CAN FD IP function

Datasheet

Ille Ondrej

December 19, 2015

Contents

1. Introduction **4**

1.1 Functionality requirements . . . 4

1.2 Commercial research . . . 4

1.3 Development tools . . . 5

2. System architecture **6**

2.1 Block diagram . . . 6

2.2 Signals architecture . . . 6

2.2.1 Driving bus . . . 6

2.2.2 Status bus . . . 6

2.3 Components description . . . 7

2.3.1 CAN Core . . . 7

2.3.1.1 Protocol control . . . 7

2.3.1.2 Operation control . . . 7

2.3.1.3 Fault confinement . . . 8

2.3.1.4 Bit stuffing . . . 9

2.3.1.5 Bit destuffing . . . 9

2.3.1.6 CRC . . . 9

2.3.1.7 Transceive buffer . . . 9

2.3.2 CAN bus sampling . . . 9

2.3.3 Prescaler . . . 10

2.3.4 Message filtering . . . 10

2.3.5 Receive buffer (RX) . . . 10

2.3.6 Transmit buffer - Time (TXT) . . . 11

2.3.7 TX arbitrator . . . 11

2.3.8 Interrupt manager . . . 11

2.3.9 Memory registers . . . 11

2.3.10 Event logger (optional) . . . 11

2.3.11 CAN top level . . . 12

2.4 Clock domains, external signals synchronization and reset . . . 13

2.5 Minimal timing . . . 13

3. Register map **14**
3.1 DEVICE_ID . . . 14
3.2 MODE_REG . . . 14
3.2.1 MODE . . . 14
3.2.2 COMMAND . . . 16
3.2.3 STATUS . . . 16
3.2.4 SETTINGS . . . 17
3.3 INTERRUPT_REG . . . 17
3.3.1 INT . . . 17
3.3.2 INT_ENA . . . 18
3.4 TIMING_REG . . . 18
3.4.1 BTR . . . 18
3.4.2 BTR_FD . . . 19
3.5 ALC_PRESC . . . 19
3.5.1 ALC . . . 19
3.5.2 BRP . . . 19
3.5.3 BRP_FD . . . 20
3.6 ERROR_TH . . . 20
3.6.1 EWL . . . 20
3.6.2 ERP . . . 20
3.6.3 FAULT STATE . . . 20
3.7 ERROR_COUNTERS . . . 21
3.7.1 RXC/CTR_PRES . . . 21
3.7.2 TXC . . . 21
3.8 ERROR_COUNTERS_SPECIAL . . . 22
3.8.1 ERR_NORM/CTR_PRES_S . . . 22
3.8.2 ERR_FD . . . 22
3.9 FILTER_X_MASK . . . 22
3.10 FILTER_X_VALUE . . . 23
3.11 FILTER_RAN_LOW . . . 23
3.12 FILTER_RAN_HIGH . . . 23
3.13 FILTER_CONTROL . . . 24
3.14 RX_INFO_1 . . . 24
3.14.1 RX_STATUS . . . 24
3.14.2 RX_MC . . . 25
3.14.3 RX_MF . . . 25
3.15 RX_INFO_2 . . . 25
3.15.1 RX_BUFF_SIZE . . . 25
3.15.2 RX_WPP . . . 25

3.15.3 RX_RPP . . . 26
3.16 RX_DATA . . . 26
3.17 TRV_DELAY . . . 26
3.18 TX_STATUS . . . 26
3.19 TX_SETTINGS . . . 27
3.20 TX_DATA_X . . . 27
3.21 RX_COUNTER . . . 28
3.22 TX_COUNTER . . . 28
3.23 LOG_TRIG_CONFIG . . . 28
3.24 LOG_CAPT_CONFIG . . . 30
3.25 LOG_STATUS . . . 31
3.25.1 LOG_STAT . . . 31
3.25.2 LOG_WPP . . . 31
3.25.3 LOG_RPP . . . 31
3.26 LOG_COMMAND . . . 32
3.27 LOG_CAPT_EVENT_1 . . . 32
3.28 LOG_CAPT_EVENT_2 . . . 32
3.28.1 EVENT_TS(15:0) . . . 32
3.28.2 EVENT_INFO . . . 33
3.29 YOLO_REG . . . 34

4. RTL Simulation **35**
4.1 Basic test-benches . . . 35
4.2 Constraint random test-benches . . . 36
4.2.1 core_top_tb1.vhd . . . 36
4.2.2 top_level_2_tb.vhd . . . 36

5. Synthesis and testing **38**
5.1 Design size . . . 38
5.2 Timing analysis . . . 39
5.3 FPGA Tests . . . 39

6. Future work and improvements **40**
6.1 Missing RTL tests . . . 40
6.1 Missing hardware tests . . . 40

Appendix A - Driving bus signals **42**

Appendix B - Status bus signals **45**

1. Introduction

Testing of automotive buses is challenging task these days. Due to this reason device described in [5] is being developed. Since CAN 2.0 specification was recently extended with Flexible Data-rate, CAN bus now offers more bandwidth than ever before. In order to integrate this functionality into device in [5] Flexible Data rate functionality needs to be covered. Predecessor of this component is CAN IP function developed in [6]. Due to absence of proper RTL tests in [6] new implementation is considered (instead of extending the previous one).

