FlexRay controller

Author: Martin Paták

Prague, 2012

Contents

List of Figures

List of Tables

Chapter 1

FlexRay controller

1.1 Overview

FlexRay controllers, available on the market, support only basic functionality (they behave as a normal FlexRay device). Therefore, these controllers are not appropriate for a testing system which requires advanced functionalities.

The controller should offer standard and non-standard functionalities such as direct controlling of states, error handling, and many others. Therefore, a new controller had to be developed for such a system which satisfies all the functionalities mentioned in chapter **??**.

The FlexRay controller, a core written in VHDL, offers adjustable number of Rx buffers with ID filter, an adjustable number of Tx buffers with extern or time stamp triggering. This controller can be used as a non cold start node, leading cold start node, passive observer, or even as a unit disturbing the communication on the bus. The controller offers more functionalities than mentioned in here, the detailed description can be found in the register description. The user have to decide which function to use and how to combine them together in order to test a specific parameter of the bus, node or cluster.

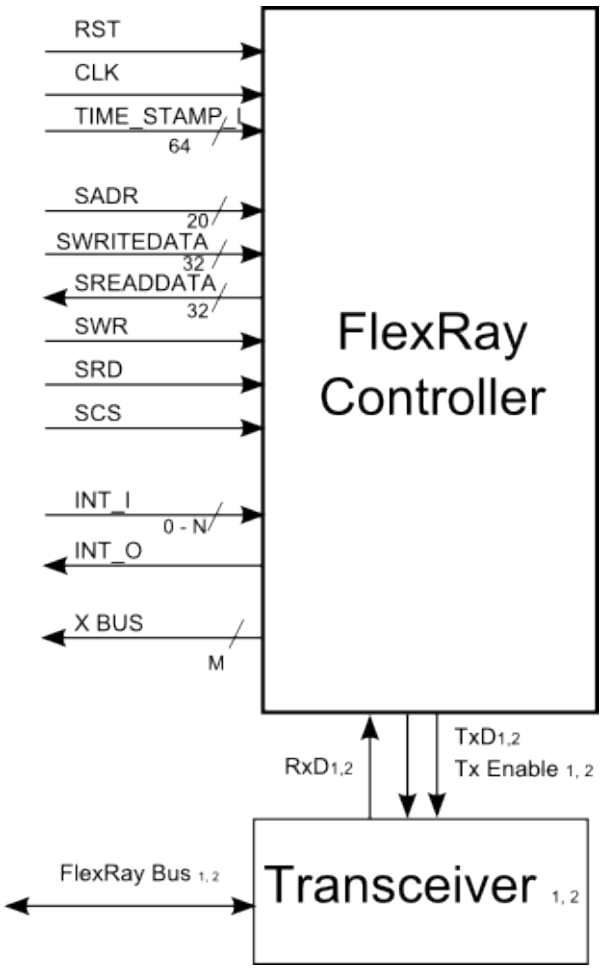

Figure 1.1: Interface

1.2 Interface

1.2.1 Overview

The FlexRay controller IP is a core with a parallel interface (32-bit). The interface is compatible with the Nios II bus system. Therefore, the Nios II core can directly access the registers (accessing the registers as a normal RAM memory)

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

Pin	Activity	Description
RST	HIGH	Reset signal
CLK	-	Clock input
TIME_STAMP_I[63:0]	-	Time stamp input (63 : 0 bit)
SADR[19:0]	-	Address of the register
SWRITEDATA[31:0]	-	Data input (32 bits)
SREADDATA[31:0]	-	Data output (32 bits)
SWR	HIGH	Bus access signal : HIGH for the write transfer
SRD	HIGH	Bus access signal : HIGH for the read transfer
CSC	HIGH	Device select bit
INT_I[(N-1):0]	HIGH	Interrupt for the TX buffer to start transfer, N - configurable number of Tx buffers
INT_O	HIGH	Interrupt caused by the FlexRayController
CSC	HIGH	Device select bit
RxD[1:0]	-	Receive data from the FlexRay transceiver 1,2
TxD[1:0]	-	Transmit data to the FlexRay transceiver 1,2
Tx_enable[1:0]	-	Enable transceiver Tx 1,2
X bus	-	The bus is described in 1.2.2

Table 1.1: Interface description

1.2.2 X bus

The X bus contains many signals connected to one bus. The purpose of this bus is to provide an easy way in the language VHDL to share data among different components. The X bus is controlled by the FlexRay controller core. The description of bits in this bus can be found in tables A.1, A.2, and A.3 in appendix B. All the connected components can connect to this bus and use signals which are required.

1.3 Register description

The device is separated into three main components. Therefore, the device also contains three main blocks of registers: General settings, Tx registers, Rx registers.

All the registers are 32-bit, however, not all bits in these registers are used. It means that a number with a range 0 to 7 uses just 3 bits (bit 2 (MSB) to bit 0 (LSB)).

This section describes these registers and provides brief explanation for each bit.

1.3.1 FlexRay controller general settings

The controller general settings registers consists of two main parts. The first one sets the basic constants of the cluster. These variables and registers are shown in table 1.2, detailed description of this variables can be found in [1] (the variables are named the same as in the FlexRay standard).

