Proyecto Integrador de Arquitectura y Ensamblador: Semáforo

Ariel Arévalo Alvarado*, Cheryl Bryden Watson†,
Génesis Herrera Knyght‡ y Yendry Jiménez Quesada§
E.C.C.I., Universidad de Costa Rica
San José, Costa Rica
*ariel.arevalo@ucr.ac.cr, †cheryl.bryden@ucr.ac.cr,
‡genesis.herreraknyght@ucr.ac.cr, §yendry.jimenezquesada@ucr.ac.cr

***Resumen*—En este proyecto se realiza la simulación de la funcionalidad de un sistema de semáforos de tránsito y peatonales. Para esto se establecen requisitos funcionales y de arquitectura mínimos para funcionamiento a un estándar realista. Con estos requisitos se lleva a cabo un diseño detallado de cada parte del sistema de control de tránsito para cumplir los mismos, y se diagrama tanto las conexiones entre estas partes, así como la lógica interna de su coordinación. Seguidamente, se implementa una simulación en código ensamblador de los componentes, la cual intenta conservar la mayor fidelidad posible al funcionamiento del circuito digital.**

I. Introducción

En este proyecto se realiza la simulación de la funcionalidad de los semáforos en un cruce de calles con pasos peatonales. Este semáforo controla el flujo de los vehículos que se dirigen a cuatro direcciones distintas, con giros a la izquierda para dos carriles y además el flujo de peatones para que estos puedan atravesar las calles de una forma segura. En Costa Rica, actualmente se tienen semáforos para controlar los cruces vehiculares donde la cantidad de vehículos es alta, y se incluyen semáforos peatonales, si al igual que con los vehículos, el flujo de peatones también es alto. Los semáforos modernos tienden a presentar lógica digital a nivel tanto de su control, así como dentro de los semáforos como tal.

II. Especificación

Semáforo

Un semáforo es un dispositivo de control y seguridad utilizado para la señalización en carreteras, permitiendo la regulación del paso de vehículos, bicicletas y transeúntes al intentar cruzar una carretera. Dependiendo del mecanismo de control de cada semáforo, la intención con la que fue colocado y el lugar de su colocación, pueden clasificarse en:

1. Semáforos para tránsito de vehículos
2. Semáforos para paso peatonal
3. Semáforos especiales

Para una intersección completa entre dos vías, con carril de giro a la izquierda en cada entrada a la intersección, se requiere un total de ocho semáforos de tránsito vehicular. En el caso de haber cruce peatonal en cada esquina, y en cada dirección, se requieren ocho semáforos peatonales. Estos semáforos reciben coordinación de un coordinador central para establecer un flujo razonable de peatones y vehículos que prevenga la colisión entre los mismos, asegurándose de alertar con antelación razonable a los conductores y peatones de sus cambios [1]. Además, el coordinador puede recibir señal de botones en las esquinas para priorizar una fase que permita el paso peatonal.

Requerimientos funcionales

1. Al determinar las fases dentro de un ciclo semafórico se debe intentar que su número sea el menor posible, para así reducir los tiempos perdidos en cada ciclo, así como que asegurar que el número de movimientos simultáneos –sin conflictos comunes– sea el máximo posible [2].
2. Los semáforos deben ser capaces de mostrar al menos una fase que permita cada tipo de movimiento posible desde cualquiera de las cuatro entradas a la intersección durante un ciclo.
3. El coordinador debe conocer el orden preestablecido de las fases de tránsito vehicular, así como debe ser lo suficientemente flexible para incorporar una fase de paso peatonal y vehicular simultáneo y seguro en su ciclo cuando un peatón lo solicite por botón en las esquinas. Además debe ser capaz de mantener cada fase por 30 segundos. Estas fases consisten en un patrón de cuáles luces deben estar en verde en una parte determinada del ciclo completo de tránsito.
4. Cada esquina debe tener dos botones, uno para cada dirección de cruce desde la misma. Esto resulta en ocho botones, que representan los cuatro cruces de vía diferentes.
5. A nivel de intersección, se debe evitar que ocurra la misma fase peatonal dos veces seguidas. Esta lógica se implementa a nivel del bus para los botones, en forma de un banderín de botón duplicado, y a nivel del coordinador, que revisa este banderín.
6. Cualquier semáforo en la intersección debe ser capaz de cumplir con un ciclo completo de cada una de sus luces, así como de cambiar a amarillo (o parpadear, en el caso de los peatonales) por tres segundos únicamente cuando es sujeto a cambio en la siguiente fase.
7. El coordinador debe ser capaz de comunicarse con cada semáforo de la intersección, y deberá ser capaz

de comunicarle a cada semáforo que es momento de cambiar de luz.

