Real Time Ada Compilers for the 68020

L. Asplund, M. Carlsson, D. Wengelin and G. Bray.
University of Uppsala

Abstract
Three cross compilers hosted on a VAX/VMS for the Motorola 68020 processor have been evaluated. The target machine, a VME system with a MVME133 processor card and a MVME340 digital IO-card, has been set up in an environment with a function generator and an oscilloscope in order to measure the real time response. The compilers supply the run time kernel. It is possible to use Ada for real time applications, with the use of optimization of accept statements task switch times can come down to the order of 15 – 20 μs .

Introduction

Ada has evolved drastically the last years. Soon after the language became an ANSI standard, only a few compilers were available and they could only be used to compile various simple programs. Many enthusiasts started out with nested generic packages and task types, only to find that the validated compilers were valid only for the test suite. The situation for the users of the first compilers was to spend a lot of time finding out where and why the compiler failed, either at compile or at run time.

The next step in the evolution came with what can be called production compilers for numeric and data handling programs. These compilers could replace most of the existing Pascal and FORTRAN compilers. Tasking could be used for moderate time critical applications. Still the target machines were the same as the host, such as the VAX. Several users and companies could now start to learn the language and start to develop tools.

At that time Ada could replace most other languages for many applications. The language was primarily designed for real time applications and especially for embedded systems. The latest development in Ada compilers are cross compilers for quite powerful microcomputers like the Motorola 680X0-family, Intel 80X86 and National 32X32. Hopefully there will be compilers for smaller micros in the future.

In a project, aiming at distributing Ada software to a loosely coupled multi computer system, three independent VME systems, with cpu, memory and Ethernet port, will be used/1/. The whole system will be programmed in Ada and it is therefore essential to choose a cpu-board and a compiler that supports low level programming and allows an efficient mapping of hardware interrupts.

Cross compilers

The three cross compilers that have been tested are the Alsycomp_011 from Alsys/2/, the System Designers Ada-Plus version 3A.00 /3/ and TeleGen2 /4/ from Telesoft/TeleLOGIC. Cross compilers from VAX/VMS to Motorola 68000 not tested here are available from Rational and Tartan.

Software tools and development support

The environment support is good enough for program development. There are, however, problems when the hardware and software do not match. If spurious interrupts occur, the system may hang, making it impossible to restart the program or even get to the kernel. The kernel was downloaded into the target prior to the loading of the compiled and linked code. There is, of course, the possibility to use EPROMs for the kernel. During this evaluation, it was most convenient to download the kernel.

The environment is essential when developing programs on one computer to be run on a separate target. However, the tests described in this article did not require any debugging.

Hardware configuration

Fig 1 shows the hardware setup for the test of the Alsys and the SD compilers. A

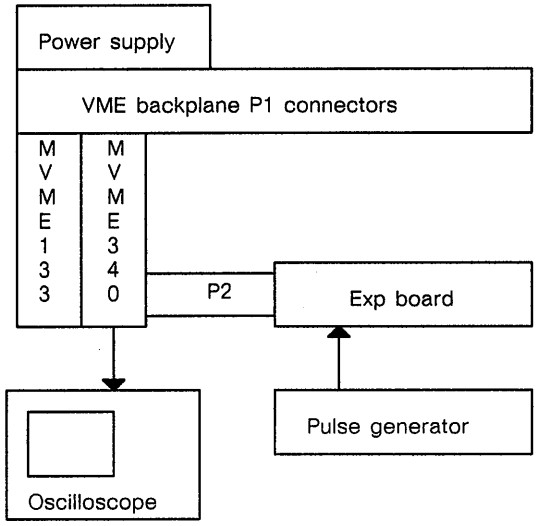

Fig 1

VME-crate (Electronic Solutions, Series 10) with 12 slots and a backplane for P1 only, with a MVME133-1 processor card. This board includes a MC68020 cpu with 16.67MHz clock frequency and one wait state. The board also comprises a real time clock, 1Mbyte of onboard memory and two serial ports. In two of the ROM-sockets a simple debugger is installed (133-BUG). With this debugger, a kernel can be loaded via a

serial link. This link is then used to download the compiled and linked code. For the tests with the Telegen2 compiler a MVME135-1 card was used running at 20MHz with zero wait states. All times are given as measured, but these times can be increased by a factor of 25% to compensated for the difference in clock frequency, but this would then be an overestimate due to the VME-timings. The MVME133 card was first rented and two of the compilers tested. The tests with the TeleGen2 compiler was performed after the other two, using the MVME135-1 board.

