

ELECTRÓNICA DIGITAL

2023

**Alumnos:**

* Bargas Darío Santiago
* Mariño Pablo

**Tema:**  Implementación y programación de algoritmo divisor en una FPGA.

# Resumen:

El presente informe integrador final tiene como objetivo mostrar y dar explicación del diseño e implementación de un divisor de cuatro bits de números enteros. Para ello utilizamos una FPGA (dispositivo semiconductor programable que se utiliza para implementar circuitos digitales personalizados), donde con el lenguaje de descripción de hardware llamado Verilog, la programamos. Se seguirá un algoritmo especifico y se utilizaran los recursos de la FPGA para la prueba del mismo.

**Palabras claves:** FPGA, FSM, Contador, divisor, LEDs, bits, top.

# Introducción:

La consigna de este Trabajo Integrador Final consiste en realizar un divisor de números enteros de cuatro bits mediante el uso de una FSM (Finite State Machine), compuesta por:

* Módulo de procesamiento de datos que permita al usuario cargar los valores del numerador, denominador y obtener el resultado.
* Bloques combinacionales y/o secuenciales que procesan la información.
* FSM que realice la operación de división entera.

La cátedra propuso el siguiente algoritmo:

1. Cargar los datos, en este caso numerador y denominador de la división.
2. Determinar si el denominador es distinto de cero. Si el denominador fuera igual a cero, la división no sería posible. En este caso el proceso termina, pudiendo informar esta situación mediante algún mensaje o señal de error. De lo contrario, es posible realizar la división.
3. Para obtener el resultado y el resto de la división, es posible restar al numerador una cantidad igual al denominador, iterando sucesivamente hasta que el numerador sea menor al denominador. Cuando esto sucede, el resultado es igual al número de iteraciones realizadas, y el resto de la división es igual al valor del numerador tras la última iteración.

En este trabajo integrador final, se propone implementar un divisor de números enteros de cuatro bits utilizando una FPGA y una FSM. La FSM estará compuesta por un módulo de procesamiento de datos que permitirá al usuario cargar los valores del numerador y denominador, y obtener el resultado de la división. La operación de división entera se realizará siguiendo un algoritmo específico.

El algoritmo propuesto consiste en cargar los datos del numerador y denominador, verificar si el denominador es distinto de cero (si no lo es, la división no es posible y se mostrará un mensaje de error), y luego iterar sucesivamente restando al numerador el valor del denominador hasta que el numerador sea menor al denominador. El resultado de la división será igual al número de iteraciones realizadas, y el resto de la división será igual al valor del numerador después de la última iteración.

Para implementar este algoritmo, se utilizará una FSM que representará el comportamiento del sistema. Se diseñará un diagrama de estados y transiciones que describe el flujo de la FSM. Además, se determinarán los elementos de hardware necesarios para llevar a cabo el algoritmo propuesto.

El diseño se implementará en Verilog, un lenguaje de programación de hardware, y se comprobará su funcionamiento mediante testbench. Finalmente, se impactará el diseño en la FPGA proporcionada por la cátedra, utilizando los LEDs para visualizar el numerador, denominador y resultado, y los botones para realizar las operaciones de carga y visualización de resultados.

# 2. Desarrollo:

Antes de dar la explicación de diagramas y figuras realizadas, daremos una breve explicación y un ejemplo del algoritmo de división (Figura 1) que usamos y programamos. Imaginemos que queremos dividir 7 dividido 2:

* Primero Inicializamos los datos: Numerador=7 Denominador=2 Resto=7 Cociente=0 donde numerador y denominador son los datos ingresados, el resto es igual al numerador y el cociente es 0 que son los datos inicializados automáticamente cuando comienza a funcionar el algoritmo.
* Luego chequeamos que el Numerador sea mayor al Denominador para seguir iterando, si no cumple esa condición se corta la división. Si se cumple la condición, el Numerador ahora será igual al numerador anterior menos el denominador. El resto será igual al resto anterior menos el denominador. El cociente será igual al cociente anterior más uno y el denominador siempre se mantiene igual.
* Si el Numerador < Denominador, cortamos la división y mostramos el cociente y el resto. Se vio en el primer caso que al resto se le asigno el valor del numerador, y esto es porque si en la primera iteración se cumple la condición mencionada, el resto ya tiene su valor y el cociente también, que serian 0 y 7 respectivamente.

