**Postgrado ITESO**

**Diseño de Sistemas Digitales**

**Práctica 1 Diseño de una Unidad Aritmética Lógica (ALU) con Display**

Realizar el diseño e implementación de una ALU con display que cumpla con las siguientes especificaciones.

ESPECIFICACIONES

1. La ALU debe ser modelada utilizando un modelo parametrizado en Verilog comportamental. Utiliza un parámetro, *LENGTH*, para definir el número de bits de los puertos de entrada *A, B*. También el puerto de salida, *Result* de la ALU debe ser definido en términos de *LENGTH*.
2. Las operaciones a realizar por la ALU se determinan por el valor de la señal *Control*, como se describe en la Tabla 1.

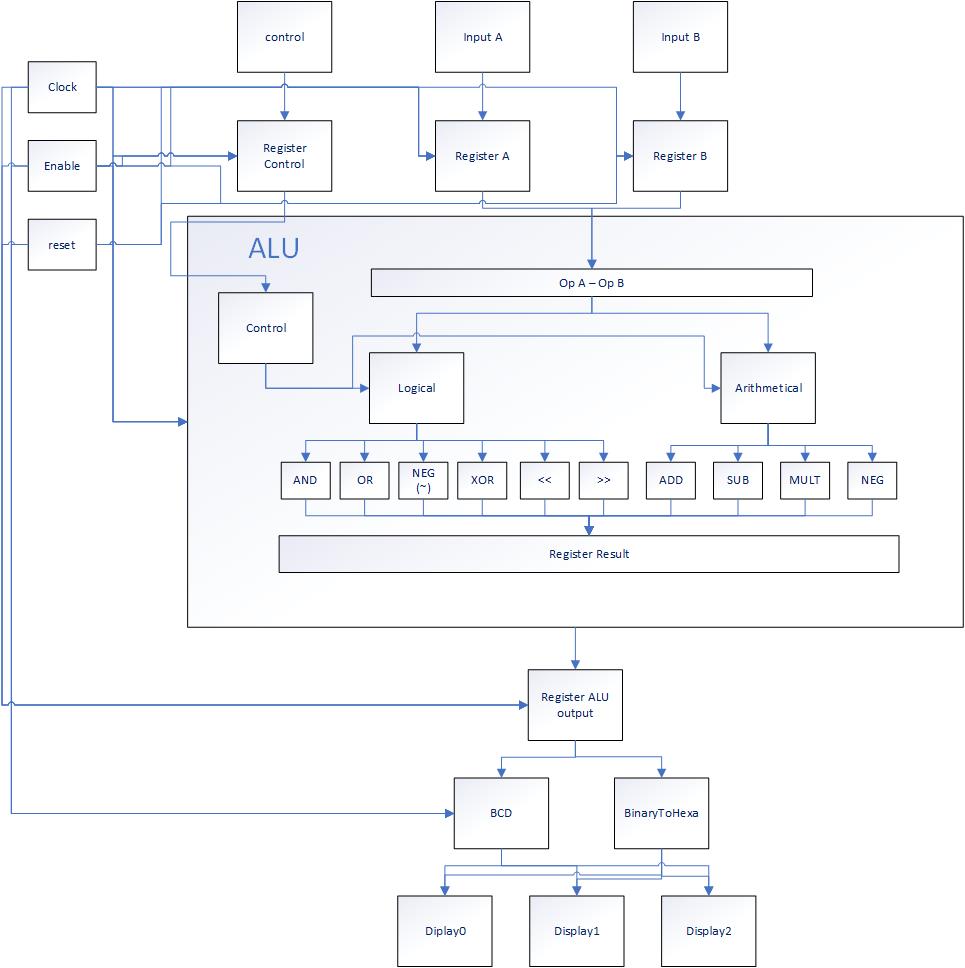
Tabla 1. Operaciones a realizar por la ALU dependiendo del valor de la señal *Control*.

