PBL: Simulación del control de un inversor trifásico que alimenta un motor de inducción (DSP)

1.1 Inicialización

Main. Se inicia llamando a las funciones de configuración, como el InitPWM que configura los registros para el PWM, el InitADC para inicializar el conversor analógico-digital, initCustomVariables para inicializar algunas variables utilizadas y el ConfigInterrupts para configurar las interrupciones.

```
113 void main(void)
114 {
115     int i=0;
116
117     DeviceInit();   // Device Life support & GPIO
118
119    // Waiting for enable flag set.
120    //Flag has to be set manually for the program to run.
121    while (EnableFlag==FALSE)
122     {
123       BackTicker++;
124     }
125
126 // Timing sync for background loops
127 // Timer period definitions found in device specific PeripheralHeaderIncludes.h
128     CpuTimer2Regs.PRD.all =  mSec100;   // Background task
129
130     InitPWM();
131     InitADC();
132
133 // Initialize user structures and variables
134     initCustomVariables();
135
136 // Call HVDMC Protection function
137     HVDMC_Protection();
138
139 //INTERRUPT CONFIGURATION
140     ConfigInterrupts();
141
142
143 // IDLE loop. Just sit and loop forever:
144     for(;;)  //infinite loop
145     {
146         BackgroundLoop1();
147
148     }
149 } //END MAIN CODE
```

En el InitPWM se configuran los registros como los del periodo del PWM, disparo de ADC, acción de las puertas A y B, tiempo muerto, etc…

```
155 void InitPWM(voi HVACI_Scalar_2833x_Base/DSP2833x_GlobalVariableDefs.c
156 {
157   // Initialize PWM modules 1, 2 and 3.
158
159   EALLOW;
160   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0; // Disable ePWM modules
161   EDIS;
162
163   EPwm1Regs.TBCTL.bit.CTRMODE = 0x3;   // Time-Base Control Register - Disable counter
164   EPwm1Regs.TBCTL.all = 0xC013;        // Time-Base Control Register - Free run,Sync with this register,Stop-freeze counter operation (default on reset)
165   EPwm1Regs.TBCTR = 0x0000;            // Time-Base Counter Register - Reset timer counter
166   EPwm1Regs.TBPRD = 15000 ;            // Time-Base Period Register - Timer = 150Mhz/5Khz/2 - PWM period = 2 x TBPRD x TTBCLK
167   EPwm1Regs.TBPHS.half.TBPHS = 0x0000; // Time-Base Phase Register - Phase=0
168   EPwm1Regs.ETSEL.all = 0x0A00;        // Event-Trigger Selection Register - Enable EPWMxSOCA(Start Of Conversion de canal A) pulse, Enable event time-base counter equal to period (TBC
169   EPwm1Regs.ETPS.all = 0x0100;         // Event-Trigger Prescale Register - Generate the EPWMxSOCA pulse on the first event: ETPS[SOCACNT] = 0,1
170   EPwm1Regs.CMPCTL.all = 0x0002;       // Counter-Compare Control Register - Load on either CTR = Zero or CTR = PRD
171   EPwm1Regs.AQCTLA.all = 0x0090;       // Action-Qualifier Output A Control Register - Set: force EPWMxA output high, Clear: force EPWMxA output low
172   EPwm1Regs.AQCTLB.all = 0x0060;       // Action-Qualifier Output B Control Register - Clear: force EPWMxB output low, Set: force EPWMxB output high
173   EPwm1Regs.AQSFRC.all = 0x0000;       // Action-Qualifier Software Force Register - Disabled \ Does nothing
174   EPwm1Regs.AQCSFRC.all = 0x0000;      // Action-qualifier Continuous Software Force Register - Disabled \ Does nothing
175   EPwm1Regs.DBCTL.all = 0x000B;        //Dead-Band Generator Control Register - Active high complementary (AHC). EPWMxB is inverted, Dead-band is fully enabled for both rising-edge del
176
177   EPwm2Regs.TBCTL.bit.CTRMODE = 0x3;   // Time-Base Control Register - Disable counter
178   EPwm2Regs.TBCTL.all = 0xC007;        // Time-Base Control Register - Free run,Sync with ePWM1,Stop-freeze counter operation (default on reset)
179   EPwm2Regs.TBCTR = 0x0000;            // Time-Base Counter Register - Reset timer counter
180   EPwm2Regs.TBPRD = 15000;             // Time-Base Period Register - Timer = 150Mhz/5Khz - PWM period = 2 x TBPRD x TTBCLK
181   EPwm2Regs.TBPHS.half.TBPHS = 0x0000; // Time-Base Phase Register - Phase=0
182   EPwm2Regs.AQCTLA.all = 0x0090;       // Action-Qualifier Output A Control Register - Set: force EPWMxA output high, Clear: force EPWMxA output low
183   EPwm2Regs.AQCTLB.all = 0x0060;       // Action-Qualifier Output B Control Register - Clear: force EPWMxB output low, Set: force EPWMxB output high
184   EPwm2Regs.AQSFRC.all = 0x0000;       // Action-Qualifier Software Force Register - Disabled \ Does nothing
185   EPwm2Regs.AQCSFRC.all = 0x0000;      // Action-qualifier Continuous Software Force Register - Disabled \ Does nothing
186   EPwm2Regs.DBCTL.all = 0x000B;        //Dead-Band Generator Control Register - Active high complementary (AHC). EPWMxB is inverted, Dead-band is fully enabled for both rising-edge del
187
188   EPwm3Regs.TBCTL.bit.CTRMODE = 0x3;   // Time-Base Control Register - Disable counter
189   EPwm3Regs.TBCTL.all = 0xC007;        // Time-Base Control Register - Free run,Sync with ePWM1,Stop-freeze counter operation (default on reset)
190   EPwm3Regs.TBCTR = 0x0000;            // Time-Base Counter Register - Reset timer counter
191   EPwm3Regs.TBPRD = 15000;             // Time-Base Period Register - Timer = 150Mhz/5Khz - PWM period = 2 x TBPRD x TTBCLK
192   EPwm3Regs.TBPHS.half.TBPHS = 0x0000; // Time-Base Phase Register - Phase=0
193   EPwm3Regs.AQCTLA.all = 0x0090;       // Action-Qualifier Output A Control Register - Set: force EPWMxA output high, Clear: force EPWMxA output low
194   EPwm3Regs.AQCTLB.all = 0x0060;       // Action-Qualifier Output B Control Register - Clear: force EPWMxB output low, Set: force EPWMxB output high
195   EPwm3Regs.AQSFRC.all = 0x0000;       // Action-Qualifier Software Force Register - Disabled \ Does nothing
196   EPwm3Regs.AQCSFRC.all = 0x0000;      // Action-qualifier Continuous Software Force Register - Disabled \ Does nothing
197   EPwm3Regs.DBCTL.all = 0x000B;        //Dead-Band Generator Control Register - Active high complementary (AHC). EPWMxB is inverted, Dead-band is fully enabled for both rising-edge del
198
199    /* Init Dead-Band Generator for EPWM1-EPWM3*/
200    EPwm1Regs.DBFED = 200;
201    EPwm1Regs.DBRED = 200;
202    EPwm2Regs.DBFED = 200;
203    EPwm2Regs.DBRED = 200;
204    EPwm3Regs.DBFED = 200;
205    EPwm3Regs.DBRED = 200;
206
207    /* Chopper unit disabled */
208    EPwm1Regs.PCCTL.bit.CHPEN = 0;     // PWM chopper unit disabled
209    EPwm2Regs.PCCTL.bit.CHPEN = 0;     // PWM chopper unit disabled
210    EPwm3Regs.PCCTL.bit.CHPEN = 0;     // PWM chopper unit disabled
211
212    EPwm1Regs.TBCTL.bit.CTRMODE = 0x2; // Time-Base Control Register - Enable counter Up-down-count mode
213    EPwm2Regs.TBCTL.bit.CTRMODE = 0x2; // Time-Base Control Register - Enable counter Up-down-count mode
214    EPwm3Regs.TBCTL.bit.CTRMODE = 0x2; // Time-Base Control Register - Enable counter Up-down-count mode
215
216   EALLOW;
217   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1; // Enable ePWM modules
218   EDIS;
219 }
```

En el InitADC se configuran los registros de los conversores, qué pines se van a leer, cuantos son, etc…

```
void InitADC(void)
{
    //Initizlization of the calibration data
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
    ADC_cal();
    EDIS;
    AdcRegs.ADCTRL3.all = 0x00E0;  /* Power up bandgap/reference/ADC circuits*/
    DELAY_US(ADC_usDELAY);         /* Delay before converting ADC channels*/

    // Initialize ADC module
    AdcRegs.ADCTRL1.bit.RESET = 1; // ADC Control Register 1 - Reset
    asm(" RPT #22 || NOP"); // Provides the required delay between writes to ADCTRL1

    AdcRegs.ADCREFSEL.bit.REF_SEL = 0;      // ADC Reference Select Register - Internal reference selected (default)
    AdcRegs.ADCTRL3.all = 0x00EC;         // ADC Control Register 3 - The bandgap and reference circuitry is powered up, The analog circuitry inside t
    AdcRegs.ADCMAXCONV.all = 0x6;           // Maximum Conversion Channels Register - Number of conversions = 7
    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x1;  // ADC Input Channel Select Sequencing Control Registers - Channels Selected ADCINA1 ( Phase A Current)
    AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x9;  // ADC Input Channel Select Sequencing Control Registers - Channels Selected ADCINB1 ( Phase  B  Current)
    AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x3;  // ADC Input Channel Select Sequencing Control Registers - Channels Selected ADCINA3 ( Phase C Current)
    AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0xF;  // ADC Input Channel Select Sequencing Control Registers - Channels Selected ADCINB7 (Phase A Voltage)
    AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0xE;  // ADC Input Channel Select Sequencing Control Registers - Channels Selected ADCINB6 (Phase B Voltage)
    AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0xC;  // ADC Input Channel Select Sequencing Control Registers - Channels Selected ADCINB4 (Phase C Voltage)
    AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x7;  // ADC Input Channel Select Sequencing Control Registers - Channels Selected ADCINA7 (DC Bus  Voltage)

    AdcRegs.ADCTRL1.all = 0x0710;           // ADC Control Register 1 - Acquisition window size = 7 + 1 times the ADCLK period,ADCCLK = Fclk/1, Casca
    AdcRegs.ADCTRL2.all = 0x0900;           // ADC Control Register 2 - Interrupt request by INT_SEQ1 is enabled, Allows SEQ1/SEQ to be started by ePW
}
```

Para las interrupciones se configura para llamar el método MainISR para disparar siempre que hace la conversión del ADC en el período de 200us.

```
void    ConfigInterrupts(void)
{
    //--- Enable the MainISR interrupt with timer
    EALLOW;
    PieVectTable.ADCINT = &MainISR;     // PIE MUXed Peripheral Interrupt Vector Table - Set method for interruption
    EDIS;
    PieCtrlRegs.PIEIER1.bit.INTx6 = 1;  // Enable the ADCINT interrupt in the PIE module
    IER |= 0x0001;                      // Interrupt Enable Register - Enable
    EINT;
    ERTM;
    test = 2;
}
```

En la inicialización hay una función que es utilizada para calcular algunos valores como el q para convertir la corriente y tensión, además de calcular la relación entre la tensión nominal y la tensión del BUS:

```
void    initCustomVariables(void)
{
    overcurrent = 0;

    //Calcular la relacion V/F
    double VmotorPU = Vmotor/(Vdc*0.6124);
    if (KC > 0) {
        if (VmotorPU > 1.154) {
            VmotorPU = 1.154;
        }
    } else {
        if (VmotorPU > 1) {
            VmotorPU = 1;
        }
    }
    RelacionVF = (VmotorPU-0)/(1-0);

    //double qADC = (fondo_ADC_positivo-fondo_ADC_negativo)/(pow(2,NBITS)-1);
    qCurrent = (fondo_corriente_positivo-fondo_corriente_negativo)/(pow(2,NBITS)-1);
    qVoltage = (fondo_tension_positivo-fondo_tension_negativo)/(pow(2,NBITS)-1);
}
```

