FRONTGRADE

APPLICATION NOTE

UT700

Understanding the LEON Buses Capability
UT700 LEON 3FT

10/22/2019
Version #: 1.0.0

Product Name	Manufacturer Part Number	SMD #	Device Type	Internal PIC Number
UT700 LEON	UT700	5962-13238	LEON Buses	WQ03

Table 1: Cross Reference of Applicable Products

1.0 Overview

This application note explains the UT700 LEON 3FT SPARC™ V8 Microprocessor data bus capability and how the LEON presents a sub-word operation to the bus with a 16-bit or 32-bit configuration. Knowing how the sub-word operates on the data bus at different segments of the memory map can allow programmers to use the appropriate programming language syntax when communicating over the bus.

Note: The description in this application note describes how to directly use the memory mapped interface of a specific hardware peripheral. If you are using an operating system such as RTEMS, Linux, and VxWorks or an environment such as BCC then it is recommended to use the infrastructure provided by those environments instead of accessing the peripheral directly as described in this application note.

2.0 Application Note Layout

This application note starts by providing a brief description of how the LEON CPU presents a sub-word operation to the bus. We follow by showing the different memory map segments and how the bus corresponding to that memory segment will operate. More details about the unique feature of each segment bus operation follow with C code examples to elaborate on the critical nature of programming the bus corresponding to the different memory segments.

These subsections are described in detail below:

- LEON Sub-word Operation
- LEON Memory Map and Bus Operations
- Bus Access Programming Examples.

3.0 LEON Sub-word Operation

The UT700 LEON 3FT SPARC™ V8 Microprocessor presents a sub-word datum on the data bus by duplicating the subword size up to 32-bit. **Table 2** and **Table 3** show how the LEON duplicates the sub-word for the 16-bit and 32-bit data bus.

Sub-word	Size (Bit)	Value (Hex)	LEON 16-bit Data Bus	Remarks
Byte/char	8	0x12	0x1212	Before read-modify-write, if supported!

Table 2: 16-bit Bus Sub-word Operation value appear on the bus

Sub-word	Size (Bit)	Value (Hex)	LEON 32-bit Data Bus	Remarks
Byte/char	8	0x12	0x1212_1212	Before read-modify-write, if supported!
Half word/short	16	0x1234	0x1234_1234	

Table 3: 32-bit Bus Sub-word Operation value appear on the bus

These sub-word arrangements allow 8-bit and 16-bit devices to operate on a 16-bit or 32-bit bus. Moreover, the address bits [1:0] are also presented over the address bus. Unlike a "true" 32-bit bus configuration, the address bits [1:0] are omitted, allowing only 32-bit peripherals to operate on the bus. The former bus implementation provides the flexibility of mixing and matching the different peripherals (8-bit, 16-bit, and 32-bit) data bus width to operate on the same bus. Nonetheless, this flexibility comes with the restriction and limitation. When accessing the peripheral on this bus, the programmer must use the correct data size corresponding to that peripheral. For example, the LEON has a mix of 32-bit and 8-bit (CAN) peripherals must require the programmer to use the programming language syntax for byte and word to access these two types of devices. Read-modify-write (RMW) operation cannot be implemented in such a bus configuration.

Note: The LEON has 32-bit and 8-bit (CAN) peripherals and the programmer must use the word and byte access methods respectively.

3.1 LEON Memory Map and Buses Operation

Table 4 shows the memory map of the LEON processor. Each memory-mapped region has its method to configure the bus width and to manipulate the datum provided by the LEON (CPU, see **Table 2** and **Table 3**). We will elaborate on each memory region and its corresponding bus behavior in the next paragraph.

Memory Area	Memory Range	Memory Chip Select Pins
PROM	0x0000_0000-0x0FFF_FFFF	ROM[0]
PROM	0x1000_0000-0x1FFF_FFFF	ROM[1]
I/O	0x2000_0000-0x2FFF_FFFF	$\overline{\text{IOS}}$
Reserved	0x3000_0000-0x3FFF_FFFF	N/A
SRAM/SDRAM	0x4000_0000-0x7FFF_FFFF	$\overline{\text{RAMS}}$[4: 0]/$\overline{\text{SDCS}}$[1: 0]
Internal Peripherals	0x8000_0000-0xFFFF_FFFF	Internal chip select

Table 4: Memory Map Overview

Each memory region supports a different bus configuration as shown in **Table 5.** Also, see the UT699E_UT700 LEON Functional Manual section 3.14, the Register for a more comprehensive information.

Memory Area	Data Bus Width	Methods of Configuration
PROM	8-bit, 16-bit, and 32-bit	GPIO and MCFG1
PROM	8-bit, 16-bit, and 32-bit	GPIO and MCFG1
I/O	8-bit, 16-bit, and 32-bit	MCFG1
Reserved	Reserved	Reserved
SRAM	8-bit, 16-bit, and 32-bit	MCFG2 and RMW
SDRAM		MCFG3, Byte Lanes, and RMW
Internal Peripherals	Peripheral Specific	Peripheral Bus Width Specific

Table 5: Type of Memory Device Configurations

3.2 PROM, I/O, and Internal Peripheral Memory Regions

In these memory regions, there are neither RMW nor byte-lanes (BL) support. Accessing the memory devices in these regions must be peripheral bus width specific; otherwise, unexpected results will be written to the destination memory location (or Register).

