

## ARQUITECTURA DE COMPUTADORES BOLETÍN DE LABORATORIO 3: ENTRADA/SALIDA (2024-2025)

### **PARTE 1: CONTENIDO DE LAS SESIONES**

#### **1. REPASO DE PROGRAMACIÓN EN C**

De cara a manejar un dispositivo de entrada/salida real se precisa un lenguaje de alto nivel que sea capaz de acceder al hardware del dispositivo. El lenguaje que siempre se ha utilizado para ello ha sido C/C++. A continuación, se realizará un repaso de programación en lenguaje C básico.

##### **1.1. ESTRUCTURA BÁSICA DE UN PROGRAMA EN C**

En este apartado se expone un breve resumen de la programación en el lenguaje C. Se considera el siguiente contenido como un repaso de lo aprendido en otras asignaturas; es por ello que no se profundizará en su explicación.

El lenguaje C es puramente imperativo, de forma que no considera la creación de clases ni objetos; hay que recordar que este lenguaje es uno de los predecesores de los lenguajes orientados a objetos. La creación de un programa en este lenguaje consta de una serie de ficheros “.c” (ficheros de código) y “.h” (ficheros de cabecera). Por la simplicidad de los programas que se realizarán en esta práctica no será necesaria la creación de ningún fichero de cabecera y todo el código se implementará bajo el mismo fichero “.c”.

La estructura general, ordenada, de un fichero “.c” puede considerarse:

- a) Inclusión de librerías:  
Estructura: `#include <nombre_de_libreria.h>`  
Ejemplo: `#include <stdio.h>`
- b) Definición de constantes:  
Estructura: `#define NOMBRE VALOR`  
Ejemplo: `#define MAX 128`
- c) Definiciones teóricas de estructuras  
Estructura: `struct nombre_struct{tipo campo1; tipo campo2; ...};`  
Ejemplo: `struct miEstructura{int c1; float c2;};`
- d) Prototipos de funciones:  
Estructura: `tipo nombre_funcion(parámetros);`  
Ejemplo: `int miFuncion(int, float);`
- e) Declaración de variables globales:  
Definición e inicialización idénticas a variables locales; únicamente varía su ámbito.
- f) Función principal (main)  
Según el compilador: `main(){...}, void main(){...}, void main(void){...}, int main(){...}, ...`
- g) Implementación del resto de funciones

Los tipos básicos (atómicos) del lenguaje C son los comúnmente utilizados en la gran mayoría de los lenguajes: ***int y char (enteros); float y double (reales en punto flotante).***

A estos tipos se les puede aplicar una serie de modificadores de tipo: ***signed, unsigned, long y short;*** y una serie de modificadores de comportamiento: ***static, auto, register, const y volatile.***

Los tipos compuestos de este lenguaje son, principalmente, dos:

- a) Vectores:
  - i. Unidimensionales: `tipo nombre[tamaño];` / `tipo nombre[tamaño]={valor0, valor1, ...};`
  - ii. Multidimensionales: `tipo nombre[dimension0][dimension1][...];`

**b) Estructuras de datos:**

- i. Definición teórica de la estructura: ***struct nombreEst{tipo0 var0; tipo1 var1; ...};***
- ii. Creación de variable: ***struct nombreEst nombreVar;***
- iii. Acceso: ***nombreVar.var0*** a operador “.”.

Un ejemplo global es el siguiente:

```
#include <stdio.h>
#define PI 3.141592
struct DatosCircunferencia{
    float area;
    float perimetro;
};

struct DatosCircunferencia calculaDatos (float);

int main (){
    float radio;
    struct DatosCircunferencia res;
    printf("Calculo del area y perimetro de una circunferencia dado el radio\n");
    printf("Introduzca el radio de la circunferencia: ");
    scanf("%f", &radio);
    res = calculaDatos (radio);
    printf("El resultado ha sido\n Area: %f\n Perimetro: %f", res.area, res.perimetro);
    return 0;
}

struct DatosCircunferencia calculaDatos (float r){
    struct DatosCircunferencia dat;
    dat.area = PI*r*r;
    dat.perimetro = 2*PI*r;
    return dat;
}
```

## 1.2. OPERADORES PROPIOS PARA ENTRADA/SALIDA

Al trabajar con dispositivos de entrada/salida, se accede a registros de datos (comúnmente para nosotros de 8 bits de tamaño), donde se requerirá consultar el estado de algunos bits concretos, además de requerir modificar el valor de ciertos bits.

Para ello, se precisa utilizar operadores binarios sobre los datos que manejemos en nuestro programa en C. En este caso, veremos dos tipos de operadores:

### 1.2.1. DESPLAZAMIENTO

En el Lenguaje C y C++ se pueden realizar desplazamientos a nivel de bits (equivalentes a sll y srl/sra del ensamblador) utilizando los siguientes operadores:

- >> : desplazamiento a la derecha
- << : desplazamiento a la izquierda

La nomenclatura de estos operadores es la siguiente:

*Valor\_a\_desplazar* **OPERADOR** *Num\_bits\_a\_desplazar*

Algunos ejemplos de funcionamiento:

- $8 \gg 1$  à Resultado: 4
- $8 \ll 2$  à Resultado: 32

Y un ejemplo con variables:

```
unsigned int a = 16, b;  
b = a >> 3;           // En 'b' se guarda un 2.
```

### 1.2.2.MÁSCARAS

En el Lenguaje C el tipo de datos más pequeño disponible es el char de 8 bits. Otros son el int (32 bits), el float (32 bits) o el double (64 bits). De igual forma, cuando se lee o escribe en un registro de E/S en C se han de leer o escribir como mínimo 8 bits (char). También es posible 16 bits (short int) o 32 bits (int).

Sin embargo, en la codificación de los registros de un controlador de entrada/salida, es bastante común utilizar bits independientes para tareas concretas. Por lo tanto, debemos ser capaces de consultar y/o modificar el valor de un bit específico dentro de una variable de tipo char, short int o int.

