Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica
IEE2473- Laboratorio de Electrónica ANalógica y Digital

Acondicionamiento y recuperación de señales

Grupo 17

Kevin Justiniano — Juan Ignacio Lorca

1. Acondicionamiento de señal analógica

Haciendo uso del OpAamp LM741 se crea una topología atenuadora inversora, con una ganancia en el pasabanda (20 [Hz]) de $G = 0{,}33$ con un desfase de 10° para la frecuencia de 1Hz y 10° para la frecuencia de 400Hz.

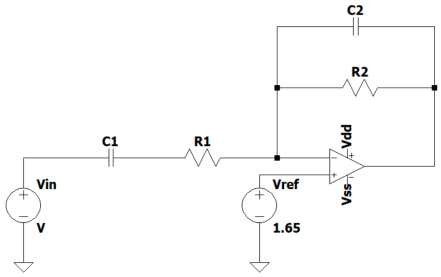

Figura 1: circuito inverso atenuador

Los valores a usar en el laboratorio fueron:

- C1 = 3.2μF
- C2 = 1.5nF
- R1 = 150kΩ
- R2 = 51kΩ

Las alimentaciones del OpAmp Vdd y Vss fueron fijadas a 12 y -12 V respectivamente, y en base a Vdd se diseñó un divisor de voltaje con un potenciómetro de 10kΩ en serie con un resistor de 5.1kΩ. La ecuación para el diseño del circuito anterior fue la siguiente:

$$G(s) = -\frac{sC_1R_2}{(1+sR_1C_1)(1+sR_2C_2)} \tag{1}$$

la cual, en su diagrama de bode considera magnitud y fase para sinusoides de distinta frecuencia.

Para lo anteriormente mencionado, se realizaron pruebas de entradas específicas, donde la señal amarilla corresponde a V_{in} mas su componente contínua, la v_{out} mas su componente contínua y la magenta a V_{in}.

a. Entrada nula

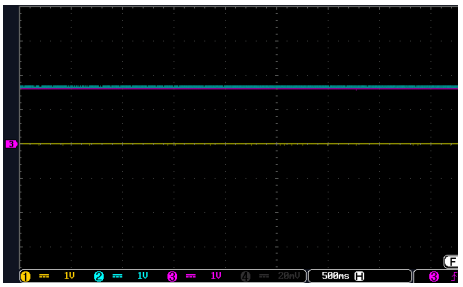

Figura 2: (a)

b. Entrada fija de 6.5V

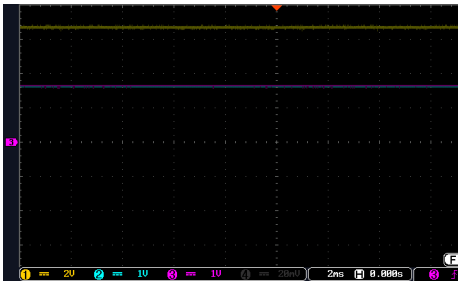

Figura 3: (b)

c. Entrada sinusoidal 1Hz, $7Vpp$ y componente continua de 6.5V

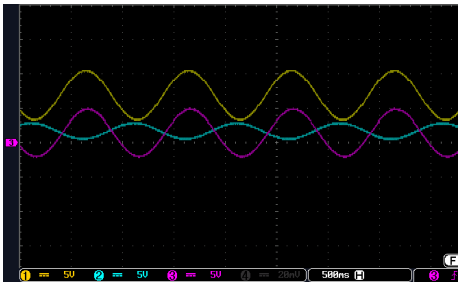

Figura 4: (c)

d. Entrada sinusoidal 1Hz, $1Vpp$ y componente continua de 6.5V

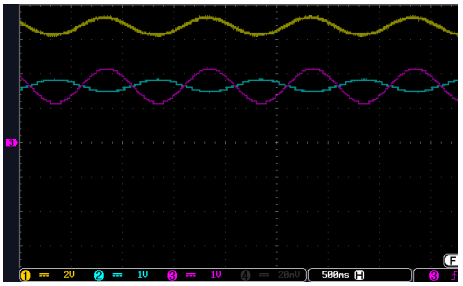

Figura 5: (d)

