Control de Potencia Monofásica por SPWM

Aplicaciones en: Control de Potencia y la Industria

Jacobi Diego - Pfarher Iván
diego@jacobi.com.ar - ipfarher@gmail.com

Universidad Tecnológica Nacional Facultad Regional Paraná
Argentina – Entre Ríos - Paraná

keywords: spwm, control, potencia, monofásica, distorsión, mosfet, cruce por cero, fuente flotante.

Resumen

Esta aplicación en electrónica de potencia se basa en el control de potencia de la onda senoidal monofásica, por medio de la técnica SPWM, de forma tal que el ancho de pulso de la modulación PWM este ralacionado con la función senoidal. Esta técnica hace que la distorsión armónica se desplace a mayor frecuencia y por consiguiente los filtros se verán simplificados.

Introducción

Cuando surge la necesidad de variar una tensión alterna, con el objetivo de entregar mayor o menor potencia en una carga particular, es donde aparecen los controles de potencia monofásicos, con los cuales se logra recortar partes de la onda senoidal, variando la potencia entregada a la carga. Las técnicas convencionales empleadas, son por control de fase, estas generan armónicas cercanas a la armónica fundamental, lo cual hace que los filtros utilizados para eliminarlas sean complejos y poco económicos. Es por esto, que aparecen técnicas como la que se utiliza en este proyecto para que las primeras armonicas se vean desplazadas a frecuencias mas altas, lo cual hace que los filtros empleados para la eliminacion de las armonicas contaminanes sean de diseño mas simple.

SPWM

El funcionamiento básico de la modulación por ancho de pulso es simple, una serie de pulsos cuyo ancho es controlado por la variable de control. Es decir, que si la variable de control se mantiene constante o varía muy poco, entonces el ancho de los pulsos se mantendrá constante o variará muy poco respectivamente.

Si hacemos que el ancho de pulso no varíe linealmente con la variable de control, de modo que el ancho de los pulsos puede ser diferentes unos de otros, entonces sería posible seleccionar el ancho de los pulsos de forma que ciertas armónicas sean eliminadas.

Existen distintos métodos para variar el ancho de los pulsos. El más común y el que incentiva esta ponencia es la modulación senoidal del ancho de pulso (SPWM).

En el control PWM senoidal se generan los anchos de pulso al comparar un voltaje de referencia triangular de amplitud Ar y y de frecuencia fr con otro voltaje semisenoidal portador de amplitud variable Ac y de frecuencia 2 fs.

El voltaje semisenoidal de referencia está en fase con el voltaje de fase de entrada pero tiene 2 veces su frecuencia. La amplitud del voltaje semisenoidal de referencia controla el indice de modulacion M que varia entre 0 y 1, es decir, 0 a 100 %. El índice de modulacion se define como:

M = Amplitud de semisenoidal de referencia / Amplitud de triangular de referencia

En una modulación SPWM, el factor de desplazamiento es la unidad y el factor de potencia se mejora en gran medida respecto del PWM normal.

Las armónicas de menor orden se eliminan o se reducen. Por ejemplo, con 4 pulsos por medio ciclo, la armónica de orden mas bajo es la quinta y con seis pulsos por medio ciclo, la armónica de orden mas bajo es la séptima.

En esta implementación hacemos 39 pulsos por medio ciclo lo que nos da una armónica de menor orden de 4KHz aproximadamente