The second part of the controller general settings registers influence the behavior of the controller. These registers enable to start the controller, set the controller's states and another functions. The list of the registers can be found in table 1.3.

1.3.2 STARTUP SETUP

bit 0 - pWakeupChannel - 0 - channel A, 1 - channel B

bit 1 - pKeySlotUsedForStartup

bit 2 - pAllowHaltDueToClock

bit 3 - vColdstartInhibit (when this option is set to '0' (cold start node), the user must set an appropriate periodic startup message into one TX register).

bit 4,5 - vPOCErrorMode ('00' = vPOC_ACTIVE, '01' or '10' vPOC_PASSIVE, '11' = vPOC_COMM_HALT)

bit 7 - skip_integration_coldstart_check - checks this state and enters immediately the cold start join state

bit 8 - Device is active on channel A, it uses channel A for communication

bit 9 - Device is active on channel B, it uses channel B for communication

bit 10 - Start as a TT-L node. If this bit is set to '1', the controller starts in TT-E mode. Two periodic messages must be set in TX registers. When this bit is '0', the device starts in TT-D mode.

1.3.3 STARTUP COMMANDS

bit 0 - CONFIG_COMPLETE - writing '1' to this register causes the controller to go from CONFIGURATION state to READY state, in another states it is ignored

Offset	Access	Name	Range	Default value
0x000	R	DEVICE ID	-	0x12345678
0x004	R/W	gdActionPointOffset	0 to 63	3
0x008	R/W	gdStaticSlot	0 to 661	50
0x00C	R/W	gMacroPerCycle	0 to 16000	5000
0x010	R/W	gNumberOfStaticSlots	0 to 1023	60
0x014	R/W	gOffsetCorrectionStart	0 to 15999	4920
0x018	R/W	pDecodingCorrection	0 to 143	56
0x01C	R/W	pdMaxDrift	0 to 1923	601
0x020	R/W	pMacroInitialOffsetA	0 to 68	5
0x024	R/W	pMacroInitialOffsetB	0 to 68	5
0x028	R/W	pMicroPerCycle	0 to 640000	200000
0x02C	R/W	pOffsetCorrectionOut	0 to 15567	1201
0x030	R/W	pRateCorrectionOut	0 to 1923	600
0x034	R/W	gdSampleClockPeriod	0 to 7	2
0x038	R/W	pClusterDriftDamping	0 to 20	1
0x03C	R/W	gdTSSTransmitter	0 to 15	11
0x040	R/W	pMicroInitialOffsetA	0 to 239	23
0x044	R/W	pMicroInitialOffsetB	0 to 239	23
0x048	R/W	pdAcceptedStartupRange	0 to 1875	300
0x04C	R/W	pDelayCompensationA	0 to 200	1
0x050	R/W	pDelayCompensationB	0 to 200	1
0x054	R	zOffsetCorr	-15567 to +15567	-
0x058	R	zRateCorr	-1923 to +1923	-
0x05C	R/W	pSamplesPerMicrotick	1 to 7	2
0x060	R/W	gdCASRxLowMax	0 to 99	50
0x064	R/W	gdWakeupSymbolTxLow	0 to 60	30
0x068	R/W	gdWakeupSymbolTxIdle	0 to 180	90
0x06C	R/W	gdWakeupSymbolRxLow	0 to 60	50
0x070	R/W	gdWakeupSymbolRxIdle	0 to 180	59
0x074	R/W	gdWakeupSymbolRxWindow	0 to 301	301
0x078	R/W	pWakeupPattern	0 to 63	16
0x07C	R	vCycleCounter	0 to 63	0
0x080	R	vMacroTick	0 to 16000	0
0x084	R/W	pdListenTimeout	1284 - 1283846	401202
0x088	R/W	vColdstartAttempts	2 - 31	10
0x08C	R/W	gMaxWithoutClockCorrectionPassive	1 - 15	10
0x090	R/W	gMaxWithoutClockCorrectionFatal	1 - 15	14
0x094	R/W	pAllowPassiveToActive	1 - 31	20
0x098	R/W	ExternRateControl	0 - 15567	0
0x09C	R/W	ExternOffsetControl	0 - 1923	0
0x0A0	R/W	cdCAS	0 - 200	30

Table 1.2: Core settings - register map

Offset	Access	Name	Range	Default value
0x100	R/W	STARTUP SETUP	-	-
0x104	W	STARTUP COMMANDS	-	-
0x108	R	POCState	-	-
0x10C	R	StartupState	-	-
0x110	R	WakeupStatus	-	-
0x114	R/W	Extended	-	-

Table 1.3: Behavior - register map

bit 1 - SEND_WAKE_UP_REQ writing '1' to this register causes the controller to enter a WAKEUP state (sends a wake up symbol to a channel specified in STARTUP SETUP register). Active in ready state only.

bit 2 - SEND_STARTUP_REQ writing '1' to this register causes the controller to enter the STARTUP state. Active in ready state only.

bit 3 - SEND_TO_CONFIG writing '1' to this register causes the controller to enter a CONFIGURATION state. Active in ready state only.

bit 4 - SEND_FROM_HALT_TO_CONFIG writing '1' to this register causes the controller to enter a CONFIGURATION state. Active in halt state only.

bit 5 - vPOCCHIHaltRequest when set the controller goes immediately to the halt state (from any state)

The device can be in one of the states mentioned shown in figure 1.2. When the device starts or it is restarted it automatically goes to DEFAULT_CONFIG [1] state from which it automatically goes to CONFIG state.