In these tests an external interrupt source was a pulse generator, which was connected via a board to an IO-card, MVME340.

The MVME340 board has three independent IO circuits, PI/Ts (Parallel Interface and Timer chip, MC68230), and circuits for interrupts. The memory layout for each of the three PI/Ts is shown in fig 2. Observe that all registers are 8 bit wide and the offset

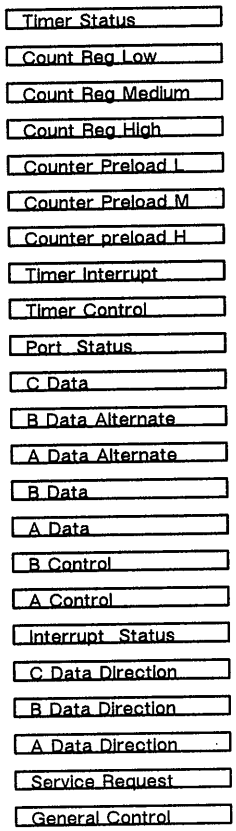

Fig 2

for these are at multiples of 2. The first of the three starts at the base of the MVME340A. The second starts at an offset of 1 from the base address. These two first are interleaved. The third starts with an offset 16#81#. Observe that this PI/T is not interleaved with a fourth PI/T leaving the memory locations between these

registers empty. Fig 3 shows the memory layout for the three different PI/Ts.

Apart from these three circuits, the MVME340 card has three general registers controlling the board. One of these registers defines the interrupt priority of the card and controls a light emitting diode at the front. This LED is used in all the different tests. It can be accessed both from assembler and Ada.

Software description of hardware

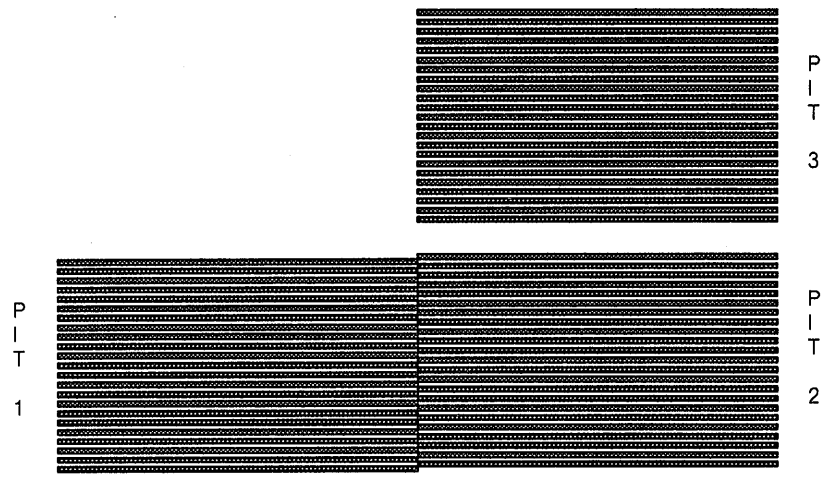

Fig 3

Fig 3 shows the hardware map of the various registers in the MVME340. The Ada code below show the specification of one of the PI/Ts. With a representation specification the mapping can be made to correspond to the hardware as shown in fig 4. Observe that there are three different 8 bit wide registers (A, B, and C) and that it would have been better if the hardware had been made in such a modular way that these three could have been represented by:

```
Type Data_Register is record
        Direction,
        Control,
        Data,
        Alternate : Byte;
 end record;
```

It would then have been possible to specify the three PI/Ts in terms of these, but unfortunately the hardware layout of register C differs from register A and B.

The compilers handle the type Byte in different ways. For the SD compiler it was possible to directly use

```
Type Byte is range 0..255;
For  Byte'size use 8;
```

This gave an eight bit variable. For Alsys there was no way of defining the size. The workaround was to use a predefined type Short_Integer.

In one of the routines, bit seven should be set and for clarity (in the absence of bit specification) the following code should be used:

```
        Some_Variable : Byte;
    begin
        Some_Variable := 2#1000_0000#;
```

but for Alsys the work around is

```
        Some_Variable := -128;
```

The TeleGen2 has no type like Short_Integer and the representation specification for 'size could not be used. In order to access only one byte on the card and not sixteen bits, which will cause a bus error due to the upper eight bits not being available, the following specification were necessary:

```
        type Byte is range 0..255;
        type Byte_Record is record
          Data : Byte;
        end record;
        For Byte_Record use record;
          Data  at 0 range 0..7;
        end record;
        Some_Variable : Byte_Record;
    begin
        Some_Variable.Data := 2#1000_0000#;
```