Requerimientos de arquitectura

1. Para el diseño del hardware se usará la herramienta Logisim y la implementación se simulará utilizando programación híbrida: un lenguaje de alto nivel (C/C++) y lenguaje ensamblador. Se simulará la interconexión del hardware y se trabajará la interfaz en el lenguaje C++. Se trabajarán los problemas que los componentes del hardware abordan con código en lenguaje ensamblado.
2. La funcionalidad de los botones para solicitar el paso de los peatones, así como la validación de la fase peatonal se programará en lenguaje ensamblador.
3. El manejo y almacenamiento de las fases, así como el cronometraje entre cada fase por parte del coordinador se programará en lenguaje ensamblador. Las fases se almacenan como una serie de enteros binarios de tres bits.
4. La lógica para controlar el estado del ciclo de luces individual que cada semáforo contiene será programada en lenguaje ensamblador.
5. La visualización en interfaz de los cambios de luces y de vías habilitadas, se trabajará en C++. Se espera poder llevar a cabo este proceso de forma simultánea con el del código en lenguaje ensamblador de forma nativa, pero en el caso donde esto no sea posible, se separará en dos secciones el código del coordinador para permitir la actualización de la interfaz, posibilitando así la simulación.
6. Como mecanismo de comunicación entre los botones y el coordinador existe un codificador 4-a-2, que traduce las cuatro direcciones de paso peatonal a un código de dos bits. Entre el coordinador y los semáforos existe un codificador 3-a-8, el cual convierte cada número entero representante de una fase (en binario) a una señal asincrónica, la cual pasa por un enrutador para aquellos de los 16 semáforos que deban cambiar sus luces para cada fase.
7. En práctica, se podría asumir el código del coordinador corriendo sobre un procesador con acceso a pines GPIO mapeados a direcciones en memoria, tomando como modelo el UP Board, con un chip Intel Atom y hasta 14 pines GPIO disponibles para uso general [3]. Sin embargo, como la implementación real de los semáforos revela, la lógica para coordinar un semáforo es lo suficientemente simple para ser implementada en un PLC a la medida [4].

III. Diseño

El diseño final del sistema de semáforo contempla tres fases principales, las cuales cumplen el propósito de entrada, coordinación, y salida, respectivamente.

Fase de entrada

1. Controlador de botones: La función principal del controlador es convertir el pulso inconstante por parte de un botón físico en una señal constante que indica alguno de los cruces peatonales. Cuando un botón, o su contra-parte del otro lado de la calle, es oprimido, el controlador se encarga de emitir, de una de sus cuatro salidas, una señal constante que indica cuál de los pares de botones ha sido activado. Para asegurarse de que solamente se emita de una de las cuatro salidas a la vez, el controlador además presenta funcionalidad de *switch*. Cuando un botón de un dado par es oprimido, el controlador interrumpe el flujo a las salidas indicadoras de los demás pares de botones.
2. Codificador de entrada: Conectado a la salida del controlador de botones hay un codificador 4-a-2. Este codificador convierte la señal de una de las salidas del controlador de botones a un código de dos bits, dependiendo de la posición de la entrada.
3. Validador de entrada: Conectado a la salida del codificador de entrada se encuentra un validador. Este se ocupa solamente de comparar la última entrada con la siguiente, señalando a través de una salida auxiliar al coordinador si las mismas son duplicadas. Además, el validador presenta una entrada auxiliar de parte del coordinador, la cual se utiliza para señalar al validador en qué momento debe sobre-escribir el valor en las salidas principales con el valor entrante, actuando como una línea de escritura.