e. Entrada sinusoidal 20Hz, $7Vpp$ y componente continua de 6.5V

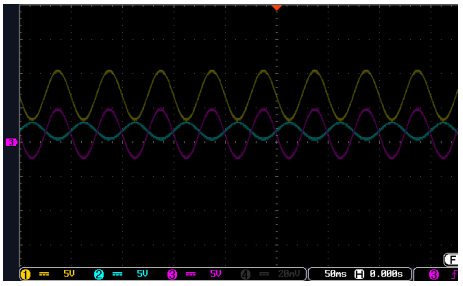

Figura 6: (e)

f. Entrada sinusoidal 1Hz, $1Vpp$ y componente continua de 6.5V

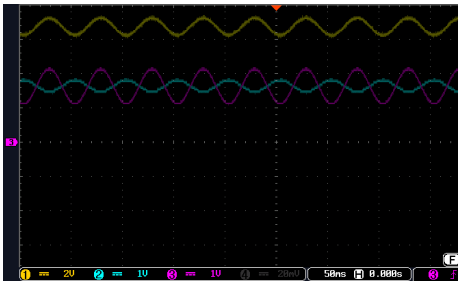

Figura 7: (f)

g. Entrada sinusoidal 400Hz, $7Vpp$ y componente continua de 6.5V

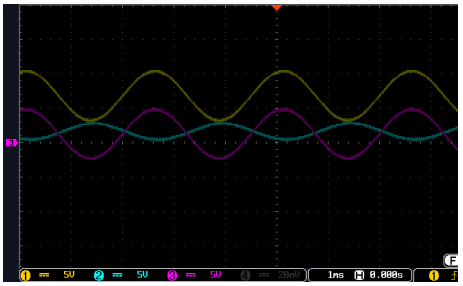

Figura 8: (g)

h. Entrada sinusoidal 1Hz, $1Vpp$ y componente continua de 6.5V

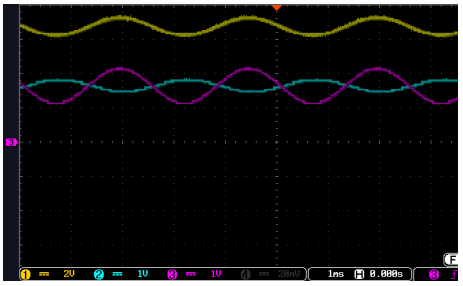

Figura 9: (h)

i. Comparación de desfases.

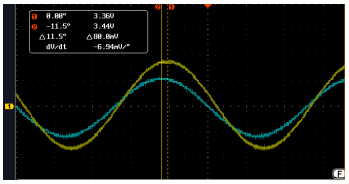

Figura 10: f=1Hz

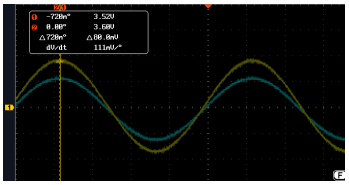

Figura 11: f=20Hz

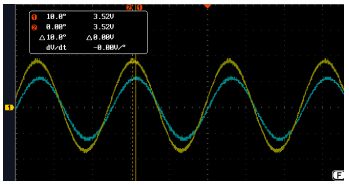

Figura 12: f=400Hz

De la figura 2 a la 9, la señal señal amarilla es la entrada, mientras que la celeste es la de salida; mientras que las figuras 10, 11 y 12 muestran los desfases. Se puede ver como para los valores ocupados los desfases son de 11.5°, 0° y 10° para las frecuencias de 1, 20 y 400 Hz respectivamente. Es notable que los desfases son similares a los de la simulación, ya que se usaron desde ese momento valores reales y no los obtenidos directamente de los cálculos, y existe una pequeña variación por la presición de los elementos circuitales.

2. MSP y digitalización de señal

Para esta parte de la experiencia, la tarjeta MSP430f5529 fue utilizada con fines de digitalización y procesamiento de señal. La principal función de esta etapa fue agrupar el señal en 4 bytes, donde los 4 bits menos significativos son un número en BCD y los 4 más significativos son una etiqueta de posición.

