[LW_B_1];
rwp = $k24 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k24 + 79
mem(rwp++) = r0
nop * 1
// #692: dutyCycle[LW_B_2] = pS[LW_B_2];
rwp = $k25 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k25 + 79
mem(rwp++) = r0
nop * 1
// #693: dutyCycle[LW_B_3] = pS[LW_B_3];
rwp = $k26 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k26 + 79
mem(rwp++) = r0
nop * 1
// #694: dutyCycle[LW_B_4] = pS[LW_B_4];
rwp = $k27 + 48
nop * 1
```

```
r0 = mem(rwp)
nop * 1
rwp = $k27 + 79
mem(rwp++) = r0
nop * 2
// #696: while (Vb_lw_ref > ESTADO_ACTUAL_B_LW) {
_cpp_loop_26_:
r0 = $Vb_lw_ref - $ESTADO_ACTUAL_4
r0 =__ZERO__
nop * 1
jpLE _cpp_continue_26_
// #697: if (I_LW_B > 0) {
r1 = __ANALOG_OUTPUT_4__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_27a_
// #698: pos = posVmin (BYPASS, Vcond_LW_B_1, Vcond_LW_B_2,
Vcond_LW_B_3, Vcond_LW_B_4, pS[LW_B_1], pS[LW_B_2], pS[LW_B_3],
pS[LW_B_4]);
push $k1
nop * 2
push $Vcond_13
nop * 2
push $Vcond_14
nop * 2
push $Vcond_15
nop * 2
push $Vcond_16
nop * 2
rwp = $k24 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k25 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
```

```
rwp = $k26 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k27 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 1
// #699: else {
goto _cpp_if_27b_
_cpp_if_27a_:
// #700: pos = posVmax (BYPASS, Vcond_LW_B_1, Vcond_LW_B_2,
Vcond_LW_B_3, Vcond_LW_B_4, pS[LW_B_1], pS[LW_B_2], pS[LW_B_3],
pS[LW_B_4]);
push $k1
nop * 2
push $Vcond_13
nop * 2
push $Vcond_14
nop * 2
push $Vcond_15
nop * 2
push $Vcond_16
nop * 2
rwp = $k24 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k25 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
```

```
rwp = $k26 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k27 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 2
_cpp_if_27b_:
// #702: ESTADO_ACTUAL_B_LW = ESTADO_ACTUAL_B_LW + 1;
$ESTADO_ACTUAL_4 = $ESTADO_ACTUAL_4 + $k2
// #703: pos_vector = LW_B_1 + pos;
$pos_vector = $k24 + $pos
nop * 2
// #704: dutyCycle[pos_vector] = porcentaje_B_LW;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_4
nop * 1
// #705: pS[pos_vector] = 1;
rwp = $pos_vector + 48
mem(rwp++) = $k2
// #706: porcentaje_B_LW = 1;
$porcentaje_4 = $k2
nop * 1
goto _cpp_loop_26_
_cpp_continue_26_:
// #709: ESTADO_ACTUAL_B_LW = ESTADO_PREVIO;
$ESTADO_ACTUAL_4 = $ESTADO_PREVIO
nop * 2
// #711: while (Vb_lw_ref < ESTADO_ACTUAL_B_LW) {
_cpp_loop_28_:
r0 = $Vb_lw_ref - $ESTADO_ACTUAL_4
r0 =__ZERO__
nop * 1
jpGE _cpp_continue_28_
// #712: if (I_LW_B > 0) {
```

```
r1 = __ANALOG_OUTPUT_4__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_29a_
// #713: pos = posVmax (INSERTED, Vcond_LW_B_1, Vcond_LW_B_2,
Vcond_LW_B_3, Vcond_LW_B_4, pS[LW_B_1], pS[LW_B_2], pS[LW_B_3],
pS[LW_B_4]);
push $k2
nop * 2
push $Vcond_13
nop * 2
push $Vcond_14
nop * 2
push $Vcond_15
nop * 2
push $Vcond_16
nop * 2
rwp = $k24 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k25 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k26 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k27 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
```

```
call _posVmax_
$pos = r0
nop * 1
// #714: else {
goto _cpp_if_29b_
_cpp_if_29a_:
// #715: pos = posVmin (INSERTED, Vcond_LW_B_1, Vcond_LW_B_2,
Vcond_LW_B_3, Vcond_LW_B_4, pS[LW_B_1], pS[LW_B_2], pS[LW_B_3],
pS[LW_B_4]);
push $k2
nop * 2
push $Vcond_13
nop * 2
push $Vcond_14
nop * 2
push $Vcond_15
nop * 2
push $Vcond_16
nop * 2
rwp = $k24 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k25 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k26 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k27 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
```