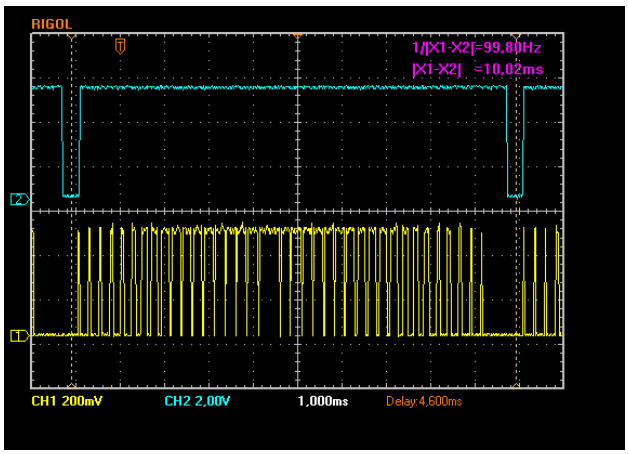

Ilustración 1: Señal SPWM y de cruce por cero generada por el microcontrolador

La señal senoidal de referencia debe ser pura, es decir que si el propósito es el control de potencia, y estamos hablando de una frecuencia de linea de 50 Hz, deberemos generar un senoide puro de 50 Hz sin obtenerlo de la misma linea.

Esto es debido a que la tensión de linea se puede encontrar con mucha distorsión armónica y generaría errores en la presición del SPWM los cuales serían aumentados por el efecto de aliasing en el ADC.

Implementación

Existen varios métodos posibles para la implementación de SPWM.

El desarrollador podría pensar que sería mas sencillo y eficiente utilizar amplificadores operacionales comparando entre la señal senoidal de referencia y la triangular. Pero, si bien este tipo de implementación es perfectamente posible, se presentan otros problemas. Generar una señal senoidal pura y coerente de 50 Hz no es tarea fácil, de la misma forma, tampoco lo es generar una señal triangular. Además, ambas señales deben ser capaces de ponerse en fase con la tensión de linea de 50 Hz lo cual presenta el desafío mas importante.

La señal senoidal de referencia debe ser entonces coerente y en fase con la tensión de linea y no ser obtenida a partir de esta ultima.

La señal triangular debe ser tan perfecta como sea posible y su frecuencia tan alta como se necesite desplazar los armónicos indeseados. Adicionalmente, como el SPWM debe estar en fase con el período de la tensión de línea, la frecuencia de la señal triangular deberá ser un multiplo entero de la de línea.

Entonces, por ejemplo, la tensión de línea es de 50 Hz, la señal senoidal de referencia debe ser de 50 Hz y la triangular de 50*k Hz.

Es por estas razones que resulta evidente que las técnicas digitales pueden realizar este trabajo sin repasar los mismos problemas.

En este proyecto se implementó ésta técnica haciendo uso de un microcontrolador AVR.

La señal senoidal es un conjunto de valores previamente calculados y guardados en la memoria interna del microcontrolador por razones de mayores prestaciones, ya que siendo este un dispositivo de 8 bits, el cálculo del valor instantáneo del seno es demaciado intenso para el procesador.

Los valores utilizados corresponden a cuarto de ciclo de senoide y con un contador se selecciona el valor correspondiente incrementando y luego decrementando, conformando asi medio ciclo del senoide.

La frecuencia interna de la referencia es entonces un semiseno pulsante de 100 Hz, ya que al realizar las comparaciones con la señal triangular elegimos realizarlo siempre con valores positivos.

La señal triangular es realizada haciendo uso del PWM interno del microcontrolador.

La sincronización se realiza usando una interrupción que disparará un semiciclo de SPWM cada vez que se detecte un paso por cero en la tensión de línea.

Estos detalles nos permiten que el microcontrolador esté en su mayor parte del tiempo inactivo, solo despertando en cada interrupción (cada 1/100 [s]) y durante cada interrupción de contador, de modo que el circuito de control consumirá muy baja potencia, siendo despreciable para cualquier propósito.

Diagrama en Bloques

El esquema que se presenta es de configuración sencilla y segura. El microcontrolador se encuentra aislado de la etapa 1, la cual detecta los cruces por cero de la señal de la red eléctrica monofásica de distribución de energía.

El control de disparo de la etapa de potencia se encuentra aislada por la etapa 2, la cual esta formada por una fuente flotante.