Para esta tarea, se utilizan operaciones lógicas adecuadas que toman como primer operando la secuencia de bits (englobada dentro de la variable correspondiente), seguro del propio operando (que puede ser de dos tipos) y, finalmente, una secuencia predeterminada, denominada “máscara”, que indique sobre qué bit (o bits) se va a actuar. Esta máscara se representará en hexadecimal o decimal (aunque se recomienda en hexadecimal para evitar confusiones). Esto es:

*Secuencia\_de\_bits* **OPERADOR** *Máscara*

Por otro lado, el tipo de operador marcará, además de qué operación binaria se realiza sobre la secuencia de bits, la máscara a aplicar.

Tendremos dos tipos de operadores para máscaras (también denominados “tipos de máscaras”):

- Operador AND à ‘&’
- Operador OR à ‘|’

Estos operadores aplican la operación binaria ‘and’ u ‘or’ sobre la secuencia de bits. No confundir con ‘&&’ y ‘||’ que equivalen a las operaciones booleanas ‘Y’ y ‘O’, no guardando ninguna relación con las operaciones binarias.

A continuación, se presentan los diversos casos de aplicación dependiendo de lo que queremos hacer sobre la secuencia de bits:

- Caso 1 – Modificar valor de bit:
  - A ‘0’ à operador AND con bit que se quiere cambiar a ‘0’
  - A ‘1’ à operador OR con bit que se quiere cambiar a ‘1’

Se pueden cambiar varios bits a la vez, sin más que representar en hexadecimal aquellos bits sobre los que se quiere actuar.

Ejemplos:

- Modificar bit 3 de la variable V a ‘1’ y guardar en W à  $W = V | 0x8$ ;
- Modificar bit 2 de la variable V a ‘0’ y guardar en W à  $W = V \& 0xB$ ;
- Caso 2 – Consultar valor de bit:
  - Se puede utilizar cualquier mascara... pero mejor operador AND
  - En general à operador AND con máscara todo a ‘0’ salvo bit a consultar
  - Subcasos
    - Caso 2.1 – Consulta si bit es ‘0’ à  $(reg \& \text{máscara}) == 0$
    - Caso 2.2 – Consulta si bit es ‘1’ à  $(reg \& \text{máscara}) != 0$

### 1.3. ENTORNO DE PROGRAMACIÓN: ONLINE GBD COMPILER

En el contexto de la asignatura no se precisa un entorno de programación que gestione múltiples proyectos y soluciones, sino un entorno sencillo que permita crear programas básicos de consola fácilmente.

Por ello, se utilizará un entorno online: Online GBD Compiler and Debugger for C/C++.

Puede utilizarlo a través de cualquier navegador en la URL:

[https://www.onlinegdb.com/online\\_c\\_compiler](https://www.onlinegdb.com/online_c_compiler)

Una vez hayamos accedido a esta web, tendremos un proyecto base ya creado y listo para ejecutar por consola.








Puede observar la pantalla principal en la siguiente imagen:








Puede observar que, por defecto, posee ya una función principal con su estructura básica y la inclusión de la librería de entrada/salida estándar (stdio.h).

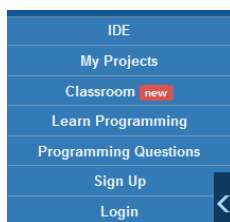
En la parte superior puede observar las siguientes opciones (en este orden):



-  Nuevo fichero (New File): permite crear un fichero nuevo en blanco, indicando el nombre.
-  Subir fichero (Upload File): permite cargar un fichero creado previamente que tengamos en local.
-  Arranque (Run): ejecuta el programa creado y muestra los resultados en la ventana negra inferior.
-  Depuración (Debug): permite ejecutar el programa paso a paso siempre y cuando hayamos incluido previamente puntos de ruptura en el código.
-  Parada (Stop): podremos parar la ejecución de un código en el momento que estimemos oportuno.
-  Compartición (Share): genera una URL accesible públicamente para el código que ha creado.
-  Guardar (Save): si se ha logueado con una cuenta propia, esta opción permitirá guardar el programa creado en su propia biblioteca de programas.

-  Embellecer el código (Beauty): permite reorganizar el código y tabularlo correctamente para mejorar su visibilidad. Mucho cuidado con los tipos de devolución de las funciones, que se separan de los nombres de las mismas.
-  Descarga de código: permite descargar el código creado como un fichero "main.c".
-  Selección del lenguaje de programación: por defecto está seleccionando "language C", que es ANSI C estándar. En este desplegable se pueden seleccionar otros lenguajes de programación que no serán utilizados en la asignatura.
-  Información: mostrará una ventana emergente con la información concerniente a los atajos de teclado.
-  Configuración: permite seleccionar el modo de visualización del entorno, incluyendo tamaño de letra, espaciado, tema, etc.

En la parte lateral puede observar las siguientes opciones (en este orden):



- IDE: vuelve a cargar el entorno online desde cero. Mucho cuidado, porque esta opción borra lo que tengamos, sin guardar nada.
- My Projects: Si nos hemos logueado previamente, esta opción permitirá visualizar los proyectos guardados.
- Classroom: no disponible en la versión gratuita.
- Learn Programming: nos redirige a una web propia con tutoriales básicos de programación.
- Programming Questions: nos redirige a un foro propio donde poder postear preguntas concretas.
- Sign Up: nos permite crear una cuenta.
- Login: Nos permite loguearnos con nuestras credenciales.

Como se ha comentado al inicio, haremos uso únicamente de aspectos básicos de programación; por lo tanto, no es necesario crear cuenta en el entorno. De igual forma, a no ser que creamos un programa muy complejo, tampoco será necesario realizar depuración.

