Ada in Embedded Boards for Scientific and Medical Instruments

Robert C. Leif
Ada_Med, a Division of Newport Instruments
5648 Toyon Road
San Diego CA 92115
(619)582-0437
E-mail rleif@rleif.com

Suzanne B. Leif
Ada_Med, a Division of Newport Instruments
5648 Toyon Road
San Diego CA 92115
(619)582-0437
E-mail rleif@rleif.com

1. ABSTRACT

The combination of Ada's new class-wide programming with tagged types, generics, and representation clauses for both enumerated and record types greatly facilitates low level programing. A generic board register class was extended to represent the specific hardware and provide high level abstractions for reading and changing the states of the hardware registers. Subprograms included in this generic board register class include functions and procedures which address these registers by name and employ high-level syntax for bit manipulation. The use of these objects derived from the register class permits the development of easily understood, maintainable software for computer boards which control and acquire data from devices including scientific and medical instruments. A software library providing these and other relevant functionalities and an application with a commercial 100 megahertz scaler board for a PC will be described.

1.1 Keywords

Object Oriented Programming, OOP, Device, Board, Low-level, Ada, Ada 95, Embedded system.

2. PROJECT DESCRIPTION:

2.1 Chemistry:

A europium containing compound (Vallarino-Leif Quantum Dye)[1],[2] is excited for 10 microseconds with ultraviolet light at 370 nanometers and emits in the red at 618 nanometers. The europium Quantum Dye has a luminescence half-life of about 300 microseconds. Since solvents and additives effect both the quantum efficiency and luminescence lifetime, it is necessary to measure both.

2.2 Hardware:

An ultrasensitive, relatively inexpensive, time gated, single photon counting luminometer has been constructed (Figure 1). The photon counter is an integrated red sensitive photomultiplier module. The luminescence is excited with a commercially available xenon flash lamp. Since a series of at least 10 data points are required to characterize the shape of the luminescence decay and the half life is approximately 300 microseconds, the time between readings should be less than 30 microseconds. Often, in order to achieve an acceptable signal to noise ratio, a series of measurements must be averaged. The capacity to average up to 500 series of measurements has to be included in the system.

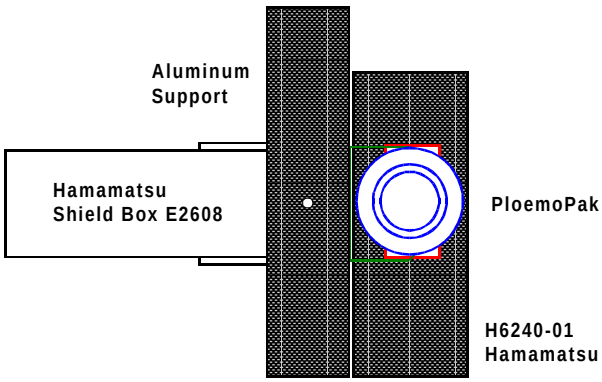

Figure 1. Luminometer Layout. The Aluminum Support (shown shaded) holds and aligns all components. The xenon flash lamp with an ellipsoidal reflector is mounted in the Shielded Box E2608. The PloemoPak, which contains the excitation filter, the dichroic mirror, and emission filter is co-linear with the E2608 and is mounted directly above the H6240-01 photon counter.

Ideally for real-time, high speed data acquisition, the use of specialized data acquisition hardware should be maximized and the use of software minimized. Unfortunately, no reasonably priced scaler with either a FIFO cache or direct memory access for data transfer was available. Real-time computer control of the interface board was the only option. An Advanced Research Instruments Corp. PC100D Dual Counter Timer PC Board (Boulder CO), which effectively has a 100 MegaHertz scaler was purchased. The high counting rate provides the dynamic range required for bright samples. Data losses are minimized because in Alternating Mode one of the two counters in the PC100D can be read while the other is counting. The scaler and light source must be synchronized either in hardware or an algorithm developed to measure the first data point, which should be obtained within 30 microseconds of triggering of the flash.

