

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

El proyecto consiste en "dibujar" una función periódica, ya sea senoide, rectangular o triangular. Se puede controlar su frecuencia desde 1 a 16383 Hz, el offset de 0 a 1023 y la amplitud a 512. El control es con los switches, que seleccionan los parámetros a editar y los botones para cambiar la magnitud de cada parámetro. El Input Driver es el análogo al Program Counter, y genera una palabra de control de 32 bits, esto es más rápido que los parámetros por separado y puede escribirse como dato en comunicación AXI para cualquier formato. Después un decodificador inicializa y asigna los parámetros por separado para entregarlos al Wave Generator, que es el generador de funciones en sí, tiene un módulo que controla el clock de entrada para generar una rampa de la frecuencia seleccionada, esta rampa es entrada para cada uno de los módulos seno, rectángulo y triángulo, que generan la onda deseada por separado. La salida va si o si a la computadora, para eliminar el riesgo de quemar la tarjeta.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%): *Descripción:* 1. Utilice components para integrar al menos tres componentes dentro de otra entity. El código cumple con las funciones que habíamos planteado en nuestro proyecto las cuales son: incorporar en una entity no sólo 3, sino 6 components distintos. De acuerdo a la arquitectura presentada en la sección I, esto se implementó en el bloque 3, cuya funcionalidad es ser salida del generador de funciones. *Nivel de Logro:* 100%. Completamente logrado, el

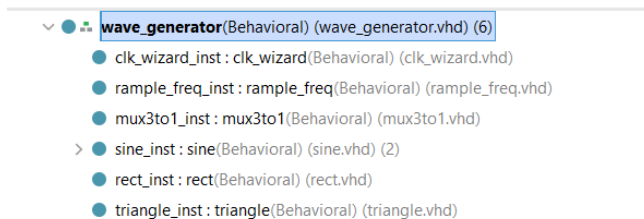


FIGURE 1: 6 components dentro de una entity llamada wave generator.

código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AO2 (100%): AO2: Genere al menos 3 diferentes packages (IP-Cores) en vivado block design para incorporar sus códigos. Utilice parametos genéricos para la configuración de sus packages Sin AXI, tenemos 3 diferentes IP Cores relevantes, Input Driver, Decoder y Wave Generator, también ILA/VIO son IP Cores válidos. Se pudo desarrollar hasta el bitstream completo. El block diagram se tuvo que probar en primera instancia como bloques RTL, sin convertir a IP Cores.

AO3:...

AC1: (100%) Genere una máquina de estados que esté asociada a parte o al total de su proyecto. Para controlar los

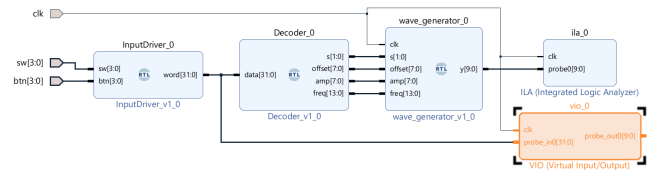


FIGURE 2: 3 IP Cores distintas, conectadas a ILA/VIO.

parámetros de dibujo (frecuencias, offset, amplitud y forma) es necesario usar los botones y switches con una máquina de estados, esta máquina define los estados como los valores de switches, los botones como suma o resta de los parámetros. La máquina se encarga de alterar los valores de la palabra de control de 32 bits que contiene la información necesaria para generar la onda. Se logró pasar hasta la síntesis por sí sola, pero la implementación por la limitación física de 32 bits no es necesaria.

```
begin
  SWITCH: process (sw)
  begin
    case sw is
      when FREQUENCY =>
        if btn(0) = '1' then
          f <= f + 10;
        end if;
        if btn(1) = '1' then
          f <= f + 100;
        end if;
        if btn(2) = '1' then
          f <= f - 10;
        end if;
        if btn(3) = '1' then
          f <= f - 100;
        end if;
        if f < 0 or f > 16383 then
          f <= 0;
        end if;
      when AMPLITUDE =>
        if btn(0) = '1' then
          a <= a + 1;
        end if;
        if btn(1) = '1' then
          a <= a + 10;
        end if;
        if btn(2) = '1' then
          a <= a - 1;
        end if;
    end case;
  end process;
end;
```

FIGURE 3: Máquina de estados en la entity Input Driver.

