Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo Fin de Grado

Sistema de control de convertidores CC-CC basado en microprocesador y reconfigurable desde un ordenador

Autor: Adrián Gómez Tur

Tutor: Antonio Lázaro Blanco

Índice general

Índice de Figuras

Índice de tablas

1. Introducción

1.1. Introducción

En este apartado se incluirán los objetivos y motivaciones del proyecto "Sistema de control de convertidores CC-CC basado en microprocesador y reconfigurable desde un ordenador", además de la estructura de la documentación.

Si tuviéramos que clasificar este proyecto, estaría en el campo de la electrónica de potencia, pero ¿qué es la electrónica de potencia? Según [1] la definición de electrónica de potencia es: "aquella rama de la electrónica encargada del estudio de los circuitos electrónicos destinados al manejo de flujos de energía eléctrica".

La historia de la electrónica de potencia comienza en 1900, con la invención del rectificador de arco de mercurio. Tras este se introdujeron: el rectificador de tanque metálico, el de tubo al vacío controlado por rejilla, el ignitrón, el fanotrón y el tiratrón. Estos dispositivos se usaron para el control de potencia, hasta aproximadamente 1950.

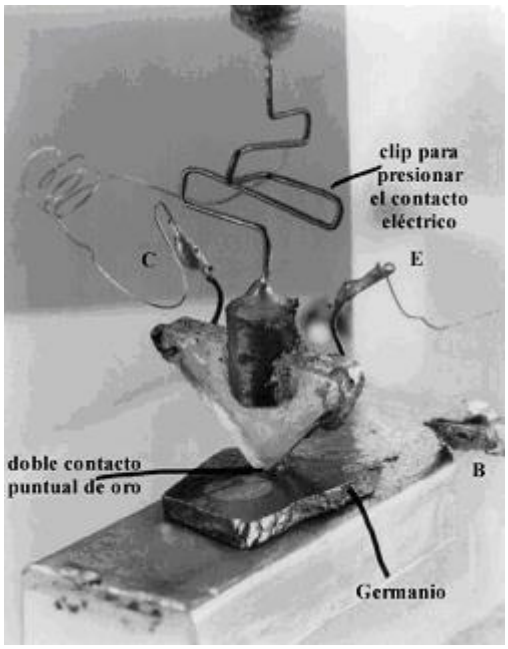


Figura 1.1-1 Fotografía del primer transistor

Podría decirse que la primera "revolución electrónica" comenzó en 1948, año en el que de la mano de Bardeen, Brattain y Schockley se inventó el transistor de silicio en los Bell Telephone Laboratories. En 1956 se inventó el siguiente adelanto en lo que a electrónica de potencia se refiere, el transistor de disparo PNPN, conocido como tiristor o rectificador controlado de silicio (SCR en inglés), también en los Bell Laboratories. La segunda "revolución electrónica" comenzó en 1958, año en el que se desarrolló el tiristor comercial, por la empresa General Electric Company. Desde ese momento hasta la actualidad, la electrónica de potencia ha ido aumentando su presencia en nuestras vidas.

1.1. Motivación

Hoy en día diferenciamos principalmente entre dos tipos de controles: analógico y digital. En este proyecto se profundiza en el control digital, ya que en los últimos años está cobrando más protagonismo, debido al avance que se está produciendo en su velocidad de trabajo, tamaño, coste, etc. El control digital ofrece numerosas ventajas, como por ejemplo incluir algoritmos complejos en un espacio físico reducido, o la comunicación con ordenadores, muy útil para detectar anomalías o transmitir datos de interés.

A pesar de las ventajas anteriormente enumeradas el control digital presenta a su vez algunos inconvenientes, como el retraso que se produce al muestrear, no debemos olvidar

tampoco que al ser un control digital trabajaremos con muestras discretas, por lo que en este aspecto el control analógico es más ventajoso.

La competencia en el sector electrónico, sumado a los avances es la industria de los semiconductores han provocado que los precios de los microcontroladores disminuyan, además la dificultad para trabajar con estos también ha disminuido, ya que no es necesario programar en lenguaje de bajo nivel (ensamblador), puede programarse en lenguaje de alto nivel como puede ser C.

Por estas razones se ha decidido que este TFG se realice el diseño de un control digital mediante un microcontrolador.

2. Control y comunicaciones

2.1. Introducción

En este capítulo se definirán las configuraciones de los diferentes dispositivos de control necesarios para implementar el control digital al convertidor CC-CC.

Con el paso del tiempo y dadas las mejoras en velocidades y consumos, el control digital cada vez va tomando mayor protagonismo en la electrónica de potencia. Por ello es el tipo de control elegido para realizar este TFG.

Para el control se empleará un microcontrolador STM32F4 (cortex M4) con arquitectura ARM y cuya plataforma de desarrollo es Discovery. A continuación se incluirá una introducción histórica sobre los microprocesadores, para finalmente mostrar la configuración de los periféricos que usaremos en el proyecto.

2.2. Historia de los microcontroladores

En 1965, el cofundador de Intel, el Dr. Gordon Moore formuló una predicción, que se conoce como la "Ley de Moore", esta ley está representada en la Figura 2.2-1, dice que el número de transistores en un chip se duplicaría cada dieciocho meses. Esta afirmación, que se ha cumplido durante los últimos treinta años, en principio estaba destinada a los dispositivos de memoria, pero también los microprocesadores la han cumplido. De esta forma los usuarios pueden disponer de mejores equipos cada relativamente poco tiempo, pero implica también la parte de negativa derivada del gasto que se debe realizar para renovar dichos equipos, se estima que esta ley seguirá vigente durante los próximos quince o veinte años.

Figura 2.2-1 Ley de Moore

El 15 de noviembre de 1971 aparece el primer microprocesador de la mano de Intel, el Intel 4004, de 4 bits que fue el primer microprocesador en un solo chip y contenía 2.300 transistores y realizaba sesenta mil operaciones por segundo. El 4004 lo diseño e implementó Federico Faggin entre 170 y 1971, Faggin creó una metodología nueva de "random logic design" con Silicon Gate, que no existía previamente, la cual se utilizó para poder encajar el microprocesador en un solo chip.

Al año siguiente, el 1 de Abril de 1972, Intel anunciaba una versión mejorada del 4004, lo llamaron 8008 y su principal ventaja era que procesaba a 8 bits, y su velocidad de reloj alcanzaba los 740 kHz. Fue el primer microprocesador de 8 bits, implantaba la tecnología PMOS, contaba con 48 instrucciones y podía ejecutar 300 mil operaciones por segundo, y direccionaba 16 Kbytes de memoria.

En 1974 National Semiconductor desarrolló el SC/MP, su nombre es un acrónimo de Simple Cost-effective Micro Processor y era popularmente conocido como "Scamp". Presentaba un bus de direcciones de 16 bits y un bus de datos de 8 bits, además contaba con una característica avanzada para su tiempo, era capaz de liberar los buses para que pudieran ser compartidos por varios procesadores. Este microprocesador fue muy utilizado, dado su bajo precio, y que provisto con kits era utilizado para propósitos educativos, de investigación, etc.

En Abril de 1974, Intel lanza el 8080, con una velocidad de reloj de 2 MHz. Un año después aparece el primer ordenador personal, cuyo nombre el Altair, que estaba basado en la microarquitectura del Intel 8080.

Motorola lanza al mercado en 1975 el Motorola MC6800. Era más conocido como 6800, y tomaba su nombre de que contenía aproximada mente 6.800 transistores. Algunas de las primeras microcomputadoras de los años 1970 lo usaron como procesador.

Federico Faggin, quien fuera diseñador jefe del Intel 4004 fundó en 1974 la compañía Zilog Inc., quien en 1976 crea el Zilog Z80. El Zilog Z80 es un microprocesador de 8 bits construido con tecnología NMOS. Un año después sale al mercado el primer ordenador que hace uso del Z80, el Tandy TRS-80 Model 1. Es uno de los procesadores de más éxito, se han producido numerosas versiones clónicas, y en la actualidad se sigue usando en multitud de sistemas embebidos.

En 1982 Intel lanzó el 80286, popularmente 286, fue el primer procesador de Intel capaz de ejecutar el software escrito para su predecesor. Desde entonces la compatibilidad del software es un sello de la familia de microprocesadores Intel. Tras seis años de su introducción un estudio estimó que había 15 millones de PC's en todo el mundo basadas en el 286.

Tres años más tarde Intel lanza el 80386, conocido como 386, estaba integrado por 275.000 transistores, más de 100 veces más que el 4004. El 386 tenía una arquitectura de 32 bits, tenía capacidad para multitarea y una unidad de traslación de páginas. Estas características hicieron más sencillo la implementación de sistemas operativos que usaban memoria virtual.

En 1993 sale al mercado el Intel Pentium, que poseía una arquitectura capaz de ejecutar dos operaciones a la vez. Tenía un bus de 64 bits, y permitía accesos de memoria de 64 bits. Ofrecía velocidades de hasta 233 MHz, gracias a las instrucciones MMX.

En el año 2000, Intel lanza el Intel Pentium 4, microprocesador de séptima generación basado en arquitectura x86. Este microprocesador estrenaba la arquitectura NetBurst, en la que Intel sacrificaba rendimiento en cada ciclo para poder obtener una mayor cantidad de ciclos por segundo y una mejora en las instrucciones SSE.

El AMD Phenom salió al mercado en el 2007 de la mano de Advanced Micro Devices (AMD) y fue la primera generación de microprocesadores de tres y cuatro núcleos basados en arquitectura K10. Estos procesadores se diseñaron para facilitar el uso inteligente de los recursos y energía del sistema, generando un óptimo rendimiento.

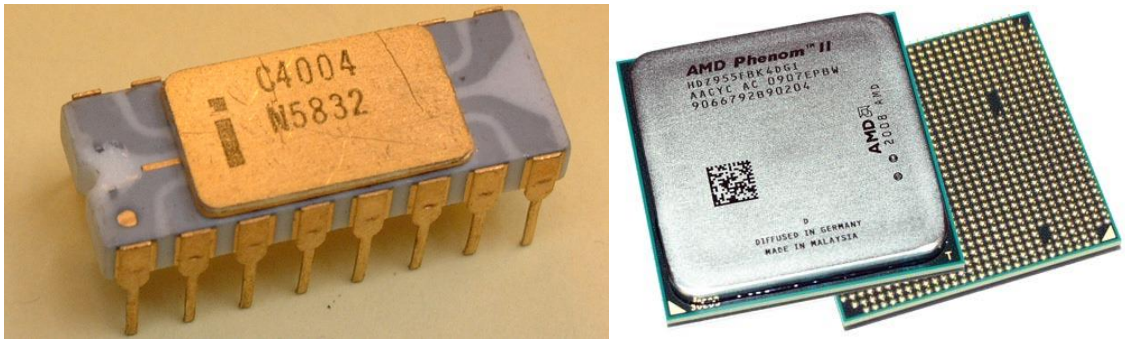


Figura 2.2-2 A la izquierda el Intel 4004 y a la derecha el AMD Phenom II

2.3. Introducción al microcontrolador STM32F4-Discovery

Este kit de desarrollo está diseñado para explorar las características de los microcontroladores Cortex-M4(los cortex-M están orientados a aplicaciones de bajo coste donde el bajo consumo es importante), a continuación se indican algunas de las características de este kit de desarrollo.

Figura 2.3-1 Microcontrolador STM32F4-Discovery

Características

- Mecanismo de depuración ST-LINK/V2 integrado.
- Alimentación mediante el USB o mediante fuente externa auxiliar de 5 V.
- Acelerómetro de 3 ejes LIS302DL.
- Ocho LEDs:
  - LD1 (rojo/verde) para indicar la comunicación USB.
  - LD2 (rojo) indicador de alimentación.
  - 4 LEDs para el usuario: LD3 (naranja), LD4 (verde), LD5 (rojo) y LD6 (azul).
- Dos pulsadores:
  - User
  - Reset
- Oscilador RC
- 1 Mbyte de memoria flash

Para la conexión entre la placa y el Pc se requiere un puerto USB libre en el pc y un cable de conexión USB A a mini-B.

2.3.1. Creación de un proyecto en Atollic

Cuando se crea un proyecto, es necesario incluir todos los drivers del microcontrolador, los archivos necesarios para la depuración, etc. Al usar Atollic podemos seleccionar el fabricante y el microcontrolador que vamos a utilizar, y de esta forma el programa incluye todos los archivos necesarios, además cuando creamos un proyecto de cero se carga un "programa de prueba", para que el usuario se familiarice con el microcontrolador.

Tras realizar la programación del proyecto es necesario depurarlo, en la depuración Atollic se encarga de buscar errores en el programa. Es necesario destacar que en este proceso Atollic solamente busca errores como por ejemplo: si falta algún punto y coma al final de alguna línea, alguna variable que se usa y no se ha declarado, etc. Es decir, que puede ocurrir, que aunque Atollic no detecte errores puede no funcionar como el programador espera.

Una vez se ha depurado el programa, y siempre y cuando no tenga ningún error podrá cargarse el programa en el microcontrolador. Si hubiera algún error será necesario arreglarlo antes de cargar el programa en el microcontrolador.

2.4. Objetivos del control

Se quiere controlar un convertidor CC-CC, que en nuestro caso será un flyback, dado que una antigua alumna de la UC3M realizó un proyecto similar y el convertidor está ya construido. Para poder realizar este control se requerirán las siguientes acciones:

- Control del estado del Mosfet
- Muestreo de la tensión a la salida del convertidor

En la siguiente tabla se indican las funciones que hay que realizar asociadas al periférico que las llevará a cabo.

Además de los periféricos anteriores, se han programado también los siguientes:

- GPIOs (puertos de entrada y salida)
- TIMERs (relojes)
- SysTick (reloj del sistema).

En la Figura 2.4-1 se muestran otros periféricos del microcontrolador.

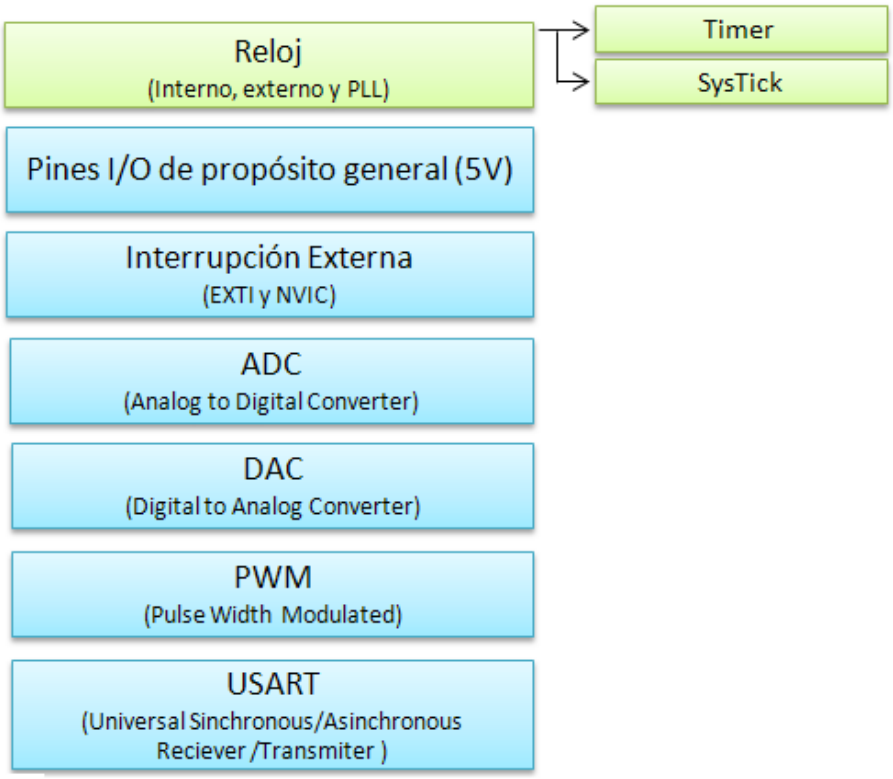


Figura 2.4-1 Algunos periféricos del microcontrolador

Para realizar el proyecto se ha usado el entorno de desarrollo Atollic TrueStudio, que tiene una versión gratuita para estudiantes, aunque para ello hay que registrarse y cumplimentar un breve cuestionario.

