TDM controller core

Jamil Khatib

April 11, 2001

Contents

1 List of authors and changes

Name	Changes	Date	Contact address
Jamil Khatib	Initial release	3-2-2001	khatib@ieee.org
Jamil Khatib	General review and CPU interface added	10-2-2001	khatib@ieee.org
Jamil Khatib	ISDN support added	3-4-2001	khatib@ieee.org
Jamil Khatib	Buffer Calculations added	9-4-2001	khatib@ieee.org

2 Project Definition

2.1 Introduction

Time devision multiplexing is a scheme used to communicate between systems or devices via shared interface lines. Each device or system gets the access to this interface in a single time slot.

2.2 Objectives

The aim of this project is to develop the basic TDM functionalities to be used by many communication systems like ISDN, E1, T1 and voice codecs.

3 Specifications

3.1 System Features Specification

1. Supports E1 bit rate and time slots (32 time slots or 32 DS0 channels at bit rate 2.048Mbps)
2. Supports ST-Bus (Serial Telecom bus) interface.
3. Routes time slots to/from HDLC controller via the backend interface and software support or to/from memory.
4. Supports read for all or partial TDM slots from the ST-bus.
5. Supports write for all or partial TDM slots to ST-bus.
6. It supports $N \times 64$ mode (i.e. it supports sampling (or writing) to N consecutive time slots)
7. Supports two serial lines one input and one output.
8. Can be connected to other ST-Bus compatible devices via serial or star configurations.
9. If no data is available for transmission it sends all ones.
10. Backend interface uses the Wishbone bus interface which can be connected directly to the system or via FIFO buffer.
11. Optional External FIFO buffer, configuration and status registers.
12. The core will be made of two levels of hierarchies, the basic functionality and the Optional interfaces and buffers which makes it easy to add extra serial lines by duplicating the TDM controllers in parallel.
13. ISDN (2B+D) support can be supported by adding three parallel HDLC controllers on the first three time slots.

3.2 External Interfaces

Signal name	Direction	Description
Control interface		
Rst_n	Input	System asynchronous reset (active low)
Size[4:0]	Input	Number of time slots (Can be fixed)
Serial Interface (ST-Bus)		
C2	Input	Bus Clock
DSTi	Input	Receive serial Data
DSTo	Output	Transmit serial Data
F0_n	Input	Framing pulse (active low)
F0od_n	Output	Delayed Framing pulse (active low)
Back-end Interface (Received)		
RxD[7:0]	Output	Receive data bus
RxValidData	Output	Valid Data
FrameErr	Output	Error in the received data
Read	Input	Read byte
Ready	Output	Valid data exists
Back-end Interface (Transmited)		
TxD[7:0]	Input	Transmit data bus
TxValidData	Input	Valid Data
Write	Input	Write byte
Ready	Output	Ready to get data

3.2.1 Back-end interface mapping to Wishbone SoC bus

The TDM backend interface is divided into two parts one for receive and one for transmit.It can be used as a slave core or master according to the below mapping. The core supports SINGLE READ/WRITE Cycle only using 8-bit data bus without address lines. The choice between master and slave is left for the system integrator and must do the configuration and glue logic as defined in the tables.

Signal Name	Wishbone signal
Master Configuration connected to FIFO	Receive channel
C2	CLK_I
Rst	not RST_I
RxD[7:0]	DAT_O(7:0)
RxValidData	STB_O
RxValidData	CYC_O
Read	ACK_I and not RTY_I
Ready	WE_O
FrameERR	TAG0_O
Slave FIFO(two-clock domain FIFO)	
Data[7:0]	DAT_I(7:0)
Chip Select	STB_I
STB_I and not FullFlag	ACK_O
FullFlag	RTY_O
Write	WE_I
Slave Configuration	
C2	CLK_I
Rst	not RST_I
RxD[7:0]	DAT_O(7:0)
RxValidData	TAG0_O
ReadByte	not WE_I
Ready	not RTY_O
STB_I and not WR_I	ACK_O
FrameERR	TAG1_O

Signal Name	Wishbone signal
Master Configuration connected to FIFO	Transmit channel
C2	CLK_I
Rst	not RST_I
TxD[7:0]	DAT_I(7:0)
Write	ACK_I and not RTY_I
Ready	not WE_O
TxValidData	TAG0_I
Always Active	CYC_O
Always Active	STB_O
Slave FIFO(two-clock domain FIFO)	
Data[7:0]	DAT_I(7:0)
EmptyFlag	RTY_O
Read	WE_I
WE_I and not EmptyFlag	ACK_O
ChipSelect	STB_I
Slave Configuration	
C2	CLK_I
Rst	not RST_I
TxD[7:0]	DAT_I(7:0)
TxValidData	STB_I
Write	WE_I
Ready	not RTY_O
STB_I and WR_I	ACK_O

3.2.2 CPU interface

This interface is used when the FIFO and registers are included in the Core. This interface is compatible to WishBone slave bus interface that supports single read/write cycles and block cycles. The interface supports the following wishbone signals.

