SISTEMA DE DEMOSTRACIÓN PARA SISTEMAS OPERATIVOS DE TIEMPO REAL SOBRE SISTEMAS EMPOTRADOS, CASO: UP ADAPTABLE NIOS II

ANEXOS

César Iván Alvear Vega
Ricardo Antonio Palacios Ledezma

Monografía presentada para optar al título de Ingeniero en Electrónica y Telecomunicaciones

Directora
Mag. Ing. Carolina Ríos Fuentes

Co-Directora
Mag. (c) Ing. Mary Cristina Carrascal

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Popayán, Octubre de 2010

Tabla de Contenido

Índice de Ilustraciones

ANEXO A

PRÁCTICAS SOBRE SISTEMAS DE TIEMPO REAL

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática

SISTEMAS DE TIEMPO REAL

A.1. PRACTICA No. 1
GUÍA DE AMBIENTACIÓN A LAS HERRAMIENTAS DE TRABAJO

OBJETIVO:

- Lograr que el estudiante se familiarice con la interfaz de monitoreo y depuración remota para Nios II con el que se va a trabajar.
- Conocer el proceso de creación y compilación de un proyecto en el entorno de desarrollo Nios II 7.2 IDE.
- Efectuar el proceso de configuración de las características de trabajo del kit de desarrollo Stratix II con procesador adaptable Nios II.

DESCRIPCIÓN:

En esta práctica se tratarán los aspectos básicos de la edición, y compilación de códigos en el entorno de desarrollo Nios II 7.2 IDE por medio de un ejemplo, en el cual se busca crear una variable que se incrementa y muestra los cambios de los estados en LEDs. Además, se debe llevar a cabo la configuración de trabajo de la tarjeta de desarrollo utilizando el software de programación Quartus II; para terminar, se lleva a cabo una sesión de monitoreo sobre la tarjeta de desarrollo utilizando la interfaz de monitoreo depuración para Nios II.

PROCEDIMIENTO:

Para crear o editar un archivo .c, vamos a utilizar el entorno de desarrollo Nios II 7.2 IDE, que nos permite crear, editar, y compilar los programas que desarrollaremos durante el curso. Iniciaremos creando un proyecto que llamaremos “ContadorBinario” y un archivo .c con el nombre de “contbin.c”, como lo indica la primera sección del manual de éste IDE; luego tomaremos como ejemplo el siguiente sencillo código, que debe ser copiado en el archivo que creamos:

```
/*Ejemplo de contador binario: Crear una variable que se incrementa
y muestra los cambios de los estados en LEDs.
*/
```

```
//Inclusión de librerías

#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"

 //Creación del la función principal

    int main (void)
    {
        alt_u8 led = 0; // declaración e inicialización de la variable "led"

         while (1)      // creación de un bucle infinito
         {
             led++;     // incremento de la variable "led"

                IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE,   led);   /*   Instrucción   que
permite a la variable ser visualizada en los leds*/

            }

        return 0;
     }
```

A continuación, se debe guardar los cambios y efectuar el proceso de compilado, como lo indica la segunda sección del manual de uso del IDE Nios II.

Después de haber llevado a cabo la compilación, se obtiene un archivo .elf, creado en la carpeta del proyecto; éste archivo es el ejecutable con la información del proyecto que tenemos que programar sobre la tarjeta de desarrollo.

Antes de programar la tarjeta de desarrollo con nuestra aplicación, se debe configurar la tarjeta de desarrollo con la configuración del hardware de las características que vamos a trabajar. Para lo que debemos seguir la guía de configuración de la tarjeta de desarrollo que se encuentra en el manual del programador Quartus II.

Ahora ya es posible acceder a la tarjeta de desarrollo para poder programarle tu archivo .elf de programación y poder llevar a cabo la depuración y monitoreo siguiendo su manual de ayuda.

SISTEMAS DE TIEMPO REAL

A.2. PRACTICA No. 2
MANEJO DE PERIFERICOS DEL KIT DE DESARROLLO STRATIX II PARA NIOS II

OBJETIVO:

- Conocer el funcionamiento de los periféricos de salida y entrada con los que cuenta el kit de desarrollo Stratix II con procesador adaptable Nios II.

PROCEDIMIENTO:

En la práctica se busca experimentar de manera directa con el funcionamiento de los periféricos tanto de entrada como de salida, disponibles en la tarjeta de desarrollo, para lo cual se presenta como apoyo un ejemplo en el que se puede visualizar el manejo y funcionamiento de éstos periféricos.

Los estudiantes deben implementar un contador de 8 bits el cual debe mostrar su valor en los 8 LED y los *display* de 7 segmentos, y aplicar a este valor una función matemática (seno, logaritmo, raíz cuadrada, etc.) diferente de acuerdo a cada pulsador que se presione, y mostrar el resultado de estas operaciones en el LCD y en el *Hyper Terminal* utilizando el puerto serial.

```
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include "system.h"
#include <stdio.h>
#include <unistd.h>

#ifndef LCD_DISPLAY_NAME

#   define LCD_CLOSE(x)
#   define LCD_OPEN() NULL
#   define LCD_PRINTF(lcd, args...) fprintf(lcd, args)

#else

#   define LCD_CLOSE(x) fclose((x))
#   define LCD_OPEN() fopen("/dev/lcd_display", "w")
#   define LCD_PRINTF fprintf

#endif

#define ESC 27
```

```
#define ESC_CLEAR "K"
#define ESC_COL1_INDENT5 "[1;5H"
#define ESC_COL2_INDENT5 "[2;5H"
#define ESC_TOP_LEFT "[1;0H"

static alt_u8 count;
volatile int edge_capture;
FILE* fp;

#ifdef BUTTON_PIO_BASE
static void handle_button_interrupts(void* context, alt_u32 id)
{
    volatile int* edge_capture_ptr = (volatile int*) context;
    *edge_capture_ptr =
IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0);
}

static void init_button_pio()
{
    void* edge_capture_ptr = (void*) &edge_capture;

    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE, 0xf);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0x0);
    alt_irq_register( BUTTON_PIO_IRQ, edge_capture_ptr,
handle_button_interrupts );
}
#endif

#ifdef SEVEN_SEG_PIO_BASE
static void sieteseg_set(int hex)
{
    static alt_u8 segmentos[16] = {
        0x81, 0xCF, 0x92, 0x86, 0xCC, 0xA4, 0xA0, 0x8F, 0x80, 0x84, /*
0-9 */
        0x88, 0xE0, 0xF2, 0xC2, 0xB0, 0xB8 };                       /*
a-f */

    unsigned int data = segmentos[hex & 15] | (segmentos[(hex >> 4) &
15] << 8);
    IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE, data);
}
#endif

static void inicia_lcd( FILE *lcd )
{
   LCD_PRINTF(lcd, "%c%s NIOS II Prueba Perifericos...",
ESC,ESC_TOP_LEFT);
}

static void mensaje_inicial()
{

printf("\n\n*****************************************************\n");
    printf("*       Prueba de Funcionamiento de Perifericos     *\n");
    printf("*       Contador                                    *\n");
    printf("*****************************************************\n");

    fprintf(fp,
"%s","\n\r\n\r*****************************************************\n\
r");
```

```
    fprintf(fp, "%s","*       Prueba de Funcionamiento de Perifericos
*\n\r");
    fprintf(fp, "%s","*       Contador
*\n\r");
    fprintf(fp,
"%s","*************************************************\n\r");
}



static void contador_led()
{
    alt_u8 b = count;
#ifdef LED_PIO_BASE
    IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, b);
#endif
}


static void contador_sieteseg()
{
#ifdef SEVEN_SEG_PIO_BASE
    sieteseg_set(count);
#endif
}


static void contador_lcd( void* arg )
{
#ifdef LCD_DISPLAY_NAME
    FILE *lcd = (FILE*) arg;
    LCD_PRINTF(lcd, "%c%s 0x%x\n", ESC, ESC_COL2_INDENT5, count);
#endif
}

static void contador_todos( void* arg )
{
    contador_led();
    contador_sieteseg();
    contador_lcd( arg );
    printf("%02x,  ", count);
    fprintf(fp, "\t%d",count);
}


static void handle_button_press(alt_u8 type, FILE *lcd)
{
    if (type == 'c')
    {
        switch (edge_capture)
        {
            case 0x1: contador_led();
            break;
            case 0x2: contador_sieteseg();
            break;
            case 0x4: contador_lcd( lcd );
            break;
            case 0x8: contador_todos( lcd );
            break;
            default:  contador_todos( lcd );
            break;
```

```
        }
    }
    else
    {
        switch (edge_capture)
        {
        case 0x1: printf( "Button 1\n");
            edge_capture = 0;
            break;
        case 0x2: printf( "Button 2\n");
            edge_capture = 0;
            break;
        case 0x4: printf( "Button 3\n");
            edge_capture = 0;
            break;
        case 0x8: printf( "Button 4\n");
            edge_capture = 0;
            break;
        default: printf( "Button press UNKNOWN!!\n");
        }
    }
}


int main(void)
{
    int i;
    int wait_time;
    FILE * lcd;

    fp = fopen ("/dev/uart1", "w");
    count = 0;
    lcd = LCD_OPEN();
    if(lcd != NULL) {inicia_lcd( lcd );}

#ifdef BUTTON_PIO_BASE
    init_button_pio();
#endif

    mensaje_inicial();

    while( 1 )
    {
        usleep(100000);
        if (edge_capture != 0)
        {
           handle_button_press('c', lcd);
        }
        else
        {
            contador_todos( lcd );
        }
        if( count == 0xff )
        {
            LCD_PRINTF(lcd, "%c%s %c%s %c%s Esperando...\n", ESC,
ESC_TOP_LEFT,
                        ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
            printf("\nEsperando...");
            edge_capture = 0;
```

```
            LCD_PRINTF(lcd, "%c%s, %c%s", ESC, ESC_COL2_INDENT5, ESC,
ESC_CLEAR);
            wait_time = 0;
            for (i = 0; i<70; ++i)
            {
                printf(".");
                wait_time = i/10;
                LCD_PRINTF(lcd, "%c%s %ds\n", ESC, ESC_COL2_INDENT5,
                    wait_time+1);

                if (edge_capture != 0)
                {
                    printf( "\nPresiono:  " );
                    handle_button_press('w', lcd);
                }
                usleep(100000);
            }
            mensaje_inicial();
            inicia_lcd( lcd );
        }
        count++;
    }
    LCD_CLOSE(lcd);
    return 0;
}
```

SISTEMAS DE TIEMPO REAL

A.3. PRACTICA No. 3
ADMINISTRACIÓN DE TAREAS

OBJETIVO:

- Poner en práctica los conceptos relacionados con la creación de tareas y su administración entendida como pausar, reanudar, eliminar, manejo de prioridades y ejecución de tareas.

- Profundizar los conocimientos teóricos de funcionamiento de tareas implementadas en un Sistema de Tiempo Real y su aplicabilidad en la ejecución de procesos.

CONCEPTOS GENERALES

Tarea

Una tarea, también llamada un subproceso, es un programa sencillo que interpreta que tiene toda la Unidad Central de Procesamiento o (CPU) para sí misma. El proceso de diseño de una aplicación en tiempo real consiste en dividir el trabajo por hacer en tareas que son responsables de una parte de éste. A cada tarea se le asigna una prioridad, un conjunto de registros de CPU y un área de pila.

Cada tarea es normalmente un bucle infinito que puede estar en cualquiera de estos cinco Estados: INACTIVO, DISPONIBLE, ACTIVO, ESPERANDO, o en Interrupción (ISR) Figura A1. El estado INACTIVO, corresponde a una tarea que reside en la memoria pero no se ha puesto a disposición para el núcleo multitarea. Una tarea está DISPONIBLE, cuando puede ejecutarse pero su prioridad es menor que la tarea actualmente en curso. Una tarea esta ACTIVA, cuando tiene el control de la CPU. Una tarea está ESPERANDO, cuando se requiere la ocurrencia de un evento (esperando a que una operación de E/S se complete, un recurso compartido esté disponible, un pulso de sincronización se produzca, un tiempo a punto de caducar etc.). Por último, una tarea se Interrumpe ISR, cuando se ha producido una interrupción y la CPU está en el proceso de atención a esta.

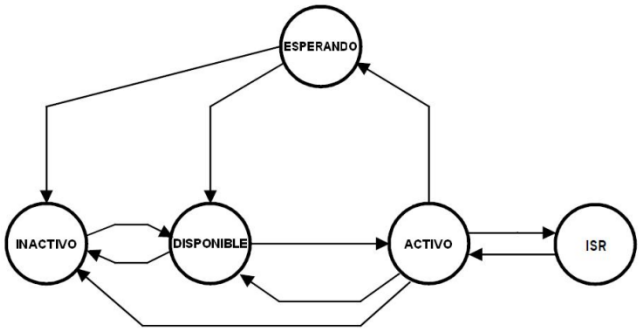


Figura A1 Estado de las tareas

Tareas múltiples

Es el proceso de programación y conmutación de la CPU cuando se están manejando varias tareas, una sola CPU cambia su atención entre varias tareas secuenciales. El funcionamiento de múltiples tareas maximiza la utilización de la CPU y admite la construcción modular de las aplicaciones. Uno de los aspectos más importantes de la multitarea es que permite al programador gestionar la complejidad inherente de las aplicaciones en tiempo real. Los programas son generalmente más fáciles de comprender, diseñar y mantener si se utiliza la Tareas múltiples.

PROCEDIMIENTO:

A continuación se presenta un ejemplo en el que se implementan tres tareas, se les asigna una prioridad a cada una, posteriormente se ejecutan, y se eliminan. En un segundo ejemplo, las tareas creadas se suspenden y reinician.

Utilizando como material de apoyo los siguientes códigos, implementar un sistema en el que se creen

En el siguiente código de creación y eliminación de tareas, además se presenta el manejo de los periféricos LEDs, Display de 7 segmentos y LCD. En la primera parte, se presenta la codificación en el SOTR Micro C/OS II:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "includes.h"
#include "Perifericos.h"

/* Definición de tareas */

#define   TASK_STACKSIZE       1024
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
OS_STK    task3_stk[TASK_STACKSIZE];
OS_STK    task4_stk[TASK_STACKSIZE];

/* Definición de prioridades de las tareas */

#define TASK2_PRIORITY      11
#define TASK3_PRIORITY      12
#define TASK4_PRIORITY      13
FILE *lcd;

unsigned int contador;

/* Prints "Hello World" and sleeps for three seconds */

void task2(void* pdata){
volatile unsigned int i=0;
```

```
  while (1){
    i++;
       LED(2);
       printf("Tarea 2\n");
      contador++;
       Setsegmentos(contador);
       if(i==20) {
                    fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
                    fprintf(lcd, "Tarea 2 Borrada\n");
                    OSTaskDel(11);
       }
    OSTimeDlyHMSM(0, 0, 1, 0);
  }     }

void task3(void* pdata){
volatile unsigned int i=0;
  while (1){
     i++;
       LED(3);
       printf("Tarea 3\n");
      contador++;
       Setsegmentos(contador);
       if(i==20) {
                     fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
                     fprintf(lcd, "Tarea 3 Borrada\n");
                     OSTaskDel(12);
       }
    OSTimeDlyHMSM(0, 0, 0, 500);
  }     }

void task4(void* pdata){
volatile unsigned int i=0;
  while (1){
     i++;
       LED(4);
       printf("Tarea 4\n");
       contador++;
       Setsegmentos(contador);
       if(i==20) {
                    fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
                    fprintf(lcd, "Tarea 4 Borrada\n");
                    OSTaskDel(13);
       }
    OSTimeDlyHMSM(0, 0, 0, 250);
  }     }

int main(void){
  lcd=fopen("/dev/lcd_display", "w");

  OSTaskCreateExt(task2, NULL, (void *)&task2_stk[TASK_STACKSIZE],  TASK2_PRIORITY,
                  TASK2_PRIORITY, task2_stk, TASK_STACKSIZE,  NULL, 0);

 OSTaskCreateExt(task3, NULL, (void *)&task3_stk[TASK_STACKSIZE], TASK3_PRIORITY,
                  TASK3_PRIORITY, task3_stk, TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(task4, NULL, (void *)&task4_stk[TASK_STACKSIZE], TASK4_PRIORITY,
                  TASK4_PRIORITY, task4_stk, TASK_STACKSIZE, NULL, 0);

  OSTaskCreateExt(task5, NULL,(void *)&task5_stk[TASK_STACKSIZE], TASK5_PRIORITY,
                  TASK5_PRIORITY, task5_stk, TASK_STACKSIZE, NULL, 0);
  OSStart();
  return 0;
}
```

En la segunda sección, se presenta la codificación en el SOTR FreeRTOS:

```
/* Standard includes. */
#include <stddef.h>
#include <stdio.h>
#include <string.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
```

```
#include "queue.h"
#include "semphr.h"
#include "Perifericos.h"
#define STACK_SIZE       configMINIMAL_STACK_SIZE

 void vTaskCode1( void *pvParameters );
 void vTaskCode2( void *pvParameters );
 void vTaskCode3( void *pvParameters );

 xTaskHandle xHandle1, xHandle2, xHandle3;
 FILE *lcd;
 unsigned int contador;

void vTaskCode1( void * pvParameters ){
    volatile unsigned int i=0;
    while(1){
        i++;
        contador++;
        Setsegmentos(contador);
        LED(1);
        printf("Tarea 1\n");
        if(i==20) {
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "Tarea 1 Borrada\n");
         vTaskDelete( xHandle1 );
        }
        vTaskDelay( ( portTickType ) 1000/portTICK_RATE_MS  );
     }   }

void vTaskCode2( void * pvParameters ){
    volatile unsigned int i=0;
     while(1){
       i++;
       contador++;
        Setsegmentos(contador);
       LED(2);
       printf("Tarea 2\n");
       if(i==20) {
       fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
       fprintf(lcd, "Tarea 2 Borrada\n");
       vTaskDelete( xHandle2 );
       }
       vTaskDelay( ( portTickType ) 500/portTICK_RATE_MS );
     }      }

 void vTaskCode3( void * pvParameters ){
    volatile unsigned int i=0;
     while(1){
       i++;
       contador++;
        Setsegmentos(contador);
       LED(3);
       printf("Tarea 3\n");
       if(i==20) {
                 fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
                 fprintf(lcd, "Tarea 3 Borrada\n");
                 vTaskDelete( xHandle3);
       }
       vTaskDelay( ( portTickType ) 250/portTICK_RATE_MS );
     }      }
```