In CONFIG state writing CONFIG_COMPLETE causes the controller to enter READY state, from this state the controller can enter WAKE_UP state by writing WAKE_UP_REQ bit. The controller sends the wake up symbol and returns to READY state and gives the result to WakeupStatus.

From REDY state the controller can enter the STARTUP state by writing STARTUP_REQ. The controller has to be configured for the startup phase. In case that the controller is a cold start node - a message (sync and startup message, it must be sent periodically every cycle) must be configured in the TX registers.

[1] In this state no value in registers is changed

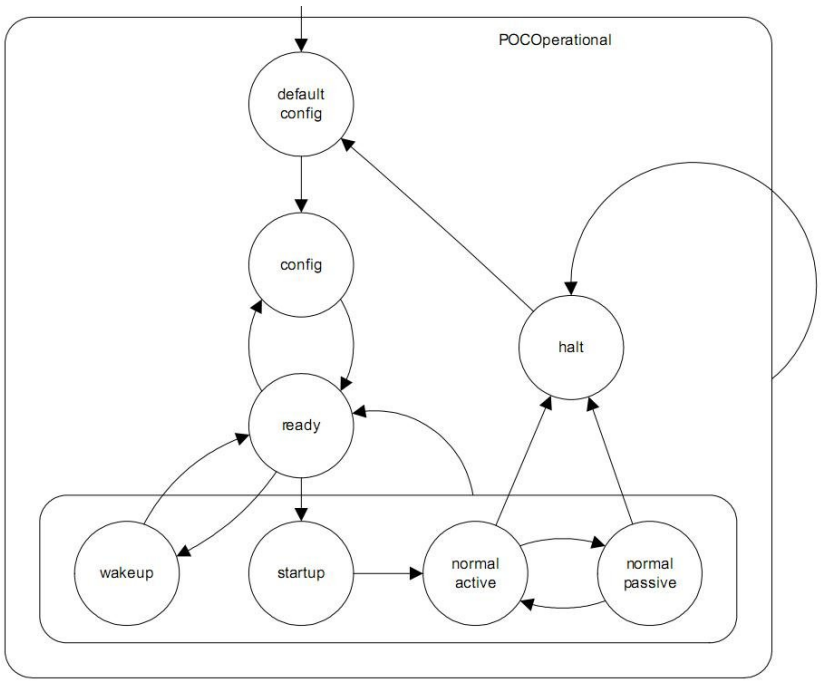

Figure 1.2: Controller states [1]

The controller can return to READY from STARTUP state when some condition mentioned in [1] occurs. In case that the device is successfully integrated, it enters NORMAL_ACTIVE state. It can enter NORMAL_PASSIVE or HALT state under conditions mentioned in the FlexRay standard.

Writing vPOCCHIHaltRequest any time causes the controller to enter HALT state. Halt state can be escaped only by writing FROM_HALT_TO_CONFIG. The device then enters CONFIG state and a new configuration can be made.

1.3.4 POCState

States as defined in the FlexRay standard ('1' means: device is in this state):

bit 0 - CONFIG

bit 1 - DEFAULT_CONFIG

bit 2 - HALT

bit 3 - NORMAL_ACTIVE

bit 4 - NORMAL_PASIVE

bit 5 - READY

bit 6 - STARTUP

bit 7 - WAKEUP

1.3.5 StartupState

States as defined in the FlexRay standard ('1' means: device is in this state):

bit 0 - UNDEFINED

bit 1 - COLDSTART_LISTEN

bit 2 - INTEGRATION_COLDSTART_CHECK

bit 3 - COLDSTART_JOIN

bit 4 - COLDSTART_COLLISION_RESOLUTION

bit 5 - COLDSTART_CONSISTENCY_CHECK

bit 6 - INTEGRATION_LISTEN

bit 7 - INITIALIZE_SCHEDULE

bit 8 - INTEGRATION_CONSISTENCY_CHECK

bit 9 - COLDSTART_GAP

1.3.6 WakeupStatus

Wake up status as defined in the FlexRay standard ('1' means device is in this state):

bit 0 - UNDEFINED

bit 1 - RECEIVED_HEADER

bit 2 - RECEIVED_WUP

bit 3 - COLLISION_HEADER

bit 4 - COLLISION_WUP

bit 5 - COLLISION_UNKNOWN

bit 6 - TRANSMITTED

1.3.7 Extended

bit 0 - When this but is set to 1, the rate correction is disabled. The controller still runs, however the rate correction process provides 0 (or vExternRateCorrection if not zero).

bit 1 - When this but is set to 1, the offset correction is disabled. The controller still runs, however the offset correction process provides (or vExternOffsetCorrection if not zero).

bit 2 - ignoreAllErrors - all errors are ignored in the normal active state, the device stays in normal active state (however the device must already by in this state in order to say in this state)

bit 3 - When this bit is set to 1, then the device skips the startup phase. Writing the SEND_STARTUP_REQ then causes the device to enter immediately (from UNDEFINED state) to normal active state (at this point also macro tick generator is started)

1.3.8 FlexRay controller Tx registers

The Tx registers offer the user to send a wake up frame, a CAS symbol, and a normal frame. The Tx buffer allows to send a frame once time - triggered by the timestamp or by an outer signal. It also allows to send the data periodically to a selected channel/channels.