Signal	Note
RST_I	Reset
CLK_I	Clock
ADR_I(2:0)	3-bit address line
DAT_O(7:0)	8-bit receive data
DAT_I(7:0)	8-bit transmit data
WE_I	Read/write
STB_I	Strobe
ACK_O	Acknowledge
CYC_I	Cycle
RTY_O	Retry
TAG0_O	TxDone interrupt
TAG1_O	RxReady interrupt

4 Internal Blocks

5 Design description

5.1 ST-Bus interface

The TDM controller interfaces to the TDM lines via serial telecom bus. The interface uses the external input clock (2.048MHz) for all of the internal serial logic. It detects the incoming framing pulse to synchronize the sampling and transmission of bits. The core reads and writes only the specified number of TDM channels (8-bits) by the size bus (No. of channels register). In the transmission mode the output pin should be disabled after writing the configured time slots. It generates also the output delayed framing pulse after it samples all the specified bits (TDM channels). This feature can be used to cascade controllers for different TDM channels.

5.1.1 Design notes

5.1.2 Timing

5.2 External FIFO

The controller has optional external FIFO buffers, one for data to be transmitted and one for data to be received. Status and control registers are available to control these FIFOs. These two blocks (FIFOs and registers) are built around the TDM controller core which make them optional if the core is to be used in different kind of applications.

The current implementation supports the following configuration: The size of the Transmit and receive FIFOs is (8×32) bits which enables the whole TDM frame to be buffered.

The transmit buffer is used to prevent underflow while transmitting bytes to the line. All bytes will be available once the transmit is enabled. If the transmit FIFO is empty the core will transmit ones. The Receive buffer is used to provide data burst transfer to the Back end interface which prevents the back end from reading each byte alone. The FIFO size is suitable for operating frequencies 2.048MHz on the serial interface and 20 MHz on the back end interface. Other frequencies can operate if the back end can read the entire TDM frame before the first byte of the next frame is written (the next calculations is an example to be applied for different frequencies)

8 bits (Time needed to receive the first byte of the next frame) / 2.048MHz = 3.9 us

32 Bytes (Maximum frame size) / 20MHz = 1.6 us

These FIFOs are implemented on Single port memory. It is the responsibility of the external interface to write/read data to/from the FIFOs.

TxDone and RxRdy interrupts are generated when the Tx buffer is empty and Rx buffer has data respectively .

5.2.1 Notes

- **Transmit Operation:** If the transmit FIFO is empty not enough data bytes is available according to no. of channels (caused by incomplete burst transfer, the core sets the Aborted bit in the TX status and control register and sends all ones in the transmit serial line.
- **Transmit Operation:** The back end (software) should write data to the Tx buffer register according to the configured number of time slots. The transmission will start only after the specified number of slots are available in the buffer other wise Aborted bit of the Tx Status register will be set and all ones will be transmitted in this slot.
- **Receive Operation:** When Receive FIFO is full It drops the second FIFO contents and sets overflow bit in the Rx Status and Control register.
- **Receive Operation:** When RxRdy Interrupt is asserted (or RxRdy bit is set) the back end interface (software) must read the specified number of slots from the Rx Data buffer register or the buffer will not be marked as empty.

5.3 ISDN support

In order to provide $(2B + D)$ ISDN support three HDLC controllers should be used on three time slots. The serial data the of first three time slots will enter (or get out) directly to (from) the three parallel HDLC controllers if HDLCen bit is set in the Tx Status and Control register. The HDLC controllers will be managed through the enable signals (each controller will be enabled on its corresponding time slot). These HDLC controllers will set in parallel with the Rx and Tx buffers (as shown in the figure) which still can be used even if the ISDN mode is enabled.

5.4 Registers

All internal registers are 8-bit width.

