

SISTEMA CON ADC, PWM Y UART

Stellaris Lunchpad LM4F120

Descripción breve

Reporte del desarrollo de un sistema que hace uso de los recursos de ADC, PWM y UART para comunicación serial haciendo el control de la intensidad de luminosidad de un LED controlado desde la entrada del ADC y el valor asignado por el usuario por medio de la comunicación serial.

Juan Miguel Vargas Sánchez
jmvarsan@gmail.com

Convertidor Analógico - Digital con Salida a un PWM y Comunicación Serial

Este documento describe el funcionamiento de un sistema básico desarrollado en una tarjeta Stellaris LM4F120 de Texas Instruments, en el cual se hace uso de tres recursos como son el módulo ADC, comunicación serial por medio de un módulo UART y PWM, cabe mencionar que para este ultimo la tarjeta no cuenta con hardware dedicado para generar la señal PWM por lo tanto no es posible usar los métodos disponibles en la “driver library” por lo que se configuran los timers 0 y timer 1 para generarlo.

La modulación por ancho de pulso (mejor conocida como PWM) de una señal es una técnica para manejar un señal periódica modificando su ciclo de trabajo, ya sea para enviar información a través de un canal o para controlar el nivel de energía que se envía en una carga, el ciclo de trabajo está definido por la siguiente ecuación.

$$D = \frac{\tau}{T}$$

Donde

- D es el ciclo de trabajo.
- τ es el tiempo en que la función es positiva.
- T es el periodo de la función.

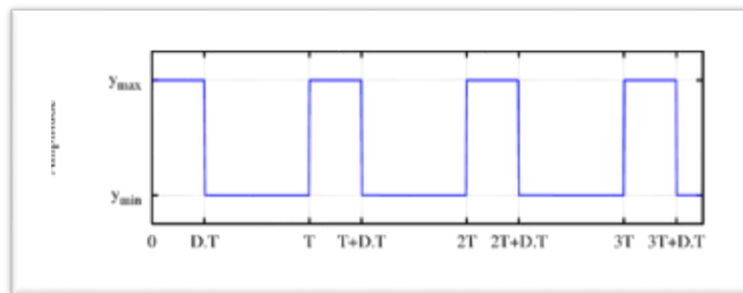


Figura 1. Señal PWM de una onda cuadrada.

La conversión de una señal analógica a digital (ADC por sus siglas en inglés) se consigue a través del muestreo de una señal analógica para así poder tratar los datos de esta de una manera más fácil, la tarjeta LM4F120 cuenta con dos módulos de ADC (ADC0 y ADC1).

La comunicación serial, en telecomunicaciones e informática es el proceso de envío de datos de un bit a la vez, de forma secuencial, sobre un canal de comunicación o un bus, la tarjeta LM4F120 cuenta con 8 módulos de UART el cual permite crear este tipo de comunicación entre la tarjeta y otro dispositivo.

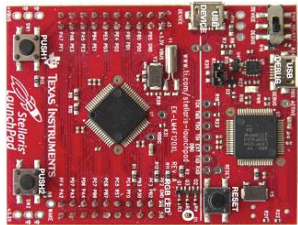


DESARROLLO

El sistema descrito en el presente documento está conformado por tres módulos principales como ya se mencionó anteriormente, su funcionamiento consiste en lo siguiente, en el pin de entrada del ADC se inyecta una voltaje de corriente directa que varía entre 0 v y 3.3 v esto con ayuda de un preset que funciona como divisor de voltaje, el resultado de la conversión del ADC es recibido en el handler de interrupciones asignado al módulo ADC el cual es tratado para obtener el ciclo de trabajo del PWM y se le asigna a este último, el resultado del PWM se ve reflejado en el LED de la tarjeta pues dependiendo del ciclo de trabajo que se le asigne varia la intensidad a la que se encenderá el LED.

Por otro lado con el uso de la comunicación serial se puede indicar el ciclo de trabajo al que se requiere que trabaje el PWM, desde la terminal de comunicación serial se indica el ciclo de trabajo el cual puede variar entre 0% y 100%, este es enviado hacia la tarjeta LM4F120, enseguida el dato es recibido por el handler de interrupciones asignado al módulo UART donde dicho dato es tratado y finalmente asignado al PWM, el resultado final se ve reflejado en el LED de la tarjeta como ya se mencionó.

Hardware y software

El hardware utilizado durante la práctica se muestra en la siguiente tabla:

Stellaris LM4F120	
USB to TTL D-SUN-V3.0	
Preset 10Kohm	

Cabe mencionar que para la comunicación serial se utiliza el USB to TTL, esto para una implementación rápida y así evitar el uso de un cable serial y el circuito integrado MAX232.

El software utilizado en esta práctica fue el Hercules el cual sirve para hacer el envío y recepción de datos por comunicación serial entre la pc y la tarjeta LM4F120.

Conexiones

Las conexiones entre las tarjetas se listan a continuación:

Conexión	LM4F120
Pin de entrada ADC	PE3
Gnd	Gnd
corriente	+3.3 v

Tabla 1. Conexiones entre Maestro y esclavo.

Conexión	LM4F120	MSP430
Ground	GND	GND
Transmisión (lm4f120)	PC5 (Tx)	RXD
Recepción (lm4f120)	PC6 (Rx)	TXD

Tabla 2. Conexión entre lm4f120 y TTL.