## RECURSOS:

En internet pueden encontrarse números cursos y tutoriales donde encontrar explicaciones detalladas del lenguaje C. A continuación, se muestran algunos de ellos

- Material didáctico de la asignatura "Fundamentos de programación" de primero disponible en rodas.us.es:  
<http://rodas.us.es/items/5c12ed9d-e398-4570-a30c-991953e92eed/1/>  
<http://rodas.us.es/items/c3145fb0-ed17-4070-9bef-7ec707caa971/1/>
- Blog "La programación no es un arte" del Prof. Riquelme:  
<http://laprogramacionnoesunarte.blogspot.com.es/>
- Libro electrónico de libre acceso: "Aprenda lenguaje ANSI C como si estuviera en primero"  
<http://www.josedomingo.org/web/mod/resource/view.php?id=806>

## 2. SIMULACIÓN DE E/S EN VISUALRISC-V

### 2.1. INTRODUCCIÓN

Los procesadores RISC-V se conectan al subsistema de entrada-salida mediante E/S mapeada en memoria, por lo que se comunica con los dispositivos de E/S a través de un espacio de direcciones común que comparte con el espacio de direcciones a memoria. De esta forma, los accesos a los registros internos de los dispositivos se realizan de la misma forma que los accesos a memoria, esto es, empleando instrucciones de carga (*load*) para leer de los dispositivos e instrucciones de almacenamiento (*store*) para escribir en ellos.

		Operación	
		Entrada (E/S → Registro)	Salida (Registro → E/S)
Acceso	8 bits	lbu rd, Inm(rs)	sb rs2, Inm(rs1)
	16 bits	lhu rd, Inm(rs)	sh rs2, Inm(rs1)
	32 bits	lw rd, Inm(rs)	sw rs2, Inm(rs1)

*Instrucciones de carga y almacenamiento para acceder los dispositivos*

Para acceder a un dispositivo es necesario consultar sus hojas de especificaciones pues éstas detallan todo lo necesario para realizar la comunicación con el dispositivo: su comportamiento general, los registros internos que posee de cada tipo (registros de control, de estado, de datos, o una mezcla de ellos) y la especificación de cada registro (tamaño de cada registro, interpretación de sus bits, comportamiento, etc.). De esta manera, para acceder a un registro interno de un dispositivo de E/S que está mapeado es necesario tener en cuenta si está disponible, qué operaciones permite (lectura y/o escritura) y el tamaño del acceso.

La tabla anterior muestra las instrucciones que se utilizan en función de la operación y del tamaño del acceso. Por lo general, las lecturas de 8 y 16 bits suelen realizarse sin signo pues la información obtenida suele interpretarse a nivel de bits. No obstante, si la información leída debe interpretarse en CA-2, que podría darse el caso al leer principalmente registros de datos, sería necesario utilizar las versiones con signo (lb y lh).

Otras instrucciones que serán de utilidad para obtener la dirección del dispositivo y manipular el contenido de un registro a nivel de bits son:

li rd, inm32	Guarda en el registro rd un inmediato de 32 bits (dirección base del dispositivo)
andi rd, rs, inm12	Operación lógica <b>and</b> con un inmediato (para máscaras)
ori rd, rs, inm12	Operación lógica <b>or</b> con un inmediato (para máscaras)
slli rd, rs1, inm5	Desplazamiento a la <b>izquierda</b> de <i>inm5</i> posiciones
srlr rd, rs1, inm5	Desplazamiento a la <b>derecha</b> de <i>inm5</i> posiciones

Puesto que el espacio de direcciones es compartido para los accesos a memoria y a los dispositivos de E/S, debe haber un mecanismo que permita determinar qué direcciones son mapeadas al subsistema de E/S o la memoria. Muchos procesadores lo resuelven reservando para la E/S un rango fijo de direcciones del espacio de direcciones, bien al principio o al final del espacio. En otros procesadores como RISC-V no se especifica esta restricción, aunque sería necesario establecer las direcciones utilizadas para E/S como no *cacheables* con el fin de evitar problemas de consistencia entre la caché y la memoria principal. Estos problemas de consistencia podrían aparecer en direcciones cacheadas debido a los efectos colaterales que se producen en las operaciones de lectura y escritura en los periféricos. En el caso del RISC-V implementado en el simulador, no hay limitaciones a la hora de mapear los dispositivos siempre que se encuentren dentro del rango de direcciones definidos para el usuario (rango 0x00010000-0xBFFFFFF0). Para mapear en direcciones fuera de este rango, debe utilizarse la directiva: *fullaccess on*.

Por otro lado, al mapear un dispositivo en el espacio de direcciones, se utiliza una dirección base a partir de la cual se mapearán todos los registros accesibles en el dispositivo de E/S. Estos registros están identificados con un offset, de forma que la dirección de acceso a cada registro del dispositivo se obtiene de:

### Dirección\_RegE/S = Dirección\_Base\_Dispositivo + Offset\_Registro

Como en RISC-V los accesos a direcciones emplean el modo de direccionamiento Registro Base + Desplazamiento (“Inm(rs)”), es habitual utilizar el registro base (rs) para indicar la dirección del dispositivo y el offset (Inm) para especificar el registro al que acceder. Por ejemplo, en un dispositivo con 3 registros de 8 bits mapeado a partir de la dirección 2000 se obtendría:

Dirección base: 2000	Dispositivo	
Dirección 2000 →	RegA (8 bits)	Offset +0
Dirección 2001 →	RegB (8 bits)	Offset +1
Dirección 2002 →	RegC (8 bits)	Offset +2

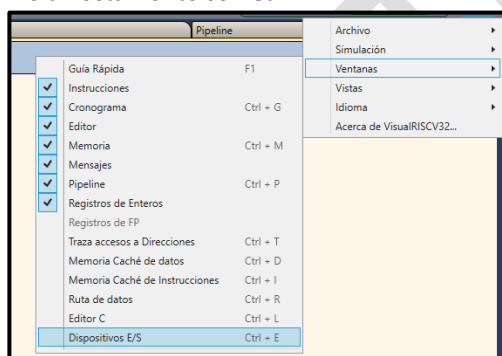
Teniendo en cuenta el ejemplo, un fragmento de código para guardar en *RegC* el contenido del *RegB* podría ser el siguiente:

```
...
addi x2,x0,2000 #use li para inmediatos>12 bits
lbu x3,1(x2)
sb x3,2(x2)
```

