Electrónica Digital lll

Implementación de un Osciloscopio Digital en la Placa LPC1769

Facultad de Ciencias Exactas, Físicas y Naturales
Universidad Nacional de Córdoba

Alumnos:

- Guadalupe Vega
- Guillermo Rubén Darío Zúñiga

Profesor:

- Ing. Gallardo

2024

Introducción

El proyecto consiste en el diseño e implementación de un osciloscopio digital utilizando el microcontrolador **LPC1769**, el cual está integrado por un ARM Cortex-M3. Este sistema es capaz de capturar señales analógicas provenientes de un generador de señales, procesarlas digitalmente y enviarlas a una computadora mediante el protocolo UART para su análisis. Además, el sistema incluye una salida reconstruida de las señales procesadas mediante un DAC (Convertidor Digital-Analógico), que puede ser visualizada en un osciloscopio físico.

Este informe detalla el diseño, configuración y funcionamiento del sistema, con énfasis en el uso de periféricos integrados como el ADC (Convertidor Analógico-Digital), DAC, UART y DMA (Acceso Directo a Memoria). También se describen las operaciones de procesamiento de señales realizadas en tiempo real, así como la interacción con el usuario mediante comandos.

Descripción del Sistema

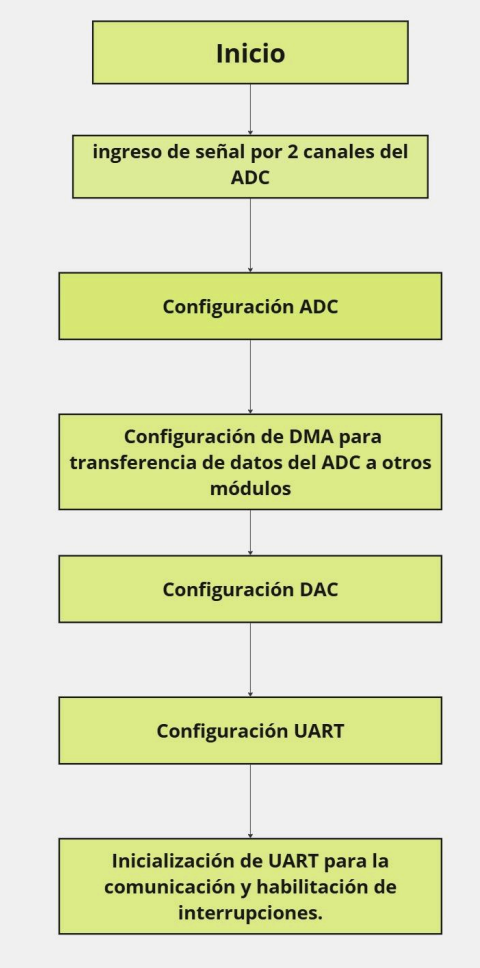
El sistema se basa en las principales configuraciones:

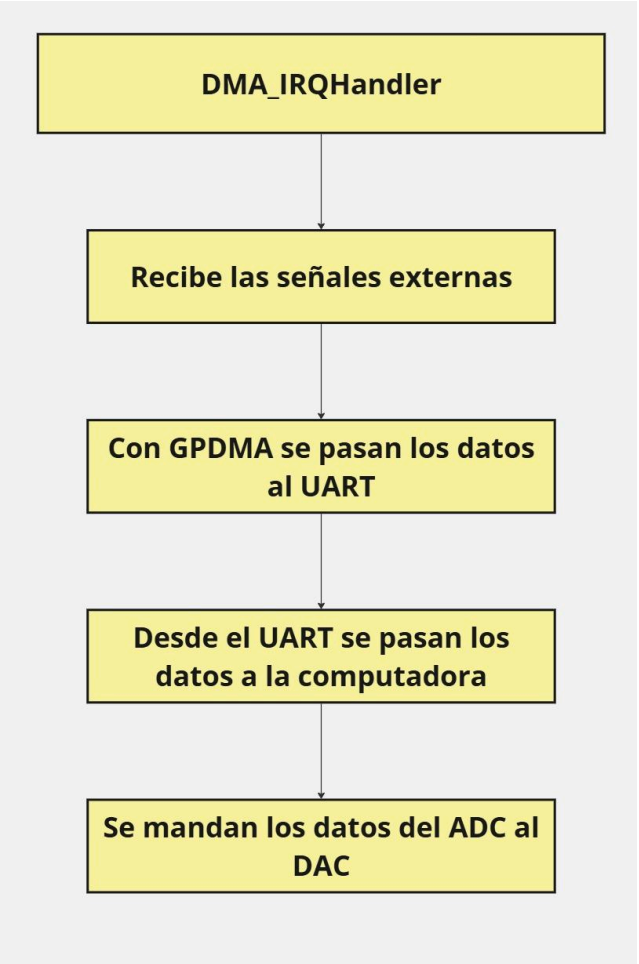
1. **Entradas analógicas**: Dos canales de entrada del ADC (ADC1 y ADC2) conectados a un generador de señales.
2. **Procesamiento digital**: Realización de operaciones básicas (suma y multiplicación) entre las señales capturadas.
3. **Transmisión de datos**:
    - Los datos del ADC son enviados a la computadora mediante UART para su análisis en un software externo.
    - Simultáneamente, los datos procesados son enviados al DAC para reconstrucción de señales.
4. **Control por comandos**: Mediante un terminal en la computadora, el usuario puede:
    - Seleccionar una señal específica (canal 1 o canal 2).
    - Realizar operaciones matemáticas (suma y multiplicación).
    - Se conoce la operación seleccionada mediante indicadores LED.
5. **Visualización**:
    - Señales reconstruidas mediante el DAC se visualizan en un osciloscopio físico.
    - Datos transmitidos por UART permiten analizar las señales en software de terceros.

