Pontificia Universidad Católica de Chile
IEE2473 - Laboratorio de Electrónica Analógica y Digital.

Informe Laboratorio n°1

16 de Septiembre de 2022

2º semestre 2022

Vicente Barrios - Elías Bruchfeld

Pseudo índice:

- Guía previa: página 1 a página 17
- Informe: página 18 en adelante

4.1.1. Acondicionamiento de Señal - Análisis previo

1. Para encontrar V_o, lo que hacemos es superponer la señal AC de pequeña señal con la DC de gran señal.
Para análisis DC, los capacitores son circuitos abiertos, por lo que nos queda:

$$V_0 = V_{ref} \cdot (1 + \frac{R_2}{R_1})$$

Mientras que en análisis AC, las fuentes DC las consideramos cortocircuitos y hacemos:

$$V_0 = V_i \cdot \frac{-Z_2}{Z_1}$$

Definiendo Z_1 y Z_2 como:

$$Z_1 = \frac{1}{sC_1} + R_1 = \frac{1 + sC_1R_1}{sC_1}$$
$$Z_2 = \frac{1}{sC_2} \parallel R_2 = \frac{R_2}{1 + sR_2C_2}$$

Así, obtenemos:

$$V_0 = V_i \cdot -\frac{R_2}{1 + sR_2C_2} \cdot \frac{sC_1}{1 + sC_1R_1}$$

Finalmente, superponiendo ambas señales nos queda:

$$V_0 = -V_i \cdot \frac{R_2}{1 + sR_2C_2} \cdot \frac{sC_1}{1 + sC_1R_1} + V_{ref} \cdot (1 + \frac{R_2}{R_1})$$

2.
Para encontrar la ganancia del circuito buscamos despejar $\frac{V_0}{V_i}$, en función únicamente de R_1 y R_2.
De esta manera C_1 debe actuar como un cortocircuito, de lo contrario la señal v_i no podría pasar. De esta forma, C_2 es el que debe actuar como un circuito abierto.
Así, la ganancía obtenida será:

$$\frac{V_0}{V_i} = \frac{-R_2}{R_1}$$

3.
Considerando C_2 como circuito abierto, nos queda:

$$\begin{aligned}
V_0 &= V_i \cdot \frac{-R_2}{\frac{1}{sC_1} + R_1} \\
V_0 &= V_i \cdot \frac{-sC_1R_2}{1 + sR_1C_1} \\
V_0 &= V_i \cdot \frac{-jwC_1R_2}{1 + jwR_1C_1} \\
V_0 &= V_i \cdot \frac{(-jwC_1R_2) \cdot (1 - jwR_1C_1)}{1 + w^2R_1^2C_1^2} \\
\frac{V_0}{V_i} &= \frac{-jwC_1R_2}{1 + w^2R_1^2C_1^2} - \frac{w^2R_1R_2C_1^2}{1 + w^2R_1^2C_1^2} \\
\angle\frac{V_0}{V_i} &= arctan(\frac{wC_1R_2}{w^2R_1R_2C_1^2}) \\
\angle\frac{V_0}{V_i} &= arctan(\frac{1}{wR_1C_1})
\end{aligned}$$

4.
Ahora, considerando C_1 como cortocircuito, nos queda:

$$
\begin{aligned}
V_0 &= V_i \cdot \frac{-R_2}{1+sR_2C_2} \cdot \frac{1}{R_1} \\
V_0 &= \frac{-V_i \cdot R_2}{R_1} \cdot \frac{1}{1+jwR_2C_2} \\
\frac{V_0}{V_i} &= \frac{-R_2}{R_1} \cdot \frac{1}{1+jwR_2C_2} \\
\frac{V_0}{V_i} &= \frac{-R_2}{R_1} \cdot \frac{1-jwR_2C_2}{1+w^2R_2^2C_2^2} \\
\angle\frac{V_0}{V_i} &= arctan(-wR_2C_2) = -arctan(wR_2C_2)
\end{aligned}
$$

5.
Entre los capacitores electrolíticos y los cerámicos existen diferencias en su construcción y también en su uso.
Con respecto al uso, los capacitores electrolíticos son usados en voltajes de corriente continua (DC) por eso estos tienen polaridad, mientras que los cerámicos se usan mas en alterna o para eliminar la componente continua (DC) de una señal de corriente alterna (AC) con componente de continua.
De esta manera, para el capacitor C_1 sería conveniente usar un capacitor cerámico, para eliminar la componente DC de la señal de entrada v_i, es decir realizar el acople AC de la señal.
Mientrás que para el capacitor C_2 debería ser útil usar un capacitor electrolítico producto de que el OpAmp por cortocircuito virtual forzará un voltaje V_{ref} continuo en su entrada inversora.

6.
De las especificaciones sabemos que en los extremos de la banda, es decir para 1 [Hz] y para 400 [Hz], la magnitud del desfase será de 15^o, lo que es equivalente a 0,2618 rad.
Así, del resultado obtenido en el punto 3, y sabiendo que es un pasabajos reemplazamos f=400, y despejamos R_1C_1:

$$
\begin{aligned}
\angle\frac{V_0}{V_i} &= arctan(\frac{1}{wR_1C_1}) \\
0,2618 &= arctan(\frac{1}{wR_1C_1}) \\
tan(0,2618) &= \frac{1}{wR_1C_1} \\
0,2679 &= \frac{1}{2\pi 1 \cdot R_1C_1} \\
R_1C_1 &= 0,5941
\end{aligned}
$$

Mientras del punto 4, despejamos R_2C_2, reemplazando f=1[Hz].

$$\begin{aligned}
\angle\frac{V_0}{V_i} &= arctan(-wR_2C_2) = -arctan(wR_2C_2) \\
15^o &= arctan(-wR_2C_2) \\
0,2618 &= arctan(-wR_2C_2) \\
0,2679 &= 2\pi 400 \cdot R_2C_2 \\
R_2C_2 &= 1,0659 \cdot 10^{-4}
\end{aligned}$$

Por otra parte, la ganancia (A) del circuito debería ser:

$$\frac{\Delta V_o}{\Delta V_i} = \frac{3,3[V] - 0[V]}{10[V] - 3[V]} = 0,4714$$

Por lo tanto del punto 2, sabemos:

$$A = \frac{V_0}{V_i} = \frac{R_2}{R_1} = 0,4714$$

De esta manera, si elegimos R_2=4,714 $[k\Omega]$ y R_1=10 $[k\Omega]$, cumplimos con ese propósito.
Y despejamos C_1 de la siguiente ecuación:

$$\begin{aligned}
R_1C_1 &= 0,5941 \\
C_1 &= 5,941 \cdot 10^{-5}
\end{aligned}$$

Mientras que C_2 lo despejamos de la siguiente ecuación:

$$\begin{aligned}
R_2C_2 &= 1,0659 \cdot 10^{-4} \\
C_2 &= 2,2611 \cdot 10^{-8}
\end{aligned}$$

7.
Así, la frecuencia de corte inferior la obtenemos mediante:

$$\begin{aligned}
f_u &= \frac{1}{2\pi R_1C_1} \\
f_u &= \frac{1}{2\pi \cdot 0,5941} \\
f_u &= 0,2679[Hz]
\end{aligned}$$

Mientras que la frecuencia de corte superior la obtenemos mediante:

$$\begin{aligned}
f_d &= \frac{1}{2\pi R_2C_2} \\
f_d &= \frac{1}{2\pi \cdot 1,0659 \cdot 10^{-4}} \\
f_d &= 1493,1508[Hz]
\end{aligned}$$

8.
Así, tendremos 3 ecuaciones, la orimera de ellas aportada por la ganancia del circuito:

$$\frac{V_o}{V_i} = \frac{R_2}{R_1}$$
$$\frac{3,3}{7} = \frac{R_2}{R_1}$$

La segunda:

$$R_1 C_1 = 0,5941$$

Y la tercera:

$$R_2 C_2 = 1,0659 \cdot 10^{-4}$$

Realizando una simulación AC, podemos obtener las frecuencias de corte.

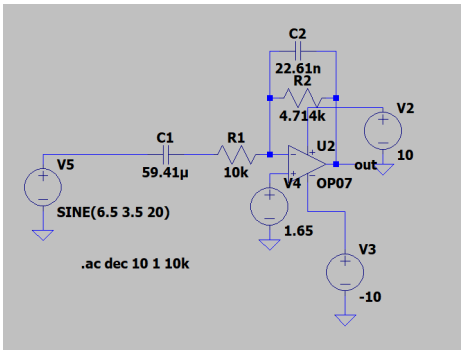

Figura 1: Simulación AC circuito en LTspice

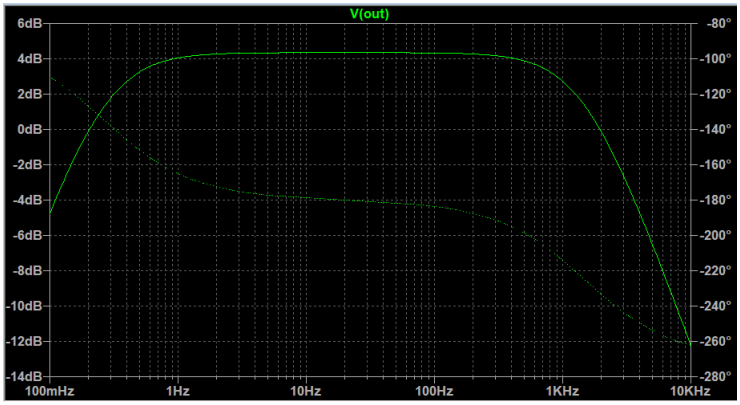

Figura 2: Gráfico de Bode

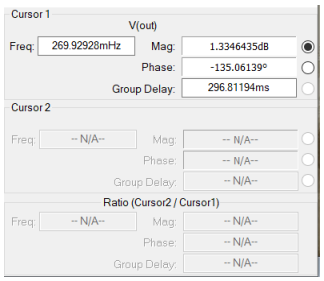

Figura 3: Frecuencia de corte inferior

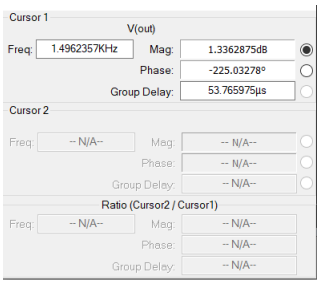

Figura 4: Frecuencia de corte superior

Para encontrar laa frecuencias de corte buscamos -3 [dB] respecto a ganancia de la banda pasante.
Así, tendremos una frecuencia de corte inferior igual a 0,269 [Hz] y una frecuencia de corte superior igual a 1496,2357 [Hz]. Entonces, podemos ver que nos dan valores muy similares a las frecuencias de corte teóricas.