|  |  |
| --- | --- |
| ***Control*** | **Operación a Realizar** |
| 0 | Suma aritmética de números signados de entrada: sistema numérico complemento a 2. |
| 1 | Resta aritmética de números signados de entrada: sistema numérico complemento a 2. |
| 2 | Negativo de *B*. Donde *B* es un numero signado. |
| 3 | Multiplicación de números signados de entrada: sistema numérico complemento a 2. |
| 4 | Operación lógica AND |
| 5 | Operación lógica OR |
| 6 | Negación lógica *A* |
| 7 | Operación lógica XOR |
| 8 | Corrimiento de bits: *Result* = A << B[3:0] |
| 9 | Corrimiento de bits: *Result* = A >> B[3:0] |

1. Las entradas y salida de la ALU deben estar registradas, los puertos no se conectan directamente a la ALU, sino que primero se hacen pasar por un registro.
2. Los resultados de la ALU se deben mostrar en un display de 7 segmentos 3 dígitos. Los resultados de las operaciones aritméticas signadas deben estar en formato decimal, prende un segmento adicional o un punto decimal para representar el signo. Los resultados de las operaciones lógicas se deben mostrar en hexadecimal.
3. El modelo de mayor jerarquía debe llamarse ALU\_Display, debe ser un modelo Verilog estructurado, donde instancies los bloques que lo componen, por ejemplo, ALU, registros, multiplexores y módulos del display. Cada módulo instanciado debe estar identificado por una etiqueta.
4. El parámetro *LENGTH* solo debe afectar a los puertos *A, B, Result*, no debe afectar a la señal *Control*.
5. La ALU debe contar con señales (banderas) de *carry, overflow, negativo y zero.*
6. Para la implementación en la tarjeta DE10-Standard, los operandos de entrada se deben definir en 5 bits y el resultado de la operación de multiplicación se debe truncar a 8 bits.
7. El diseño de mayor jerarquía debe contar con señales de *clk, rst, y enable*. Esta última señal es la que habilita el almacenamiento de los datos de entrada y resultado. Debido a que el numero de *switches* y *push-buttons* es limitado en la tarjeta DE10-Standard, propón una idea creativa, efectiva y funcional para el usuario, para manejar las señales de *control*, *rst* y *enable*.
8. Todos tus archivos Verilog deben estar debidamente comentados en las definiciones de puertos y variables y deben contar con un encabezado de comentarios donde expliques la función de los puertos y la descripción del diseño. Revisa la Fig. 1 al final de este documento, donde se muestra un ejemplo de modelo Verilog comentado.
9. La implementación no debe contener *latches*, es decir, que no aparezcan *warning* de *laches* en el reporte de compilación.

ENTREGABLES

Reporte subido en Canvas en la fecha de entrega definida. Usa este mismo documento para entregar tu reporte. El reporte debe contener los siguientes puntos.

1. Diagrama de bloques (arquitectura) del diseño de mayor jerarquía [20%].
2. Explicación del planteamiento de la solución al diseño [15%].

Se requiere entregar una ALU parametrizada para la cual en este caso utilizaremos 5 bits de entrada en A y otros 5 bits para B, se necesitan bits de enable, reset así como de control para poder elegir la operación a realizar, adicionalmente la salida deberá ser observada en los displays de 7 segmentos que se encuentran en la tarjeta, para las operaciones aritméticas es necesario mostrar si es positivo, negativo y su valor en decimal, para las operaciones lógicas es necesario desplegar el resultado en hexadecimal.

Se necesitan encender banderas de negativo, zero, carry y overflow.

Primero que nada se crearon módulos de registro (Flip-Flops) para poder registrar el valor de A y B cuando se apretara el botón enable, y como no se tienen suficientes botones y switches para la variable control\_top, se creo un registro en el cual se obtendrá su valor de los primeros 3 bits de A al apretar el botón control en la tarjeta, de esta manera ya podemos elegir la operación a realizar.