The number of Tx registers is adjustable in the source code (constant TX_BUFFERS). It is also possible to set the maximal length of the payload. Using the constant BUFF_LENGTH (the same length for Rx and Tx registers) 0 causes that the payload is always empty. In case that the payload of a message is longer than BUFF_LENGTH, an empty bytes will be send out.

Offset	Access	Name	Range
0x200	R/W	TX_BUF	0 to TX_M - 1
0x204	R/W	TX_HEADER1	-
0x208	R/W	TX_HEADER2	-
0x20C	R/W	TX_TRG	-
0x210	R/W	TX_MSG_TYPE	-
0x214	R/W	TX_OPTION	-
0x220	R/W	TX_TIMESTAMP_LOW	-
0x224	R/W	TX_TIMESTAMP_HIGH	-
0x228	R/W	TX_BYTES - N	-
0x22C	R/W	BYTE $(N \cdot 4 + 3)$ to BYTE $(N \cdot 4)$	-

Table 1.4: Tx registers

bit	description
28	Reserved bit (from the header)
27	Payload preamble indicator
26	Null frame indicator
25	Sync frame indicator
24	Startup frame indicator
23:13	Frame ID
12:6	Payload length
5:0	-

Table 1.5: TX_HEADER1 register description

1.3.8.1 TX_BUF

This read/write register selects the TX buffer [2].

1.3.8.2 TX_HEADER_x

The description of bits in these registers is shown in tables 1.5 and 1.6

[2]The number of buffers is adjustable. The maximal number of Tx registers is TX_M (0 to TX_M-1)

bit	description
10:0	Header CRC

Table 1.6: TX_HEADER2 register description

1.3.8.3 TX_TRG

This read/write register determines which signal triggers the transmission.

bit 0 - OUTER - An outer signal (External interrupt) is used to trigger the transmission.

bit 1 - TIMEST_GENERATOR - The time stamp is used to trigger the transmission. If the time is equal to the time stamp, a message is to be transmitted.

bit 2 - TIMEST_MACRO - The time stamp macro tick is used to trigger the transmission. If the time (macrotick of the cluster and the one specified in the registers) is equal to the time stamp, a message is to be transmitted.

bit 3 - TIMEST_MACRO_CYCLE - The time stamp macro tick and cycle (macroticks are equal) is used to trigger the transmission. If the time is equal to the time stamp, a message is to be transmitted.

bit 4 - EVERY_CYCLE - Send message every cycle in the static segment according to a proper action point offset. This bit should be set for all frames in static segment.

bit 5 - EVERY_ODD_CYCLE - The same as EVERY_CYCLE but message is sent only every odd cycle.

bit 6 - EVERY_EVEN_CYCLE - The same as EVERY_CYCLE but message is sent only every even cycle.

bit 7 - IN_DYNAMIC_SEGMENT - message is sent in the dynamic segment. When this bit is set to 0, then it is sent in the static slot[3].

bit 8 - TXNW - The message is sent immediately

bit 31 - BUSY - 0 - Buffer is empty
1 - Buffer is full, when the message is transmitted and the periodic sending is not set, this bit is set to 0. The user must set this bit to 1 in order to be this tx buffer active. The user also has to select what type of event the registers contain in the TX_MSG_TYPE.

Bits, except EVERY_CYCLE, EVERY_ODD_CYCLE, and EVERY_EVEN_CYCLE, are cleared when a frame is transmitted. The bits EVERY_CYCLE, EVERY_ODD_CYCLE, and EVERY_EVEN_CYCLE are cleared only when a SINGLE_MESSAGE flag is used.

[3]Valid for EVERY_CYCLE, EVERY_ODD_CYCLE, and EVERY_EVEN_CYCLE

1.3.8.4 TX_MSG_TYPE

This register determines about the type of the frame. The following bits have the following meaning:

bit 0 - WAKE_UP - Send a wake up symbol

bit 1 - SEND_CAS - Send CAS symbol

bit 2 - SINGLE_MESSAGE - Send a message once when a trigger event occur

bit 3 - SEND_MESAGE_PERIODICALLY - Send message periodically in every cycle. Valid only for EVERY_CYCLE, EVERY_ODD_CYCLE, and EVERY_EVEN_CYCLE.