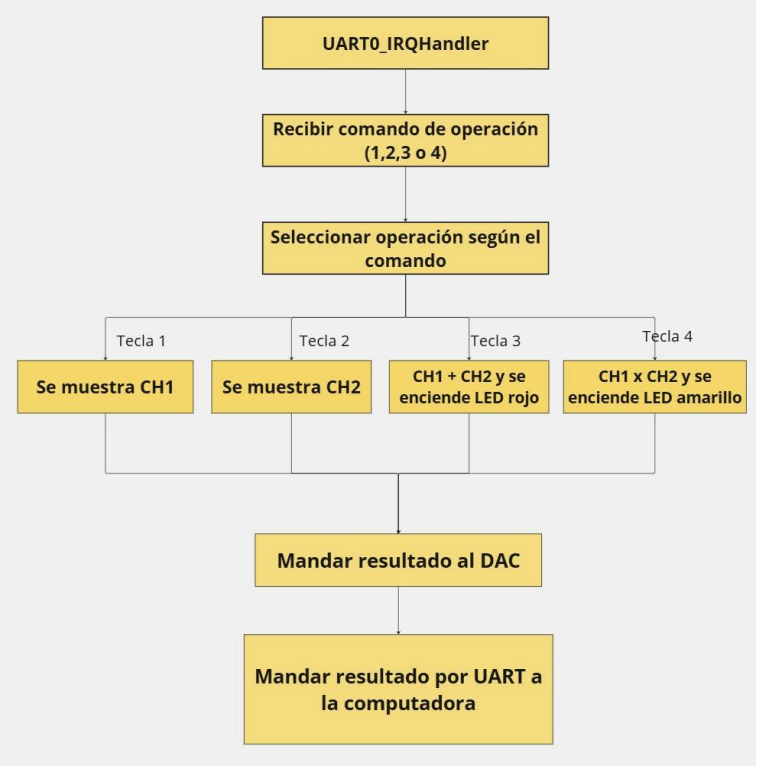
Periféricos

1. **Configuración del ADC**:
    - Se emplean los canales ADC1 y ADC2 para capturar señales analógicas.
    - Frecuencia de muestreo configurada a 200 kHz.
    - Interrupciones habilitadas para transferencia de datos mediante DMA.
2. **Uso del DMA**:
    - **Canal 0**: Transferencia de datos desde el ADC al DAC para la reconstrucción en tiempo real.
    - **Canal 1**: Transferencia de datos desde el ADC hacia un buffer UART para su transmisión.
    - Ventaja del DMA: Reducción de la carga de la CPU, permitiendo un procesamiento más eficiente.
3. **DAC**:
    - Configurado para generar una salida analógica que puede ser visualizada en un osciloscopio digital.
    - Muestra tanto señales individuales como el resultado de operaciones matemáticas.
4. **UART**:
    - Configurado para transmisión a una computadora a través de un módulo serie.
    - Comunicación bidireccional para recibir comandos del usuario y enviar datos procesados.
5. **Control mediante comandos**:
    - **Comando 1**: Muestra la señal del canal 1 del ADC.
    - **Comando 2**: Muestra la señal del canal 2 del ADC.
    - **Comando 3**: Realiza la suma de las señales de ambos canales, la envía al DAC y enciende un LED rojo.
    - **Comando 4**: Realiza la multiplicación de ambas señales, la envía al DAC y enciende un LED amarillo.
6. **Indicadores LED**:
    - **LED rojo**: Encendido durante la operación de suma.
    - **LED amarillo**: Encendido durante la operación de multiplicación.

Diagrama de Flujo







Implementación del Código

El programa está dividido en las siguientes secciones:

1. **Configuración de periféricos**:
    - Inicialización del ADC, DAC, UART y DMA.
    - Configuración de pines mediante la librería `lpc17xx_pinsel`.
2. **Manejo de interrupciones**:
    - **DMA_IRQHandler**: Gestión de interrupciones generadas por la transferencia de datos del ADC al DAC y UART.
    - Control de errores y reinicio de transferencias en caso de fallos.
3. **Procesamiento de datos**:
    - Operaciones de suma y multiplicación realizadas en tiempo real.
    - Aplicación de filtros básicos para suavizar las señales capturadas.
4. **Comunicación UART**:
    - Recepción de comandos desde la computadora.
    - Transmisión de datos procesados en formato legible por software externo.

Pruebas

1. **Prueba de Captura de Señales**:
    - Se conectaron dos señales sinusoidales (6 Hz y 10 Hz) a los canales ADC1 y ADC2.
    - Las señales capturadas fueron transmitidas correctamente al DAC y al UART.
2. **Pruebas UART**:
    - Se verificó la recepción de comandos desde un terminal serie.
    - Los datos procesados fueron enviados correctamente y visualizados en la computadora.
3. **Validación de Operaciones**:
    - La suma y multiplicación de señales se comprobaron mediante su salida reconstruida en el DAC y en el terminal.
4. **Indicadores LED**:
    - Funcionaron correctamente según la operación seleccionada.

