System-on-Chip Wire (SoCWire) User Manual

Issue Draft, Revision 1

SoCWire V1.1

02.02.2012

Prepared by
Björn Osterloh

Page intentionally left blank

DOCUMENTATION CHANGE RECORD

Issue	Rev.	Pages	Date	Changes	Author
Draft	1	All	22.04.2009	Draft Issue	BO
	2	8	02.02.2012	Inverted Endianness removed	HM

TABLE OF CONTENTS

1 General Overview

SoCWire is a Network-on-Chip (NoC) approach based on the ESA SpaceWire interface standard [1] to support dynamic reconfigurable System-on-Chip (SoC). SoCWire has been developed to provide a robust communication architecture for the harsh space environment and to support dynamic partial reconfiguration in future space applications.

SoCWire provides:

- Reconfigurable point-to-point communication
- High speed data rate
- Hot-plug ability to support dynamic reconfigurable modules
- Link error detection and recovery in hardware
- Easy implementation in dynamic partial reconfigurable systems.
- Scalable data word width (8-8192)
- Configurable Switch with 2 to 32 ports

For more background information about the SoCWire motivation see [2].

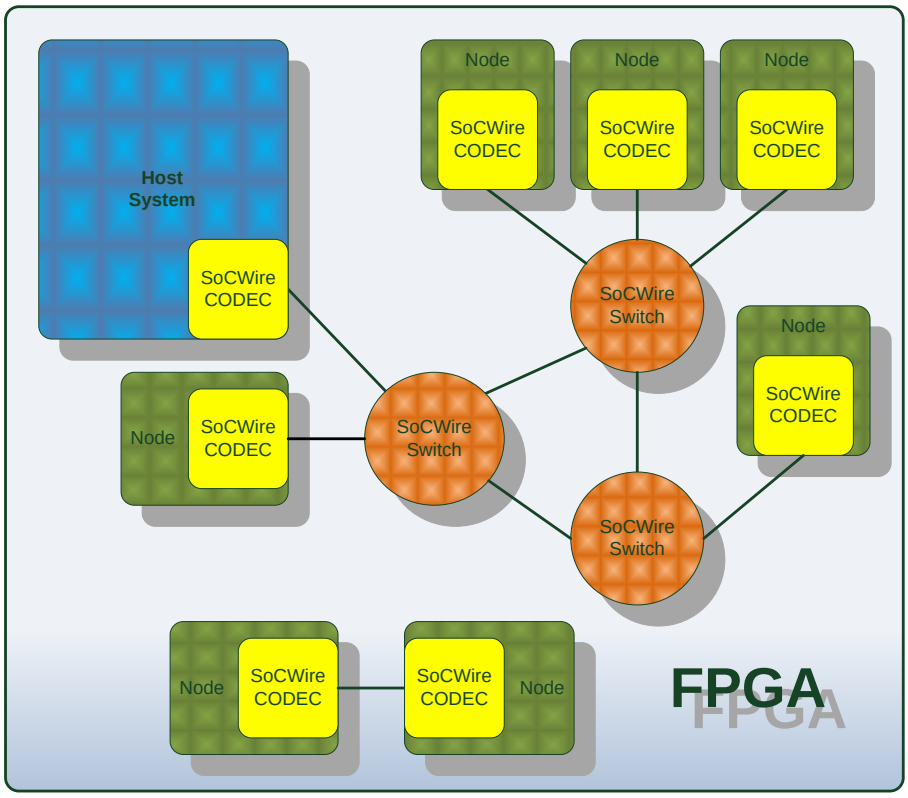


Figure 1: SoCWire architecture network example

1.1 Software License

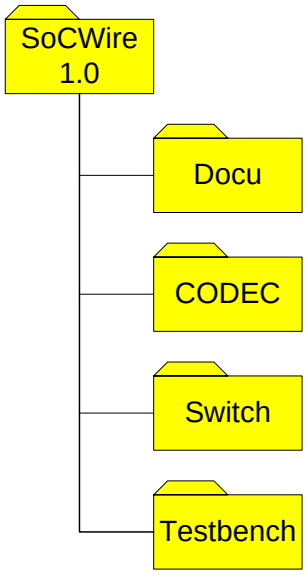
This license governs the use of this software, and your use of this software constitutes acceptance of this license. Agreement with all points is required to use this software.

1. These source files may be used and distributed without restriction provided that the software license statement is not removed from the file and that any derivative work contains the original software license notice and the associated disclaimer.
2. The source files are free software; you can redistribute it and/or modify it under the restriction that **UNDER NO CIRCUMTANCES this Software is to be used to CONSTRUCT a SPACEWIRE INTERFACE**. This implies modification and/or derivative work of this Software.
3. This source is distributed in the hope that it will be useful, but **WITHOUT ANY WARRANTY**; without even the implied warranty of **MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.**

Your rights under this license are terminated immediately if you breach it in any way.

1.2 Software content

This software package comprises:



Docu	This document
CODEC	SoCWire CODEC source files
Switch	SoCWire Switch source files
Testbench	SoCWire CODEC Testbench. 1 SoCWire CODEC configured in loopback mode, packets with different length are send over the link
	SoCWire network testbench. 1 Switch and 4 nodes communication example

2 SoCWire

SoCWire is based on the SpaceWire standard [1]. The SpaceWire interface is a well established standard, providing a layered protocol (physical, signal, character, exchange packet, network) and proven interface for space applications. It is an asynchronous communication, serial link, bi-directional (full-duplex) interface including flow control, error detection and recovery in hardware, hot-plug ability and automatic reconnection after a link disconnection. SpaceWire is a serial link interface and performance of the interface depends on skew, jitter and the implemented technology. SoCWire is a NoC approach in a complete on-chip environment. Therefore SpaceWire interface has been modified to a parallel data interface. The advantage of this approach is that significantly higher data rates can be achieved as compared to the SpaceWire standard. Additionally, a scalable data word width to support medium to very high data rates has been implemented. On the other hand the advantageous features of the SpaceWire standard including flow control, hot-plug ability, error detection and link re-initialization are still fully support.

3 Background

The following sections describe the SoCWire interface. The chapter briefly describes the SoCWire relevant sections from the SpaceWire standard and the differences. For more detailed information please refer to [1]. The SoCWire character level, exchange level, packet level and network level is derived from the SpaceWire standard and refers to it.
In Contrast to SpaceWire the SoCWire architecture does not require the physical layer. Furthermore Time-Code characters and logical addressing has not been implemented to safe resources.

3.1 Character Level

The character level describes data and control characters used to manage the flow of data across the link. It follows the SpaceWire standard without Time-Code distribution. SpaceWire does not provide a global time base; nodes are synchronized through a system time distribution with Time-Codes, which have a high priority in the network Time-Code characters are not necessarily required because a global time base distribution can be easily implemented in a complete on-chip environment through dedicated signals, which saves resources.

3.1.1 Data Character

A data character is formed by 1 parity bit, 1 data-control flag and 8 data bits to be transmitted.

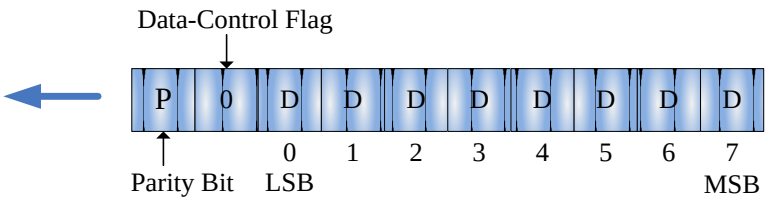


Figure 2: Data Character

The data-control flag indicates if the current character is a data (0) or control character (1).

3.1.2 Control Character

Control characters (4 bit length) are used for flow control: Flow Control Token (FCT), End of Packet (EOP), Error End of Packet (EEP) and an Escape Character (ESC) which is used to form the higher level control code (8 bit length) NULL (ESC+FCT). Odd parity is assigned to data and control characters to support the detection of transmission errors. The parity bit covers the previous eight bits of a data character or two bits of a control character. *Table 3-1* shows the different characters. If the data width is exceeds 8 bits the highest significant bits are appended with zeros. Parity depends on the data or control character send in the previous clock cycle.

Parity	Control	Data	Type
P	'1'	X"00"	FCT Flow Control Token
P	'1'	X"02"	EOP End of Packet
P	'1'	X"01"	EEP Exceptional End of Packet
P	'1'	X"03"	ESC Escape
P	'1'	X"0B"	Null

Table 3-1 : Control Characters & Control Code

3.1.3 Parity

Odd parity is assigned to data and control characters to support the detection of transmission errors. The parity bit covers the previous bits of a data or control character, the parity bit, and the data-control flag. This parity algorithm is a SpaceWire legacy. SpaceWire employs this parity algorithm, because of the serial nature of the transmission.

