La Máquina Sencilla y su Sistema de Entrada Salida en la placa Nexys3

Barijhoff H , Gil S ,Giordano F, Sicardi S, Bahoz Torrico M, Ishikame E,
Guzmán Lamperti M, Luján E, Cingolani L, Kantor S, Romano S, Temmyk D, Vega L

Materia: Diseño de Sistemas con FPGA. Profesora P. Borensztejn
Departamento de Computación, FCEyN, UBA
autor de contacto: mirko.torrico@gmail.com

1 de agosto de 2016

Resumen

La Máquina Sencilla fue diseñada en 1988 por profesores de la Facultad de Informática de Barcelona con el objetivo de contar con una herramienta para la enseñanza de una de las materias introductorias de la carrera de Informática.

En este trabajo se ha realizado una implementación de la Máquina Sencilla en una placa nexys3 de Digilent. Adicionalmente, se extendió su juego de instrucciones original para incluir el soporte de subrutinas y de instrucciones de entrada/salida. Se definió una interface común para los controladores de dispositivos permitiendo la extensión de las funcionalidades del procesador en el hardware. Se implementaron varios dispositivos estandard. Toda la implementacion se acompaña con las siguientes herramientas de software que fueron desarrolladas ad-hoc: ensamblador, desensamblador y simulador de código.

Índice

1. Introducción

La Máquina Sencilla fue diseñada en 1988 por profesores de la Facultad de Informática de Barcelona[1] con el objetivo de contar con una herramienta para la enseñanza de una de las materias introductorias de la carrera de Informática. Su repertorio de instrucciones consta de sólo 4 instrucciones (ADD, MOV, CMP y BEQ) y sus operandos se acceden

[1] Miguel Valero, Eduard Ayguadé: La Máquina Sencilla: Introducción a la Estructura Básica de un Computador. 1988. Centre Publicacions D Abast. Barcelona. España

indicando su dirección de memoria en la instrucción. La instrucción CMP modifica el único flag de estado de la arquitectura: flag de cero (Z). Los datos, las instrucciones y la ALU son de 16 bits. Dado que las instrucciones son de 16 bits, y como máximo deben especificar dos operandos en memoria, las direcciones de los operandos son de 7 bits permitiendo acceder a un máximo de 128 palabras de memoria. Para ser consistentes con la arquitectura llamada Von Neumann, los datos y sus instrucciones residen en la única memoria de 128 palabras. La Máquina Sencilla no tiene un banco de registros visible al programador: todas las operaciones son memoria/memoria. La Máquina Sencilla se convirtió en un excelente diseño para implementarse en FPGA. En particular, la existencia de LUTs RAM hacen posible integrar la Memoria dentro de la UP sirviendo así de doble función: Banco de Registros y Memoria de Instrucciones.

2. Implementación de la Máquina Sencilla con sus extensiones

La implemetación de la Máquina Sencilla se realizó siguiendo exactamente el diseño de sus autores, por lo tanto se separó en dos módulos verilog: la UP (Unidad de Proceso) y UC (Unidad de Control).

2.1. Memoria

La UP contiene (instancia) el módulo Memoria, donde se localizan datos e instrucciones. La Memoria está implementada como Memoria Distribuida (LUT RAM), con escritura síncrona y lectura asíncrona.

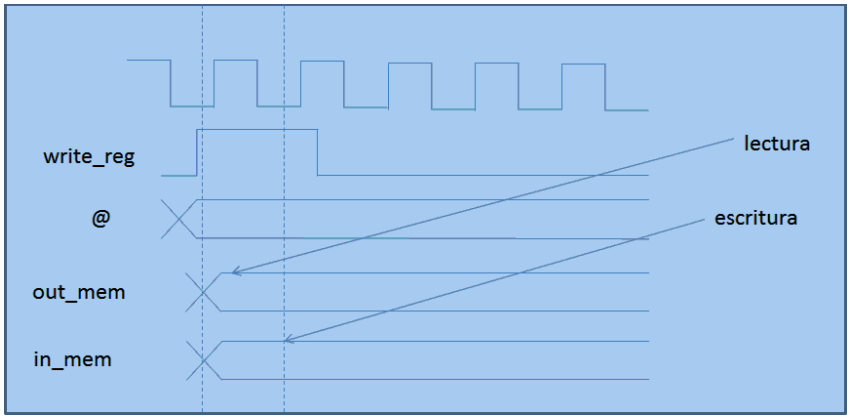


Figura 1: Diagrama de Tiempo del módulo Memoria.

```
module RAM(
    input clk, le,
    input [6:0] dir,
    input [15:0] ent,
    output [15:0] sal
    );

    parameter RAM_WIDTH = 16;
    parameter RAM_ADDR_BITS = 7;
    reg [RAM_WIDTH-1:0] M[(2**RAM_ADDR_BITS)-1:0];

     initial
         begin
              $readmemb("shifter.hex",M);
         end

    always @(posedge clk)
       if (le)
          M[dir] <= ent;
     assign sal = M[dir];

 endmodule
}
```

2.2. UP

La UP implementa el Camino de Datos de la Máquina Sencilla. Las instrucciones de la arquitectura básica son memoria-memoria.

```
Instrucciones

ADD F, D ; (D) <= (F) + (D)
MOV F, D ; (D) <= (F)
CMP F, D ; FZ <=(F) xor (D)
BEQ D ; si (FZ) PC <= D
```

El formato y tamaño de instrucción es el que sigue:

```
Tamaño: 16 bits
Código de operación: 2 bits
Campo para especificar dirección de memoria: 7 bits
```

La UP está formada por cinco registros no visibles al programador, la ALU, y un multiplexor para selección de dirección de instrucción.