A continuación se muestra el diagrama del sistema

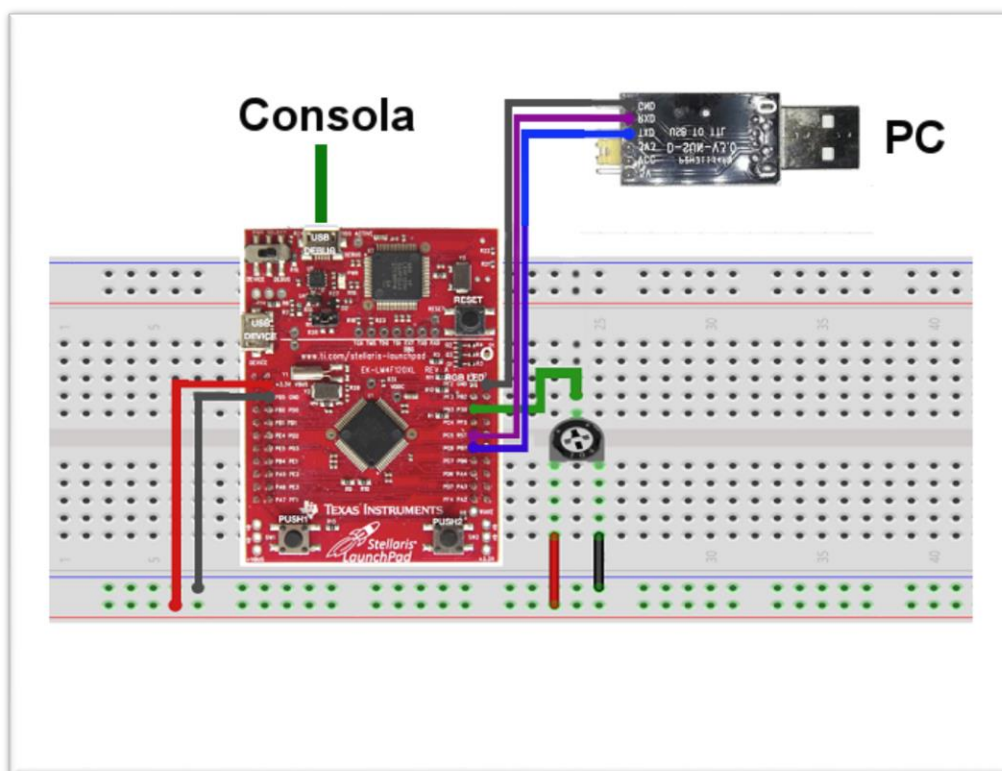


Figura 2. Sistema con módulos ADC, PWM y UART.

