Gaisler Reseach

GRSPW Spacewire Codec IP Core

Based on GRLIB-1.0.6

Jiri Gaisler, Marko Isomaki

1 Introduction

1.1 Overview

The GRSPW core implements a Spacewire Codec with RMAP support and AMBA host interface. The core implements the Spacewire standard with the protocol identification extension (ECSS-E-50-12 part 2) and RMAP protocol draft C. Receive and transmit data is autonomously transferred between the Spacewire Codec and the AMBA AHB bus using DMA transfers. Through the use of receive and transmit descriptors, multiple Spacewire packets can be received and transmitted without CPU involvement. The GRSPW control registers are accessed through an APB interface. For critical space applications, a fault-tolerant version of GRSPW is available with full SEU protection of all RAM blocks.

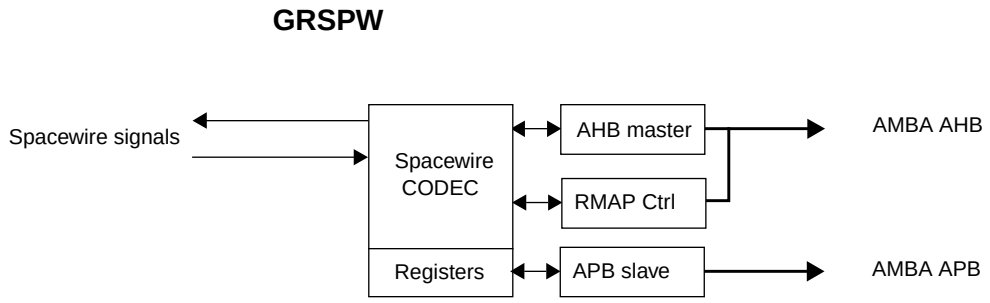


Figure 1. GRSPW block diagram

1.2 Implementation characteristics

The GRSPW is inherently portable and can be implemented on most FPGA and ASIC technologies. The table below shows the approximate Cell/LUT count and frequency for four different GRSPW configurations on Actel RTAX, Xilinx Virtex2 and ASIC technologies.

TABLE 1. Implementation characteristics (Cells / RAM blocks / AHB MHz / SPW MHz)

Core configuration	RTAX	Virtex2	ASIC
GRSPW	2,800 / 2 / 40 / 100	1,900 / 3 / 80 / 200	10,000 gates
GRSPW + RMAP	3,600 / 2 / 40 / 100	2,800 / 3 / 80 / 200	15,000 gates
GRSPW-FT	2,900 / 4 / 40 / 100	2,000 / 6 / 80 / 200	11,000 gates
GRSPW-FT + RMAP	3,700 / 4 / 40 / 100	2,900 / 6 / 80 / 200	16,000 gates

The GRSPW core is available in VHDL source code or as a pre-synthesized netlist. It can be delivered for stand-alone operation or with a wrapper for GRLIB AMBA plug&play interface.

2 GRSPW - SpaceWire codec with AHB host interface and RMAP support

2.1 Overview

The GRSPW provides an interface between an AHB bus and a SpaceWire network. It implements the SpaceWire standard with the protocol identification extension (ECSS-E-50-12 part 2) and there is also an optional Remote Access Memory Protocol (RMAP) command handler. The RMAP handler implements draft C of the standard.

The GRSPW is configured through a set of registers accessed through an APB interface. Data is transferred through DMA channels using an AHB master interface. Currently, there is one DMA channel but the core can easily be extended to use separate DMA channels for specific protocols.

There are three clock domains: one for the AHB interface (system clock), one for the transmitter and one for the receiver. The receiver clock can be twice as fast and the transmitter clock four times as fast as the system clock whose frequency should be at least 10 MHz.

The core only supports byte addressed 32-bit big-endian host systems.

Figure 2 shows a block diagram of the GRSPW module.

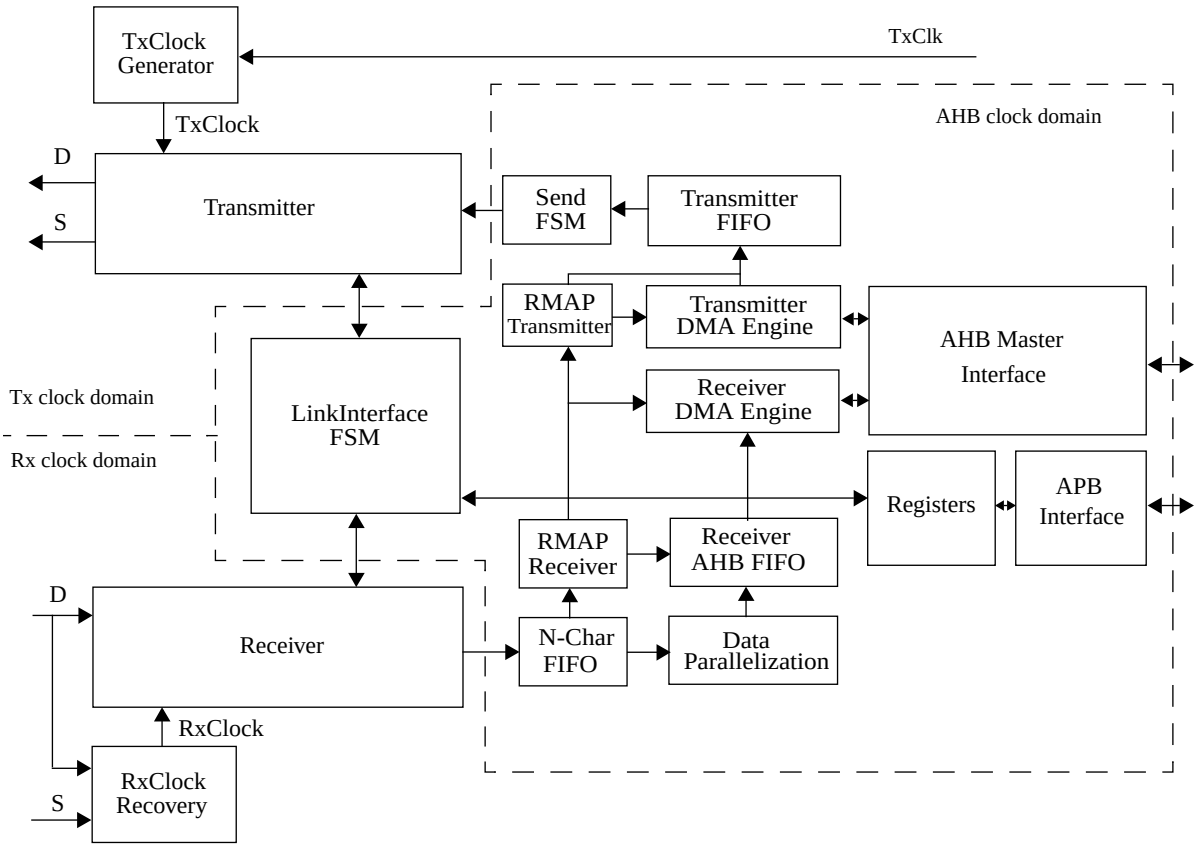


Figure 2. GRSPW block diagram.

