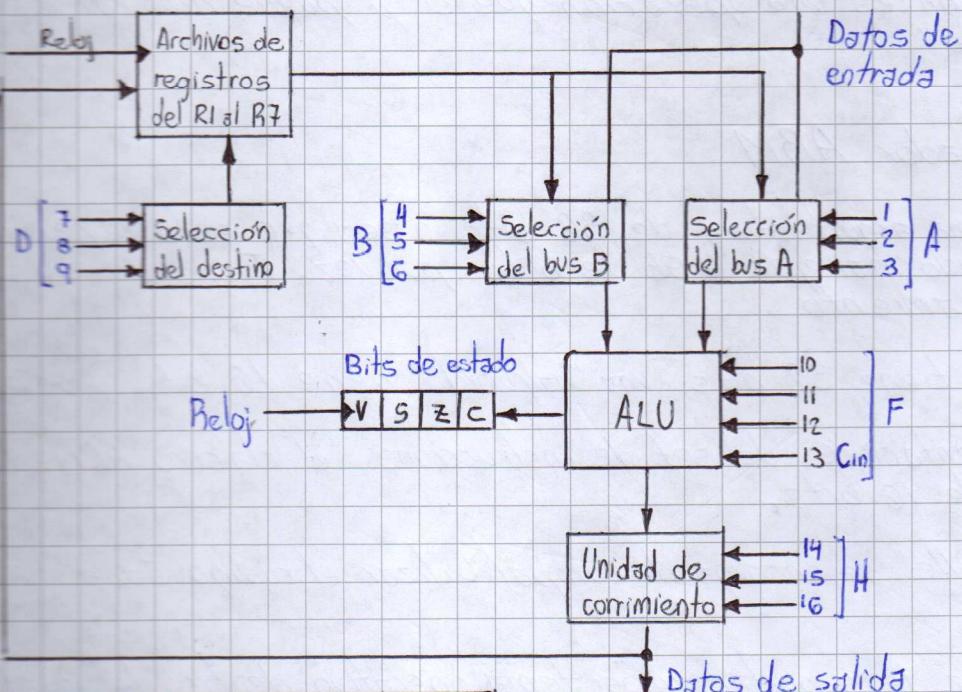


Codificación de la palabra de control de la unidad procesadora

Función de campos de selección						
Código Binario	A	B	D	F con $Cin = 0$	F con $Cin = 1$	H
000	Entrada	Entrada	Ninguno	$F = A$	$F = A + 1$	No hay corrimiento
001	R_1	R_1	R_1	$F = A + B$	$F = A + B + 1$	SHL
010	R_2	R_2	R_2	$F = A + \bar{B}$	$F = A - B$	SHR
011	R_3	R_3	R_3	$F = A - 1$	$F = A$	$Bus = 0$
100	R_4	R_4	R_4	$F = A \wedge B$	-	-
101	R_5	R_5	R_5	$F = A \vee B$	-	ROL
110	R_6	R_6	R_6	$F = A \oplus B$	-	ROR
111	R_7	R_7	R_7	$F = \bar{A}$	-	-



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16			
A	B	D	F				H											

(Palabra de control)

→ La palabra de control tiene 16 bits.

Microoperación	Designación Simbólica					Palabra de Control				
	A	B	D	F	H	A	B	D	F	H
$R_1 \leftarrow A_2 - A_3$	A_2	A_3	R_1	$F = A_2 - A_3$	No hay comm.	0	1	0	0	1
$R_4 \leftarrow shr(R_5 + R_6)$	R_5	R_6	R_4	$F = A + B$	SHR	1	0	1	1	0
$R_2 \leftarrow R_2 + 1$	R_2	-	R_2	$F = A + 1$	No hay comm.	1	1	0	0	1
$R_1 \leftarrow R_2$	R_2	-	R_1	$F = A$	No hay comm.	0	1	0	0	0
$Salida \leftarrow R_3$	R_3	-	Ninguna	$F = A$	No hay comm.	0	1	0	0	0
$R_4 \leftarrow ROL/R_4$	R_4	-	R_4	$F = A$	ROL	1	0	0	0	0
$R_5 \leftarrow 0$	-	-	R_5	-	$Bus = 0$	0	0	0	0	0

11.05.17

- **SUPERVISOR**: entra en acción cuando estoy en el reset y cuando se produce una interrupción por software (en la analogía, el intercomunicador era la interrupción por software; es la única sincronizada, porque las de hardware son asincrónicas). El sincronismo es debido al clock. Que se produzca en el reset quiere decir que se produce cuando arranca.
- **ABORT**: usado para manejar violaciones de acceso a la memoria. Son una especie de **AUTODIVERRUPCIONES**, si estás trabajando normalmente y detectas que la memoria ha sido "violada", automáticamente sale del modo usuario y entra en este modo.
- **UNDEF**: usado para manejar instrucciones indefinidas. En un µ de 32 bits, hay capacidad p/ 46 de instrucciones, pero yo manejo 50 como mucho, y el resto son muchas instrucciones indefinidas. Cuando detecta una de esas, entra en este modo.
- **SYSTEM**: modo privilegiado.

→ Set de registros ARM

→ Los primeros 12 registros no tienen nombre, pero los últimos 3 pueden realizar distintas tareas

- **r15 (pc)** → contador de programa → indica la instrucción que se está ejecutando o la que se tiene que ejecutar. Una vez que realice una tarea, el "pc" ya debe tener la dirección de la próxima instrucción.
- **r14 (lr)** → enlace de registros → cuando estoy ejecutando una tarea y necesito hacer una subrutina, necesito volver a la rutina que estaba ejecutando (enlace rutinas).
- **r13 (sp)** → puntero a la pila.
- **CPSR** → current program status register → registro de estado de programa → Nosotros antes lo habíamos visto como las banderas de estado, actualizan el mismo significado (Carrier, Overflow, Zero y Signo)

→ Cuando el micro pasa de un modo al otro tengo que "salvar o guardar" el valor de todos los registros para no perder lo anterior. Lo que se hace es salvar estos registros mencionados anteriormente, y solo modificar o borrar los R0 - R12, salvo en el modo FIQ que predio salvar desde R8-R14.

→ Las interrupciones son provenientes del exterior, y las excepciones, como los cambios de modo que vimos recién, son una especie de AUTODIVERRUPCIONES.

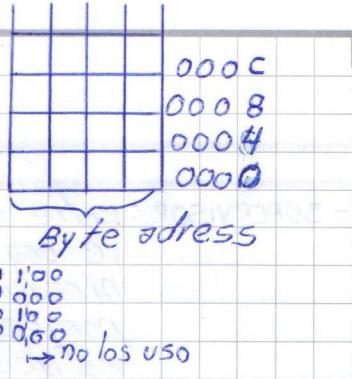
- Cuando ocurre una excepción, el ARM:
- Copia el CPSR en SPSR

→ Registro de estado de programa

- Podemos hacer incaps solo en el primer bloque, son las banderas que ya conocemos (N, Z, C, V).
- Cuando I = 1 → habilita IRQ
- " " F = 1 → " FIQ
- " " T = 0 → !! procesador ARM ; T = 1 → THUMB state

→ Program counter (r15)

- Las instrucciones tienen 32 bits
- " " que estar alineadas a word → a las instrucciones las coloco en la 0, 4, 8, ...
- El valor del program counter es almacenado en los bits 2 al 31, y con los bits 0 y 1 no están definidos (no se usan) → el pc me indica la tarea que estoy ejecutando o pur ejecutar



→ Ejecuciones condicionales y banderas

→ Las instrucciones de Arm pueden ejecutarse condicionalmente

CMP r3, #0
BEQ SKIP
ADD r0, r1, r2
SKIP

BRANCH EQUAL

CMP r3, #0
ADDNE r0, r1, r2 → NOT EQUAL (r3 ≠ 0)

↓ menor cantidad de líneas → MEJOR

→ Por defecto las instrucciones de procesamiento de datos NO afectan las banderas del código de condiciones, pero las banderas pueden ser opcionalmente establecidas usando "S". El COMP no necesita "S".

SUBS r1, r1, #1 → decrementa r1 y setea la bandera
BNE loop

↓ si no seteo la bandera, no se aplica la instrucción que hice.

→ El compare restó 0 a R3, la bandera "Z" no se activó porque R3 ≠ 0 (en este caso)

→ Código de condiciones

Sufijo	Descripción	Bandera
EQ	Igual	Z = 1
NE	No igual	Z = 0
CS/HS	(Sin signo) igual o mayor	C = 1
CC/LO	(Sin signo) menor	C = 0
MI	Menos	N = 1
PL	Positivo o Cero	N = 0
VS	Overflow	V = 1
VC	No Overflow	V = 0
HI	(Sin Signo) mayor	C = 1 & Z = 0
LS	(Sin Signo) menor o igual	C = 0 or Z = 1
GE	Mayor o igual	N = V
LT	Menor que	N! = V
GT	Mayor que	Z = 0 & N = V
LE	Menor o igual que	Z = 1 or N = !V
AL	Siempre	

Single Register data transfer

- Una cadena de caracteres es un arreglo cuyos elementos son elementos del código ASCII
- El código ASCII es un código de 7 bits, que contiene 128 caracteres
- Entonces para trabajar con caracteres, voy a trabajar con bytes, no con palabras.
- O sea que si me piden almacenar caracteres en la memoria, voy a tener que trabajar con 8 bits

LDR STR - Word - 32 bits
 LDRB STRB - Bytes - 8 bits
 LDRH STRH - Halfword - 16 bits

- Cuando quiero copiar un byte sin signo lo almaceno en los 8 bits menos significativos y a los 24 restantes los completo con "0"
- Cuando el byte tiene signo, completo los 24 con el bit de signo

LDRSB 101011 → nº de 6 bits con signo. P/saber el valor absoluto → Complemento a 2.
 LDRSH 010100

 ...000000000000010101
 ...1111111111101011

↓
 hasta el primer "1" ←
 inclusive → luego invierto los bits

Número extendido a 32 bits

- Sintaxis: ↗ si es byte, halfword, word
 ↗ Adónde va a parar el dato desde la memoria

LDR {cond} {<size>} Rd, <address> } → dirección de la memoria *

STR {cond} {<size>} Rd, <address> } → dirección de la memoria *

↓
 condicional o no condicional; mayor, igual, menor, etc...

