Diseño e implementación de una caja musical programable en FPGA

Mora Nicolas, Rufiner Santiago
FICH-UNL. Nicolas2002Mora@gmail.com , santiagorufiner@gmail.com

Resumen — En este trabajo se diseñó, codificó, simuló e impactó una "caja musical" que sea capaz de reproducir la melodía de "feliz cumpleaños", mediante la integración de distintos módulos que realizan tareas específicas. La codificación se realizó en el lenguaje de descripción de hardware Verilog, mientras que para impactar los resultados y poder visualizarlos se utilizó una FPGA (Field Programmable Gate Array) llamada EDU-CIAA-FPGA, equipada con el microcontrolador ICE40HX4K-TQ144, donde se conectó un parlante con una resistencia en uno de sus pines para poder escuchar la melodía. La implementación final demostró que el sistema propuesto es capaz de sintetizar y reproducir con precisión la melodía propuesta, validando así la efectividad del diseño basado en Verilog.

Palabras clave: FPGA, HDL, Verilog, microcontrolador, caja musical.

Introducción

En el ámbito de la electrónica digital, el diseño y la implementación de sistemas complejos requieren herramientas y tecnologías avanzadas que permitan describir y configurar el comportamiento de hardware. Dos de estas herramientas fundamentales son las matrices de puertas lógicas programables en campo (FPGA, por sus siglas en inglés) y los lenguajes de descripción de hardware (HDL, por sus siglas en inglés).

Una FPGA (Field-Programmable Gate Array) es un dispositivo semiconductor digital que puede programarse luego de su fabricación para realizar una amplia gama de funciones lógicas. Las FPGAs son populares debido a su flexibilidad y capacidad para ser reconfiguradas en campo, es decir, después de haber sido desplegadas en un sistema, en contraposición a los circuitos integrados ordinarios que están limitados a una función específica impuesta por su diseño de fábrica. Las FPGAs se componen de:

- **Lógica Configurable:** Bloques lógicos básicos que se pueden interconectar de diferentes maneras para formar circuitos complejos.
- **Interconexiones Programables:** Redes de conexiones que permiten la comunicación entre los bloques lógicos configurables.
- **Memoria Integrada:** Bloques de memoria que pueden ser utilizados para almacenar datos y estados durante el procesamiento.
- **Interfaces de Entrada/Salida:** Permiten la interacción con el mundo exterior, conectando la FPGA con otros componentes y sistemas.

Para este trabajo se utilizó una FPGA llamada EDU-CIAA-FPGA [1], diseñada por el Grupo de Investigación y Desarrollo de Sistemas Embebidos (ASE) de la UTN FR Haedo, orientada a la enseñanza en nivel secundario, terciario y universitario, y diseñada con tres objetivos fundamentales en mente: bajo costo, flexibilidad y facilidad de uso.

La placa se basa en una FPGA Lattice iCE40 HX4K (equipada con un microcontrolador ICE40HX4K-TQ144) a la cual se agregaron los mínimos componentes necesarios para su funcionamiento y fácil utilización:

1. Memoria flash para almacenar la configuración de la placa;
2. Interfaz FTDI para comunicar la placa con la PC mediante USB;
3. Pulsadores y LEDs incorporados para probar ejemplos sencillos sin necesidad de componentes externos.

La Figura 1 muestra un diagrama en bloques simplificado y la forma en que se interconectan varios de los componentes de la placa de desarrollo, y la Figura 2 muestra la EDU-CIAA-FPGA física.