1.1 Functionality requirements

In order to perform tests on CAN bus, several functions of the CAN controller are required. These requirements CAN be implemented in hardware (controller itself) or software (firmware or driver). Software implementation is usually more complicated than hardware one (due to strict timing requirements). Because of this reason all test requirements are implemented in hardware as part of CAN FD IP core in VHDL language. These requirements are:

1. Handle CAN FD protocol according to [1].
2. Provide "Logical Link Control" extension of MAC layer. It enables to send and receive frames by manipulation with buffers (instead of direct manipulation with state machine). This approach simplifies firmware implementation.
3. Be able to capture external time stamp when received frame is stored in buffer, as well as start transmitting frame when specific value of external time stamp is reached.
4. Record various events on the bus with time stamp.
5. When frame is inserted with specified transmittion time, be able to determine when it was actually sent
6. Be able, to trigger event recording (point 4) by specific event on the bus.
7. Be able to manipulate Fault confinement state and counters.

1.2 Commercial research

Short commercial research was made before implementation to avoid overcharging of implementation. Following product was found:

- CAN FD IP Core by CAST inc. , unknown prize for RTL source code.
- IFI CAN FD IP Core Ingenieurbüro Für Ic-Technologie , prize: 12375 euro, encrypted net-list only

Both products are overly expensive (for purpose of [5]) and don't fulfill requirements 4-7 in 1.1. On the other hand both IP cores are rigorously verified and ready for ASIC manufacture. Since the main application in [5] is FPGA based, this advantage is not important. Due to these reasons own implementation is considered.

1.3 Development tools

To develop this IP function following software or hardware tools were used:

- ModelSim ALTERA Edition 6.5b for VHDL implementation and RTL test-benches.
- Quartus II 9.1sp Web Edition for synthesis (target device from ALTERA EP4CE55F23C8N) and Timing analysis.
- Code composer studio v 5.0 for accessing IP function as periphery of Texas Instruments processor from C code.
- HALCoGen v04.04 for processor configuration (refer to [5] where whole test system is described).
- CANoe program (with CAN/USB converter) for post-synthesis verification by communication with reference controller

2. System architecture

The implementation is divided into several modules, every module covering just its main function, thus keeping modularity design rule. Whole system is implemented as synchronous design with asynchronous reset (performed by input pin of FPGA). IP core acts as memory mapped periphery. In order to be compatible with Flexray IP core implemented in [18] Avalon bus compatible interface is selected. This interface is in detail described in [17]. Simplified description is provided in description of "Memory registers" block.

2.1 Block diagram

Block scheme is in the Figure 1. Every block is separate VHDL source code file or (in case of Controller core) is assembled from several sub-blocks.

2.2 Signals architecture

In order to simplify VHDL interconnection in top level entity, two paralel "buses" were created: Driving bus (not on picture), Status bus. These buses have form of std_logic vector and one source block and many destination blocks.

2.2.1 Driving bus

The Driving bus is used to control functionality of all blocks from user registers. Exact definition of driving bus enables separation of memory registers from other blocks. This enables changing the registers structure without modification of internal signaling. List of signals in driving bus is in Appendix A. Driving bus has following form:

```
signal drv_bus: std_logic_vector(1023 downto 0)
```

2.2.2 Status bus

Status bus is used to provide information about state and state registers of CAN Core for other blocks. It's source is in CAN Core. Status bus signals are used by Memory registers and Event logger component. List of signals in status bus is in Appendix B. Status bus has following form:

```
signal stat_bus: std_logic_vector(511 downto 0)
```