The development of both research and clinical laboratory chemistries almost always is a multi-discipline undertaking. For instance, the development of the software described below was motivated by the need to build a new research instrument (hardware), which in turn, was required for the development and characterization of a new class of compounds. The major goal of the study of these compounds is to develop very sensitive assays for HIV and similar analytes. Ada 95 is a very good, forgiving language, which helps individuals who are educated in fields other than computer science to create useful, reliable, efficient software:

The major constraint was to minimize data loss. Other significant constraints were: one person was the entire software and hardware engineering staff; both unit and development costs had to be minimized; and finally, the description of the PC100D board was primarily given as example C and BASIC programs. Microsoft DOS was selected as the operating system because it has one major advantage. It does nothing. It only causes a minimum number of interrups. And at present, no Ada compiler is available for Microsoft Windows CE. Aonix ObjectAda was selected because it supports the PharLap 32 bit extensions to DOS, Aonix supports DOS, and the Aonix semantics employed for machine code insertions are simpler than those employed by GNAT, which are based on GNU.

The 7 eight bit registers of the PC100D board are described in Table 1. Each of the 2 counters and preset timers is individually addressable. The Control Register starts and resets the Counters and Timers. It also loads 8 bit data from each counter into the output register, which is sequentially read by the software. The Mode_Interrupts_Enable register sets the counting mode: Two Independent, Two Simultaneous or Two Alternating; it also enables the interrupts for each of the Timers. The Status register reports the counting mode, whether each counter is busy and the whether the timer interrupts are enabled. Read and write register pairs which have entirely different functions share the same port address.

Table 1 : PC100D Registers

Child packages of PC100	Port_Object_ Name	Direc-tion	Base Addr. Offset	Port Num.
Counters	Counter_1	Read	0	1
“ “	Counter_2	Read	1	2
Preset_Timers	Timer_1	Write	0	1
“ “	Timer_2	Write	1	2
Control_Reg	Control_Register	Write	2	1
Status	Status	Read	2	1

Table 1 : PC100D Registers

Child packages of PC100	Port_Object_ Name	Direc-tion	Base Addr. Offset	Port Num.
Counters	Counter_1	Read	0	1
“ “	Counter_2	Read	1	2
Preset_Timers	Timer_1	Write	0	1
“ “	Timer_2	Write	1	2
Control_Reg	Control_Register	Write	2	1
Mode_Interrupts_ Enable	Mode_Interrupts_ Enable	Write	3	1

2.3 Example of Vendor’s software:

```
// Start counting (Set Mode 3)
 outportb(WRT_MODE, 3);

 // Determine which counter is busy.
 // Read status register.
 // If bit 4 = 1, Counter 1 is busy.
 // If bit 5 = 1, Counter 2 is busy.
 status = inportb(RD_STAT);

 if(status & 0x10)
  // bit 4 is 1
  goto COUNT1_DATA;
 else
 {
  // bit 4 is 0
   if(!(status & 0x20))
  {
   // bit 5 is 0
   fprintf(stderr, “Error! Both
 counters stopped.\n”);
   goto END;
  }
  else
   // bit 5 is 1
   goto COUNT2_DATA;
 }
```

2.4 Equivalent Ada Main Procedure:

```
Pc100_Board.Read_Control_File(
 Pc100_Board.Control_File_Name_W_ext_bd
 , Control_Values =>
 Pc100_Board.Control_Values);

Pc100_Board.Reset_Pc100_Counters;

Pc100_Board.Set_Timer_1;

Pc100_Board.Read_Data_2_alternating
 Counts_Array =>
```

```
  Pc100_Board.Counts_Array,
 Control_Values =>
  Pc100_Board.Control_Values);
```

Admittedly, this is a comparison between apples and oranges. The Ada main procedure calls Ada packages, which themselves are based on layered abstractions. However, the final software employed by the application programmer can be very similar to the Ada example above. The details of how the Ada example above was created will now be described. The C software encodes both the Mode and Status as integers. In fact, it is virtually unreadable without extensive comments and the Ada does not have or require any comments except those in the individual packages which describe the purpose and, if necessary, the algorithmic structure of the subprograms. The source text of the main procedure is a stylized version of the following verbal description of the events. The values used to control the board are read from a previously created Control_File. The counters are reset; and, the Timer is set (to the value in the Control_Values). The data is then collected in alternating mode and stored in an array.

