FRONTGRADE

APPLICATION NOTE

UT700

Enable the SpaceWire Interface UT700 LEON 3FT

9/29/2017
Version #: 1.0.2

Table 1: Cross Reference of Applicable Products

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

1.0 Overview

The UT700 LEON 3FT SPARC™ V8 Microprocessor provides four SpaceWire (SPW) modules with Remote Memory Access Protocol supports. It also armed each SPW module with DMA capability to enhance the SPW transmission throughput while maintaining load balancing access throughout the system. Once configured, the SPW operates autonomously without software intervention.

In order to achieve this performance capability, one needs to understand the SPW system framework, the programming paradigm, and how to configure the module to achieve the best throughput. With this information, the programmer can apply the appropriate SPW configuration to their operational needs.

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 the SPW memory map, the associated registers, and the different mode of operation. The description of the SPW falls under the SPW Hardware sections.

After the SPW Hardware section, this application note provides a high-level flow diagram to depict the correct sequential steps to initialize the SPW. We describe each block in the order as shown in the flow diagram. The description of the flow diagram falls under the SPW Initialization sections.

Finally, we use C Programming codes to show how to configure the SPW to perform different tasks. The C code programming examples fall under the SPW Programming sections.

These subsections are described in detail below:

- SPW Hardware
- SPW Initialization
- SPW Programming

3.0 SPW Hardware

The UT700 has four identical SPW modules with Remote Memory Access Protocol support. It implements the

SpaceWire standard (ECSS-E-ST-50-12C) with the protocol identification extension (ECSS-E-ST-50-51C). The Remote Memory Access Protocol (RMAP) target implements the ECSS standard (ECSS-E-ST-50-52C). For more information about the SpaceWire refers to the UT700 Reference Manual **Chapter 12.**

The SPW module requires the system clock for transmission and an external clock for receiving data **(SPW_CLK)**. Elaboration on the external clock is provided to help clarify the ambiguity of its usage versus its terminology in a later subsection. Also, refer to the UT700 Reference Manual for the SPW memory map and registers. Important to note that configuring these SPW registers alone is not enough to enable the SPW, the SPW system framework must be setup as shown in **Figure 1.** This framework is a requirement to support the SPW high data throughput.

We elaborate each sub-block in **Figure 1** in a later subsection and show how to setup each sub-block in the SPW Initialization section.

3.1 SPW Transmit and Receiver (SPW_CLK) Clock

The SPW transmitter clock maximum frequency is 200 MHz and the Receiver clock (SPW_CLK) is used to reconstruct receiver data by sampling the data and strobe signals. The Receiver clock (SPW_CLK) frequency is governed by Equation 1 and it needs to be a multiple of 10 in order to achieve the 10 MHz start frequency.

SPW_CLK > 3/4 Receive Data Rate (max) **– Equation 1**

For example, from **Equation 1,** for 100 MHz receive data rate, the **SPW_CLK** would be 75 MHz. Since the **SPW_CLK** needs to be a multiple of 10, the **SPW_CLK** should be 80 MHz to satisfy both **Equation 1** and the requirement of multiple of 10.

Note: Receiver data is sampled on rising and fall edge of the **SPW_CLK** and the **SPW_CLK** needs to be a multiple of 10 in order to achieve the 10 MHz start up frequency.

3.2 SPW System Framework

In order to minimize software intervention and maximum data throughput, each SPW module has its own transmit and receive DMA. The DMA requires a list of information to instruct the DMA what to do and where to retrieve and store the data. This list of information is stored in the descriptors; a memory array of information resides in the main memory.

As shown in **Figure 1,** we have Receive (up to 128) and Transmit (up to 64) Descriptors for receive and transmitter DMA respectively. Each descriptor is also associated with a data buffer for the DMA to retrieve or store data.

We show how to setup this framework **(Figure 1)** in the SPW Initialization section.

