Cika Electrónica

Tutorial: CTU-003
Título: **Rabbit 3000**
Autor: Sergio R. Caprile, Senior Engineer

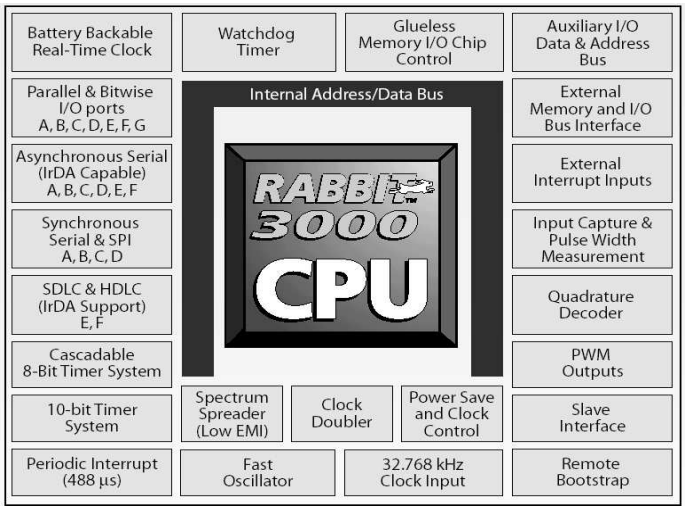
Revisiones	Fecha	Comentarios
0	29/08/03	
1	18/08/04	Corrección ports paralelo y captura de eventos

El presente es un tutorial sobre microprocesadores Rabbit 3000, el segundo miembro de la familia de microprocesadores Rabbit, basados en la arquitectura del Z-80/180. El tutorial cubre las principales diferencias y mejoras de estos procesadores con respecto al Rabbit 2000, por lo que se recomienda la lectura del tutorial sobre Rabbit 2000 si el lector no tiene conocimientos previos de la familia Rabbit.

Índice de contenido

Introducción

Rabbit Semiconductor se forma expresamente para diseñar microprocesadores orientados a control de pequeña y mediana escala; su segundo producto es el microprocesador Rabbit 3000. El diseño del mismo se centra en agregar nuevas prestaciones por sobre las ya excelentes prestaciones de Rabbit 2000.



Novedades

- Encapsulado de 128 pines, TFBGA ó LQFP. Clock de hasta 54MHz. Rango de temperatura comercial e industrial.
- Operación a 3V, entradas 5V tolerant.
- Características especiales para baja interferencia electromagnética (EMI), como spectrum spreader en el oscilador principal, pines de alimentación separados para la CPU y los I/O, y bus auxiliar de I/O.
- Menor consumo por MHz que Rabbit 2000.
- Más cantidad de ports paralelo y serie.
- 4 salidas PWM.
- 2 entradas para codificadores en cuadratura.
- 6 ports serie con capacidad IrDA.
- 2 ports serie con capacidad SDLC/HDLC.
- Más timers
- Más modos de operación de bajo consumo (Ultra-Sleepy)
- Control de memoria optimizado para bajo consumo (short CS)
- Timing de acceso a memoria extendido para operación en alta frecuencia

Principales diferencias con Rabbit 2000

	R3000	***R2000***
Frecuencia máxima de reloj	54 MHz	30 MHz
Frecuencia máxima de cristal (puede duplicarse internamente)	27 MHz	32 MHz
Oscilador de 32,768 kHz	Externo	Interno
Tensión máxima de operación	3,6 V	5,5 V
Tensión máxima en pines de I/O	5,5 V	5,5 V
Corriente de operación	2 mA/MHz a 3.3 V	4 mA/MHz a 5 V
Número de pines en el encapsulado	128	100

	R3000	***R2000***
Tamaño del encapsulado	16 × 16 × 1,5 mm LQFP 10 x 10 x 1,2 mm TFBGA	24 × 18 × 3 mm PQFP
Espaciado entre pines	0,4 mm (16 mils) LQFP 0,8 mm TFBGA	0,65 mm (26 mils) PQFP
Modos de clock	1x, 2x, /2, /3, /4, /6, /8	1x, 2x, /4, /8
Modos de operación en bajo consumo	Sleepy (32 kHz) Ultra-sleepy (16, 8, 2 kHz)	Sleepy (32 kHz)
Cantidad de ports de I/O de 8 bits	7	5
Cantidad de ports serie	6	4
Ports con capacidad SPI	4 (A, B, C, D)	2 (A, B)
Máxima velocidad en modo asincrónico	Clock/8	Clock/32
Timers de 8 bits	10	5