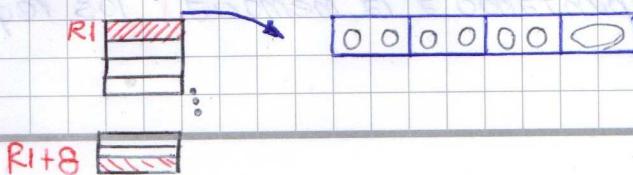
* El puntero en "C", es una variable cuyo contenido es una dirección de memoria.

{ LDRB R1, [R2] → Cuando está entre corchetes, el dato está en la memoria
 ↓
 ↗ Especificación de la dirección de la memoria donde tengo que buscar el dato

Voy a cargar en R1 el contenido de la memoria que está apuntado por R2. R2 representa la dirección de memoria donde está el dato que me interesa.

→ La dirección es accedida por las instrucciones LDR/STR especificada por un registro base y un desplazamiento.

LDR R0, [R1, #8] →



byte menos significativo → dirección menos significativa

29.06.17

Load/Store Exercise

array[10] = array[5] + y

$$\cdot r_3 = \text{array}[5]$$

LDR r3, [r2, #20] → r2 es la dirección base, tiene un desplazamiento de 20 (decimal) → ahí apunta al arreglo de 5, y luego lo llevo a r3
 $*20 = 4 \times 5$

ADD r3, r3, r1; r3 = array[5] + y → "y" estaba en r1, entonces sumo r3 + r1, y el resultado lo guardo en r3

Ahora lo tengo que llevar a arreglo de 10.

$$* = 40 = 4 \times 10$$

STR r3, [r2, #40]; array[5] + y = array[10] → Llevo r3 a la memoria, donde me indique r2 + un desplazamiento de 40.
* cada elemento de un arreglo, ocupa 4 bytes porque es tipo "word"

Halfword

LDRH r11, [r0]; Carga una media palabra dentro de r11 en la memoria (en lo apuntado por r0)

E	E	r0
F	F	
9	0	
:	:	

r11 antes carga: 1 2 3 4 5 6 7 8

r11 desp. carga: 0 0 0 0 F F E E

Mapa de memoria con direccionamiento de 3 bytes

Halfword → 2 bytes

→ toma media palabra, la cual apunta r0, luego va a r11 y los dos primeros bytes (halfword) van a la parte baja o menos significativa de r11 → los dos bytes más significativos, los relleno, con "0" porque la instrucción no es "LDRHS", o sea no es con signo (si fuera con signo, relleno con el signo)

Multiple

Ahora vamos a ver un proceso similar, a diferencia de que ahora utilizamos más de un registro

opcional

Syntax: <LDM|STM> {<cond>} <addressing-mode> Rb {!} <register list>

STMXX r10, {r0, r1, r4}

!" implica actualización del registro "Rb"

Es el registro base que está apuntando a la memoria

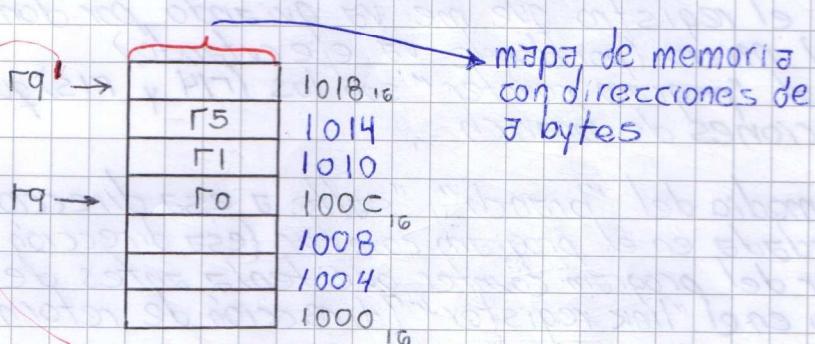
Dirección donde voy a "salvar" los registros r0, r1, r4

DA

$Rb \rightarrow Rlo \rightarrow$

r_4
r_1
r_0

Rlo apunta a r_4 y, una vez que guardo esa posición, incrementa (o decremente) } Rlo siempre apunta a la misma dirección base, el contenido se desplaza luego



STMIA $r9!$, { r_0, r_1, r_5 }

Multiply and Divide

→ hay multiplicaciones que producen resultados en 32 bits y otras en 64 bits → siempre la multiplicación duplica la cantidad de bits del operando

↓
si los resultados son en 32 y 64, la operación que acepta es de media palabra por media palabra (16 y 32 bits).

↓
si tengo que multiplicar un R de 32 × otro R de 32, el resultado va a estar en 64 → el resultado va a estar en 2 registros.

MUL $r_0, r_1, r_2 ; r_0 = r_1 * r_2 \rightarrow$ Toma la mitad de r_1 , la mitad de r_2 y lo guarda en r_0 (antes los multiplica)

[U/S] MULL $r_4, r_5, r_2, r_3 ; r_5:r_4 = r_2 * r_3 \rightarrow$ Toma los 32 bits de cl/ y los guarda en el par $r_5:r_4$ y el -significativo en r_4 .

MLAL $r_4, r_5, r_2, r_3 ; r_5:r_4 = (r_2 * r_3) + r_5:r_4 \rightarrow r_1 * r_2 + r_3$ y guarda en r_0

Branch instructions (Instrucciones de salto)

→ No hay ninguna situación en programación que no pueda ser resuelta por:
- Secuencia:
- Selección: if - ifelse - switch
- Repetición: for - while

→ En "C" el "GO TO" está prohibido porque rompe secuencias.

→ En el micro, podemos utilizar la REPETICIÓN para llegar a una BIFURCACIÓN

→ En "c" cuando llamamos a una función también rompo estructuras

Branch with link: BL {<cond>} subroutine - label → En algún registro queda guardada la dirección de retorno

→ El compilador genera el código de máquina de un lenguaje de alto nivel

→ El ensamblador traduce la sentencia en assembler el código de ARM

→ El "program counter" es el registro que me va guiando por donde me voy moviendo con el micro (por donde va ejecutando).
El "link register" junto al "program counter" son los (R14 y R15) que intervienen en las instrucciones de branch.

→ Cuando le decimos por medio del "branch", "salte a esa dirección", esa dirección queda guardada en el program counter (esa dirección contiene la subrutina); luego el valor del program counter que tenía antes de esta operación queda salvado en el "link register" (dirección de retorno).

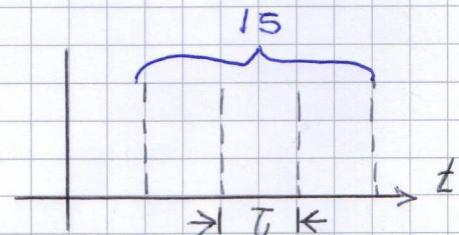
Register Usage

RS-232

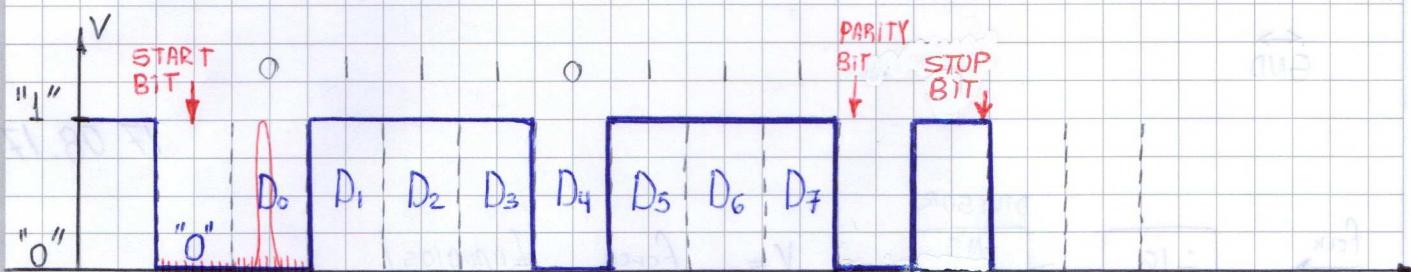


$\left\{ \begin{array}{l} \text{DATA} \\ \text{TERMINAL} \\ \text{EQUIPMENT} \end{array} \right.$ $\left\{ \begin{array}{l} \text{DATA} \\ \text{COMMUNICATION} \\ \text{EQUIPMENT} \end{array} \right.$

$$T = \text{tiempo de duración de un bit} \Rightarrow V = \frac{1}{T} \left[\frac{\text{bits}}{\text{s}} \right]$$



- bits que no aportan inf \rightarrow bimots
- \rightarrow los baudios siempre son menores que los bps.



