1 Gnu Ada95 Cross Compiler for M68xx0

The GnuAda95 compiler is available in different versions. Two versions are installed at Computer Engineering Linux systems. One native Linux PC compiler and a Cross compiler that generates code for a Motorola 68k processor. The cross compiler is also running on a Linux system. To use the compilers /cab/ce/sw/gada68/Linux/bin must be included in path. One way to get it included is to add the following line to your .tcshrc file:

```
set path=( $path /cab/ce/sw/gada68/Linux/bin . )
```

It is also recommended (but not strictly needed) to set the following ENVIRONMENT variable:

```
GADA=/cab/ce/sw/gada68/ada-linux
```

After this, do the following command:

```
source $GADA/setall
```

These commands will normally be set in the .tcshrc file.

This paper gives a brief description on how to use Gnu Ada for cross compiling and running Ada programs on the department's Motorola 68340 microprocessors.

1.1 Compiling, linking and execution

1.1.1 Filename conventions

If a file name is given on the form *ppp.sss*, then *ppp* is it's prefix and *sss* it's suffix.

- One compilation unit (procedure, package specification or package body) per file.
- Programs (main procedure) and packet bodies are stored in files with suffix .adb
- Packet specifications are stored in files with suffix .ads
- The prefix of the file name shall be the same as the name for the compilation unit.

1.1.2 Libraries

No separate "Ada library" is needed by Gnat - it is built into the compiler. When compiling you should be positioned in the Linux directory where the program files are located.

1.1.3 Compiling

Assume that you have the following Ada program in the file *hello.adb*:

```
With TEXT_IO;
procedure HELLO is
begin
```

```
        TEXT_IO.PUT_LINE(“Hello”);
    end HELLO;
```

Observe that the name of the main program is HELLO, the same as the prefix hello. To get an executable version of the program, it have to be compiled and linked. The compilation command is named **gada68**. It takes the program name containing the Ada program as argument. To compile the Ada program in the file *hello.adb* give the command:

```
gada68 hello.adb
```

If there are errors in the program you will get error messages. You will have to correct the program and recompile.

The compile command generates an object file with the suffix .o, here *hello.o*. The object file is in machine code format (coff-m68k), but all references are not resolved, that is all program modules are not known (for example the routine PUT_LINE).

1.1.4 Linking

To get an executable version of the program, it has to be linked with the other needed program modules. This is done with the command **gbind68** with the name of the main procedure as argument

```
gbind68 hello
```

The command links the object program *hello.o* with routines from the compilers Ada library and produces a load module named *hello.x* on Motorola S-record format. Furthermore, a file named hello is produced on coff-m68k format. This file is only needed by the symbolic debugger **gdb68**.

There is another variant of gbind68 named **gbind68f**. This variant links the program with libraries located in a flash memory on the mc68 card. This reduces the load time considerably for programs that use Ada tasking. The command is used in the same way as gbind68:

```
gbind68f hello
```

1.1.5 Execution

To execute the program, it first have to be downloaded to the mc68 microprocessor. The terminal emulator **68term** is used to communicate with the microprocessor. This command have to be run in an X-window at the workstation where the microprocessor is connected. Start with the command:

```
68term
```

Now all characters that is typed in this window is sent to the microprocessor. The microprocessor have a debug monitor, *db68*, that answers with the prompt `db68:`. Write *help* to get a list of commands accepted by db68. Command that begins with the character (tilde) is interpreted by the 68term program.To download a program to the microprocessor, first give the command ~l. Now 68term answers with:

```
filename(default main.x):
```

Write *hello.x* here followed by *Enter*. The file *hello.x* is now downloaded to the microprocessor on a 38400 baud serial link. To indicate that loading is in progress the printout `loading...` is given with regular intervals. When the loading is complete the printout `loading complete` is given. To start execution of the program, give the command: `go 3000` where 3000 is the start address for the program.

1.2 Separate compilation

A program may be divided into several *compilation units* that are separately compiled. A compilation unit may be a packet specification, a packet body or a main program.

When a packet is written, the specification and the body is placed in different file (with suffix .ads and .adb). The files must be compiled in the correct order. Packet specifications must be compiled before the matching packet body. Furthermore files that reference packets in other files cannot be compiled until the referenced packet specifications are compiled. The packet bodies on the other hand, may be compiled later.