El sistema físico final se muestra en la siguiente figura

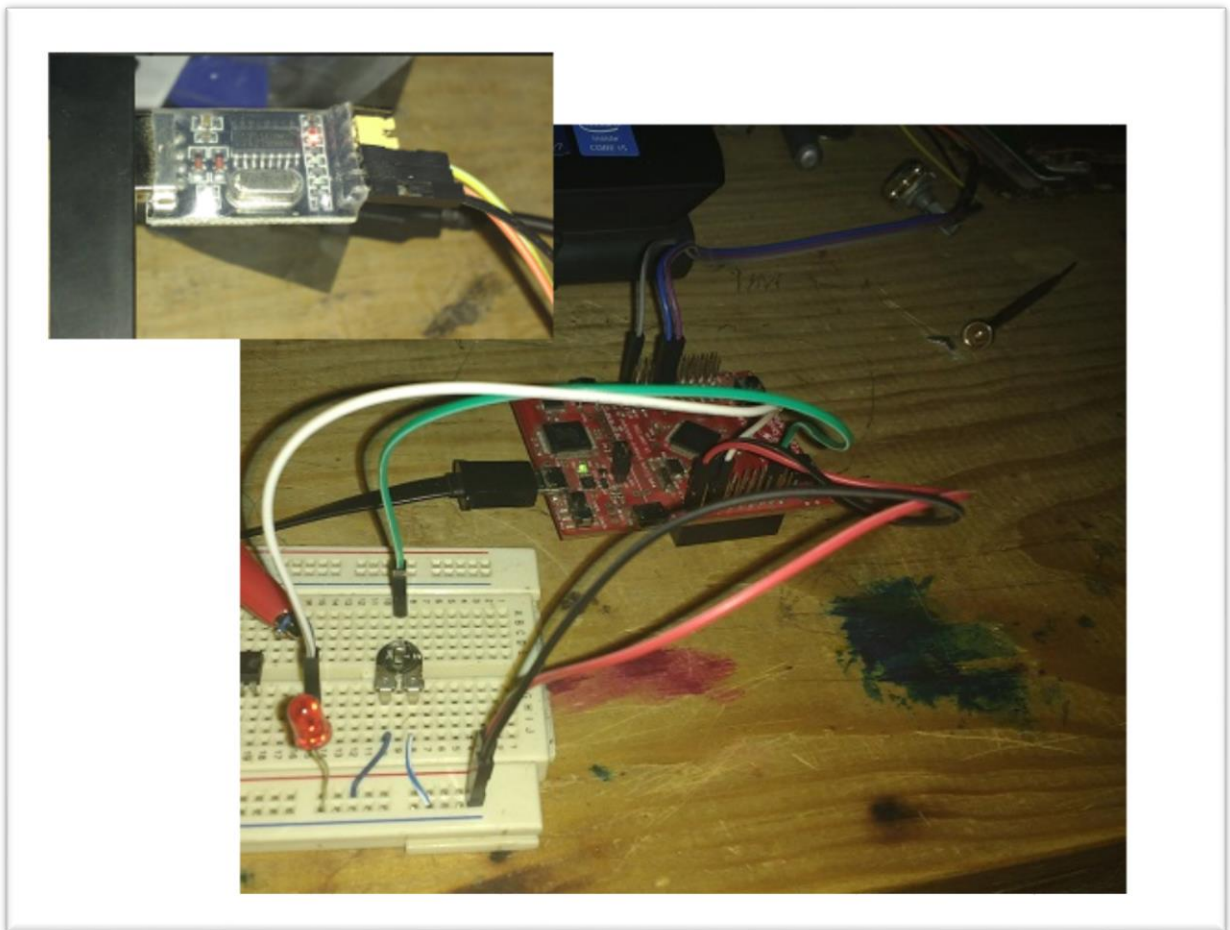


Figura 3. Sistema físico ADC_PWM_UART.

A continuación se mostrarán los resultados del funcionamiento del sistema desarrollado en este documento, cabe mencionar que en las terminales de la comunicación serial se usan para visualizar el log de resultado de las tareas que va ejecutando el sistema.

Al iniciarse el sistema se despliegan logs de información de que el sistema ha iniciado correctamente junto con instrucciones para el uso de la comunicación serial.

En la siguiente figura se muestran dos consolas, la de la izquierda es la consola de información de la comunicación serial entre la PC y la tarjeta y la de la derecha es la consola que despliega los logs de información general durante la ejecución.

En la consola de la derecha se puede ver los resultados de la conversión de la señal de entras del ADC, se muestra que se está asignando un ciclo de trabajo del 23% el cual equivale aproximadamente a 774 mV de entrada con un leve variación.

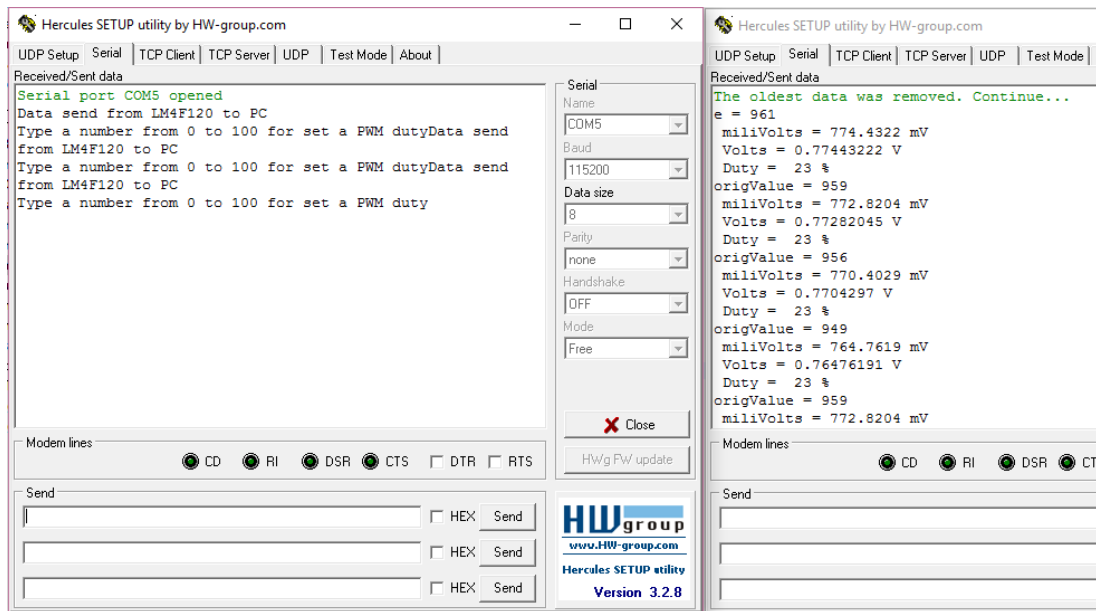


Figura 4. A la izquierda consola de comunicación serial pc -lm4f120 y a la derecha consola de logs del sistema mostrando el clico de trabajo del 23% asignado al PWM.

En la siguiente figura se puede ver los resultados de la conversión de la señal de entras del ADC, se muestra que se está asignando un ciclo de trabajo del 49% el cual equivale aproximadamente a 1.64 V de entrada con un leve variación.

