Esta práctica tiene como objetivo la familiarización con el entorno de compilación y el entorno de simulación de Lustre.

# 1 Entorno de compilación y pruebas en Lustre

Supongamos que necesitamos un programa en Lustre que detecta flancos de subida o de bajada en un flujo de tipo booleano. Este tipo de programa es muy útil en el contexto de un sistema de control de ascensores diseñado para responder a las interacciones de usuarios mediante un pulsador. Este pulsador genera un flujo booleano:

- 0: El pulsador no está presionado (falso).
- 1: El pulsador está presionado (verdadero).

En este contexto, existe la necesidad de que el sistema detecte las transiciones (flancos) para determinar cuándo un usuario ha pulsado y soltado el botón, evitando activaciones repetidas por mantenerlo presionado.

La implementación del programa descrito se encuentra en el fichero `edge.lus` que se encuentra disponible en Moovi, cuyo contenido es el siguiente:

```
node EDGE(X:bool) returns (Y:bool);
let
    Y = R_EDGE(X) or R_EDGE(not(X));
tel

node R_EDGE (X:bool) returns (Y:bool);
let
    Y = false → X and not pre(X);
tel
```

cuya evaluación ciclo a ciclo es la siguiente:

| Ciclo | X | pre(X) | R_EDGE(X) | R_EDGE(not(X)) | Y |
|---|---|---|---|---|---|
| 1 | 0 | - | false | false | false |
| 2 | 0 | 0 | false | false | false |
| 3 | 1 | 0 | true | false | true |
| 4 | 1 | 1 | false | false | false |
| 5 | 0 | 1 | false | true | true |
| 6 | 0 | 0 | false | false | false |
| 7 | 1 | 0 | true | false | true |
| 8 | 0 | 1 | false | true | true |

Figure 1: Evaluación paso a paso de EDGE

Si uno pretende saber si el código es sintácticamente correcto, solo necesita invocar:

```
$lv6 edge.lus -n EDGE
```

Destacar que la opción `-n` (o `-node`) establece el nodo de primer nivel (es obligatorio).

```
$lv6 edge.lus -node EDGE
$lv6 edge.lus -n R_EDGE
```

Si todo es correcto, el resultado será el siguiente:

```
This program is syntactically correct. If you want to do something with it, you
could try one the options: -2c, -2c-exec, -exec, -ec, -lic, or -h for more options.
```

Este mensaje indica que el programa no contiene errores de sintaxis. Sin embargo, el mensaje también señala que Lustre no hará nada adicional a menos que se utilice una de las opciones proporcionadas para realizar acciones específicas.

Por tanto, una vez comprobado que no hay errores sintácticos, es necesario generar el código C, para lo que se puede usar la opción `-2c`.

```
$lv6 edge.lus -n EDGE -2c
```

Los ficheros generados a partir de `edge.lus` son una serie de archivos todos ellos necesarios para ejecutar o integrar el modelo en un sistema de programación en C. A continuación se va a explicar cada uno de los ficheros generados:

1. **edge_EDGE.c**:

   Este archivo contiene la implementación en C del nodo `EDGE` definido en el código Lustre. La función `EDGE` se traduce directamente a código C e implementa la lógica que calcula la salida `Y` a partir de la entrada `X`, utilizando internamente el nodo `R_EDGE`.

2. **edge_EDGE.h**:

   Este archivo es un encabezado en C que define las interfaces necesarias para interactuar con el nodo `EDGE`. Contiene las declaraciones de funciones, tipos de datos y estructuras requeridas para integrar el nodo en el resto del programa. Este archivo podría incluir los prototipos [1] de las funciones que ejecutan la lógica del nodo `EDGE` y gestionan sus entradas y salidas.

3. **edge_EDGE_loop.c**:

   Este archivo en C implementa la lógica de bucle necesaria para la ejecución iterativa o continua del nodo `EDGE`. En aplicaciones de tiempo real o sistemas embebidos, los cálculos se realizan de manera cíclica. Por ello, este archivo incluye el código que actualiza los valores de entrada `X` y calcula las salidas `Y` en cada ciclo de ejecución.

4. **edge_EDGE_loop_io.c**:

   Este archivo implementa la lógica de entrada y salida asociada al ciclo de ejecución del nodo `EDGE`. Incluye las funciones que gestionan la lectura de la entrada `X` y el cálculo y retorno de la salida `Y` en cada iteración del bucle, facilitando la integración con el resto del sistema.