Fase de coordinación

La fase de coordinación comprende el trabajo llevado a cabo por el coordinador. El coordinador mantiene el orden de las fases en el ciclo, así como almacena la fase actual del ciclo. Además, lleva a cabo la selección de la próxima fase basado en esto anterior, así como el estado del validador de entrada. El coordinador se encarga, por otra parte, de tomar el tiempo adecuado entre cada parte del ciclo de las luces del semáforo. La forma en que el coordinador ejecuta las fases es emitiendo un pulso de tres bits como salida cada vez que los semáforos de una fase deben cambiar de una luz a la próxima. El coordinador lleva acabo la mitad final del ciclo de luces para una fase, e inicia el ciclo de luces para la próxima. Antes de iniciar la mitad final del ciclo actual, revisa las condiciones para decidir la próxima fase.

Fase de salida

1. Decodificador de salida: Conectado a la salida del coordinador hay un decodificador 3-a-8. Este decodificador recibe un pulso en una línea de escritura, el cual es dirigido de acuerdo a una entrada de tres bits que representa una de las ocho fases del sistema de semáforos en binario. El pulso es emitido por alguna de las ocho salidas del decodificador, con cada posición equivalente a una fase del sistema de semáforos.
2. Enrutador de salida: Conectado a la salida del decodificador se encuentra un enrutador de salida. El enrutador implementa la lógica pre-escrita que decide cuáles semáforos le pertenecen a cada fase. De esta forma, el

enrutador permite que el pulso del decodificador incida sobre los semáforos que deben de avanzar en su ciclo de luces. La ruta específica para cada fase se escoge con el afán de seguir de cerca las leyes de tránsito reales hasta donde las limitaciones del hardware escogido lo permita, aunque cabe destacar que cuál ruta se asigna a cuál fase es básicamente arbitrario, y es decidido por convención.

3. Semáforos: Finalmente, el pulso emitido por el enrutador alcanza aquellos semáforos que son parte de la fase que se está llevando a cabo. Los semáforos cambian de luz cada vez que reciben un pulso, llevando a cabo de esta forma su ciclo de luces. La única distinción entre los semáforos peatonales y lo semáforos de tránsito, a nivel de diseño, es la ausencia de amarillo en los semáforos peatonales. Los semáforos peatonales presentan, en ausencia de una luz amarilla, una fase de luz verde parpadeante, la cual alerta a los peatones de que se agota el tiempo para cruzar la calle.

IV. Implementación

El objetivo principal para la implementación fue el de conservar la mayor fidelidad posible entre el funcionamiento del circuito digital, y el funcionamiento del código en ensamblador. La totalidad de los componentes descritos en estas fases son programados en lenguaje ensamblador, salvo los semáforos, los cuales son simulados por parte de la interfaz gráfica. El uso de lenguaje de alto nivel en lo que respecta a los componentes se limita a la integración de los mismos entre sí, integración necesaria ya que no es posible correr múltiples instancias de lenguaje ensamblador a nivel de un solo programa serial, y el diseño de un programa paralelo sería un esfuerzo desproporcionado para los propósitos de este proyecto. Además se utiliza lenguaje de alto nivel para hacer una integración similar con la interfaz, la cual solamente reacciona ante métodos de alto nivel en su implementación actual, métodos que un programa de lenguaje ensamblador no puede fácilmente ejecutar, ya que son dependientes de la instancia del objeto de la interfaz.

Interfaz

La interfaz implementada corre a base del software para interfaces gráficas en C++ conocido como *QT*. Más allá de su simple diseño gráfico, la interfaz es un elemento que cambia de una fase a otra al llamarse un método público suyo que permite hacer ciclo de la fase dada como parámetro. Basado en este parámetro, y en un contador interno de estado, la interfaz llama el método interno indicado para cada estado de cada fase del ciclo completo de la intersección según corresponda. Además, la interfaz presenta pares de botones para cada dirección de cruce peatonal, los cuales llaman un método privado que actualiza un valor interno de la interfaz que marca cuál fue el último botón oprimido. A pesar de que el circuito de semáforo cuenta con lógica para evitar sobre-escritura del cruce en cola por otro botón peatonal, es necesario además implementar esta lógica a nivel de interfaz, dado el hecho de que la interfaz no entrega el valor escogido inmediatamente al circuito de semáforo para ser procesado, sino que, debido a la falta de genuina concurrencia, debe almacenar este valor hasta que el circuito se lo solicite, y debe asegurarse que no sea sobre-escrito mientras esto suceda.