2.5. Configuración de los periféricos

Los periféricos deben configurarse a través de registros, cada uno tiene al menos los siguientes tres registros:

- Registro de control
- Registro de estado
- Registro de datos

Para los registros que se pueden modificar, dicha modificación se realiza mediante máscaras lógicas (compuestas por AND y OR).

De forma que se facilite la configuración, para realizarla de una forma más intuitiva, se programan los drivers. En el IDE elegido (Atollic) los drivers se incluyen cuando se crea el proyecto. La lectura y comprensión de los drivers cuyos periféricos se van a utilizar es una tarea que resulta muy útil para la realización de cualquier proyecto.

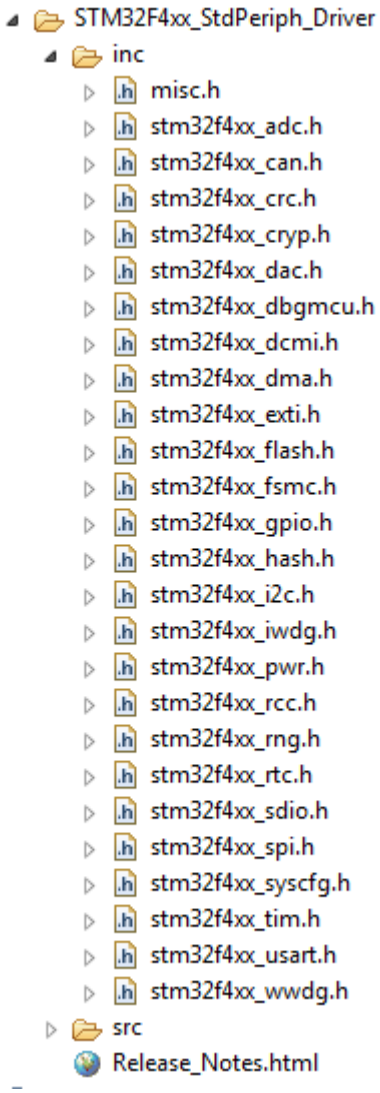


Figura 2.5-1 Drivers del microcontrolador STM32F4- Discovery

En este proyecto se utilizan los siguientes periféricos:

- Reset and Clock Control (RCC). Se trata del reloj del microcontrolador. Este periférico es necesario programarlo siempre, ya que sin él el resto de periféricos no funcionarán.
- System Tick (SysTick). Es el “tick” del sistema, se puede definir el “tick” que se desee, y en este caso se usa para definir la frecuencia de muestreo.
- General Purpouse Input/Output (GPIOS). Son los puertos de salida y entrada. Es necesario configurarlos si se va a usar otro periférico que necesite alguno de los puertos. En este caso hay dos entradas, el ADC y otra para la comunicación serie y dos salidas, una PWM y un puerto de salida para la comunicación.
- Nested Vector Interrupt Controller (NVIC). Es el vector de interrupciones. En este caso la comunicación serie se programa por interrupción.
- Analog to Digital Converter (ADC). Se trata del conversor analógico digital, en este caso se usa para muestrear la salida del convertidor.

- Pulse Width Modulator (PWM). Es el periférico que genera la señal de salida PWM. En este proyecto se usa una salida PWM.

2.5.1. Reset and Clock Control (RCC)

Este microcontrolador funciona a una frecuencia de 168 MHz, pero además puede trabajar con frecuencias más bajas, lo que sirve para bajar también el consumo. En esta clase de microcontroladores es necesario optimizar la frecuencia del reloj para nuestra aplicación, de forma que optimizaremos también el consumo de energía.

Es fundamental activar los relojes de los periféricos al principio del programa, ya que si no estos no funcionaran. Los periféricos están agrupados en diferentes buses, hay que localizar estos buses ya que para cada bus tienen ligeras variaciones. Para saber a qué bus está conectado cada periférico es necesario consultar los drivers.

```
void RCC_Configuration(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    /* Enable UART clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
 }
```

2.5.2. System tick (SysTick)

El SysTick es un reloj del Sistema que genera una interrupción cada cierto tiempo. Este tiempo puede configurarse, y es muy útil para contar tiempo por ejemplo. En este proyecto se usa para generar la frecuencia de muestreo.

Lo primero que hay que hacer es configurar el número de ticks que queremos que se produzcan en un segundo.

```
RCC_GetClocksFreq(&RCC_Clocks);
SysTick_Config(RCC_Clocks.HCLK_Frequency/500000);
```

En este caso se ha programado una interrupción cada 2 microsegundos. Además en la interrupción asociada al SysTick, se ha programado un contador, de forma que cuando cuenta 5 interrupciones activa un flag para que se tome una muestra en ese momento.

2.5.3. Nested Vector Interrupt Controller (NVIC)

Las interrupciones son eventos que deben ser atendidos de inmediato. Las interrupciones que programa el usuario provienen de los periféricos, por ejemplo: el ADC ha realizado una conversión, que le llega algún dato por el puerto serie, etc. El uso de interrupciones supone un gran avance en el uso de los recursos del microcontrolador, ya que el programa se ejecuta normalmente, y en el momento en el que se recibe una interrupción se detiene la ejecución del programa principal y se ejecuta la interrupción. Las interrupciones deben ser breves, y no incluir demasiado código. Cuando se termina de ejecutar la interrupción se vuelve a ejecutar el programa principal donde se había dejado.

Como su propio nombre indica el NVIC es un vector, en él se encuentran las direcciones de las rutinas de atención a la interrupción (RAI). Es probable que se puedan producir varias interrupciones a la vez, por lo que para evitar conflictos es necesario establecer prioridades a las interrupciones.

Cuando se produce una interrupción se activa un bit, de forma que antes de salir de la RAI es necesario resetearlo, ya que si no se hace puede provocar que se ejecute continuamente dicha interrupción.

```
void NVIC_Config(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  // Enable the USARTx Interrupt
  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority =  0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

2.5.4. General purpouse Input/Output (GPIO)

El GPIO es uno de los periféricos que más se usan, y que más sencillo es de configurar, permite que el micro se comunique con el exterior, ya sea con comunicaciones, con el ADC o bien algo tan sencillo como encender un LED para indicar algún evento.

Los GPIO se pueden configurar en cualquier parte del programa, si bien como es lógico se necesitan configurar antes de intentar usarlos, también es necesario activar los relojes pertinentes como se indica en el apartado del RCC.

A continuación se incluye la configuración de dos pines, uno de salida (primero) y uno de entrada (segundo), que servirán como representación de los demás.

```
GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

2.5.5. Analog to Digital Converter (ADC)

El conversor analógico digital (ADC) es el periférico que permite obtener el equivalente digital de una señal analógica, y así usarla internamente en el microprocesador.

Uno de los parámetros más importantes del ADC es la resolución, y se define como el valor mínimo de tensión que es capaz de medir el ADC. En este proyecto el ADC se ha configurado con 12 bits de resolución, por lo tanto la resolución es la siguiente:

$$Resolución = \frac{3V}{2^{12} - 1} = 0.73\ mV \qquad (1)$$

La lectura del ADC se realiza por polling (se comprueba continuamente si se ha realizado el evento), por lo que no es necesario programar la interrupción del periférico. La lectura de datos se realiza a la frecuencia de muestreo, y se implementa con el SysTick.

Para el ADC, es necesario configurar el pin como entrada analógica. El código para configurar el ADC es el siguiente:

```
void ADC_Config(void)
{
  /* Enable The HSI  */
  RCC_HSICmd(ENABLE);
  /* Enable the GPIOF or GPIOA Clock */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  /* Enable ADC1 clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
   /* Configure PA.01 (ADC Channel) in analog mode */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
  /* Check that HSI oscillator is ready */
  while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET);

  //ADC_BankSelection(ADC1, ADC_Bank_B);

  ADC_StructInit(&ADC_InitStructure);
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC1, &ADC_InitStructure);


  /* ADC1 regular channel1 configuration */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_3Cycles);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Start ADC1 Software Conversion */
  ADC_SoftwareStartConv(ADC1);

  /* Wait until ADC Channel 5 or 1 end of conversion */
  while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET)
  {
  }

}
```

2.5.6. Pulse Width Modulator (PWM)

La PWM es de gran utilidad, ya que no solo se puede usar para el control de convertidores, sino que también puede usarse en el control de motores. Su funcionamiento no es nada sencillo, pero gracias a los drivers puede programarse de una forma relativamente fácil. La PWM es una de las cuatro funciones de los TIMERs: Timming (temporizador), Output Compare (crea intervalos de tiempo y puede usarse para generar ondas de salida), Input Capture (cronómetro) y la PWM.

Para empezar hay que consultar en que pines están disponibles los TIMERs. Seguidamente hay que configurar la PWM para el disparo del MOSFET.

El código necesario para configurar la PWM es el siguiente:

```
void
{
    // Configuración del PWM através del timer TIM11 y el pin GPIO_Pin_9 del puerto GPIOB

    /*---------------------- System Clocks Configuration -----------------------*/
     /* TIM13 clock enable */
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM13 , ENABLE); //Activa el reloj del Timer 13
     /* GPIOB clock enable */
     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE); // Activa el reloj del GPIOA y B

     /* ----------------------------GPIO Configuration ----------------------------*/
     GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
     GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
     GPIO_Init(GPIOA, &GPIO_InitStructure);

     /* Remap PB9 pin to TIM11 */
     GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_TIM13);


     /* Time base configuration del Timer 13 PWM principal*/
     TIM_TimeBaseStructure.TIM_Period = PWM_Periodo_TIM13;
     TIM_TimeBaseStructure.TIM_Prescaler = 0;
     TIM_TimeBaseStructure.TIM_ClockDivision = 0;
     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

     TIM_TimeBaseInit(TIM13, &TIM_TimeBaseStructure);


     /* PWM1 Mode configuration: Timer 13 Channel1 */
     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
     TIM_OCInitStructure.TIM_Pulse = 0; //33;//CCR1Val; Aquí se configura el ciclo de trabajo inicial
     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

     TIM_OC1Init(TIM13, &TIM_OCInitStructure);

     TIM_OC1PreloadConfig(TIM13, TIM_OCPreload_Enable);

     TIM_ARRPreloadConfig(TIM13, ENABLE);
     /* TIM13 enable counter */
     TIM_Cmd(TIM13, ENABLE);


}
```

2.5.7. Universal Synchronous/Asynchronous Receiver/Transmiter (USART)

Para la configuración de la USART es necesario haber realizado previamente la configuración de interrupciones, incluyéndolo. La configuración correcta de los pines de recepción (RxD) y transmisión (TxD) es muy importante:

```
void USART_Configuration(void)
{
  USART_InitTypeDef USART_InitStructure;

  /* USART configured as follow:
        - BaudRate = 9600 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled*/

  USART_InitStructure.USART_BaudRate = 9600;//9600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

  // USART configuration
  USART_Init(USART2, &USART_InitStructure);

  //Habilitando interrupciones por transmisión y recepción.
  USART_ITConfig(USART2, USART_IT_RXNE,ENABLE);

  // Enable the USART2
 USART_Cmd(USART2, ENABLE);

}
```

La recepción la realizaremos mediante interrupciones, dado que se desconoce en qué momento se recibirá información.

El código de la interrupción para recibir datos es el siguiente:

```
void USART2_IRQHandler(void)
{
    if(USART_GetITStatus (USART2, USART_IT_RXNE)    != RESET)
    {
    RxBuffer[RxCounter] = ((uint8_t) USART_ReceiveData(USART2));
    RxCounter++;

    if(RxCounter == 21)
        {
           RxCounter = 0;
        }
    USART_ClearITPendingBit(USART2,USART_IT_RXNE);
    }

}
```

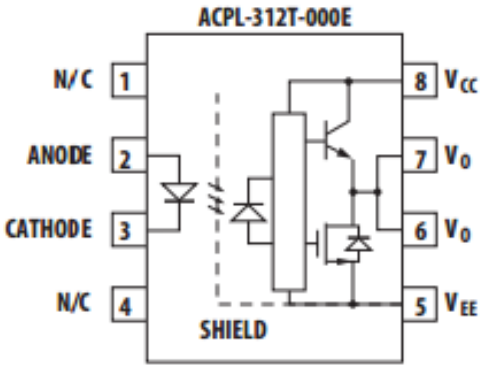
La información se va almacenando en un buffer llamado RxCounter mientras se va recibiendo. Este buffer es un vector de 21 posiciones.

Cuando se han recibido los 21 caracteres se convierten los datos, ya que se reciben como caracteres y es necesario transformarlos en números para asignarlos a sus correspondientes variables.

2.5.8. Hardware necesario para los periféricos

Para poder realizar correctamente el lazo de control del convertidor es necesario añadir algunos circuitos:

- Circuito de protección del ADC, está compuesto por una resistencia en serie con la puerta y un diodo zener. De esta forma la tensión de la puerta no superará los 3 V que admite la puerta del microcontrolador, que podría resultar dañada.
- Circuito opto-driver a la salida de las PWM. Para poder disparar los MOSFET no es suficiente con los 5 V y los 20 mA que aportan los puertos de salida. El opto-driver ACPL-312T-000E, puede manejar tensiones de 15 V y corrientes de hasta 2.5 A. Además es lo suficientemente rápido como para no introducir retrasos significativos en las conmutaciones. Su funcionamiento está basado en un fotodiodo, de forma que si ocurre alguna avería en el sistema, esta no dañaría el microcontrolador, por lo que también funciona como protección. Sus principales características están indicadas en la siguiente figura:



- 2.5 A maximum peak output current
- 2.0 A minimum peak output current
- 25 kV/µs minimum Common Mode Rejection (CMR) at V_{CM} = 1500 V
- 0.5 V maximum low level output voltage (V_{OL}) - Eliminates need for negative gate drive
- I_{CC} = 5 mA maximum supply current
- Under Voltage Lock-Out protection (UVLO) with hysteresis
- Wide operating V_{CC} range: 15 to 30 Volts
- 500 ns maximum switching speeds

Figura 2.5.8-1 Características opto-driver ACPL-312T-000E

- Regulador LM7815 para alimentar los dos opto-drivers con 15 V.

Para la configuración de los opto drivers se han seguido las instrucciones dadas por el fabricantes en la hoja de características, para el regulador LM7815 se ha seguido el mismo procedimiento.

Los valores de los componentes se definen en la Tabla 2.5.8-1.

Resistencia R13	10KΩ
Diodo Zener DZ1	3.3 V
Resistencia R14, R16	40 Ω
Resistencia R15, R17	100 Ω
Condensadores electrolíticos C8 y C9	10 µF (63 V)
Condensadores cerámicos C11 y C10	1 µF

Tabla 2.5.8-1 Resumen de los componentes para configurar los periféricos

Para la comunicación es necesario un transductor USB-TTL, que sirve para "traducir" la comunicación serie mediante niveles TTL a USB, para conectarlo al PC.

2.6. Comunicaciones

La finalidad de comunicar el microcontrolador con el PC, es poder controlar en tiempo real el convertidor CC-CC.

Las comunicaciones se establecen mediante un protocolo de comunicación serie. En este protocolo se envía los datos bit a bit mediante la USART. Para comunicar el PC mediante un puerto USB es necesario un transductor USB-TTL, del cual que hay que instalar en el ordenador los drivers.

Para que la comunicación sea factible es necesario acceder al hiperterminal, para ello se usa el programa Tera Term, de libre distribución. Para que esta comunicación pueda realizarse es también imprescindible utilizar los mismos parámetros en el Tera Term que los que hemos usado al configurar el periférico USART.

Velocidad de transmisión	9600 baudios
Tamaño de los datos	8 bits
Tipo de paridad	N – No paridad
Bit de parada	1 bit

Tabla 2.6-1 Parámetros de la comunicación serie

En la Figura 2.6-1 se muestra como iniciar las comunicaciones, una vez realizado esto se puede comunicar perfectamente el microcontrolador con el PC.