The specification of the records defining the PI/T was, for all three compilers, written as in fig 4.

```
Type PIT_Record is record
   General_Control,
   Service_Request,
   Data_Direction_A,      Data_Direction_B,     Data_Direction_C,
   Interrupt_Status,
   Control_A,             Control_B,
   Data_A,                Data_B,
   Alternate_A,           Alternate_B,
   Data_C,
   Status,
   Timer_Control,
   Timer_Interrupt,
   Counter_Preload_3,     Counter_Preload_2,    Counter_Preload_1,
   Count_3,               Count_2,              Count_1,
   Timer_Status
                          :Byte;
end record;
```

Fig 4 record type describing a single PI/T chip. Compare fig 2.

The components of the record are located at every second byte. The record specification in fig 5 is used.

```
for PIT_Record  use record at mod 8;
   General_Control      at 16#00#  range 0..7;
   Service_Request      at 16#02#  range 0..7;
   Data_Direction_A     at 16#04#  range 0..7;
   Data_Direction_B     at 16#06#  range 0..7;
   Data_Direction_C     at 16#08#  range 0..7;
   Interrupt_Status     at 16#0A#  range 0..7;
   Control_A            at 16#0C#  range 0..7;
   Control_B            at 16#0E#  range 0..7;
   Data_A               at 16#10#  range 0..7;
   Data_B               at 16#12#  range 0..7;
   Alternate_A          at 16#14#  range 0..7;
   Alternate_B          at 16#16#  range 0..7;
   Data_C               at 16#18#  range 0..7;
   Status               at 16#1A#  range 0..7;
   Timer_Control        at 16#20#  range 0..7;
   Timer_Interrupt      at 16#22#  range 0..7;
   Counter_Preload_3    at 16#26#  range 0..7;
   Counter_Preload_2    at 16#28#  range 0..7;
   Counter_Preload_1    at 16#2A#  range 0..7;
   Count_3              at 16#2E#  range 0..7;
   Count_2              at 16#30#  range 0..7;
   Count_1              at 16#32#  range 0..7;
   Timer_Status         at 16#34#  range 0..7;
end record;
```

Fig 5 representation specification of the record in fig 4.

The three individual PI/Ts were declared in the following way for the three different compilers.

```
Alsys:
PIT1 : PIT_Record; for PIT1 use at To_Address (16#200_000#);
PIT2 : PIT_Record; for PIT2 use at To_Address (16#200_001#);
PIT3 : PIT_Record; for PIT3 use at To_Address (16#200_041#);

SD:
PIT1 : PIT_Record; for PIT1 use at Convert_Address ("200000");
PIT2 : PIT_Record; for PIT2 use at Convert_Address ("200001");
PIT3 : PIT_Record; for PIT3 use at Convert_Address ("200041");

TeleGen2:
PIT1 : PIT_Record; for PIT1 use at Addr (16#200_000#);
PIT2 : PIT_Record; for PIT2 use at Addr (16#200_001#);
PIT3 : PIT_Record; for PIT3 use at Addr (16#200_041#);
```

```
package MVME340 is
   procedure SEQ;
   procedure REQ;
   procedure Int_Enable;
   procedure Int_Disable;
   procedure Reset_Interrupt;
   --...
end MVME340;
```

Fig 6. Package specification for IO-board.

In a package, MVME340, the board is described using these record definitions. Some of he visible routines are shown in fig 6. There are routines to enable and disable the

interrupt. The previously mentioned LED can be accessed using the routines SEQ and REQ to set and reset it respectively.

Programs

The programs used to evaluate the compilers are short and can therefore be included in this report. The most important thing to measure is the latency time for interrupts. This has been measured in two different ways.

Ada is very heavily standardized. The Language Reference Manual (LRM) /5/ gives small opportunities to depart from the standard. Chapter 13 is devoted to implementation dependent features and representation specifications. Here we now have three different compilers with identical host and target and no two of them handle addresses in the same way. The type address given in package system is for all declared as private. For SD a routine Convert_Address is supplied. This function takes as argument a string, which has to be a sequence of hexadecimal numbers, not a based integer literal as defined in Ada. The corresponding function for Alsys is the function To_Address which as its argument requires a Long_Integer. The TeleGen2 can convert from Long_Integer via the procedure Addr.

