[IPD432] Diseño Avanzado de Sistemas Digitales Tarea # 1

Mauricio Aravena Cifuentes
Departamento de Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
maurio.aravena@sansano.usm.cl

I. Introducción

En el presente trabajo, se tiene como objetivo principal retomar y *desempolvar* los conceptos asociados al diseño e implementación de circuitos digitales utilizando un entorno de descripción de hardware. Así como también, permitir repasar el funcionamiento, definición y análisis de máquinas de estado.

II. Debouncer para pulsadores electromecánicos

A. Análisis de comportamiento

Realizando la simulación temporal para analizar el comportamiento de ambos debouncer (lógico y basado en *FSM*), se llega a la representación temporal mostrada en la Figura 1.

Como se puede observar, el comportamiento de ambos dispositivos antirebotes es análogo. Al momento de recibir una pulsación desde `PB`, se determina si es una pulsación válida o un *glitch*, para ambos debouncer la metodología para determinar esta condición es similar, un contador de al menos 15 ciclos continuos se debe cumplir, para aceptar la señal como válida. En el caso de ser una pulsación válida, se genera un pulso `PB_pressed_pulse`, que indica que una pulsión ha ocurrido. Luego de esto, una señal limpia acorde al tiempo aproximado que el botón se mantiene presionado se activa, `PB_pressed_status`. Finalmente cuando se suelta el botón, se baja la señal de presión y se envía un pulso que indica que el botón ha sido liberado, `PB_released_pulse`. Una diferencia que se puede apreciar al analizar la duración de la señal `PB_pressed_status`, es que en el caso del debouncer basado en FSM este es menor que la misma señal en el debouncer lógico. En la tabla I, se observa que existe un delta $\Delta = 32$ *ns*, para las señales entre debouncers. Analizando la implementación en código del debouncer basado en FSM y comparando con el deboucer lógico, se puede concluuir que la diferencia se atribuye a que este último, compensa los ciclos de presión del botón que se *consumen* al momento de determinar si la pulsación detectada es válida y no producto de la oscilación de los terminales del botón. Luego de que la pulsación deja de ser detectada el debouncer lógico espera 15 ciclos antes de bajar la señal de presión estable. En cambio, el debouncer por FSM, al momento de bajar la señal `PB` realiza el cambio de estado, sin realizar una compensación por los ciclos utilizados para determinar si la señal estaba estable.

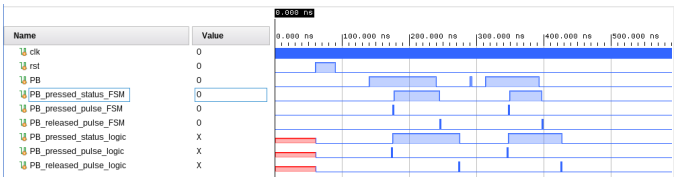

Fig. 1. Representación del comportamiento en el tiempo, para los dos tipos de debouncer analizados.

TABLE I
Análisis de la diferencia temporal para la señal `PB_pressed_status` entre ambas máquinas

PB *ns*	**PB_pressed_status_FSM** *ns*	**PB_pressed_status_logic** *ns*
100	68	100
80	48	80

Realizando una modificación del *testbench* para poder llevar al límite la funcionalidad de los debouncers, se escoge un pulso que sea similar al número de ciclos necesario para determinar la estabilidad del pulso (mayor a 15 ciclos). Experimentalmente se determinó que para ambos casos de debouncers, se tiene un overhead de aproximadamente 15 ciclos, por lo que definimos un pulso para la señal `PB` de 31 ciclos. El resultado de esta simulación se encuentra en la Figura 2. Como se puede observar, para el caso del debouncer implementado mediante una *FSM*, solo se obtiene el pulso asociado a la presión del botón. Este comportamiento se relaciona a la manera en que la FSM fue implementada, dado que se tiene que el estado asociado a la salida `PB_pressed_status_FSM`, solo es alcanzable si el pulso asociado a la presión sostenida del botón sigue estando activo, de esta forma se obtiene un resultado que solo entrega información de que se ha pulsado el botón[1]. En comparación el debouncer lógico es capaz de entregar las tres señales asociadas a la pulsación del botón de manera correcta.

