PRÁCTICA N°0: MÁQUINAS DE ESTADO FINITO USANDO ARDUINO

Presentado por:
SEBASTIÁN TOBÓN ECHAVARRÍA
CRISTIAN LOPERA TRUJILLO
JOSE MANUEL MACHADO LOAIZA

PROYECTO INSTRUMENTAL II

Presentado a:
AUGUSTO CARMONA VALENCIA

INGENIERÍA FÍSICA
DEPARTAMENTO DE CIENCIAS FÍSICAS
ESCUELA DE CIENCIAS
UNIVERSIDAD EAFIT
MEDELLÍN
2020

CONTENIDO

LISTA DE FIGURAS

INTRODUCCIÓN

Una Máquina de Estado Finito (Finite State Machine), llamada también Autómata Finito, es una abstracción computacional que describe el comportamiento de un sistema reactivo mediante un número determinado de estados y un número determinado de transiciones entre dicho estados. Dicho modelo computacional es utilizado para la simulación de lógica secuencial en áreas afines a la electrónica digital, matemáticas discretas, inteligencia artificial, etc.

Cualquier dispositivo que almacena el estado de algo en un momento dado puede ser modelado por una Máquina de Estado Finito. El estado cambiará en función de las entradas, proporcionando la salida resultante para los cambios implementados. Las FSM son de gran utilidad debido a que se puede tener un conjunto fijo de estados en los que la máquina puede estar, cada estado tiene un conjunto de transiciones y cada transición está asociada con una entrada y apunta a un estado.

En el presente informe, se muestra el desarrollo de la realización de un programar mediante MEF en un sistema embebido para el control de un semáforo junto con las metodologías asociadas al autómata finito [1].

1. OBJETIVOS

- Desarrollar una Máquina de Estados Finitos para el control de un semáforo con control peatonal especificando los requerimientos de un problema de automatización.
- Analizar el funcionamiento de estructuras funcionales en un sistema embebido junto con los tipos de variables estáticas y constantes.
- Representar gráficamente el funcionamiento del dispositivo empleando diagramas de estado.
- Obtener la respuesta frecuencial de la planta de primer orden de un circuito RC de manera experimental.

2. MARCO TEÓRICO

Las Máquinas de estados finitos (conocidas como Finite State Machines) nos sirven para realizar procesos bien definidos en un tiempo discreto. Reciben una entrada, hacen un proceso y nos entregan una salida.

En otras palabras, es una máquina capaz de seguir una secuencia finita de pasos al introducir un conjunto de datos en ella, que solamente se puede leer un dato en cada paso que se realice, por tanto el número de pasos a seguir está dado por el número de datos a introducir. Cada entrada diferente genera una salida diferente, pero siempre el mismo resultado con los mismos datos de entrada. Por lo tanto una computación es capaz de resolver un problema, sí y solo sí tiene una solución algorítmica, es decir, puede ser descrito mediante una secuencia finita de pasos bien definidos [2].

Formalmente, podemos definir un autómata finito como un conjunto

$$A = \{Q, q_0, F, \Sigma, \delta\}$$

donde: Q : Conjunto finito de estados.

q_0 : Estado inicial donde $q_0 \in Q$. Debe haber uno y sólo un estado inicial.

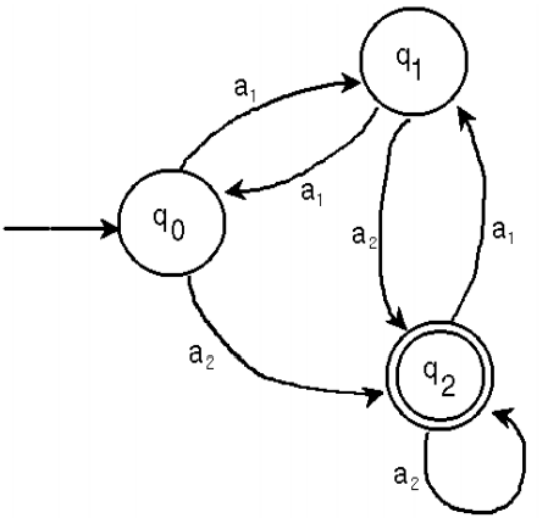
F : Conjunto de estados finales $F \subseteq Q$. El estado q_0 también puede ser final.

$\sum$: Alfabeto finito de entrada.

δ : Función de Transición $Q \times \Sigma \to Q$

Mediante un *diagrama de estados*, o también llamado diagrama de transición, podemos representar una máquina de estados finitos. Dicho diagrama muestra qué estado se moverá un autómata finito dado, basándose en el estado actual y otras entradas. Dicho en otras palabras, es una representación gráfica en la cual algunas de las entradas son el estado actual, y las salidas incluyen el siguiente estado, junto con otras salidas. En la **Figura 1** se muestra el diagrama de transición para una máquina de estados finitos

Figura 1. Diagrama de estados.



