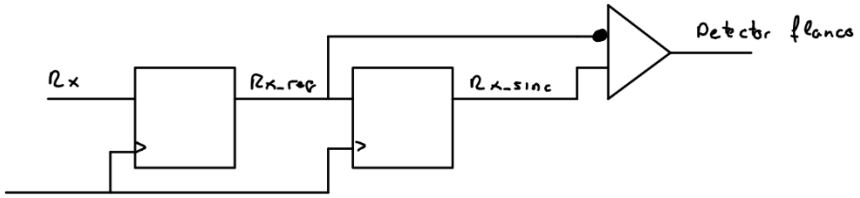
Miguel Aliende **Adrián Lázaro**

Memoria Sistemas Electrónicos de Telecomunicaciones

Serial RX

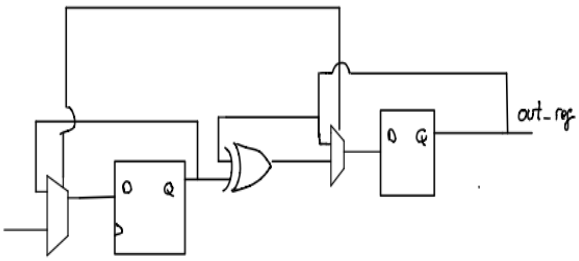
Lo que hace es básicamente recibir datos en serie y transformarlos en un bus de 10 bits. Para esto utilizamos un contador de 10 para ir guardando cada uno de los bits, otro contador de 868 que en el primer bit que se detecte se iniciará en 434 en vez de en 0 para situarnos justo en mitad de el bit que es donde es mas estable. Para el detector de flancos usamos la siguiente estructura:

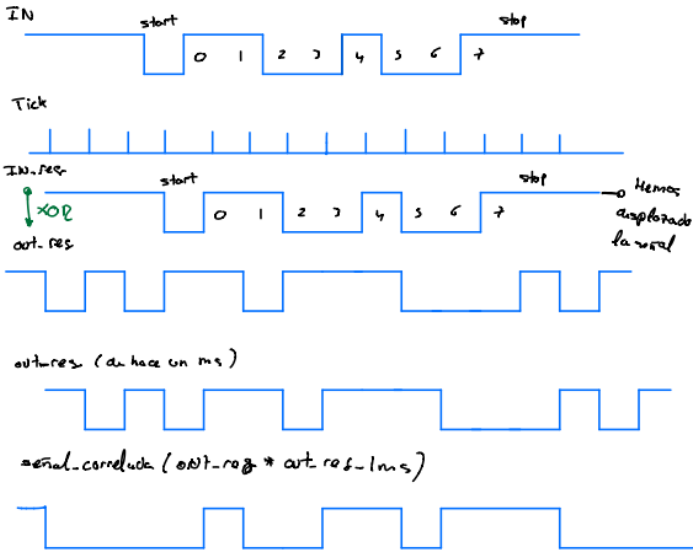


Una vez hemos guardado todos los datos pasamos a un registro para que la señal que mandemos solo cambie una vez.

Coder

Vamos a crear una señal para sincronizar las llegadas de diferentes palabras para que empiecen sincronizadas con el contador de bit y duren lo mismo, porque sino depende de cuando llegue la palabra el primer bit durara menos que el resto.

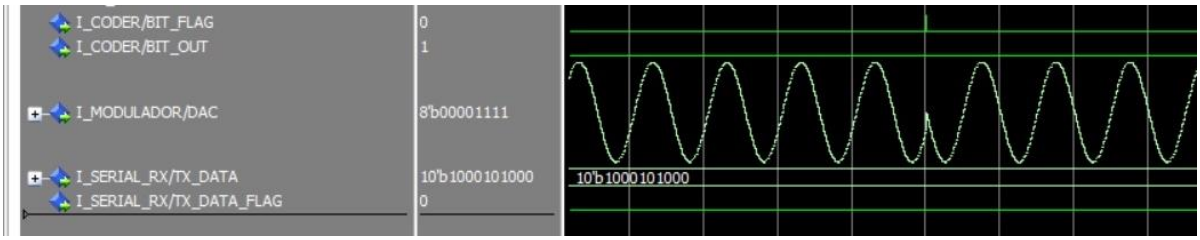




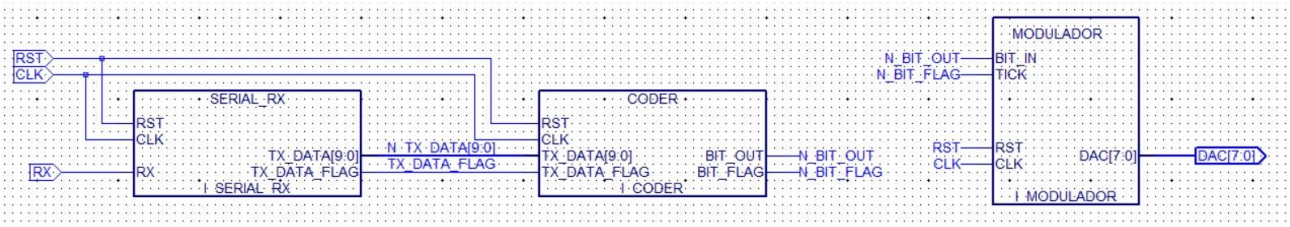
Para comprobar que esta bien lo que hacemos es retrasar la señal out_reg 1 ms y correlarla con la original, será como hacer una modulación, esto no es sintetizable por lo tanto apagaremos la herramienta de síntesis con "*Synopsys Translate off*" .