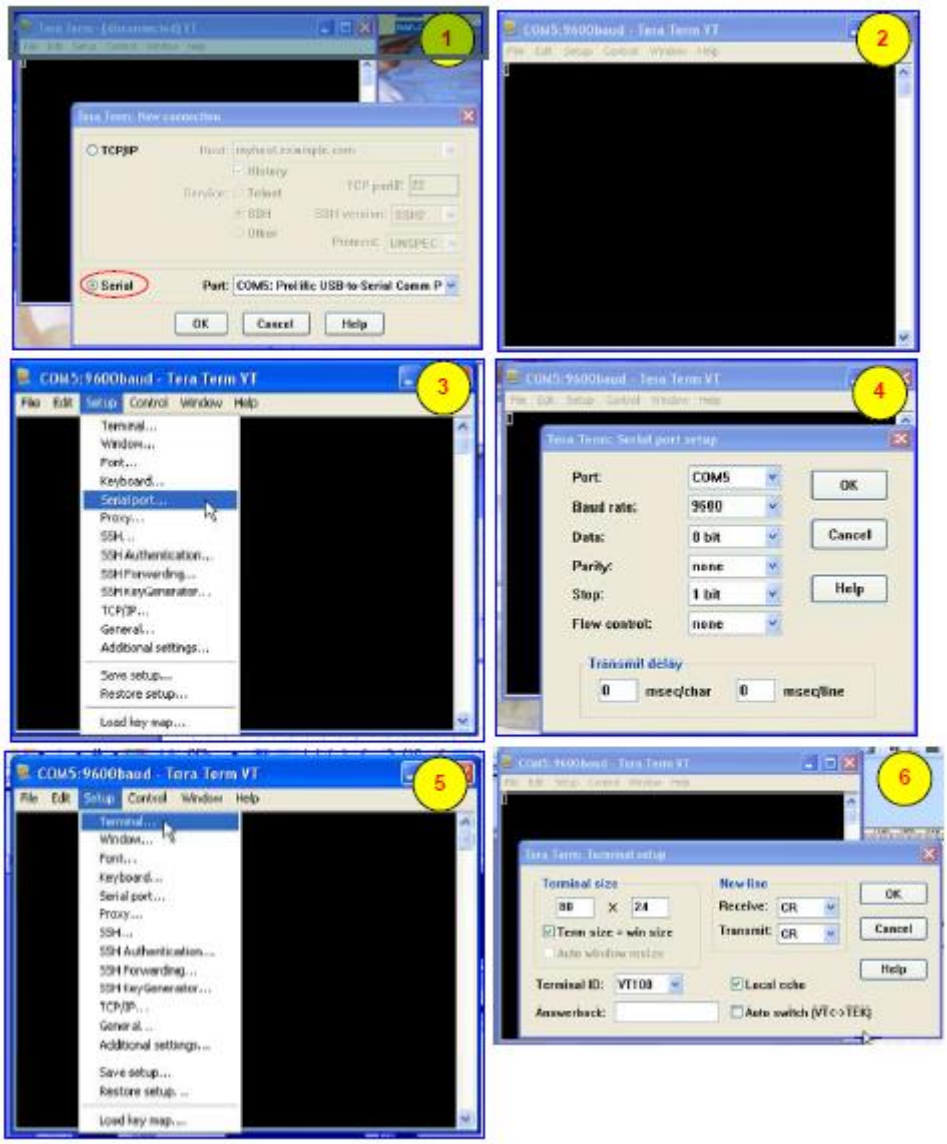


Figura 2.6-1 Guía para iniciar las comunicaciones entre el PC y el microcontrolador

El vector que se recibe consta de 21 posiciones, en la Tabla 2.6-2 se describe el índice y longitud de cada uno de los datos, además de una breve descripción.

Nombre	Índice	Longitud	Descripción
Tipo	0	1	Indica el tipo de regulador, 1 si es PI y 2 si es PID
Vref	1	5	Indica el valor de la tensión de referencia, directamente ligada con la tensión que se desea a la salida
Proporcional	6	5	Indica el valor de la constante proporcional
Integral	11	5	Indica el valor de la constante integral
Derivativa	15	5	Indica el valor de la constante derivativa

Tabla 2.6-2 Resumen del vector de comunicaciones

3. Reguladores

3.1. Introducción

Incluso en la naturaleza, todos los sistemas son regulados. El objetivo de regular un sistema es:

1- Mejorar la estabilidad.
   a. Conseguir un sistema estable a partir de uno inestable
   b. Mejorar la estabilidad de un sistema con una estabilidad baja
2- Régimen permanente.
   a. Seguimiento, sin error, de una señal.
   b. Eliminar de la salida las influencias de algunas perturbaciones
3- Respuesta transitoria
   a. Transitorio rápido
   b. Correcto amortiguamiento, baja sobre oscilación…

Los objetivos son los mencionados anteriormente, pero como se suele decir nada es gratis, por lo que en algunas ocasiones es necesario sacrificar por ejemplo velocidad en el transitorio para no tener una sobre oscilación demasiado elevada.

En cualquier sistema controlado se pueden definir tres acciones básicas de control:

Acción proporcional	La señal de control es proporcional a la señal de entrada (señal de error).
Acción integral	La señal es proporcional a la integral del error. Se elimina así el error en régimen permanente
Acción derivativa	La señal es proporcional a la derivada de la señal (la derivada del error es la variación de este). Ayuda a reducir las variaciones ante sobre oscilaciones.

Tabla 3.1-1 Resumen acciones básicas de control

Existen numerosos tipos de controladores, en el siguiente apartado se mencionan los más utilizados por su sencillez y robustez, son los siguientes: regulador proporcional (P), regulador proporcional integral (PI), regulador proporcional derivativo (PD) y el regulador que combina el PI y el PD, el PID.

3.1.1. Introducción a los reguladores (P, PI, PD y PID)

El análisis de los reguladores se va a hacer en el dominio de la frecuencia, conocido como dominio de Laplace, aunque estos reguladores poseen un equivalente en el dominio discreto o z. Será necesario conocer el equivalente en el dominio z de los reguladores para poder implementarlos en el microcontrolador.

La Tabla 3.1.1-1 resume los principales tipos de reguladores y su ecuación característica.

Regulador	Definición	Expresión matemática
P Proporcional	-Con el regulador proporcional es posible desplazar los polos del sistema realimentado, siempre en las ramas del lugar de las raíces.	$R(s) = K$ $u(t) = k \cdot e(t)$
PI Proporcional Integral	-Anula los errores de posición debido a la acción integral. -El polo en el origen aumenta el tipo de sistema y elimina el error en régimen permanente.	$R(s) = K \cdot \left(1 + \frac{1}{Ts}\right)$ $u(t) = k \cdot \left(e(t) + \frac{1}{T}\int e(t)dt\right)$
PD Proporcional Derivativo	-Predice linealmente el valor futuro del error. -Permite mejorar la respuesta del sistema en cuanto a sobre oscilación y tiempo de respuesta, sin afectar al error en régimen permanente.	$R(s) = K \cdot (1 + Ts)$ $u(t) = k \cdot \left(e(t) + T\frac{de(t)}{dt}\right)$
PID Proporcional Integral Derivativo	-Agrupa los efectos del PI y del PD	$R(s) = K \cdot \left(1 + \frac{1}{Ts} + Ts\right)$ $u(t) = k \cdot \left(e(t) + T\frac{de(t)}{dt} + \frac{1}{T}\int e(t)dt\right)$

Tabla 3.1.1-1 Resumen de los principales reguladores y sus ecuaciones

El diseño de los reguladores puede realizarse con las siguientes técnicas:

- Métodos empíricos: permite calcular un valor para los parámetros del PID cuando no disponemos del modelo del sistema a controlar.
    - Método Ziegler-Nichols en bucle abierto (respuesta ante escalón de entrada).
    - Método Ziegler-Nichols en bucle cerrado.
- Métodos analíticos o de asignación de polos. Fija los polos deseados del sistema en bucle cerrado, siguiendo los criterios de estabilidad y requisitos de funcionamiento. Para ello es necesario conocer la función de transferencia del sistema que se desea controlar.
    - Diseño basado en el lugar de las raíces.

3.1.2. Criterios de diseño de reguladores

Para diseñar un regulador para una fuente DC-DC las principales exigencias están relacionadas con:

- Impedancia de salida
- Respuesta transitoria
- Audiosuceptibilidad

Estas tres propiedades están relacionadas con la ganancia del regulador. Para obtener la ganancia del regulador, multiplicamos las ganancias de los bloques del mismo, esto implica que los retrasos producidos por cada uno de los bloques se suman.

La estabilidad de un regulador está relacionada con su ganancia y su fase. Por ello el análisis en frecuencia es vital para conocer la estabilidad del sistema.

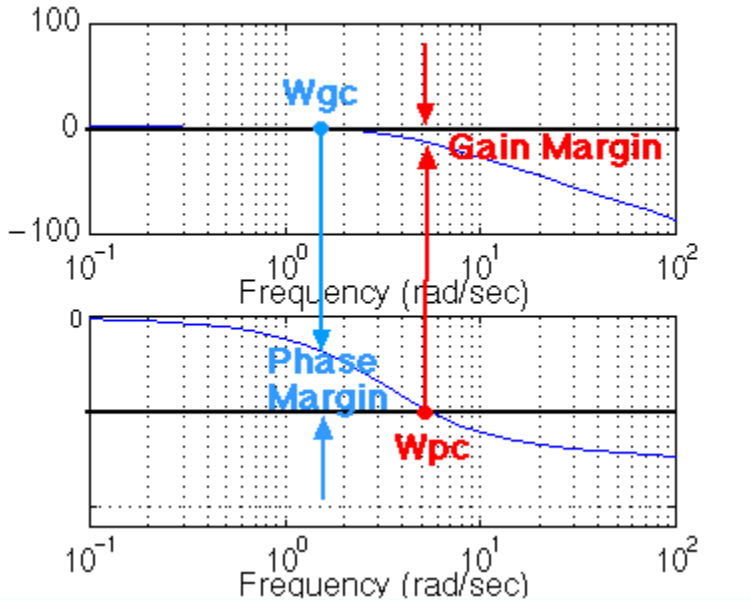


Figura 3.1.2-1 Diagrama de bode de un sistema, con el margen de fase y ganancia indicados

Considerando la figura 3.1.2-1, T(s) es la ganancia en lazo abierto del sistema, y definimos:

- Margen de ganancia (gain margin en inglés): es la ganancia del sistema (referida a 0 dB) cuando la fase es 180˚.
- Margen de fase (phase margin en inglés): es la diferencia entre 180˚ y la fase en el momento en el que la ganancia corta los 0 dB.
- Ancho de banda (bandwith en inglés): es la distancia en el eje X (se mide en frecuencia), desde el origen hasta que la ganancia cruza por 0 dB, en la imagen sería Wgc.

Criterios de diseño

1. El sistema oscilará durante los transitorios si el margen de fase en lazo abierto es muy pequeño, es decir si al cruzar la ganancia por cero la fase del sistema está muy próxima a 180˚.
2. Con un ancho de banda pequeño obtendremos una respuesta transitoria lenta.
3. El sistema será estable sin sacrificar respuesta dinámica si:
    a. El margen de fase en bucle abierto está alrededor de 50 ˚.
    b. A la frecuencia de conmutación la ganancia debe estar atenuada.

El criterio b es necesario para un buen funcionamiento, pero en el criterio a será necesario estudiar la planta y ver las limitaciones de nuestro control. Para facilitar el cálculo de reguladores en el mercado hay software que puede ayudarnos, por ejemplo SmartCtrl, que introduciendo los datos de la planta y del tipo de regulador nos ofrece un mapa de soluciones donde el diseñador puede seleccionar la que cumpla sus necesidades.

3.1.3. Transformada Z.

Aunque la transformada z fue introducida por el matemático polaco W. Hurewicz en 1947, para poder resolver ecuaciones diferenciales lineales con coeficientes constantes, la idea ya existía en tiempos de Laplace en 1744. Pero el nombre de transformada z se lo dieron Ragazzini y Zadeh en 1952.

La transformada de Laplace y la transformada z son similares, ambas modelan eventos en el tiempo, pero la diferencia radica en que la transformada z modela eventos discretos, mientras que la transformada de Laplace modela eventos continuos en el tiempo. Por ello la transformada de Laplace se usa en el diseño de reguladores analógicos, y la transformada z en reguladores digitales.

Ambas transformadas están tan relacionadas, que es posible diseñar un regulador mediante un lugar de las raíces por ejemplo (regulador continuo) y discretizarlo para de esta forma implementarlo en un microcontrolador. Esto es posible desde un punto de vista teórico, es decir de forma matemática, pero en la realidad hay que tener en cuenta otros factores, no se trata simplemente de realizar la discretización, sino que es necesario tener en cuenta el retraso producido al aproximarse a la frecuencia de muestreo, que puede provocar problemas de estabilidad.

La equivalencia matemática entre z y s se ve en la siguiente igualdad $z = e^{sT}$ donde T es el periodo de muestreo. En ambos dominios existe una zona estable de diseño, figura 3.1.3-1.

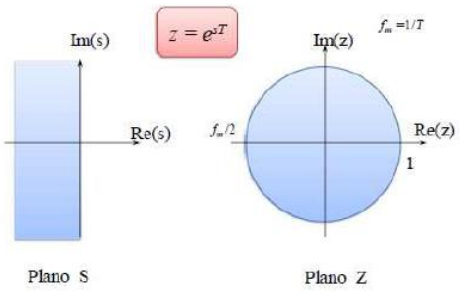


Figura 3.1.3-1 Equivalencia entre el domino s y z

3.1.4. Discretización de reguladores analógicos

Para discretizar un regulador existen tres métodos:

1- Método Euler I. Es conocido como forward.

$$s = \frac{z-1}{T} \tag{1}$$

2- Método Euler II. Es conocido como backward.

$$s = \frac{z-1}{T \cdot z} \tag{2}$$

3- Método Trapezoidal, de Tustin o transformación bilineal.

$$s = \frac{2(z-1)}{T \cdot z} \tag{3}$$

Estos métodos son una aproximación a la integral de la señal como un sumatorio de pequeñas áreas, figura 3.1.4-1. En la figura, siendo e (t) la señal continua en el tiempo que deseamos discretizar. Estas áreas pueden ser rectángulos (1) y (2), o trapecios (3).

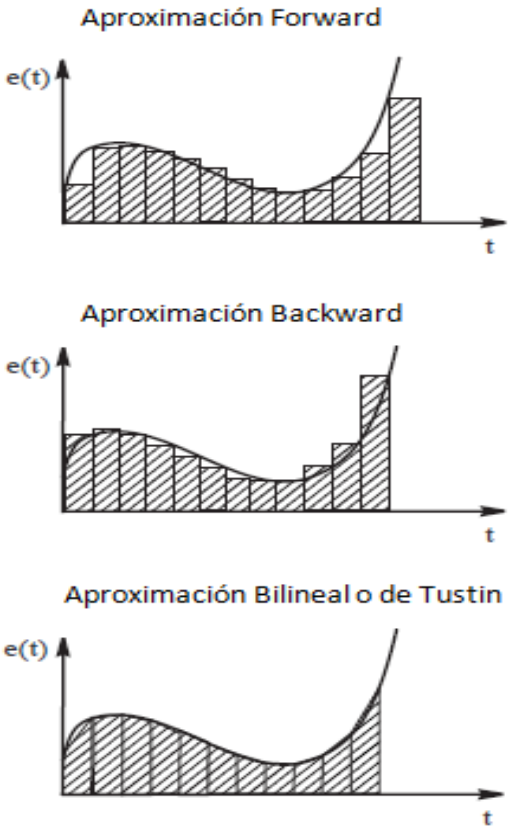


Figura 3.1.4-1 Tipos de aproximación de una señal continua a una discreta

3.2. Regulador PI

El diagrama de bloques del PI analógico se muestra a continuación. Siendo su función de transferencia:

$$\frac{Y(s)}{X(s)} = \left(K_p + \frac{K_i}{s}\right) \tag{4}$$

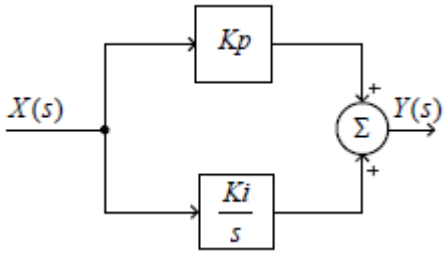


Figura 3.2-1 Diagrama de bloques de un regulador PI

En el diagrama de bloques K_p es el coeficiente proporcional y K_i el integral. La ecuación (4) puede expresarse también como (5).

$$\frac{Y}{X} = K \cdot \left(\frac{1 + sT}{sT}\right) \tag{5}$$

Siendo:

$$K_p = K \quad y \quad K_i = \frac{K}{T} = \frac{K_p}{T} \tag{6}$$

Si utilizáramos un amplificador operacional para implementar el PI, habría que usarlo en su configuración inversora, tal como indica la figura 3.2-2.