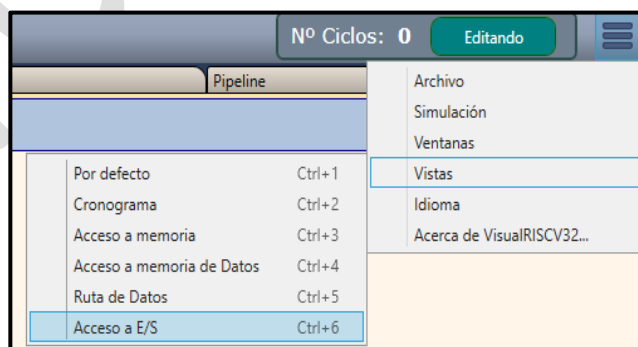
## 2.2. ENTORNO DE VISUALRISC-V PARA LA SIMULACIÓN DE LA E/S

El simulador VisualRISC-V integra un mecanismo para el acceso a la E/S muy simple que permite acceder a una serie de dispositivos virtualizados. Esta característica ya estaba disponible en la versión v22.3.22 pero en versiones posteriores se han realizado cambios importantes que requieren el uso de estas nuevas versiones para el desarrollo de las prácticas de E/S relacionadas con este simulador.

Para el acceso a la E/S en VisualRISC-V se necesita, además de las ventanas utilizadas en las prácticas anteriores, la ventana ‘Dispositivos E/S’ que se encuentra en Menú → Ventanas → Dispositivos E/S o directamente con *Ctrl+E*.



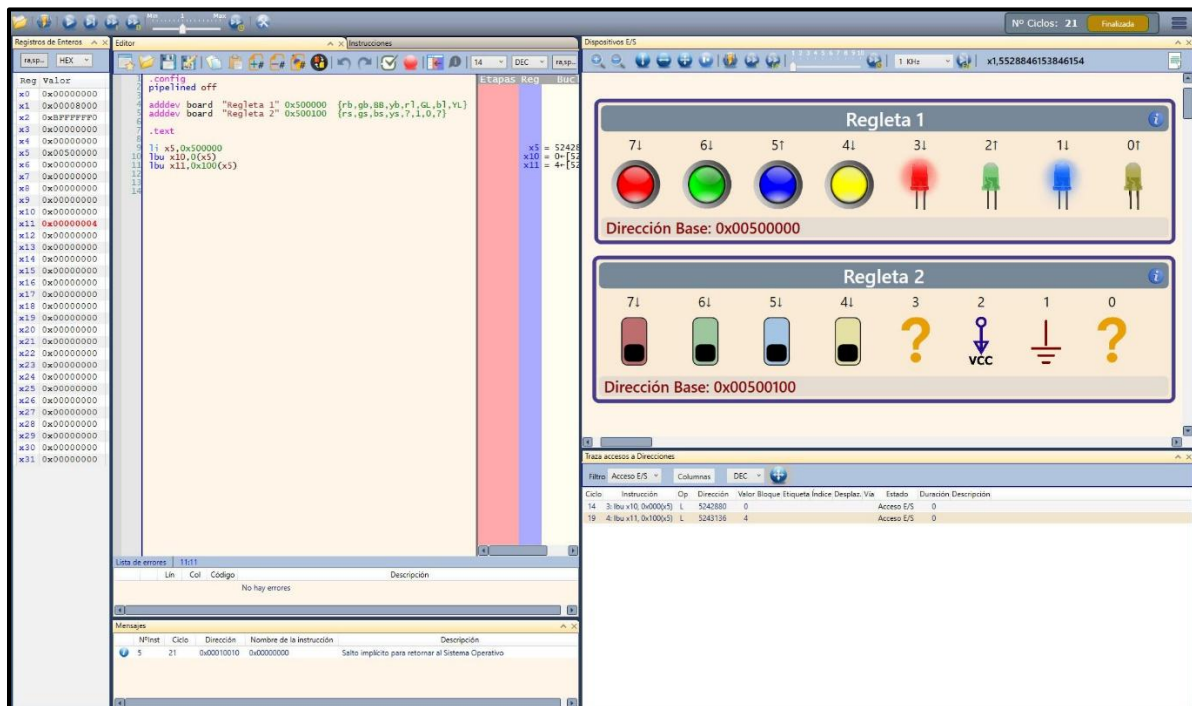
Menú de acceso a la ventana ‘Dispositivos de E/S’



Menú de Acceso a la Vista ‘Acceso a E/S’

Para distribuir las ventanas automáticamente de una forma más conveniente para el estudio de los accesos a E/S puede utilizarse la vista ‘Acceso a E/S’ que se obtiene con *Ctrl+6* o accediendo a Menú → Vistas → Acceso a E/S. La siguiente imagen corresponde a una captura con la vista ‘Acceso a E/S’ conectados a 2 dispositivos ‘board’.