Dato a transmitir: "EE" = "11101110"

$$\frac{1}{9600} = 1,04 \times 10^{-4} = 104 [\mu\text{s}]$$

Reg. despl: ~~D₀~~

* Siempre se debe acordar:

- velocidad de transmisión
- cant. de datos a transmitir
- bits de paridad
- cant. de stop bits

→ Paridad: EVEN PAR 0
 ODD IMPAR 1

10.08.17



{
DTR →
DSR ←

{
RTS →
CTS ←

← DCD

← RI

→ TX

← RX

↔ GND

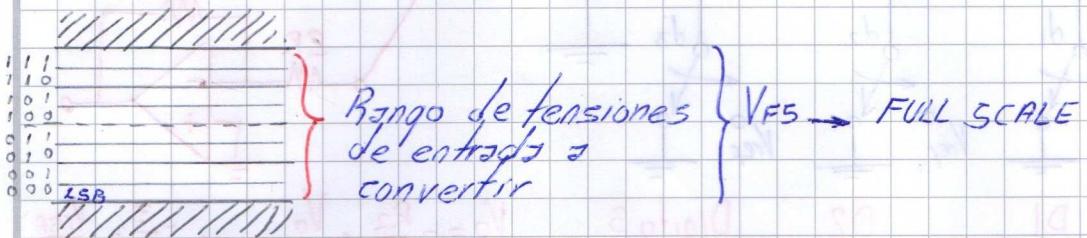
17.08.17

$$f_{clk} = 1.8432 \text{ MHz}$$

$$\frac{\div 16}{\text{DIVISOR}} \rightarrow \boxed{\begin{matrix} M5 \\ L5 \end{matrix}}$$

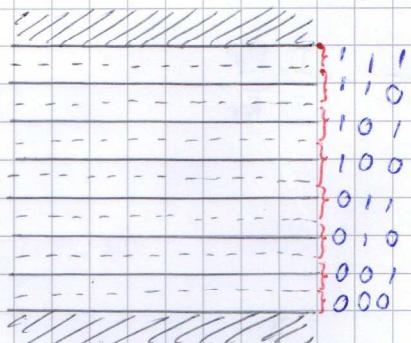
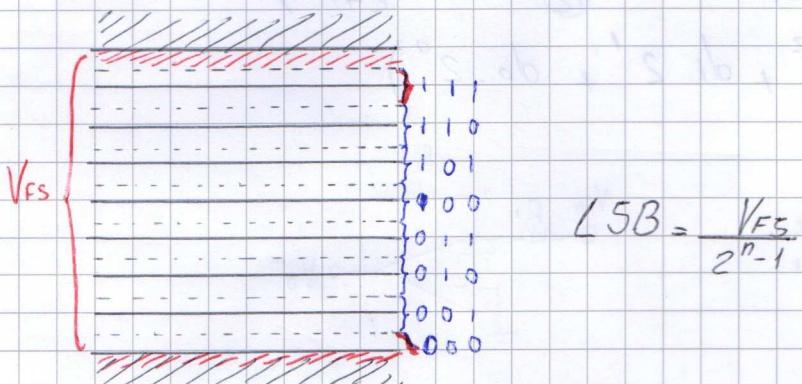
$$V = \frac{f_{clk}}{16 \times \text{DIVISOR}} \text{ [BAUDIOS]}$$

• Conversores A-D

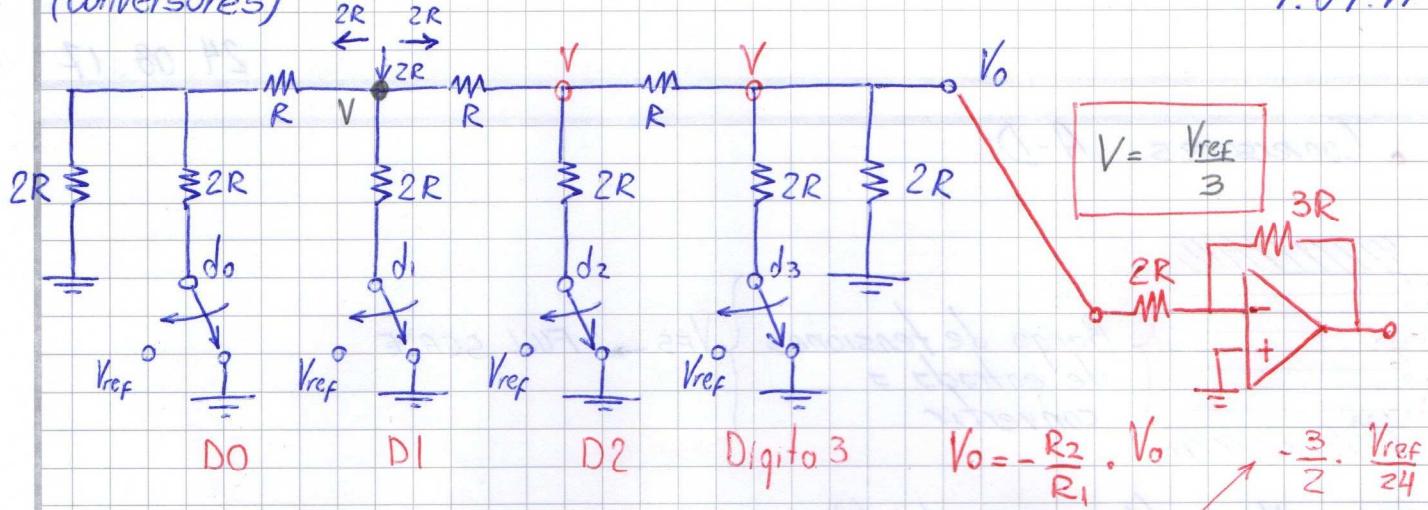


$$\text{LSB} = \frac{V_{FS}}{2^n} \quad (\text{less significant bit})$$

$$e_q = \pm \frac{1}{2} \text{ LSB} \quad (\text{error de cuantificación})$$

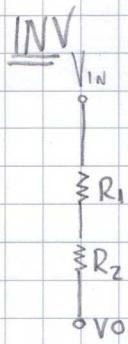


(Conversores)

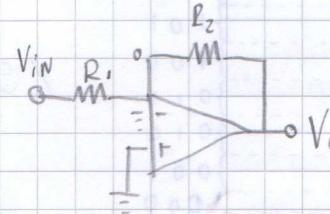


$$V_o = \frac{V_{ref}}{24} \left(d_3 \cdot \left(\frac{24}{3}\right)^8 + d_2 \cdot \left(\frac{24}{6}\right)^4 + d_1 \cdot \left(\frac{24}{12}\right)^2 + d_0 \cdot \left(\frac{24}{24}\right)^1 \right)$$

$$V_o = \frac{V_{ref}}{24} \left(d_3 \cdot 2^3 + d_2 \cdot 2^2 + d_1 \cdot 2^1 + d_0 \cdot 2^0 \right)$$



$$V_o = -\frac{R_2}{R_1} \cdot V_{in}$$



$$\frac{V_o}{V_{in}} =$$

$$V_{in} = I \cdot R_1$$

$$\frac{V_o}{V_{in}} = -\frac{I \cdot R_2}{I \cdot R_1}$$

$$V_o = -I \cdot R_2$$

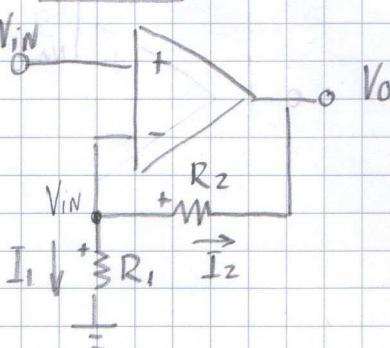
NO-INV

$$I_1 = I_2$$

$$I_1 = \frac{V_{in} - 0}{R_1} = \frac{V_{in}}{R_1}$$

$$I_2 = \frac{V_{in} - V_o}{R_2} = \frac{V_{in}}{R_2} - \frac{V_o}{R_2}$$

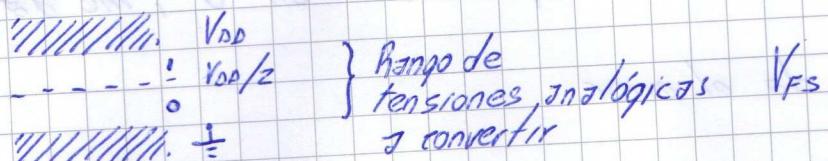
$$\left. \begin{aligned} -\frac{V_{in}}{R_1} &= \frac{V_{in}}{R_2} - \frac{V_o}{R_2} \\ \frac{V_{in}}{R_1 + R_2} &= -\frac{V_o}{R_2} \end{aligned} \right\} \frac{V_o}{V_{in}} = -\frac{R_2}{R_1 + R_2}$$



