UNIVERSITY OF MICROCHIP
U OF M
MASTERs 2007

11028 CCS

A Comprehension of CCS C Compiler Advanced Techniques

Class Objectives

- **When you finish this class you will:**
  - Learn about the newest features added to the CCS C Compiler
  - Be able to use compiler output to diagnose problems
  - Know how to force ROM/RAM placement.
  - Get an overview of some of CCS's power features

Agenda

- **New Features**
- **Output Files and Messages**
- **Memory Control**
  - RAM
  - ROM
  - Bootloader Example
- **addressmod**
- **RTOS**

New Features

SPI

- **Configurable SPI library**
- **#use spi(parameters)**
  - HW or SW pins
  - Multiple streams
  - Clock rate and clock mode configurable
- **in = spi_xfer(STREAM_SPI, out)**

Bit Arrays

- **Array of bits:**
  - int1 flags[30]={FALSE};
  - flags[i] = TRUE;
  - if (flags[10]) { /* some code */ }
- **Bits are packed**
- **Pointers not supported!**

Fixed Point Decimal

- **Represent decimal numbers with integers, instead of floats**
  - Faster, Smaller
  - 100% precision
- **Example Declaration:**

```
[type] _fixed(y) [declarator]
int16 _fixed(2) money;
```

- **Supported by printf()**

Fixed Point Decimal Examples

- `int16 _fixed(2) cash;`
  - Range: 0.00 to 655.35
- `cash = 20.50;`
- `cash += 5;      //adds 5.00`
- `cash += value;`
- `printf(“%w”, cash);`

String Parameters

- String can be function parameter
- `LCDPuts(“Hello”);`
- Example (RAM):

```
#device PASS_STRINGS=IN_RAM
void LCDPuts(char *str);
```

- Example (Const RAM):

```
#device PASS_STRINGS=IN_RAM
#device CONST=ROM
void LCDPuts(const char *str);
```

- Example (ROM):

```
void LCDPuts(rom char *str);
```

#import()

- **Import various files into project.**
- **#import(File=name, Type)**
- **Type = Relocatable (.o, .cof)**
  - ‘only’ and ‘except’ specifies what C symbols
- **Type = HEX (.hex)**
  - ‘range’ parameter specifies range
- **Type = RAW**
  - ‘location’ gets or sets location
  - ‘size’ parameter gets size

Examples / Libraries

- **Variable number of parameters (…)**
- **Borrowed from C++:**
  - Default parameters
  - Function Overloading
- **New examples / libraries:**
  - FAT
  - XTEA Cipher
  - Modbus

Output Messages and Output Files

Output Files

- HEX – Compiled Application
- COF – Debug Application
- ERR – Compile output messages
- LST – C to Assembly comparison
- SYM – Memory Map
- STA – Statistics
- TRE – Call Tree

SYM – Memory Map

- **First section is RAM memory map**

```
005-014     main.buffer
na      main.index
```

- **‘na’ indicates no RAM**
- **Following sections include:**
  - Other Input files
  - ROM memory map
  - PIC® MCU / Compiler Settings

STA Statistics

- Review ROM/RAM/Stack used
- Statistics for each function:

```
Page ROM  %  RAM  Functions:
---- --- --- ---  ----------
0     26  0    1  @delay_ms1
0    284  1    3  ee_reset
```

STA Statistics

- **Statistics for each segment:**

```
Segment        Used   Free
---------     -----   ----
00000-00006       4      4
00008-000B2     172      0
000B4-03FFE   15826    378
```

TRE Statistics

- **Review call tree**
  - Function Name - Segment/ROM – RAM

```
project
  main  0/84  Ram=0
   init  0/194  Ram=1
      RELAY_INIT  0/16  Ram=0
          RELAY1_OFF  (Inline)  Ram=0
          @delay_ms1  0/26  Ram=1
```

- **Segment will be ? if it won’t fit**

Out of ROM

- **Full output:**

  ```
  Out of ROM, A segment or the program is too
  large: XXXXXX
  Seg w-x, y left, need z
  Seg 0-3ff, 12C left, need 12F
  ```

- **Tips:**
  - Be aware of processor segment size

    ```
    Seg 0-3FF, 3FF left, need 412
    ```

  - Optimize code
  - Split large functions
  - Reduce stack space

How Can I Reduce Code Space?