3. Software architecture:

The software has been split into two parts: a set of Ada_Utilities, which are general tools, and a part specific for computer interface boards, particularly the present PC100 board. Both parts were designed to maximize reuse including future commercialization.

3.1 Ada_Utilities:

Two of the Ada_Med Ada '83 library packages, Strings and Num_Types, were reused. They were previously employed for the commercially available QC-Tracker Flow Cytometry quality control program.[3] The original Num_Types and Strings packages were derived from the book, ADA in Action.[4] They have been updated for Ada 95 and significantly extended. In order to maximize the reliability of the software, bounded strings were substituted for strings. This permitted the elimination of the use of access types and minimized the possibility of Constraint Errors. In order to facilitate debugging, the generation of test reports, and minimize typing; virtually all data types included a function, Img, to generate a string describing their value.

The Ada Utilities consist of the following collections (subdirectories of source text) of packages: Strings, File_Types, Num_Types, and Time_Types. The authors hope the term text describes the human readable information in their Ada packages rather than code.

The Strings collection consists of the following packages: Strings, Bounded_Strings, Generic_File_Names, Generic_File_Names.Dirs, File_Names. The Generic_File_Names package permits the same subprograms to be employed for DOS, Windows 95, and hopefully other operating systems. File_Names is the present instantiation for DOS. The file name and extension are each brought in as a string and the combination separated by a period is returned as a bounded string.

The File_Types collection, at the time of this writing, consists of Output_Text_Files and its two children Log and Tab_Delimited. Although Log exists only as a specification, it is withed by most of the other packages. Log provides the subprograms for the Log_File, which contains much of the data generated by the program including the exception messages. The final data output of the photon counter will be stored in a separate tab delimited format file (Tab_Delimited), which will be read by a spreadsheet.

In order to avoid a circular dependency, the subprogram bodies in the Generic_File_Names package output their exception error messages to an Error_File, which has a different name than the Log_File. Since Output_Text_Files.Log withs File_Names, which is the instantiation of the Generic_File_Names package, the Generic_File_Names package can not with Output_Text_Files.Log.

This use of a Log_File eliminates two major problems: 1) possible loss of error messages, because with present, commercial graphical user interfaces, there is no absolute certainty that the user will receive the error messages raised by exceptions and 2) inaccurate error message descriptions by the users to technical support providers. The user can always read the Log_File and send it or its contents to software vendor technical support group.

The Num_Types collection consists of: Booleans, Floats, Integers, Unsigneds, Data, and Si_Prefixes. The purpose of the Luminometer is to produce data, which is in the form of an unsigned 32 bit integer, Data_32. The safety of Data_32, which is a modular type, was enhanced by redefining the addition, subtraction, multiplication, and numerical type conversion functions to raise an exception if the result of an operation would have been greater than maximum value of Data_32 or negative. The initial approach of creating Data_32 as a private type had the problem that there was no simple way to set a Data_32 object to be equal to a Universal_Integer.

Since the Luminometer is to be used by laboratory workers, exponential notation had to be replaced by a simpler format. The Si_Prefixes package contains a function which transforms a Float_15 into a string followed by a metric (thousands prefix and a standard fixed representation. For instance, 16.0E-6 produced 16.0 micro. The package Time_Types.Seconds uses this function to report the length of time in seconds with the appropriate prefix. This function is used to report the preset time setting of the timers on the PC100 board.

3.2 PC100 board:

The software consists of four parts: A Generic_Port package, a Pc100 package, its 5 child packages each of which models one type of port, and packages which describe the data including its creation and manipulation.