La alimentación del microcontrolador, como de las etapas adyacentes se realiza por una fuente de alimentación externa de 12 y 5V.

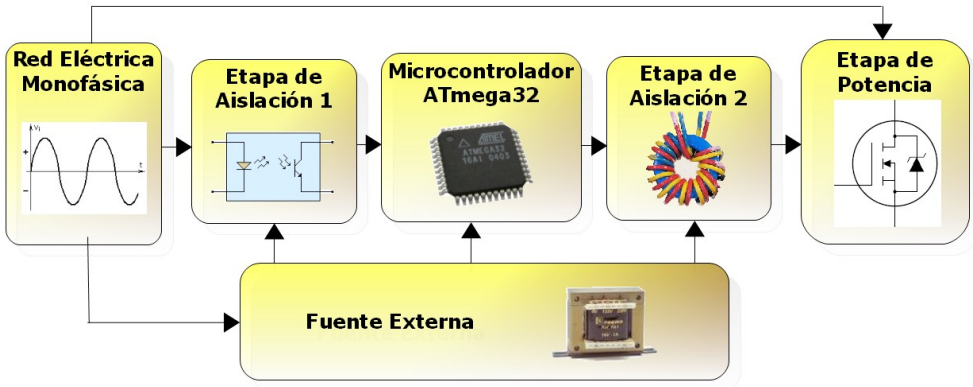

Ilustración 2: Diagrama en Bloques General

Circuito

El circuito del microcontrolador Atmega 32 posee pocos componentes asciados y pines utilizados, entre ellos se encuentran: un preset conectado al canal cero del ADC con el cual se varia la señal SPWM, el circuito oscilador externo compuesto por el cristal de 16MHz y sus

capacitores, el circuito de reset, la entrada de detección del cruce por cero y la alimentacion del mismo.

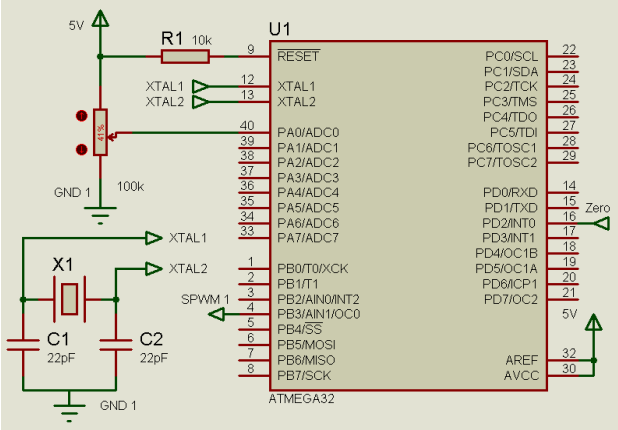

Ilustración 3: Microcontrolador Atmega32

El circuito de detección de cruce por cero esta compuesto por un puente rectificador, el cual entrega la onda rectificada al diodo emisor del optocoplador, este conmuta de estado ON a OFF en cada cruce por cero de la onda senoidal de 50Hz. La tensión del colector del transistor de salida del optoacoplador dispara el transistor driver obteniendose un pulso cada 0,01 seg. en el colector. Esta es la señal de entrada al microcontrolador en el pin de interrupcion cero.

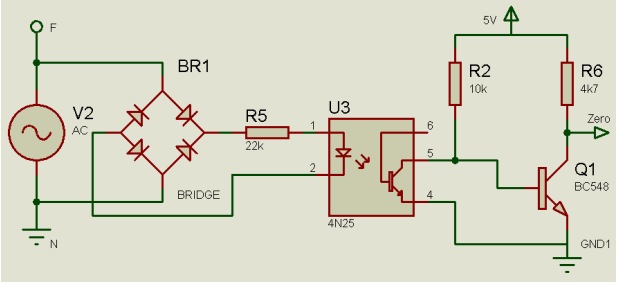

Ilustración 4: Etapa de aislacion 1: Detección de cruce por cero de la señal senoidal de 50Hz

Para tener aislación entre la etapa lógica y la etapa de potencia, mas presisamente la etapa de disparo del MOSFET, se diseño una fuente flotante.