$$\frac{V_o}{V_{in}} = \frac{R_2}{R_1 + R_2}$$

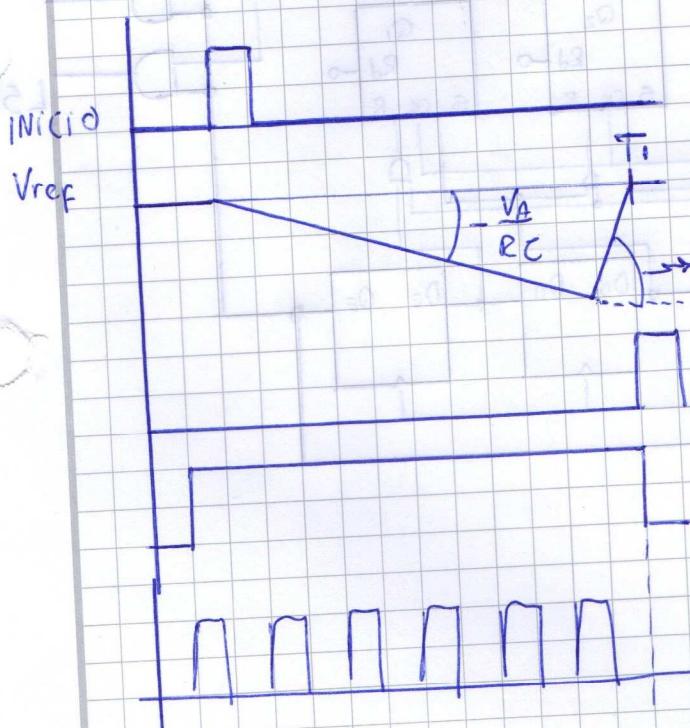
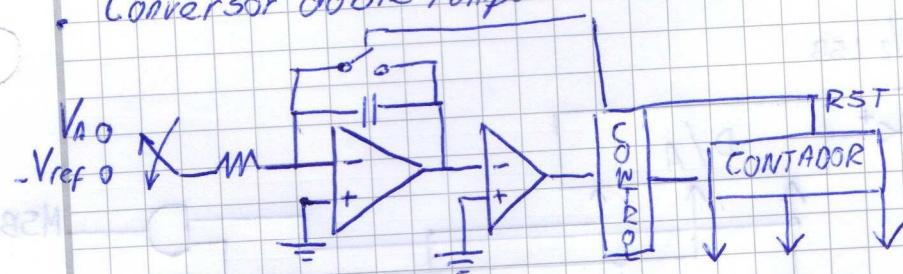
$$V_o = -\frac{V_{ref}}{2^n} \left(d_{n-1} \cdot 2^{n-1} + d_{n-2} \cdot 2^{n-2} + d_{n-3} \cdot 2^{n-3} + \dots + d_0 \cdot 2^0 \right)$$

• Comparador de 1 bit



$$\begin{aligned} V_o &= A(V_{in^+} - V_{in^-}) \\ V_{in^+} &= V_{DD} = 5V \\ V_{in^-} &= V_{DD}/2 \end{aligned}$$

• Conversor doble rampa



* Por más que R o C varíen, como ambas rampas dependen de RC , T_i no varía

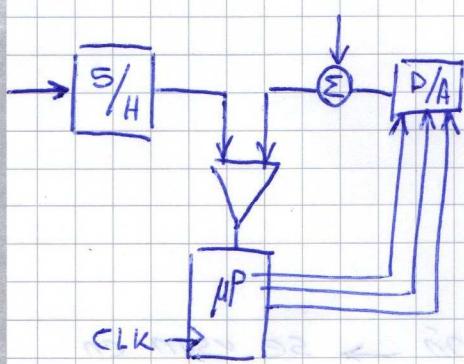
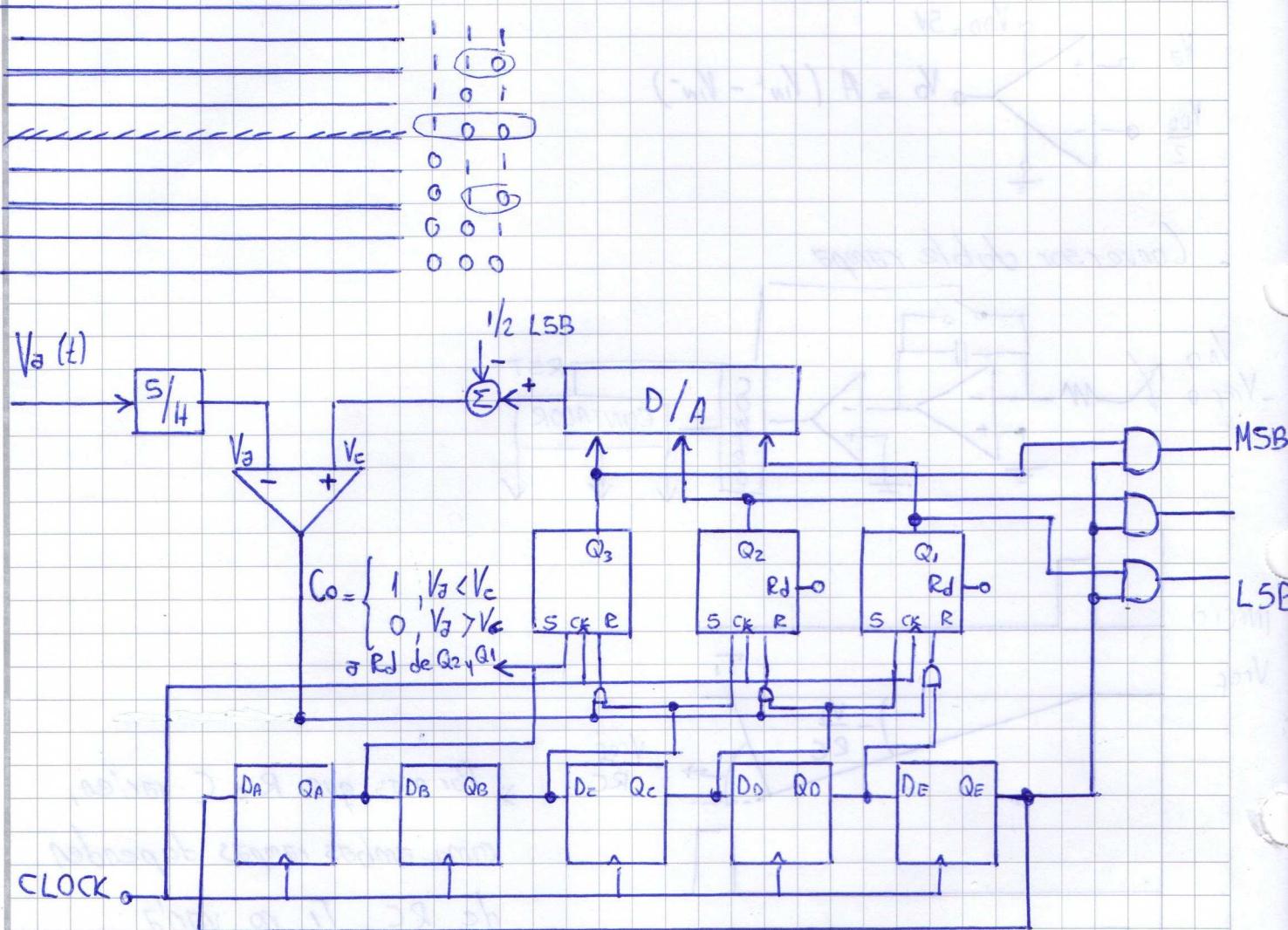
El flash tiene un periodo de clock

+ No importa la velocidad pero sí la precisión → se usan en voltmetros

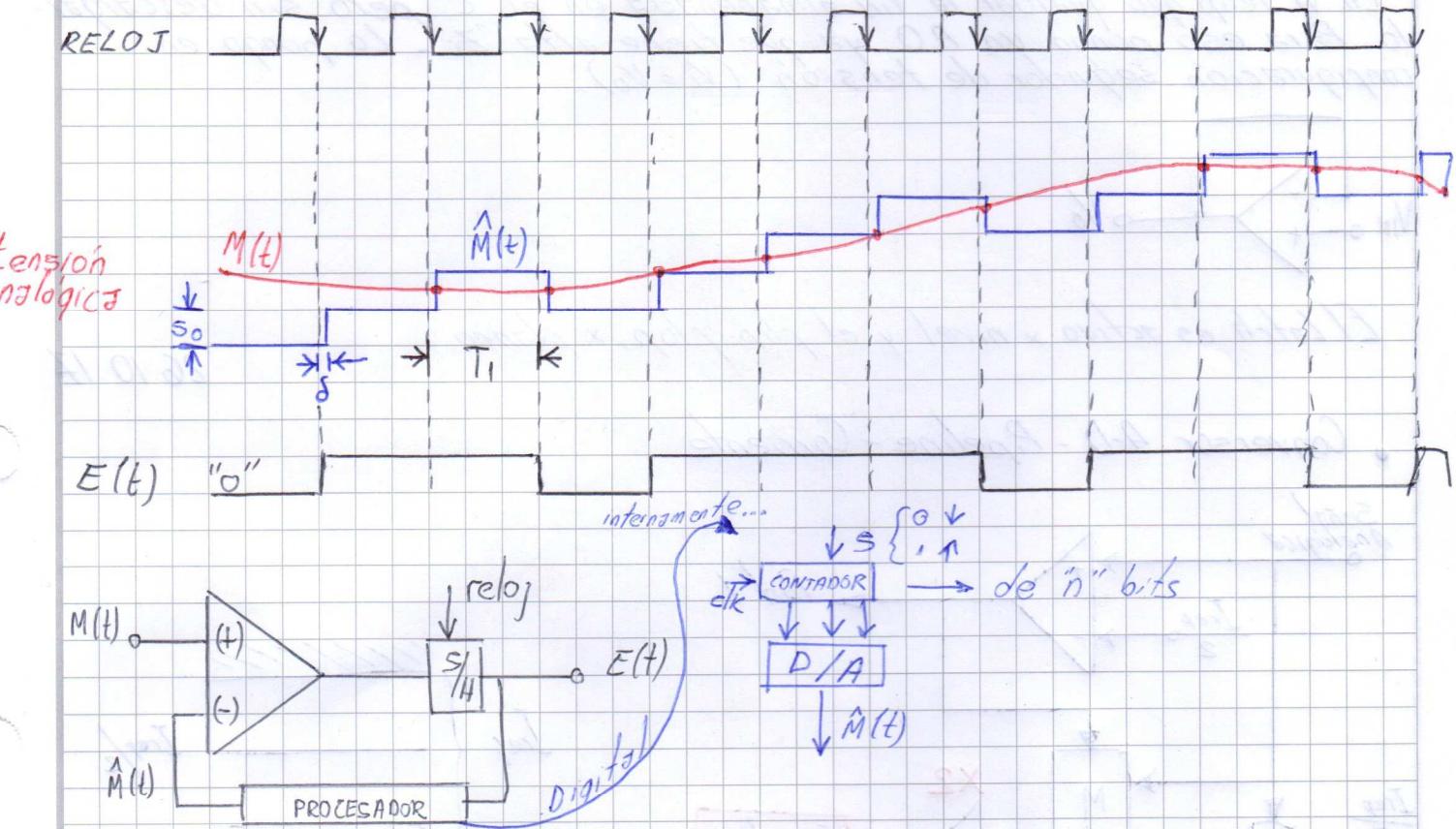
• Conversor Aproximaciones sucesivas

28.09.17

- El rango lo defino como el doble de la pesa + grande
- El margen de error lo defino como la mitad de la pesa + chico
- El bit + significativo puesta en "1" y el resto en "0", me da la mitad del rango
- La mitad del LSB me da el error



