ANEXO D:
GUÍA DE REFERENCIA PARA PROGRAMACIÓN DEL DSK TMS320C6416T

ZULMA BERNARDA PABÓN PIPICANO
ANCIZAR SANTACRUZ AHUMADA

Director
I. E. Juan Fernando Flórez Marulanda

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
POPAYÁN
2007

ANEXO D:
GUÍA DE REFERENCIA PARA PROGRAMACIÓN DEL DSK TMS320C6416T

D.1. CARACTERÍSTICAS:

El C6416T DSK es una plataforma de desarrollo en un solo equipo de bajo costo que les posibilita a los usuarios evaluar y desarrollar aplicaciones para la familia de DSP TI C64XX. El DSK también sirve como un diseño de referencia Hardware para los DSPs TMS320C6416T. Los modelos esquemáticos, las ecuaciones lógicas y las notas de aplicación están disponibles para facilitar el desarrollo HW.

El DSK viene con un completo conjunto de dispositivos en la tarjeta que sirven para una amplia variedad de ambientes de aplicación. Las características claves incluyen:

- Un DSP TMS320C6416T operando a 1GHz.
- Un códec estéreo AIC23.
- 16 MB de RAM síncrona (SDRAM).
- 512 KB de memoria Flash no volátil.
- 4 leds y conmutadores DIP accesibles al usuario.
- Configuración de la tarjeta por software a través de los registros implementados en CPLD.
- Opciones de arranque configuradas y selección de entrada de reloj.
- Conectores de expansión estándar para uso de una tarjeta auxiliar.
- Emulación JTAG a través de un emulador JTAG en tarjeta con interfaz host USB o emulador externo.
- Fuente única de suministro de voltaje (5V).

D. 2. FUENTE DE ALIMENTACIÓN

El DSK opera a partir de una única fuente de alimentación externa de +5V conectada a la entrada de potencia principal (J5). Internamente, la entrada de +5V es convertida en +1,2V y +3.3V usando un regulador de voltaje dual. La fuente de +1.2V es usada por el núcleo del DSP mientras que la fuente de +3.3V es usada por los búfferes de entrada/salida del DSP y todos los otros circuitos integrados de la tarjeta. El conector de potencia es un plug tipo barrel de 2.5mm.

Hay 3 puntos de prueba de potencia en el DSK en JP1, JP2 y JP4. toda la corriente de las entradas/salidas del 6416T pasa a través de JP2 mientras toda la corriente del núcleo pasa a través de JP1. Toda la corriente del sistema para a través de JP4, normalmente estos puentes están cerrados. Para medir esta corriente que pasa a través remueva estos puentes y conecte los pines con un dispositivo de medición de corriente tal como un multímetro o un probador de corriente.

D. 3. FUNCIONAMIENTO

El DSP sobre el DSK 6416T interactúa con los periféricos en la tarjeta a través de 2 buses, la EMIFA de 64 bits de ancho y la EMIFB de 8 bits de ancho. La SDRAM, flash y CPLD están cada uno conectado a estos dos buses. La EMIFA es también conectada a los conectores de tarjeta de expansión auxiliar los cuales se usan para anexar tarjetas en aplicaciones con trabajo compartido.

El códec AIC en la tarjeta permite que el DSP transmita y reciba señales analógicas. Se usa McBSP1 para interfaz de control del códec y McBSP2 para los datos. La entrada/salida analógica es hecha a través de cuatro jacks de audio de 3.5mm que corresponden a la entrada de micrófono, entrada de línea, salida de línea, salida de audífonos. La salida analógica es conducida tanto por los conectores de salida de línea (ganancia fija) y audífonos (ganancia ajustable). Se puede reenrutar McBSP1 y McBSP2 a conectores de expansión mediante software.

Un dispositivo lógico programable llamado un CPLD es usado para implementar la lógica agregada que enlaza los componentes de la tarjeta juntos. El CPLD también tiene una

interfaz de usuario basada en registro que le permite al usuario configurar la tarjeta al leer y escribir en los registros del CPLD.