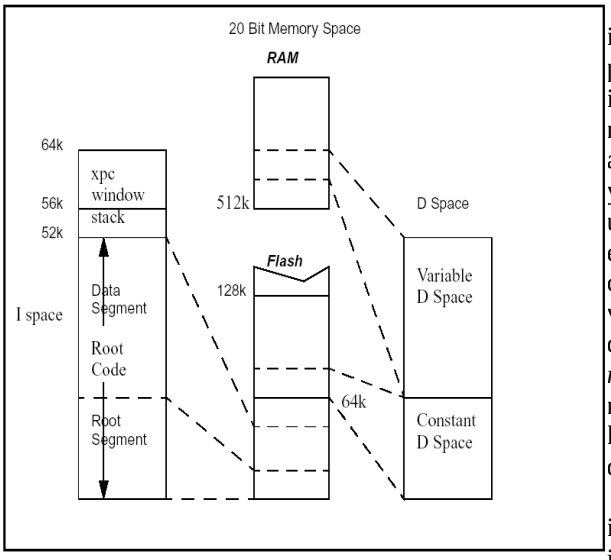
Mejoras y diferencias

Manejo de Memoria

Espacios separados de Instrucciones y Datos

En el modelo de memoria normal, los datos deben compartir un espacio de 64K con el código *base*[1], el *stack*, y la ventana *XPC*. Típicamente, esto nos deja con unos potenciales 40K o menos como espacio de datos; debido a que se requieren 8K para XPC, unos 4K para stack y la mayoría de los sistemas necesitan unos 12K de código en el segmento *base*, como mínimo. Esta cantidad de memoria para datos es suficiente para muchas aplicaciones de sistemas dedicados.

Una forma de obtener mayor espacio de datos es colocarlos en RAM o flash que no mapee dentro del espacio de 64K, y luego accederlos usando funciones especiales o en assembler, con instrucciones de 20 bits. El problema es que estas instrucciones son mucho menos eficientes que las de 16 bits.



El Rabbit 3000 soporta espacios separados para instrucciones (*I*) y datos (*D*). Esto significa que el procesador realiza una distinción entre tomar una instrucción de la memoria y leer o escribir datos en memoria. Cuando se habilita esta función, la misma aplica sólo para las direcciones en los segmentos *base* y *data*, combinando ambos de forma tal de obtener unos 52K de memoria para código (espacio *I*). El espacio *D* se forma asignando el segmento *base* para constantes, mapeado en flash, y el segmento *data* para variables, mapeado en RAM. Se incrementa el espacio disponible para código en área *root* y datos en área *root* debido a que ya no es necesario que compartan el mismo espacio de memoria lógica, aún cuando utilicen las mismas direcciones lógicas, dado que el procesador discrimina instrucciones de datos.

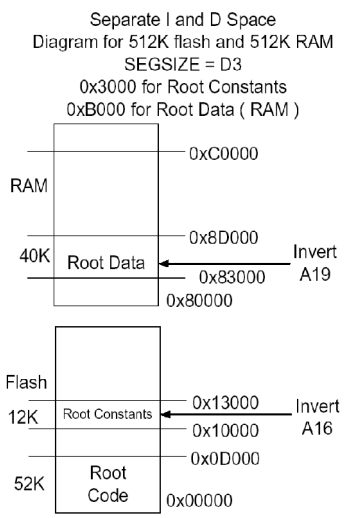
Generalmente, la separación de espacios *I* y *D* es implementada como figura en el diagrama a la izquierda. En el espacio *I*, los segmentos *base* y *data* se

1 La bibliografía de Rabbit utiliza el término *root* para referirse al segmento *base*. A fin de evitar confusiones, trataremos de utilizar ese término para referirnos al área (*base* + *data*) y no al segmento en sí.

combinan en un solo segmento de código comunmente llamado *root*. En el espacio *D*, los segmentos *base* y *data* mapean por separado a flash y RAM, en ese orden, de modo de albergar datos constantes y variables, respectivamente.

La forma de realizar esto en el hardware es mediante un simple truco: invertir una de las líneas de direcciones (A16 ó A19) cuando el acceso es de datos; la inversión puede especificarse por separado para los segmentos *base* y *data*. Normalmente, se invierte A16 cuando se acceden datos en el segmento *base*, esto ocasiona que el acceso se haga en una posición desplazada 64K (2^{16}) de la dirección física utilizada por el espacio *I*. Si el acceso de datos es en el segmento *data*, entonces se invierte A19, ocasionando un desplazamiento de 512K (2^{19}) respecto al espacio *I*.