Componentes

A nivel de la implementación de los componentes, la intención es tener pre-establecidas ubicaciones de memoria que representan las entradas y salidas de los componentes electrónicos simulados. Se simula una ejecución de uno de estos componentes, y esto modifica la dirección de memoria que representa el estado de sus salidas. Se interrumpe el flujo del programa en lenguaje ensamblador para así actualizar la interfaz de acuerdo al valor devuelto por el programa en lenguaje ensamblador. Este paradigma se escogió tanto para mantener fidelidad con el circuito digital, como para facilitar el desarrollo, permitiendo que los componentes sean bastante desacoplados entre sí, y fuertemente acoplados a la implementación de la simulación en sí. Además, permite que la interfaz sea desacoplada del circuito, con la única comunicación entre el circuito y la interfaz ocurriendo a través del main.

1. Controlador de botones: El controlador se ve reducido, en esta fase, a un programa que al correr revisa si uno de cuatro bytes en un arreglo en memoria es diferente a cero. De ser así, cambia todos los valores en un arreglo de salida a cero, y luego cambia el valor en el arreglo de salida de misma posición que el valor que es diferente a cero en el arreglo de entrada. Luego, cambia todos los valores del arreglo de entrada a cero.
2. Codificadores: Los codificadores, dependiendo de su dirección, convierten a/desde una dirección de byte única en memoria, que representa un valor único binario, desde/a un arreglo de bytes en memoria de tamaño k, para representar los k diferentes estados de entrada/salida.
3. Validador: La entrada del validador consiste en un programa que compara el valor en un arreglo de entrada de dos bytes en memoria con el valor en un arreglo de salida de dos bytes en memoria. Si los valores son iguales, cambia una dirección de byte en memoria *dupl* a uno, si no son iguales, la cambia a cero. La salida del validador consiste en un programa que prueba si el valor del banderín de escritura *write* es diferente de cero. Si es diferente de cero, escribe los valores del arreglo de entrada al arreglo de salida.
4. Coordinador: El coordinador preserva su lógica descrita/diagramada.
5. Enrutador: El enrutador consiste en un programa que prueba si algún elemento del arreglo de entrada es diferente de cero. Al encontrar el primer elemento diferente de cero, escribe la posición del elemento a la dirección de salida.
6. Semáforos: La funcionalidad de los semáforos es absorbida por la interfaz gráfica.

Como se puede apreciar, se mantienen varias ineficiencias en la implementación de los componentes con la única motivación de asegurar un funcionamiento que emule el circuito digital de su diseño lo más cercanamente posible.

Integración

La forma en que se llevó a cabo el proceso del main a manera de integrar estos dos componentes no obedece los patrones recomendados al trabajar con *QT*. Esto debido al costo de desarrollar una solución apropiadamente concurrente, como lo pediría *QT* para estos casos. Se intentó simular el funcionamiento de un programa concurrente a través del almacenamiento del botón oprimido a nivel de la interfaz, y a través de un ciclo con quiebres en la lógica entre las esperas del coordinador, permitiéndole así a la interfaz actualizarse. Si en cualquiera de estos quiebres, se detecta que un botón fue oprimido en los momentos anteriores, se hace la emisión al circuito, durante el quiebre, de este evento, usando un método que revela el circuito para la entrada apropiada. Al final de cada quiebre, se toma la ruta que actualmente presenta el coordinador como salida, y se usa para avanzar el estado de los semáforos de la interfaz.

Pruebas y Mejoras

La entrega inicial de la implementación fue una de éxito en cuanto al cumplir con los lineamientos impuestos por la especificación inicial del proyecto. Sin embargo, se esperó proceder con las mejoras que se pudieran lograr en el último lapso del período del proyecto, así como con las pruebas formales de cada componente de la interfaz.