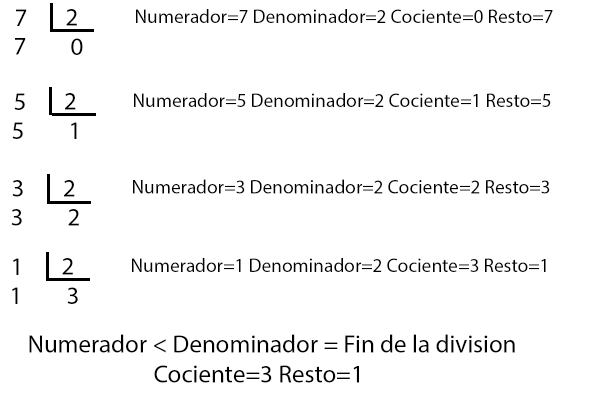


Figura 1. Ejemplo del algoritmo para la división

Una vez comprendido el algoritmo de división y visto el respectivo ejemplo anteriormente, procederemos a mostrar el diagrama de flujo (Figura 2). El mismo nos mostrara el funcionamiento de nuestra FSM.

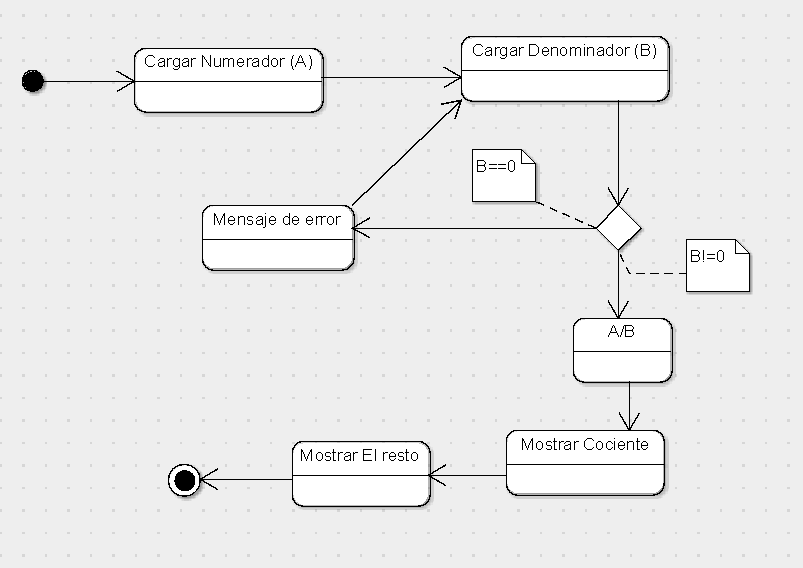


Figura 2. Diagrama de flujo de datos

A continuación, se describe cada uno de los pasos:

* Paso 1: Cargar numerador
* Paso 2: Cargar denominador
* Paso 3: Si el denominador es igual a cero, mostramos un mensaje de error y vamos a cargar el denominador nuevamente
* Paso 4: Si el denominador es distinto de cero vamos a realizar la división de A/B
* Paso 5: Mostramos el cociente
* Paso 6 Mostramos el resto
* Paso 7: Finaliza la división

Una vez planteado el proceso en un diagrama de flujo, se establecieron las decisiones de diseño para lograr una implementación eficiente y funcional.