AC2: (100%) Utilice Vio e ILA para monitorear y modificar señales de su proyecto. Se pudo implementar exitosamente Vio e ILA para ver las señales por sí solas del módulo wave generator, el bitstream está guardado aparte.

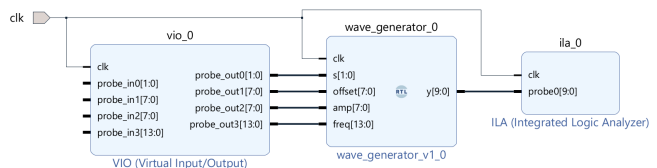


FIGURE 4: Implementación básica ILA/VIO.

AC3: (100%) Utilice al menos 2 operadores y 2 atributos diferentes. Particularmente el módulo componente triangle

del wave generator usa 3 atributos diferentes (std logic vector, integers y un booleano flag), y 7 operaciones (igualdad, asignación, negación, suma, resta, multiplicación, división). Esto para construir la función triangle con la rampa de entrada y controlarla con los parámetros offset y amplitud.

```

architecture Behavioral of triangle is
    signal flag : boolean := FALSE;
    signal t_buffer: std_logic_vector(7 downto 0);
    signal n_tr: integer range 0 to 1023;

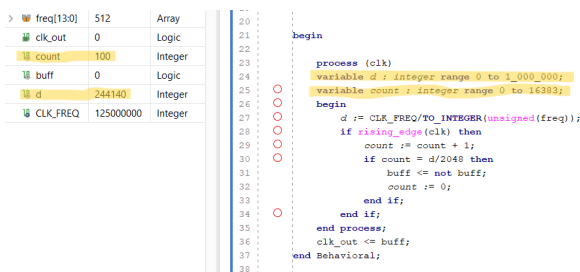
begin
    process (a, offset, amp)
    begin
        -- Calcular n_tr basado en el valor de a, offset y amp
        if a = 1023 then
            flag <= not flag;
        end if;

        if flag = TRUE then
            n_tr <= to_integer(unsigned(a)) * to_integer(unsigned(amp)) / 512 + to_integer(unsigned(offset));
        else
            n_tr <= (1023 - to_integer(unsigned(a))) * to_integer(unsigned(amp)) / 512 + to_integer(unsigned(offset));
        end if;
        if n_tr < 0 then
            n_tr <= 0;
        end if;
        -- Convertir n_tr a std_logic_vector y asignarlo a t
        t <= std_logic_vector(to_unsigned(n_tr, t'length));
    end process;
end Behavioral;

```

FIGURE 5: Operadores y atributos.

AC4: (100%) *Utilice variables para actualizar valores instantaneamente en alguna parte de la lógica programada* En el componente clock wizard, que controla el período de la rampa, se usan dos variables llamadas d y count. Esto para no abusar el uso de señales dentro de la lógica del programa.



```

begin
    process (clk)
        variable d : integer range 0 to 1_000_000;
        variable count : integer range 0 to 16383;
    begin
        d := CLK_FREQ/TO_INTEGER(unsigned(freq));
        if rising_edge(clk) then
            count := count + 1;
            if count = d/2048 then
                buff <= not buff;
                count := 0;
            end if;
        end if;
    end process;
    clk_out <= buff;
end Behavioral;

```

FIGURE 6: En destacado, las variables usadas y su correcta implementación a la izquierda.

AC5: (100%) *Utilice código secuencial y código concurrente y evidencie claramente la diferencia de funcionamiento de ambos tipos de códigos* La lógica concurrente encontrada en la figura 6 corresponde al multiplexor. Usa when y no tiene un proceso asociado, es decir el valor de las salidas no depende de entradas anteriores. Por lo que es un código puramente concurrente.

La lógica secuencial, que se encuentra comunmente en los componentes de dibujo de wave generator. En clock wizard es necesario usar un contador que cuenta, valga la redundancia, hasta el período de la frecuencia pedida, pero escalada en 10 nanosegundos, que es el período del clock de la Zybo, la parte secuencial está en el contador, que depende de la entrada un ciclo antes.

