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

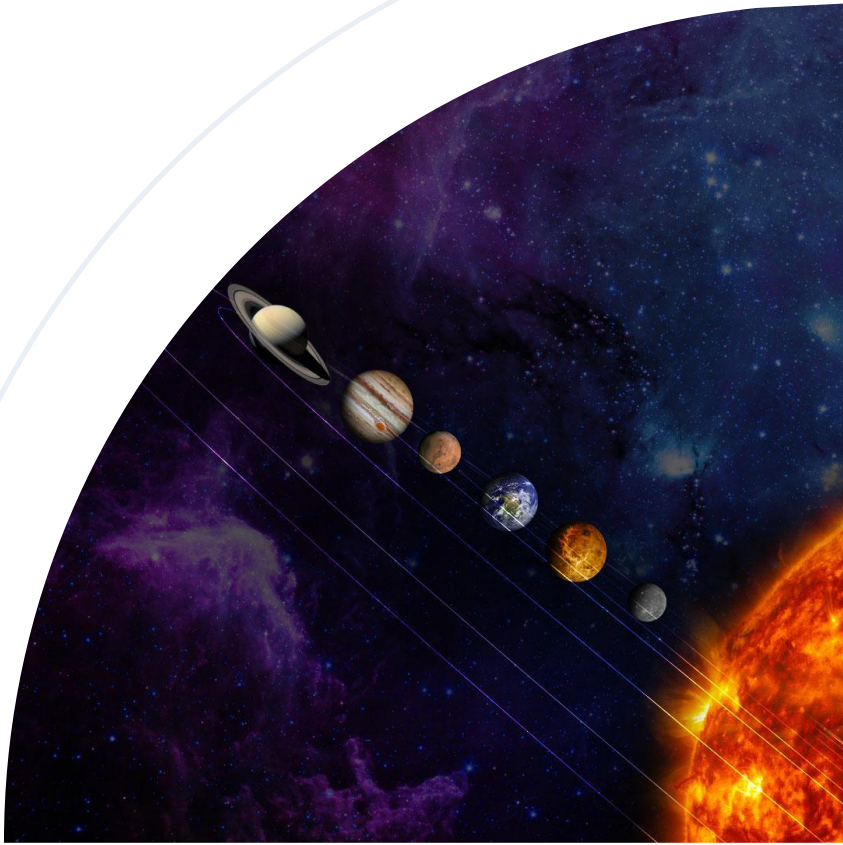
Patricia Lopez Cueva, Marco Panunzio

Ada-Europe – 19/06/2018 - Lisbon

ThalesAlenia Space
a Thales / Leonardo company

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

Q-Word 64bits at offset 0	U	D	T	Q	C		
	u u u u u u u u u u u u u u u u	d d d d d d d d d d d d d d d d	t t t t	q q q q q	c c c c c c c c		

The memory mapping corresponding to the physical representation above is:

Bit Big-Endian	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
Bit Little-Endian	63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Byte Big-Endian	0	1	2	3	4	5	6	7
Byte Little-Endian	7	6	5	4	3	2	1	0
	u u u u u u u u	u u u u u u u u	d d d d d d d d	d d d d t t t t	t q q q q q q q	c c c c c c c c		

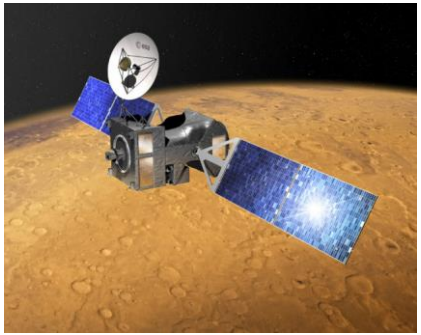
The equivalent Ada record representation clause :