```
int main( void ){
  lcd=fopen("/dev/lcd_display", "w");
    static unsigned char ucParameterToPass;

    xTaskCreate( vTaskCode1, "NAME1", STACK_SIZE, &ucParameterToPass,
1, &xHandle1 );
    xTaskCreate( vTaskCode2, "NAME2", STACK_SIZE, &ucParameterToPass,
2, &xHandle2 );
    xTaskCreate( vTaskCode3, "NAME3", STACK_SIZE, &ucParameterToPass,
3, &xHandle3 );

    vTaskStartScheduler();
    return 0;
}
```

En el siguiente código de ejemplo, se crean tres tareas, se implementan y se suspenden, utilizando también, los periféricos de salida.

En la primera sección se presenta el código desarrollado sobre el SOTR Micro C/OS II:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "includes.h"
#include "Perifericos.h"
#include "sys/time.h"

static unsigned long ulLedStates;
#define partstNUM_LEDS          ( 8 )

/* Definición de tareas */

#define   TASK_STACKSIZE       1024

OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
OS_STK    task3_stk[TASK_STACKSIZE];
OS_STK    task4_stk[TASK_STACKSIZE];
OS_STK    task5_stk[TASK_STACKSIZE];

/*Definición de las prioridades de las tareas; recuerden que  en el
sistema operativo Micro C/OS-II   el número menor indica mayor
prioridad*/

#define TASK2_PRIORITY      11
#define TASK3_PRIORITY      12
#define TASK4_PRIORITY      13
#define TASK5_PRIORITY      14

FILE *lcd;
unsigned int contador;

/* implementación de las tareas  */

void task2(void* pdata){
volatile unsigned int i=0;
  while (1){
    i++;
       LED(2);
       printf("Tarea 2\n");
```

```
      contador++;
       Setsegmentos(contador);
       if(i==20) {
                  fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
                  fprintf(lcd, "T2 Suspendida\n");
                   i=0;
                  OSTaskSuspend (11);             }
    OSTimeDlyHMSM(0, 0, 1, 0);
  }   }
void task3(void* pdata)
{
volatile unsigned int i=0;
  while (1){
     i++;
       LED(3);
       printf("Tarea 3\n");
      contador++;
       Setsegmentos(contador);
       if(i==20) {
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "T3 Suspendida\n");
        i=0;
       OSTaskSuspend (12);  }
    OSTimeDlyHMSM(0, 0, 0, 500);
  }   }

void task4(void* pdata){

volatile unsigned int i=0;
  while (1){
     i++;
       LED(4);
       printf("Tarea 4\n");
       contador++;
       Setsegmentos(contador);
       if(i==20) {
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "T4 Suspendida\n");
        i=0;
        OSTaskSuspend (13); }
    OSTimeDlyHMSM(0, 0, 0, 250);}   }

void task5(void* pdata){
  while (1) {
    if (contador ==60){
                usleep(1000000);
                fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
                fprintf(lcd, "  Reiniciando\n     Tareas\n");
                OSTaskResume (11);
                OSTaskResume (12);
                OSTaskResume (13);
                contador =0;
    }
    OSTimeDlyHMSM(0, 0, 1, 0);
  }   }

/* Creación del la función principal */

int main(void){
  lcd=fopen("/dev/lcd_display", "w");
```

```
  OSTaskCreateExt(task2,NULL,(void *)&task2_stk[TASK_STACKSIZE],
                  TASK2_PRIORITY, TASK2_PRIORITY, task2_stk,
                  TASK_STACKSIZE, NULL,0);

 OSTaskCreateExt(task3,NULL,(void *)&task3_stk[TASK_STACKSIZE],
                  TASK3_PRIORITY, TASK3_PRIORITY, task3_stk,
                  TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(task4, NULL, (void *)&task4_stk[TASK_STACKSIZE],
                  TASK4_PRIORITY, TASK4_PRIORITY, task4_stk,
                  TASK_STACKSIZE, NULL,0);

  OSTaskCreateExt(task5,NULL, (void *)&task5_stk[TASK_STACKSIZE],
                  TASK5_PRIORITY, TASK5_PRIORITY, task5_stk,
                  TASK_STACKSIZE, NULL,0);
  OSStart();
  return 0;
}
```

Ejemplo de código de creación de tareas desarrollado en el SOTR FreeRTOS:

```
/*  includes Standard */
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
//HAL includes
#include <sys/time.h>

/* includes SO FreeRTOS */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
/* includes manejo perifericos*/
#include "Perifericos.h"

#define STACK_SIZE       configMINIMAL_STACK_SIZE

 void vTaskCode1( void *pvParameters );
 void vTaskCode2( void *pvParameters );
 void vTaskCode3( void *pvParameters );
 void vReiniciar( void *pvParameters );

 xTaskHandle xHandle1, xHandle2, xHandle3, xHandle4;
 FILE *lcd;
 unsigned int contador;

void vTaskCode1( void * pvParameters ){
    volatile unsigned int i=0;
    while(1){
        i++;
        contador++;
        Setsegmentos(contador);
        LED(1);
        printf("Tarea 1\n");
        if(i==20) {
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "  T1 Suspendida\n");
        i=0;
        vTaskSuspend( xHandle1 );
        }
```

```
        vTaskDelay( ( portTickType ) 1000/portTICK_RATE_MS  );
     }      }

void vTaskCode2( void * pvParameters ) {
    volatile unsigned int i=0;
     while(1){
       i++;
       contador++;
        Setsegmentos(contador);
       LED(2);
       printf("Tarea 2\n");

       if(i==20) {
       fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
       fprintf(lcd, "  T2 Suspendida\n");
       i=0;
       vTaskSuspend( xHandle2 );
       }
       vTaskDelay( ( portTickType ) 500/portTICK_RATE_MS );
     }      }

 void vTaskCode3( void * pvParameters ){
    volatile unsigned int i=0;

     while(1){
       i++;
       contador++;
        Setsegmentos(contador);
       LED(3);
       printf("Tarea 3\n");
       if(i==20) {
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "  T3 Suspendida\n");
        i=0;
         vTaskSuspend( xHandle3 );
       }
       vTaskDelay( ( portTickType ) 250/portTICK_RATE_MS );
     }      }

void vReiniciar( void * pvParameters ){
          while(1){
       if(contador==60){
        usleep(1000000);
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "  Reiniciando\n     Tareas\n");
        vTaskResume( xHandle1 );
        vTaskResume( xHandle2 );
        vTaskResume( xHandle3 );

        contador=0;
        }
       vTaskDelay( ( portTickType ) 1000/portTICK_RATE_MS );
     }       }
  int main( void ){
  lcd=fopen("/dev/lcd_display", "w");
        static unsigned char ucParameterToPass;
    xTaskCreate( vTaskCode1, "NAME1", STACK_SIZE, &ucParameterToPass,
2, &xHandle1 );
    xTaskCreate( vTaskCode2, "NAME2", STACK_SIZE, &ucParameterToPass,
3, &xHandle2 );
```

```
    xTaskCreate( vTaskCode3, "NAME3", STACK_SIZE, &ucParameterToPass,
4, &xHandle3 );
    xTaskCreate( vReiniciar, "NAME4", STACK_SIZE, &ucParameterToPass,
1, &xHandle4 );
    vTaskStartScheduler();
    return 0;
}
```

SISTEMAS DE TIEMPO REAL

A.4. PRACTICA No. 4
SINCRONIZACIÓN Y COMUNICACIÓN DE TAREAS

OBJETIVO:

- Poner en práctica los conceptos relacionados con la sincronización y comunicación de tareas aquí utilizaremos los conceptos de creación y manejo de colas, mutex y semáforos.

- Profundizar los conocimientos teóricos de funcionamiento de tareas implementadas en un Sistema de Tiempo Real y su aplicabilidad en la ejecución de procesos utilizando los SOTR FreeRTOS, y Micro C/OS II.

CONCEPTOS GENERALES

Al utilizar variables globales, cada proceso debe asegurarse de que tiene acceso exclusivo a los recursos para evitar corrupción en los datos. Entre los mecanismos que permiten esta comunicación se encuentran:

- **Semáforos:** un semáforo es una variable protegida o tipo abstracto de datos que constituye un método clásico de controlar el acceso por varios procesos a un recurso común en un entorno de programación paralelo. Un semáforo generalmente toma una de dos formas: binario o contador. Un semáforo binario es un indicador de condición que registra si un recurso está disponible o no; estos solo pueden tomar dos valores "verdadero/falso" (bloqueado / desbloqueado), si esta en verdadero el recurso está disponible, y el proceso lo puede utilizar; si es falso, el recurso no está disponible y el proceso debe esperar. Y el semáforo como contador, trabaja llevando la cuenta de los recursos que se encuentren disponibles. Cualquier tipo de semáforo puede ser empleado para prevenir accesos simultáneos a un mismo recurso .

- **Cola de mensajes:** es uno de los mecanismos más utilizados para comunicar procesos. Éste permite que un número de mensajes, cada uno de una longitud variable, sea ordenado en una estructura de cola (*First In, First Out* FIFO o basada en prioridades) a la espera de ser leídos. Una vez que ha sido creada, cualquier proceso puede escribir y leer mensajes sobre ellas. Por tanto, es posible que varios procesos envíen mensajes a una misma cola y que múltiples procesos reciban mensajes de una de ellas. Es conveniente borrar las colas de mensajes cuando no van a ser utilizadas, ya que en caso contrario se malgastará de forma innecesaria la memoria del sistema.

Cuando se habla de sincronización se hace referencia a la coordinación de procesos que se ejecutan simultáneamente para completar una tarea, con el fin de obtener un orden de ejecución correcto y evitar así estados inesperados, independientemente del tipo de interacción existente entre los distintos procesos activos, en un sistema con multiprogramación éstos comparten un conjunto de elementos que deben ser accedidos de forma controlada para evitar situaciones de inconsistencia. Estos elementos compartidos ya sean dispositivos de E/S o zonas de memoria comunes son considerados como críticos y la parte del programa que los utiliza se conoce como región o sección crítica. Es muy importante que sólo un programa pueda acceder a su sección crítica en un momento determinado. Por esta razón, el sistema debe ofrecer mecanismos que hagan posible una correcta sincronización de los distintos procesos activos en los accesos a los recursos que comparten.

El mecanismo más utilizado en la sincronización utilizado por los STR es la Exclusión Mutua o MUTEX, la cual es una forma de acceso en la que un proceso excluye temporalmente a todos los demás de utilizar un recurso compartido, con el fin de asegurar la integridad del sistema. Si el recurso compartido es una variable, la exclusión mutua asegura que, como máximo, un proceso tendrá acceso a ella durante las actualizaciones críticas. En el caso de compartir dispositivos, la exclusión mutua es mucho más obvia si se consideran los problemas que pueden derivarse de su uso incontrolado.

PROCEDIMIENTO:

Después de haber estudiado los conceptos relacionados con la comunicación y sincronización de tareas, y utilizando como apoyo los siguientes ejemplos crear dos tareas que generen números aleatorios los cuales se escriben en una cola, garantizando el acceso a esta por parte de cada tarea utilizando semáforos o MUTEX, y una tercer tarea que obtenga los datos de la cola y los imprima en el LCD utilizando cada uno de los SOTR: FreeRTOS y Micro C/OS II.

A continuación se muestra un ejemplo de creación y utilización de semáforos con 3 tareas deferentes, y se despliega el resultado en los periféricos de la tarjeta de desarrollo.

En la primera parte, se presenta la codificación de un ejemplo en el cual se muestra la creación de un semáforos el cual controla el acceso a un recurso, en este caso el LCD con el que cuenta la tarjeta de desarrollo, utilizando el SOTR Micro C/OS II:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "includes.h"
#include "Perifericos.h"
#include "sys/time.h"
#include "alt_ucosii_simple_error_check.h"
#include <signal.h>
#include "altera_avalon_pio_regs.h"

static unsigned long ulLedStates;
#define partstNUM_LEDS          ( 8 )

/* Creación de las tareas */

#define    TASK_STACKSIZE       2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
```

```
/* Definición de prioridades de las tareas */

#define TASK1_PRIORITY      12
#define TASK2_PRIORITY      10
FILE *lcd;

/* Definición del semaforo */
OS_EVENT *shared_resource_sem;

void task1(void* pdata){
    INT8U return_code = OS_NO_ERR;
    OSSemPend (shared_resource_sem, 100, &return_code);// Solicito el
semaforo
   for( ;  ;  ){
            fprintf(lcd, "Tarea 1 sin sem");
             printf("1\n");
            OSSemPost (shared_resource_sem);//Retorno el semaforo
            OSTimeDlyHMSM(0, 0, 1, 0);
      }     }

void task2(void* pdata){
    INT8U return_code = OS_NO_ERR;
   for( ;  ;  ){
            OSSemPend   (shared_resource_sem,   100,   &return_code);//
Solicito el semaforo
            fprintf(lcd, "sin semáforo 2\n");
            printf("2\n");
            OSSemPost (shared_resource_sem);//Retorno el semaforo
            OSTimeDlyHMSM(0, 0, 0, 500);
      }     }

int main(void){
    lcd=fopen("/dev/lcd_display", "w");
    shared_resource_sem = OSSemCreate(1);
    if(shared_resource_sem!=NULL)printf("Semáforo creado\n");

OSTaskCreateExt(task1,   NULL,   (void   *)&task1_stk[TASK_STACKSIZE],
TASK1_PRIORITY,
                  TASK1_PRIORITY, task1_stk, TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(task2,   NULL,   (void   *)&task2_stk[TASK_STACKSIZE],
TASK2_PRIORITY,
                  TASK2_PRIORITY, task2_stk, TASK_STACKSIZE, NULL, 0);
  OSStart();
  return 0;
}
```

En la segunda parte, se presenta la codificación del ejemplo anterior en el SOTR FreeRTOS:

```
/* Standard includes. */
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
```

```
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "altera_avalon_pio_regs.h"
#include <Perifericos.h>
#define STACK_SIZE       configMINIMAL_STACK_SIZE

/* Variables globales */

FILE *lcd;
void vTarea1( void * pvParameters );
void vTarea2( void * pvParameters );

static xSemaphoreHandle sem_lcd=NULL;

void vTarea1( void * pvParameters ) {
    int t11=0;
    alt_u8 t1=0;
     for( ;  ;  ){
      t1++;
      if ( xSemaphoreTake( sem_lcd, 500 ) == pdTRUE ) { // Solicito el
semaforo
            fprintf(lcd, "T1: %u\n", t1);
             printf("1\n");
             xSemaphoreGive( sem_lcd );//Retorno el semaforo
        }
        else {printf("semaforo no tomado 1\n");}
        t11= (rand() % 255) + 0;
        IOWR_ALTERA_AVALON_PIO_DATA( LED_PIO_BASE, t11 );
        vTaskDelay(  300 );
      }     }

 void vTarea2( void * pvParameters ){
   int t22=0;
   alt_u8 i=0;
     for( ;  ;  ){
        i++;
        if ( xSemaphoreTake( sem_lcd, 500 ) == pdTRUE ) {   // Solicito
el semaforo
            fprintf(lcd, "T2: %u\n", i);
            printf("2\n");
             xSemaphoreGive( sem_lcd );//Retorno el semaforo
        }
        else {printf("semaforo no tomado 2\n");}
        t22= (rand() % 255) + 0;
        Setsegmentos(t22);
        vTaskDelay(  200 );
      }     }

int main( void )
{
   lcd=fopen("/dev/lcd_display", "w");
    usleep(500000);
    static unsigned char ucParameterToPass;
    //inicializo el semaforo
    vSemaphoreCreateBinary(sem_lcd);
    if(sem_lcd!=NULL)printf("semaforo creado");
    xTaskHandle xHandle1, xHandle2;
    xTaskCreate( vTarea1, "TAREA1", STACK_SIZE, &ucParameterToPass, 4,
&xHandle1 );
```