AC7: (100%) *Implemente alguna función que no se haya presentado en el curso.* Se usó el IP Core Distributed Memory Generator, que no se vió en el curso, pero es conocido en otra asignatura. Este bloque de memoria se inicializa con un archivo coe que tiene una muestra de 1024 partes de un período de onda sinusoidal. Funciona básicamente como n°

```

architecture Behavioral of mux3to1 is
begin

y <= x1 when s = "00" else
    x2 when s = "01" else
    x3 when s = "10" else
    "0000000000";

end Behavioral;

```

FIGURE 7: .

```

process (clk)
    variable d : integer range 0 to 1_000_000;
    variable count : integer range 0 to 16383;
begin
    d := CLK_FREQ/TO_INTEGER(unsigned(freq));
    if rising_edge(clk) then
        count := count + 1;
        if count = d/2048 then
            buff <= not buff;
            count := 0;
        end if;
    end if;
end process;
clk_out <= buff;
end Behavioral;

```

FIGURE 8: .

muestra - DMG - muestra.

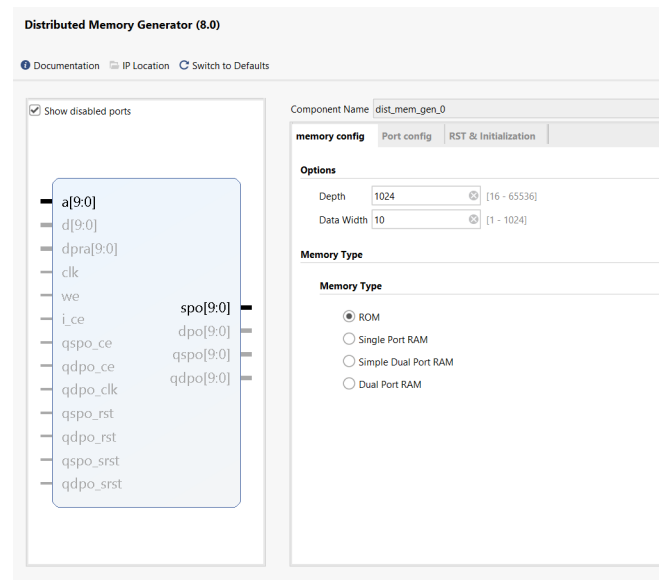


FIGURE 9: SETUP del IP Core DMG 8.0 de Xilinx

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

Arriba está la figura de simulación de comportamiento (ideal) y abajo, la simulación post-implementación

- Identificación de delays
- (0.75puntos) La diferencia entre la actualización de señales y variables.
- (2puntos) Mostrar la correcta ejecución de las transacciones AXI Lite y AXI Full. Utilice Testbench o ILA.

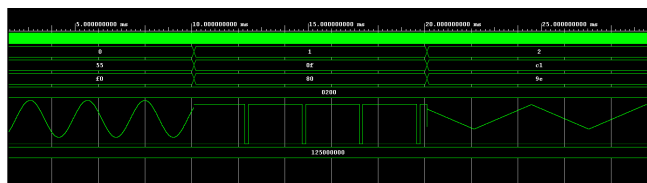


FIGURE 10: Simulación de las 3 funciones a distintos offset y amplitudes.

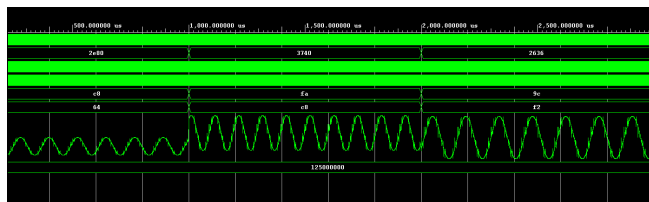


FIGURE 11: Simulación de 3 sinusoides post-implementación

Identifique claramente los handshake asociados a cada transacción.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

Los resultados de implementación fueron buenos, a pesar de que la implementación tenía un tamaño considerable, y corrió lento. Pudo completarse la totalidad del código en la tarjeta.

V. CONCLUSIONES(0.2)

Enumere las conclusiones más importante de su proyecto. Recuerde que:

- Las conclusiones deben ser cuantitativas (con datos). Evite escribir cosas como: "los resultados fueron buenos", " el sistema funcionó bien" o "se enviaron algunos datos exitosos y otros no". Se esperan conclusiones del tipo "La cantidad de datos esperados eran de XXX en YY ciclos de reloj, sin embargo solo se recibieron AAA en BBB ciclos de reloj. La reducción de datos esperados se debió a WWWW". Siempre cuantifique con números y datos sus conclusiones.
- Recuerde que las conclusiones no es un resumen de su trabajo. Evite escribir cosas como: "En este trabajo se hizo desarrollo una implementación de luego se hizo....., posteriormente implementamos.....". Las conclusiones deben ser breves, no es necesario explayarse innecesariamente.