- Registros (no visibles al programador)
  - PC: dirección de instrucción (7 bits)
  - IR: instrucción (16 bits)
  - A, B: registros fuentes para ALU (16 bits)
  - FZ: flag de cero (1 bit)
- ALU (Unidad Aritmético Lógica): Soporta las tres operaciones básicas : sumar, xor, dejar pasar un dato de la entrada a la salida
- Multiplexor: Selección de la dirección de memoria

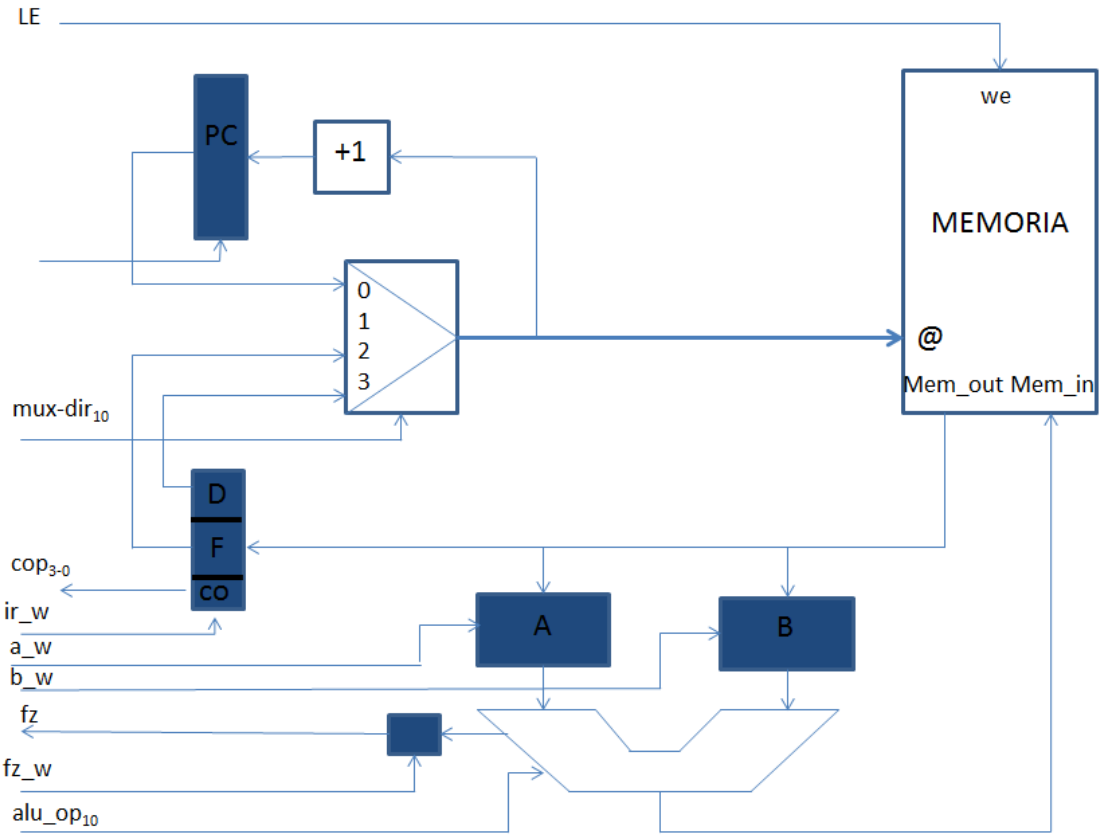


Figura 2: Máquina Sencilla

2.3. UC

La UC es la responsable de generar las señales de control para la UP. Temporiza estas señales mediante un grafo de estados que representan el ciclo de ejecución de cada una de las cuatro instrucciones. Según se muestra en la Figura 4 , las instrucciones ADD y CMP se ejecutan en 5 ciclos, la instrucción MOV en 4 ciclos y la instrucción BEQ en 2 ciclos. La UC es un típico sistema secuencial formado por un registro para guardar el estado de la ejecución, y dos bloques combinatorios: uno que genera el siguiente estado, y el otro que genera las salidas del estado actual. En ese sentido, es un sistema secuencial típico modelado con grafo de estado de Moore: las salidas solo despenden del estado actual.

