LIN controller

Designed by

PatakEngineering
www.patakengineering.eu

Version: 1.0 Date: May 2014

Contents

List of Figures

List of Tables

LIN controller

1.1 Overview

LIN is very similar to UART but it contains synchronization break which is 13 bits long and therefore it cannot be implemented by a classic UART unit, which is a part of every μController. Some controllers support the LIN frame functionality, but the interface is the same as for an UART.

The LIN frame consists of a synchronization break, byte 0x55, ID, up to eight data bytes, and a checksum. Implementing this in the μController takes a lot of computing time, because each byte must be sent and read separately. Most of IPs designed for LIN functionality are designed this way.

The LIN controller, designed in here, offers the whole frame transfer and therefore, it is much easier and more effective to handle the frame transfer by a μController.

The LIN controller, the core written in VHDL, offers one Rx buffer (buffer for a full frame), filter of IDs, and a Tx buffer. This controller can therefore be used as a master, a slave, or just for listening to all messages on the bus.

It significantly reduces the processor computation time. For example, it can be set to transmit whole frame, and then cause an interrupt (either because a frame has been successfully transferred, or an error occurred).

1.2 Interface

1.2.1 Overview

The LIN controller IP is a core with an Avalon MM bus interface.

The interface signals are shown in figure 1.1 and their description can be found in table 1.1.

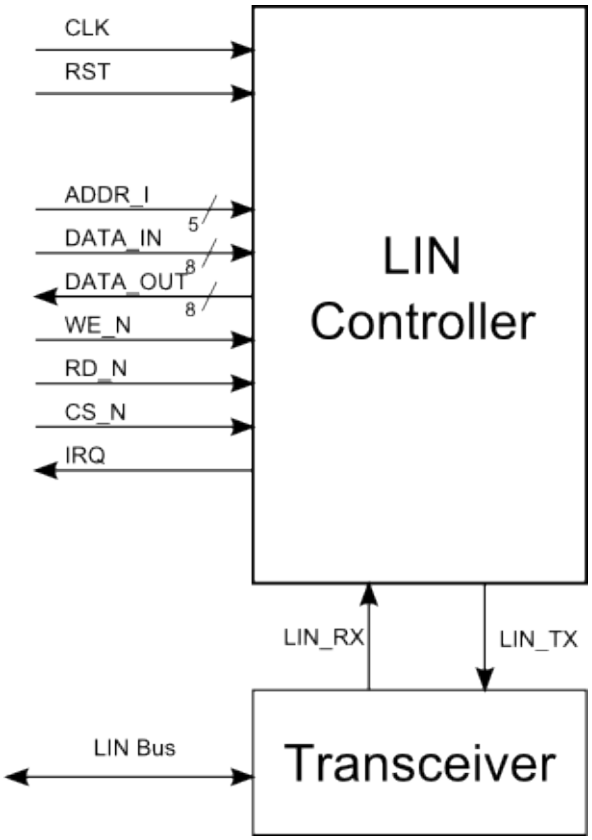

Figure 1.1: Interface

Pin	Activity	Description
CLK	-	Clock input
RST	HIGH	Reset signal
LIN_RX	-	Receive data from the LIN transceiver
LIN_TX	-	Transmit data to the LIN transceiver
ADDR_I[4:0]	-	Address of the register
DATA_IN[7:0]	-	Data input (8 bits)
DATA_OUT[7:0]	-	Data output (8 bits)
WE_N	LOW	Bus access signal : LOW for the write transfer
RD_N	LOW	Bus access signal : LOW for the read transfer
CS_N	LOW	Device select bit
IRQ	HIGH	“Interrupt output” bit