Modulador

Lo que hacemos es que cunado llegue la señal "*shot*" analicemos si lo que tenemos es un "0" o un "1" y en funcion de eso cambiar o mantener la fase de nuestro seno de 100 KHz, debemos hacerlo cunado el seno pase por 0 para que el cambio de fase se note.

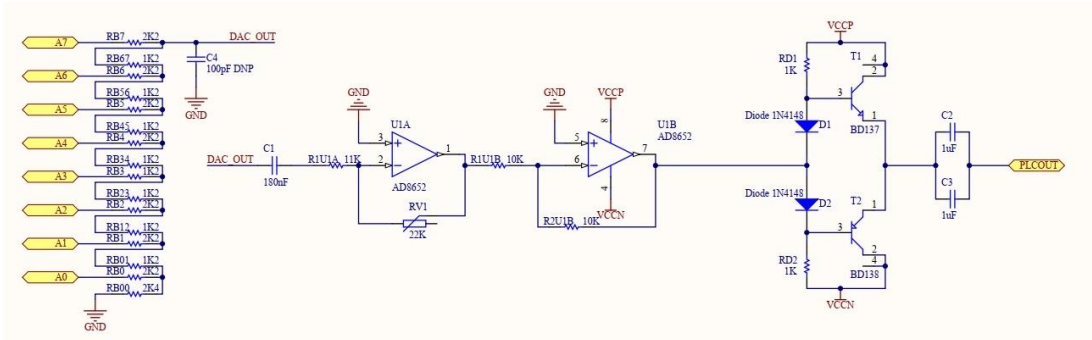


Esquema de conexiones

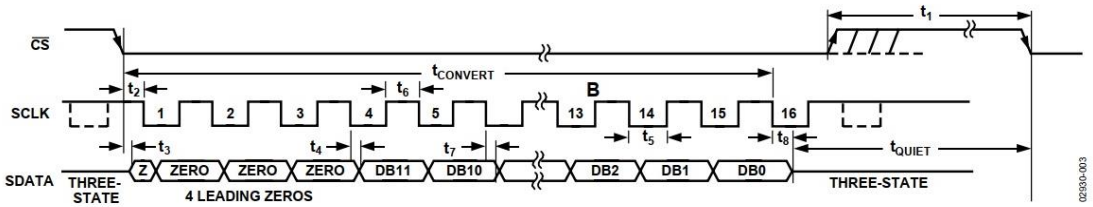


Conversor A/D

Es un converor flash de 8 bits que utiliza divisores de tension, despues de el conversor va una etapa de simetria complentaria para amplificar la señal

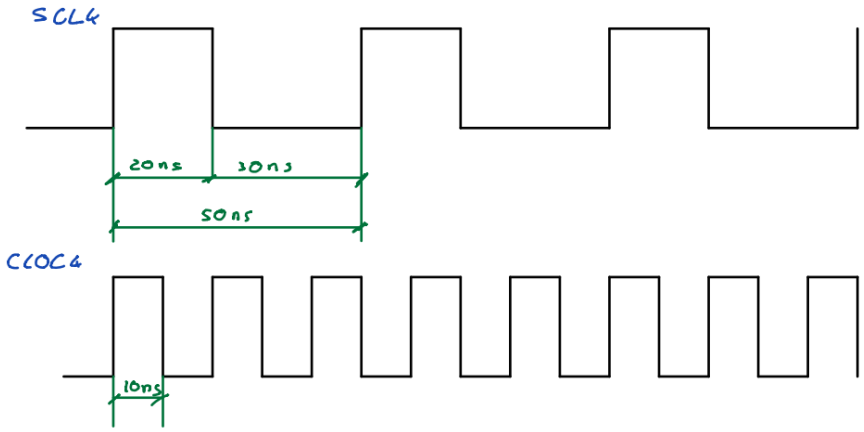


Control ADC



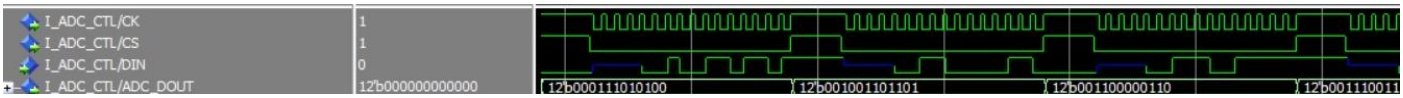
El problema es que SCLK va a 100 MHz y nuestra señal de reloj es de 50 MHz, por lo tanto la tenemos que acelerar.

Vamos a generar la señal SCLK a partir del CLOCK de 10 ns.

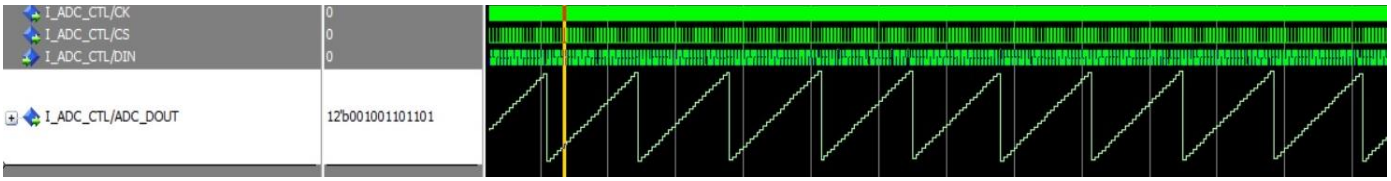


Ahora vamos a generar la señal *"CS"* tiene periodo de 1 us por lo tanto con nuestra frecuencia de reloj lo podemos conseguir con un contador de 4 y otro de 19, que el primero llega hasta 4 el de 19 se incrementa 1.