Modulador Delta



La señal analógica $M(t)$ es comparada con la digital $\hat{M}(t)$; si $\hat{M}(t)$ es menor a la continua, aumenta un " δ " o escalaón hasta el próximo pulso descendente del clock. En este pulso, vuelve a comparar y sube un " δ " o lo bajo.

Si además pongo un FPB, la señal de salida se verá: —



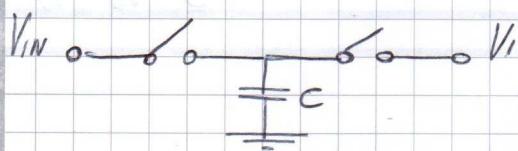
Circuit diagram of a first-order low-pass filter (LPF) with resistor R and capacitor C connected to ground, and output V_A .

Si aumenta la freq. de muestreo, los pulsos del clk. estarán más juntos y la recuperación de la señal es mayor

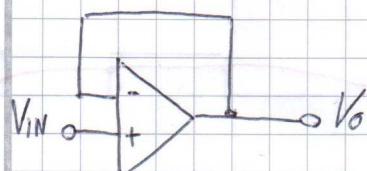
Los "n" bits del procesador dependen del rango de valores de la señal analógica

19.10.17

(Sample & Hold)



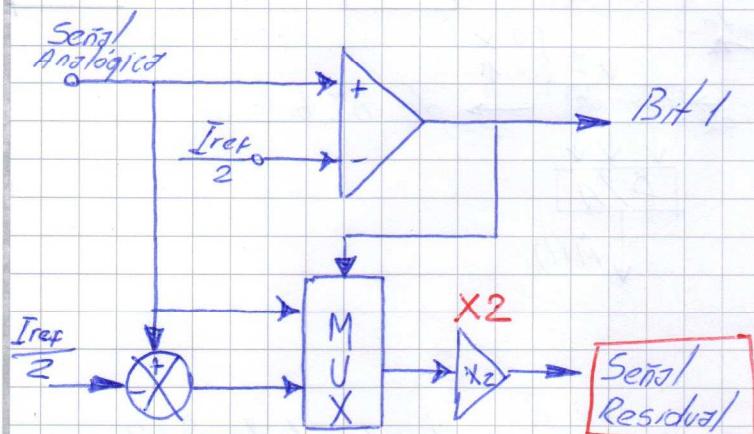
En V_h tengo que guardar la V_{in} almacenada en el "C" pero sin descargarla. Para eso pongo un A.O. ya que tiene alta "Z_i". Lo pongo en configuración seguidor de tensión ($V_o = V_h$).



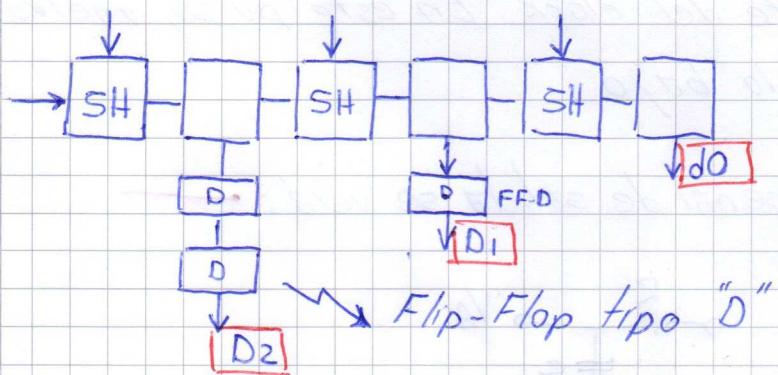
El latch es activo x nivel y el flip-flop, x flanco

26.10.17

• Conversor A-D - Pipeline - Corriente

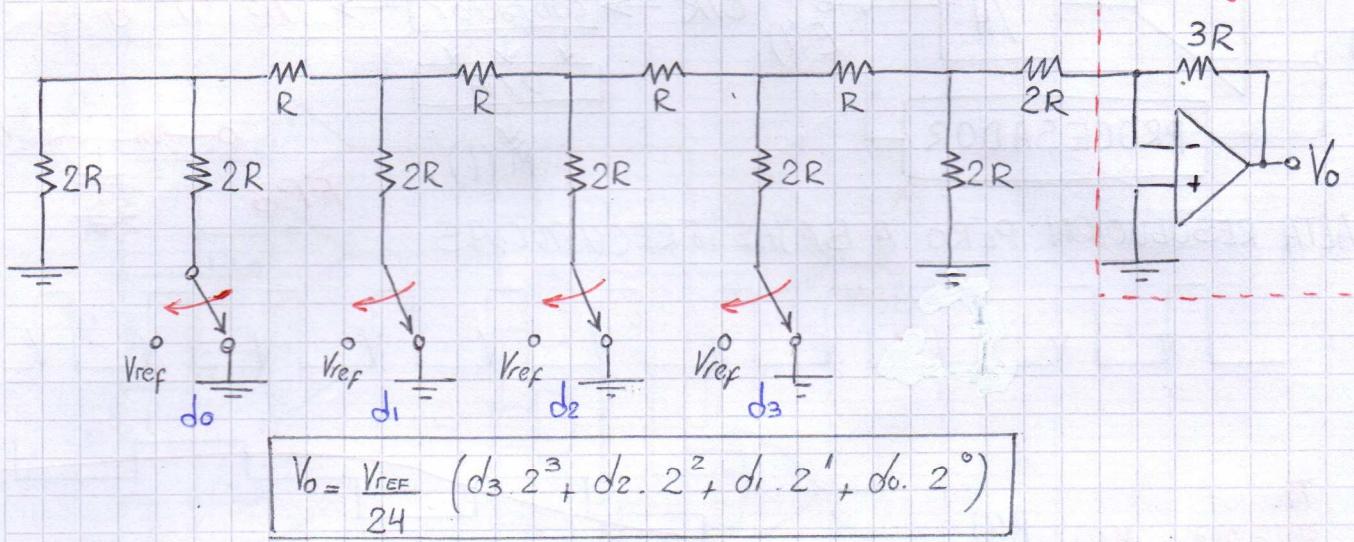


↑ Celda esquemática de 1bit



$$f_{ck} = \frac{1}{T_{ck}}$$

Conversor D-A 3 R - 2R



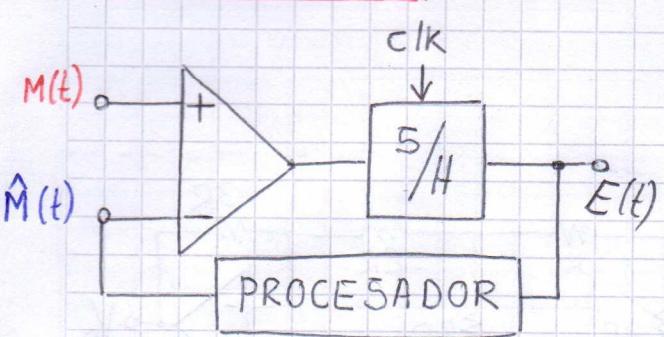
Ventajas: • Solamente se emplean resistencias de dos valores diferentes

- La variación de las resistencias con la temperatura será similar en todas ellas, y se pueden emplear valores pequeños cuando se necesite alta velocidad
- Alta velocidad de conversión
- Funcionamiento sencillo

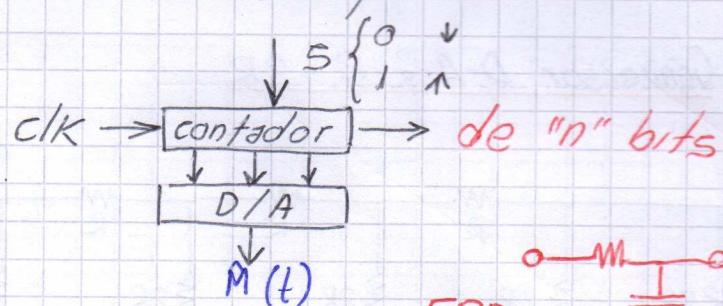
Desventajas: • Se requiere el doble de resistencias que en el caso del conversor de resistencias ponderadas.

- Los valores de resistencias tienen que ser precisos, sobre todo los R de los bits más significantes.