2.1. ADC

Para el muestreo de la señal se utilizó el módulo ADC12 de la MSP430x2xx que entrega la señal analógica en 12 bits según la ecuación:

$$N_{ADC} = 4095 \cdot \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \tag{2}$$

El rango de voltajes de entrada V_{in} utilizado fue de [0.0, 3.3] y se definieron las referencias internas cómo: $V_{R+} = 3{,}3\ [V]$ y $V_{R-} = 0\ [V]$. De esta manera el valor más alto para la señal fue $2^{12} - 1 = 4095$ y el más bajo 0 (en 12 bits).

En cuanto al código y los pines utilizados, el pin analógico de lectura fue el P6_0 asociado a la entrada analógica A0. Para su lectura o muestreo, se utilizó el comando **analogRead(P6_0)** del framework Arduino almacenando el valor instantáneo en un entero como variable auxiliar.

2.2. Algoritmo

Para descomponer el número en su decenas, unidades, primer decimal y segundo decimal es necesario obtener el valor instantáneo y flotante del voltaje muestreado. Para ello, se realiza el siguiente mapeo al la variable entera auxiliar:

$$V_{in} = N_{ADC} \cdot \frac{V_{R+} - V_{R-}}{4095} + V_{R-} \tag{3}$$

Luego, para la obtención de los valores mencionados anteriormente se aplica el siguiente algoritmo al valor V_{in}:

Algorithm 1:

Data: V_{in}
Result: $Unit$
$v_{init} \leftarrow V_{in} \cdot 100$;
$i = 0$;
while $i + 1 < 5$ **do**
  $Unit = integerPart(\frac{v_{init}}{10^{3-i}})$;
  $v_{init} = v_{init} - UnitS \cdot 10^{3-i}$;
  $i = i + 1$;
end

El algoritmo anterior tiene como salida $Unit$ el cual respresenta a un número entero en binario. Debido a que el el algoritmo asegura entregar números entre 0 y 9, los 4 dígitos menos significativos que retorna el algoritmo son el valor de dicho entero en BCD, por lo que queda agregar la etiqueta de posición en los otros 4 dígitos que quedan.

Para esto se definieron 4 variables escritas en formato uint8 que representan si el valor al que se asocien es decena, unidad, décima o centésima; estas son:

Etiqueta	Nombre variable	Valor
Decena	bcd1	0b00000001
Unidad	bcd2	0b00000010
Décima	bcd3	0b00000100
Centésima	bcd3	0b00001000

Finalmente, las 4 salidas que toma $Unit$, son denominadas en base al valor que deben de tomar al ser mostradas enla FPGA. Para que esta última sepa en que orden mostrar los valores, las variables *bcd* y los resultados $Unit$ se concatenan y generan una nueva variable en formato uint8 denominada $UnitS$ de la siguiente forma:

```
uint8_t UnitS = (bcd_i << 4) | Unit
```

Entonces, suponiendo que la MSP recibe un valor 01.65 en su pin analógico, se obtendría que:

Posición	Valor	bcd_i	Unit	UnitS
Decena	0	0b00000001	0b00000000	0b00010000
Unidad	1	0b00000010	0b00000001	0b00100001
Décima	6	0b00000100	0b00000110	0b01000110
Centésima	5	0b00001000	0b00000101	0b10000101

luego, $UnitS$ es lo que se estaría enviando por transmisión UART.

2.3. Blinking LEDs

El valor que retorna el ADC se envía a la función `compare` la cual retorna un valor entero entre 0 y 2. Si el valor que recibe la función es menor al 25 % de 3.3, se retorna un 0; si es que se encuentra el 25 % y 75 %, se retorna un 1; y en caso contrario se retorna un 2. El siguiente código ejemplifica lo anteriormente mencionado:

```
int compare(float a){
  int value = 0;
  if (a < 0.25 * 3.3){
    value = 0;
  } else if (0.25 * 3.3 < a  and  a < 0.75 * 3.3){
    value = 1;
  } else{
    value = 2;
  }
  return value;
}
```

Los valores que retorna `compare` son leídos por un conjunto de condicionales, donde si el valor es 0, no se enciende ningún LED. Si se retorna un 1 se enciende el LED verde que va vinculado al pin $P4\_7$, por lo que se envía una señal de `HIGH` a dicho pin. Por último, si se retorna un 2, se enciende el LED rojo que va vinculado al pin $P1\_0$, por lo que se envía una señal de `HIGH` a dicho pin.

