Práctica FPGA 2: Diseño de un intérprete de comandos en VHDL

Hipólito Guzmán Miranda
Profesor Contratado Doctor
Departamento de Ingeniería Electrónica
hguzman@us.es

Objetivo de la práctica: Realizar un diseño digital de complejidad media.
Evaluación de la práctica: Se debe escribir una breve memoria (puede ser una continuación del documento de la práctica anterior) con los resultados obtenidos. También se deben entregar los códigos VHDL del diseño y testbenches.

Se desea diseñar un circuito digital capaz de procesar comandos. Cada uno de estos comandos puede venir acompañado, adicionalmente, de uno o varios datos.

Funcionamiento:

El circuito recibirá sus comandos a través de una FIFO. Mientras la FIFO no esté vacía, el circuito leerá valores de la FIFO y los procesará. Ya que tanto los comandos como sus datos asociados vienen por el mismo canal (la FIFO), se distingue entre dato y comando de la siguiente manera: siempre que no se esté esperando un dato, se interpretará que el valor leído de la FIFO es un comando. Los datos se deben interpretar como enteros sin signo.

El circuito tiene dos registros internos de 8 bits de anchura, reg0 y reg1.

Para simulación, debe generarse una FIFO de 8 bits de anchura con Xilinx CORE Generator (la profundidad de la FIFO es irrelevante, aunque se recomienda que tenga al menos 32 posiciones).

Puertos:

Entradas:
`clk` : Señal de reloj. Activo por flanco de subida.
`rst` : Reset asíncrono, activo a nivel alto.
`data_in` : 8 bits. Datos o comandos de entrada que vienen de la FIFO.
`empty` : Activa a nivel alto. Indica que la FIFO desde la que se leen los datos está vacía.

Salidas:

`rd_en` : Activa a nivel alto. Indica a la FIFO que debe proporcionar un nuevo dato en data_in [1]

`data_out` : 16 bits. Datos de salida.
`valid` : Activa a nivel alto. Indica que data_out es válida
`err` : Activa a nivel alto. Indica que se ha producido un error. Cuando err se activa, el circuito deja de procesar comandos. Err sólo puede desactivarse activando la entrada de reset asíncrono rst.

Comandos:

Los comandos recibidos se interpretan de la siguiente manera:

Bits 7 a 6: origen del operando 1:

1. "00" : reg0
2. "01" : reg1
3. "10" : dato que debe ser leído de la FIFO
4. "11" : ninguno

Bits 5 a 4: origen del operando 0:

5. "00" : reg0
6. "01" : reg1
7. "10" : dato que debe ser leído de la FIFO
8. "11" : ninguno

Bits 3 a 0: código del comando:
`Send_to_reg_0` : código 0x0. Almacena el operando 0 en el registro 0.
`Send_to_reg_1` : código 0x1. Almacena el operando 0 en el registro 1.
`Inc_0` : código 0x2. Incrementa en uno el valor almacenado en el registro 0.
`Inc_1` : código 0x3. Incrementa en uno el valor almacenado en el registro 1.
`Sum` : código 0x4. Saca por `data_out` la suma de los operandos 0 y 1.
`Mult` : código 0x5. Saca por `data_out` el resultado de multiplicar los operandos 0 y 1.
`Median` : código 0x6. Saca por `data_out` el valor medio entre los operandos 0 y 1.
`Sqroot` : código 0x7. Saca por data_out la raíz cuadrada[2] del operando 0.
`Halt` : código 0xE. Activa la salida `err`.
`Flush` : Código 0xF. Lee todos los valores almacenados en la FIFO, sin interpretarlos, hasta que la FIFO quede vacía.

Los comandos `send_to_reg_0`, `send_to_reg_1` y `sqroot` deben activar la salida `err` si como origen del operando 0 reciben "ninguno" y/o si como origen del operando 1 reciben un valor distinto de "ninguno".

---

[1] Debido al funcionamiento de la FIFO, entre la activación de `rd_en` y el cambio en `data_in` existe una latencia de 1 ciclo de `clk`, salvo que se seleccione la opción "First Word Fall Through" al crear la FIFO con Core Generator.
[2] Se recomienda calcular la raíz cuadrada por el método de aproximaciones sucesivas.

Los comandos `inc_0` e `inc_1` deben activar la salida `err` si como origen de cualquiera de sus operandos reciben un valor distinto de “ninguno”.

Los comandos `halt` y `flush` ignoran los campos que definen el origen de los operandos 0 y 1.

Si se recibe un comando desconocido se debe activar la salida `err`.

Arquitectura:

La arquitectura de este diseño se debe plantear centrada en la lectura e interpretación de los comandos. Una posible manera de hacerlo es esperando a que la FIFO deje de estar vacía, en ese momento activar `rd_en` y en el siguiente estado interpretar el comando recibido:

```
case state is
      when idle =>
        if (empty = '0') then
          rd_en <= '1';
          p_state <= parse_cmd;
        end if;

      when parse_cmd =>
        case (data_in(3 downto 0)) is
          when SEND_TO_REG_0 => state <= handle_send_to_reg_0;
          when SEND_TO_REG_1 => state <= handle_send_to_reg_1;
          [...]
          when others => state <= handle_error;
        end case;
```

Para que el código resulte más legible, se recomienda el uso del keyword `constant`:

```
constant CMD_SEND_TO_REG_0  : std_logic_vector(3 downto 0) := "0000";
constant CMD_SEND_TO_REG_1  : std_logic_vector(3 downto 0) := "0001";
[...]
constant OP_REG_0  : std_logic_vector(1 downto 0) := "00";
constant OP_REG_1  : std_logic_vector(1 downto 0) := "01";
[...]
```

Desarrollo:

Se debe desarrollar el VHDL del circuito propuesto. En la memoria, se debe describir la funcionalidad de la entidad, realizando las explicaciones que sean necesarias sobre cómo funciona el código que la describe. También se debe demostrar que la funcionalidad es correcta, utilizando uno o varios testbenches e incluyendo las capturas relevantes en la memoria. Finalmente, se deben realizar consideraciones sobre la posibilidad de implementar el diseño utilizando una arquitectura pipeline.