Simulación



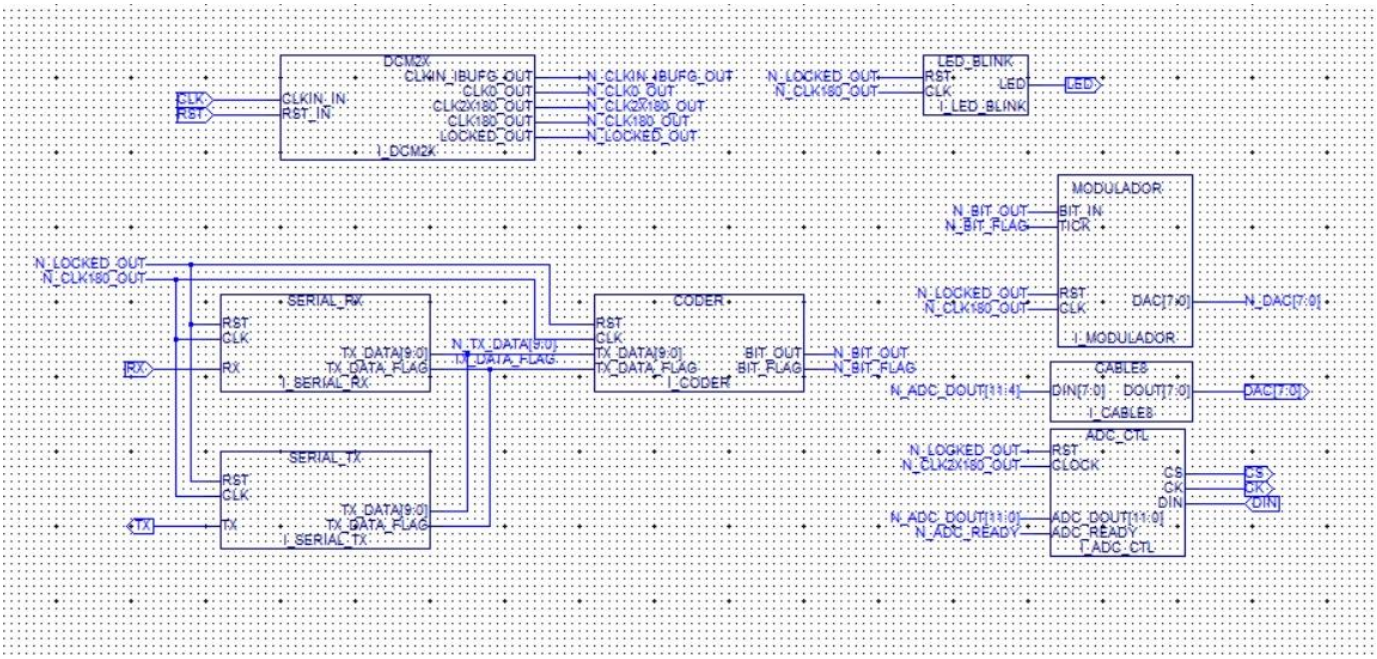
Visualizacion en formato analogico para ver la rampa



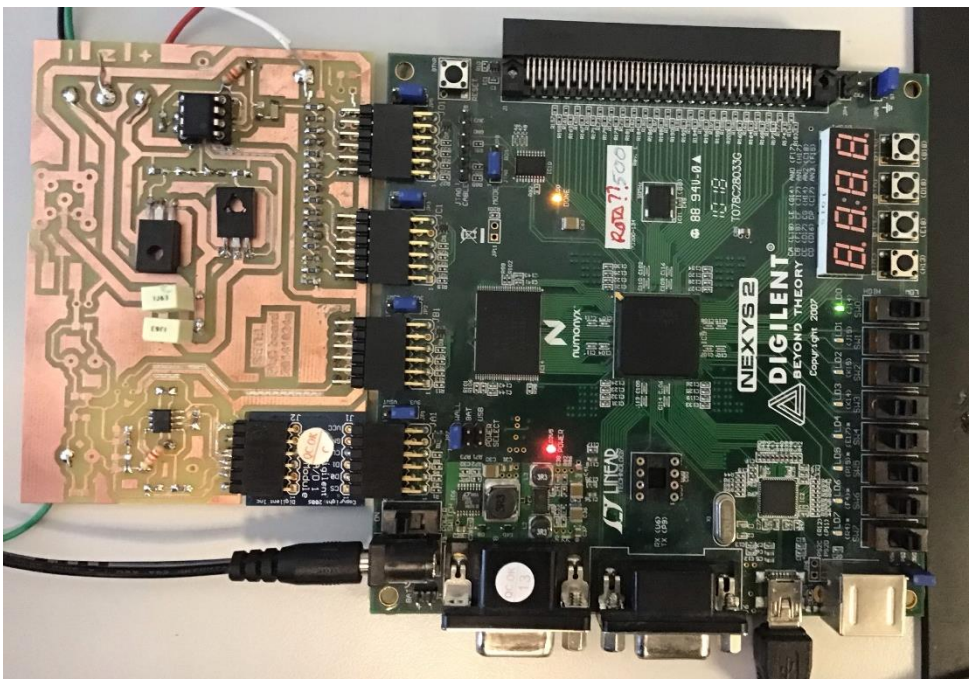
Ahora tenemos que meter lo que hemos hecho de control y el ADC en el esquematico del transmisior. Metemos ADC_Control en la FPGA y AD7476A en PLACA.

El siguiente paso es poner en FPGA el DCM para generar la señal de reloj a 100 MHz. De esta manera le llegaran a todos los bloques la señal de reloj de 50 MHz excepto a ADC_CONTROL que le llegara la de 100 MHz.

