Managing the Endianness of Software Building Blocks with GNAT Ada Attributes: a Case Study

Patricia Lopez Cueva, Marco Panunzio

Ada-Europe – 19/06/2018 - Lisbon

Thales Alenia Space
Outline

- Recap on Endianness
- Context
- The PUS standard
- Scalar Storage Order Attribute
- Application to selected case study
- Conclusions

(+ ) We acknowledge Adrien Prieux, now with Thales Services, for his essential contribution to this investigation
Endianness

**Big endian**

Big-endian format requires storage with the most significant byte first. Namely, that the most significant bit is stored at the lowest memory address then the following bytes are stored in decreasing order of significance.

Example: one hundred twenty-three

**Little endian**

Little-endian format reverses the order of the sequence and stores the least significant byte at the first location in memory and the most significant byte is stored last.
Endianness

**Endian independent layout**

The memory mapping corresponding to the physical representation above is:

<table>
<thead>
<tr>
<th>Bit Big-Endian</th>
<th>Bit Little-Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Big-Endian</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Byte Little-Endian</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

The equivalent Ada record representation clause:

```ada
for T_BE use record
  U at 0 range 0 .. 15;
  D at 0 range 16 .. 28;
  T at 0 range 29 .. 33;
  Q at 0 range 34 .. 39;
  C at 0 range 40 .. 47;
end record;
```

```ada
for T_LE use record
  U at 0 range 48 .. 63;
  D at 0 range 35 .. 47;
  T at 0 range 30 .. 34;
  Q at 0 range 24 .. 29;
  C at 0 range 16 .. 23;
end record;
```
Why is this relevant to space software?

Current US or EU spacecrafts mostly use Big-Endian Processor Architectures
- SPARC (mostly in Europe, some cases in US)
- ERC32, LEON2, LEON3, LEON4
- PowerPC (US, Europe in some cases)
- PPC 750, PPC 8548, etc..

New ARM-based hardware targets (Little Endian) will start to be used in the next future
- ARM R5 (Project developments already ongoing)
- ARM R52 (H2020 DAHLIA)
- ARM A9 (CNES Hyperion / ESA OPSAT)
- ARM A53 (new NASA on-board computer)

Porting consolidated building blocks with 10+ years of heritage across architectures requires to address the problem of endianness in a satisfactorily manner.
Example of next ARM hardware target
Goals and case study

- Facilitate retargeting from current big-endian hardware to another platform with little-endian hardware
- Maintain a unique version of OBSW code which works on both architectures
- Produce an endian-agnostic application and solve interoperability with ground software
- Remain compatible with Ada 2005 and with both GNAT compiler for ARM and SPARC architectures
- Come up with a solution should be affordable and easy to implement

Selected case study: application to a subset of the TAS PUS library

Overall 30000 Ada SLOCs
"Telemetry and Telecommand Packet Utilisation"  
ECSS-E-ST-70-41C

The European standard to describe the contents and format of spacecraft Telecommands (TC) and Telemetry (TM)

- Description of common services to operate the spacecraft
- Mostly focuses on the data format of the exchange
- With some data exchanges protocol

<table>
<thead>
<tr>
<th>Packet Header (48 Bits)</th>
<th>Packet Data Field (Variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet ID</td>
<td>Packet Sequence Control</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

Note: The reserved service type identifiers were used in previous versions of this Standard. This Standard no longer promotes the use of these service types but does not preclude that existing implementations are reused for new missions.
### PUS – Datafield examples

#### 8.1.2.8 TM[1,8] failed completion of execution verification report

a. Each telemetry packet transporting a failed completion of execution verification report shall be of message subtype 8.

   **NOTE** For the corresponding system requirements, refer to clause 6.1.3.1.

b. For each telemetry packet transporting a failed completion of execution verification report, the source data field shall have the structure specified in Figure 8-5.