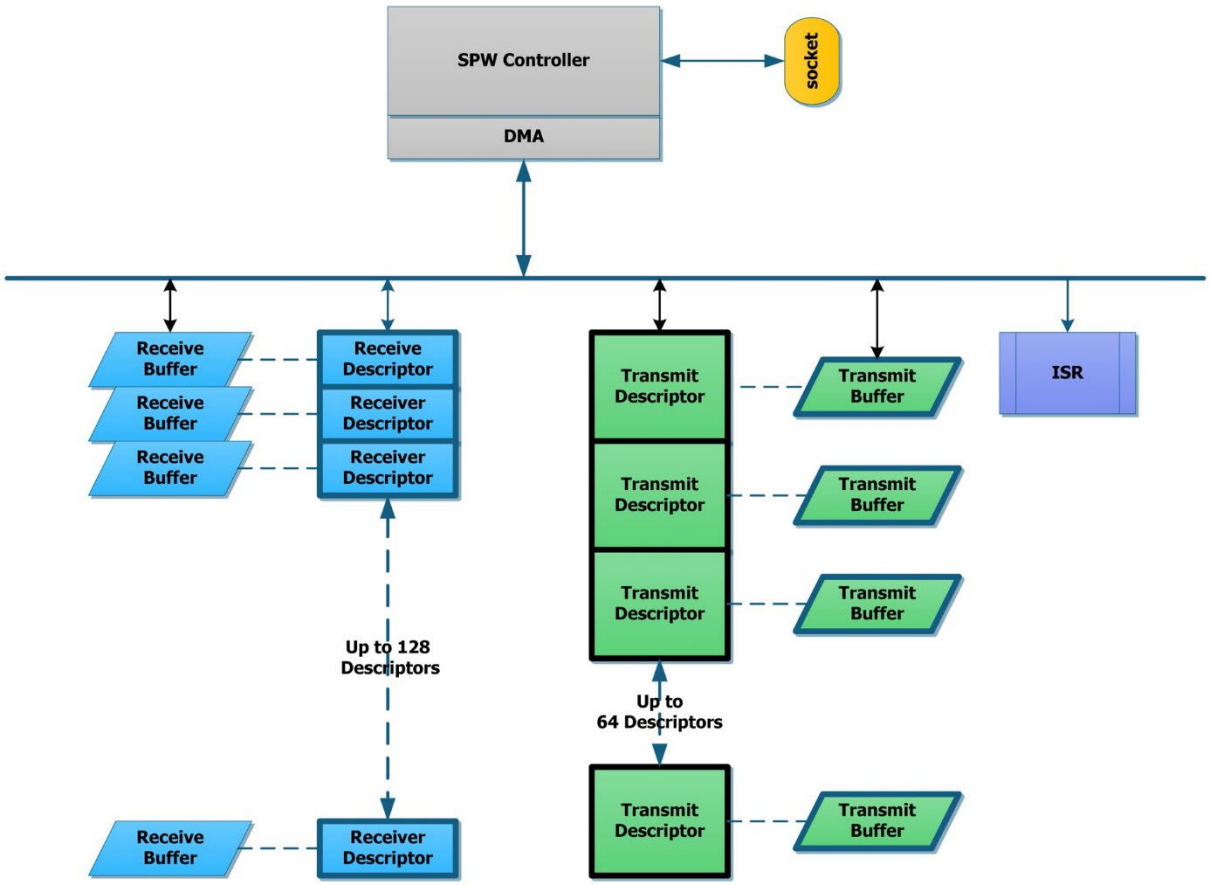

Figure 1: SpaceWire System Framework and Programming Model

3.3 SPW Receive Descriptor

The Receive Descriptor is a two words long memory structure; word 0 controls the behavior of the DMA and the length of the data, and word 1 consists of the data buffer address location.

The number of receiver descriptors is limited to 128 because of the physical constraint of the SPW Receive Descriptor Register.

Note: Maximum SPW receive descriptors are 128

3.4 SPW Transmitter Descriptor

The Transmitter Descriptor is a four words long memory structure; word 0 controls the behavior of the DMA and the length of the header, word 1 consists of the header address, word 2 consists of the data length, and word 3 consists of the data address.

The number of transmitter descriptors is limited to 64 because of the physical constraint of the SPW Transmitter Descriptor Register.

Note: Maximum SPW transmitter descriptors are 64

4.0 SPW Initialization

In this section, we explain the correct steps to initialize the SPW framework and controller in Figure1 and Figure 2 in the order as shown in Figure 2.