El DSK incluye cuatro leds y cuatro conmutadores DIP de posición como una manera simple de proveer al usuario con una realimentación interactiva. Ambos son accesados al leer y escribir en los registros del CPLD.

Se usa una fuente de potencia externa de 5V incluida para energizar la tarjeta. Los reguladores de voltaje conmutado en tarjeta proveen el voltaje de núcleo del DSP de 1.2V y el voltaje de alimentación de entrada/salida de 3.3V. la tarjeta se mantiene en reset hasta que estas fuentes están entre las especificaciones de operación. Un regulador separado energiza las líneas de 3.3V en la interfaz de expansión.

Code Composer se comunica con el DSK a través de un emulador JTAG embebido con una interfaz host USB. El DSK también puede ser usado con un emulador externo a través del conector JTAG externo.

D.4. OPERACIÓN BÁSICA

El DSK está diseñado para trabajar con Code Composer Studio (CCS) de TI en ambientes de desarrollo y se envía con una versión específicamente “diseñada” para trabajar con la tarjeta. Code Composer se comunica con la tarjeta a través del emulador JTAG sobre la tarjeta. Para iniciar, siga las instrucciones de la Guía de Inicio Rápido para instalar Code Composer. Este proceso instalará todas las herramientas de desarrollo, documentación y controladores necesarios.

Después de completar la instalación, siga estos pasos para ejecutar el Code Composer. El DSK debe estar completamente conectado para lanzar la versión DSK del Code Composer.

1) Conecte la fuente de alimentación incluida al DSK.
2) Conecte el DSK a su PC con un cable USB estándar (también incluido).

3) Abrir el Code Composer desde el ícono en su escritorio,

La información detallada acerca del DSK incluyendo un tutorial, ejemplos y material de referencia está disponible en el archivo de ayuda del DSK. Puede acceder al archivo de ayuda a través del menú de ayuda del Code Composer. También puede abrirlo directamente dando doble click sobre el archivo c6416Tdsk.hlp en el subdirectorio docs\hlp del Code Composer.

El DSK provee +3.3V hasta 1A para la tarjeta auxiliar. La fuente de +3.3V se deriva de la fuente de potencia de +5V vía el regulador principal de +3.3V. También es posible proveer a la tarjeta auxiliar con +12V y -12V cuando se usa el conector de potencia externa (J6).

D.5. CONFIGURACIÓN Y MAPA DE MEMORIA

La familia C64XX de DSPs tiene un “espacio de direcciones direccionable con un byte de extensión. El código y los datos de programa pueden ser localizados en cualquier parte del espacio de direcciones unificado. Las direcciones son siempre de 32 bits de ancho.

El mapa de memoria (Tabla 1) muestra en la izquierda el espacio de direcciones de un procesador genérico 6416T con detalles específicos de cómo se usa cada región en la derecha. Por defecto, la memoria interna se sitúa en el comienzo del espacio de direcciones. Las porciones de memoria pueden ser reasignadas en software como caché L2 en lugar de RAM fija.

Cada EMIF (Interfaz de Memoria Externa) tiene cuatro regiones direccionables separadas llamadas espacio de habilitación de chip (CE0 – CE3), la SDRAM ocupa el CE0 de la EMIFA mientras que el CPLD y la Flash son mapeadas en el CE0 y CE1 de la EMIFB respectivamente. Las tarjetas auxiliares usan el CE2 y CE3 de la EMIFA.

Tabla 1. Mapa de memoria

Dirección	Espacio Genérico C6416	DSK 6416
0x00000000	Memoria Interna	Memoria Interna
0x00100000	Espacio Reservado o Registros de Periféricos	Reservado o Periféricos
0x60000000	EMIFB CE0	CPLD
0x64000000	EMIFB CE1	Flash
0x68000000	EMIFB CE2	
0x6C000000	EMIFB CE3	
0x80000000	EMIFA CE0	SDRAM
0x90000000	EMIFA CE1	
0xA0000000	EMIFA CE2	
0xB0000000	EMIFA CE3	Tarjeta Auxiliar

D.5.1. Memoria de datos.