Debido a que las señales de `HIGH` son estacionarias, previo a entrar al condicional se envían señales de `LOW` a ambos pines.

2.4. UART TX

Los valores que toma $UnitS$ son guardados en las siguientes 4 variables:

Posición	Nombre variable
Decena	thUnitS
Unidad	ctUnit
Décima	dcUnitS
Centésima	unUnitS

Finalmente, a través del comando `write`, y definiendo previamente a Serial1 como el canal tansmisor a un baud rate de 115200, se emiten las variables $UnitS$ tal que:

```
Serial1.write(thUnitS);
Serial1.write(ctUnitS);
Serial1.write(dcUnitS);
Serial1.write(unUnitS);
```

3. FPGA

Para la implementación del protocolo de comunicación de recepción de datos y el despliegue de estos en el display de 7 segmentos de la FPGA se siguió el siguiente diagrama de bloques (actualizado) para su codificación en el lenguaje *Verilog* de Vivado:

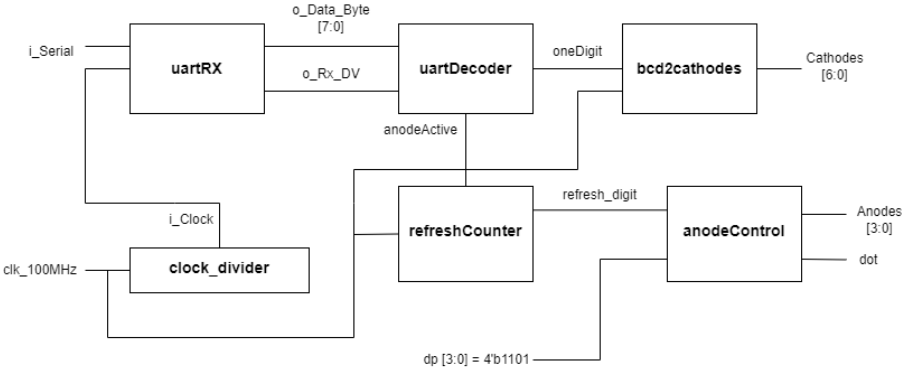

Figura 13: Diagrama de Bloques FPGA

De acuerdo al diagrama, el sistema corresponde a uno de tipo MIMO con los siguientes inputs y outputs:

1. **Inputs:**
   a) clk_100MHz: representa al clock maestro de la FPGA.
   b) i_Serial: representa la señal codificada en UART de acuerdo al puerto serial de la MSP. Esta señal posee una sección IDLE, 1 bit de partida, 8 bits para los datos y 1 bit final.
2. **Outputs:**
   a) *Cathodes* : representan los cátodos asociados al display de 7 segmentos. Estos son comunes para los 4 ánodos.
   b) *Anodes* : son los 4 ánodos del display de 7 segmentos. Si queremos encender los cátodos correspondientes al número enviado por la MSP, debemos enceder el ánodo respectivo (1 *lógico*) y apagar los cátodos asociados a ese número (0 *lógico*).
   c) *dot*: representa al punto que separa a los valores enteros de los decimales para el display de números flotantes en la FPGA.

Adjunto a este informe está el zip *verilogCodes* que contiene los códigos de los módulos enseñados en formato de texto (.txt).

3.1. Módulos FPGA

Para su implementación se utilizó una envoltura o *wrapper* de todos estos en un módulo llamado **top**. Adjunto a este informe, en el archivo codes.zip se encuentran los códigos en formato .txt.

3.1.1. clock_divider:

Módulo encargado de la división de la frequencia del reloj. Su código se encuentra parametrizado por un valor de división de acuerdo a:

$$div\_val = \frac{100\ [MHz]}{2 \cdot f^*} - 1 \tag{4}$$

dónde la frecuencia f^* es la frecuencia del reloj deseada.