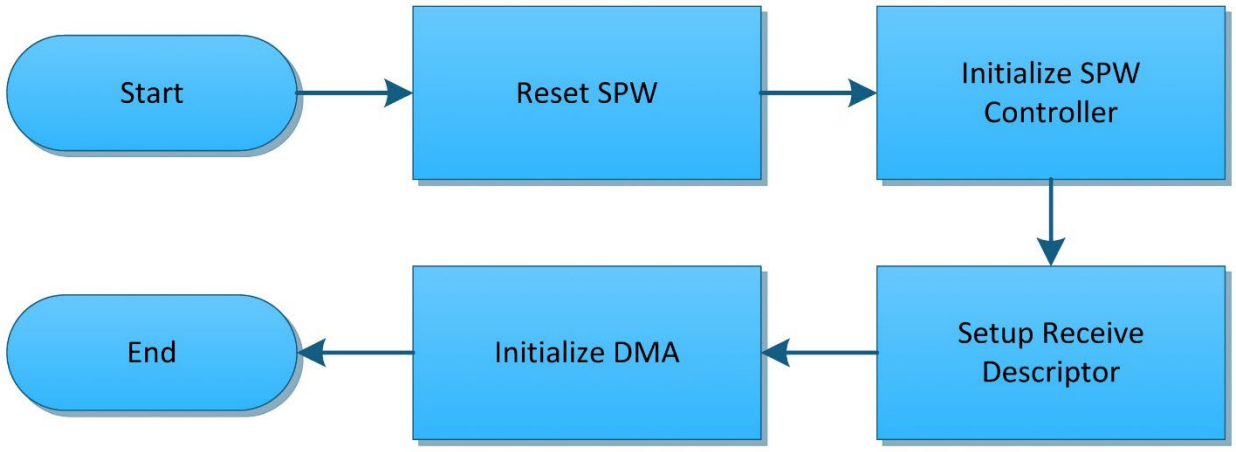

Figure 2: SpaceWire Initialization

4.1 Reset SPW Controller

By default, on power-on reset, the SPW Controller is reset. Nonetheless it is always recommended to issue a reset command to the SPW Controller before programming it.

```
        SPW->SPWCTR.B.RS = 1;          // reset the SPW Controller
```

4.2 Setup SPW Framework

The SPW controller uses the DMA to receive and transmit high data rate. In order to maintain this high data throughput, a framework was designed to provide instruction and information to the DMA. This framework **(Figure 1)** is divided into four parts; the receive descriptor, the receive data buffer, the transmit descriptor, and the transmit data buffer.

The receive **(3.3)** and transmit **(3.4)** descriptor buffers must be 1Kbyte aligned (Maximum size: 1K byte). The buffer size is based on the expected cargo size and must be word aligned.

4.2.1 Setup the Receive and Transmit Descriptors

The descriptors are setup as follows:

```
        struct TXDESCRIPTOR *txd;              // see appendix A
        struct RXDESCRIPTOR *rxd;              // see appendix A
txd = (TXDESCRIPTOR *) almalloc (1024);        // allocation buffer with 1K alignment
rxd = (RXDESCRIPTOR *) almalloc (1024);
```

4.2.2 Initialize the SPW Controller

The initialization of the SPW Controller varies depending on the port you using, the speed of the transmission, etc. In this section we provide a generic initialization for transmission on port 0 at 100 MHz.

```
SPW->SPWNAR.R = nodeaddr;                //set node address
```