El divisor comenzará en la carga de datos del numerador. Se podrá ingresar un valor del 0 al 15 en binario utilizando los botones de la FPGA. El número correspondiente seleccionado se irá mostrando en la FPGA con los LEDs encendidos, una vez que el contador llega al número 15 vuelve al 0. Cuando se confirma el valor requerido, el divisor pasará a la acción de elegir un valor para el numerador, con la misma lógica mencionada anteriormente. Si se selecciona un valor igual a 0 la FPGA hará un peque titilo en los LEDs, y volverá al estado de carga de denominador. Una vez validado el valor del denominador, el sistema realizará las operaciones asignadas y pasará a la muestra de resultados. En la misma se mostrará primero el cociente de la división a través de los LEDs de la FPGA. Si se pasa al siguiente estado el sistema mostrará el resto de la división, una vez mostrado el resto, se puede volver al estado de cargar otros valores para el numerador y denominador.

Como la FPGA consta de 4 botones, el control de los mismos se definió de la siguiente manera:

* **BOTÓN 1:** Incremento de los valores a ingresar.
* **BOTÓN 2:** Decremento de los valores a ingresar.
* **BOTÓN 3:** Ok, avanza al siguiente paso.
* **BOTON 4:** X.

Para la implementación de la FSM, se pensaron diferentes módulos que ayudaron a completar lo que seria el divisor completo. Hay varios caminos y opciones a la hora de realizar estos algoritmos y testear su funcionalidad en test-bench, sin embargo, a la hora de impactarlo en una FPGA pueden ocurrir bugs o ambigüedades y podría influir en su funcionamiento. Para evitar estos problemas se respetó la sintaxis del libro “FPGA Prototyping By Verilog Examples”, garantiza consistencia y estabilidad en el circuito impactado.

A continuación, se presentarán los mismos con su respectivo gráfico:

* **Módulo contador:**

Se comenzó programando el módulo contador, donde su funcionalidad es incrementar y decrementar valores entre 0 y 15 a través de una FSM. Se crearon 16 estados donde cada uno de esos estados corresponde a un número, por ejemplo, el estado cero corresponde al número 0, el estado uno corresponde al número 1, y así sucesivamente. A continuación, se ve la Figura 3 que representa el contador de 0 a 15.