Código completo

```
#include "lpc17xx_adc.h"
#include "lpc17xx_dac.h"
#include "lpc17xx_pinsel.h"
#include "lpc17xx_nvic.h"
#include "lpc17xx_gpdma.h"
#include "lpc17xx_gpio.h"
#include "debug_frmwrk.h"
#include "math.h"

/************************** PRIVATE DEFINITIONS
************************/
/* ADC sample frequency */
#define ADC_CONVERSION_RATE 200000

/** DMA size of transfer */
#define DMA_SIZE        1

/************************** PRIVATE VARIABLES
************************/

/* Terminal Counter flag for Channel 0 */
volatile uint32_t Channel0_TC;
volatile uint32_t Channel0_Err;
volatile uint32_t Channel1_TC;
volatile uint32_t Channel1_Err;

/* Signal operation status */
uint8_t operation = 0;  // Suma por default

void DMA_IRQHandler(void) {
    if (GPDMA_IntGetStatus(GPDMA_STAT_INT, 0)) {
        if (GPDMA_IntGetStatus(GPDMA_STAT_INTTC, 0)) {
            GPDMA_ClearIntPending(GPDMA_STATCLR_INTTC, 0);
            Channel0_TC++;
        }
        if (GPDMA_IntGetStatus(GPDMA_STAT_INTERR, 0)) {
            GPDMA_ClearIntPending(GPDMA_STATCLR_INTERR, 0);
            Channel0_Err++;
        }
    }
```

```
    if (GPDMA_IntGetStatus(GPDMA_STAT_INT, 1)) {
        if (GPDMA_IntGetStatus(GPDMA_STAT_INTTC, 1)) {
            GPDMA_ClearIntPending(GPDMA_STATCLR_INTTC, 1);
            Channel1_TC++;
        }
        if (GPDMA_IntGetStatus(GPDMA_STAT_INTERR, 1)) {
            GPDMA_ClearIntPending(GPDMA_STATCLR_INTERR, 1);
            Channel1_Err++;
        }
    }
}

void UART0_IRQHandler(void) {
    // Comprobar si la interrupción es por recepción de datos
    if (UART_GetIntId(LPC_UART0) & UART_IIR_INTID_RDA) {
        uint8_t operation_aux = UART_ReceiveByte(LPC_UART0); // Leer el
dato recibido

        // if(('1' <= operation_aux) && (operation_aux <= '4')){
        if(operation_aux != '\n'){
            operation = operation_aux;

            // Control de LEDs
            switch (operation) {
                case '1':
                    GPIO_ClearValue(0, (1 << 27));
                    GPIO_ClearValue(0, (1 << 28));
                    break;
                case '2':
                    GPIO_ClearValue(0, (1 << 27));
                    GPIO_ClearValue(0, (1 << 28));
                    break;
                case '3':
                    GPIO_SetValue(0, (1 << 27));
                    GPIO_ClearValue(0, (1 << 28));
                    break;
                case '4':
                    GPIO_SetValue(0, (1 << 28));
                    GPIO_ClearValue(0, (1 << 27));
                    break;
                default:
                    GPIO_ClearValue(0, (1 << 27));
                    GPIO_ClearValue(0, (1 << 28));
```

```
                    break;
            }
        }
    }
}

void config(void) {
    PINSEL_CFG_Type PinCfg;
    GPDMA_Channel_CFG_Type GPDMACh0Cfg;
    GPDMA_Channel_CFG_Type GPDMACh1Cfg;
    GPDMA_LLI_Type LLICh1Cfg;
    GPDMA_LLI_Type LLICh2Cfg;
    uint32_t adc_value1;
    uint32_t adc_value2;
    uint32_t result = 0;    // Resultado de la operacion de los dos
canales

    /* Initialize ADC */
    PinCfg.Funcnum = PINSEL_FUNC_1;
    PinCfg.OpenDrain = PINSEL_PINMODE_NORMAL;
    PinCfg.Pinmode = PINSEL_PINMODE_PULLUP;
    PinCfg.Pinnum = PINSEL_PIN_24;
    PinCfg.Portnum = PINSEL_PORT_0;
    PINSEL_ConfigPin(&PinCfg);

    PinCfg.Pinnum = PINSEL_PIN_25;
    PINSEL_ConfigPin(&PinCfg);

    // UART0 Config
    debug_frmwrk_init();

    // Habilitar interrupciones de UART
    UART_IntConfig(LPC_UART0, UART_INTCFG_RBR, ENABLE); // Habilita
interrupciones de recepción
    NVIC_EnableIRQ(UART0_IRQn); // Habilita la interrupción del UART0