La fuente flotante se compone de dos transistores driver 2n3904, los cuales se encargan de conmutar el primario del transformador toroidal con la señal portadora de 50KHz modulada por la señal SPWM.

Los pulsos de alta frecuencia del secundario son filtrados, teniendo de esta forma la señal SPWM 2 que dispara al MOSFET en la etapa de potencia.

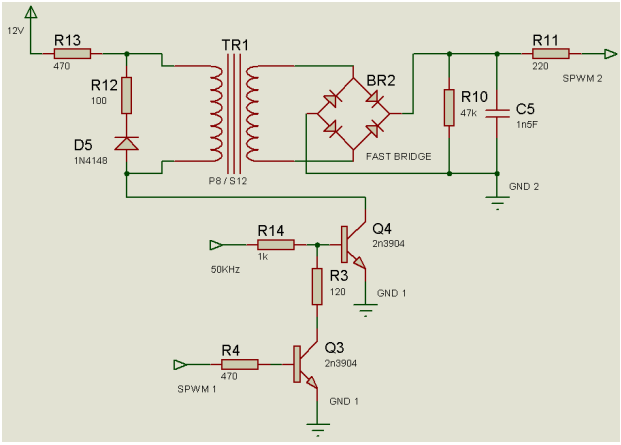

Ilustración 5: Etapa de aislación 2: Fuente flotante por transformador toroidal

La etapa de potencia se basa en una configuración de intrruptor de corriente alterna monofásico con puente rectificador. Esta configuración permite conmutar cargas de CA con dispositivos de CC.

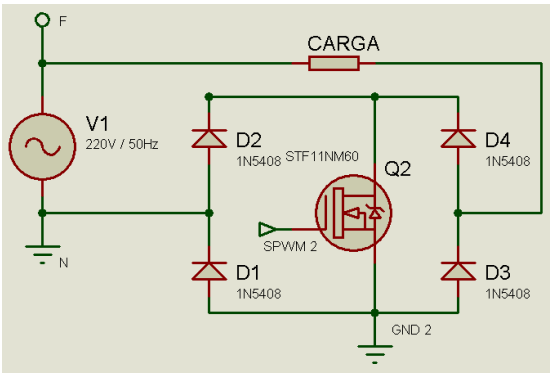

Ilustración 6: Etapa de Potencia: Interruptor de CA monofásico con puente rectificador

En las siguientes dos gráficas se puede observar los resultados obtenidos respecto a la conmutación del Mosfet y la caida de tensión en la carga.

Ademas, se puede ver que la primer componente que cobra importancia en el espectro en frecuencia se sitúa aproximadamente en los 4KHz.

