1 Conversor SAR

1.1 Introducción teórica

Un conversor SAR o conversor por aproximaciones sucesivas es un tipo de implementación de un conversor analógico digital.

El conversor utiliza un algoritmo de búsqueda binaria para lograr establecer el valor digital cuya conversión analógica es la más cercana al valor de la entrada que se está intentando convertir.

1.1.1 Algoritmo de búsqueda binaria

El algoritmo de búsqueda binaria es un algoritmo que permite encontrar un valor buscado V_b dentro de un conjunto discreto y ordenado de valores V_{set}. Este algoritmo resulta eficiente por su manera de buscar, que va descartando mitad de los elementos de V_{set} por cada comparación realizada.

El algortimo compara el valor objetivo al elemento medio de V_{set} según el ordenamiento utlizado. En caso de no resultar iguales,, el algoritmo elimina la mitad derecha V_{subDer} o la mitad izquierda V_{subIzq} de los elementos de V_{set}c omo posibles valores V_b según el resultado anterior de la comparación con el elemento medio y continúa realizando la comparación con el elemento de la mitad del subconjunto restante hasta que encuentre el elemento V_b o el nuevo subconjunto $V_{sub} \subseteq V_{set}$ tiene un único elemento que no es V_b.

Luego de ejecutado el algoritmo, se obtendrán dos posibles resultados:

1. Se obtiene el valor V_b buscado, como se mencionó anteriormente.
2. Se obtiene un valor V_{ob} no igual a V_b tal que $V_{ob} \epsilon V_{sub}$ que resulta ser el elemento perteneciente a V_{set} que más se acerque al valor buscado, ya sea por izquierda o por derecha.

1.2 Modelo y funcionalidad general del conversor

El esquema general de un conversor SAR de 8 bits es el siguiente:

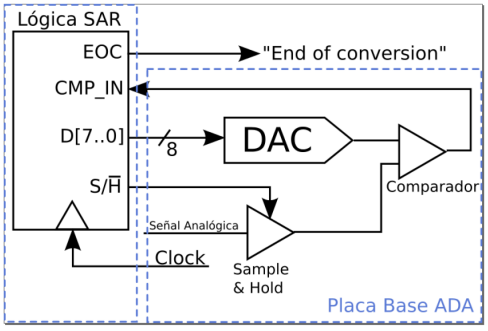

Figure 1: Conversor SAR

Se procede a explicar la implementación del algoritmo de búsqueda binaria para lograr convertir la tensión de entrada analógica "Señal Analógica", ahora en más denominada "V_a" a un valor digital apropiado "V_d".

1. La señal analógica V_a pasará por un sample and hold que logrará retener un cierto valor de la señal cada un intervalo de tiempo bien definido τ, como se explica en el Trabajo Práctico 1. Denominaremos al valor muestreado en el tiempo t_o como $V_a(t_o)$, el cual será mantenido por un intervalo temporal τ.

2. El módulo "Lógica SAR" es el encargado de implementar la lógica del algoritmo de búsqueda binaria: El mismo decidirá con qué valor inicial comparar la señal analógica (el "elemento medio", como se mencionó en la explicación del algoritmo) para finalmente obtener V_d. Comenzará con un valor digital tentativo V_d'.

3. V_d' es convertido a valor analógico V_a' mediante el uso de un DAC.

4. El comparador arrojará un valor lógico que indica si V_a' es menor, mayor o igual a $V_a(t_o)$. Este valor lógico retornará al módulo lógico para que el mismo obtenga el nuevo valor tentativo V_d' y se continúe comparando hasta que se llegue al final de la conversión/búsqueda (EOC), en cuyo caso se V_a' resulta ser el valor igual o más cercano a $V_a(t_o)$ y por ende se deduce que el V_d' asociado resulta ser la representación digital de la entrada, lográndose así convertir la señal analógica a la digital para toda muestra.

1.3 Implementación de la lógica digital para la búsqueda binaria

Únicamente para los efectos de este conversor, se tomará la convención de que un bit "apagado" será equivalente a un 0 lógico mientras que un bit "prendido" será equivalente a un 1 lógico.