en el NVIC

    // Configurar el ADC para una frecuencia de muestreo más alta
    ADC_Init(LPC_ADC, ADC_CONVERSION_RATE); // 200 kHz
    ADC_ChannelCmd(LPC_ADC, ADC_CHANNEL_1, ENABLE); // Activar canal 1
    ADC_ChannelCmd(LPC_ADC, ADC_CHANNEL_2, ENABLE); // Activar canal 2

    /* Configuration for DAC */
```

```
    PinCfg.Funcnum = 2;
    PinCfg.Pinnum = 26;
    PINSEL_ConfigPin(&PinCfg);
    DAC_Init(LPC_DAC);

    /* GPDMA block section
     * EL DMA GUARDA TODO EL REGISTRO DE 32b
     */
    NVIC_DisableIRQ(DMA_IRQn);
    NVIC_SetPriority(DMA_IRQn, 9);  // CAMBIAR SI NO ANDA. ES LO MISMO
QUE LO DE ABAJO
    // NVIC_SetPriority(DMA_IRQn, ((0x01 << 3) | 0x01));
    GPDMA_Init();

    // GPDMA Ch 0 -> ADC Ch 1
    GPDMACh0Cfg.ChannelNum = 0;
    GPDMACh0Cfg.SrcMemAddr = 0;
    GPDMACh0Cfg.DstMemAddr = (uint32_t)&adc_value1;
    GPDMACh0Cfg.TransferSize = DMA_SIZE;
    GPDMACh0Cfg.TransferWidth = 0;
    GPDMACh0Cfg.TransferType = GPDMA_TRANSFERTYPE_P2M;
    GPDMACh0Cfg.SrcConn = GPDMA_CONN_ADC;
    GPDMACh0Cfg.DstConn = 0;
    GPDMACh0Cfg.DMALLI = 0;
    GPDMA_Setup(&GPDMACh0Cfg);

    // GPDMA Ch 1 -> ADC Ch 2
    GPDMACh1Cfg.ChannelNum = 1;
    GPDMACh1Cfg.SrcMemAddr = 0;
    GPDMACh1Cfg.DstMemAddr = (uint32_t)&adc_value2;
    GPDMACh1Cfg.TransferSize = DMA_SIZE;
    GPDMACh1Cfg.TransferWidth = 0;
    GPDMACh1Cfg.TransferType = GPDMA_TRANSFERTYPE_P2M;
    GPDMACh1Cfg.SrcConn = GPDMA_CONN_ADC;
    GPDMACh1Cfg.DstConn = 0;
    GPDMACh1Cfg.DMALLI = 0;
    GPDMA_Setup(&GPDMACh1Cfg);

    Channel0_TC = 0;
    Channel0_Err = 0;
    Channel1_TC = 0;
    Channel1_Err = 0;
    NVIC_EnableIRQ(DMA_IRQn);
```

```
// LEDs P0.27 P0.28
GPIO_SetDir(0, (1 << 27) | (1 << 28), 1);

while (1) {
    GPDMA_ChannelCmd(0, ENABLE);
    ADC_ChannelCmd(LPC_ADC, ADC_CHANNEL_1, ENABLE);

    // Iniciar la conversión del canal 1 y esperar a que finalice
    ADC_StartCmd(LPC_ADC, ADC_START_NOW);
    while (Channel0_TC == 0);

    GPDMA_ChannelCmd(0, DISABLE);
    GPDMA_ChannelCmd(1, ENABLE);
    ADC_ChannelCmd(LPC_ADC, ADC_CHANNEL_1, DISABLE);
    ADC_ChannelCmd(LPC_ADC, ADC_CHANNEL_2, ENABLE);

    // Iniciar la conversión del canal 2 y esperar a que finalice
    ADC_StartCmd(LPC_ADC, ADC_START_NOW);

    while (Channel1_TC == 0);
    ADC_ChannelCmd(LPC_ADC, ADC_CHANNEL_2, DISABLE);
    GPDMA_ChannelCmd(1, DISABLE);

    // Mapear a 12 bits
    adc_value1 = ADC_DR_RESULT(adc_value1);
    adc_value2 = ADC_DR_RESULT(adc_value2);
    // adc_value1 = 2000;
    // adc_value2 = 4000;

    // Enviar valor del ADC al UART para monitoreo
    switch (operation) {
        case '1':
            result = adc_value1;
            break;
        case '2':
            result = adc_value2;
            break;
        case '3':
            result = adc_value1 + adc_value2;
            break;
        case '4':
            result = adc_value1 * adc_value2;
```

```
                break;
            default:
                result = adc_value1;
                break;
        }
        _DBD16((uint16_t)(result & 0x03FF));
        _DBG("\n");

        // Enviar el valor capturado al DAC. Mapeado a 10 bits
        DAC_UpdateValue(LPC_DAC, (uint16_t)(result >> 2));

        GPDMA_Setup(&GPDMACh0Cfg);  // Reconfigurar DMA
        GPDMA_Setup(&GPDMACh1Cfg);  // Reconfigurar DMA
        Channel0_TC = 0;  // Reset del contador terminal
        Channel0_Err = 0; // Reset de error
        Channel1_TC = 0;  // Reset del contador terminal
        Channel1_Err = 0; // Reset de error
    }
    ADC_DeInit(LPC_ADC);
}

int main(void) {
    config();
    return 0;
}
```