Luckily, the programmer need not do this compilations manually, because all dependencies are recorded in the Ada library and there is a command, **gnatmake68**, that compiles all the files in correct order. Thus to compile a program that consists of several files, just do:

```
gnatmake68 main
```

Here the main program is assumed to be in the file main.adb. Gnatmake68 compiles all the files that need to be recompiled in the correct order and then runs *gbind68* to link everything together to an object file named *main.x*.

If linking against the library in the flash memory is wanted use:

```
gnatmake68f main
```

This command works in the same way as *gnatmake68*, but uses *gbind68f* for linking. These commands usually requires that all files that belongs to the program are available.

1.3 Debugger

A partially Ada adapted version of gdb is available for use with the Gnat M68xx0 cross compiler. Normally *gdb* is run on the same computer as the program that is debugged. Small microprocessors thad don't have a file system is not able to run gdb. In this case 'remote debugging' may be used, that is gdb is run on a workstation that communicates with the program to be debugged over a serial link.

In our case the microprocessor have another architecture (M68k) than the Intel based workstation. In this case a cross version of gdb must be used that understands Motorola 68K code. This version is called **gdb68**.

Gdb68 communicates with the program to be debugged, using a serial link and a special protocol. The program to be debugged, thus needs access to a communication routine that understands this protocol. This communication routine is included in the *db68* debug monitor and can be activated by giving the command, **gdb**, to *db68*. The communication routine is never explicitly called by the debugged program bat is called via the exception handling (for example than a breakpoint instruction is executed). No changes in needed in the debugged program in order to use gdb68.

Assume that we want to execute the program hello.adb using gdb68. Compilation and linking is done in the usual way. After this the file *hello.x* need to be downloaded to the microprocessor using **68term** (There is a load command in gdb68, but this do not work). Then loading is finished *db68* answers with the prompt `db68:` We now give the following commands in order to activate the communication routine and start the execution of *hello*:

```
db68: gdb
```

```
db68: go 3000
```

Because the gdb command have placed a breakpoint in the beginning of the program (at address 0x3046) the communication routine is called. This routine gives the printout `$S05#b8` and waits for commands from **gdb68**. We now terminates 68term using ~x (or CTRL D) and gets prompt from the work station. Gdb68 is started with the command:

```
cepc224> gdb68 hello
```

After a few initial printouts gdb68 answers with a prompt (gdb). We now have to tell gdb68 that we want to use a serial link with the command:

```
(gdb) target remote /dev/ttyS0
```

Usually gdb68 answers with:

```
Remote debugging using /dev/ttS0

0x3046 in copydata ()

(gdb)
```

From now on the same commands are used as when debugging an Ada program executing on the work station.

Usually the debugger is able to keep track of which programming language the program uses, but sometimes it has to be informed that it is an Ada program that is debugged. This is done with the command:

```
(gdb) set language ada
```

If you want to set a breakpoint that allows you to single step the program you write:

```
(gdb) break hello
```

Breakpoints can also be set in other places in the program, for example in the beginning of a subroutine:

```
break subroutine_name
```

Subprograms declared in a package are named packetname.subprogramname.Exceptions can be trapped by the following breakpoint:

```
(gdb) break __gnat_raise
```

(Observe two underline characters)

The commands ’step’, ’next’ and’cont’ can be used to single step and continue the execution. When the program stops information on where the stop is and which calls that are active is given by the command:

```
(gdb) where
```

Before the debugger is terminated it good to give the command ‘kill’. It makes the microprocessor return to the db68 monitor. If this command is not given, reset have to be pressed at the microprocessor box to get t running again. The command ‘q’ terminates gdb68. There are many more commands to gdb68. To get information write:

```
(gdb) help
```

Short command list

Command	**Comment**
gada68 *name*.adb	Compiles
gbind68 *name*	Links
gbind68f *name*	Lines with flash library
gnatmake68 *name*	Compiles all files needed by *name*
gnatmake68f *name*	As gnatmake68 but links with flash library
gnatls68 -v *name*	Lists library information for *name*
gclean	Remove temporary files