9.
El circuito implementado en LTspice nos queda como:

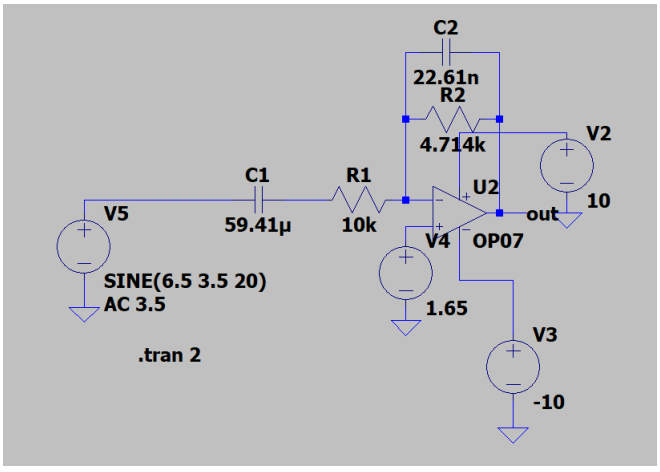

Figura 5: Circuito en LTspice

a.
Para una entrada nula obtenemos:

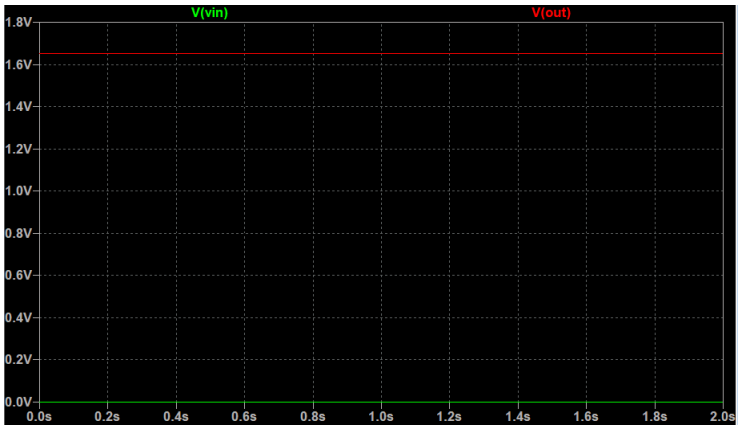

Figura 6: V_{in} y V_{out}

b.
Para una entrada de 6.5 V obtenemos:

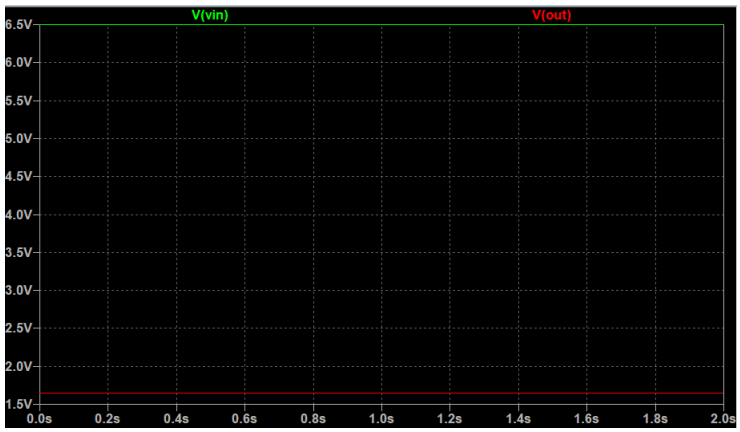

Figura 7: V_{in} y V_{out}

c.
Para una entrada de 1 Hz, 7 V_{pp}, y componente continua de 6.5 V obtenemos:

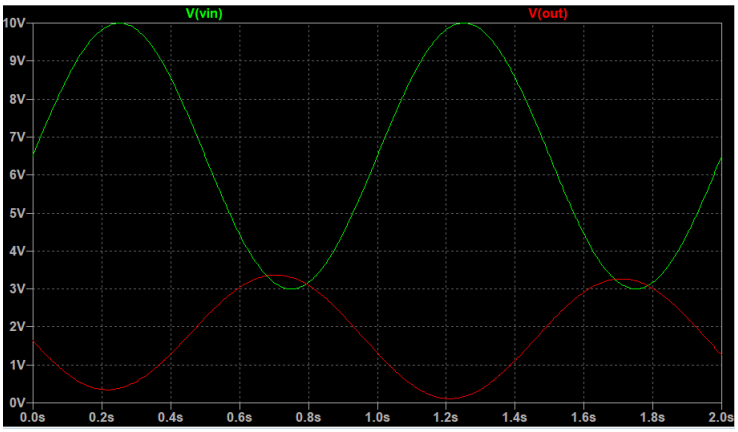

Figura 8: V_{in} y V_{out}

d.
Para una entrada de 1 Hz, 1 V_{pp}, y componente continua de 6.5 V obtenemos:

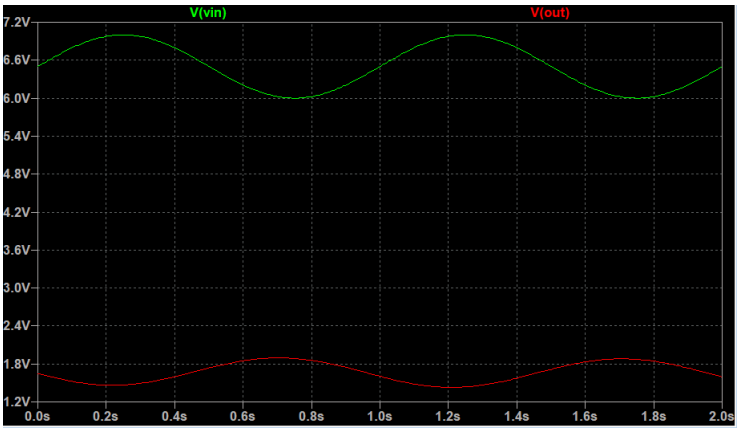

Figura 9: V_{in} y V_{out}

e.
Para una entrada de 20 Hz, 7 V_{pp}, y componente continua de 6.5 V obtenemos:

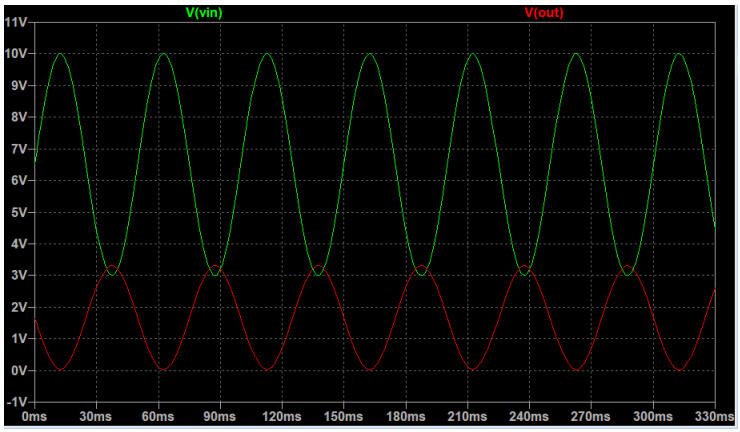

Figura 10: V_{in} y V_{out}

f.
Para una entrada de 20 Hz, 1 V_{pp}, y componente continua de 6.5 V obtenemos:

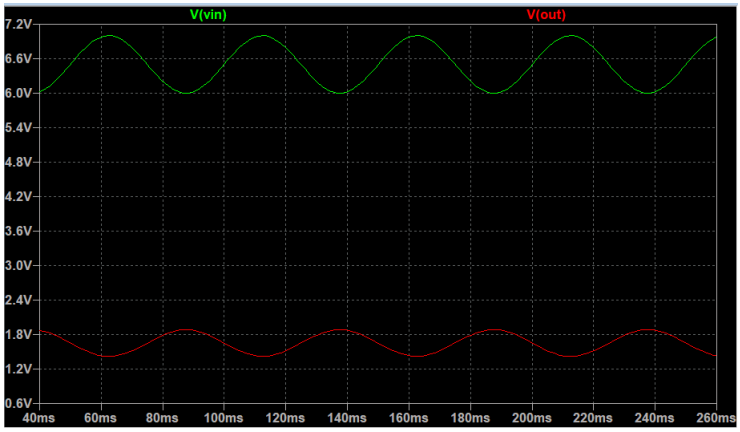

Figura 11: V_{in} y V_{out}

g.
Para una entrada de 400 Hz, 7 V_{pp}, y componente continua de 6.5 V obtenemos:

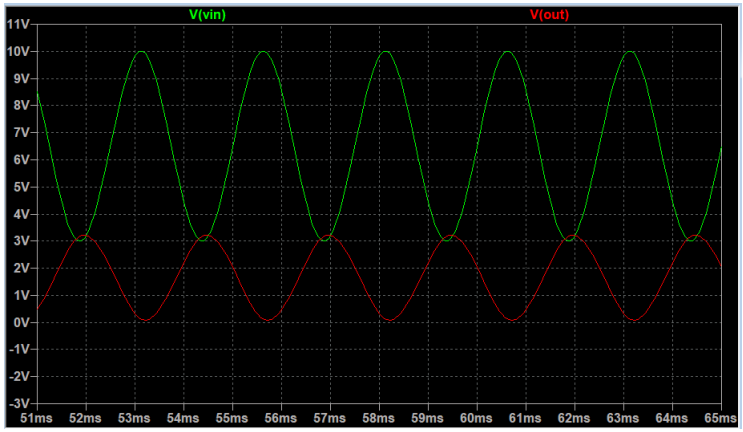

Figura 12: V_{in} y V_{out}

h.
Para una entrada de 400 Hz, 1 V_{pp}, y componente continua de 6.5 V obtenemos:

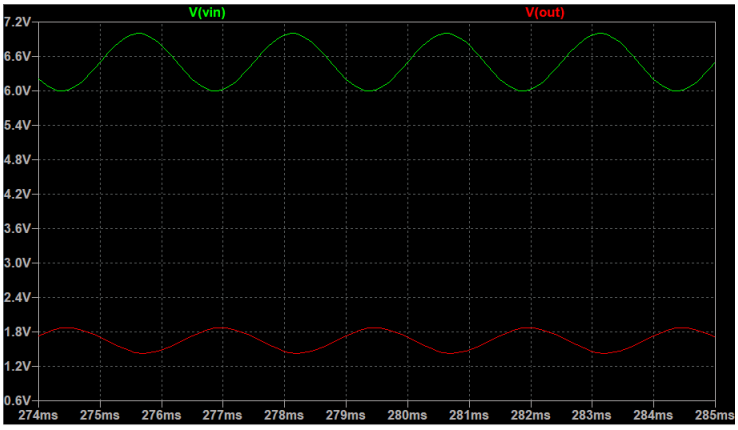

Figura 13: V_{in} y V_{out}

i.
El desfase relativo entre la señal de entrada y salida, lo podemos encontrar mediante la siguiente fórmula.

$$w = \frac{\Delta\theta}{\Delta T}$$
$$2\pi f = \frac{\Delta\theta}{\Delta T}$$
$$\Delta\theta = 2\pi f \cdot \Delta T$$

Así, para el punto c, nos queda:

$$\Delta\theta = 2\pi 1[Hz] \cdot (694,02174(ms) - 251,63043(ms)) = 2,7796(rad) = 159,2609^o$$

Los Δ T fueron obtenidos mediante los cursores de LTspice.
Mientras que para el punto e:

$$\Delta\theta = 2\pi 20[Hz] \cdot (137,78263(ms) - 112,57783(ms)) = 3,1673(rad) = 181,4729^o$$

Los Δ T fueron obtenidos mediante los cursores de LTspice.

Finalmente para el punto g:

$$\Delta\theta = 2\pi 400[Hz] \cdot (319,46288(ms) - 318,10356(ms)) = 3,4163(rad) = 195,7421^o$$

Los Δ T fueron obtenidos mediante los cursores de LTspice.

4.1.2. MSP430F5529 - Análisis previo

1.

Según el datasheet de la MSP430, el voltaje de entrada al ADC debe estar entre los rangos V_{R+} y V_{R-}, donde dichos valores son programables. Además, V_{R-} debe ser mayor o igual a 0V y V_{R+} debe ser menor o igual al voltaje de alimentación de la MSP (obviamente con $V_{R-} < V_{R+}$. En caso de no programar dichos valores, se utilizan como referencia simplemente la tierra y el voltaje de alimentación. Para que el ADC funcione bien, se recomienda un volaje de alimentación de entre 2.2V y 3.6V. Existe la opción de utilizar 2 conversores ADC; el de 10 bit y el de 12 bit. Esto simplemente limita la resolución de la conversión, ya que el de 10 bit entrega valores entre 0 y 1023, y el de 12 entre 0 y 4095. La formula de conversión es la siguiente:

$$N_{ADC} = (2^n - 1)\frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$

Entonces por ejemplo, si se tiene que $V_{R-} = 0V$, $V_{R+} = 3{,}6V$, se tiene una señal de entrada de $2V$ y se quiere usar el conversor de 12 bits, el resultado de la conversión sería:

$$N_{ADC} = (2^{12} - 1)\frac{2V - 0V}{3{,}6V - 0V} = 2275$$

2.
La comunicación en serie utiliza un solo cable. La información se envía bit por bit desde

el transmisor hasta el receptor. Si se quisiera mandar un byte, habría que, enviar los 8 bits individualmente de un lado a otro. Por otro lado, la comunicación en paralelo utiliza múltiples cables. La información se envía toda junta del transmisor al receptor. Si se quisiera mandar un byte, se necesitarían al menos 8 cables para mandarlo, pero las señales se enviarían todas al mismo tiempo.

La comunicación en paralelo tiene la ventaja de que la comunicación es muy rápida. Si hablamos de mandar bytes, se pueden mandar 8 bytes en paralelo en el tiempo en que se mandó 1 en serie. Sin embargo, tiene las complicaciones de que si se trabaja a frecuencias muy altas, es muy común que los bits de información lleguen a destino en tiempos diferentes.

Por otro lado, la comunicación en serie sirve mucho mejor para trabajar con altas frecuencias o con distancias grandes entre comunicador y receptor. Además, los cables que utilizan son mucho más pequeños y baratos que los de comunicación en paralelo. Sin embargo tiene la desventaja ya antes mencionada de que la comunicación es más lenta.

En el MSP430F5529, pueden ser utilizados como transmisor los pines 4.4 y 3.3, y como receptor los pines 4.5 y 3.4.

4.1.3. FPGA - Análisis previo

1.
El diagrama de bloques de funcionamiento de la FPGA es el siguiente:

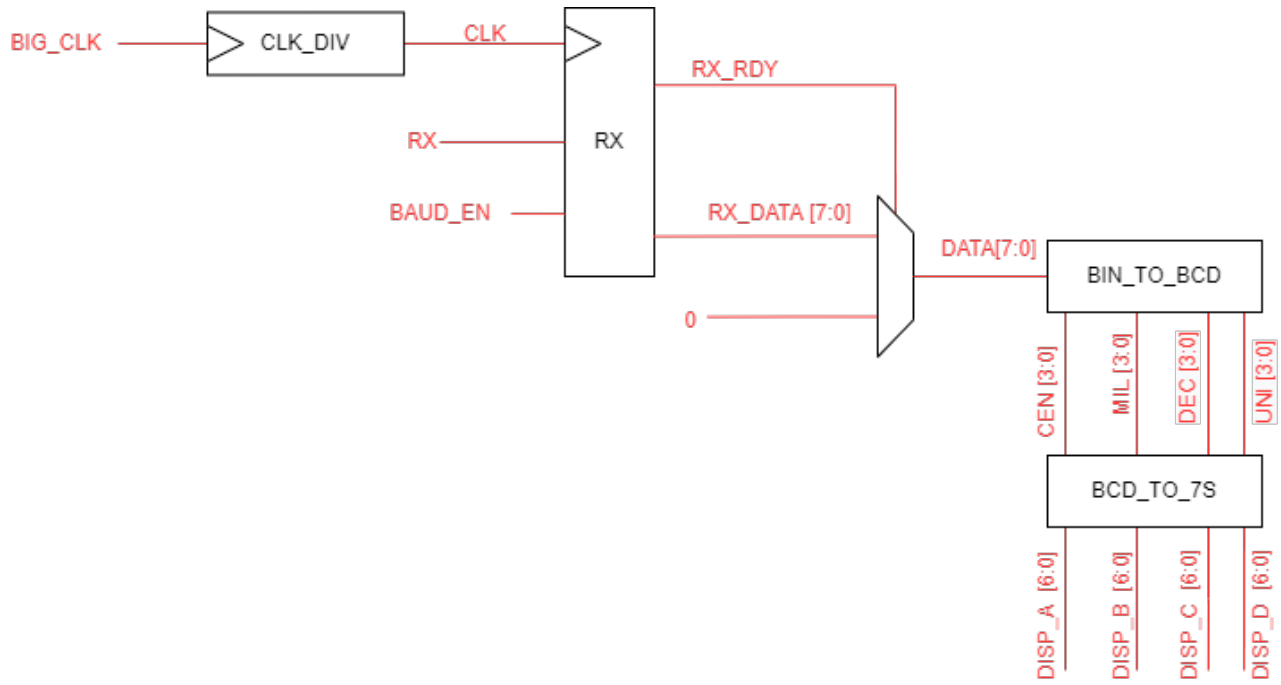

Figura 14: Diagrama de bloques

En primer lugar, hay un bloque divisor de clock que sirve para disminuir la frecuencia del reloj de la Basys.

Luego viene el módulo receptor que recibe como entrada la señal de reloj modificada, una señal llamada baud enable que permite hacer un muestreo de los datos recibidos durante un periodo de tiempo (solo cuando es 1) y una señal RX que es el mensaje recibido. Como UART es un protocolo de comunicación en serie, entonces recibe los datos uno por uno (datos de 1 bit). El bloque tiene como salida una señal rx_rdy que indica si ya se terminó de recibir el mensaje (1) o no (0). La otra salida es un bus de 8 bits con el mensaje que se acaba de recibir.

Luego la señal pasa por un multiplexor cuya salida es el byte recibido en caso de que ya se haya terminado de recibir, y 0 si es que todavía no se recibe completo (o no se recibe nada).

El número está en formato de un byte, pero para poder mostrarlo en el display primero hay que pasarlo a formato BCD (Binary-Coded Decimal). La entrada de ese bloque es un bus de 8 bits, y la salida son cuatro buses de 4 bits cada uno correspondientes a las unidades, decenas, centenas y miles del numero anterior.

Finalmente, los números hay que pasarlos a formato 7 segmentos. Para eso hay un bloque cuya entrada son los tres buses de 4 bits, pero cuya salida son 4 buses de 7 bits (por los 7 segmentos).

2.
Cuando trabajamos con una FPGA contamos con un oscilador (frecuencia fija), en el caso de la Basys 3 cuanta con un oscilador de 100 MHz. Pero, en muchos casos queremos controlar periféricos como encender o apagar un LED a una frecuencia mucho menor.
De esta forma mediante un un divisor de clock, recibimos como entrada una señal de clock y entrega una nueva señal de clock con una menor frecuencia.
La implementación de un divisor de clock puede ser realizada mediante un flip flop tipo D, en donde logramos tener un clock de salida de frecuencia igual a la mitad de frecuencia de entrada.