2.2 Operation

2.2.1 Overview

The GRSPW can be split into three main parts: the link interface, the AMBA interface and the RMAP handler. A block diagram of the internal structure can be found in figure 2.

The link interface consists of the receiver, transmitter and the link interface FSM. They handle communication on the SpaceWire network. The AMBA interface consists of the DMA engines, the AHB master interface and the APB interface. The link interface provides FIFO interfaces to the DMA engines. These FIFOs are used to transfer N-Chars between the AMBA and SpaceWire domains during reception and transmission.

The RMAP handler is an optional part of the GRSPW which can be enabled with a generic. It handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The GRSPW is controlled by writing to a set of user registers through the APB interface and through one signal (tick-in). The controlled parts are clock-generation, DMA engines, RMAP handler and the link interface.

The link interface, DMA engines, RMAP handler and AMBA interface are described in section 2.3, 2.4, 2.6 and 2.7 respectively.

2.2.2 Protocol support

The GRSPW only accepts packets with a destination address corresponding to the one set in the node address register. Packets with address mismatch will be silently discarded. The node address register is initialized to the default address 254 during reset. It can then be changed to some other value by writing to the register.

The GRSPW also requires that the byte following the destination address is a protocol identifier as specified in part 2 of the SpaceWire standard. It is used to determine to which DMA-channel a packet is destined. Currently only one channel is available to which all packets (except RMAP commands) are stored but the GRSPW is prepared to be easily expandable with more DMA channels. Figure 3 shows the packet type expected by the GRSPW.

RMAP (Protocol ID = 0x01) commands are handled separately from other packets if the hardware RMAP handler is enabled. When enabled, all RMAP commands are processed, executed and replied in hardware. All RMAP replies received are still stored to the DMA channel. If the RMAP handler is disabled, all packets are stored to the DMA channel. More information on the RMAP protocol support is found in section 2.6.

All packets arriving with the extended protocol ID (0x00) are dumped to the DMA channel. This means that the hardware RMAP command handler will not work if the incoming RMAP packets use the extended protocol ID. Note also that the reserved extended protocol identifier (ID = 0x000000) are not ignored by the GRSPW. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the GRSPW.

Figure 3 shows a packet with a normal protocol identifier. The GRSPW also allows reception and transmission with extended protocol identifiers but then the hardware RMAP CRC calculations will not work.

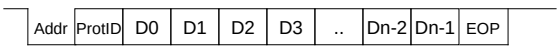


Figure 3. The SpaceWire packet with protocol ID that is expected by the GRSPW.

2.3 Link interface

The link interface handles the communication on the SpaceWire network and consists of a transmitter, receiver, a FSM and FIFO interfaces. An overview of the architecture is found in figure 2.

2.3.1 Link interface FSM

The FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM in the host domain handles the exchange level.

The link interface FSM is controlled through the control register. The link can be disabled through the link disable bit, which depending on the current state, either prevents the link interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started state when either the link start bit is set or when a NULL character has been received and the autostart bit is set.

The current state of the link interface determines which type of characters are allowed to be transmitted which together with the requests made from the host interfaces determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 2.3.4).

When the link interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver N-Char FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested or the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO for further handling by the DMA interface. Received Time-codes are handled by the time-interface.

2.3.2 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the DMA-interface are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because one usually wants to run the SpaceWire link on a different frequency than the host system clock. The GRSPW has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 2.8.1. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 4. The transmitter handles sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the trans-

mitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals.

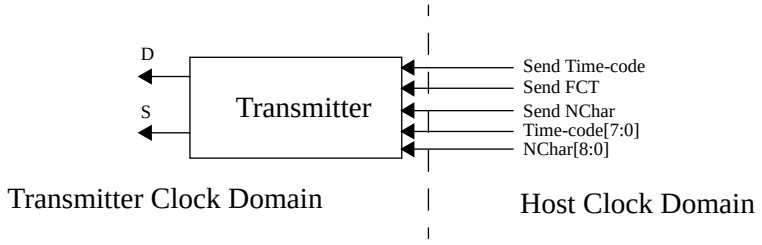


Figure 4. Schematic of the link interface transmitter.

A transmission FSM reads N-Chars for transmission from the transmitter FIFO. It is given packet lengths from the DMA interface and appends EOPs/EEPs and RMAP CRC values if requested. When it is finished with a packet the DMA interface is notified and a new packet length value is given.

2.3.3 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream on the data and strobe signals. It is also located in a separate clock domain which runs on a clock generated from the received data and strobe signals. More information on the clock-generation can be found in section 2.8.1.

The receiver is activated as soon as the link interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error reset state. Disconnections are handled in the link interface part in the system clock domain because no receiver clock is available when disconnected.

Received Characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 5. L-Chars are the handled automatically by the host domain link interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first are discarded.

There are no signals going directly from the transmitter clock domain to the receiver clock domain and vice versa. All the synchronization is done to the system clock.

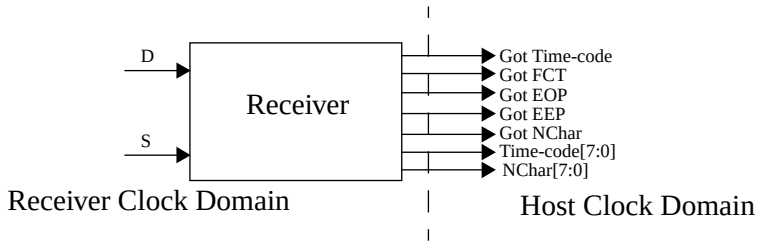


Figure 5. Schematic of the link interface receiver.

2.3.4 Time interface

The time interface consists of a time-counter register, tick-in signal, tick-out signal, tick-in register field and a tick-out register field. The time-counter register is set to 0 after reset and is incremented each time the tick-in signal is asserted for one clock-period. This also causes the link interface to send the new value on the network. Tick-in can be generated either by writing a one to the register field or by asserting the tick-in signal. A Tick-in should not be generated to often since if the time-code after the previous Tick-in has not been sent the register will not be incremented and no new value will be sent. The tick-in field is automatically cleared when

the value has been sent and thus no new ticks should be generated until this field is zero. If the tick-in signal is used there should be at least 4 system-clock and 25 transmit-clock cycles between each assertion.

A tick-out is generated each time a valid time-code is received. When the tick-out is generated the tick-out signal will be asserted one clock-cycle and the tick-out register field is asserted until it is read, which causes it to clear itself.

The current time counter value can be read from the status register but cannot be written. The only way to affect the counter value from the host is to generate a tick-in.

The two control-bits are always set to zero for transmitted time-codes and received control bits are discarded. There is no user-readable register containing the control-bits.