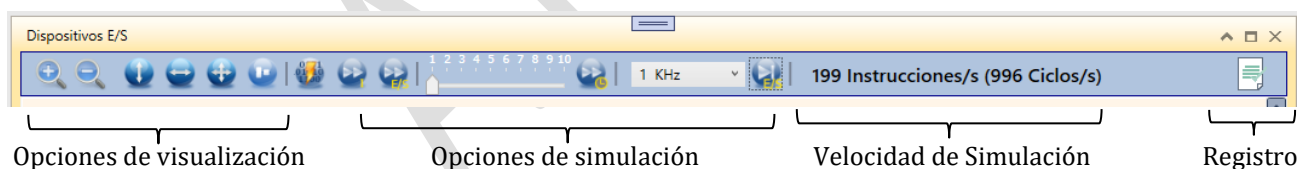




Captura de VisualRISC-V en la vista 'Acceso a E/S'

Las Mayoría de las ventanas que se muestra en la imagen ya fueron estudiadas a través de prácticas anteriores, aunque la ventana 'Trazo de accesos a Direcciones' vuelve a tomar protagonismo pues resultará especialmente útil para verificar los accesos realizados a los dispositivos. Esta ventana contiene la lista de todos los accesos y permite aplicar filtros para obtener únicamente los Accesos de E/S.

En la parte superior derecha se encuentra la ventana 'Dispositivos E/S' que ofrece un espacio de trabajo en el que aparecerán los dispositivos conectados al computador. Esta ventana dispone de una cinta de opciones para asistir a la visualización y facilitar la depuración. El aspecto de la cinta se muestra a continuación:



Las 'Opciones de Visualización' permiten ajustar el tamaño y distribuir los dispositivos en el espacio de trabajo de forma automática. Posteriormente se encuentra el botón *reset* que se carga de reiniciar la simulación (actúa de la misma que el botón reset que aparece en la ventana principal).

Las opciones de simulación se encargan de controlar el ritmo de la simulación. Cada grupo de elementos dentro de estas opciones realizan la simulación de forma ligeramente diferente con el objetivo de adecuarse a las necesidades del usuario. Las opciones de simulación que se encuentran más a la izquierda se utilizan en las fases iniciales de la programación pues van encaminadas a simulaciones paso a paso y las que se encuentran más a la derecha realizan una simulación en 'modo continuo' teniendo en cuenta la temporización.

El primer botón de las opciones de simulación (lleva inscrita una 'I'), realiza la simulación de la siguiente instrucción. Cuando el procesador está configurado en modo secuencial (*pipelined off*) resulta especialmente útil para verificar las operaciones a nivel de bit que el simulador ejecuta.

El segundo botón realiza la simulación de una serie de ciclos hasta el siguiente acceso a la E/S. Esta opción puede ser de utilidad para comprobar de una forma más rápida si los valores leídos del dispositivo de E/S o escritos en él son los esperados.



Los métodos anteriores, al basarse en la simulación paso a paso, tiene importantes limitaciones en determinadas situaciones, por ejemplo, interactuar con dispositivos básicos como botones puede resultar imposible (no se puede estar pulsando un botón con el ratón para verificar su comportamiento mientras se accede a las opciones de simulación ciclo a ciclo). En cambio, las siguientes dos opciones sí realizan una simulación de forma automática, lo que va a permitir interactuar con los dispositivos más fácilmente, aunque de forma ligeramente diferente entre ellas.

La tercera opción para realizar la simulación se controla mediante un botón de deslizamiento para seleccionar la frecuencia de simulación (entre 1 y 10 Hz) y el botón adjunto (que muestra un reloj). En este método, la interfaz gráfica del simulador se actualiza completamente en cada ciclo de reloj: valores de los registros, direcciones de memoria, traza de accesos, cronograma, ruta de datos, etc., lo que permite visualizar el estado del computador durante la simulación de forma íntegra. Esta opción sólo está disponible hasta 10 ciclos de reloj pues frecuencias mayores no sería útil al no poder percibirse el estado debido a que actualiza cada periodos de tiempo muy cortos.

El último método de simulación se ejecuta a frecuencias relativamente mayores (1KHz por defecto) y resulta adecuado cuando el dispositivo requiere muchos ciclos de espera o simplemente para interactuar con los dispositivos de una forma más fluida. En este caso la interfaz gráfica de la aplicación no se actualiza en cada ciclo a excepción de la ventana que contiene los dispositivos. Al detener la simulación, se actualiza la interfaz gráfica para mostrar el estado actual del computador. La frecuencia máxima que soporta la simulación está condicionada por la potencia del computador en el que corre simulador. Ha de tenerse en cuenta que, en este modo, el simulador sigue calculando y registrando el estado del sistema en cada uno de los ciclos simulados, aunque no se muestren durante la simulación. Por este motivo, no podrá alcanzarse altas frecuencias de simulación y será necesario mucha memoria principal si se ejecuta durante muchos ciclos. En este modo, la cinta de opciones mostrará también los ciclos por segundos reales al que se está realizando la simulación.

El botón de registro, que se encuentra en el extremo derecho de la cinta, muestra los accesos que se han realizado a los dispositivos de forma incorrecta, por ejemplo, si se intenta escribir en un registro de sólo lectura o si intenta modificar algún bit que debe permanecer intacto.

### 2.3. DISPOSITIVOS EN VISUALRISC-V

El simulador integra un conjunto reducido de dispositivos, aunque su número podrá aumentar en futuras versiones. En este boletín se recoge la descripción de 3 de los dispositivos de incluye y que serán utilizados en prácticas: **board**, **arrows** y **matrix**.

Los dispositivos se añaden al simulador mediante directivas '**adddev**' en la sección 'config' del programa. Puede añadirse varios dispositivos del mismo tipo si fuese necesario y que podrán distinguirse mediante un nombre que los identifique. En algunos casos, la ventana de propiedades permite configurar algunos parámetros por defecto, pero lo habitual será configurarlo en la directiva.

#### 2.3.1.DISPOSITIVO 'BOARD'

Consiste en una regleta de 8 pines que permite conectar un elemento básico por cada pin. Por tanto, dispone de un puerto de 8 bits en el que cada bit se corresponde con uno de los pines al que está conectado un elemento. En cada pin puede conectarse: leds, pulsadores (button) o interruptores (switch). Además, puede simular que un pin está conectado a masa para que su valor sea siempre '0', a Vcc para sea siempre '1' o conectarse a un elemento especial identificado por "?" en el que su valor es indeterminado. Un ejemplo del dispositivo *board* ya configurado se muestra a continuación:



Ejemplo de un dispositivo *board* con los componentes básicos más comunes.