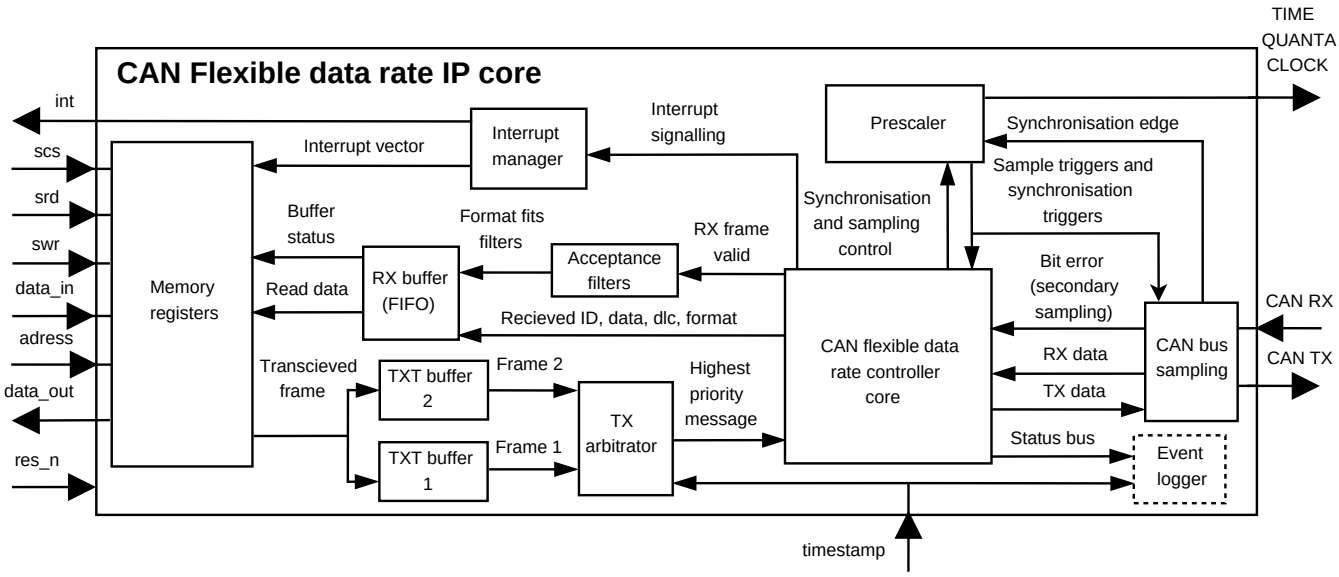

Figure 1: IP function block diagram

2.3 Components description

2.3.1 CAN Core

File: core_top.vhd

CAN Core covers the functionality of serial data transmission according to CAN FD standard. Storing frame from input, storing received frame, transmission, reception, arbitration, bit stuffing, bit destuffing, CRC calculation, error handling and fault confinement are implemented within this unit. Block diagram (just most important signals are present) is on picture 2 ref. Furthermore valid CRC selection, transmit trigger and receive trigger creation, status bus assignment and bus traffic measurement is implemented within this module.

2.3.1.1 Protocol control

File: protocolContorol.vhd

Protocol control is main state machine handling CAN FD protocol. Some states has sub-states which represent different situations within one state (e.g. protocol control state : control , sub state: two bits, base_id). Protocol control processes received data with rec_trig signal which is signal two clock cycles delayed from sample signal. Data are transmitted with tran_trig signal (in synchronization segment of bit time). Circuit operation is started from Intermission state. Reception of message is started when hard_sync_valid input is in logic 1. Transmission is started when intermission lasted for at least 3 bit times and data are available for transmission. When message is available and hard_sync_valid is in logic 1 then transmission also starts. In this case the arbitration mechanism is applied. State transition diagram is not provided because of too many transitions! For more details view the source code file where every state is separated by "case" statement of signal "PC_State"