Después de esto se creó la lógica principal de la ALU en el modulo ALU el cual trabaja también con reset, clk, enable y control para realizar las operaciones, mandar el resultado a la variable result y trabajar con las banderas.

Se necesita desplegar el resultado en los displays por lo tanto se crearon los módulos BinaryToBCD y BinaryToHexadecimal, enviamos el resultado al módulo BinaryToBCD para convertirlo y ya dentro de BinaryToHexadecimal decidimos si utilizar el resultado crudo que viene directamente de la ALU o si utilizar el resultado que viene del modulo BCD dependiendo la operación que se realizó (si fue aritmética o lógica).

El módulo BinaryToHexadecimal envía la respuesta a los displays de 7 segmentos.

1. Modelos Verilog comentados, de cada una de los bloques del sistema y del diseño de mayor jerarquía [25%].

Modulo ALU\_Display

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente



Modulo register

Texto

Descripción generada automáticamente

Modulo register\_control

Texto

Descripción generada automáticamente

Modulo ALU

Texto

Descripción generada automáticamente

Modulo BinaryToBCD

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Modulo BinaryToHexadecimal

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Diagrama esquemático que genera la herramienta Quartus Prime (Tools > Netlist Viewers > RTL Viewer) [5%].

Diagram, schematic

Description automatically generated

1. Reporte de recursos utilizados por el sistema con comentarios de lo que observas [5%].

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Podemos observar que esta utilizando 108 ALMs (Adaptive Logic Module) que es menos del 1% de lo que podemos utilizar, 97 registros entre todo lo que utilizamos para hacerlo funcionar, 39 pines entre entradas y salidas y un bloque DSP (Digital Signal Processing).

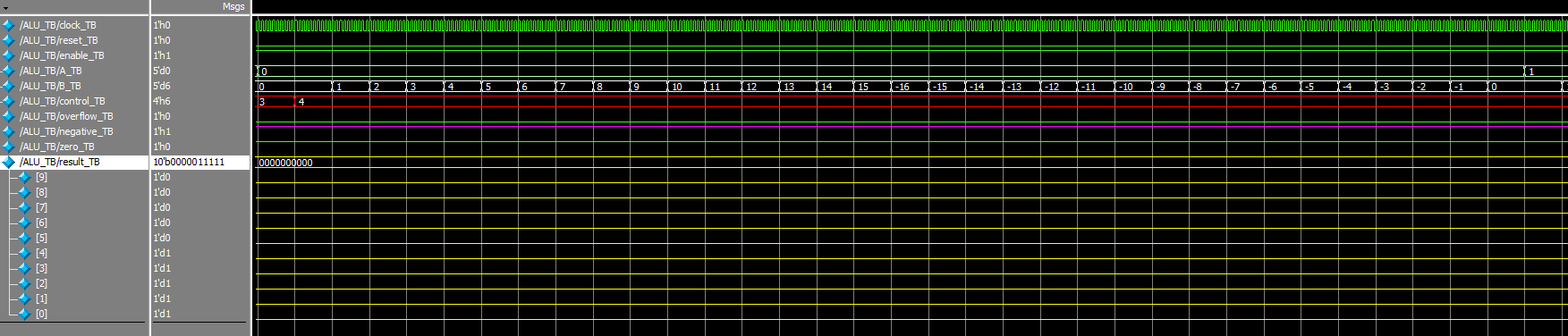
1. El *test-bench* y formas de onda de la simulación de la ALU, solamente la ALU, sin display ni decodificador a decimal, en dos partes:

NOTA: Para la demostración del testbench se agregaran algunos casos representativos y más significativos para ilustrar el funcionamiento ya que son demasiadas combinaciones y no sería practico añadirla todas al reporte

1. las operaciones lógicas con los valores en binario;

AND:

El arreglo “control” tiene un valor igual a 4 por lo cual será operación AND



A mantiene su valor en 0 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

A picture containing text, electronics

Description automatically generated

A mantiene su valor en 9 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

A picture containing diagram

Description automatically generated

A mantiene su valor en 15 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

A picture containing graphical user interface

Description automatically generated

A mantiene su valor en -16 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

OR:

El arreglo “control” tiene un valor igual a 5 por lo cual será operación OR

A picture containing text

Description automatically generated

A mantiene su valor en 0 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

A screenshot of a computer

Description automatically generated with medium confidence

A mantiene su valor en 9 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

Diagram

Description automatically generated with low confidence

A mantiene su valor en 15 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

A picture containing diagram

Description automatically generated

A mantiene su valor en -16 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

Negación de A:

El arreglo “control” tiene un valor igual a 6 por lo cual será operación Negación de A

Graphical user interface

Description automatically generated with medium confidence

En esta operación podemos ver la mayoría de las opciones posibles

XOR:

El arreglo “control” tiene un valor igual a 7 por lo cual será operación XOR

A picture containing graphical user interface

Description automatically generated

A mantiene su valor en 0 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

A screenshot of a computer

Description automatically generated with medium confidence

A mantiene su valor en 9 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

Diagram, schematic

Description automatically generated

A mantiene su valor en 15 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

Schematic

Description automatically generated with medium confidence

A mantiene su valor en -16 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

A << B:

El arreglo “control” tiene un valor igual a 8 por lo cual será operación A << B

A screenshot of a computer

Description automatically generated with medium confidence

A mantiene su valor en 1 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

A picture containing schematic

Description automatically generated

A mantiene su valor en 15 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

En esta operación podemos ver claramente como le bit se va corriendo a la izquierda y también podemos ver cuando se presenta el caso de overflow

A >> B:

El arreglo “control” tiene un valor igual a 9 por lo cual será operación A >> B

A picture containing diagram

Description automatically generated

A mantiene su valor en 9 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

Graphical user interface, timeline

Description automatically generated with medium confidence

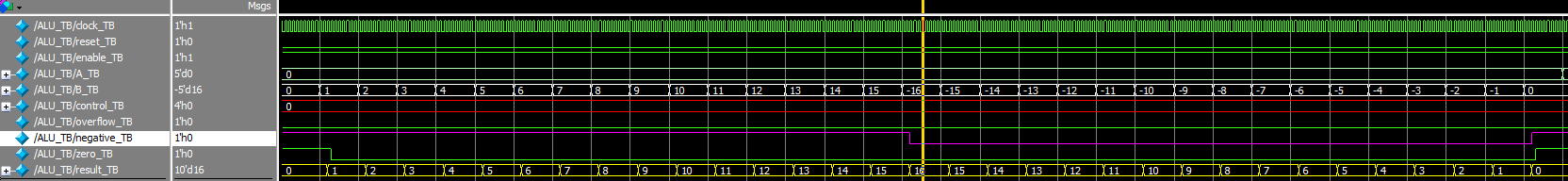
A mantiene su valor en -16 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0

En esta operación podemos ver como se hace el corrimiento de los bits hacia la derecha

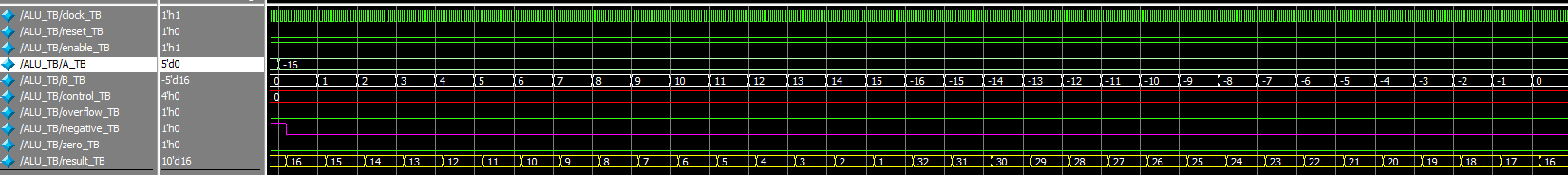
1. las operaciones aritméticas con los valores en decimal signado.