3.1.4 EOP and EEP

The host data interface can be source or destination for the transferred data. To distinguish between user data and packet marker (EOP and EEP) the following coding is used.

Data Control Flag	Data bits (MSB..LSB)	
0	XXXXXXXX	user data
1	00000000	coded EOP
1	00000001	coded EEP

Table 3-2: Coding of EOP and EEP for 8 bit data word width

The data control flag is the MSB of the dat_din(datawidth) or dat_dout(datawidth) vector. All lower bits shall be set to '0's (EOP) or '0's + '1' for the LSB (EEP).

3.2 Exchange Level

The exchange level manages the connection and flow across the link. The exchange level is separated into two types: Link-Characters (L-Char) and Normal-Characters (N-Char). N-Char comprises data character, EOP and EEP and are passed to the network level. L-Char are used in the exchange level and are not passed to the network level. They comprise FCT and ESC characters and are responsible for link connection and flow control.

3.2.1 Flow Control

To avoid buffer overflows and therefore data loss, a credit-based flow control is implemented. After link connection is established, FCTs are transmitted over the link. Each FCT signify that one end of the link is ready to receive 8 N-Chars.

3.2.2 State Machine

The SoCWire CODEC is based on a finite state machine derived from the SpaceWire standard as depicted in *Figure 3*.

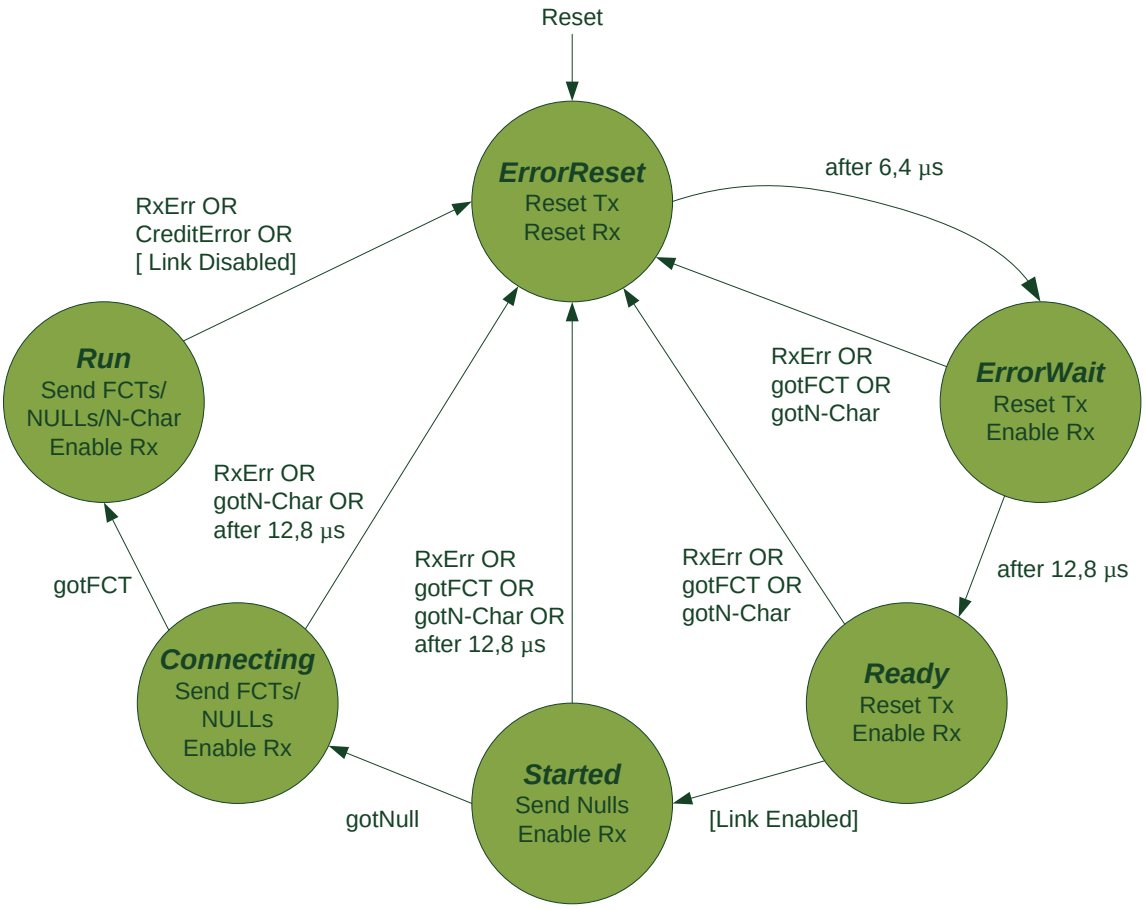


Figure 3: SoCWire CODEC state machine (adapted from [1])

The "ErrorReset" is the initial state after an external reset, a link operation is terminated or if an error occurs during link initialization. In "ErrorReset" state both transmitter and receiver are in reset. If reset is de-asserted this state will be move after 6,4 µs into "ErrorWait", In "ErrorWait" the receiver is enabled and wait for 12,8 µs. This time period make sure that both ends of the link are ready to receive data before either ends begins transmission. The "Ready" state checks if the interface has permission (Link Enable) to buildup a link. If Link Enable is true the state machine moves on into "Started" and waits 12,8 µs for NULL-characters. If during this time period NULL-characters are received the state machine moves into "Connecting" and waits 12,8 for µs FCTs. If an FCT is received the state machine moves into "Run". The "Run" state is the state of normal operation.

3.2.3 Link Connection

Figure 4 shows the SoCWire link connection flow.

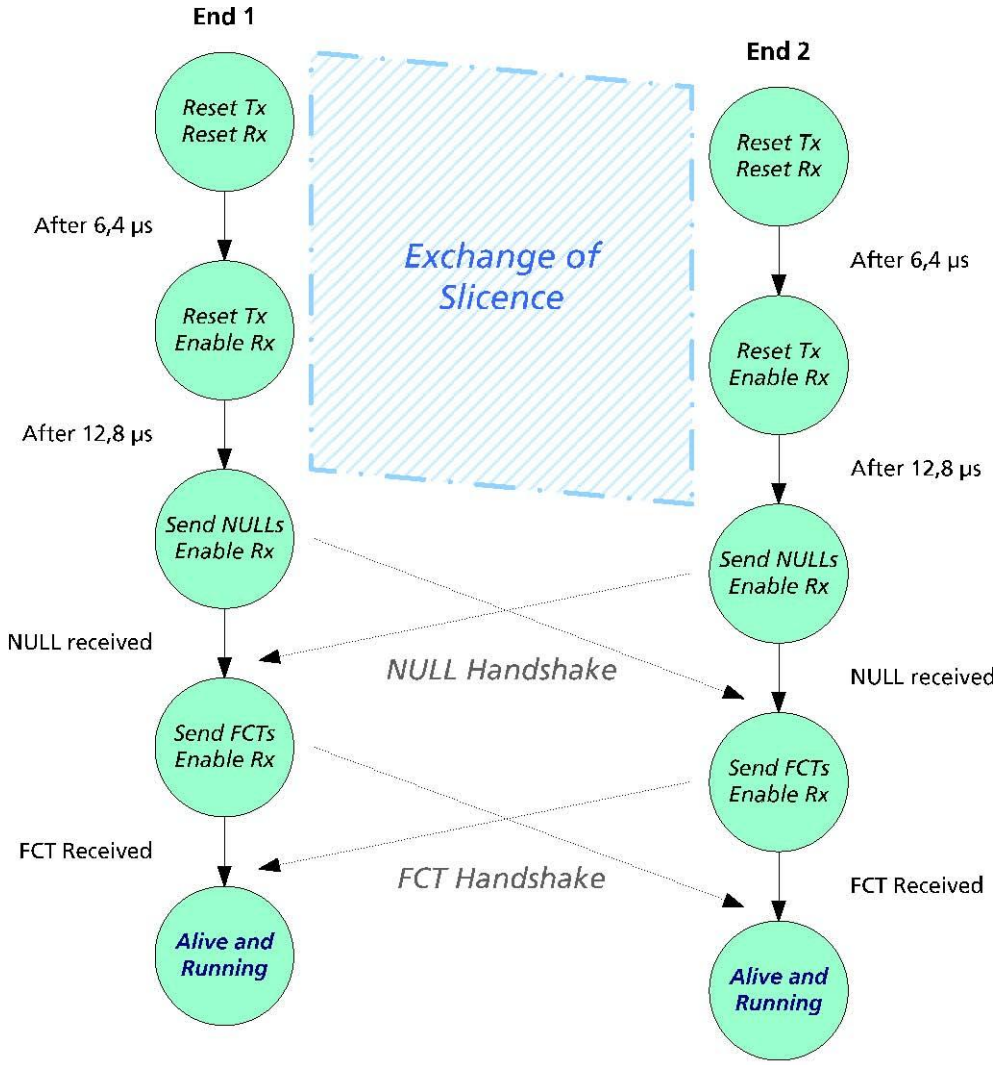


Figure 4: SoCWire link connection (adapted from [1])

The timeouts After 6,4 µs and After 12,8 µs follow the SpaceWire standard. Since SoCWire is in a complete synchronized on-chip environment these timeouts can be decreased. Additional detection of disconnection timeout has a window of 850 ns in the SpaceWire standard and can also be decreased.