3.

```
// Code your design here
// Fuente: https://www.fpga4student.com/2017/08/verilog-code-for-clock-divider-on-fpga.html
 `timescale 10ns / 1ps

module clk_divider(clk_in,clk_out);

input clk_in; // input clock on FPGA
output reg clk_out; // output clock after dividing the input clock by divisor
  reg[27:0] counter=28'd0; //Este es el contador que nos indica cuando cambia clk_out

parameter DIVISOR = 28'd2;   //En este caso dividiremos la mitad de la frecuencia
// La frecuencia del clk_out será el clk_in dividido por el DIVISOR.

// For example: Fclk_in = 50Mhz, if you want to get 1Hz signal to blink LEDs
// You will modify the DIVISOR parameter value to 28'd50.000.000
// Then the frequency of the output clk_out = 50Mhz/50.000.000 = 1Hz

always @(posedge clk_in)
    begin
        counter <= counter + 28'd1;
        if(counter>=(DIVISOR-1))  //En esta condición reseteamos el contador, counter parte de cero.
            counter <= 28'd0;
        clk_out <= (counter<DIVISOR/2)?1'b1:1'b0;
    end
endmodule

```

Figura 15: Diseño Modulo Divisor de clock

El módulo clock divider fue obtenido de la página FPGA4Student y eL link está comentado en el código.

```
// Code your testbench here
// or browse Examples
 `timescale 10ns / 1ps //Unidad de tiempo y escala de precisión.
module slow_clk_TB;
  reg clk_in;
  wire clk_out;

  //Unit Under Test (UUT)

  clk_divider uut(.clk_in(clk_in),
  .clk_out(clk_out));

  initial begin
   $dumpfile("dump.vcd");
   $dumpvars;
   #120$finish;
  // Initialize Inputs
  end

  initial begin
    // Initialize Inputs
    clk_in = 0;
  end

  always #5 clk_in = ~clk_in;

endmodule
```

Figura 16: Testbench Módulo Divisor de clock

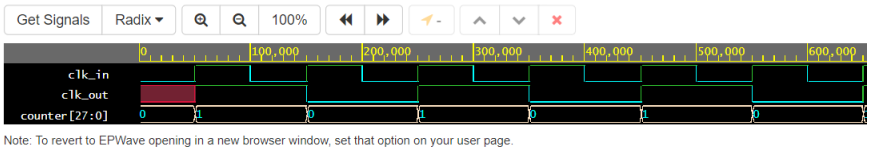

Figura 17: Resultado Testbench Módulo Divisor de clock

4.
Se realizó el siguiente módulo receptor:

```
`timescale 1ns / 1ps
//----------------------------------------------------
// Nombre del módulo    : uart_rx
// Nombre del archivo   : uart_rx.v
// Función              : Recibe los datos
// Diseñador            : Grupo 5
// Fecha                : 26 de noviembre de 2021
//----------------------------------------------------

module uart_rx(
    input clk,
    input rst,
    input rx,
    input baud_en,
    output reg [7:0] rx_data,
    output reg rx_rdy
    );

    // Estados
    parameter IDLE = 3'b000;
    parameter START = 3'b001;
    parameter BIT = 3'b010;
    parameter STOP = 3'b011;
    parameter WAIT = 3'b100;

    reg [1:0] state, nextstate;
    reg [3:0] bit_cnt;
    reg [4:0] baud_cnt;

    always @(posedge clk, posedge rst) begin //bloque con valores de contadores segun estado
        if (rst)
            state <= IDLE;
        else begin
            state <= nextstate;
            if (baud_en && (state == START || state == STOP || state == WAIT)) begin
                if (baud_cnt == 15)
                    baud_cnt <= 0;
                bit_cnt <= 0;
                baud_cnt <= baud_cnt + 1;
            end
            else if (baud_en && state == BIT) begin
                bit_cnt <= bit_cnt + 1;
                baud_cnt <= baud_cnt + 1;
            end
            else if (baud_en) begin
                bit_cnt <= 0;
                baud_cnt <= 0;
            end
        end
    end

    always @(*) begin //bloques definiendo el proximo estado según las condiciones
```

```
always @(*) begin //bloques definiendo el proximo estado según las condicion SV/Veril
    case(state)

        IDLE: begin
            nextstate = IDLE;
            if (baud_en && rx)
                nextstate = START;
        end

        START: begin
            if (baud_en && rx)
                nextstate = IDLE;
            else if (baud_en && baud_cnt < 7)
                nextstate = START;
            else if (baud_en && baud_cnt == 7 && !rx)
                nextstate = BIT;
            else nextstate = START;
        end

        BIT: begin
            if (baud_en && bit_cnt < 7)
                nextstate = BIT;
            else if (baud_en && bit_cnt == 7)
                nextstate = STOP;
        end

        STOP: begin
            if (baud_en && baud_cnt < 15)
                nextstate = STOP;
            else if (baud_en && baud_cnt == 15)
                nextstate = WAIT;
        end

        WAIT: begin
            if (baud_en && baud_cnt < 7)
                nextstate = WAIT;
            else if (baud_en && baud_cnt == 7)
                nextstate = IDLE;
        end

    endcase
end


always @(*) begin //bloques que definen las salidas
    case (state)

        IDLE: begin
            rx_rdy = 0;
        end

        START: begin
```

```
100             rx_rdy = 0;
101         end
102
103         START: begin
104             rx_rdy = 0;
105         end
106
107         BIT: begin
108             rx_data = rx_data << 1;
109             rx_data[0] = rx;
110             rx_rdy = 1;
111         end
112
113         STOP: begin
114             rx_rdy = 0;
115         end
116
117         WAIT: begin
118             rx_rdy = 1;
119         end
120     endcase
121   end
122
123 endmodule
124
```

Figura 18: Módulo receptor UART

Con el siguiente testbench:

```
`timescale 1ns / 1ps

module uart_rx_tb;
  reg rx;
  reg clk = 0;
  reg baud_en = 0;
  reg rst = 0;

  wire rx_rdy;
  wire [7:0] rx_data;

  uart_rx uut(clk, rst, rx, baud_en, rx_data, rx_rdy);


  initial begin
    #1600
    rx = 0;
    #1600
    rx = 1;
    #1600
    rx = 0;
    #1600
    rx = 1;
    #1600
    rx = 0;
    #1600
    rx = 1;
    #1600
    rx = 0;
    #1600
    rx = 1;
    #1600
    rx = 0;
    #1600
    rx = 1;



  end

  always #50 clk = ~clk;

  always #100 begin
    baud_en = 1;
    #1
    baud_en = 0;
  end

endmodule
```

Figura 19: Testbench receptor

Resultados de la experiencia

0.1. Parte analógica

Para poder armar el circuito descrito en la guía previa, presentamos inconvenientes principalmente porque en el laboratorio no se encontraban los componentes con los valores exactos que fueron calculados. Por ejemplo, no había un capacitor de $59,41\mu F$, por lo que lo tuvimos que resolver conectando en paralelo uno de 10μ con uno de 47μ, para tener en total 57μ. Consideramos que el 57 era suficientemente cercano al 59,41 como para que la diferencia en el resultado sea significativa. En la otra capacitancia usamos una de $22nF$ en vez de $22,61n$ también con el mismo criterio.

Para el caso del voltaje V_{ref}, optamos por sacarlo del voltaje V_{DD} a partir de un divisor de voltaje. En un principio calculamos los valores exactos de las resistencias del divisor para obtener un voltaje de $1,65V$. Sin embargo, por probablemente no idealidades de las resistencias nunca logramos que la salida sea exactamente la que queríamos. Es por eso que finalmente optamos por utilizar un potenciómetro y variarle la resistencia hasta que la salida sea justo la deseada.

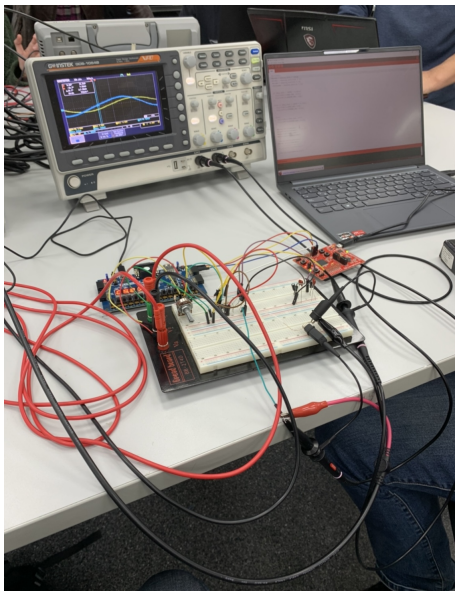

Figura 20: Implementación en laboratorio.

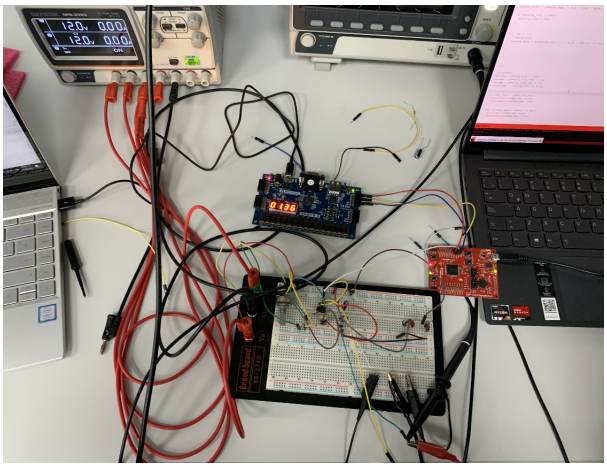

Figura 21: Implementación en laboratorio.

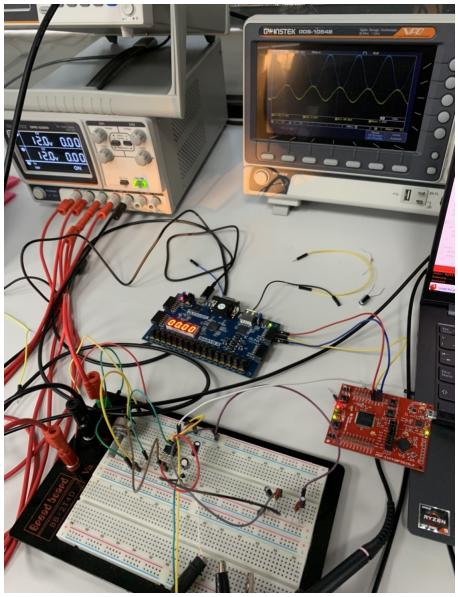

Figura 22: Implementación en laboratorio.