3. PROCEDIMIENTO

Los códigos del semáforo a realizar e implementar usando el sistema Arduino deben cumplir con los siguientes requerimientos [1]:

- La duración máxima permitida para la luz verde es de 90 segundos, antes de cambiar el estado del semáforo a naranja, el led debe titilar 5 veces con intervalos entre encendido y apagado de 1 segundo (todo esto incluido en los 90 segundos). Claramente si este led está encendido el led rojo del peatón estará encendido.
- El semáforo pasa a estado naranja después del proceso previamente descrito para el led verde, en este estado tendrá una duración de 5 segundos, para luego pasar a estado rojo.
- En estado rojo el semáforo tardará 20 segundos, en los cuales el peatonal estará en verde y en ese mismo tiempo debe titilar en sus últimos 5 segundos, 5 veces para luego pasar a rojo.
- Para pasar a verde el vehicular deberá haber pasado al menos 2 segundos desde el instante en que el peatonal haya cambiado a rojo.

- Existe un pulsador que el peatón puede oprimir para acelerar el cambio del semáforo a estado rojo. Si éste se oprime por más de 5 veces, el estado del semáforo debe pasar a verde titilando.

Para el desarrollo del laboratorio se realiza el montaje como se muestra en la **Figura 2** que finalmente se implementa en una Protoboard como se expone en la **Figura 3**

Figura 2. Esquema de conexión.

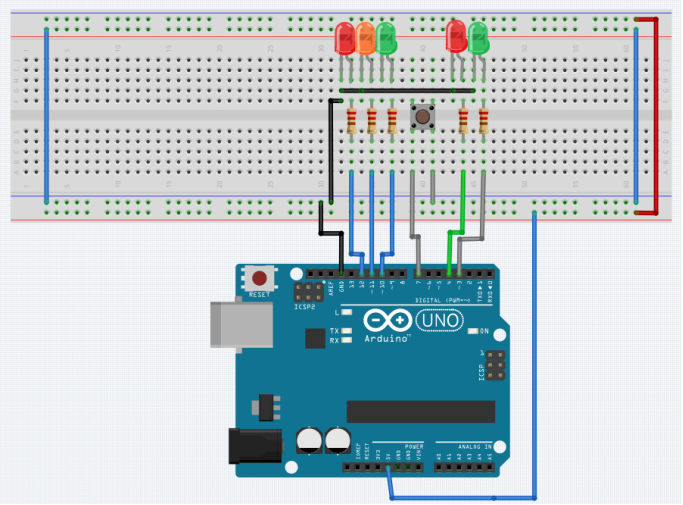
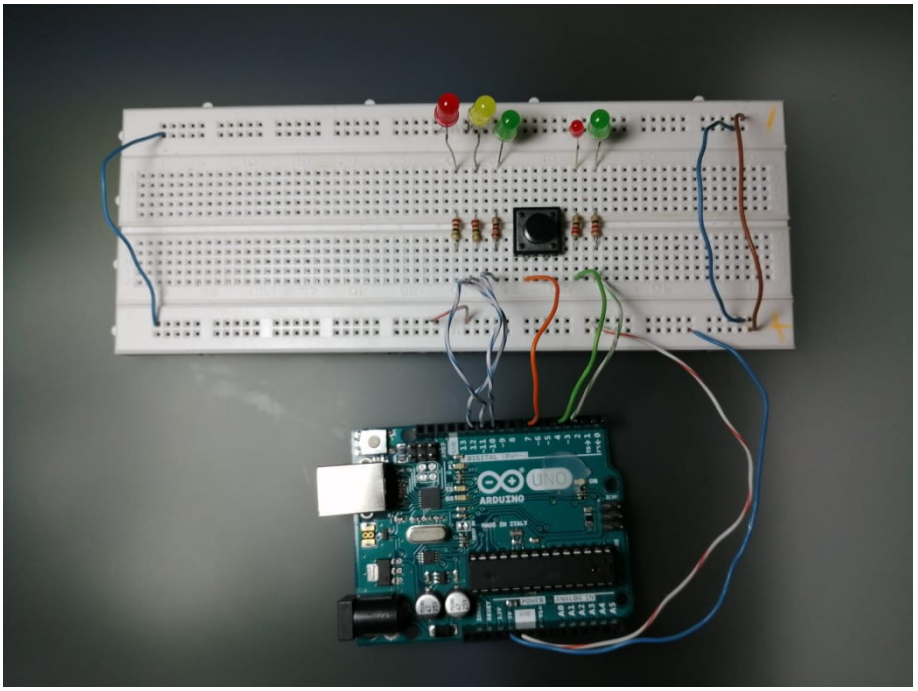


Figura 3. Montaje experimental.