Suma:

El arreglo “control” tiene un valor igual a 0 por lo cual será operación SUMA



En esta imagen se puede observar la suma donde A conserva el valor de 0 y B esta incrementado su valor en uno cada 10ns los puntos a destacar aquí ver la salida que esta registrado un valor correcto y los bits “zero” y “negative” se están activando cuando nuestro resultado tiene un valor negativo o un valor igual a cero

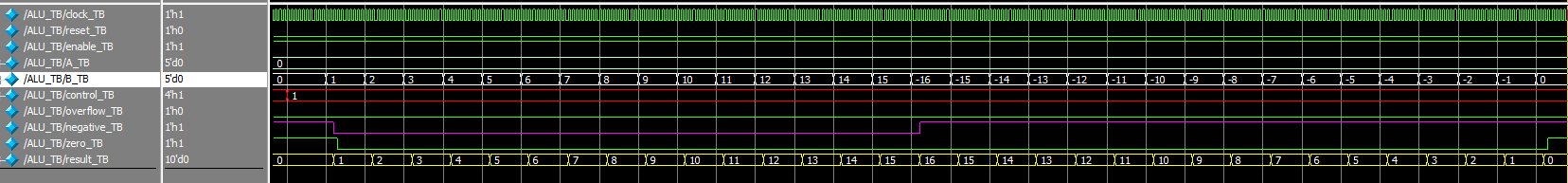


Ahora en este otro caso de la suma vemos como A mantiene su valor en mínimo que es menos 16 y B su valor va desde 0 hasta 15 y luego -16 a 0.

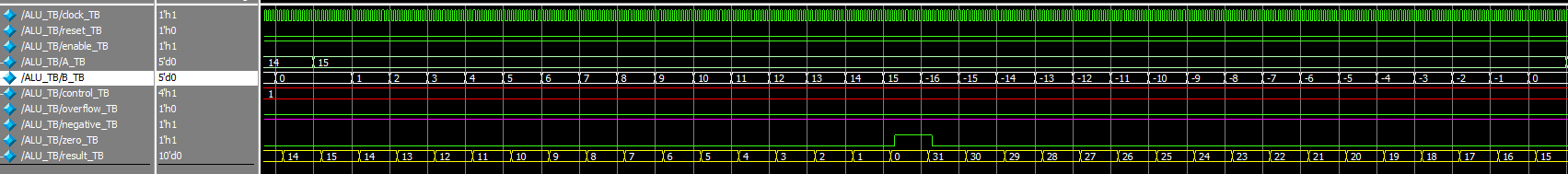
De la misma manera vemos el resultado correcto a la salida y como los bits “zero” y “negative” se están activando cuando nuestro resultado tiene un valor negativo o un valor igual a cero

Resta:

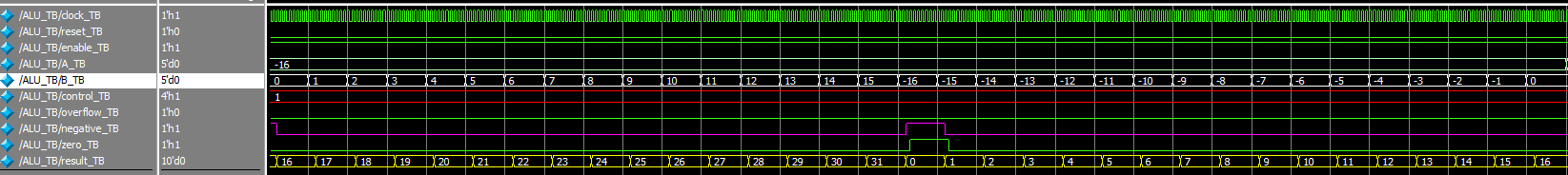
El arreglo “control” tiene un valor igual a 1 por lo cual será operación RESTA



A mantiene su valor en 0 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.