Generic	Range (1 = 1 ns)
after64	1 to 6400
after128	1 to 12800
disconnect_detection	1 to 850

Table 3-3: SoCWire timeout generics

In simulation after64=64, after128=128 and disconnect_detection = 85 has been successfully tested. This has to be verified in hardware!

3.3 Packet Level

The packet level describes the format to support routing of packets over a SoCWire network. A SoCWire packet comprises Destination Address plus Cargo plus EOP/EEP as depicted in *Figure 5*

Figure 5: SoCWire packet

The Destination Address is required to send packets over a SoCWire network to a certain target. Depended on the network topology multiple destination addresses can follow, before the Cargo begins. For point-to-point communication the destination address is not required. The Cargo contains the user data. Regular packets are completed with an EOP marker. EEP marker is exclusively send by the user to indicate an erroneous packet. To this the target can react accordingly and reject the packet. In summary, the SpaceWire packet level is highly flexible and permits to implement a wide range of protocols.

3.4 Network

SoCWire network nodes can be either connected by links or connected by routing switches. The network operates exclusively with the objects of the packet level. All lower levels are completely masked from the network. The network topology can be configured as e.g. tree, cloud or cube. The SoCWire Switch supports wormhole routing; packets arriving at one port are routed immediately to the output port, if the port is free, which reduces buffer space and latency. The SoCWire network level supports the simple and effective header deletion technique to transfer packets across an arbitrary sized network. When a packet is received at a routing switch the destination port is determined from the header. The destination header is then deleted and the remaining packet content is transferred through the output port. If a second identifier exists it can be used for any subsequent routing. Therefore at each stage of the network a packet can be regarded as a packet comprising a single destination identifier header, cargo and end of packet. The SoCWire network level comprises patch dressing. With path addressing a sequence of destination identifier within a packet is used to guide the packet across the network.

The SoCWire error recovery scheme covers the exchange and network level. In the exchange level the following errors can be detected:

- Disconnect error
- Parity error
- Escape error
- Character sequence error (invalid token at invalid time)
- Credit error

The response to any of these errors is:

1. Detect error
2. Disconnect link
3. Report error to network level
4. Attempt to reconnect the link if the link is still enabled

In the network level the following errors can be detected:

- Link error (exchange level error)
- EEP received
- Invalid destination address

If a link error is detected the network level response as follows:

1. Error is received by the network level
2. Current received packet is terminated with EEP
3. If the error occurred in destination or source node, the error shall reported to the host system

4 SoCWire CODEC

The SoCWire CODEC connects a node or host system to a SoCWire network. SoCWire CODECs are the atomic components of the network and are source and destination of a SoCWire link.

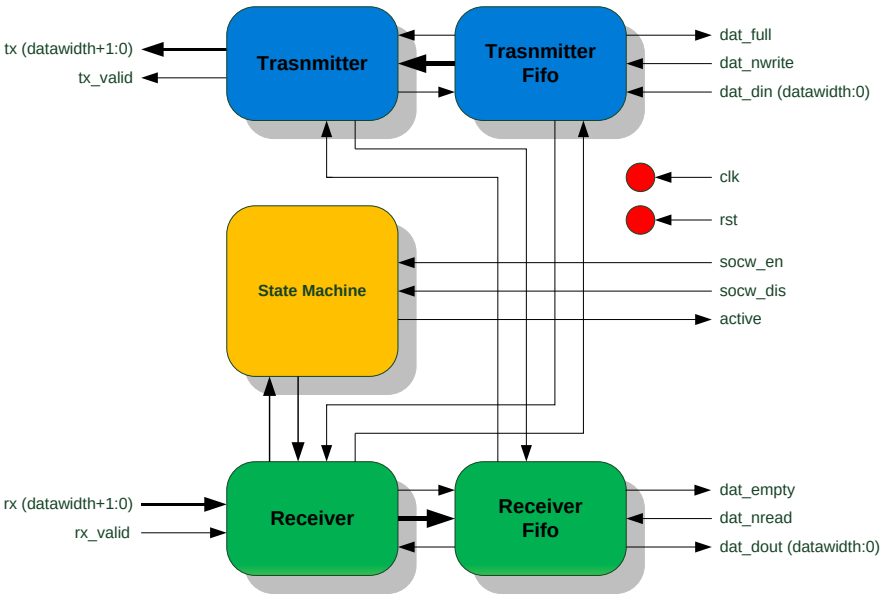
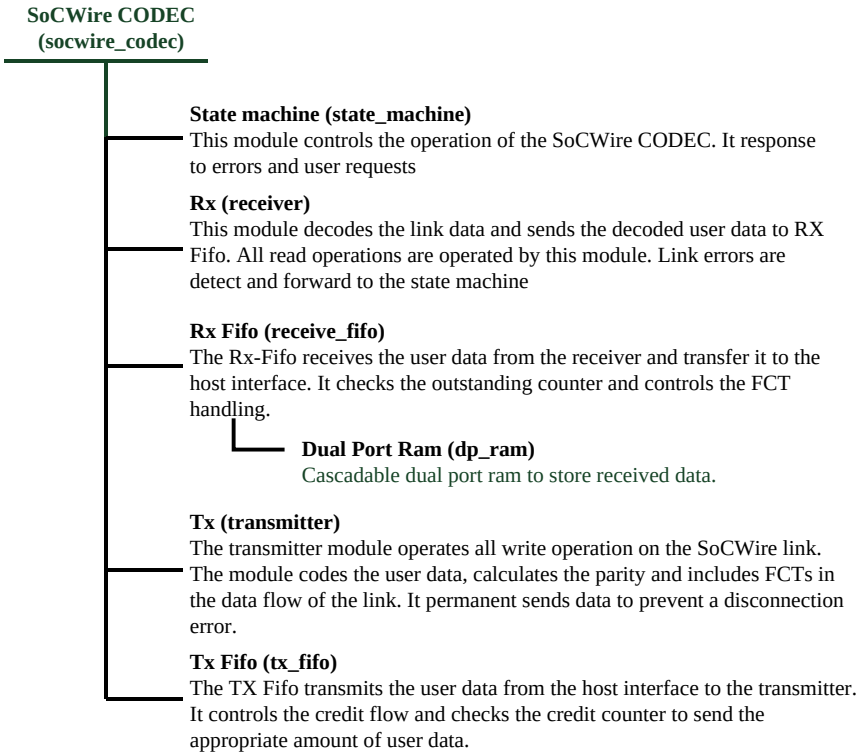


Figure 6: SoCWire CODEC

4.1 Structure

The SoCWire CODEC consists of 6 modules:



4.1.1 State Machine

The state machine describes the SoCWire CODEC finite state machine. Its states are analog to *Figure 3* and response to the external signals “socw_en” and “socw_dis”. “socw_en” is necessary to move from the state “ready” into “started”. The “socw_dis” signal forces the state machine to move from “run” into “ErrorReset”. Additional the signal “active” indicates the “run” state, where link connection is established. Time conditions for time dependent state transitions are triggered by watchdog timer, which are adapted to the system clock period. The watchdog clock cycles are calculated with:

$$watchdog = \frac{t_w}{t_{clk}} - 1$$

t_w is the number of watchdog clock cycles and t_{clk} is the clock period in ns. The current state of the state machine is distributed to all other modules.

4.1.2 Rx Fifo

The rx_fifo can be access from the host interface with the low active “dat_nread” signal. The fill state of the fifo is indicated with the high active “dat_empty” signal. If “dat_empty” is ‘0’, data can be read from the fifo. The “dat_dout” signal vector provides the user data. It is 1+n bit width, n is the data word width and the MSB is the Data Control Flag. The fifo is currently realized with 16 Kbyte BlockRAM (dp_ram). The fifo depth is 1024, which is the default depth of the Virtex-4 architecture. The fifo is cascadable to adapt it to data word width from 8 to 8192 bit. This is automatically generated and the resource utilization of BlockRam increase with the data word width.

The fifo is currently implemented for the Xilinx Virtex-4 architecture, but it can be simply implemented in any architecture with the replacement of the BlockRam primitive in the dp_ram module. Additonal the rx_fifo controls the FCTs transfer. This is done by the “fct_empty” signal. With “fct_nread” the transmitter can indicate that its credit counter is full and no more FCTs have to be sent.