2.4 Receiver DMA engine

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels. Currently there is only one receive DMA channel available but the GRSPW has been written so that additional channels can be easily added if needed.

2.4.1 Basic functionality

The receiver DMA engine reads N-Chars from the N-Char FIFO and stores them to a DMA channel. Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the GRSPW it reads a descriptor from memory and stores the packet to the memory area pointed to by the descriptor. Then it stores status to the same descriptor and increments the descriptor pointer to the next one.

2.4.2 Setting up the GRSPW for reception

A few registers need to be initialized before reception can take place. First the link interface need to be put in the run state before any data can be sent. The DMA channel has a maximum length register which sets the maximum size of packet that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only be incremented in steps of four bytes. If the maximum length is set to zero the receiver will *not* function correctly.

The node address register needs to be set to hold the address of this SpaceWire node. Packets received with the incorrect address are discarded. Finally, the descriptor table and control register must be initialized. This will be described in the two following sections.

2.4.3 Setting up the descriptor table address

The GRSPW reads descriptors from a area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on a 1 kB aligned address. It is also limited to be 1 kB in size which means the maximum number of descriptors is 128.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap automatically by setting a bit in the descriptors. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 B aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

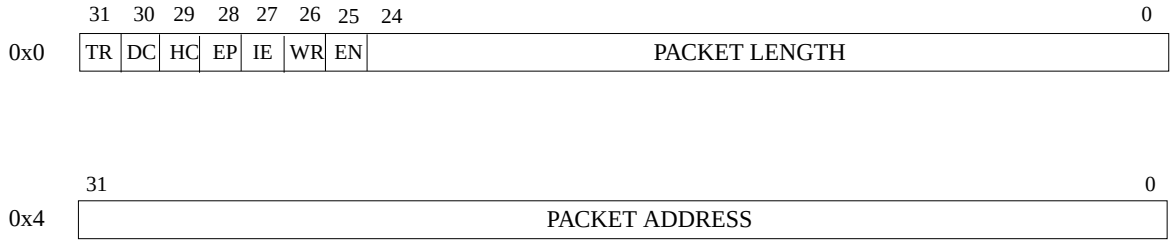
If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

2.4.4 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 B in size and the layout is shown in figure 6. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for this to happen.

The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the rxunaligned or rmap generic is set to 1 this restriction is removed and a number of bytes can be received to any packet address without excessive bytes being overwritten.



24-0: Packet Length - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW.

25: Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.

26: Wrap (WR) - If set, the next descriptor used by the GRSPW will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kB in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kB boundary.

27: Interrupt Enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.

28: EEP Termination (EP) - This packet ended with an Error End of Packet character.

29: Header CRC (HC) - 1 if a CRC error was detected for the header and 0 otherwise.

30: Data CRC (DC) - 1 if a CRC error was detected for the data and 0 otherwise.

31: Truncated (TR) - Packet was truncated due to maximum length violation.

31-0: Packet Address - The address pointing at the buffer which will be used to store the received packet. If the rxunaligned and rmap generics are both set to zero only bit 31 to 2 are used.

Figure 6. SpaceWire Receive descriptor.
Address offsets are shown in the left margin.

2.4.5 Setting up the DMA control register

To final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 2.11). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the GRSPW might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

2.4.6 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded. If the receiver is enabled the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is ‘0’ and the nospill bit is ‘0’ the packets will be discarded. If nospill is one the grspw waits until rxdescav is set.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to ‘0’ and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will cause all packets received afterwards to be discarded. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but no more descriptors will be read until it is set again. Rxdescav is also cleared by the GRSPW when it reads a disabled descriptor.

2.4.7 Address recognition and packet handling

When the receiver N-Char FIFO is not empty, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address which is compared to the node address register. If it does not match, the complete packet is discarded (up to and including the next EOP/EEP). Otherwise the next action taken depends on whether the node is configured with RMAP or not. If RMAP is disabled all packets are stored to the DMA channel and depending in the conditions mentioned in the previous section, the packet will be received or not. If the packet is received complete packet including address and protocol ID but excluding EOP/EEP is stored to the address indicated in the descriptor, otherwise the complete packet is discarded.

If RMAP is enabled the protocol ID and 3rd byte in the packet is first checked before any decisions are made. If incoming packet is an RMAP packet (ID = 0x01) and the command type field is 01b the packet is processed by the RMAP command handler which is described in section 2.6. Otherwise the packet is processed by the DMA engine as when RMAP is disabled.

At least 2 non EOP/EEP N-Chars needs to be received for a packet to be stored to the DMA channel. If it is an RMAP packet with hardware RMAP enabled 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than these sizes are discarded.

2.4.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The GRSPW can also be made to generate an interrupt for this event as mentioned in section.

RMAP CRC is always checked for all packets if the CRC logic is included in the implementation (rmapcrc or rmap generic set to 1). If the received packet is not of RMAP type the CRC error indication bits in the descriptor should be ignored. If the received packet is of RMAP type the bits are valid and the HC bit is set if a header CRC error was detected. In this case, the data CRC will not be calculated at all and the DC bit is undefined. If the header CRC was correct the DC bit will also contain a valid value and is set to one if a data CRC error was detected.

2.4.9 Error handling

If a packet reception needs to be aborted because of congestion on the network, the suggested solution is to set link disable to ‘1’. Unfortunately, this will also cause the packet currently being transmitted to be truncated but this is the only safe solution since packet reception is a passive operation depending on the transmitter at the other end. A channel reset bit could be provided but is not a sat-

isfactory solution since the untransmitted characters would still be in the transmitter node. The next character (somewhere in the middle of the packet) would be interpreted as the node address which would probably cause the packet to be discarded but not with 100% certainty. Usually this action is performed when a reception has stuck because of the transmitter not providing more data. The channel reset would not resolve this congestion.

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

2.5 Transmitter DMA engine

The transmitter DMA engine handles transmission of data from the DMA channel to the SpaceWire network. Currently there is only one DMA channel available but the GRSPW has been written so that additional DMA channels can be easily added if needed.

2.5.1 Basic functionality

The transmit DMA engine reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the GRSPW reads them and transfer the amount data indicated.