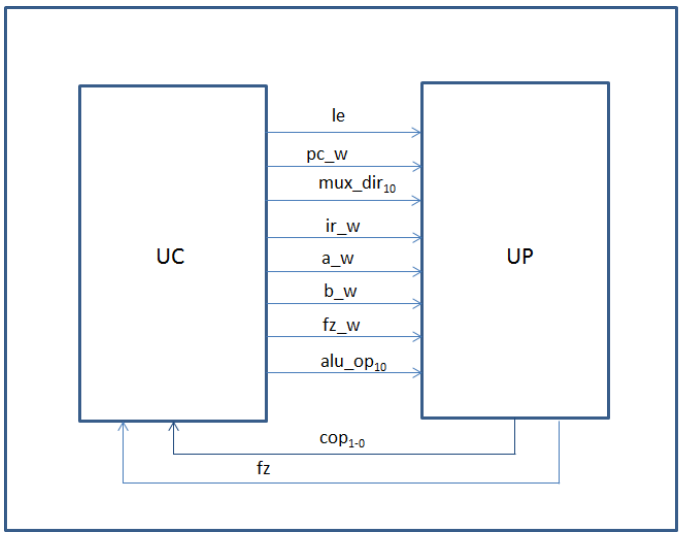


Figura 3: UC-UP

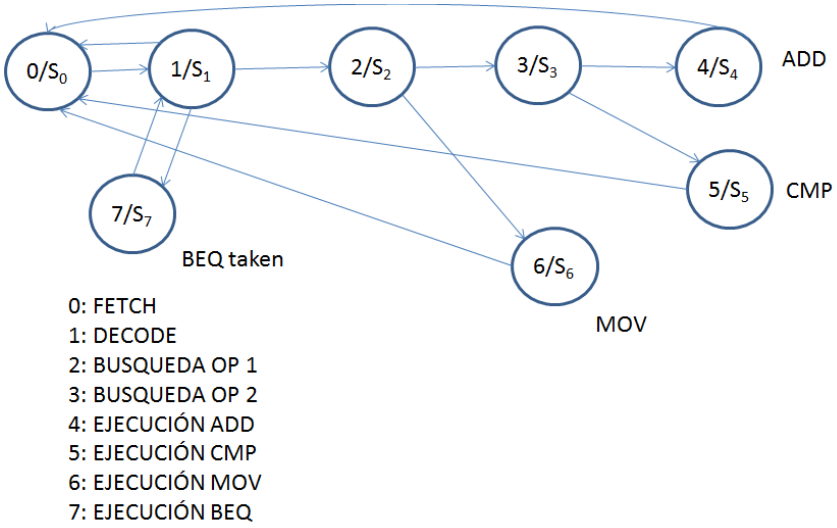


Figura 4: Grafo de Estados de las cuatro instrucciones básicas

2.4. Extesiones de la MS: instrucciones IN-OUT

Para acceder al espacio de entrada y salida se agregan las instrucciones :

```
IN dirport, D  ;;; (D) <= inport
OUT dirport, D ;;; outport<= (D)
```

El código de operación se extiende a 4 bits, y la dirección del puerto de entrada/salida ocupa los 5 bits siguientes al código de operación, permitiendo especificar 32 destinos externos. El dato que se lee o se escribe es de 16 bits y se especifica mediante su dirección de memoria, como en el resto de las instrucciones. Los cambios introducidos en la UP son: un multiplexor para la entrada de datos de memoria (salida de la ALU o inport), y exportar hacia afuera del módulo las señales que salen de la memoria (outport) y los cinco bits del registro IR que identifican el puerto de entrada-salida. Los cambios introducidos en la UC son: grafo de estado para las dos nuevas instrucciones, las señales correpondientes para el control del UP y la nueva señal WE.

La Máquina Sencilla exporta ahora 5 señales: dirport, inport, outport y la señal de escritura (WE). La simulación que se muestra en la figura 5 ejecuta dos instrucciones:

```
IN 1,a
OUT 0, a
```

La dirección 1 podria estar asignada a los switches, por ejemplo, y la dirección 0 a los leds. En la próxima sección veremos toda la definición e implementación del sistema de entrada salida.