Por cuestiones notacionales, se numerará a los bits en forma creciente, siendo el bit más significativo el bit b_0 y el menos significativo el bit b_{N-1}, siguiendo la siguiente figura:

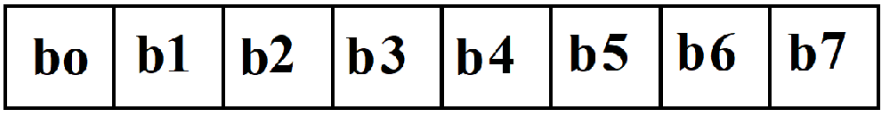

Figure 2: Notación para los bits

Se elige que la representación binaria/digital de la tensión de entrada que se almacene en el módulo lógico luego de finalizada la conversión asocie su valor mínimo a todos sus bits apagados y su valor máximo a todos sus bits prendidos, lo que se correspondería con una representación positiva (no admitiría números negativos) de los números. Esto se debe a que muchos DACs interpretan su input digital de la misma forma. Además, la lógica utilizada para implementar el algoritmo resulta ser más fácil de escribir de esta manera.

Se considera ahora al arreglo de N bits que es la salida del módulo lógico (en nuestro caso, N=8) como el arreglo en el cual se realizará la búsqueda del valor V_d, es decir, los todos los números que son representables con N bits formarán el conjunto V_{set}. El valor medio inicial de este conjunto estará dado por el bit más significativo prendido y el resto apagado.

El algoritmo procederá haciendo un análisis "bit a bit" y es por eso que denominamos b_a al bit que está siendo analizado, es decir, al bit para el cual se está decidiendo si deberá prenderse o apagarse para lograr el valor V_d' final correcto. De lo anterior, inicialmente $b_a = b_0$, por lo que $a = 0$.

(1) Al convertir el valor tentativo digital dado por el estado actual de los N bits a analógico y luego comparar con la tensión de entrada retenida (el resultado de la comparación será recibido por el pin CMP_IN de la figura) como se explicó en la subsección anterior, se seguirá la siguiente regla de decisión:

- Si el valor tentativo resulta menor al de entrada V_a, el bit b_a deberá prenderse.
- Si el valor tentativo resulta mayor al de entrada V_a, el bit b_a deberá apagarse.

Luego, si $a \neq N - 1$, se elige incremente el índice a y se vuelve a realizar la comparación, volviendo al punto (1).

Al terminar el ciclo iterativo, la configuración en la que se encuentran los N bits será el valor V_d', que convertido a analógico estará más cercano al valor de la entrada V_a.

Cabe aclarar que cada iteración será realizada en un ciclo de clock interno del módulo lógico, que será menor en duración al intervalo temporal τ durante el cual se retiene la muestra tomada en el instante t_o, ya denominada $V_a(t_o)$. Sin embargo, estos dos tiempos estarán íntimamente relacionados y es por eso que los tiempos de muestreo y retención del sample and hold estarán también manejados por el módulo lógico.

Esta afirmación y la relación entre tiempos serán explicadas con detalle en la siguiente subsección.

1.4 Interacción temporal entre los distintos componentes

Por cuestiones notacionales definimos a t_c como el tiempo que dura un ciclo de clock, es decir, el tiempo durante el cual la señal de clock se mantiene en un estado lógico constante 0 ó 1.

El módulo lógico no resulta independiente del funcionamiento del resto de los componentes que forman parte del conversor en cuanto a cuestiones temporales. Se irá caso por caso, llamando la atención sobre los factores y tiempos a tener en cuenta.

1.4.1 Interacción entre módulo lógico y Sample and Hold

La dependencia entre tiempos de componentes se debe principalmente a que la implementación del algoritmo de búsqueda mencionada anteriormente depende de que $V_a(t_o)$ se mantenga constante durante las N iteraciones que tiene que realizar el módulo lógico para poder convertir el valor $V_a(t_o)$ a digital. Para que esto último suceda, el tiempo por el cual se retiene una muestra tiene que ser, como condición necesaria pero no suficiente, mayor al tiempo en que tarda el módulo lógico en realizar N iteraciones de comparación ($N \cdot t_c < \tau$). Además, las N iteraciones tienen que realizarse sobre la misma muestra, por lo que la señal de clock y el tiempo en el que se comienza a iterar tienen que estar sincronizados con la señal de Sample y de Hold de manera tal que sólo se comience a iterar cuando se comenzó el tiempo de hold de una muestra.

Sin embargo, en la práctica es necesario pedir una condición extra cuando se habla de la interacción específica entre S&H y módulo lógico, y es que hay que tener en cuenta los tiempos de establecimiento de la muestra cuando es retenida por el S&H. Es por esto que se podrá comenzar a iterar sobre la muestra únicamente cuando la muestra retenida ya se ha establecido, por lo que si llamamos a este tiempo de establecimiento t_e resulta ser que $N \cdot t_c + t_e < \tau$.