2.5.2 Setting up the GRSPW for transmission

Four steps need to be performed before transmissions can be done with the GRSPW. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register is written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

2.5.3 Enabling descriptors

The descriptor table address register works in the same way as the receiver's corresponding register which was covered in section 2.4.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 B and the Maximum data length is 16 Mb - 1. When the pointer and length fields have been set the enable should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

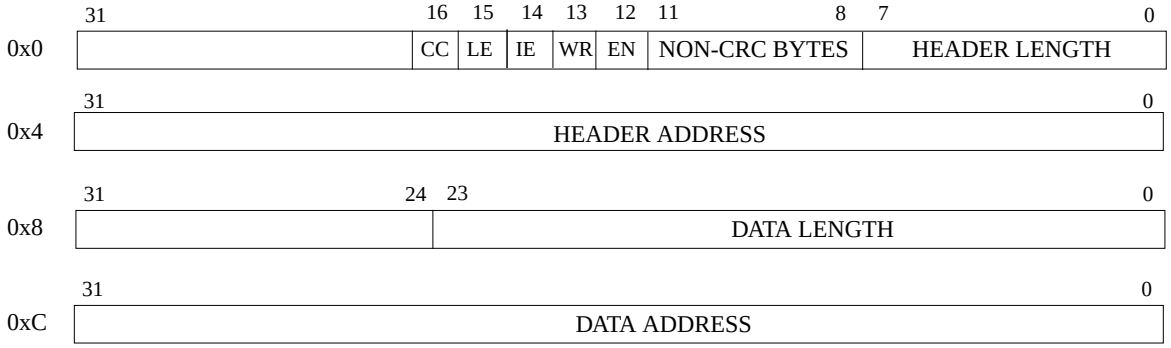
The transmit descriptors are 16 B in size so the maximum number in a single table is 64. The different fields of the descriptor is shown in figure 8 together with the memory offsets.

The CC field should be set if RMAP CRC should be calculated and inserted for the current packet. This field is only used by the GRSPW if the CRC logic is available (rmap or rmapcrc generic set to 1). The first CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation.

The CRC is skipped if the corresponding field is zero. If both fields are zero nothing will be sent not even an EOP.

2.5.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the GRSPW to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable register bit should be set. This has to be done because each time the GRSPW encounters a disabled descriptor this register bit is set to 0.



7-0: Header Length - Header Length in bytes. If set to zero, the header is skipped.

11-8: Non-CRC bytes - Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.

12: Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The GRSPW clears this bit when the transmission has finished.

13: Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.

14: Interrupt Enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.

15: Link Error (LE) - A Link error occurred during the transmission of this packet.

16: Calculate CRC (CC) - If set, two CRC values according to the RMAP specification will be generated and appended to the packet. The first CRC will be appended after the data pointed to by the header address field and the second is appended after the data pointed to by the data address field.

31-0: Header Address - Address from where the packet header is fetched. Does not need to be word aligned.

23-0: Data Length - Length of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

31-0: Data Address - Address from where data is read. Does not need to be word aligned.

Figure 8. SpaceWire Transmitter descriptor.
Address offsets are shown in the left margin.

2.5.5 The transmissions process

When the txen bit is set the GRSPW starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

2.5.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the 1 kB limit for the descriptor table is reached or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

2.5.7 Error handling

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 waitstates). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

When an AHB error is encountered during transmission the currently active DMA channel is disabled, the packet is truncated and an EEP is inserted (if the transmission has started) and the transmitter goes to the idle mode. A bit in the DMA channel's control/status register is set to indicate this error condition. The client using the channel has to correct the error and enable the channel again.

2.6 RMAP

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire Link. Some common operations are reading and writing to memory, registers and FIFOs. The GRSPW has an optional hardware RMAP command handler which is enabled with a generic. This section describes the basics of the RMAP protocol and the command handler implementation.

2.6.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations write, read and read-modify-write. These operations are posted operations which means that a source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must be implemented in the user application which sends the commands. Data payloads of up to 16 Mb - 1 is supported in the protocol. A destination can be requested to send replies and to verify data before executing an operation. A complete description of the protocol is found in the RMAP standard.

2.6.2 Implementation

The GRSPW includes an handler for RMAP commands which processes all incoming packets with protocol ID = 0x01 and type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver which is seen in figure 2.

The GRSPW implements all three commands defined in the standard with some restrictions. The implementation is based on draft C of the RMAP standard. Support is only provided for 32-bit big-endian systems. This means that the first byte received is the msb in a word. The command handler will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

Packets with a mismatching destination logical address are never passed to the RMAP handler. There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to '1'.

Detection of all error codes is supported. When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the GRSPW. It is shown in table 2.

TABLE 2. The order of error detection in case of multiple errors in the GRSPW. The error detected first has number 1.

Detection Order	Error Code	Error
1	2	RMAP command not supported by node
2	3	Invalid destination key
3	11	RMW data length error
4	9	Verify buffer overrun
5	10	Authorization failure
6	5/6	Early EOP/EEP
7	4	Invalid data CRC
8	7/8	Late EOP/EEP

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmitting. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

2.6.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 B and the address must be aligned to the size. That is 1 B writes can be done to any address, 2 B must be halfword aligned, 3 B are not allowed and 4 B writes must be word aligned. Since there will always be only on AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

2.6.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero.

2.6.5 RMW commands

All read-modify-write sizes are supported except 6 which will lead to 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command,

2.6.6 Control

The RMAP command handler mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the GRSPW which can be used to completely disable the RMAP command handler. When it is set to ‘0’ no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the command handler stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the command handler to only use one buffer which prevents this situation

The last control option for the command handler is the possibility to set the destination key which is found in a separate register.

TABLE 3. GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command /Response	Write /Read	Verify data before write	Acknowledge	Increment Address		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incrementing address.	Executed normally. No restrictions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Modify-Write incrementing address	Executed normally. If length is not one of the allowed rmw values nothing is done and error code is set to 11. If the length was correct, alignment restrictions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, single-address, do not verify before writing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incrementing address, do not verify before writing, no acknowledge	Executed normally. No restrictions. No reply is sent.
0	1	1	0	1	0	Write, single-address, do not verify before writing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	1	1	Write, incrementing address, do not verify before writing, send acknowledge	Executed normally. No restrictions. If AHB error occurs error code is set to 1. Reply is sent.

TABLE 3. GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command /Response	Write /Read	Verify data before write	Acknowledge	Increment Address		
0	1	1	1	0	0	Write, single address, verify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restrictions apply as for rmw. No reply is sent.
0	1	1	1	0	1	Write, incrementing address, verify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restrictions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, verify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incrementing address, verify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

2.7 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 2.11. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the fifo size in length. The last burst might be shorter. If the rmap or rxunaligned generics are set to 1 there might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

2.7.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

2.7.2 AHB master interface

The GRSPW contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner

requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

The AHB accesses are always word accesses (HSIZE = 0x010) of type incremental burst with unspecified length (HBURST = 0x001) if rmap and rxunaligned are disabled. Otherwise the accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010). Byte and halfword accesses are always NONSEQ.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the GRSPW does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the core provides full support for ERROR, RETRY and SPLIT responses.

2.8 Synthesis and hardware issues

2.8.1 Clock-generation

Figure 10 shows the clock recovery scheme for the receiver. Data and strobe are coupled directly from their pads to an xor gate which generates the clock. The output from the xor is then connected to a clock network. The specific type of clock network depends on the technology used. The xor gate is actually all that logically belongs to the Rx clock recovery module in figure 2.

The clock output drives all flip-flops in the receiver module found in figure 2. The data signal which is used for generating the clock is also coupled to the data inputs of several flip-flops clocked by the Rx clock as seen in figure 10. Care must be taken so that the delay from the data and strobe signals through the clock network are longer than the delay to the data input + setup time.