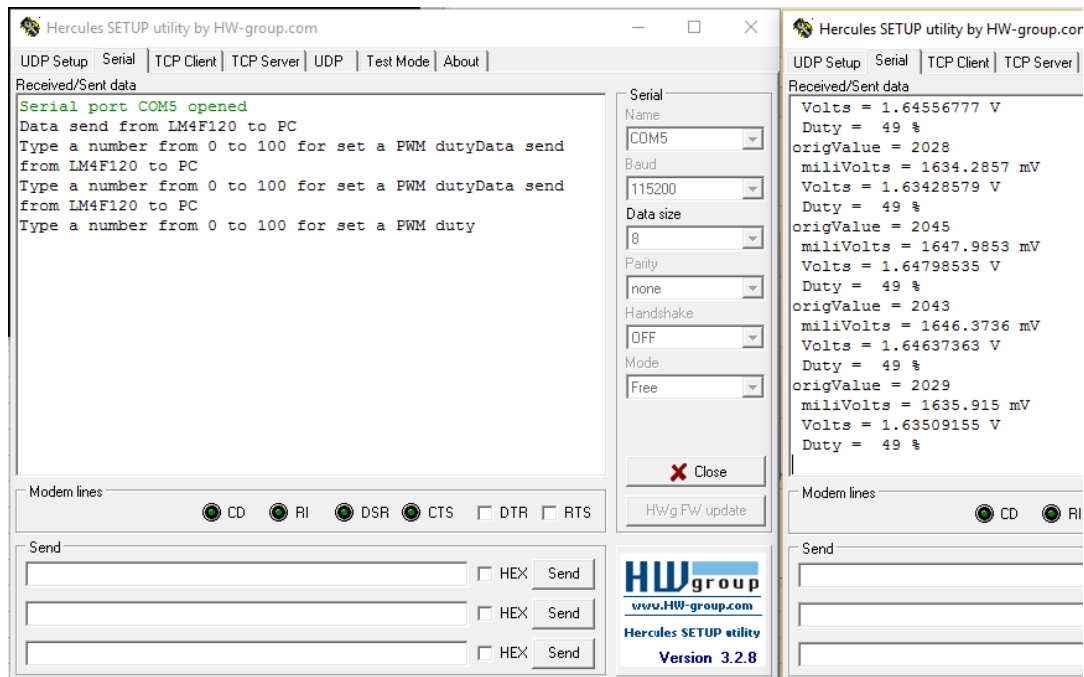


Figura 5. A la izquierda consola de comunicación serial pc -lm4f120 y a la derecha consola de logs del sistema mostrando el clico de trabajo del 49% asignado al PWM.

En la siguiente figura se puede ver los resultados de la conversión de la señal de entras del ADC, se muestra que se está asignando un ciclo de trabajo del 75% el cual equivale aproximadamente a 2.51 V de entrada con un leve variación.

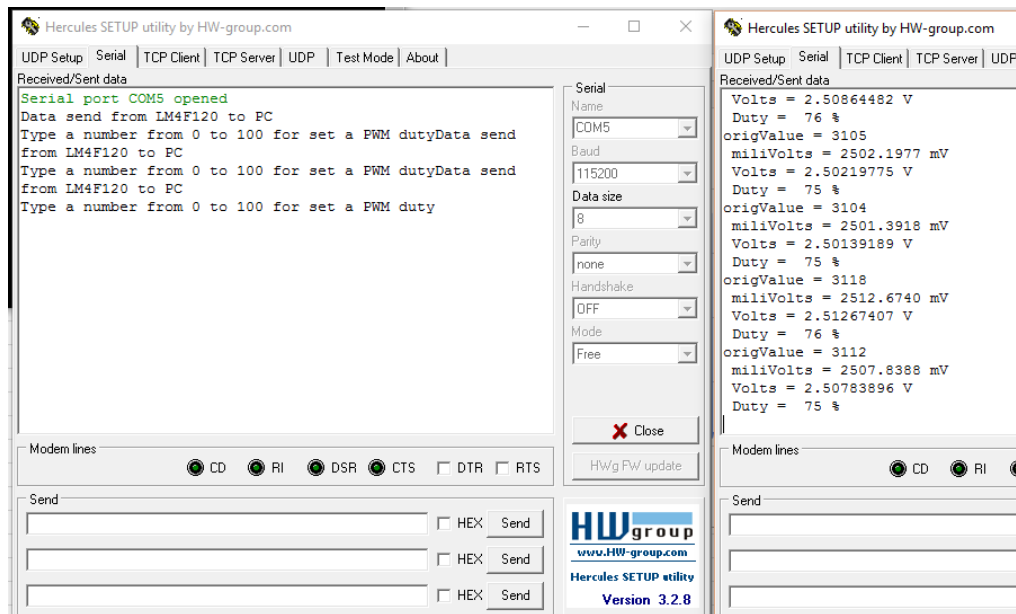


Figura 6. A la izquierda consola de comunicación serial pc-lm4f120 y a la derecha consola de logs del sistema mostrando el clcio de trabajo del 75% asignado al PWM.

En la siguiente figura se puede ver los resultados de la conversión de la señal de entras del ADC, se muestra que se está asignando un ciclo de trabajo del 99% el cual equivale aproximadamente a 3.29 V de entrada con un leve variación.