B. Circuitos inferidos

A partir de las descripciones en HDL de cada modulo, se realizó diagramas de los circuitos digitales asociados. El circuito asociado al debouncer lógico se encuentra en la Figura 3 y el circuito asociado al debouncer basado en *FSM* se

[1]En la siguiente sección, cuando se realice una análisis al diagrama de estados del debouncer mediante FSM, esta situación será más evidente.

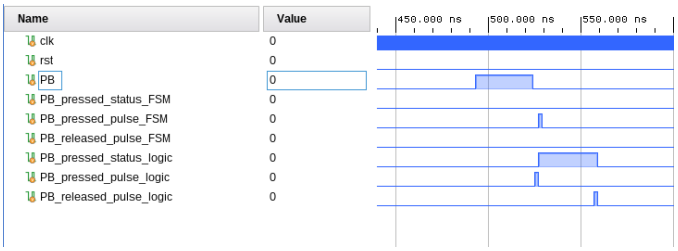

Fig. 2. Simulación para una situación límite para los debouncer

encuentra en la Figura 4. Comparando estos circuitos inferidos por los entregados por Vivado, mediante la herramienta `RTL ANALYSYS`. Se obtiene para el debouncer lógico el circuito mostrado en la Figura 5 y para el debouncer basado en *FSM*, en la Figura 6. Realizando una comparación entre ambos, se puede observar que son muy similares. Se podría decir que el diagrama inferido y el obtenido para el debouncer lógico son iguales, en cambio para el debouncer *FSM* existen algunas diferencias en la forma que se implementa la lógica de los cambios de estado, sin embargo, la topología de los circuitos es símil y la elección de componentes representativos es acorde a lo deducido. Comparando ahora, con los circuitos obtenidos luego de realizar la síntesis de cada módulo [2], se puede observar que estos son distintos a nivel de componentes e interconexión. Esto se debe a que el esquemático obtenido en `RTL ANALISYS` es una representación del módulo definido pre-optimización específica del hardware. Se utilizan elementos genéricos que son independientes de una plataforma particular. En cambio, luego de realizar la síntesis el circuito obtenido, este es una representación de un proceso de optimización, el cual utiliza las componentes disponibles en el hardware para la generación del módulo. Por lo que este diagrama depedende de forma neta de la arquitectura que se esté utilizando.

C. Reporte de la utilización de recursos

En las Tablas II y III, se adjunta la utilización de recursos para cada implementación.

[2]Estos diagramas, dada su complejidad son obviados, dado que no podrían ser visualizados correctamente en este documento.