El método MainISR es responsable de ejecutar toda la lógica de generación de la moduladora, incluso la aplicación de la rampa para el cambio de consigna, lectura de los valores, cálculo de la moduladora senoidal y la protección de sobrecorriente.

```
interrupt void MainISR(void)
{
    int i=0;
    test=5;
// Verifying the ISR
    IsrTicker++;

    vhz1.SpFreq = SpeedRef;     //USER INPUT: Speed set-point for control function, as example.
    test = 1;
// ---------------------------------------------------------------------------
//    Call the Volt/hertz profile and ramp
// ---------------------------------------------------------------------------
    rampaVF();
// ---------------------------------------------------------------------------
//    ADC measurement calculation. Read AdcMirror registers and calculate phisical value.
// ---------------------------------------------------------------------------
    readAndProcessADCValues();
// ---------------------------------------------------------------------------
//    Call the Sinusoidal modulation
// ---------------------------------------------------------------------------
    double OutputSenalSenoidal[3];
    SinusoidalModulation(OutputSenalSenoidal);

// ---------------------------------------------------------------------------
//  Connect inputs to the PWM compare levels
// ---------------------------------------------------------------------------

    (*ePWM[1]).CMPA.half.CMPA = (OutputSenalSenoidal[0]+1)*7500; // Counter-Compare A Register - +1 = offset \ 7500 = PWMPeriod/SysClockPeriod
    (*ePWM[2]).CMPA.half.CMPA = (OutputSenalSenoidal[1]+1)*7500; // Counter-Compare A Register - +1 = offset \ 7500 = PWMPeriod/SysClockPeriod
    (*ePWM[3]).CMPA.half.CMPA = (OutputSenalSenoidal[2]+1)*7500; // Counter-Compare A Register - +1 = offset \ 7500 = PWMPeriod/SysClockPeriod
// ---------------------------------------------------------------------------
// CONTROL DE SOBRECORRIENTE - SI LA CORRIENTE > 5
// ---------------------------------------------------------------------------
    //llamada a la rutina que comprueba si hay o no sobrecorriente en cualquiera de las 3 fases
    current_protection();
    //Si hay sobrecorriente abrimos todos

    if( overcurrent > 0 )
    {
        EALLOW;
        (*ePWM[1]).TZFRC.bit.OST = 1;
        (*ePWM[2]).TZFRC.bit.OST = 1;
        (*ePWM[3]).TZFRC.bit.OST = 1;
        EDIS;
    }
// ---------------------------------------------------------------------------
//    Connect inputs of the debug buffers and update buffers, for graph viewing
// ---------------------------------------------------------------------------

// ---------------------------------------------------------------------------
//    Interrupt management
// ---------------------------------------------------------------------------

// Resetear el SEQ1 al estado CONV00
    AdcRegs.ADCTRL2.bit.RST_SEQ1=1;
    AdcRegs.ADCTRL2.bit.RST_SEQ2=1;

// Limpiar el flag de la interrupción ADC SEQ1
    AdcRegs.ADCST.bit.INT_SEQ1_CLR=1;
    AdcRegs.ADCST.bit.INT_SEQ2_CLR=1;

// Acknowledge interrupt to recieve more interrupts from PIE group 3
    PieCtrlRegs.PIEACK.bit.ACK6 = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}   // ISR Ends Here
```

1.2 Funciones MainISR

En esta función los valores leídos del ADC Mirror son convertidos con el "q" calculado en la inicialización para convertir los valores del contador para los valores de corriente y tensión. Incluso, también se calcula la potencial total y real monitoreando la corriente y la variable ángulo para identificar el desfase entre la tensión que pasa por el 0 con pi radianes, con cuando la corriente pasa por el 0. Luego, es posible calcular la potencia real con esta diferencia de ángulo entre tensión y corriente.

```
313 void    readAndProcessADCValues(void)
314 {
315     ADCCONV[0] = AdcMirror.ADCRESULT0;
316     ADCCONV[1] = AdcMirror.ADCRESULT1;
317     ADCCONV[2] = AdcMirror.ADCRESULT2;
318     ADCCONV[3] = AdcMirror.ADCRESULT3;
319     ADCCONV[4] = AdcMirror.ADCRESULT4;
320     ADCCONV[5] = AdcMirror.ADCRESULT5;
321     ADCCONV[6] = AdcMirror.ADCRESULT6;
322 
323     current[0] = AdcMirror.ADCRESULT0 * qCurrent + fondo_corriente_negativo;
324     current[1] = AdcMirror.ADCRESULT1 * qCurrent + fondo_corriente_negativo;
325     current[2] = AdcMirror.ADCRESULT2 * qCurrent + fondo_corriente_negativo;
326 
327     voltage[0] = AdcMirror.ADCRESULT3 * qVoltage + fondo_tension_negativo;
328     voltage[1] = AdcMirror.ADCRESULT4 * qVoltage + fondo_tension_negativo;
329     voltage[2] = AdcMirror.ADCRESULT5 * qVoltage + fondo_tension_negativo;
330 
331     busVoltage = AdcMirror.ADCRESULT6 * qVoltage + fondo_tension_negativo;
332     potencia = current[0]*voltage[0] + current[1]*voltage[1] + current[2]*voltage[2];
333 
334     if (stage == 0) {
335         if (corrienteMax < current[0]) {
336             corrienteMax = current[0];
337         }
338         if (current[0] < 0 ) {
339             double deltaAngulo = integral-PI_Num;
340             potenciaTotal = sqrt(3)*Vdc*0.612*RelacionVF*velocidad*corrienteMax/sqrt(2);
341             potenciaReal = potenciaTotal*cos(deltaAngulo);
342             corrienteMax = 0;
343             stage = 1;
344         }
345     } else if (stage == 1 && current[0] > 0.3) {
346         stage = 0;
347     }
348 }
```

En la función rampaVF se controla el incremento en la velocidad/frecuencia de la moduladora sumando incrementos en cada ciclo hasta que la diferencia entre la consigna imputada y la velocidad sea menor que el incremento.

```
354 void    rampaVF(void)
355 {
356     //Calcular la relacion V/F
357     /*double VmotorPU = Vmotor/(busVoltage*0.6124);
358     if (KC > 0) {
359         if (VmotorPU > 1.154) {
360             VmotorPU = limiteSuperior;
361         }
362     } else {
363         if (VmotorPU > 1) {
364             VmotorPU = limiteSuperior;
365         }
366     }
367     RelacionVF = (VmotorPU-0)/(1-0);*/
368
369     //Rampa
370     if (SpeedRef < OFFSET) {
371         SpeedRef = OFFSET;
372     } else if (SpeedRef > 1) {
373         SpeedRef = 1;
374     }
375
376     float diferencia = SpeedRef - velocidad;
377     double incremento = RAMPA * T;
378     if (diferencia > incremento) {
379         velocidad = velocidad + incremento;
380     } else if (diferencia < -incremento) {
381         velocidad = velocidad - incremento;
382     }
383 }
```

Para generar la señal senoidal, se aplica la relación entre tensión y frecuencia y la variable velocidad después de aplicar la rampaVF. Si está con tercer armónico activo suma el tercer armónico en la señal, si no, la salida es la senoide pura. EL ángulo es integrado, así como en la simulación en Simulink, y si es mayor que 2Pi, se substrae 2Pi para evitar el overflow.

```
389 void    SinusoidalModulation(double OutputSenal[3])
390 {
391     integral = 2*PI_Num*velocidad*FREQFUNDAMENTAL*T + integral;
392     if (integral > 2*PI_Num) {
393         integral = integral-2*PI_Num;
394     }
395
396     if (KC > 0) {
397         double terceraArmonica = velocidad * KC * sin (3*integral);
398         OutputSenal[0] = RelacionVF * (velocidad * sin(integral + 0) + terceraArmonica);
399         OutputSenal[1] = RelacionVF * (velocidad * sin(integral - 2*PI_Num/3) + terceraArmonica);
400         OutputSenal[2] = RelacionVF * (velocidad * sin(integral + 2*PI_Num/3) + terceraArmonica);
401     } else {
402         OutputSenal[0] = RelacionVF * velocidad * sin(integral + 0);
403         OutputSenal[1] = RelacionVF * velocidad * sin(integral - 2*PI_Num/3);
404         OutputSenal[2] = RelacionVF * velocidad * sin(integral + 2*PI_Num/3);
405     }
406 }
```

Para la protección de sobre corriente, se compara los valores leídos en el ADC con lo limite para caso sea mayor, actuar la protección de sobre corriente.

```
412 void    current_protection(void)
413 {
414     if (abs(current[0]) > overcurrent_limit ||
415         abs(current[1]) > overcurrent_limit ||
416         abs(current[2]) > overcurrent_limit) {
417         overcurrent = 1;
418     }
419 }
```

1.3 Ejercicio opcional (calcular potencia real)

Para calcular la potencia real, se divide el cálculo en 2 etapas, una primera cuando la corriente está por encima de 0 y está siempre buscando el valor máximo de la corriente hasta que sea menor que cero.

Como el motor es una carga inductiva, se supone que el ángulo Pi de la señal moduladora es cuando la fundamental de la tensión está a 0, luego, la corriente está atrasada. Con eso, cuando la corriente ha pasado por el eje 0, tenemos que el ángulo actual, menos el ángulo cuando la tensión pasa por el eje 0 (Pi) es la diferencia del ángulo entre la corriente y tensión.

Con eso, tenemos que el valor RMS de la corriente, es el valor máximo de la corriente divido por raíz de 2 y la tensión fundamental RMS puede ser obtenida por la formula del filtro que es Vdc por 0.612 por el índice de modulación de la amplitud. Por fin, para sacar la potencia trifásica del motor, hay que multiplicar la tensión RMS, por la corriente RMS por raíz de 3.

Calculada la potencia total, la potencial real es la total multiplicada por el cos del ángulo entre la corriente y tensión. Obtenidas las potencias, prosigo para el estado 1 donde aguarda que la corriente pase de nuevo por el eje 0 y quede positivo, el valor utilizado es 0.3 porque por error de medición si es elegido mayor que 0, puede volver debajo de cero en la siguiente medición por error.

```
334     if (stage == 0) {
335         if (corrienteMax < current[0]) {
336             corrienteMax = current[0];
337         }
338         if (current[0] < 0 ) {
339             double deltaAngulo = integral-PI_Num;
340             potenciaTotal = sqrt(3)*Vdc*0.612*RelacionVF*velocidad*corrienteMax/sqrt(2);
341             potenciaReal = potenciaTotal*cos(deltaAngulo);
342             corrienteMax = 0;
343             stage = 1;
344         }
345     } else if (stage == 1 && current[0] > 0.3) {
346         stage = 0;
347     }
```

1.4 Simulaciones en bancada

Comprobación del dead band en todas las salidas del PWM:

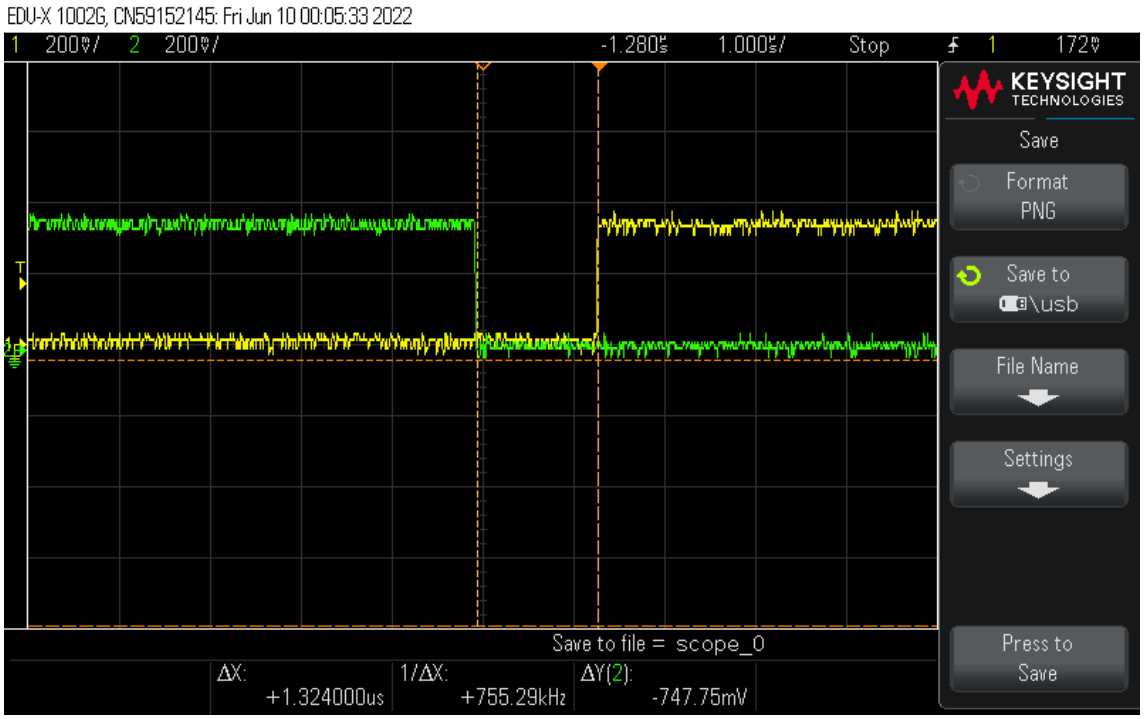


Ilustración 1 - Dead band del PWM1A y PWM1B

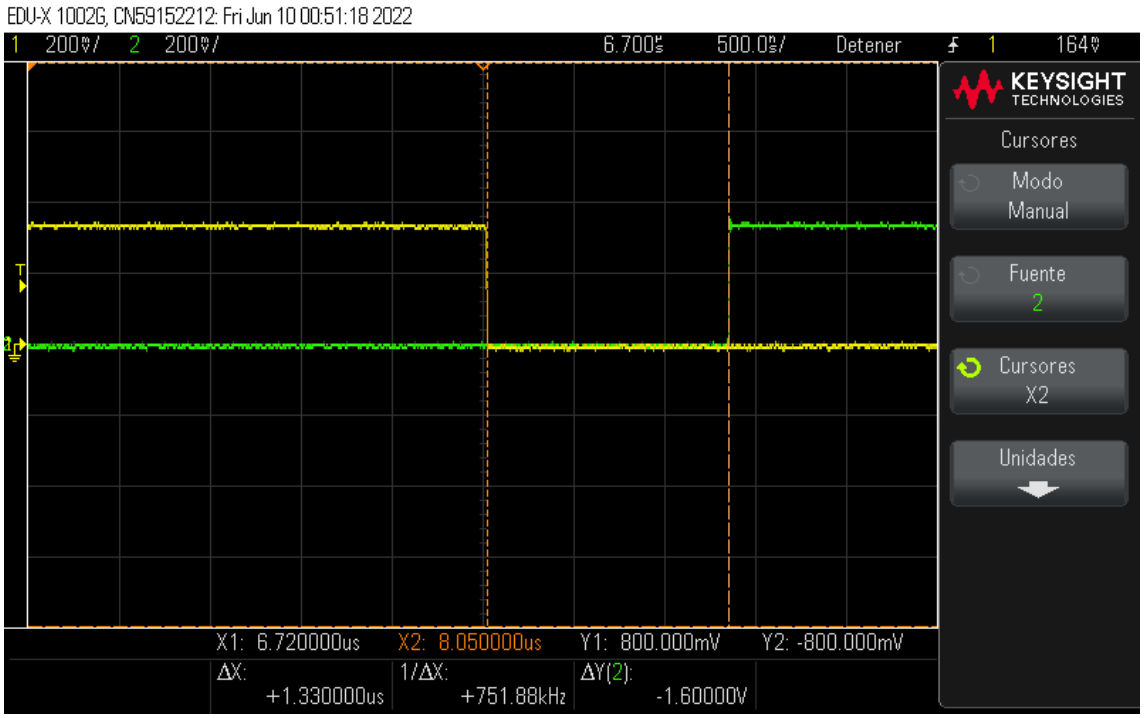


Ilustración 2 - Dead band del PWM2A y PWM2B

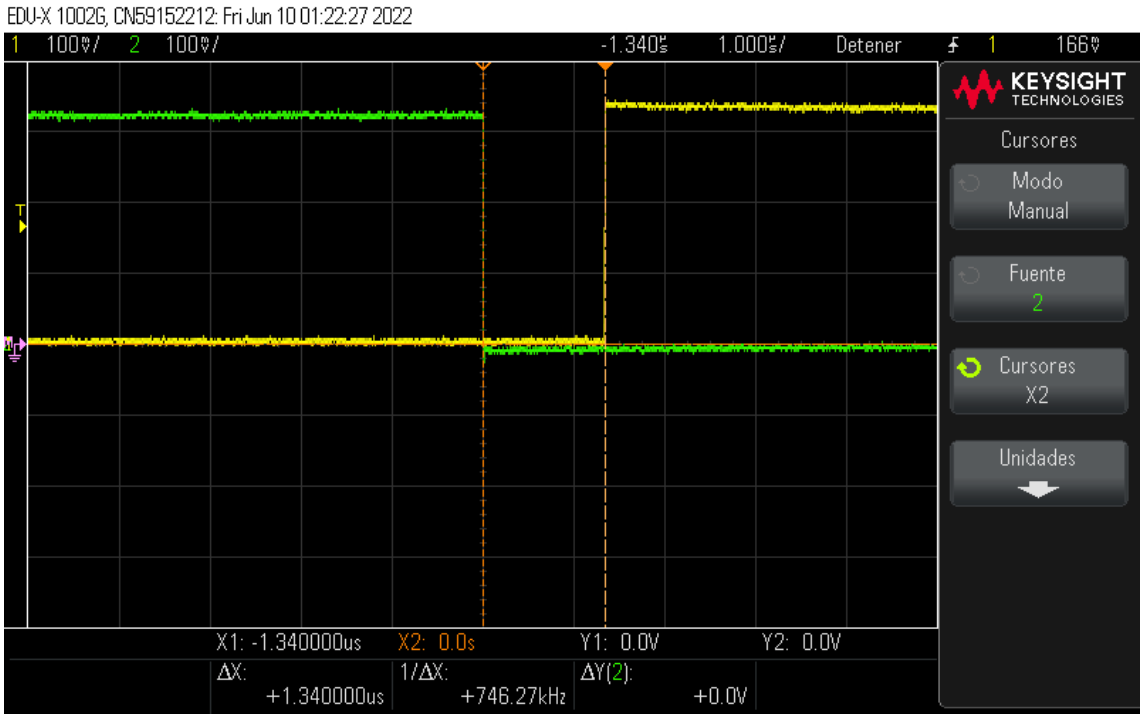


Ilustración 3 - Dead band del PWM3A y PWM3B

Comprobación del cambio de velocidad en todas las salidas del PWM

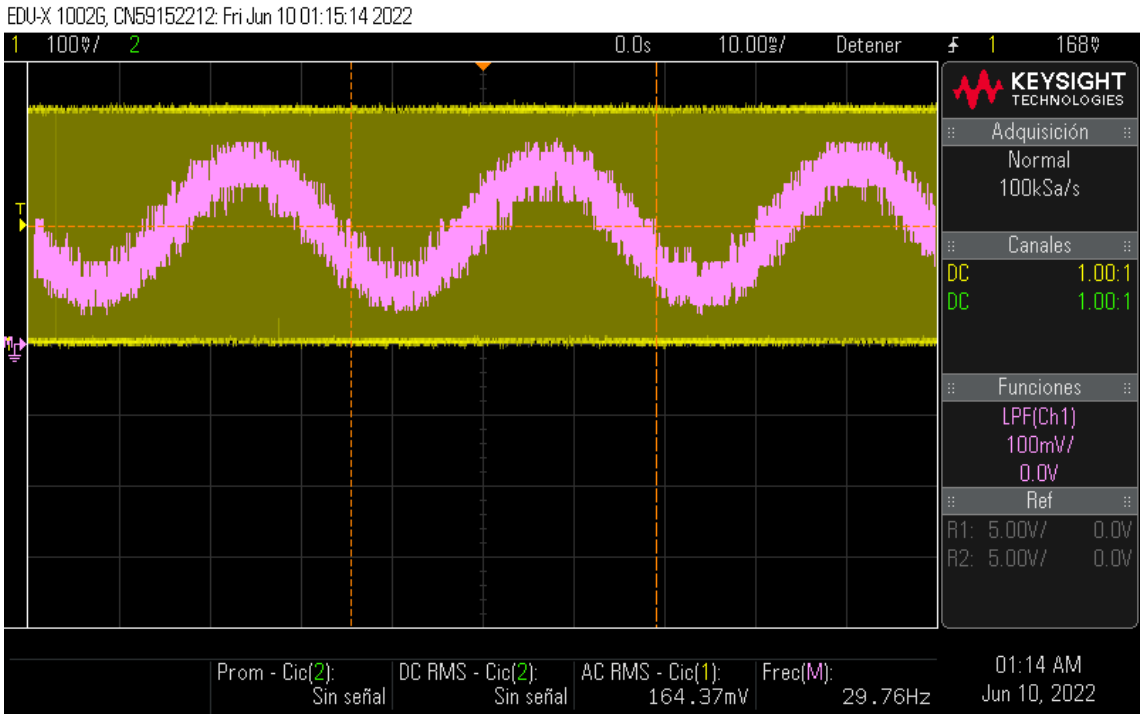


Ilustración 4 - PWM1 A 30Hz

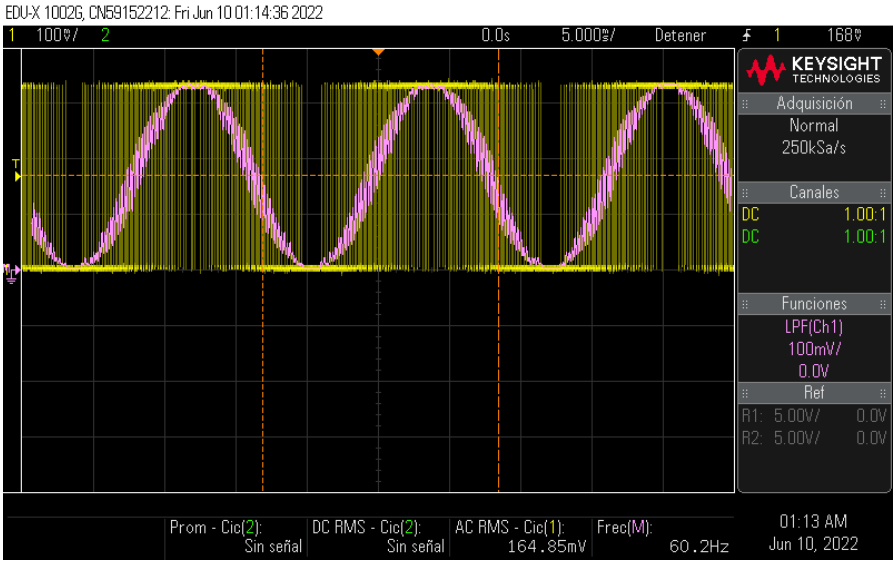


Ilustración 5 - PWM1 A 60Hz

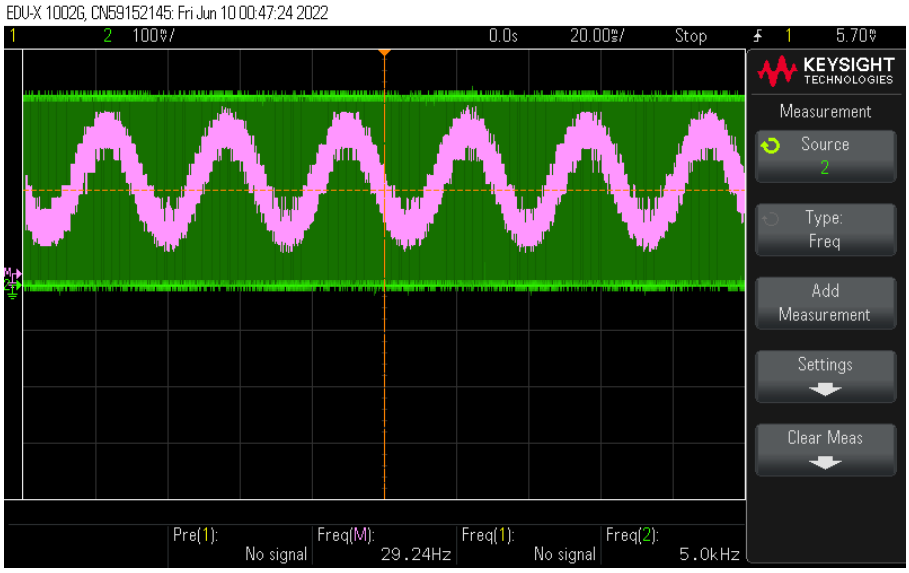


Ilustración 6 - PWM1 B 30Hz

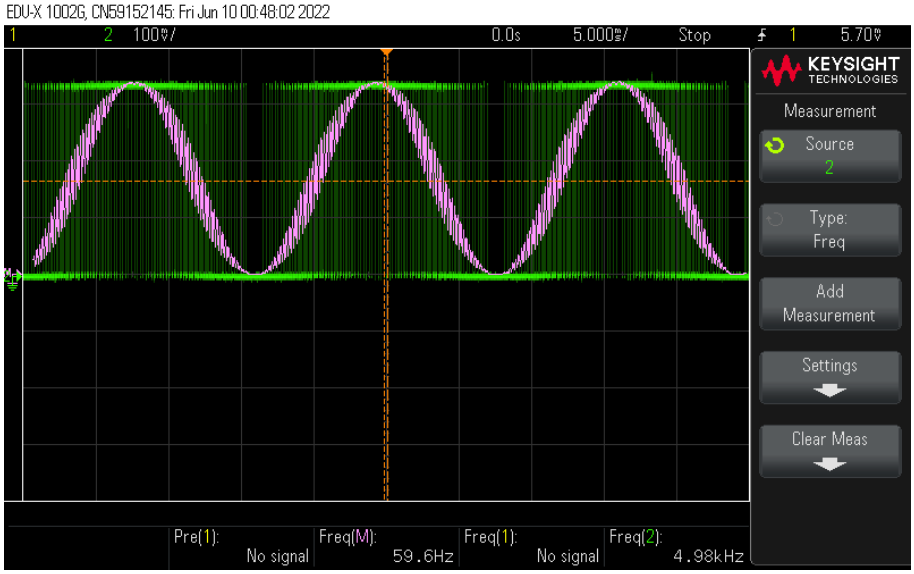


Ilustración 7 - PWM1 B 60Hz

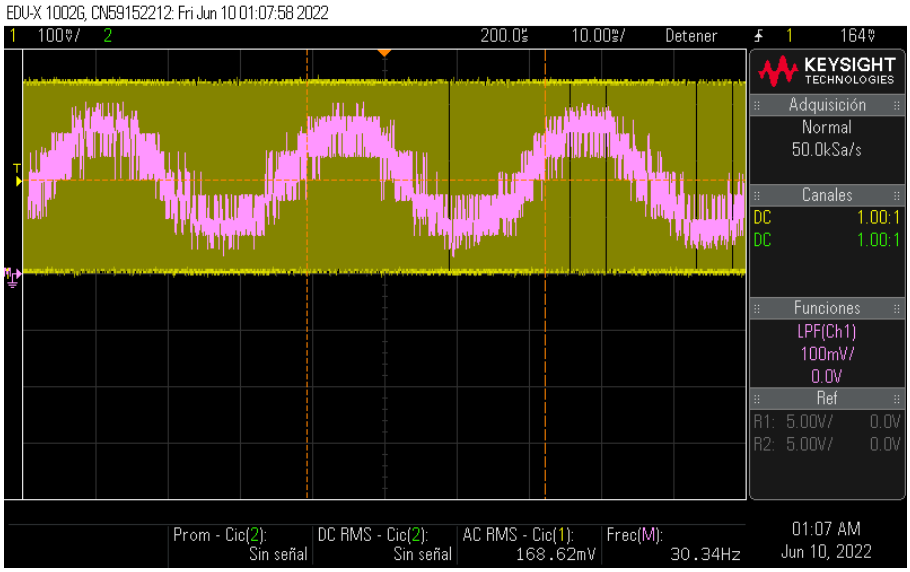


