Supporting Embedded Innovation Since 1983
ST7 C Compiler

IDEA
Source editor

COMPILER
Code generator
Linker

DEBUGGER
Simulator, Emulator
IDEA

- Windows Integrated environment
- Integrates with C and Assembler
- Project management
- Automatic make and build
- Integrated Editor
- Automatic Error Handling
- Integration of documentation
IDEA Flowchart

Source Files -> Compiler -> Object Files

Editor

Project Manager

Final Application

Debugger

Executable

Linker

ST7 DEVELOPMENT TOOLS
IDEA Main Window
ST7 C Compiler Features

- **ANSI / ISO Compliant**
  - Full language implementation
  - Standard C libraries (subset for embedded)
  - Compatible with native compilers

- **Memory Spaces**
  - Short range (zero page) / Long range
  - Direct access to I/O Registers
  - EEPROM support
ST7 C Compiler Features

- **Full Floating Point Support**
  - Single Precision IEEE-754
- **C and Assembly Library Source Code**
  - Optional Integer only library support
- **Bit Optimizations**
  - Extensive use of bit instructions
  - Bit variables
  - Option to reverse bit ordering
  - 8-Bit Bitfield support
ST7 C Compiler Features

- **Stack Implementation**
  - Physical Stack (@stack)
    - fully recursive and reentrant
  - Overlay Static Memory (@nostack)
    - Optimized Memory Allocation by Linker
- Both allowed in the same application

- **And Also**
  - Overflow control for signed compares
  - Absolute C and Assembler Listings
Compiler Architecture

Switches to enforce
- Prototyping
- Strict checking for code inconsistencies
- ...

Switches to tune
- first Levels of Optimization
- listings
- ...

C Parser

I.L.

Code Generator

Assembler

C

Error Messages

ST7 DEVELOPMENT TOOLS
Compiler Architecture

Full Control on any optimization Feature

Optimizer

Assembler

Assembler

Object

C/ASM Listings
Compiler Architecture

Smart Linker, only necessary Library Modules are loaded

Linker

Executable Object

Libraries

Convertisers

S-Record, IEEE695 ELF/DWARF Third Party Debug Environment

ZAP-HLL-Debugger

Utilities

Map File: Symbols, Stack, Segments, Info

Absolute Listings

HLL Debug Info

Inspect Objects Absolute Listings ...

Simulation, Emulation, ROM Monitor

ST7 DEVELOPMENT TOOLS

11
Memory Spaces

Short Addressing
(@tiny)

Static byte access:
`ld a,adr`

Long Addressing
(@near)

Stack byte access:
`ld x,s`
`ld a,(0x100,x)`
Memory Models

- **Physical Stack**
  - Stack long
  - Stack short

- **Static Memory**
  - Memory large
  - Memory medium
  - Memory small
  - Memory short
  - Memory compact
**Memory Models**

<table>
<thead>
<tr>
<th>Model</th>
<th>Stack</th>
<th>Globals</th>
<th>Pointers</th>
</tr>
</thead>
<tbody>
<tr>
<td>modsl</td>
<td>phys</td>
<td>long</td>
<td>16 bits</td>
</tr>
<tr>
<td>mods</td>
<td>phys</td>
<td>short</td>
<td>16 bits</td>
</tr>
<tr>
<td>modml</td>
<td>long</td>
<td>long</td>
<td>16 bits</td>
</tr>
<tr>
<td>modmm</td>
<td>long</td>
<td>short</td>
<td>16 bits</td>
</tr>
<tr>
<td>modms</td>
<td>short</td>
<td>long</td>
<td>16 bits</td>
</tr>
<tr>
<td>modm</td>
<td>short</td>
<td>short</td>
<td>16 bits</td>
</tr>
<tr>
<td>modc</td>
<td>short</td>
<td>short</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

**ST7 DEVELOPMENT TOOLS**
Mixing Models

• One default model for the whole application
• One matching library type for the whole application
• Possible mixing in the same application of
  - mods and modsl
  - modm and modms
  - modmm and modml
• Use modifiers to adapt behaviour:
  - @stack forces arguments and locals on the physical stack
  - @nostack forces arguments and locals in simulated stack
• Interrupt functions defaulted to stack model for nested interrupts
C Language Extensions