Como sabemos, Dynamic C y los módulos utilizan flash a partir de la posición 0x00000 (512K inferiores) y RAM a partir de 0x80000 (512 K superiores). Además, el segmento *base* generalmente mapea a la dirección 0x00000 para arrancar ejecutando desde flash. Con estos datos, es fácil deducir que al activar la separación de espacios *I* y *D*, los datos en segmento *base* (constantes) mapean a partir de la dirección 0x10000 (64K por encima del inicio del segmento base), en flash, y los datos en segmento *data* (variables) mapean a partir de la dirección 0x80000+*data* (512K por encima del inicio del segmento *data*). Esto puede apreciarse en el diagrama a la derecha.



Tanto el segmento *stack* como el *XPC* no son afectados por este esquema, y mapean normalmente; el sistema no distingue entre instrucciones y datos en estos casos.

La ventaja de esta separación es generar mayor espacio de código en el área *root*; esto significa que el código se ejecuta más rápido porque se lo llama con instrucciones de direccionamiento en 16 bits, que son inherentemente más rápidas, comparadas con las de 20 bits. Los datos, a su vez, se acceden de forma más rápida y simple debido a que no es necesario manipularlos con instrucciones de 20 bits ni traerlos desde *xmem*.

Bus auxiliar de I/O

En los diseños tradicionales de microprocesadores, se utilizan los mismos buses de direcciones y datos tanto para las memorias como para los periféricos de I/O; esto suele forzar compromisos o complicar el diseño. Generalmente, el bus de memoria tiene un timing más crítico y menos tolerante de la carga capacitiva adicional que suele imponer el bus de I/O. Con el Rabbit 3000, el diseñador tiene la opción de utilizar buses separados para memoria e I/O.

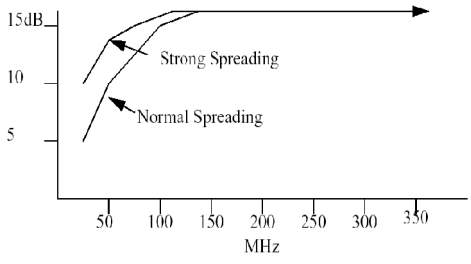
El bus auxiliar comparte los pines con el *slave port*, por lo que ambos son mutuamente excluyentes. El *port A* provee las líneas de datos y el *port B* provee seis líneas de address, las seis menos significativas de las dieciseis en total del espacio de I/O. Estas cambian de estado en cada acceso I/O y permanecen así hasta el próximo acceso. Los ciclos de I/O se ejecutan en paralelo en ambos buses (principal y auxiliar), por lo que, de ser necesario, es posible tomar líneas de address adicionales del bus principal.

Al conectar dispositivos periféricos al bus auxiliar, se libera al bus principal de la carga capacitiva adicional y es posible trabajar más rápido con la memoria. Como el bus de I/O tiene menos actividad, y es más lento que el de memoria, puede extenderse su longitud sin generar problemas de interferencias electromagnéticas (EMI). Por último, dado que las entradas de Rabbit 3000 son 5V tolerant, es posible conectar aquí dispositivos de 5V, lo cual podría causar problemas en el bus principal si se utilizan memorias de 3,3V.

Reducción de interferencias (EMI)

Spectrum spreader

El sistema principal de reloj puede modificarse por un módulo *spectrum spreader* interno. Cuando se activa este bloque, el clock se acelera y retrasa alternadamente, modificando el espectro harmónico de la señal. Esto reduce interferencias electromagnéticas y mejora de 15 a 20dB la respuesta a los tests de emisiones en frecuencias críticas.