Ilustración 8 - PWM2 A 30Hz

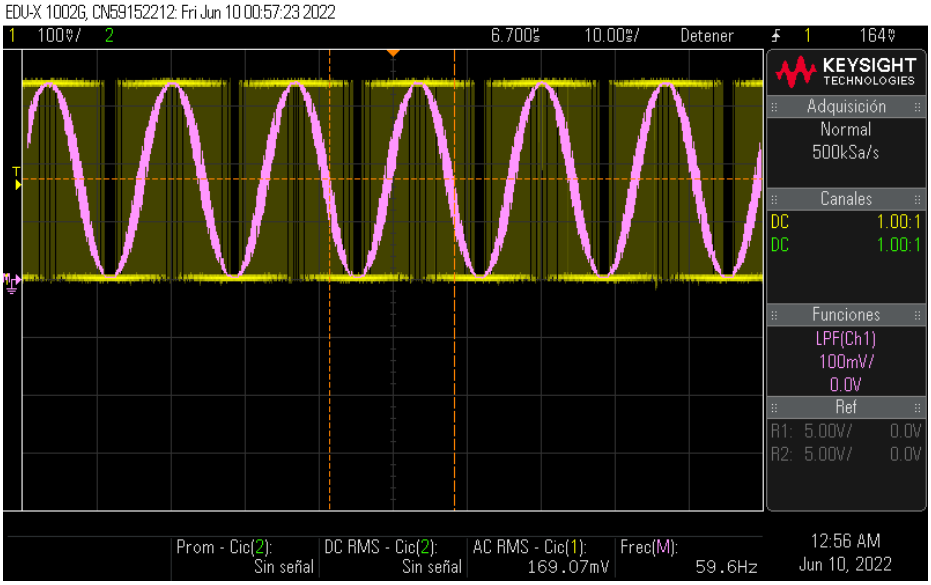


Ilustración 9 - PWM2 A 60Hz

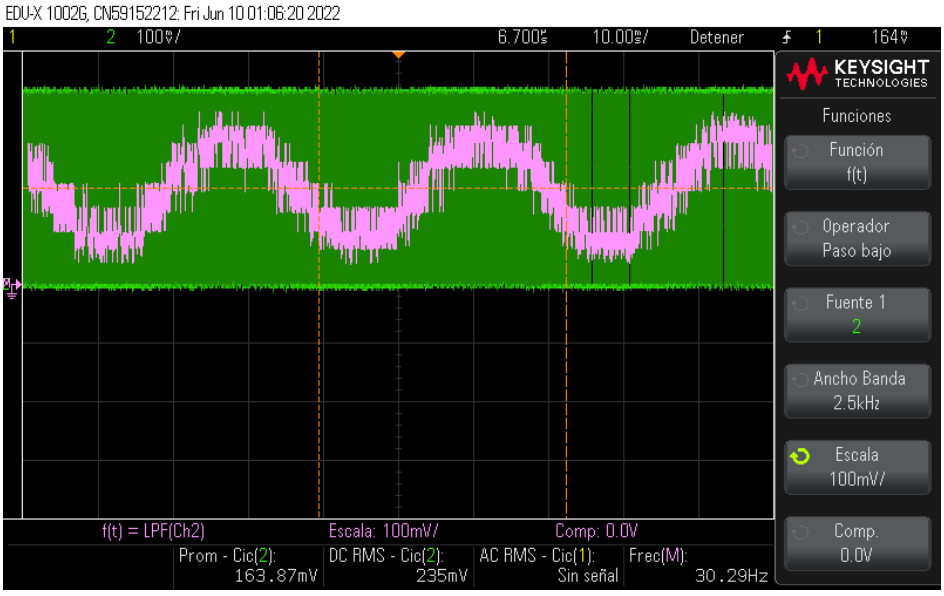


Ilustración 10 - PWM2 B 30Hz

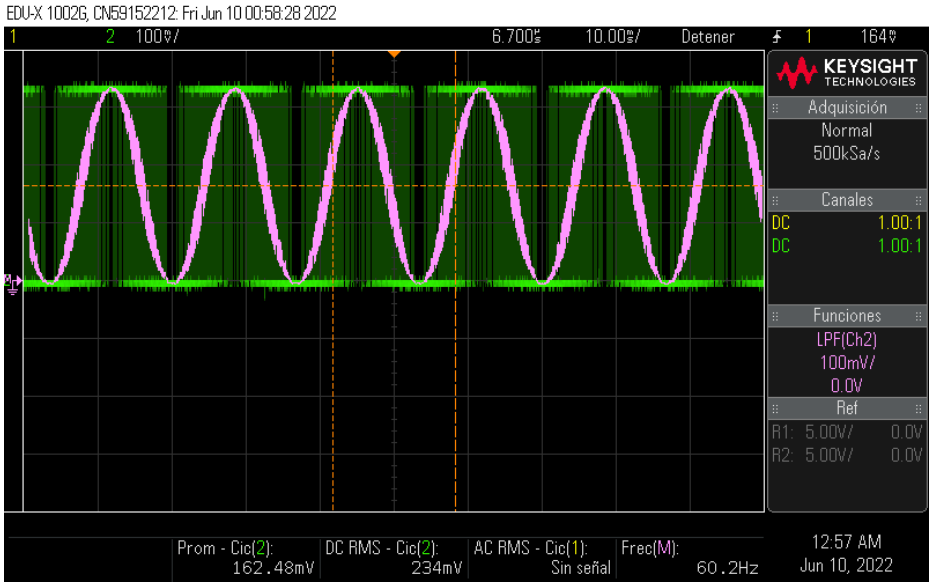


Ilustración 11 - PWM2 B 60Hz

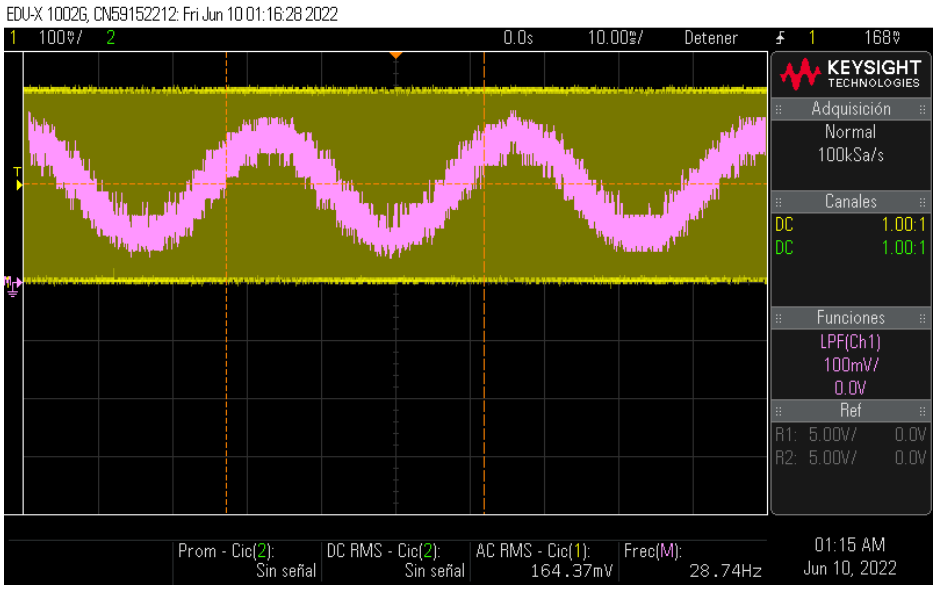


Ilustración 12 - PWM3 A 30Hz

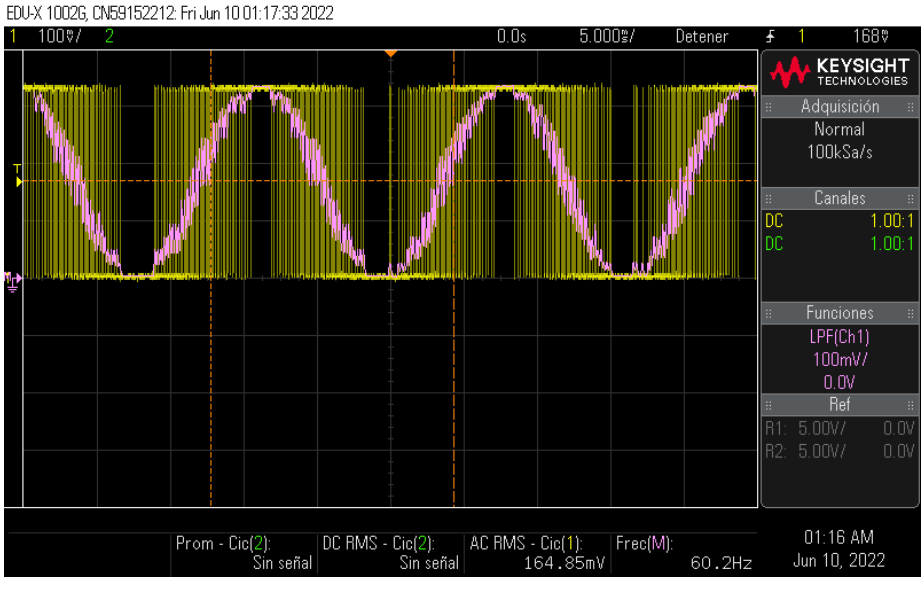


Ilustración 13 - PWM3 A 60Hz

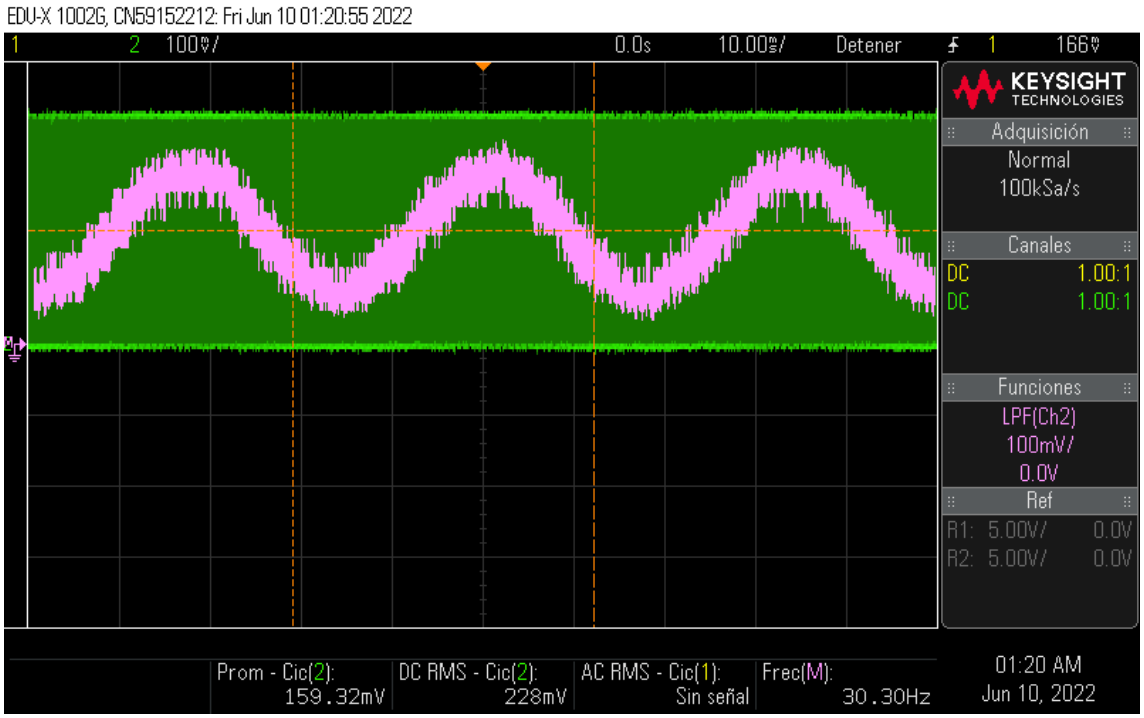


Ilustración 14 - PWM3 B 30Hz

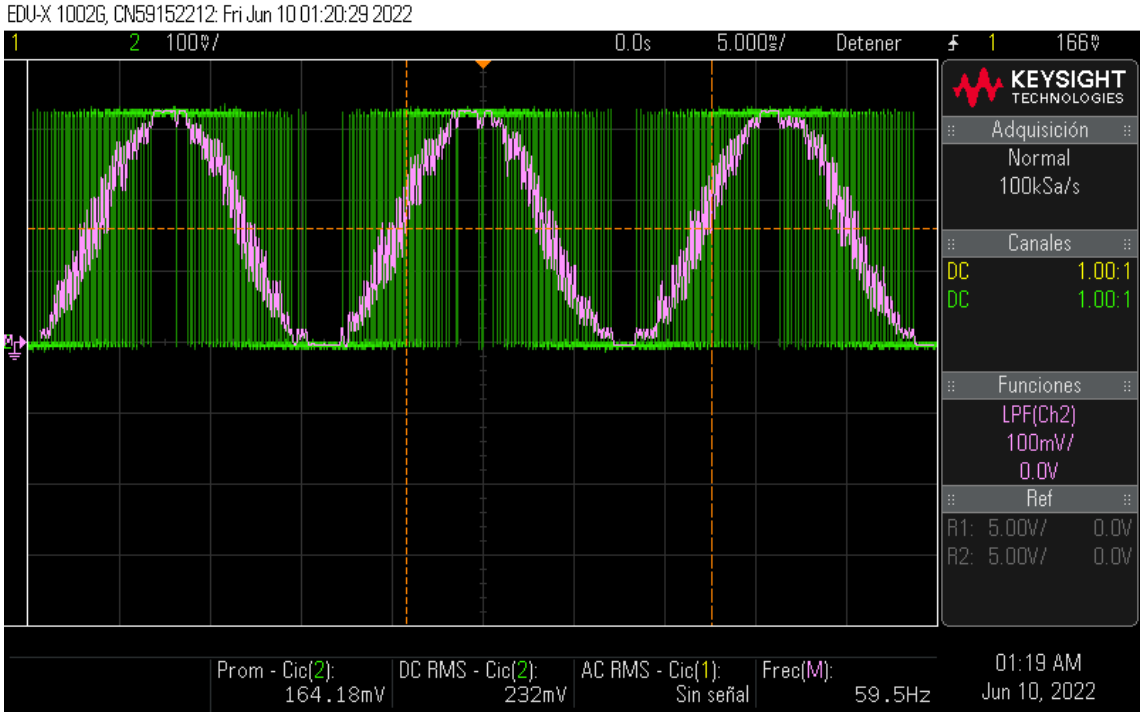


Ilustración 15 - PWM3 B 60Hz

Por fin, para comprobar el desfase entre las salidas, hemos sacado las siguientes gráficas:

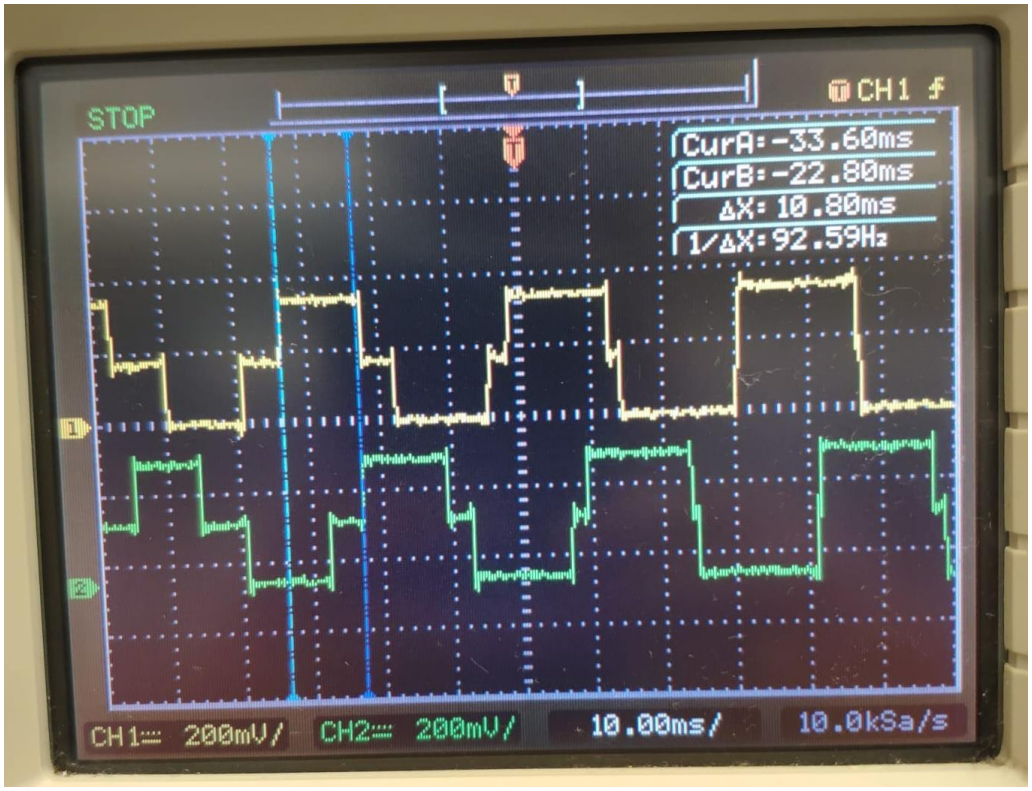


Ilustración 16 - Desfase ePWM1 vs ePWM2

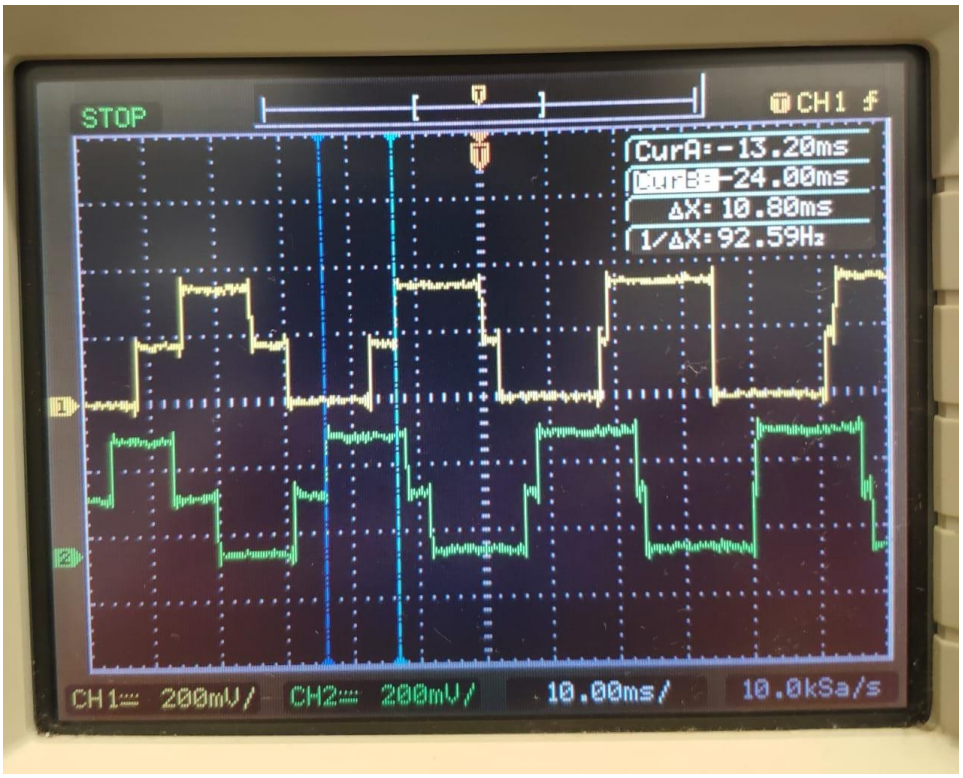


Ilustración 17 - Desfase ePWM1 vs ePWM3

1.5 Cuantificación de los errores

A continuación, se muestran los errores cuantificados para las corrientes.

Estas diferencias se deben a que la resolución de 12 bits que empleamos solo puede representar mediciones en múltiplos de q al traducirlo al mundo digital, y por ello, siempre tendremos errores de medición de como máximo q/2.

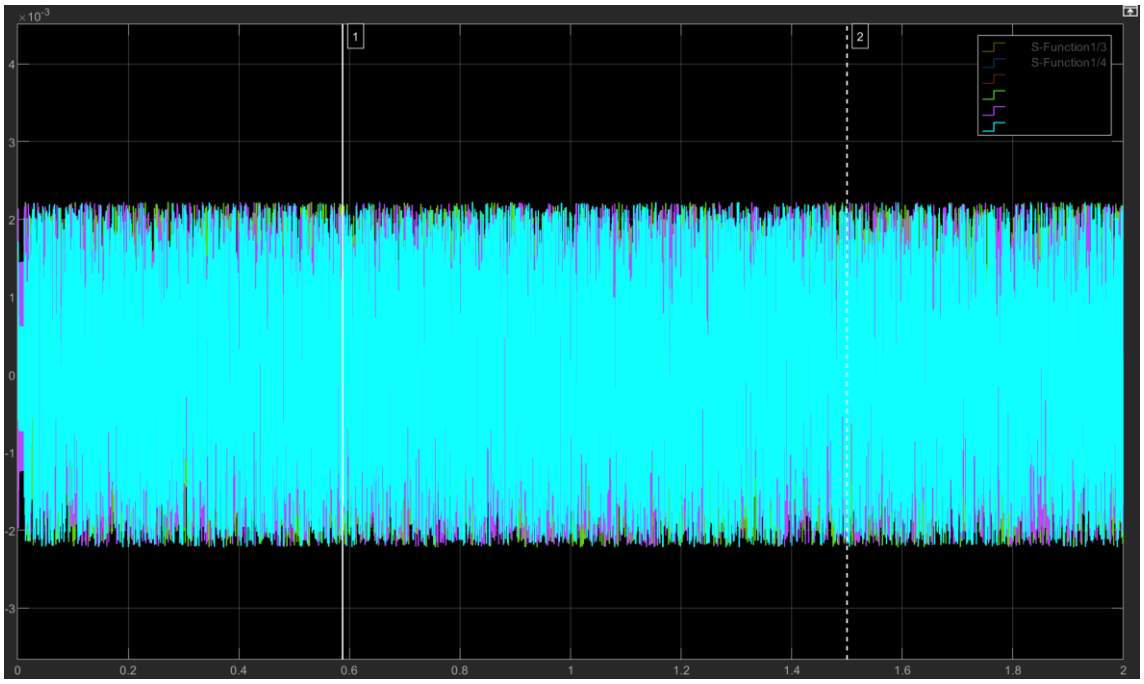
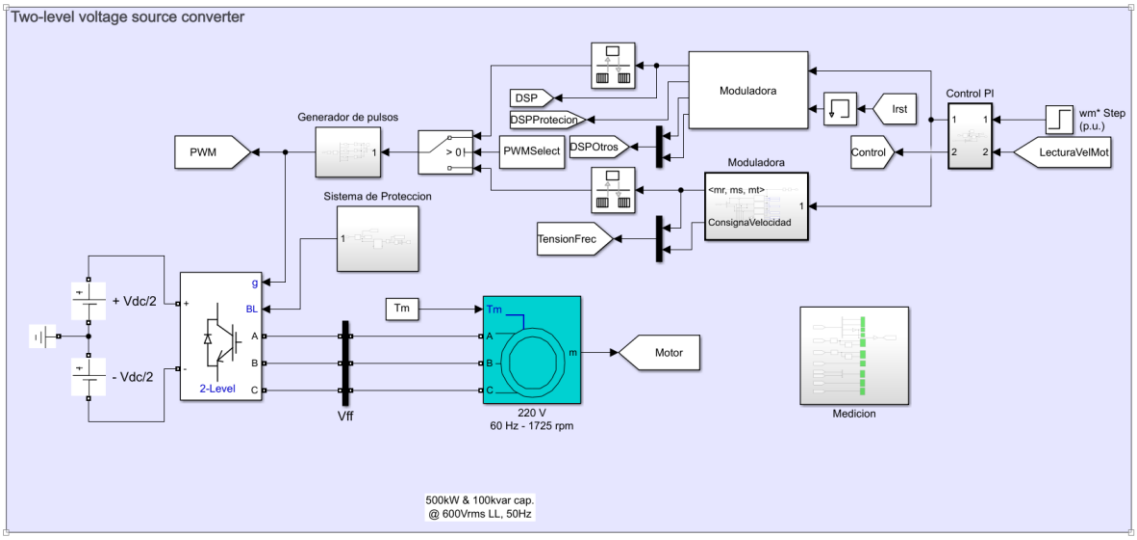


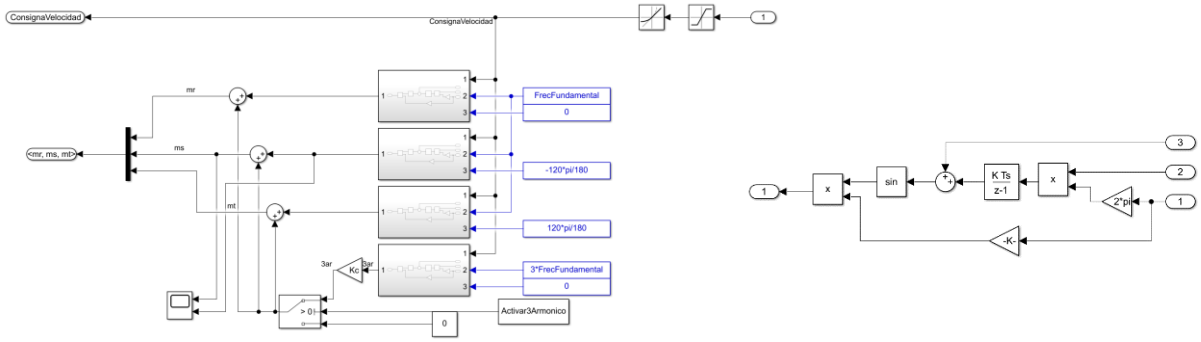
Ilustración 18 – Errores de cuantificación de corrientes

Práctica 5: Simulación del control de un inversor trifásico que alimenta un motor de inducción

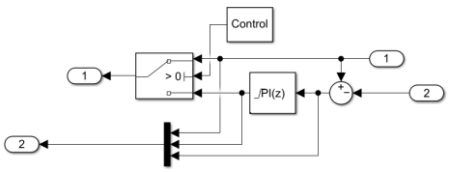
5.1 Modelización



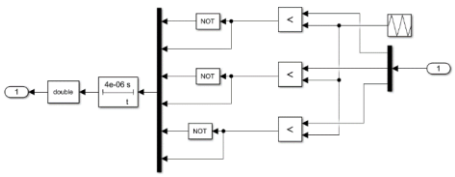
Moduladora en Simulink:



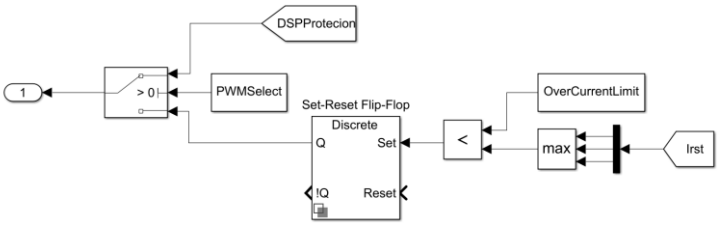
Control PI:



Generador de pulsos:



Sistema de protección:

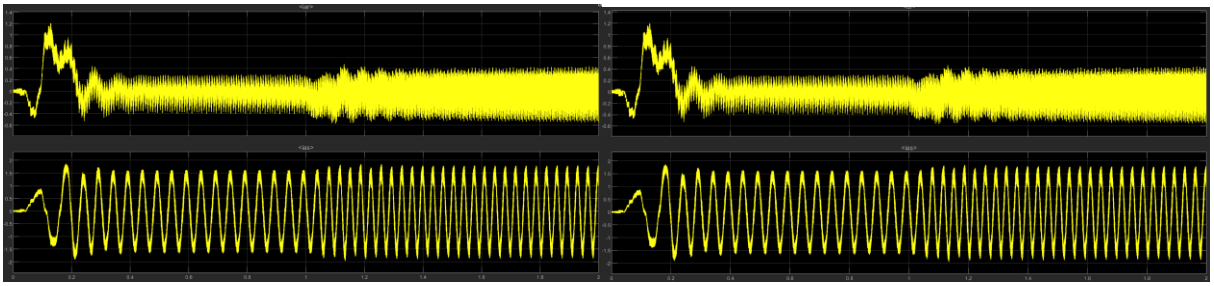


5.2 Ejercicio obligatorio (modulación escalar SPWM)

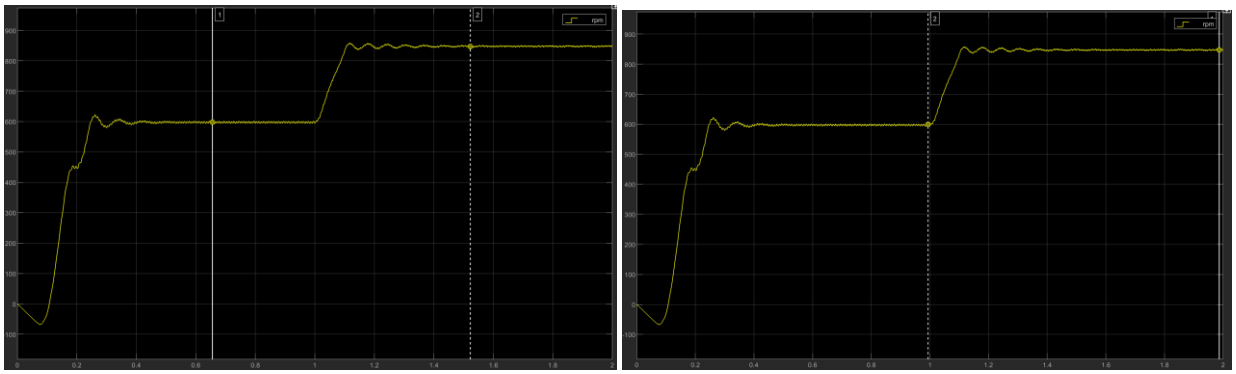
Antes de simular se han de fijar los siguientes valores de los parámetros de los switches:

PWMSelect = 0 y 1*% <=0 para seleccionar Moduladora Simulink; >0 para seleccionar Moduladora S-Function.*
Control = 1; *% <= 0 para seleccionar Control PI; > 0 para seleccionar Control Step.*
Activar3Armonico = 1; *% <= 0 para desactivar 3er armónico; >0 para activar 3er armónico*

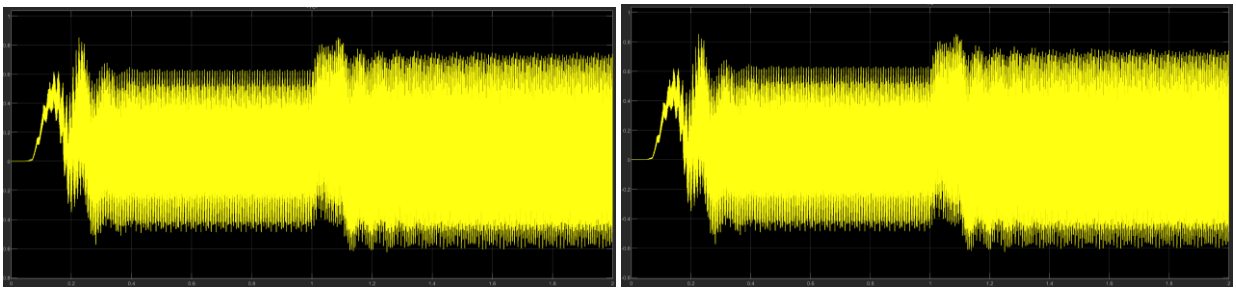
Corrientes de estator y Rotor [A] (Sin control PI; izquierda Potencia; derecha S-Function):



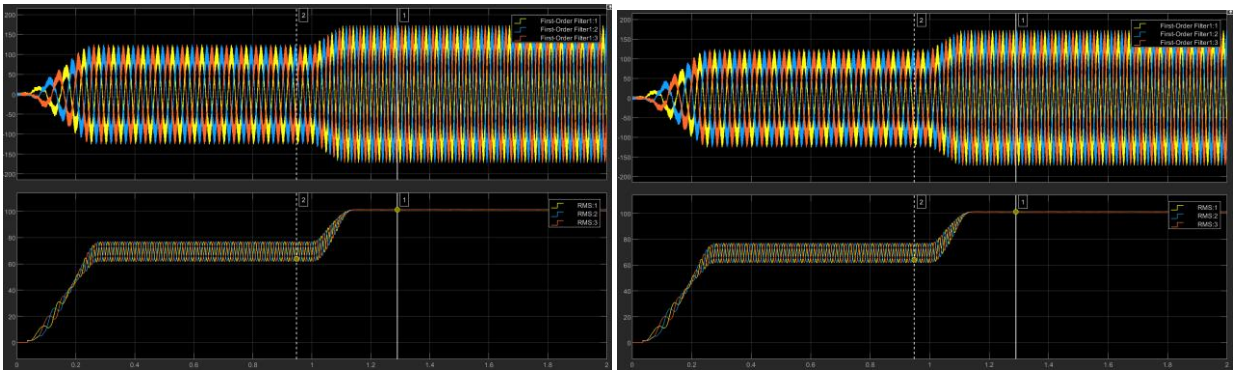
RPMs del motor [rpm] (Sin control PI; izquierda Potencia; derecha S-Function):



Par motor [Nm] (Sin control PI; izquierda Potencia; derecha S-Function):



Tensiones fase-fase filtradas y RMS [V] (Sin control PI; izquierda Potencia; derecha S-Function):

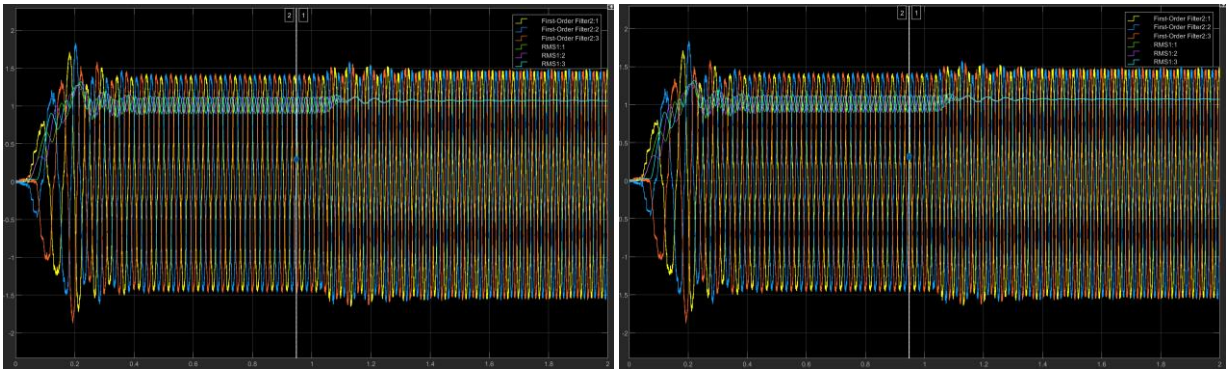


Se puede observar que todas las gráficas son prácticamente idénticas para el caso de simulación con bloques en Simulink y para el caso de simulación con la S-function.

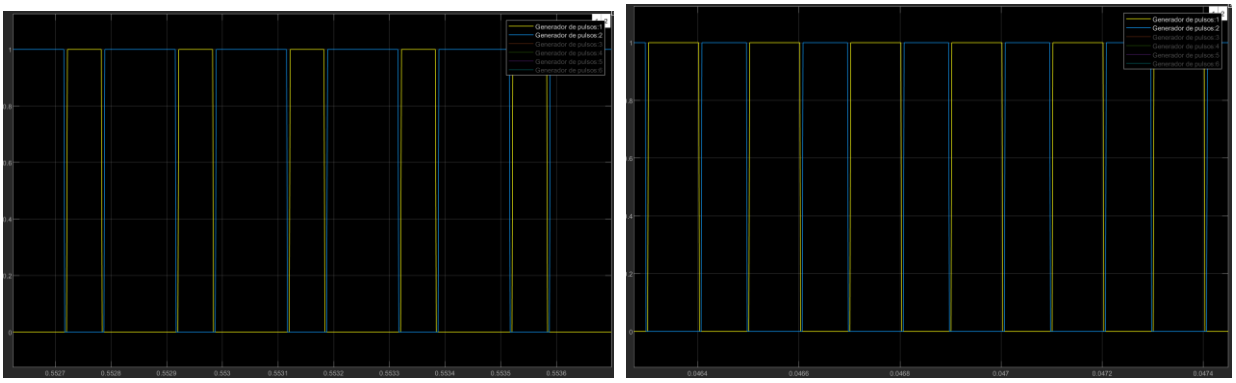
Para los PWM se observa cómo cuando se abre el IGBT A, el homólogo IGBT B estará cerrado, y viceversa, sin solaparse en ningún momento para evitar cortocircuito.

Tanto la moduladora como la velocidad siguen perfectamente la consigna, como se aprecia en las siguientes gráficas.

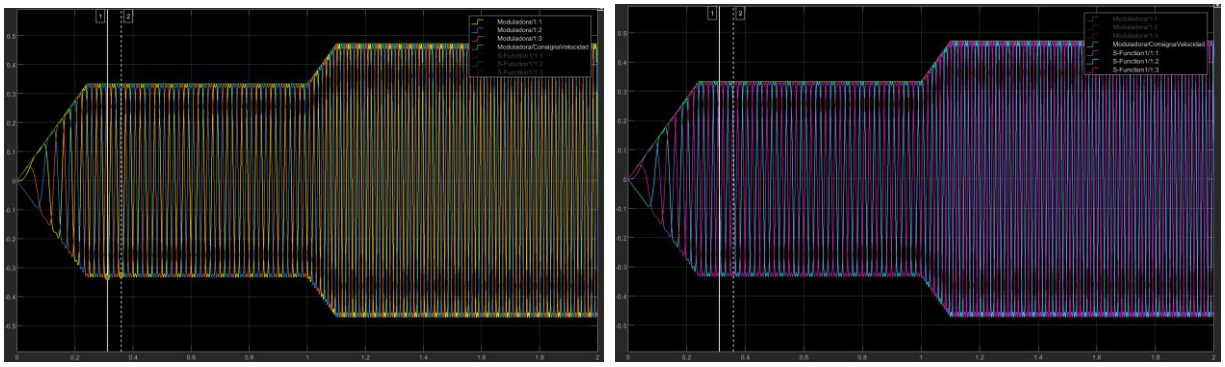
Corrientes filtradas y RMS [A] (Sin control PI; izquierda Potencia; derecha S-Function):



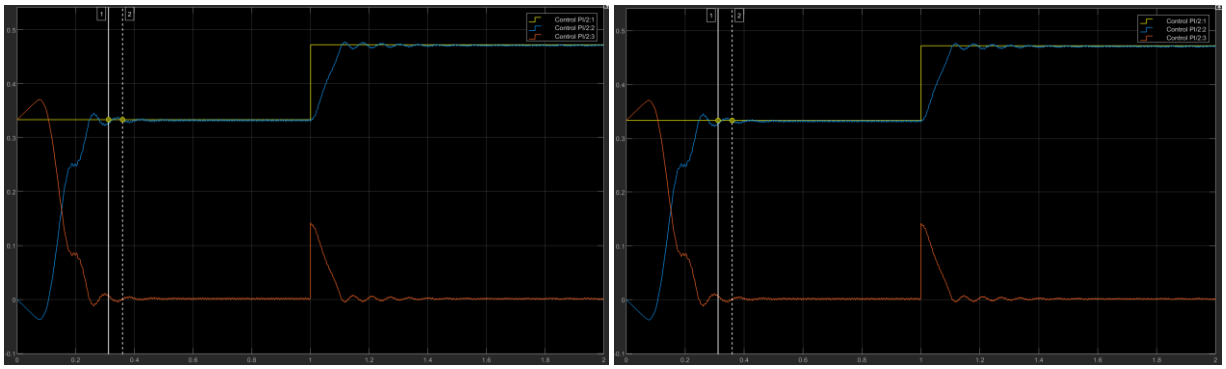
PWM1 A y B (Sin control PI; izquierda Potencia; derecha S-Function):



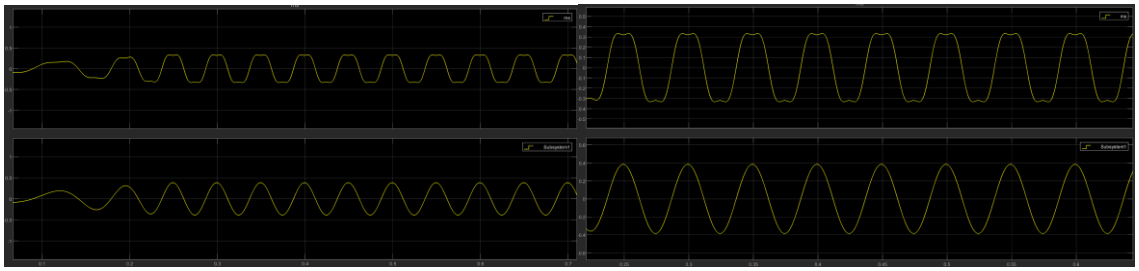
Moduladora y Consigna velocidad (Sin control PI; izquierda Potencia; derecha S-Function):