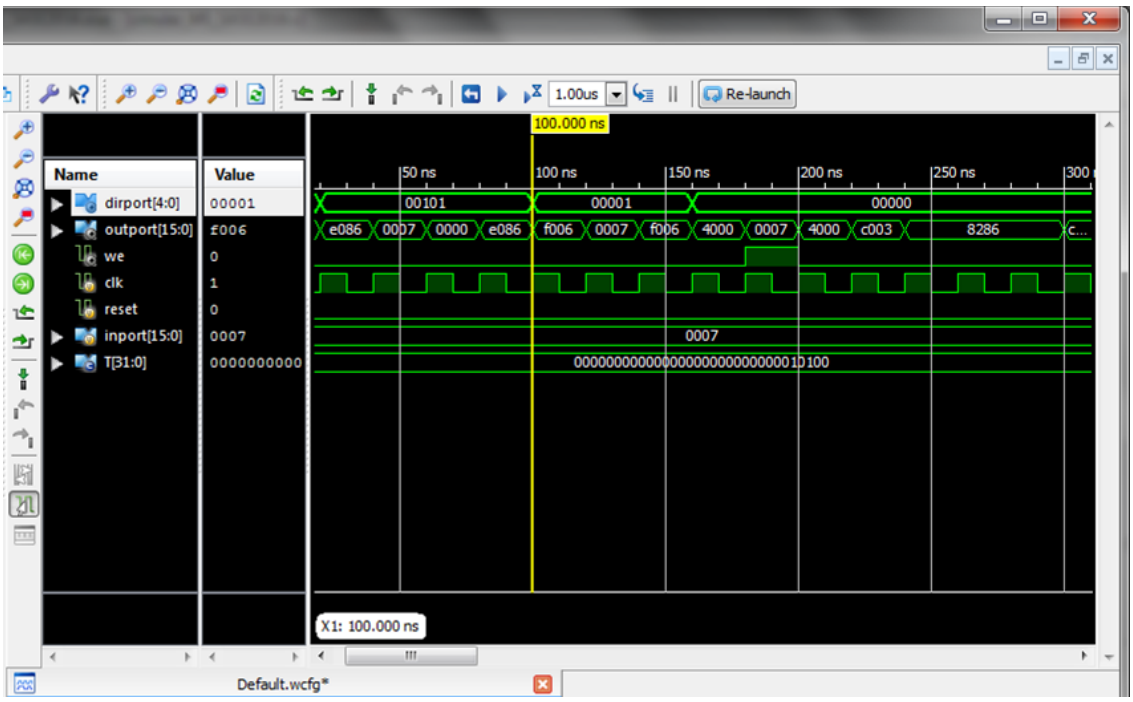


Figura 5: Simulacion E/S

2.5. Extesiones de la MS: instrucciones CALL-RET

Finalmente, se incluyeron las instrucciones de llamada y retorno de subrutina. Para evitar usar recursos adicionales, la Memoria se utiliza como pila de direcciones de retorno, desde las posiciones mas altas. Los recursos adicionales en la UP han sido: un registro SP para implementar las operaciones de PUSH y POP de la pila de direcciones de retorno.Un incrementador y decrementador para actualizar el valor del SP despues de las correspondientes operaciones, y una entrada mas en el multiplexor de entrada a la memoria. Las modificaciones en la UC son: la inclusión de los grafos de estado para las nuevas instrucciones, y las nuevas señales requeridas para la UP , carga del SP, decremento o incremento del SP, y selección del multiplexor a memoria.

La Máquina Sencilla, con estas modificaciones queda como se muestra en la Figura 6. También se muestra la utilización de recursos en la spartan6, en la Figura 7

3. Definición e Implementación de una interface común para los controladores de entrada/salida

Objetivo:

- Definir una interface común para todos los dispositivos de entrada/salida
- Implementar el decodificador de direcciones y ruteo de datos entre el procesador y los controladores
- Utilizar la misma interface tanto para controladores de entrada/salida como para módulos en verilog

3.1. LógicaIO

Es el módulo responsable de implementar el diálogo entre el procesador y los controladores de E/S. Soporta 8 controladores de E/S. Cada uno de ellos tiene cuatro registros.

Su interface con el procesador es la siguiente:

- Entradas del procesador
  - dev_sel: dirport[4:2] identifica al dispositivo permitiendo solo 8 posibles dispositivos conectados simultáneamente. Con estos 3 bits se activa la señal CS (chip select) del dispositivo seleccionado. Recordamos que dirport está especificada en la instrucción.
  - reg_sel: dirport[1:0] identifica uno de los 4 registros disponibles del controlador. Se exporta al controlador del dispositivo directamente.
  - data_out: dato de 16 bits . Es el dato que se envía al dispositivo seleccionado. Si el dispositivo es de 8 bits, el programa deberá hacerse cargo de la conversión.
  - we: señal de escritura. Se exporta directamente del procesador al controlador.
- Salidas al procesador

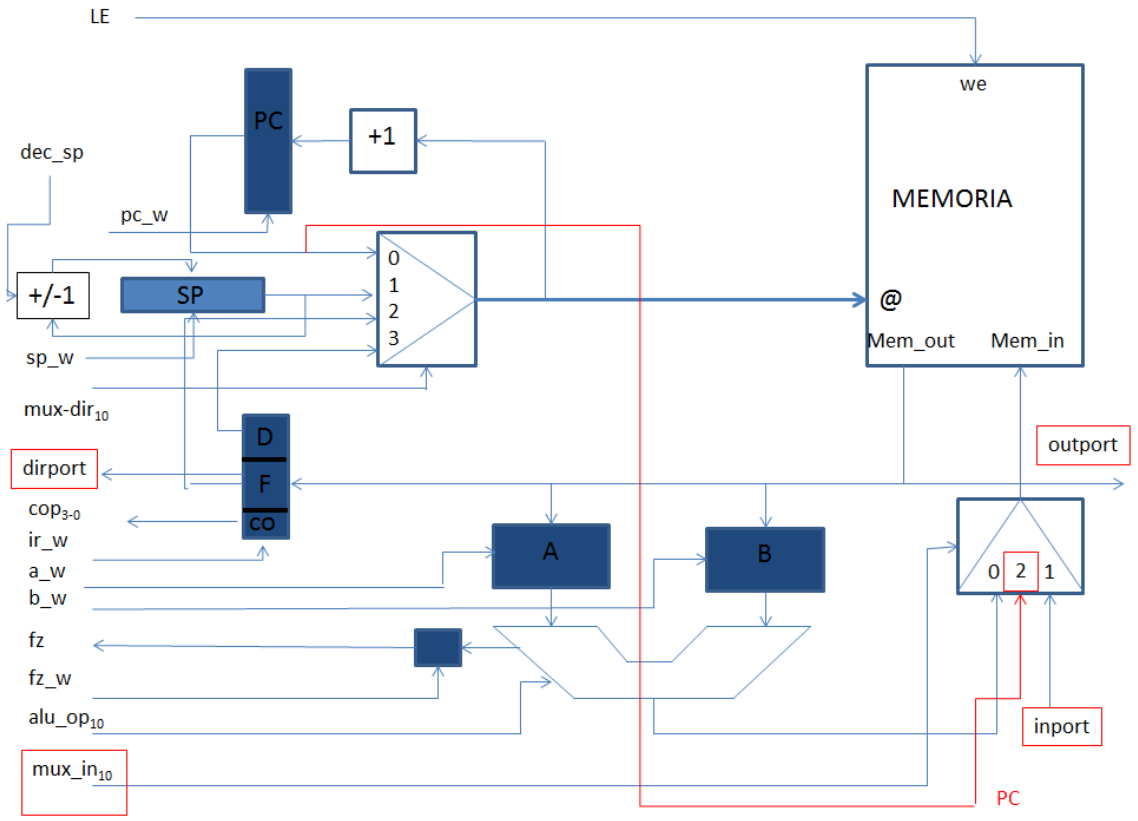


Figura 6: Máquina Sencilla: UP con extensiones

- data_in: dato de 16 bits . Es el dato que envía el dispositivo seleccionado. Si el dispositivo es de 8 bits, los bits mas altos son nulos.