Modulador Delta

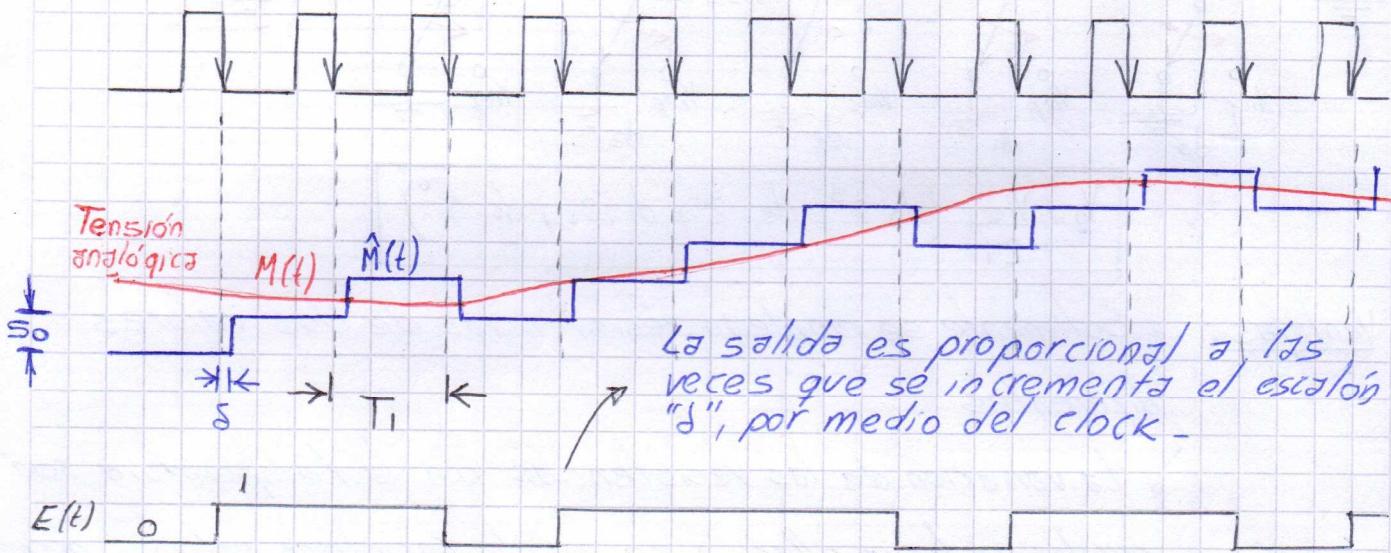


Procesador digital:



FPB M $\frac{V_A}{2}$

ALTA RESOLUCIÓN PERO A BAJAS FRECUENCIAS



La señal analógica $M(t)$ es comparada con la digital $M-hat(t)$; si $M-hat(t)$ es menor a la continua, aumenta un "d" o escalón hasta el próximo pulso descendente del clock.

Si aumento la frecuencia de muestreo, los pulsos del clock están más juntos, y la recuperación de la señal es mayor.

Los "n" bits del procesador dependen del rango de valores de la señal analógica.

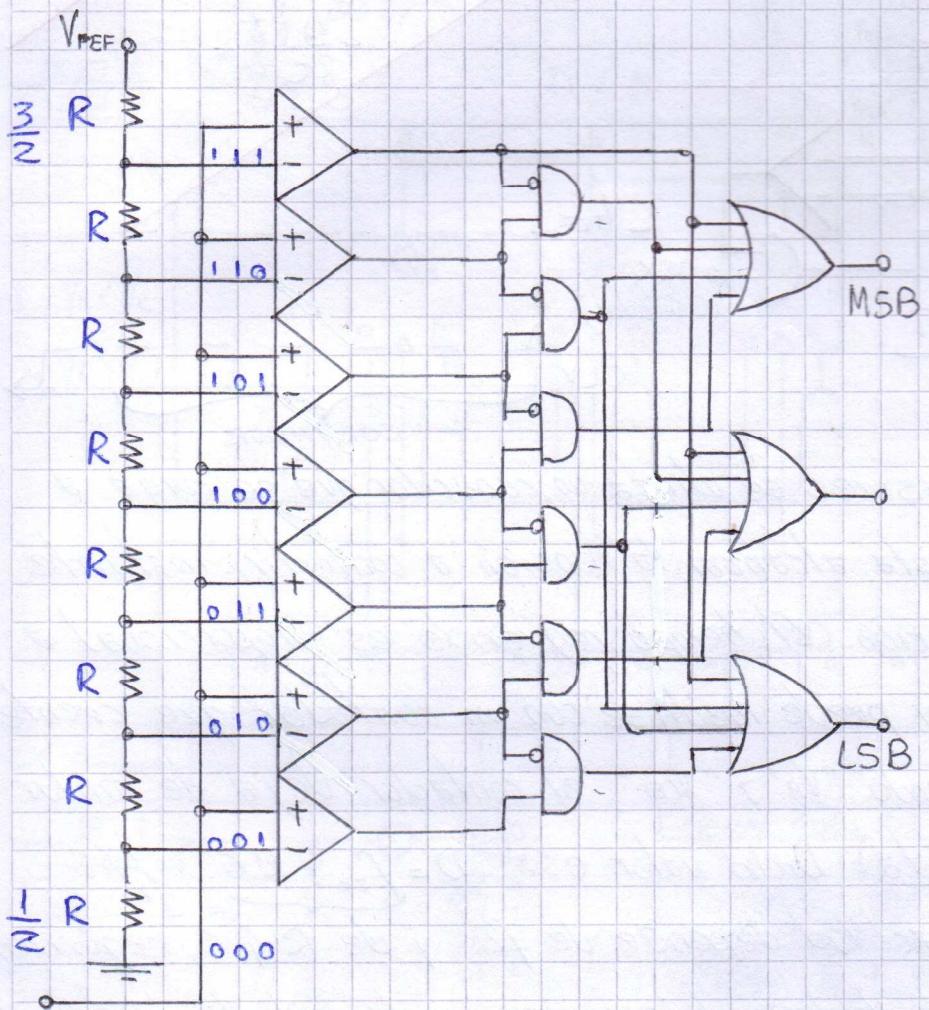
Las señales son muestreadas a una tasa mucho mayor que la de Nyquist, pero con un solo bit de resolución de amplitud.

La desventaja de este modulador de alta resolución, es la velocidad. El hardware tiene que operar a una tasa de muestreo mucho más alta que el BW de la señal, demandando un circuito complejo.

No se requiere un circuito externo de "S & H" dada la alta tasa de muestreo en la entrada y la baja presión del ADC.

NOTA

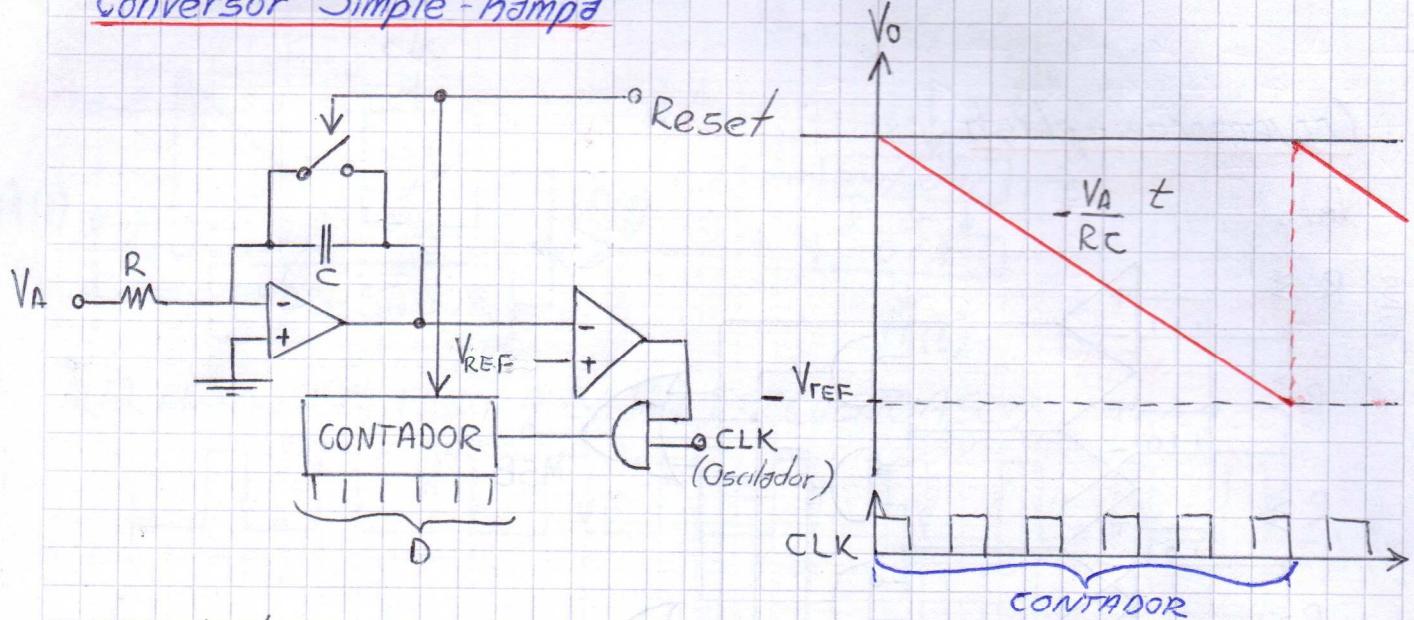
Conversores flash



Consiste en una serie de comparadores que comparan la señal de entrada con una referencia para cada nivel. El resultado de las comparaciones entra en un circuito lógico que cuenta los comparadores activados. Los valores de las R extremas difieren de los restantes para lograr que la comutación de un código al siguiente se produzca a mitad del camino del intervalo que corresponde a ese código. (Si $V_{ref} = 8V$, las comutaciones $\Rightarrow 0,5, 1,5, 2,5$)

La ventaja de este tipo de conversores es que la conversión es prácticamente en tiempo real, salvo el tiempo de comutación de los comparadores y la lógica. La desventaja es que cuando la resolución es alta requerirá gran cantidad de comparadores, cuyo offset debe ser menor a 1 LSB.