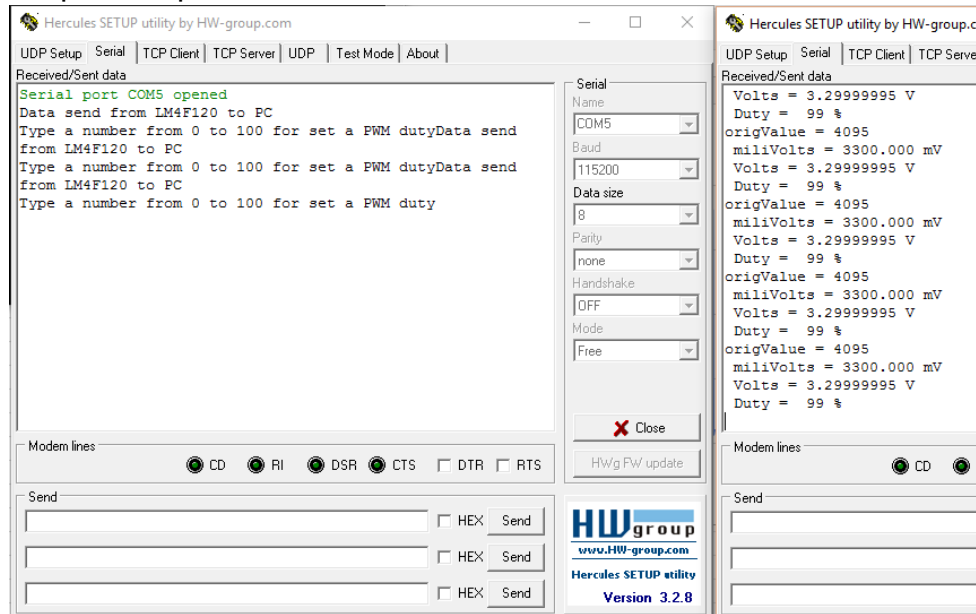


Figura 7. A la izquierda consola de comunicación serial pc-lm4f120 y a la derecha consola de logs del sistema mostrando el clcio de trabajo del 99% asignado al PWM.

Por ultimo desde la pc se envía el valor del ciclo de trabajo que se desea asignar al PWM desde la consola de comunicación serial en esta prueba se asignó el 50%.

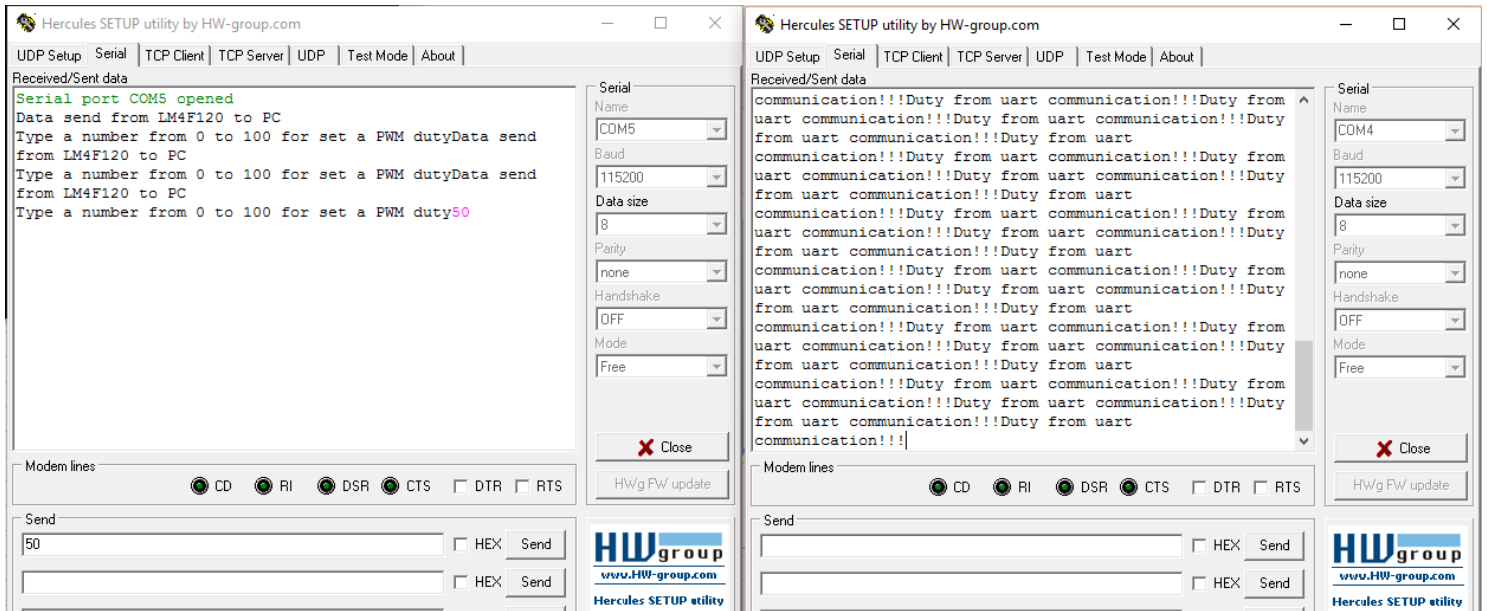


Ilustración 8. A la izquierda la consola de comunicación serial mostrando el ciclo de trabajo asignado al PWM por el usuario y a la derecha la consola de logs indicando que el ciclo de trabajo ha sido configurado desde la comincacion serial.

El resultado final del ciclo de trabajo asignado es mostrado en la siguiente figura.