Table 1.1: Interface description

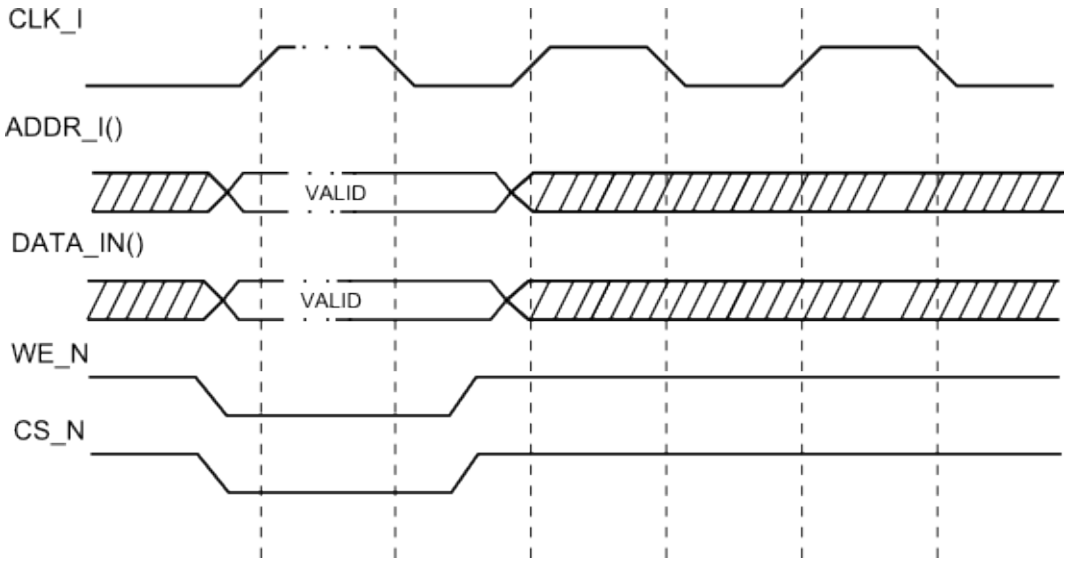

Figure 1.2: Writing to registers

1.2.2 Writing data to registers

Writing data to registers is shown in figure 1.2. Data are written to a register on the rising edge of CLK when the WE_N goes low and CS_N is low.

1.2.3 Reading data from registers

Reading data from registers is shown in figure 1.3. Correct data are present on DATA_OUT latest on the next rising edge of CLK when RD_N went low and CS_N was low.

1.3 Register description

1.3.1 LIN controller general settings

General settings described in the table 1.2 set behavior and indicate the basic statuses of the LIN controller. Values after reset can be found in the table 1.3.