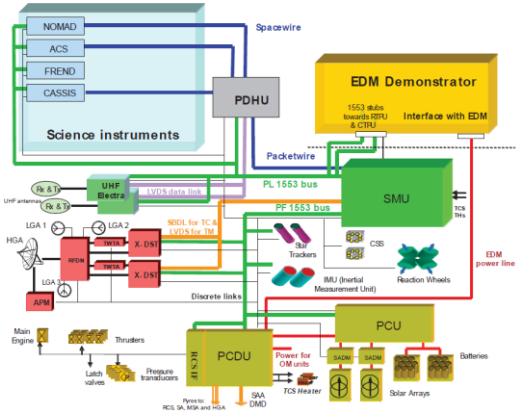
```
for T_BE use record
   U at 0 range  0 .. 15;
   D at 0 range 16 .. 28;
   T at 0 range 29 .. 33;
   Q at 0 range 34 .. 39;
   C at 0 range 40 ..  47;
end record;
```

```
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 endiannes in a satisfactorily manner

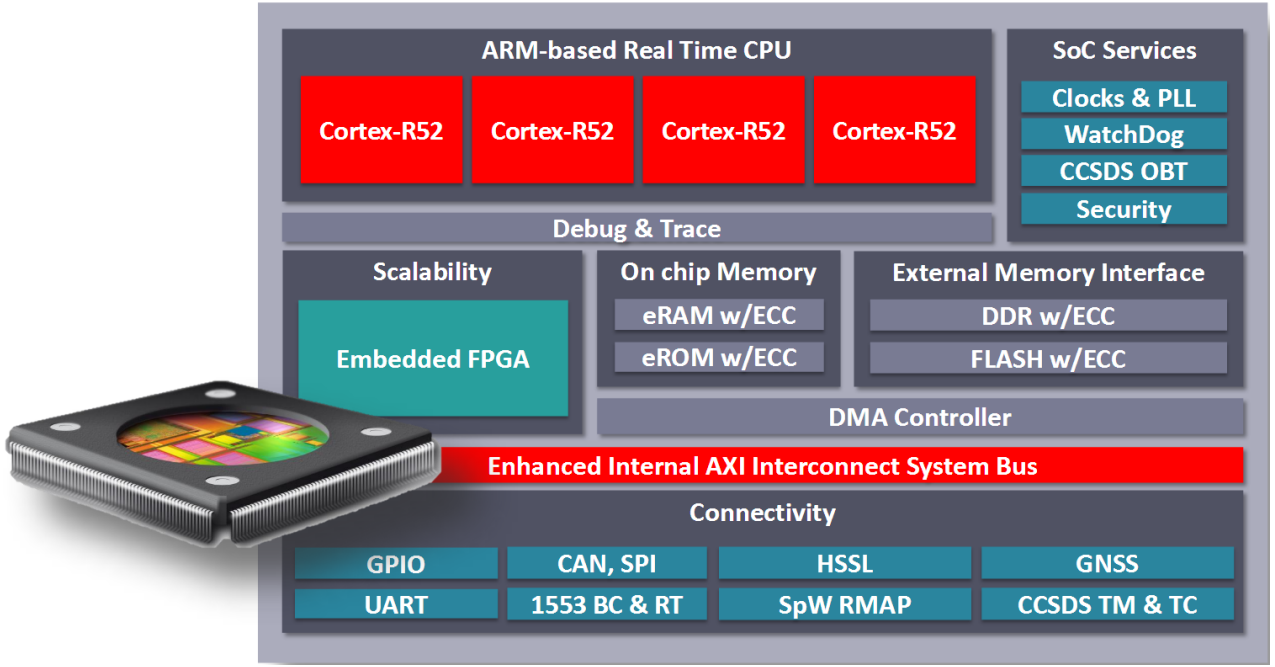




ThalesAlenia Space
a Thales / Leonardo company

Example of next ARM hardware target

DAHLIA



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

PUS

- "Telemetry and Telecommand Packet Utilisation" ECSS-E-ST-70-41C
- The European standard to describe the contents and format of spacecaft 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

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Application Data	Spare	Packet Error Control (see Note 2)
Version Number (=0)	Type (=1)	Data Field Header Flag	Applica-tion Process ID	Sequence Flags	Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	16

service type	
name	ID
request verification	1
device access	2
housekeeping	3
parameter statistics reporting	4
event reporting	5
memory management	6
(reserved)	7
function management	8
time management	9
(reserved)	10
time-based scheduling	11
on-board monitoring	12
large packet transfer	13
real-time forwarding control	14
on-board storage and retrieval	15
(reserved)	16
test	17
on-board control procedure	18
event-action	19
parameter management	20
request sequencing	21
position-based scheduling	22
file management	23
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 6.1.5.3.2.

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

request ID						failure notice	
packet version number	packet ID			packet sequence control		code	data
	packet type	secondary header flag	application process ID	sequence flags	packet sequence count		
enumerated (3 bits)	enumerated (1 bit)	Boolean (1 bit)	enumerated (11 bits)	enumerated (2 bits)	unsigned integer (14 bits)	enumerated	deduced
							deduced presence

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 6.22.9.2.

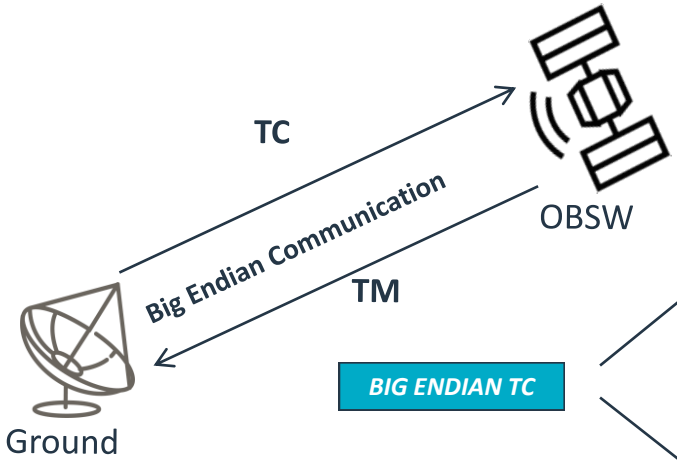
b. For each telemetry packet transporting a position-based schedule detail report, the source data field shall have the structure specified in Figure 8-246.

repeated N times

N	sub-schedule ID	group ID	position tag	activity persistency status	persistent activity periodicity	request
enumerated	enumerated	enumerated	deduced	enumerated	unsigned integer	TC packet
					deduced presence	
	optional	*optional*		*optional*		

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



PROCESSING

SPARC Processor

BIG ENDIAN

VALID DATA

Big-Endian	1 Byte	1 Byte	1 Byte	1 Byte
Memory Address	0	1	2	3
32-bit integer	12	34	56	78
Read-Value	0x12345678			

PROCESSING

ARM Processor

LITTLE ENDIAN

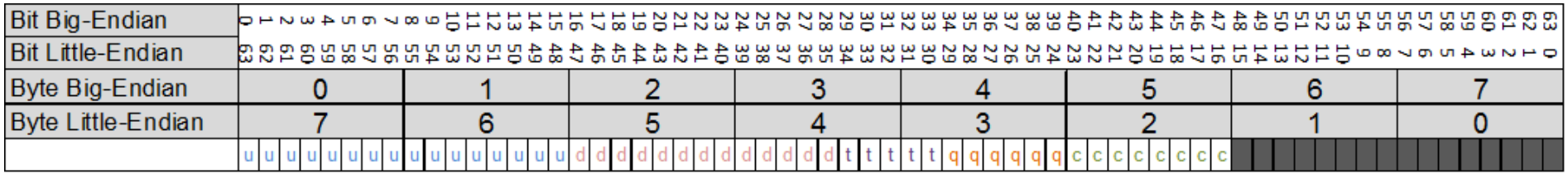
WRONG DATA

Little-Endian	1 Byte	1 Byte	1 Byte	1 Byte
Memory Address	0	1	2	3
32-bit integer	12	34	56	78
Read-Value	0x78563412			

Investigation on Scalar Storage Order

- Permits to specify bit / byte order of storage