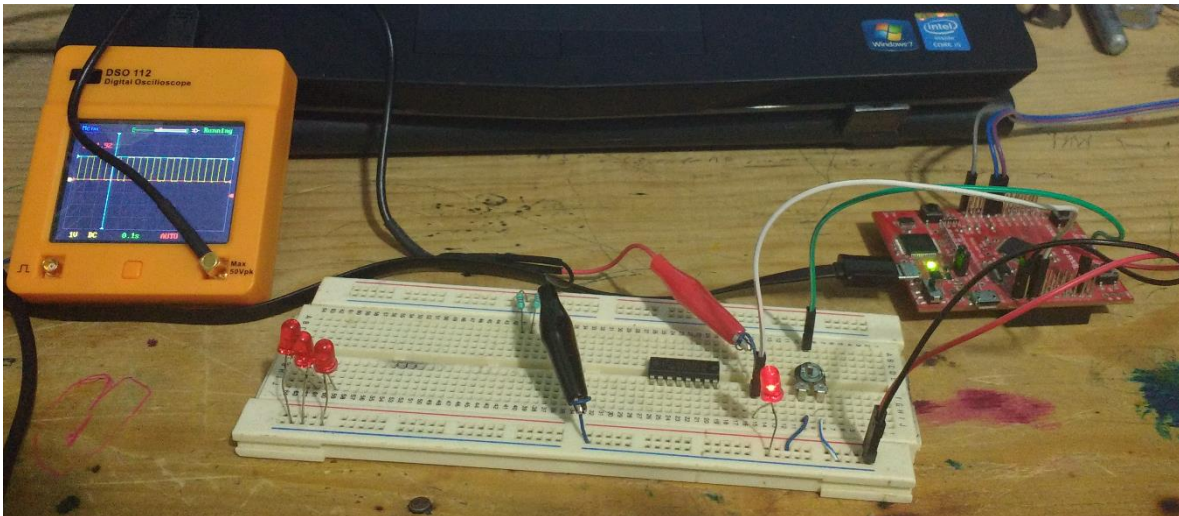


Ilustración 9. Senal PWM de salida con un ciclo de trabajo del 50%

Código del sistema ADC_PWM_UART

Primeramente se definen las librerías y variables globales necesarias para los módulos y funciones del programa.

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "utils/uartstdio.h"
#include "driverlib/adc.h"
#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"

#define RED 0

unsigned long ulADC0Value[1];
volatile float ulmV;
volatile float ulV;

unsigned long ulPeriod, ulDutyR, ulDutyExt;
unsigned int uiOn = RED;
unsigned char charGet;
int uartEnable_ = 0;
int UARTFire = 0;
float dutyValue;
```

Método InitConsole

En este método se hace una configuración básica de la uart 0, donde dicho modulo hace la función de una consola para mostrar logs de información durante la ejecución del sistema haciendo uso de la librería uartstdio.

```
void InitConsole(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);

    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTStdioInit(0);
}
```

Método blinked

Como su nombre lo dice este método sirve para ejecutar un parpadeo en el led rojo de la tarjeta stellaris, el cual sirve para notificar al usuario cuando se ejecutó alguna acción en el sistema.

```
void blinked(void)
{
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
    SysCtlDelay(SysCtlClockGet()/75);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
}
```

Método InitUARTComm

Este método hace la configuración necesaria para usar el módulo UART 1 de la tarjeta Stellaris, en el cual se habilitan periféricos y pines a utilizar, así como la configuración para la comunicación serial y habilitación de interrupciones.

```
Void InitUARTComm(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);

    GPIOPinConfigure(GPIO_PC4_U1RX);
    GPIOPinConfigure(GPIO_PC5_U1TX);
    GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5);

    UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 115200,
UART_CONFIG_WLEN_8|UART_CONFIG_PAR_NONE|UART_CONFIG_STOP_ONE);

    IntEnable(INT_UART1);
    UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);
}
```

Método sendDataUART

Este método sirve para hacer el envío de datos desde la tarjeta Stellaris hacia la PC por medio del módulo UART1.

```
void sendDataUART(unsigned char *data){
    while(UARTBusy(UART1_BASE));
    while(*data != '\0')
        UARTCharPut(UART1_BASE, *data++);
    Blinked
}
```

Método UARTIntHandler

El método UARTIntHandler es el handler que se dispara cuando una interrupción es generada al momento de recibir datos de la PC en la tarjeta Stellaris a través

de la comunicación serial, cabe mencionar que en este handler se espera recibir un numero en el rango 0 - 100 el cual es considerado en este caso como el ciclo de trabajo a establecer al PWM.

Cabe mencionar que el número recibido es un arreglo de caracteres el cual es almacenado en la variable `uartDuty`, posteriormente se le hace un corrimiento al arreglo para hacer la conversión de una cadena a un número entero y después a un porcentaje. Después de la conversión el valor obtenido es establecido al timer el cual simula un pwm.