```
    xTaskCreate( vTarea2, "TAREA2", STACK_SIZE, &ucParameterToPass, 5,
&xHandle2 );
    vTaskStartScheduler();
   return 0;
}
```

En la siguiente sección se muestra un ejemplo de creación y manejo de una Cola, en la que se crean tres tareas y una cola por tarea, una cuarta tarea toma los datos de la cola y los muestra en los LED, el LCD y el Display de 7 segmentos, respectivamente, en la primera parte se muestra el desarrollo sobre el SOTR Micro C/OS II:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "includes.h"
#include "Perifericos.h"
#include "sys/time.h"
#include "altera_avalon_pio_regs.h"

static unsigned long ulLedStates;
#define partstNUM_LEDS          ( 8 )

/* Definición de tareas */

#define   TASK_STACKSIZE       2048
OS_STK    ContaLed_stk[TASK_STACKSIZE];
OS_STK    ContaDis_stk[TASK_STACKSIZE];
OS_STK    ContaLcd_stk[TASK_STACKSIZE];
OS_STK    ImprimeTask_stk[TASK_STACKSIZE];

/* Establecimiento de las prioridades de cada tarea */

#define CONTALED_PRIORITY         10
#define CONTADIS_PRIORITY         11
#define CONTALCD_PRIORITY         12
#define IMPRIMETASK_PRIORITY      13


/* Definición de la cola de mensajes: tamaño del mensaje */
#define   MSG_QUEUE_SIZE  50
/*Puntos de manejo de la cola */
OS_EVENT  *msgqueue1;
OS_EVENT  *msgqueue2;
OS_EVENT  *msgqueue3;
/*Almacenamiento de mensajes*/
void      *msgqueueTbl1[MSG_QUEUE_SIZE];
void      *msgqueueTbl2[MSG_QUEUE_SIZE];
void      *msgqueueTbl3[MSG_QUEUE_SIZE];

FILE *lcd;

void ContaLed(void* pdata){
  volatile alt_u8 contador=0;
        for( ;  ;  ){
        contador++;
      OSQPost (msgqueue1, (void *)contador);
         OSTimeDlyHMSM(0, 0, 0, 200);
      }     }

void ContaDis(void* pdata){
   volatile alt_u8 contador=0;
```

```
        for( ;  ;  ){
        contador=contador + 2;
      OSQPost (msgqueue2, (void *)contador);
         OSTimeDlyHMSM(0, 0, 0, 200);
      }     }

void ContaLcd(void* pdata){
    volatile alt_u8 contador=0;
        for( ;  ;  ){
         contador=contador + 5;
         OSQPost (msgqueue3, (void *)contador);
         OSTimeDlyHMSM(0, 0, 0, 200);
              }      }

void ImprimeTask(void* pdata){
   alt_u8 temp_meas1;
   alt_u8 temp_meas2;
   alt_u8 temp_meas3;

     while(1){
      temp_meas1 = OSQPend(msgqueue1, 200, &temp_meas1);
             /* imprime el resultado en un puerto */
             printf("Contador #%u\n", temp_meas1);
             IOWR_ALTERA_AVALON_PIO_DATA( LED_PIO_BASE, temp_meas1 );

      temp_meas2 = OSQPend(msgqueue2, 200, &temp_meas2);
             printf("Contador #%u\n", temp_meas2);
             Setsegmentos(temp_meas2);

      temp_meas3 = OSQPend(msgqueue3, 200, &temp_meas3);

             printf("Contador #%u\n", temp_meas3);
             fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
             fprintf(lcd, "lcd: %u", temp_meas3 );
    printf("imprime cola\n");
  }     }

int main(void){
    lcd=fopen("/dev/lcd_display", "w");

  msgqueue1 = OSQCreate(&msgqueueTbl1[0], MSG_QUEUE_SIZE);
  msgqueue2 = OSQCreate(&msgqueueTbl2[0], MSG_QUEUE_SIZE);
  msgqueue3 = OSQCreate(&msgqueueTbl3[0], MSG_QUEUE_SIZE);

  OSTaskCreateExt(ContaLed,                NULL,                (void
*)&ContaLed_stk[TASK_STACKSIZE], CONTALED_PRIORITY,
            CONTALED_PRIORITY, ContaLed_stk, TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(ContaDis,                NULL,                (void
*)&ContaDis_stk[TASK_STACKSIZE], CONTADIS_PRIORITY,
            CONTADIS_PRIORITY, ContaDis_stk, TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(ContaLcd,                NULL,                (void
*)&ContaLcd_stk[TASK_STACKSIZE], CONTALCD_PRIORITY,
           CONTALCD_PRIORITY, ContaLcd_stk, TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(ImprimeTask,             NULL,                (void
*)&ImprimeTask_stk[TASK_STACKSIZE], IMPRIMETASK_PRIORITY,
                  IMPRIMETASK_PRIORITY,            ImprimeTask_stk,
TASK_STACKSIZE, NULL, 0);
  OSStart();
```

```
  return 0;
}
```

A continuación se muestra el ejemplo anterior codificado en el SOTR FreeRTOS:

```
/* Standard includes. */
#include <stddef.h>
#include <stdio.h>
#include <string.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "altera_avalon_pio_regs.h"
#include <Perifericos.h>

#define STACK_SIZE       configMINIMAL_STACK_SIZE

/* Variables globales */
xQueueHandle xQueue, xQueue1, xQueue2;
FILE *lcd;
void vImprimeTask( void * pvParameters );
void vContaTask( void * pvParameters );

void vContaLed( void * pvParameters ){
     volatile alt_u8 contador=0;
     for( ;  ;  ){
      contador++;
      xQueueSend(xQueue, &contador, 0);
      vTaskDelay(  200 );
      }     }

 void vContaDis( void * pvParameters ){
     volatile alt_u8 contador=0;
     for( ;  ;  ){
      contador=contador + 2;
      xQueueSend(xQueue1, &contador, 0);
      vTaskDelay(  200 );
      }     }

 void vContaLcd( void * pvParameters ){
     volatile alt_u8 contador=0;
     for( ;  ;  ){
      contador=contador+5;
      xQueueSend(xQueue2, &contador, 0);
      vTaskDelay(  200 );
      }     }

void vImprimeTask( void * pvParameters ){
    alt_u8 temp_meas;
     while(1){
      if (xQueueReceive(xQueue, &temp_meas, portMAX_DELAY)){
             /* Imprime el resultado en el puerto */
             printf("Contador #%u\n", temp_meas);
             IOWR_ALTERA_AVALON_PIO_DATA( LED_PIO_BASE, temp_meas );
         }

      if (xQueueReceive(xQueue1, &temp_meas, portMAX_DELAY)) {
          printf("Contador #%u\n", temp_meas);
```

```
            Setsegmentos(temp_meas);
        }

      if (xQueueReceive(xQueue2, &temp_meas, portMAX_DELAY)){
            printf("Contador #%u\n", temp_meas);
            fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
            fprintf(lcd, "lcd: %u", temp_meas );
        }
       printf("imprime cola\n");
     }      }

int main( void ) {
   lcd=fopen("/dev/lcd_display", "w");
    static unsigned char ucParameterToPass;
    xTaskHandle xHandle1, xHandle2, xHandle3, xHandle4;

   xQueue = xQueueCreate (5, sizeof (alt_u8));
   xQueue1 = xQueueCreate (5, sizeof (alt_u8));
   xQueue2 = xQueueCreate (5, sizeof (alt_u8));

     if ( (xQueue != 0) && (xQueue1 != 0)&& (xQueue2 != 0) ){      //Si
pudo crear las colas creo las tareas
    xTaskCreate( vContaLed, "CUENTAL", STACK_SIZE, &ucParameterToPass,
5, &xHandle1 );
    xTaskCreate( vContaDis, "CUENTAD", STACK_SIZE, &ucParameterToPass,
4, &xHandle2 );
    xTaskCreate(        vContaLcd,        "CUENTALC",        STACK_SIZE,
&ucParameterToPass, 3, &xHandle3 );
    xTaskCreate(       vImprimeTask,       "IMPRIME",       STACK_SIZE,
&ucParameterToPass, 2, &xHandle4 );
   }
    vTaskStartScheduler();
      return 0;
}
```

A continuación se presenta la codificación de un ejemplo en el cual se muestra la creación de un MUTEX, utilizado para hacer exclusión mutua para poder acceder al LCD, creado sobre el SOTR Micro C/OS II:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "includes.h"
#include "Perifericos.h"
#include "sys/time.h"
#include "alt_ucosii_simple_error_check.h"
#include <signal.h>
#include <sys/alt_irq.h>
#include "altera_avalon_pio_regs.h"

static unsigned long ulLedStates;
#define partstNUM_LEDS          ( 8 )

/* Definición de tareas */

#define   TASK_STACKSIZE       2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];

/* Definición de prioridad en las tareas */
```

```
#define TASK1_PRIORITY      11
#define TASK2_PRIORITY      10

FILE *lcd;

/* Definición del MUTEX */
OS_EVENT *resource_Mutex;

static void NewPrintString(const char  *pcString) {
    INT8U return_code = OS_NO_ERR;
         OSMutexPend (resource_Mutex, 200, &return_code);
            fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
            fprintf(lcd, "%s",pcString);
         OSMutexPost (resource_Mutex);
  }

void task1(void* pdata){
   char *pcStringToPrint;
  pcStringToPrint = ( char * ) pdata;
        for( ;; ) {
             NewPrintString( pcStringToPrint );
              printf("tarea 1 \n");
             OSTimeDlyHMSM(0, 0, 0, 700);
         }      }

void task2(void* pdata){
    char *pcStringToPrint;
    pcStringToPrint = ( char * ) pdata;

       for( ;; )  {
                 NewPrintString( pcStringToPrint );
                 usleep(1000000);
                 printf("tarea 2 \n");
                 OSTimeDlyHMSM(0, 0, 0, 500);
           }       }

int main(void)
{INT8U return_code = OS_NO_ERR;
    lcd=fopen("/dev/lcd_display", "w");
    resource_Mutex = OSMutexCreate (9, &return_code);
    printf("Mutex creado\n");
OSTaskCreateExt(task1,      "Task      1      *********\n",       (void
*)&task1_stk[TASK_STACKSIZE],
                  TASK1_PRIORITY,      TASK1_PRIORITY,      task1_stk,
TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(task2,      "Task      2      ---------\n",       (void
*)&task2_stk[TASK_STACKSIZE],
                  TASK2_PRIORITY,      TASK2_PRIORITY,      task2_stk,
TASK_STACKSIZE, NULL, 0);
  OSStart();
  return 0;
}
```

A continuación se presenta el ejemplo de codificación de un MUTEX sobre el SOTR FreeRTOS:

```
///* Standard includes. */
#include <stddef.h>
#include <stdio.h>
#include <string.h>
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <sys/alt_irq.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "altera_avalon_pio_regs.h"
#include <Perifericos.h>

#define STACK_SIZE       configMINIMAL_STACK_SIZE
void vTarea1( void * pvParameters );
void vTarea2( void * pvParameters );

static xSemaphoreHandle xMutex;
FILE *lcd;

static void prvNewPrintString( const portCHAR *pcString )
{ xSemaphoreTake( xMutex, portMAX_DELAY );
    {  fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
       fprintf(lcd, "%s", pcString ); }
    xSemaphoreGive( xMutex );
}

static void prvPrintTask1( void *pvParameters )
{ char *pcStringToPrint;
  pcStringToPrint = ( char * ) pvParameters;
   for( ;; ) {
   prvNewPrintString( pcStringToPrint );
   vTaskDelay( 700); }
}

static void prvPrintTask2( void *pvParameters )
{ char *pcStringToPrint;
  pcStringToPrint = ( char * ) pvParameters;
  for( ;; ) {
   prvNewPrintString( pcStringToPrint );
   usleep(1000000);
   vTaskDelay( 500); }
}

int main( void ) {
 lcd=fopen("/dev/lcd_display", "w");
    xMutex = xSemaphoreCreateMutex();
    srand( 567 );

    if( xMutex != NULL ) {
xTaskCreate( prvPrintTask1, "Print1", 1000, "Task 1 *********\n", 1,
NULL );
        xTaskCreate( prvPrintTask2, "Print2", 1000, "Task 2 ---------
\n", 2, NULL );
        vTaskStartScheduler(); }
    for( ;; );
}
```

SISTEMAS DE TIEMPO REAL

A.5. PRACTICA No. 5
ADMINISTRACIÓN DE TIEMPO

OBJETIVO:

- Afianzar los conocimientos de los conceptos adquiridos en el estudio de la administración de tiempo que están disponibles para su uso en sistemas operativos de tiempo real.

- Iniciar al estudiante para la implementación de un Sistema de Tiempo Real donde sea necesario el uso de los conceptos estudiados.

CONCEPTOS GENERALES

En el trabajo con Sistemas en Tiempo Real, la noción de tiempo y sobre todo de su administración es primordial desde todo punto de vista, ya que la exactitud con que los procesos, las tareas, los retardos, etc., se lleven a cabo es la base de su funcionamiento. Por lo tanto, adquirir los fundamentos en la administración del tiempo toma gran importancia en la formación dentro de un laboratorio cuyo propósito sea ofrecer experiencias didácticas de formación sobre Sistemas en Tiempo Real.

Tiempo

Para hacer posible la administración del tiempo se requiere que se proporcione un método de obtener una interrupción periódica para mantener un seguimiento de retardos de tiempo y tiempos de espera. Esta fuente de tiempo periódica se llama señal de reloj y debe ocurrir entre 10 y 100 veces por segundo o hercios. La frecuencia real de la señal de reloj depende de la resolución deseada para el trabajo de de la aplicación que la va a necesitar. Sin embargo, cuanto mayor sea la frecuencia de las señales o tics, mayor es la sobrecarga en el uso de los recursos del sistema.

Tic de reloj

Una señal de reloj es una interrupción especial que se produce periódicamente. Este ciclo de interrupciones puede ser visto como el ritmo cardíaco del sistema. El tiempo entre las interrupciones es específico de la aplicación que se esté desarrollando y por lo general se encuentra entre los 10 y 200 ms. Este tic de reloj hace posible que se permita interrumpir o retrasar las tareas un número entero conocido de tics de reloj, y así contar con tiempos de espera cuando las tareas están a la espera de posibles acontecimientos que se produzcan. Es posible permitir que las tareas presenten retrasos por un determinado número de tics de reloj. La resolución en los retrasos de

las tareas es de 1 pulso de reloj, sin embargo, esto no significa que su precisión sea exactamente de 1tic de reloj.

PROCEDIMIENTO:

Después de realizar un estudio de los conceptos que definen las características del manejo de tiempo en los STR. Utilizando el código ejemplo en el que

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "includes.h"
#include "Perifericos.h"
#include "sys/time.h"
static unsigned long ulLedStates;
#define partstNUM_LEDS          ( 8 )

/* Definición de tareas */

#define   TASK_STACKSIZE       1024

OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
OS_STK    task3_stk[TASK_STACKSIZE];
OS_STK    task4_stk[TASK_STACKSIZE];

/* Definición de prioridad en las tareas */

#define TASK2_PRIORITY      11
#define TASK3_PRIORITY      12
#define TASK4_PRIORITY      13

FILE *lcd;
unsigned int contador;

void task2(void* pdata){
alt_u8 t1=0;

  while (1){
    t1++;
    IOWR_ALTERA_AVALON_PIO_DATA( LED_PIO_BASE, t1 );
    OSTimeDlyHMSM(0, 0, 0, 100);
      OSTimeDly (100);
       }        }

void task3(void* pdata){
alt_u8 t2=0;
  while (1){
      t2++;
      Setsegmentos(t2);
       OSTimeDlyHMSM(0, 0, 0, 100);
       }        }

void task4(void* pdata){
alt_u8 t3=0;

    while(1){
        t3++;
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "  T3: %d", t3);
```

```
        OSTimeDlyHMSM(0, 0, 0, 100);
  }     }

int main(void){
  lcd=fopen("/dev/lcd_display", "w");

  OSTaskCreateExt(task2,   NULL,   (void   *)&task2_stk[TASK_STACKSIZE],
TASK2_PRIORITY,
                  TASK2_PRIORITY, task2_stk, TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(task3,   NULL,   (void   *)&task3_stk[TASK_STACKSIZE],
TASK3_PRIORITY,
                  TASK3_PRIORITY, task3_stk, TASK_STACKSIZE, NULL, 0);

 OSTaskCreateExt(task4,   NULL,   (void   *)&task4_stk[TASK_STACKSIZE],
TASK4_PRIORITY,
                  TASK4_PRIORITY, task4_stk, TASK_STACKSIZE, NULL, 0);
 OSStart();
  return 0;
}
```

Y el desarrollo del siguiente ejemplo codificado en le SOTR FreeRTOS se presenta a continuación:

```
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"

#include "Perifericos.h"

#define STACK_SIZE   configMINIMAL_STACK_SIZE

 void secuencia();
 void vTaskCode1( void *pvParameters );
 void vTaskCode2( void *pvParameters );
 void vTaskCode3( void *pvParameters );

 xTaskHandle xHandle1, xHandle2, xHandle3;
 FILE *lcd;
 unsigned int contador;

void vTaskCode1( void * pvParameters ){
    int t1=0;
    portTickType xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();

     while(1){
       t1++;
       IOWR_ALTERA_AVALON_PIO_DATA( LED_PIO_BASE, t1 );
       vTaskDelayUntil( &xLastWakeTime, 100 );
           }         }

void vTaskCode2( void * pvParameters ){
   int t2=0;
     while(1){
        t2++;
```

```
        Setsegmentos(t2);
        vTaskDelay( 100/portTICK_RATE_MS );
       }  }

 void vTaskCode3( void * pvParameters ){
    alt_u8 t3=0;

    while(1){
        t3++;
        fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
        fprintf(lcd, "  T3: %d", t3);
        vTaskDelay( 100/portTICK_RATE_MS );
       }      }

int main( void ){
  lcd=fopen("/dev/lcd_display", "w");

    static unsigned char ucParameterToPass;

    xTaskCreate( vTaskCode1, "NAME1", STACK_SIZE, &ucParameterToPass,
2, &xHandle1 );
    xTaskCreate( vTaskCode2, "NAME2", STACK_SIZE, &ucParameterToPass,
3, &xHandle2 );
    xTaskCreate( vTaskCode3, "NAME3", STACK_SIZE, &ucParameterToPass,
4, &xHandle3 );
    vTaskStartScheduler();
    return 0;
}
```

SISTEMAS DE TIEMPO REAL

A.6. PRACTICA No. 6
EJECUTIVO CICLICO

OBJETIVO:

- Verificar la apropiación y correcto uso de los conceptos de ejecutivos cíclicos y planificación estática de tareas.
- Comprobar el manejo de las condiciones para planificar un conjunto de tareas utilizando un ejecutivo cíclico.
- Conocer el nivel de comprensión de las clases teóricas y su aplicabilidad.