- **Absolute addressing**
  ```c
  unsigned char PORTB @0x03;
  ```
- **Short range addressing**
  ```c
  @tiny char shvar;
  ```
- **Long range addressing**
  ```c
  @near int lgvar;
  ```
- **Internal EEPROM**
  ```c
  @eeprom short eevvar;
  ```
- **Bit variables**
  ```c
  _Bool bitvar;
  ```
Data Allocation

Variable

- Short range
  - Initialized
    - .bsct
  - Non initialized
    - .ubsct

- const
  - .const

- _Bool
  - .bit

-eprom
  - .eprom

- Long range
  - Initialized
    - .data
  - Non initialized
    - .bss

@tiny char shvar;
const int cvar;
_Bool bvar;
eeprom int eeval;
@near char lgvar;
Code Allocation

- **Functions**: Pointer to `.text`
- **Literals**: Pointer to `.const`

### Options
- **+split**: one function per section
- **+nocst**: literals and const in code section
- **+nobss**: all data considered as initialized
Section Renaming

- **Code section**
  
  ```
  #pragma section (sname)
  .text → .sname
  ```

- **Data section**
  
  ```
  #pragma section @tiny [name]
  #pragma section @near {sname}
  ```

  ```
  const, @eeprom, _Bool
  ```
Function call

\[ x = f(a, b, c); \]

| Local Variables | \( a \) | Return Address | \( b \) | \( c \) |

return in A (char) or X:A (int, pointer) or c_lreg (long, float)

Simulated stack:

```
1d a, _f$L-3
```

Physical stack:

```
1d x, s
1d a, (0x103, x)
```
Inline assembly

```
#asm
rim
ld a,_var
#endasm

#pragma asm
sim
jp _main
#pragma endasm
```

- inside or outside a function
- connection with global C objects
- no connection with local C objects
Inline assembly

```asm
$N:
  dec  a
  jrne  $L
#endasm
```

- $N creates a new label
- $L uses the current label
**Inline assembly**

result = _asm("asm code", input);

- Result copied from returned value in A (char) or X:A (int)
- Input expression evaluated in A (char) or X:A (int)

Assembler code:

```
crc = _asm("add a,#$80\n rlc a", crc);
```

- LD A,_crc
- ADD A,#$80
- RLC A
- LD _crc,A

**ST7 DEVELOPMENT TOOLS**
• **library functions**

```c
@inline char *memcpy(char*, char*, int);
@inline char *memset(char*, char, int);
@inline char *strlen(char*);
```

• **user functions**

```c
@inline void func(int arg);
```

- cannot return any value
- any kind and number of arguments
- replace function call by its body
Bit Variables

_Bool Bitvar;

- Conform to ANSI standard C99 (C9X)
- Global and local bit variables packed into bytes
- Argument and function return values 1 bit in 1 byte
- Localized bit definition

_Bool PA3 @PORTA:3;
Interrupt Functions

```c
@interrupt void it_func(void);
@interrupt @nostack void it_func(void);
```

- `c_x` (2 bytes) used for extending X register to 16 bits
- `c_y` (2 bytes) used for extending Y register to 16 bits
- `c_lreg` (4 bytes) used for long and float operations

*Automatically and selectively* saved with the Y register by interrupt functions

If there is a function call inside the interrupt routine:

- `Y, c_x, c_y` saved even if not explicitly used, unless `@nosvf` is specified
- `c_lreg` *NOT* saved if not explicitly used, unless `@svlreg` is specified
Linker

Sections in object files

Segments in executable file

ST7 DEVELOPMENT TOOLS
Linker Controls

Segment definition:

```
+seg .text -b 0x8000 -m 0x2000 -n code
```

- **Section type**
- **Start address**
- **Maximum size**
- **Segment name**

```
+seg .const -a code -n const
```

ST7 DEVELOPMENT TOOLS
Symbol definition:

```
+def symbol=value
```

Symbol name

- `_ssize` = 0x80
- `_hstart` = `_ssize`
- `_bstart` = `.bss`
- `_cstart` = `start(code)`

**Absolute value**

**Other symbol**

**Section reference**

**Segment reference**

`start()`

`end()`

`size()`
## Linker Controls

**Linker command file:**

```
+seg .text -b 0xe000 -m 0x1ffe0 -n code
+seg .const -a code -n const
+seg .bsct -b 0x80 -m 0x80 -n ram

crt.s.o  Startup code
appli.o  Application code
libm.st7 library
+seg .const -b 0xffff0 -n vectors
vectors.o Interrupt vectors
```
Supporting Embedded Innovation Since 1983