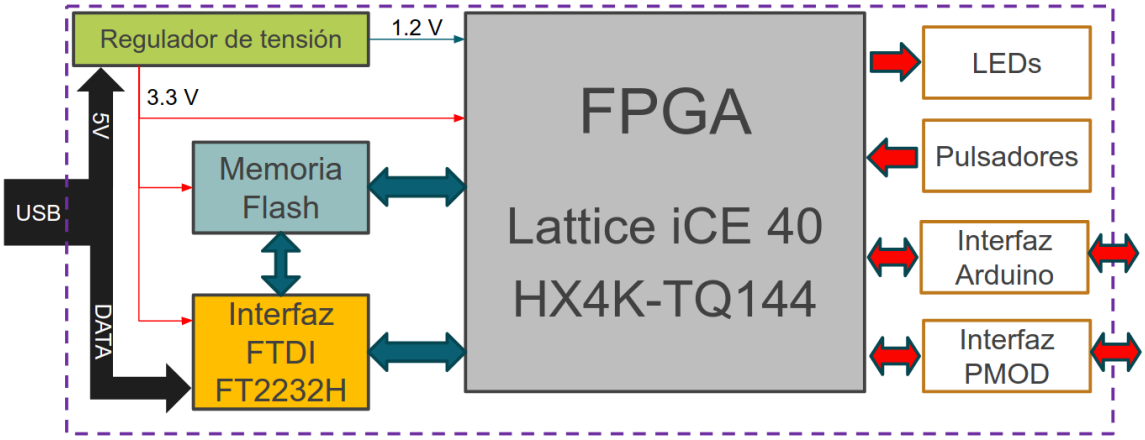

Fig. 1: Diagrama en bloques de los componentes de la EDU-CIAA-FPGA.

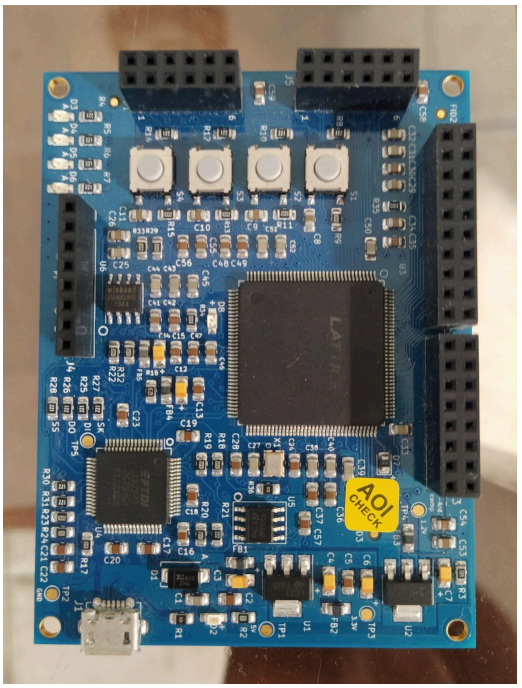

Fig. 2: Imagen de la EDU-CIAA-FPGA física.

Por otro lado, HDL (Hardware Description Language) es un lenguaje de modelado especializado utilizado para describir el comportamiento, la estructura y la organización de circuitos electrónicos digitales [2]. A diferencia de los lenguajes de programación tradicionales que se utilizan para escribir software que corre en un procesador, los HDL se utilizan para modelar circuitos electrónicos que se implementarán en hardware físico, como en FPGAs, ASICs (Application-Specific Integrated Circuits) y otros dispositivos lógicos programables.

Para este trabajo se utilizó el lenguaje de descripción de hardware Verilog, que es uno de los lenguajes más utilizados en la industria y en el ámbito académico para el diseño de sistemas digitales. Fue desarrollado originalmente en la década de 1980 y se ha convertido en un estándar debido a su capacidad para describir sistemas desde niveles muy abstractos hasta niveles muy detallados, permitiendo a los ingenieros definir el comportamiento y la estructura de circuitos digitales en una forma textual que se puede simular y sintetizar en hardware físico.

El proceso de diseño para una FPGA implica escribir código en un lenguaje HDL como Verilog, simular el diseño para verificar su funcionalidad mediante un testbench y visualizar el mismo por medio de un visor de ondas como GTKWave, para luego sintetizar el diseño en una configuración que la FPGA puede implementar físicamente.

El objetivo de este Trabajo Final Integrador (TFI) es poner en práctica los conocimientos abordados a lo largo de la asignatura de Electrónica Digital para implementar una "caja musical" que sea capaz de reproducir la melodía del “feliz cumpleaños".

Diseño

Para resolver el enunciado se nos brindó como información las frecuencias de las notas musicales en hercios (Hz) (ver Fig. 3) y la secuencia a seguir para reproducir la melodía del “feliz cumpleaños” (ver Fig. 4). Esta melodía se reproduce por medio de un parlante (mini speaker) conectado a uno de los pines de la FPGA, con una resistencia conectada en otro de sus pines. Además, para que el parlante conectado a la FPGA emita sonido, debe ser capaz de generar una señal cuadrada de la frecuencia determinada por cada nota. Este parlante debe estar conectado en el PIN 107, mientras que el clock de la FPGA está disponible en el PIN 94 y su frecuencia es de 12 Mhz. Adicionalmente, se implementaron controles para la gestión de la melodía: el PIN 33 se utilizó para la función de reinicio (reset), permitiendo reiniciar la melodía desde el principio, y el PIN 34 se asignó para la función de detener (stop), que permite finalizar la reproducción de la melodía sin necesidad de interrumpir el suministro de energía a la FPGA. Estos pines y su distribución se ilustran en la Figura 5.