4.1.3 Rx

The receiver module receives the parallel data of the link. Since SoCWire is a synchronous implementation it requires a additional signal to indicate the validation of the data. This is implemented with the high active “rx_valid” signal. The receiver is responsible to detect link errors. The errors are pass to the state machine with the signals “err_par” (parity error),”err_esc” (escape error),”err_dsc”(disconnection error),”err_nchar”(character error) and “err_fct”(fct error). A disconnection error is triggered if the link is inactive for $disconnect\_detection_{ns}$ (generic in the vhdl model) after the first link connection was established. Inactive is related to the rx_valid low period. This state is implemented with a counter (dsc_count).
The clock cycles can be calculated with:

$$watchdog = \frac{disconnect\_detection_{ns}}{t_{clk}} - 1$$

The receiver communicates with the rx_fifo, to transfer data with the “dat_dout” signal. A valid received N-Char is displayed with the “dat_empty” signal. Additional valid received FCTs are displayed with the “fct_empty” signal. The rx_fifo signalize its internal fifo fill state to the receiver. If the fifo is full and still a N-Char is received a character error is triggered. The receiver is fully pipelined to provide at every clock cycle a full SoCWire data word.

4.1.4 Tx Fifo

The transmit fifo can be access with the low active “dat_write” signal”. As soon as the fifo is full the “dat_full” signal is ‘1’. FCTs received by the receiver are forwarded with the “fct_nwrite” signal to the transmitter. The “fct_nwrite” increments the credit counter. If the credit counter receives 7 FCTs the “fct_full” signal is set. For a bi-directional full-duplex transfer the FCTs have to be included in the data transfer. The tx fifo does not require a dedicated ram, because it just has to store 1 data word.

4.1.5 Tx

The tx module codes and transfers the user data and operates the handshake mechanisms. Additional it calculates the parity bit in a 2 stage pipeline. The “tx_valid” signal indicates the validity of the data and is connect with “rx_valid” of the receiver. *Figure 7* shows the state diagram.

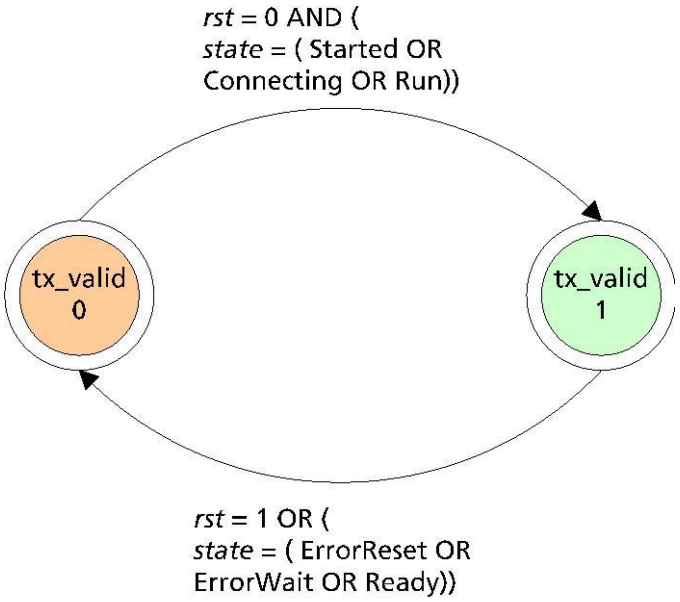


Figure 7: Control of tx_valid

In idle mode the transmitter sets all bits, except the parity bit, to '0'. The parity bit is set to '1' to cover the odd parity. In "run" state the transmitter sends NULL character to sustain the link connection.

4.2 Configuration options

Generic	Function	Allowed range	Default
datawidth	data word width	8-8192	8
speed	CODEC speed set to system clock in ns	1-100	10
after64	6,4 us timeout, unit ns	1-6400	6400
after128	12,8 us timeout, unit ns	1-12800	12800
disconnect_detection	disconnect detetction timeout	1-850	850

4.3 Signal description

Signal name	Range	Type	Function	Active
rst		Input	Reset	High
clk		Input	Clock	
socw_en		Input	Link Enable - When assert CODEC can move to Ready state and	High
socw_dis		Input	Link Disable - When assert and in Run state CODEC moves immediately to Error Reset state	High
rx	(datawidth+1:0)	Input	SoCWire CODEC receive link	
rx_valid		Input	When assert current rx link data is valid	High
tx		Output	SoCWire CODEC transmit link	
tx_valid		Output	When assert current tx link data is valid	High
dat_full		Output	Input Data Interface - Indicates the input Fifo is full, write is rejected	High
dat_nwrite		Input	Input Data Interface - Write data to CODEC	Low
dat_din	(datawidth:0)	Input	Input Data Interface - User data (datawidth-1:0), Data control flag (datawidth)	
dat_nread		Input	Read data from CODEC	Low
dat_empty		Output	Indicates fifo empty, no data available	High
dat_dout	(datawidth:0)	Output	User data (datawidth-1:0), Data control Flag (datawidth)	
active		Output	When active , link is connected and running, transmission can start	High

4.4 Write access

Figure 8 shows a SoCWire CODEC write transfer completed with an EOP.

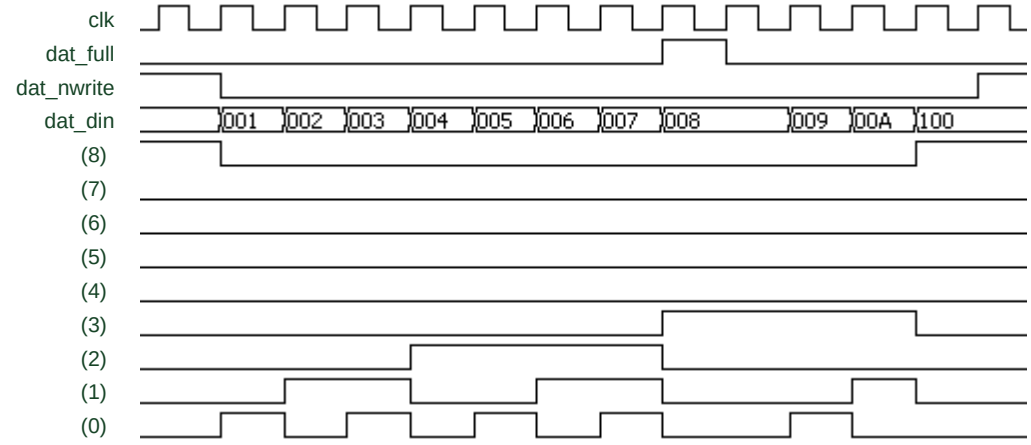


Figure 8: SoCWire CODEC write access for 8 bit data word width

4.5 Read access

Figure 9 shows a SoCWire CODEC read transfer completed with an EOP.

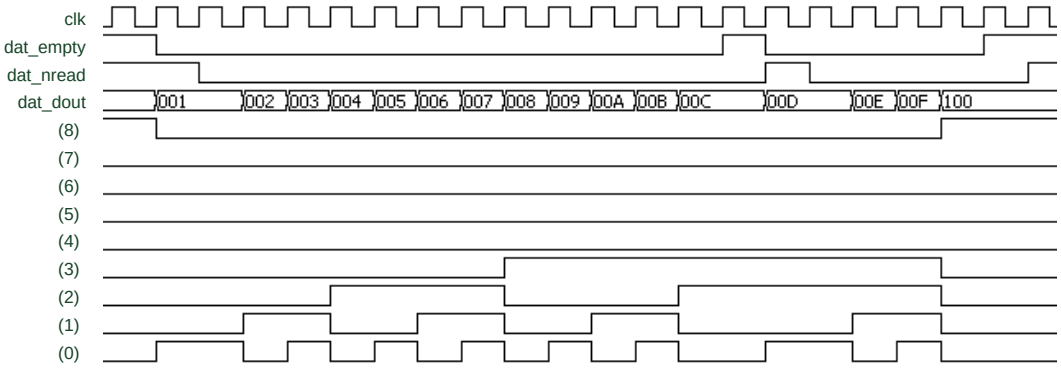


Figure 9: SoCWire CODEC read access 8 bit data word width

4.6 Data Rates

For bi-directional (full-duplex) data transfer the FCTs need to be included in the transfer. After initialization phase, every eight data characters are followed by one FCT. The maximum data rate for a bi-directional (full-duplex) transfer can therefore be calculated to:

$$DRate_{Bi}\left[\frac{\text{Mb}}{\text{s}}\right] = f_{Core(MHz)} \times DWord\ Width \times \frac{7}{8}$$

For a unidirectional data transfer the FCTs are processed in parallel and the maximum data rate can be calculated to:

$$DRate_{Uni}\left[\frac{\text{Mb}}{\text{s}}\right] = f_{Core(MHz)} \times DWord\ Width$$