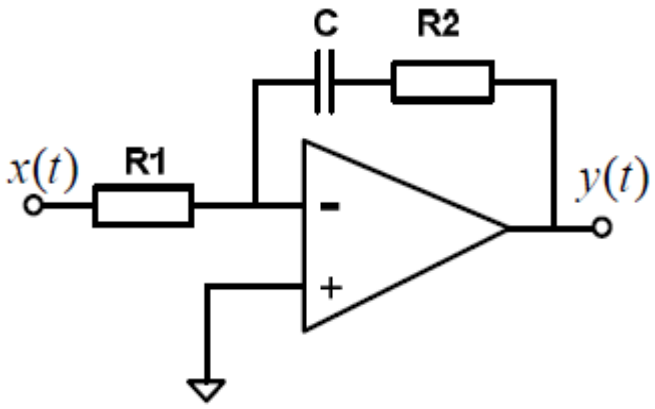


Figura 3.2-2 Regulador PI implementado con un amplificador operacional

En esta configuración se invierte la señal, dato importante que se debe tener en cuenta si se usa este circuito para implementar el regulador PI. La función de transferencia de la figura 3-5 es (7).

$$\frac{Y}{X} = -\left(\frac{R_2 + \frac{1}{Cs}}{R_1}\right) \tag{7}$$

Siendo:

$$K_p = \frac{R_2}{R_1} \tag{8}$$

$$K_i = \frac{1}{R_1 \cdot C} \tag{9}$$

3.2.1. Discretización del regulador PI.

Para el cálculo de la ecuación en diferencia se utiliza la aproximación de Tustin, que también se conoce como aproximación trapezoidal.

$$s \approx \frac{2 \cdot (z-1)}{Tm \cdot (z+1)} \tag{10}$$

$$z = e^{sTm} \tag{11}$$

La expresión para poder realizar la discretización es la siguiente:

$$x(k-1) = X(k) \cdot z^{-1} \tag{12}$$

$$x(k-2) = x(k-1) \cdot z^{-1} = x(k-2) \cdot z^{-2} \tag{13}$$

En las expresiones anteriores:

- $x(k)$ es la última muestra.
- $x(k-1)$es la muestra tomada en el instante anterior.
- z^{-1} indica el retraso introducido por el muestreo, con un periodo Tm.

Si sustituimos (12) en (13), se calcula la ecuación en diferencia del PI en función de la muestra anterior.

$$y(k) = y(k-1) + K_1 \cdot \big(x(k) - x(k-1)\big) + \frac{Tm \cdot K_2}{2} \cdot \big(x(k) + x(k-1)\big) \tag{14}$$

Los valores K_1 y K_2 se obtienen gracias a la funcionalidad “s2z Converter” de PSIM, que discretiza el PI analógico partiendo de k y T, que se obtienen a su vez de los valores del PI analógico que nos da SmartCtrl.

Dado que la frecuencia de muestreo es más de diez veces mayor que la frecuencia de corte del sistema, podemos asumir:

$$x(k) \approx x(k-1) \tag{15}$$

En este caso la ecuación (14) pasa a ser la ecuación (16).

$$y(k) = y(k-1) + K_i \cdot x(k) + K_p \cdot x(k) \tag{16}$$

El término y (k-1) sólo almacena los valores acumulados por la parte integral del control.

3.2.2. Implementación de un PI en un microcontrolador.

En la figura se muestra un regulador PI digital simplificado.

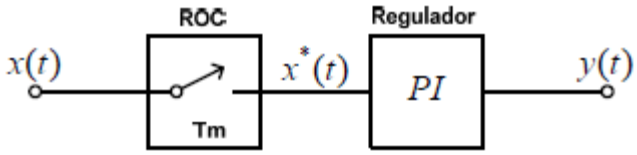


Figura 3.2.2-1 Diagrama de un regulador PI discreto

En la Figura 3.2.2-1:

- ROC es el bloqueador de orden cero, que se encarga de mantener el valor de x (t) constante a la salida durante un periodo.
- x*(t) es la señal x (t) discretizada.

Para que el microcontrolador trabaje en un punto fijo, la señal se digitaliza mediante un ADC.

Al usar el ADC introducimos una ganancia al sistema:

$$G_{ADC} = \frac{2^N - 1}{Vref} \tag{17}$$

En (17):

- N es el número de bits del ADC.
- Vref es la tensión de referencia.

Para que la ganancia del PI que implementaremos en el microcontrolador sea igual a la ganancia del PI analógico, hay que dividir el PI en punto fijo por la ganancia del ADC (G_{ADC}).

3.2.3. Re-escalado de los coeficientes

En alguna ocasión puede pasar que los valores que obtenemos con la calculadora S2z converter de PSIM sean menores que la unidad. Si esto pasa es necesario multiplicar el menor de los valores por un factor de escalado para obtener un número entero con suficiente número de dígitos significativos.

Llamaremos a este factor K_{ESC} , y su valor dependerá de la precisión que se quiera lograr. A continuación se explicará un ejemplo de cómo calcularlo.

Supongamos que mediante s2z converter hemos obtenido los siguientes valores:

K1(Kp)	0.2864
K2(KI)	114.7865

Tabla 3.2.3-1 coeficientes del regulador

En este caso deberíamos tener al menos dos cifras significativas, por lo que el valor mínimo será 100, ya que es el valor que necesita K1 para obtener dichas cifras. Por lo tanto los nuevos valores serán:

K1(Kp)	28
K2(Ki)	11478
K_{ESC}	100

Tabla 3.2.3-2 coeficientes del regulador escalados

De igual forma que hicimos con el ADC, debemos introducir una compensación a la salida del PI.

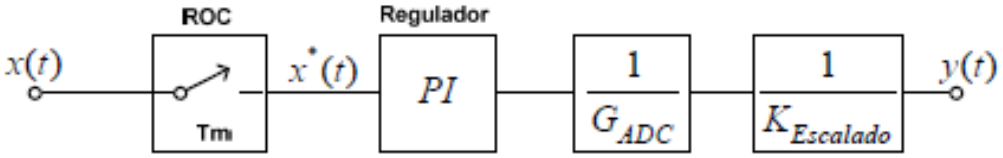


Figura 3.2.3-1 Regulador PI en punto fijo con reescalado

3.2.4. Saturación de variables en punto fijo en el PI

Al diseñar un control en punto fijo es necesario comprobar que las variables no desborden, esto se debe a que en dispositivos como por ejemplo microcontroladores o FPGAs las variables tienen un tamaño limitado.

En este caso debemos tener especial cuidado con la variable y(k-1).

Por ejemplo en el caso de nuestro microcontrolador, a una frecuencia de 100 KHz (f_{SW}) tiene una resolución de 1680 bits. Este valor se calcula de la siguiente forma:

$$R_{PWM} = \frac{F_{clock}}{f_{SW}} = \frac{168 \cdot 10^6}{10^5} = 1680 \qquad (18)$$

Pero dado que este TFG se aplicará a un control de un convertidor CC-CC, el ciclo de trabajo no puede superar la unidad, además es necesario que exista una transferencia de energía. Por lo que limitaremos el ciclo de trabajo a un 70%(d=0,7). Por lo tanto para definir el límite usaremos la ecuación (19).

$$y_{max} = {70 \cdot [(R_{PWM} - 1) \cdot G_{ADC} \cdot K_{ESC}]}/{100} \qquad (19)$$

3.3. Regulador PID

El diagrama de bloques del PID analógico se muestra a continuación. Siendo su función de transferencia:

$$\frac{Y(s)}{X(s)} = \left(K_p + \frac{K_i}{s} + K_d s\right) \tag{20}$$

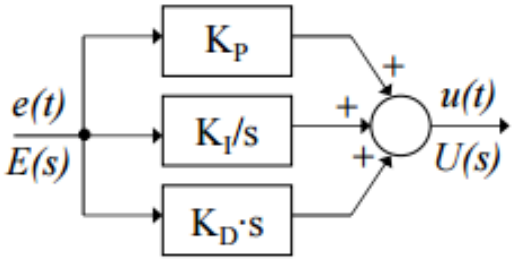


Figura 3.3-1 Diagrama de bloques de un regulador PID

En el diagrama de bloques K_p es el coeficiente proporcional, K_i el integral y K_d el diferencial

Si utilizáramos un amplificador operacional para implementar el PID, sería necesario el circuito que se muestra en la figura X.

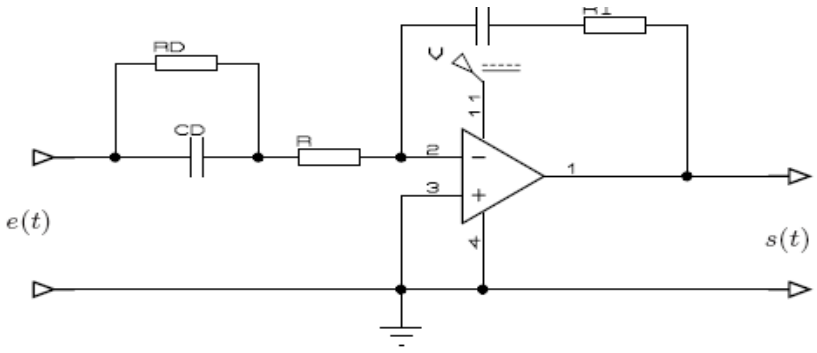


Figura 3.3-2 Regulador PI implementado con un amplificador operacional

Con esta configuración:

$$K_p = \frac{R_I}{R + R_D} \tag{22}$$

$$K_i = \frac{1}{R_I \cdot C} \tag{23}$$

$$K_d = R_D \cdot C_D \tag{24}$$

3.3.1. Discretización del regulador PID.

Para el cálculo de la ecuación en diferencia, y de forma análoga a lo que se ha hecho en el regulador PI se utiliza la aproximación de Tustin, que también se conoce como aproximación trapezoidal.

$$s \approx \frac{2 \cdot (z-1)}{Tm \cdot (z+1)} \tag{25}$$

$$z = e^{sTm} \tag{26}$$

Tras realizar la aproximación de Tustin obtenemos:

$$y(k) = y(k-1) + K_1 \cdot y(k) + K_2 \cdot y(k-1) * K_3 \cdot (k-2) \tag{27}$$

Dónde:

$$K_1 = K_p + \frac{K_i}{2} \cdot T + \frac{K_d}{T} \tag{28}$$

$$K_2 = -K_p + \frac{K_i}{2} \cdot T - \frac{2 \cdot K_d}{T} \tag{28}$$

$$K_3 = \frac{K_d}{T} \tag{28}$$

3.3.2. Implementación de un PID en un microcontrolador.

En la Figura 3.3.2-1 se muestra un regulador PID digital simplificado.

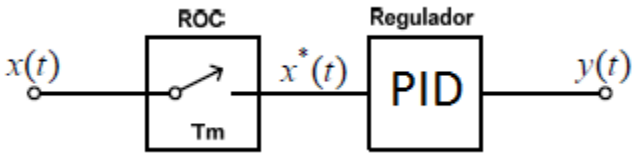


Figura 3.3.2-1 Diagrama de un regulador PID discreto

En la Figura 3-10:

- ROC es el bloqueador de orden cero, que se encarga de mantener el valor de x(t) constante a la salida durante un periodo.
- x*(t) es la señal x(t) discretizada.

Para que el microcontrolador trabaje en un punto fijo, la señal se digitaliza mediante un ADC.

Al usar el ADC introducimos una ganancia al sistema:

$$G_{ADC} = \frac{2^N - 1}{Vref} \tag{29}$$

En (29):

- N es el número de bits del ADC.
- Vref es la tensión de referencia.

Para que la ganancia del PID que implementaremos en el microcontrolador sea igual a la ganancia del PI analógico, hay que dividir el PI en punto fijo por la ganancia del ADC (G_{ADC}).

3.3.3. Re-escalado de los coeficientes

Tal y como pasaba en el regulador PI, en alguna ocasión puede pasar que los valores que obtenemos con la calculadora S2z converter de PSIM sean menores que la unidad. Si esto pasa es necesario multiplicar el menor de los valores por un factor de escalado para obtener un número entero con suficiente número de dígitos significativos.

De esta forma el diagrama de bloques queda de la siguiente forma:

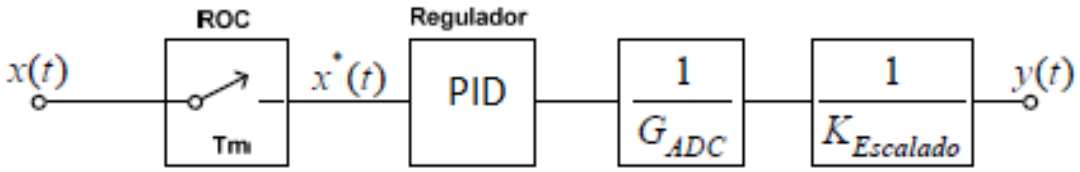


Figura 3.3.3-1 Regulador PID en punto fijo con reescalado

3.3.4. Saturación de variables en punto fijo en el PID

Como ocurría en el caso del PI, es necesario tener en cuenta que algunas variables acumulativas requieren una atención especial, para de esta forma evitar errores.

Igual que en el PID, la parte integral podría llegar a saturarse, por lo que es necesario implementar un control anti-windup, este control es exactamente igual que el implementado en la parte del PI.

4. Evaluación experimental

4.1. Introducción

El objetivo de este proyecto es diseñar un sistema de control para convertidores CC-CC, lo primero que se explicará en este capítulo es la etapa de potencia sobre la que se realizarán las pruebas. Se ha elegido la topología Flyback, ya que existía un convertidor diseñado y montado por una antigua alumna en su proyecto fin de carrera.

En [2] se puede encontrar toda la información relativa al funcionamiento del convertidor Flyback, por lo que se partirá de que el lector conoce como funciona este convertidor.

Para continuar se explicarán las pruebas realizadas para comprobar el funcionamiento del sistema de control.

4.2. Descripción de la etapa de potencia

En la figura 4.2-1 se muestra el esquemático del convertidor Flyback, compuesto por: fuente de alimentación, condensador de entrada, bobina, Mosfet, circuito snubber, carga resistiva y un circuito para programar escalones de carga.

Además están incluidos otros circuitos: transductor corriente-tensión basado en un sensor LEM, para monitorizar la tensión en el primario de la bobina, así como la medida de la corriente de salida, para poder evaluar la respuesta dinámica del sistema.

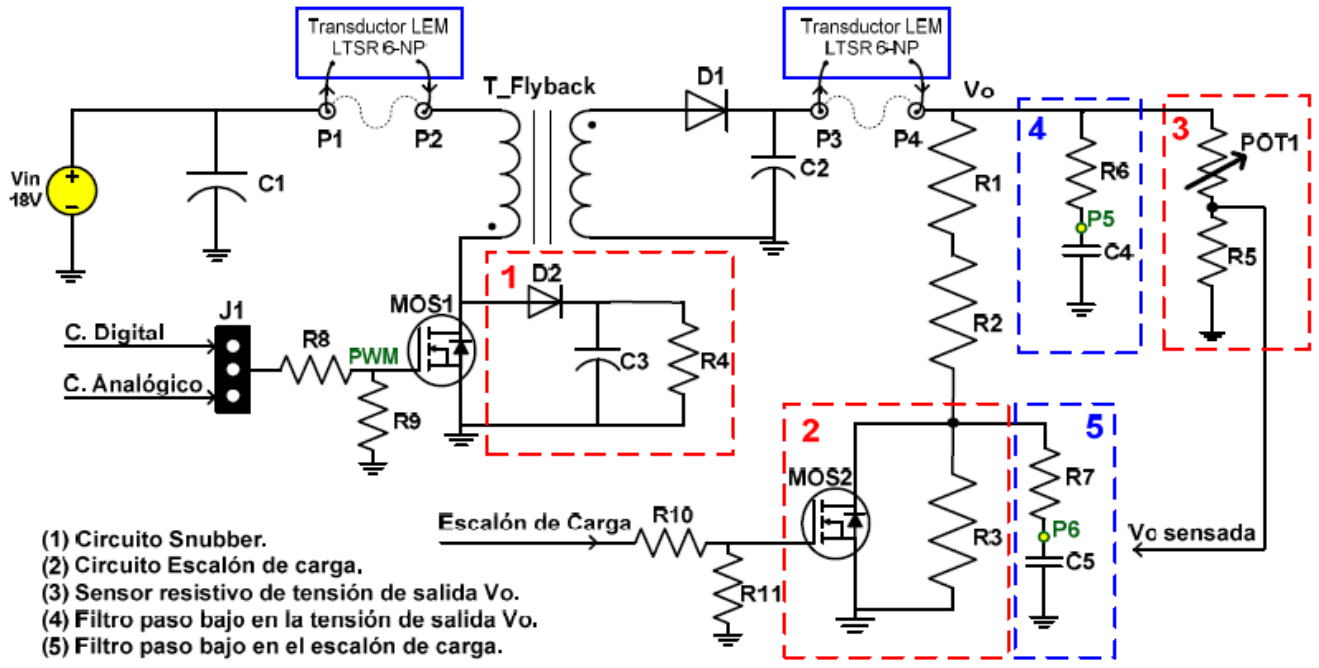


Figura 4.2-1 Esquemático del convertidor Flyback completo

Las características del convertidor Flyback se encuentran en la tabla 4-1.