1.3.1.1 CONTROL1

This read/write register contains LIN controller settings

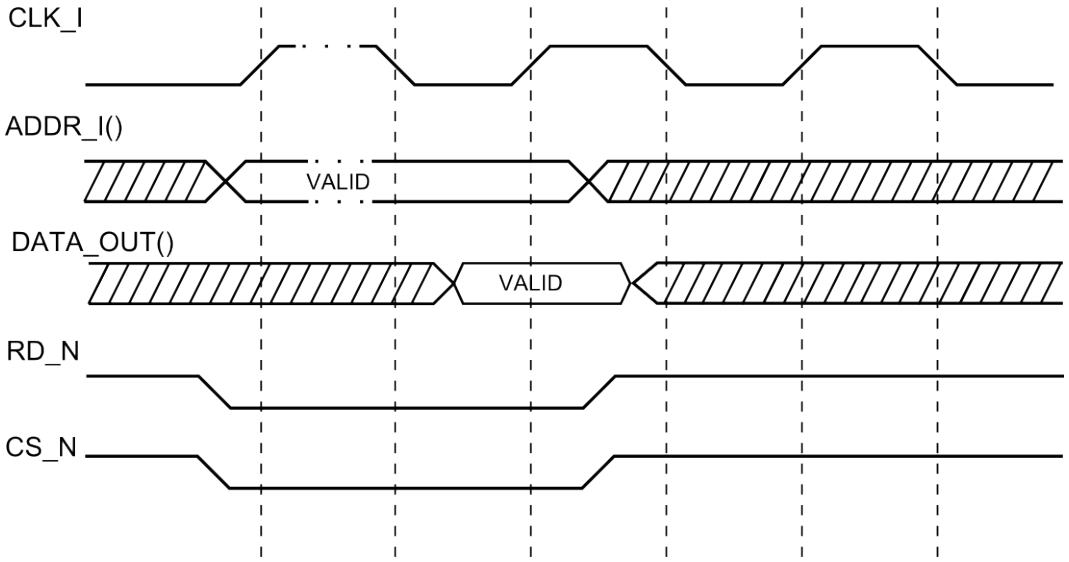

Figure 1.3: Reading data from registers

Offset	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	CONTROL1	-	-	-	-	-	-	ABAU	MASL
0x01	CONTROL2	-	-	-	IWAKE	IERR	IHDR	ITX	IRX
0x02	STATUS1	-	-	IDL	WAKE	ERR	HDR	TDRE	RDRE
0x03	ERRORS	-	-	NOSLR	XMIT	FRAM	PAR	CKS	ORUN
0x04	ID_MASK	-	-	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x05	ID_FILT	-	-	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x06	CLK_DIV1	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x07	CLK_DIV2	-	-	-	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
0x08	FRM_CONF	ENHA	-	-	-	LEN3	LEN2	LEN1	LEN0

Table 1.2: General registers

MASL 0 - Controller is slave
1 - Controller is master

ABAU 0 - auto baud rate disabled
1 - auto baud rate enabled (slave controller only) [1]

1.3.1.2 CONTROL2

This read/write register contains the LIN driver interrupt settings

IRX interrupt when the Rx register is not empty
0 - interrupt disabled
1 - interrupt enabled

ITX interrupt when the Tx register is empty
0 - interrupt disabled
1 - interrupt enabled

IHDR interrupt when header received
0 - interrupt disabled
1 - interrupt enabled

IERR interrupt when an error occurred
0 - interrupt disabled
1 - interrupt enabled

IWAKE interrupt when a wake up signal occurred
0 - interrupt disabled
1 - interrupt enabled

1.3.1.3 STATUS1

This read/write register contains information about the state of the LIN controller

RDRE 0 - Rx buffer is empty
1 - Rx buffer is not empty

[1] CLK_DIV register must be set to a bitrate which might be the fastest on the bus. This baudrate is used to measure break symbol. The autobaudrate can then be max CLK_DIV. The lowest bitrate is limited to $CLK\_DIV \cdot 4$, that means that the detected baudrate cannot be more than four times slower than the one specified in CLK_DIV registers.

TDRE 0 - All Tx registers are full
1 - Not all Tx registers are full

HDR 0 - No header received
1 - A header received
Cleared when STATUS1 register is read

ERR 0 - No error flags set
1 - An error flag is set
Cleared when ERRORS register is read

WAKE 0 - No break was generated on the bus
1 - a break or wake up signal was generated
Cleared when STATUS1 register is read [2]

IDL 0 - The controller is active
1 - The controller is idle , this flag can be used just when the controller is configured as a master

1.3.1.4 ERRORS

This read-only register contains the LIN bus error flags. It is cleared to the reset state when read.

ORUN 1 - Message was lost. The Rx buffer was full.

CKS 1 - Checksum error in a received frame.

PAR 1- Parity error was detected in the ID.

FRAM 1 - Framing error was encountered.

XMIT 1 - Bit error during transmission (send '0' received '1' or send '1' and received '0').

NOSLR 1 - No slave response within the time-out [3] ($T_{Frame\_Maximum}$).

[2]It is generated when dominant state appeared on the bus for longer than 11 bits. It is generated even if master initializes the start.

[3]Only active when autobaudrate is disabled.

CONTROL1	0x0
CONTROL2	0x0
STATUS1	0x2
ERRORS	0x0
ID_MASK	0x0
ID_FILT	0x0
CLK_DIV1	0x2
CLK_DIV2	0x0

Table 1.3: Reset values

1.3.1.5 ID_MASK

Device supports to filter incoming frames. It is supported in slave mode only. If

$$ID\_MASK \; and \; ID\_FILT \; != \; received\_ID \; and \; ID\_MASK \tag{1.1}$$

whole frame is ignored (where 'and' means bitwise 'and'). Therefore, if the ID_MASK is set to 0x00, all frames are received, if it is set to 0x3F, only the frame with ID equal to ID_FILT is received.

This read/write register sets the mask of the ID.