The Generic_Port package contains Port_Representation_Type, which is tagged private. This record after instantiation and the subsequent addition of the actual layout of the register, creates a record which describes the port and the data it inputs or outputs.

```
type Port_Representation_Type is tagged
 record
  Port_Part : Port_Type := Port_Value;
  Register_Address_Part :
   Register_Address_Type :=
    Register_Address;
  Data_Direction_Part :
   Data_Direction_Type :=
    Data_Direction;
  Port_Object_Name_Part :
   Port_Object_Name_Type
    := Port_Object_Name;
end record;
```

Since Port_Type, Port_Value, Register_Address_Type, Register_Address, Data_Direction_Type Data_Direction, and Port_Object_Name are all generic, all types of computer ports can be modeled and manipulated by the subprograms contained in the generic. The actual modeling of the port content occurs after the instantiation. The use of generic type, permits the Port_Value to be 8, 16, 32, etc. bits wide. Similarly, the Register_Address can be a memory or port location. The procedure Write_Port_Representation stores the contents of the tagged record in the Log_File; and the procedure Create_Port is employed when a second port of the same type needs to be created.

Package Pc100 is the parent of the 7 ports. Except for the Port_Object_Name and the actual value of the offset from the Base_Address, it creates all the types and provides the values for instantiations of the Generic_Port package, which occur in the children of Pc100.

```
Pc100port_Data_Direction_Type is
 (Write_Only, Read_Only);
Pc100port_Base_Address : constant
 Unsigned_16 := 440;
subtype Pc100port_Address_Type is
 Unsigned_16 range
  Pc100port_Base_Address
  ..Pc100port_Base_Address +3;
subtype Port_8_Bit_Type is Unsigned_8;
subtype Port_Number_Type is Unsigned_16
 range 1..2;
```

Package Pc100 also provides the two procedures Read_Pc100port and Write_Pc100port, which are employed for controlling the ports and acquiring the data.

The values of the individual registers are shown above in Table 1.

Table 2 Naming Conventions

Item	Suffix
Object	None
Package	_Pkg or Plurals
Types	_Type
Record_Part	_Part
Record	_Rec
Array	_Array
Access	_Ac

Ada Source Text Writing Rules:

1. Follow inflected style in Ada as a Second Language, N. Cohen, McGraw Hill, 1996[5].
   Attach suffixes to describe the entity. Only objects do not have suffixes:
2. At the top of packages, rename withed child packages to that of the child;
   ```
   Package Integers renames
    Numtypes.Integers;
   ```
3. Similarly, create subtypes with the name of the type in the child.
   ```
   Subtype Integer_32 is
    Num_types.Integers.Integer_32;
   ```
4. Employ the Formal => Actual notation except where the Formal and Actual parameters are identical or the situation is very obvious.

3.3 Example PC Port Child Package:

The child package, PC100.Status, will be described. Status provides an interesting example of the modeling of a port. The parts of the Generic_Port which are specific to the Status port are: It is Read_Only, its address is offset +2 from the PC100port base address, and its name is “Status”. The actual modeling of the Status Register includes a private type

```
private
   type Counter_Mode_Type is
   (Independent, Simultaneous,
 Alternating);
   for Counter_Mode_Type'Size use 2;
   for Counter_Mode_Type use(
      Independent  => 2#00#,
```

```
      Simultaneous => 2#01#,
      Alternating  => 2#11#);
```