Tensión de entrada (Vin)	18 V
Tensión de salida (Vo)	15 V
Frecuencia de conmutación (fsw)	100 kHz
Potencia de salida (Po)	24 W
Modo de trabajo	MCC
Temperatura máxima	60 ˚C

Tabla 4.2-1 Resumen de las características del convertidor Flyback

4.2.1. Fuente de alimentación

Para la alimentación se usará un alimentador portátil universal AC/DC de 39 W. La tensión de salida de la fuente es 18 V. En la PCB (printed circuit board), se colocará un conector hembra.

4.2.2. Condensadores de entrada y salida y diodo del filtro de salida

A continuación se explican los valores y funciones de los condensadores de entrada y salida y del diodo del filtro de salida.

- Condensador electrolítico de entrada. La función de este condensador es desacoplar la entrada del primario del convertidor del cableado de la fuente de tensión.

Capacidad	470 µF
ESR	10 mΩ
Tensión	25 V

Tabla 4.2.2-1 Características del condensador de entrada

- Condensador electrolítico de salida. El valor de la capacidad de este condensador se determina por el valor máximo de rizado permitido en la tensión de salida. Además este condensador tiene una influencia directa en la dinámica del sistema.

Capacidad	330 µF
ESR	10 mΩ
Tensión	35 V

Tabla 4.2.2-2 Características del condensador de salida

- Diodo del filtro de salida del convertidor. El diodo del filtro de salida se elige según la corriente máxima que atraviesa el filtro de salida del convertidor y la potencia que debe disipar.

Modelo	STPS10L60CFP
If	5 A

Tabla 4.2.2-3 Características diodo del filtro de salida

4.2.3. Bobinas acopladas

Usualmente cuando se habla del convertidor Flyback, se dice que tiene un transformador que transmite la energía desde el primario al secundario, pero este término n oes correcto, ya que el convertidor Flyback en realidad tiene lo que se denominan bobinas acopladas.

Las diferencias entre bobinas acopladas y un transformador son las siguientes:

- Desde el punto de vista constructivo. Aunque ambas se arrollan en torno a un núcleo, las bobinas acopladas se suelen diseñar con un entrehierro donde se almacena la energía, no así los transformadores.
- Desde el punto de vista energético. Los transformadores no almacenan energía, es decir transmiten la energía desde el primario hasta el secundario de manera instantánea. Pero las bobinas acopladas transmiten la energía en dos fases, durante la primera almacenan la energía, y en la segunda parte la transmiten al secundario.

El convertidor Flyback que se va a utilizar para realizar las pruebas de este trabajo tiene una bobina modelo FA 2901, del fabricante Coilcraft.

Part number[1]	L at 0 A[2] ±10% (µH)	L at Ipk[3] min (µH)	DCR max (mOhms)[4] pri	sec	aux	Leakage L (µH)[5]	Turns ratio[6] pri : sec	pri : aux	Ipk[3] (A)	Output[7]
FA2677-AL_	50	40.8	26	2.6	328	2.20	1 : 0.12	1 : 0.41	2.9	3.3 V, 7.5 A
FA2898-AL_	50	40.8	26	4.0	315	1.05	1 : 0.18	1 : 0.41	2.9	5 V, 5 A
FA2899-AL_	50	40.8	26	13	315	0.473	1 : 0.35	1 : 0.41	2.9	9 V, 2.8 A
FA2900-AL_	50	40.8	26	20	315	0.409	1 : 0.47	1 : 0.41	2.9	12 V, 2.1 A
FA2901-AL_	50	40.8	26	37	305	0.381	1 : 0.59	1 : 0.41	2.9	15 V, 1.7 A

Figura 4.2.3-1 Características de la bobina FA2901

Potencia	25 W
Tensión de entrada	18 a 75 V
[2] Inductancia del primario medida a 25 kHz, 0.7 Vrms, 0 A dc	50 µH
[5] Inductancia de pérdidas, medidas en el primario con los pines del secundario cortocircuitados	0.381 µH
[6] Relación de transformación del primario y secundario conectados en paralelo	1:0.59
[7] Salida de corriente del secundario	15 V, 1.7 A

Tabla 4.2.3-1 Resumen de las principales características de la bobina FA2901

La inductancia de pérdidas (o dispersión), corresponde a la fracción de la energía que circula por el primario pero no se transmite al secundario. Por ello se debe colocar un circuito snubber para que esta energía no se soporte íntegramente por el MOSFET, ya que podría suponer su rotura.

4.2.4. MOSFET

El transistor es el encargado de la conmutación en el convertidor Flyback, en este caso trabaja en las zonas de corte y saturación, por lo que se comporta como un interruptor.

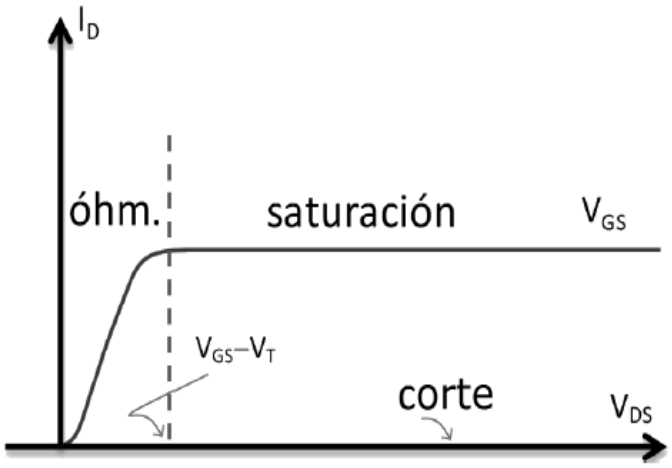


Figura 4.2.4-1 Zonas de funcionamiento de un MOSFET

La señal de apagado o encendido proviene del circuito de control, es decir del microcontrolador. Pero para que el MOSFET trabaje necesita una corriente y tensiones mínimas, que el microcontrolador no es capaz de proporcionarle, por lo tanto se ha optado por incluir un opto driver ACPL-312T-000E.

Para elegir el MOSFET deben tenerse en cuenta:

- Máxima tensión drenador-fuente, V_{DSS}. Tensión máxima que soportará el MOSFET sin romperse.
- Máxima corriente media del drenador, I_D.
- Tensión umbral y máxima de puerta, V_{GS}. Tensión proporcionada en nuestro caso por el opto-driver. Esta tensión varía con la temperatura.
- Velocidad de conmutación. Este parámetro define el tiempo que le MOSFET tarda en pasar de corte a saturación.
- Resistencia en conducción, R_{DS}. Este valor es fundamental ya que las pérdidas por conducción del MOSFET están relacionadas con este valor. Por lo tanto interesa que sea lo más bajo posible.

El MOSFET de este convertidor es el IRFB31N20D con un encapsulado TO-220.

4.2.5. Snubber RCD

Para evacuar la energía que no se transmite al secundario del convertidor, esto se debe a la L de dispersión, el convertidor cuenta con un circuito snubber, cuya misión principal en este caso es evitar el “picotazo” de tensión durante el paso de corte a saturación.

Los dos tipos de snubber más utilizados son:

- Snubber RC (condensador y resistencia).
- Snubber RCD (diodo, condensador y resistencia).

En este caso el snubber es del tipo RCD, su principal inconveniente es que disipa la potencia en la resistencia.

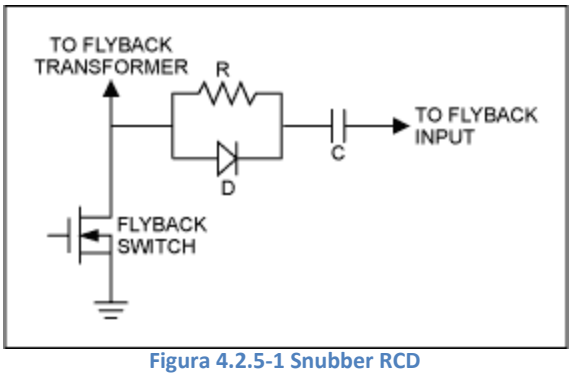


Figura 4.2.5-1 Snubber RCD

Cuando se diseña el snubber se debe tener en cuenta la frecuencia de corte, ya que debe ser mucho menor que la frecuencia de conmutación del convertidor, para que la tensión en el condensador no varíe significativamente durante los apagados y encendidos del MOSFET.

Los valores del snubber de este circuito son los indicados en la tabla 4.2.5-1.

Diodo	MUR 460
Condensador	33 µF y 100 V
Resistencia	3.3 kΩ y 5 W

Tabla 4.2.5-1 Resumen de los componentes del snubber

4.2.6. Resistencia de carga

La resistencia de carga de este convertidor es de 9.4 Ω, y está compuesta por dos resistencias acorazadas de ARCOL de 4.7 Ω. Este valor se obtiene con la ecuación (1), sabiendo que la tensión de salida es de 15 V y disipa una potencia de 25 W.

$$R = \frac{V_o^2}{P} = \frac{15^2}{25} = 9\ \Omega \qquad (1)$$

Además este convertidor cuenta con una resistencia ARCOL de 20 Ω, para poder generar un escalón de carga, para que el sistema pase de MCC (modo conducción continuo) a MCD (modo conducción discontinuo). Este circuito se muestra en la figura 4.2.6-1.

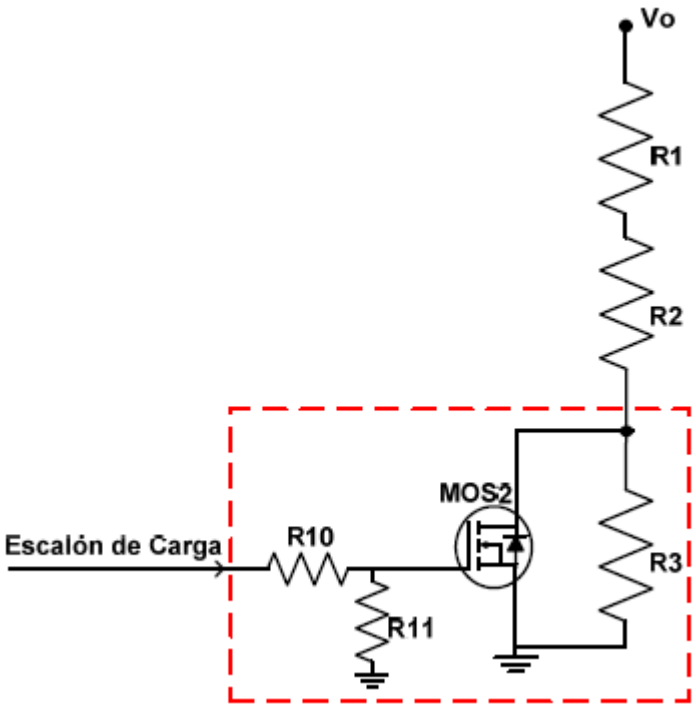


Figura 4.2.6-2 Circuito de carga incluyendo escalón de carga

Los valores de este circuito están recogidos en la Tabla 4.2.6-1.

R1	4.7 Ω (Acorazada)
R2	4.7 Ω (Acorazada)
R3(escalón de carga)	20 Ω (Acorazada)
MOS2	IRF 540
R11	100 kΩ
R10	33 Ω y 0.5 W

Tabla 4.2.6-1 Resumen de componentes del circuito de carga

Este elemento es el que más energía disipa en todo el circuito, por lo que está equipado con un disipador para evitar sobrecalentamiento, se trata de un disipador LAM 150x50mm con ventilador de 12 V.

4.3. Mediciones experimentales

A continuación se explicarán las pruebas que se han realizado para comprobar el funcionamiento del lazo de control programado en el microcontrolador.

4.3.1. Instalación utilizada

Todas las pruebas se han realizado en el laboratorio de electrónica de potencia de la UC3M, siendo todos los aparatos a excepción del ordenador portátil propiedad de la propia universidad. En la figura 4.3.1-1 se muestran todos los instrumentos que se han utilizado para realizar las pruebas.

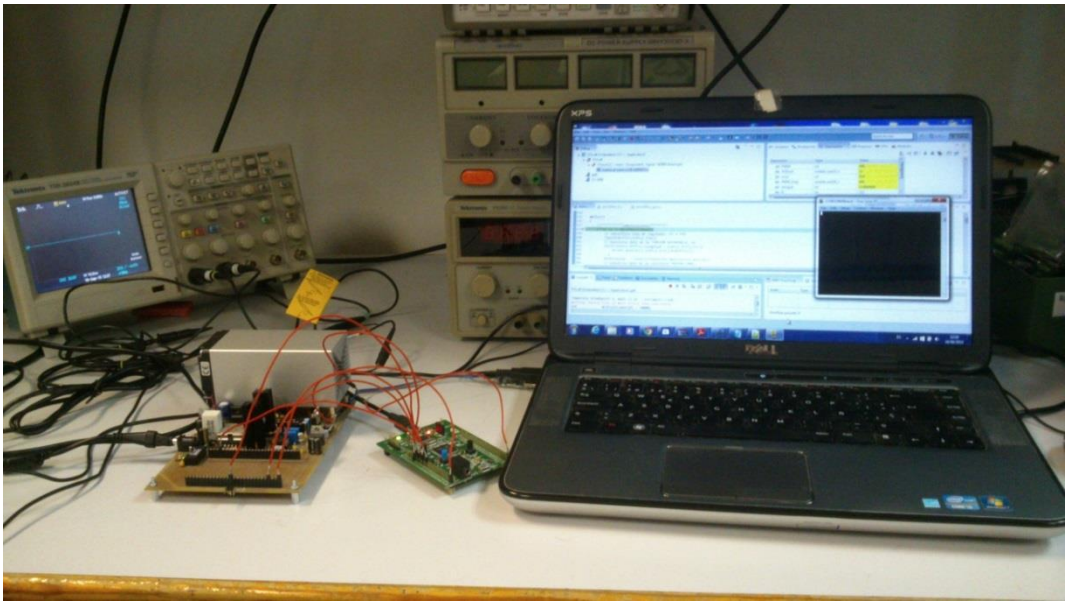


Figura 4.3.1-1 Instalación para realizar medidas

- Osciloscopio Tectronix TDS2024B.
- Fuente de tensión Lendher IMHY3003D-3.
- Ordenador portátil Dell.
- PCB con la etapa de potencia descrita anteriormente.
- Cargador de portátil universal AC/DC 39 W, para alimentar la etapa de potencia.
- Microcontrolador STM32 F4 Discovery de STMicroelectronics.

Además el software utilizado es, ver Figura 4.3.1-2.