Note: Accessing these memory regions must be **PERIPHERAL** bus width specific, for example, a 32-bit, 16-bit or 8-bit device must be accessed using word, half-word, and byte access method respectively.

3.2.1 SRAM Memory Regions

The SRAM memory region supports the RMW feature. When the system starts, power-on reset (POR), the RMW bit is set (RM=1, MCFG2 [6]). The user must not clear the RM bit. If cleared, unexpected datum will be written to the destination memory location (see section **3.2**). When EDAC is enabled, the check bits are generated based on an aligned 32-bit word even with an 8-bit bus configuration. EDAC is not supported in a 16-bit configuration.

The RMW operation starts by setting the associated datum in the data cache status bit as dirty (assuming there is an associated datum in the data cache). The LEON implements a write-through data cache replacement policy will write the data to the write FIFO. When the memory controller reads that datum from the FIFO, it also reads the memory location corresponding to that write datum address. The write and read data are masked accordingly to form the desired datum and write back to that memory location. In the same instant, the snooping logic reads that new datum and writes it back the data cache and clear the dirty bit.

Note: This memory region supports RMW (RM=1, MCFG2 [6]).

3.2.2 SDRAM Memory Regions

This region supports the byte-lanes operation. In the byte-lanes operation, the address and the opcode are used to decode the byte-lanes enable signals. For example, a word consists of four bytes with the following arrangement: HH, HL, LH, and LL bytes. When writing the upper half-word, the byte-lanes associated with HH and HL will be enabled allowing the writing to that memory location. Similarly, when writing an LH byte, the byte-lane associated with LH is enabled. Our example elaborates only on a 32-bit configuration.

When EDAC is enabled, the memory controller enables the RMW feature that is independent of the RM bit of the SRAM region. The SDRAM read-modify-write operation is similar to that of the SRAM region (see section **3.2.1**).

Note: This memory region supports byte-lane access method and RMW when EDAC is enabled.

4.0 Bus Access Programming Examples

C-Bit fields do have several advantages as follows:

- Efficiency—storage of data structures by packing.
- Readability—members can be easily addressed by the names assigned to them.
- Low level programming—the biggest advantage of bit fields is that one does not have to keep track of how flags and masks actually map to the memory. Once the structure is defined, one is completely abstracted from the memory representation as in the case of bit-wise operations, during which one has to keep track of all the shifts and masks.

C-Bit fields are one of the gotchas of non-portable code. If the project you're working on is architecture specific, and you know how the compiler works and you don't care about portable code, you're only half golden! You still need to understand how the bus system works.

The LEON ROM, I/O, and Internal Peripheral buses don't support the use of C-Bit fields (see **Table 2**). Here is an example showing why you shouldn't use C-Bit fields when using these specified buses.

```
2   #include <stdint.h>
3   #define SPW0_BASEADDR 0x80000A00
4
5   typedef volatile uint32_t vuint32_t;
6
7   struct SPW_TAG {
8       union {
9           vuint32_t R; // Table 12.13: Description of SpaceWire Clock Divisor Register
10          struct {
11                  vuint32_t REV31_16  :16;    // Reserved
12                  vuint32_t LINK      :8;     // 8-Bit Clock Divisor Link State Value
13                  vuint32_t RUN       :8;     // 8-Bit Clock Divisor Run State Value
14          } B;
15      } SPWCLK;
16  }
17
18  #define SPW0 (*(volatile struct SPW_TAG*) SPW0_BASEADDR)
19
20  SPW0.SPWCLK.B.RUN   = 0x12;
21  SPWO.SPWCLK.B.LINK  = 0xAB;
```

Code 1: C-Bit Fields Declaration

Code line	Size	Datum	Bus	Memory	Result	Remarks
20	8 bits	0x12	0x1212_1212	0xFFFF_FFFF	0x1212_1212	overwritten
21	8 bits	0xAB	0xABAB_ABAB	0x1212_1212	0xABAB_ABAB	overwritten

Table 6: 32-bit Bus Access with C-Bit Fields Examples (ROM, I/O and Internal Peripheral)

Code 1 shows the C-Bit field declaration for the examples in **Table 6.** These examples show that they overwrite the former memory content instead of writing the data in the respective byte positions (the correct datum: 0xFFFF_AB12).

The correct method to access the bus (see **Code 2**) is to perform a read, mask, and OR the data as shown:

```
24  vuint32_t *SPW      = (vuint32_t *)SPW0_BASEADDR;
25  vuint32_t memVal    = 0;
26
27  memVal              = *SPW;
28  memVal              = (memVal & 0xFFFF_0000) | 0x0000_AB12;
29  *SPW                = memVal;
30
31  *SPW                = 0x0000_AB12;
```

Code 2: Manual RMW

Code lines 24 to 29 show how one can write the correct value to the memory. If one doesn't care about the former content in the memory location, use code line 31 example.

5.0 Summary

Bus accesses in the SRAM and SDRAM regions with C-Bit fields can achieve expected results because these two memory regions are supported with RMW and byte-lanes hardware. However, refer to section **4.0** if the user wants to use C-Bit fields.

Revision History

Date	Revision #	Author	Change Description	Page #
2019-10-22	1.0.0	MTS	Initial Release	