La memoria de datos en la tarjeta DSK6416 está formada por la memoria interna del DSP en los niveles L1 y L2 y las memorias externas flash ROM y SDRAM comunicadas por buses EMIF.

La memoria L2 de 1MB se puede utilizar como caché de datos y programa de segundo nivel o como RAM de usuario. Esta memoria es la que utiliza el compilador de C por defecto.

La memoria flash ROM de 512 KB es de propósito general y puede ser utilizada como memoria persistente de datos o como memoria de programa, en cuyo caso es necesario configurar el DSP para que inicie la aplicación leyendo las instrucciones de esta memoria.

La memoria SDRAM es una memoria de propósito general configurada para comunicación a 100 MHz. Tiene una longitud de 16 MB y se usa como memoria auxiliar a la memoria interna mediante la interfaz EMIFA.

D.5.2 Memoria de Programa.

El DSP usa como memoria de programa o caché de programa la porción L1P de 16KB de extensión. Cuando se requiere programas de mayor tamaño se puede almacenarlos en una memoria externa, de preferencia en la Flash y leerlos hacia la memoria L1P para ser ejecutados. En este caso es necesario configurar el DSP para que las operaciones de carga apunten hacia esta memoria.

D.6. ELABORACIÓN DE UN MAPA DE MEMORIA.

Para disponer de los recursos del DSP se requiere de un mapa de memoria donde se indique en que localidades de memoria y de qué manera se almacenan los diferentes tipos de datos, ya sea manejados por programas ensamblador o generados por el compilador.

El mapa de memoria le dice al depurador (en inglés, debuger) a cuales áreas de memoria puede y no puede tener acceso. El mapa de memoria varía dependiendo de la aplicación. Típicamente, el mapa de memoria alcanza la definición *MEMORY* en el archivo de comandos del enlazador.

Cuando se invoca CCS por primera vez, el mapa de memoria está desactivado. Se puede tener acceso a cualquier localidad de memoria; el mapa no interfiere. Cuando el mapeo de memoria está habilitado, el depurador confronta cada uno de los accesos a memoria contra el mapa establecido. Si se trata de acceder a un área indefinida o protegida, el depurador prevendrá que el acceso ocurra y desplegará el valor como una serie de líneas (-).

Nota: El simulador usa configuraciones de mapa de memoria fuertemente codificados para proveer una representación genérica de la familia del procesador simulada por el software.

D.6.1 Secciones

La unidad más pequeña de un archivo objeto es llamada una *sección.* Una sección es un bloque de código o datos que finalmente ocupa un espacio continuo en el mapa de memoria. Cada sección está separada y es distinta. Los archivos objeto COFF siempre tienen tres secciones por defecto:

- Sección .text: Usualmente contiene el código ejecutable.
- Sección .data: Usualmente contiene los datos inicializados.
- Sección .bss: Usualmente reserva espacio para variables no inicializadas.

Adicionalmente, el ensamblador y el enlazador (En inglés, linker) le permiten a usted crear, nombrar y vincular secciones nombradas que son usadas como secciones .data, .text y .bss.

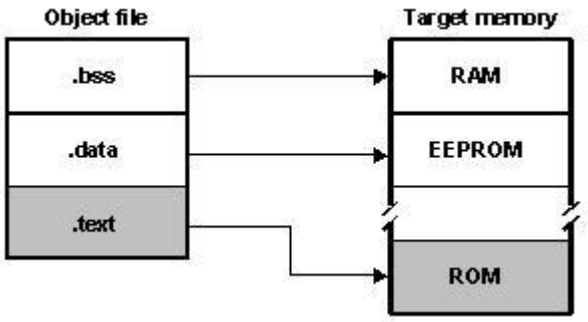
Hay dos tipos básicos de sección:

- Secciones inicializadas: Contienen datos o código. Las secciones .data y .text son inicializadas; las secciones nombradas creadas con la directiva ensamblador .sect también son inicializadas.