Figure 10 shows data rates for different data word width, unidirectional and bi-directional (full-duplex) data transfer at a core clock frequency of 200 MHz

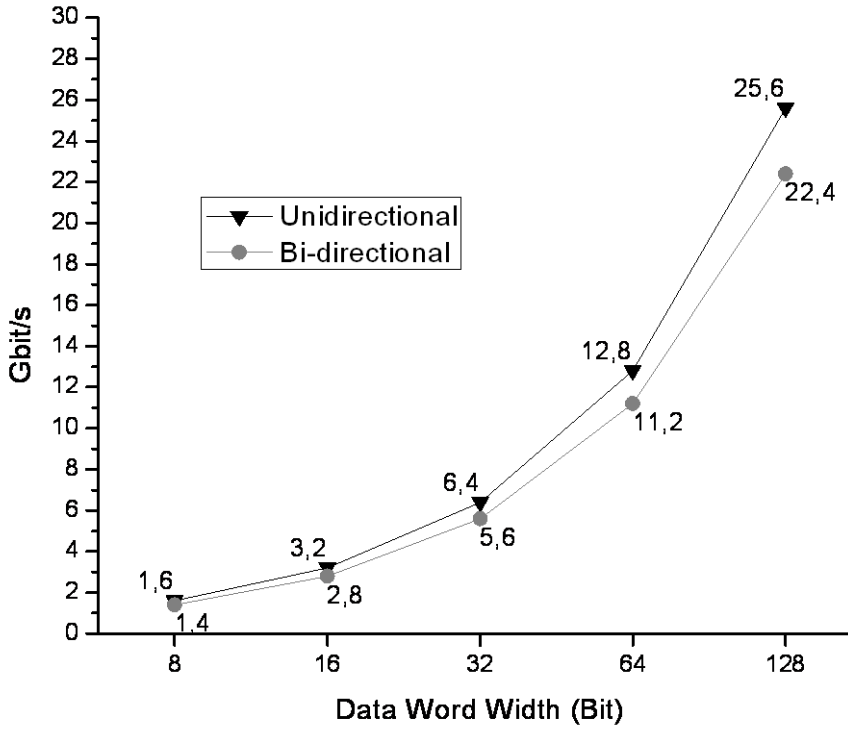


Figure 10: SoCWire CODEC data rates at core clock frequency 200 MHz

4.7 Resource utilization and timing

The SoCWire CODEC has been implemented and tested in Xilinx Virtex-4 LX60-10. *Figure 11* shows the occupied area, absolute values and maximum clock frequency.
Appendix A shows detailed information of the SoCWire CODEC implementation.

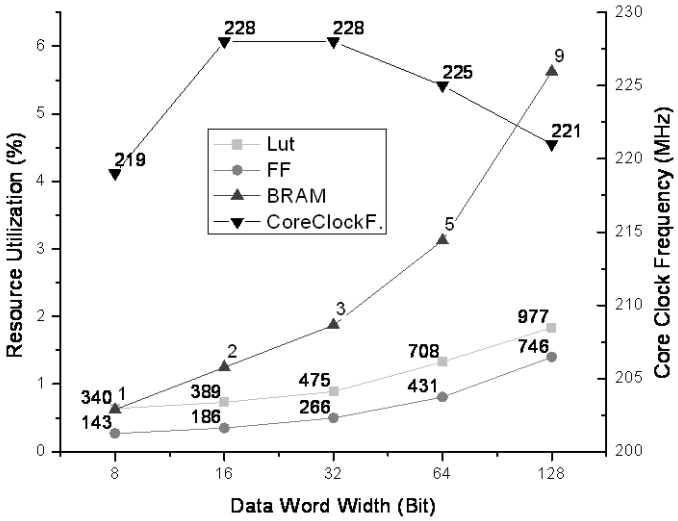


Figure 11: SoCWire CODEC synthesis report Xilinx Virtex 4 LX60-10

5 SoCWire Switch

The SoCWire Switch enables the transfer of packets arriving at one link interface to another link interface on the switch. The SoCWire Switch provides a configurable number of ports, realized by internal SoCWire CODECs.

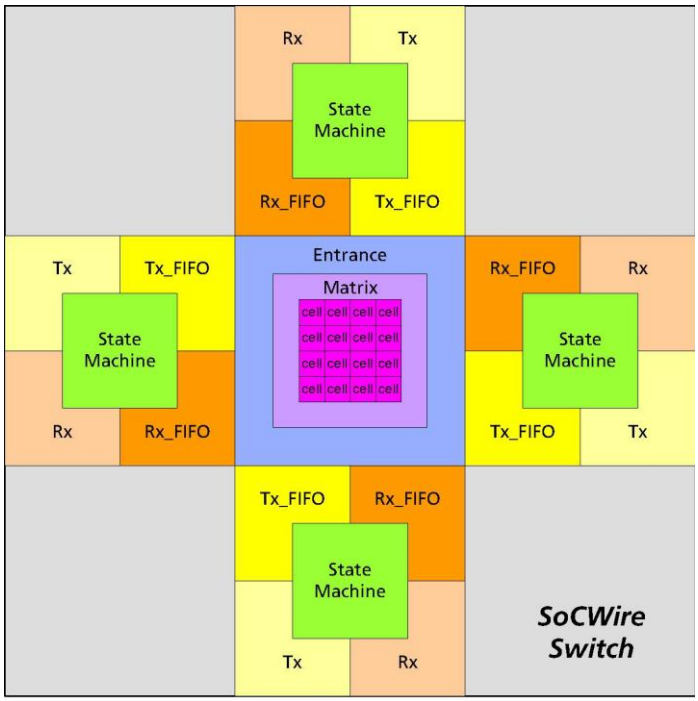
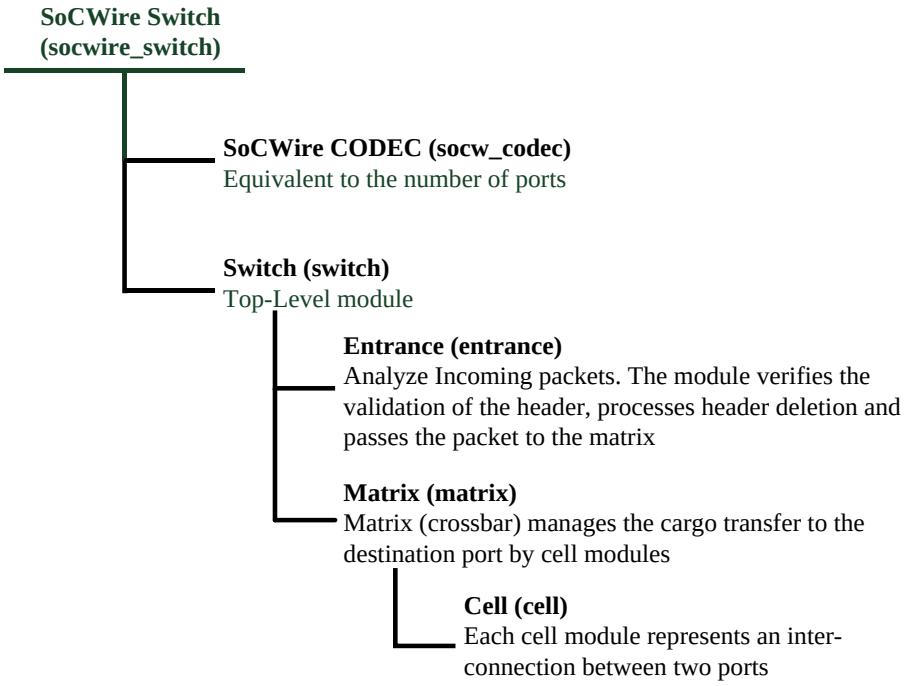


Figure 12: SoCWire Switch

5.1 Structure

The SoCWire Switch comprises at least 5 modules. The number of SoCWire CODECs are equivalent to the number of ports. Each cell module represents an inter-connection between two ports and therefore a switch with 4 ports comprises 16 cells or with 32 ports 1024 cells



5.1.1 Entrance

The entrance module analyzes the header of incoming packets. If the packet destination port is valid the module deletes the header and forwards the cargo to the matrix with the destination port information. For each port a entrance module is instantiated, therefore a 4 port switch has 4 entrance modules. This is necessary to provide an independent communication of each port. Additional the entrance module markes a port as “full” if a cargo is transferred. The entrance module itself does not receive information if the port is busy. The requested port of the current packet is therefore transferred to the matrix with the “wanted” signal. This signal tests autonomously if the connection to the destination port can be established. This information is provided to the entrance module with the “nwrite” signal. If it is ‘0’ the transfer can be started. If the entrance receives an EOP or EPP, the header deletion is enabled and destination address is determined for the incoming next packet.