En la imagen puede distinguirse los siguientes elementos: en el pin 7 un pulsador rojo configurado a nivel bajo (identificado por ↓), en el pin 6 un diodo led rojo activo a nivel bajo que se encuentra encendido, el pin 5 permanecerá siempre a '1', el pin 4 es un interruptor azul (switch) activo a nivel bajo y que actualmente está abierto (en off), en el pin 3 un led azul activo por nivel alto (identificado por ↑) que está apagado, el pin 2 está a masa por lo que siempre vale '0', un pulsador verde en el pin 1 y, finalmente, un elemento '?' en el pin 0 que indica que su valor es impredecible ya que cambia aleatoriamente en cada reset del simulador.

La sintaxis para añadir un dispositivo *board*:

***adddev board [Identificador] [Dirección\_base] [Patrón]***

**Identificador:** cadena entre comillas dobles que le proporciona un nombre único al dispositivo.

**Dirección\_base:** dirección del espacio de direcciones en el que se mapea el único puerto que posee.

**Patrón:** secuencia de 8 elementos entre llaves y separados por comas que identifica los 8 componentes básicos conectados al *board*.

Para especificar cada elemento del patrón se utiliza:

<b>1</b>	Conectado a Vcc
<b>0</b>	Conectado a Masa
<b>?</b>	Conectado a un componte desconocido

Y los componentes básicos se especifican mediante un código de 2 dígitos:

XY	
<b>X</b>	Identifica el color del componente. r → red, g → green, b → blue y → yellow
<b>Y</b>	Determina el tipo de componente. l → led, b → button, s → switch

Si el código está en mayúsculas configura el pin a nivel alto (lógica positiva). En caso contrario a nivel bajo (lógica negativa, activación por '0')

Por Ejemplo, YB conectará un botón amarillo (*Yellow Button*) a nivel alto.

Teniendo en cuenta esta descripción, el dispositivo que se muestra en la imagen del ejemplo se obtiene mediante el patrón: **{rb,rl,1,bs,BL,0,GB,?}**

Y la directiva que configura el ejemplo sería:

***adddev board "Mi Regleta" 0xA0000000 {rb,rl,1,bs,BL,0,GB,?}***

### 2.3.2.DISPOSITIVO 'ARROWS'

Es un dispositivo que emula los cursores de un teclado. Su ventaja respecto a los pulsadores que se emplean en *board*, además de su diseño, es que pueden ser pulsados con el teclado del ordenador. Las teclas de los cursores se conectan a un puerto de 8 bits, bien sea a nivel alto y/o bajo, y es posible conectar una tecla a varios bits del puerto o dejarla sin conectar.



Imagen del dispositivo 'arrows'

La sintaxis de configuración es similar al dispositivo *board*, aunque el patrón es diferente:

***adddev arrows [Identificador] [Dirección\_base] [Patrón]***

**Patrón:** secuencia de 8 caracteres entre llaves (no va separados por comas) que especifica a qué pin se conecta cada tecla de los cursores. Para especificar cada tecla se utiliza la siguiente codificación:

<b>L/l</b>	Izquierda ( <b>Left</b> )
<b>R/r</b>	Derecha ( <b>Right</b> )
<b>U/u</b>	Arriba ( <b>Up</b> )
<b>D/d</b>	Abajo ( <b>Down</b> )

Como en el dispositivo anterior, si se especifica en mayúsculas la pulsación provoca un nivel alto, y en caso contrario provoca un nivel bajo (la pulsación provocaría un '0' en la entrada).

Igualmente pueden utilizarse los elementos comodines '1', '0' y '?' para valores fijos o indeterminados.

Un ejemplo de conexión de este dispositivo sería:

```
adddev arrows "cursores" 0x30000 {1r?U0DL}
```

En este ejemplo, el cursor de la izquierda está conectado al bit 7 a nivel bajo y al pin 0 a nivel alto simultáneamente.

### 2.3.3.DISPOSITIVO 'MATRIX'

Consiste en una pantalla de píxeles a color cuya resolución es configurable. El control de la pantalla se realiza mediante comandos. Los comandos que actúan sobre un píxel específico deberán especificar la dirección del píxel. La siguiente imagen muestra un ejemplo del dispositivo:



Ejemplo de dispositivo 'Matrix' tras dibujar una diagonal en rojo

Para el control del dispositivo se utilizan los siguientes registros internos:

0	Fila	Dirección Fila, Columna para especificar un píxel de la pantalla.	
1	Columna		
2	Rojo	Componentes Rojo, Verde y Azul para especificar el color de un píxel en formato RGB	
3	Verde		
4	Azul		
5	Estado/ Control	Registro utilizado como estado y control. En este registro se distinguen los siguientes bits:	
		B/b:	Bit 'busy'. Indica si el dispositivo está ocupado o disponible para procesar un nuevo comando
		C/c:	Comando 'Clear'. Borra la pantalla.
		U/u:	Comando 'Unset'. Desactiva el pixel (pasa a negro) especificado en los registros Fila, Columna
		S/s:	Comando 'Set'. Activa el pixel especificado en los registros Fila, Columna con el color RGB indicado en los registros Rojo, Verde y Azul.

Para añadir el dispositivo puede utilizarse una de las siguientes directivas:

```
adddev matrix [NOMBRE] [DIRECCION] [FILAS] [COLUMNAS]
```

```
adddev matrix [NOMBRE] [DIRECCION] rows=[FILAS] columns=[COLUMNAS]  
latency=([LATMIN],[LATMAX]) pattern=[PATRON]
```

**FILAS** y **COLUMNAS** especifican la resolución de la matriz de puntos siendo 64x64 la resolución máxima.

**LATMIN** y **LATMAX** especifican el número de ciclos mínimo y máximo que será necesario para que el dispositivo quede nuevamente disponible tras el envío de un comando (véase el bit 'Busy').