2.3.1.2 Operation control

File: operationContorol.vhd

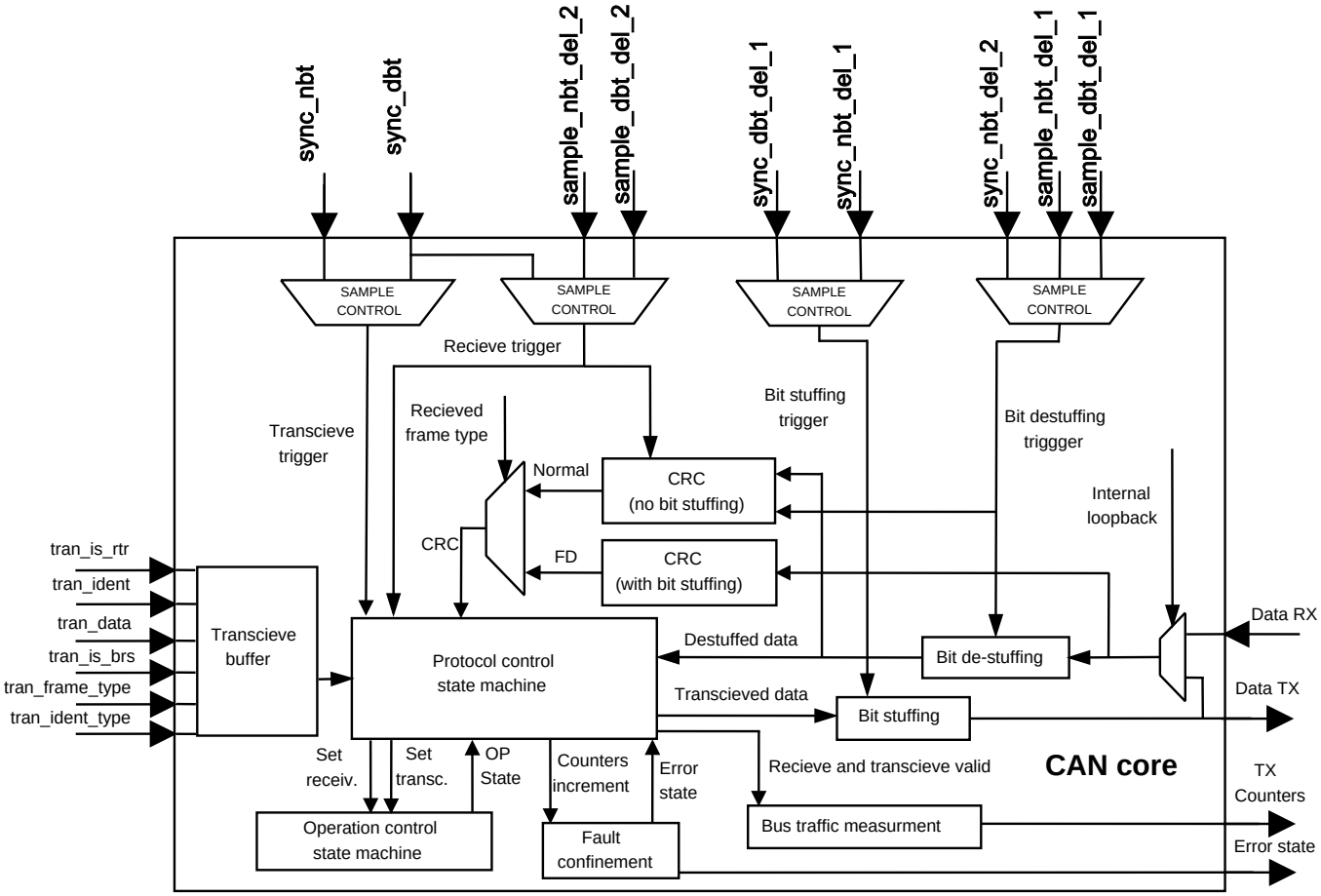

Figure 2: CAN Core block diagram

Operation control is state machine handling "Operation mode" as defined in [1]. This covers unit integration after start, transmission mode (transmitter), reception mode (receiver) and bus idle. Other modes defined in [1] (bus monitoring mode, self test mode) are implemented via dedicated signals of driving bus (drv_bus_mon_ena, drv_self_test_ena signals).

2.3.1.3 Fault confinement

File: faultConf.vhd

Fault confinement module implements Fault confinement state (Error active, Error passive or Bus off), error counters, bit error and stuff error detection. Error counters for Fault confinement (rx_counter, tx_counter) are incremented via inc_one, inc_eight, dec_one signals from Protocol control, therefore all exceptions in [1] (Fault confinement chapter) are managed by Protocol control. These error counter are implemented to be read/write (via driving bus), so that Fault confinement state can be manipulated by user. This provides extended testing functionality of CAN controller. Furthermore two additional counters (err_counter_norm, err_counter_fd), are implemented to distinguish between errors which appeared in Nominal bit rate and Data bit rate.

2.3.1.4 Bit stuffing

File: bitStuffing_v2.vhd

Bit stuffing module implements functionality of Bit stuffing into serial data stream. Number of equal consecutive bits is variable (stuff_length input) as well as fixed bit stuffing method for CRC field of CAN FD frames (fixed_stuff input). Circuit processes input data with triggering signal one clock cycle delayed from transmit trigger (tran_trig_1 input). Stuff bit insertion is signaled to Protocol Control to stop the data transmission via data_halt signal.

2.3.1.5 Bit destuffing

File: bitDestuffing.vhd

Bit destuffing module implements reverse functionality to Bit stuffing. It discards stuff bits from serial data stream with trigger one clock cycle delayed from sample signal (trig_spl_1 signal). Additionally stuff error is detected when stuff_length+1 same consecutive bits are detected.

2.3.1.6 CRC

File: CRC.vhd

CRC module implements cyclic redundancy check calculation according to [1]. Circuit operation is started with rising edge on enable input (circuit is still synchronous to clk_sys. Rising edge on "enable" input means detection of 0 to 1 transition) . Input data are processed with trig signal. After finishing the calculation CRC value remains valid until next rising edge on enable input. The result value is propagated to outputs (crc15,crc17,crc21 signals) of the circuit. All three CRC values are calculated at the same time. CRC calculation is implemented via shift register in order to avoid long combination paths if only combinational implementation would be used.

2.3.1.7 Transceive buffer

File: tranBuffer.vhd

Auxiliary component for storing the frame to be transmitted. Circuit stores input frame when logic 1 is detected on frame_store input.