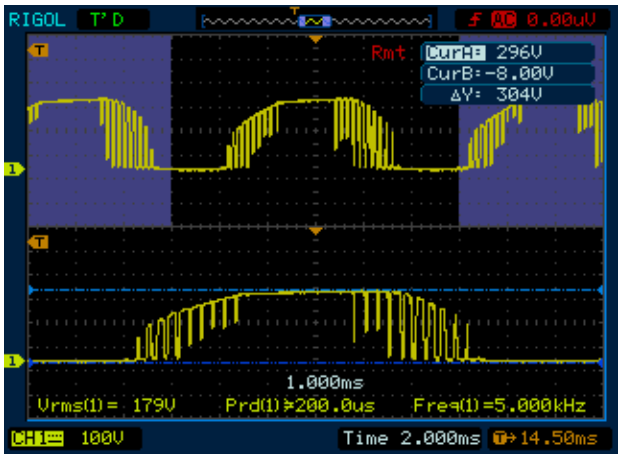

Ilustración 7: Señal de conmutación del MOSFET sobre una carga de 75W

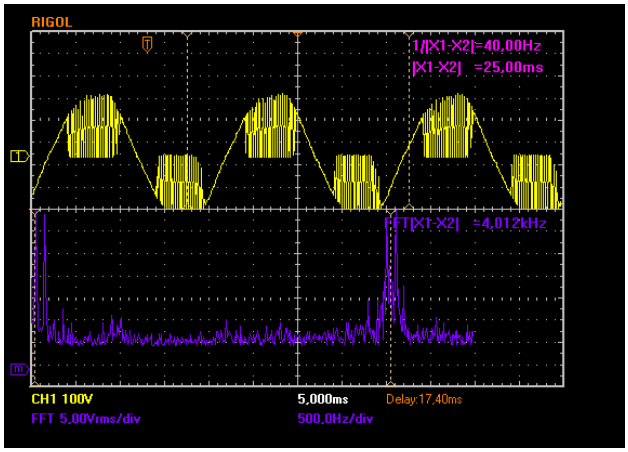

Ilustración 8: Espectro en frecuencia: armónica fundamental en 50Hz y primer armónica en 4KHz