**PATRON** especifica el patrón del registro Control/Estado (Registro 5). Este patrón consiste en 8 caracteres encerrados entre llaves que especifican las posiciones de los bits asociados al registro 5: 'Busy' (**B/b**), 'Clear' (**C/c**), 'Unset' (**U/u**) y 'Set' (**S/s**). Estos bits podrán especificarse una sola vez dentro del patrón, bien en mayúsculas para utilizar lógica positiva o minúsculas en caso contrario. El resto de los bits del patrón, podrán ser, como en dispositivos anteriores '1', '0' o '?' según deban mantenerse siempre a 1, a 0, o con un valor indeterminado.

Todos los parámetros excepto **NOMBRE** y **DIRECCIÓN** son opcionales por lo que, en caso de omitirse, tomará por defecto los parámetros que aparecen en la ventana de propiedades. Ha de tenerse en cuenta que la modificación de los valores por defecto se mantiene de una sesión a otra. La siguiente imagen muestra los valores por defecto que aparecen en la ventana de propiedades:

Dispositivo : **matrix**

**Registro de Datos 'ROW' (Offset 0)**  
Número de Filas (1 - 64)

**Registro de Datos 'COLUMN' (Offset 1)**  
Número de Columnas (1 - 64)

**Registro de Datos 'PIXEL\_RED' (Offset 2)**

**Registro de Datos 'PIXEL\_GREEN' (Offset 3)**

**Registro de Datos 'PIXEL\_BLUE' (Offset 4)**

**Registro de Estado/Control (Offset 5)**

7	6	5	4	3	2	1	0
B	?	0	?	0	U	S	c

Latencia [Min - Max]:  -  Ciclos

Legend:  
B/b: Busy/Busy  
C/c: Clear/Clear  
S/s: Set/Set  
U/u: Unset/Unset

Valores por defecto del dispositivo 'Matrix' en la ventana propiedades.

## **PARTE 2: EJERCICIOS DE LABORATORIO**

**Ejercicio 1:** Indique, al menos, 5 errores cometidos en el siguiente código (para lenguaje ANSI C) y qué haría para solucionarlos.

```
#include <stdio.h>
int main(){
    unsigned int res, num;
    scanf_s("%f", num);
    res = fib(num);
    printf("Resultado: \n", res);
    return 0;
}
void fib(unsigned int n){
    int a=0, b=1, r;
    if(n=0)
        r = a;
    if else(n=1)
        r = b;
    else
        for(int i=2; i<n; i++)
        {
            r = a + b;
            a = res;
            b = r;
        }
}
```

**Ejercicio 2:** Rellene los huecos para que el programa de la derecha tenga el mismo comportamiento que el de la izquierda. **Nota:** si no sabe por dónde empezar, fíjese en las diferencias y busque información acerca de punteros.

<pre>float calcArea(float r); int main(){     float radio, res;     scanf_s("%f", &amp;radio);     res = calcArea(radio);     printf("Area: %.2f\n", res);     return 0; } float calcArea(float r){     return (3.141592*r*r); }</pre>	<pre>void calcArea(float r, _____); int main(){     float radio, res;     scanf_s("%f", &amp;radio);     calcArea(radio, _____);     printf("Area: %.2f\n", _____);     return 0; } void calcArea(float r, _____){     _____ = 3.141592*r*r; }</pre>
--	--

**Ejercicio 3:** Rellene los siguientes apartados:

Defina un tipo de estructura llamada <b>coche</b> , que contenga internamente dos campos: una cadena de 50 caracteres que indique el <b>tipo de coche</b> y un número entero con el <b>número de bastidor</b> .	
Cree una variable del tipo estructura definido antes, y llámela <b>miCarro</b> .	
Modifique el tipo de coche de la variable <b>miCarro</b> , por <b>"DeLorean DMC-12"</b> .	
Cree un vector bidimensional de elementos de tipo entero, de 50 filas y 20 columnas. Llámelo <b>matriz</b> .	
Modifique el valor del elemento situado en la primera fila y última columna por <b>32</b> .	

**Ejercicio 4:** Rellene el siguiente programa en C para que almacene en un vector (v2) los elementos pares de otro vector (v1) y, finalmente, muestre por pantalla el resultado.

```
#include <stdio.h>
int main()
{
    int v1[10]={3, 8, 2, 0, 18, 5, 9, 4, 10, 1}, v2[10];
    ...
    return 0;
}
```

**Ejercicio 5:** Realice un programa que pida por teclado un número entero sin signo y, posteriormente, lo muestre por pantalla utilizando varios modificadores de tipos sobre la función “printf”: %d (para mostrarlo como entero), %u (para mostrarlo como entero sin signo) y %x (para mostrarlo como hexadecimal).

**Ejercicio 6:** Utilice los operadores de desplazamiento de bits que aporta C para, con el dato recibido por teclado del ejercicio anterior, mostrar por pantalla el resultado de desplazar dicho número dos bits a la izquierda y dos bits a la derecha.

**Ejercicio 7:** Realice un programa que lea de teclado un número entero sin signo, y pinte por pantalla este mismo entero, pero mostrando solo la parte baja de la palabra (16 bits menos significativos).  
**Nota:** haga uso de las máscaras en C para conseguir tal propósito.

**Ejercicio 8:** Repita el ejercicio anterior pero, en este caso, para la parte alta.

**Ejercicio 9:** Combine el ejercicio 7 y el 8, de forma que el programa deberá mostrar en una línea la parte alta y en la siguiente la parte baja del número introducido desde teclado.