- Secciones no inicializadas: Reservan espacio en el mapa de memoria para datos no inicializados. La sección .bss es no inicializada; las secciones nombradas creadas con la directiva ensamblador .usect también son no inicializadas.

Diversas directivas ensamblador permiten asociar varias porciones de código y datos con las secciones apropiadas. El ensamblador construye estas secciones durante el proceso de ensamble, creando un objeto archivo organizado como se muestra en la ilustración 1.

Ilustración 1. Particion de memoria en bloques lógicos



Una de las funciones del enlazador es restablecer en el mapa de memoria del sistema de la tarjeta; esta función es llamada *asignación* (En inglés, allocation). Debido a que la mayoría de sistemas contienen diferentes tipos de memoria, puede ayudarle a usar la memoria de la tarjeta más eficientemente. Todas las secciones son reasignables independientemente; se puede colocar cualquier sección en cualquier bloque asignado de la memoria de tarjeta. Por ejemplo, se puede definir una sección que contiene una rutina de inicialización y luego asignar la rutina en una porción del mapa de memoria que contiene la ROM.

D.6.2. Especificar dónde asignar secciones en la memoria.

El compilador produce bloques reasignables de código y datos. Estos bloques, llamados secciones, son asignados en memoria en una variedad de maneras de conformidad a una variedad de configuraciones de sistemas.

El compilador crea dos tipos de secciones básicas: inicializadas y no inicializadas. La tabla 2 resume las secciones:

Cuando se enlaza un programa se debe especificar la asignación de las secciones en la memoria. En general, las secciones inicializadas son vinculadas en la ROM o RAM; las secciones no inicializadas son vinculadas en la RAM. Con excepción de .text, las

secciones inicializadas y no inicializadas creadas por el compilador no pueden ser asignadas en la memoria interna.

Tabla 2. Secciones

Nombre	Contiene
a) Secciones inicializadas	
.cinit	Tablas para variables globales y estáticas inicializadas explícitamente
.const	Variables constantes estáticas y globales que están explícitamente inicializadas y que son literales de cadena.
.pinit	Tabla de constructores para ser llamada al inicio
.switch	Tabla para implementación de sentencias tipo *swtich*
.text	Código ejecutable y constantes
b) Secciones no inicializadas	
.bss	Variables estáticas y globales
.stack	Pila del sistema
.sysmem	Memoria para funciones *malloc*

El enlazador provee las directivas *MEMORY* y *SECTIONS* para asignación de secciones.

El enlazador donde son asignadas las secciones de salida en la memoria, debe tener un modelo de memoria de tarjeta que cumpla con esto. La directiva *MEMORY* le permite especificar un modelo de memoria de tarjeta tal que pueda definir los tipos de memoria que su sistema contiene y los rangos de direcciones que ocupa. El enlazador mantiene el modelo tanto como las secciones de salida que asigna y usa este modelo para determinar cuales localidades de memoria pueden ser usados por el código objeto.

Las configuraciones de memoria para los sistemas TMS320C6000 difieren de una aplicación a otra. La directiva *MEMORY* le permite especificar una variedad de configuraciones. Después de usar *MEMORY* para definir un modelo de memoria, puede usar la directiva *SECTIONS* para asignar las secciones de salida en la memoria definida.

La directiva *SECTIONS* describe como se combinan las secciones de entrada en las secciones de salida, define las secciones de salida en el programa ejecutable, especifica donde son puestas las secciones de salida en memoria (en relación a cada una de las otras y al espacio entero de memoria) y permite renombrar las secciones de salida.

Las directivas *MEMORY* y *SECTIONS* son especificadas en el archivo de comandos (.cmd) por las palabras *MEMORY* y *SECTIONS* (en mayúsculas), respectivamente. La palabra *SECTIONS* va seguida de una lista de especificaciones de secciones de salida encerradas entre llaves. Cada especificación de sección, iniciando por el nombre, define una sección de salida, es decir, una sección en el archivo de salida (.out). Después del nombre, se coloca una secuencia de propiedades como el formato de la memoria, el tipo de sección y sobretodo, la asignación de memoria.