5. **edge_EDGE_loop_io.h**:

[1] Puede contener las declaraciones de las funciones necesarias para implementar `EDGE`, pero no su implementación completa. En resumen, el archivo contiene una descripción de "qué hace" cada función o componente relacionado con el nodo `EDGE`, pero no "cómo lo hace" (esa parte se encuentra en el archivo `.c`).

Este archivo es un encabezado en C que define las interfaces de entrada/salida para el bucle de ejecución del nodo `EDGE`. Contiene declaraciones de funciones diseñadas para permitir la interacción con otras partes del sistema, como la lectura de valores de entrada `X` y la escritura de valores de salida `Y`, integrándose con dispositivos físicos o subsistemas en sistemas de control en tiempo real.

6. **lustre_consts.h**:

   Este encabezado en C contiene las definiciones de las constantes utilizadas en el sistema generado a partir del modelo Lustre. Estas constantes pueden incluir valores relacionados con configuraciones de tiempo, límites del sistema o parámetros específicos del modelo, garantizando la consistencia en todo el programa.

7. **lustre_types.h**:

   Este archivo es un encabezado en C que define los tipos de datos utilizados en la implementación del modelo Lustre. Incluye definiciones de estructuras, enumeraciones y tipos personalizados que facilitan la organización y manipulación de los datos necesarios para las entradas, salidas y otras partes del modelo.

8. **lustre_consts.c**:

   Este archivo en C implementa las constantes definidas en el archivo `lustre_consts.h`. Aquí se inicializan y gestionan las constantes que se utilizan en el sistema generado, asegurando que estén disponibles en todas las partes del programa de manera uniforme.

9. **EDGE.sh**:

   Este archivo es un script de shell diseñado para automatizar tareas en sistemas Unix/Linux, como la compilación y ejecución del código generado. El script `EDGE.sh` se encarga de compilar el código C correspondiente al nodo `EDGE` y crear el fichero ejecutable `EDGE.exec` del modelo generado, permitiendo su simulación o integración en el entorno de control.

En resumen:

Cuando compilas un modelo de Lustre con el comando `lv6 edge.lus -n EDGE -2c`, se generan todos estos archivos en C que permiten ejecutar e integrar el modelo de control de Lustre en un sistema embebido o de tiempo real. El proceso incluye la conversión del código Lustre en código C ejecutable, además de generar encabezados para la interacción con el sistema (entradas, salidas, y constantes), la configuración del ciclo de ejecución y la automatización de la ejecución.

Cada archivo tiene un propósito específico para que el sistema resultante funcione correctamente y permita la simulación, la compilación y la ejecución del modelo de control.
Finalmente, para generar el código binario ejecutable, solo será necesario ejecutar el script `.sh` o por el contrario, generar el código C junto con su ejecutable con la opción `-2c-exec`.

```
$sh EDGE.sh
```

o

```
$lv6 edge.lus -n EDGE -2c-exec
```

# 2 Entorno de simulación de Lustre

Para ilustrar el comportamiento del entorno de simulación vamos a hacer uso en esta práctica de la implementación de la especificación **ABRO** y a continuación se desarrollará la implementación de la especificación **MOUSE**.

## 2.1 Especificación ABRO

La especificación **ABRO** se puede formular de la siguiente manera: *Emitir una salida O tan pronto como dos entradas A y B ocurran (no tienen porque ocurrir simultáneamente). Reinicializar el sistema cada vez que ocurre R.*

El programa Lustre que implementa dicha funcionalidad es el siguiente:

```
node EDGE(X:bool) returns (Y:bool);
let
    Y = false → X and not pre(X);
tel

node ABRO (A,B,R:bool) returns (O: bool);
    var seenA, seenB : bool;
let
    O = EDGE(seenA and seenB);
    seenA = false → not R and (A or pre(seenA));
    seenB = false → not R and (B or pre(seenB));
tel
```

Con un editor de texto, como `nano` o `gedit`, escribe el siguiente programa y guárdalo con el nombre `abro.lus`. A continuación, utiliza el compilador `Lustre` para procesar este archivo. Como resultado, se generará un fichero denominado `abro.c`, el cual puede emplearse como núcleo reactivo dentro de una aplicación. Dicha aplicación puede integrarse en un sistema más amplio o interactuar directamente con dispositivos de hardware.