A mantiene su valor en 15 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

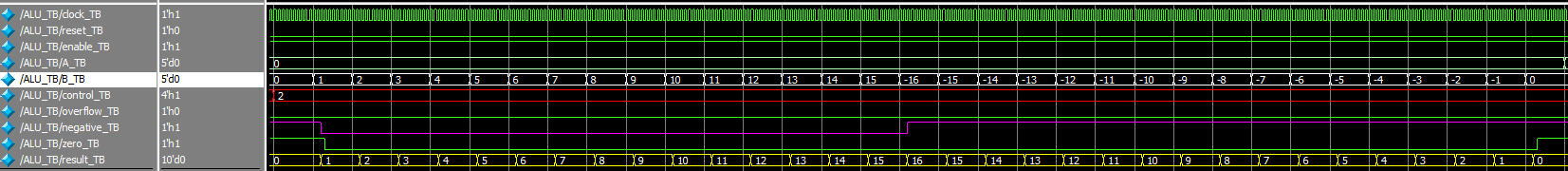


A mantiene su valor en -16 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

En los tres casos de la resta vemos como los bits “zero” y “negative” se están activando cuando nuestro resultado tiene un valor negativo o un valor igual a cero

Negativo:

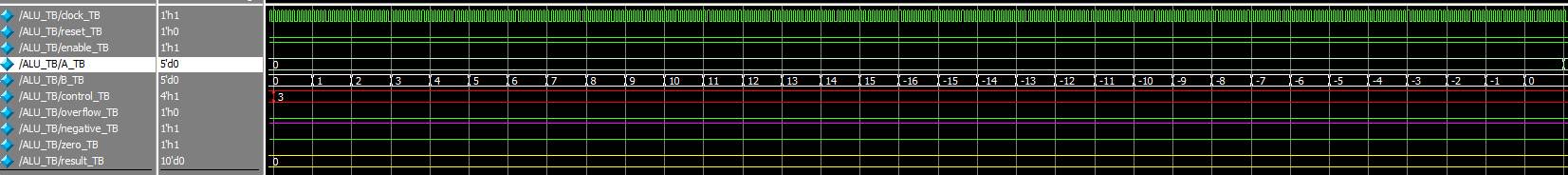
El arreglo “control” tiene un valor igual a 2 por lo cual será operación negativo



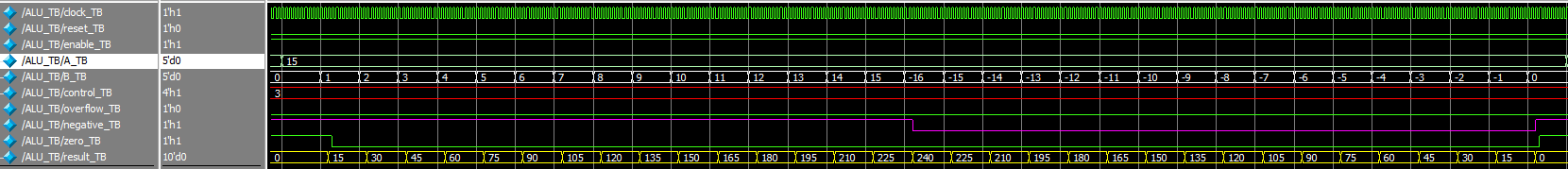
En la operación de control si vemos todos los casos posibles y de la misma manera vemos como los bits “zero” y “negative” se están activando cuando nuestro resultado tiene un valor negativo o un valor igual a cero

Multiplicación:

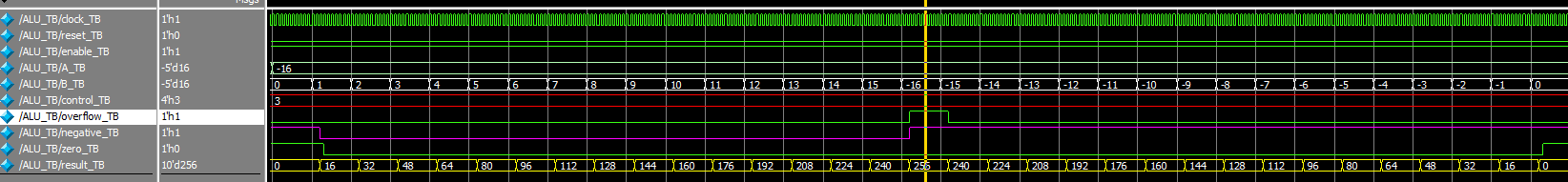
El arreglo “control” tiene un valor igual a 1 por lo cual será operación multiplicación



A mantiene su valor en 0 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.