Un ejemplo de esto puede verse en las siguientes sentencias de configuración:

```
MEMORY
{
   RAM        :      origin = 0x00000001,        length = 0xFFFFFFFE
}

SECTIONS
{
   .text      :      ALIGN(32)   {} > RAM
   .const     :      ALIGN(8)    {} > RAM
   .data      :      ALIGN(8)    {} > RAM
   .bss       :      ALIGN(8)    {} > RAM
   .cinit     :      ALIGN(4)    {} > RAM     ; cflag option only
   .pinit     :      ALIGN(4)    {} > RAM     ; cflag option only
   .stack     :      ALIGN(8)    {} > RAM     ; cflag option only
   .far       :      ALIGN(8)    {} > RAM     ; cflag option only
   .sysmem    :      ALIGN(8)    {} > RAM     ; cflag option only
```

```
   .switch     :     ALIGN(4)     {} > RAM     ; cflag option only
   .cio        :     ALIGN(4)     {} > RAM     ; cflag option only
}
```

D.6.3. Agregar un nuevo rango de mapa de memoria.

1. Seleccione de la barra de menús: Option -> Memory map…
   (Insertar imagen)
2. Si su configuración de memoria de tarjeta actual o simulada soporta múltiples páginas, el cuadro de texto Memory Map contiene una tabla separada para cada tipo de página de memoria (por ejemplo: Programa, Datos, Entrada/Salida). Seleccione la tabla apropiada para el tipo de memoria a modificar. Las tablas separadas no aparecen para los procesadores que solo tienen una página de mapa de memoria.

3. Pulse el botón Done para aceptar la selección.

D.6.4. Memoria de asignación HEAP

La asignación dinámica de memoria no es parte del lenguaje C/C++ estándar. La librería de soporte en tiempo de ejecución suministrada con el compilador C6000 que contiene diversas funciones (tales como malloc, calloc y realloc) que permite asignar memoria dinámicamente para variables en tiempo de ejecución.

La memoria es asignada de un banco global (heap) que es definido en la sección .sysmem. El compilador C/C++ usa una sección no inicializada llamada .sysmem para el banco de memoria de C usado por *malloc*(). Se puede establecer el tamaño el tamaño de este banco de memoria al tiempo de enlazar al usar la opción *–heap* en el archivo de comandos. La sintaxis para esta opción es

-heap tamaño.

Donde tamaño es una constante. El enlazador crea la sección .*sysmem* solamente si hay una sección .sysmem en el archivo de entrada. También crea un símbolo global __SYMEM_SIZE y le asigna un valor igual al tamaño de heap. El tamaño por defecto es un 1K palabras.

Los objetos asignados dinámicamente no son direccionados directamente (a ellos siempre se accede mediante punteros) y el banco de memoria es uns sección separada (.sysmem); sin embargo, el banco de memoria dinámica puede tener un tamaño limitado solamente por la cantidad de memoria en el sistema. Para conservar el espacio en la sección .bss se puede asignar grandes arreglos desde el heap en vez de definirlos como global o estático. Por ejemplo, en lugar de una definición tal como:

struct big table[100];

use un puntero y llame a la función malloc:

```
struct big *table;
table = (struct big *)malloc(100*sizeof(struct big));
```

D.6.5. Accediendo a memoria no existente

Cuando el depurador compara los accesos con el mapa de memoria, desarrolla este chequeo en software, no en hardware. El depurador no puede prevenir que el programa intente tener acceso a memoria no existente.

Hay dos maneras de definir un rango de mapa de memoria válido en la tarjeta:

- Definir el mapa interactivamente mientras se usa el depurador. Esto puede ser inconveniente a causa que, se configurará un mapa de memoria antes que inicie la depuración y entonces se usará un mapa para todas las sesiones de depuración.

- Definir el mapa de memoria usando las funciones embebidas GEL (Lenguaje de Extensión General). GEL provee un completo conjunto de funciones de mapeo de

memoria. El método más fácil de implementar un mapeo de memoria es colocar las funciones de mapeo de memoria en un archivo de texto GEL y ejecutar el archivo al inicio