1.3.8.5 TX_OPTION

This register determines to which channel the data should be send (it is possible to select both channels). It also can influence when to send this message and another options.

bit 0 - CHANNEL A - Send message to channel A

bit 1 - CHANNEL B - Send message to channel B

bit 29 - WRONG_CRC - Send a message with a wrong payload CRC (CRC is inverted)

bit 30 - SEND_EARLIER - The message is sent (X macroTicks - Y μTicks) before a proper action point.

$$time = action\ point - X + Y \tag{1.1}$$

Where AP and X are in macroTick and Y is in μTicks.

The time is specified in macro and μTicks in the TX_TIMESTAMP register in the format given by table 1.7. This option is valid for EVERY_CYCLE, EVERY_ODD_CYCLE, and EVERY_EVEN_CYCLE option for frames in static segment only.

bit 31 - SEND_LATER - The message is sent X μTicks after a proper action point. The time is specified in μTicks in the TX_TIMESTAMP register in the format given by table 1.7. This option is valid for EVERY_CYCLE, EVERY_ODD_CYCLE, and EVERY_EVEN_CYCLE option for frames in static segment only.

The options SEND_EARLIER and SEND_EARLIER allows to send message earlier or later. However, the synchronization process of this controller takes these messages as if they would be transmitted with a zero delay.

	bit	
register	31:16	15:0
TIMESTAMP_LOW	microTick	
TIMESTAMP_HIGH	cycle	macrotick

Table 1.7: Timestamp micro and macro tick description

1.3.8.6 TX_TIMESTAMP

This register contains the timestamps depending on the type of the TIMESTAMP_TYPE. In case that the timestamp is generated by the timestamp generator, the value in the registers gives a 64-bit timestamp.

In case that the micro and macro tick (eventually cycle) are used, the registers have the meaning described in table 1.7.

1.3.8.7 TX_BYTES - N

The value in this register determines which data bytes can be read from the register BYTE $N \cdot 4 + 3$ - BYTE $N \cdot 4$.

1.3.8.8 BYTE $(N \cdot 4 + 3)$ to BYTE $(N \cdot 4)$

This register holds the data bytes of the frame. The selected byte range can be changed by writing another value to the register TX_BYTES - N.

1.3.9 FlexRay controller Rx registers

The Rx registers provide the user the following functionalities:

- Receive symbols
- Receive messages, filtered according to ID and another
- Timestamp events (timestamp generator, FlexRay cluster time(absolute, differential to action point))

The FlexRay Rx registers are shown in the figure 1.3. The depth of the buffer is adjustable (in the source code, variable COUNT_OF_RX_REG, the maximal number of stored payload bytes for each buffer is adjustable in parameter MAX_RX_LENGTH in the source code, when the message is longer than the buffer, the rest of the payload

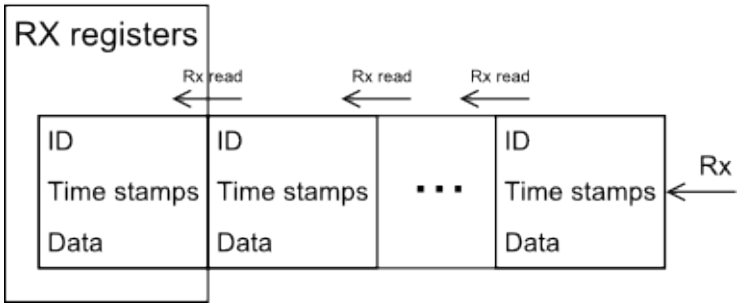

Figure 1.3: Rx buffer

Offset	Access	Name	Range
0x300	R/W	ID_FILTER	0 to 2047
0x304	R/W	ID_MASK	0 to 2047
0x308	R/W	SOURCE	-
0x30C	R/W	TIMESTAMP_TYPE	-
0x310	R/W	RX_IRQ	-
0x320	R	MSG_CNT	0 to COUNT_OF_RX_REG
0x324	R	MESSAGE_STATUS	-
0x328	R	TIMESTAMP_LOW	-
0x32C	R	TIMESTAMP_HIGH	-
0x330	R	HEADER	-
0x334	R/W	RX_BYTES - N	-
0x338	W	RX_READ	-
0x340	R	BYTE $(N \cdot 4 + 3)$ to BYTE $(N \cdot 4)$	-

Table 1.8: Rx registers

is ignored). It behaves like a FIFO (data are stored in the RAM; only a pointer is incremented). It is possible to read the oldest unread message.

The recommended minimal value of the COUNT_OF_RX_REG variable is 4 for device connected to both channels, 2 for device connected to only one channel.

1.3.9.1 ID_FILTER

This read/write register sets the IDs to be received.

1.3.9.2 ID_MASK

This read/write register sets the mask of the ID.
0 - bits must match
1 - bit does not need to match

bit	description
0	record data from channel A
1	record data from channel B
2	record symbols
3	record all frames
4	record startup frames

Table 1.9: SOURCE description

1.3.9.3 SOURCE

The SOURCE description can be found in table 1.9. This option allows the user to catch only desired type of events.

The user can select more than one event to record.

1.3.9.4 TIMESTAMP_TYPE

The user can select type of timestamp and when the timestamp should be recorded (Only one type of timestamp in one time is supported):

bit 0 - Record secondary TRP or time when the event occurred. Timestamp generator is used.

bit 1 - Record secondary TRP or time when the event occurred. Micro and macro tick are used (the macro tick generator must be running, otherwise the timestamp is 0).

bit 2 - Record the time difference between the expected time and time when the message was received (in μTick).

