1 Introducción

Esta práctica tiene como objetivo estudiar brevemente la interacción de un programa escrito en Lustre con uno externo escrito en C.

2 Interacción entre Lustre y uno externo como C

Para hacer que un programa escrito en Lustre interactúe con uno externo escrito en `C`, necesitas establecer una interfaz entre ambos. `Lustre` es un lenguaje de programación sincrónico utilizado principalmente en sistemas de tiempo real y sistemas embebidos, mientras que `C` es un lenguaje de propósito general. La interacción entre ambos se puede lograr mediante la generación de código `C` a partir del código `Lustre` y luego integrarlo con el programa en `C`.

2.1 Casos Prácticos

Vamos a estudiar dos casos. El primero, el código Lustre es una simple función y en el segundo es un nodo (con memoria, que puede llamar a otros nodos o funciones etc.)

2.1.1 Caso 1: Descripción del problema

Supongamos que tenemos un programa en `Lustre` que realiza un cálculo simple, como sumar dos números enteros en cada ciclo de reloj. Queremos que este programa interactúe con un programa en `C` que proporciona los valores de entrada y recibe el resultado de la suma.

1. **Paso 1: Escribir el programa en Lustre**

   El programa en Lustre consiste en una función que realizará la suma de dos entradas (`a` y `b`) y producirá una salida (`c`). Esta función se almacena en un fichero `suma.lus`.

   ```
   function suma(a, b: int) returns (c: int);
   let
       c = a + b;
   tel
   ```

   A continuación, vamos a tener que compilar el código `Lustre` para convertirlo a `C`.

2. **Paso 2: Compilar el programa `Lustre` a `C`**

   Para compilar el código `Lustre`, tendremos que ejecutar el siguiente comando en el terminal:

   ```
   $lv6 suma.lus -node suma -cc
   ```

   Esto genera una serie de ficheros entre los que se encuentran `suma_suma.c` y un archivo de cabecera `suma_suma.h`. Concretamente, `suma_suma.c` contiene la implementación en `C` del nodo `Lustre` y `suma_suma.h` contiene las declaraciones de las funciones generadas.

3. **Paso 3: Escribir el programa en `C`**

   A continuación, tenemos que implementar un código en `C` que llamará a la función generada por `Lustre` (`suma`) y manejará la entrada y salida. En este caso, será el fichero `main.c` cuya implementación se detalla más abajo:

```c
#include <stdio.h>
#include "suma_suma.h"  // Incluye el archivo de cabecera generado por Lustre
int main() {
    int a, b, c;

    // Ejemplo de entrada
    printf("Introduce el valor de a: ");
    scanf("%d", &a);
    printf("Introduce el valor de b: ");
    scanf("%d", &b);

    // Llamar a la función Lustre
    // Pasa un puntero a 'c' para almacenar el resultado
    suma_suma_step(a, b, &c);

    // Mostrar el resultado
    printf("La suma es: %d\n", c);
    return 0;
}
```

En este fichero se observa como se comienza incluyendo el archivo de cabecera `#include "suma_suma.h"`, el cual contiene la declaración de la función generada por Lustre. Concretamente el contenido de dicho fichero es:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lustre_consts.h"

// User typedef
#ifndef _suma_suma_TYPES
#define _suma_suma_TYPES
// Memoryfull soc ctx typedef
//suma_suma does not have memory
void suma_suma_step(_integer ,_integer ,_integer *);