```
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;
```



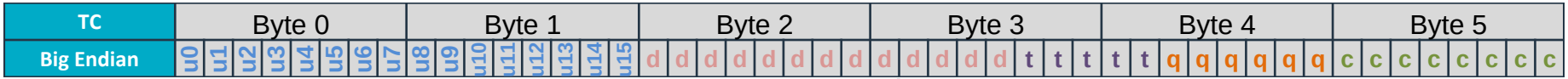
19/06/2018

Ref.: <reference>

Template : 83230347-DOC-TAS-EN-005

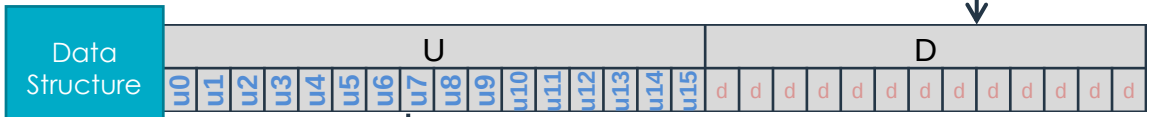
TC reception

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



- Unchecked conversion to endian pragma structure

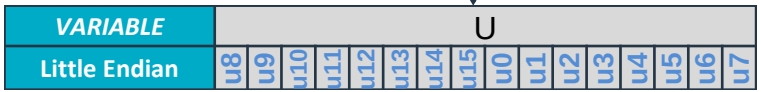
Unchecked conversion



Swap

Assign an element to a variable

- Assign to a variable

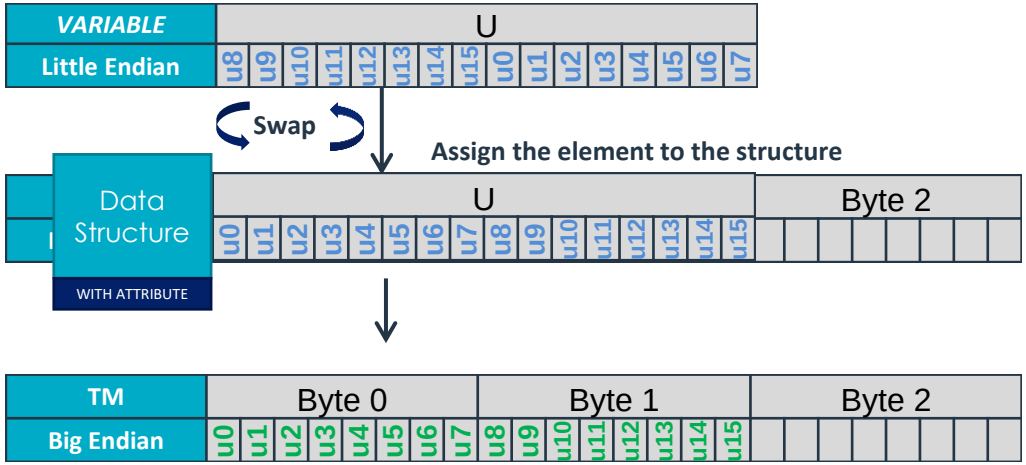


TC to Data Structure

