Using Ada in non-Ada systems

Industrial Presentation

Ada-Europe 2018

White Elephant GmbH

Experience report on incorporating packages written in Ada into predominantly non-Ada microprocessor based systems.

Why aren't we already using exclusively Ada?

Because in the late eighties and early nineties Ada compilers were:

- Rare – especially for microprocessors
- Slow
- Required vast resources
- Expensive

So we used Modula-2

- Invented by Niklaus Wirth.
- ISO standard 10514

Pascal based, shares many features with Ada

- Verbose, non ambiguous syntax
- Specification and implementation are separately compiled units

Originally we used :

- Cross compilers
  - Now we translate the Modula-2 into C and then compile that to target
- Wind River Diab tool chain
  - Now we use GCC
- M68332 & Motorola Coldfire
  - Now we use ARM

The GCC we use to compile our C for ARM is the same compiler that we use to compile our Ada for ARM.

Namely v6.3.1 as distributed by AdaCore as their GPL 2017

Because we use the same compiler their objects are compatible

So, at least in theory, we can write a program that is a mixture of Ada and Modula-2 / C

Except:

Our existing system has a run-time and
so does Ada

Probably not a good idea for an application to have more than one run-time so either:

1. Ada uses ours
2. We use Ada's
3. Restrict Ada so that it doesn't require a runtime

The latter is generally known as the Zero Footprint profile for Ada (ZFA)

Implemented by :

- Pragma restrictions in the file System.ads
- Whether or not certain specification files can be found in the search path.

ZFA defined by restrictions in the file System.ada

- pragma Restrictions (No_Exception_Propagation);
- pragma Restrictions (No_Elaboration_Code);
- pragma Restrictions (No_Implicit_Dynamic_Code);
- pragma Restrictions (No_Finalization);
- pragma Restrictions (No_Tasking);
- pragma Restrictions (No_Delay);
- pragma Discard_Names;

These restrictions mean we lose Ada features.

- Tasks
- Protected Objects
- Controlled Types
- The *delay* statement
- Dynamic storage allocation using *new*
- Exception propagation

Additional voluntary restrictions:

pragma Restrictions (No_Floating_Point)

Because our target has no hardware support for floating point

pragma Restrictions (No_Seconday_Stack)

Because our target has very little RAM

Without a secondary stack we cannot write functions that return an unconstrained type.

This means that functions can’t return Strings nor can we use ‘img or ‘image

Originally chose not to support elaboration.

Implemented by
pragma restrictions (No_Elaboration_Code);

Without elaboration:

- Global variables can only be initialised to values known at compile time
- No package bodies
- Pre-elaborated packages can only call other pre-elaborated or pure packages

Still useful?

Even a very reduced Ada is better than Modula-2 and presumably is much much better than writing in C!

For example:

- Named parameters
- Named fields in constructors
- Private types, functions and procedures
- Bit level specification in representation clauses

A simple example.

C calling a parameter-less Ada procedure

Ada name mangling:

- package name + “_” + “_” + procedure name
- Name entirely in lowercase

C declaration:

```
extern void adaunit__adaprocedure(void);
```

C call:

```
adaunit__adaprocedure();
```

Then…

- Compile Ada
- Compile C
- Link together to form an executable

That’s all you need because ZFA doesn’t require any libraries.

Some language features require the package specification but a body isn't required.

Because the implementation is intrinsic (built-in) to the compiler

For example:

ada.unchecked_conversion

Requires the file a-unccon.ads to be found in the path.

Note the “crunched” name!

Historical throwback?

Others we needed were:

- a-except.ads = ada.exceptions
- interfac.ads= interfaces
- s-maccod.ads = system.machine_code
- s-stoele.ads = system.storage_elements
- s-unstyp.ads = system.unsigned_types

Debugging

-dwarf-3

Standard DWARF debugging information

Can continue to use our own tool but…

Needed to be enhanced to support objects with a size and position not necessarily a byte multiple.

Link time optimisation

-lto

In-lining over units

Unfortunately loses original name of compilation units.

Solution = Always name mangle (even if local)

Fortunately this is what Ada does!

Exception Handling

Even simple code requires exception handling

Without a runtime exceptions must be handled locally

Effectively a jump.

Consider:

```
X : Natural;
X := X + 1;
```

This can raise constraint_error

We could provide a last chance handler but instead we chose to either:

1. Rewrite the code so that the exception can’t happen
2. Catch and handle the exception locally

Tip:

Use switch –gnat.x

Warns if implicit or explicit exception not covered by local handler.

But…

The current compiler v6.3.1 doesn't always get it right ☹

Example:

```
type Handler is access procedure;
The_Handler : Handler;

procedure Test is
begin
   if The_Handler /= null then
      The_Handler.all;
   end if;
end Test;
```

The compiler warns that an exception can be raised – which in fact it obviously can't!

Two solutions:

1. Disable warnings
   - Still links. Which shows that an exception wasn't really generated
   - But Disable/Enable warnings is messy
2. Use Pragma supress
   - Hope that a future compiler will one day warn us that the pragma is superfluous

Example:

```
type Handler is access procedure;
The_Handler : Handler;

procedure Test is
   pragma Suppress (Access_Checks);
begin
   if The_Handler /= null then
      The_Handler.all;
   end if;
end Test;
```

Global variables

Hard not to use globals

Often need to be initialised

Could use an initialisation routine and hope that we remember to call it (error prone)

However if the value is known at compile time then because Ada uses the same convention as C, Ada global variables are initialised by the same mechanism as C variables are.

