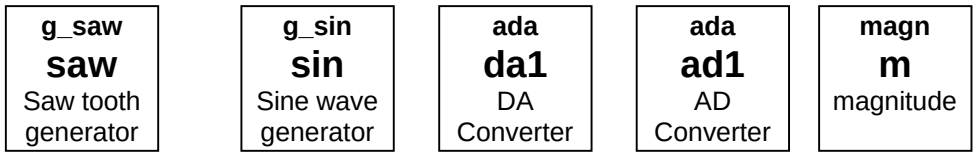
Programming with functional blocks using FIBULA

Why a functional block language ?

Digital Signal Processors have very specific architectural characteristics which make them particularly efficient for most signal processing tasks (MAC instruction, parallel instructions, modulo addressing, DMA channels, wired DO loops etc. ...). *However this implies that several instructions are bound together in indivisible blocks, in order to preserve this high performance.* Furthermore required resources such as memory, registers and DMA channels affected to the block's function must have been properly reserved and initialized at the beginning of the program.
Conventional languages such as C / C++ are translated on an instruction by instruction basis and therefore fail in performance as compared to DSP native assembly language. Speed ratios between assembly and C higher than 10 can frequently be observed.
The best language for a DSP is a set of basic functional blocks. You build your application simply by connecting them together .