0 - bit must not match

1 - bits must match

1.3.1.6 ID_FILT

This read/write register sets the filter.

1.3.1.7 CLK_DIV

This read/write registers set the bit rate.

$$CLK\_DIV = \frac{clk}{16 \cdot bit\_rate} \tag{1.2}$$

Divisors 1 and 0 are invalid.

1.3.1.8 FRM_CONF

Frame configuration, set the count of data bytes (LEN) and type of checksum. Master device must set these information before a header is transmitted. Slave device sets this

Offset	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0A	TX_ID	-	-	ID 5	ID 4	ID 3	ID 2	ID 1	ID 0
0x0B	TX_SEND	-	-	-	-	-	WAKE	HDR	FRM
0x0C	TX_BYTE_0	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x0D	TX_BYTE_1	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x0E	TX_BYTE_2	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x0F	TX_BYTE_3	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x10	TX_BYTE_4	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x11	TX_BYTE_5	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x12	TX_BYTE_6	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x13	TX_BYTE_7	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0

Table 1.4: Tx registers

register after an ID is received and before the first data byte is received.

LEN message length (allowed values 1 to 8)

ENHA 0 - use classic checksum
1 - use enhanced checksum

1.3.2 LIN controller Tx registers

Tx register stores whole message which should be transmitted.

1.3.2.1 TX_ID

If device is configured as a master - this ID is transmitted to a bus (data bytes are transmitted if bit FRM in TX_SEND is set). In case that the device is configured as slave - device automatically reply to a received ID if it is equal to TX_ID.

1.3.2.2 TX_SEND

FRM 1 - Send whole frame (device configured as master) or a response (device configured as slave)

Offset	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x15	RX_ID	IGN	-	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x16	RX_READ	-	-	-	-	-	-	-	-
0x18	RX_BYTE_0	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x19	RX_BYTE_1	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x1A	RX_BYTE_2	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x1B	RX_BYTE_3	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x1C	RX_BYTE_4	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x1D	RX_BYTE_5	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x1E	RX_BYTE_6	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0x1F	RX_BYTE_7	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0

Table 1.5: Rx registers

HDR 1 - Controller sends just header (device must be configured as a master). Master device can send a header any time allowing to stop the current communication. Therefore, the μController should send a break after a successful data transfer is finished or device is in idle (bit IDL in STATUS1 register) or after a timeout (bit NOSLR in ERRORS register).

WAKE 1 - Send a wake up symbol. This symbol is equal to break (13 bits), but it can be transmitted by master as well as by slave.

1.3.2.3 TX_BYTE

This read/write registers set bytes in the message

1.3.3 LIN controller Rx registers

Tx register stores whole message which was received.

1.3.3.1 RX_ID

This read/write register holds the ID of the received message (read only).

Writing bit IGN causes controller to stop receiving a message. This can be used e.g. in case that the received ID passed ID filter but the device does not know the count of data bytes. Therefore, to avoid an error, after receiving and ID with unknown count of data bytes/type of checksum, write IGN bit to this register and the controller will wait for a next header. This can be only used in slave mode and when the device in not transmitting data.

1.3.3.2 RX_READ

This write-only register (any data) sets the flag that this message was read. The next message reception is allowed. If a received message was not marked as read and a new ID is received a flag ORUN in ERROR register is set and message is ignored.

1.3.3.3 RX_BYTE

This read-only register holds the data bytes of the received message.

1.4 Examples how to use the LIN core

This section describes briefly how to handle the communication with core co control the frame transfer.

1.4.1 LIN controller configured as a master

After start the μController must configure the registers:

- CONTROL1 - bit MASL to '1'
- CLK_DIV to have proper bit rate
- bits in CONTROL2 register if an interrupt is required. It is not necessary if pooling is used.

TX data Core is going to transmit whole frame (header + data bytes) - master to slave communication. The following registers should be set:

  - TX_ID - set the ID which should be transmitted.

