Cika Electrónica

Tutorial: CTU-001
Título: **Dynamic C**
Autor: Rabbit Semiconductor, trad. y adaptación: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	19/8/03	
1	25/11/03	Agrega funciones de I/O y documentación

El presente es un tutorial sobre Dynamic C, el entorno integrado de programación y debugging de Z-World para los microprocesadores Rabbit. Las áreas cubiertas abarcan desde cómo portar aplicaciones a Dynamic C, pasando por las diferencias entre éste y ANSI C, hasta una breve descripción de su estructura interna.

Índice de contenido

Introducción

Dynamic C integra las siguientes operaciones de desarrollo:

- Edición
- Compilación
- Linkeado
- Cargado en el controlador
- Depuración

Dynamic C difiere de la programación tradicional en C en sistemas UNIX o PCs, ya que no es posible usar C standard en un entorno dedicado sin hacer adaptaciones. Como es sabido, C standard hace muchas asunciones que no son aplicables a sistemas dedicados; por ejemplo, implícitamente se asume que un sistema operativo se halla presente y que el programa se inicia con un entorno limpio, cuando es sabido que los sistemas dedicados deben enfrentar el ser encendidos y apagados. Dynamic C es la extensión de Z-World del lenguaje C en aquellas áreas necesarias para adecuarlo a las necesidades de los sistemas dedicados.

Cómo portar aplicaciones a Dynamic C

Dynamic C tiene una gran cantidad de características sobresalientes y diferencias, si se lo compara con otros compiladores C. Generalmente, otros compiladores requieren un gran número de archivos de inclusión (include files), potencialmente diferentes para cada fuente, para proveer definiciones y prototipos de funciones (function prototypes). En Dynamic C, las bibliotecas de funciones (source libraries) se ocupan de manejar esto de forma automática; la mayor parte de la tarea de portar código consiste en reorganizar los fuentes y prototipos de funciones en bibliotecas de Dynamic C.

Existen, además, algunas diferencias en las reglas empleadas por Dynamic C con respecto a otros compiladores para sistemas dedicados. Estas resultan en su mayoría del hecho de que Dynamic C compila la totalidad del código del archivo fuente y las bibliotecas (source libraries), en vez de compilar varios módulos independientes y luego linkearlos en un paso aparte.

Cuando es posible y deseable, Dynamic C sigue los lineamientos del standard ISO/ANSI. Como éste no toma en consideración las necesidades especiales de los sistemas dedicados, es necesario, y a veces deseable, alejarse del mismo en algunas áreas, como las relacionadas con memorias de solo lectura o inclusión de código assembler. Por esta razón, los compiladores orientados a sistemas dedicados no suelen cumplir completamente con el standard, sino que en realidad lo utilizan como una guía.

A modo de somera aproximación, se requiere de un día de trabajo para portar unas mil líneas de código, sin considerar diferencias de hardware de I/O, drivers y código de startup.

Funciones para operaciones de I/O

Si bien las funciones de I/O no son standard, es común, en el ambiente PC, el utilizar funciones tales como *inportb()* y *outportb()* para realizar operaciones de entrada-salida. Dynamic C incluye funciones específicas para acceder a ports tanto en el espacio de I/O interno (I, periféricos en chip) o externo (E):

- ➔ *BitRdPortE(), BitRdPortI()* : Lee el estado de un bit en un port
- ➔ *BitWrPortE(), BitWrPortI():* Escribe el estado de un bit en un port
- ➔ *RdPortE(), RdPortI()*: Lee un port
- ➔ *WrPortE(), WrPortI()*: Escribe un port

Mejoras

Estas son algunas de las mejoras por sobre el standard ANSI C que encontramos en Dynamic C:

- Encadenado de funciones (Function chaining), un concepto único de Dynamic C. Permite que segmentos especiales de código puedan estar embebidos en una o más funciones. Cuando se ejecuta una cadena, se ejecutan todos los segmentos que la componen.
- Co-sentencias (costatements). Implementan construcciones del tipo usado en máquinas de estados, permitiendo simular procesos paralelos concurrentes en un mismo programa.
- Co-funciones (cofunctions), permiten simular procesos cooperativos en un mismo programa.