Su interface con los Controladores es la siguiente:

- Salidas hacia el controlador:
  - deviceNout: donde N es un número del 0 al 7. Formado por 3 campos: bit 18= we, bit 17:16= reg_sel, bits 15:0 = datos al controlador.
  - deviceNcs: habilitación del dispositivo
- Entradas desde el controlador:
  - deviceNin [15:0] = datos del controlador

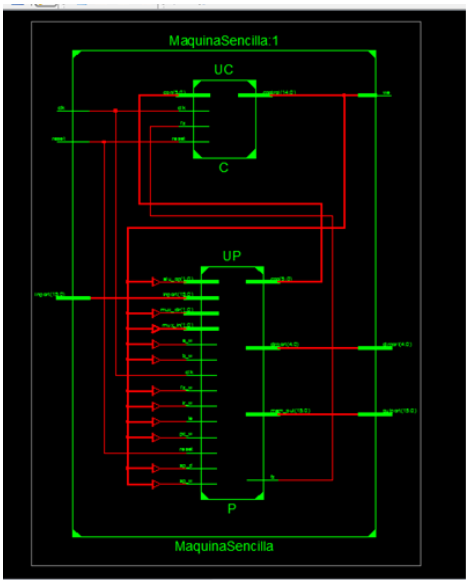
3.2. Interface común para los Controladores

Cada controlador tiene cuatro registros accesibles. Su interface con el controlador del bus (módulo LogicaIO) es la siguiente:

- Entradas
  - CS : un bit para seleccionar el controlador (1 bit)
  - Dir_reg: dos bits para identificar 4 registros del controlador (2 bits)
  - We: escritura del registro especificado
  - data_out(16 bits)
- Salidas
  - Datos: data_in (16 bits) proviene del controlador. Si el dispositivo es de 8 bits, el programa recibirá ceros en los bits mas significativos.

3.3. Placa

Todos los contraladores, así como la LogicaIO y la Máquina Sencilla se instancian en el Módulo Principal del Proyecto llamado Placa. El módulo Placa exporta todas las señales que salen de la FPGA, por ejemplo, switches, leds, etc. Todos los dispositivos básicos se acceden a través de un controlador conforme a la interface que se ha definido. Existe un mapa de direcciones que indica en que dirección del espacio de E/S está instalado cada controlador. Ese mapa es variable y



Device utilization summary:

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers:	77 out of 18224	0%
Number of Slice LUTs:	134 out of 9112	1%
Number used as Logic:	102 out of 9112	1%
Number used as Memory:	32 out of 2176	1%
Number used as RAM:	32	

Figura 7: Uilización de recursos en spartan6

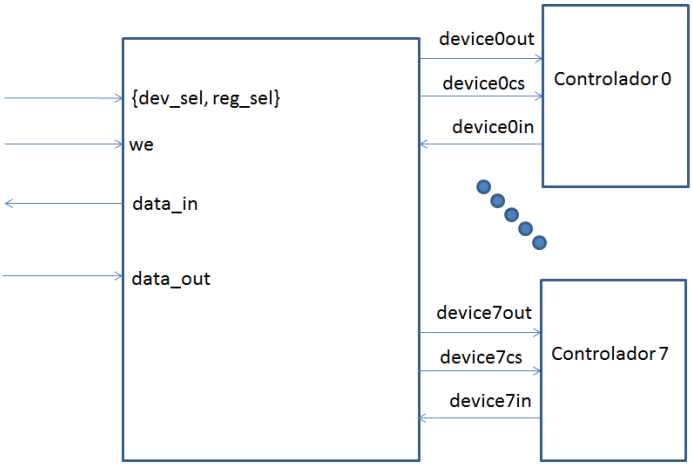


Figura 8: Logica de E/S: interfaces

lo determina el usuario modificando correspondientemente el archivo Placa que instancia al controlador. En la Figura 9 se muestra la asignación de controladores realizada en este proyecto.

La utilización de los recursos de la nexys3 con todos los controladores instanciados se muestra en la Figura 10

4. Controladores

En este trabajo se implementaron varios controladores. Podemos distinguir dos tipos de controladores: aquellos que se concectan con dispositivos o interfaces presentes en la placa nexys3, como por ejemplo los leds, switches o uart, y aquellos que amplian las funcionalidades del procesador, por ejemplo el shifter, el timer y la ALM. Dado que la Máquina Sencilla cuenta con un conjunto muy reducido de instrucciones, operaciones mas complejas pueden ser realizadas por la lógica externa con muy poco costo en tiempo. Por ejemplo, para utilizar el shifter es necesario ejecutar tres instrucciones: dos OUT y una IN con el resultado. Cada instrucción IN o OUT ejecuta en 3 ciclos, por lo tanto el resultado estará disponible en 9 ciclos. Si consideráramos implementar la instrucción dentro de la máquina, deberíamos distinguir dos casos, derecha e izquierda, con lo cual necesitamos usar dos códigos de operación. Se debería tambien ampliar los bits de control para la ALU, y finalmente la lógica del shifter. En ese caso, la instrucción shift , tardaría 5 ciclos.