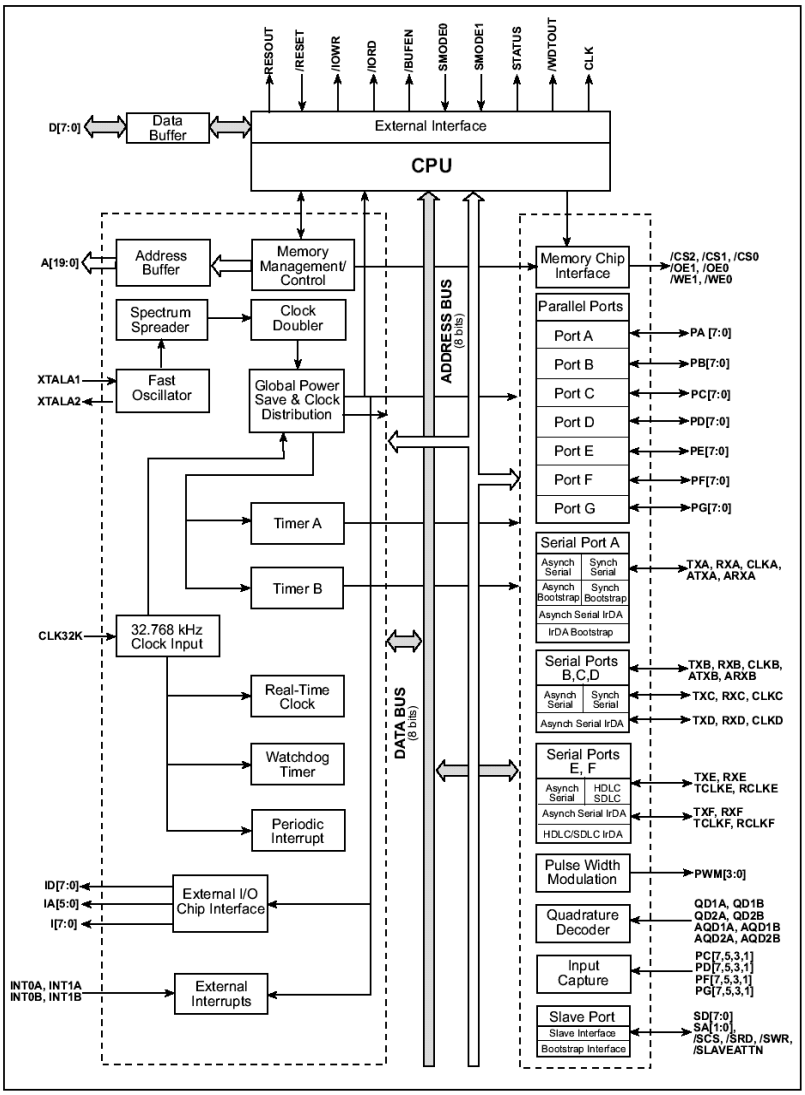
El módulo tiene tres opciones de operación: *desconectado*, *normal*, e *intenso*. El uso del mismo requiere un tiempo de acceso a memoria más rápido, de unos 2 a 3ns para el modo *normal* y de 6 a 9ns para el modo *intenso*. Si bien esto altera baud rates y demás temporizaciones al introducir clock jitter, el efecto es lo suficientemente bajo como para no tomarse en cuenta.

Pines de alimentación separados para core e I/O

Se ha dividido la oblea de silicio que constituye el Rabbit 3000 en dos partes: core logic y anillo de I/O. Esta última se localiza en los cuatro bordes y contiene a los transistores más grandes, que se utilizan precisamente como buffers de I/O. La core logic, en el interior, contiene al procesador y la lógica periférica, que operan a velocidades altas y con grandes corrientes transitorias que generan ruido, el cual se comunica al exterior por los pines de alimentación. Los buffers de I/O, por el contrario, tienen tiempos de conmutación más bajos y operan mayormente a frecuencias mucho más bajas. La separación de pines permite al diseñador proveer the alimentación filtrada al anillo de I/O, minimizando el ruido de alta frecuencia que aparece en los pines de salida, y por consiguiente la EMI.

Periféricos en chip

El diagrama que figura a continuación muestra un esquema de la estructura interna del Rabbit 3000, donde pueden observarse, entre otras cosas, los diferentes bloques internos que lo componen y los diversos periféricos que han sido incluidos en el mismo.



Rabbit 3000 dispone de 56 líneas de entrada/salida, repartidas en siete ports de 8 bits designados como *port A*, *B*, *C*, *D*, *E*, *F* y *G*. La mayoría de los pines utilizados tiene funciones alternativas, como port serie o chip select strobe.

Los ports *D*, *E*, *F* y *G* tienen la capacidad de sincronizar sus salidas con un timer. Todos ellos menos el E pueden configurarse como open drain.