- Sentencias de partición (slice statements), permiten procesos de tipo preemptive en un mismo programa.
- Soporte de código assembler.
- Palabras clave (keywords) para identificar datos compartidos por contextos diferentes o almacenados en memoria no volátil: *shared* y *protected.*

Diferencias

- Si una variable es inicializada explícitamente en su declaración (ej.: *int x=0;*), será guardada en memoria Flash y no podrá ser alterada más tarde por una sentencia de asignación. A partir de DC7.0, está situación generará una advertencia (warning) que podrá eliminarse declarándola como constante: ***const*** *int x=0;*
- Para inicializar variables de tipo estático (static vars) en SRAM, se emplean las secciones *#GLOBAL_INIT*. Nótese que otros compiladores C ponen atomáticamente a cero todas las variables de tipo estático que no hayan sido explícitamente inicializadas al llamarse a la parte principal del código (main). En Dynamic C, pueden agregarse segmentos a la function chain *GLOBAL_INIT*, la misma se ejecutará automáticamente al inicio del programa. **Dynamic C no pone a cero las variables de tipo estático que no hayan sido explícitamente inicializadas**, esto es así debido a que en un sistema dedicado es deseable preservar datos en RAM con pila de respaldo (battery back-up), aún después de un reset.
- La clase por defecto es *auto*. En versiones anteriores era *static*. Puede cambiarse mediante la directiva *#class.*
- Dynamic C posee un sistema de bibliotecas (libraries) que provee automaticamente la información que comunmente se halla en prototypes y headers, por esta razón, no utiliza include files. Esto es importante para aquellos usuarios que escriben sus propias bibliotecas de funciones; la sección sobre Módulos en el Manual del Usuario de Dynamic C contiene información importante al respecto. Es importante notar que la directiva *#use* es reemplazo de la directiva *#include*, la cual no está soportada.
- Al declarar punteros de funciones (function pointers), no deben usarse argumentos en la declaración. Los mismos pueden ser usados al llamar funciones indirectamente por medio del puntero, pero el compilador no revisará los argumentos para ver si corresponden con la definición de la función.
- No existe soporte para bit fields.
- Existen, además, diferencias menores en el empleo de las keywords *extern* y *register*.
- Dynamic C incluye bibliotecas de funciones totalmente en código fuente. Estas soportan programación en tiempo real e I/O a nivel código de máquina; proveyendo funciones standard de manejo de cadenas de caracteres (strings) y funciones matemáticas.
- Dynamic C compila directamente a la memoria del sistema objeto.

Variables protegidas (protected vars)

Una característica importante de Dynamic C es la habilidad de declarar variables como *protected*. Una variable así declarada se encuentra protegida en caso de falla de alimentación o reset, debido a que el compilador genera código que crea una copia de seguridad de la variable antes de modificarla. Si el sistema llegara a resetearse al momento de modificar dicha variable, el valor original puede recuperarse al reiniciar el sistema. Esta función especial requiere de la existencia de RAM con pila de respaldo (battery-backed RAM).

Variables compartidas (shared vars)

Si un sistema comparte datos entre diversas tareas o entre rutinas de interrupción, es probable que una interrupción en medio de la escritura de una variable multi-byte (un entero, por ejemplo) ocasione la destrucción de los datos. Para evitar ésto, debe declararse la variable como tipo *shared*. Todos los cambios a una variable shared son atómicos, es decir, se inhiben las interrupciones mientras se la modifica.

Tipos de variables

integer	2 bytes
short integer	2 bytes
long integer	4 bytes
unsigned char	1 bytes

unsigned int	2 bytes
unsigned short int	2 bytes
unsigned long	4 bytes
single- precision floating point	4 bytes

Restricciones

Macros

Es posible definir macros mediante la directiva *#define*. Dynamic C tiene un preprocesador que expande las macros antes de proceder a la compilación. La longitud del nombre no debe exceder los 31 caracteres.