La idea de implementar instrucciones complejas en hardware externo tiene que ver con agregar al sistema solo lo que se va a utilizar, al contrario de procesadores dotados con un juego inmenso de instrucciones de las cuales solo se usan el 10 % , nuestro procesador tiene solo las instrucciones imprescindibles. Las demas se obtienen del hardware externo, solo en caso de que sean necesarias. En otro caso, no se instancia el controlador correspondiente.

Dirección	Dispositivo	Reg. 0	Reg. 1	Reg. 2	Reg. 3	Observaciones
0	ALM	d0(w)	d1(w)	op(w)	out(r)	Ver Docs/ALM.md
1	Shifter	op(w)	value(w)	data_out(r)		op = {{11{x}}, 1'b(¬l/r), 4'b(#shifts)}
2	Timer	rounds(r) rounds_goal(w)	prescaler(r) prescaler_goal(w)	status(r) start(w)	done(r) reset(w)	Ver Docs/timer-calc.html para el cálculo de tiempos
3	UART	tx_data(w)	rx_data(r) reset(w)	tx_full(r)	rx_empty(r)	Opera con los 8 bits *menos* significativos
4	7 Segmentos	7seg_data(w)				
5	Botones	btnU(r)	btnD(r)	btnL(r)	btnR(r)	Expande con ceros
6	Switches	swt_data(r)				
7	LEDs	led_data(w)				

Figura 9: Mapa de los Controladores

```
Selected Device : 6slx16csg324-2


Slice Logic Utilization:
 Number of Slice Registers:          447  out of  18224    2%
 Number of Slice LUTs:               728  out of   9112    7%
 Number used as Logic:               680  out of   9112    7%
 Number used as Memory:               48  out of   2176    2%
  Number used as RAM:                 48
```

Figura 10: Utilización de nexys3 con Entrada Salida

4.1. Controladores implementados hasta el momento

Los controladores implementados hasta el momento son los que estan descriptos en la Figura 9. El dispositivo ALM es un Módulo Aritmético Lógico que soporta las operaciones descritas en la Figura 11. Este módulo consume una importante cantidad de recursos, en particular el multiplicador y divisor que estan realizados utilizando los IP cores de xilinx.

4.2. Controlador Timer

Como ejemplo de controlador presentamos el Timer con dos funcionalidades: contar tiempo transcurrido y esperar un tiempo dado. La primera funcionalidad es útil particularmente cuando el código incluye esperas activas (polling). Para programarlo, el Timer ofrece cuatro registros :

- ROUNDS: Registro 0. Especifica un número entre 0 y 65536. Se escribe un valor para esperar tiempo. En la funcionalidad contar tiempo, se puede leer.
- PRESCALER: Registro 1. Especifica un número entre 0 y 65536 Se lo programa para establecer hasta cuanto hay que contar para lograr un round. También se puede leer.
- START: Registro 2. Se debe poner un 1 para iniciar el conteo en las dos funcionalidades
- DONE: Registro 3. Se consulta en el primera modalidad para saber si ha terminado la cuenta.

Por ejemplo, para esperar un segundo, podemos programar los registros de la siguiente manera: Si la frecuencia del reloj es de 100 Mhz, un ciclo de reloj dura 10nseg. Un segundo son 10^9 nanosegundos, o sea debemos esperar 10^8 ciclos. Usando la calculadora de tiempos del timer[2] los valores mas aproximados a 1 segundo se obtienen haciendo:

- Round= 1526
- Prescaler=65530

El código para actualizar un contador en los siete segmentos cada un segundo, sería el siguiente:

```
MOV @1,a
OUT 16, a
```

[2]Santiago Gil, Documentación del Timer : timer-calc.html

Código (OP)	Descripción	Resultado (OUT)
0	AND	D0 & D1
1	OR	D0 | D1
2	NOT	~D0
3	NAND	~(D0 & D1)
4	XOR	D0 ^ D1
5	NOR	~(D0 | D1)
6	XNOR	~(D0 ^ D1)
7	BITSEL	D0[D1]
8	MERGE_H	{D1[15:8], D0[15:8]}
9	MERGE_L	{D1[7:0], D0[7:0]}
10	MERGE_HL	{D1[15:8], D0[7:0]}
11	MERGE_LH	{D1[7:0], D0[15:8]}
12	SWAP_HL	{D0[7:0], D0[15:8]}
13	MULT	D0[7:0] * D1[7:0]
14	DIV	D0 / D1
15	DIV_REM	D0 % D1

Figura 11: Módulo Aritmético Lógico

```
iterar: MOV @1526, rounds
MOV @65535, prescaler
OUT 8, rounds
OUT 9, prescaler
MOV @1, start
OUT 10, start
polling: IN 11, done
CMP @0,done
BEQ polling
INC a
OUT 16,a
JMP iterar
```

Observación: dos instrucciones JMP e INC no pertenecen al juego de instrucciones, son pseudoinstrucciones generadas por el ensamblador, del que hablaremos en la próxima sección. También podemos ajustar los cálculos de tiempo considerando el tiempo de ejecución de cada instrucción.

4.3. Instrucciones para agregar un Controlador

- Se deberá crear el controlador con la siguiente interface:

```
Controlador_Nuevo (
input clk, reset, we, cs,
     input [1:0] reg_sel,
     input [15:0] in,
     output [15:0] out )
```

- Se deberá asignar una dirección al controlador (si fuere necesario , se reutilizará una dirección de alguno de los controladores por defecto no utilizados)
- Se deberá incluir la instanciación del controlador en el archivo placa.v
- Los accesos desde el programa asm deben realizarse utilizando el número de dispositivo asignado.
- En caso que el dispositivo se comunique con la placa , (por ejemplo, VGA port, memoria externa, ethernet, etc) se deberá incluir en su interface
- Las entradas y salidas de la interface del módulo placa deberán incluirse en el UCF del proyecto.
- IMPORTANTE: El controlador deberá registrar sus entradas y/o salidas.