+Do	+C	523.25 Hz
Do	C	261.63 Hz
Re	D	293.66 Hz
Mi	E	329.63 Hz
Fa	F	349.23 Hz
Sol	G	392.00 Hz
La	A	440.00 Hz
Si	B	493.88 Hz
La#	A#	466.16 Hz

Fig. 3: Tabla con las frecuencias correspondientes a cada nota musical.

C C D C F E
C C D C G F
C C +C A F E D
A# A# A F G F

Fig. 4: Secuencia de notas de la melodía del “feliz cumpleaños”.

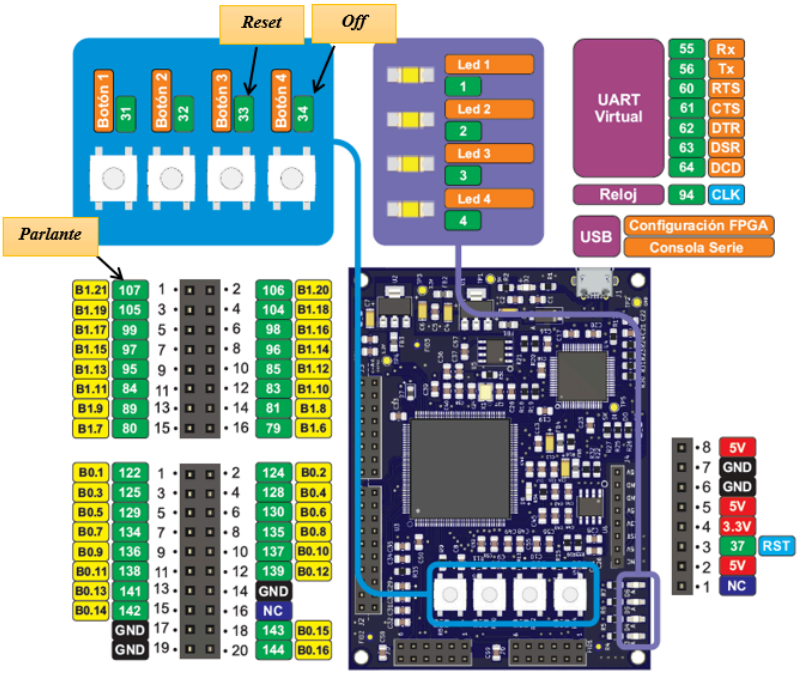

Fig. 5: Asignación de pines en la EDU-CIAA-FPGA.

En el diagrama de bloques mostrado en la Figura 6, se presenta el diseño de la caja musical, el cual se basa en la modularización del programa por medio de la implementación de distintos módulos que ejecutan tareas específicas. El diseño consta de tres módulos principales:

1. **Generador de Onda Cuadrada (‘square_wave_gen’):** Este módulo se encarga de convertir la frecuencia de la nota en una señal de onda cuadrada.

2. **Secuenciador de Notas (‘secuencia_notas’):** Este módulo define las frecuencias de las notas musicales y establece la secuencia necesaria para reproducir la melodía deseada.

3. **Módulo Principal (‘caja_musical’):** Este módulo es el que controla la reproducción de la melodía. Utiliza un contador para avanzar a través de la secuencia de notas y proporciona el índice actual al módulo de ‘secuencia_notas’, que devuelve la frecuencia de la nota correspondiente. Esta frecuencia se envía al módulo de ‘square_wave_gen’, que genera una onda cuadrada modulada a esa frecuencia. Esta onda cuadrada es luego enviada directamente al parlante, produciendo la melodía deseada. El módulo principal también maneja la pausa y la reanudación de la melodía en respuesta a las señales de control de los botones ‘reset’ y ‘stop’.