Nuestro propósito es la simulación del núcleo reactivo, para lo cual la compilación en `Lustre` se realizará:

```
$ lv6 abro.lus -node ABRO -2c-exec
```

lo que genera un fichero binario ejecutable `ABRO.exec` cuyo comportamiento puede ser simulado mediante la traza del programa. Para obtener esta traza, simplemente se deberá ejecutar el fichero ejecutable con:

```
$ ./ABRO.exec
```

De esta forma se podria obtener la siguiente traza del programa:

```
#inputs "A":bool "B":bool "R":bool
#outputs "O":bool
#step 1
true true false
0
#step 2
true true false
1
#step 3
true false true
0
#step 4
false true false
0
#step 5
true false false
1
```

Obsérvese dos cosas. En primer lugar, tal y como ya quedó reflejado en la Figura 1, el primer paso de la simulación es un paso en blanco. En segundo lugar, cuando se trabaja con booleanos, la simulación admite los valores `true/false` y `1/0` para representarlos.

Otro modo de realizar la simulación es mediante simulación gráfica. Para ello es necesario ejecutar las siguientes órdenes:

```
$ luciole abro.lus ABRO
```

En este momento se abre una ventana con el aspecto de la Figura 2.

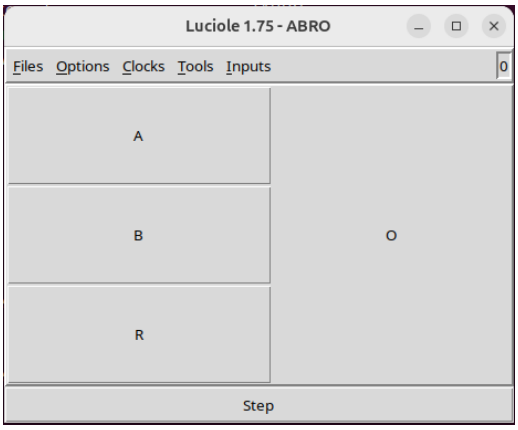


Figure 2: Herramienta Luciole

Se trata de una herramienta de análisis y diagnóstico para monitorear y depurar el tráfico de entrada y salida en el sistema Lustre, identificando los patrones de uso y proporcionar una vista en tiempo real de las operaciones que se están realizando en el sistema. Con esta herramienta se puede visualizar el comportamiento del sistema, facilitando la comprensión de problemas complejos.

Concretamente, tiene varios configuraciones posibles atendiendo a: 1) considerar una única entrada por ciclo de reloj o considerar varias entradas simultáneas por ciclo de reloj; 2) considerar que reloj lógico marcado por el usuario o un reloj en tiempo real, el cual se puede configurar su frecuencia. Vamos a ver ejemplos para cada uno de ellos, atendiendo a la Figura 3.

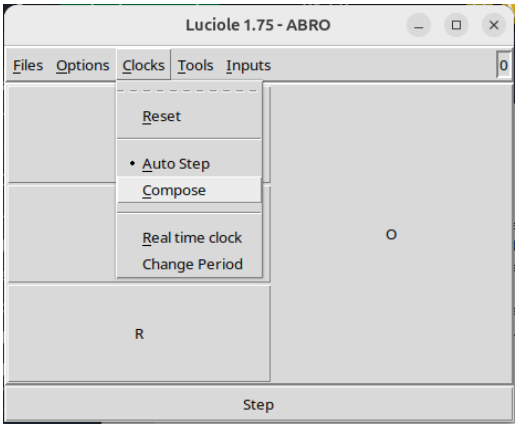


Figure 3: Opciones del menú `Clocks`

Por tanto, para considerar una única entrada por cada ciclo de reloj se tiene que mantener en el menú `Clocks` la opción `Auto step` seleccionada, mientras que en el caso de querer poder introducir varias entradas simultáneas, será necesario seleccionar la opción `Compose`.

En caso de querer establecer un reloj en tiempo real, en el menú `Clocks` será necesario seleccionar la opción `Real time clock`. En caso contrario, se tendrá que desactivar esa opción.