**Ejercicio 10:** Realice un programa que recoja de teclado un número entero sin signo de 8 bits (“unsigned char”, utilizando el identificador “%hhu” de la función ‘scanf’). Seguidamente, modificará el bit número 3 (cuarto bit menos significativo) a ‘1’ y mostrará el resultado por pantalla (utilizando la representación en hexadecimal “%x”). Nota: intente que el número introducido no tenga un ‘1’ en binario en la posición 3 para poder ver los resultados. Se recomienda probar con el número 4 que, tras la modificación, debería salir por pantalla 0xC (12 si lo pinta como número entero).

**Ejercicio 11:** Realice un programa que recoja de teclado un número entero sin signo de 8 bits (“unsigned char”, utilizando el identificador “%hhu” de la función ‘scanf’), modificará el bit número 5 (sexto bit menos significativo) a ‘0’ y mostrará el resultado por pantalla (utilizando la representación en hexadecimal “%x”). Nota: intente que el número introducido SÍ tenga un ‘1’ en binario en la posición 5 para poder ver los resultados. Se recomienda probar con el número 32 en decimal que, tras la modificación, debería salir por pantalla 0.

**Ejercicio 12:** Realice un programa que recoja de teclado un número entero sin signo de 8 bits (“unsigned char”, utilizando el identificador “%hhu” de la función ‘scanf’) y que consulte el estado del bit número 2 (tercer bit menos significativo). Si dicho bit tiene el valor ‘0’ deberá ponerlo a ‘1’, y si tiene el valor ‘1’ deberá ponerlo a cero. Muestre el resultado por pantalla utilizando la representación hexadecimal.

**Ejercicio 13:** Realice un programa que recoja continuamente valores enteros sin signo de 8 bits desde teclado hasta que el usuario introduzca uno cuyo bit 7 (octavo bit menos significativo) sea ‘1’. Para ello, programe un bucle “while” en el que continuamente realice llamadas a “scanf”, y que compruebe la condición en cada iteración. Si se cumple, saldrá del bucle y finalizará el programa; si no es el caso, seguirá solicitando números al usuario. Nota: un número únicamente con el bit 7 a ‘1’ es el 128 (en decimal); por lo tanto, para que se cumpla la condición en un número de 8 bits sin signo, el valor debe ser superior o igual a 128.

**Ejercicio 14:** Haciendo uso del VisualRISCv utilice el siguiente fragmento de código para añadir 2 dispositivos *board* con componentes básicos y realice los siguientes apartados:

```
.config
fullaccess on
pipelined off
adddev board "Panel1" 0x50000 {1,BS,gs,BL,r1,GL,0,RS}
adddev board "Panel2" 0x50001 {RL,GL,RL,GL,RL,GL,RL,GL}
```

- Seleccione la vista 6 en el programa y realice un reset para visualizar los dispositivos.
- Realice un programa que, al accionar el interruptor rojo, se enciendan todos los leds del panel2 y se apaguen al desactivar el interruptor.



- c) Modifique el código anterior para que se enciendan todos los leds del panel2 si el interruptor rojo está inactivo y el azul activo. Ante cualquier otra combinación, los leds deberán permanecer apagados.
- d) Modifique nuevamente el programa anterior para que se activen los leds si, además, el interruptor verde está desactivado (tenga en cuenta que es activo a nivel bajo).
- e) Modifique una vez más el código para que interruptor rojo solo actúe si el interruptor azul está activo. En caso de estar desactivado el interruptor azul, el interruptor rojo es ignorado
- f) Desarrolle un programa que, al activar el interruptor rojo, se enciendan solo los leds rojos del panel2 y que, al desactivarlo, se enciendan solo los leds verdes.
- g) Implemente un programa que, al activar el interruptor azul y luego desactivarlo, se enciendan las luces del panel2, pero que, al repetir el proceso sobre el interruptor azul, las luces se apaguen. Este comportamiento deberá repetirse constantemente.
- h) Escriba un código que encienda el led del pin0 del panel2. A partir de ese momento, cada vez que el interruptor rojo cambie de estado debe encenderse la luz inmediatamente a la izquierda (pin1) y apagar la anterior. Este proceso se repetirá hasta encender el led del pin 7, momento en el que pasará nuevamente al pin 0 con el siguiente desplazamiento.
- i) Realice un programa que encienda el led azul al accionar interruptor azul y lo apague al desactivar el interruptor.
- j) Modifique el programa anterior para que el led azul se encienda y el led rojo se apague al accionar el interruptor azul y viceversa.

**Ejercicio 15:** Mediante el simulador VisualRISCV cargue el siguiente fragmento de código que añade los dispositivos *board*, *arrows* y *matrix* para realizar los siguientes apartados:

```
.config
    pipelined off
    adddev board "Panel"      0x600000 {0,0,0,0,0,RS,GS,BS}
    adddev arrows "Cursores" 0x700000 {11LRUD11}
    adddev matrix "Pantalla" 0x800000 rows=32 columns=32 latency=(0,0) pattern={0C?S?B1U}
.text
    li x6,0x600000
    li x7,0x700000
    li x8,0x800000
```

- a) Realice un programa que dibuje una línea diagonal de color rojo en la pantalla sin tener en cuenta el bit de ocupado (Configurado con latencia 0).
- b) Cambie la latencia en la directiva del dispositivo *matrix* a *latency=(80,100)* y modifique el código para que realice la espera activa del bit *busy* antes de enviar un comando.
- c) Modifique nuevamente el código para que dibuje constantemente la diagonal según el estado en el que se encuentren los interruptores rojo, verde y azul, teniendo en cuenta que cada interruptor corresponde a una componente de color (si un interruptor está activo establecerá la componente de color asociada al interruptor en 255, en caso contrario escribirá un 0 en dicha componente).
- d) Basándose en el código del apartado anterior, realice un programa que vaya rellenando la pantalla por filas desde abajo hacia arriba de un color cualesquiera.
- e) Realice un programa que dibuje en la pantalla un pixel y que se desplace según se pulsan los cursores. No es necesario tener en cuenta los casos en los que el cursor se sale de la pantalla.