Driver para configurar UART

```
#include "debug_frmwrk.h"
#include "lpc17xx_pinsel.h"

/* If this source file built with example, the LPC17xx FW library
configuration
 * file in each example directory ("lpc17xx_libcfg.h") must be
included,
 * otherwise the default FW library configuration file must be included
instead
 */
#ifdef __BUILD_WITH_EXAMPLE__
#include "lpc17xx_libcfg.h"
```

```
#else
#include "lpc17xx_libcfg_default.h"
#endif /* __BUILD_WITH_EXAMPLE__ */

#ifdef _DBGFWK
/* Debug framework */

void (*_db_msg)(LPC_UART_TypeDef *UARTx, const void *s);
void (*_db_msg_)(LPC_UART_TypeDef *UARTx, const void *s);
void (*_db_char)(LPC_UART_TypeDef *UARTx, uint8_t ch);
void (*_db_dec)(LPC_UART_TypeDef *UARTx, uint8_t decn);
void (*_db_dec_16)(LPC_UART_TypeDef *UARTx, uint16_t decn);
void (*_db_dec_32)(LPC_UART_TypeDef *UARTx, uint32_t decn);
void (*_db_hex)(LPC_UART_TypeDef *UARTx, uint8_t hexn);
void (*_db_hex_16)(LPC_UART_TypeDef *UARTx, uint16_t hexn);
void (*_db_hex_32)(LPC_UART_TypeDef *UARTx, uint32_t hexn);
uint8_t (*_db_get_char)(LPC_UART_TypeDef *UARTx);


/*********************************************************************/
/**
 * @brief        Puts a character to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
 * @param[in]    ch        Character to put
 * @return        None

*********************************************************************/
void UARTPutChar (LPC_UART_TypeDef *UARTx, uint8_t ch)
{
    UART_Send(UARTx, &ch, 1, BLOCKING);
}


/*********************************************************************/
/**
 * @brief        Get a character to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
 * @return        character value that returned

*********************************************************************/
uint8_t UARTGetChar (LPC_UART_TypeDef *UARTx)
{
    uint8_t tmp = 0;
```

```
    UART_Receive(UARTx, &tmp, 1, BLOCKING);
    return(tmp);
}


/*********************************************************************/
/**
 * @brief        Puts a string to UART port
 * @param[in]    UARTx     Pointer to UART peripheral
 * @param[in]    str     string to put
 * @return        None

**********************************************************************/
void UARTPuts(LPC_UART_TypeDef *UARTx, const void *str)
{
    uint8_t *s = (uint8_t *) str;

    while (*s)
    {
        UARTPutChar(UARTx, *s++);
    }
}


/*********************************************************************/
/**
 * @brief        Puts a string to UART port and print new line
 * @param[in]    UARTx    Pointer to UART peripheral
 * @param[in]    str        String to put
 * @return        None

**********************************************************************/
void UARTPuts_(LPC_UART_TypeDef *UARTx, const void *str)
{
    UARTPuts (UARTx, str);
    UARTPuts (UARTx, "\n\r");
}


/*********************************************************************/
/**
 * @brief        Puts a decimal number to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
```

```
 * @param[in]    decnum    Decimal number (8-bit long)
 * @return        None