During reset the clock link divider register in GRSPW2 gets its value from GPIO[7:4], which must be pulled up/down to set the divider correctly.

```
SPW->SPWCLK.B.LINK = clkdivs;          // set transmission clock divisor
SPW->SPWCLK.B.RUN = clkdiv;            // 0

SPW->SPWRXL.B.LEN = dma.rxmaxlen;      // set rx maxlength

SPW->SPWKEY.B.KEY = destkey;           // rmap

SPW->SPWCHN.R = 0xFFFE01E0;            // clear status, set ctrl for dma chan

// set tx descriptor pointer
dma.txd = (struct TXDESCRIPTOR *) almalloc(1024));

SPW->SPWTXD.B.ADDR = ((uint32_t) spw->dma.txd) >> 10;

// set rx descriptor pointer
dma.rxd = (struct RXDESCRIPTOR *) almalloc(1024))

SPW->SPWRXD.B.ADDR = ((uint32_t) spw->dma.rxd) >> 10;

SPW->SPWSTR.R = 0xFFF;                 // clear status bits

SPW->SPWCTR.B.LOOP = spw->port;        // port loop back
SPW->SPWCTR.B.RX = spw->timerxen;      // time rx enable
SPW->SPWCTR.B.TT = spw->timetxen;      // time tx enable
SPW->SPWCTR.B.AS = 1;                  // link start

SPW->SPWCHN.B.NS = spw->dma.nospill;   // set dma control
```

At this point, we have initialized the SPW controller and we are ready to proceed to the SPW Programming section.

5.0 SPW Programming

In this section, we provide programming examples how to command the SPW controller to perform specific tasks. Refer to **Appendix A** for the programming syntax used in this section and refer to the UT700 Functional Manual for more information on the registers description.

5.1 Disable the SPW

Setting a 1 to the LD bit disables the SPW; setting a 0 to the LD bit has no effect to the SPW.

```
        SPW->SPWCTR.B.LD = 1;
```

5.2 Enable the SPW

Refer to the SPW control register in Chapter 12, the appended example shows how to enable the SPW.

```
        SPW->SPWCTR.R        = SPW->SPWCTR.R & 0x220F7E;
```

5.3 Start the SPW

Setting a 1 to the LS bit enables the Start the SPW link; it allows a transition from ready to start state. Setting a 0 to the LS bit has no effect to the SPW.

```
        SPW->SPWCTR.B.LS    = 1;
```

5.4 Stop the SPW

This example shows a proper to stop the SPW.

```
        SPW->SPWCTR.R        = SPW->SPWCTR.R & 0x220F7D;
```

5.5 Set Clock Link Divisor

This example shows how to setup the link clock.

```
        if ((clkdiv < 0) || (clkdiv > 255)) return 1;
SPWCLK.B.LINK = clkdiv;
```

5.6 Set Clock Run Divisor

This example shows how to setup the run clock.

```
        if ((clkdiv < 0) || (clkdiv > 255)) return 1;
        SPWCLK.B.RUN = clkdiv;
```

5.7 Set SPW Node Address

This example shows how to setup the node address.

```
        if ((nodeaddr < 0) || (nodeaddr > 255) ||
        (mask < 0) || (mask > 255)) return 1;

        SPW->SPWNAR.B.MASK = mask;
        SPW->SPWNAR.B.ADDR = nodeaddr;
```

5.8 Set DMA Channel Address

This example shows how to setup DMA Channel Address clock.

```
        if ((dma.addr < 0) || (dma.addr > 255) ||
        (dma.mask < 0) || (dma.mask > 255)) return 1;

        SPW->SPWDADR.B.MASK = dma.mask;
        SPW->SPWDADR.B.ADDR = dma.addr;
```

5.9 Set RX Max Length

This example shows how to setup the maximum receive length.

```
        if ((dma.rxmaxlen < 4) || (dma.rxmaxlen > 33554431)) return 1;
                SPW->SPWRXL.R = dma.rxmaxlen;
```

5.10 Send Time

This example shows how to send the time stamp.

```
        while (SPW->SPWCTR.B.TI==1)
        {
           for (i = 0; i < 16; i++) {}
        }

SPW->SPWCTR.B.TI = 1;
```

5.11 Send Time Expire

This example shows how to send expiring time.

```
        code = ((ctrl << 6) & 0xC0) | time;
        while (spw->SPW->SPWCTR.B.TI==1)
        {
           for(i = 0; i < 16; i++) {}
        }

        SPW->SPWTIM.R = (SPW->SPWTIM.R & 0xFFFFFF00) | (0xFF & code);
        SPW->SPWCTR.B.TI = 1;
```