Firmware

```
/* Sources:
** http://www.nongnu.org/avr-libc/user-manual/modules.html
**
** Burn with:
** avrdude -c usbasp -p atmega32 -U flash:w:main.hex -U eeprom:w:main.eep
*/

#include "main.h"

// La amplitud del senoide es asignada por un canal de ADC
//      esta controla la potencia total entregada a la carga.
uchar amplitud_seno;

// Esta funcion iniciará el contador interno como PWM del microcontrolador,
//    configurandolo de tal forma que obtengamos una buena frecuencia para empujar las armónicas
//    distorsionantes a mayores frecuencias pero a su vez lo suficientemente baja para que el
//    microcontrolador no se vea muy exigido y tenga tan poco consumo electrico como sea posible.

inline void triangularSTART(void)

{
   TCCR0 = BIT( WGM00 ) | BIT( COM01 ) | BIT( CS01 );
   // PWM Phase Correct (triangular)
   // Prescaller de 8 da 39 subidas y 39 bajadas.
   // El Pin con OC0 debe estar configurado como salida.
   OCR0 = 0;
   TCNT0 = 0xFF;     // El contador empezará decrementando.


   TCCR1A = BIT(COM1A0); //Toogle
   TCCR1B = BIT(WGM12) | BIT(CS10); //WGM13:0=4 => CTC, CS12:0=1 => prescaler=1
   OCR1A = 159; //para generar 50KHz
}


inline void triangularSTOP(void)

{

// Apaga el contador interno que controla el PWM
   TCCR0 = 0;

}


inline void spwmSTART(void)

{

// Iniciara el Sinusoidal PWM, incluyendo desde el contador correspondiente a la señal triangular
//     hasta el contador correspondiente al senoide.
// Este ultimo contador tiene una frecuencia de actualización mucho menor, ya que no requiere terminar su
//     ciclo tantas veces como la triangular.
   triangularSTART();

   TCCR2 = BIT( CS21 ) | BIT( WGM21 );
   TCNT2 = 0;
   OCR2 = 78;   // 39 subidas mas 39 bajadas
   SBI( TIMSK , OCIE2 );

   // Mostrar inicio y fin de un periodo senoidal en pin PB0
   SBI( PORTB, PB0 );
}

inline void spwmSTOP(void)

{

// Detenemos toda la generacion del SPWM.
// Esta funcion es llamada al terminar el PWM luego de cada semiperiodo de 50 Hz.
// Casi inmediatamente despues, el SPWM es vuelto a iniciar por la siguiente interrupción.
   TCCR2 = 0;
   CBI( TIMSK , OCIE2 );
   triangularSTOP();

   CBI( PORTB, PB0 );
}
```

```
// El senoide de referencia es un conjunto de valores calculados en un software de hoja de calculo y
//     formateada en forma de arreglo que será almacenado en la flash o memoria de programa del
//     microcontrolador.

// Este método es necesario ya que es demaciado costoso para un microcontrolador calcular el valor
//     instantáneo de un seno.
// Se pueden ahorrar un par de instrucciones mas copiando este vector de 126 elementos a memoria
//     durante el inicio del microcontrolador, en vez de acceder a la memoria flash cada vez que necesita
//     ser leido.
// Cambiando los valores de este vector podemos realizar distintos tipo de controles, como por ejemplo
//     utilizar un seno al cuadrado como señal de referencia, haciendo que el SPWM concentre su mayor
//     operación en la zona de mayor tensión de la tensión de linea.
#define NUM_SAMPLES 126
static const uchar Seno[NUM_SAMPLES] PROGMEM = {
 0,  3,  6, 10, 13, 16, 19, 22, 26, 29, 32, 35, 38, 41, 45, 48,
51, 54, 57, 60, 63, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94,
97,
100, 103, 106, 109, 111, 114, 117, 120, 123, 126, 128,
131, 134,
137, 139, 142, 145, 147, 150, 152, 155, 158, 160, 163,
165, 167,
170, 172, 175, 177, 179, 181, 184, 186, 188, 190, 192,
194, 196,
199, 201, 202, 204, 206, 208, 210, 212, 214, 215, 217,
219, 220,
222, 223, 225, 226, 228, 229, 231, 232, 233, 235, 236,
237, 238,
239, 240, 242, 243, 243, 244, 245, 246, 247, 248, 249,
249, 250,
250, 251, 252, 252, 253, 253, 253, 254, 254, 254, 254,
255, 255,
255, 255, 255 };

// La interrupcion del timer 2 controla el valor instantaneo del seno de referencia que esta siendo usado en
//     cada comparación con la triangular.
ISR(TIMER2_COMP_vect)

{
   static uchar k = 1;
   static char inc = 1;


      UInteger senov;
// La constante k se incremente hasta el ultimo elemento del vector y luego se decrementa hasta su inicio.
// De esta forma se realiza por simetria un semiciclo completo de seno, ahorrando otros 125 valores
//     en el vector.
      k+=inc;

      if (k == 125)
        inc = -1;
      if (k == 0)
         {
         k = 1;
         inc = 1;

         spwmSTOP();

// Al terminar el semiciclo completo el SPWM se apaga temporalmente (o reinicia).
         }

// La variable "senov" es una union de 16 bits de longitud de dos partes. Un entero que en la familia AVR es
//    de 16 bits, y un array de 2 caracteres de 8 bits. Ambos tipos de datos ocupan el mismo espacio de