Tambien sustituimos el reset por la señal LOCKED_OUT para asegurar que las señales de reloj de DCM_2X estan estables. Hay que tener cuidado ya que esta señal se activa en alto, por lo que tuvimos que cambiar la manera de funcionamineto del reset en nuestro sistema.



Una vez acabado el codigo debemos soldar en la PCB los operaciones y la etapa en simetria complentaria

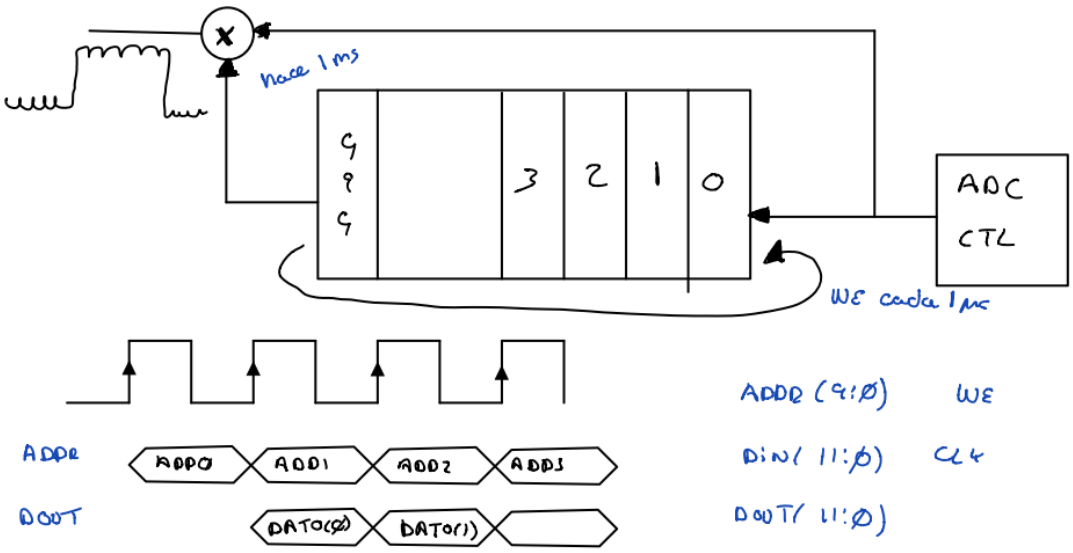


Para comprbar que funciona lo que hacemos es mandar una señal sinusiodal desde el generador de señales, digitalizarla en la FPGA, volver a convertirla a analogica y visualizarla en el osciloscopio para compribar que las dos conversiones funcionan bien.

Capturas del osciloscopio a diferentes frecuencias

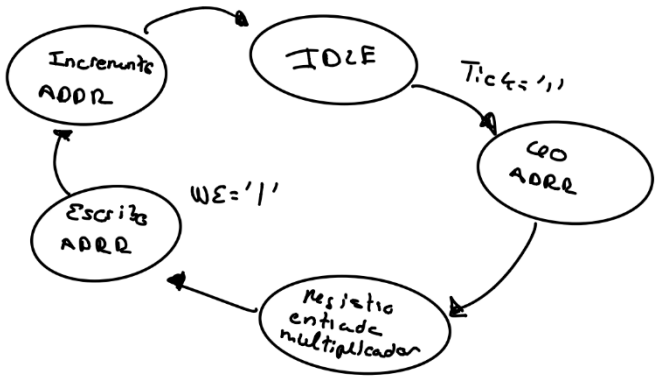
RAM

los datos que llegan del ADC_CTL se van guardando en la RAM que tiene el siguiente esquema:

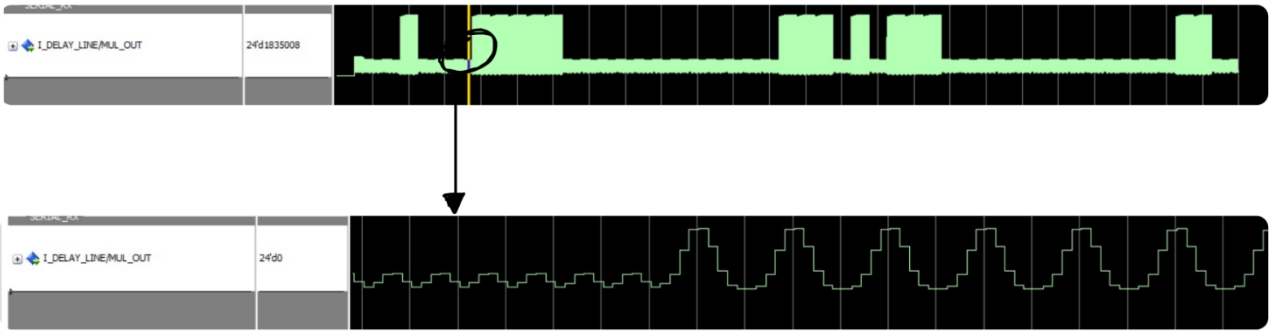


Delay Line

Para controlar la línea de retraso en la salida MUL_OUT tendremos la multiplicación de los datos multiplicados por si mismo pero retrasados 1 ms, para esto usamos la memoria RAM, ya que nos permite tener los datos retrasados a la vez que los actuales. Para hacer esto necesitaremos una sencilla máquina de estados:



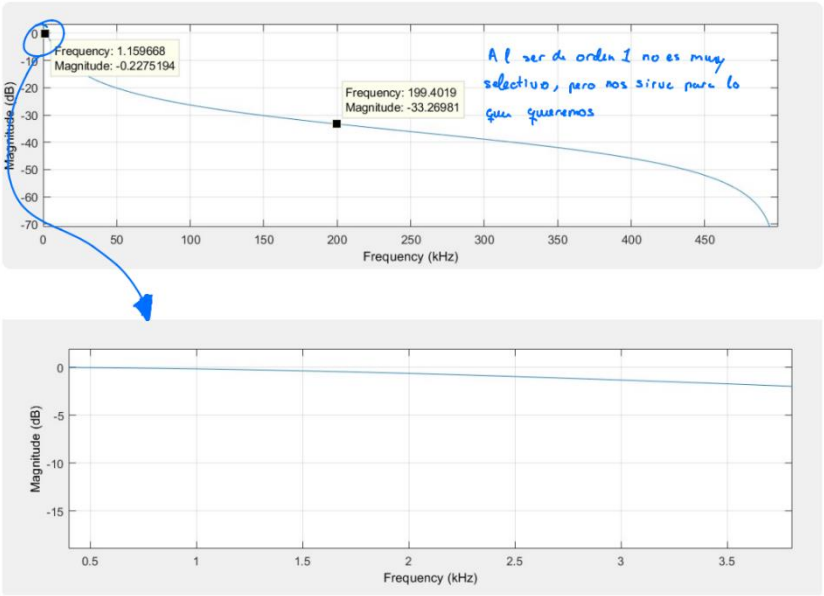
Simulación de la señal MUL_OUT:



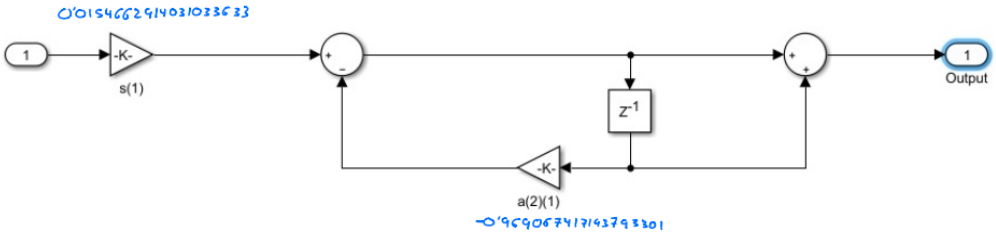
Solo queda aplicarle el filtro, como nuestra señal es de 1 KHz y esta otra es de 200KHz con un filtro paso bajo sencillo de primer orden es suficiente.

Filtro Paso Bajo

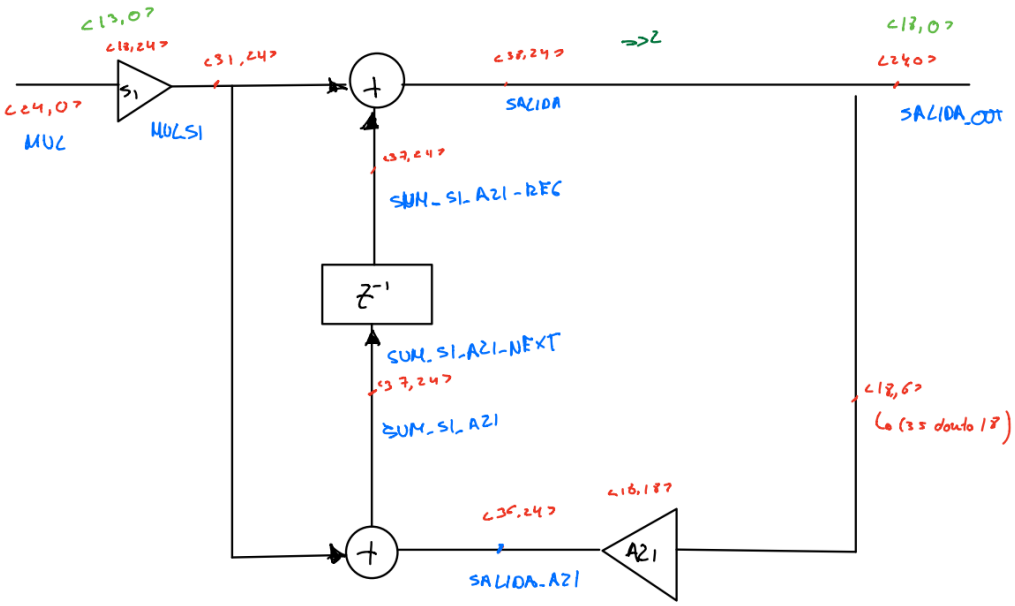
Para diseñar el filtro hemos utilizado una herramienta de diseño de filtro de Matlab.