5.12 Check Time

This example shows how to check the time.

```
        uint32_t tmp = SPW->SPWSTR.B.TO;
       if (tmp)
              SPW->SPWSTR.B.TO = 1;
       return tmp;
```

5.13 Get Time

This example shows how to get the time.

```
return SPW->SPWTIM.B.COUNTER;
```

5.14 Get Time Ctrl

This example shows how to get time control.

```
return SPW->SPWTIM.B.CTRL;
```

5.15 Reset

This example shows how to reset the SPW.

```
SPW->SPWCTR.B.RS = 1;
```

5.16 RMAP Enable

This example shows how to enable the RMAP; RMAP is enabled upon Power-on Reset.

```
SPW->SPWCTR.B.RE = 1;
```

5.17 RMAP Disable

This example shows how to disable RMAP.

```
SPW->SPWCTR.B.RE = 0;
```

5.18 Set the Destination Key

This example shows how to setup the destination key.

```
if ((destkey < 0) || (destkey > 255)) return 1;
  SPW->SPWKEY.B.KEY = destkey;
```

5.19 Enable Separate Address for this Channel

This example shows how to enable separate address for this channel.

```
SPW->SPWCHN.B.EN = 1;
```

5.20 Disable Separate Address for this Channel

This example shows how to disable separate address for this channel.

```
SPW->SPWCHN.B.EN = 0;
```

5.21 Enable RX

This example shows how to enable RX.

```
SPWCHN.B.RE = 1;
```

5.22 Disable RX

This example shows how to disable RX.

```
SPWCHN.B.RE = 0;
```

5.23 Disable Promiscuous Mode

This example shows how to disable the promiscuous mode.

```
SPWCTR.B.PM = 0;
```

5.24 Enable Promiscuous Mode

This example shows how to enable the promiscuous mode.

```
SPWCTR.B.PM = 1;
```

6.0 Summary and Conclusion

This application note is not a tutorial to SpaceWire; it is an application note to reinforce the understanding of UT700 SPW registers.

Frontgrade provides SPW device drivers for RTEMS, Linux and VxWorks Operating Systems. User can download the RTEMS and Linux device drivers from our website at www.frontgrade.com.

For more information about our UT700 LEON 3FT/SPARC™ V8 Microprocessor and other products please visit our website, www.frontgrade.com or email us at https://www.frontgrade.com/contact-us.