*********************************************************************/
void UARTPutDec(LPC_UART_TypeDef *UARTx, uint8_t decnum)
{
    uint8_t c1=decnum%10;
    uint8_t c2=(decnum/10)%10;
    uint8_t c3=(decnum/100)%10;
    UARTPutChar(UARTx, '0'+c3);
    UARTPutChar(UARTx, '0'+c2);
    UARTPutChar(UARTx, '0'+c1);
}

/*********************************************************************/
/**
 * @brief        Puts a decimal number to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
 * @param[in]    decnum    Decimal number (8-bit long)
 * @return        None

*********************************************************************/
void UARTPutDec16(LPC_UART_TypeDef *UARTx, uint16_t decnum)
{
    uint8_t c1=decnum%10;
    uint8_t c2=(decnum/10)%10;
    uint8_t c3=(decnum/100)%10;
    uint8_t c4=(decnum/1000)%10;
    uint8_t c5=(decnum/10000)%10;
    UARTPutChar(UARTx, '0'+c5);
    UARTPutChar(UARTx, '0'+c4);
    UARTPutChar(UARTx, '0'+c3);
    UARTPutChar(UARTx, '0'+c2);
    UARTPutChar(UARTx, '0'+c1);
}

/*********************************************************************/
/**
 * @brief        Puts a decimal number to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
 * @param[in]    decnum    Decimal number (8-bit long)
 * @return        None
```

```
*********************************************************************/
void UARTPutDec32(LPC_UART_TypeDef *UARTx, uint32_t decnum)
{
    uint8_t c1=decnum%10;
    uint8_t c2=(decnum/10)%10;
    uint8_t c3=(decnum/100)%10;
    uint8_t c4=(decnum/1000)%10;
    uint8_t c5=(decnum/10000)%10;
    uint8_t c6=(decnum/100000)%10;
    uint8_t c7=(decnum/1000000)%10;
    uint8_t c8=(decnum/10000000)%10;
    uint8_t c9=(decnum/100000000)%10;
    uint8_t c10=(decnum/1000000000)%10;
    UARTPutChar(UARTx, '0'+c10);
    UARTPutChar(UARTx, '0'+c9);
    UARTPutChar(UARTx, '0'+c8);
    UARTPutChar(UARTx, '0'+c7);
    UARTPutChar(UARTx, '0'+c6);
    UARTPutChar(UARTx, '0'+c5);
    UARTPutChar(UARTx, '0'+c4);
    UARTPutChar(UARTx, '0'+c3);
    UARTPutChar(UARTx, '0'+c2);
    UARTPutChar(UARTx, '0'+c1);
}

/*********************************************************************/
/**
 * @brief        Puts a hex number to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
 * @param[in]    hexnum    Hex number (8-bit long)
 * @return        None

*********************************************************************/
void UARTPutHex (LPC_UART_TypeDef *UARTx, uint8_t hexnum)
{
    uint8_t nibble, i;

    UARTPuts(UARTx, "0x");
    i = 1;
    do {
        nibble = (hexnum >> (4*i)) & 0x0F;
```

```
        UARTPutChar(UARTx, (nibble > 9) ? ('A' + nibble - 10) : ('0' +
nibble));
    } while (i--);
}


/*****************************************************************/
/**
 * @brief        Puts a hex number to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
 * @param[in]    hexnum    Hex number (16-bit long)
 * @return        None

*****************************************************************/
void UARTPutHex16 (LPC_UART_TypeDef *UARTx, uint16_t hexnum)
{
    uint8_t nibble, i;

    UARTPuts(UARTx, "0x");
    i = 3;
    do {
        nibble = (hexnum >> (4*i)) & 0x0F;
