**DISEÑO LÓGICO**

Profesores: Martín Vázquez y Lucas Leiva

**TRABAJO PRÁCTICO: PROCESADOR**

**UNTREF**

2° Cuatrimestre 2018

Integrantes: Maximiliano Andreoli y Franco De Alesandro

**Introducción:**

El objetivo de este trabajo es armar un procesador digital con una variedad de componentes diferentes aprendidos durante la cursada. Para esto utilizamos el lenguaje de programación VHDL y el IDE ModelSim proporcionado por los profesores.

Comenzamos haciendo la Unidad Aritmética Lógica (ALU) pedida en la primer práctica, cumpliendo con las operaciones demandadas, para luego implementar dentro del Procesador. Una vez terminado de realizar todos los componentes y conectarlos entre sí, se realizó un Test Bench para probar el correcto funcionamiento.

Durante este informe detallaremos características de los componentes y sus comportamientos. También concluiremos con las dificultades a la hora de la realización del trabajo práctico.

**Desarrollo:**

A continuación, se detallan las funcionalidades de los componentes utilizados para el Procesador:

\* Todos los componentes están en un nivel de abstracción RTL (Register Transfer Level).

* **PC:** contador que consta de una salida de 7 bits, clock y reset. Devuelve un valor que será la dirección de la instrucción a utilizar. Estilo descriptivo: algorítmico.
* **ROM\_prog:** memoria de sólo lectura con una entrada *address* de 7 bits y una salida de 16 bits. Almacena todas las posibles instrucciones y devuelve la que se encuentre en la posición de la entrada. Estilo descriptivo: flujo de datos.
* **IR:** registro de instrucción con entrada y salida de 16 bits. Almacena la instrucción próxima a ejecutar. Estilo descriptivo: algorítmico.
* **Decode:** decodificador con una entrada de 8 bits y salidas *we* (habilitación de escritura) para el Reg\_out, Reg\_A y Regs, una señal *bus select* para el MUX de 2 bits y una señal *operación* de 3 bits a utilizar por la ALU. Determina las señales de control que deben ejecutarse para que se lleve a cabo la operación de la instrucción. Estilo descriptivo: flujo de datos.
* **MUX:** multiplexor con 3 entradas y una salida de 8 bits, y señal de 2 bits. La señal determina cuál entrada es la salida. Estilo descriptivo: flujo de datos. Aclaración: en el programa, es una señal dentro del Procesador.
* **Regs:** banco de registros con entradas de lectura y escritura de 4 bits, una señal *we*, clock, reset y entrada y salida de 8 bits. Almacena las operaciones realizadas en cada ciclo de clock. Estilo descriptivo: algorítmico.
* **Reg\_A:** registro acumulador con entrada y salida de 8 bits, señal *we*, clock y reset. Almacena al siguiente valor B de la ALU. Estilo descriptivo: algorítmico.
* **ALU:** unidad aritmética lógica con operandos como entradas de 8 bits, *operación* de 3 bits y salida de 8 bits. Resuelve una serie de operaciones que son ejecutadas según la señal a la cuál cada una de las mismas corresponde. Puede realizar la devolución de la entrada A, puede desplazar a la derecha y a la izquierda, sumar, restar y realizar AND, OR y XOR. Estilo descriptivo: flujo de datos.
* **Reg\_out:** registro de salida final con entrada y salida de 8 bits, señal *we*, clock y reset. Almacena el valor que será la salida del procesador. Estilo descriptivo: algorítmico.
* **Procesador:** es el componente que contiene a todos los demás mencionados. Tiene una entrada y una salida de 8 bits, clock y reset. Une todos los componentes con cables (señales). Estilo descriptivo: estructural.

El *ejercicio 1* de la práctica 2 consiste en completar el siguiente cuadro, el cuál es la combinación de salidas del decodificador según la entrada que ingrese en él:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instrucción** | **bus\_sel** | **alu\_op** | **reg\_a\_we** | **out\_we** | **reg\_we** |
| IN (0 x 01) | 10 | 000 | 0 | 0 | 1 |
| OUT (0 x 02) | 00 | 000 | 0 | 1 | 0 |
| MOV (0 x 03) | 00 | 000 | 0 | 0 | 1 |
| LDA (0 x 04) | 00 | 000 | 1 | 0 | 0 |
| LDI (0 x 05) | 01 | 000 | 1 | 0 | 0 |
| ADD (0 x 10) | 00 | 010 | 0 | 0 | 1 |
| SUB (0 x 11) | 00 | 011 | 0 | 0 | 1 |
| AND (0 x 12) | 00 | 100 | 0 | 0 | 1 |
| OR (0 x 13) | 00 | 101 | 0 | 0 | 1 |
| XOR (0 x 14) | 00 | 110 | 0 | 0 | 1 |
| SHL (0 x 20) | 00 | 001 | 0 | 0 | 1 |
| SHR (0 x 21) | 00 | 111 | 0 | 0 | 1 |

El Test Bench realizado para la ALU no posee entity, y sólo agregamos el componente a probar, sin clock ni reset, ya que hicimos un proceso que aumenta el valor de la señal de operación por cada delay de nano-segundos.

En el caso del Test Bench del Procesador, tampoco utilizamos entity pero sí agregamos el clock y el reset, los cuales son utilizados cada uno con un proceso por separado.

Ambos Tests los hicimos con comprobaciones assert, probando cada salida posible del componente utilizado.

**Conclusión:**

Después de finalizar el trabajo, podemos decir que nos llevó más tiempo de lo que nosotros mismos pensabamos que iba a ser. Principalmente esto se debe a que es un lenguaje que no conocíamos y que el IDE (ModelSim) en un principio no fue muy cómodo de utilizar, ya que primeramente no muestra bien los errores (al menos no como estamos acostumbrados). Podemos decir que la finalización del trabajo fue “muy sobre la hora” porque ambos integrantes tuvimos que ir al recuperatorio de la cursada y hasta el momento de aprobarlo no habíamos practicado ni realizado ninguna tarea con respecto al mismo.

La metodología que utilizamos para hacerlo fue conectándonos mediante *Skype*, no quisimos hacer tareas por separado ya que esto podría crear conflicto en la diferente realización de lógica o código a lo largo del proyecto. A todo esto ambos tenemos nuestros respectivos horarios de trabajo y obligaciones, los cuales afectaron los momentos en los que nos podíamos poner al mismo tiempo.

Pero mas allá de todo lo dicho, creemos que el tiempo fue acorde, sólo que no teníamos mejor forma de distribuirlo.

Creemos que esta práctica nos ayuda muchísimo a comprender más los temas vistos, siendo que de esta forma, al poder plasmarlo en código, se nos familiariza más y es más fácil entender el por qué de los procedimientos de cada componente.

Todo el trabajo práctico también está subido a GitHub, por lo cual les compartimos el link: <https://github.com/francodealesandro/DisenoLogicoProce>