1.3.9.5 RX_IRQ

The user can request an interrupt when a new message was received. The following bit can be set to cause an interrupt event.

bit 0 - Cause an interrupt when a Rx buffer is not empty

bit 1 - Cause an interrupt when a Rx buffer contains (RX_M - 1) messages. There must be an empty space in the buffer for receiving frames. Therefore, at least one empty buffer should remain free.

bit 31 - 1 - buffer was full and a message was lost. Reading this register clears this event.

bit	description
0	Wake up symbol received
1	CAS symbol received
2	Message received
3	Frame received on channel A
4	Frame received on channel B
5	Error in message reception

Table 1.10: MESSAGE_STATUS description

Interrupt is cleared when the count of messages in the buffer is lower than the minimal count specified in this register.

1.3.9.6 MSG_CNT

This read-only register holds the number of messages in the Rx buffer.

1.3.9.7 MESSAGE_STATUS

The table 1.10 shows the meaning of bits.

1.3.9.8 TIMESTAMP_LOW, TIMESTAMP_HIGH

This register contains the timestamps depending on the type of the TIMESTAMP_TYPE. In case that the timestamp is generated by the timestamp generator, the value in the registers gives a 64-bit timestamp.

In case that the micro and macro tick are used, the registers have the meaning described in table 1.7.

In case that the time difference timestamp type is used, the value is stored in TIMESTAMP_LOW(15:0) bits.

1.3.9.9 HEADER

The HEADER register contains data as described in table 1.11.

1.3.9.10 RX_BYTES - N

The value in this register determines which data bytes can be read from the register BYTE $N \cdot 4 + 3$ - BYTE $N \cdot 4$.

bit	description
28	Reserved bit (from the header)
27	Payload preamble indicator
26	Null frame indicator
25	Sync frame indicator
24	Startup frame indicator
23:13	Frame ID
12:6	Payload length
5:0	Cycle count

Table 1.11: HEADER description

1.3.9.11 RX_READ

This write-only register sets the flag that this message was read. The next message (if any) is shifted in the RX registers.

1.3.9.12 BYTE $(N \cdot 4 + 3)$ to BYTE $(N \cdot 4)$

This read-only register holds the data bytes of the received message. The selected byte range can be changed by writing another value to the register RX_BYTES - N.

1.4 Example

This section shows an example how to start the controller as a cold start node.

The controller is switched on (or restarted), it goes through default config to config state. The user must set the configuration of all registers mentioned in table 1.2.

As a next step, the TX registers must be set. At least one buffer has to be configured as a sync startup frame (as the controller is a cold start node). The following code shows this procedure:

```
//select the TX buffer 0
WRITE_VALUE(FLEXRAY_BASE, TX_BUF, 0);
//SET the HEADER_1
WRITE_VALUE(FLEXRAY_BASE, TX_HEADER1,
        (0 << 28)  // reserved bit = 0
        | (0 << 27)  // no preamble indic
```

```
        | (1 << 26)  // null frame
        | (1 << 25)  // synch frame
        | (1 << 24)  // startup frame
        | (1 << 13)  // ID
        | (0x10 << 6)); // payload length
// compute the header CRC and insert here to HEADER_2
// in this case - 0xF2
WRITE_VALUE(FLEXRAY_BASE, TX_HEADER2, 0xF2);
// it is a periodic message - send every cycle
WRITE_VALUE(FLEXRAY_BASE, TX_MSG_TYPE, (1 << 3));
// send message to channel A and B
WRITE_VALUE(FLEXRAY_BASE, TX_OPTION, 3);
// fill data
for (i = 0; i < 0x10 / 4; i++)
{ // divided by 4 because - 1 write cycle = writing 4 bytes
 //tx byte selection N
 WRITE_VALUE(FLEXRAY_BASE, TX_BYTES_N, i);
 //tx byte N*4+3 to N * 4, fill payload
 WRITE_VALUE(FLEXRAY_BASE, TX_BYTES_SELECTED,
                byte(N*4+3) << 24
                | byte(N*4+2) << 16
                | byte(N*4+1) << 8
                | byte(N*4+0));
}
// trigger options:
WRITE_VALUE(FLEXRAY_BASE, TX_TRG,
        (1 << 4) //trigger every cycle (send message every cycle)
        |(1 << 31)); //and set that this TX register is used
```

The code above sets One TX register is set and used for data transmitting. This frame is send every cycle with an ID equal to 1. The controller can start the communication as a cold star node:

```
// device is using channel A and B
WRITE_VALUE(FLEXRAY_BASE, CORE_STARTUP_SETUP, (3 << 8));
// command for a controller to go from config state to ready
```

```
WRITE_VALUE(FLEXRAY_BASE, STARTUP_COMMANDS, 0x1);
// go from ready to initialize startup
WRITE_VALUE(FLEXRAY_BASE, STARTUP_COMMANDS, 0x1 << 2);
```

After a couple of cycles the controller might go to NORMAL_ACTIVE state. The state can be obtained from POCState and StartupState registers. The controller might go back to READY state in cases described in [1]. The μController should set the controller to try another attempt by writing a command shown on the last line in the code listing.