5. Ensamblador y otras herramientas

En esta sección hablaremos del ensamblador de la MS. Su desarrollo fué una herramienta indispensable en este proyecto pues la introducción manual de programas en binario constituia una fuente de errores imposibles de detectar. Una guía completa del ensamblador se encuentra en [3]

Sus características principales son:

[3] Bahoz Torrico M., Ishikame E.,: Tokuwaga: Un Ensamblador para la Máquina Sencilla

- Soporta las 8 instrucciones básicas: ADD, CMP, BEQ, MOV, IN, OUT, CALL, RET
- Soporta pseudoinstrucciones: JMP (salto incondicional), DEC, INC, LEA (Load Efective Address), SUB (resta)
- Soporta directivas de almacenaje de datos DW
- Soporta constantes
- Soporta nombres para los controladores básicos: LEDS, SWITCHES, etc
- Informa tamaño memoria ocupada
- Tiene un desensamblador para verificación de código
- Tiene un simulador

5.1. Constantes y Etiquetas

La MS no contiene modo de direccionamiento inmediato, es decir, no soporta constantes.Esto obligaba, antes de disponer de un ensamblador, a tener que almacenar las constantes en posiciones de memoria, y referenciarlas con esa dirección absoluta.Lo mismo sucedía con las etiquetas, había que llevar la cuenta de las direcciones.Este trabajo manual generaba muchísimos errores. El ensamblador reserva automáticamente lugar para las constantes y las etiquetas.

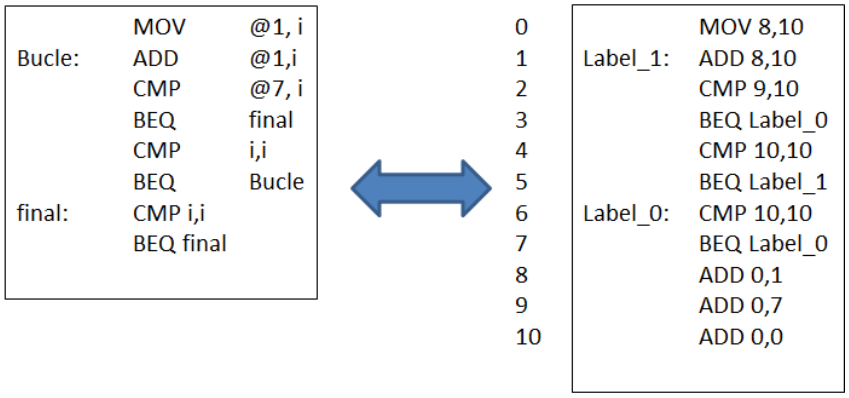


Figura 12: Traducción automática de constantes y etiquetas

En la Figura 12 se ha utilizado la herramienta desensamblador para observar el código generado. La variable i ha sido ubicada en la dirección 10, la constante 1 en la dirección 8, la constante 7 en la dirección 9.Dado que el código de instrucción de ADD es 00, el desensamblador muestra ADD en las direcciones 8,9 y 10.

5.2. Pseudoinstrucciones

El ensamblador soporta un conjunto de pseudoinstrucciones. Estas son: JMP (salto incondicional), INC, DEC, SUB, SHIFTR, SHIFTL, LEA y MOV con indirección. Algunas de estas pseudoinstrucciones son verdaderos algoritmos, como la instrucción SUB. Otras, son implementadas invocando un controlador de E/S, como las pseudoinstrucciónes SHIFTR y SHIFTL como se muestra en la Figura 13.

La pseudoinstrucción LEA (Load Efective Address) en conjunto con las instrucciones MOV con indirección permiten trabajar con punteros.

5.3. Desensamblador y Simulador

Para testear el código generado por el compilador, se implementó un reensamblador, el cual genera instrucciones (si las existen) a partir del cóodigo binario. Dado que las pseudo instrucciones no pueden ser reensambladas y las variables y constantes no son visibles en el código original, el código obtenido es, en general, de mayor tamaño. Además, como algunas pseudoinstrucciones crean la instrucción en el momento de ejecución, se implementó un simulador de la máquina sencilla para poder realizar el seguimiento sobre dichas instrucciones.

6. Un ejemplo completo: Resolución de una ecuación por el método de las Diferencias Finitas

El programa resuelve de forma numérica, por el método de las diferencias finitas, el perfil de temperaturas de una barra de acero, en estado estacionario, mediante la ecuación unidimensional de difusión de temperatura, bajo apropiadas condiciones de borde. El planteo matemático que describe este problema es el siguiente: Sea $u : \mathbb{R} \rightarrow \mathbb{R}$ el campo escalar de temperaturas [$^{\circ}C$]. x, $x_0 \leq x \leq x_n$, la variable independiente que representa la dimensión espacial