```
Void UARTIntHandler(void)
{
    unsigned long ulStatus;
    int i = 0;
    int j = 0;
    char c = 0;
    int x = 0;
    int dutyValueTemp = 0;
    unsigned char uartDuty[3] = {'\0'};

    UARTFire ++;
    ulStatus = UARTIntStatus(UART1_BASE, true);
    UARTIntClear(UART1_BASE, ulStatus);
    while(UARTCharsAvail(UART1_BASE))
    {
        charGet = UARTCharGetNonBlocking(UART1_BASE);
        uartDuty[i]= charGet;
        i++;

        blinkled();
    }
    for(j = i; j > 0; j--){
        if(j == 3){
            c = uartDuty[i-j];
            x = (c - '0')*100;
        }
        if(j == 2){
            c = uartDuty[i-j];
            x = (c - '0')*10;
        }
        if(j == 1){
            c = uartDuty[i-j];
            x = c - '0';
        }

        dutyValueTemp += x;
    }
    dutyValue = (99 - dutyValueTemp) * 0.01;
    uartEnable_ = 1;
    ulDutyExt = (unsigned long)(ulPeriod-1)*dutyValue;
    if(UARTFire == 1){
        TimerMatchSet(TIMER2_BASE, TIMER_A, ulDutyExt);
    }else if(UARTFire == 2){
        UARTFire = 0;
    }
}
```

Como parte de la configuración del proyecto es necesario hacer la configuración del handler UARTIntHandler en el archivo "lm4f120h5qr_startup_ccs.c", agregando las siguientes líneas:

```
#include <stdint.h>

.
.
.

// External declaration for the reset handler that is to be called when the
// processor is started
//
//*****
extern void _c_int00(void);
extern void UARTIntHandler(void);

.
.
.

IntDefaultHandler,           // GPIO Port E
IntDefaultHandler,           // UART0 Rx and Tx
    UARTIntHandler,          // UART1 Rx and Tx
.
.
.
```

Función initPWMTimer

La función que se muestra a continuación se encarga de configurar el timer para los timer A y B para simular un señal PWM, los metodos usados para la configuracion son los siguientes:

- **SysCtlPeripheralEnable:** Habilita el periférico del puerto B.
- **SysCtlPeripheralEnable:** Habilita el Timer 2 de la tarjeta.
- **GPIOPinTypeGPIOOutput:** Configura el pin 0 del puerto b como salida.
- **GPIOPinWrite:** Inicializa el pin de salida.
- **GPIOPinConfigure:** Establece al puerto B como salida para un Timer CCP (Capture Compare PWM).
- **GPIOPinTypeTimer:** Se indica el puerto de salida de la señal PWM generada.
- **TimerConfigure:** Se establece el modo de operación del Timer 2.
 - **TIMER_CFG_SPLIT_PAIR:** Dos timers de media onda cada uno.
 - **TIMER_CFG_A_PWM:** Salida PWM de media onda.
- **ulPeriod:** Se define el periodo de la señal de salida.
- **ulDutyExt:** Se define el ciclo de trabajo inicial del PWM en 50%.
- **TimerLoadSet:** Estable el periodo para la señal de salida.
- **TimerMatchSet:** Establece el ciclo de trabajo del PWM.
- **TimerEnable:** Habilita el Timer 2.

```

void initPWMTimer(void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);

    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0x00);
    GPIOPinConfigure(GPIO_PB0_T2CCP0);
    GPIOPinTypeTimer(GPIO_PORTB_BASE, GPIO_PIN_0);

    TimerConfigure(TIMER2_BASE, TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_PWM);

    ulPeriod = (SysCtlClockGet() / 5000)/2;
    ulDutyExt = (unsigned long)(ulPeriod-1)*0.5;

    TimerLoadSet(TIMER2_BASE, TIMER_A, ulPeriod-1);
    TimerMatchSet(TIMER2_BASE, TIMER_A, ulDutyExt);
    TimerEnable(TIMER2_BASE, TIMER_A);
}

```

Función initADCModule

La función initADCModule se encarga de inicializar el módulo ADC de la tarjeta Stellaris LM4F120, a continuación se muestran los métodos utilizados para hacer dicha inicialización.

- **SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0):** Habilita el módulo ADC 0 de la tarjeta.
- **SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS):** Establece una tasa de muestreo de 250 Kilo-Muestras.
- **SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE):** Habilita el periférico del puerto E.
- **GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3):** Establece el pin 3 del puerto E como pin de entrada para la señal analógica.
- **ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0):** Se establece la secuencia de muestras con las siguientes opciones:
 - 3: captura una muestra simple.
 - ADC_TRIGGER_PROCESSOR: al ejecutar ADCProcessorTrigger() se inicia la conversión del dato/s capturado.
 - 0: Se establece una prioridad alta con respecto a otras secuencias de muestras configuradas en caso de que las hubiera.
- **ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE | ADC_CTL_END):** Configura un paso de la secuencia de muestreo con las siguientes opciones.
 - ADC_CTL_CH0 configura al canal de muestra 0 en modo de una sola terminación o single-ended (default).

- ADC_CTL_IE configura la bandera de interrupcion que sera establecida cuando la muestra esté completa.
- ADC_CTL_END configura la lógica del ADC con lo cual se indica que sera la ultima conversión en secuencia 3.
- **ADCSequenceEnable(ADC0_BASE, 3):** Habilita la secuencia de muestra 3.
- **ADCIntClear(ADC0_BASE, 3):** Limpia el estado de la bandera de interrupción antes de empezar el muestreo de la senal analogica.