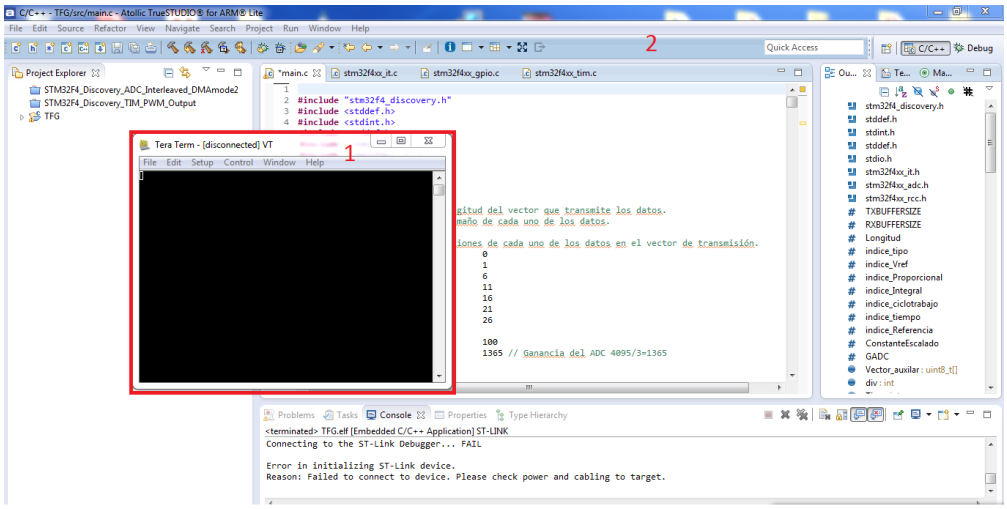
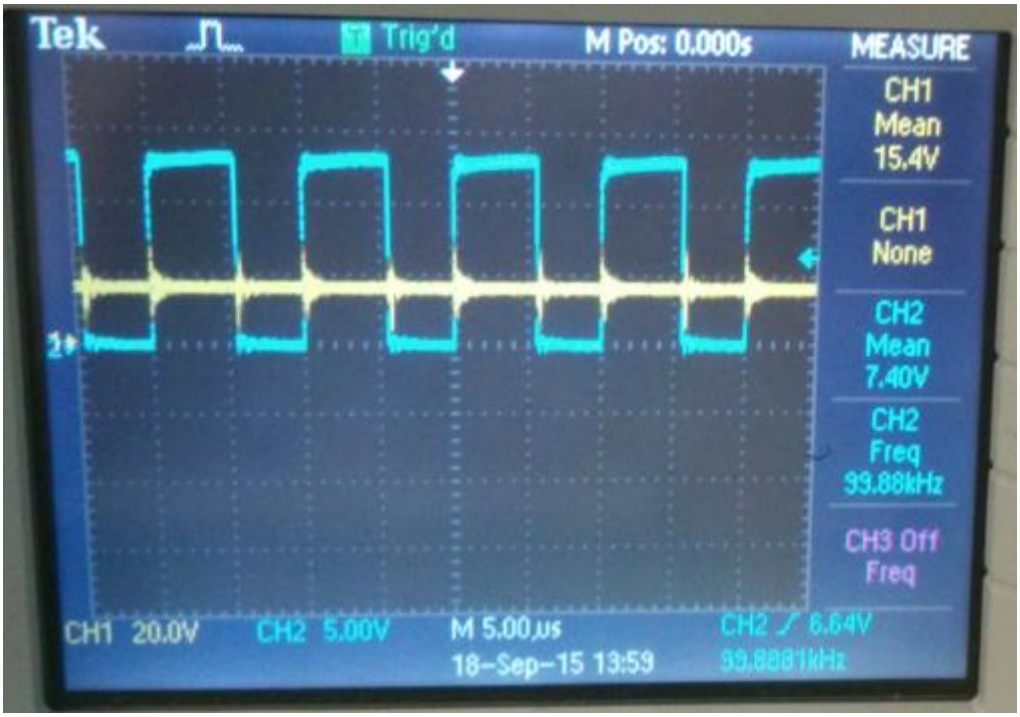


Figura 4.3.1-2 Software utilizado para las pruebas

1. Atollic TrueStudio, para descargar el programa en el microcontrolador.
2. Tera Term para las comunicaciones entre el microcontrolador y el PC.

4.3.2. Mediciones con regulador PI

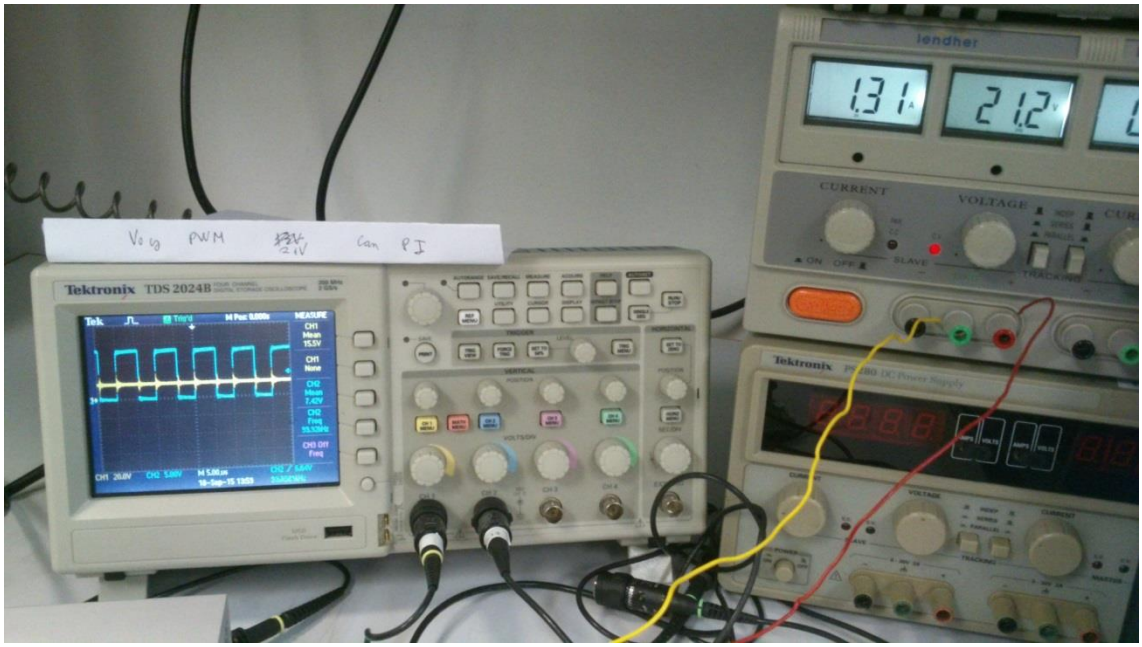
En la Figura 4-8 podemos observar la PWM y la tensión de salida cuando alimentamos el convertidor Flyback con el cargador de portátil universal de 39 W, es decir se está alimentando a 18 V.



CH1 – Tensión de salida CH2 – señal PWM

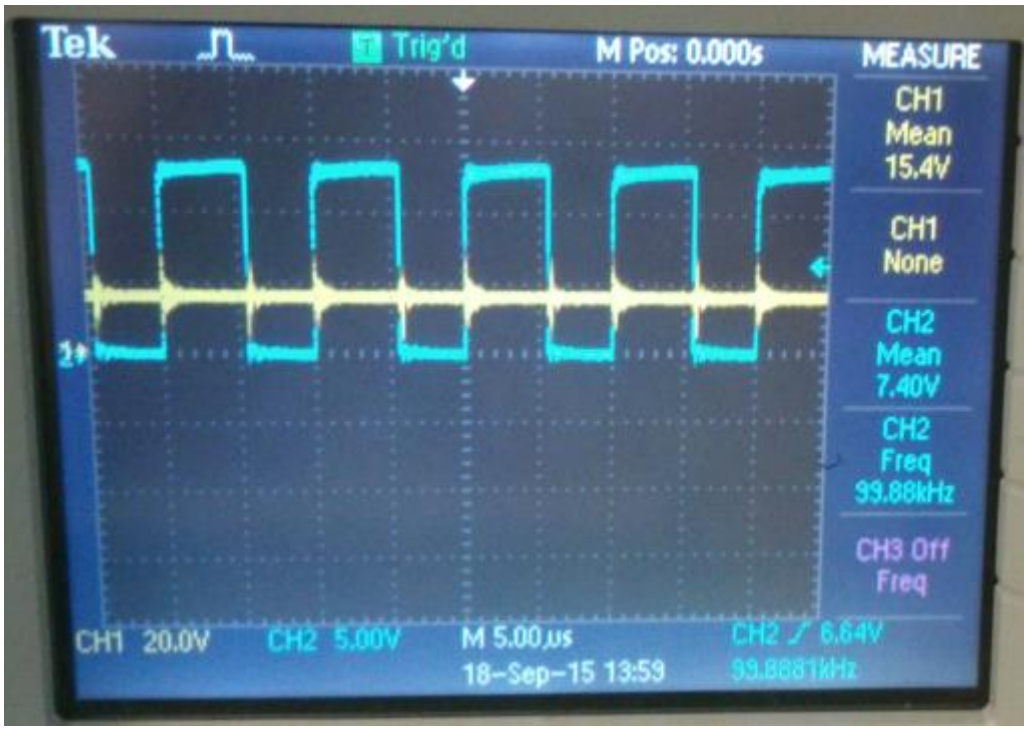
Figura 4.3.2-1 Medición de señal PWM y tensión de salida con regulador PI

Para comprobar que el regulador está cumpliendo su función se ha realizado la siguiente



prueba: se ha cortado la pista de la tensión de entrada del convertidor flyback, y en su lugar se ha soldado un cable, de forma que puede alimentarse el convertidor mediante una fuente de tensión externa. De esta forma se han realizado medidas a 21 y 13 V aproximadamente. Así conseguimos variar la tensión de entrada y comprobar que efectivamente el regulador funciona, Ver Figuras 4.3.2-2, 4.3.2-3, 4.3.2-4 y 4.3.2-5

Figura 4.3.2-2 Instalación con una tensión de entrada de 21.2 V



CH1 – Tensión de salida CH2 – señal PWM

Figura 4.3.2-3 Medición de señal PWM y tensión de salida, regulador PI, Vi=21.2 V

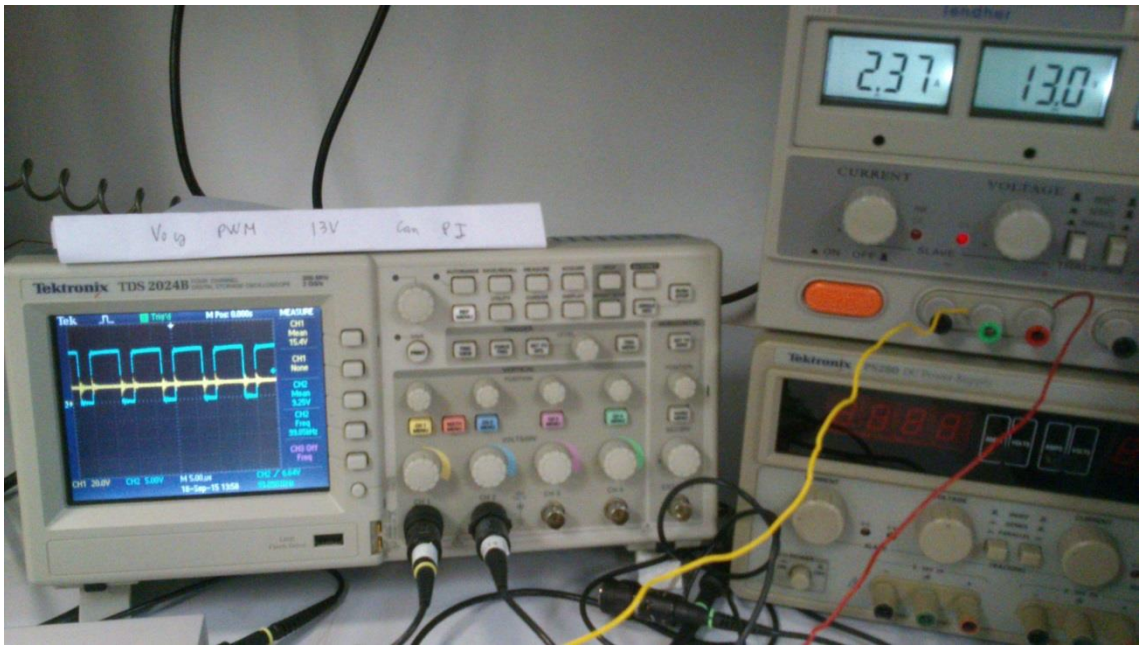
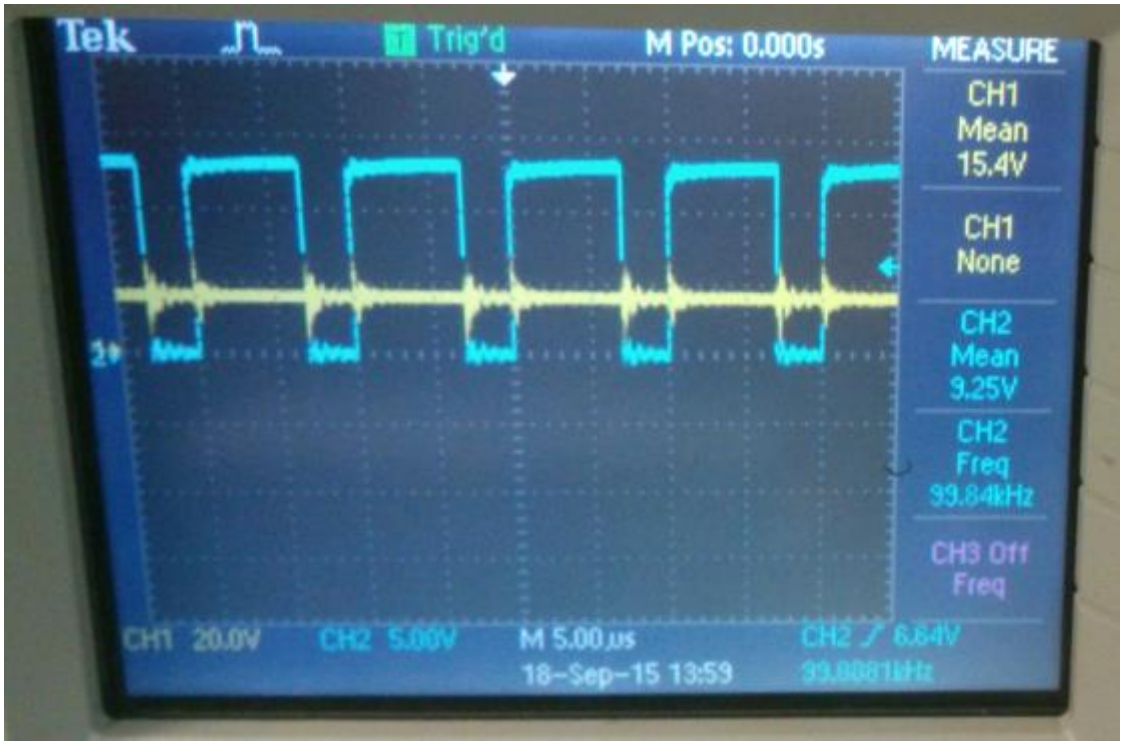


Figura 4.3.2-4 Instalación con una tensión de entrada de 13.0 V



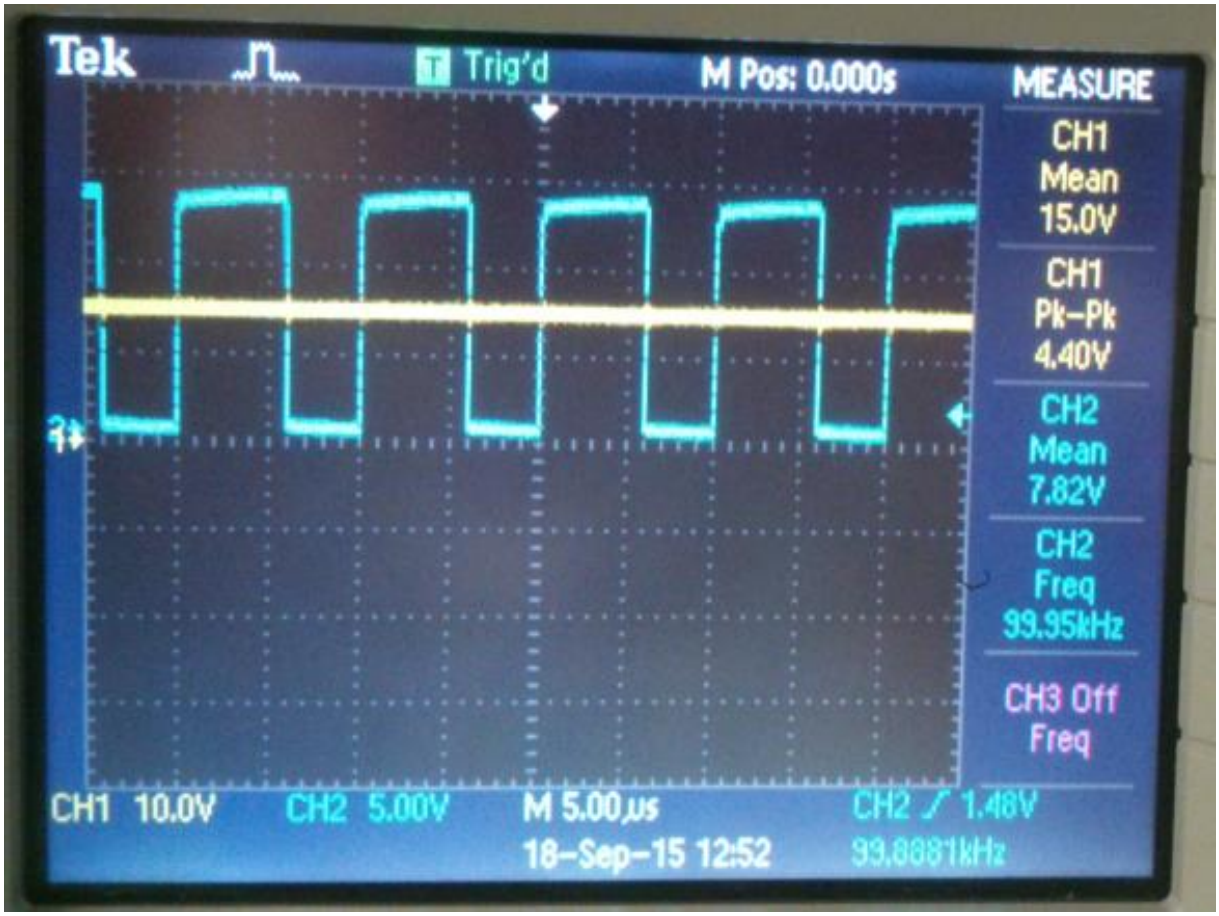
CH1 – Tensión de salida CH2 – señal PWM

Figura 4.3.2-5 Medición de señal PWM y tensión de salida, regulador PI, Vi=13.0 V

Como se puede ver en ambos casos la tensión de salida se mantiene, mientras que gracias al regulador la señal PWM se ha ajustado, demostrando así que el regulador funciona.