- TX_BYTEs - set all data bytes
- Set ENHA and LENx bits in register FRM_CONF. This provides the type of checksum which to use and number of data bytes which should be transmitted.
- Write FRM bit to register TX_SEND to transmit whole message.
- Wait till the bit TDRE in STATUS1 register is set to '1' (message transmitted) or wait for an interrupt.

RX data Controller sends just a header, slave device responds with data bytes and checksum - slave to master communication.

- TX_ID - set the ID which should be transmitted
- Set ENHA and LENx bits in register FRM_CONF. This provides the type of checksum which to use and number of data bytes which should be received.
- Write HDR bit to register TX_SEND to transmit header only.
- Wait till the bit RDRE in STATUS1 register is set to '1' (message received) or wait for an interrupt.

1.4.2 LIN controller configured as a slave

After start the μController must configure the registers:

- CLK_DIV to have proper bit rate, even if the autobaudrate is used, it is necessary to set the CLK_DIV registers. It should be set to a highest bitrate that can be detected on the bus.
- bits in CONTROL2 register if an interrupt is required. It is not necessary if pooling is used.
- Set ID_MASK and ID_FILT if device should receive just selected IDs.

TX data Core is going to transmit data bytes after a specific ID is received - slave to master communication. The following sequence should be executed:

- Wait till a header (ID) is received (either interrupt or pooling), HDR bit in STATUS1 register is set. ID is stored in RX_ID register.
- TX_BYTEs - set all data bytes

- TX_ID - set the ID to match the received ID
- Set ENHA and LENx bits in register FRM_CONF. This provides the type of checksum which to use and number of data bytes which should be transmitted.
- Write FRM bit to register TX_SEND to transmit a message.
- Wait till the bit TDRE in STATUS1 register is set to '1' (message transmitted) or wait for an interrupt.

RX data Core is going to receive data bytes after a specific ID is received - master to slave communication. The following sequence should be executed:

- Wait till a header (ID) is received (either interrupt or pooling), HDR bit in STATUS1 register is set. ID is stored in RX_ID register.
- Set ENHA and LENx bits in register FRM_CONF. This provides the type of checksum which to use and number of data bytes which should be transmitted.
- Wait till the bit RDRE in STATUS1 register is set to '1' (message received) or wait for an interrupt.
- received data are available in RX_BYTE registers.

1.5 Controller architecture

The LIN controller consists of eight components: configuration registers, core state machine, checksum generator, clk divider, break detector, transmitter, majority sampler and receiver.. The block diagram[4] is shown in figure 1.4.

1.5.1 Checksum gen

The checksum is calculated according to the LIN specification. This component contains a parallel data input and output, and control signals for communication with core state machine.

[4]However, not all connections are showed.