CONCEPTOS GENERALES:

Los STR se pueden presentar como un conjunto de tareas de control y/o de supervisión. Éstas tienen unos plazos de activación y respuesta estrictos. Existen multitud de métodos a emplear para resolver el problema, la mayor parte de los cuales requieren la existencia de un Sistema Operativo de Tiempo Real para administrar correctamente los recursos de la máquina. Dado que existe una inmensa cantidad de STR con una complejidad *software* relativamente baja, en ocasiones es más adecuado el uso de una planificación de tareas estática, realizada durante el diseño del sistema. Si todas las tareas que intervienen en el mismo son periódicas y de tiempo de cómputo conocido, podríamos beneficiarnos del ejecutivo cíclico que está especialmente diseñado para sistemas con ese tipo de tareas.

El ejecutivo cíclico realiza una planificación muy sencilla que no requiere el uso de interrupciones, por lo que está especialmente indicado para los sistemas que carecen de las mismas. El hecho de que el sistema no desaloje forzosamente a los programas que se encuentran en ejecución, elimina muchas complicaciones:

- Sólo existe un hilo de ejecución a nivel de aplicación. Esto indica que no se usarán funciones para la gestión de hilos.
- Desaparecen los problemas de compartición de recursos. Implícitamente todos los recursos se utilizan en exclusión mutua, incluida la CPU.
- No existen funciones de sincronización.
- Los mecanismos de comunicación entre tareas son muy simples.

- Es relativamente sencillo determinar la ejecutabilidad del sistema.

PROCEDIMIENTO:

Tomando como base el ejemplo de creación de un ejecutivo cíclico presentado a continuación, se debe crear uno para 3 tareas. Estas tareas deben imprimir un mensaje con su activación y desactivación para verificar su ejecución, y verificar que se cumplan los tiempos establecidos en el diseño.

A continuación se presenta un ejemplo donde se crea un ejecutivo cíclico que planifica la ejecución de 4 tareas.

```
#include<time.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/alt_alarm.h>


#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"

void tareaA();
void tareaB();
void tareaC();
void tareaD();
void sistema();

int main(void)
    {
        sistema();
    }

void sistema()
    {   int ciclo=0;

    while (1) {

            usleep(100000);
            switch (ciclo) {
                            case 0: tareaA(); tareaB(); tareaC();
tareaD();break;
                            case 1: tareaA(); tareaB(); break;
                            case 2: tareaC(); tareaA(); tareaB();
break;
                            case 3: tareaB(); tareaA(); break; }
            ciclo++;
            if(ciclo>3) {ciclo=0;
printf("\n\n**********************\n");}}}

void tareaA(){
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 1);
            usleep(40000);
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0);
            printf("TareaA()\n"); }

void tareaB(){
            OWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 2);
            usleep(20000);
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0);
```

```
            printf("TareaB()\n"); }

void tareaC(){
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 4);
            usleep(30000);
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0);
            printf("TareaC()\n"); }

void tareaD(){
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 8);
            usleep(10000);
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0);
            printf("TareaD()\n"); }
```

SISTEMAS DE TIEMPO REAL

A.7. PRACTICA No. 7
PRACTICA FINAL

OBJETIVO:

- Poner en práctica los conceptos y experiencias obtenidas a lo largo del desarrollo de las guías realizadas en el curso.

DESCRIPCIÓN:

En esta práctica se busca realizar un ejercicio que incluya algunos de los conceptos que fueron objeto de experimentación durante el desarrollo del curso con el fin de afianzar estos conocimientos y fortalecer las habilidades del estudiante para iniciar nuevos desarrollos en los que se trabaje con Sistemas de Tiempo Real.

PROCEDIMIENTO:

De acuerdo a lo aprendido durante el desarrollo de las prácticas, proponer e implementar un proyecto donde se utilice uno de los SOTR estudiados en el curso, haciendo uso de los periféricos de entrada y salida disponibles en el kit de desarrollo Stratix II con procesador adaptable Nios II.

ANEXO B
MANUAL DE USUARIO SISTEMA DE MONITOREO Y DEPURACION NIOS II

Este manual presenta el uso de esta herramienta y los pasos que se deben seguir para realización correcta de la depuración de las prácticas realizadas.

B.1 Manual Interfaz de usuario

1. Iniciar la herramienta, el acceso directo se encuentra en el escritorio con el nombre de **Cliente Nios II**, la interfaz se muestra en la Figura B**.**

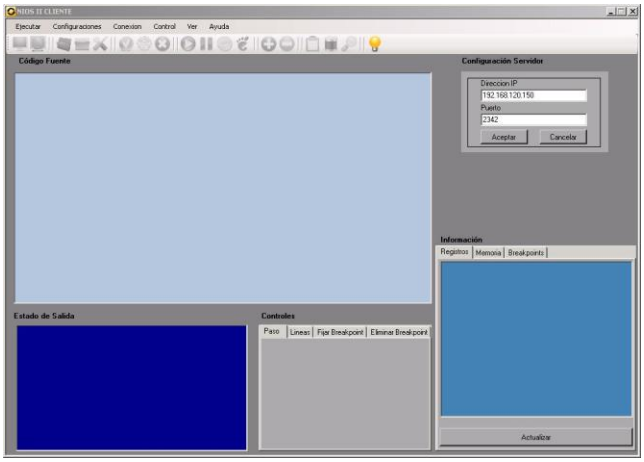


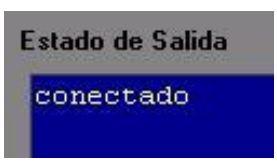
Figura B2 Interfaz de depuración

2. Introducir dirección IP y puerto del equipo Servidor en el cual se encuentra conectada la tarjeta de desarrollo Stratix II, Nios II, luego Aceptar Figura B.

Figura B3 Configuración de Servidor

3. El botón que permite conectarse al servidor se activara, al presionarlo, indicara através de un mensaje si la conexión se realizo como se muestra en la

4. Figura B, o si el servidor está ocupado, si el servidor se encuentra ocupado, se deshabilitara la opción de conexión, hasta recibir el mensaje de **recurso**



liberado como se muestra en la Figura B.

a. Icono conexión **b. mensajes de conexión en consola cuadro de texto**

Figura B4 Conexión al servidor

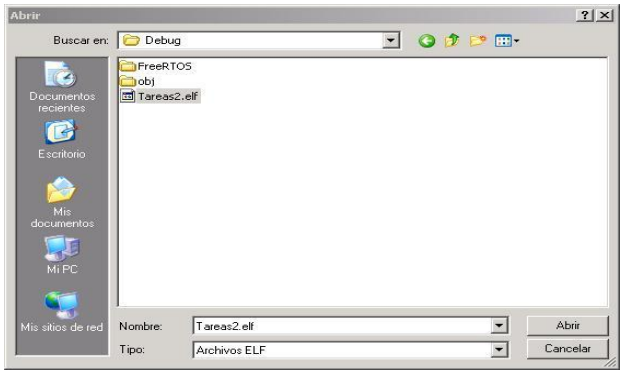
a. Mensajes de servidor ocupado

b. barra de iconos desactivada

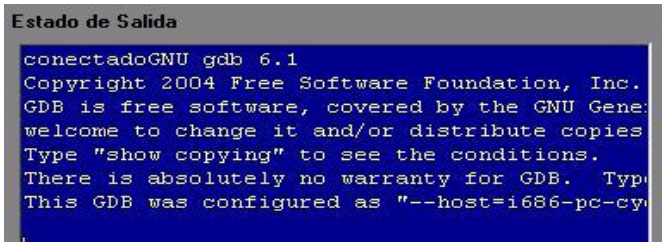
Figura B5 Mensaje y estado de desconexión

5. Si se logra la conexión se activaran las opciones de desconectar y Abrir Archivo ejecutable, como se muestra en la Figura B a. b. y c., las cuales permiten desconectarse del servidor y abrir el ejecutable de la aplicación de extensión **.elf,** el cual se encuentra en la carpeta **C:/Nombre_del_proyecto/debug/ejecutable.elf**, que luego se cargara en la tarjeta de desarrollo.

a. Icono de desconexión y de archivo activado



b. Cuadro de texto abrir archivo ejecutable

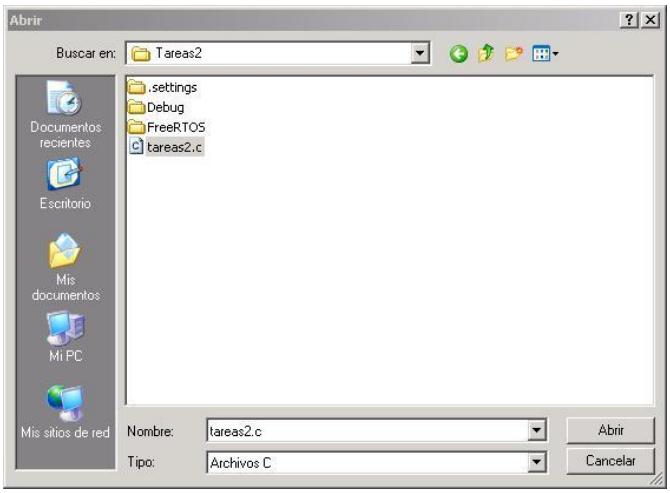


c. Mensaje de de archivo ejecutable abierto

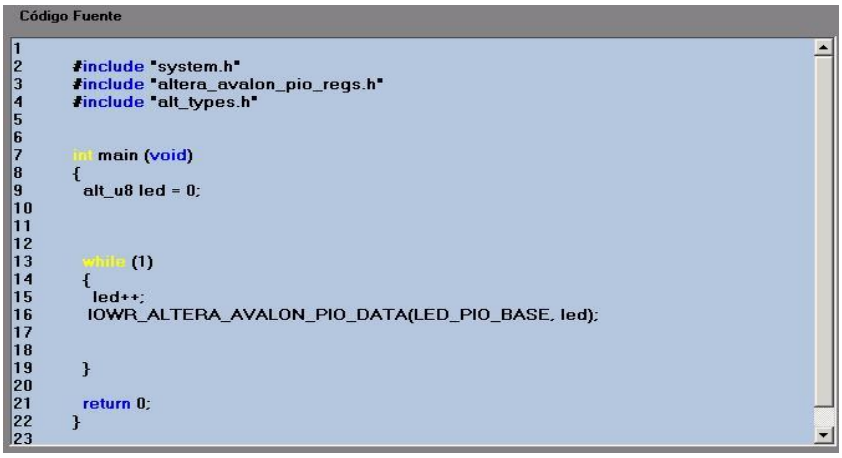
Figura B6 Proceso de apertura de archivo ejecutable

6. Después de abrir el archivo ejecutable, se activa la opción Abrir Código Fuente Figura B7, la que permite cargar el código desarrollado, cuya ubicación es **C:/Nombre_del_proyecto/codigo_fuente.c,** para ser mostrado en al campo código Fuente de la aplicación.

a. Icono cargar archivo de código



b. Cuadro de dialogo, cargar archivo de código



c. Cuadro de texto con el código del proyecto

Figura B7 Proceso de abrir código del proyecto

7. Se activa la opción Conectar, la cual permite conectarse remotamente a la tarjeta de desarrollo Figura B8.

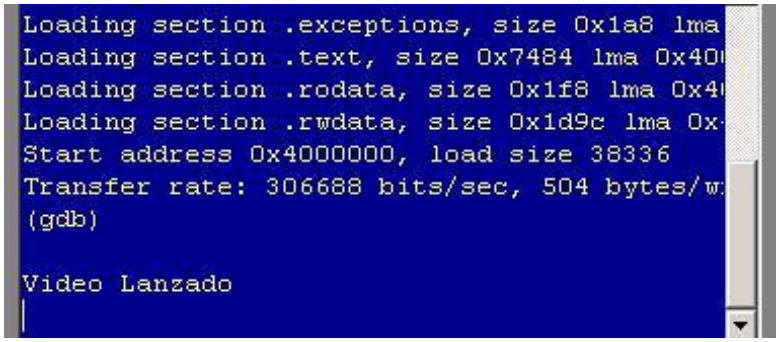
a. Icono de conexión con la tarjeta de desarrollo

b. Mensaje de conexión exitosa, muestra la IP equipo conectado

Figura B8 Conexión con la tarjeta de desarrollo

8. Ya conectado a la tarjeta de desarrollo, se activan las opciones Cargar y Desconectar, las cuales permiten descargar el archivo ejecutable .elf Figura B9, en la tarjeta y desconectarse de la tarjeta para cargar un nuevo archivo ejecutable y código fuente, he inicia la recepción de video desde el servidor, permitiendo visualizar el estado de los periféricos a través de la ejecución del código como se muestra en la Figura B9.

a. Activación del icono de programación del ejecutable sobre la tarjeta, e icono de desconexión de ésta



b. Mensaje de archivo cargado con éxito, y video lanzado

c. Video lanzado

Figura B9 Proceso de programación de la tarjeta y lanzamiento de video

9. Se activa la opción Ejecutar la cual permite poner en ejecución el código cargado en la tarjeta como se muestra en la Figura B10.

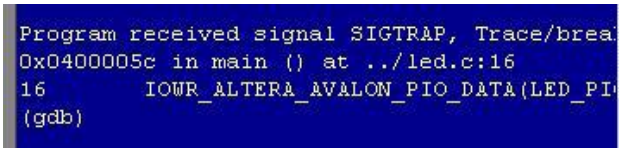
a. Icono de ejecución activado

b. Mensaje de programa en ejecución

Figura B10 Proceso de ejecución del programa sobre la tarjeta

10. La opción Detener se activa permitiendo parar la ejecución de la aplicación, Figura B11.

a. Icono de detención del programa activado



b. Mensaje de programa detenido

Figura B11 Proceso de detención del programa

11. Al detener la ejecución, se activan las opciones, Next, Step, Breakpoints, Quitar Breakpoint, Registros, Memoria, Ver Breakpoints como se muestra en la Figura B12.

Figura B12 Iconos de comandos de ejecución e información activados

12. Next Figura B13, permite la ejecución paso a paso del programa, sin entrar en el código de las funciones implementadas, la opción next del menú control activa la pestaña líneas la cual permite ingresar varios pasos a la vez.

a. Icono de detención del comando Next activado

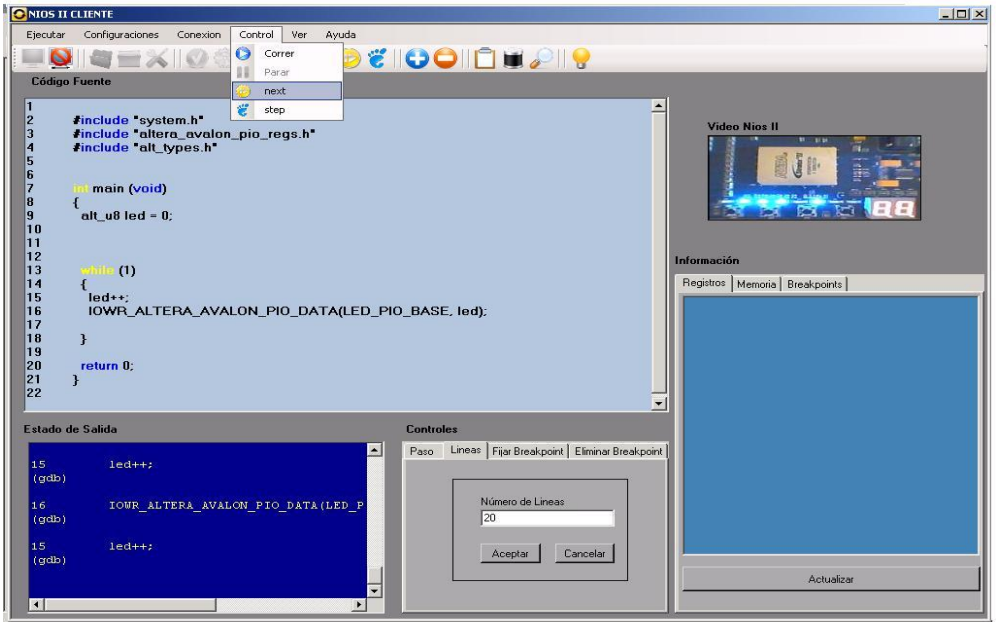


Figura B13 Ejecución del comando Next

13. Step Figura B14, permite la ejecución paso a paso del programa, entrando en el código de las funciones implementadas, la opción step del menú control activa la pestaña paso la cual permite ingresar varios pasos a la vez.