For details on how this works see follow-up article in the Ada User Journal

Which members of Ada-Europe receive as part of their membership.

Visit our web site
www.ada-europe.org/join

And leave your contact details

But what if the value isn’t known at compile time?

This type of initialisation is performed by elaboration code

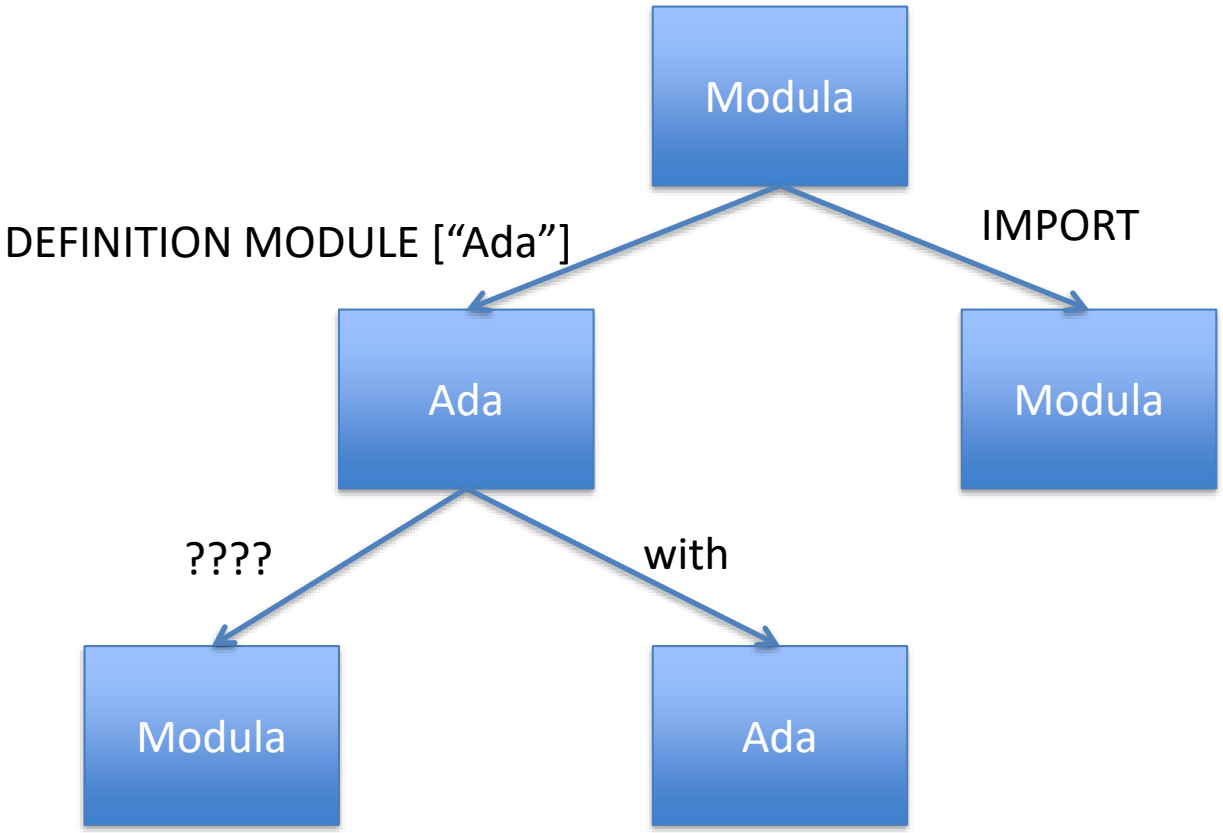
We originally hoped that we could do without elaboration

But… Modula elaborates it‘s module bodies and restricting our Ada to only calling Modules without bodies would have been too prohibitive.

Ada packages that require elaboration generate routines named __elabb or __elabs

All you need to do is call them…

In the right order.


Modula
DEFINITION MODULE ["Ada"]
IMPORT
Ada
Modula
????
with
Modula
Ada

```
Pragma Modula_Import (ModulaUnit);
```

-gnatwG

Suppress warnings on unrecognised pragmas

However the Ada RM requires that such warnings appear.

Therefore our IDE had to take over the detection of unrecognised pragmas

Because only it knows the names our our new pragmas

```
extern void __attribute__((weak)) ModulaUnit_BEGIN(void);
extern void __attribute__((weak)) adaunit___elabb(void);

typedef void (*Unit_List[ 1])(void);

static const Unit_List Unit_Body_the_list = {
   ModulaUnit_BEGIN,
   adaunit___elabb};
```

Interrupt Routines

ARM interrupt routines are simply parameter-less procedures whose address is placed into the vector table.

~~~~
procedure Interrupt_Handler
with attach_handler (36);
~~~~

```
procedure Interrupt_Handler with Export;
pragma Use_Vector (36);
```

Embedded Assembler

```
with System.Machine_Code; use System.Machine_Code;

procedure Disable_Interrupts with Inline is
begin
   Asm ("msr primask, %0;",
        Inputs   => Integer'asm_input ("r", 1),
        Clobber  => "memory",
        Volatile => True);
end Disable_Interrupts;
```

Summary

Does it work?

Yes.

We converted two ARM based projects:

1. Hard real-time motor control program used to position an astronomical telescope.
2. Modem used to relay IP and serial communications over high voltage power lines.

Problems

- INC / DEC
- ‘size is in bits
- Modula-2 expressions from left to right
- Can’t modify “in” parameters

Questions?