Variables globales compartidas (shared global vars)

Las variables *SEC_TIMER*, *MS_TIMER* y *TICK_TIMER* son de tipo *shared*, lo que hace que sus modificaciones se manejen de forma atómica. Son actualizadas por interrupciones periódicas y no deben ser modificadas por programas de usuario, ya que algunas co-sentencias y co-funciones dependen de ellas.

Sentencias (statements)

A excepción de los comentarios, todo en un programa C se define como statement. Casi todos deben terminar con un punto y coma. Un programa C es tratado como un stream de caracteres donde los renglones (generalmente) no son tenidos en cuenta. Cualquier sentencia puede escribirse ocupando la cantidad de renglones que sean necesarios. El editor limita la longitud de una línea a 512 bytes, incluyendo expansión de macros.

Procesamiento multitarea con Dynamic C

Una tarea es una lista ordenada de operaciones a realizar. En un entorno multitarea, más de una tarea (cada una representando una secuencia de operaciones), pueden aparentar ejecutarse en paralelo. En realidad, un solo procesador puede ejecutar sólo una instrucción a la vez; si una aplicación debe realizar múltiples tareas, el software de multitarea usualmente saca provecho de las demoras naturales en cada tarea para mejorar la performance global del sistema. Cada tarea realiza parte de su trabajo mientras las otras están esperando que ocurra algún evento. De esta forma, las tareas se ejecutan casi en paralelo.

Multitarea cooperativo es una forma de realizar varias tareas diferentes, virtualmente a la vez. Un ejemplo sería realizar una secuencia de operaciones en una máquina y dialogar a la vez con el operador mediante un teclado. Cada tarea por separado cede voluntariamente su tiempo de procesador cuando no necesita realizar ninguna actividad inmediata. En procesamiento multitarea de tipo preemptive (preemptive multitasking), por el contrario, un ejecutor (task scheduler) asigna y remueve el control a la fuerza, repartiendo el tiempo de procesamiento por prioridades y/o porciones de tiempo fijo entre las diversas tareas.

Dynamic C incluye extensiones para soportar ambos tipos de procesamiento multitarea, para el caso de multitarea cooperativo, las extensiones son *costate* y *cofunction*; para tipo preemptive, la extensión es *slice*, aunque también puede emplearse el real-time kernel µC/OS-II, que se vende como una biblioteca de funciones extra.

Supongamos que un programador debe resolver un problema de programación en tiempo real que involucra diversas tareas con diferentes escalas temporales. Ante la ausencia de un kernel multitarea de tipo preemptive, la solución generalmente será una maquina de estados con un GRAN LAZO principal.Esto significa que el programa consiste de un gran loop infinito, dentro del cual cada tarea es un fragmento de programa que avanza a través de una serie de estados; cada estado se codifica generalmente como valores numéricos en alguna variable. Con Dynamic C, cada tarea puede asociarse, por ejemplo a un *costate*.

Co-sentencias (costates)

Son las extensiones de Dynamic C que simplifican la implementación de máquinas de estados (state machines). Son cooperativos ya que su ejecución debe suspenderse y reanudarse voluntariamente. El cuerpo de

un costate es una tarea, una lista ordenada de operaciones a realizar. Cada *costate* tiene su propio puntero de programa (statement pointer) para determinar qué item en la lista debe ejecutarse cuando se le da oportunidad. Como parte del proceso de inicialización, el puntero se asigna a la primera instrucción.

Co-funciones (cofunctions)

Las *cofunction*s son similares a los *costate*s, pero su formato es similar a las funciones; es decir, se les puede pasar argumentos y devuelven un resultado.

