Small Device C Compiler

史斌

2015-11-21

What is SDCC

- Cross tool chain: preprocessor, compiler, assembler, linker, debugger, simulator, libraries
- Targetting 8-bit MCU: MCS-51 / DS390 / DS400, ST7 / STM8, HC08 / HCS08, Z80 / Z180 / GBZ80 / R2000 / R3000A / TLCS90, PIC16 / PIC18 (incomplete), AVR (abandoned)
- C89 / C99 / C11 with extentions (not support C++)
- Comand line only without IDE
- Open source (GPLv2 and others)
- Production applicable

Similar Tools

- Keil (MCS-51, ARM)
- IAR (ARM, AVR, PIC*, MCS-51, MSP430, ……)
- GCC (AVR, ARM)
- Cosmic (STM8)
- MPLAB (PIC*)
- Metroworks (Freescale HC*)

Components

- sdcpp: the proprecessor based on gcc-4.6
- sdcc: the compiler
- sdas: the assembler compitable with GNU as
- sdld: the linker
- sdbinutils: based on a very old binutils
- ucsim: the simulator support many targets
- libraries: common libraries and arch-depedant libraries
- other gadgets

Extentions

- __data, __idata, __xdata, __pdata, __bit, __code

  __idata int a;

  __xdata long * __data p;

  __bit c;

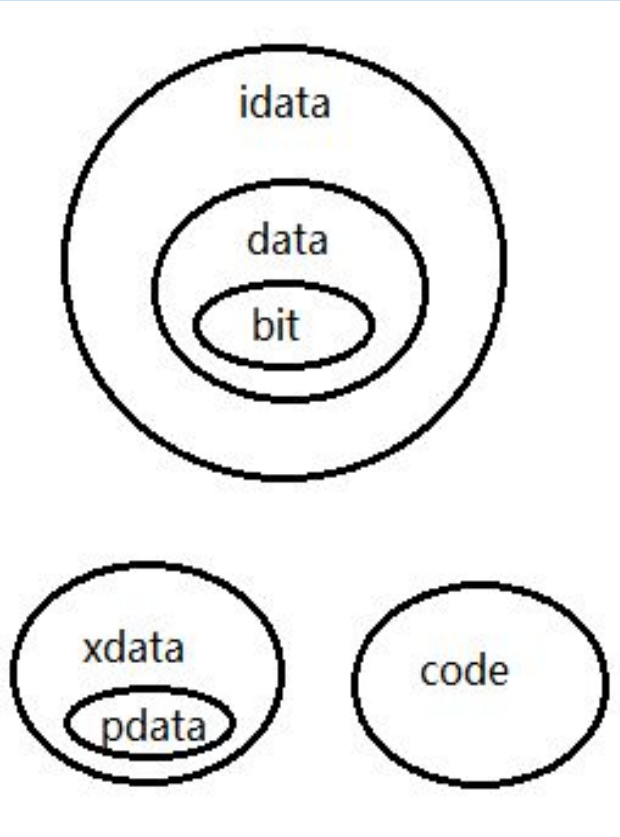
  char (* __pdata fp)(void);

- __sfr, __sfr16, __sft32, __sbit

  __sfr __at (0x80) P0;

  __sfr16 __at (0x8C8A) TMR0;

  __sbit __at (0xd7) CY;



Extentions

- __at

  __xdata __at (0x7ffe) unsigned int chksum;

  __bit __at (0x02) bvar;
- __interrupt

  void timer_isr (void) __interrupt (1) {...}
- __using

  void timer_isr (void) __interrupt (1) using (2) {...}
- __critical

  int foo (void) __critical {...}

Extentions

- __reentrant
  int foo (void) __reentrant {...}
- __naked
  void foo (void) __naked {...}
- __asm__
  __asm__ (”; This is a comment\nlabel:\n\tnop”);
- __asm / __endasm
- __eeprom
  __eeprom int w = 4;

Command Line Options

- --model-small, --model-medium, --model-large
- --stack-auto
- --xstack
- --callee-saves
- --code-size / --iram-size / --xram-size / --stack-size
- --code-loc / --idata-loc / --xdata-loc / --stack-loc

Stages

- Flex: source -> tokens
- Bison: tokens -> AST
- Intermediate representation: AST -> icode
- Common optimization: icode -> icode
- Register allocation
- Code generating: icode -> asm
- Arch-dependant optimization: asm -> asm
- Assemble: asm -> objective
- Link: objective -> executable

SDCC icode

- Arithmatic: add / sub / mul / div / mod
- Comparason: > / >= / < / <= / == / !=
- Logic: and / or / not
- Bitwise logic: and / or / not / xor
- Stack operation: push / pop
- Shift: left / right / arithmatic / logic
- Flow control: jump / call / return
- Pointer: address of variable / value of pointer
- Others: assign / cast

Common Optimization

- Common subexpression elimination
- Loop optimizations (loop invariant, strength reduction of induction variables, loop reversing)
- Dead code elimination
- Copy propagation
- Constant folding & propagation
- Jump tables for switch statements
- Live analysis / register allocation

My Work Since Feb 2014

- Front end bug fix
- STM8 back end optimization and bug fix
- 138 commits
- 80 bug fixes
- 5 new features