A mantiene su valor en 15 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

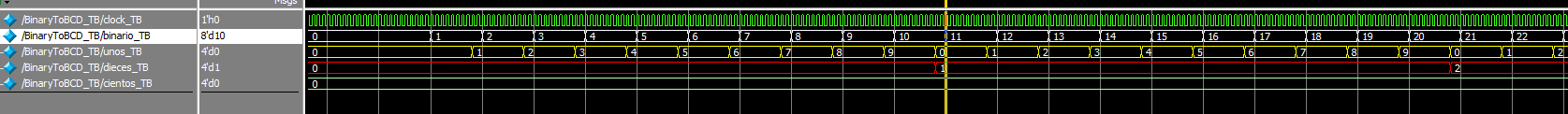


A mantiene su valor en -16 y B incrementa su valor desde 0 hasta 15 y luego -16 a 0.

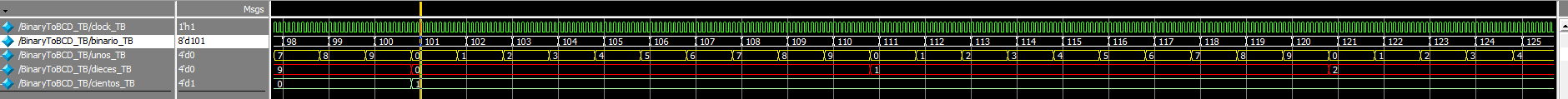
Un punto que destacar en esta operación es cuando A y B tienen un valor de -16 podemos ver que el bit de “overflow” se activa, siendo la única operación que lo dispara

Además, incluye el testbench y formas de onda de la simulación individual del decodificador de binario a decimal, donde reportes el resultado en grupos de 4 bits (BCD) [25%].

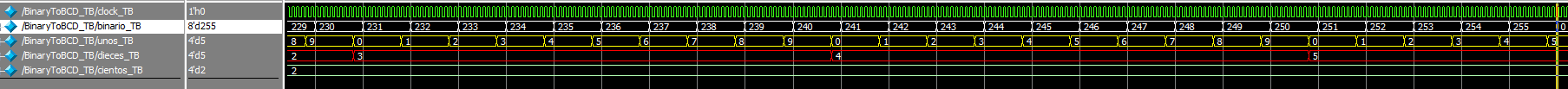
Para demostrar el testbech del módulo BCD se mostrarán casos representativos que nos permitan entender el funcionamiento



En esta imagen vemos como el arreglo “binario\_TB” va incrementado su valor de uno en uno, ese arreglo representa la entrada del BDC y dos casos a destacar en esta imagen es cuando el valor de “binario\_TB” es igual a 10 y a 20, vemos como el arreglo “dieces\_TB” incrementa a 1 y 2 respectivamente en los casos mencionado y el registro “unos \_TB” regresa a 0



En esta imagen tenemos otros 2 casos a destacar, podemos ver que cuando el valor de “binario\_TB” es igual a 99 los registros “unos \_TB” y “dieces\_TB” es igual a 9 cada uno, pero cuando pasa a 100 el valor de “binario\_TB” estos dos registros pasan a tener un valor de 0 y ahora el registro “cientos\_TB” tiene un valor de 1



Y para finalizar podemos ver la representación de 255 como los registros de la salida del módulo BCD tienen un valor de “unos \_TB” = 5, “dieces\_TB” = 5 y “cientos\_TB” = 2

1. Conclusiones y reflexiones sobre los aprendizajes [10%].

Al finalizar esta práctica uno de los conceptos elementales que se estuvieron reforzando y aprendiendo fue el lenguaje de modelado de HW Verilog, al no tener una gran experiencia previa con el lenguaje esta practica ayudo a practicar el desarrollo usando este lenguaje de modelado de HW.