Si sigue estas reglas para escribir las conclusiones tendrá los 0.25 puntos.

VI. TRABAJOS FUTUROS (0.2)

Identifique las oportunidades de trabajos futuros en caso que algún compañero quisiera continuar con una siguiente versión de su proyecto. Debe dar detalles técnicos de los desafíos que quedan por resolver. Eso le dará los 0.25 puntos.

VII. INTRODUCCIÓN A LATEX

En las siguientes líneas se dejan como ejemplo los comandos más utilizados para escribir un informe. Es probable que no necesite más comentarios que estos. Para hacer una lista de puntos hagala así:

- item uno
- item 2
- otro item

Si las quiere con números, entonces use:

- 1) la primera cosa
- 2) la segunda
- 3) y así sucesivamente.

Texto en negrita va así **este texto está en negrita**. Texto en cursiva va así *este texto está en negrita*. Si quiere escribir algo en color, use el comando **esto va en rojo**.

Para incorporar una figura en latex simplemente use el texto que viene a continuación indicando el directorio (en overleaf) de donde ha guardado su figura.

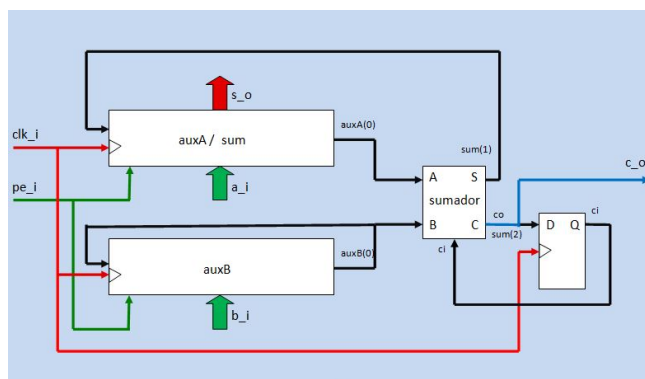


FIGURE 12: Este es un ejemplo de como incorporar una figura de un esquema.

Si quiero referenciar mi figura en el texto, simplemente uso este comando para decir que en la Fig. 12 se está describiendo la estructura de mi proyecto. El texto que va dentro de "ref{" es el label de la figura. Si desea agregar una tabla, lo más conveniente es hacerla en el software que más le acomode y luego agregarla como una figura.

Si quiero agregar una subsección, simplemente uso el siguiente comando:

A. EQUATIONS

En esta subseccion veremos que para agregar una ecuación, simplemente uso el código siguiente:

$$E = mc^2. \quad (1)$$

Para una lista de comandos matemáticos puede visitar [aquí](#).

Para referencia mi ecuación, simplemente digo: En la eq. (1), se describe la relación entre masa y energía.

En la siguiente sección se listan las referencias utilizadas en su trabajo. Si ud quiere hacer uso de ellas, debe simplemente usar el comando "cite". Así, por ejemplo diremos que en [1], encontré una idea de proyecto en el cual me

basé, además en [2] encontré un ejemplo de como crear un IPCore, y use una función que me sirvió para dividir dos datos, además, utilice los documentos [3]–[5] para aprender como funciona el protocolo AXI.

Con toda esta información es suficiente para que pueda escribir su proyecto.

REFERENCES

- [1] G. O. Young, “Synthetic structure of industrial plastics,” in *Plastics*, 2nd ed., vol. 3, J. Peters, Ed. New York, NY, USA: McGraw-Hill, 1964, pp. 15–64.
- [2] W.-K. Chen, *Linear Networks and Systems*. Belmont, CA, USA: Wadsworth, 1993, pp. 123–135.
- [3] J. U. Duncombe, “Infrared navigation—Part I: An assessment of feasibility,” *IEEE Trans. Electron Devices*, vol. ED-11, no. 1, pp. 34–39, Jan. 1959, 10.1109/TED.2016.2628402.
- [4] E. P. Wigner, “Theory of traveling-wave optical laser,” *Phys. Rev.*, vol. 134, pp. A635–A646, Dec. 1965.
- [5] E. H. Miller, “A note on reflector arrays,” *IEEE Trans. Antennas Propagat.*, to be published.

...