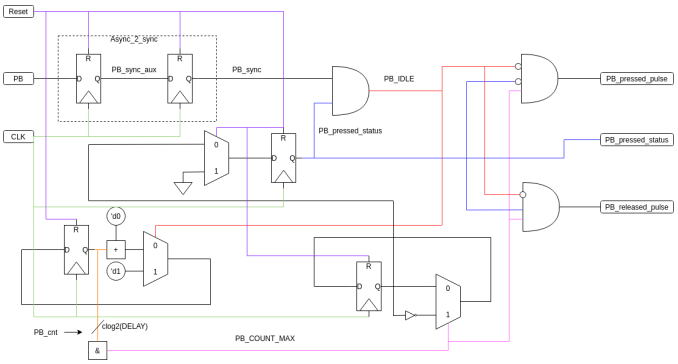

Fig. 3. Circuito inferido para debouncer lógico

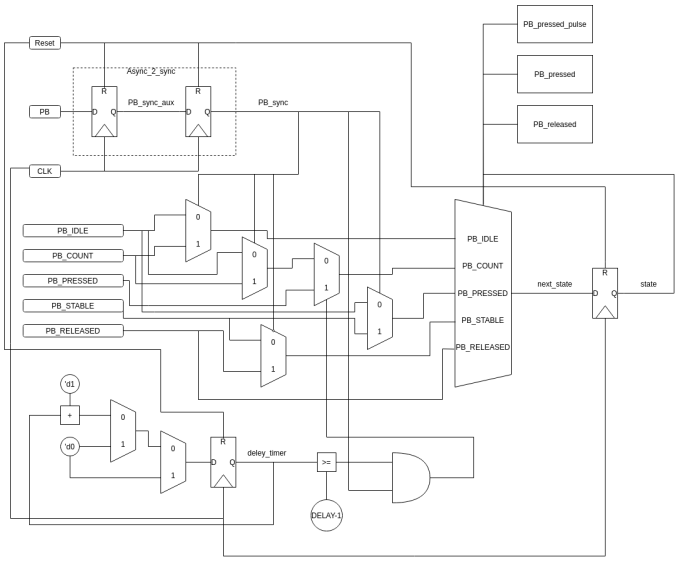

Fig. 4. Circuito inferido para debouncer mediante *FSM*

TABLE II
REPORTE DE UTILIZACIÓN DE RECURSOS PARA DEBOUNCER LÓGICO

Recurso	Utilización	Disponible	Utilización %
LUT	8	63400	0.02
FF	7	126800	0.01
IO	6	210	2,86

TABLE III
REPORTE DE UTILIZACIÓN DE RECURSOS PARA DEBOUNCER MEDIANTE *FSM*

Recurso	Utilización	Disponible	Utilización %
LUT	10	63400	0.02
FF	11	126800	0.01
IO	6	210	2,86

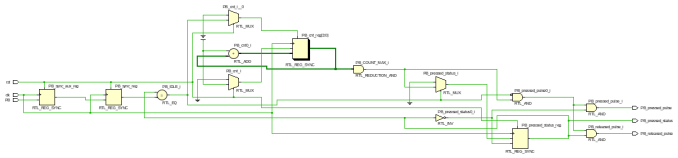

Fig. 5. Circuito digital obtenido mediante análisis RTL en Vivado, para el debouncer lógico

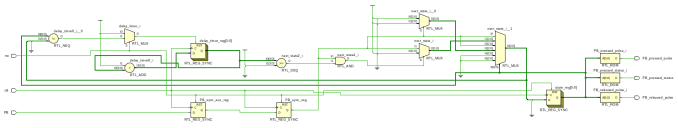

Fig. 6. Circuito digital obtenido mediante análisis RTL en Vivado, para el debouncer mediante *FSM*

D. Análisis de ventajas comparativas entre los debouncers

En términos funcionales, ambos módulos son capaces de entregar las señales requeridas, sin embargo, como se determinó en la etapa de análisis de simulación, el debouncer lógico tiene un mejor comportamiento cuando las señales de presión del botón son muy cercanas a los ciclos necesarios para determinar si la pulsación es estable, si una situación como esta ocurre el módulo basado en FSM podría no entregar los resultados buscados. En términos de recursos utilizadas ambas opciones son bastante similares, siendo la lógica por muy poco la opción más conservadora de recursos. Por otro lado, en su facilidad de descripción a nivel de código e interpretación el debouncer implementado por máquina de estado tiene la ventaja, dado que los estados permiten un diseño más sencillo a nivel humano.

E. Flip-Flops de sincronización

Los Flip-Flops a la entrada del circuito, corresponden a un circuito simple que permite tomar una señal asincrónica como es la pulsación de un botón y permiten sincronizarla con el reloj del sistema. De no incluir esta etapa de *filtraje*, se podría tener comportamiento asincrónico, lo que no es deseado en un sistema controlado por reloj.

III. Análisis de FSM debouncer

A partir de la implementación en código, se puede determinar que la *FSM* implementada corresponde a una *timed-FSM*, por lo que se puede modelar en términos de diagramas de bloque, como se muestra en la Figura 7. Donde se tiene una máquina de estados principal, en este caso, la que controla los valores de las salidas y una máquina de apoyo, en este caso el contador, para la verificación de la estabilidad del pulso. De esta forma, la implementación y representación del módulo se simplifica bajo el principio de *divide and conquer*.

En la Figura 8, se muestra la representación de la máquina de estado principal y en la Figura 9, se llevó el contador a una representación de máquina de estado. De esta forma, la implementación del módulo se simplifica, haciendo que cada máquina tome una tarea, en comparación con la dificultad a nivel de definición e implementación que conllevaría realizar todo en una sola máquina.