Finalmente, además de pulsar sobre las señales `A`, `B` y `R`, ya bien sea de forma simultánea o no, y enviarse al sistema reactivo con el objetivo de generar el evento de salida, es posible visualizarlo a lo largo del tiempo mediante la herramienta `sim2chro`. Para ello, será necesario ir al menú `Tools` y seleccionar ficha opción, tal y como se ve en la Figura 4.

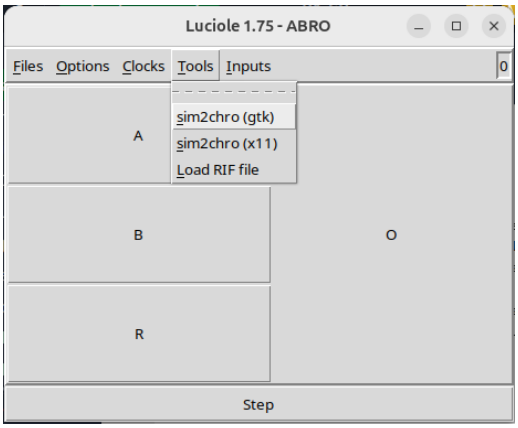


Figure 4: Opciones del menú `Tools`

El resultado será la ventana que se abre y se observa en la Figura 5.

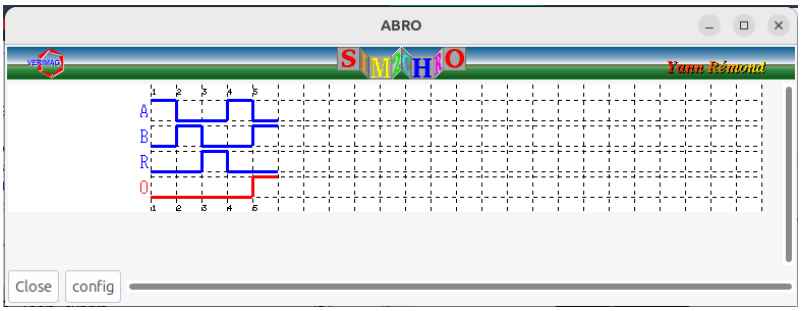


Figure 5: Ventana sim2chro

**ACTIVIDAD:** `Tarea 1: Eliminar el paso en blanco inicial del código y mostrar mediante sim2chro la traza del programa anterior, pero esta vez con el nuevo código.`

## 2.2 Especificación MOUSE

Se desea implementar un sistema en Lustre que simule el comportamiento de un ratón (**MOUSE**) capaz de detectar clicks simples (`SINGLE`) y dobles (`DOUBLE`). El sistema debe cumplir con las siguientes condiciones, tal y como se ve en la Figura 6:

Las señales de entrada son:

- `CLOCK`: Una señal booleana que marca el paso del tiempo. Cada vez que CLOCK es true, se considera que ha transcurrido un ciclo.
- `CLICK`: Una señal booleana que indica si el usuario ha hecho un click. Cuando `CLICK` es true, se registra un click.

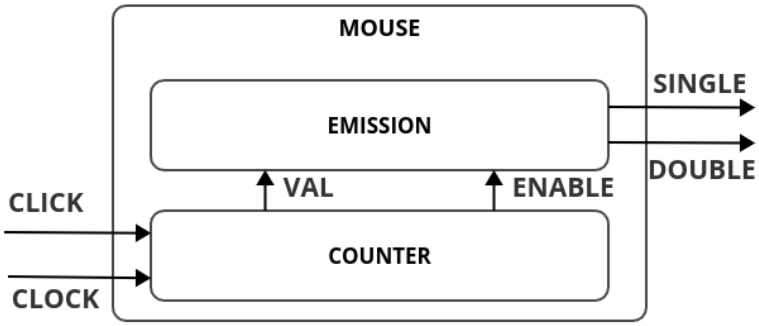


Figure 6: Especificación **MOUSE**

Las señales de salida son:

- `SINGLE`: Una señal booleana que se activa (true) si se detecta un click simple.
- `DOUBLE`: Una señal booleana que se activa (true) si se detecta un doble click.

Requisitos funcionales:

- Las salidas `SINGLE` y `DOUBLE` solo deben emitirse cada dos ciclos de `CLOCK`.
- Un click simple (`SINGLE`) se detecta si hay exactamente un click en los dos ciclos anteriores.
- Un doble click (`DOUBLE`) se detecta si hay dos o más clicks en los dos ciclos anteriores.
- Después de emitir las salidas, el sistema debe reiniciarse para comenzar a contar clicks nuevamente.