2.3.2 CAN bus sampling

File: busSync.vhd - file name kept although module functionality changed

Circuit samples the CAN bus values. Optional (recommended) synchronization chain with two flip flops is used to avoid metastability since CAN_RX signal is input from CAN physical layer transceiver (asynchronous signal from outside of FPGA) . Bus is sampled with sample signal from Prescaler. Sync_edge output signals that valid (recessive to dominant) edge has appeared on CAN_RX input. Furthermore transceiver delay measurement and secondary sampling point generation is implemented here via two shift registers and counter. One is dedicated to storing transmitted data in order to compare it with delayed received data (bit error detection for transmitter in Data bit time). Second one stores sample points generated by Prescaler thus providing secondary sampling point.

2.3.3 Prescaler

File: prescaler_v3.vhd

Prescaler implements the functionality of bus timing. It counts Time quanta and Bit time clock. It contains simple state machine for phase of bit time (Synchronisation, Propagation, Phase 1, Phase 2). It generates synchronization signals (and appropriate delayed signals) and sample signals (and appropriate delayed signals). Furthermore it covers the functionality of Hard synchronization and Resynchronization (sync_control signal). Based on sample_control signal Nominal bit time or Data bit time is used.

2.3.4 Message filtering

File: messageFilter.vhd

Message filter validates wheter input frame identifier matches three (A,B,C) mask filters or one range filter. Output is valid if identifier matches at least on filter. Furthermore frame type and identifier type can be selected for each filter. Note that range filter input (from registers) does not have the same format as transmitted and received identifier!

2.3.5 Receive buffer (RX)

File: rxBuffer.vhd

Receive buffer implements FIFO memory for storing received CAN frames. Basic unit of buffer is 32 bit wide word. Size of the buffer is configurable (generic buff_size) before synthesis. However because of addressing logic, only powers of 2 can be used as buffer size. One frame in received buffer contains from 5 to 20 words. Therefore if size of less than 32 words is used, long CAN FD frames won't be stored and data overrun will appear even if the buffer is empty. Frame format is the same as transmitted frame format. First word is according to TX_DATA_1, second to TX_DATA_2 ... This structure is in detail shown in description of register TX_DATA_X.

When rec_message_valid input signal is in logic one first word is stored. In following up to 19 clock cycles remaining words are stored. This requires the received data to be valid for at least 20 clock cycles (register in CAN Core). Since frame is validated at the end of EOF field, until received data are erased by next message bus is in the intermission field. Having minimum 10 clock cycles per nominal bit time this gives minimum 30 clock cycles (3 bit times is minimal length of intermission field) when received data, frame type, frame format, identifier are stable. Always one word (given by read_pointer signal) is on output of the circuit to be read. Read_pointer is incremented by one via rising edge on drv_read_start signal (part of driving bus). Memory registers block drives this signal to automatically increment the read_pointer by one when data from RX buffer are read.

Receive buffer also implements Data overrun flag functionality. When rec_message_valid signal is in logic one, first frame size (based on rec_dlc) is calculated and compared with free memory in the buffer. If there is not enough memory, frame is not stored and Data overrun flag is set. Apart from read_pointer, write_pointer and free_memory also amount of CAN frames stored in the buffer is available.

Receive buffer is implemented as array of std_logic_vector (see signal "memory" in source code). Due to complicated storing logic and universality it is intended to be synthesized as flip-flops. This significantly increases size of the design when large buffer sizes are used. Usage of specific RAM elements (available in FPGAs) is possible, however this would require modification of source code. Entity which exactly defines the memory to be inferred must be inserted and signals as "write_pointer", "read_pointer", "memory" in the design must be connected to it's data and address inputs. Change to inferred RAM block can significantly reduce size of the design, however universality of the source code is lost. More about implementing inferred RAMs in ALTERA devices in .

It is assumed that this change will be realized in the next version of IP function. Several architectures of receive buffer are about to be implemented for different available RAM memories.

2.3.6 Transmit buffer - Time (TXT)

File: txtBuffer.vhd

Transmit buffer is memory which contains one CAN frame to be transmitted. It is accessed via committing content of TX_DATA_1 to TX_DATA_20 registers into either TXT buffer 1 or TXT buffer 2. If the buffer is full an data are committed new data are not stored into buffer. To avoid this situation buffer status can be read. In order to save LUTs on FPGA it is possible to synthetise buffer which only supports frame length up to 8 bytes (via generic useFDsize). TX_DATA_7 to TX_DATA_20 registers have no meaning then. When this "reduced" buffer is synthesized it is still possible to insert message dlc with length up to 64 bytes. However data 9 to 64 will be lost and only zeroes will be transmitted.