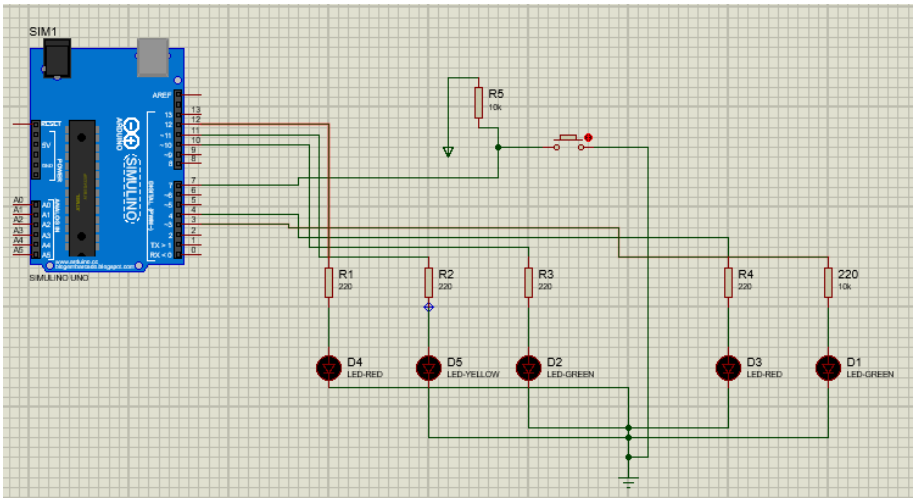


4. RESULTADOS Y ANÁLISIS

Adjunto a este informe se encuentran los códigos para la realización del laboratorio en un repositorio en GitHub. Dichos códigos están debidamente comentados y resuelven las tareas del autómata descrito anteriormente.

Dichos archivos incluye el "método de fuerza bruta"junto con la implementación de la MEF. Para simular la ejecución de dichos códigos se utilizo 'Simolino' en el sofware Proteus, permitiendo así cargar el programa y ver la ejecución del código. Para ello se utilizó una montaje en Proteus como se muestra en la **Figura 4**

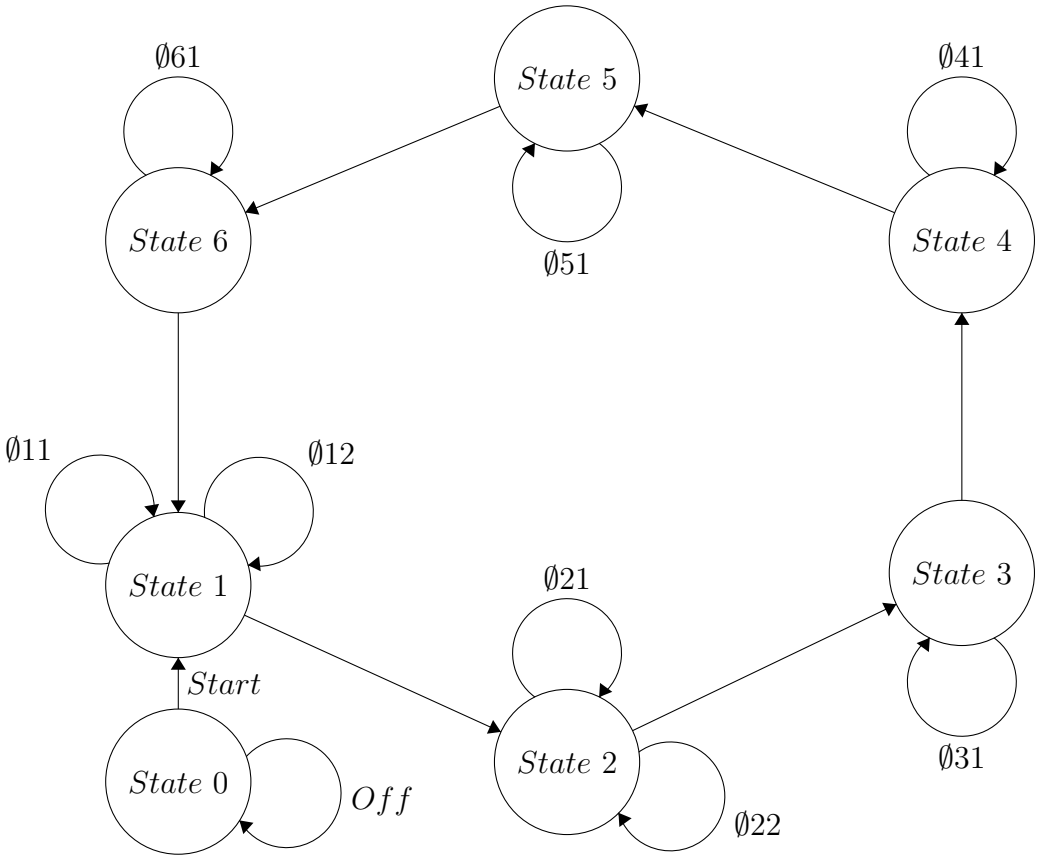
Figura 4. Simulación en Proteus.