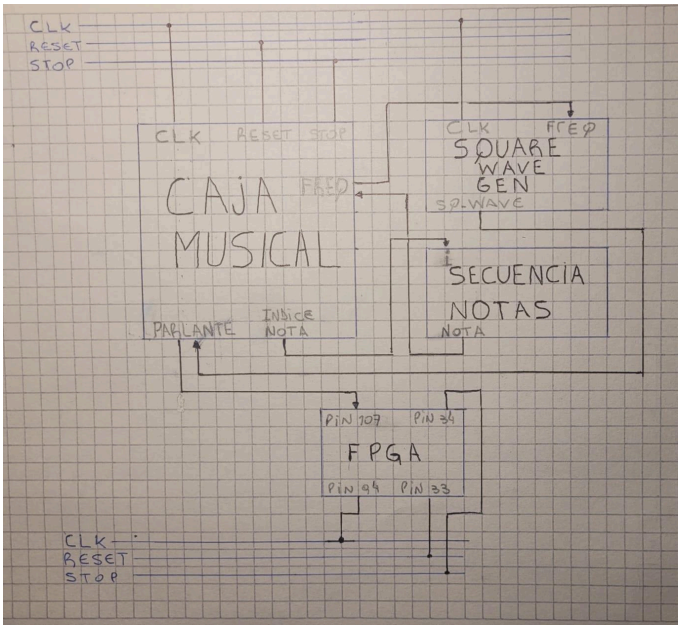

Fig. 6: Diagrama de bloques donde se representa el diseño de la caja musical.

Codificación

Generador de onda cuadrada (‘square_wave_gen’)

Al comenzar el desarrollo, inicialmente comenzamos con este módulo, que resultó ser quizás el que menos teníamos en claro sobre cómo llevarlo adelante, por lo que decidimos buscar documentación sobre la teoría detrás de la generación de onda cuadrada y cómo implementar un generador en Verilog [3]. Básicamente, lo que hace este generador es implementar un divisor de reloj para dividir la frecuencia del reloj de la FPGA, utilizando un contador para contar los pulsos de reloj entrantes y alternar el estado de la salida cuando el número de pulsos alcance un conteo específico (en nuestro caso, la frecuencia de la nota deseada). El estado de salida ‘state’ se asigna de manera continua a la onda cuadrada ‘sq_wave’, produciendo la señal que se enviará al parlante. La lógica del divisor de reloj se puede ver en la Figura 7.

```
15  always @(posedge clk) begin
16      if (cont >= freq) begin
17          cont <= 0;
18          state <= ~state;
19      end
20      else begin
21          cont <= cont + 1;
22      end
23  end
24
25  assign sq_wave = state;
26
27  endmodule
```

Fig. 7: Extracto del módulo square_wave_gen donde se realizó la variación de la onda cuadrada.

Secuenciador de notas (‘secuencia_notas’)

Para el módulo del secuenciador de notas, se definió como entrada el índice de la nota deseada, representada mediante un registro de 5 bits, ya que esto nos alcanza para representar una melodía de $2^5 = 32$ notas, suficientes para la melodía del feliz cumpleaños; y como salida la nota deseada en un registro de 16 bits, ya que 16 bits representan frecuencias de hasta 2^16 = 65.5KHz (se utilizaron 16 bits por convención, ya que 8 bits no alcanzaban para representar todas las frecuencias). Se definieron las frecuencias de las notas utilizando una constante ‘localparam’, pasandole a cada nota en formato decimal su equivalente en 16 bits (por ejemplo, nota7 = 16'd440), y se agregó una nota muda para representar los silencios de la melodía (nota_muda = 16'd0).

Para la parte lógica, se utilizó un bloque ‘always’ con un case adentro. El case lo que hace es, dependiendo de qué índice de nota se le pase, asigna continuamente la nota que se tiene que devolver en ese momento. Esta asignación se hace teniendo en cuenta la frecuencia del reloj de la FPGA (12 MHz), dividida por la frecuencia deseada multiplicada por 2, ya que necesito un pulso alto y otro bajo para generar la onda cuadrada. Esto sucede ya que el tiempo de cada estado (alto o bajo) es la mitad del periodo de la onda cuadrada, es decir, T/2, de manera que:

$$T_{alto} = T_{bajo} = T_{nota} / 2 = 1 / 2 * f_{nota}$$

Este resultado de nota nos dará el umbral de ciclos de reloj en el que el estado de la nota tiene que permanecer tocada (en alto), y luego el mismo umbral nos servirá para tener el estado de pausa (en bajo), generando así la señal cuadrada.

Módulo principal (‘caja_musical’)

El módulo principal presenta tres entradas, que son el clock (clk), el reset y el stop (off), y una única salida definida como parlante, que es la onda cuadrada de cada nota. Se crearon algunas variables auxiliares como el índice de la nota, que sirvió para seguir la secuencia de la melodía; su duración, que se fijó en 0.5 segundos por nota; la frecuencia modulada que debe tener el reloj de la FPGA para producir la nota deseada; y el pausar/pausa_prev que son variables que utilizamos dentro del bucle que controla el stop para finalizar la melodía. Dentro de este módulo se instancian los submódulos del secuenciador de notas y el generador de onda cuadrada, para luego inicializar los contadores y realizar la secuencia de la melodía.

En la Figura 8 se visualiza la lógica implementada para el conteo, reseteo, y finalización de la melodía. Cuando el contador alcanza la duración de la nota, este se reinicia a cero y pasa al siguiente índice de nota. Se repite el proceso hasta llegar al índice de la nota máxima (número 29), que corresponde a la nota muda, y se vuelve a iniciar la secuencia.

En el caso de que se presione el botón de resetear, al apretar el botón que corresponde al reset, este se mantiene inicialmente en alto, produciendo una secuencia alto/bajo que desencadena la lógica negativa del reset (!reset). Dentro de esta, el contador se reinicia a cero a la vez que se cambia al índice de la nota muda para inicializar la secuencia nuevamente.

En el caso de que se presione el botón de pausa, si quisiéramos implementar la misma lógica del reset, la pausa solo duraría el tiempo en el que el botón se mantenga presionado, por lo que necesitamos implementar la lógica de pausa de una manera distinta. Para eso, solo detectamos el flanco descendiente al pulsar el botón de stop, y nos guardamos el estadio previo de este. Cuando se apriete el botón de stop, (!stop && pausa_prev) dará verdadero, y cambiará el estado de pausa a activo, desencadenando la lógica de la pausa. Dentro de esta, se lleva a la nota actual a la nota muda (en caso de que la nota al momento de presionar el stop no sea la nota muda) y se mantiene el contador para que este no avance, evitando que se produzca el cambio de nota.

```
40  always @(posedge clk or negedge reset) begin
41      if (!reset) begin
42          cont <= 0;
43          indice_nota <= nota_maxima; // Inicia desde la nota muda
44          pausar <= 0; // Salimos de la pausa en un reset
45          pausa_prev <= 0;
46      end else begin
47          if (!stop && pausa_prev) begin
48              pausar <= ~pausar;
49          end
50
51          // Siempre actualiza el estado anterior de la pausa (esto da 1 ya q el botón sin apretar es = 1)
52          pausa_prev <= stop;
53
54          if (pausar) begin
55              if (indice_nota != nota_maxima) begin
56                  indice_nota <= nota_maxima; // Nota muda
57                  cont <= 0;
58              end else begin
59                  cont <= cont; // Mantengo el contador para no avanzar
60              end
61          end else if (cont >= duracion_nota) begin
62              cont <= 0;
63              if (indice_nota < nota_maxima) begin
64                  indice_nota <= indice_nota + 1;
65              end else begin
66                  indice_nota <= 0; // Se reinicia la secuencia del feliz cumple
67              end
68          end else begin
69              cont <= cont + 1;
70          end
71      end
72  end
```

Fig. 8: Lógica de conteo, reseteo y finalización de la melodía.

Simulación

Se simuló el código para testear su funcionamiento por medio del testbench ‘caja_musical_tb.v’, en donde se generó una señal de clk con un periodo de 83.33 ns (lo que corresponde a una frecuencia de 12 MHz), y en donde se instanció el módulo de ‘caja_musical’, conectando esta instancia a la señal de reloj y al parlante. Ignoramos las señales de reset y stop a la hora de realizar el testbench. Como escala de tiempo se utilizó ‘timescale 1ns/1ps’, y se configuró la duración de la nota en el módulo de ‘caja_musical’ para que sea de 10 milisegundos (duracion_nota = 120000), para que el test no se haga tan extenso y poder testar múltiples veces de manera más rápida y eficiente. Se le dió un tiempo considerable al test (250000000 unidades de tiempo, que equivalen a 0.25 segundos) para poder visualizar la melodía completa.

