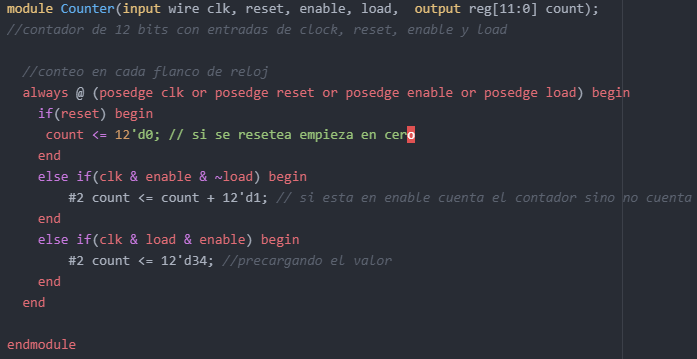
**Laboratorio #8**

**Enlace al repositorio de github:** [**https://github.com/gar19421/Lab-08-Digital.git**](https://github.com/gar19421/Lab-08-Digital.git)

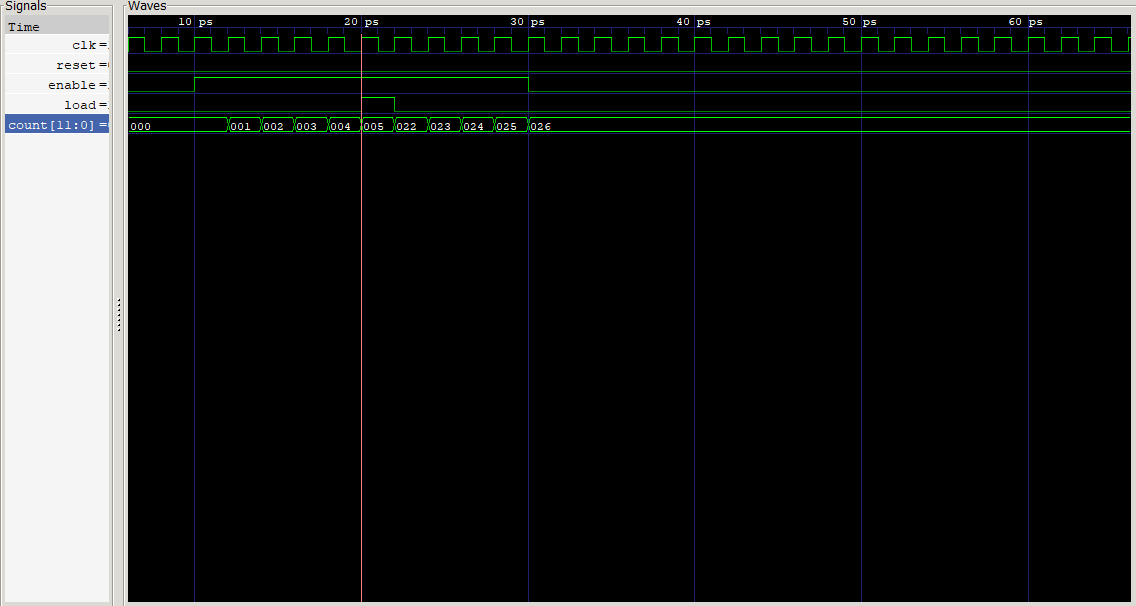
**Ejercicio 1:**

En este ejercicio consistió un contador el cual contara +1 en cada flanco cuando la señal de enable de entrada estuviera en 1 y que precargara un valor cuando la señal de load estuviera en 1, de forma asincrónica.

Se colocó dentro de un bloque always para verificar continuamente el valor del reset, el clock, el enable y el load y dependiendo del valor que tomara cada el contador aumentaba en uno cuando el enable estuviera activado, de lo contrario se mantenía en el mismo valor. Precargaba un valor en el contador de 12 bits todo esto se analizaba en cada flanco de reloj.

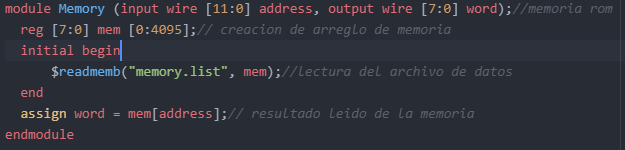


Como se puede observar en el diagrama de timing el funcionamiento anteriormente descrito funciona adecuadamente, ya que cuando el enable se activa espera para empezar a contar en el siguiente flanco de reloj y va sumando mientras este activado luego se activa load y se puede ver como se carga en el siguiente flanco de reloj un valor en el contador y continúa contando desde allí hasta que el enable se desactiva y deja de contar