```
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 Stucture
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



Assembler level

	BIG ENDIAN	LITTLE ENDIAN
X86	401a8e: 8b 45 ec mov -0x14(%ebp),%eax 401a91: 89 45 e0 mov %eax,-0x20(%ebp)	401a8e: 8b 45 ec mov -0x14(%ebp),%eax 401a91: 0f c8 bswap %eax 401a93: 89 45 e0 mov %eax,-0x20(%ebp)
ARM	3b48: 689b ldr r3, [r3, #8] 3b4a: 461a mov r2, r3	3b48: 689b ldr r3, [r3, #8] 3b4a: ba1b rev r3, r3 3b4c: 461a mov r2, r3

```
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

```
type TC_HEADER_T is
   record
      CCSDS_HEADER : CCSDS_TC_HEADER_T;
      PUS_HEADER   : PUS_TC_HEADER_T;
   end record;

for TC_HEADER_T use
   record
      CCSDS_HEADER at 0 range 0 .. 47;
      PUS_HEADER   at 0 range 48 .. 71+8*PUS.CST.SRC_DEST_ID_FIELD_SIZE_C;
   end record;

for TC_HEADER_T'SIZE use 72+8*PUS.CST.SRC_DEST_ID_FIELD_SIZE_C;
for TC_HEADER_T'Bit_Order use System.High_Order_First;
for TC_HEADER_T'Scalar_Storage_Order use System.High_Order_First;
```

```
type CCSDS_TC_HEADER_T is
   record
      PACKET_ID      : CCSDS_TC_PACKET_ID_T;
      SEQUENCE_FLAGS : PUS.TYPES.BITS2_T;
      SEQUENCE_COUNT : PUS.TYPES.SEQUENCE_COUNT_T;
      PACKET_LENGTH  : PUS.TYPES.UNSIGNED_HALF_WORD_T;
   end record;

for CCSDS_TC_HEADER_T use
   record
      PACKET_ID      at 0 range 0 .. 15;
      SEQUENCE_FLAGS at 0 range 16 .. 17;
      SEQUENCE_COUNT at 0 range 18 .. 31;
      PACKET_LENGTH  at 0 range 32 .. 47;
   end record;

for CCSDS_TC_HEADER_T'SIZE use 48;