2.3.7 TX arbitrator

File: txArbitrator.vhd

TX Arbitrator circuit manages the functionality of sending the CAN (CAN FD) frames in time specified in TX_DATA_2 and TX_DATA_3 when committed into TXT buffer. If external time stamp value is lower than time stamp specified for frame. Frame is propagated as valid to CAN Core. If timestamps in both TXT buffers are lower than external time stamp then the one with lower time stamp is propagated to CAN Core for transmitting. When both timestamps are the same and lower than external time stamp message with lower identifier (ID_BASE&ID_EXT) is selected for transmitting. Additionally circuit enables to forbid transmitting from each of the TXT Buffers.

2.3.8 Interrupt manager

File: intManager.vhd

Interrupt manager provides interrupt via int output of the CAN_top_level entity. Interrupt sources are configurable from driving registers and every interrupt is marked into interrupt vector (register INT).

2.3.9 Memory registers

File: registers.vhd

Memory registers provide interface between Avalon compatible 32 bit bus and driving bus used for controlling whole CAN FD IP function. Driving bus assignment is implemented in this module. Register structure is in detail described in Chapter 3.

2.3.10 Event logger (optional)

File: logger.vhd

Event logger is module allowing to capture events on the can bus with its timestamps. FIFO memory is implemented with configurable size (generic "memory_size"). Event logger implements state machine with CONFIG state which is dedicated to reading out previously logged data, configuring triggering and capturing event types. When command is given (via driving bus,see register LOG_COMMAND description). state machine is moved to READY state where it is waiting for triggering condition to move to RUNNING state. In RUNNING state, events are being captured along with its time stamp and additional information (see register EVENT_TYPE description). When the memory is full circuit automatically moves to CONFIG state. This circuit provides additional testing capability beyond the CAN FD specification. Events can be read out from memory from EVENT_INFO_1 and EVENT_INFO_2 register. Always event at the position of read_pointer is read out. Read_pointer position can be manipulated via LOG_CMD register.

FIFO Memory is implemented in the same way as RX Buffer memory, with Flip-Flops. For inferring RAM memory available in FPGAs see [20].

Note that several event types are possible to be captured during one run of Event logger. However event logger is able to capture only one event within one clock cycle! Therefore if two events should be captured and they occur at the same clk_sys cycle only the one with higher priority is captured (lower the position of the event in the LOG_CAPT_CONFIG higher the priority of event). Therefore it might happen that some event will be lost! In order to avoid this do not combine following event types together or with other event types when capturing:

- Synchronization edge appeared
- Stuff bit was inserted
- Bit was destuffed
- Data overrun appeared
- Error appeared
- Fault Confinement state changed
- Error warning limit reached

2.3.11 CAN top level

File: CAN_top_level.vhd

Entity covering the functionality of whole IP function. All configurable features of the CAN IP function (set as generic constants of this entity) are listed in table 1.

Name	Type	Default value	Meaning
use_logger	boolean	true	If Event logger should be synthesized as component
rx_buffer_size	natural	128	Size of recieve buffer in 32 bit words
useFDSize	boolean	true	Size of transmitt buffer should be for FD frames
use_sync	boolean	true	Use synchronisation chain for recieved data
ID	natural	1	ID of the controller. Adress(19:16) to which the IP core is mapped
logger_size	natural	8	Size of event logger memory in number of events.

Table 1: CAN FD IP core configuration options

2.4 Clock domains, external signals synchronization and reset

Whole design is synchronous to one clock signal, clk_sys. Every other period of time is relative to clk_sys (Time quantum, Bit time...). Every register has asynchronous reset, res_n, which is active low, by default. The design is intended to be latch-free.

Input signals of Avalon memory interface and time stamp value are expected to be synchronous to clk_sys and no signal synchronization is implemented on these signals. CAN_RX signal is synchronized by simple synchronization chain with two flip-flops. This synchronization chain is optional, but it is recommended to use it unless synchronization chain is automatically inserted by synthesizer.

2.5 Minimal timing

CAN standard defines so called information processing time which is different for every implementation of CAN controller. This time determines how many clock cycles controller needs after sampling bus value before determining following bit value. Implementation of this IP function works with so called "triggering signals". For sampling, three triggering signals are used. These signals are mutually delayed by one clock cycle (one for sampling, two for bit destuffing, three for processing by CAN Core). This gives minimal duration of PH2 to 4 clock cycles (one more to actualize state variables of Fault Confinement or Protocol Control) . When Prescaler value is set to 1 (time quanta equal to clk_sys), minimal timing (e.g. Data phase of FD) is following:

- SYNC = 1 (this segment is always equal to 1)
- PROP = 1
- PH1 = 1
- PH2 =4

Note that when higher value of prescaler is used PROP or PH1 segments can be 0. However still at least three clock cycles must be the duration of SYNC+PROP+PH1. Considering that the maximal frequency of circuit is 50 MHz (see Timing analysis) this gives us maximal Bit Rate 7,14 MBit/s. If higher Bit Rates are needed, other FPGA with different technology than EP4CE55F23C8N must be used.