3.1.2. uartRx:

Este módulo representa el receptor encargado de leer la información serial i_Serial enviada por la MSP a través del uso de una máquina de estados con los estados s_IDLE, s_ST_BIT, s_DATA_BITS y s_STP_BIT, los cuales son las diferentes partes del mensaje codificado en UART. Para ello, es necesario que este módulo lea a la misma tasa de bits enviados (115200 de baudRate). Esto último se hace utilizando un divisor de reloj para lograr que un contador de reloj dado por $CLKS\_PER\_BIT$ sea 16 y se logre hacer el *sampling* de los bits justo en medio de la información serial enviada.

$$CLKS\_PER\_BIT = \frac{clock\_divider(100\ [MHz])}{baudRate} \tag{5}$$

Cómo salidas, este módulo posee 2: un registro que representa la data de 8 bits o_Data_Byte dado por el número en BCD y su respectiva etiqueta que le indica el lugar en los ánodos del display a aparecer, y el registro o_Rx_DV que es el bit que indica que se está en el último estado s_STP_BIT y por ende va a comenzar un nuevo ciclo con un nuevo mensaje.

3.1.3. uartDecoder

Este módulo posee como entrada la información enviada por **uartRx** y salidas *anodeActive* y *oneDigit*. Su forma de funcionar es através de otra máquina de estados que va llenado 2 registros de 4 bits cada uno en cada estado. Para gatillar el pasar de un estado a otro se utiliza el bit enviado del módulo anterior o_Rx_DV que indica que se va a enviar un nuevo número por UART. Así el primer registro se llena con la información del número en BCD que se envió y se pasa a la salida *oneDigit* y el segundo con la etiqueta posicional que este lleva y se la pasa a la salida *anodeActive*.

3.1.4. Módulos Display 7 segmentos: refreshCounter, bcd2cathodes y anodeControl

Estos módulos son la base o los módulos *core* para realizar un display de 7 segmentos de cátodo común en la Basys 3. Sus códigos son muy similares a los que se pueden ecnontrar en la documentación y tutorial de FPGA 4 Students. A continuación una breve descripción de cada uno:

1. **refreshCounter:** este módulo controla el periodo de refresco del conjunto de ánodos y cátodos. Dado que no se pueden encender los 4 ánodos a la vez porque sólo se poseen 8 cátodos, este módulo, controlado por *anodeActive*, indica qué anodo a activar y sus respectivos cátodos para representar el número enviado. Esto útlimo lo hace a través de $refresh\_digit$.

2. **bcd2cathodes:** asigna los cátodos a 0 *lógico* o apagados con tal de encender los diodos respectivos para iluminar el número enviado por *oneDigit*.

3. **anodeControl:** asigna el ánodo común para los 8 cátodos a estar en 1 *lógico* con tal de enceder los leds del display de 7 segmentos.

A modo de conclusión, la estructura del código lo que hace es tomar la información contenida en o_Data_Byte y tratar de enviarla lo más rápido posible al display de 7 segmentos asociado a la etiqueta contenida en los 8 bits de datos.

4. Conclusiones

Como grupo podemos concluir que la experiencia en su totalidad fue muy nutritiva en cuanto al aprendizaje que uno se puede llevar de ella. Como futuros ingenieros eléctricos creemos que es fundamental saber usar las distintas funcionalidades y módulos que nos ofrecen los controladores y FPGAs. De esta forma, se logró interiorizar el buen uso de estas, además de comprender que los microcontroladores son especialmente útiles en medida de lograr instrucciones complejas, mientras que la FPGA es especialmente útil de recibir señales y operarlas de forma paralela, haciendo que varias señales sean trabajadas de forma ágil y rápida.

Del mismo modo, es importante saber medir señales analógicas en los distintos instrumuentos que ofrece el laboratorio y entender como funcionan los procesos de discretización, como el del ADC. De esta manera, esta experiencia nos logró introducir al uso del ADC12 y los distintos puertos seriales del microcontrolador MSP-EXP430F5529LP y también la implementación del estándar comunicación UART entre esta última con la FPGA Basys3.