////////////////////////////////////////////
#endif // end of  _suma_suma_TYPES
```

Ahí se observa que la función que realiza la suma es `suma_suma_step(_integer ,_integer ,_integer *)` (Lustre ha añadido el sufijo `_step`) y requiere de 3 variables: las dos primeras son las de entrada, y la última es un puntero a un entero que almacena el resultado. Al tener la función un tipo de retorno void, eso significa que no devuelve un valor directamente. En su lugar, almacena el resultado en la variable apuntada por el tercer argumento.

Volviendo al código principal, nada más empezar la ejecución del mismo se mostrará por pantalla una serie de mensajes con el objetivo de que el usuario introduzca los valores de `a` y `b`.

Más adelante, se realiza la llamada a la función `suma_suma_step(a, b, &c)` que invoca la función generada en Lustre, pasando los valores capturados previamente en `a` y `b` como entradas y almacenando el resultado en la dirección de `c` (la salida).

Finalmente, el resultado de la función se imprime.

4. **Paso 4: Compilar y enlazar los programas**

   Para seguir adelante con la ejecución del código generado en C tenemos que compilar el archivo suma_suma.c generado a partir del código Lustre. Esto es:

   ```
   $gcc -c suma_suma.c -o suma_suma.o
   ```

   Una vez eso hecho, pasamos a compilar el programa principal main.c que acabamos de crear:

   ```
   $gcc -c main.c -o main.o
   ```

   Finalmente, los dos archivos objeto (suma_suma.o y main.o) se tienen que enlazar para crear el ejecutable final:

   ```
   $gcc suma_suma.o main.o -o programa_suma
   ```

5. **Paso 5: Ejecutar el programa resultante**

   Ya solo queda ejecutar el programa previo para comprobar el resultado:

   ```
   $./programa_suma
   ```

   Un ejemplo de salida podría ser el siguiente:

   ```
   Introduce el valor de a: 5
   Introduce el valor de b: 3
   La suma es: 8
   ```

2.1.2 Caso 2: Descripción del problema

Supongamos ahora que usamos el programa de Lustre implementado en la primera práctica, abro.lus (disponible en Moovi), cuya especificación era que se emitía una salida O tan pronto como dos entradas A y B ocurriesen y se reinicializaba el sistema cada vez que ocurría R. Este se encuentra disponible en Moovi. Ahora queremos que este programa interactúe con un programa en C que proporcione los valores de entrada y reciba el resultado O.

Los pasos son similares al mostrado en el caso 1. Lo primero que vamos a hacer es compilar el código Lustre, ejecutando el siguiente comando:

```
$lv6 abro.lus -n ABRO -cc
```

Al igual que antes, esto genera una serie de ficheros entre los que se encuentran abro_ABRO.c y el archivo de cabecera abro_ABRO.h. Concretamente, abro_ABRO.c contiene la implementación en C del nodo Lustre y abro_ABRO.h contiene las declaraciones de las funciones generadas.

A continuación, se muestra el código en C correspondiente al fichero principal `main.c`. En este código podemos enviar señales de activación (`A`, `B`, `R`) en momentos específico. Vamos a verlo con más detalle:

```
#include <stdio.h>
#include <stdbool.h>
#include "abro_ABRO.h" // Incluir el archivo de cabecera generado

int main() {
    // Variables de entrada y salida
    _boolean A = false; // Señal A
    _boolean B = false; // Señal B
    _boolean R = false; // Reset
    _boolean O = false; // Salida O

    abro_ABRO_ctx_type abro_ctx; // Contexto del nodo ABRO
    abro_ABRO_ctx_reset(&abro_ctx); // Inicializar el contexto

    // Simulación de interacción con el usuario
    while (true) {
        // Solicitar al usuario qué señal desea enviar
        char opcion;
        printf("¿Qué señal desea enviar? (a = A, b = B, r = Reset, s = salir): ");
        scanf(" %c", &opcion);

        if (opcion == 'a') { // Activar la señal A
            A = true;
            B = false; // Reiniciar la señal B
            R = false; // Reiniciar el reset
        } else if (opcion == 'b') { // Activar la señal B
            B = true;
            A = false; // Reiniciar la señal A
            R = false; // Reiniciar el reset
        } else if (opcion == 'r') { // Activar el reset
            R = true;
            A = false; // Reiniciar la señal A
            B = false; // Reiniciar la señal B
        } else if (opcion == 's') { // Salir del programa
            break;
        } else {
            printf("Opción no válida.\n");
            continue;
        }