The Alsys compiler maps a task entry to an interrupt source by a call to the run time system. The following code is used:

```
specification:
  entry INTERRUPT;

body:
   INTERRUPT_HANDLER.INIT (SYSTEM.TO_ADDRESS( 16#100#));
  accept INTERRUPT do
```

If several entries are valid for the task, the mapping is to the first entry only. The first part of the interrupt code has to be written in assembler. In the assembler code, the interrupt has to be cleared. Then with a call to the run time system a rendezvous is made with the task.

The SD compiler declares the interrupt according to chapter 13 in LRM. An entry in a table that describes all interrupt vectors had to be modified. This table is written in assembler.

```
specification:
  entry INTERRUPT;
  for Interrupt use at system.Convert_Address("100");
```

The TeleGen2 compiler from TeleLOGIC has a package called Interrupt where a private type, Descriptor, is declared. An object of this type can be initialized by a call to the function Source, where the hardware interrupt vector is defined as a parameter. The mapping of the task entry is done to the variable of type Descriptor. No assembler programming was necessary.

```
MVME340_Device : Interrupt.Descriptor := Interrupt.Source (Addr (16#100#));
...
entry INTERRUPT;
for INTERRUPT use at MVME340_Device'Address;
```

The first program given in fig 7 incorporates a select statement but only with an **or terminate** alternative. A task, that shall handle interrupts, is, in a data acquisition program, responsible for reading in external data when an interrupt occurs. Such a task needs, besides the interrupt entry, others entries to obtain information data that has been read in. The way to solve this is by the use of a **select** statement.

```
with MVME340, SYSTEM, INTERRUPT_HANDLER;
procedure Int_Entry is
   task HANDLER is
      pragma PRIORITY(10);
      entry INTERRUPT;
      for Interrupt use at compiler specific.
   end HANDLER;
   task body HANDLER is
   begin
      loop
         select
            accept INTERRUPT do
               MVME340.SEQ;
               MVME340.Reset_Interrupt;
            end Interrupt;
            MVME340.REQ;
         or
            terminate;
         end select;
      end loop;
   end HANDLER;
begin
      MVME340.Int_Enable;
      delay 60.0;
      MVME340.Int_Disable;
end Int_Entry ;
```

Fig 7.

The code shown in fig 7 is the code used, with minor modifications for the three compilers. The entry Handler.Interrupt is, by the different techniques given by the compilers, defined to handle the hardware interrupt from an external source. When such an interrupt occurs, the do part of the accept will set the LED on the front panel, reset the interrupt (by reading a register on the board), and then reset the LED. The Alsys compiler did the reset of the interrupt in assembler. On the oscilloscope the time is measured from the negative edge of the pulse to the LED being turned on.

Since the select statement takes some time to handle, another test was made. This test measures the time to enter a task, reset the interrupt request, and return. The code is shown in fig 8. The accept statement for Alsys does no require any do part, as the interrupt is cleared in assembler.

```
with MVME340, SYSTEM, INTERRUPT_HANDLER;
procedure Int_Task is
   task HANDLER is
      pragma PRIORITY(10);
      entry INTERRUPT;
      for Interrupt use at compiler specific.
   end HANDLER;
   task body HANDLER is
      begin
      loop
         Alsys:
         accept INTERRUPT;
         SD:
         accept Interrupt do
            MVME340.Reset_Interrupt;
         end Interrupt;
      end loop;
   end HANDLER;
begin
   MVME340.Int_Enable;
   loop
      MVME340.SEQ;
      MVME340.REQ;
   end loop;
   MVME340.Int_Disable;
end Int_Task ;
```

Fig 8

The procedures in fig 7 and 8 measure the time for interrupts. These involve task switches that are tested separately in the procedure shown in fig 9. With an interval of 0.01 seconds, all the chained tasks are called. The activity is measured with an oscilloscope. After 100 of these chains another task is added. This program was used with all compilers without rewriting. Two of the compilers (SD and Alsys) gave a warning message, that the construction could lead to a task calling itself.

```
with MVME340;
procedure RendezV is
   type Mail;
   type Mail_Pointer is access Mail;
   task type Mail is
      entry Init (Next : Mail_Pointer);
      entry Request;
   end Mail;
   Root : Mail_Pointer;
   task body Mail is
      My_Next : Mail_Pointer;
   begin
      accept Init (Next : Mail_Pointer) do
         My_Next := Next;
      end Init;
      loop
         select
            accept Request;
            MVME340.SEQ;
            MVME340.REQ;
            if My_Next /= Null then
               My_Next.Request;
            end if;
         or
            terminate;
         end select;
```

```
      end loop;
   end Mail;
begin
   loop
      declare
         Temp : Mail_Pointer := new Mail;
      begin
         Temp.Init (Root);
         Root := Temp;
      end;
      for i in 1..100 loop
         delay 0.01;
         Root.Request;
      end loop;
   end loop;
end RendezV;
```

Fig 9. Procedure to measure simple rendezvous times.