Para el diseño del amplificador inversor, optamos por utilizar un OpAmp LM741, el que tiene recomendaciones de alimentación entre ± 10[V] y ± 15 [V], por lo que optamos en alimentarlo con ± 12 [V]. En primera instancia, la salida del circuito no mostraba lo que debía mostrar (de hecho era prácticamente nula). Finalmente nos dimos cuenta de que era debido a que el OpAmp que estabamos usando estaba malo, por lo que debimos cambiarlo. La entrada y la salida del acondicionador de señal para cada caso particular se muestra a continuación:

a) Para una entrada nula.

Figura 23: Entrada nula.

b) Para una entrada fija en 6.5[V].

Figura 24: Entrada fija en 6.5 [V].

c) Para una entrada sinusoidal de 1 [Hz], 7 [V_{pp}] y componente continua de 6.5 [V].

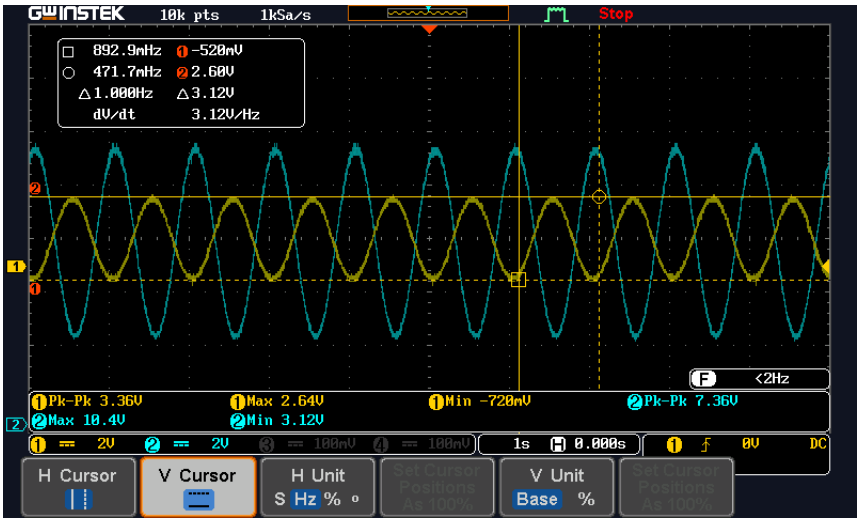

Figura 25: Entrada de 1 [Hz], 7 [V_{pp}] y off-set 6.5 [V].

d) Para una entrada sinusoidal de 1 [Hz], 1 [V_{pp}] y componente continua de 6.5 [V].

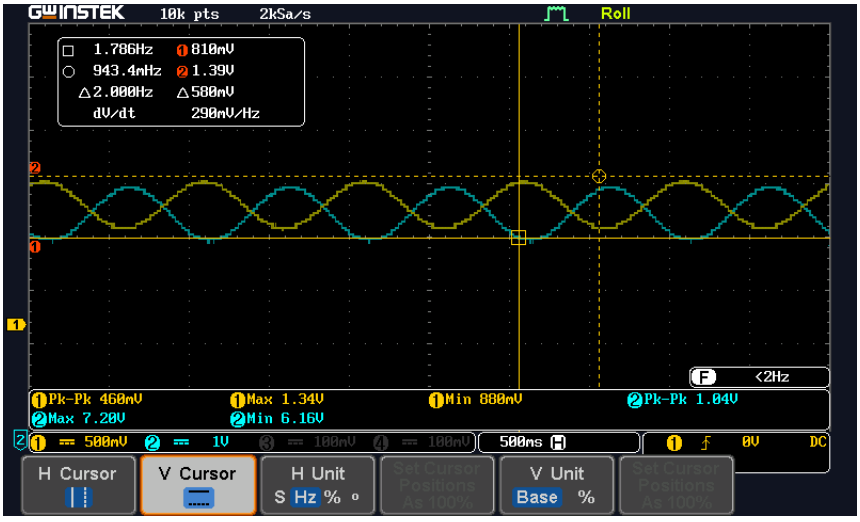

Figura 26: Entrada de 1 [Hz], 1 [V_{pp}] y off-set 6.5 [V].

e) Para una entrada sinusoidal de 20 [Hz], 7 [V_{pp}] y componente continua de 6.5 [V].

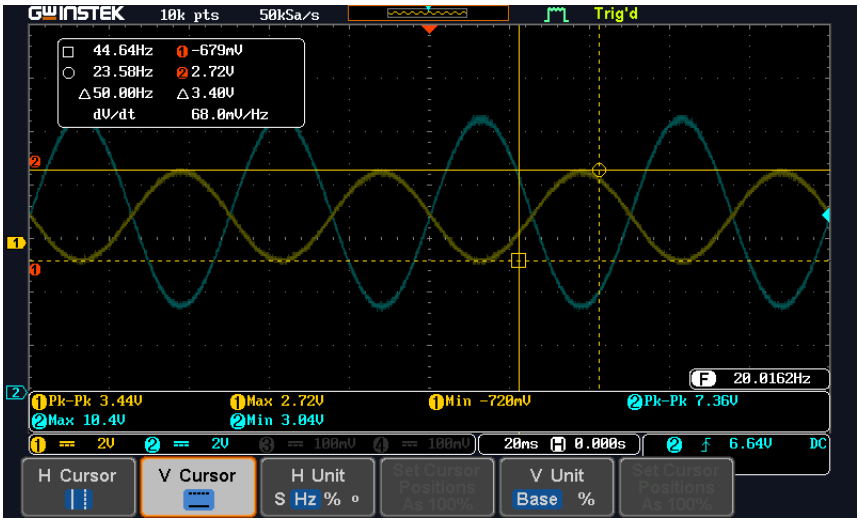

Figura 27: Entrada de 20 [Hz], 7 [V_{pp}] y off-set 6.5 [V].

f) Para una entrada sinusoidal de 20 [Hz], 1 [V_{pp}] y componente continua de 6.5 [V].

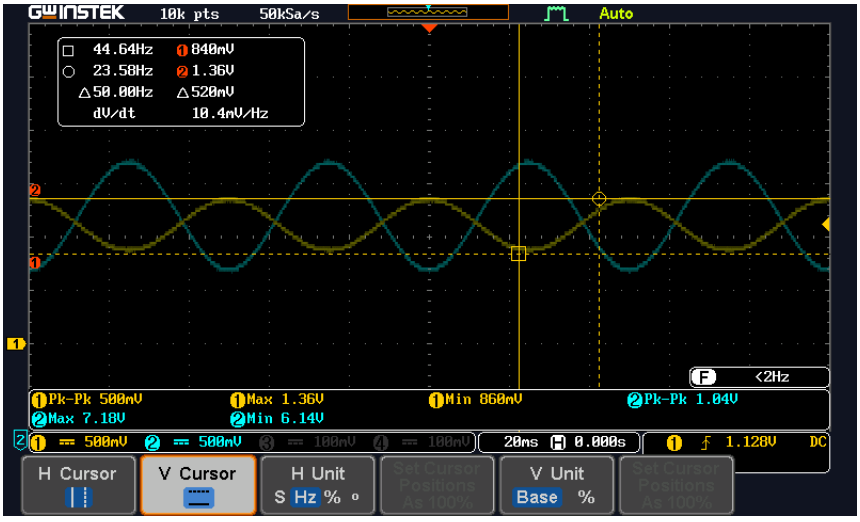

Figura 28: Entrada de 20 [Hz], 1 [V_{pp}] y off-set 6.5 [V].

g) Para una entrada sinusoidal de 400 [Hz], 7 [V_{pp}] y componente continua de 6.5 [V].

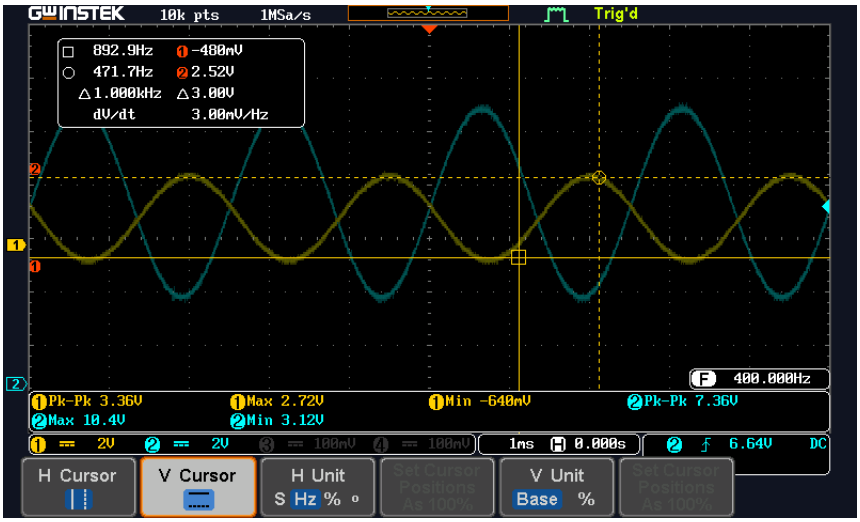

Figura 29: Entrada de 400 [Hz], 7 [V_{pp}] y off-set 6.5 [V].

h) Para una entrada sinusoidal de 400 [Hz], 1 [V_{pp}] y componente continua de 6.5 [V].

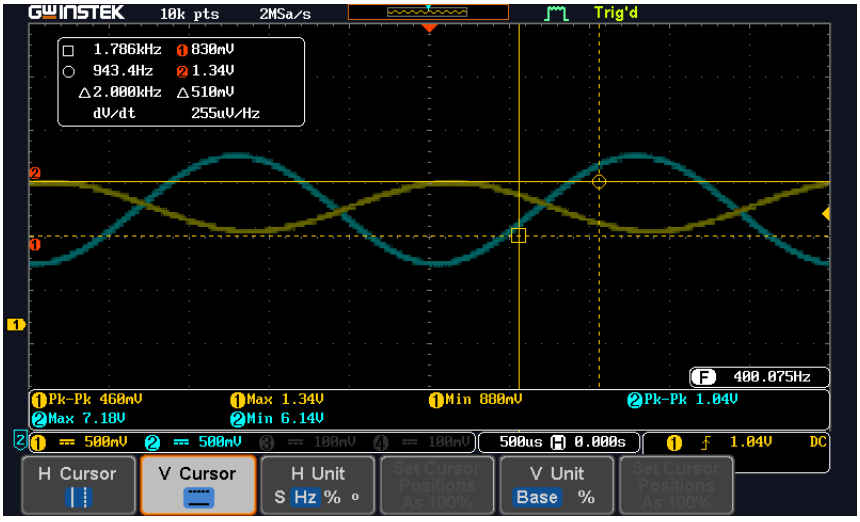

Figura 30: Entrada de 400 [Hz], 1 [V_{pp}] y off-set 6.5 [V].

i) Medición de desfases.