En la Figura 9 se pueden visualizar en el visor de ondas GTKWave las señales de clk, la frecuencia de la nota, su índice correspondiente y la onda cuadrada que recibe el parlante. Notar que las señales de freq e índice de nota se corresponden con lo que se esperaría de la melodía del feliz cumpleaños: primera y segunda nota iguales, estas dos iguales a la séptima y octava nota, así como también a las notas 14 y 15, además de poder observarse el cambio correcto de frecuencias de más graves a más agudas.

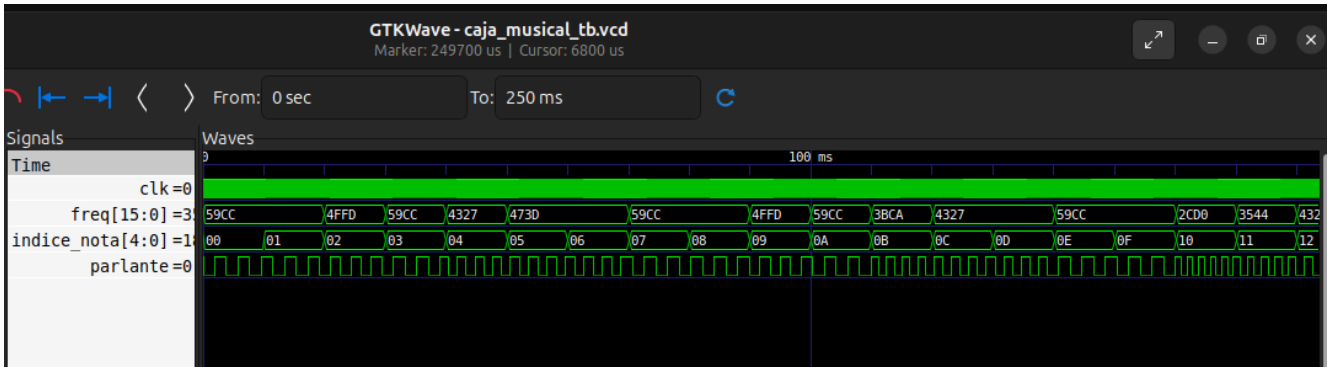

Fig. 9: Visualización de la simulación del código realizado mediante testbench en GTKWave.

Conclusión

En este trabajo se diseñó, codificó, simuló e impactó una caja musical programable en FPGA, mediante el uso de herramientas de electrónica digital como la utilización de módulos, contadores, ondas cuadradas, cálculo de frecuencia, circuitos secuenciales, circuitos combinacionales, entre otros. Codificado en Verilog como lenguaje de descripción de hardware, su herramienta GTKWave para la simulación y visualización de las señales por computadora y la FPGA EDU-CIAA-FPGA para el ensayo real del trabajo, el objetivo propuesto se cumplió y su implementación final demostró que el sistema planteado es capaz de sintetizar y reproducir con precisión la melodía del feliz cumpleaños propuesta por la cátedra, validando así la efectividad del diseño basado en Verilog.

Agradecimientos

Queremos agradecer a los profesores de la asignatura de Electrónica Digital que nos guiaron durante el desarrollo de este trabajo. Agradecemos su disposición para responder a nuestras consultas y aclarar las dudas que nos fueron apareciendo a lo largo de la realización del mismo, así como también el proporcionarnos la FPGA necesaria para poder impactar y probar nuestro código funcional previamente testeado.

Referencias

[1] Proyecto CIAA. EDU-CIAA-FPGA.
https://www.proyecto-ciaa.com.ar/devwiki/doku.php%3Fid=desarrollo:edu-fpga.html

[2] Wikipedia. Lenguaje de descripción de hardware.
https://es.wikipedia.org/wiki/Lenguaje_de_descripci%C3%B3n_de_hardware#:~:text=Un%20lenguaje%20de%20descripci%C3%B3n%20de,anal%C3%B3gico%2Ddigital%20o%20cualquier%20antena

[3] Generating simple square waves using FPGA. (2016, February 9). Numato Lab.
https://numato.com/kb/generating-square-wave-using-fpga/