4.3.3. Mediciones con regulador PID

Para el regulador PID se han realizado las mismas mediciones que en el caso del regulador anterior. En la figura X el convertidor está alimentado con el cargador de portátiles usado en el caso anterior.



CH1 – Tensión de salida CH2 – señal PWM

Figura 4.3.3-1Medición de señal PWM y tensión de salida con regulador PI

Al igual que en el caso del regulador PI, es necesario comprobar que el regulador está realizando su función, por lo que igual que en el caso anterior se ha procedido a realizar varias medidas variando la tensión de entrada. Ver Figuras 4.3.3-2, 4.3.3-3, 4.3.3-4 y 4.3.3-5.

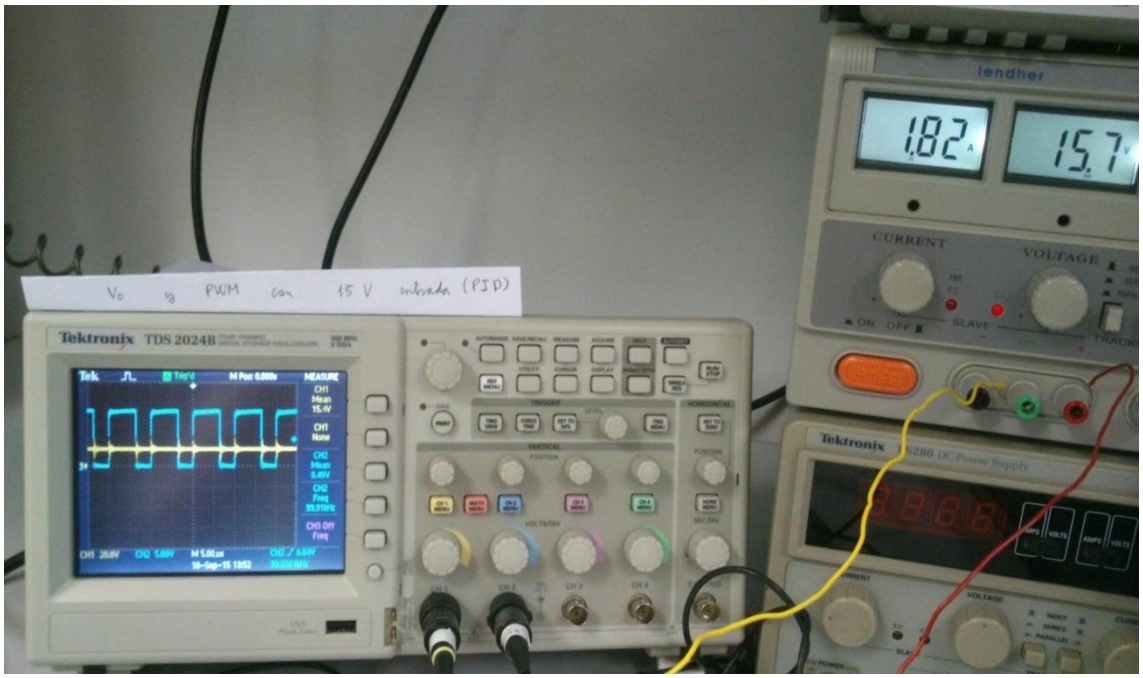
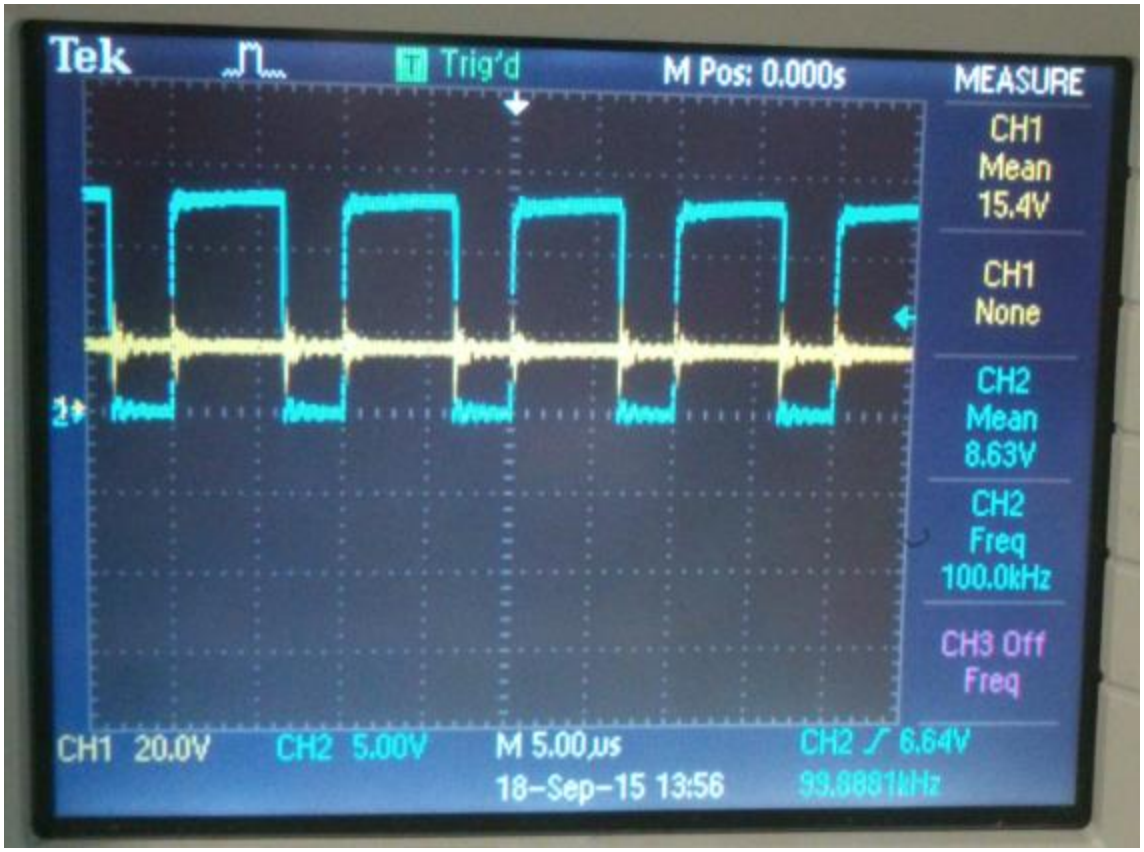


Figura 4.3.3-3 Instalación con una tensión de entrada de 15.7 V



CH1 – Tensión de salida CH2 – señal PWM

Figura 4.3.3-4 Medición de señal PWM y tensión de salida, regulador PI, Vi=13.0 V

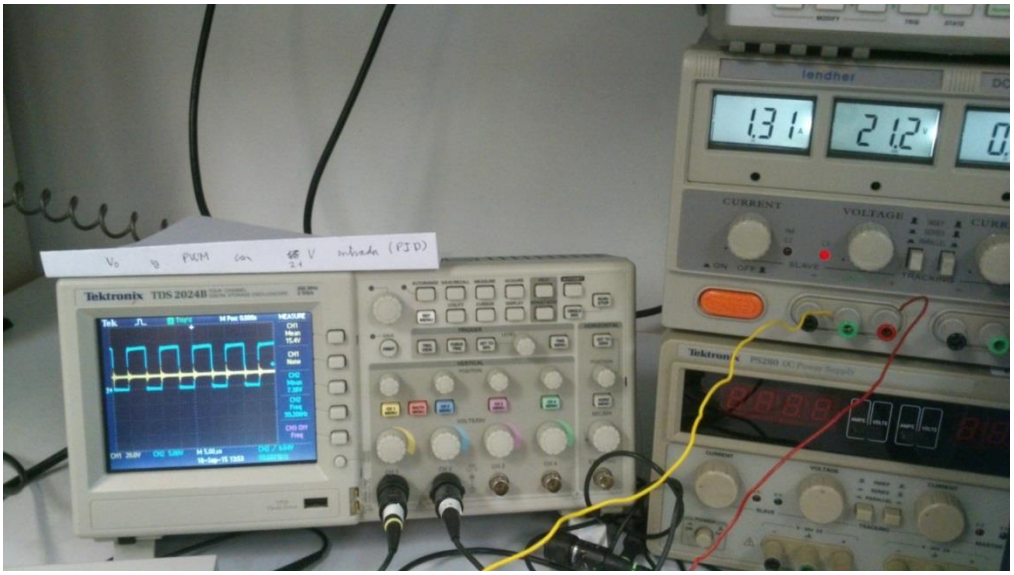
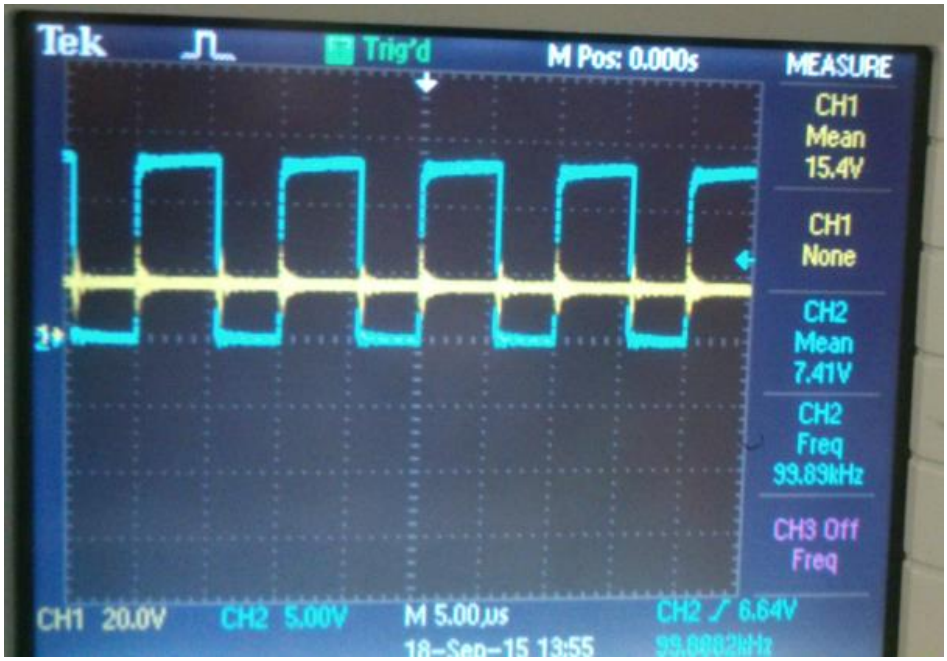


Figura 4.3.3-5 Instalación con una tensión de entrada de 15.7 V



CH1 – Tensión de salida CH2 – señal PWM

Figura 4.3.3-6 Medición de señal PWM y tensión de salida, regulador PI, Vi=13.0 V

Como podemos observar en este caso la tensión de salida se mantiene constante aunque variemos la tensión de entrada, por lo que el regulador está cumpliendo su función, ajustar la PWM para obtener así la tensión de salida deseada.

5. Conclusiones y trabajos futuros.

5.1. Introducción

En este capítulo se explicarán las conclusiones obtenidas tras la realización de diseño del sistema de control de convertidores CC-CC basado en microprocesador y reconfigurable desde un ordenador, así como los trabajos futuros que pueden realizarse tomando como base este TFG.

5.2. Conclusiones.

- Se ha procedido a realizar un Sistema de control de convertidores CC-CC, por lo que se tuvo que decidir la plataforma donde realizar las pruebas, se hizo en un convertidor diseñado por una antigua alumna en su PFC. El primer paso fue familiarizarse con dicho proyecto, y por lo tanto con el convertidor en cuestión.
- Se ha deducido las expresiones del PI y PID digitales partiendo de los reguladores analógicos, para después implementarlos en el microcontrolador STM32F4 con su plataforma de desarrollo Discovery.
- Una vez realizada la programación necesaria, se empiezan con las pruebas en el laboratorio, donde gracias a la opción Debug (permite ejecutar línea a línea el código), se encuentran y arreglan los errores que se hubieran podido cometer. Esta parte consta de lo que se denomina ensayo y error, se analiza dónde puede estar el error y se cambia la programación necesaria.
- Se programó y probó la comunicación entre el PC y el microcontrolador mediante un transductor TTL- USB.
- Se comprobó que los reguladores programados cumplían su función, para ello se realizó una modificación en la PCB del convertidor para poder variar la tensión de entrada, y se comprobó que efectivamente los reguladores funcionaban correctamente.

5.3. Trabajos futuros

A continuación se exponen una serie de trabajos que podrían realizarse en el futuro:

1- Realizar pruebas en otro convertidor CC-CC, tales como elevador , reductor, etc.
2- Introducir nuevos reguladores más complejos, como por ejemplo un regulador Tipo 2.

6. Presupuesto

6.1. Introducción

En este capítulo se expone el coste total del proyecto. El coste total se divide entre el coste de material y el coste de personal, que viene dado por las horas empleadas en el proyecto por el ingeniero, en este caso el estudiante que opta a dicho título.

6.2. Costes de personal

Los costes asociados al personal se indican en la Tabla 6-1.

Apellidos, Nombre	Categoría	Dedicación (Horas al mes)	Coste unitario	Duración (meses)
Gómez Tur, Adrián	Ingeniero	80	35	4
			Coste personal…….	11.200,00 €

Tabla 6.2-1 Resumen de los costes de personal

6.3. Coste de material

Los costes asociados al material están indicados en la tabla 6-2.