5.4.1 Transmit

Tx Status and Control Register: Tx_SC Offset Address = 0x0

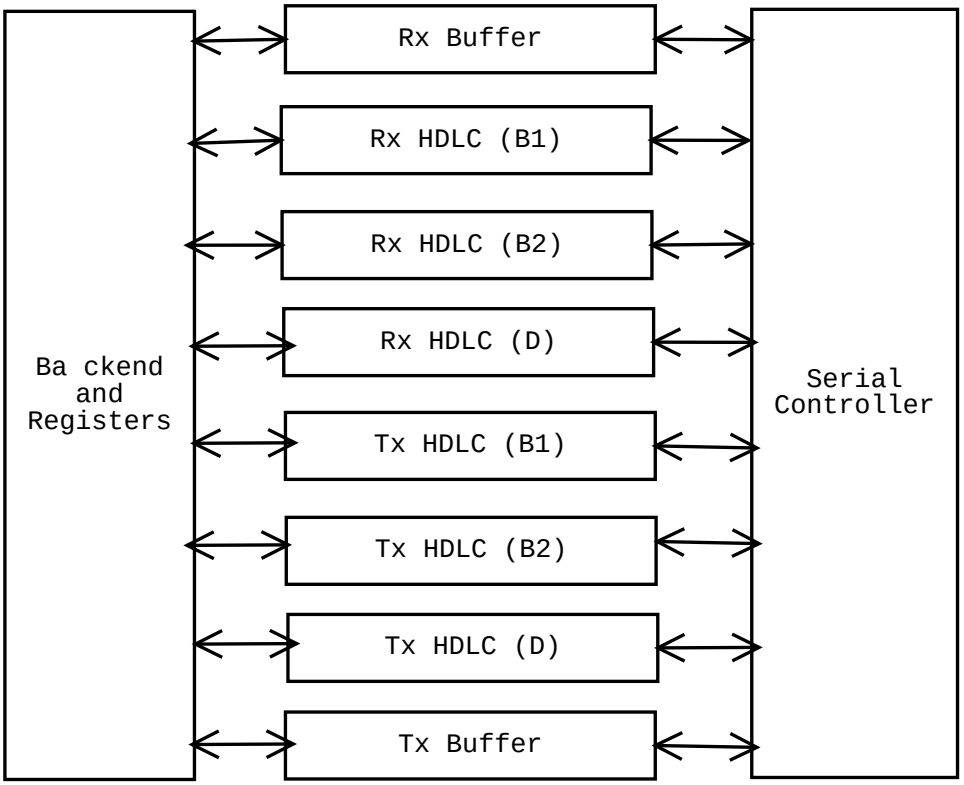

Figure 1: ISDN support

BIT	7	6	5	4	3	2	1	0
FIELD	N/A	N/A	N/A	N/A	HDLCen	Aborted	TxEnable	TxReady(empty)
RESET	0	0	0	0	0	0	0	0
R/W	RO	RO	RO	RO	WO	RO	WO	RO

Tx FIFO buffer register: Tx_Buffer Offset Address = 0x1

BIT	7-0
FIELD	Transmit Data byte
RESET	0x0
R/W	WO

5.4.2 Receive

Rx Status and Control Register: Rx_SC Offset Address = 0x2

BIT	7	6	5	4	3	2	1	0
FIELD	N/A	N/A	N/A	N/A	N/A	FrameError	Drop	RxReady(Full)
RESET	0	0	0	0	0	0	0	0
R/W	RO	RO	RO	RO	RO	RO	WO	RO

Rx FIFO buffer register: Rx_Buffer Offset Address = 0x3

BIT	7-0
FIELD	Received Data byte
RESET	0x0
R/W	RO

configuration register: CFG Offset Address = 0x4

BIT	7-0
FIELD	No. of channels
RESET	0xFF
R/W	RO

This register defines number of time slots will be sampled and written after the framing pulse.

HDLC registers Each HDLC controller its own registers as described in the HDLC controller document but with the offset address as 0xY0 + z where Y represents the HDLC channel number and z the internal HDLC register offset. For example Tx_SC register of the second HDLC controller in the TDM controller will be mapped to 0x20 + 0x0 = 0x20

5.5 Diagrams

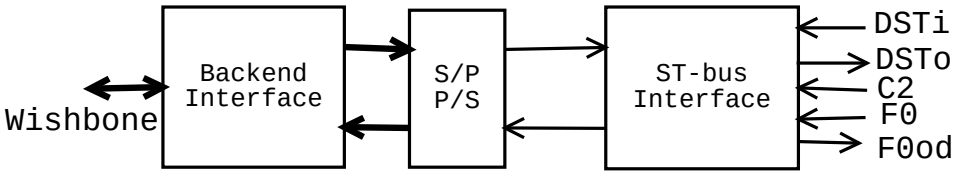

Figure 2: TDM core

6 Testing and verifications

Requirement	Test method	Validation method
Interface timing		
Functionality		

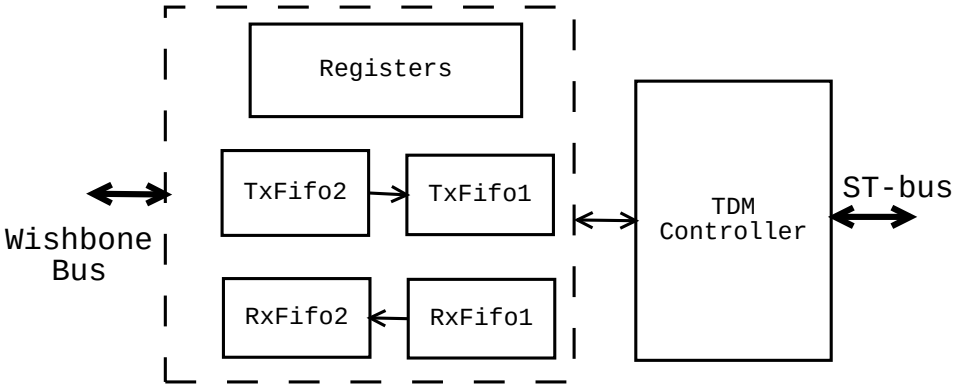

Figure 3: TDM controller

6.1 Simulation and Test benches

6.2 Verification techniques and algorithms

6.3 Test plans

7 Implementations

7.1 Scripts, files and any other information

7.2 Design conventions and coding styles

8 Reviews and comments

9 References