Otro punto básico o fundamenta a destacar fue el desarrollo de los test benchs, también al comienzo fue algo complicado el hacer el desarrollo, pero sin duda al finalizar esta practica el conceptos y entendimientos de como elaborar un TB han mejorado y ahora desarrollo de un TB es más fácil

Dejando de lado un poco el ambiente de desarrollo o de pruebas de Verilog, el concepto de complemento a 2, ALU y BCD ahora quedan claros al nivel que fuimos capaces de desarrollar módulos funcionales y verificar su comportamiento probándolo tanto en los TB como el la misma FPGA

1. Referencias consultadas [5%].

[1]Arithmetic Logic Unit.[online].Available: <http://en.wikipedia.org/wiki/Arithmetic_logic_unit>.

[2] FPGA Architecture White Paper.[online]. Available: www.altera.com.cn/literature/wp/wp-01003.pdf.

[3]Wikipedia contributors. (2022, 12 agosto). Binary-coded decimal. Wikipedia. Recuperado 20 de septiembre de 2022, de <https://en.wikipedia.org/wiki/Binary-coded_decimal>

Presentación mostrando la implementación de la ALU\_Display en la tarjeta DE10-Standard. Sin presentación no se toma en cuenta el reporte.

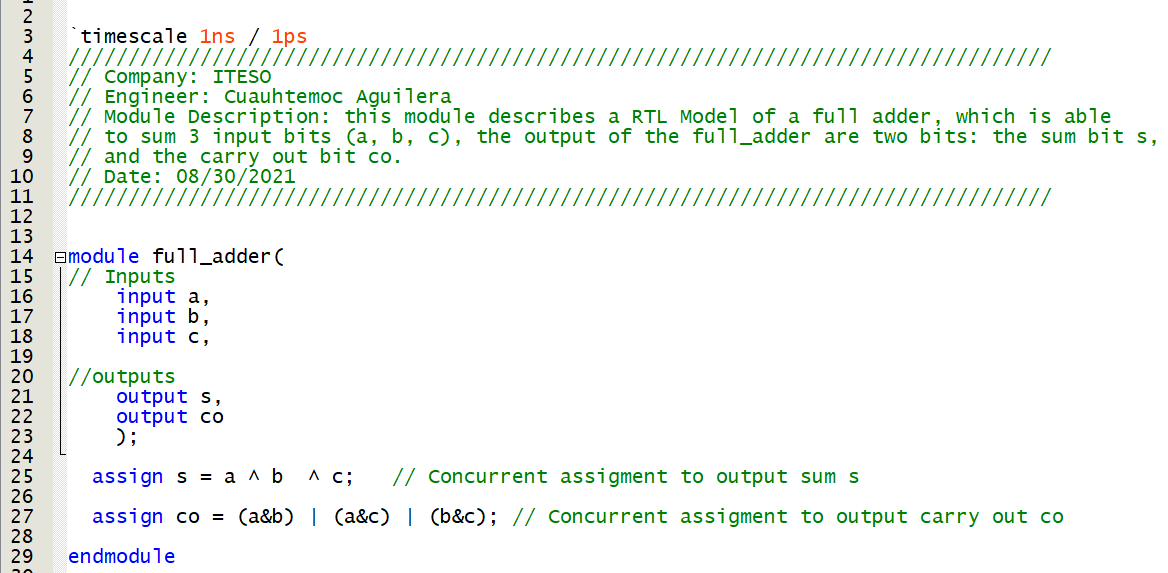


Fig. 1 Ejemplo de cómo se debe documentar, comentar y describir un modelo en Verilog