If an user wants to restart the controller, whatever state it is in, it can be done in the following way:

```
// go from any state to HALT state - halt request
WRITE_VALUE(FLEXRAY_BASE, STARTUP_COMMANDS, 0x1 << 5);
// go from halt to default config
WRITE_VALUE(FLEXRAY_BASE, STARTUP_COMMANDS, 0x1 << 4);
```

Note: any value is changed in the registers during the default config state. Therefore, it is not needed to execute the default configuration after this state.

This was a short example how to set the controller as a cold start node. The attached CD contains more examples where the controller starts as a leading cold startup node, non-leading cold startup node, a leading cold startup mode using TT-L data transmission, or a non-cold startup node (a node integrating to a running communication).

1.5 Controller architecture

The controller consists of 3 main parts as we can see in figure 1.4. They are Core, Rx and Tx registers.

1.5.1 Core

The core is the most important part of the whole system. It consists of 11 subcomponents. Here is a very brief description:

uTickGenerator is a component provide the micro tick clock and sampling frequency of the bus

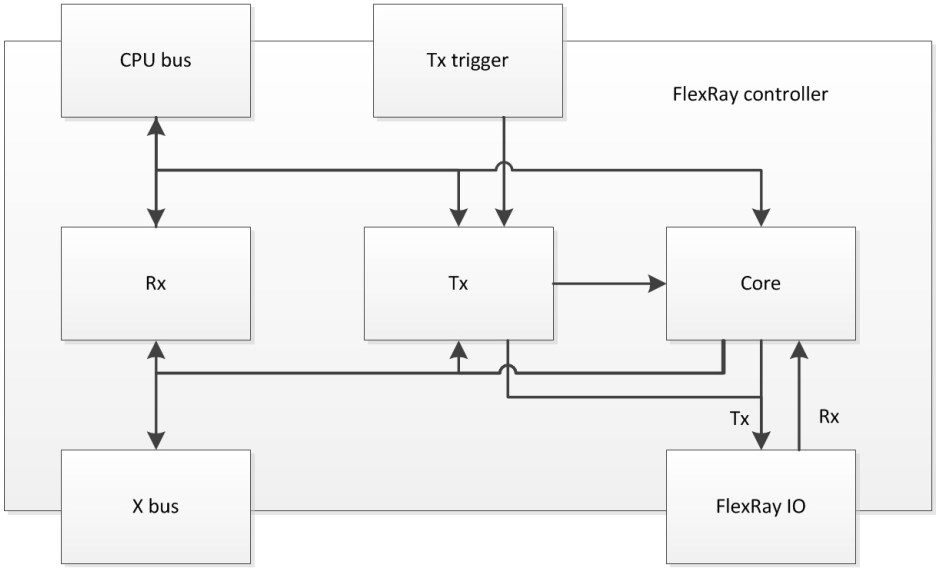

Figure 1.4: Controller architecture

majority_sampler is a component which samples signal from the bus and provides a filtered (voted) output as it is described in section Sampling and majority voting in [1]

receiver receives filtered signal and strobes the signal and provides a strobed value and last byte

symbol_encoding component takes bit strobed value from the receiver and decodes symbols such as TSS, CAS, WUS

receive_state_machine component receives bytes from receiver component and it decodes the frame. It provides a header, payload, and trailer. It also checks the CRC.

integrationStartUp is a component which is responsible for starting of a macro generator. In case that the unit is in initialize schedule, this component starts the macro tick generator.

macro_generator is a component which provides the macro tick clock and it handles also the clock distribution. It can be started either "directly", starting from zero by the ProtocolOperationControl in case that the controller is a leading cold start node, or it can integrate to a running cluster by signals provided by integrationStartUp component.

frameExpectation provides the action points (a starting point when a message can start) and the delay of the last received message

csp - clock synchronization process is a component responsible for synchronization. It needs the delay of frames from frameExpectation and it computes new values the macro_generator should apply

configuration_registers hold all settings of the core component, it is possible to communicate with this component via a parallel interface.

ProtocolOperationControl is a state machine representing and executing the state state machine (including all main state such as READY or HOLD and it also handles the process of STARTUP) of the whole system

Some of these components are twice in the design, because it is required them for channel A and B.

1.5.2 Receiver

This component receives all frames from the X-bus and stores them to RAM memory (if it is configured to store this information). It also contains registers which are directly accessible by the μController.

1.5.3 Transmitter

Transmitter consists of three main components: send_msg, CRC24_DATA8 and the main tx_reg itself.

send_msg is a component which sends a symbol or byte. It signals whether it is empty to execute a next command or it is busy.