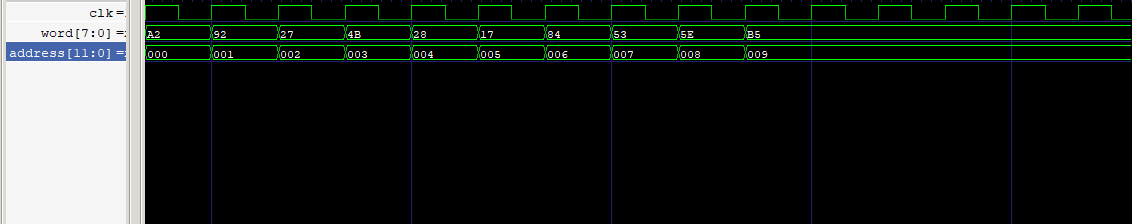


**Ejercicio 2:**

Este ejercicio consitió en construir en verilog una memoria ROM, la cual se implemento con un array, de la cual se pudiera leer datos de un archivo. Para realizar esto se utilizo el comando $readmemb para leer los datos del archivo memory.list de la memoria de 4kx8 construida, pidiendo una direccion de memoria y devolviendo la informacion contenida dentro de esta. Existe otra variación del comando mencionado anteriormente la cual es $readmemh, la diferencia entre estos dos comandos es que $readmemb sirve para leer archivos con datos en binario mientras que $readmemh para leer datos en hexadecimal.



Que como se puede observar en cada flanco de reloj se obtiene el valor de cada posición de memoria de los 10 datos solicitados para la lectura de la memoria ROM.

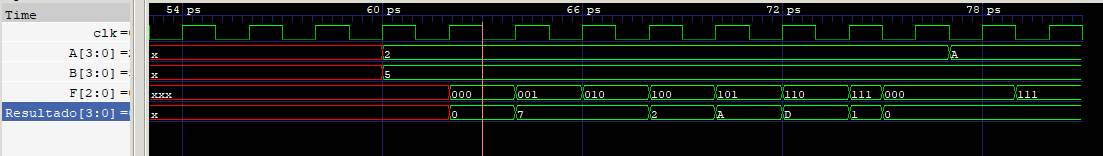


**Ejercicio 3:**

En este ejercicio se implementó la ALU de la sección 5.2.4. del libro con un switch case para realizar las distintas acciones del ALU con base al parámetro de entrada controlado por la variable F y se implementó las operaciones propias del ALU con implementación Behavioral.

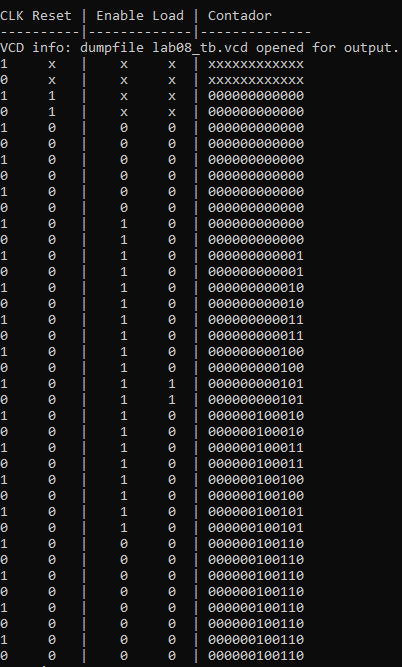
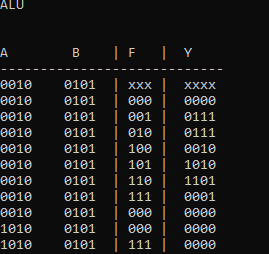
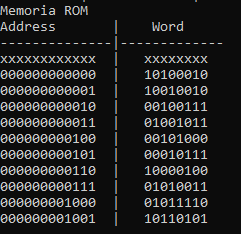


Como se puede observar en el diagrama de timing el resultado obtenido en cada una de las operaciones realizadas es el correcto con las distintas combinaciones de entrada de A y B y también el valor de salida es cero cuando no se realiza ninguna operación.

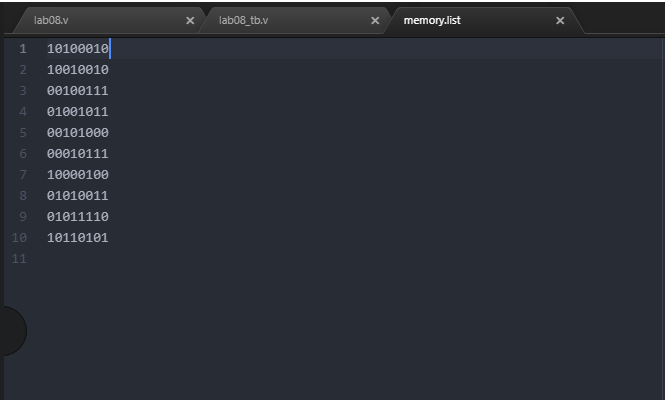


**Otras evidencias(screenshots):**

- Pruebas en consola:

Archivo de datos:



Testbech:



