Embedded Controller Programming

Section 2:

Embedded C Compilers

- Ken Arnold
ecp2@hte.com

1

Embedded Compilers

- Embedded Tools are Different!
- Embedded vs. Desktop Compilers
- Cross Compilers (vs. Native)
- Development Configurations
- Compiler Extensions
- Enhancements
- Memory Allocation
- Memory Models

2

8051 C Compilers for PCs

- SDCC Free Compiler – Unlimited
  - Open Source, Windows, Linux++
- Keil C Evaluation – Limited to 2K code
  - Windows, $1695 - now owned by ARM
  - Current eval version won't run on SDK
- Other 8051 Compilers
  - Hi-Tech Software www.htsoft.com
  - IAR www.iar.com

3

Two Compilers

- SDCC
  - Open-source and Free
  - Command Line
- MIDE
  - IDE for 8051 SDCC
  - Harder to compile multi-file projects
  - Small projects
- Keil
  - Full Feature Tool Chain
  - Professional
  - Integrated Tools
  - Simulator
  - In System Debugger
  - Project Management
  - 2K Limited Evaluation

4

Compiler Setup

- Installation and Setup on Win PC
- Development Components
- User Interface
- Libraries

5

Overview of SDCC Tools

- SDCC Freeware Compiler
  - Sandeep Dutta's C Compiler
  - Small Device C Compiler
  - http://sdcc.sourceforge.net
- Freeware Assembler(s), Utilities
- Command Line
- Or IDE from Opcube:

  http://www.opcube.com

6

Screen Shot: Setup SDCC

- Add SDCC screen capture video here

7

Screen Shot: Setup MIDE

- Add MIDE screen capture video here

8

SDCC Directories & Docs

- Directories in \sdcc
  - \LIB - C Compiler Library Files
  - \INCLUDE - .h Header Files
  - \DOC - Documents
  - \BIN - Executable Files
- Documentation
  - http://sdcc.sourceforge.net/doc/sdccman.pdf

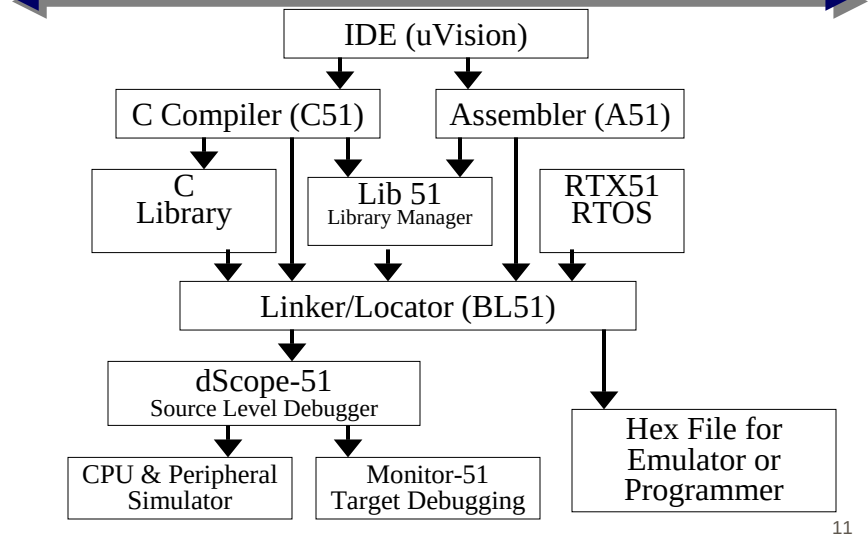
9

Overview of Keil Tools

- Keil uVision IDE
- C51 Optimizing C Cross Compiler
- A51 Macro Assembler
- 8051 Utilities
  - Linker,Object File Converter
  - Library Manager
- dScope Debugger/Simulator
- 8051 Development Model

10

The Development Process



11

Screen Shot: Setup Keil

- Add Keil screen capture video here

12

Important Directories

- Directories in \keil
  - \ASM - Assembler Include Files
  - \BIN - Executable Files
  - \DS51 - dScope
  - \Examples – Sample Applications
  - \INC - C Compiler Include Files
  - \LIB - C Compiler Library Files
  - \MON51 - Target Monitor Files
  - \TS51 - tScope-51 for DOS IOT drivers
  - \RTX51 - Keil Real Time Executive

13

Language Extensions - Keil

- In Compiler manual: c51.pdf)
- **8051 Directives – Page 7**
- **8051 Keywords – Page 57**

  bdata large _task_ bit pdata using code reentrant xdata
  compact sbit data sfr idata sfr16
  interrupt small
- **NOTE: These keywords can be disabled by using the NOEXTEND control directive**

14

Language Extensions SDCC

- Most Extensions Same as Keil:
  - Varibles: data, idata, xdata, code...
  - Functions: reentrant
- There are differences
  - Parameter passing, byte order...
- See section 3 of sdccman.pdf for details on the SDCC implementation

15

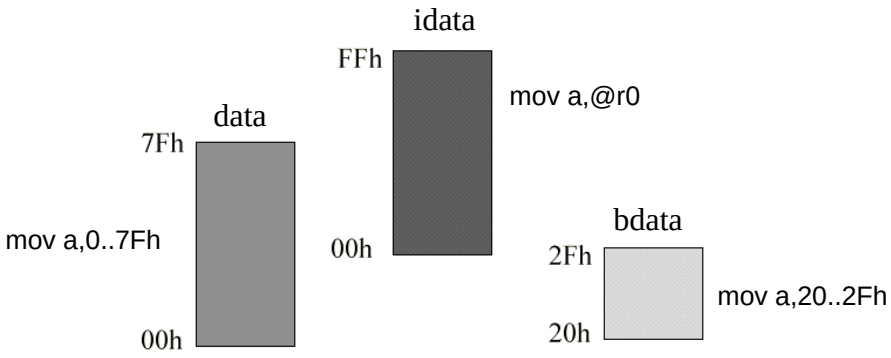
8051 Memory Areas

- What are the 8051's Memory Spaces Called in C?
- Why do we need C memory extensions?

16

Internal Data Memory

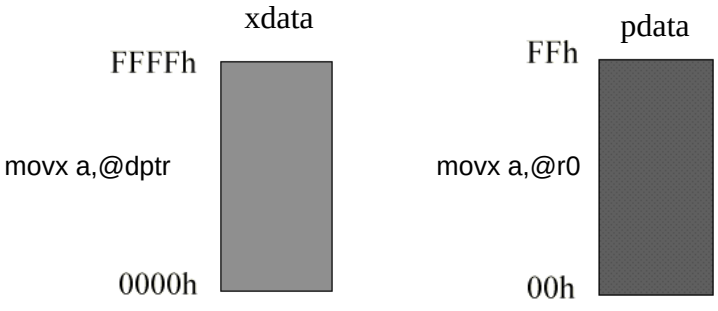
- data, idata, bdata keywords



17

External Data Memory

- xdata, pdata keywords

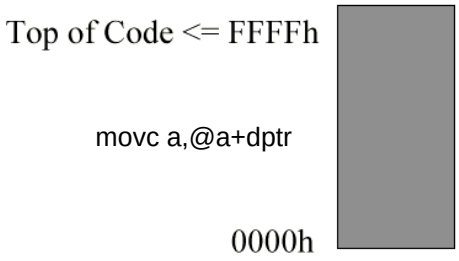


18

Code Memory

- code keyword

Note: most support external code/data beyond 64K address space, using banked memory



19

Memory Type Examples

```
int var1, var2;
char data var3;
char code text[] = “Enter Value:”;
unsigned long xdata array[500];
unsigned int pdata dimension;
unsigned char xdata
vector[10][4][4];
char bdata flags;
```

- If a type is omitted, the implicit type is used !
- Implicit type depends on memory model...

20

8051 Memory Models: small

- SMALL Model (“near” for SDCC & others)
  - Most often the best choice
  - All variables default to internal memory
    - What memory keyword is this?
  - Stack/Variables share internal space
- What are advantages/disadvantages?
  - Code Size, Speed, Data Size Limitations…

21

Memory Models: large

- LARGE: (or “far” in SDCC & others)
  - Second Choice
- LARGE Model Has:
  - All variables in external memory
- What keyword is this?
- What are advantages/disadvantages ?

22

Memory Models: compact

- COMPACT:
  - Seldom used
- COMPACT Model:
  - All variables in *one page (256) of ext data*
  - What keyword is this?
  - Stack in internal space
- What are advantages/disadvantages ?

23

8051 Memory Types

- code - Program Memory (64K bytes)
  - accessed by opcode MOVC @A+DPTR
  - Good for constants, lookup tables
  - Usually NON-volatile memory
- data - Internal Data (128 bytes 0..7F)
  - accessed directly over lower 128 bytes
  - Fastest, Smallest code!
  - a.k.a. “near” in SDCC and others

24

8051 Memory Types (cont'd)

- idata - Internal Data (256 bytes:0..FF)
  - accessed indirectly by @R0, @R1
- bdata - Bit Addressable (16 bytes: 20-2F)
  - accessed as bit or byte variables
- xdata - External Data (64K bytes: 0-FFFF)
  - accessed indirectly by MOVX @DPTR
  - a.k.a. "far" in SDCC and others
- pdata - External Data (256 byte pages)
  - accessed indirectly by MOVX @Rn

25

8051 Data Types - page 64

Data Type	Bits	Bytes	Value Range
bit	1	…	0,1
signed char	8	1	-128..+127
unsigned char	8	1	0..255
enum	16	2	-32,768..+32767
signed short	16	2	-32,768..+32767
unsigned short	16	2	0..65535
signed int	16	2	-32,768..+32767
unsigned int	16	2	0..65535

26

8051 Data Types (cont'd)

Data Type	Bits	Bytes	Range
signed long	32	4	-2147483648..+2147483647
unsigned long	32	4	0..4,294,967,295
float	32	4	+/-1.175494 E-38 to +/-3.402823 E+38

Extended types for 8051 Architecture:

bit	*1*	*--*	*0,1*
sbit	*1*	*--*	*0,1*
sfr	*8*	*1*	*0..255*
sfr16	*16*	*2*	*0..65535*

27

Nonstandard C Data Types – bit

- Keil bit types
  - Page 65 Bit Type - Example

```
bit testfunc( bit flag1, bit flag2)
  {
  ...
  return (0);
  }
```

28

Nonstandard Data Types – bit

- Bit variables have some limitations;
  - A bit can't be declared as a pointer: bit *ptr is invalid
  - An array of type bit is invalid: bit flags [5] is invalid
- Functions using an explicit register bank cannot return a bit
- Only data or idata memory types can be used with bit
- A maximum of 128 bit variables can be used in a program (why?)

29

Bit Addressable Objects - Keil

Declarations:

```
int bdata base; // base is in bit addressable mem
char bdata bitarray [4]; // bit addressable array
bit mybit0 = base^0; // bit 0 (lsb) of base

bit barray35 = bitarray[3]^5; // bit 5 of element 3
```

Assignments:

```
barray35 = 0;   // Clear bit 5 in array element 3
bitarray[3] = 0xFF; // Set element 3 to FF
mybit0 = 1;       // Set bit 0 of base
```

30

Bit Objects in SDCC

- When a variable is declared as a bit, it is allocated into the bit addressable memory, e.g.:

  bit test_bit;

- Writing 1 to this variable:

  test_bit = 1 ;

- Generates the Assembly code:

  setb _test_bit

- The bit addressable memory consists of 128 bits which are located from 0x20 to 0x2f in data memory.

31

Bit Addressable Objects - Keil

Bits can also be used in struct and unions;

```
bdata struct bitaddr
{
    char m1;
    int m2;
}tcp;
```

32

Differences: SDCC v. Keil

- Variable byte order
  - Keil is big-endian: MS byte at low address
  - SDCC is little-endian: LS byte at low address
- bit and sbit declarations are different
- Pointers are declared differently
- Parameters passed in different registers
- See documents for details

33

Data Types: SFRs - Keil

Special Function Registers;
sfr, sfr16, and sbit types can be used with SFRs
see reg52.h in keil\c51\inc for more examples

```
sfr P0 = 0x80;  // Port 0, defined at address 0x80h
sfr P1 = 0x90;  // Port 1, defined at address 0x90h

sfr16 T2 = 0xCCCD; // Timer 2, T2L, T2H 0CCh, 0CDh
                   // Note order is lo byte, hi byte!

sbit EA = 0xAF; // SFR enable bit in IE register
sbit EA = IE^7 // EA = IE register bit 7
sbit EA = 0xA8^7 // EA = Address A8, bit 7
```

34

Vars at Absolute Addresses

SDCC: Variables can be located at absolute locations using the at keyword;

```
xdata at 0x8000 int time
                    /* int at xdata 0x8000 */
```

This is useful for things that are located at fixed addresses, such as memory mapped I/O devices!

But this is not enough... need volatile for I/O.

35

Vars at Absolute Addresses

KEIL: Variables can also be located at absolute locations using the _at_ keyword;

```
xdata int time _at_ 0x8000
                    /* int at xdata 0x8000 */
```

This is useful for things that are located at fixed addresses, such as memory mapped I/O devices!

Such variables must be OUTside a function def.!

36

Memory Mapped I/O

- Like the Government and Microsoft, The Optimizer Will “Help” You... :)
- Optimizer changes:
- From:

```
  mov dptr, #ioport
poll: ...
  movx a, @dptr
  jnb acc.1, poll
```

- To:

```
  mov dptr, # ioport
  movx a, @dptr ;
poll:
  jnb acc.1, poll
```

37

Keyword: Volatile

- *volatile* unsigned char ioport
- Volatile tells the compiler that this variable is something that can be changed by the hardware (or an ISR or another task...)
- volatile keyword prevents optimization of accesses to the variable

38

Pointer != Integer

- Some C Code Assumes That a Pointer is the Same Length as an Int
  - Poorly Designed Code Assigns Ints with the Value of a Pointer & vice versa
- This is NOT the Case in Most Systems!
  - Truncation & Other Errors Will Occur
- In these Versions of C, Pointers can be 1, 2 or 3 bytes...

39

Pointer Types

- **Generic Pointers –** Can point anywhere in any memory space – standard pointers
  - Takes 3 bytes
  - 1st Byte is for memory type
  - 2nd Byte – High order byte of offset
  - 3rd Byte – Low order byte of offset
  - Most library routines use generic pointers
  - Converted to/from other types by casting

40

Keil C Generic Pointers

- Generic Pointer First Byte Values

Memory Type:	Code Value:
idata/data/bdata	0x00
xdata	0x01
pdata	0xFE
code	0xFF

41

Pointer Types

- Memory Specific Pointers – These can only be used with a specific memory type:
  - idata, data, bdata, and pdata require 1 byte
  - code and xdata require 2 bytes
  - These can be much more efficient !!

42

Pointer Types

- Examples of Memory Specific Pointers:

```
char data *cdptr;          int xdata *xdptr;
char data chdata;          int xdata x_temp;
cdptr = &chdata;           x_temp = 0x1000;
                           *xdptr = x_temp;
char code *coptr;
char code carray[20]; // array by address
coptr = carray;
```

43

Pointer Type/Code Efficiency (Keil)

Idata Pointer	Xdata Pointer	Generic Pointer
Char idata *ip;	char xdata *xp;	char *p
Char val;	char val;	char val;
Val = *ip;	val = *ip;	val = *ip;
8051 Code		
MOV R0, ip	MOV DPL, xp+1	MOV R1, p+2
MOV val, @R0	MOV DPH, xp	MOV R1, p+1
	MOV A, @DPTR	MOV R3, p
	MOV val, A	CALL CLDPTR
Efficiency & Size		
Ptr. Size – 1 byte	2 bytes	3 bytes
Code Size – 4 bytes	9 bytes	11 bytes + lib.
Speed – 4 cycles	7 cycles	13 cycles

- **Code Size and (Speed) can be affected by pointer type!!**

44

Pointer Type Conversion

- Compilers can convert between memory specific and generic pointers either explicitly using type casts or through the compiler
  - If no function prototype is present, a memory specific pointer is always converted to a generic pointer – this can cause an error if function is expecting a memory specific pointer
  - Prototype functions in code or with #include

45

Pointer Type Conversions (Keil)

Examples:

```
Int *p1;                  /* Generic Pointer (3 bytes) */
Int xdata *p2;            /* xdata ptr (2 bytes ) */
Int idata *p3;            /* idata ptr (1 byte) */
Int code *p4;             /* code ptr (2 bytes) */

P1 = p2;                  p1 = p3;
  MOV p1,#01h               MOV p1, #00h
  MOV p1+01h,p2             MOV p1+01h, #00h
  MOV p1+02h,p2+01h         MOV p1+02h, p3
```

46

Function Declarations

- C Function Code Can Be Modified:
  - Selecting a specific memory model:
    result foo (arg) small
  - Choosing a specific register bank:
    result foo (arg) using 3
  - Specifying an interrupt function:
    void foo (void) interrupt 2
  - Specifying reentrant code generation:
    result foo (arg) reentrant

47

Additional SDCC Features

- Critical:
  - Interrupts disabled before, and restored after
    int foo () critical { ... } ;
- Naked:
  - No register preservation (no auto push/pop)
    int foo () interrupt naked { ... } ;
- Semaphore
  - Test using atomic test and clear (JBC opcode)

48

Function Parameter Passing: SDCC

- SDCC always uses the global registers DPL, DPH, B and ACC to pass the first parameter to a routine
- 2nd parameter onwards is either allocated on the stack (for reentrant routines or if --stack-auto is used) or in data/xdata memory (depending on the memory model)

49

Keil Arguments & Return

- Parameters can be passed either through registers (default) or in specific memory locations
- Bit values should be at the end of the argument list if there are non bit arguments, otherwise other args are not passed in registers

50

Function Return Val - Keil

- Return values are *always* passed in regs
- Bit – In carry flag
  - Char – R7
  - Int – R6 & R7
  - Long, float – R4-R7
  - Generic ptr – R1-R3

51

Interrupt Functions p 92

- Interrupt Service Routines (ISR's) can be coded just like any other function
- Ex: Timer 0 ISR Function using Bank 2:

```
void timer0 (void) interrupt 1 using 2 {
  if (++interruptcnt == 4000) //count to 4000
  { second++;       // incr seconds counter
    interruptcnt = 0; //clear interrupt cnt
  }
}
```

52

Interrupt Functions (cont'd)

- The contents of ACC, B, DPH, DPL, and PSW, when required, are saved automatically on stack
- All working registers used in the ISR are stored and restored by default
- Registers R0..7 *may* be saved...
- RETI instruction is used to exit the function

53

Interrupt Function Rules:

- No function arguments can be used with interrupt functions – WHY ?
- No return values can be used with interrupt functions – WHY ?
- You cannot directly invoke an interrupt function – WHY ?

54

Interrupt Rules (cont'd)

- Don't call an interrupt function through a function pointer
- Functions that are called by interrupt functions must use the same register bank as the interrupt function.
- The range of valid interrupts is limited (see documentation)

55

Interrupt Numbers

#	Source	Interrupt Address
0	External /INT0	0x0003 / 0x4003
1	Timer Flag 0: TF0	0x000B / 0x400B
2	External /INT1	0x0013 / 0x4013
3	Timer Flag 1: TF1	0x001B / 0x401B
4	Serial RI or TI bit	0x0023 / 0x4023
5	Timer Flag 2: TF2	0x002B / 0x402B

- and so on …

56

Reentrant Functions

- Reentrant functions are functions which can be invoked one or more times before completing and be shared by several processes (i.e. by RTOS or interrupt calls)
- Allows you to selectively define a function as being reentrant (for local variables only)
- Note that library functions are generally NON-reentrant!
- Necessary but not sufficient to guarantee the function is reentrant!

57

Reentrant Fns (cont'd)

- For each reentrant function, a reentrant stack area is simulated in internal or external memory, depending on the memory model used
- Small model uses the idata memory
- Compact model uses the pdata memory
- Large model uses the xdata memory
- You can also specify which area to use for a reentrant function

58

References

- Patterns For Time-Triggered Embedded Systems
  - Michael J. Pont
- C and the 8051 vol. I
  - Thomas W. Schultz
- SDK Manual
- The 8051 Microcontroller
  - James Stewart and Kai X. Miao

59

Summary

- Tool Intro
- Data Types
- Extensions:
  - Variables
  - Memory Types
  - Special Function Declarations
- Embedded Compilers Really Are Different!

60

Homework

- Write your own putchar.c function to use the same serial port as the SDK monitor: output to the memory mapped UART on connector P5 directly rather than through the SDK monitor
- Preferred: use pointers to UART registers
- Write putchar.c for both SDCC and Keil
- Detailed instructions will be e-mailed

61