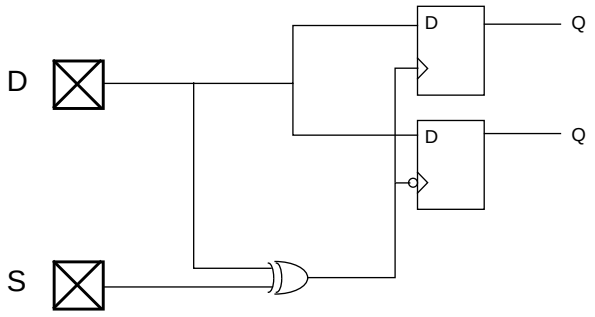


Figure 10. The clocking scheme for the receiver. The clock is generated from the data and strobe signals using an xor gate.

The transmitter clock is generated from the txclk input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the txclk signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

The txfreq generic should be set to the frequency in kHz txclk input. It is used to determine the value of clock select and the clock divisor value (see figure 11) during initialization. In the run-state these values are taken from the user accessible registers.

The clock divider generates the following frequencies: txclk/(2*(divisor+1)). The minimum divisor is 2 (the effective value, not the one entered in the register) so if the maximum frequency is wanted the txclk must be coupled directly to the transmitter. This is done with the maxfreqen bit in the ctrl register. Only even values are allowed so only 10, 20, 40, 60, 80,... MHz txclks are allowed if the SpaceWire standard is not to be violated (requires initialization frequency of 10 MHz).

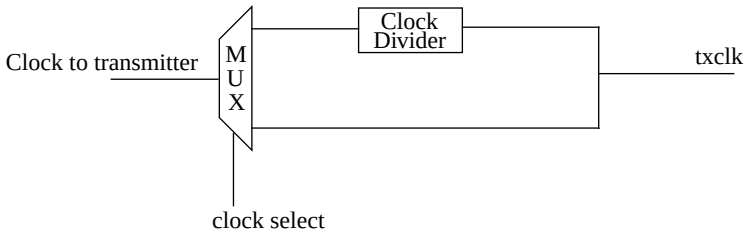


Figure 11. The transmitter clock generator.

2.8.2 Synchronization

The system clock frequency must be at least 10 MHz to guarantee disconnect timing limits. The sysfreq generic is set to the clk input frequency in kHz. It is used to generate the disconnect, 6.4 us and 12.8 us timers. If the system clock frequency is 10 MHz or above the disconnect time will be within the limits specified in the SpaceWire standard.

The generic nsync selects how many synchronization registers are used between clock domains. The default is one and should be used when maximum performance is needed. It allows the transmitter to be clocked 4 times faster than the system clock and the receiver 2 times faster. These are theoretical values without consideration for clock skew and jitter. Note also that the receiver clocks data at both negative and positive edges. Thus, the bitrate is twice as high as the clock-rate.

The synchronization limits the Tx and Rx clocks to be at most 4 and 2 times faster than the system clock. But it might not be possible to achieve such high clock rates for the Tx and Rx clocks for all technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses has a completely asynchronous reset. To make sure that nothing bad happens the is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

2.8.3 Fault-tolerance

The GRSPW core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*). Note: the GPL version of GRLIB does not include fault-tolerance, and the GRSPW core will not work unless the *ft* generic is 0.

2.8.4 Synthesis

Since the receiver and transmitter may run on very high frequency clocks their clock signals have been coupled through a clock buffer with a technology wrapper. This clock buffer will utilize a low skew net available in the selected technology for the clock.

The clock buffer will also enable most synthesis tools to recognize the clocks and it is thus easier to find them and place constraints on them. The fact there are three clock domains in the GRSPW of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths.

In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

The type of clock buffer is selectable with a generic and the value zero provides a normal feed through which lets the synthesis tool infer the type of net used.

2.9 Configuration options

The GRSPW has the following configuration options (VHDL generics):

TABLE 4. GRSPW configuration options (VHDL generics)

Generic	Function	Allowed range	Default
tech	Technology for fifo memories.	0 - NTECH	inferred
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by GRSPW.	0 - NAHBIRQ-1	0
sysfreq	Frequency of clock input “clk” in kHz.	-	10000
txfreq	Frequency of clock input “txclk” in kHz.	-	10000
nsync	Number of synchronization registers.	1 - 2	1
rmap	Include hardware RMAP command handler. RMAP CRC logic will also be added.	0 - 1	0
rmapcrc	Include RMAP CRC logic in the core. Only needs to be set if the RMAP generic is set to 0 since it is included automatically in the other case.	0 - 1	0
fifosize1	Sets the number of entries in the 32-bit receiver and transmitter AHB fifos.	4 - 32	32
fifosize2	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 64	64
rxclk-buftype	Select clock buffer type for receiver clock. 0 selects hardwired clock while 1 selects routed clock.	0 - 2	0
txclk-buftype	Select clock buffer type for transmitter clock. 0 selects hardwired clock while 1 selects routed clock.	0 - 2	0
rxu-naligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes.	0 - 1	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
ft	Enable fault-tolerance against SEU errors	0 - 2	0

2.10 Vendor and device id

The module has vendor ID 0x01 (Gaisler Research) and device ID 0x1F. For description of vendor and device IDs see GRLIB IP Library User’s Manual.

2.11 Registers

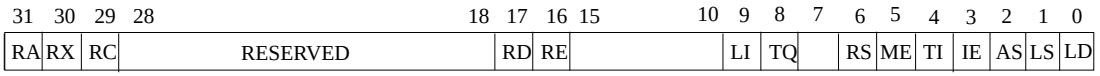
The GRSPW is programmed through 9 registers mapped into APB address space.

TABLE 5. GRSPW registers

Register	APB Address offset
Control	0x0
Status/Interrupt-source	0x4
Node address	0x8

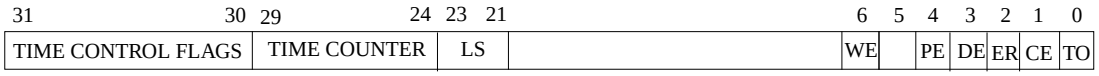
TABLE 5. GRSPW registers

Register	APB Address offset
Clock divisor	0xC
Destination key	0x10
DMA channel 1 control/status	0x20
DMA channel 1 rx maximum length	0x24
DMA channel 1 transmit descriptor table address.	0x28
DMA channel 1 receive descriptor table address.	0x2C



0: Link Disable (LD) - Disable the SpaceWire codec. Reset value: ‘1’.
1: Link Start (LS) - Start the link, i.e. allow a transition from ready to started state. Not reset.
2: Autostart (AS) - Automatically start the link when a NULL has been received. Not reset.
3: Interrupt Enable (IE) - If set, an interrupt is generated when one of bit 8 to 10 is set and its corresponding event occurs. Reset value: ‘0’.
4: Tick In (TI) - The host can generate a tick by writing a one to this field. This will increment the timer counter and the new value is transmitted after the current character is transferred. A tick can also be generated by asserting the tick_in signal. Reset value : ‘0’.
5: Max Frequency Enable (ME) - Use the transmit clock without scaling, i.e. transmitter will be clocked directly with the txclk input. Reset value: ‘0’.
6: Reset (RS) - Make complete reset of the SpaceWire node. Self clearing. Reset value: ‘0’.
8: Tick-out IRQ (TQ) - Generate interrupt when a valid time-code is received. Not reset.
9: Link error IRQ (LI) - Generate interrupt when a link error occurs. Not reset.
16: RMAP Enable (RE) - Enable RMAP command handler. Only available if rmap generic is set to 1. Reset value: ‘1’.
17: RMAP buffer disable (RD) - If set only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively. Reset value: ‘0’.
29: RMAP CRC available - Set to one if RMAP CRC is enabled in the core. Only readable.
30: Rx Unaligned access - Set to one if unaligned writes are available for the receiver. Only readable.
31: RMAP available - Set to one if the RMAP command handler is available. Only readable.

