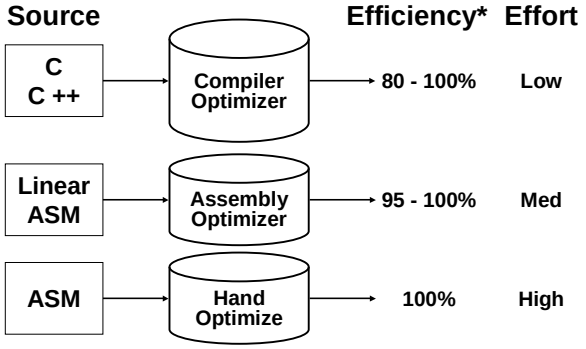
Lecture 3 - Software

Outline

- Programming Methods
- Code Composer Studio (CCS)
- Projects
- Working with C

Programming the 'C6000



* Typical efficiency vs. hand optimized assembly

Source Code Examples

```
float mac(float *m, float *n, int count)
{ int i, float sum = 0;

  for (i=0; i < count; i++) {
    sum += m[i] * n[i]; }   ...
```

C

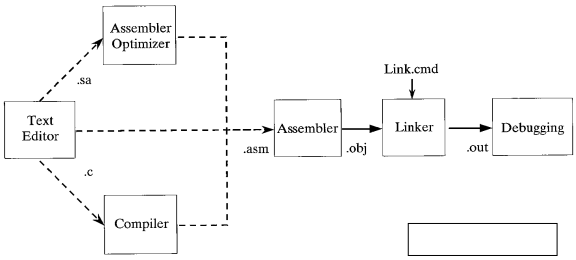
```
        MVK     .S      40, cnt
loop:
        LDH     .D      *ap++, a
        LDH     .D      *xp++, x
        MPY     .M      a, x, prod
        ADD     .L      y, prod, y
        SUB     .L      cnt, 1, cnt
[cnt]   B       .S      loop
```

Linear ASM

```
;** --------------------------------------------------*
LOOP:   ; PIPED LOOP KERNEL
        LDDW    .D1     A4++,A7:A6
||      LDDW    .D2     B4++,B7:B6
||      MPYSP   .M1X    A6,B6,A5
||      MPYSP   .M2X    A7,B7,B5
||      ADDSP   .L1     A5,A8,A8
||      ADDSP   .L2     B5,B8,B8
|| [A1] B       .S2     LOOP
|| [A1] SUB     .S1     A1,1,A1
;** --------------------------------------------------*
```

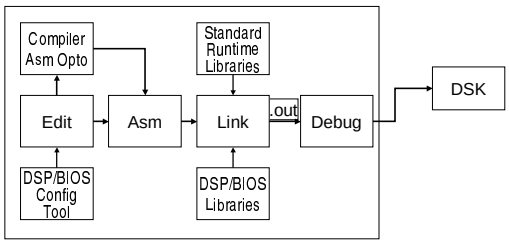
ASM

C6x Software Tools



.c = C source file
.sa = linear-assembly source file
.asm = assembly source file
.obj = object file
.out = executable file
.cmd = linker command file

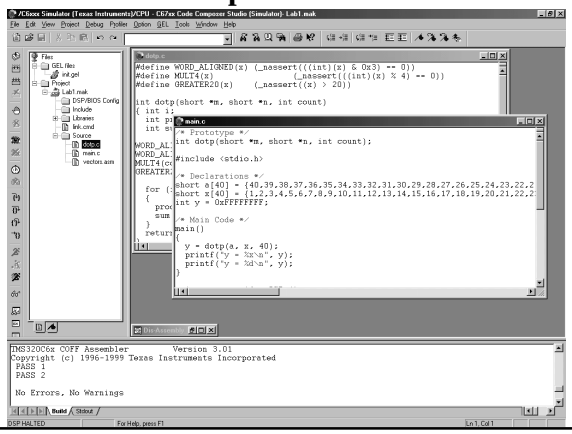
Code Composer Studio



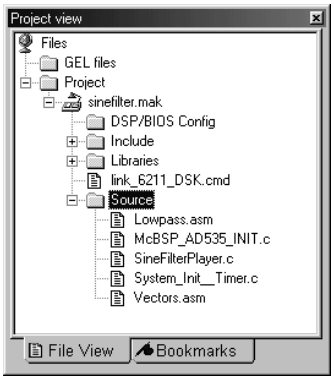
Code Composer Studio Includes:

- Integrated Edit / Debug GUI
- Code Generation Tools
- BIOS: Real-time kernel
  Real-time analysis

Code Composer Environment

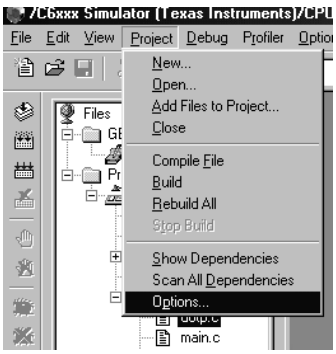


What's a Project?



- **Code Composer keeps track of all project files:**
  - **Source**
  - **Library**
  - **Command**
  - **Etc.**
- **You can add files to a project with drag-n-drop onto the .MAK file**

Project Options

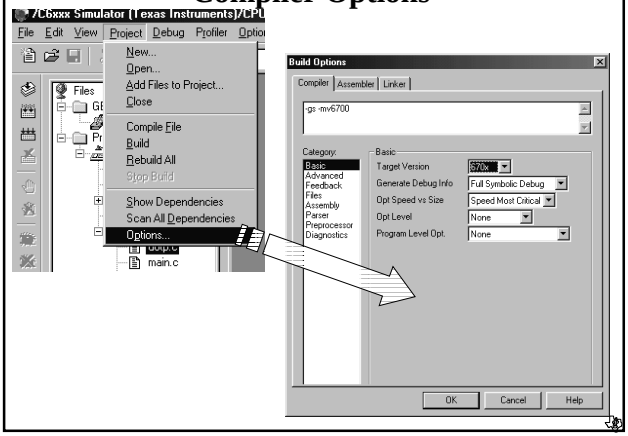


Indicates how output file should be constructed

- **Which Optimizations**
- **Where to find files/libs**
- **'C62x or 'C67x**
- **How to link files**
- **Etc.**

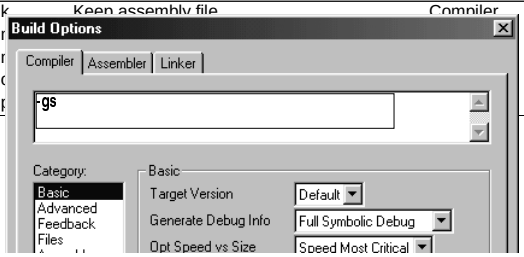
Full list of options found in *Optimizing C/C++ Compiler User's Guide (SPRU187)*

Compiler Options



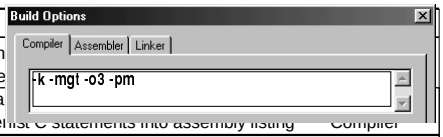
Debug Compiler Options

	Options	Description	CC Tab
	-mv6700	Generate 'C6700 code ('C6200 is default)	Compiler
	-fr <dir>	Directory containing source files	Compiler
debug	**-g**	**Enables src-level symbolic debugging**	**Comp/Asm**
	-s	**Interlist C statements into assembly listing**	**Compiler**
	-k	Keep assembly file	Compiler



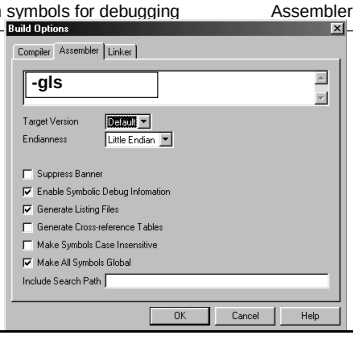
Optimization Compiler Options

	Options		
	-mv6700	Gen	
	-fr <dir>	Dire	
	-g	Ena	
	-s	Interlist C statements into assembly listing	Compiler
speed opto	**-k**	Keep assembly file	Compiler
	-mg	Enables minimum debug to allow profiling	Compiler
	-mt	No aliasing used	Compiler
	-o3	Invoke optimizer (-o0, -o1, -o2/-o, -o3)	Compiler
	-pm	Combine all C source files before compile	Compiler
	-ms	Minimize code size (-ms0/-ms, -ms1, -ms2)	Compiler
	-oi0	Disables automatic function inlining	Compiler