for CCSDS_TC_HEADER_T'Bit_Order use System.High_Order_First;
for CCSDS_TC_HEADER_T'Scalar_Storage_Order use System.High_Order_First;
```

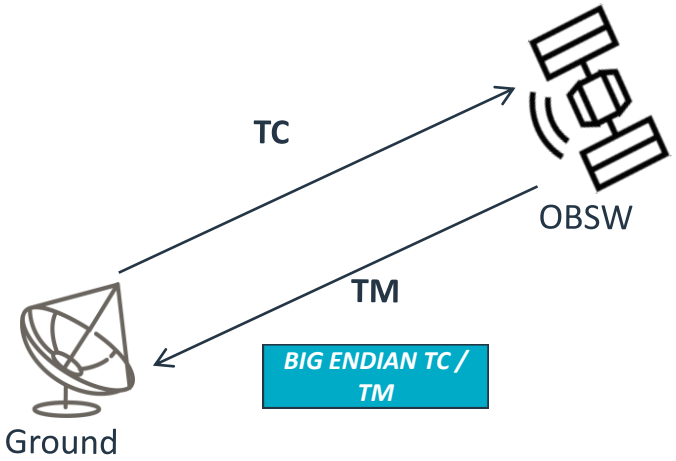
```
type PUS_TC_HEADER_T is
   record
      SECONDARY_HEADER_FLAG : PUS.TYPES.BIT_T;
      VERSION               : PUS.TYPES.BITS3_T;
      ACK_COMPLETE          : PUS.TYPES.BIT_T;
      ACK_PROGRESS          : PUS.TYPES.BIT_T;
      ACK_START             : PUS.TYPES.BIT_T;
      ACK_ACCEPTANCE        : PUS.TYPES.BIT_T;
      SERVICE_TYPE          : PUS.TYPES.SERVICE_TYPE_T;
      SERVICE_SUBTYPE       : PUS.TYPES.SERVICE_SUBTYPE_T;
#if SRC_DEST_FIELD_USED    -- if source id/destination id fields are included in the PUS header
      SOURCE_ID             : PUS.TYPES.BYTE_T;
#end if;
   end record;

   for PUS_TC_HEADER_T use
      record
         SECONDARY_HEADER_FLAG at 0 range 0 .. 0;
         VERSION               at 0 range 1 .. 3;
         ACK_COMPLETE          at 0 range 4 .. 4;
         ACK_PROGRESS          at 0 range 5 .. 5;
         ACK_START             at 0 range 6 .. 6;
         ACK_ACCEPTANCE        at 0 range 7 .. 7;
         SERVICE_TYPE          at 0 range 8 .. 15;
         SERVICE_SUBTYPE       at 0 range 16 .. 23;
#if SRC_DEST_FIELD_USED    -- if source id/destination id fields are included in the PUS header
         SOURCE_ID             at 0 range 24 .. 31;
#end if;
      end record;

   for PUS_TC_HEADER_T'SIZE use 8*PUS.CST.PUS_HEADER_SIZE_C;
   for PUS_TC_HEADER_T'Bit_Order use System.High_Order_First;
   for PUS_TC_HEADER_T'Scalar_Storage_Order use System.High_Order_First;
```

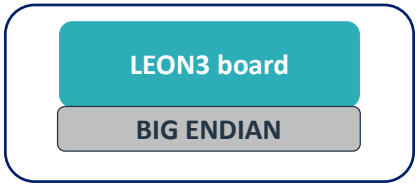
Case study execution

HMI / HTML

TC sender

TM receiver





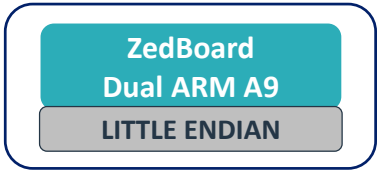
SPARC binary image

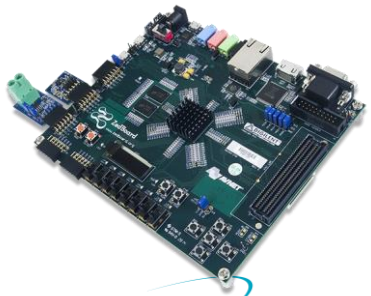
PUS Lib

Simplified OBSW

PikeOS Hypervisor

ARM binary image





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

ThalesAlenia Space
a Thales / Leonardo company

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 then 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?