Sentencias de control (control statements)

Se utilizan para distribuír el procesamiento entre las diversas tareas, detectando los estados de espera.

waitfor (expresión);

La palabra clave *waitfor* indica una sentencia especial, y no una llamada a una función:
La expresión entre paréntesis se computa cada vez que se ejecuta el *waitfor*. Cualquier función C válida que retorne algún valor puede ser utilizada como parámetro de espera.

waitfordone {tarea};

Algunas veces una tarea tiene principio y fin, por ejemplo una *cofunction* para transmitir un string de caracteres por el port serie comienza cuando se la llama por primera vez y continúa a través de sucesivas llamadas durante el loop de programa. El fin ocurre luego de que se ha enviado el último caracter y se satisface la condición *waitfordone*. Este tipo de llamada a *cofunction* se vería así:
waitfordone *{ SendSerial ("string of characters");}*

Multitarea tipo preemptive: slice

Basado en el lenguaje de *costate*, *slice* permite que el programador asigne la ejecución de un bloque de código durante un tiempo específico. La sentencia *slice* puede activarse en cualquier momento, eliminando la posibilidad de anidado y su uso en funciones que puedan ser directa o indirectamente llamadas dentro de una sentencia *slice*. Las únicas formas soportadas de salir de un *slice* son: ejecutar hasta la última sentencia dentro del *slice*, o ejecutar una sentencia *abort*, *yield* o *waitfor*. Las sentencias *return*, *continue*, *break* y *goto* no están soportadas. Tampoco pueden utilizarse sentencias *slice* si se utiliza µC/OS-II o TCP/IP.
Debido a que una sentencia *slice* tiene su propio stack, las variables locales de tipo auto y los parámetros de llamada no pueden accederse dentro del contexto del mismo. Cualquier función llamada desde allí dentro opera normalmente.

Resumen

Aunque el procesamiento multitarea puede disminuir ligeramente la performance del procesador, es un concepto muy importante. Un controlador normalmente está conectado a más de un dispositivo externo, un esquema multitarea permite escribir el programa de control sin tener que pensar en todos los dispositivos a la vez. En otras palabras, multitarea es una forma más fácil de pensar el sistema.

Bibliotecas de funciones (libraries)

Dynamic C incluye muchas bibliotecas de funciones, es decir, archivos en código fuente con funciones útiles. Se localizan en el subdirectorio LIB, dentro de la instalación de Dynamic C, y su extensión por defecto es .LIB
Dynamic C utiliza las funciones y datos de estas bibliotecas para compilarlas junto con el programa de aplicación, que podrá ser enviado al controlador o guardado en un archivo (extensión .BIN).
Un programa de aplicación (extensión por defecto .C) consiste de un archivo fuente que contiene una función llamada *main* y generalmente otras funciones definidas por el usuario. Cualquier otro archivo adicional es considerado una biblioteca de funciones (aún cuando su extensión sea .C) y tratado como tal.
Las bibliotecas de funciones, ya sean definidas por el usuario o por Z-World, se vinculan con la aplicación mediante el uso de la directiva *#use*. La misma identifica a un archivo del cual pueden extraerse funciones y datos, siendo posible la inclusión anidada. Recordemos que la directiva *#use* es reemplazo de la directiva *#include*, la cual no está soportada.

Cualquier biblioteca a ser incluída por un programa en Dynamic C, debe estar listada en el archivo LIB.DIR. Sin embargo, el usuario puede especificar un archivo diferente en las opciones del compilador, de modo de facilitar el trabajo con múltiples proyectos.

Encabezados (headers)

Dynamic C utiliza dos tipos de encabezados:

- Encabezados de Módulo (Module Headers): cumplen la función de hacer que Dynamic C reconozca las funciones y variables globales presentes en ese módulo, que podrán ser utilizadas. Una biblioteca típica contiene varios módulos, los mismos organizan el contenido de la biblioteca de modo de facilitar la reducción del código compilado en la aplicación que usa esa biblioteca de funciones. Para crear sus propias bibliotecas, el usuario debe escribir los módulos siguiendo los lineamientos descriptos en el manual del usuario de Dynamic C
- Encabezados de Función (Function Description Headers): describen funciones, son la base para el sistema de ayuda de búsqueda e inserción de funciones (function lookup help). Cada función de una biblioteca que pueda ser llamada por el usuario, debe ser precedida por un encabezado que la describa. Esta información se utiliza para proveer ayuda on-line. El encabezado es un comentario con un formato específico, como muestra este ejemplo:

```
/* START FUNCTION DESCRIPTION ********************
WrIOport                                          <IO.lib>
SYNTAX: void WrIOport (int portaddr, int value);
DESCRIPTION:
Writes data to the specified I/O port.
PARAMETER1: portaddr – register address of the port.
PARAMETER2: value – data to be written to the port.

RETURN VALUE: None
KEY WORDS: parallel port

SEE ALSO:  RdIOport
END DESCRIPTION **********************************/
```