Referencia	Descripción	Cantidad	Precio Unitario	Total
RS: 734-4938P	Optoacoplador // Modelo: ACPL-312T-000E	5	4,79	23,95
POSAN	Fuente de alimentación+hembra para PCB // 39W	1	25	25
RS: 774-2827	Disipador para la carga // 12 V 50x50x100mm	1	30,56	30,56
RS:199-7854	Resistencia del snubber //1,5k Ohm 5W	2	0,276	0,552
RS:713-3930	Led verde	1	1,772	1,772
RS:262-2955	Led naranja	1	0,264	0,264
RS:522-0063	Potenciómetro multivuelta // 1k Ohm //Encapsulado T93YB	2	2,09	4,18
RS:522-0215	Potenciómetro multivuelta // 5k Ohm //Encapsulado T93YB	2	3,37	6,74
RS: 521-9710	Potenciómetro multivuelta // 20k Ohm //Encapsulado T93YB	2	3,37	6,74
RS:117-814	Condensador polipropileno 1nF // 63 V	4	1,564	6,256
RS:240-4690	Condensador polipropileno 10nF // 63 V	4	0,524	2,096
RS:157-544	Resistencia acorazada HS25 // 4,7 Ohm	2	3,08	6,16
RS:157-566	Resistencia acorazada HS25 // 22 Ohm	1	2,88	2,88
RS:720-3256	Detector de corriente // montaje superficial	2	0,576	1,152

Farnell: 1702010	Test pin PCB, 1.0 MM (paquete 100 pines)	1	18,69	18,69
Farnell: 1700858	Diodo Schottky// modelo : SB160 // 1 A	10	0,32	3,2
Farnell: 9592504	Quad buffer // Modelo: 74LV125	10	0,28	2,8
Farnell: 522636	kit aislante, TO-220	20	0,27	5,4
Farnell: 8648751	Mosfet , N , 200V, 31A, encapsulado TO-220// IRFB31N20DPBF	10	3,38	33,8
			Total (sin IVA)	182,19 €
			Total (con IVA)	220,45 €

Tabla 6.3-1 Resumen de los costes de material

6.4. Coste total

El coste total del proyecto está resumido en la tabla 6-3.

Concepto	Coste
Coste de personal	11.200,00 €
Coste de materiales	220,45 €
Total	11.420,45 €

Tabla 6.4-1 Resumen de costes totales

Bibliography

[1] Muhammad H. Rashid, “Power Electronics: Circuits Devices and its Applications”, 2003.

[2] Daniel W. Hart “Electrónica de Potencia”, 2001.

Infobae, “http://www.infobae.com/2011/11/15/1038036-el-microprocesador-cumplio-40-anos-historia”, consultada por última vez 14-agosto-2015.

ST, “UM1472”, User manual.

ST, “RM0090”, Reference manual.

Katsuhiko Ogata, “Sistemas de control en tiempo discreto”, 1996.

Firtec, “Introducción a la arquitectura ARM: ARM Cortex M4”, 2014.

Luis Moreno, “Apuntes de la asignatura Ingeniería de control II”, 2009.

Juan Vázquez y Michael G. Lorenz, “Apuntes de la asignatura Microprocesadores”, 2014.

Anexo 1: Código

En primer lugar se incluirá el código main.c y a continuación el stm32f4xx_it.c.

main.c

```
#include "stm32f4_discovery.h"
#include <stddef.h>
#include <stdint.h>
#include <stddef.h>
#include <stdio.h>
#include "stm32f4xx_it.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_rcc.h"

#define TXBUFFERSIZE    1
#define RXBUFFERSIZE    21//Longitud del vector que transmite los datos.
#define Longitud        5  //Tamaño de cada uno de los datos.

//Lista de índices de las posiciones de cada uno de los datos en el vector de transmisión.
#define indice_tipo                 0
#define indice_Vref                 1
#define indice_Proporcional         6
#define indice_Integral             11
#define indice_derivativa           16

#define ConstanteEscalado           100
#define GADC                        1365 // Ganancia del ADC 4095/3=1365

uint8_t Vector_auxilar[Longitud]; // Vector auxiliar utilizado en la función: ConvertirDatos2decimal
int div;
int Tipo;               // El tipo puede ser 1(PI) o 2 (PID)
int Vreferencia;            // Tensión de referencia (condiciona la tensión a la salida)
int Kp;                     // Constante proporcional
int Ki;                 // Constante integral
int Kd;                     //constante derivativa

int error_anterior = 0;

int i;                      // Variable indice

static volatile uint32_t aux;   //Variable indice en ConvertirDatos2decimal
static volatile uint32_t valor; //Variable indice en ConvertirDatos2decimal

int TxCounter = 0;
int RxCounter = 0;

//static volatile uint32_t valor2;
static volatile uint32_t a;
static volatile uint32_t TimingDelay;
static volatile uint32_t multiplicador[5];

static volatile int NbrOfDataToTransfer = TXBUFFERSIZE;
static volatile int NbrOfDataToRead = RXBUFFERSIZE;

#define ADC_CONVERT_RATIO  731//806    /* (3300mV / 4095)* 1000 */
#define PWM_Periodo_TIM13  840 // ver cálculo dentro de la funcion


__IO uint16_t PWM_Duty = 0;
int error = 0;
int integral = 0;
int proporcional;
int derivativo;
int PWM;
```

```
__IO uint16_t ADCdata = 0;
__IO uint32_t Valor_actual_ADC = 0;
__IO uint16_t PWM_Duty_max = 0;
__IO uint32_t VDDmV = 0;
__IO uint32_t systick;
__IO uint32_t MuestrearAhora;

/* Private typedef */
GPIO_InitTypeDef    GPIO_InitStructure;
NVIC_InitTypeDef    NVIC_InitStructure;
USART_InitTypeDef   USART_InitStructure;
RCC_ClocksTypeDef   RCC_Clocks;

TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
TIM_OCInitTypeDef  TIM_OCInitStructure;
ADC_InitTypeDef         ADC_InitStructure;


/*****************************************************************************************/
void NVIC_Config(void);
void RCC_Configuration(void);
void GPIO_Configuration(void);
void USART_Configuration(void);
void Delay(uint32_t nTime);
void TimingDelay_Decrement(void);
void USART2_IRQHandler(void);
void EnviarDatos(uint8_t TxBuffer[1]);
int ConvertirDatos2decimal(uint8_t Vector_auxilar[Longitud]);
void SysTick_Handler(void);

void ADC_Config(void);
void PWM_Config(void);
void PWM_Control(__IO uint16_t PWM_Duty);

/*****************************************************************************************/

int main(void)
{
    // Inicialización de las variables.
    multiplicador[0] = 10000;
    multiplicador[1] = 1000;
    multiplicador[2] = 100;
    multiplicador[3] = 10;
    multiplicador[4] = 1;
    aux = 0;
    valor = 0;
    a = 0;
    i = 0;
    div=0;

    systick=0;
    MuestrearAhora=0;
// Inicialización de las configuraciones de los periféricos.
  RCC_Configuration();
  GPIO_Configuration();
  NVIC_Config();
  USART_Configuration();
  ADC_Config();
  PWM_Config();
```

```
RCC_GetClocksFreq(&RCC_Clocks);
SysTick_Config(RCC_Clocks.HCLK_Frequency/500000); //500k para muestrear a 100k

//Fija el ciclo de trabajo a un valor máximo evitando así
//cualquier avería en el convertidor en casos imprevistos.
PWM_Duty_max = (uint16_t)(((uint32_t)(PWM_Periodo_TIM13*70))/ 100);

while(1)
{
    //***********Comunicaciones*********
    if (a == NbrOfDataToRead){
        // Identifica tipo de regulador: PI o PID
        Tipo=RxBuffer[indice_tipo];
        // Convierte dato de la TENSIÓN REFERENCIA -Vo-
        for(i=indice_Vref;i<(Longitud + indice_Vref);i++){
            Vector_auxilar[i-indice_Vref]=RxBuffer[i];
          }
        Vreferencia  = ConvertirDatos2decimal(Vector_auxilar);
        // Convierte dato de la constante PROPORCIONAL.
        for(i=indice_Proporcional ;i<(Longitud + indice_Proporcional );i++){
            Vector_auxilar[i-indice_Proporcional ]=RxBuffer[i];
        }
        Kp = ConvertirDatos2decimal(Vector_auxilar);
        // Convierte dato de la constante INTEGRAL.
        for(i=indice_Integral ;i<(Longitud + indice_Integral);i++){
            Vector_auxilar[i-indice_Integral]=RxBuffer[i];
        }
        Ki = ConvertirDatos2decimal(Vector_auxilar);
        // Convierte dato de la constante DERIVATIVA.
        for(i=indice_derivativa  ;i<(Longitud + indice_derivativa );i++){
            Vector_auxilar[i-indice_derivativa ]=RxBuffer[i];
        }
        Kd = ConvertirDatos2decimal(Vector_auxilar);

        a = 0;
    }
    if ( MuestrearAhora==1){


        //***********Regulador PI********
        if (Tipo == 1){ //
          MuestrearAhora=0;
          GPIO_SetBits(GPIOB,GPIO_Pin_11);
            //Lectura PC2
            ADCdata =  ADC_GetConversionValue(ADC1);
            //Conversión de 12bits del ADC a 32bits
            VDDmV = (uint32_t)ADCdata;
            error = Vreferencia - VDDmV;
            //Cálculo de la parte integral del PI
            integral = integral + Ki*error;

            //Control Anti-windup para integral
            if (integral>=(PWM_Periodo_TIM13*ConstanteEscalado*GADC))
            {
                integral = PWM_Periodo_TIM13*ConstanteEscalado*GADC;
            }
            if (integral<= 0)
            {
                integral = 0;
            }
```

```
            PWM = (Kp*error + integral);
            div=GADC*ConstanteEscalado;
            PWM = PWM / div;
            PWM_Duty = (uint16_t)PWM;

            PWM_Control( PWM_Duty);
            GPIO_ResetBits(GPIOB, GPIO_Pin_11);

             }
}

    if(Tipo==2)
    {
      GPIO_SetBits(GPIOB,GPIO_Pin_11);
      MuestrearAhora=0;
      //Lectura PC2
      ADCdata =  ADC_GetConversionValue(ADC1);
      //Conversión de 12bits del ADC a 32bits
      VDDmV = (uint32_t)ADCdata;
      error = Vreferencia - VDDmV;
      proporcional=error*Kp;
      integral=integral+(Ki*error);
      derivativo=Kd*(error-error_anterior);
      error_anterior = error;

      //Control Anti-windup para integral
      if (integral>=(PWM_Periodo_TIM13*ConstanteEscalado*GADC))
      {
          integral = PWM_Periodo_TIM13*ConstanteEscalado*GADC;
      }
       if (integral<= 0)
       {
           integral = 0;
       }
       //sumamos las tres partes del regulador
       PWM=Kp*error+integral+derivativo;
       div=GADC*ConstanteEscalado;
       PWM = PWM / div;
       PWM_Duty = (uint16_t)PWM;

       PWM_Control( PWM_Duty);
       GPIO_ResetBits(GPIOB, GPIO_Pin_11);


     }

   MuestrearAhora = 0;
}

}
```

```
//*****************************************************************************
int ConvertirDatos2decimal(uint8_t Vector_auxilar[Longitud]){
    aux = 0;
    valor = 0;
    while(aux<=Longitud){
        if(Vector_auxilar[aux]== '0'){
           valor = valor;
         }
        if(Vector_auxilar[aux]== '1'){
           valor = valor + 1*multiplicador[aux];
        }
        if(Vector_auxilar[aux]== '2'){
           valor = valor + 2*multiplicador[aux];
        }
        if(Vector_auxilar[aux]== '3'){
           valor = valor + 3*multiplicador[aux];
        }
        if(Vector_auxilar[aux]== '4'){
            valor = valor + 4*multiplicador[aux];
        }
        if(Vector_auxilar[aux]== '5'){
           valor = valor + 5*multiplicador[aux];
        }
        if(Vector_auxilar[aux]== '6'){
           valor = valor + 6*multiplicador[aux];
        }
        if(Vector_auxilar[aux]== '7'){
           valor = valor + 7*multiplicador[aux];
         }
        if(Vector_auxilar[aux]== '8'){
           valor = valor + 8*multiplicador[aux];
         }
         if(Vector_auxilar[aux]== '9'){
           valor = valor + 9*multiplicador[aux];
         }
         aux++;
    }
return valor;
}
//*****************************************************************************
void RCC_Configuration(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    /* Enable UART clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
 }
//*****************************************************************************
void GPIO_Configuration(void)
{
  // Configure USART Tx
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2|GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_OType  = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
  //GPIO_Init(GPIOB, &GPIO_InitStructure);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource2, GPIO_AF_USART2);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource3, GPIO_AF_USART2);

}

//**************************************************************************
void USART_Configuration(void)
{
  USART_InitTypeDef USART_InitStructure;

  /* USART configured as follow:
        - BaudRate = 9600 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled*/

  USART_InitStructure.USART_BaudRate = 9600;//9600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

  // USART configuration
  USART_Init(USART2, &USART_InitStructure);


  //Habilitando interrupciones por transmisión y recepción.
  USART_ITConfig(USART2, USART_IT_RXNE,ENABLE);

  // Enable the USART2
 USART_Cmd(USART2, ENABLE);

}
//**************************************************************************

void NVIC_Config(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  // Enable the USARTx Interrupt
  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority =  0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

```
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);

}
//*****************************************************************************
void TimingDelay_Decrement(void)
{

}

void ADC_Config(void)
{
  /* Enable The HSI  */
  RCC_HSICmd(ENABLE);
  /* Enable the GPIOF or GPIOA Clock */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  /* Enable ADC1 clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
   /* Configure PA.01 (ADC Channe1) in analog mode */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
  /* Check that HSI oscillator is ready */
  while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET);
```

```
  ADC_StructInit(&ADC_InitStructure);
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC1 regular channel1 configuration */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_3Cycles);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Start ADC1 Software Conversion */
  ADC_SoftwareStartConv(ADC1);

  /* Wait until ADC Channel 5 or 1 end of conversion */
  while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET)
  {
  }

}
//*****************************************************************************


void PWM_Config(void)
{
    // Configuración del PWM através del timer TIM11 y el pin GPIO_Pin_9 del puerto GPIOB

    /*---------------------- System Clocks Configuration -----------------------*/
      /* TIM13 clock enable */
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM13 , ENABLE); //Activa el reloj del Timer 13
      /* GPIOB clock enable */
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA); // Activa el reloj del GPIOA

      /* ----------------------------GPIO Configuration -----------------------------*/
      GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
      GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
      GPIO_Init(GPIOA, &GPIO_InitStructure);

      /* Remap PB9 pin to TIM11 */
      GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_TIM13);
```

```
    TIM_TimeBaseStructure.TIM_Period = PWM_Periodo_TIM13;
    TIM_TimeBaseStructure.TIM_Prescaler = 0;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    TIM_TimeBaseInit(TIM13, &TIM_TimeBaseStructure);


    /* PWM1 Mode configuration: Timer 13 Channel1 */
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 0; //33;//CCR1Val; Aquí se configura el ciclo de trabajo inicial
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

    TIM_OC1Init(TIM13, &TIM_OCInitStructure);

    TIM_OC1PreloadConfig(TIM13, TIM_OCPreload_Enable);

    TIM_ARRPreloadConfig(TIM13, ENABLE);
    /* TIM13 enable counter */
    TIM_Cmd(TIM13, ENABLE);


}
//**************************************

// void PWM_Control(uint16_t PWM_Duty)
void PWM_Control(uint16_t PWM_Duty_int)

{
    // Limitación del ciclo de trabajo maximo al 90%
    if (PWM_Duty_max < PWM_Duty_int)
   {
       PWM_Duty_int = PWM_Duty_max;
   }
    if (PWM_Duty_int <= 0 )
       {
           PWM_Duty_int = 0;
       }
   // Escribiendo el registro CCR1 se ajusta dinámicamente el ciclo de trabajo

    TIM_SetCompare1(TIM13, PWM_Duty_int);


}
```

stm32f4xx_it.c

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_it.h"

//declaración de variables
extern int systick;
extern int MuestrearAhora;
extern int a;
uint8_t  RxBuffer[21];
extern  RxCounter;

void SysTick_Handler(void)
{
    systick++;
    if(systick==5)
    {
        MuestrearAhora=1;
        systick=0;
    }

}

//interrupción de usart
//*****************************************************************************
void USART2_IRQHandler(void)
{
    if(USART_GetITStatus (USART2, USART_IT_RXNE)    != RESET)
    {
    RxBuffer[RxCounter] = ((uint8 t) USART ReceiveData(USART2));

    RxCounter++;

    if(RxCounter == 21)
        {
           RxCounter = 0;
        }
    USART_ClearITPendingBit(USART2,USART_IT_RXNE);
    }

}
//*****************************************************************************
```