Figure 12. GRSPW control register.



0: Tick Out (TO) - A new time count value was received and is stored in the time counter field. Cleared when written with a one. Reset value: ‘0’.

1: Credit Error (CE) - A credit has occurred. Cleared when written with a one. Reset value: ‘0’.

2: Escape Error (ER) - An escape error has occurred. Cleared when written with a one. Reset value: ‘0’.

3: Disconnect Error (DE) - A disconnection error has occurred. Cleared when written with a one. Reset value: ‘0’.

4: Parity Error (PE) - A parity error has occurred. Cleared when written with a one. Reset value: ‘0’.

5: Reserved.

6: Write synchronization Error (WE) - A synchronization problem has occurred when receiving N-Chars. Reset value: ‘0’.

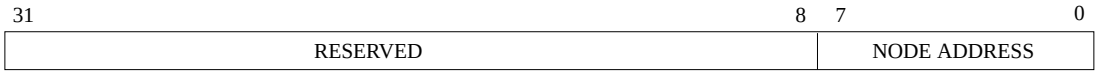
23-21: Link State (LS) - The current state of the start-up sequence. 0 = Error-reset, 1 = Error-wait,

2 = Ready, 3 = Started, 4 = Connecting, 5 = Run. Reset value: 0.

29-24: Time Counter - The current value of the system time counter. Reset value: 0.

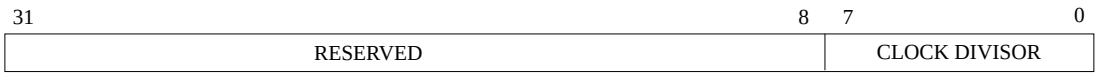
31-30: Time Control Flags - The current value of the time control flags. Currently always set to zero.

Figure 13. GRSPW status register



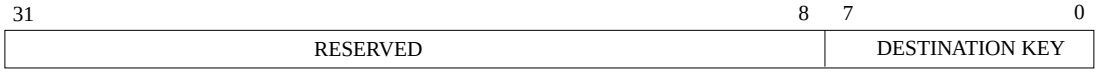
7 - 0: 8-bit node address used for node identification on the SpaceWire network. Reset value: 254.

Figure 14. GRSPW node address register.



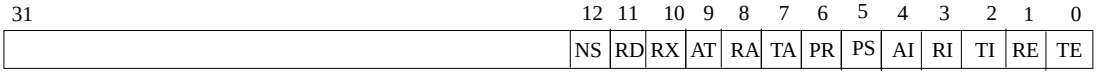
7 - 0: 8-bit Clock divisor value used for the clock-divider when the link-interface is in the run-state. Not reset.

Figure 15. GRSPW clock divisor register



7 - 0: RMAP destination key. Reset value: 0.

Figure 16. GRSPW destination key register.



0: Transmitter Enable (TE) - Write a one to this bit each time new descriptors are activated in the table. Writing a one will cause the SW-node to read a new descriptor and try to transmit the packet it points to. This bit is automatically cleared when the SW-node encounters a descriptor which is disabled. Reset value: ‘0’.

1: Receiver Enable (RE) - Set to one when packets are allowed to be received to this channel. Reset value: ‘0’.

2: Transmit Interrupt (TI) - If set, an interrupt will be generated each time a packet is transmitted. The interrupt is generated regardless of whether the transmission was successful or not. Not reset.

3: Receive Interrupt (RI) - If set, an interrupt will be generated each time a packet has been received. This happens both if the packet is terminated by an EEP or EOP. Not reset.

4: AHB Error Interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus. Not reset.

5: Packet Sent (PS) - This bit is set each time a packet has been sent. Never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.

6: Packet Received (PR) - This bit is set each time a packet has been received. never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.

7: TX AHB Error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.

8: RX AHB Error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.

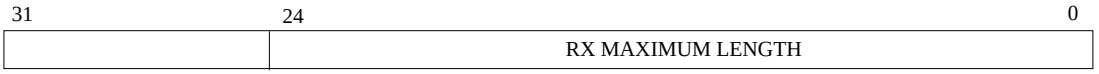
9: Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no transmission is active the only effect is to disable transmissions. Self clearing. Reset value: ‘0’.

10: RX Active (RX) - Is set to ‘1’ if a reception to the DMA channel is currently active otherwise it is ‘0’. Only readable.

11: Rx Descriptors Available (RD) - Set to one, to indicate to the GRSPW that there are enabled descriptors in the descriptor table. Cleared by the GRSPW when it encounters a disabled descriptor: Reset value: ‘0’.

12: No Spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the GRSPW will wait for a descriptor to be activated.

Figure 17. GRSPW DMA channel control/status register.



24 - 0: Receiver packet maximum length in bytes. Only bits 24 - 2 are writable. Bits 1 - 0 are always 0. Not reset.

Figure 18. GRSPW DMA channel receiver max length register.

3 - 0: Reserved.

9 - 4: Descriptor selector - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. Reset value: 0.

31 - 10: Descriptor table base address - Sets the base address of the descriptor table. Not reset.

Figure 19. GRSPW transmitter descriptor table address register

2 - 0: Reserved.

9 - 3: Descriptor Selector - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. Reset value: 0.

31 - 10: Descriptor table base address - Sets the base address of the descriptor table. Not reset.

Figure 20. GRSPW receiver descriptor table address register.

2.12 Signal description

GRSPW signals are described in table 6.

TABLE 6. GRSPW signals

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
TXCLK	N/A	Input	Transmitter default run-state clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SWNI	D	Input	Data input	-
	S	Input	Strobe input	-
	TICKIN	Input	Time counter tick input	High
SWNO	D	Output	Data output	-
	S	Output	Strobe output	-
	TICKOUT	Output	Time counter tick output	High
* see GRLIB IP Library User's Manual				

2.13 Library dependencies

Table 7 shows libraries that should be used when instantiating a GRSPW.

TABLE 7. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPACEWIRE	Signals, component	GRSPW component and record declarations.