Appendix A: Header File

This header file is designed for this application note purpose only.

```
/*****************************************************************************\
* MODULE: SpaceWire (SPW)                                                     *
\*****************************************************************************/

#ifndef__LEON_SPACEWIRE_H__
#define__LEON_SPACEWIRE_H__

#include <stdint.h>

typedef volatile uint32_t vuint32_t;

#define SPW0_BASEADDR 0x80000A00
#define SPW1_BASEADDR 0x80000B00
#define SPW2_BASEADDR 0x80000C00
#define SPW3_BASEADDR 0x80000D00

struct RXDESCRIPTOR
{
        union {
        vuint32_t R;                        // Table 12.1: Description of SpaceWire
Receive Descriptor Word 0
                struct {
                  vuint32_t TR:1;           // Truncated
                  vuint32_t DC:1;           // Data CRC
                  vuint32_t HC:1;           // Header CRC
                  vuint32_t EP:1;           // EEP Termination
                  vuint32_t IE:1;           // Interrupt Enable
                  vuint32_t WR:1;           // Wrap
                  vuint32_t EN:1;           // Enable Descriptor
                  vuint32_t LEN:25;         // number of bytes received by the buffer
                } B;
        } ctrl;//SPWRDW0

        union {
        vuint32_t R;                        // Table 12.2: Description of SpaceWire
Receive Descriptor Word 1
        struct {
                  vuint32_t ADDR:32;        // address pointing to the buffer
                } B;
  } daddr;      //SPWRDW1
};
struct TXDESCRIPTOR
{
```

```
        union {
        vuint32_t R;    // Table 12.3: Description of SpaceWire Transmitter Descriptor
Word 0
                struct {
                  vuint32_t REV31_18:14;    // Reserved
                  vuint32_t DC:1;           // Append data CRC
                  vuint32_t HC:1;           // Append header CRC
                  vuint32_t LE:1;           // Link Error
                  vuint32_t IE:1;           // Interrupt Enable
                  vuint32_t WR:1;           // Wrap
                  vuint32_t EN:1;           // Enable
                  vuint32_t NON_CRC_BYTE:4;// not be included in the CRC calculation
                  vuint32_t HEADER_LEN:8;   // Header length in bytes
                } B;
  } ctrl;

        union {
        vuint32_t R;    // Table 12.4: Description of SpaceWire Transmitter Descriptor
Word 1
                struct {
                  vuint32_t ADDR:32;        // Address from where the packet header
                } B;
  } haddr;

        union {
        vuint32_t R;    // Table 12.5: Description of SpaceWire Transmitter Descriptor
Word 2
                struct {
                  vuint32_t REV31_24:8;     // Reserved
                  vuint32_t DATA_LEN:24;    // Length of data part of the packet in
bytes
                } B;
  }dlen;

        union {
        vuint32_t R;    // Table 12.6: Description of SpaceWire Transmitter Descriptor
Word 3
                struct {
                  vuint32_t DATA_ADDR:32;   // Address from where data is read
                } B;
  } daddr;
};
```

```
//.....
// GRSPW2 Registers
//.....
struct SPW_TAG {                // Table 12.9: GRSPW2 Registers
        union {
        vuint32_t R;            // Table 12.10: Description of SpaceWire Control
Register

                struct{
                  vuint32_t RA:1;           // RMAP Available
                  vuint32_t RX:1;           // RX Unaligned Access
                  vuint32_t RC:1;           // RMAP CRC Available
                  vuint32_t REV28_27:2;     // Reserved
                  vuint32_t PO:1;           // Number of available SpaceWire ports
minus 1
                  vuint32_t REV25_23:3;     // Reserved
                  vuint32_t LOOP:1;         // Port loop back:
                  vuint32_t REV21_18:4;     // Reserved
                  vuint32_t RD:1;           // RMAP Buffer Disable
                  vuint32_t RE:1;           // RMAP Enable
                  vuint32_t REV15_12:4;     // Reserved
                  vuint32_t TR:1;           // Time RX Enable
                  vuint32_t TT:1;           // Time TX Enable
                  vuint32_t LI:1;           // Link Error IRQ
                  vuint32_t TQ:1;           // Tick-Out IRQ
                  vuint32_t REV7:1;         // Reserved
                  vuint32_t RS:1;           // Reset
                  vuint32_t PM:1;           // Promiscuous Mode
                  vuint32_t TI:1;           // Tick In
                  vuint32_t IE:1;           // Interrupt Enable
                  vuint32_t AS:1;           // AutoStart
                  vuint32_t LS:1;           // Link Start
                  vuint32_t LD:1;           // Link Disable
                } B;
        } SPWCTR;

        union {
        vuint32_t R;            // Table 12.11: Description of SpaceWire Status
Register
                struct {
                  vuint32_t REV31_24:8;     // Reserved
                  vuint32_t LS:3;           // Link State
                  vuint32_t REV20_10:11;    // Reserved
                  vuint32_t AP:1;           // Active port