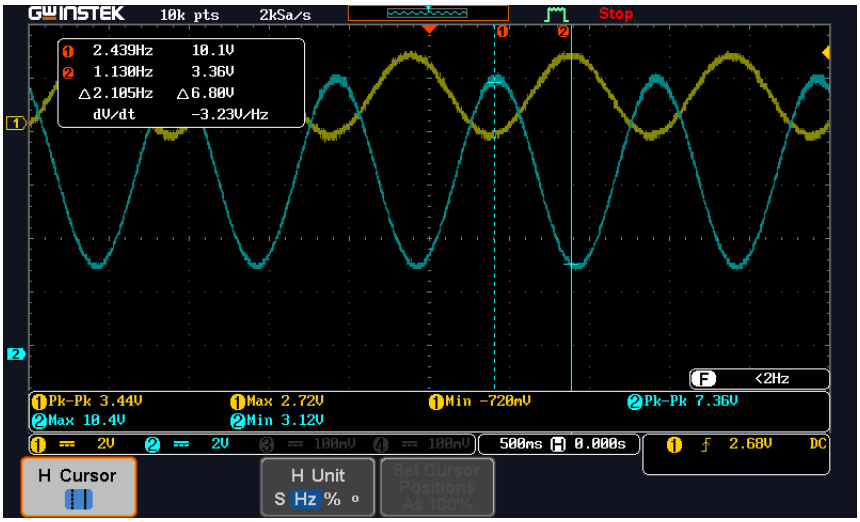

Figura 31: Medición punto (c).

$$\Delta\theta = 2\pi f[Hz] \cdot \frac{1}{\Delta f}$$

$$\Delta\theta = 2\pi 1[Hz] \cdot \frac{1}{2,105[Hz]} = 2,9849(rad) = 171,0214^o$$

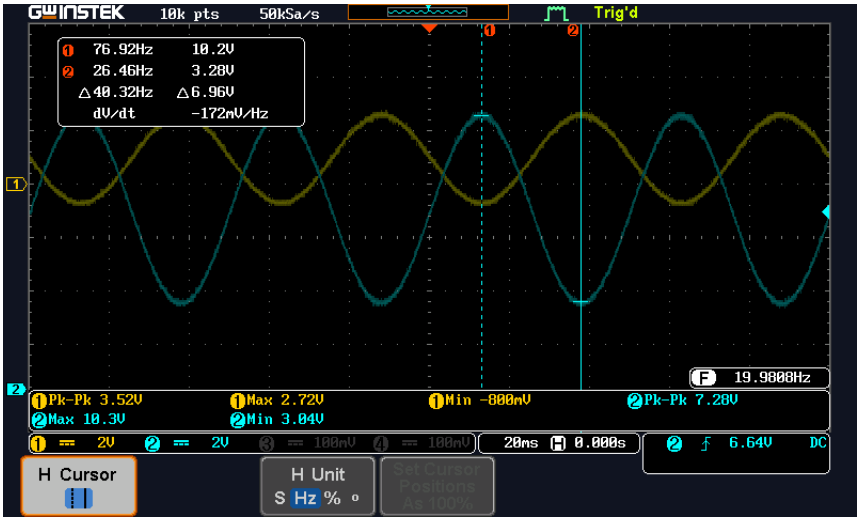

Figura 32: Medición punto (e).

$$\Delta\theta = 2\pi f[Hz] \cdot \frac{1}{\Delta f}$$

$$\Delta\theta = 2\pi \cdot 20[Hz] \cdot \frac{1}{40,32[Hz]} = 3,1167(rad) = 178,5714^o$$

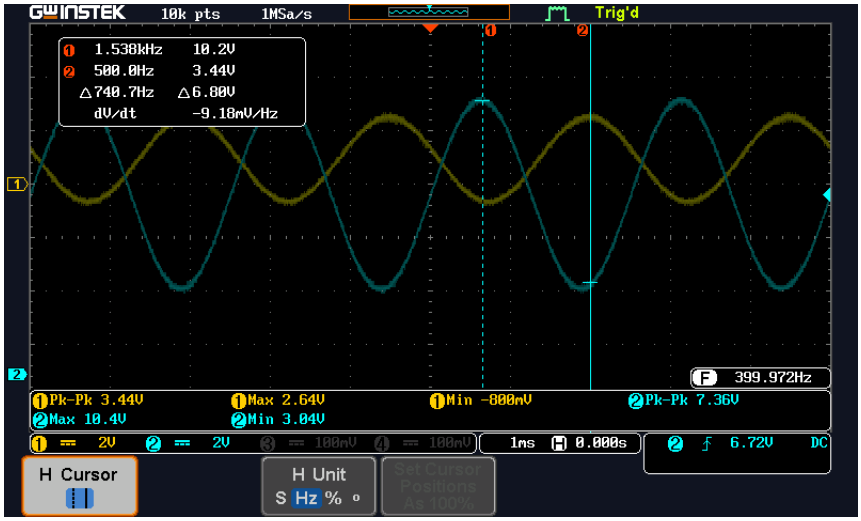

Figura 33: Medición punto (g).

$$\Delta\theta = 2\pi f[Hz] \cdot \frac{1}{\Delta f}$$

$$\Delta\theta = 2\pi \cdot 400[Hz] \cdot \frac{1}{740,7[Hz]} = 3,3931(rad) = 194,4107^o$$

0.2. Diagramas de bloques del sistema implementado

A continuación se mostrará un diagrama de bloques de todo el sistema implementado (incluyendo la parte analógica y digital).

En primer lugar se tiene obviamente el circuito analógico (filtro pasa banda), donde la entrada V_i correspondiente a una sinusoide de amplitud $7V$ y *offset* de $6,5V$, y la salida V_o es la señal luego de pasar por el filtro (con las características descritas en la guía previa). También se tienen las entradas de tierra de señal y de los voltajes de realimentación:

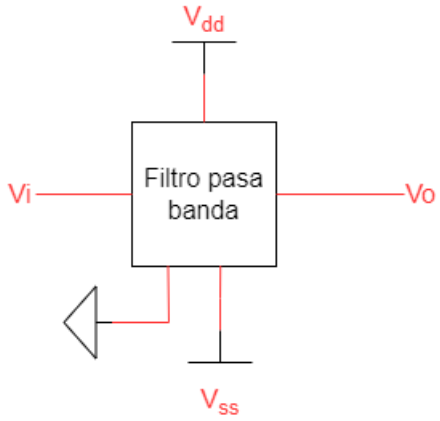

Figura 34: Diagrama filtro pasa banda

Luego viene la MSP. Esta recibe como input la señal analógica V_o y una señal binaria proveniente de un *switch* de la *Basys*. La MSP convierte la señal analógica a digital (es decir en una escala de 0 a 4095). Luego, esa señal hay que enviarla mediante UART a la Basys. Sin embargo, la Basys debe recibir los números en una escala del 0 al $3,3$ o del 3 al 10 dependiendo de si el switch está arriba o abajo. Por lo tanto, si el switch es 0, la MSP convierte en escala de 0 a $3,3$ y si es 1 convierte de 3 a 10. Luego, los números resultantes los separa en decena, unidad, décima y centésima y los envía uno por uno a la Basys. Además, si el número esta entre el 25 % de los valores más pequeños que podría tomar se enciende un LED rojo, si está entre el 75 % más grande se prende uno verde, y si está entre 25 % y 75 % se prenden ambos. El código se puede entender mejor con el siguiente diagrama:

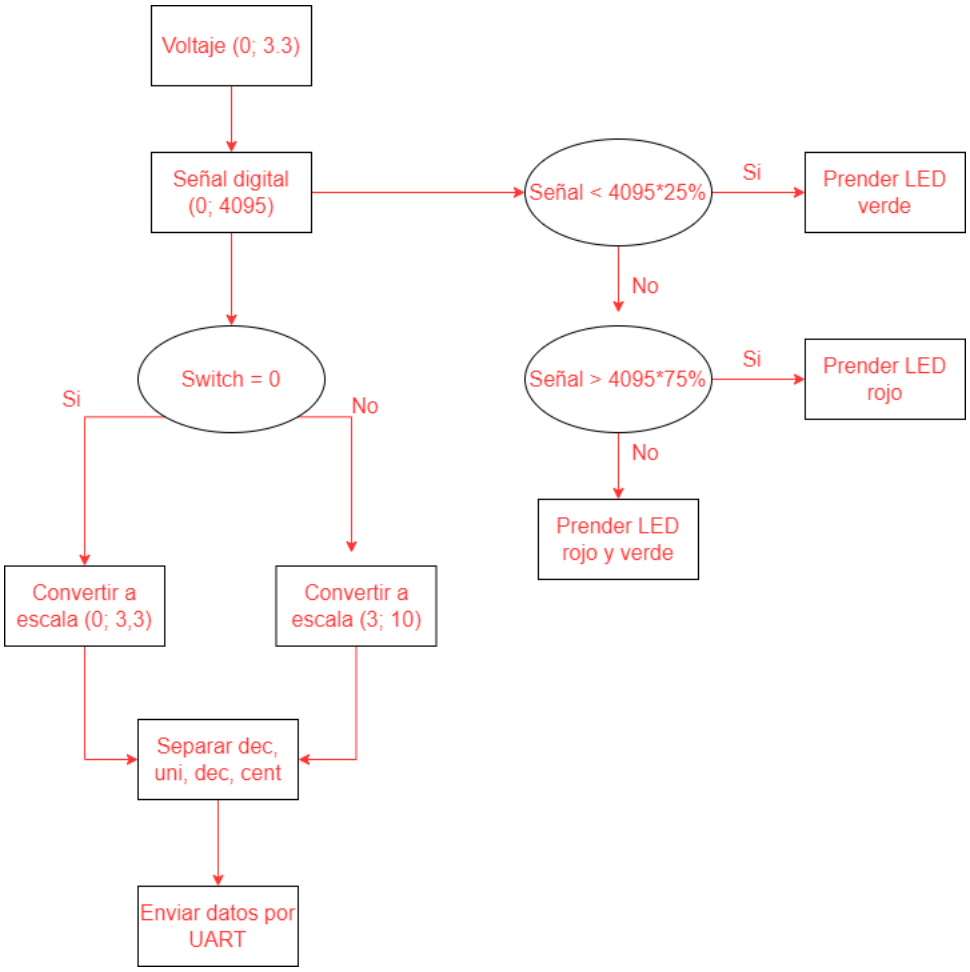

Figura 35: Algoritmo MSP

El bloque de la MSP sería así

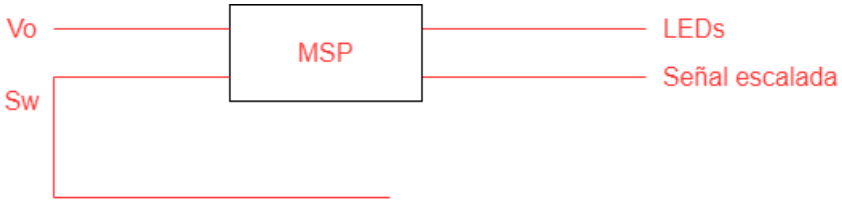

Figura 36: Bloque MSP MSP

Ahora, se presentarán los diagramas de bloque de los módulos de la Basys. **Solo se presentarán los bloques. El desglose de cómo funciona cada uno estará detallado en la próxima sección.**

Primero viene un bloque generador de *baud rate* que va conectado al receptor. Las entradas y salidas se presentan a continuación:

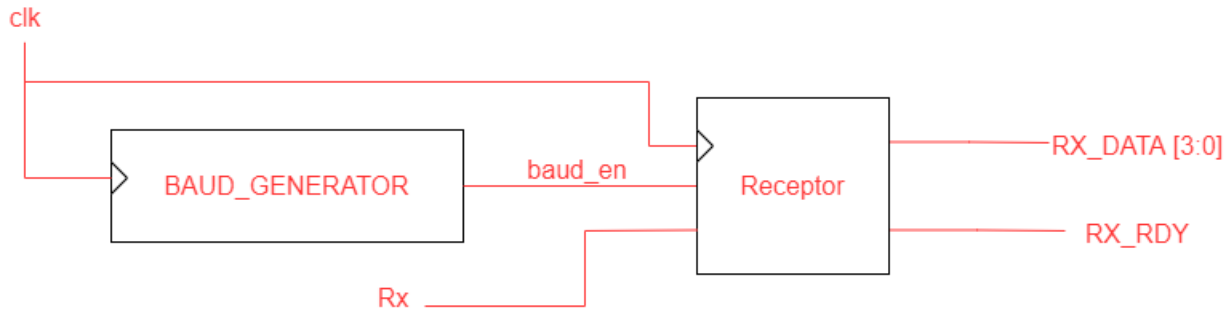

Figura 37: Baud generator y receptor

Los números que van llegando se van transformando a formato 7 segmentos gracias al módulo *BCDtoSeg*

Figura 38: Módulo BCDtoSeg

Luego, hay un módulo llamado *where* que, gracias a la señal *rx_rdy* actúa como contador para saber si se está recibiendo el dígito correspondiente a la decena, unidad, décima o centésima. Cada vez que *rx_rdy* es 1, aumenta en una unidad. Cuando llega a 4 (o sea que ya recibió los 4 dígitos correspondientes al número), vuelve a 0. La salida es simplemente la posición en el display de número que llegó. (si llega la decena va en el primer display, unidad en el segundo, etc):

Figura 39: Módulo where

Después, hay un buffer de 28 bits llamado *numero_grande*, que básicamente va guardando los números en formato 7 segmentos según la posición que le indica el *pos_disp*. Es decir, si *pos_disp* es 0, guarda el número en la posición [6:0], si es 1 lo guarda en [13:7] y así sucesivamente:

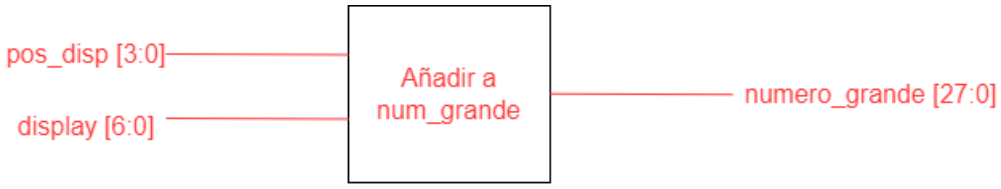

Figura 40: Se añade al buffer

Por último, el *numero_grande* se pasa por un módulo *SevenSegController* para poder mostrarlo correctamente en los display:

Figura 41: SevenSegController

No está demás recordar el switch, que mediante un comando assign arroja una salida que va directamente a la MSP:

Figura 42: Switch

Finalmente ya tenemos el circuito completo. Si juntamos todos los bloques se llega a lo siguiente:

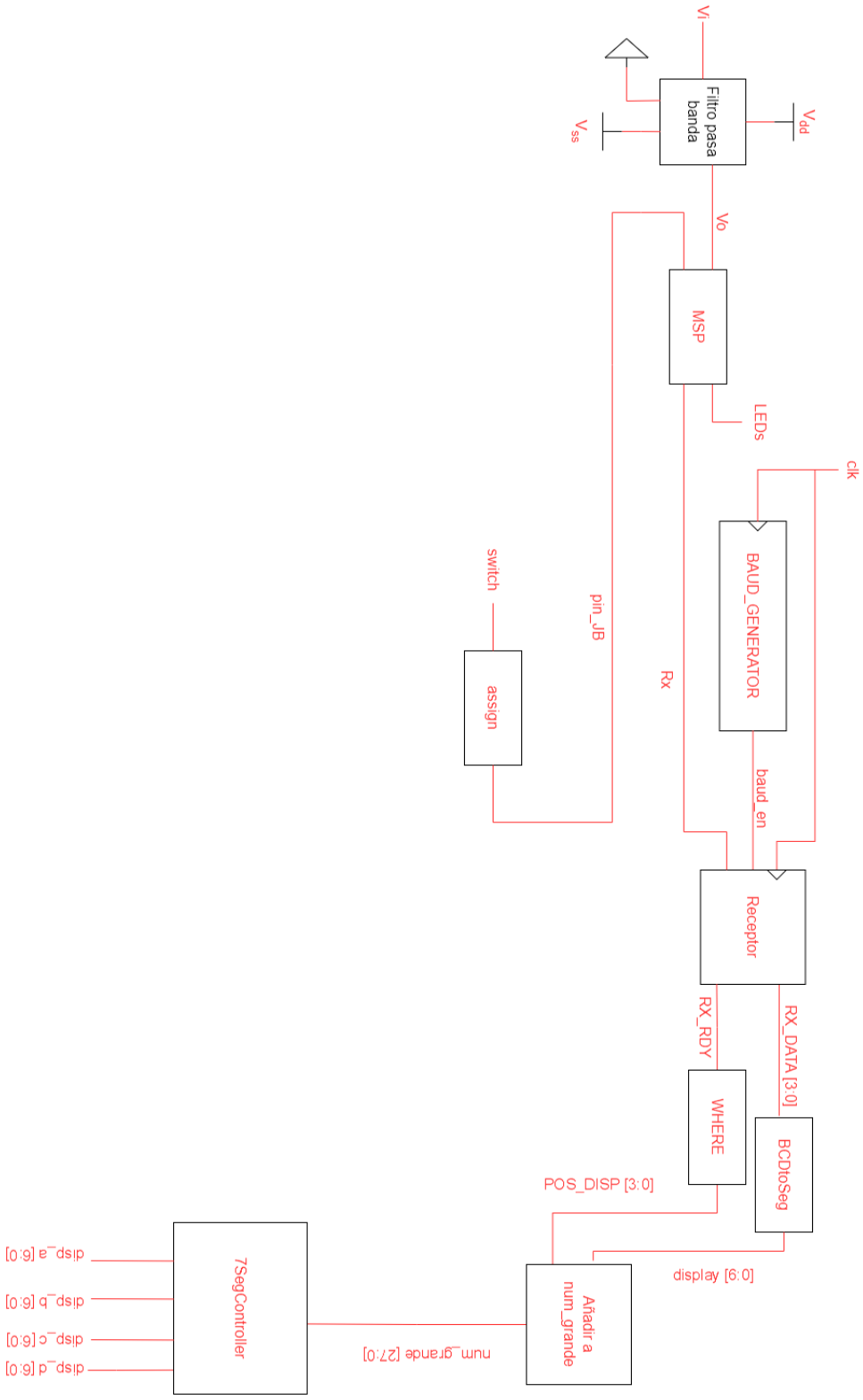

Figura 43: Circuito completo

Es importante notar también que el circuito terminó siendo bastante más complejo que el que habíamos planificado en la guía previa.

0.3. Detalle de los Módulos Digitales

En esta sección se detallará en detalle como funciona cada módulo de la Basys.