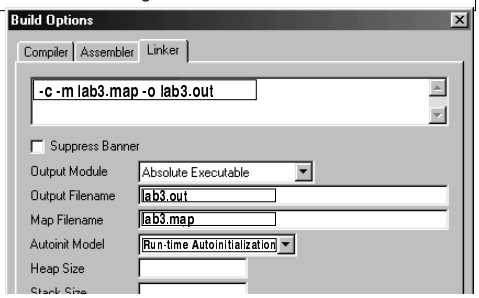
Assembler Options

Options	Description	CC Tab
- g	Enables src-level symbolic debugging	Comp/Asm
- l	Create assembler listing file (small -L)	Assembler
- s	Retain asm symbols for debugging	Assembler



Linker Options

Options	Description	CC Tab
- o <file>	Output file name	Linker
- m <file>	Map file name	Linker
- c	Auto-initialize global/static C variables	Linker

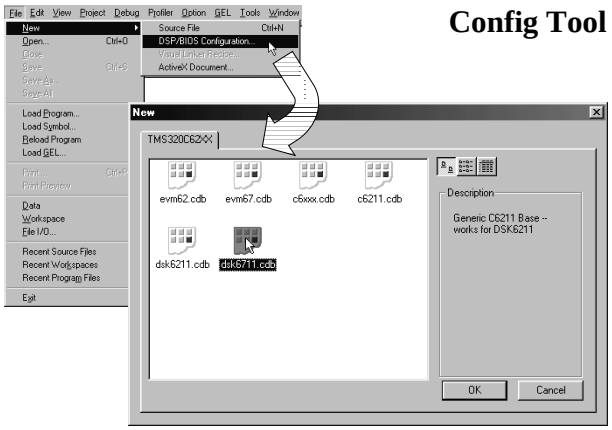


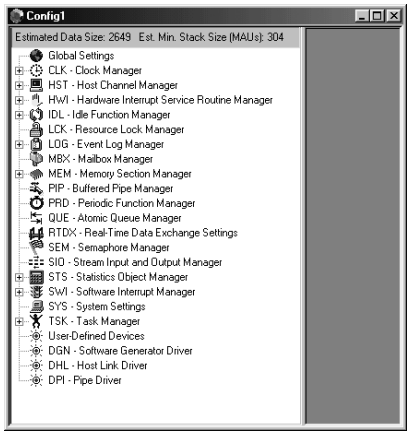
	Options	Description	Options Tab
	-mv6700	Generate 'C6700 code ('C6200 is default)	Compiler
	-fr <dir>	Directory containing source files	Compiler
debug	-g	Enables src-level symbolic debugging	Comp/Asm
debug	-s	Interlist C statements into assembly listing	Compiler
	-k	Keep assembly file	Compiler
	-mg	Enables minimum debug to allow profiling	Compiler
speed opto	-mt	No aliasing used	Compiler
speed opto	-o3	Invoke optimizer (-o0, -o1, -o2/-o, -o3)	Compiler
	-pm	Combine all C source files before compile	Compiler
	-l	Create assembler listing file (small -L)	Assembler
	-s	Retain asm symbols for debugging	Assembler
	-o <dir>	Output file name	Linker
	-m <dir>	Map file name	Linker
	-c	Auto-Init C variables (-cr turns off autoinit)	Linker
	-l	Library archive file	Comp/Linker
	-i	Defines #include search path	Comp/Linker

'C6000 C Data Types

Type	Size	Representation
char, signed char	8 bits	ASCII
unsigned char	8 bits	ASCII
short	**16 bits**	**2's complement**
unsigned short	16 bits	binary
int, signed int	**32 bits**	**2s complement**
unsigned int	32 bits	binary
long, signed long	**40 bits**	**2's complement**
unsigned long	40 bits	binary
enum	32 bits	2's complement
float	**32 bits**	**IEEE 32-bit**
double	64 bits	IEEE 64-bit
long double	64 bits	IEEE 64-bit
pointers	32 bits	binary

Config Tool





Config Tool

- Simplifies system design
- HWI automatically creates vector table