        // Llamar a la función principal de ABRO
        abro_ABRO_step(A, B, R, &O, &abro_ctx);
        // Mostrar el estado de la salida
        if (O) {
            printf("¡Salida O activada!\n");
        } else {
            printf("Salida O desactivada.\n");
        }
    }
    return 0;
}
```

- Se ha incluido `#include abro_ABRO.h`. Este archivo contiene las declaraciones de las funciones y tipos necesarios para interactuar con el nodo Lustre `ABRO`.
- Se han definido las variables `_boolean A`, `_boolean B`, `_boolean R`, todas ellas de tipo booleano. Todas estas variables representan las señales de entrada al nodo Lustre. Del mismo modo, `_boolean O` es la variable booleana que almacena el resultado.
- Se define el contexto de `ABRO`, es decir, se declara una variable de tipo `abro_ABRO_ctx_type`, que es una estructura que almacena el estado interno del nodo Lustre (por ejemplo, las memorias `seenA` y `seenB`).
- Luego, se define `abro_ABRO_ctx_reset(&abro_ctx);` cuya misión es inicializar el contexto del nodo Lustre. Este paso es necesario para asegurar que el estado interno comience en un valor conocido (por ejemplo, `seenA = false` y `seenB = false`).
- A partir de ahí se crea un bucle infinito que permite interactuar con el usuario repetidamente hasta que decida salir. Concretamente, si el usuario inserta:
  - `'a'`, se activa la señal `A` y se desactivan las señales `B` y `R`.
  - `'b'`, se activa la señal `B` y se desactivan las señales `A` y `R`.
  - `'r'`, se activa el reset (`R`) y se desactivan las señales `A` y `B`.
  - `'s'`, se sale del bucle y termina el programa.
  - una opción no válida, se muestra un mensaje de error y se vuelve a solicitar una opción.
- A continuación se realiza la llamada a la función principal de `ABRO` mediante `abro_ABRO_step` que es la función generada por Lustre que implementa la lógica del nodo. Recibe los parámetros `A`, `B`, `R`, el puntero a la variable de salida `O`, es decir, `&O`; y el puntero al contexto del nodo Lustre `&abro_ctx`.
- Finalmente, se muestra el estado de las salidas.

ACTIVIDAD:
`Modificar el código para que permita insertar dos y tres señales a la vez.`

3 Entregable

Supongamos un programa Lustre que implemente el funcionamiento de un reloj. El programa almacenará la hora actual en una estructura con tres enteros: la hora, los minutos y los segundos y dispondrá de las siguientes señales.

- Tres señales de entrada `hora_signal`, `minuto_signal` y `segundo_signal` que serán transmitidas desde el programa en C al núcleo reactivo cuando el usuario así lo decida, indicando cual de las señales se quiere modificar.
- Tres entradas de tipo entero `nueva_hora`, `nuevo_minuto` y `nuevo_segundo` que serán transmitidos desde el programa en C al núcleo reactivo cuando el usuario así lo decida, indicando el nuevo valor a modificar.
- Una señal de salida `tiempo` que será de tipo entero y que mostrará por pantalla el número de segundos transcurridos desde el mediodía.

Cada vez que esté presente una de las señales `hora_signal`, `minuto_signal` o `segundo_signal`, el reloj en Lustre actualizará sus valores e informará por pantalla del número total de segundos transcurridos.

Entrada `hora`	Entrada `minuto`	Entrada `segundo`	Salida `tiempo` (segundos desde el mediodía)
12	0	0	0
13	15	30	4530
14	45	10	9910
0	0	0	43200 (medianoche siguiente)
11	59	59	86399

Figure 1: Ejemplo de funcionamiento

Para ello será necesario implementar en el reloj el cálculo del tiempo transcurrido entre la hora actual y el mediodía (si no se inserta alguno de los tre enteros, se tomará el valor del ciclo anterior).

El programa en C proporcionará una interfaz interactiva para el usuario, permitiéndole modificar la hora, los minutos o los segundos de forma independiente. Al recibir una actualización, el sistema transmitirá la señal correspondiente al núcleo reactivo, que procesará el cambio y actualizará la salida tiempo.

El cálculo del tiempo transcurrido desde el mediodía se realizará mediante la fórmula:

$$tiempo = (hora\_actual - 12)x3600 + minuto\_actualx60 + segundo\_actual$$

El sistema deberá manejar correctamente la actualización de los valores para evitar inconsistencias, asegurando que los minutos y segundos se mantengan en el rango válido (0-59) y que la hora permanezca dentro del rango de 0 a 23.

El modelo Lustre será compilado para generar código C, que se integrará con la interfaz de usuario. Esta última permitirá a los usuarios seleccionar qué valores actualizar y visualizar en tiempo real la cantidad de segundos transcurridos desde las 12:00.