3. Register map

Register structure is selected to be formally compatible with SJA1000 controller [8]. Since SJA1000 has 8 bit address, in order to fit into 32 bit address several 8 bit registers were concatenated (to save address space). If register is read or written all 32 bits are read or written at once. Therefore by one read/write also other registers can be affected. Therefore it is necessary to read the register and then write back just partially modified value! Reading or writing just 8 or 16 bits is not possible. In some situations registers which are read-only or write-only are concatenated together. Read of write-only register has no effect and always return 0. Write into read-only register also doesn't have any effect. Some registers (e.g. INT in INTERRUPT REG) are erased when read.

3.1 DEVICE_ID

Type: Read

Register contains identifier of IP function. It is used for determining whenever CAN IP function is mapped correctly on its base address.

Bit offset	31-0
Name	DEVICE_ID
Default value	0x0000CAFD

3.2 MODE_REG

3.2.1 MODE

Type: Read / Write

MODE register (part of MODE_REG) sets special operating modes of the controller. All bits are active in logic 1.

Bit offset	7	6	5	4	3	2	1	0
Name	ACF	TSM	RTRP	FDE	AFM	STM	LOM	RST
Default value	0	0	1	1	0	0	0	0

RST Activating this bit resets the controller. It has the same effect as logic 0 on res_n input of controller

LOM Listen only mode. In this mode controller only receives data and sends only recessive bits on the bus. When dominant bus is sent it is rerouted internally so that bus value remains the same.

STM Self test mode. In this mode transmitted frame is considered valid even when acknowledge is not received.

Address offset	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0	Register name
0x0	DEVICE_ID				DEVICE_ID
0x4	SETTINGS	STATUS	COMMAND	MODE	MODE_REG
0x8	INT_ENA		INT		INTERRUPT_REG
0xC	BTR_FD		BTR		TIMING_REG
0x10	BRP_FD	BRP	Reserved	ALC	ALC_PRESC
0x14	FAULT_STATE		ERP	EWL	ERROR_TH
0x18	TXC		RXC/CTR_PRES		ERR_COUNTERS
0x1C	ERR_FD		ERR_NORM/CTR_PRES_S		ERR_COUNTERS_SPECIAL
0x20	FILTER_A _MASK				FILTER_A_MASK
0x24	FILTER_A _VALUE				FILTER_A_VAL
0x28	FILTER_B _MASK				FILTER_B_MASK
0x2C	FILTER_B _VALUE				FILTER_B_VAL
0x30	FILTER_C _MASK				FILTER_C_MASK
0x34	FILTER_C _VALUE				FILTER_C_VAL
0x38	FILTER_RAN _LOW				FILTER_RAN_LOW
0x3C	FILTER_RAN _HIGH				FILTER_RAN_HIGH
0x40	Reserved		FILTER_CONTROL		FILTER_CONTROL
0x44	Reserved	RX_MF	RX_MC	RX_STATUS	RX_INFO_1
0x48	Reserved	RX_RPP	RX_WPP	RX_BUFF _SIZE	RX_INFO_2
0x4C	RX_DATA				RX_DATA
0x50	Reserved		TRV_DELAY		TRV_DELAY
0x54	Reserved			TX_STAT	TX_STATUS
0x58	Reserved			TX_SET	TX_SETTINGS
0x5C	TX_DATA_1				TX_DATA_1
0x60	TX_DATA_2				TX_DATA_2
...	...				
0xA8	TX_DATA_20				TX_DATA_20
0xAC	RX_COUNTER				RX_COUNTER
0xB0	TX_COUNTER				TX_COUNTER
0xB4	Reserved				
0xB8	LOG_TRIG_CONFIG				LOG_TRIG_CONFIG
0xBC	Reserved				
0xC0	LOG_CAPT_CONFIG				LOG_CAPT_CONFIG
0xC4	LOG_RPP	LOG_WPP	LOG_STAT		LOG_STATUS
0xC8	Reserved			LOG_CMD	LOG_COMMAND
0xCC	EVENT_TIME_STAMP(47:16)				LOG_CAPT_EVENT_1
0xD0	EVENT_TIME_STAMP(15:0)		EVENT_INFO		LOG_CAPT_EVENT_2
0xD4	Reserved				
0xD8	YOLO_REGISTER				

Table 2: Register map

AFM Acceptance filters mode. Activating this bit enables usage of acceptance filters.

FDE Flexible data rate support enable. When this bit is inactive, receiving recessive EDL bit (Flexible data rate frame) causes form error.

RTRP RTR Frame preferred behavior. When RTR frame is sent also non-zero dlc code can be inserted. This bit specifies behavior of controller when this situation happens. If bit is active then all zeros are sent and written dlc is ignored. If bit is inactive then written dlc is sent.