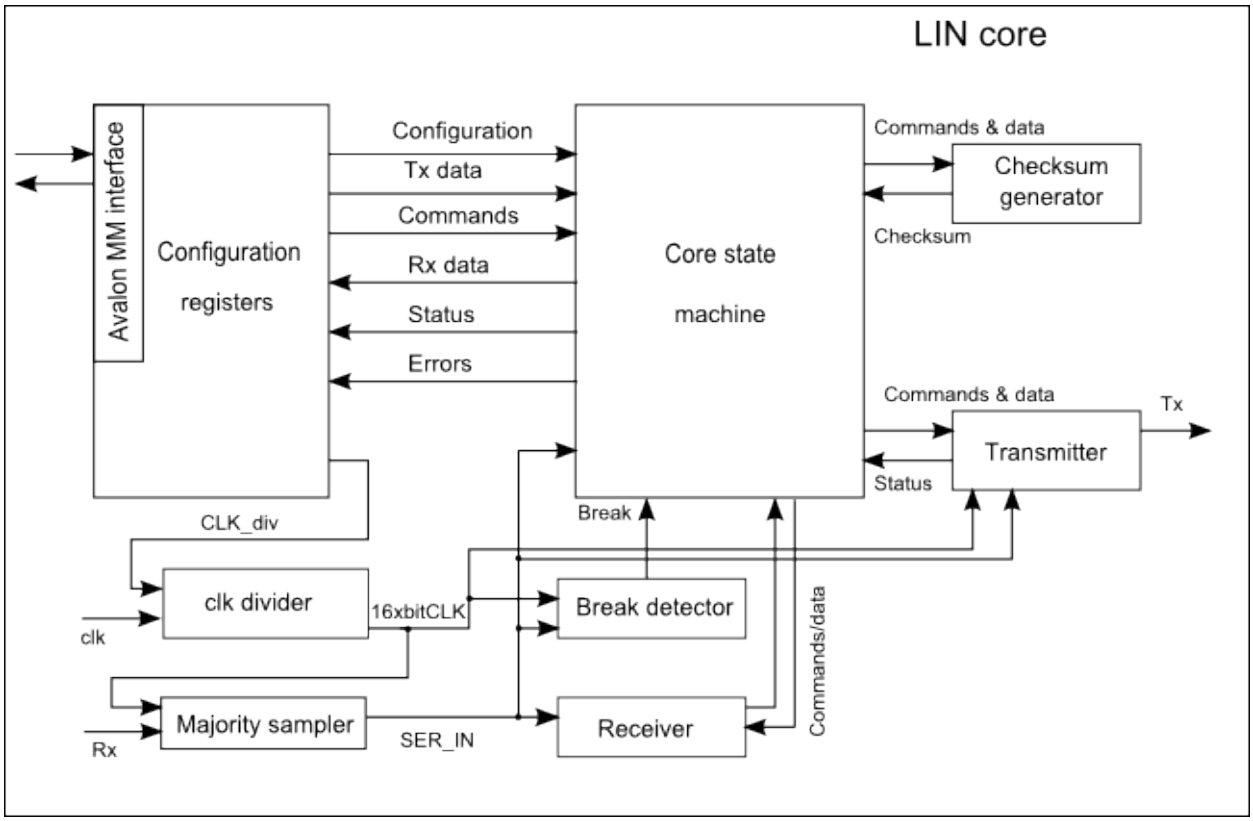

Figure 1.4: Schematics

1.5.2 Clock divider

The clock divider generates a bit rate from the clock input. This bit rate is adjustable by clk_div1 and clk_div2 registers. It generates a clock which is 16 times faster than the bit rate.

1.5.3 Majority sampler

The majority sampler uses bit_clkx16 (16 times faster clock than a bit clock), and makes an average of the last 16 samples. It causes the output (SER_IN) to be delayed by 0.5 bit.

1.5.4 Receiver

The receiver is implemented as a shift register. It is synchronized on the start bit and it takes the sample in the middle of the bit. The majority sampler delays the input for 0.5 bit and receiver samples in the middle of the bit. Therefore, the bit reception is delayed by one bit. The receiver also supports calculating the automatic bit rate for which a 18 bit counter is needed[5].

1.5.5 Transmitter

The transmitter receives the whole byte and transmits it to the bus (generating the start and stop bits) or a break symbol. After the byte is transmitted, the transmitter sets a signal that the transmission is finished.

The transmitter is also responsible for checking whether the transmitted data are equal to the received data. If they do not match, the xmit_error flag is set.

1.5.6 Configuration registers

The configuration registers store all settings for the LIN controller. They are accessible via the Avalon MM slave interface (+clock, reset, interrupt). All settings are stored in registers.

[5] 13 bits are used because CLK_DIV can have 13 bits, 4 more bits are needed for dividing CLKx16, and the other 1 bit is used to count up to two bits

total logic elements	total registers	total memory bits	max frequency [MHz]
715 (16%)	397	0 (0%)	131.2

Table 1.6: Implementation in the FPGA

1.5.7 Core state machine

The core state machine includes three main processes: slave_prc, master_prc and slave_tx_prc.

- Slave_prc process receives all data from the bus.
- master_prc process sends the header (break, sync field and ID).
- slave_tx_prc process is responsible for sending data fields and checksum.

1.6 Summary

The LIN driver has been tested on the Cyclone II EP2C5T144C8. The compilation was made with the respect to the highest speed in the Quartus II version 12.0. The result can be found in table 1.6. The maximal frequency (according to slow model) is 131.2 MHz, which is sufficient to support the maximum LIN bit rate of 20 kbps and easy handling by a μController.