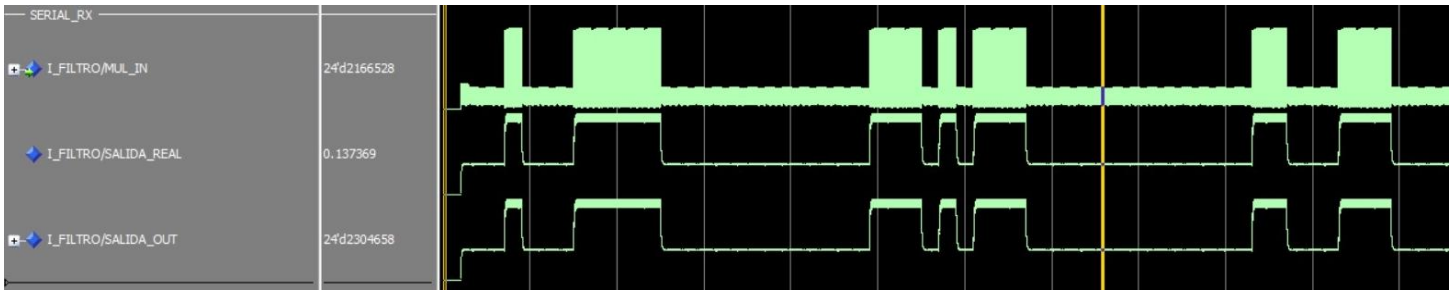
Esquema



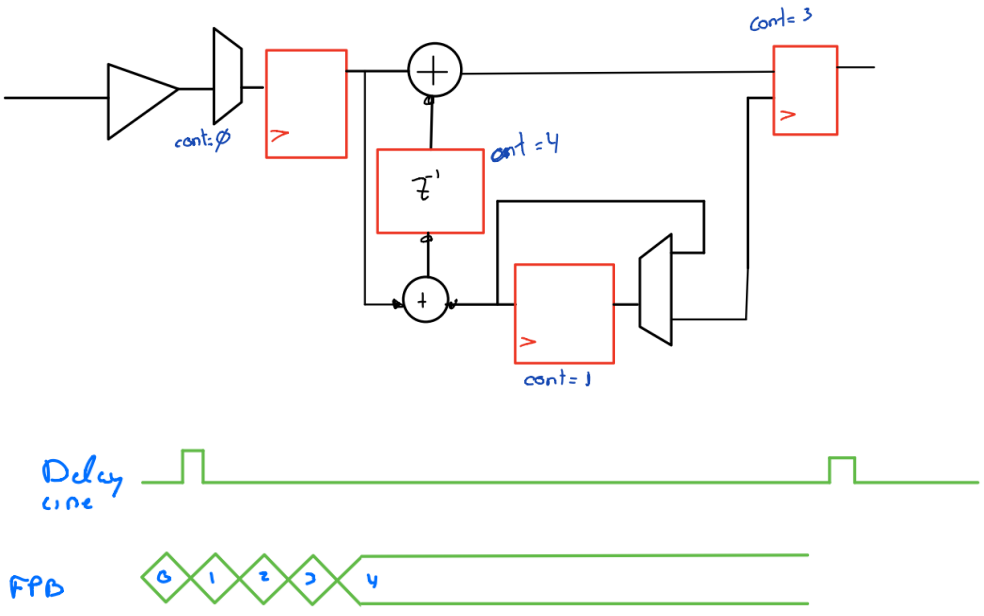
Codificación



Simulacion del filtro con reales y codificado en VHDL



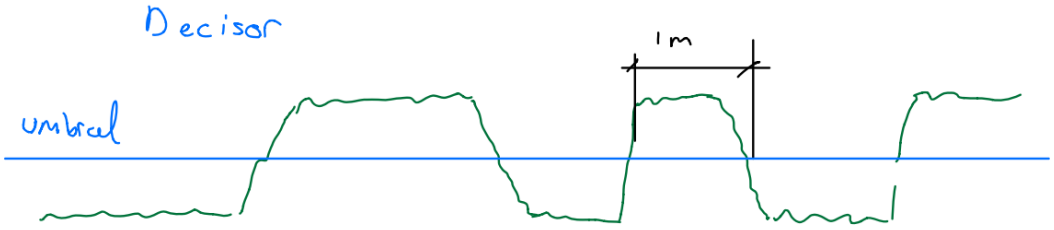
El problema de este filtro que al sisntetizar y hacer place and route no cumplia con los tiempos, para solucionarlo lo que hicimos fue segmentar el filtro utilizando biestables en ciertos lugares:



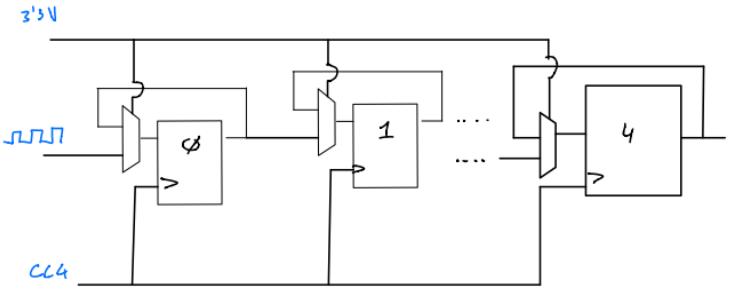
De esta manera hacemos un contador que cuente hasta 4 y cada biestable se activara según esta en el dibujo. De esta manera cumplimos con los tiempos.

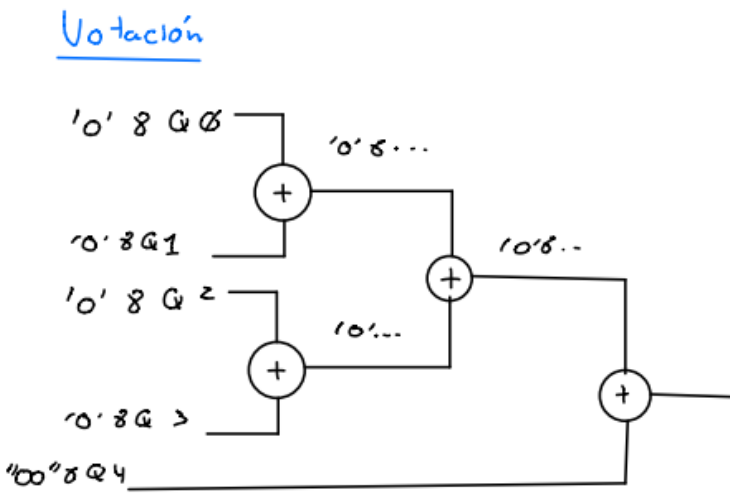
Decisor

Con la salida del filtro paso bajo seleccionamos un umbral y en función de si la señal esta por encima o por debajo decidimos un “0” o un “1”.