- Use int1 or bit fields for flags
- Use fixed point decimal, not float
- Divide large functions
- Avoid ->, move structure to local
- Use access bank mode
  - #device *=8
  - read_bank(b,o), write_bank(b,o,v)

Out of RAM

- **Full error message:**

  `Not enough RAM for all variables`

- **Review SYM file**
- **Tips:**
  - Be aware of PIC® MCU bank size
  - Use bit flags
  - Remove unused variables

Memory Management

#locate

- **Force location of variable**
- **#locate ident=X**
  - Assigns the C variable ident to location X
  - X can be a literal, or variable identifier
  - If not specified, ident is treated as a byte
  - Compiler allocates X
  - Can be any structure or type (not const)

#locate examples

- Overlaying variable onto SFR:

```
#locate STATUS=5
```

- Repeat, but using get_env():

```
#locate STATUS=get_env(“SFR:STATUS”)
```

- Overlaying two variables:

```
char buffer[512];
struct { /*protocol */} header;
#locate header=buffer+2
```

#byte and #bit

- **#byte ident=X**
  - Same as #locate, but no allocation
- **#bit ident=X.b**
  - Declares boolean at address X, bit b
  - Examples:
    - **`#bit CARRY=STATUS.0`**
    - **`#bit CARRY=get_env(“BIT:C”)`**

#rom

- **Place raw data into program memory**
- **#rom address={data….data}**
- **Application Ideas:**
  - Place strings into ROM
  - Initialize the internal data EEPROM
  - Manually set configuration bits
  - Manually set ID location

#inline and #separate

- **#inline**
  - Makes following function inline
- **#separate**
  - Makes following function separate (called)
  - Disables stack overflow check
- **Generally you should let the optimizer determine if a function should be separate or inline**

#org

- **Create segment, force code into segment**
- **#org start, end**
  - Place following function/constant in this segment
- **#org start**
  - Continue previous segment

#org Example

- Force following code into 0x100-0x1FF

```
#org 0x100, 0x1FF
void func1(void) {/*code*/}

#org 0x100
const cstring[]=“Hello”;

#org 0x100
void func2(void) {/*code*/}

//Valid Protoype
#seperate void func1(void);
```

#org

- **#org start, end DEFAULT**
  - Forces all following function/constants into this segment.
- **#org DEFAULT**
  - Terminate previous DEFAULT
- **#org start, end { }**
  - Reserve ROM

#org Example 2

- Force following code into 0x100-0x1FF

```
#org 0x100, 0x1FF default
void func1(void) {/*code*/}

const cstring[]=“Hello”;

void func2(void) {/*code*/}

#org default
```

#org

- View .STA to view current segment usage:

```
Segment         Used   Free
0000-0003:      4      0
0004-00FF:      250    2
0100-01FF:      190    66
0200-07FF:      1337   199
```