El sistema debe implementarse utilizando tres nodos en Lustre:

- Nodo MOUSE:

  Este es el nodo principal que recibe las entradas `CLOCK` y `CLICK` y produce las salidas `SINGLE` y `DOUBLE`.

  Debe llamar a los nodos `COUNTER` y `EMISSION` para realizar el procesamiento.

- Nodo COUNTER:

  Este nodo cuenta los clicks (`CLICK`) que ocurren en un período de dos ciclos de CLOCK.

  Debe generar una señal `ENABLE` que se active cada dos ciclos de `CLOCK` para indicar cuándo se deben emitir las salidas.

  Debe generar una señal `VAL` con valor de contador interno que se incrementa cada vez que ocurre un `CLICK`.

  Debe reiniciar el contador de clicks después de cada emisión.

- Nodo EMISSION:

  Este nodo recibe el valor del contador (VAL) y la señal `ENABLE`.

  Debe emitir las salidas `SINGLE` y `DOUBLE` según el valor de `VAL`, pero solo cuando `ENABLE` es true.

Por tanto, el núcleo reactivo va a consistir en un nodo principal llamado MOUSE con dos nodos internos llamados EMISSION y COUNTER, almacenados todos ellos en un fichero mouse.lus. Para compilar los tres nodos en Lustre la orden será:

```
$ lv6 mouse.lus -node MOUSE -2c-exec
```

Esto generará todos los ficheros necesarios para crear el fichero MOUSE.exec.

**ACTIVIDAD:**

- Tarea 1: Diseñar el Nodo COUNTER:
  1. Implementar un contador que cuente los clicks (CLICK) en un período de dos ciclos de CLOCK.
  2. Generar una señal ENABLE que se active cada dos ciclos de CLOCK.
  3. Reiniciar el contador después de cada emisión (cuando ENABLE es true).
- Tarea 2: Diseñar el Nodo EMISSION:
  1. Recibir el valor del contador (VAL) y la señal ENABLE.
  2. Emitir SINGLE si VAL = 1 y ENABLE es true.
  3. Emitir DOUBLE si VAL > 1 y ENABLE es true.
  4. Emitir false en ambas salidas si ENABLE es false.
- Tarea 3: Diseñar el Nodo MOUSE:
  1. Conectar las entradas CLOCK y CLICK al nodo COUNTER.
  2. Recibir las salidas VAL y ENABLE del nodo COUNTER.
  3. Llamar al nodo EMISSION con VAL y ENABLE para obtener las salidas SINGLE y DOUBLE.
- Tarea 4: Mostrar la salida mediante sim2chro para los tres nodos.

## Node: MOUSE

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| CLOCK (bool) | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ |
| CLICK (bool) | ☐ | ☑ | ☐ | ☐ | ☑ | ☑ | ☐ |
| SINGLE (bool) | ☐ | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ |
| DOUBLE (bool) | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ |
| VAL (int) | 0 | 1 | 1 | 0 | 1 | 2 | 0 |
| ENABLE (bool) | ☐ | ☐ | ☑ | ☐ | ☐ | ☑ | ☐ |

## Node: EMISSION

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| VAL (int) | 0 | 1 | 1 | 0 | 1 | 2 | 0 |
| ENABLE (bool) | ☐ | ☐ | ☑ | ☐ | ☐ | ☑ | ☐ |
| SINGLE (bool) | ☐ | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ |
| DOUBLE (bool) | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ |

## Node: COUNTER

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| CLOCK (bool) | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ |
| CLICK (bool) | ☐ | ☑ | ☐ | ☐ | ☑ | ☑ | ☐ |
| VAL (int) | 0 | 1 | 1 | 0 | 1 | 2 | 0 |
| ENABLE (bool) | ☐ | ☐ | ☑ | ☐ | ☐ | ☑ | ☐ |
| v (int) | 0 | 1 | 1 | 0 | 1 | 2 | 0 |
| RESET (bool) | ☐ | ☐ | ☐ | ☑ | ☐ | ☐ | ☑ |
| countdown (int) | 2 | 1 | 0 | 2 | 1 | 0 | 2 |