In the project where one of these compilers will be used, heavy numerical calculations will be performed. The following two procedures measure the speed of floating point operations. The first test procedure used a simple Taylor expansion of sine. The second procedure makes use sine from a math-package, available in the Alsys and TeleGen2 compilers.

```
Function SIN (Arg : Float) Return Float;

With MVME340;
With Sin;
Procedure Cosine is
   Level : Float := 0.00;
   Step  : Float := 0.01;
begin
   loop
      For Omega in 0..627 loop
         If Sin(Float(Omega)/100.0) > Level then
            Level := Level + Step;
            MVME340.SEQ;
         else
            Level := Level - Step;
            MVME340.REQ;
         end if;
      end loop;
   end loop;
end Cosine;
```

Fig 10. Code for evaluating floating point performance

Results

All measurements are performed with an oscilloscope, which is connected to the LED at the front. The time to switch this LED on and off are 0.8 µs in assembler. In fig 8 the main program is an endless loop where the LED is switched on and off. The times to Set or Reset the LED and handle the loop is

Compiler	SEQ/REQ	loop
Alsys	4 µs	1.5 µs
System Designer	12 µs	3 µs

TeleGen2	4.5 us	1.5 µs

The programs listed in figs 7 and 8 were used to measure the response times for interrupts. In the setup shown in fig 1, a pulse generator is applied to the MVME340 and and oscilloscope to the LED at the front of the card. In this way, it is possible to come close to a realistic use.

The interrupts of the MVME340 have to be reset by a reading one of the data registers on the board. For the Alsys compiler this read operation has to be done in an assembler routine before entering the Ada runtime. The time taken to enter the select statement in fig 7 is

Alsys	310 µs
System Designer	132 µs
TeleGen2	18 µs

The TeleGen2 gives the value 18 µs when the Pragma Function_Mapping is used, which restricts the code within the do part from making a rendezvous with other tasks or access dynamic variables (those on the stack).

The program in fig 7 contains an accept statement with an **or terminate** alternative. These take more time than a pure accept statement. In order to test the latency for such a construct, the procedure in fig 8 is used. The total time spent in the task is

Alsys	240 µs
System Designer	380 µs
TeleGen2	28 µs (with Pragma Function_Mapping)
TeleGen2	225 µs

This test now shows the fastest way of context switching when hardware interrupts are involved. The program with no other entries has limited practical use.

The procedure given in fig 9 measure the time to do a regular rendezvous with no argument. The time between successive rendezvous is

Alsys	200 µs
System Designer	170 µs
TeleGen2	125 µs

The number of task that could run in was different for the various compilers, but no optimization was done to increase the number.

The program in figs 10 tests the floating point performance. With the Alsys and TeleGen2 compilers, a predefined math package is supplied. A second test was made where this package was used. For the SD compiler only the procedure utilizing the Taylor expansion of sine was used. The effect of the programs will be a cosine function over the LED. The frequency of the intensity function was measured.

In the case of Alsys there is a choice of using the hardware coprocessor 68881 or to

do the arithmetic in software. The period times of the cosine are given in table 11.

	Function sine	Function Math.sine
Alsys with HW float	235 ms	315 ms
Alsys with SW float	330 ms	480 ms
System Designers	192 ms	Not applicable
TeleGen2	110 ms	54 ms

Fig 11

Future

The results from these tests form a basis for the next step in the project. One of the compilers will be purchased and the results from distributing Ada on a Vax running VMS will now be implemented on three separated VME-systems. The distribution methods are to be implemented on a missile and guidance control software system. The problem with many Ada Run time designers is probably that they regard even interrupt routines to be handled of this run time system. For time critical interrupt handlers there are no time more than to read ore write data to external hardware.

Acknowledgments

This study has been made possible by FOA, as part of a study of multi-processor systems. Thanks are given to Joseph Nordgren for allowing the extended use of the hardware.

Thanks also to Olle Hansson at TeleLOGIC for the support end help, to Magnus Eriksson and Örjan Lehringe at MariaData (Alsys) for making the compiler available as a loan and for their assistance.

References

1 Distributed Ada Run-Time System, M. Carlsson *et al.* Draft.
2 Reference Manual for the Alsys compiler Alsy_Comp_011
3 Reference Manual for System Designers compiler Ada-Plus
4 Reference Manual for the TeleGen2 compiler
5 Ada Language Reference Manual