```
void initADCModule (void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);

    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE | ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3);

    ADCIntClear(ADC0_BASE, 3);
}
```

Función getDataAdc

La funcion getDataAdc se encarga de obtener los datos obtenidos de la entrada analogica en cada muestreo, cabe mencionar que cada que se obtiene una nueva muestra de la entrada analogica este dato es tratado para obtener el ciclo de trabajo que se estableciera al PWM, a continuacion se muestra la descripción de los metodos utilizados para su funcionamiento.

- **ADCProcessorTrigger(ADC0_BASE, 3):** Inicia la obtencion de muestras de la senal analogica.
- **ADCIntStatus(ADC0_BASE, 3, false):** Obtiene el estado de la interrupcion.
- **ADCIntClear(ADC0_BASE, 3):** Limpia la interrupcion.
- **ADCSequenceDataGet(ADC0_BASE, 3, uIADC0Value):** Obtiene el dato capturado por la secuencia de muestreo y lo almacena en la variable "uIADC0Value".
- **TimerMatchSet(TIMER2_BASE, TIMER_A, uIDutyExt):** Establece el ciclo de trabajo del PWM.

```
void getDataAdc ()
{
    int dutyValueTemp = 0;
    int dutyValuePrint = 0;

    ADCProcessorTrigger(ADC0_BASE, 3);

    while(!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }
    ADCIntClear(ADC0_BASE, 3);
}
```

```

ADCSequenceDataGet(ADC0_BASE, 3, ulADC0Value);

ulmV = ((ulADC0Value[0] * 3.3)/4095)*1000;
ulV = ulmV/1000;

int ulmVEnt1 = ulmV;
float ulmVFrac = ulmV - ulmVEnt1;
int ulmVEnt2 = ulmVFrac * 10000;
int ulVEnt1 = ulV;
float ulVFrac2 = ulV - ulVEnt1;
int ulVEnt2 = ulVFrac2 * 10000;
float ulVFrac3 = (ulVFrac2 * 10000) - ulVEnt2;
int ulVEnt3 = ulVFrac3 * 10000;

dutyValuePrint = ((100 * ulADC0Value[0])/4096);
dutyValueTemp = 99 - ((100 * ulADC0Value[0])/4096);
dutyValue = dutyValueTemp * 0.01;

ulDutyExt = (unsigned long)(ulPeriod-1)*dutyValue;

TimerMatchSet(TIMER2_BASE, TIMER_A, ulDutyExt);

UARTprintf("origValue = %3d \n miliVolts = %d.%03d mV\n Volts = %d.%03d%03d V\n
Duty = %3d %%\n", ulADC0Value[0], ulmVEnt1, ulmVEnt2, ulVEnt1, ulVEnt2, ulVEnt3,
dutyValuePrint);
}

```

Programa principal del sistema

En el programa principal se hace la configuracion de la frecuencia del reloj principal a utilizar asi como de todos los modulos del sistema y los led indicadores de la tarjeta, tambien se inicia el programa ejecutando la funcion que obtiene los datos de la senal analogica, a continuacion se describen algunos de estos metodos.

- **SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN):** Configura la frecuencia del reloj principal en 40 MHz.
- **SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF):** Habilita el puerto F de la tarjeta.
- **GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2|GPIO_PIN_1):** Configura los pines 1 y 2 del puerto F como perifericos de salida.
- **InitConsole():** Ejecuta la funcion que configura la consola para el despliegue de logs.
- **InitUARTComm():** Ejecuta la funcion que configura el modulo UART 1 para la comunicaci3n serial.
- **initPWMTimer():** Ejecuta la funcion que configura el timer en modo PWM.
- **initADCModule():** Ejecuta la funcion que configura el modulo ADC 0 de la tarjeta.
- **IntMasterEnable():** Habilita el interruptor maestro de interrupciones.

```

int main(void) {
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2|GPIO_PIN_1);

    InitConsole();
    UARTprintf("UARTPrintf inicializada.\n");

    InitUARTComm();
    UARTprintf("Módulo UART inicializado.\n");

    initPWMTimer();
    UARTprintf("PWM inicializado.");

    initADCModule();
    UARTprintf("ADC incializado.");

    UARTprintf("Prueba de PWM y ADC ->\n");
    UARTprintf(" Tipo: Pwm controlado por una entrada adc\n");
    UARTprintf(" Entrada: Senal analogica de 3.3 V\n\n");

    IntMasterEnable();

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);
    SysCtlDelay(SysCtlClockGet() / 150);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
    UARTprintf("Led indicador en la tarjeta LM4F120 configurado.\n");

    UARTprintf("Sistema iniciado con éxito.\n");

    unsigned char *buf = "Dato enviado desde LM4F120 a la PC\n\r";

    sendDataUART(buf);
    while(UARTBusy(UART1_BASE));

    buf = "Envie un numero del 0 al 100 para establecer el ciclo de trabajo del PWM\n";
    sendDataUART(buf);

    while(1)
    {
        if(uartEnable_ == 1)
        {
            UARTprintf("Ciclo de trabajo establecido desde la PC. \n");
        }
        else
        {
            getDataAdc();
        }
        SysCtlDelay(SysCtlClockGet() / 12);
    }
}

```