<table>
<thead>
<tr>
<th>packet version number</th>
<th>packet ID</th>
<th>packet sequence control</th>
<th>code</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet type</td>
<td>secondary header flag</td>
<td>application process ID</td>
<td>sequence flags</td>
<td>packet sequence count</td>
</tr>
<tr>
<td>enumerated (3 bits)</td>
<td>enumerated (1 bit)</td>
<td>enumerated (11 bits)</td>
<td>enumerated (2 bits)</td>
<td>unsigned integer (14 bits)</td>
</tr>
</tbody>
</table>

**NOTE** The request ID field alone cannot be used to identify the request since it does not contain the identifier of the source of that request. That source identifier corresponds to the destination identifier of the secondary header of the related telemetry packet, refer to clause 7.4.3.1.

#### 8.22.2.10 TM[22,10] position-based schedule detail report

a. The telemetry packet transporting a position-based schedule detail report shall be of message subtype 10.

   **NOTE** For the corresponding system requirements, refer to clause 6.2.9.2.

b. For each telemetry packet transporting a position-based schedule detail report, the source data field shall have the structure specified in Figure 8-246.

- **TC packet**
  - N
  - sub-schedule ID
  - group ID
  - position tag
  - activity persistency status
  - persistent activity periodicity

**NOTE 1** The structure of the position tag field is driven by requirement 8.22.1b.

**NOTE 2** For the activity persistency status enumerated values, refer to requirement 8.22.3d.
Big Endian vs. Little Endian

**SPARC Processor**
- **BIG ENDIAN**

**ARM Processor**
- **LITTLE ENDIAN**

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit integer</td>
<td>12</td>
<td>34</td>
<td>56</td>
<td>78</td>
</tr>
<tr>
<td>Read-Value</td>
<td>0x12345678</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PROCESSING**
- **VALID DATA**
- **WRONG DATA**

**Example:** 0x12345678

**Big-Endian Memory Address**
- 32-bit integer
- 1 Byte 1 Byte 1 Byte 1 Byte
- 0 1 2 3
- 12 34 56 78

**Little-Endian Memory Address**
- 32-bit integer
- 1 Byte 1 Byte
- 0 1 2 3
- 1 Byte 1 Byte
- 12 34 56 78

19/06/2018
2018 Thales Alenia Space
Investigation on Scalar Storage Order

Permits to specify bit / byte order of storage

```plaintext
for T_2 use record
  U at 0 range 0 .. 15;
  D at 2 range 0 .. 12;
  T at 2 range 13 .. 17;
  Q at 2 range 18 .. 23;
  C at 5 range 0 .. 7;
end record;
for T_2:Bit_Order use System.High_Order_First;
for T_2:Scalar_Storage_Order use System.High_Order_First;
```
TC reception

- Received data – Big Endian (Byte-stream received from low-level drivers)

- Unchecked conversion to endian pragma structure

- Assign to a variable

- Unchecked conversion

- Assign an element to a variable
type TC_HEADER_T is record
  U : UINT16_T;
  D : UINT13_T;
  T : UINT5_T;
  Q : UINT6_T;
  C : UINT8_T;
end record;
for TC_HEADER_T use record
  U at 0 range 0 .. 15;
  D at 0 range 16 .. 28;
  T at 0 range 29 .. 33;
  Q at 0 range 34 .. 39;
  C at 0 range 40 .. 47;
end record;
for TC_HEADER_T'Bit_Order use System.High_Order_First;
for TC_HEADER_T'Scalar_Storage_Order use System.High_Order_First;

--Received data Exemple component U is 16#1234#
subtype TC_T is UINT8_ARRAY_NC_T(0 .. 5);
TC  : TC_T := (16#12#,16#34#,16#56#,16#78#,16#9a#,16#bc#);
--Header Structure
TC_HEADER : TC_HEADER_T;
-- to convert the TC packet from bytes list to an TC_HEADER_T structure
function TO_TC_HEADER is new UNCHECKED_CONVERSION (TC_T, TC_HEADER_T);
--Memory map of TC_HEADER  : 16#12#, 16#34#
TC_HEADER := TO_TC_HEADER (TC);
--Memory map of U_var  : 16#34# , 16#12#
U_var := TC_HEADER.U;