Since, the time between readings will be less than 30 microseconds, the PC100D board is operated in Alternating mode to minimize dead-time. This requires simultaneous rather than sequential monitoring of the state of both counters. The enumerated Counters_State_Type is described in two grandchildren of PC100. The one (PC100.Status.Both) employed during data acquisition for both the Alternating and Simultaneous mode reads the state of both counters.

```
type Counters_State_Type is
 (Both_Off,
 C_1_On_And_C_2_Off,
 C_1_Off_And_C_2_On,
 Both_Counting); --raises an exception
for Counters_State_Type'Size use 2;

for Counters_State_Type use
 (Both_Off => 2#00#,
 C_1_On_And_C_2_Off =>2#01#,
 C_1_Off_And_C_2_On =>2#10#,
 Both_Counting => 2#11#);
--------------------------------------
--The Status Register can now be created
--in the grandchild by combining one
--enumerated type from the child with an
--enumerated and two boolean types from
--the grandchild.
--------------------------------------
type Status_Reg_Type is record
 Counter_Mode_Part : Counter_Mode_Type:=
  Alternating;
 Counters_State_Part :
  Counters_State_Type := Both_On;
 Timer_1_Interrupt_Enabled_Part :
  Boolean := False;
 --Blocks access to Timer_1.
 Timer_2_Interrupt_Enabled_Part :
  Boolean := False;
end record;
for Status_Reg_Type'Size use 8;
for Status_Reg_Type use record
 Counter_Mode_Part at 0 range 0..1;
 Counters_State_Part at 0 range 4..5;
 Timer_1_Interrupt_Enabled_Part at 0
  range 6..6;
 Timer_2_Interrupt_Enabled_Part at 0
  range 7..7;
end record;
--------------------------
```

The addition of The Status_Reg_Type to the tagged type created in Status from the instantiation of the Generic_Port package results in the complete description of the Status Register:

```
type Status_Port_Type is new
 Status_Port_Pkg.Port_Representation_
  Type with record
  Status_Reg_Item : Status_Reg_Type;
end record;
 Status_Port : Status_Port_Type;
```

3.4 Data Collection Example:

Below is an abbreviated version of the present data acquisition procedure. The present source text includes the capability of logging each step. Paradoxically, since the reference code is given as reading and writing integers to the registers, a function based on an Unchecked_Conversion had to be created to translate the states of the Status_Port.Status_Reg_Item into an unsigned 8 bit integer. This permitted the integer value equivalents of the higher level commands to be checked against the values given in the reference C and Basic examples.

```
procedure Read_Data_2_Alternating
 --abbreviated
 (Counts_Array: out Counts_Array_Type;
 Control_Values: in Control_Values_Type)
is
 Counts_Array_Var : Counts_Array_Type :=
  Create_Zero_Counts_Array;
 --Start with all data items in the
--array set to zero

 Counter_1_Port :
  Counters.Counter_Port_Type;
 Counter_2_Port :
  Counters.Counter_Port_Type
   := Counters.Create_Port_2;
 C_1_Read, C_2_Read : Boolean := False;
 Control_Register_Port :
  Cont_Reg.Control_Register_Port_Type;
 Counters_State :
  Status_Both.Counters_State_Type;

 Both_Counters_On, Both_Counters_Off :
  exception;
begin --Read_Data_2_alternating
Mode_Interups_Enable.Set_Counter_Mode
 (Counter_Mode =>
  Mode_Interups_Enable.Alternating);

Fill_Counts_Array: --Collect the data
for I in Counts_Array_Type'range loop
 Counters_State :=
 Status_Both.Read_Counters_State;
 --4 possibilities
 case Counters_State is
```

```
  when Status_Both.C_1_Off_And_C_2_On
    => if not C_1_Read then
 --Counter 2 Busy and Counter 1 not
 --Busy, then Read Counter 1
   Count_Data.Fill_4_Bytes
    (Data_32 => Counts_Array(I),
    Counter_Port => Counter_1_Port);
 --Takes 4 bytes sequentially and stuffs
 --them into one Data_32 object,
 --and stores the Data_32 as the I^th
 --element of the Array, which will be
 --exported.

 Cont_Reg.Reset_Timer_1_Counter
  (Control_Register_Port, To => False);
 C_1_Read := True;
 --Since C_2 was busy, when it is
 --finished, it should be eligible for
 --reading.
 C_2_Read := False;
   end if;

 --Second possibility......
   when Status_Both.C_1_On_And_C_2_Off
 => --...
 --Third possibility......
   when Status_Both.Both_Counting =>
    raise Both_Counters_On;
   --Fourth possibility......
   when Status_Both.Both_Off  => raise
    Both_Counters_Off;
  end case;

end loop Fill_Counts_Array;
exception
 when N : Both_Counters_On =>
  Log.Log_Exception
  (The_Exception_Occurrence => N,
  Error_Message => "Both_Counters_On"
   & "C_1_read = )". Bool.Img(C_1_Read)
   & "C_2_read = " & Bool.Img(C_2_Read)
   & "The Status_Register = " &
    Status_Both.Status_Register_Image);
end Read_Data_2_Alternating;
```