Pruebas

Una intersección de semáforos presenta baja complejidad debido a su baja cantidad de entradas posibles por probar en los diferentes escenarios que presenta. Dado que el semáforo opera casi totalmente como una máquina de estados, la interacción entre los componentes de entrada es casi nula, por lo que sólo fue necesario probar cada componente en cada estado, inicialmente. La forma en que se llevó a cabo las pruebas consistió en la activación de cada uno de los botones durante cada uno de los posibles estados del ciclo del semáforo, cerciorándose de que se activara la fase apropiada para cada posibilidad. La única interacción existente es entre un mismo semáforo peatonal, y el ciclo. Para cada botón del semáforo peatonal, se confirmó que el oprimir el botón dos veces dentro de un mismo ciclo de la intersección no produjera una activación duplicada inmediatamente, sino que esta sólo ocurriera hasta que hubiera transcurrido un ciclo completo. Esto produce en los resultados un línea de fallos que son más bien el resultado deseado. Los resultados completos de las pruebas llevadas a cabo se puede apreciar en el cuadro I.

Cuadro I
Prueba de inducción de su fase para cada botón y fase

	Botón							
Fase activa	0	1	2	3	4	5	6	7
0	✓	✓	✓	✓	✓	✓	✓	✓
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	-	-	✓	✓	✓	✓	✓	✓
4	✓	✓	-	-	✓	✓	✓	✓
5	✓	✓	✓	✓	-	-	✓	✓
6	✓	✓	✓	✓	✓	✓	-	-

Los botones se encuentran numerados en orden de las manecillas del reloj, iniciando en el semáforo izquierdo del cruce superior. El comportamiento en las fases peatonales es deseado, ya que la fase presente es la inducida.

Mejoras

Hubo varias mejoras que se quisieron implementar como forma de aprovechar el tiempo antes de esta última presentación.

Las mejoras que se planearon implementar fueron las siguientes:

- Agregar parpadeo a los semáforos peatonales
- Remover luces en flechas de llegada por distracción
- Encontrar solución al estado de ”No responde”durante la espera del semáforo
- Mejorar la estética de la interfaz

De estas mejoras propuestas, se logró completar la remoción de las luces fuera de lugar y la mejora de la calidad estética de la interfaz. El intento por resolver el estado ocasional de falta de respuesta de parte del programa no tuvo éxito. La ausencia de respuesta inmediata debido a la espera estilo *busy-wait* por parte del programa es una necesidad provocada por el deseo de programar la espera en lenguaje ensamblador en *Windows*. La ausencia de alguna herramienta para llevar a cabo una espera programáticamente adecuada en *Windows* nos lleva a utilizar una espera definida por la ausencia de respuesta en sí que caracteriza un programa atrapado en un número extremadamente alto de ciclos. Esta espera es una pieza fundamental del funcionamiento de nuestro semáforo, y no se logró hallar una forma de sobrepasar este obstáculo sin deconstruir gran parte del trabajo necesario para una entrega funcional. En cuanto al agregado de parpadeo para los semáforos peatonales, esto también falló. Aunque el *.gif* que se previó para representar la luz parpadeante del semáforo peatonal funciona por sí solo, a la hora de incrustarlo en la interfaz gráfica el *.gif* se queda atascado en su primer fotograma, probablemente debido a la *busy-wait* previamente mencionada.

Referencias

[1] C. Mcshane, “The origins and globalization of Traffic Control Signals,” *Journal of Urban History*, vol. 25, no. 3, pp. 379–404, Mar. 1999.

[2] P. Koonce, ”Traffic Signal Timing Manual”, U.S. Department of Transportation Federal Highway Administration, Portland, OR, FHWA-HOP-08-024, Mar, 2008.

[3] “UP Board Series,” *UP Bridge the Gap*, 2022. [Online]. Available: https://up-board.org/up/specifications/. [Accessed: 18-Apr-2022].

[4] ”Traffic Signals 101”, Office of Traffic, Safety & Technology, St. Paul, MN: Minnesota Dept. of Transportation, Jan, 2018.