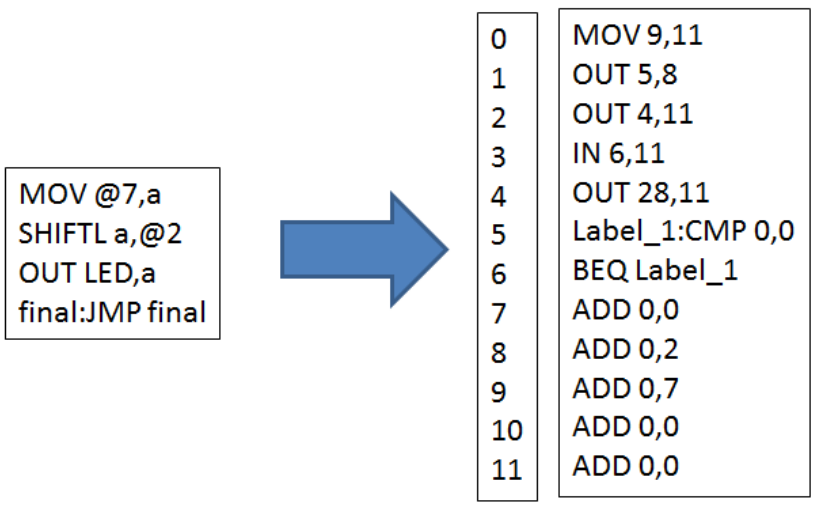


Figura 13: Traducción de instrucción SHIFT

[cm]. D la constante que representa la conductividad térmica del material. Se resuelve $\frac{\partial}{\partial x}\left(D\frac{\partial u}{\partial x}\right) = 0$. Con condiciones de borde $u(x = x_0) = u_0$ y $u(x = x_n) = u_n$. $u_0 = 100, u_n = 0, x_0 = 0, x_n = 150$.

El código ensamblador asociado envía los datos generados a través de la UART de la máquina sencilla.

Debido a que este algoritmo necesita de números reales para funcionar, y que la máquina sencilla implementada no tiene este tipo de representación se optó por lo siguiente: $u'(x) = u(x) * 327{,}67$. En el caso de la condición de Diriclet del extremo izquierdo, u'_0=32767. De modo que una vez calculados los u'(x), usando la relacción anterior, se pueden obtener los valores deseados. Si bien esta alternativa pierde precisión es suficientemente buena para resolver el problema que se plantea.

La solución a esta ecuación simple se realiza de forma numérica, por el método de las diferencias finitas.

El esquema numérico generado es el siguiente: $r\ u_{i+1} - 2\ r\ u_i + r\ u_{i-1} = 0$ donde r = $\frac{D}{2\ dx}$

El peril de temperaturas generados es:

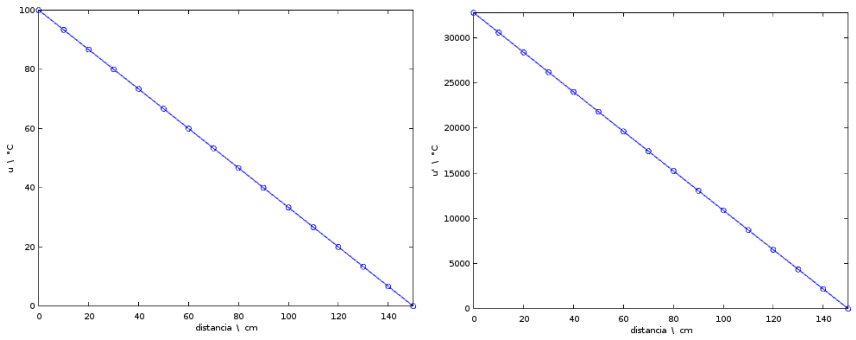


Figura 14: Perfil de Temperaturas

Por problemas de espacio, la solución completa incluyendo el programa ensamblador y la solución obtenida se adjunta en el documento [4]

7. Proyección Futura y Conclusiones

En este trabajo se han cumplido dos objetivos. Por un lado, ha constituido una herramienta de aprendizaje para el Diseño de Sistemas con FPGA dado que se ha trabajado diseñando sistemas secuenciales y combinacionales, poniendo especial cuidado en minimizar el número de registros de forma de aprovechar al máximo los recursos de forma paralela. Por otro lado, consideramos que hemos logrado un diseño sencillo y abierto, de forma que es posible seguir trabajando en él aprovechando sus características didácticas, de cara a futuros estudiantes.

Respecto a trabajos futuros, indicamos algunas extensiones o mejoras:

- La Máquina Sencilla tiene muy poca memoria direccionable, de forma que se podría encarar alguna manera de aumentar su espacio de direcciones, ya sea incluyendo mas modos de direccionamiento, o bien creando una arquitectura Harvard con dos espacios diferentes para instrucciones o datos, o , en fin, cualquier otra idea. Dado que no estamos restringidos por nada físico y sobran recursos, cualquier idea es implementable

[4] Luján E, Guzmán Lamperti M : Resolución de ecuación 1d de calor por diferencias finitas.

- La Máquina Sencilla tiene un problema con la carga de programas en memoria: cada vez que se carga un programa es necesario sintetizar e implementar todo el proyecto. Esto tambien puede encararse tomando la idea del PicoBlaze que permite cargar en BRAM los programas sin modificar el resto de los módulos.
- Implementación de Interrupciones en la MS
- Implementación de más controladores: Memoria BRAM, Memoria Externa, VGA, etc
- etc, etc

Por último queremos resaltar la experiencia que tuvimos al trabajar varios grupos de alumnos en un proyecto común , donde fué necesario sincronizar los cambios en un repositorio común y donde era necesaria la interacción permanente para ponerse de acuerdo en las necesidades y definiciones.

Todo el material desarrollado incluyendo manuales de uso y documentación mas detallada se encuentra en :

`https://github.com/marcegl91/sase_project`