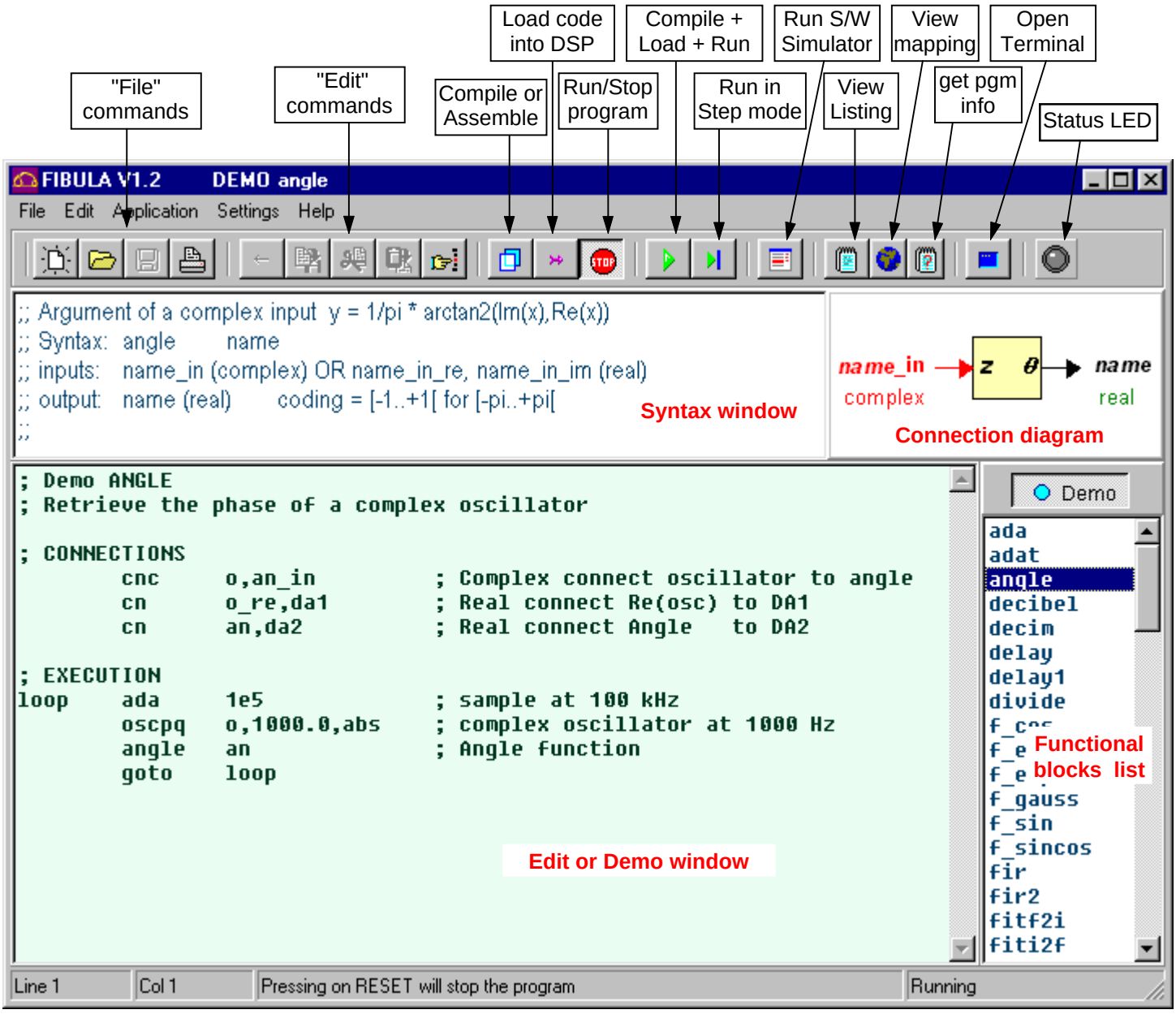


What is a functional block ?
A block groups following elements:

- Hardware resource reservations
- State variables reservation
- Constant data creation
- Initialization executable code
- Real time executable code
- Optional debug code
- Register modify report
- Documentation with demo

The FIBULA development environment

The FIBULA Integrated Development Environment is aimed to ease the code generation process and the debugging of DSP56300 programs.



The instruction list and the syntax window are available in **Fibula compiler mode.** They appear with the command **Help | Functional Blocks**.

When a demo is available for the selected instruction, this demo may be loaded in the edit window by pressing the Demo button, and may be compiled and run.

Closing the demo restores the current program.

Status Led Color	Description
GRAY	No serial comm.
DARK GREEN	DSP Ready
AQUA, blinking	Compilation active
LIGHT GREEN	Success (compilation or download)
RED	Compilation errors (syntax)
BLACK	Compiler internal problem (may be caused by cyclic références or macro infinite recursion).
PURPLE blinking	Downloading failed (Error during communication)
BLUE, blinking	DSP running
YELLOW	Step mode active.

Compiling modes:

1 FIBULA language and assembly:
In this mode, you may use a high level interconnected blocks textual description for your program, using several macros from the libraries. Assembly instructions may be used, but you must conform to the rules of the FIBULA language (naming conventions, sections where data and code reside).

2 Assembly with minimal I/O library and startup program
In this mode, your program will be assembled with an epilogue containing an INIT routine, the AD-DA analog I/O, and the SCI serial port I/O routines. Use this mode if you want to learn about the DSP assembly language.

3 Absolute assembly, no library
Use this mode if you want to check a code segment without any external interference.

Running in step mode

When compiling a program in step mode, a software interrupt is added at the end of each block, which allows the user to view on the terminal window input and output values of each executed block.
Pressing the space bar will execute the following block.
Pressing "g" (go) will run the application at it's normal speed.
Pressing "h" (halt) will return in the step mode
Pressing "f" (fast stepping) will run the program in step mode, as fast as possible, limited by the serial communication baud rate.
Pressing Escape will quit the application program and returns to the resident debugger.
If a big block is made from several sub-blocks, stepping through will execute the big block as an entity unless the debug level has been raised by one or more units.

Software simulation

If you are getting trouble while running a program written in assembly language and you want to understand the processor's behavior, you may open the software simulator. The simulation applies to the last compiled program. The simulator is a high performance Motorola product that you might have to configure to meet your specific needs. Every register and memory can be observed while stepping at the assembly instruction level. Inputs and outputs can be simulated using data files.

Opening the terminal

The terminal window displays the ASCII serial communication between the PC and the DSP card. However the terminal display function is inhibited during code downloading. You may use the terminal to manually interact with the DSP card using the resident debugger:
Viewing / modifying the memory content:
Type **x 123 <enter>** to view the content of address $000123 in the **X:** space.
Type **<space>** to go to the next address, or **"/"** to go to the previous address;
Type **.567 <enter>** to change the content to a new fractional value or
type **345678 <enter>** to change the content to a new hexadecimal value.
In the same manner, you may view / modify memory contents in the fields **Y:, P:, L:.**
Running a program:
Type **g 100** to run a program located at P:$100
If the program has been downloaded in the **.lod** format, you may use source symbols to retrieve addresses e.g. **x sine_wave <enter>** displays the variable named "sine_wave" in the source program.