```
call _posVmin_
$pos = r0
nop * 2
_cpp_if_29b_:
// #717: ESTADO_ACTUAL_B_LW = ESTADO_ACTUAL_B_LW - 1;
$ESTADO_ACTUAL_4 = $ESTADO_ACTUAL_4 - $k2
// #718: pos_vector = LW_B_1 + pos;
$pos_vector = $k24 + $pos
nop * 2
// #719: dutyCycle[pos_vector] = porcentaje_B_LW;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_4
nop * 1
// #720: pS[pos_vector] = 0;
rwp = $pos_vector + 48
mem(rwp++) = $k1
// #721: porcentaje_B_LW = 0;
$porcentaje_4 = $k1
nop * 1
goto _cpp_loop_28_
_cpp_continue_28_:
// #728: pos = 0;
$pos = $k1
// #730: ESTADO_PREVIO = ESTADO_ACTUAL_C_LW;
$ESTADO_PREVIO = $ESTADO_ACTUAL_5
// #732: dutyCycle[LW_C_1] = pS[LW_C_1];
rwp = $k32 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k32 + 79
mem(rwp++) = r0
nop * 1
// #733: dutyCycle[LW_C_2] = pS[LW_C_2];
rwp = $k33 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k33 + 79
mem(rwp++) = r0
nop * 1
// #734: dutyCycle[LW_C_3] = pS[LW_C_3];
rwp = $k34 + 48
```

```
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k34 + 79
mem(rwp++) = r0
nop * 1
// #735: dutyCycle[LW_C_4] = pS[LW_C_4];
rwp = $k35 + 48
nop * 1
r0 = mem(rwp)
nop * 1
rwp = $k35 + 79
mem(rwp++) = r0
nop * 2
// #737: while (Vc_lw_ref > ESTADO_ACTUAL_C_LW) {
_cpp_loop_30_:
r0 = $Vc_lw_ref - $ESTADO_ACTUAL_5
r0 =__ZERO__
nop * 1
jpLE _cpp_continue_30_
// #738: if (I_LW_C > 0) {
r1 = __ANALOG_OUTPUT_6__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_31a_
// #739: pos = posVmin (BYPASS, Vcond_LW_C_1, Vcond_LW_C_2,
Vcond_LW_C_3, Vcond_LW_C_4, pS[LW_C_1], pS[LW_C_2], pS[LW_C_3],
pS[LW_C_4]);
push $k1
nop * 2
push $Vcond_21
nop * 2
push $Vcond_22
nop * 2
push $Vcond_23
nop * 2
push $Vcond_24
nop * 2
rwp = $k32 + 48
nop * 1
r0 = mem(rwp)
```

```
nop * 2
push r0
nop * 2
rwp = $k33 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k34 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k35 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 1
// #740: else {
goto _cpp_if_31b_
_cpp_if_31a_:
// #741: pos = posVmax (BYPASS, Vcond_LW_C_1, Vcond_LW_C_2,
Vcond_LW_C_3, Vcond_LW_C_4, pS[LW_C_1], pS[LW_C_2], pS[LW_C_3],
pS[LW_C_4]);
push $k1
nop * 2
push $Vcond_21
nop * 2
push $Vcond_22
nop * 2
push $Vcond_23
nop * 2
push $Vcond_24
nop * 2
rwp = $k32 + 48
nop * 1
r0 = mem(rwp)
```

```
nop * 2
push r0
nop * 2
rwp = $k33 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k34 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k35 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 2
_cpp_if_31b_:
// #743: ESTADO_ACTUAL_C_LW = ESTADO_ACTUAL_C_LW + 1;
$ESTADO_ACTUAL_5 = $ESTADO_ACTUAL_5 + $k2
// #744: pos_vector = LW_C_1 + pos;
$pos_vector = $k32 + $pos
nop * 2
// #745: dutyCycle[pos_vector] = porcentaje_C_LW;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_5
nop * 1
// #746: pS[pos_vector] = 1;
rwp = $pos_vector + 48
mem(rwp++) = $k2
// #747: porcentaje_C_LW = 1;
$porcentaje_5 = $k2
nop * 1
goto _cpp_loop_30_
_cpp_continue_30_:
// #750: ESTADO_ACTUAL_C_LW = ESTADO_PREVIO;
```