                  vuint32_t EE:1;           // Early EOP/EEP Read
                  vuint32_t IA:1;           // Invalid Address Read
                  vuint32_t REV6_5:2;       // Reserved
                  vuint32_t PE:1;           // Parity Error Read
                  vuint32_t DE:1;           // Disconnect Error Read
                  vuint32_t ER:1;           // Escape Error Read
                  vuint32_t CS:1;           // Credit Error Read
                  vuint32_t TO:1;           // Tick Out Read
```

```
                } B;
        } SPWSTR;

        union {
        vuint32_t R;    // Table 12.12: Description of SpaceWire Node Address Register
                struct {
                  vuint32_t REV31_16:16;    // Reserved
                  vuint32_t MASK:8;         // Address mask
                  vuint32_t ADDR:8;         // node address
                } B;
        } SPWNAR;

        union {
        vuint32_t R;    // Table 12.13: Description of SpaceWire Clock Divisor
Register
                struct {
                  vuint32_t REV31_16:16;    // Reserved
                  vuint32_t LINK:8;         // 8-Bit Clock Divisor Link State Value
                  vuint32_t RUN:8;          // 8-Bit Clock Divisor Run State Value
                } B;
        } SPWCLK;

        union {
        vuint32_t R;    // Table 12.14: Description of SpaceWire Destination Key
Register
                struct {
                  vuint32_t REV31_8:24;     // Reserved
                  vuint32_t KEY:8;          // RMAP Destination Key
                } B;
        } SPWKEY;

        union {
        vuint32_t R;    // Table 12.15: Description of SpaceWire Time Register
                struct {
                  vuint32_t REV31_8:24;     // Reserved
                  vuint32_t CTRL:2;         // Time Control Flags
                  vuint32_t COUNTER:6;      // Time Counter
                } B;
        } SPWTIM;

        union {

        vuint32_t Padding18;                // Padding
        vuint32_t Padding1C;                // Padding

        union {
        vuint32_t R;    // Table 12.16: Description of SpaceWire DMA Channel Receiver
Max Length Register
                struct {
                  vuint32_t REV31_17:15;    // Reserved
                  vuint32_t LE:1;           // Link error disable
                  vuint32_t SP:1;           // Strip PID
```

```
                vuint32_t SA:1;          // Strip Address
                vuint32_t EN:1;          // Enable Address
                vuint32_t NS:1;          // No Spill
                vuint32_t RD:1;          // RX Descriptors Available Read
                vuint32_t RX:1;          // RX Active
                vuint32_t AT:1;          // Abort TX
                vuint32_t RA:1;          // RX AHB Error
                vuint32_t TA:1;          // TX AHB Error
                vuint32_t PR:1;          // Packet Received 0: No new packet
                vuint32_t PS:1;          // Packet Sent
                vuint32_t AI:1;          // AHB Error Interrupt
                vuint32_t RI:1;          // Receive Interrupt
                vuint32_t TI:1;          // Transmit Interrupt
                vuint32_t RE:1;          // Receiver Enable
                vuint32_t TE:1;          // Transmitter Enable Read
              } B;
        } SPWCHN;

        union {
        vuint32_t R;    // Table 12.17: Description of SpaceWire DMA Channel Receiver
Max Length
              struct {
                vuint32_t REV31_25:7;    // Reserved
                vuint32_t LEN:25;        // Receiver Packet Maximum Length
              } B;
        } SPWRXL;

        union {
        vuint32_t R;// Table 12.18: Description of SpaceWire Transmitter Descriptor
Register
              struct {
                vuint32_t ADDR:22;       // Transmitter Descriptor Table Base
Address
                vuint32_t SEL:6;         // Transmitter Descriptor Selector
                vuint32_t REV3_0:4;      // Reserved
              } B;
        } SPWTXD;

        union {
        vuint32_t R;    // Table 12.19: Description of SpaceWire Receiver Descriptor
Register
              struct {
                vuint32_t ADDR:22;       // Receiver Descriptor Table Base Address
                vuint32_t SEL:7;         // Receiver Descriptor Selector
                vuint32_t REV2_0:3;      // Reserved
              } B;
        } SPWRXD;

        union {
        vuint32_t R;    // Table 12.20: Description of SpaceWire DMA Channel address
register
```

```
                struct {
                  vuint32_t REV31_16:16;    // Reserved
                  vuint32_t MASK:8;         // Mask
                  vuint32_t ADDR:8;         // Address
                } B;
        } SPWDADR;
};

typedef volatile struct SPW_TAG SPWPORT;

#define SPW0 (*(volatile struct SPW_TAG*) SPW0_BASEADDR)
#define SPW1 (*(volatile struct SPW_TAG*) SPW1_BASEADDR)
#define SPW2 (*(volatile struct SPW_TAG*) SPW2_BASEADDR)
#define SPW3 (*(volatile struct SPW_TAG*) SPW3_BASEADDR)

#endif
```

Revision History

Date	Revision #	Author	Change Description	Page #
06/23/2017	1.0.0	MTS	Initial Release	
09/06/2017	1.0.1	MTS	Remove 3.1	
09/29/2017	1.0.2	MTS	Add note, page 1	