CRC24_DATA8 is a component which computes a 24 bit CRC

tx_reg glues all components together. It includes two send_msg and two CRC24_DATA8 components (each for a separate channel, CRC must be computed also twice, because each channel contains a different CRC). This component also includes the registers to which or from which it is possible by an interface.

total logic elements	total registers	total memory bits	max frequency [MHz]
9886 (53%)	4897	24576 (10%)	82.42

Table 1.12: Implementation in the FPGA

1.6 Conclusion

The FlexRay controller has been tested on the Cyclone II EP2C20F484C7N with four RX and TX registers. The compilation was made with the respect to the highest speed in the Quartus II version 9.1. The result can be found in table 1.12. The maximal frequency is 82.42 MHz, which is sufficient to support the maximum FlexRay bit rate of 10 Mbps.

The controller is able to integrate to a running cluster as well as initialize the communication as a leading coldstart node. It contains configurable number of RX and TX registers, it supports many options according to the standard and it even contains options which are not allowed in the FlexRay standard. These options allow to test a wide range of parameters of the FlexRay node or cluster.

Bibliography

[1] FlexRay Consortium, *FlexRay Communications System Protocol Specification, Version 2.1, Revision A*, 2005.

Appendix A

X-bus description

This section contain X-bus bits description in tables A.1, A.2, and A.3.

Bit	Description
0	Tx - Channel A (non filtered)
1	Tx - Channel B (non filtered)
2	Tx - Channel A (filtered - majority voting)
3	Tx - Channel B (filtered - majority voting)
4	ChannelSampleClock - Sampling clock
5	uTick - utick clock
6	TSS symbol decoded on A
7	TSS symbol decoded on B
8	CAS symbol decoded on A
9	CAS symbol decoded on B
10	bit strobed on A
11	bit strobed on B
12	byte received on A
13	byte received on B
14	secondary TRP signal received on A
15	secondary TRP signal received on B
16	wake up symbol decoded on A
17	wake up symbol decoded on B
18	header received on A
19	header received on B
20	received frame is in odd cycle - A
21	received frame is in odd cycle - B
22	received frame's ID on A is odd
23	received frame's ID on B is odd
24	received frame on A is a startup frame
25	received frame on B is a startup frame
26	received frame on A is a sync. frame
27	received frame on B is a sync. frame
28	first header received on A by the component integration startup during the startup phase
29	first header received on B by the component integration startup during the startup phase
30	second header received on A by the component integration startup during the startup phase
31	second header received on B by the component integration startup during the startup phase
32	device started integration on A
33	device started integration on B
34	macro tick generator is running
35	impulse at the beginning of a new cycle when a macro generator is running
41:36	cycle counter

Table A.1: X bus - description part 1

Bit	Description
61:42	micro tick counter
75:62	macro tick counter
76	computing of offset in the csp component is finished
77	action point on A
78	action point on B
79	static slot finished on A
80	static slot finished on B
86:81	cycle of the received on A
92:87	cycle of the received on B
103:93	frame ID received on A
114:104	frame ID received on B
115	null frame received on A
116	null frame received on B
117	payload preambule received on A
118	payload preambule received on B
119	reserved bit received on A
120	reserved bit received on B
127:121	payload length A
134:128	payload length B
142:135	byte received - data A
150:143	byte received - data B
151	trailer received on A
152	trailer received on B
153	bus idle on A
154	bus idle on A
170:155	message delay (message received later/earlier to action point) on channel A - 16 bit signed nuber
186:171	message delay (message received later/earlier to action point) on channel B - 16 bit signed nuber
187	header received on A with a wrong CRC
188	header received on A with a wrong CRC
189	send CAS request
190	send WUS on channel A request
191	send WUS on channel B request
192	send startup frames only
193	wake up collision detected
194	system should remain in halt state
195	system should remain in passive state
196	macro generator can be running
197	start of the macro generator requested

Table A.2: X bus - description part 2

198	POCState is in CONFIG state
199	POCState is in DEFAULT_CONFIG state
200	POCState is in HALT state
201	POCState is in NORMAL_ACTIVE state
202	POCState is in NORMAL_PASSIVE state
203	POCState is in READY state
204	POCState is in STARTUP state
205	POCState is in WAKEUP state
206	StartupState is in UNDEFINED state
207	StartupState is in COLDSTART_LISTEN state
208	StartupState is in INTEGRATION_COLDSTART_CHECK state
209	StartupState is in COLDSTART_JOIN state
210	StartupState is in COLDSTART_COLLISION_RESOLUTION state
211	StartupState is in COLDSTART_CONSISTENCY_CHECK state
212	StartupState is in INTEGRATION_LISTEN state
213	StartupState is in INITIALIZE_SCHEDULE state
214	StartupState is in INTEGRATION_CONSISTENCY_CHECK state
215	StartupState is in COLDSTART_GAP state
216	WakeupStatus is in UNDEFINED state
217	WakeupStatus is in RECEIVED_HEADER state
218	WakeupStatus is in RECEIVED_WUP state
219	WakeupStatus is in COLLISION_HEADER state
220	WakeupStatus is in COLLISION_WUP state
221	WakeupStatus is in COLLISION_UNKNOWN state
222	WakeupStatus is in TRANSMITTED state
223	start macro generator request
222	zSyncCalcResult is WITHIN_BOUNDS
228:225	number of zStartUpNodes
222	start macro generator request

Table A.3: X bus - description part 3