```
$ESTADO_ACTUAL_5 = $ESTADO_PREVIO
nop * 2
// #752: while (Vc_lw_ref < ESTADO_ACTUAL_C_LW) {
_cpp_loop_32_:
r0 = $Vc_lw_ref - $ESTADO_ACTUAL_5
r0 =__ZERO__
nop * 1
jpGE _cpp_continue_32_
// #753: if (I_LW_C > 0) {
r1 = __ANALOG_OUTPUT_6__
nop * 2
r0 = r1 - $k1
r0 =__ZERO__
nop * 1
jpLE _cpp_if_33a_
// #754: pos = posVmax (INSERTED, Vcond_LW_C_1, Vcond_LW_C_2,
Vcond_LW_C_3, Vcond_LW_C_4, pS[LW_C_1], pS[LW_C_2], pS[LW_C_3],
pS[LW_C_4]);
push $k2
nop * 2
push $Vcond_21
nop * 2
push $Vcond_22
nop * 2
push $Vcond_23
nop * 2
push $Vcond_24
nop * 2
rwp = $k32 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k33 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k34 + 48
nop * 1
r0 = mem(rwp)
```

```
nop * 2
push r0
nop * 2
rwp = $k35 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmax_
$pos = r0
nop * 1
// #755: else {
goto _cpp_if_33b_
_cpp_if_33a_:
// #756: pos = posVmin (INSERTED, Vcond_LW_C_1, Vcond_LW_C_2, Vcond_LW_C_3, Vcond_LW_C_4, pS[LW_C_1], pS[LW_C_2], pS[LW_C_3], pS[LW_C_4]);
push $k2
nop * 2
push $Vcond_21
nop * 2
push $Vcond_22
nop * 2
push $Vcond_23
nop * 2
push $Vcond_24
nop * 2
rwp = $k32 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k33 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
nop * 2
rwp = $k34 + 48
nop * 1
r0 = mem(rwp)
```

```
nop * 2
push r0
nop * 2
rwp = $k35 + 48
nop * 1
r0 = mem(rwp)
nop * 2
push r0
noop * 1
call _posVmin_
$pos = r0
nop * 2
_cpp_if_33b_:
// #758: ESTADO_ACTUAL_C_LW = ESTADO_ACTUAL_C_LW - 1;
$ESTADO_ACTUAL_5 = $ESTADO_ACTUAL_5 - $k2
// #759: pos_vector = LW_C_1 + pos;
$pos_vector = $k32 + $pos
nop * 2
// #760: dutyCycle[pos_vector] = porcentaje_C_LW;
rwp = $pos_vector + 79
mem(rwp++) = $porcentaje_5
nop * 1
// #761: pS[pos_vector] = 0;
rwp = $pos_vector + 48
mem(rwp++) = $k1
// #762: porcentaje_C_LW = 0;
$porcentaje_5 = $k1
nop * 1
goto _cpp_loop_32_
_cpp_continue_32_:
// #765: Vuz = loopMicroseconds();
r0 = __LOOPMICROSECONDS__
nop * 2
$Vuz = r0
nop * 1

jpNT loop
halt

_posVmin_:

// #141: double Vmin;
#variable -S80 Vmin
```

```
push k0
nop * 2
// #142: Vmin = 1600;
$Vmin = $k0
// #144: int pos;
#variable -S80 pos
push k0
nop * 2
// #145: pos = 0;
$pos = $k1
// #147: if ((pS_1 == estado_buscado) && (Vcond_1 < Vmin)) {
r0 = $pS_1 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_1_
r0 = $Vcond_1 - $Vmin
nop * 1
r0 = LT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_1_:
r0 = r0 & __ONE__
nop * 2
jpEQ _cpp_if_1a_
// #148: pos = 0;
$pos = $k1
// #149: Vmin = Vcond_1;
$Vmin = $Vcond_1
nop * 2
_cpp_if_1a_:
// #152: if ((pS_2 == estado_buscado) && (Vcond_2 < Vmin)) {
r0 = $pS_2 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_2_
r0 = $Vcond_2 - $Vmin
nop * 1
r0 = LT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_2_:
r0 = r0 & __ONE__
nop * 2
jpEQ _cpp_if_2a_
// #153: pos = 1;
```