1.4.2 Interacción con el DAC

Por cada iteración, el DAC tendrá que convertir el valor digital tentativo a un valor analógico para luego poder comparar ese valor con la entrada. La respuesta del DAC en tiempo no es ideal (nula), sino que tiene un valor definido por el intergrado o circuito a utilizar. Cada iteración tendrá que esperar un tiempo t_{dac} de respuesta del conversor DAC para poder recibir la respuesta de comparación y así poder decidir el valor del bit actualmente analizado. Es de aquí que se agrega esta condición para las N iteraciones que se realizan dentro de un tiempo de hold.

De aquí, $N \cdot t_c + t_e + N \cdot t_{dac} < \tau$

1.4.3 Interacción con el comparador

El tiempo de respuesta del comparador ha de ser tenido en cuenta. Dado que la salida del comparador sólo será observada durante el tiempo en que el S&H está en estado Hold y luego de pasado el tiempo de establecimiento del mismo, entonces se deberá sumar el tiempo de respuesta del comparador a la condición para el tiempo de clock. De esta manera, llamando t_{comp} al tiempo de respuesta del comparador, obtenemos $N \cdot t_c + t_e + N \cdot (t_{dac} + t_{comp}) < \tau$.

1.4.4 Condición de Clock resultante

Luego de establecidas las condiciones a tener en cuenta y habiendo llegado a la expresión:

$$N \cdot t_c + t_e + N \cdot (t_{dac} + t_{comp}) < \tau$$

Podemos despejar la condición para el tiempo máximo en el que el clock realiza un ciclo como:

$$t_c < \frac{\tau - t_e}{N} - t_{dac} - t_{comp}$$

Este tiempo máximo tendrá asociada un frecuencia mínima de clock dada por:

Nos preguntamos luego cómo definir el tiempo de Hold para que el sistema funcione correctamente. El tiempo que impondrá las condiciones para el tiempo de clock será el tiempo de sample en este caso, ya que el integrado de sample and hold tendrá un tiempo de acquisición mínimo que deberá cumplirse y es más restrictivo que el tiempo de establecimiento del hold.

Dada la máxima frecuencia de muestreo del sistema, f_m de $44.1kHz$, la cual para dar márgenes de error aproximamos a f_m=$44.5kHz$, podemos obtener el máximo período de sampleo $T_m = 22.45\mu s$.

Llamando a su vez $t_a = t_e + N \cdot (t_{dac} + t_{comp})$, podemos expresar a un periódo de muestreo a partir del siguiente gráfico, donde t_{sar} es el tiempo en el que se realiza las iteraciones:

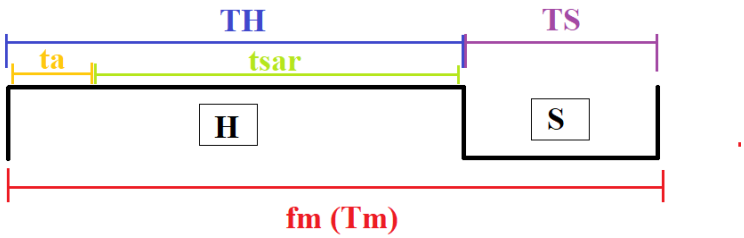

Figure 3: Tiempo de muestreo, sample, hold y retardos adicionales

Requerimos utilizar el menor tiempo de sample posible, por lo que utilizando el valor de frecuencia f_m mencionado anteriormente y teniendo en cuenta un tiempo de acquisición de $10\mu s$ (el sample and hold a utilizar será el LF398), observamos que si se elige $T_s = T_m \cdot 0.45 = \frac{0.45}{f_m} \approx 10.1\mu s$, se cumple la condición. De aquí se deduce que $\tau = T_H = 0.55 \cdot T_m \approx 12.5\mu s$.

Restando t_a obtenemos luego el tiempo en el cual se realizarán cada una de las N=8 iteraciones, lo cual arroja un $t_c = t_{sar} = \frac{\tau - t_a}{N} =$. Este tiempo de clock arroja un valor mínimo de frecuencia de clock de $f_{clk} = \frac{1}{t_c} =$.

1.5 Implementación física del módulo lógico

Se decidió implementar al módulo lógico mediante el uso de una FPGA, programándola con el uso de Quartus II.