Velocidad real, consigna y error (Sin control PI; izquierda Potencia; derecha S-Function):



Moduladora antes y después de inyectar el tercer armónico (Sin control PI; izquierda Potencia; derecha S-Function):

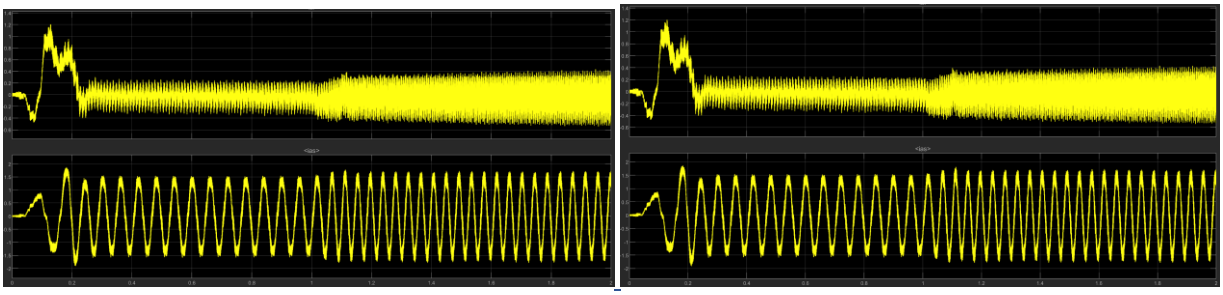


5.3 Ejercicio opcional (regulación PI de velocidad)

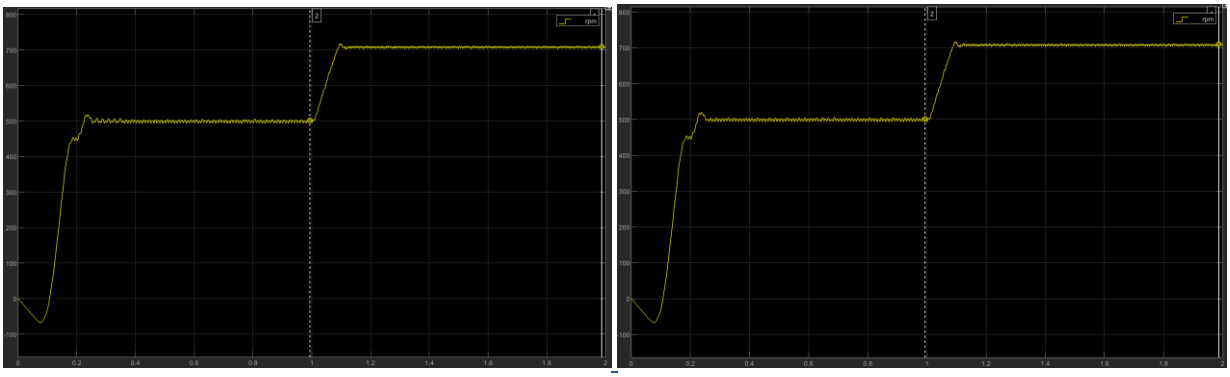
Antes de simular se han de fijar los siguientes valores de los parámetros de los switches:

PWMSelect = 0 y 1; *% <=0 para seleccionar Moduladora Simulink; >0 para seleccionar Moduladora S-Function.*
Control = 0; *% <= 0 para seleccionar Control PI; > 0 para seleccionar Control Step.*
Activar3Armonico = 1; *% <= 0 para desactivar 3er armónico; >0 para activar 3er armónico*

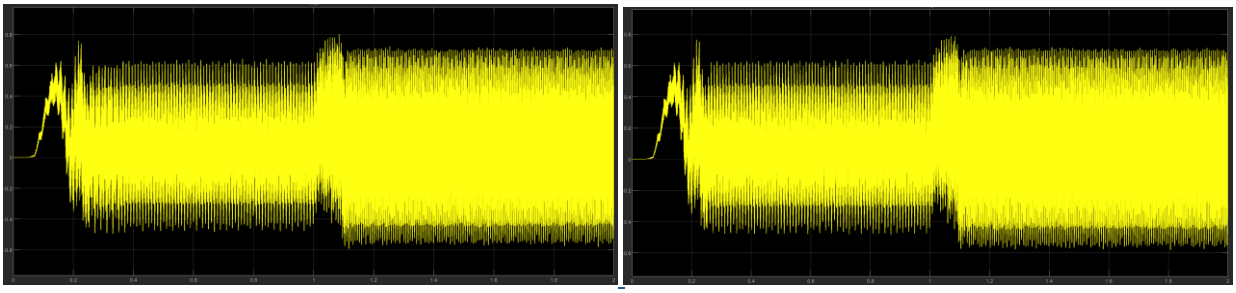
Corrientes de estator y Rotor [A] (Con control PI; izquierda Potencia; derecha S-Function):



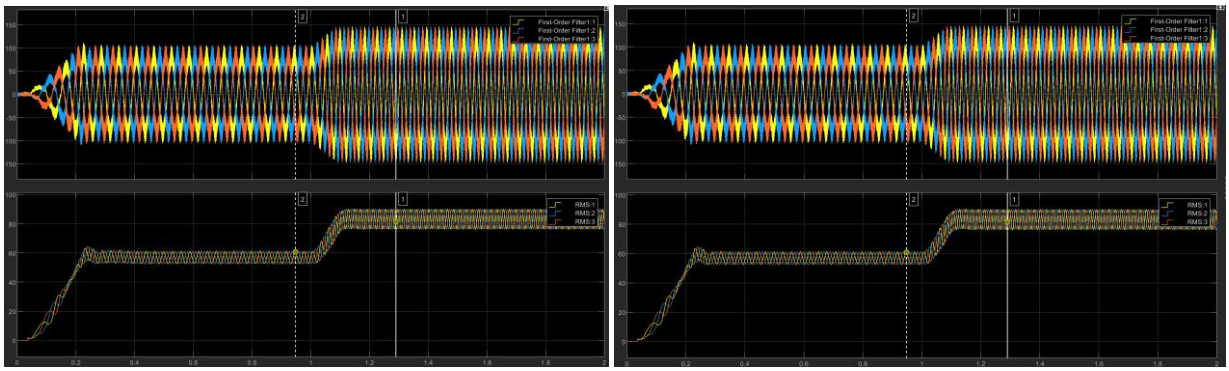
RPMs del motor [rpm] (Con control PI; izquierda Potencia; derecha S-Function):



Par motor [Nm] (Con control PI; izquierda Potencia; derecha S-Function):

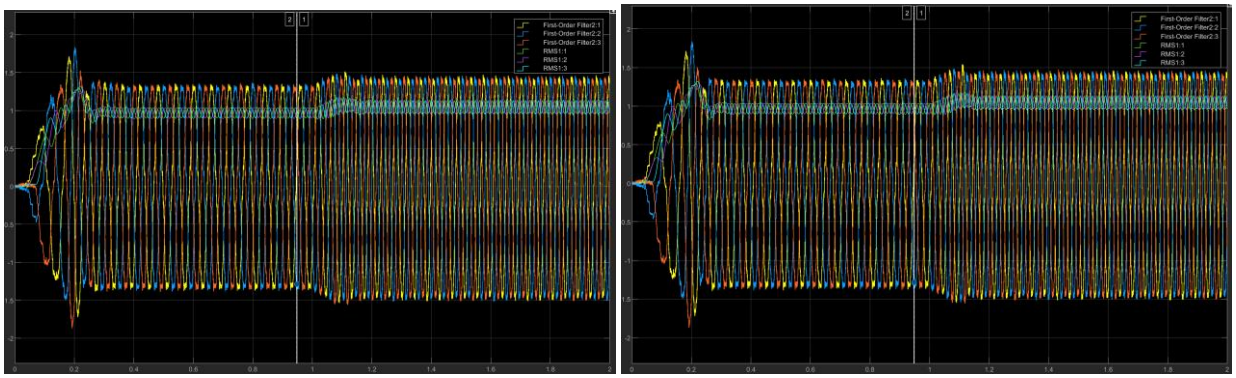


Tensiones fase-fase filtradas y RMS [V] (Sin control PI; izquierda Potencia; derecha S-Function):

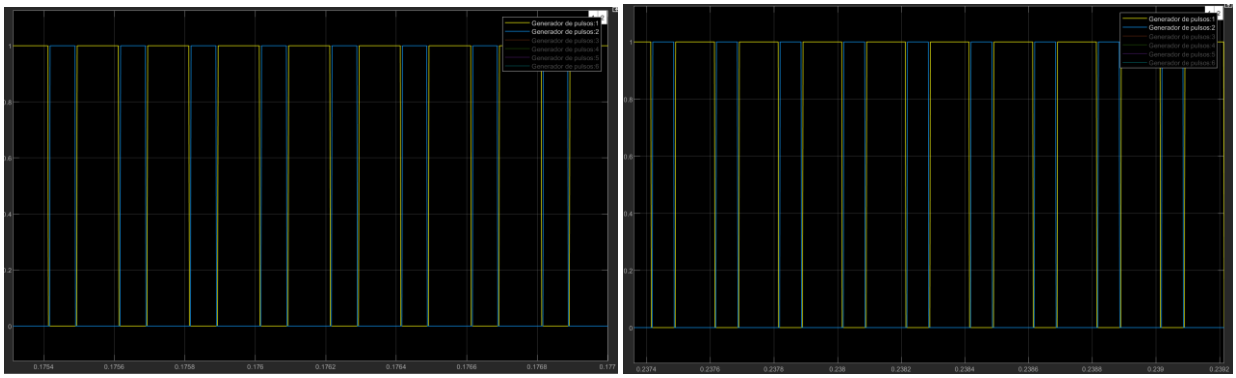


Con el regulador PI activo, el comportamiento es adecuado. Si bien, se detecta en la gráfica de regulación de velocidad que existe un offset con respecto a la consigna. Este es un punto que se debería de tratar de estudiar más en detalle, puesto que se cree que este comportamiento no es del todo correcto, pero se desconoce el origen de la desviación.

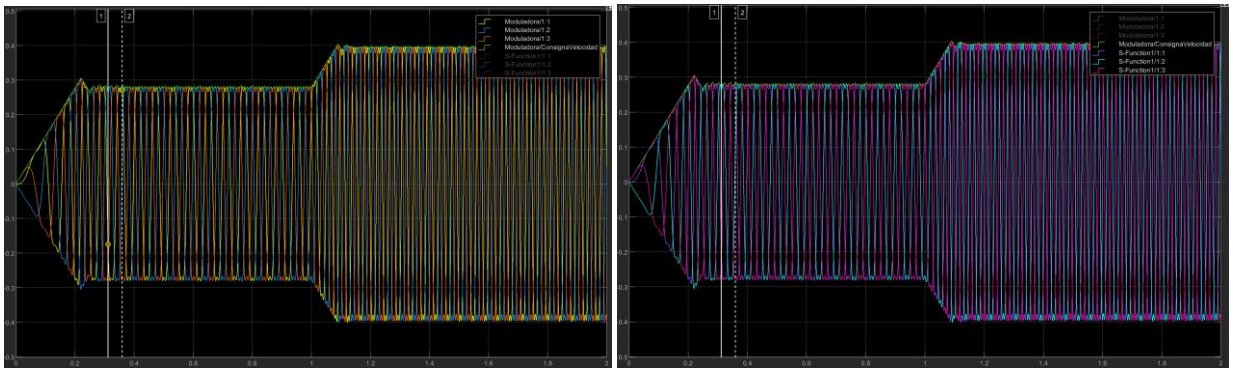
Corrientes filtradas y RMS [A] (Con control PI; izquierda Potencia; derecha S-Function):



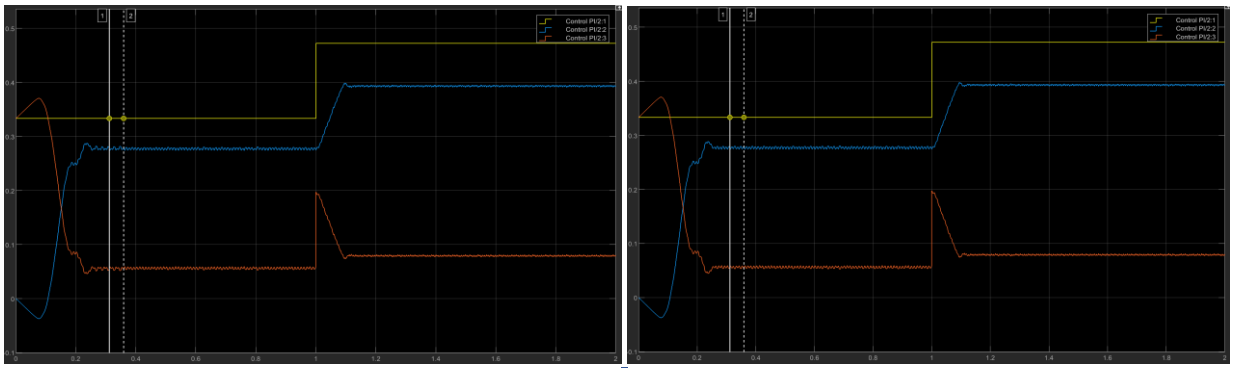
PWM1 A y B (Con control PI; izquierda Potencia; derecha S-Function):



Moduladora y Consigna velocidad (Con control PI; izquierda Potencia; derecha S-Function):



Velocidad real, consigna y error (Con control PI; izquierda Potencia; derecha S-Function):