a. Icono de detención del comando Step activado

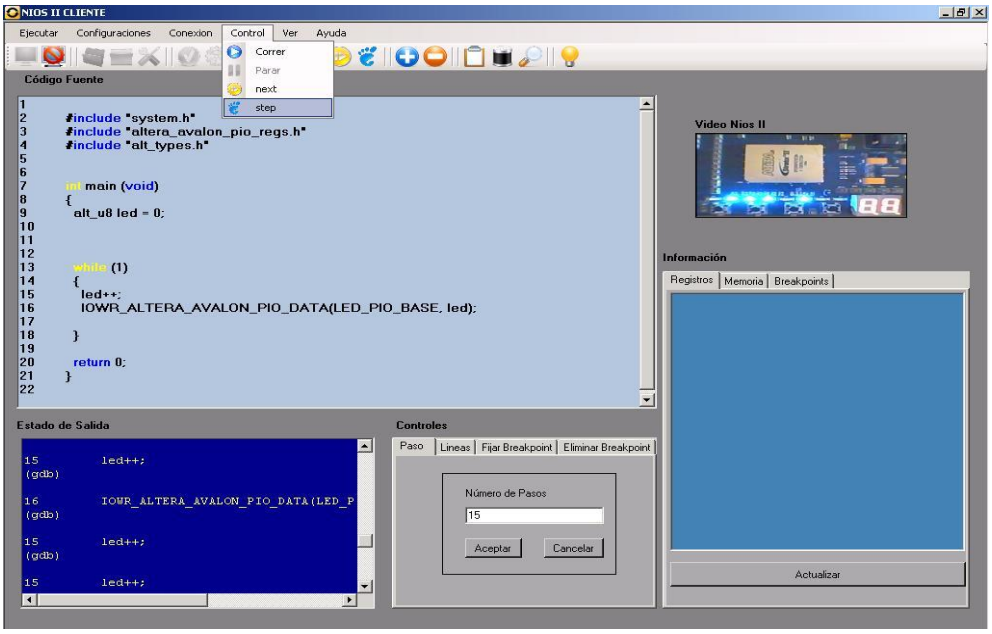
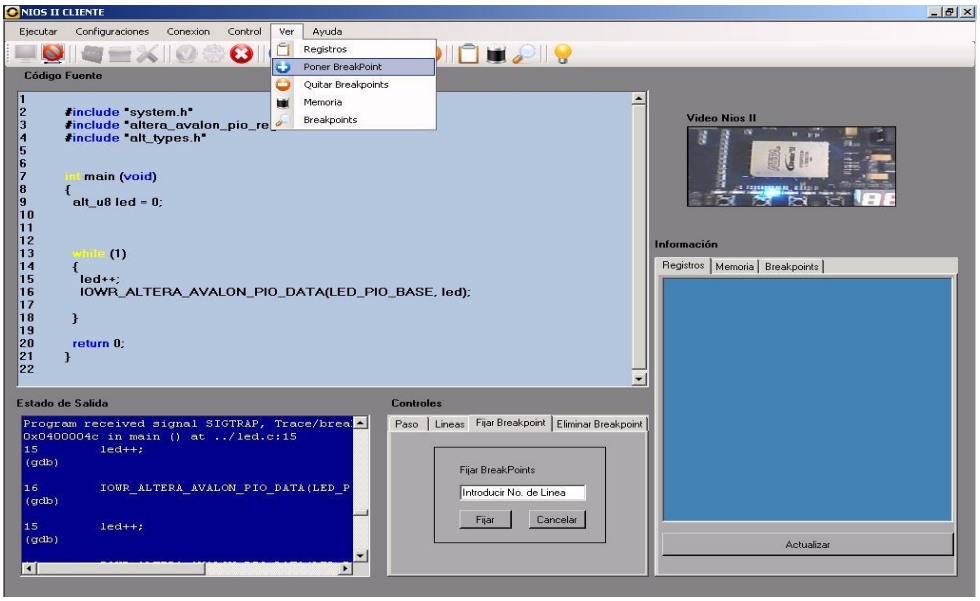


Figura B14 Proceso de ejecución del comando Step

14. Breakpoints y Quitar Breakpoints Figura B15, permiten insertar puntos de interrupción en la ejecución del programa, estas opciones también están

disponibles en el menú ver, estas activan las pestañas fijar Breakpoints y Eliminar Breakpoints, aquí se debe especificar la línea en la que se quiere establecer o eliminar los Breakpoints.

a. Icono de fijar y eliminar Breakpoints activados



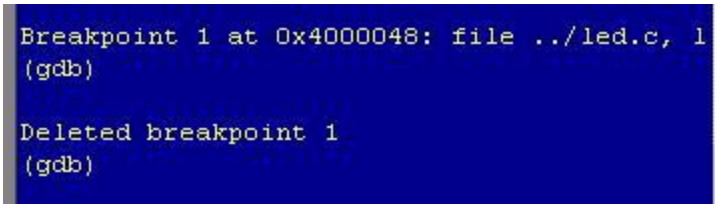


Figura B15 Proceso de fijar y eliminar Breakpoints y su respuesta en consola

15. Registros, Memoria, Ver Breakpoints, permiten visualizar el estado de los registros de datos del microprocesador, el estado de la memoria y los Breakpoints fijados en el código, cada uno de estos habilita su respectiva pestaña con la información requerida como se muestra en la Figura B16.

a. Iconos de información activados

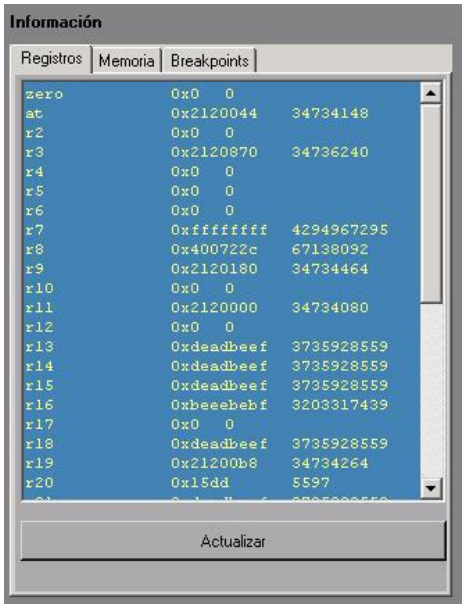


Figura B16 Visualización de estado de registros

16. Una vez terminada la depuración, se debe proceder a 1.Desconectarse de la tarjeta de desarrollo y 2. Desconectarse del servidor, 3. cerrar aplicación como se indica en la Figura B17.

Figura B17 Proceso de ejecución del comando Step

17. Luego de desconectarse del servidor, cerrar la aplicación utilizando la opción salir del menú ejecutar.

B.2 Manual Interfaz Servidor

1. Iniciar la herramienta, cuyo ejecutable se encuentra en la carpeta Servidor 2.7\Servidor2\AppServidor\AppCliente\bin\Debug\AppCliente.exe, la interfaz se muestra en la Figura B18.

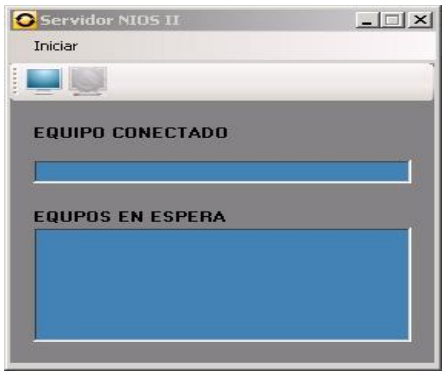


Figura B18 Interfaz del Servidor

2. Dar click en el botón Iniciar Server, para iniciar la escuche de peticiones de conexión, activando el cable de descarga *USB BLASTER,* permitiendo el acceso a la tarjeta y activando el botón Cerrar Server, estando conectado, mostrara la dirección IP y el puerto del equipo que ha accedido al recurso en el campo equipo conectado y en el campo equipos en espera, los equipos que no se pudieron conectar, ver Figura B19 , al los cuales se le enviara un mensaje de recurso liberado, una vez el equipo conectado se desconecte.

Figura B19 Usuario conectado y en espera

3. El botón Cerrar Server termina la escucha de peticiones de conexión.

ANEXO C
MANUAL DE CONFIGURACION DE PROYECOTS CON LOS SOTR FreeRTOS y Micro C/OS-II

C.1. Creación de un proyecto con el SOTR: Micro C/OS-II

Se presenta un ejemplo sencillo que ejecuta algunas características básicas del sistema operativo Micro C/OS-II. Figura C20 es un diagrama simplificado de la aplicación.

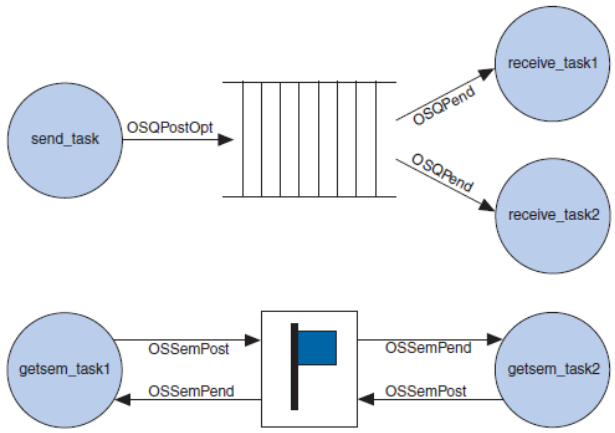


Figura C20 Ejemplo basico con Micro C/OS-II

Como se muestra en la figura, el diseño tiene cinco tareas activas. send_task llena una cola de mensajes con incremento de datos. receive_task1 y receive_task2 extraen periódicamente mensajes de la cola de mensajes. getsem_task1 y getsem_task2 compiten por un recurso compartido que está protegido por un semáforo. El diseño también tiene dos tareas, que no se muestra en la Figura C21: una para la inicialización y otra para imprimir la información de estado.

El proceso para crear una imagen de software de Micro C/OS-II para el procesador Nios II implica los siguientes pasos:

1. Crear un nuevo proyecto de Nios II IDE.
2. Configurar el proyecto de biblioteca de sistema de Nios II.
3. Generar y ejecutar el proyecto de software de Nios II.

Estos pasos se describen en la siguiente sección.

1. Crear un nuevo proyecto de Nios II IDE.

En esta sección se crea un nuevo proyecto de Nios II IDE mediante una plantilla de software. Realice los pasos siguientes:

1. En el menú Inicio de Windows, seleccione en Programas, **Altera, Nios II EDS 7.2** y haga clic en **Nios II 7.2 IDE**.

2. En el menú **Archivo**, seleccione **Nuevo** y haga clic en la aplicación de **Nios II c/c++**. Se abre la primera página del Asistente para nuevo proyecto.

3. En la parte de abajo, seleccionar la plantilla de proyecto, elija el **tutorial de Micro C/OS-II (*MicroC/OS-II Tutorial*)**. El nombre del proyecto y la ruta de proyecto se completan automáticamente. Mantenga estos valores predeterminados.

4. Haga clic en **Browse** debajo Seleccione el hardware destino.

5. Busque el directorio del ejemplo estándar para la placa de desarrollo de la Nios II: <altera>\<72>\<nios2eds >\<examples>\<verilog>**(versión de la tarjeta de la lista, para nosotros)**< niosII_stratixII_2s60_RoHS>\<standard>\

   - niosII_cyclone_1c20 para la placa de ciclón EP1C20.
   - niosII_cycloneII_2c35 para la placa de ciclón EP2C35.
   - niosII_stratix_1s10 para la placa de Stratix EP1S10.
   - niosII_stratix_1s10_ES para la placa de Stratix EP1S10ES.
   - niosII_stratix_1s40 para la placa de Stratix EP1S40.
   - niosII_stratixII_2s60 para la placa de Stratix EP2S60.
   - niosII_stratixII_2s60_ES para la placa de Stratix EP2S60ES.
   - **niosII_stratixII_2s60_rohs para la placa de ROHS de EP2S60 de Stratix.**

6. Haga clic en el archivo: **NiosII_stratixII_2s60_RoHS_standard_sopc.ptf**

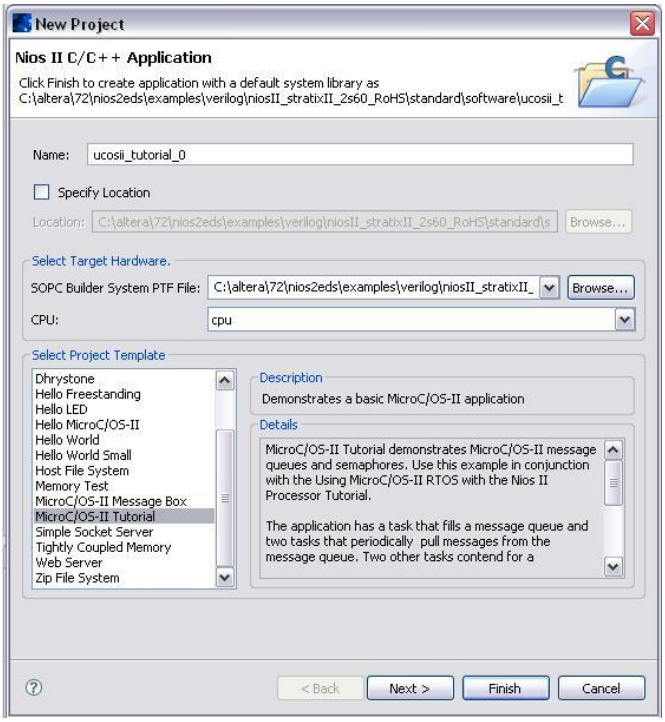


Figura C21 Asistente para nuevo proyecto, recuadro 1

7. Haga clic en **Open**, volver al asistente para **New Project**. Como se muestra en la Figura C2122, en el cuadro ***SOPC Builder System,*** Seleccione el hardware destino, que contiene la ruta de acceso al archivo **.ptf** para el diseño de ejemplo estándar. Además, el cuadro de CPU contiene el nombre de la CPU en el ejemplo de sistema generador de SOPC.

8. Haga clic en **Next** para ir a la segunda página del Asistente para el nuevo proyecto.

9. Seleccione la opción: ***Select or create a system library***.

10. Haga clic en ***New System Library Project*** para abrir la página de la biblioteca de sistema de Nios II. Consulte la Figura C2222.

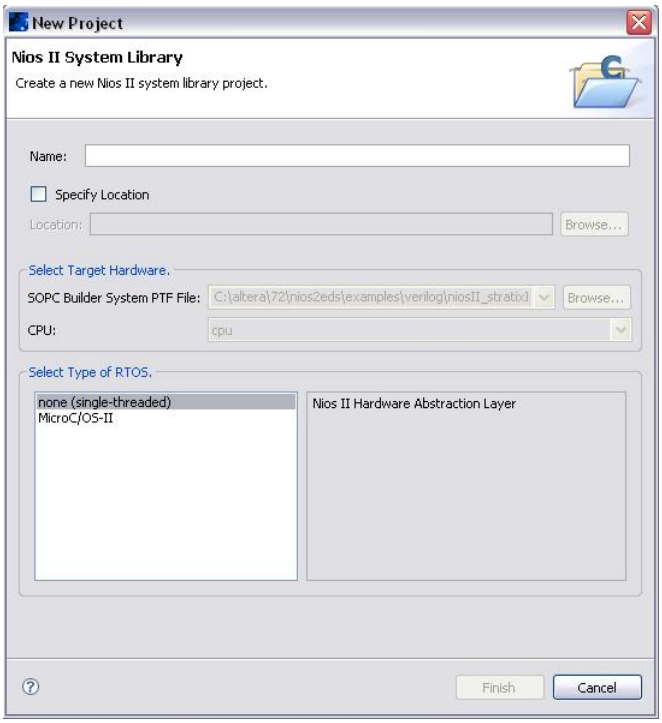


Figura C22 Cuadro de diálogo de Nuevo Sistema de Bibliotecas

11. En el cuadro **Name**, escriba **std_system_lib**.

12. Seleccione **MicroC/OS-II** en el cuadro **Select Type of RTOS**.

13. Haga clic en **Finish** para copiar los archivos de plantilla y volver al asistente para **New Project**. Consulte la Figura C2323

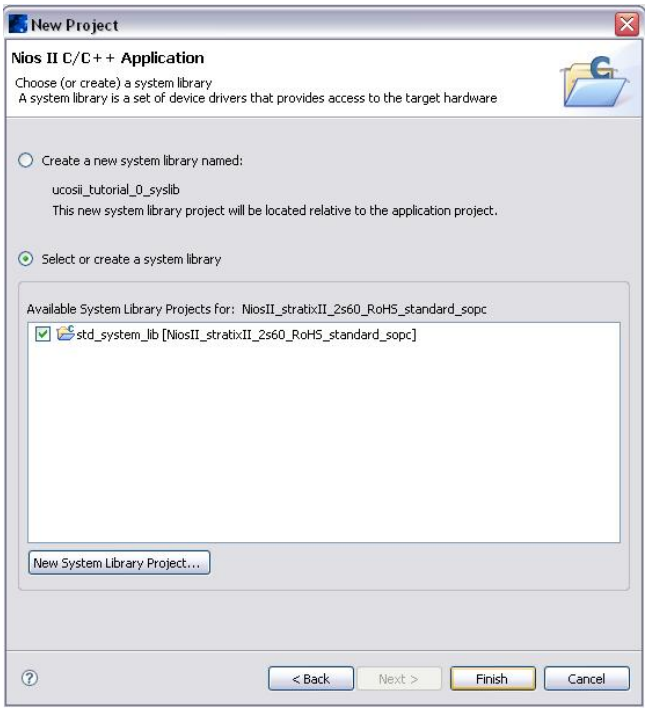


Figura C23 Asistente para Nuevo Proyecto, pagina 2

14. Haga clic en **Finish** para completar la creación de su nuevo proyecto. El asistente crea dos proyectos en el visor de proyectos de C/C ++ de Nios II, como se muestra en la Figura C2424 **ucosii_tutorial_0** es el proyecto de aplicación, y **std_system_lib** es el proyecto de biblioteca de sistema.

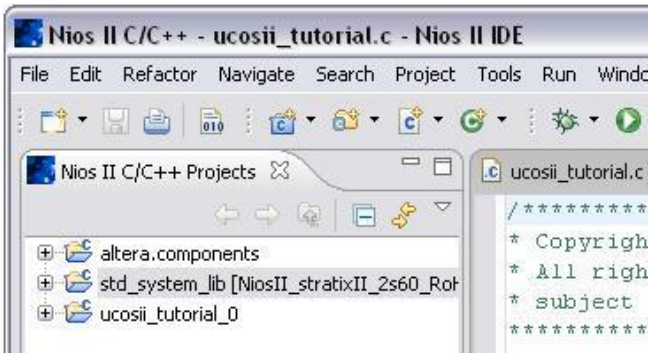


Figura C24 Proyectos de Nios II C/C++

Configurar la biblioteca de sistema.

En general, después de crear una librería nueva de sistema, se debe configurar, por ejemplo, la definición de stdin, stdout, stderr, etc. Ahora, se debe configurar MicroC/OS-II. Durante la configuración, el IDE de Nios II guarda los valores correspondientes en el archivo **system.h**. Realice los pasos siguientes para configurar el núcleo de MicroC/OS-II.