        UARTPutChar(UARTx, (nibble > 9) ? ('A' + nibble - 10) : ('0' +
nibble));
    } while (i--);
}

/*****************************************************************/
/**
 * @brief        Puts a hex number to UART port
 * @param[in]    UARTx    Pointer to UART peripheral
 * @param[in]    hexnum    Hex number (32-bit long)
 * @return        None

*****************************************************************/
void UARTPutHex32 (LPC_UART_TypeDef *UARTx, uint32_t hexnum)
{
    uint8_t nibble, i;

    UARTPuts(UARTx, "0x");
    i = 7;
    do {
        nibble = (hexnum >> (4*i)) & 0x0F;
```

```
        UARTPutChar(UARTx, (nibble > 9) ? ('A' + nibble - 10) : ('0' +
nibble));
    } while (i--);
}

///*********************************************************************
*//**
// * @brief        print function that supports format as same as
printf()
// *                 function of <stdio.h> library
// * @param[in]    None
// * @return        None
//
**********************************************************************/
//void  _printf (const  char *format, ...)
//{
//    static  char  buffer[512 + 1];
//            va_list     vArgs;
//            char    *tmp;
//    va_start(vArgs, format);
//    vsprintf((char *)buffer, (char const *)format, vArgs);
//    va_end(vArgs);
//
//    _DBG(buffer);
//}

/*********************************************************************/
/**
 * @brief        Initialize Debug frame work through initializing UART
port
 * @param[in]    None
 * @return        None

**********************************************************************/
void debug_frmwrk_init(void)
{
    UART_CFG_Type UARTConfigStruct;
    PINSEL_CFG_Type PinCfg;

#if (USED_UART_DEBUG_PORT==0)
    /*
     * Initialize UART0 pin connect
     */
```

```
    PinCfg.Funcnum = 1;
    PinCfg.OpenDrain = 0;
    PinCfg.Pinmode = 0;
    PinCfg.Pinnum = 2;
    PinCfg.Portnum = 0;
    PINSEL_ConfigPin(&PinCfg);
    PinCfg.Pinnum = 3;
    PINSEL_ConfigPin(&PinCfg);
#elif (USED_UART_DEBUG_PORT==1)
    /*
     * Initialize UART1 pin connect
     */
    PinCfg.Funcnum = 1;
    PinCfg.OpenDrain = 0;
    PinCfg.Pinmode = 0;
    PinCfg.Pinnum = 15;
    PinCfg.Portnum = 0;
    PINSEL_ConfigPin(&PinCfg);
    PinCfg.Pinnum = 16;
    PINSEL_ConfigPin(&PinCfg);
#endif

    /* Initialize UART Configuration parameter structure to default
state:
     * Baudrate = 9600bps
     * 8 data bit
     * 1 Stop bit
     * None parity
     */
    UART_ConfigStructInit(&UARTConfigStruct);
    // Re-configure baudrate to 115200bps
    UARTConfigStruct.Baud_rate = 9600;

    // Initialize DEBUG_UART_PORT peripheral with given to
corresponding parameter
    UART_Init((LPC_UART_TypeDef*)DEBUG_UART_PORT, &UARTConfigStruct);

    // Enable UART Transmit
    UART_TxCmd((LPC_UART_TypeDef*)DEBUG_UART_PORT, ENABLE);

    _db_msg    = UARTPuts;
    _db_msg_ = UARTPuts_;
    _db_char = UARTPutChar;
```

```
    _db_hex = UARTPutHex;
    _db_hex_16 = UARTPutHex16;
    _db_hex_32 = UARTPutHex32;
    _db_dec = UARTPutDec;
    _db_dec_16 = UARTPutDec16;
    _db_dec_32 = UARTPutDec32;
    _db_get_char = UARTGetChar;
}
#endif /*_DBGFWK */
```