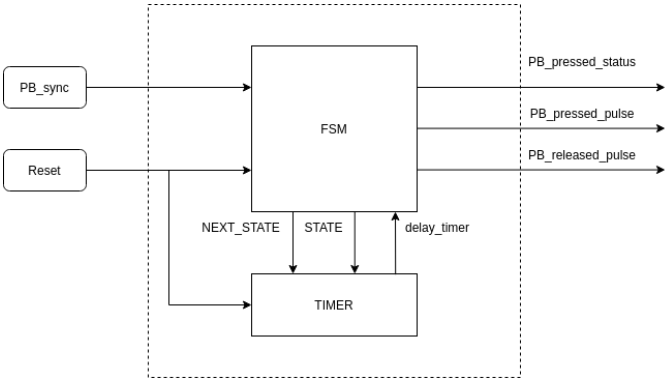

Fig. 7. Representación de la *timed-FSM*

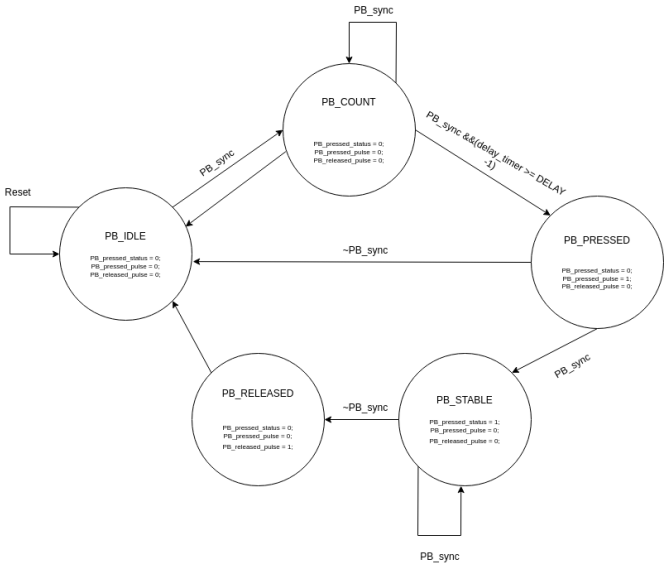

Fig. 8. Representación de la FSM principal

IV. Diseño y verificación de FSM

A partir de los requisitos solicitados, se definió la máquina de estados, representada en Figura 10 para implementar en código, los archivos correspondientes, fueron adjuntados con el presente documento.

V. Diseño e implementación de reloj alarma

Para la implementación del reloj, se propone un circuito que siga las descripciones dadas en el diagrama de alto nivel, Figura 11. En las siguientes secciones se detallará el funcionamiento de los bloques.