1. Haga clic derecho sobre la librería del sistema, **std_system_lib**, en la vista de **Nios II C/C ++ Projects**.

2. Presione clic derecho en librerías del sistema, en **Properties**, en el menú emergente para abrir su cuadro de diálogo. Si estás usando uC/OS-II en el modo de evaluación como distribuidos por Altera, puede que vea la notificación de la licencia en la Figura C2525 y Haga clic en Aceptar. El cuadro de diálogo de opciones de librerías del sistema aparece como se muestra en la Figura C26.

Figura C25 Notificación de Licencia de uC/OS-II

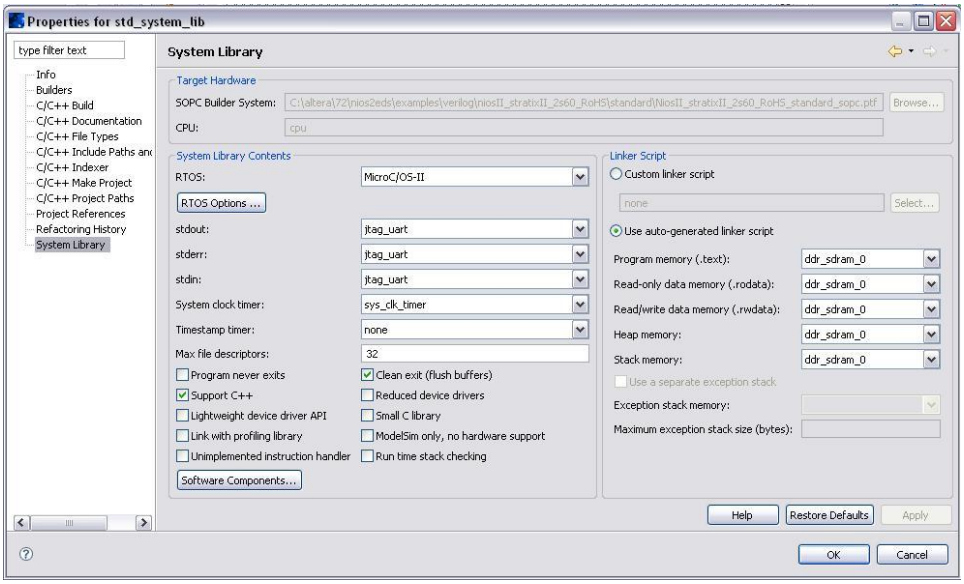


Figura C26 Opciones de Librerías del Sistema

2. Haga clic en **RTOS Options**. Se abre el cuadro de diálogo de opciones de RTOS de MicroC/OS-II, como se muestra en la Figura C2727.

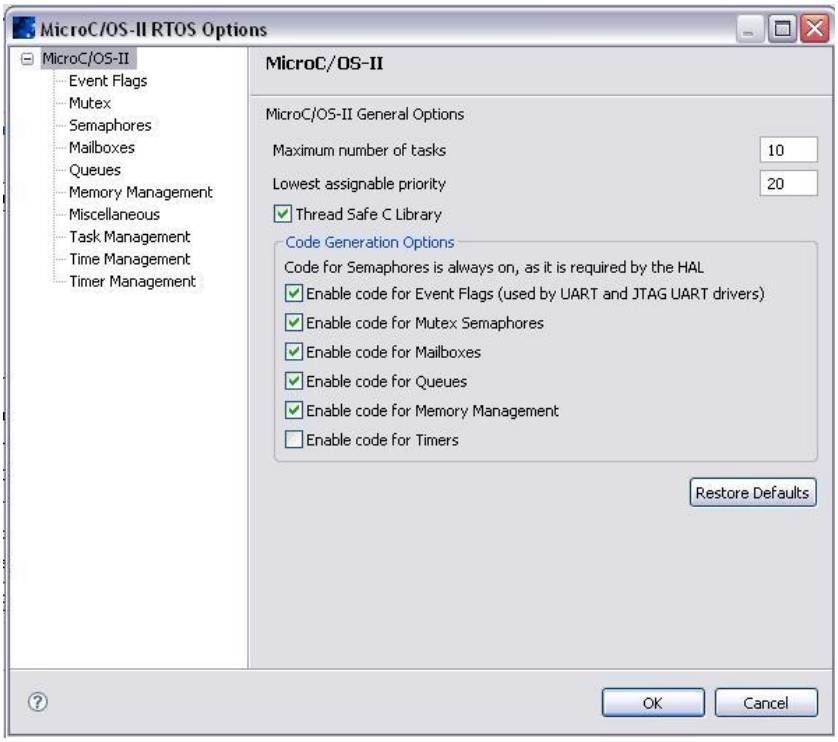


Figura C27 Opciones para Micro C/OS-II RTOS

3. Haga clic en "**+**" en el panel de la izquierda, para expandir el contenido de **MicroC/OS-II**. Se advierte que MicroC/OS-II es altamente configurable. Las opciones seleccionadas en este cuadro de diálogo se guardan en el archivo de **system.h** y determinan qué opciones de MicroC/OS-II están incluidas en la imagen binaria. Examinar las opciones que se pueden seleccionar haciendo clic en cada una de las categorías enumeradas en la parte izquierda del cuadro de diálogo.

4. Haga clic en Aceptar para utilizar la configuración predeterminada. Se vuelve al cuadro de diálogo de propiedades de sistema de biblioteca.

5. Haga clic en **OK** para completar la configuración.

Ha terminado la configuración del sistema de librerías, y está listo para generar y ejecutar el ejemplo, como se describe en la siguiente sección.

Generar y ejecutar el proyecto de software de Nios II

En esta sección, se ejecuta el diseño de ejemplo en un tablero de desarrollo. Utilizando el IDE de Nios II, se construye la aplicación, se configura la placa de desarrollo con un archivo de configuración válido (.sof) y se descarga el ejecutable y el Linkable Format File (.elf).

1. En la ventana que se visualizan los proyectos de Nios II C/C++, se selecciona el proyecto **ucosii_tutorial_0**.

2. En la barra de menús, seleccionamos **Tools**, se hace clic en **Quartus II Programmer**... para abrir el programador de Quartus II.

3. En el menú de archivos del programador de Quartus II, haga clic en **Open**.

4. Seleccione el archivo **NiosII_stratixII_2s60_RoHS_standard.sof,** como se muestra en la Figura C2828.

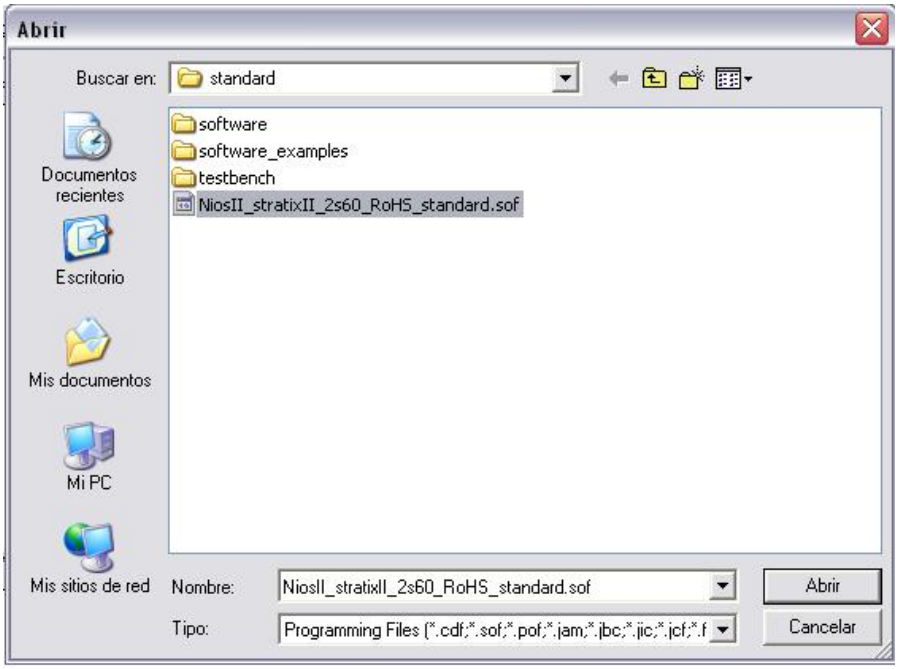


Figura C28 Cuadro de dialogo Abrir

5. Haga clic en **Abrir**. Volverá al **Quartus II Programmer**.

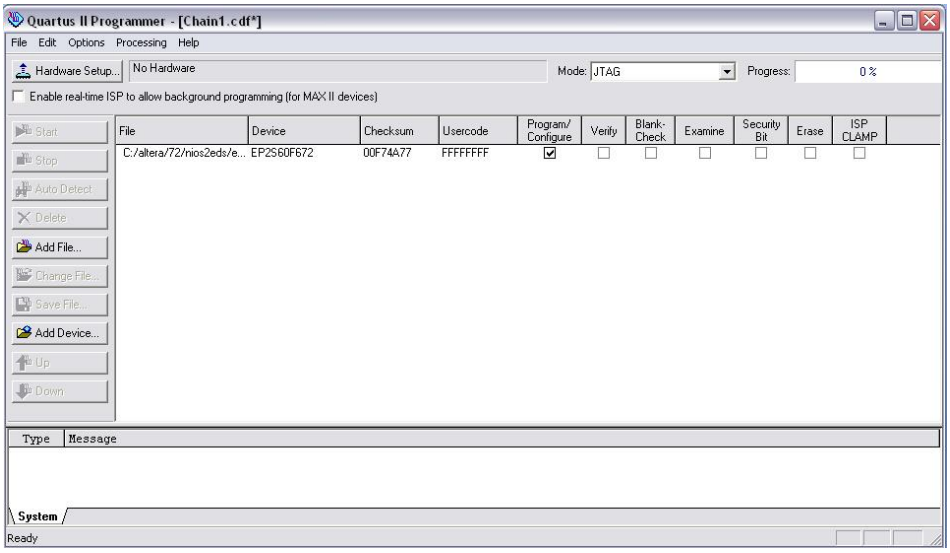


Figura C29 Herramienta de programación de Quartus II

6. Active la casilla de verificación de **Program/Configure** a la derecha del nombre de archivo **.sof**, como se muestra en la Figura C2929.

7. Haga clic en **Start** para configurar la FPGA en la placa de desarrollo con el archivo SOF. Si se supera, el programador de Quartus II muestra mensajes similares a este:

   Info: Configuration succeeded -- 1 device(s) configured
   Info: Successfully performed operation(s)
   Info: Ended Programmer operation at Wed Jan 03 17:50:03 2007

8. En el menú **File**, haga clic en **Exit** para cerrar el programador de Quartus II. Volver al IDE de Nios II.

   Nota: Si el programador de Quartus II le pregunta si desea guardar los cambios en el archivo **chain1.cdf**, haga clic en no.

9. En el menú **Run**, haga clic en **Run As**, y se hace clic en **Nios II Hardware** para crear el programa, descárguelo en la placa para ejecutarlo.

Una vez haya concluido la descarga, la ventana de la consola del **IDE** de Nios II, se actualiza periódicamente debido a **print_status_task ()** como se muestra en la Figura C3030. Los números mostrados varían en función del tipo de placa de desarrollo.

```
*********************************************************************
Hello From MicroC/OS-II Running on NIOS II.  Here is the status:

The number of messages sent by the send_task:         123

The number of messages received by the receive_task1: 73

The number of messages received by the receive_task2: 24

The shared resource is owned by: getsem_task2

The Number of times getsem_task1 acquired the semaphore 240

The Number of times getsem_task2 acquired the semaphore 185

*********************************************************************
```

Figura C30 Salida de la función ***print_status_task ()***

C.2. Creación de un proyecto con el SOTR: FreeRTOS

Para la creación de un proyecto utilizando FreeRTOS, se debe seguir los pasos los pasos descritos a continuación.

1. Haga clic derecho sobre la librería del sistema, **std_system_lib**, en la vista de **Nios II C/C ++ Projects.**

2. En el menú **Archivo**, seleccione **Nuevo** y haga clic en la aplicación de **Nios II c/c++**. Se abre la primera página del Asistente para nuevo proyecto.

3. En la parte de abajo, seleccionar la plantilla de proyecto, elija el **Blank Project**. Escriba el nombre del proyecto, la ruta de proyecto se completan automáticamente. Mantenga estos valores predeterminados.

4. Haga clic en **Browse** debajo Seleccione el hardware destino.

5. Seleccione la siguiente ruta C:altera\72\nios2eds\examples\verilog\niosII_stratixII_2s60_RoHS\standard\ **NiosII_stratixII_ 2s60_RoHS_ standard_sopc.ptf,** luego abrir.

6. Luego click en finish, ver figura Figura C31

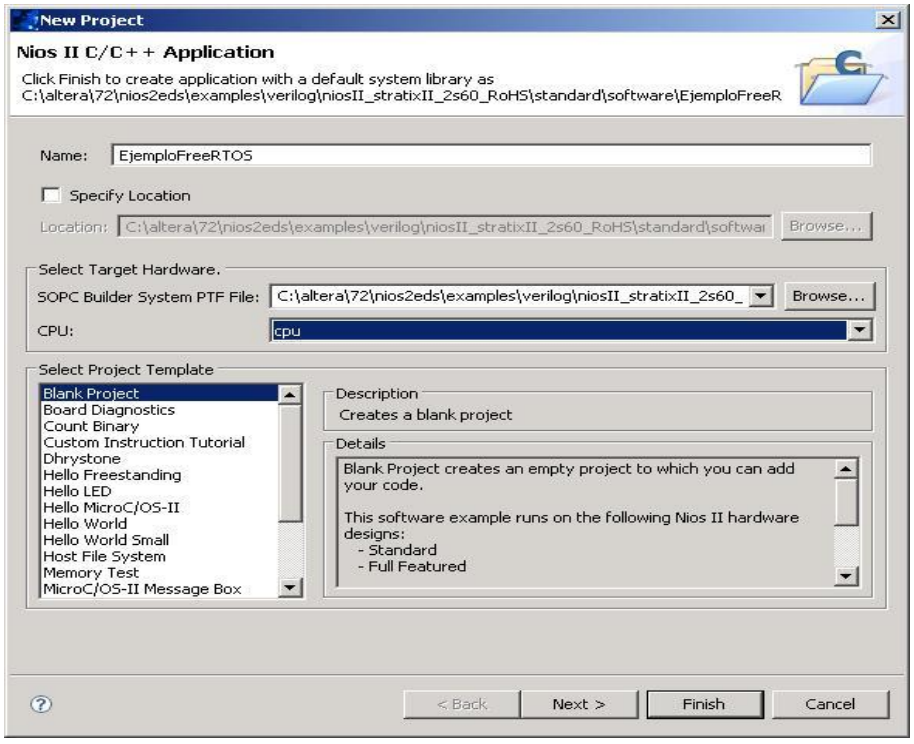


Figura C31 Creacion nuevo proyecto

7. Dar click derecho en la carpeta nombre_del_proyecto, luego click en Import…, en la ventana import seleccionar General/File System, luego next, Figura C32

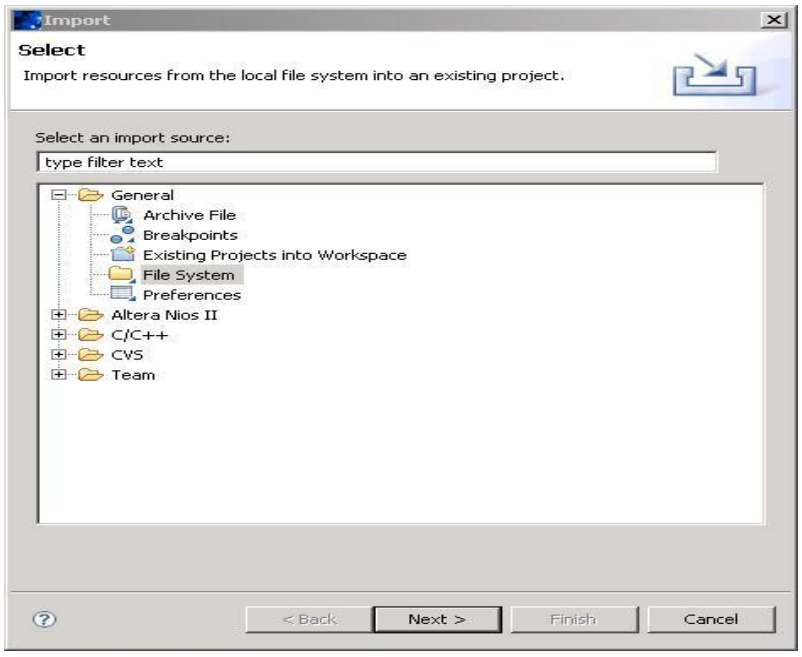


Figura C32 Importar Archivos

8. Importamos el sistema operativo FreeRTOS como muestra la Figura C33, dar aceptar, seleccionamos los archivos a importar como muestra la Figura C34, luego click en finish.