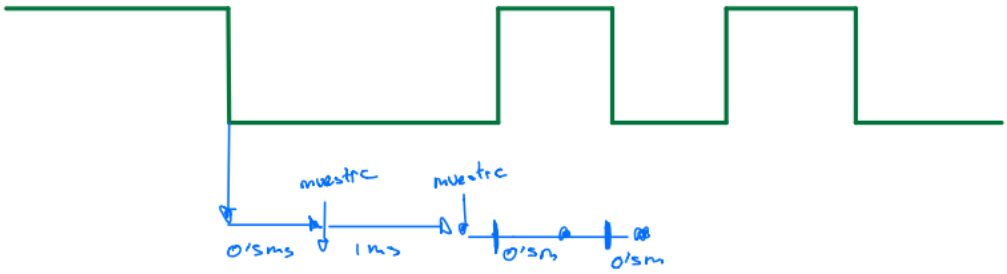


Para evitar que haya errores cogemos la señal decidida y la pasamos por 5 biestables para hacer una votación.





Una vez tengamos la señal votada y sin errores lo que tenemos que hacer es muestrear, muestreamos con un periodo de 1 ms, pero si detectamos un flanco de subida o de bajada volveremos a sincronizar para evitar errores.

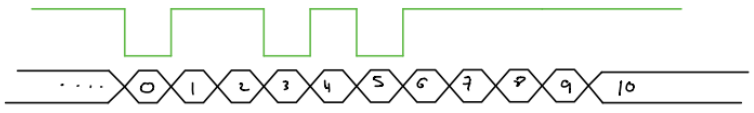


Para hacer el muestreador he hecho que solo se active cunado se están enviando datos, y que cunado este en la línea de reposo no haga nada

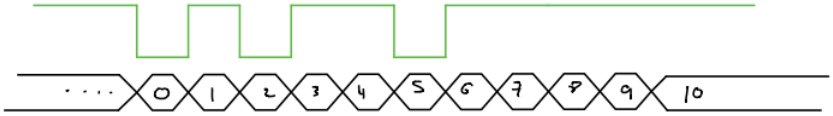
Luego enviamos la salida del filtro al serial_TX que lo que hace es un proceso inverso a Serial_RX y ya tendríamos los datos.

Problemas

Tuvimos un problema que fue que los datos nos salían al revés, y tuvimos que darle la vuelta con el siguiente algoritmo:



Esta es la señal que deberiamos ver los datos son 0110101111, pero en lugar de esta teniamos:



Esto es lo que no llega que es lo mismo pero al revés, no era tan sencillo como darle a vuelta porque se perderían los bits de Start y Stop. Se nos ocurrió el siguiente algoritmo:

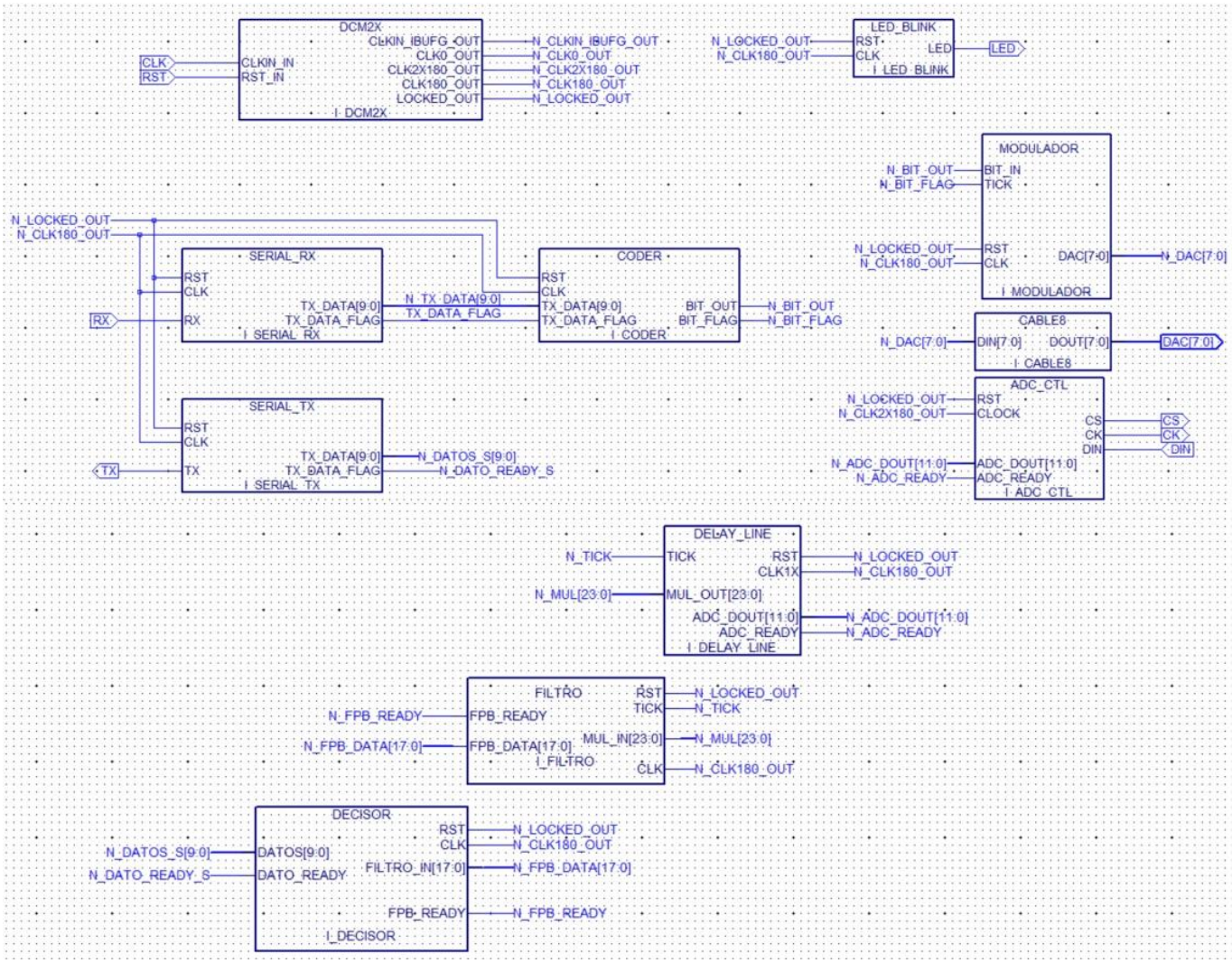
1. Guardamos en un std_logic_vector los datos que nos llegan, en nuestro caso seria : **0101101111**