The use of a case statement permits meaningful descriptions of the state of the counters. The capacity of describing the state of the counters either as individual booleans or as an enumerated type consisting of the permutations of the states of two together eliminates the need to employ the hack of representing the states, which clearly are not numeric values, by integers. The inclusion of a comprehensive error messages of up to 256 characters is far more useful than the present register dumps.

The full use of Ada object technology at this point in the development of the system where the compilations are still being performed in debug mode apparently still provides ample performance. A nonproductive (waiting period) circuit through the Fill_Counts_Array loop takes less than 4 microseconds with a Pentium 90 CPU. The PC100D manual on page 20 states, "A 486 33 Mhz computer in Basica shows minimum time base of 2ms. A faster, more efficient language will push the limit below the millisecond range. Hopefully, Ada will be that "faster, more efficient language," and maybe the fastest most efficient language.

4. Conclusions:

Ada 95 provides excellent tools: for object oriented programing (generics, tagged types, and child packages). Generics with embedded tagged types provide an excellent model of real-world hardware. The rich set of types and control of their low level representation provided by Ada permits and facilitates the creation of readable packages which describe and control hardware. Readability is not only an advantage for future maintenance; it is also a necessity for meaningful code reviews.

With Ada, one has a sporting chance at decrypting the vendor's documentation. The development of the Ada Utilities and Port Class infrastructure was expensive in time. The original cost for creating Ada source text is significantly higher than for creating read-once C or BASIC code. The cost of modeling the registers of the next board will be small.

5. Acknowledgments:

The project described was supported in part by Small Business Technology Transfer grant number 1R41CA73089 from the National Cancer Institute. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the (NIH awarding component). The assistance of Jerry van Dijk, Ed Falis, and Aonix Technical Support is gratefully acknowledged. Stephanie Leif provided very helpful comments.

6. REFERENCES

[1]. R. C. Leif and L. M. Vallarino, "Rare-Earth Chelates as Fluorescent Markers in Cell Separation and Analysis". ACS Symposium Series 464, Cell Separation Science and Technology, D. S. Kompala and P. W. Todd Editors, American Chemical Society, Washington, DC, pp. 41-58 (1991).

[2]. Macrocycle complexes of Yttrium, the Lanthanides and the Actinides having Peripheral Coupling Functionalities, L. M. Vallarino and R. C. Leif, U.S. Patent 5,373,093 (1994).

[3]. R. C. Leif, R. Rios, M. C. Becker, C. K. Becker, J. T. Self, and S. B. Leif, "The Creation of a Laboratory Instrument Quality Monitoring System with AdaSAGE". Advanced Techniques in Analytical Cytol-

ogy, Optical Diagnosis of Living Cells and Biofluids, Ed. T. Askura, D. L. Farkas, R. C. Leif, A. V. Priezzhev, and B. J. Tromberg. A. Katzir Progress in Biomedical Optics Series Editor SPIE Proceedings Series, Vol. 2678, 232-239 (1996).

[4]. D-W. Jones, “Ada in Action, With Practical Programming Examples,” John Wiley & Sons, inc., N.Y. ISBN 0-471-6078-8 (1989).

[5]. N. H. Cohen, “Ada as a Second Language,” McGraw Hill, N.Y. ISBN 0-07-011607-5 (1996).