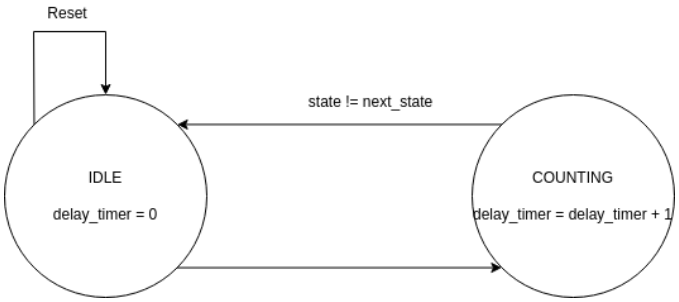

Fig. 9. Representación de la FSM secundaria

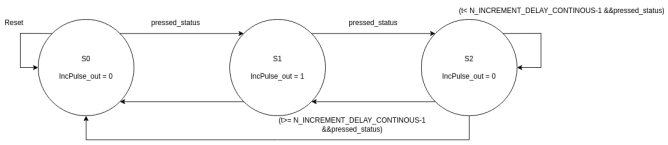

Fig. 10. Máquina FSM derivada de los requisitos solicitados

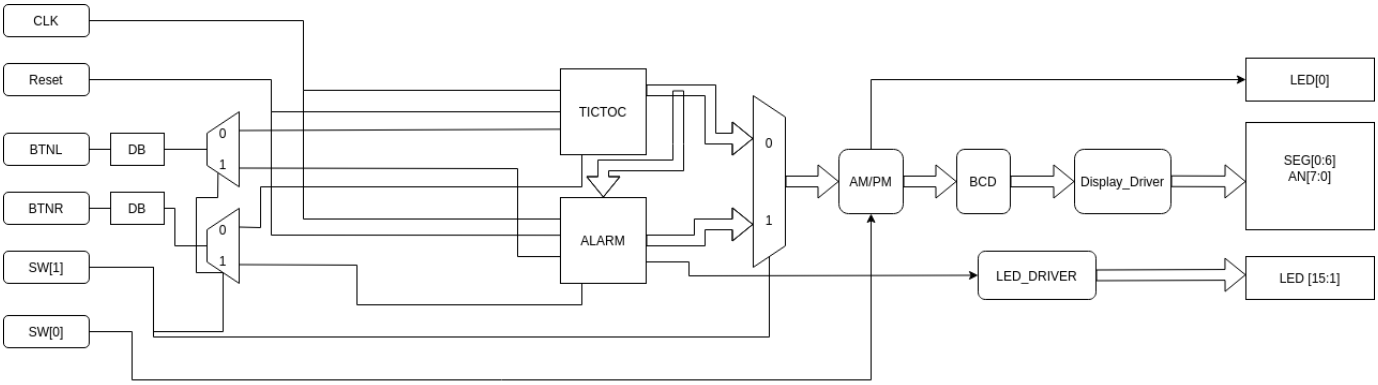

Fig. 11. Propuesta para la implementación del reloj digital

A. Bloque: DB

Este bloque corresponde a la conjunción de un debouncer basado en la FSM utilizada en la primera sección, con su estructura y diagramas de estados descrito en las Figura 7, 8 y 9, y la máquina *level-to-pulse*, definida en 10.

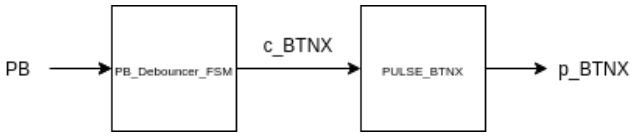

Fig. 12. Módulo DB

Su función es tomar la señal proveniente de `BTNR/BTNL`, realizar debounce y generar un pulso asociado al tiempo que se ha mantenido presionado el botón.

B. Bloque: TICTOC

Este bloque corresponde a el mecanismo principal del reloj, tiene la tarea de contar realizar la conversión de ciclos de 100 *MHz* a segundos, utilizando esto para mantener tres contadores que llevan el paso de segundos, minutos y horas. Se ha implementado como una *timed-FSM*, similar a la implementada en 7. Se puede representar los estados de la máquina principal, como se puede ver en la Figura 13.

La máquina de estado secundaria, se encarga de mantener un registro de los contadores que llevan la cuenta de los segundos, minutos y horas. Dependiendo del estado en el que se encuentre la máquina principal, la máquina secundaria añade una unidad al contador respectivo, i.e. si la máquina principal está en el estados SEGS, la máquina secundaria añadiría una unidad al contador `count_segs`. Esta máquina también es encargada de recibir los pulsos provenientes de `BTNR/BTNL` y realizar los ajustes correspondientes a los contadores asociados, para poder registrar la configuración de hora que ingrese el usuario.

C. Bloque: ALARM

Este bloque corresponde a la alarma que se pide en los requerimientos del reloj. Su implementación sigue las misma línea que la máquina TICTOC al ser implementada como una *timed-FSM*, con algunas diferencias como se ve en la Figura 14. Cuando el usuario quiere *setear* la alarma, activa el switch `SW[1]`, lo que lleva a la alarma al estado `SET`, que permite responder a los cambios de minutos y hora accionado por los respectivos botones. Luego de que el usuario ha definido una hora para la alarma, se transiciona de vuelta a un estado de espera, `IDLE`, hasta que la hora reportada por el mecanismo `TICTOC` se equivalente a la guardada por la alarma, donde se transiciona hacia el estado del mismo nombre, que activa una señal para la animación correspondiente. La alarma, puede estar en este estado por un máximo de 5 *s*, por lo que cumplido este tiempo, se retorna a un estado de espera. Del mismo modo que la máquina `TICTOC`, se tiene una máquina secundaria, que dependiendo del estado de la máquina principal, hace las sumas a los registros correspondientes, para mantener la hora de la alarma, que se desea.