#build

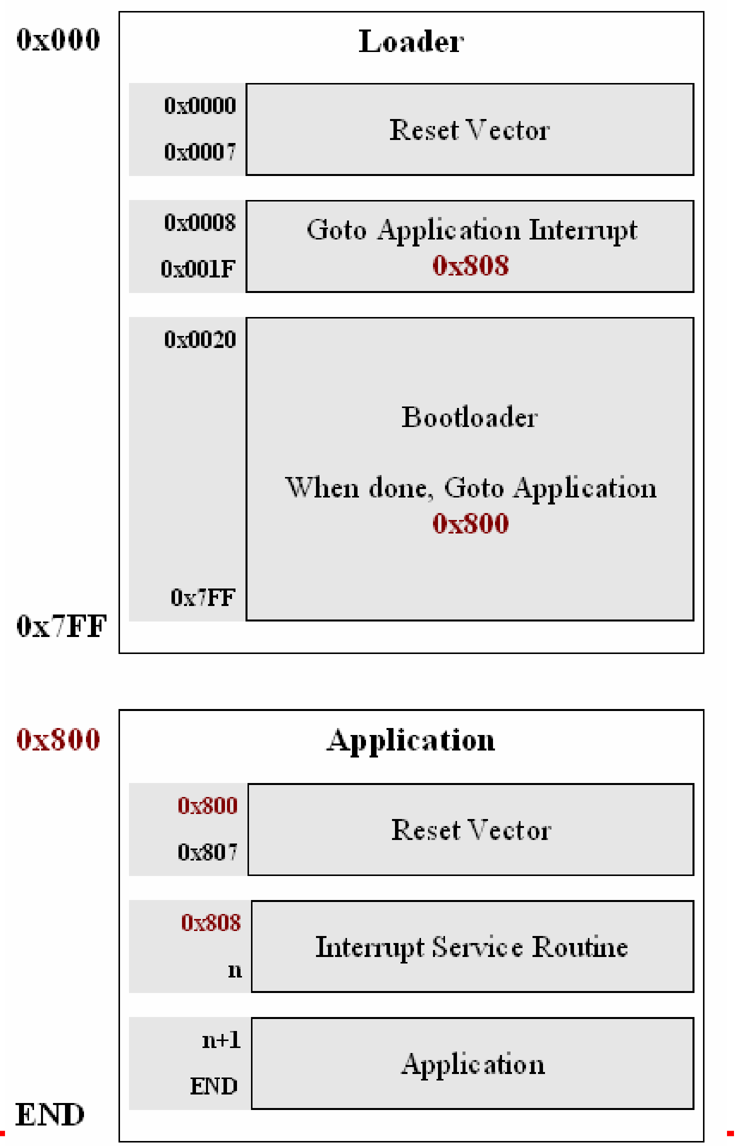
- **Can change reset and interrupt segment**
- **#build(segment=start:end)**
- **Valid segments:**
  - reset – the location of the reset vector
  - interrupt – the location of the interrupt vector
  - memory – external memory for CPU mode
- **Examples:**
  - #build(reset=0x800, interrupt=0x808)
  - #build(memory=0x10000:0x1FFFF)

Memory Management

Bootloader Example

Bootloader Overview

- Two programs:
  - Loader
  - Application
- Each program #org'd to their own space
  - Loader in low memory (0-7FF)
  - Application high memory (800-end)



0x000 Loader

0x0000 – 0x0007: Reset Vector

0x0008 – 0x001F: Goto Application Interrupt 0x808

0x0020 – 0x7FF: Bootloader

When done, Goto Application 0x800

0x7FF

0x800 Application

0x800 – 0x807: Reset Vector

0x808 – n: Interrupt Service Routine

n+1 – END: Application

END

Bootloader/Application Common Code

```
#define BOOT_END          (0x7FF)
#define APP_START         (BOOT_END+1)
#define APP_ISR           (APP_START+8)
#define PROGRAM_END
           getenv(“PROGRAM_MEMORY”)
```

Bootloader Example
Loader Code

```
//prevent bootloader from using application
#org APP_START , PROGRAM_END { }

#int_global
void isr(void)  {
  //goto interrupt in application
  jump_to_isr(APP_ISR);
}

void main(void) {
  if (IsBootloadEvent())
       Bootload();
   #asm
    goto APP_START
   #endasm
}
```

Bootloader Example Application Code

```
#import(file=loader.hex, range=0:BOOT_END)

#build(reset=APP_START,interrupt=APP_ISR)

#org 0,BOOT_END { }

#int_timer0
void timer(void) { /* do timer0 isr */ }

Void main(void)  {
   /* code */
}
```

Memory Managemt

addressmod

addressmod

- **Application defined storage**
  - ISO/IEC TR 18037 (Embedded C)
- **typemod, on steroids!**
- **`nv char productId[PID_LEN];`**
- **Can be used on any data type**
  - Pointers, structs, unions and arrays
  - const not allowed

addressmod Syntax

- **addressmod(identifier,[read,write,]start,end)**
  - identifier is the new qualifier
  - read(int32 addr, int8 *ptr, int8 len)
  - write(int32 addr, int8 *ptr, int8 len)
    - **The IO method to access this memory**
    - **Optional**
  - start/end are the memory range