Conversor Simple-Rampa

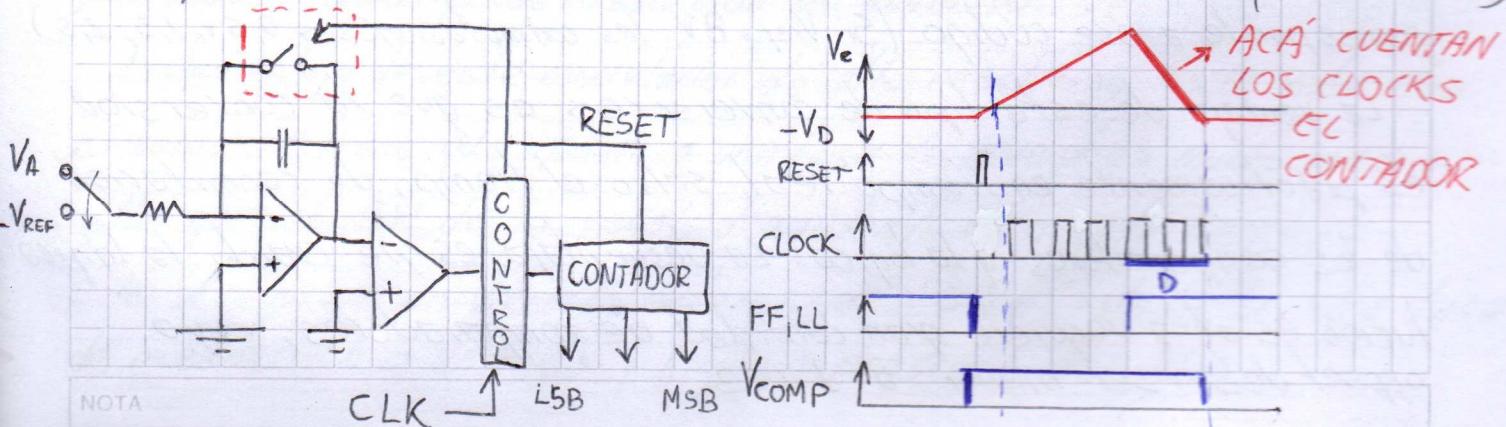


En este tipo de conversores se utiliza un capacitor que se carga a pendiente constante hasta alcanzar la tensión a convertir, instante en el que cesa la integración. El tiempo requerido es proporcional a la tensión de entrada y puede medirse con un contador que cuente ciclos de un reloj. Cuando $V_A > V_{REF}$, el contador deja de contar ciclos de reloj y comienza. Dicho valor es: $D = f_{CK} \cdot R.C. \frac{V_A}{V_{REF}} = 2^n$

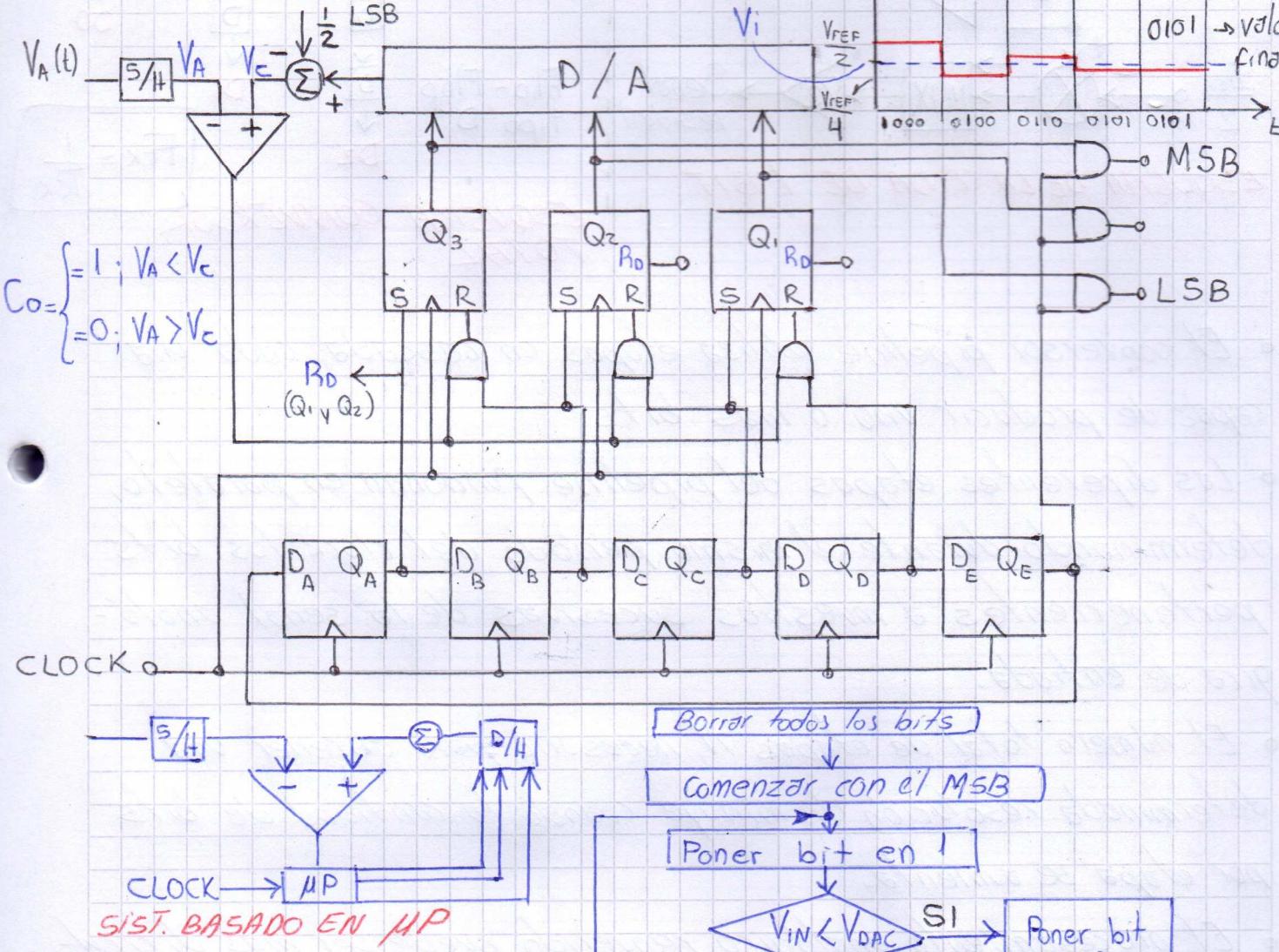
Desventajas: • La exactitud depende de f_{CK} y de R y C , requiriendo componentes no solo de baja tolerancia, sino tamb. V deriva térmica.

Conversor Doble-Rampa

Este esquema permite independizarse de la presición de f_{CK} , R y C . En la primera etapa, se realiza una integración de la tensión de entrada durante un tiempo fijo, y en la segunda se produce la descarga, con pendiente fija durante un tiempo que depende de la cantidad de carga acumulada. NO SON RÁPIDOS PERO SI PRECISOS (VOLTMETROS)



Conversor aproximaciones sucesivas



Al dar una señal de inicio de conversión, el registro aplica "1" en el MSB del conversor AD, y en los bits restantes, "0". La salida se ubica en la mitad de la escala ($V_{REF}/2$). Si $V_i > V_{REF}/2$, el MSB queda en "1". Si no sucede esto, el MSB vuelve a cero.

En el paso siguiente, el bit " $n-1$ " es llevado a "1" y se procede de la misma forma hasta llegar al LSB.

Con este tipo de conversor, el tiempo de conversión es de " n " ciclos de reloj, en lugar de 2^n como en otros casos. A diferencia del flash, en este caso se requiere que la entrada se mantenga rigurosamente constante, sino se producen errores. En efecto, una vez que los bits más significativos han quedado fijos, ya no es posible cambiarlos hasta la próxima conversión, por eso el proceso continúa buscando la mejor aproximación que sea posible con los restantes bits. Por esta razón se requiere un SAMPLE & HOLD.