```
$pos = $k2
// #154: Vmin = Vcond_2;
$Vmin = $Vcond_2
nop * 2
_cpp_if_2a_:
// #157: if ((pS_3 == estado_buscado) && (Vcond_3 < Vmin)) {
r0 = $pS_3 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_3_
r0 = $Vcond_3 - $Vmin
nop * 1
r0 = LT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_3_:
r0 = r0 & __ONE__
nop * 2
jpEQ _cpp_if_3a_
// #158: pos = 2;
$pos = $k3
// #159: Vmin = Vcond_3;
$Vmin = $Vcond_3
nop * 2
_cpp_if_3a_:
// #162: if ((pS_4 == estado_buscado) && (Vcond_4 < Vmin)) {
r0 = $pS_4 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_4_
r0 = $Vcond_4 - $Vmin
nop * 1
r0 = LT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_4_:
r0 = r0 & __ONE__
nop * 2
jpEQ _cpp_if_4a_
// #163: pos = 3;
$pos = $k4
nop * 2
_cpp_if_4a_:
// #167: return pos;
r0 = $pos
```

```
nop * 1
ret (dsp += 11)

_posVmax_:

// #173: double Vmax;
#variable -S80 Vmax
push k0
nop * 2
// #174: Vmax = -1600;
$Vmax = $k5
// #176: int pos;
#variable -S80 pos
push k0
nop * 2
// #177: pos = 0;
$pos = $k1
// #179: if ((pS_1 == estado_buscado) && (Vcond_1 > Vmax)) {
r0 = $pS_1 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_5_
r0 = $Vcond_1 - $Vmax
nop * 1
r0 = GT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_5_:
r0 = r0 & __ONE__
nop * 2
jpEQ _cpp_if_5a_
// #180: pos = 0;
$pos = $k1
// #181: Vmax = Vcond_1;
$Vmax = $Vcond_1
nop * 2
_cpp_if_5a_:
// #184: if ((pS_2 == estado_buscado) && (Vcond_2 > Vmax)) {
r0 = $pS_2 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_6_
r0 = $Vcond_2 - $Vmax
nop * 1
```

```
r0 = GT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_6_:
r0 = r0 & __ONE__
nop * 2
jpEQ _cpp_if_6a_
// #185: pos = 1;
$pos = $k2
// #186: Vmax = Vcond_2;
$Vmax = $Vcond_2
nop * 2
_cpp_if_6a_:
// #189: if ((pS_3 == estado_buscado) && (Vcond_3 > Vmax)) {
r0 = $pS_3 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_7_
r0 = $Vcond_3 - $Vmax
nop * 1
r0 = GT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_7_:
r0 = r0 & __ONE__
nop * 2
jpEQ _cpp_if_7a_
// #190: pos = 2;
$pos = $k3
// #191: Vmax = Vcond_3;
$Vmax = $Vcond_3
nop * 2
_cpp_if_7a_:
// #194: if ((pS_4 == estado_buscado) && (Vcond_4 > Vmax)) {
r0 = $pS_4 - $estado_buscado
r0 =__ZERO__
nop * 1
jpNE _cpp_expr_8_
r0 = $Vcond_4 - $Vmax
nop * 1
r0 = GT ? __ONE__ : __ZERO__
nop * 2
_cpp_expr_8_:
r0 = r0 & __ONE__
nop * 2
```

```
jpEQ _cpp_if_8a_
// #195: pos = 3;
$pos = $k4
nop * 2
_cpp_if_8a_:
// #199: return pos;
r0 = $pos
nop * 1
ret (dsp += 11)

_panelPV_:

// #219: double Ipv;
#variable -S80 Ipv
push k0
nop * 2
// #221: if (Vpv < 700) {
r0 = $Vpv - $k6
r0 =__ZERO__
nop * 1
jpGE _cpp_if_9a_
// #222: Ipv = (CONSTANTE_A * Vpv) + CONSTANTE_B;
p0 = $k7 * $Vpv
nop * 11
$Ipv = p0 + $k8
nop * 1
// #224: else {
goto _cpp_if_9b_
_cpp_if_9a_:
// #225: Ipv = ((((CONSTANTE_D * Vpv + CONSTANTE_E) * Vpv +
CONSTANTE_F) * Vpv + CONSTANTE_G) * Vpv + CONSTANTE_H) * Vpv +
CONSTANTE_I;
p0 = $k9 * $Vpv
nop * 11
r1 = p0 + $k10
nop * 2
p0 = r1 * $Vpv
nop * 11
r1 = p0 + $k11
nop * 2
p0 = r1 * $Vpv
nop * 11
r1 = p0 + $k12
```

```
nop * 2
p0 = r1 * $Vpv
nop * 11
r1 = p0 + $k13
nop * 2
p0 = r1 * $Vpv
nop * 11
$Ipv = p0 + $k14
nop * 2
_cpp_if_9b_:
// #228: return Ipv;
r0 = $Ipv
nop * 1
ret (dsp += 2)
```