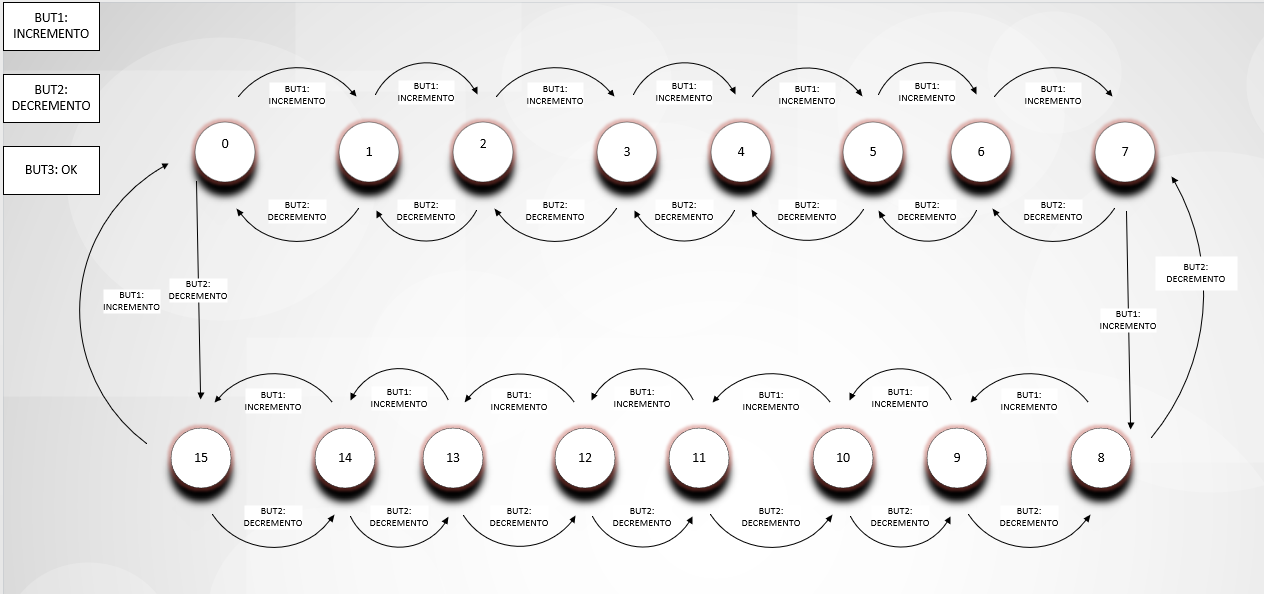


Figura 3. Diagrama de Transición de Estados del Módulo Contador

Un problema que se nos presentó durante la implementación de la misma fue que al apretar el botón en la FPGA para que el contador se incremente, ese incrementaba una gran cantidad de veces haciendo que no sea posible seleccionar el valor requerido, ya que el reloj de la FPGA oscila a 12MHz. Lo que significa que, si apretamos el botón 1 segundo, el contador avanzara millones de veces, por lo que sería imposible elegir el valor que queremos. Este problema se solucionó colocando una bandera, en la siguiente explicación se ve la resolución de este problema y básicamente el módulo completo:

Primero se declararon los registros estado actual, estado siguiente, bandera actual y bandera siguiente.

Luego se declararon dos bloques always donde la funcionalidad del primer always es:

Se ejecuta en cada flanco de subida del reloj (posedge clk) o en el flanco de subida del reset (posedge reset). Cuando el reset está activo, el estado se reinicia a cero y la bandera se pone en cero también. En caso contrario, el estado y la bandera se actualizan con los valores del próximo estado y la próxima bandera respectivamente.

Y la funcionalidad del segundo bloque always es:

* Controla la transición de estados basada en las señales de entrada “increase” y “prev”, y actualiza el estado y la bandera correspondientes.
* Si la señal increase está activa (botón increase pulsado) o la señal prev (botón prev pulsado) y la bandera actual es cero, se establece la próxima bandera en uno y se realiza una asignación de casos para incrementar o decrementar el valor.
* Si la señal increase está activa (botón increase pulsado) o la señal prev (botón prev pulsado) y la bandera actual es uno, mantenemos el estado para poder solucionar el error del reloj de los 12MHz de la FPGA, así solamente detecta como si lo pulsamos una sola vez
* Si la señal increase está inactiva (botón increase no pulsado) o la señal prev (botón prev no pulsado) y la bandera actual es uno, actualizamos la bandera a cero para poder volver a incrementar o decrementar los valores.

Una vez testeado su correcto funcionamiento mediante test bench, se procedió a implementarlo en la FPGA con la lógica de leds instanciando dentro del contador un módulo llamado FPGA.

El módulo FPGA conecta los botones de entrada (BTN1, BTN2, BTN3, BTN4) y las salidas de los LEDs (LED0, LED1, LED2, LED3) a un módulo contador llamado “cont (UUT)”. Los botones controlan el contador, y el valor del contador se refleja en los LEDs.

* **Módulo divisor:**

Para el divisor (Figura 4) se implementó la lógica mencionada anteriormente, donde constamos de 4 estados. Los cuales son:

* Carga
* Validación
* Cálculo
* Finalizado

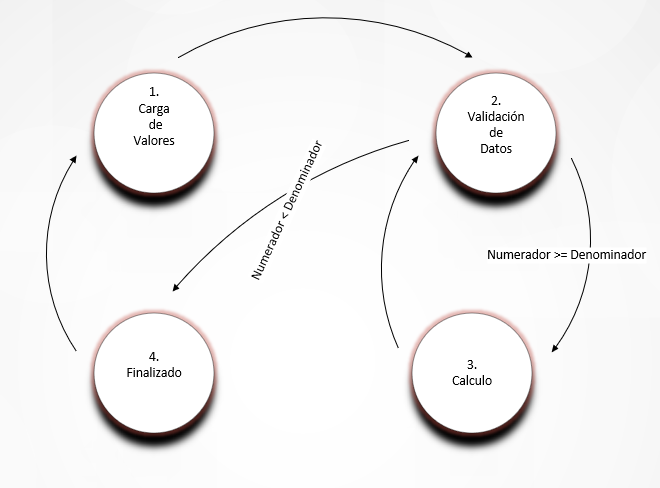


Figura 4. Diagrama de flujo de datos FSM divisor

La diferencia con el contador es que no necesitamos implementar botones, sino que se realiza con el flanco de subida del clk. Sus entradas son el clk(reloj), los reseteos sincrónico y asincrónico, el comienzo(start), los datos de entrada (numerador y denominador) y los datos de salida (cociente, resto y fin). Se definieron registros para almacenar el estado actual y siguiente de la FSM. También se definieron dos bloques always donde la funcionalidad de cada uno es:

* El primer bloque always se encarga de controlar la sincronización y el restablecimiento del circuito en función de las señales de reloj (clk) y reset (rst). Dentro de este bloque se verifica si la señal de reset (rst) está activa. Si es así, se asignan los valores iniciales a todas las variables internas, estableciendo el estado en carga y los registros en cero. Además, la señal finish se establece en cero. Si la señal de reset no está activa, se asignan los valores actuales de las variables internas a los registros de retención correspondientes. En resumen, este primer bloque always asegura que el circuito comience en un estado predefinido cuando se active la señal de reset (rst), y que las variables internas mantengan su valor actual cuando no haya un reset activo. Esto proporciona un mecanismo para inicializar y sincronizar el circuito correctamente.
* El segundo bloque always implementa la lógica del algoritmo de división, actualizando las variables internas en función del estado actual y las señales de entrada. Esto permite realizar los cálculos necesarios para la división y controlar el flujo del circuito hasta que se complete la operación.
* **Modulo multiplexor:**

Este módulo, como el mismo nombre lo dice, cumple la función de ser un multiplexor. Pero, ¿para que usamos un multiplexor? Lo usamos justamente para multiplexar lo que queremos mostrar. Es decir, nosotros en la FPGA, debemos mostrar el numerador, el denominador, el cociente y el resto, pero esto obviamente no se puede mostrar al mismo tiempo, por lo que un multiplexor de 2 bits (Figura 5) para elegir que queremos mostrar nos fue de muchísima utilidad. Al mismo le pasamos cada bit, de lo que queremos mostrar, y un selector. Entonces dependiendo el selector va a mostrar el numerador, el denominador, el cociente y el resto. Dejo aquí un pequeño grafico para que sea mas legible el entendimiento del mismo:

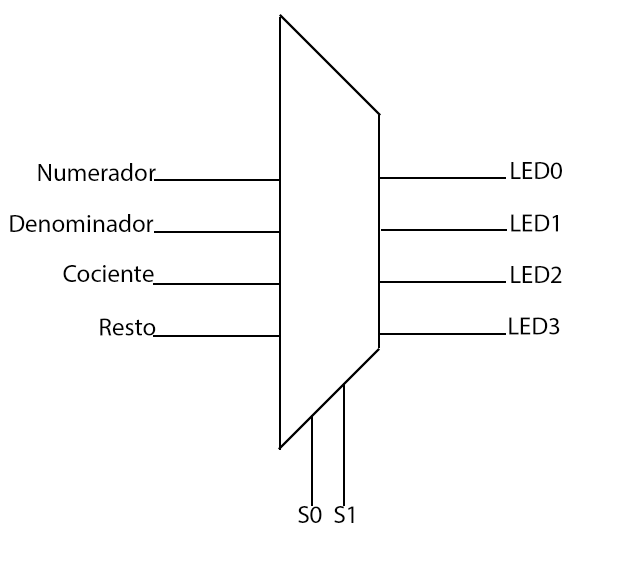


Figura 5. Multiplexor

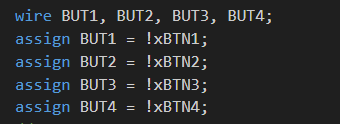
* **Modulo top:**

En términos de implementación, es el módulo más largo. Este módulo, representado en la Figura 6, es el hace la llamada a los módulos anteriores y se hace la programación del algoritmo completo.