Igualmente, el repositorio cuenta con un video del montaje experimental cuando es ejecutado el programa del autómata lo que evidenciando así la funcionalidad del programa.

4.1. Diagrama de estados Para la realización del diagrama de estados, se utiliza la herramienta *Finite State Machine Designer* disponible en $https://www.cs.unc.edu/\ otternes/comp455/fsm_designer/$. Esta estrategia permite tener una mayor claridad a la hora de implementar el código del autómata.

A continuación, se expone el diagrama de estados correspondiente a los requerimientos paran el código implementado del autómata.



Dicho diagrama se caracteriza por seguir el orden de los requerimientos para realizar la Máquina de Estados Finitos correspondiente al sistema en cuestión. Para lograr una mayor comprensión del diagrama mostrado con anterioridad, se propone una tabla explicativa para cada uno de los estados, representado como cada nodo en el diagrama; las condiciones y las transiciones que componen el diagrama.

Tabla 1. Definición de los estados del autómata

Número de estado	**Condiciones**
State 0	Off: apagado Start: se compila el código y se sube al Arduino
State 1	$\emptyset 11 : tiempo - ejecucion < 80$ $\emptyset 12 : numero - pulsaciones < 6$
State 2	$\emptyset 21 : 80s <= tiempo - ejecucion < 90s$ $\emptyset 22 : numero - pulsaciones >= 6$
State 3	$\emptyset 31 : tiempo - ejecucion < 95s$
State 4	$\emptyset 41 : tiempo - ejecucion < 115s$
State 5	$\emptyset 51 : tiempo - ejecucion < 120s$
State 6	$\emptyset 61 : 80s <= tiempo - ejecucion < 90s$

4.2. Declaraciones funcionales en lenguaje C++ Las variables tipo *static* son utilizadas para crear variables que son visibles únicamente para una función. Sin embargo, a diferencia de las variables locales que se crean y destruyen cada vez que se llama a una función, las variables estáticas persisten más allá de la llamada a la función, preservando sus datos entre las llamadas a la función. En resumen, las variables declaradas como estáticas solo se crearán e inicializarán la primera vez que se llame a una función [3].

Por otro lado, las variables tipo *const* son un un calificador de variable que modifica el comportamiento de la variable, convirtiendo una variable en "solo lectura". Esto significa que la variable se puede usar como cualquier otra variable de su tipo, pero su valor no se puede cambiar. Dichas constantes obedecen las reglas de alcance de variables que rigen otras variables. Esto, y las dificultades de usar define, hacen que la palabra clave *const* sea un método superior para definir constantes y se prefiere al uso de define.

De este modo, podemos concluir que las variables *static* y *const* son de suma utilidad para evitar la construcción repetida de distintos temporarios en cada punto de uso. Además, se corroboró a través de un multímetro digital que dichas variables reducen los tiempos y el consumo energético del sistema implementado. Igualmente, existen otras variables como lo es *define* que permite el sistema de .Cpp para çuidar"sus variables. Esta declaración permite dar un valor constante a una asignación previo a la compilación. Además, la variable *scope* es útil cuando un programa resulta muy extenso y tiene un grado de complejidad alto dado que es una variable local (su utilidad radica en que asegura que sólo una función tenga acceso a sus propias variables. Esto evita errores de programación cuando una función modifica involuntariamente las variables utilizadas por otra función) [4] .

5. CONCLUSIONES

- A través del Método de Estados Finitos (MEF) se puede obtener un código más depurado y organizado permitiendo así la optimización de harware de los sistemas embebidos y el consumo energético del sistema.
- Se comprobó experimentalmente que las simulaciones realizadas en Proteus corresponden a la correcta funcionalidad del código.
- A través de los diagramas de estado se puede visualizar la funcionalidad del código implementado mostrando los estados y las transiciones que el sistema genera.

Referencias

[1] P. I. . E. D. ESCUELA DE CIENCIAS, DEPARTAMENTO DE CIENCIAS FÍSICAS, "Máquinas de Estado Finito usando Arduino," 2020.

[2] J. A. Gutierrez, "Máquinas de Estados Finítos." http://delta.cs.cinvestav.mx/ mcintosh/cellularutomata/, 2008.

[3] Arduino., "Variable Scope Qualifiers." https://www.arduino.cc/reference/en/language /variables/variable-scope–qualifiers/static/, 2019.

[4] Arduino., "Variable Scope Qualifiers." https://www.arduino.cc/reference/en/language/ variables/variable-scope–qualifiers/scope/, 2019.