< gdb > x /2xb &TC_HEADER
0x245fe78:  0x12  0x34
< gdb > x /2xb &U_var
0x245fe8a:  0x34  0x12
On-board parameter sent as TM

---

**VARIABLE**

<table>
<thead>
<tr>
<th>LittleEndian</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

Swap

**Data Structure**

Assign the element to the structure

**TM**

<table>
<thead>
<tr>
<th>BigEndian</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
</tr>
</tbody>
</table>
## Assembler level

<table>
<thead>
<tr>
<th></th>
<th>BIG ENDIAN</th>
<th>LITTLE ENDIAN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>X86</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>401a8e: 8b 45 ec mov -0x14(%ebp),%eax</td>
<td>401a8e: 8b 45 ec mov -0x14(%ebp),%eax</td>
</tr>
<tr>
<td></td>
<td>401a91: 89 45 e0 mov %eax,-0x20(%ebp)</td>
<td>401a91: 0f c8 bswap %eax</td>
</tr>
<tr>
<td></td>
<td>401a93: 89 45 e0 mov %eax,-0x20(%ebp)</td>
<td>401a93: 89 45 e0 mov %eax,-0x20(%ebp)</td>
</tr>
<tr>
<td><strong>ARM</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3b48: 689b ldr r3, [r3, #8]</td>
<td>3b48: 689b ldr r3, [r3, #8]</td>
</tr>
<tr>
<td>+---------------------------------+---------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3b4a: 461a mov r2, r3</td>
<td>3b4a: balb rev r3, r3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3b4c: 461a mov r2, r3</td>
</tr>
</tbody>
</table>

```c
NB_TC_SELECTION_REC : NB_TC_SELECTION_REC_T
  := ( NUMBER_OF_TC , 16#2233# , 16#4455# );
```

```
401a10: 66 c7 45 e2 33 22 movw $0x2233,-0x1e(%ebp)  
401a16: c7 45 e4 55 44 00 00 movl $0x4455,-0x1c(%ebp)
401a10: 66 c7 45 e2 22 33 movw $0x3322,-0x1e(%ebp)  
401a16: c7 45 e4 00 00 44 55 movl $0x55440000,-0x1c(%ebp)
```
“Propagation” of attribute specification

```pli
type TC_HEADER_T is
record
    COSDS_HEADER : COSDS_TC_HEADER_T;
    FUS_HEADER   : FUS_TC_HEADER_T;
end record;

for TC_HEADER_T use
record
    COSDS_HEADER at 0 range 0 .. 47;
    FUS_HEADER   at 0 range 48 .. 71+5*POS.CST.SRC_DEST_ID_FIELD_SIZE C;
end record;

for TC_HEADER_T SIZE use 72+5*POS.CST.SRC_DEST_ID_FIELD_SIZE C;
for TC_HEADER_T Bit_Order use System.High_Order_First;
for TO_HEADER_T Scalar_Storage_Order use System.High_Order_First;
```
Case study execution

Tested on a subset of PUS services: 1, 3, 5, 17

- HMI / HTML
- TC sender
- TM receiver

LEON3 board
BIG ENDIAN

SPARC binary image

PUS Lib
Simplified OBSW

PikeOS Hypervisor

ARM binary image

ZedBoard
Dual ARM A9
LITTLE ENDIAN
Conclusions

This study had the goal to investigate new techniques for managing endianness of code

- By maintaining a single code baseline for both big-endian and little-endian architectures
- With a solution that could be as simple and elegant as possible

We therefore focused on the attribute “Scalar Storage Order” attribute

The prototype realised during the investigation permitted to confirm the correct behavior of the attribute

The solution is more interesting than some others (e.g., macros or code-swap functions)

- Applies to data declaration
- A priori does not impact functional / business code

However a tall has to be paid

- In some cases, ad-hoc intermediate data types have to be introduced in order to “force endianness checks”
  - E.g., to cope with extraction of scalar values with a given endianness

The solution works well if the overall software architecture was already layered so as to isolate most of the endiannes concerns in 1-2 modules of the architecture
End of Presentation

QUESTIONS?