Figura 6. Diagrama de flujo de datos de FSM top

El código de este módulo comienza invirtiendo los botones de la FPGA (Figura 7), ya que tienen la particularidad de que mandan una señal 1 cuando no son presionados y 0 cuando lo son. Como los módulos internos fueron implementados con una lógica inversa, los botones de entrada deben ser invertidos. Lo realizamos de la siguiente manera:



xBTN1, xBTN2, xBTN3 y xBTN4 (botones que se ingresan)

Figura 7. Inversión de los botones

Luego de hacer la inversión de los botones, definimos los estados de la maquina de estado, el mismo tendrá los estados:

* Cargar Numerador: Sera el encargado del control de la carga del numerador.
* Cargar Denominador: Sera el encargado del control de la carga del denominador.
* Verificar: en este estado se verifica que el denominador sea distinto de cero.
* Calcular: se hará la asignación de los valores de la división.
* Mostrar Cociente: estado encargado de mostrar el cociente de la división.
* Mostrar Resto: estado encargado de mostrar el resto de la división.

Valga la redundancia, se siguió básicamente casi la misma sintaxis que los módulos anteriores, creando registros de estado actual y estado siguiente. Estas declaraciones (Figura 8) con la forma de variable actual y variable siguiente también se asignaron, al numerador ingresado, al denominador ingresado, al cociente, al resto y al resto de las variables para el divisor. Dejo a continuación un fragmento del código para observar cómo se llevó a cabo:

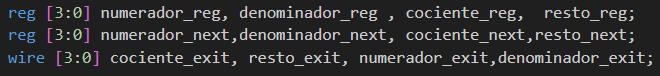


Figura 8. Sintaxis de la declaración de registros

Se puede ver como también se definió un “wire” que el mismo se utiliza para declarar una variable que representa una conexión o un bus de señales en un diseño digital. Estos nos servirán a la hora de llamar, al resto de los módulos, ya que por ejemplo al módulo divisor (Figura 9), necesitaremos pasarle las variables de entrada como son el denominador, y el numerador; lo mismo para las variables de salida, un cociente y un resto. A continuación, dejare un fragmento del código de como se llamo al modulo divisor. Recordemos que este modulo es el que realiza el cálculo de la división:

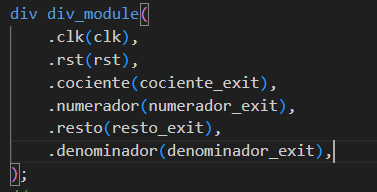


Figura 9. Llamada al módulo div.

La misma declaración de variables y la misma sintaxis de llamada se realizo para el modulo del contador (Figura 10), con la única diferencia que como variable de entrada al mismo se le asigno el par de botones, el BTN1 y el BTN2; recordemos que el contador tenia para incrementar o decrementar los numero del 0 al 15 por lo que al BTN1 se le asigno la tarea de incrementar y al BTN2 la tarea de decrementar. Se hicieron dos llamadas a los módulos contadores, uno para asignar el numerador y el otro para asignar denominador. Nuevamente dejare un pequeño fragmento del código para observar la llamada y la asignación del botón:

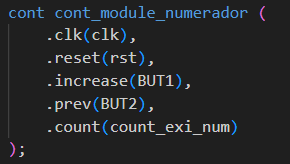


Figura 10. Llamada al módulo cont.

Luego de la asignación de los datos y de todas las llamadas de los módulos correspondientes, se procederá a explicar ambos always como se han explicado en los módulos anteriores, sin antes declarar una ultima variable, una bandera. Esta bandera nos permitirá resolver el problema que se nos presentó en el módulo del contador. Respetando la sintaxis de estado actual, y estado siguiente.