5.1.2 Matrix

The matrix (crossbar) manages the cargo transfer to the destination port by cell modules. Each cell module represents an inter-connection between two ports and therefore a switch with 4 ports comprises 16 cells or with 32 ports 1024 cells. To provide parallel data transfer between different ports, the matrix conceives data transfers of all ports as bit vector. Therefore the bit width of the matrix is datawordwidth+1(data control flag) * number of ports. As depicted in *Figure 13*.

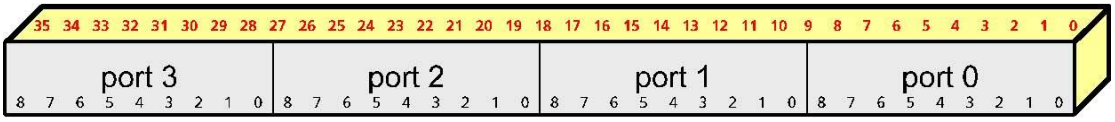


Figure 13: Matrix bit vector

Data word width is in this case 8 bit and the switch comprises 4 ports. The matrix can manipulate bits in this vector within 1 clock cycle to provide write access to ports. To prevent the concurrent access from 2 ports writing to 1 port the round robin scheduling mechanism is implemented. The access to ports is divided in time slots. Each time slot represents write access of 1 port. Therefore the maximum latency of the Switch is number of ports -1; for a 4 port Switch 3 clock cycles. If 2 ports access 1 destination port, one data transfer will be blocked until the first received packet is fully transmitted.

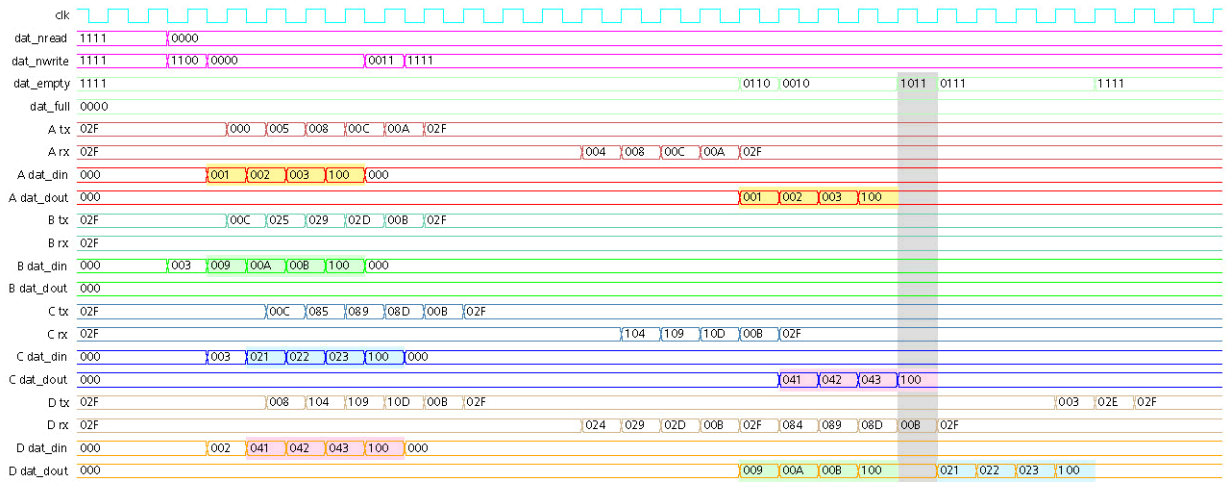
5.2 Configuration options

Generic	Function	Allowed range	Default
datawidth	data word width	8-8192	8
nports	Number of ports	2-32	3
speed	Switch speed set to system clock in ns	1-100	10
after64	6,4 us timeout, unit ns	1-6400	6400
after128	12,8 us timeout, unit ns	1-12800	12800
disconnect_detection	disconnect detetction timeout	1-850	850

5.3 Signal description

Signal name	Range	Type	Function	Active
rst		Input	Reset	High
clk		Input	Clock	
rx	((datawidth+2)*nports-1:0)	Input	SoCWire Switch receive link	
rx_valid	(nports-1 DOWNTO 0)	Input	When assert current rx link data is valid	High
tx	((datawidth+2)*nports-1:0)	Output	SoCWire Switch transmit link	
tx_valid	(nports-1 DOWNTO 0)	Output	When assert current tx link data is valid	High
active		Output	When assert, link is connection and running, deassert link inactive	High

5.4 Switch example



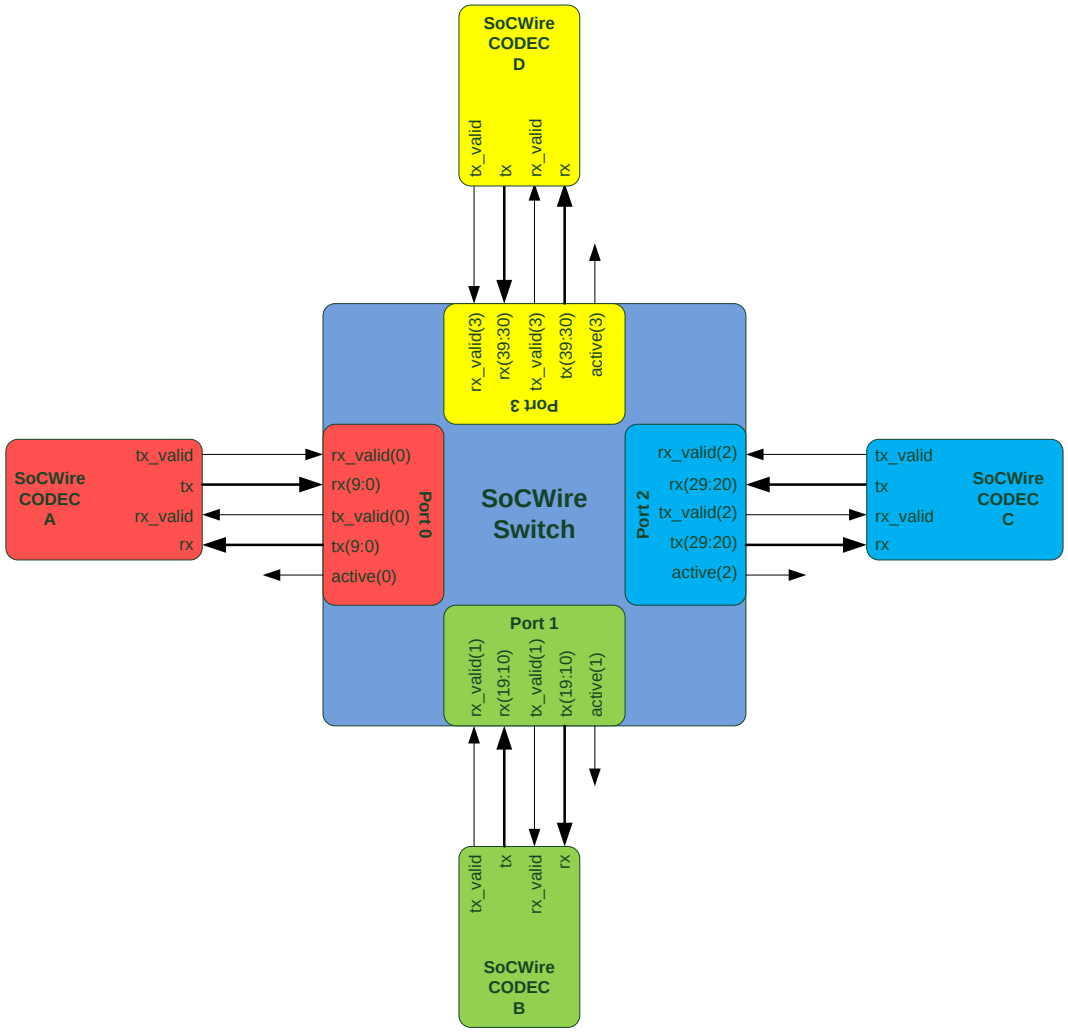


Figure 14: SoCWire Switch example with 4 nodes

Figure 14 shows a SoCWire network with 4 nodes in a star topology. The simulation shows the nodes A,B,C and D from top to bottom. For each node the transmit (tx, dat_in) and receive (rx, dat_dout) signals are shown. The signals dat_nread, dat_nwrite, dat_empty and dat_full are represented by bit vectors. The MSB (left) is therefore node D and LSB (right) node A. In the first operation all nodes are set in read mode with dat_read = '0'. Concurrently node A and B start a write cycle. Node A sends data to its own port (loopback) and sets the header to "000" (orange marked). Node Bs destination is node D and therefore sets the header to"003" (green marked). One clock cycle later node C sends data to node D (purple marked) and node D sends data to C (blue marked). The following transmissions make clear that the packets are received in the order they have sent. The packet from C to D (blue) has to wait until the packet from B (green) is completely received by D. Between the 2 packets the receiver of node D receives a NULL character (grey), this is caused by the round robin scheduling of the switch.

5.5 Data Rates

The SoCWire Switch basically consists of a number of SoCWire CODECs according to the number of ports and additional fully pipelined control machines. The maximum data rate is therefore equivalent to the SoCWire CODEC.

5.6 Resource utilization and timing

The SoCWire Switch is a fully scalable design supporting many data word widths (8-128bit) and 2 to 32 ports. It is a totally symmetrical input and output interface with direct port addressing including header deletion. The SoCWire Switch has been implemented and tested in a Xilinx Virtex-4 LX60-10. *Figure 15* shows the occupied area and maximum clock frequency for an 8 bit data word width switch. For detailed resource utilization and timing see Appendix A

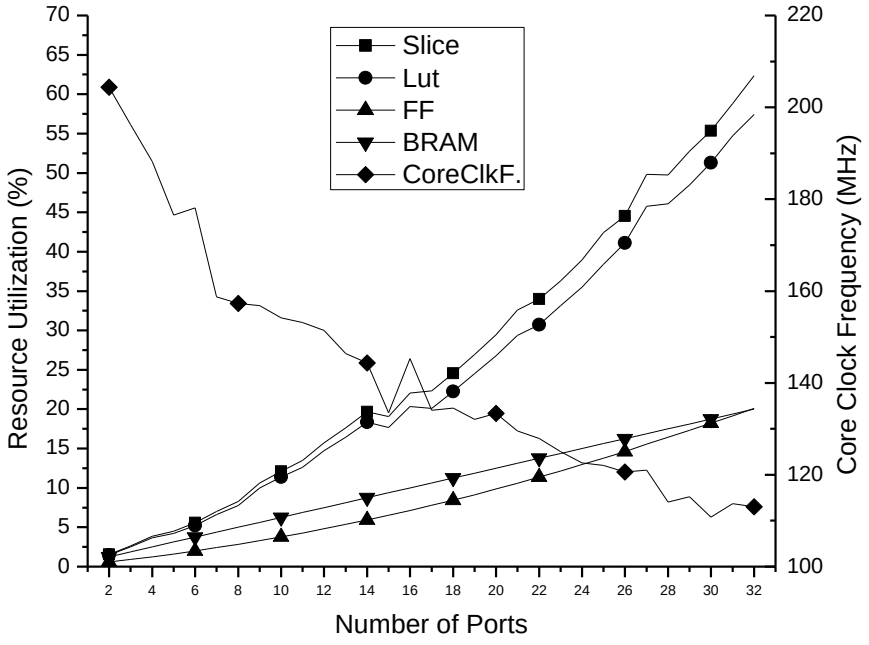


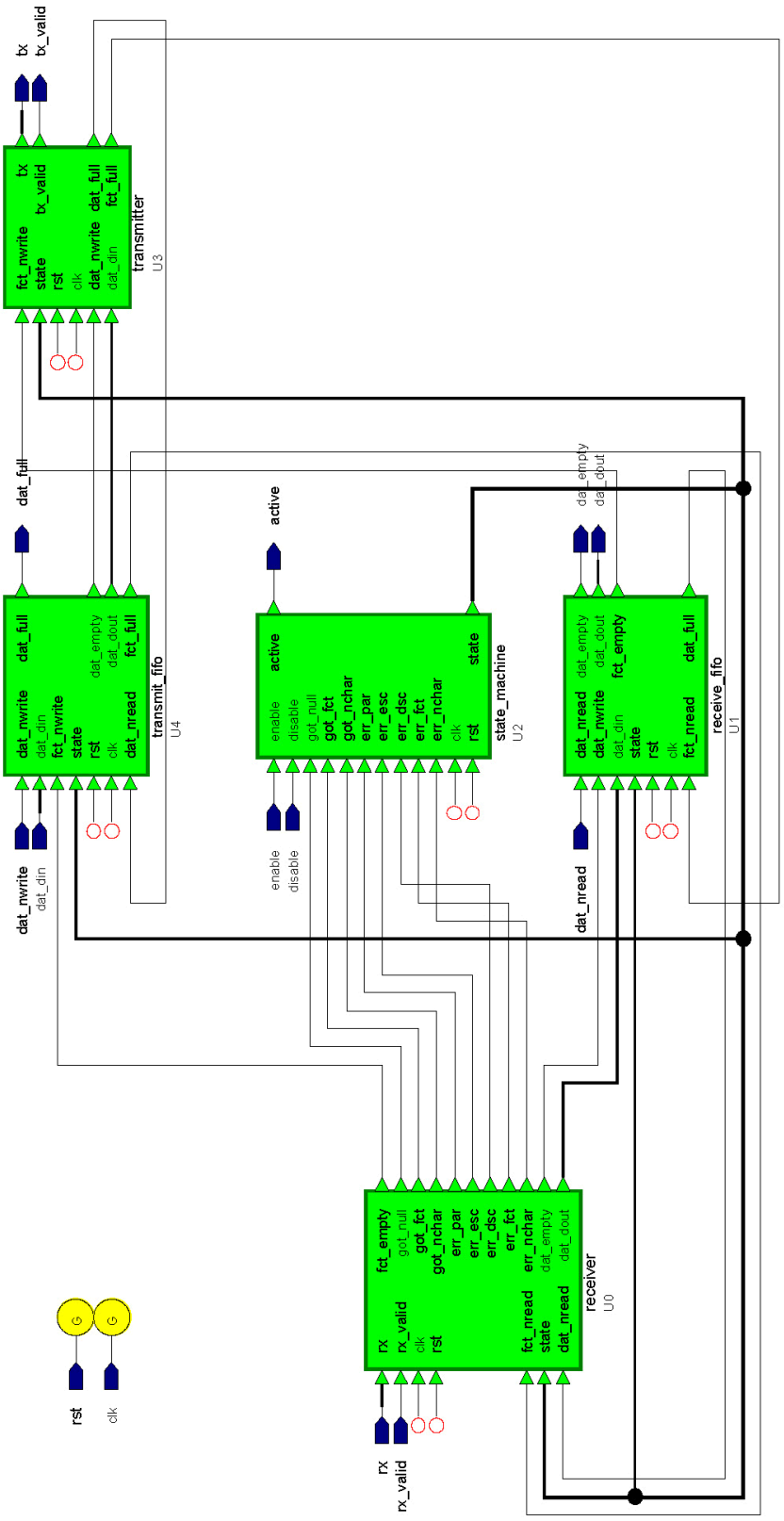
Figure 15: SoCWire Switch 8 bit data word width synthesis report

6 Reference

[1] ECSS, *Space Engineering: SpaceWire–Links, nodes, routers, and networks*, ESA-ESTEC, Noordwijk Netherlands, January 2003, ECSS-E-50-12A

[2] B. Osterloh, H. Michalik, and B. Fiethe, "SoCWire: A Robust and Fault Tolerant Network-on-Chip Approach for a Dynamic Reconfigurable System-on-Chip in FPGAs," in *Architecture of Computing Systems - ARCS 2009*. vol. 5455 Delft,Netherlands: Springer Berlin / Heidelberg, 2009, pp. 50-59.

7 Appendix



SoCWire CODEC schematic

SoCWire Codec resource utilization synthesize report
Xilinx Virtex-4 LX60-10

					LX 60 Total number of					
					SLICEs:	26624	LUTs:	53248		
Data Word Width	SoCWire CODEC Resource Utilization				FFs:	53248	BRAMs:	160	Core CLK Frequency MHz	Maximum Data Rate (Gbit/s)
	Slice	Lut	FF	BRAM	Slice %	Lut %	FF %	BRAM %		
8	180	340	143	1	0,676%	0,639%	0,269%	0,625%	219.450	1,76
16	209	389	186	2	0,785%	0,731%	0,349%	1,250%	227.905	3,65
32	257	475	266	3	0,965%	0,892%	0,500%	1,875%	228.084	7,30
64	388	708	431	5	1,457%	1,330%	0,809%	3,125%	225.601	14,44
128	549	977	746	9	2,062%	1,835%	1,401%	5,625%	220.510	28,23