addressmod Declaration

```
addressmod(nv, read_nv, write_nv,
           0, NV_SIZE);

void read_nv(int32 addr,
             int8 *ram, int8 n)
{ /* read n from addr */ }

void write_nv(int32 addr,
              int8 *ram, int8 n)
{ /* write n from ram */ }
```

addressmod Usage

```
nv NVBuffer[8192];
nv NVID;
nv *NVPtr;

#locate NVID=0

NVBuffer[i]=55;
*NVPtr++ = 0;
```

addressmod Block

- **#type default=qualifier**
  - Following declarations will use this qualifier
  - If qualifier blank, goes back to default

```
#type default=nv

char buffer[8192];
#include <memoryhog.h>

#type default=
```

addressmod Ideas

- External RAM / Flash
- Character/Grahic LCD access thru a multi-dimensional array
- Debug/trap critical variables

RTOS

RTOS Basics

- **Multitasking thru time-sharing**
  - Tasks appear to run at same time
- **'Real Time'**
- **Task is in one of three states:**
  - Running
  - Ready
  - Blocked

The CCS RTOS

- Cooperative Multitasking
- Tightly integrated with compiler
- Supports ALL PIC® MCUs with a Timer
- Available to IDE customers

RTOS Setup

- `#use rtos(timer=X, [minor_cycle=cycle_time])`
  - Timer can be any timer available
  - Minor_Cycle is rate of fastest task
  - Example:

```
#use rtos(timer=1, minor_cycle=50ms)
```

RTOS Tasks

- `#task(rate=xxxx, [max=yyyy], [queue=z])`
  - Following function is RTOS task
  - Will be called at specified rate
  - Max is slowest execution time, used for budgeting.
  - Queue defines RX message size

RTOS Start and Stop

- **rtos_run()**
  - Starts the RTOS
  - Will not return until rtos_terminate()
- **rtos_terminate()**
  - Stops the RTOS

```
#use rtos(timer=1)

#task(rate=100ms, max=5ms)
void TaskInput(void)
  { /* get user input */ }

#task(rate=25ms)
void TaskSystem(void)
  { /* do some stuff */ }

void main(void) {
  while(TRUE) {
     rtos_run();
     sleep();
  }
}
```

RTOS Task Control

- **rtos_enable(task)**
- **rtos_disable(task)**
  - Dynamic task control
  - Enable/Disable the specified task
  - Task is the function name
  - All tasks are enabled at start

RTOS Messaging

- **rtos_msg_send(task, char)**
  - Sends char to task
- **avail=rtos_msg_poll()**
  - TRUE if a char is waiting for this task
- **byte=rtos_msg_read()**
  - Read next char destined for this task

RTOS Yielding

- **rtos_yield()**
  - Stops processing current task
  - Returns to this point on next cycle
- **rtos_await(expression)**
  - rtos_yield() if expression not TRUE

```
#task(rate=100ms, max=5ms)
void TaskInput(void) {
  if (KeyReady())
     rtos_msg_send(TaskSystem, KeyGet());
}

#task(rate=25ms, queue=1)
void TaskSystem(void) {
  SystemPrepare();
  rtos_await(rtos_msg_poll());
  SystemDo(rtos_msg_read());
  rtos_yield();
  SystemVerify();
}
```

RTOS Semaphores

- **Semaphore**
  - Determine shared resource availability
  - A user defined global variable
  - Set to non-zero if used
  - Set to zero if free
- **rtos_wait(semaphore)**
  - rtos_yield() until semaphore free
  - Once free, sets semaphore as used
- **rtos_signal(semaphore)**
  - Release semaphore

RTOS Timing Statistics

- **overrun=rtos_overrun(task)**
  - TRUE if task took longer than max
- **rtos_stats(task, rtos_stats)**
  - Get timing statistics for specified task

```
typedef struct {
  int32 total;  // total ticks used by task
  int16 min;    // minimum tick time used
  int16 max;    // maximum tick time used
  int16 hns;    // us = (ticks*hns)/10
} rtos_stats;
```

RTOS Application Ideas

- User I/O
- Communication Protocols

Class Summary

- **New Features**
- **Output Files and Messages**
- **Memory Control**
  - RAM
  - ROM
  - Bootloader Example
- **addressmod**
- **RTOS**

Q & A

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KeeLoq, KeeLoq logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.