//    memoria pero son tratados diferente por el compilador.
// Usamos este método para poder realizar una multiplicacion de 8 bits por 8 bits que dará como resultado
//    un valor maximo de 16 bits y luego seleccionar unicamente la parte alta (ultimos 8 bits) del resultado.
// Con esta operacion que puede parecer complicada, evitamos utilizar variables de tipo "float" en el vector
//    de los valores seno. Este tipo de variable es de 32 bits (4 veces lo que estamos ocupando) y su manejo
//    en un procesador de 8 bits es muy costoso, y debe evitarse a toda costa.
// Primero inicializamos senov leyendo el valor del seno desde la memoria de programa:
      senov.c[0]=pgm_read_byte((PGM_P)&Seno[k]);

      senov.c[1]=0;
      senov.i = amplitud_seno*senov.i;

      OCR0 = senov.c[1];  // Es mucho menos costoso en tiempo que usar floats
// Un problema con esta implementación, es que el OCR0 tiene un doble buffer y no se actualiza al instante,

//     se actualiza recien al terminar de contar (alcanzar su maximo o minimo), lo cual crea un defasaje entre
//     el senoide y la pwm. Por eso empezamos con k=1, para que el primer valor del seno no sea cero.
// Como pseudosolución, se puede por hardware activar la interrupción de cruce por cero ligeramente
//     antes (unos pocos nanosegundos medidos visualmente con osciloscopio) con un ancho de pulso
//     ajustable, de form que si el cruce por cero exacto es el centro del pulso de interrupción, entonces el
//     flanco descendente será unos nanosegundos antes, permitiendo a los contadores iniciar nanosegundos
//     antes.
}

ISR(ADC_vect)
{
// El ADC controla la amplitud del seno para poder ajustar la potencia de salida.
// Si necesitaramos no solo variar la potencia si no tambien regularla, deberíamos usar otro canal del ADC
//     para obtener el estado de la salida y detener o no la funcion triangular.
   amplitud_seno = ADCH;
}

ISR(INT0_vect, ISR_NOBLOCK)
{
// La interrupción externa INT0 debe ser activada por un detector de cruce por cero por flanco
```

```
//     descendente. Esta hace que se dispare una ciclo
completo de PWM en forma sincronizada con la
//     tensión de linea, resolviendo de esta forma los
problemas que tendríamos al generar las señales de
//     referencia en forma analógica.

// Detener el PWM no debería ser necesario ya que este
se auto termina al finalizar el ciclo, el cual termina
//     antes de que ocurra la siguiente interrupción, pero su
inclusión no perjudica y nos asegura que ante un
//     error, las variables serán reiniciadas.
    spwmSTOP();
    spwmSTART();
}

ISR(BADISR_vect)
{
// Interrupcion por defecto ante una interrupcion
desconocida.
// Si no se define esto, el micro se resetea ante un error.
// Por supuesto, un error nunca debería ocurrir.
}

static void hardwareInit(void)
{
    DDRB = BIT( SPWMBIT ) | BIT( PB0 );
    PORTB = 0;

    DDRD = BIT( PD5 ); // PD5 como salida  de 50KHz


// Podemos usar el puerto C para debugear en forma
sencilla.
    DDRC = 0xFF;          // PortC: outputs
    PORTC = 0x00;         // Leds apagados


// Habilitar interrupciones:
    SBI( GICR , INT0 );
    MCUCR |= _BV( ISC01 );      // INT0 por flanco de
bajada
// Usar ADC0 en Free Running, pin AREF como
referencia, Ordenar a la izquierda para 8 bits en ADCH
    SFIOR = 0;

    ADMUX = BIT( ADLAR );
    ADCSRA = BIT( ADEN ) | BIT( ADSC ) | BIT( ADATE ) |
BIT( ADIE ) | BIT( ADPS2 ) | BIT( ADPS1 ) |
BIT( ADPS0 );     // ADC Enable, AutoTrigger, Enable
interrupt

    amplitud_seno = 255;
}

//=========================================
=====================
//====================   main
=====================================
//=========================================
=====================
int    main(void)
{

  //wdt_enable(WDTO_1S);

    // Even if you don't use the watchdog, turn it off here.
On newer devices,
    // the status of the watchdog (on/off, period) is
PRESERVED OVER RESET!

    _delay_loop_2(10000);
    // Delay para eliminar cualquier error que pueda
generarse en el pin de interrupcion
    //     durante el encendido de la placa, y dispare todo un
ciclo de pwm.

  hardwareInit();
  sei();   // Activar interrupciones
  for(;;)
    {
       }
    return 0;

}

/* ------------------------------------------------------------------------
*/
```

Bibliografía

"Electrónica de Potencia" Muhammad H. Rashid, 2º Edicion, Prentice Hall

Manual online de programacion de AVR
http://www.nongnu.org/avr-libc/user-manual/

ATMEL ATmega32's Datasheet
http://www.atmel.com

Softwares
AVR Studio 4
Proteus