Ports Serie

El Rabbit 3000 tiene seis ports serie, designados como *port A*, *B*, *C*, *D*, *E* y *F*. Todos pueden funcionar en modo asincrónico, pero los ports *A*, *B*, *C*, y *D* pueden además funcionar en modo sincrónico. Los ports *E* y *F* soportan además comunicaciones SDLC/HDLC. Todos los ports soportan un modo opcional RZ (Return to Zero) con timing de 3/16 (SIR) ó 1/4 (MIR) de bit; esta opción se utiliza normalmente para IrDA

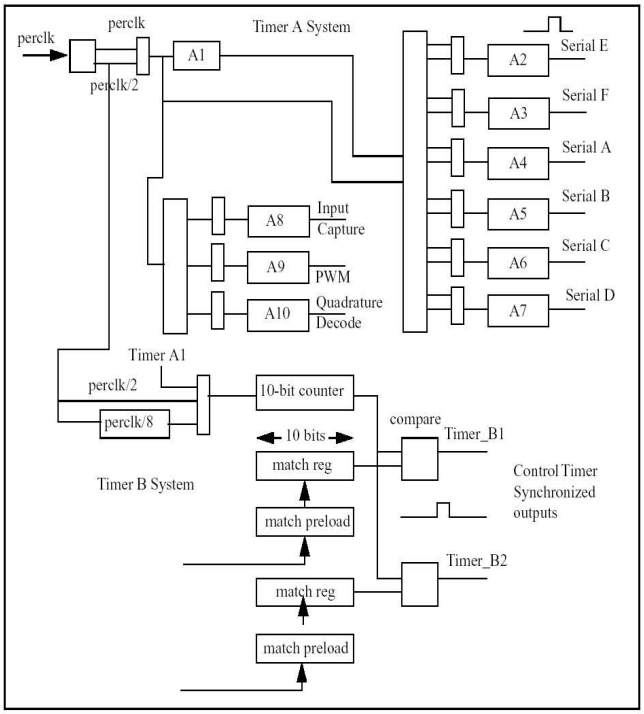
System Clock

El clock de 32,768KHz proviene ahora de un oscilador externo, existen documentos de Rabbit con un circuito sugerido de muy bajo consumo. Este oscilador continúa siendo la fuente de reloj del wtachdog timer y el port serie para la operación de bootstrap.

El clock principal tiene más opciones de división.

Timers

El diagrama a continuación muestra la nueva estructura de timers, puede observarse que se ha ampliado notablemente el *Timer A*, con el fin de soportar los ports serie adicionales, y las nuevas funciones que veremos más adelante.



Captura de eventos

Las entradas de captura de eventos se utilizan para determinar el momento en que se produce un evento particular. Esto es señalizado mediante un flanco cualquiera (o ambos) en alguno de los dieciseis pines que pueden ser configurados para este propósito, asignados a uno de dos canales posibles. Para llevar cuenta del tiempo, se emplea un contador de 16 bits que recibe reloj del *Timer A8*.

Se reconocen dos tipos de eventos: *condición de inicio* y *condición de detención*. La primera condición puede utilizarse para iniciar la cuenta y la segunda para detenerla, sin embargo el contador puede contar de forma contínua o hacerlo hasta que detecte la *condición de detención*. Ambas condiciones pueden además utilizarse

para registrar la cuenta en el momento de producirse el evento, en vez de iniciar y/o detener el contador. Ambas condiciones pueden asignarse indistintamente a un flanco y/o un pin determinados.

Una ventaja de esta arquitectura es que puede medirse fácilmente la duración de los pulsos, dado que puede configurarse al contador para arrancar en un flanco y detenerse en el otro. Esto permite medir eventos muy cortos, en los cuales no podría esperarse atender interrupciones y leer dos cuentas diferentes para luego obtener la duración restando ambas cuentas.

Entradas para codificadores en cuadratura

Un codificador en cuadratura (quadrature encoder) es un dispositivo electromecánico que se utiliza para seguir la rotación de un eje. Se implementan generalmente con un disco que alterna franjas opacas y transparentes, excitando dos detectores ópticos. Las señales de salida son dos ondas cuadradas en cuadratura; la dirección de rotación se detecta observando cuál de las dos señales adelanta en fase.

El Rabbit 3000 dispone de dos unidades, cada una con dos entradas (normal y cuadratura). Un contador bidireccional de 8 bits registra los eventos contando en una u otra dirección; la cuenta puede extenderse mediante una interrupción de desborde en ambos sentidos (overflow/underflow). Las señales externas se sincronizan con un reloj interno provisto por el *Timer A10*.

Salidas PWM

Estas salidas generan un tren de pulsos periódicos en una trama de 1024 pulsos. El ciclo de trabajo puede variar entre 1/1024 y 1024/1024 (100%). Existen cuatro unidades independientes, que reciben clock del *Timer A9*, controlando la longitud de los pulsos.