SoCWire Switch resource utilization synthesize report
Xilinx Virtex-4 LX60-10

Data Word Width: 8 Bit					LX 60 Total number of				
	SoCWire Switch Resource Utilization				SLICEs:	26624	LUTs:	53248	Core CLK Frequency MHz
					FFs:	53248	BRAMs:	160	
Ports	Slice	Lut	FF	BRAM	Slice %	Lut %	FF %	BRAM %	
2	410	778	312	2	1,5%	1,5%	0,6%	1,3%	204.365
3	699	1333	489	3	2,6%	2,5%	0,9%	1,9%	196.277
4	1033	1944	660	4	3,9%	3,7%	1,2%	2,5%	188.237
5	1194	2250	855	5	4,5%	4,2%	1,6%	3,1%	176.519
6	1481	2784	1062	6	5,6%	5,2%	2,0%	3,8%	178.102
7	1870	3523	1283	7	7,0%	6,6%	2,4%	4,4%	158.741
8	2204	4139	1512	8	8,3%	7,8%	2,8%	5,0%	157.306
9	2825	5324	1756	9	10,6%	10,0%	3,3%	5,6%	156.837
10	3225	6058	2014	10	12,1%	11,4%	3,8%	6,3%	154.175
11	3589	6724	2277	11	13,5%	12,6%	4,3%	6,9%	153.170
12	4181	7850	2563	12	15,7%	14,7%	4,8%	7,5%	151.412
13	4693	8743	2862	13	17,6%	16,4%	5,4%	8,1%	146.371
14	5233	9775	3161	14	19,7%	18,4%	5,9%	8,8%	144.368
15	5075	9406	3470	15	19,1%	17,7%	6,5%	9,4%	133.478
16	5874	10824	3801	16	22,1%	20,3%	7,1%	10,0%	145.331
17	5945	10699	4143	17	22,3%	20,1%	7,8%	10,6%	134.065
18	6541	11847	4492	18	24,6%	22,2%	8,4%	11,3%	134.533
19	7175	13052	4860	19	26,9%	24,5%	9,1%	11,9%	132.036
20	7842	14259	5254	20	29,5%	26,8%	9,9%	12,5%	133.384
21	8680	15636	5631	21	32,6%	29,4%	10,6%	13,1%	129.565
22	9045	16363	6069	22	34,0%	30,7%	11,4%	13,8%	127.924
23	9658	17654	6462	23	36,3%	33,2%	12,1%	14,4%	125.024
24	10370	18897	6902	24	38,9%	35,5%	13,0%	15,0%	122.556
25	11293	20459	7342	25	42,4%	38,4%	13,8%	15,6%	122.027
26	11851	21894	7781	26	44,5%	41,1%	14,6%	16,3%	120.570
27	13270	24363	8288	27	49,8%	45,8%	15,6%	16,9%	121.036
28	13240	24534	8758	28	49,7%	46,1%	16,4%	17,5%	114.058
29	14048	25816	9208	29	52,8%	48,5%	17,3%	18,1%	115.195
30	14740	27310	9715	30	55,4%	51,3%	18,2%	18,8%	110.746
31	15657	29134	10192	31	58,8%	54,7%	19,1%	19,4%	113.718
32	16603	30587	10698	32	62,4%	57,4%	20,1%	20,0%	113.041

Data Word Width: 16 Bit					LX 60 Total number of				
	SoCWire Switch Resource Utilization				SLICEs:	26624	LUTs:	53248	Core CLK Frequency MHz
					FFs:	53248	BRAMs:	160	
Ports	Slice	Lut	FF	BRAM	Slice %	Lut %	FF %	BRAM %	
2	477	890	404	4	1,8%	1,7%	0,8%	2,5%	205.768
3	822	1562	623	6	3,1%	2,9%	1,2%	3,8%	196.277
4	1199	2252	856	8	4,5%	4,2%	1,6%	5,0%	190.647
5	1432	2680	1096	10	5,4%	5,0%	2,1%	6,3%	181.033
6	1782	3324	1368	12	6,7%	6,2%	2,6%	7,5%	178.102
7	2226	4161	1629	14	8,4%	7,8%	3,1%	8,8%	155.366
8	2623	4891	1888	16	9,9%	9,2%	3,5%	10,0%	154.018
9	3393	6352	2193	18	12,7%	11,9%	4,1%	11,3%	156.705
10	3817	7131	2545	20	14,3%	13,4%	4,8%	12,5%	156.844
11	4260	7945	2853	22	16,0%	14,9%	5,4%	13,8%	152.527
12	4901	9103	3186	24	18,4%	17,1%	6,0%	15,0%	153.576
13	5561	10280	3566	26	20,9%	19,3%	6,7%	16,3%	154.588
14	6185	11483	3935	28	23,2%	21,6%	7,4%	17,5%	151.755
15	6190	11407	4131	30	23,2%	21,4%	7,8%	18,8%	133.499
16	7001	12750	4716	32	26,3%	23,9%	8,9%	20,0%	147.265
17	7301	12833	5056	34	27,4%	24,1%	9,5%	21,3%	134.397
18	7906	14045	5258	36	29,7%	26,4%	9,9%	22,5%	134.533
19	8165	14805	5633	38	30,7%	27,8%	10,6%	23,8%	130.180
20	8920	16208	6138	40	33,5%	30,4%	11,5%	25,0%	130.334
21	9985	17771	6572	42	37,5%	33,4%	12,3%	26,3%	125.432
22	10816	19107	7218	44	40,6%	35,9%	13,6%	27,5%	127.897
23	11549	20803	7550	46	43,4%	39,1%	14,2%	28,8%	126.755
24	12151	22118	8121	48	45,6%	41,5%	15,3%	30,0%	122.462
25	13846	24709	8609	50	52,0%	46,4%	16,2%	31,3%	125.611
26	14495	25947	8949	52	54,4%	48,7%	16,8%	32,5%	125.723
27	15392	28094	9435	54	57,8%	52,8%	17,7%	33,8%	122.760
28	15792	29031	9985	56	59,3%	54,5%	18,8%	35,0%	115.747
29	17096	31019	10507	58	64,2%	58,3%	19,7%	36,3%	115.941
30	18443	33590	11050	60	69,3%	63,1%	20,8%	37,5%	117.128
31	18892	34727	11591	62	71,0%	65,2%	21,8%	38,8%	112.040
32	19423	35721	12271	64	73,0%	67,1%	23,0%	40,0%	115.819

Data Word Width: 32 Bit					LX 60 Total number of				
	SoCWire Switch Resource Utilization				SLICEs:	26624	LUTs:	53248	Core CLK Frequency MHz
					FFs:	53248	BRAMs:	160	
Ports	Slice	Lut	FF	BRAM	Slice %	Lut %	FF %	BRAM %	
2	592	1094	568	6	2,2%	2,1%	1,1%	3,8%	206.511
3	1030	1955	859	9	3,9%	3,7%	1,6%	5,6%	196.277
4	1482	2784	1165	12	5,6%	5,2%	2,2%	7,5%	190.647
5	1760	3280	1506	15	6,6%	6,2%	2,8%	9,4%	181.033
6	2243	4158	1831	18	8,4%	7,8%	3,4%	11,3%	178.102
7	2884	5342	2178	21	10,8%	10,0%	4,1%	13,1%	155.366
8	3430	6348	2552	24	12,9%	11,9%	4,8%	15,0%	172.375
9	4357	8063	2941	27	16,4%	15,1%	5,5%	16,9%	161.430
10	4928	9118	3348	30	18,5%	17,1%	6,3%	18,8%	157.270
11	5559	10304	3734	33	20,9%	19,4%	7,0%	20,6%	152.581
12	6192	11365	4165	36	23,3%	21,3%	7,8%	22,5%	152.154
13	7232	13255	4628	39	27,2%	24,9%	8,7%	24,4%	154.588
14	7999	14704	5101	42	30,0%	27,6%	9,6%	26,3%	151.361
15	8520	15735	5486	45	32,0%	29,6%	10,3%	28,1%	139.396
16	9036	16127	6021	48	33,9%	30,3%	11,3%	30,0%	145.064
17	9636	16582	6408	51	36,2%	31,1%	12,0%	31,9%	132.239
18	10507	18156	6807	54	39,5%	34,1%	12,8%	33,8%	134.533
19	11351	20163	7346	57	42,6%	37,9%	13,8%	35,6%	132.036
20	11951	21212	7958	60	44,9%	39,8%	14,9%	37,5%	132.965
21	13405	23308	8231	63	50,3%	43,8%	15,5%	39,4%	127.445
22	14418	25135	9057	66	54,2%	47,2%	17,0%	41,3%	127.736
23	15282	27349	9617	69	57,4%	51,4%	18,1%	43,1%	125.024
24	16498	29680	10129	72	62,0%	55,7%	19,0%	45,0%	126.592
25	17970	31628	10661	75	67,5%	59,4%	20,0%	46,9%	123.666
26	18882	33230	11312	78	70,9%	62,4%	21,2%	48,8%	124.096
27	19397	35201	11847	81	72,9%	66,1%	22,2%	50,6%	116.197
28	20811	38166	12327	84	78,2%	71,7%	23,2%	52,5%	109.093
29	22720	40856	12885	87	85,3%	76,7%	24,2%	54,4%	109.631
30	23556	42052	13391	90	88,5%	79,0%	25,1%	56,3%	113.501
31	24768	45530	14185	93	93,0%	85,5%	26,6%	58,1%	111.849
32	25593	46720	14851	97	96,1%	87,7%	27,9%	60,6%	113.162