Laboratorio 1: Implementación de una ROM

Allison Lisby, Harry Lisby

***Resumen*—En el presente laboratorio se implementará una ROM [Read-Only Memory] al sistema controlador implementado anteriormente, esto requerirá de ciertas modificaciones tanto en la instancia de la ALU, como en las señales y máquina de estados anteriores.**

I. Descripción del sistema

II. Listados de programa

En el siguiente listado se presentará el código de descripción de hardware implementado para este laboratorio.

Se implementa casi la misma descripción de entidad y arquitectura de la ALU del laboratorio anterior. Lo único que cambia es que las entradas A y B, y la salida R ahora serán de 8 bits.

Adicionalmente fué creada una ROM para almacenar el programa a ser ejecutado durante la presente evaluación.

Con lo anterior también fué necesario crear una instancia de la ROM en el controlador con el objetivo de poderlo utilizar en el proceso correspondiente.

Primeramente se crea la entidad correspondiente a la ROM, en este caso solo es necesario una entrada y una salida, ambos vectores.

```
ENTITY ROM IS
  PORT(addr: in std_logic_vector(3 downto 0);

       data: out std_logic_vector(11 downto 0)
      );
END ENTITY ROM;
```

Por otra parte se crea la arquitectura, la cual contiene un condicional *CASE* que dependiendo del valor entrante *addr*, escribirá un valor asociado a tal dirección en *data*:

```
ARCHITECTURE ROM_ARCH OF ROM IS
BEGIN
 PROCESS(addr)
 BEGIN
  CASE(addr) IS
   WHEN "0000" => data <= "100100001111";
   WHEN "0001" => data <= "110000001100";
   WHEN "0010" => data <= "010111110000";
   WHEN "0011" => data <= "011000000011";
   WHEN "0100" => data <= "010011111111";
   WHEN "0101" => data <= "110001100100";
   WHEN "0110" => data <= "110111001000";
   WHEN "0111" => data <= "110001100100";
   WHEN "1000" => data <= "010100001100";
   ...
```

El proceso de instanciar la ROM se efectuó de la siguiente manera:

```
XROM: ENTITY work.ROM PORT MAP
 (addr => addrReg,
  data => dataReg
 );
```

El proceso correspondiente al nuevo sistema, el cual incluye la nueva ROM y la pone a trabajar en conjunto con la ALU implementada anteriormente se muestra en el siguiente código.

Esta es una máquina de estados que inicialmente tiene un condicional con el solo propósito de tener la posibilidad de hacer reset al sistema. Luego de este condicional nos encontramos un case que evalúa el estado actual y dependiendo de cada uno de estos se realiza un conjunto de operaciones diferentes.

El estado *progmemRead* se encarga de leer introducir el valor del contador de programa en el registro de dirección que controla la ROM, al mismo tiempo se obtiene el valor almacenado en esta dirección el cual es guardado en el *Instruction Register [IR]*. Adicionalmente se mueve el valor almacenado en el registro *CoBuffer* a la entrada de acarreo para la siguiente operación.

En el estado *moveToRegisters* se divide o trunca IR en dos registros diferentes, uno de ellos es S que almacena la instrucción u operación que hará la ALU y otro llamado *regB* el cual contiene la información o dato a ser operado. Luego de esto se mueve el valor de S a *opIn* de la ALU y al mismo tiempo se inserta el valor almacenado en W al registro W o entrada A en la ALU. Con esto solo

queda mover el acarreo resultante de la operación a *CoBuffer* para poder ser utilizado en la siguiente operación.

Finalmente en el estado *resultToW* se lee el valor en la salida de la ALU y es guardado en el registro W para poder ser utilizado en una siguiente operación. En este mismo se incrementa en uno el valor del contador de programa *[PC* y con esto se tiene listo un ciclo de cálculo del presente controlador.

```
PROCESS(state,carryInput,RST,PC)
 BEGIN
  IF(rst='0') THEN
   PC<="0000";
   W<="00000000";
  ELSE
   CASE state IS
    WHEN progmemRead =>
      addrReg <= PC;
      IR <= dataReg;
      nState <= moveToRegisters;
      CarryInput <= CoBuffer;

    WHEN moveToRegisters =>
      S <= IR(11 downto 8);
      regB <= IR(7 downto 0);
      regW <= W;
      opIn <= S;
      nState <= resultToW;
      CoBuffer <= CoReg;

    WHEN resultToW =>
     W <= rValue;
      nState <= progmemRead;
      PC <= addrReg + '1';
   END CASE;
  END IF;
 END PROCESS;
```

III. CONCEPTOS APRENDIDOS

IV. ASPECTOS POR MEJORAR

En el presente laboratorio se encontraron diversas oportunidades de mejora, principalmente con adiciones a la implementación evaluada. Uno de estos casos es en el momento en que se termina el programa en la memoria del programa y en lugar de seguir recorriendo toda la memoria hasta que el valor sea nuevamente ”0000”, que exista un comando de reinicio.

También sería posible implementar un selector de programa, con la finalidad de tener múltiples programas disponibles y seleccionables según el uso que se le quiera dar al microcontrolador.

Podría existir un flag con el objetivo de desactivar o limpiar el carry out o in, esto para situaciones donde se cambia a otra operación totalmente diferente a la cadena o proceso anterior y no se quiere preservar un carry out resultante de tal proceso.

Otra mejora podría ser que por un pulso de reloj se corra por todo el programa y solo se muestre el resultado final, así se podría pasar a combinar diferentes programas para lograr un resultado y no necesariamente ir de estado en estado y dirección por dirección de memoria.