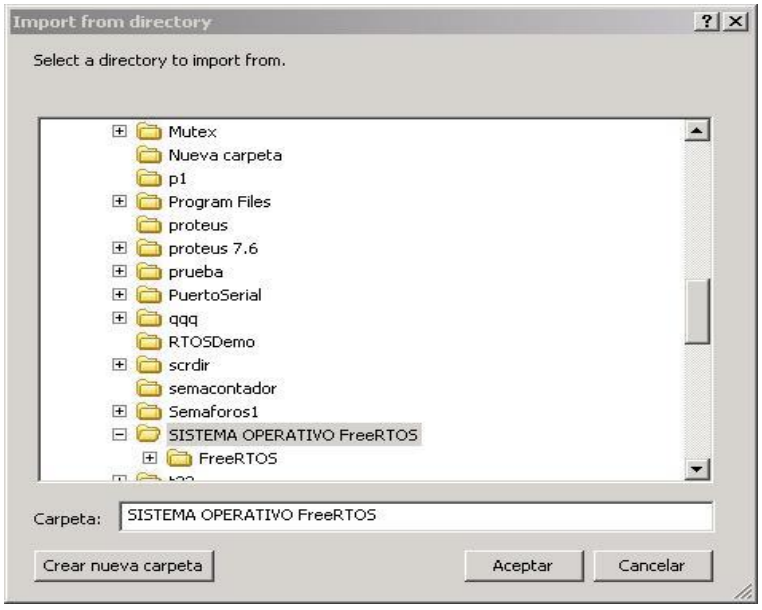


Figura C33 Importar FreeRTOS

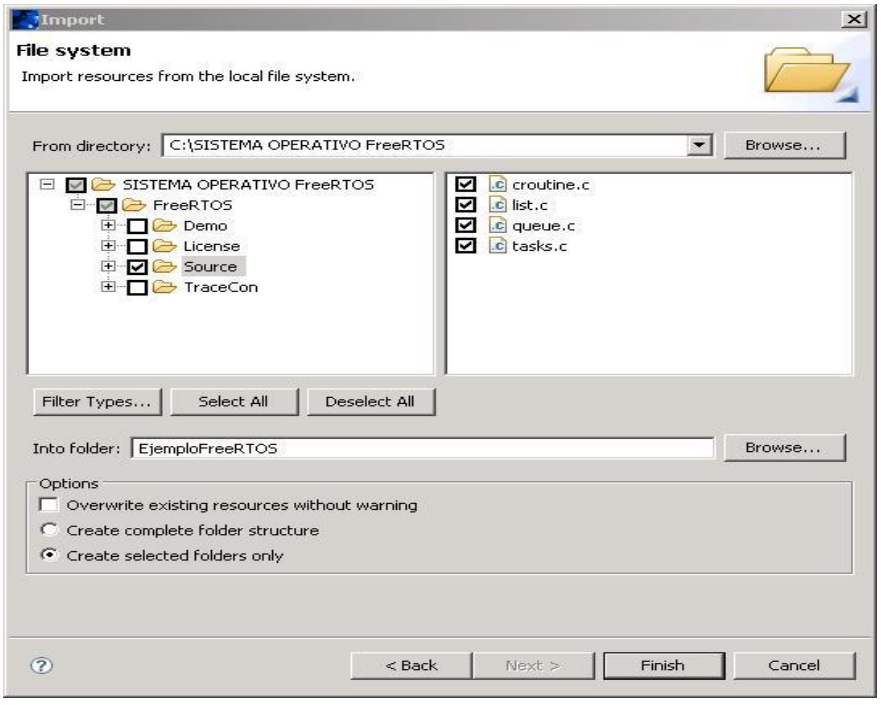


Figura C34 Archivos a importar

9. Luego dar click derecho en la carpeta nombre_del_proyecto, *properties, C/C++ Build, tool settings, Nios II compiler, general, Include Paths, Add, workspace,* he incluir el path de los siguientes directorios:

- /Nombre_del_proyecto
- /Nombre_del_proyecto/FreeRTOS/Source/Include
- /Nombre_del_proyecto/FreeRTOS/Source//portable/GCC/NiosII
- /Nombre_del_proyecto/FreeRTOS/Source//portable/MemMang

Luego dar ok, como se muestra en la Figura C35.

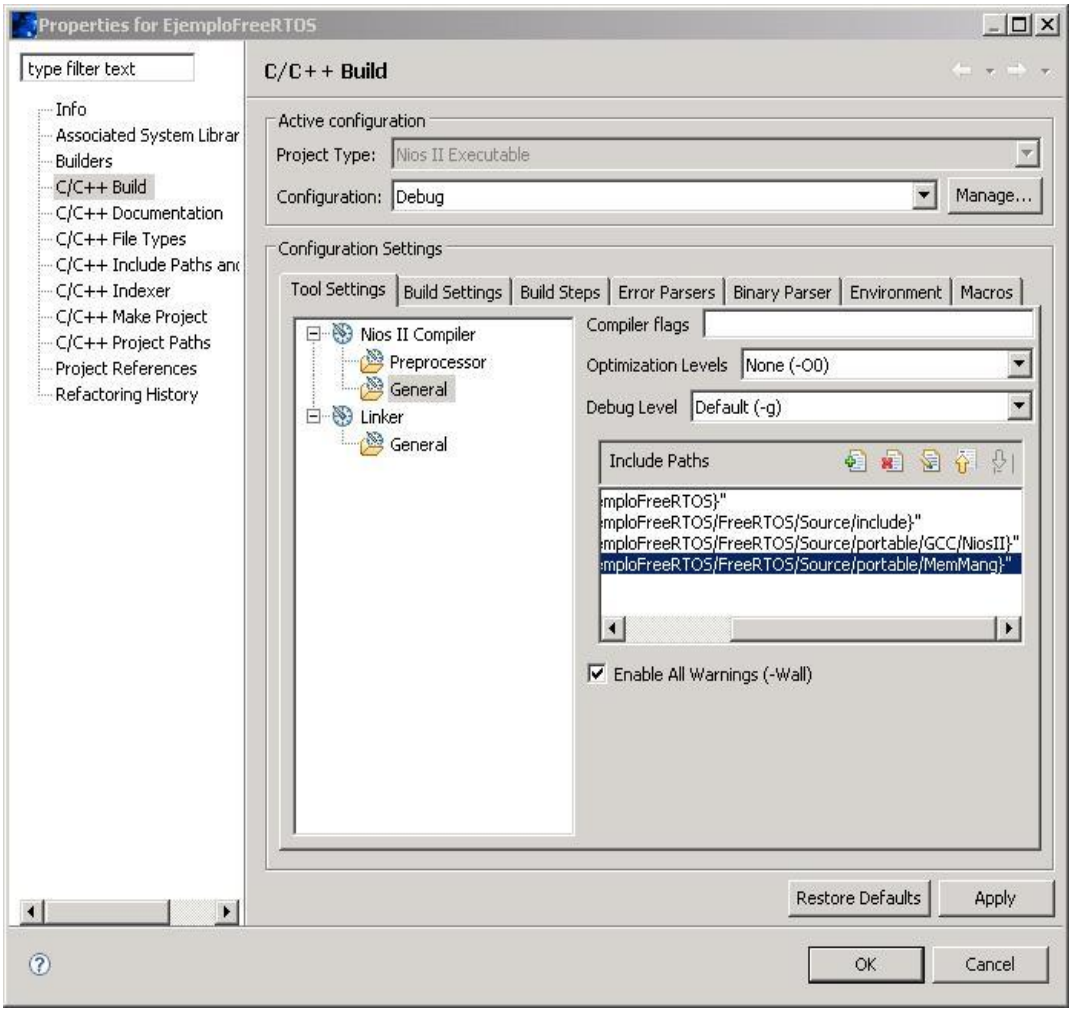


Figura C35 Incluir directorios al entorno de trabajo

10. dar click derecho en la carpeta nombre_del_proyecto, *New*, *Source File,* ingresar el nombre del archivo fuente con extensión .c en el campo *Source File*, luego *finish*.

11. Incluir los archivos de cabecera .h correspondientes a FreeRTOS y codificar aplicación, como se muestra en la Figura C36.

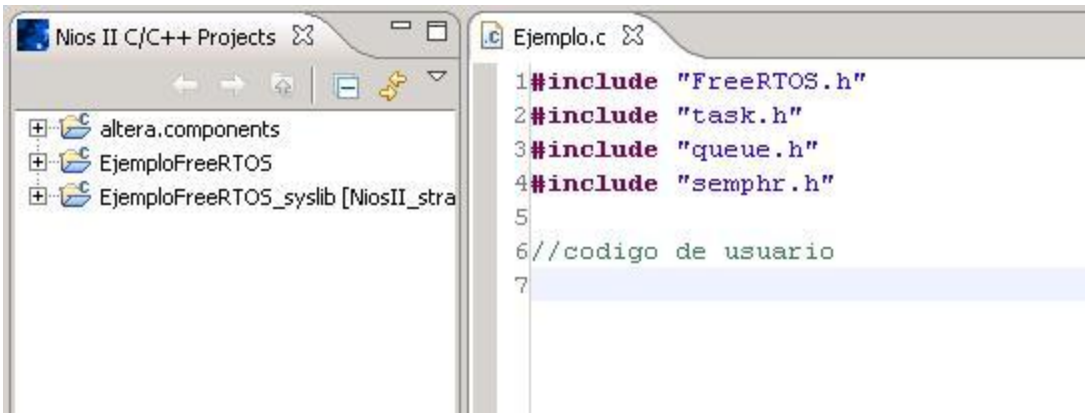


Figura C36 Codificación de la Aplicación

12. Para la realización de la compilación y la ejecución de la aplicación referirse al Anexo D, sección D.2.

ANEXO D

MANUAL DE CONFIGURACIÓN DE LA TARJETA DE DESARROLLO Y USO DEL NIOS II IDE

D.1. Configuración de la tarjeta de desarrollo desde la herramienta de programación del Quartus II

Procedimiento para programar el archivo de configuración de hardware en la tarjeta Statix II con procesador adaptable Nios II.

1. Después de instalar el paquete de Software de ALTERA, se ejecuta el programa ***Quartus II 7.2 (32-Bit),*** Figura D37

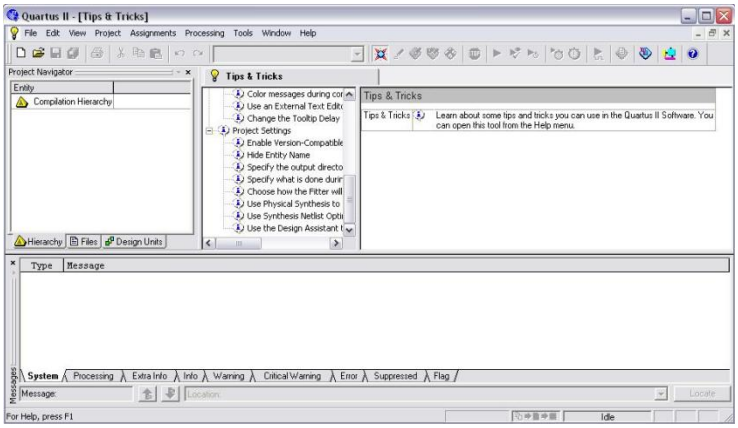


Figura D37 Entorno de desarrollo *Quartus II 7.2*

2. En la barra de herramientas, en el menú ***File*****,** Figura D38, hacemos clik sobre el icono llamado ***Open Project***:

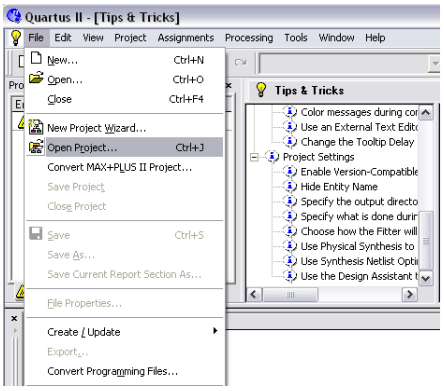


Figura D38 Proceso de apertura de proyecto *Quartus II 7.2*

3. Se despliega la ventana que es utiliza como buscador del archivo ***.qpf*** el cual contiene la configuración del hardware de la tarjeta con la que vamos a trabajar Figura D39.

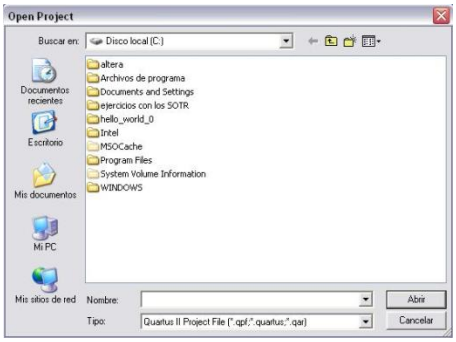


Figura D39 Ventana de apertura de proyecto

4. Buscamos el archivo en la carpeta que corresponde al nombre del modelo del kit de desarrollo correspondiente, para nuestro caso elegimos el correspondiente al **(*niosII_stratixII_2s60_RoHS*),** en la dirección: ***C:\altera\72\nios2eds\examples\verilog*** Figura D40**:**

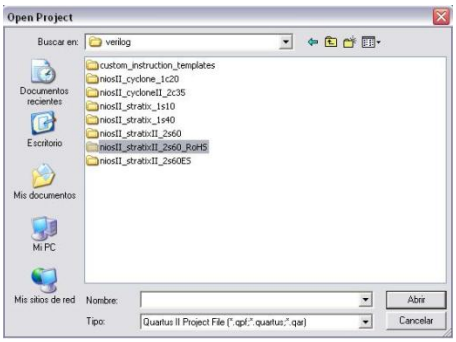


Figura D40 Ventana de Nombre del kit de desarrollo

5. En el interior de esta carpeta encontramos varias posibilidades que contienen diferentes tipos de configuraciones con características disimiles, su elección depende de la aplicación que se vaya a desarrollar, de estas elegimos ***standard*** Figura D41, la cual contiene las características generales con las que vamos a trabajar:

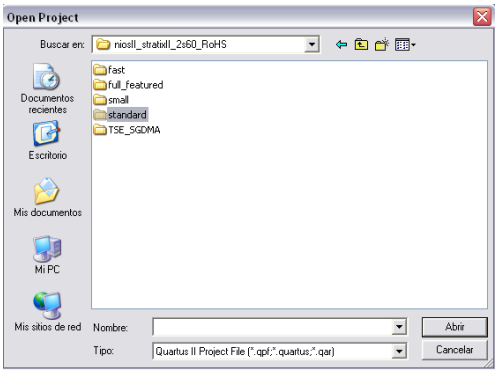


Figura D41 Ventana de tipos de configuración por proyectos

6. De esta carpeta elegimos el archivo de configuración **NiosII_stratixII_2s60ES_standard.qpf**, y pulsamos **Abrir,** Figura D42**.**

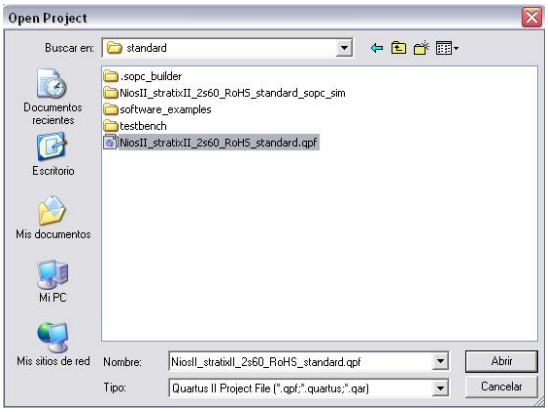


Figura D42 Archivo de configuración de tarjeta de desarrollo .qpf

7. A la izquierda del entorno de Quartus II, aparece el nombre del archivo de configuración que cargamos en la sección anterior. En la parte derecha encontramos el archivo *readme.txt* Figura D43, con las características principales de la tarjeta.

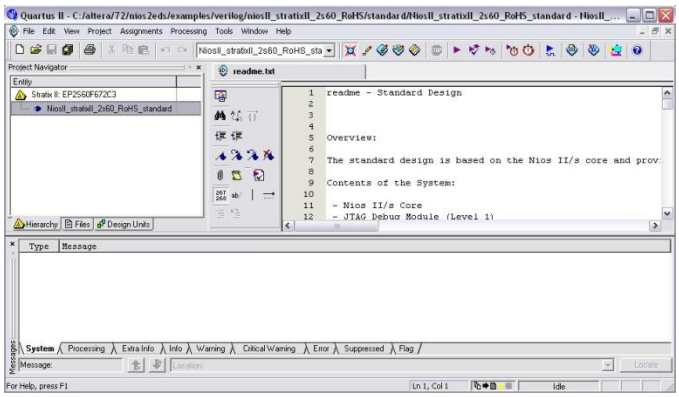


Figura D43 Entrono con archivo cargado

A continuación vamos a efectuar el proceso de programación del archivo de configuración sobre la tarjeta Stratix II con procesador adaptable Nios II:

8. En la barra superior del entorno de Quartus II, en ***Tools***, seleccionamos ***Programmer,*** Figura D44, así:

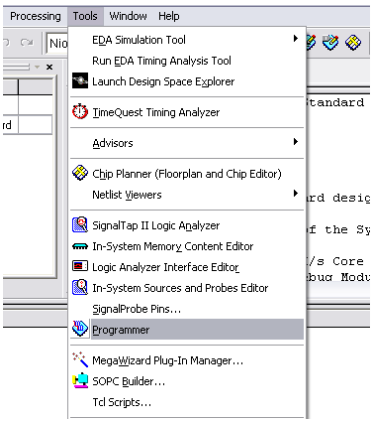


Figura D44 Apertura de herramienta de programación

En la parte derecha del entorno de Quartus II, visualizamos la herramienta que nos permite programar la tarjeta. Donde encontramos entre otras características el medio de transmisión de la información, el progreso de descarga del archivo de configuración, el nombre del archivo de configuración Figura D45, entre otras.

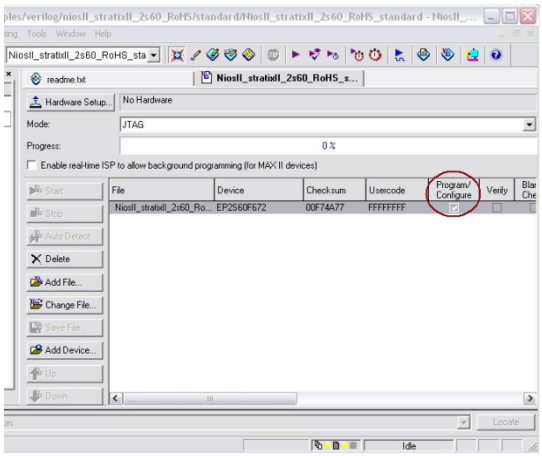


Figura D45 Configuración de herramienta de programación

Debe asegurarse que la cacilla de ***Program/Configure*** este activada para que la programación de la tarjeta se lleve a cabo de manera correcta.