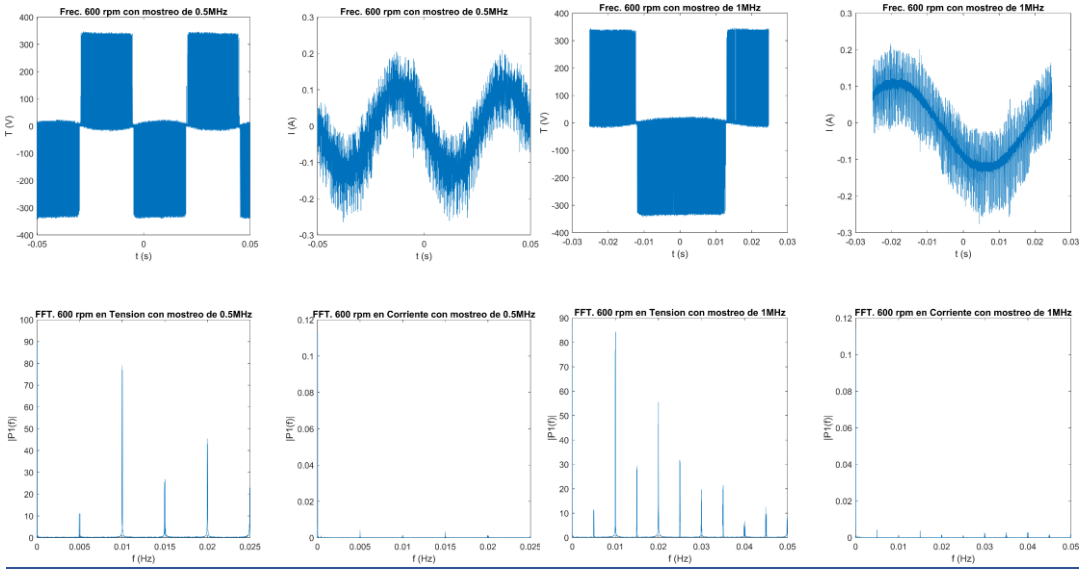
5.3 Comparación real y simulado

De los test reales, se han extraído ficheros con resolución de 1MHz de muestras y otro con resolución de 0.5MHz de la tensión y corriente, tanto para 600 RPM cuanto para 850 RPM.

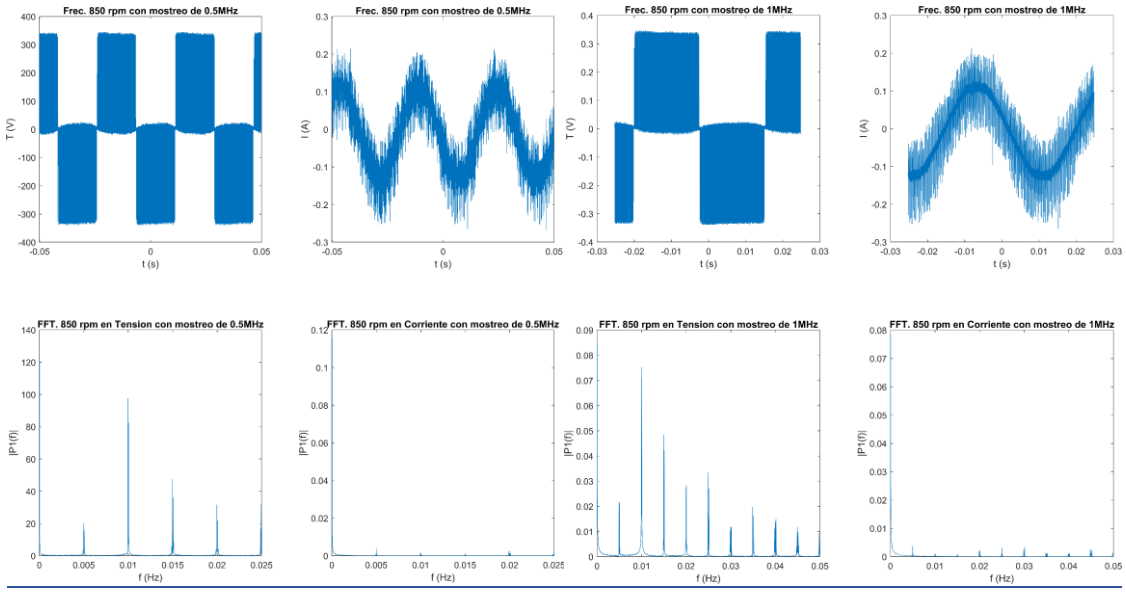
Como se aprecia en las gráficas abajo, en la extracción con 1 MHz, el valor de la fundamental y otras es un poco menor que la de 0.5 MHz. Además, como observado en las gráficas de la corriente y tensión en el tiempo, hay un ruido considerable en la medición, tanto de la corriente y tensión, por cuenta de los equipos de medición y características no ideales de los componentes del sistema.

Otro punto observado, son que las armónicas fueran identificadas como esperado en la tensión y por la corriente están filtradas por el motor que tiene característica predominante inductiva.

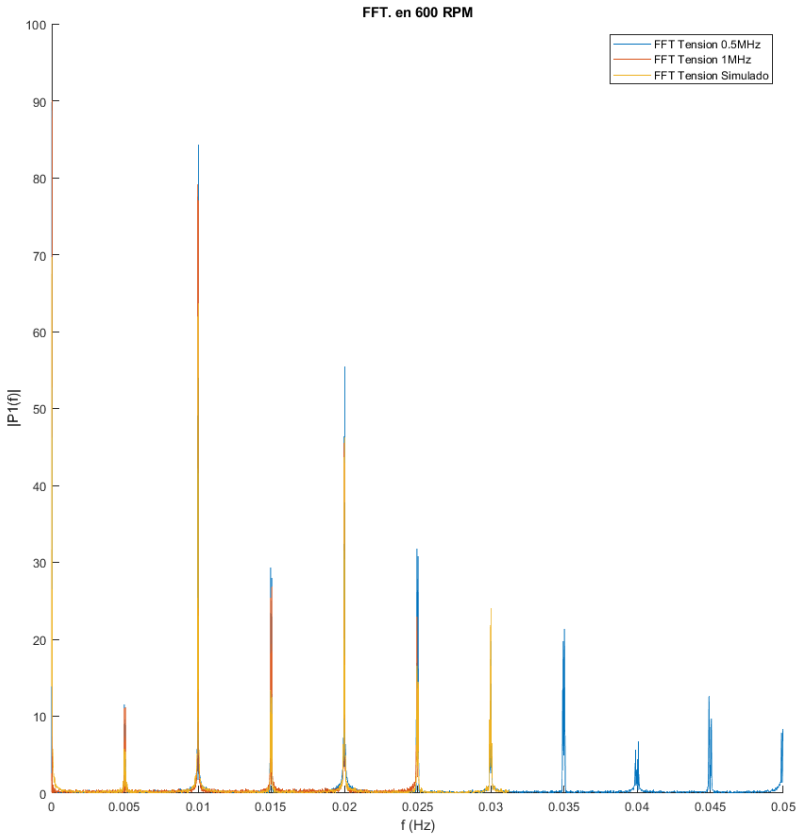
FFT y Señal de tensión y corriente real para 600 RPM con 0.5 MHz y 1 MHz de muestreo respectivamente

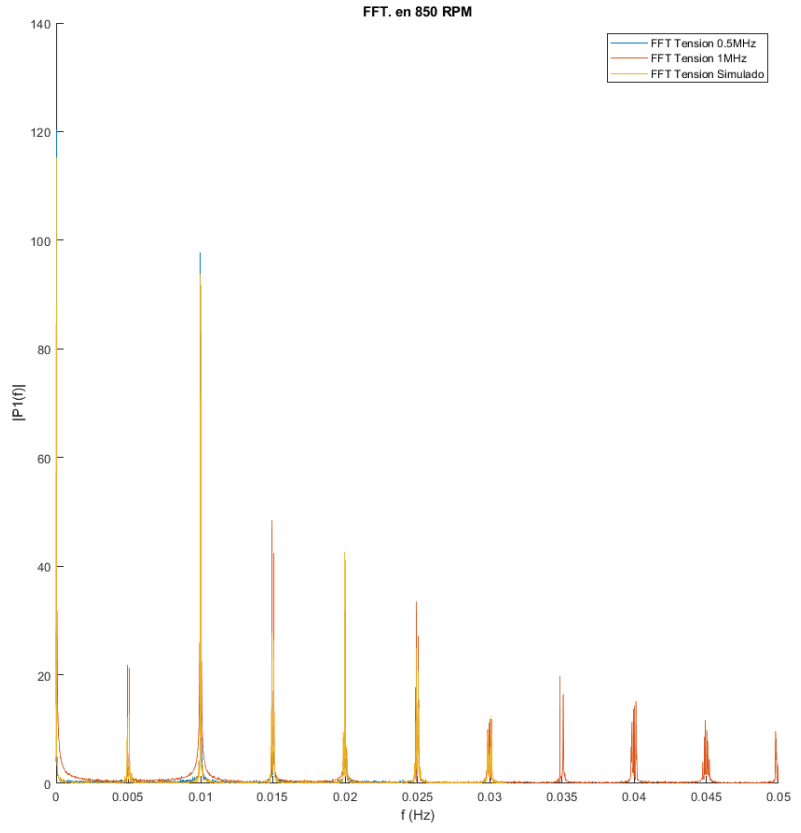


FFT y Señal de tensión y corriente real para 850 RPM con 0.5 MHz y 1 MHz de muestreo respectivamente



Por fin, las gráficas abajo son comparativas de la FFT de tensión entre 0.5 MHz, 1 MHz y la simulación, donde se comprobó los efectos de errores de medición que resultan en un pico menor para las tensiones reales medidas y valores mayor en las frecuencias acerca que la simulación.



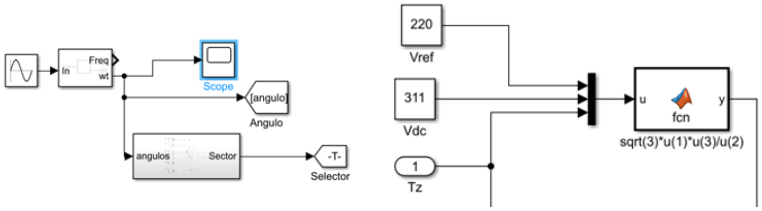


5.4 Ejercicio opcional (Modulación Vectorial)

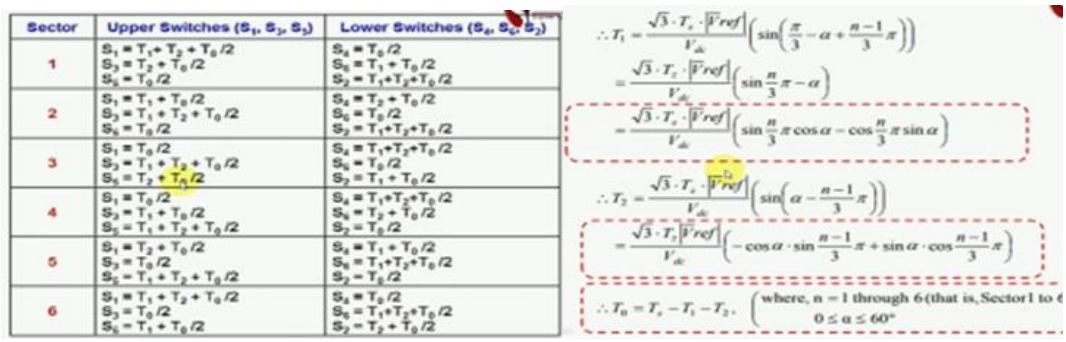
Para llevar a cabo la modulación vectorial hemos configurado el un Solver de tiempo variable en auto.

En primer hemos parametrizado un bloque que regenera una señal periódica que depende de la frecuencia que queramos que o motor gire y con una amplitud constante de 2pi, luego este valor nos sirve como un contador que va de 0 a 360º. El bloque siguiente te lo saca un pulso a cada 60º de esto señal y luego te regenera los 6 sectores para los vectores de espacios.

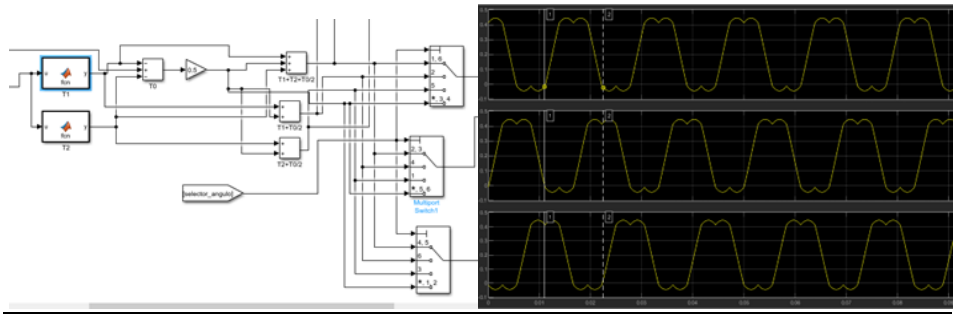
A continuación, hemos creado una función que regenera el vector voltaje referencia.



Mediante la tabla abajo, para todos los sectores hemos configurado 3 bloques de selectores que representan los estados de activación de los 3 interruptores de arriba de la rama de inversores mediante con tiempos determinados T0, T1 y T2.



Notase que en la salida de los 3 selectores se obtiene las señales periódicas ya con tercer armónico agregado en que sus amplitudes dependen de Tz (en nuestro caso es la consigna de tensión) y su periodo depende de la consiga en frecuencia. Estas son las señales que van a ser comparadas con la portadora que tiene la frecuencia de conmutación.



En aplicación tenemos como resultado en malla abierta lo gráfico en rpm de la rotación del motor, con mejor aspecto en comparación con la modulación no vectorial.