El doble del bit MSB, me da el rango
La mitad del bit LSB, me da el error

Borrar todos los bits
Comenzar con el MSB

Poner bit en 1

$V_{IN} < V_{DAC}$

SI → Poner bit en 0

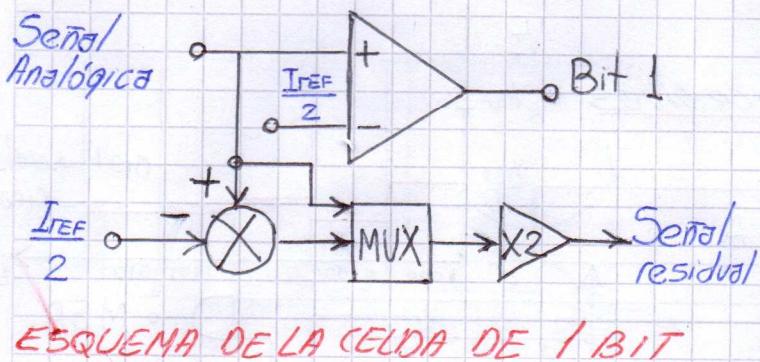
NO

Ir al bit LSB siguiente
Se llegó "n" bits

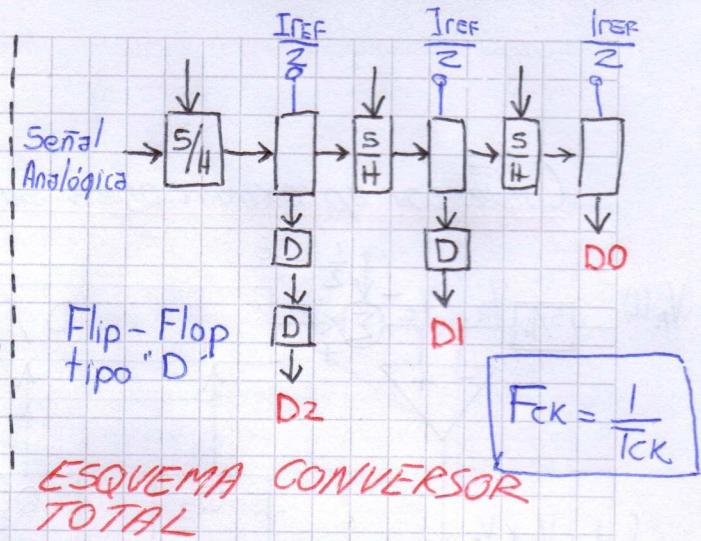
Transferring Data - FIN

DIAGRAMA DE FLUJO - SOFTWARE

Conversor Pipeline



ESQUEMA DE LA CELDA DE 1 BIT



- El conversor pipeline utiliza etapas en cascada, cada una capaz de producir uno o más bits
- Las diferentes etapas del pipeline funcionan en paralelo, determinando durante el mismo período del clock los bits pertenecientes a muestras sucesivas de la señal analógica de entrada.
- El número total de etapas N , necesario para obtener una determinada resolución disminuye cuando la cantidad de bits por etapa se aumenta.
- El procesamiento analógico requerido puede ser más difícil cuando la velocidad y el consumo de una simple etapa aumenta.

VELOCIDAD DE TRANSMISIÓN: En la transmisión sincrona, por cada carácter se envía el menor, y de inicio, y final que precede los bits de transmisión.

- A continuación del LSB se envía el bit (0 los bits) del final que hace que la linea se ponga a "1" por lo menos durante el tiempo mínimo de un bit.
- A continuación del LSB se envía el bit (0 los bits) del final que hace con los intervalos que marca el reloj de transmisión. Por lo que, transmite 5,8 bits.
- A continuación se envían todos los bits del dato a transmisor.
- Durante la transmisión, si la linea es 0, se envía un "0", y si es 1 se envía un "1".
- Cuando se envía un carácter, se envía un bit de inicio "0" durante un tiempo de bit.
- Cuando se envía un carácter, se envía un espacio alfa.
- Durante la transmisión, si la linea es 0, se envía un bit de inicio "0", y si es 1 se envía un espacio alfa.

REGLAS DE TRANSMISIÓN ASÍNCRONA: • Cuando no se envían datos por la linea, existe una máquina en estados alfa.

Transmisión asíncrona: No existe una linea de reloj común que establece la duración de un bit y al carácter precede ser enviado en celdas para cada uno de los caracteres.

TRANSMISIÓN ASÍNCRONA: No existe una linea de reloj común que establece la duración de un bit y al carácter precede ser enviado en celdas para cada uno de los caracteres.

Full duplex: Comunicación en dos líneas - (simultánea)

Semi duplex: Comunicación en una sola linea pero ambos sendidos.

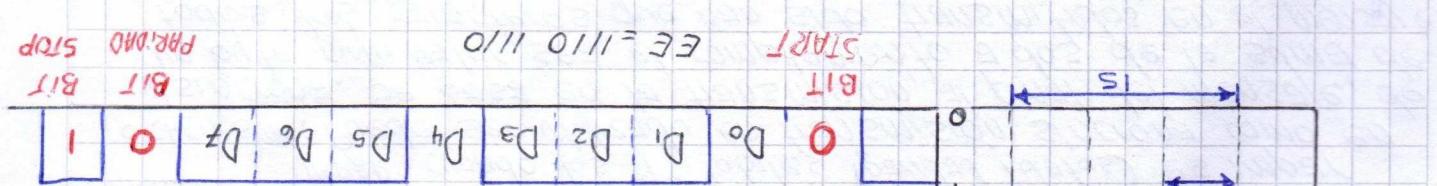
No ambos. La comunicación tiene una dirección.

LÍNEAS DE COMUNICACIÓN: SIMPLEX: Existe un transmisor y un receptor, definida basíicamente por el clock.

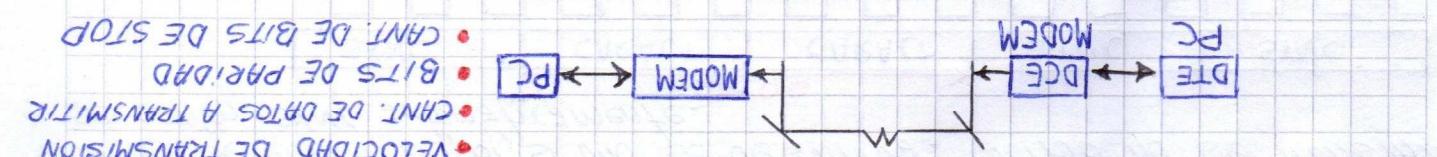
Por segundo - Se diferencia cuando es necesario más de un bit para representar más de dos estados de la señal.

que proceden ser representados por un bit, que identifica dos unidades de información), en fonics Baudio es equivalente a 8 bits.

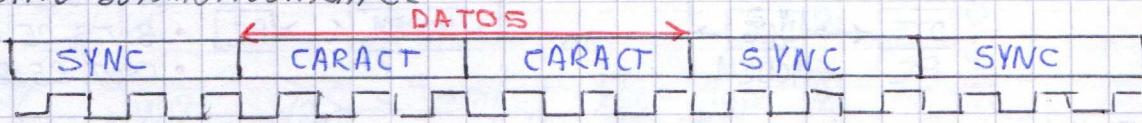
de estados de la señal por segundo, si solo existe dos estados de segundos de la señal como el número



$$T = \text{tiempo de duración de un bit} \rightarrow V = \frac{1}{f} \left[\frac{\text{bits}}{\text{s}} \right]$$



TRANSMISIÓN SÍNCRONA: Lo primero que se envía es un octeto de sincronismo (SYNC), indica al receptor que se va a enviar un mensaje. Estos SYNC deben diferenciarse de los datos del usuario. A veces se envían 2 por si uno se desvirtúa. Cuando no se transmite, se envían SYNC automáticamente.



ERRORES DE COMUNICACIÓN :

AISÍNCRONA :

- **Paridad:** PAR=0 cuando los "1" sea par ^ PAR=1 cantidad de "1" impar
IMPAR cuando los "1" totales (paridad + datos) es impar
- **CHECKSUM:** Puede ser utilizado en transmisión síncrona como en asíncrona. Se basa en la transmisión, el final del mensaje, de un byte cuyo valor, sea el complemento a dos de la suma de todos los caracteres que han sido transmitidos en el mensaje. Transmitir $40H, 35H$ y $0EH = \Sigma = 83H \rightarrow C_2 \rightarrow 7DH \rightarrow$ último dato a transm.

EVEN PAR 0
ODD IMPAR 1

CONVERSIÓN SERIE / PARALELO

Por Software :

- **Transmisión de datos:** La rutina de transmisión puede ser llamada por interrupción, cada vez que un dato serie quiere ser transmitido o porque se establezca en una o varias partes del flujo del programa que ejecuta el microcontrolador.
La rutina debe en primer lugar colocar la línea a cero durante el tiempo de un bit para establecer el bit de inicio, posteriormente el acumulador será enviado a puerto RA0, donde transmitirá el LSB.
- **Recepción de datos:** Debe detectar (puede ser por interrupción) el bit de inicio. Luego espera el tiempo de duración de 1 bit para encuestar la primera unidad de información del dato a recibir en la mitad de su intervalo. Luego el contador es establecido con el número de bits de datos que serán recibidos, el cual irá decrementando.
Luego el programa debe checar los bits de parada. También se puede corroborar el bit de paridad.

Por Hardware :

- Los datos paralelos a ser convertidos entran al registro buffer y son transferidos al registro para la transmisión de datos. Los datos son desplazados a la salida a través de una línea de salida serie a una velocidad determinada por el reloj y la sección de control. Los bits de inicio, parada y paridad son añadidos automáticamente por la UART.

A la UART se le deben especificar varios parámetros

- Bits de datos por carácter (5 a 8)
- Bits de parada (1, 1.5 - 2)
- Bit de paridad, para utilizar su capacidad de detección error
- Velocidad de transmisión.

- **Error de sobreescritura:** La UART presenta una estructura interna con doble buffer, esto le permite tener un carácter almacenado en el buffer de la recepción mientras que el registro de desplazamiento sigue paralelo continuando ensamblando un nuevo carácter. Si el µP no lee el dato almacenado en el buffer antes de que ensamblé un nuevo carácter, se produce OVERWRITE (el buffer)