De este modo, las funciones creadas por el usuario también podrán verse al utilizar la facilidad de búsqueda e inserción de funciones, que se genera automáticamente al iniciar Dynamic C.

Archivos de soporte (support files)

Los siguientes son algunos de los principales archivos de soporte que regulan el modo como se compila una aplicación:

DCW.CFG: contiene datos de configuración del controlador utilizado, es decir, el hardware.

DC.HH: contiene prototipos, definiciones de tipos básicos, *#define*s y modos por defecto.

LIB.DIR: contiene los paths de todas las bibliotecas que deban ser conocidas. El programador puede agregar o remover bibliotecas de esta lista si así lo desea. Al momento de la instalación, este archivo contiene todas las bibliotecas existentes en el disco de distribución. Cualquier biblioteca que deba ser utilizada en algún programa debe listarse en este archivo, o en algún otro específicamente indicado por el usuario en las opciones.

FACTORY.DCP, DEFAULT.DCP: estos archivos contienen el entorno de compilación por defecto,es decir, el que se instala por primera vez.

Driver virtual (virtual driver)

Dynamic C denomina Virtual Driver a los servicios de inicialización y un grupo de servicios que presta una interrupción periódica.

Servicios de inicialización

- Llamar a *GLOBAL_INIT()*.
- Inicializar las variables globales de temporización.
- Arrancar la interrupción periódica.

Servicios de la interrupción periódica:
- ✗Decrementar los watchdog timers virtuales (software).
- ✗Resetear el watchdog timer (hardware).
- ✗Incrementar las variables globales de temporización.
- ✗Manejar el procesamiento multitarea tipo preemptive de µC/OS-II.
- ✗Manejar las sentencias slice (multitarea tipo preemptive).

Al iniciar cualquier programa, por defecto, y sin intervención del usuario, Dynamic C comienza ejecutando el Virtual Driver, antes de llamar a *main()*.

El kernel (BIOS) ejecuta primero una función llamada *premain()*, que a su vez ejecuta *VDInit()*, que realiza los servicios de inicialización, incluyendo el arranque de la interrupción periódica. Luego, finalmente, llama a *main()*.

Si el usuario inhabilitara el Virtual Driver eliminando la llamada a *VDInit()* en *premain()*, ninguno de los servicios que presta la interrupción periódica estarían disponibles, y el sistema se resetearía a menos que se desactive el watchdog timer. De no ser incompatible con algún tipo de requerimientos muy exigentes de timing y realmente no sean necesarios sus servicios, se recomienda no inhabilitar el Virtual Driver.

Watchdog timers

La CPU Rabbit tiene un hardware watchdog. El Virtual Driver lo resetea periódicamente y provee 10 watchdog timers virtuales por software.

Documentación

Dynamic C viene acompañado de la siguiente documentación en CD:
- ✔**Manual del Usuario**: Descripción y utilización de Dynamic C, incluyendo uso de la multitarea
- ✔**Manual de Referencia**: Descripción de cada una de las funciones de Dynamic C, agrupadas de forma alfabética o por grupo funcional
- ✔**Ejemplos de uso**: Agrupados en directorios por aplicación o por familia de hardware, encontrará aquí ejemplos de cómo utilizar las bibliotecas de funciones y cómo resolver algunos problemas sencillos
- ✔**Manual del usuario de TCP/IP**: Inicialización, introducción y referencia de las funciones que componen el conjunto de bibliotecas de funciones de TCP/IP
- ✔**Introducción a TCP/IP**: Introducción a redes, stacks de protocolos y TCP/IP, incluye su implementación en Dynamic C.