D. Bloque: AM/PM

Corresponde a un bloque combinacional accionado por `SW[0]`. Este bloque controla como su nombre lo indica el formato de la hora a mostrar. Su implementación consiste de un bloque mux, que determina si al registro de la hora, se le

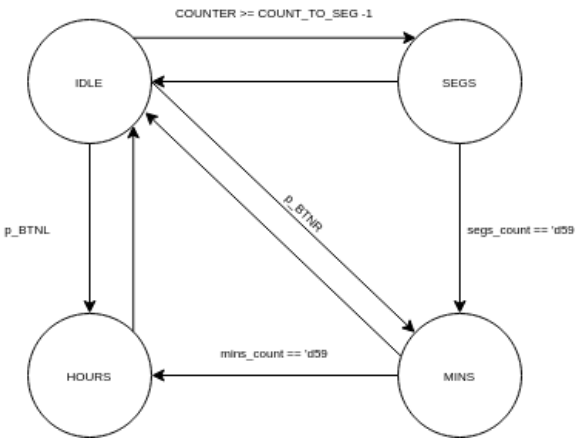

Fig. 13. Máquina de estados para el mecanismo del reloj

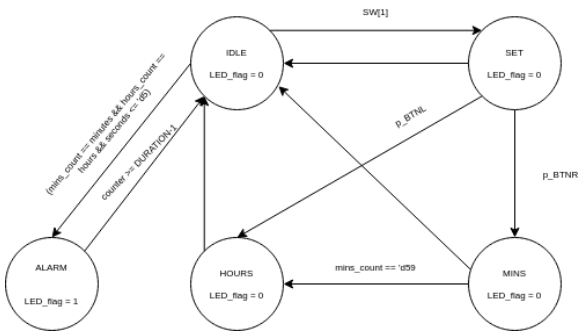

Fig. 14. Máquina de estados para el mecanismo de alarma

debe restar 12 horas, para llevarlo a formato de 12 horas o si el valor pasa directo, para formato 24 horas.

E. Bloque: BCD

Corresponde a un bloque que realiza la conversión de un número decimal sin signo a uno que representa su valor en un formato que puede ser enviado a un display de 7-segmentos. Para este módulo se reutilizó un módulo usado al momento de cursar el Laboratorio de Digitales.

F. Bloque: Display Driver

Corresponde a un bloque que recibe un número a mostrar en formato BCD, y realiza la multiplexación y el control temporal del display para representarlo. Para este módulo se reutilizó un módulo usado al momento de cursar el Laboratorio de Digitales.

G. Bloque: LED_DRIVER

Es un bloque, que dada una señal de activación, activa los LEDS[15:1], en una rutina de intermitencia de periodo 0.5 *s*. La señal de activación en este caso corresponde a la señal proveniente de la alarma.

H. Warnings no eliminados

1) Derivados de Síntesis: De la etapa de síntesis 15 warnings quedaron al momento de culminar el diseño del reloj, estos corresponden conexiones no utilizadas de bloques, como por ejemplo, de los debouncer, que poseen tres salidas, pero solo una se utilizó.

I. Derivado de implementación

No hay warnings derivados de esta etapa.

J. Reporte de uso de recursos lógicos

En la tabla IV, se detalla el uso de recursos lógicos para el reloj implementado.

K. Tamaño archivo configuración FPGA

El tamaño de este archivo corresponde a 3,8 *Mb*

TABLE IV
REPORTE DE UTILIZACIÓN DE RECURSOS PARA RELOJ DIGITAL

Recurso	Utilización	Disponible	Utilización %
LUT	311	63400	0.49
FF	359	126800	0.28
IO	37	210	17.62

VI. CONCLUSIONES

Al cierre de este trabajo, se puede afirmar que se ha logrado el objetivo principal del trabajo, retomar los conceptos medianamente olvidados del diseño e implementación de circuitos digitales, en un entorno de descripción de hardware. Por otro lado, se ha logrado también conocer nuevos aspectos de la herramienta que en un comienzo fueron obviados pero ahora toman interés como es el análisis de utilización de recursos, los circuitos generados, tanto a nivel representativo como en implementación del hardware específico. Es importante reconocer la importancia de las herramientas de simulación que entrega la plataforma, las cuales permiten realizar un trabajo de diseño más serio, sin depender tanto de la *prueba y error* directo en el hardware, una práctica díficil de dejar de lado.