Para el primer always lo que se hizo fue encargarse de controlar la sincronización y el restablecimiento del circuito en función de las señales de reloj (clk) y reset (rst). Dentro de este bloque se verifica si la señal de reset (rst) está activa. Si es así, se asignan los registros en cero. Si la señal de reset no está activa, se asignan los valores actuales de las variables internas a los registros de retención correspondientes. En resumen, este primer bloque always asegura que el circuito comience en un estado predefinido cuando se active la señal de reset (rst), y que las variables internas mantengan su valor actual cuando no haya un reset activo. Esto proporciona un mecanismo para inicializar y sincronizar el circuito correctamente. Es básicamente lo que hacen los always anteriores.

Para el segundo always lo que se hizo (como en los módulos anteriores), se implemento la lógica del algoritmo completo, como se ve en la Fig5. Primero se definen las asignaciones para las variables state\_next, numerador\_next, denominador\_next, cociente\_next, resto\_next y sel\_next. Estas variables representan los valores que tomarán en el siguiente ciclo de reloj.

La lógica en este bloque está controlada por la condición bandera\_reg == 0 & BUT3 == 1, lo que significa que solo se ejecutará si la bandera está en 0 y el botón BUT3 está presionado. En caso de que se cumpla esta condición, se realiza la asignación correspondiente a cada variable según el estado actual (state\_reg), donde los estados son los mencionados anteriormente. Al final se observa en el código como se declara un “default”, lo que significa es que, si el estado actual no corresponde a ninguno de los seis que están en la Figura 6, automáticamente las variables tendrán unos valores predeterminados. En la Figura 10 se ve la implementación del “default”

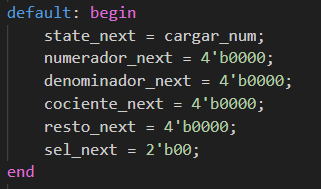


Figura 10. Fragmento del código que tiene un default

En el caso que la bandera este en 1 y el botón 3 este presionado, (bandera\_reg == 1 & BUT3==1) no cambiamos de estado, esto es para solucionar el problema del reloj de la FPGA ya mencionado.

Finalmente, en el caso que la bandera esta activa y el botón no está apretado, (bandera\_reg==1 & BUT3 == 0) se detectó que el botón no está siendo apretado entonces podemos continuar con el proceso, y para ello declaramos la bandera en 0.

# 3. Conclusión

En conclusión, este informe integrador final ha abarcado el diseño e implementación de un divisor de cuatro bits de números enteros utilizando una FPGA y una FSM. Se ha seguido un algoritmo específico para realizar la operación de división entera y se han utilizado los recursos de la FPGA, junto con el lenguaje de descripción de hardware Verilog, para programar y probar el funcionamiento del divisor.

El algoritmo propuesto se basa en cargar los datos del numerador y denominador, verificar la viabilidad de la división, y realizar iteraciones sucesivas restando al numerador el valor del denominador hasta que el numerador sea menor al denominador. El resultado de la división se obtiene contando el número de iteraciones realizadas, y el resto se determina como el valor del numerador después de la última iteración.

Para llevar a cabo este algoritmo, se ha diseñado una FSM que representa el comportamiento del sistema. Se han establecido decisiones de diseño para garantizar la eficiencia y funcionalidad del circuito. Durante el proceso de desarrollo, se han enfrentado desafíos técnicos, como la sincronización del contador y la resolución de problemas relacionados con la frecuencia del reloj de la FPGA. Sin embargo, se han encontrado soluciones efectivas para superar estos obstáculos y lograr un diseño estable y consistente.

El resultado final es un divisor de cuatro bits que cumple con los requisitos planteados, permitiendo al usuario cargar los valores del numerador y denominador, realizar la división y mostrar el cociente y el resto a través de los LEDs de la FPGA. Se ha realizado una implementación robusta y se ha verificado su funcionamiento mediante pruebas de simulación y la utilización de un testbench .

# 4. Referencias

[1] P. P. Chu, FPGA Prototyping by Verilog Examples. Hoboken, New Jersey: John Wiley & Sons, Inc., 2008