En ***Hardware Setup…*** encontramos el medio por el cual se realiza la programación de la tarjeta, en nuestro caso utilizamos el ***JTAG USB Blaster*** Figura D46**,** que es el cable que permite la programación, como se indica a continuación, presionamos ***Close*****:**

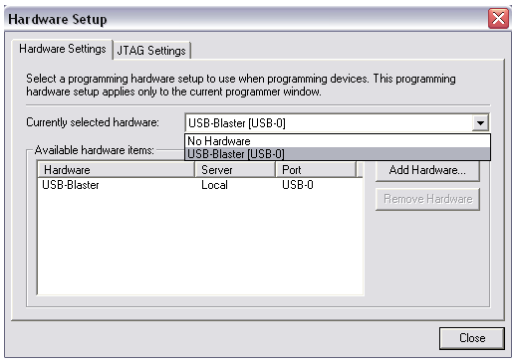


Figura D46 Selección cable de descarga *USB Blaster*

9. A continuación procedemos a presionar ***Start*** para efectuar la programación de la tarjeta Figura D47.

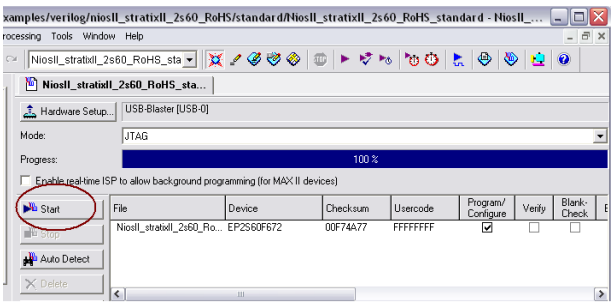


Figura D47 Proceso de descarga

Cuando ***process:*** llegue a 100%, la tarjeta está programada con su configuración de servicios y lista para utilizar.

D.2. Manual de uso del Nios II 7.2 IDE

Procedimiento que se sigue para crear un proyecto, compilarlo y programarlo sobre la tarjeta de desarrollo.

1. Después de instalar el paquete de Software de ALTERA, se ejecuta el programa Nios II 7.2 IDE Figura D48.

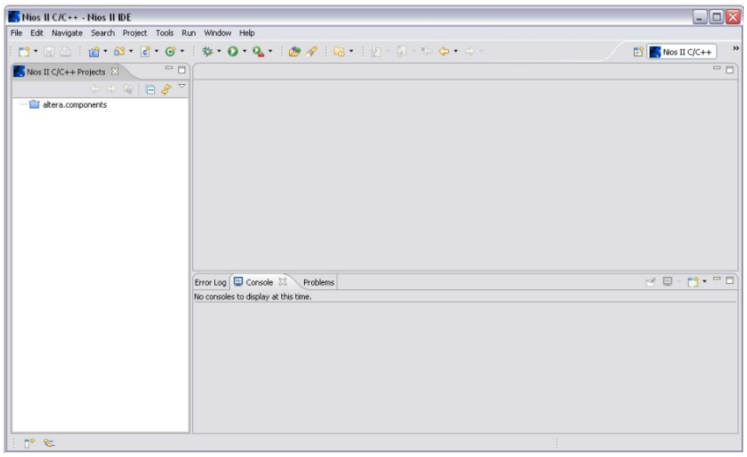


Figura D48 Entorno de desarrollo Nios II 7.2 IDE

2. En la barra de herramientas vamos a ***File*** luego, ***New*** y *se elije la opción* ***Nios II C/C++ Aplication*** Figura D49***,*** esto nos indica que hemos elegido construir una aplicación basada en el lenguaje C/C++:

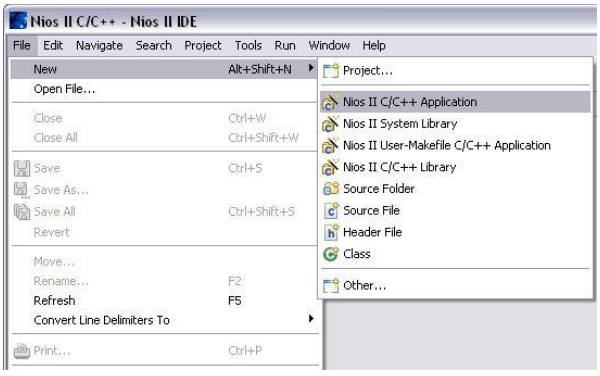


Figura D49 Selección del tipo de proyecto

3. Hemos accedido al cuadro de dialogo en el que se crea el proyecto de la siguiente manera: en la parte superior, en ***Name:*** introducimos el nombre del proyecto, sin espacios entre palabras, (si se ha elegido un ejemplo, su nombre aparecerá por defecto); luego, al seleccionarla ***Spesifyn Location*** se activará ***Location***, aquí se ingresa la dirección de creación del proyecto, aparece una dirección por defecto, sin embargo por comodidad, con el ***Browse…*** seleccionamos **C:**. En la a continuación se selecciona la tarjeta destino, aquí seleccionamos la nuestra **C:\altera\72\nios2eds\examples \verilog\niosII_stratixII_2s60_RoHS\standard\NiosII_stratixII_2s60_RoHS_**

standard_sopc.ptf-, (aparece como opción por defecto). Por último, en ***Select Project Template***, elegimos ***Blank Project***; en la parte derecha aparece la descripción y detalles del proyecto. Presionamos ***Finish*** para terminar Figura D50.

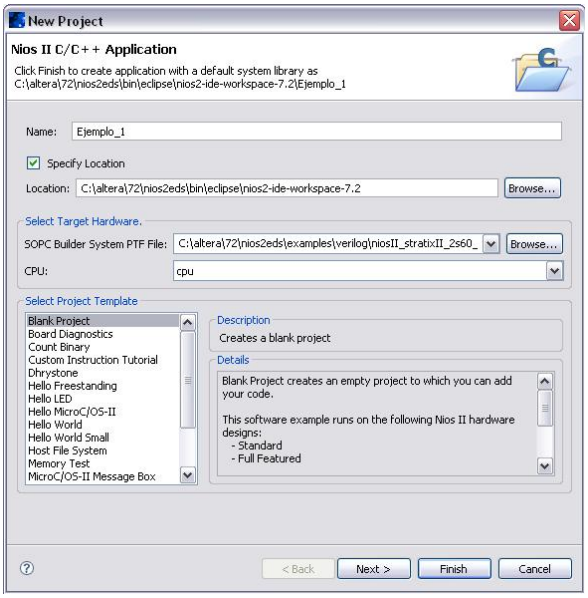


Figura D50 Ventana de creación del proyecto

4. Una vez creado el proyecto, a la derecha del Nios II SDK, aparecerán dos carpetas, son nuestro proyecto **Ejemplo_1** y la carpeta de librerías de nuestra tarjeta de desarrollo **Ejemplo_1_syslib**, presionando clic derecho sobre la carpeta del proyecto en New seleccionamos **C Source Folder** hemos elegido crear el archivo **.c**, donde haremos la codificación Figura D51.

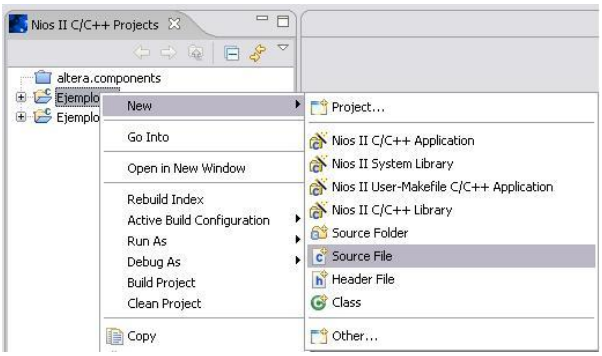


Figura D51 Creación de un archivo .c

En cuadro de dialogo, se elije el folder de creación (nuestro proyecto), y el nombre del archivo de texto que crearemos, "no olvidar colocar su extensión **.c**", a continuación, a codificar Figura D52.

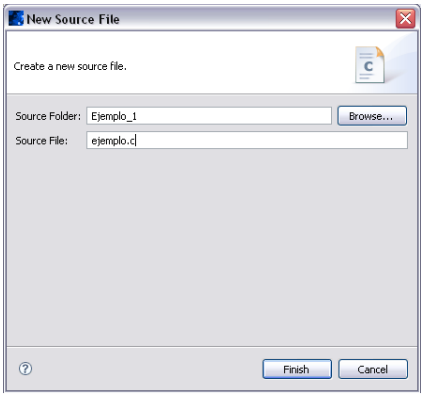


Figura D52 Configuración del archivo .c

5. A continuación, en el entorno de desarrollo de Nios II SDK, en la carpeta de nuestro proyecto aparece el archivo .C donde procedemos a codificar, Figura D53.

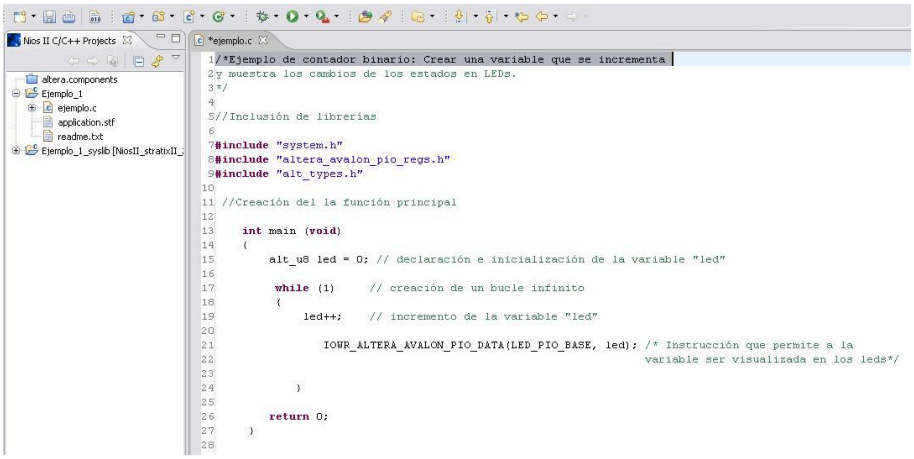


Figura D53 Ventana de creación de código

6. Para realizar la compilación de nuestro proyecto, nos ubicamos con el cursor sobre la carpeta de éste, presionamos clic derecho y seleccionamos ***Build Proyect,*** Figura D54 .

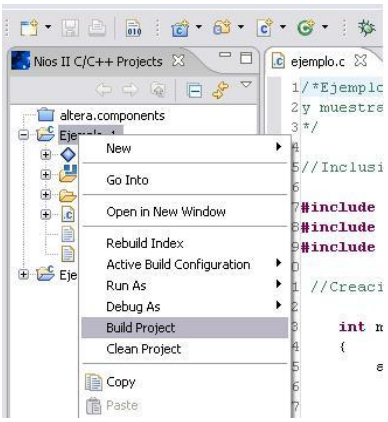


Figura D54 Compilación del proyecto

Para llevar a cabo el proceso de programación de la tarjeta de desarrollo, directamente desde el entorno de Nios II SDK, se debe asegurar que la el kit este configurado, y se procede a continuación de la siguiente manera:

1. En la barra de herramientas, ingresamos a ***Run***, aquí elegimos ***Run…***. Figura D55

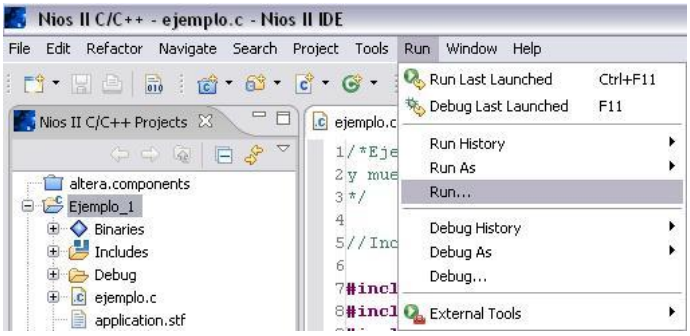


Figura D55 Correr el proyecto

2. En el cuadro de dialogo de programación que a continuación se despliega nos ubicamos en la parte derecha, en ***Nios II Hardware*** aparecerá el nombre de nuestro proyecto, presionando clic derecho sobre éste seleccionamos en el menú New Figura D56:

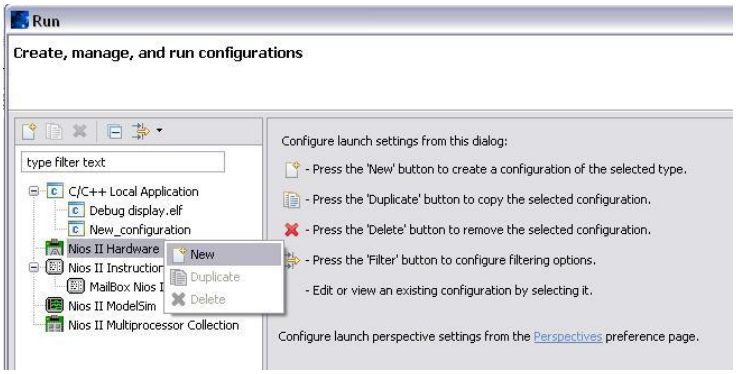


Figura D56 Programación sobre la tarjeta

3. Aparecerá el nombre de nuestro proyecto, al igual que en ***Project***, presionamos ***Run***, y listo, se ejecuta el proceso de programación sobre la tarjeta de desarrollo Stratix II con procesador adaptable Nios II, Figura D57.

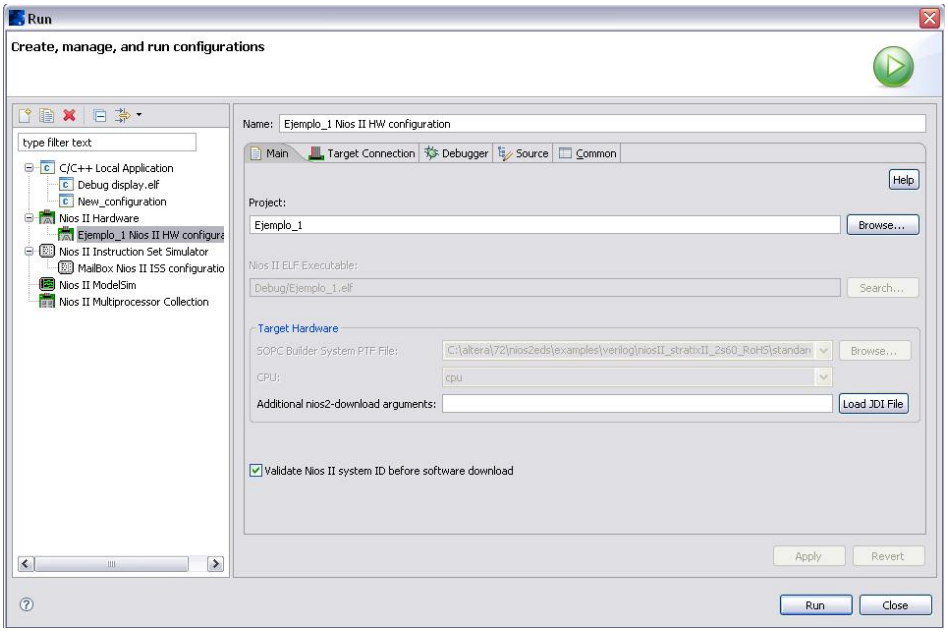


Figura D57 Descarga del proyecto sobre la tarjeta de desarrollo

ANEXO E

ENCUESTA PARA EVALUACION DE LA INTERFAZ DE MONITOREO Y DEPURACION PARA NIOS II

OBJETIVO: El objetivo de esta encuesta es evaluar la interacción de los estudiantes de laboratorio con la interfaz de monitoreo y depuración para Nios II (IMDN), con el fin de conocer su nivel de funcionalidad y desempeño.

Favor marcar con una x en la casilla correspondiente a el nivel percepción que usted tiene acerca del funcionamiento de la Interfaz de Monitoreo y Depuración para Nios II.

Nomenclatura de la calificación:

E: Excelente: El funcionamiento de la aplicación cumple con el objetivo de manera satisfactoria.

B: Bueno: El funcionamiento de la aplicación cumple con el objetivo pero podría ser mejor.

R: Regular: El funcionamiento de la aplicación cumple con el objetivo pero con fallas.

M: Malo: El funcionamiento de la aplicación no cumple con el objetivo.

Característica	Descripción	Calificación			
		E	B	R	M
Funcionalidad	Se evalúa los atributos relacionados con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son el grupo de herramientas que satisfacen las necesidades del usuario.				
Confiabilidad	Se evalúa los atributos relacionados con la capacidad de la herramienta para mantener su nivel de rendimiento bajo condiciones indicadas para un período de tiempo				

	establecido.				
Usabilidad	Se evalúa los atributos relacionados con el esfuerzo necesario para su uso.				
Eficiencia	Se evalúa los atributos que influyen en la relación entre el nivel de rendimiento del software y la cantidad de recursos utilizados, bajo condiciones establecidas.				
Mantenibilidad	Un conjunto de atributos relacionados con el trabajo necesario para hacer las modificaciones especificadas.				
Portabilidad	Un conjunto de atributos que inciden en la capacidad del software para ser transferido de un ambiente a otro.				

Observaciones:_____
_____
_____
_____
_____
_____

_____ _____

Nombre del estudiante **Código**