- *BAUD_GENERATOR:* La única función de este módulo es crear una señal que llamamos *baud_en* por baud enable. En el módulo receptor se explicará más en detalle para que sirve, pero por ahora, es una señal que se vuelve 1 un número de 16 veces en lo que dura un ciclo de recepción de un bit (considerar que se reciben 9600 bps). Por lo tanto, su única entrada es una señal de clock, y su única salida es el *baud_en*.
- *RX:* Este es el módulo encargado de recibir la información que viene de la MSP. Como se vio antes, una de sus entadas es el *baud_en* que es 16 veces más rápido que el baud rate. Esto es porque como en UART los relojes del transmisor y receptor no están sincronizados entonces podríamos perder información por no detectar que llegó un byte. Si se muestrea la señal que entra 16 veces mas rápido de lo que llega, nos aseguramos de que no se pierda nada. Obviamente las otras entradas de este módulo son un clock para generar su propio baud rate, y la señal de entrada que se quiere recibir a lo largo del tiempo. Dentro del módulo, se procesa la información como una máquina de estados. Se tiene el estado IDLE, que es básicamente cuando se está esperando que llegue algo. Luego el estado START que es cuando la máquina detecta que ya llegó el start bit, luego el estado BIT en donde se reciben los bits de información, después el estado STOP donde se recibe el stop bit, y por último el estado WAIT que espera a que el registro interno *baud_cnt* sea 7 para pasar al estado IDLE de nuevo. *baud_cnt* es un registro que cuentta el número de pulsos de *baud_en* y que se reinicia al llegar a 16. Por otro lado, también se tiene un registro llamado *bit_cnt* que cuenta el número de bits que han llegado hasta completar un byte. Por último, la salida del módulo es la información obtenida en formato de un byte, y una señal binaria llamada *rx_rdy* que es 1 cuando se termina de recibir la información, y es 0 el resto del tiempo. Al final de esta sección hay un diagrama de la máquina de estados en donde se muestra el correcto funcionamiento de esta junto con el paso de un estado a otro.
- *BCDtoSeg:* Como es sabido, para poder mostrar un número en el display de 7 segmentos este tiene que estar en un formato especial correspondiente a un número de 7 bits, donde cada bit representa un segmento del display. Por lo tanto, la entrada es el byte que se recibió en formato BCD y la salida es el mismo número pero en formato 7 segmentos. Este módulo es simplemente un *case* donde las entradas posibles son todos los números del 0 al 9, y las salidas son la conversión de estos al formato antes descrito.
- *WHERE:* Este es un contador que, gracias a la señal *rx_rdy* salida del receptor, indica cual es el byte que está llegando. Nosotros enviamos 4 bytes para hacer un número. El correspondiente a las décimas, unidades, decenas y centenas. Entonces cada vez que llega uno, la salida del módulo where aumenta en 1 número. Cuando esta llega a 4 significa que ya se recibió el número completo, por lo que se vuelve a poner en 0. La única entrada de este módulo es el *rx_rdy* y la salida es una llamada *pos_disp* que es simplemente un número entre el 0 y el 3.

- Añadir a *numero_grande*: Este en realidad no es un módulo de por sí, ya que es código que se encuentra dentro del módulo MAIN, sin embargo cumple con la función de un módulo. Básicamente se tiene un número de 28 bits llamado *numero_grande* que junta los 4 números descritos en la explicación del módulo where. Entonces, si el número corresponde a la decena, lo guarda entre los bits [6:0], si corresponde a las unidades lo guarda en [13:7], si corresponde a las decenas lo guarda en [20:14] y si es una centena lo guarda en [27:21]. Se sabe a que corresponde el número gracias a la señal obtenida del módulo where que lo indica.
- switch: Este tampoco es un módulo de por sí porque también se encuentra dentro del main, pero básicamente a uno de los pines JB de la Basys se le asigna mediante el comando assign el mismo número que entrega el primer switch.
- *SevenSegController*: Este módulo sirve para poder mostrar los números guardados en *numero_grande* en los display. Por como está hecha la Basys no se puede mostrar números distintos al mismo tiempo en los display, por lo que lo que se necesita un módulo especial que lo logre. Lo que hace es ir prendiendo y apagando cada display con su respectivo número pero de manera muy rápida, de forma de que el ojo humano crea que está todo prendido a la vez.

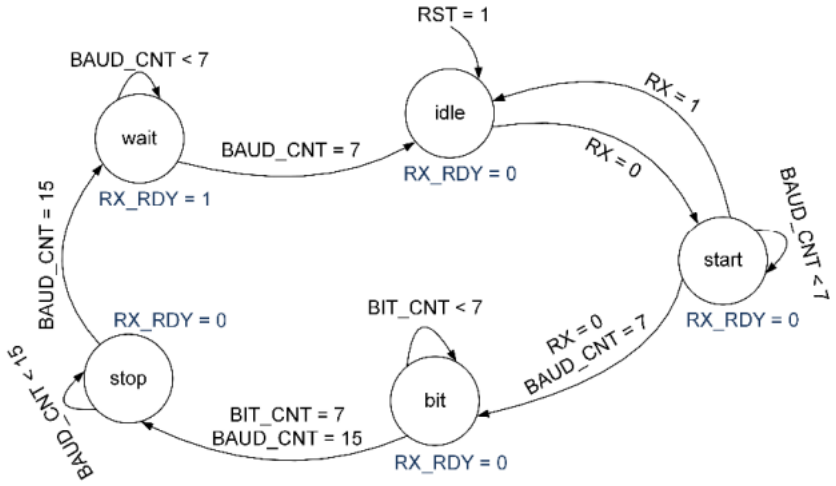

Figura 44: FMS de Rx UART

0.4. Discusión y análisis de los resultados

Primero, aquí hay un pequeño video del circuito funcionando.

La parte analógica, nos salio no exactamente igual pero muy parecida a las simulaciones. En las simulaciones calculamos los valores de los componentes para que la fase en $20Hz$ sea de 0 grados respecto a la entrada, y para que en $1Hz$ y $400Hz$ tenga un valor absoluto de 15 grados. En la realidad, en $1Hz$ nos dio una fase de $15,1$ grados y en $400Hz$ de $10,6$ grados. Creemos que esto se puede deber principalmente a las no idealidades de los componentes, ya que tanto las resistencias como las capacitancias tienen cierto margen de tolerancia. También puede deberse a que no pudimos usar los valores 100 % exactos de los componentes porque en el laboratorio no había, por lo que teníamos que usar valores muy cercanos (pero no iguales). El *offset* de la salida si nos dio exacto el valor que necesitábamos pero porque pudimos corregir el error con un potenciómetro. A grandes rasgos, se podría decir que los resultados obtenidos a partir del filtro son suficientemente satisfactorios y parecidos a lo esperado. Presentan un error totalmente esperable y tolerable.

Con respecto a la parte de la MSP específicamente, la recepción de datos funcionaba de manera correcta. Gracias a la opción de "imprimir" datos que ofrece Energia pudimos comprobar que los datos que recibíamos cuadraban con lo esperado. Las luces que implementamos que se prendían de acuerdo al rango de voltaje que se estaba recibiendo también funcionaban bien y ayudaron a comprobar el funcionamiento. La conversión de datos para poder ser enviados la hicimos con una ecuación de la recta, y también comprobamos su funcionamiento "imprimiendolos"para ver que si se hacía bien. Con respecto al envío de datos, primero comprobamos que funcionara enviando datos directamente al osciloscopio para ver si en la pantalla se veían los start y stop bits, y el byte enviado, y si se veía. Después lo comprobamos enviando los datos a Realterm (terminal serial) y también se veían bien. En general, se puede decir que la MSP funcionaba totalmente de acuerdo a lo esperado.

Finalmente, falta hablar de la parte de la Basys. Probamos cáda módulo por separado y funcionaban de manera correcta. Sin embargo al juntarlos todos, en los display se veía como que los números avanzaban demasiado rápido (incluso a bajas frecuencias). Intentamos agregarle un delay al código en la MSP y ahí si funcionó todo bien. Esto significa que simplemente era un problema de timing. Obviamente la idea no es tener un delay porque eso hace que se pierdan datos, sin embargo sin este era realmente imposible ver lo que estaba pasando. Dado que de verdad nos aseguramos que todos los módulos estaban funcionando bien, la única explicación que tenemos es que la frecuencia con la que se prenden y apagan los números en el display (controlado por el 7SegController) no cuadraba con lo que necesitábamos. Lamentablemente no tuvimos tiempo para intentar arreglando eso antes de la evaluación.

0.5. Conclusiones

En primer lugar, en esta experiencia aprendimos mucho gracias a la enorme cantidad de problemas que tuvimos. Durante el armado del circuito nos tocaron jumpers malos, una sonda mala, 2 OpAmp quemados, se nos descalibró el ADC de la MSP y tuvimos que cambiar la tarjeta. Perdimos mucho tiempo en estas cosas, pero al mismo tiempo creemos que nos ayudó a ganar experiencia para próximas ocasiones, donde gracias a esto vamos a tener más recursos para resolver problemas.

Respecto al armado del circuito en sí pudimos comprobar de primera mano las no idealidades de los componentes y como estos afectan al resultado final. De todas formas creemos que podemos identificar de manera correcta cuando estas no idealidades son lo suficientemente marginales como para que no afecten de manera significativa. Cabe mencionar que ambos integrantes del grupo hicimos el laboratorio de mediciones de manera online, entonces en verdad este fue el primer circuito que armamos y que funcionara, lo que también es muy bueno.

Aprendimos a convertir datos de analógico a digital, y también a trabajar con ambos "mundos."al mismo tiempo, lo que es sumamente interesante. En la parte de la MSP creo que reforzamos mucho todo lo que es comprobar que todo esté funcionando bien. Esto lo logramos haciendo pruebas con el osciloscopio, con terminales seriales, enviando datos, etc. Creemos que esto es una parte fundamental del trabajo ya que es muy difícil que todo salga bien a la primera, pero si sale mal también es difícil encontrar el por qué. Mientras más recursos tenga uno para poder resolver problemas, más posibilidades hay de poder optimizar el trabajo.

A pesar de que no presentamos mayores dificultades trabajando con los módulos en verilog, nos hubiese gustado poder resolver el problema final que tuvimos con el display que avanzaba muy rápido. Para una próxima ocasión al menos ya sabemos que podemos hacer si nos vuelve a pasar lo miso. Obviamente no estamos del todo seguros de si ese era realmente el error, pero al menos parece ser lo más lógico y sin duda sería lo primero que se debería intentar.

Bibliografía

- Diferencias entre comunicación en serie y paralelo: `https://www.totalphase.com/blog/2020/10/differences-between-serial-parallel-communication/`
- Información sobre transmición y recepción UART: Taller 13 de Sistemas Digitales IEE2713 período, 2021-2. Ayudantes Miguel Espinoza, Felipe Rubio y Javier Villegas.
- Código módulo receptor: Hecho por ayudantes de Sistemas Digitales IEE2713 período, 2021-2. Ayudantes Miguel Espinoza, Felipe Rubio y Javier Villegas.