2.14 GRSPW instantiation

This example shows how a GRSPW can be instantiated. Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The GRSPW in the example is configured with non-ft memories of size 4, 64 and 8 entries for AHB FIFOs, N-Char FIFO and RMAP buffers respectively. The system frequency (clk) is 40 MHz and the transmitter frequency (txclk) is 60 MHz.

The memory technology is inferred which means that the synthesis tool will select the appropriate components. The tx clk buffer uses a routed clock while the rx clk buffer uses a hardwired clock.

The hardware RMAP command handler is enabled which also automatically enables rxunaligned and rmapcrc. The Finally, the DMA channel interrupt line is 2 and the number of synchronization registers is 1.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- spacewire signals
    di :: in  std_ulogic;
    si :  in  std_ulogic;
    do :: out std_ulogic;
    so :  out std_ulogic
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni : grspw_in_type;
```

```
  signal swno : grspw_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRSPW
  sw0 : grspw
  generic map (tech => inferred, hindex => 5, pindex => 7, paddr => 7, nsync => 1,
  rmap => 1, rxunaligned => 0, rmapcrc => 0, txclkbuftype => 1, rxclkbuftype => 0,
  sysfreq => 40000, txfreq => 60000, pirq => 2, fifosize1 => 4, fifosize2 => 64,
  rmapbufs => 8, ft => 0)
  port map (rstn, clk, apbi, apbo(7), ahbmi, ahbmo(5), swni, swno);

swni.tickin <= '0';
swni.d      <= di;
swni.s      <= si;
do          <= swno.d;
so          <= swno.s;

end;
```

2.15 API

A simple Application Programming Interface (API) is provided together with the GRSPW. The API is located in $(GRLIB)/software/spw. The files are rmapapi.c, spwapi.c, rmapapi.h, spwapi.h. The spwapi.h file contains the declarations of the functions used for configuring the GRSPW and transferring data. The corresponding definitions are located in spwapi.c. The rmapapi is structured in the same manner and contains a function for building RMAP packets.

These functions could be used as a simple starting point for developing drivers for the GRSPW. The different functions are described in this section.

2.15.1 GRSPW basic API

The basic GRSPW API is based on a struct spwvars which stores all the information for a single GRSPW core. Then information includes its address on the AMBA bus as well as SpaceWire parameters such as node address, and clock divisor. This a pointer to this struct is used as a input parameter to all the functions. If several cores are used a separate struct for each core is created and used when the specific core is accessed.

TABLE 8. The spwvars struct

Field	Description	Allowed range
regs	Pointer to the GRSPW	-
rmap	Indicates whether the core is configured with RMAP. Set by spw_init.	0 - 1
rxunaligned	Indicates whether the core is configured with rxunaligned support. Set by spw_init.	0 - 1
rmapcrc	Indicates whether the core is configured with RMAPCRC support. Set by spw_init.	0 - 1
maxfreq	The maxfreq value set used for the core.	0 - 1
clkdiv	The clock divisor value used for the core.	0 - 255
nodeaddr	The node address value used for the core.	0 - 255
destkey	The destination key value used for the core.	0 - 255
rxmaxlen	The Receiver maximum length value used for the core.	0 - 33554431
rxpnt	Pointer to the next receiver descriptor.	0 - 127

TABLE 8. The spwvars struct

Field	Description	Allowed range
rxchkpnt	Pointer to the next receiver descriptor that will be polled.	0 - 127
txpnt	Pointer to the next transmitter descriptor.	0 - 63
txchkpnt	Pointer to the next transmitter descriptor that will be polled.	0 - 63
txd	Pointer to the transmitter descriptor table.	-
rxd	Pointer to the receiver descriptor table	-

The following functions are available in the basic API:

```
int spw_setparam(int nodeaddr, int clkdiv, int destkey, int rxmaxlen, int maxfreq, int
            spwadr, struct spwvars *spw);
```

Used for setting the different parameters in the spwvars struct. Should always be run first after creating a spwvars struct. This function only initializes the struct. Does not write anything to the SpaceWire core.

TABLE 9. Return values for spw_setparam

Value	Description
0	The function completed successfully
1	One or more of the parameters had an illegal value

TABLE 10. Parameters for spw_setparam

Parameter	Description	Allowed range
nodeaddr	Sets the node address value of the struct spw passed to the function.	0-255
clkdiv	Sets the clock divisor value of the struct spw passed to the function.	0-255
destkey	Sets the destination key of the struct spw passed to the function.	0-255
rxmaxlen	Sets the receiver maximum length field of the struct spw passed to the function.	$0 - 2^{25}-1$
maxfreq	Sets maxfreq field of the struct spw passed to the function.	0 - 1
spwadr	Sets the address to the GRSPW core which will be associated with the struct passed to the function.	$0 - 2^{32}-1$

```
int spw_init(struct spwvars *spw);
```

Initializes the GRSPW core located at the address set in the struct spw. Sets the following registers: node address, destination key, clock divisor, Receiver Maximum length, Transmitter descriptor table address, Receiver descriptor table address. The descriptor tables are allocated to an

aligned area using malloc. The status register is cleared and lastly the link interface is enabled. The run state frequency will be set according to the values stored in maxfreq and clkdiv,

TABLE 11. Return values for spw_init

Value	Description
0	The function completed successfully
1	One or more of the parameters could not be set correctly or the link failed to initialize.

TABLE 12. Parameters for spw_init

Parameter	Description	Allowed range
spw	The spwvars struct associated with the GRSPW core that should be initialized.	-

```
int set_txdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Transmitter descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_tx and spw_checktx (Explained in the section for those functions).

TABLE 13. Return values for spw_txdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

TABLE 14. Parameters for spw_txdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int set_rxdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Receiver descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_rx and spw_checkrx (Explained in the section for those functions).

TABLE 15. Return values for spw_rxdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

TABLE 16. Parameters for spw_rxdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_disable(struct spwvars *spw);
```

Disables the GRSPW core (the link disable bit is set to ‘1’).

TABLE 17. Parameters for spw_disable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_enable(struct spwvars *spw);
```

Enables the GRSPW core (the link disable bit is set to ‘0’).

TABLE 18. Parameters for spw_enable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_start(struct spwvars *spw);
```

Starts the GRSPW core (the link start bit is set to ‘1’).

TABLE 19. Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_stop(struct spwvars *spw);
```

Stops the GRSPW core (the link start bit is set to ‘0’).

TABLE 20. Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_setclockdiv(struct spwvars *spw);
```

Sets the clock divisor register with the clock divisor value stored in the spwvars struct.

TABLE 21. Return values for spw_setclockdiv

Value	Description
0	The function completed successfully
1	The new clock divisor value is illegal.

TABLE 22. Parameters for spw_setclockdiv

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_nodeadr(struct spwvars *spw);
```

Sets the node address register with the node address value stored in the spwvars struct.