2. Le damos la vuetla: **1111011010**

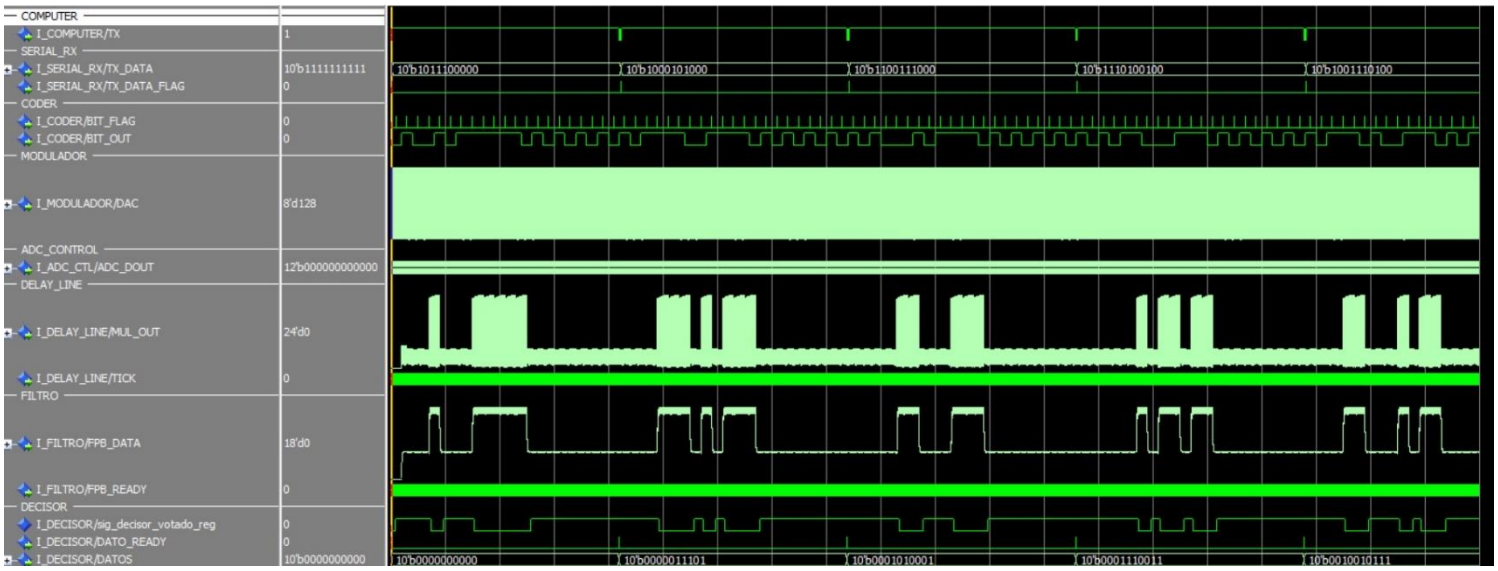
3. No dimos que lo unos que hay al principio son los mismo que debería haber al final en caso de que estuviera bien, por lo tanto lo que hacemos es detectar los "1" que ha delante del bit Start y lo ponemos al final.

Otro problema que tuvimos que al hacer la síntesis no nos dimos cuenta de que utilizamos una señal del bloque Serial_RX en el decisor, que físicamente no están conectados, por lo tanto la simulación la hacia bien pero en la síntesis fallaba, la solución fue simplemente volver a generar esta señal que nos interesaba en el decisor en vez de usar la del Serial_Rx.

Esquema final del trabajo



Simulación final



Reports

```
Design Summary:
Number of errors:      0
Number of warnings:    4
Logic Utilization:
  Number of Slice Flip Flops:          406 out of   9,312    4%
  Number of 4 input LUTs:              502 out of   9,312    5%
Logic Distribution:
  Number of occupied Slices:           400 out of   4,656    8%
    Number of Slices containing only related logic:     400 out of     400 100%
    Number of Slices containing unrelated logic:          0 out of     400   0%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:        637 out of   9,312    6%
    Number used as logic:              502
    Number used as a route-thru:       135

  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

  Number of bonded IOBs:                16 out of     232    6%
  Number of RAMB16s:                     1 out of      20    5%
  Number of BUFGMUXs:                    3 out of      24   12%
  Number of DCMs:                        1 out of       4   25%
  Number of MULT18X18SIOs:               4 out of      20   20%
```

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
Autotimespec constraint for clock net N_CLK180_OUT	SETUP	N/A	17.872ns	N/A	0
	HOLD	0.921ns		0	0
Autotimespec constraint for clock net N_CLK2X180_OUT	SETUP	N/A	5.054ns	N/A	0
	HOLD	0.967ns		0	0