TABLE 23. Return values for spw_set_nodeadr

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

TABLE 24. Parameters for spw_set_nodeadr

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_rxmaxlength(struct spwvars *spw);
```

Sets the Receiver maximum length register with the rxmaxlen value stored in the spwvars struct.

TABLE 25. Return values for spw_set_rxmaxlength

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

TABLE 26. Parameters for spw_set_rxmaxlength

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_set_maxfreq(struct spwvars *spw);
```

Sets the maxfreq bit in the spwvars struct and also sets the bit in the control register.

TABLE 27. Parameters for spw_set_maxfreq

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_disable_maxfreq(struct spwvars *spw);
```

Clears the maxfreq bit in the spwvars struct and also clears the bit in the control register.

TABLE 28. Parameters for spw_disable_maxfreq

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_tx(int crc, int skipcrcsize, int hsize, char *hbuf, int dsize, char *dbuf, struct
spwvars *spw);
```

Transmits a packet. Separate header and data buffers can be used. If CRC logic is available the GSPW inserts RMAP CRC values after the header and data fields if crc is set to one. This function only sets a descriptor and initiates the transmission. Spw_checktx must be used to check if the packet has been transmitted. A pointer into the descriptor table is stored in the spwvars struct to keep track of the next location to use. It is incremented each time the function returns 0.

TABLE 29. Return values for spw_tx

Value	Description
0	The function completed successfully
1	There are no free transmit descriptors currently available
2	There was illegal parameters passed to the function

TABLE 30. Parameters for spw_tx

Parameter	Description	Allowed range
crc	Set to one to append RMAP CRC after the header and data fields. Only available if hardware CRC is available in the core.	0 - 1
skipcrcsize	The number of bytes in the beginning of a packet that should not be included in the CRC calculation	0 - 15
hsize	The size of the header in bytes	0 - 255
hbuf	Pointer to the header data	-
dsize	The size of the data field in bytes	0 - 2^{24}-1
dbuf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should transmit the packet	-

```
int spw_rx(char *buf, struct spwvars *spw);
```

Enables a descriptor for reception. The packet will be stored to buf. Spw_checkrx must be used to check if a packet has been received. A pointer in the spwvars struct is used to keep track of the next location to use in the descriptor table. It is incremented each time the function returns 0.

TABLE 31. Return values for spw_rx

Value	Description
0	The function completed successfully
1	There are no free receive descriptors currently available

TABLE 32. Parameters for spw_rx

Parameter	Description	Allowed range
buf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should receive the packet	-

```
int spw_checkrx(int *size, struct rxstatus *rxs, struct spwvars *spw);
```

Checks if a packet has been received. When a packet has been received the size in bytes will be stored in the size parameter and status is found in the rxs struct. A pointer in the spwvars struct is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

TABLE 33. Return values for spw_checkrx

Value	Description
0	No packet has been received
1	A packet has been received

TABLE 34. Parameters for spw_checkrx

Parameter	Description	Allowed range
size	When the function returns 1 this variable holds the number of bytes received	-
rxs	When the function returns 1 this variable holds status information	-
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

TABLE 35. The rxstatus struct

Field	Description	Allowed range
truncated	Packet was truncated	0 - 1
dcrcerr	Data CRC error bit was set. Only indicates an error if the packet received was an RMAP packet.	0 - 1
hcrcerr	Header CRC error bit was se.t. Only indicates an error if the packet received was an RMAP packet.	0 - 1
eep	Packet was terminated with EEP	0 - 1

```
int spw_checktx(struct spwvars *spw);
```

Checks if a packet has been transmitted. A pointer is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

TABLE 36. Return values for spw_checktx

Value	Description
0	No packet has been transmitted
1	A packet has been correctly transmitted
2	A packet has been incorrectly transmitted

TABLE 37. Parameters for spw_checktx

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void send_time(struct spwvars *spw);
```

Sends a new time-code. Increments the time-counter in the GRSPW and transmits the value.

TABLE 38. Parameters for send time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int check_time(struct spwvars *spw);
```

Check if a new time-code has been received.

TABLE 39. Return values for check_time

Value	Description
0	No time-code has been received
1	A new time-code has been received

TABLE 40. Parameters for check_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int get_time(struct spwvars *spw);
```

Get the current time counter value.

TABLE 41. Return values for get_time

Value	Description
0 - 63	Returns the current time counter value

TABLE 42. Parameters for get_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void spw_reset(struct spwvars *spw);
```

Resets the GRSPW.

TABLE 43. Parameters for spw_reset

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be reset	-

```
void spw_rmapen(struct spwvars *spw);
```

Enables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW.

TABLE 44. Parameters for spw_rmapen

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
void spw_rmapdis(struct spwvars *spw);
```

Disables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW

TABLE 45. Parameters for spw_rmapdis

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
int spw_setdestkey(struct spwvars *spw);
```

Set the destination key of the GRSPW. Has no effect if the RMAP command handler is not available. The value from the spwvars struct is used.

TABLE 46. Return values for spw_setdestkey

Value	Description
0	The function completed successfully
1	The destination key parameter in the spwvars struct contains an illegal value

TABLE 47. Parameters for spw_setdestkey

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set.	-

2.15.2 GRSPW RMAP API

The RMAP API contains only one function which is used for building RMAP headers.

```
int build_rmap_hdr(struct rmap_pkt *pkt, char *hdr, int *size);
```

Builds a RMAP header to the buffer pointed to by hdr. The header data is taken from the rmap_pkt struct.

TABLE 48. Return values for build_rmap_hdr

Value	Description
0	The function completed successfully
1	One or more of the parameters contained illegal values

TABLE 49. Parameters for build_rmap_hdr

Parameter	Description	Allowed range
pkt	Pointer to a rmap_pkt struct which contains the data from which the header should be built	

TABLE 49. Parameters for build_rmap_hdr

Parameter	Description	Allowed range
hdr	Pointer to the buffer where the header will be built	
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

TABLE 50. rmap_pkt struct fields

Field	Description	Allowed Range
type	Selects the type of packet to build.	writecmd, readcmd, rmwcmd, writerep, readrep, rmwrep
verify	Selects whether the data should be verified before writing	yes, no
ack	Selects whether an acknowledge should be sent	yes, no
incr	Selects whether the address should be incremented or not	yes, no
destaddr	Sets the destination address	0 - 255
destkey	Sets the destination key	0 - 255
srcaddr	Sets the source address	0 - 255
tid	Sets the transaction identifier field	0 - 65535
addr	Sets the address of the operation to be performed. The extended address field is currently always set to 0.	0 - 2^{32}-1
len	The number of bytes to be writte, read or read-modify-written	0 - 2^{24}-1
status	Sets the status field	0 - 11
dstspalen	Number of source path address bytes to insert before the destination address	0 - 228
dstspa	Pointer to memory holding the destination path address bytes	-
srcspalen	Number of source path address bytes to insert in a command. For a reply these bytes are placed before the return address	0 - 12
srcspa	Pointer to memory holding the source path address bytes	-