TSM Triple sampling mode. Bus value is sampled three times when this bit is active. Usage is recommended only at very low Bit rates.

ACF When this bit is active acknowledge is not sent even when received CRC matches the calculated one.

3.2.2 COMMAND

Type: Write

Writing logic 1 gives command to the controller. Meaning of command is different for every bit. This register is automatically erased when command is finished.

Bit offset	15-12	11	10	9	8
Name	Reserved	CDO	RRB	AT	Reserved
Default value	-	0	0	0	-

AT Writing logic 1 into this bit aborts actual transmission or reception.

RRB Release receive buffer. Writing logic 1 deletes all data from receive buffer.

CDO Clear data overrun flag. Writing logic 1 will clear overrun flag.

3.2.3 STATUS

Type: Read

Register signals various states of CAN controller which are not mutually exclusive. Every bit is active in logic 1.

Bit offset	23	22	21	20	19	18	17	16
	BS	ES	TS	RS	Reserved	TBS	DOS	RBS
Default value	1	0	0	0	-	0	0	0

RBS Active value of this bit means Receive buffer is not empty.

DOS Data overrun status (flag). Active value of this bit signals message was lost due to not enough space in receive buffer.

TBS TX buffer status. Active value of this bit means both TXT buffers are full and message cant be inserted for sending

RS Active value of this bit signals that controller is receiving a message.

TS Active value of this bit signals that controller is transmitting a message.

ES Error status. Active value signals that error warning limit was reached at any of error counters

BS Bus status. Active value signals that bus is idle, controller is integrating or bus_off. Therefore this bit is active when there is no activity on the bus.

3.2.4 SETTINGS

Type: Read / Write

Register where functionality of controller is switched on. Also configuration of retransmittion limit for failed frames is possible here. Furthermore configuration of ISO FD CAN or CAN FD 1.0 is done here. Every bit is active in logic 1.

Bit offset	31	30	29	28-25	24
	FD_TYPE	ENA	INT_LOOP	RTR_TH	RTRLE
Default value	0	0	0	0	0

RTRLE Active value means that limit of retransmission is enabled.

RTR_TH Maximal number of retransmittion attempts.

INT_LOOP Active value in this bit means that internal loop-back option is permanently enabled (used only for testing).

ENA Whenever functionality of whole CAN FD peripheria is enabled

FD_TYPE Selection betwenn two possible CAN FD frame formats (0 ISO CAN FD, 1 CAN FD 1.0)

3.3 INTERRUPT_REG

3.3.1 INT

Type: Read - automatically erased after read

Register contains interrupt vector of interrupts that were thrown since last read.

Bit offset	15-11	10	9	8	7	6	5	4	3	2	1	0
Name	Reserved	BSI	RFI	LFI	BEI	ALI	EPI	Reserved	DOI	EI	TI	RI
Default value	-	0	0	0	0	0	0	-	0	0	0	0

BSI Bit rate shifted interrupt

RFI Receive buffer full interrupt

LFI Event logging finished interrupt

BEI Bus Error interrupt

ALI Arbitration lost interrupt

EPI Node became error passive or bus off interrupt

DOI Data Overrun interrupt

EI Error warning limit interrupt reached

TI Message was successfully transmitted interrupt

RI Message was successfully received interrupt

3.3.2 INT_ENA

Type: Read / Write

Register enables interrupts by different sources. Logic 1 in each bit means interrupt is allowed

Bit offset	31-27	26	25	24	23	22	21	20	19	18	17	16
Name	Reserved	BSIE	RFIE	LFIE	BEIE	ALIE	EPIE	Reserved	DOIE	EIE	TIE	RIE
Default value	-	0	0	0	0	0	1	-	1	1	0	0

BSIE Bit rate shifted interrupt enable

RFIE Receive buffer full interrupt enable

LFIE Event logging interrupt enable

BEIE Bus Error interrupt enable

ALIE Arbitration lost interrupt enable

EPIE Node became error passive or bus off interrupt enable

DOIE Data Overrun interrupt enable

EIE Error warning limit reached interrupt enable

TIE Message was successfully transmitted interrupt enable

RIE Message was successfully received interrupt enable

3.4 TIMING_REG

3.4.1 BTR

Type: Read / Write

Length of bit time segments for Nominal bit time in Time quanta. Note that SYNC segment always lasts one Time quanta.

Bit offset	15-11	10-6	5-0
Name	PH2	PH1	PROP
Default value	5	3	5

PROP Propagation segment

PH1 Phase 1 segment

PH2 Phase 2 segment

3.4.2 BTR_FD

Type: Read / Write

Length of bit time segments for Data bit time in Time quanta. Note that SYNC segment always lasts one Time quanta.

Bit offset	31	30-27	26	25-22	21-20	19-16
Name	Reserved	PH2_FD	Reserved	PH1_FD	Reserved	PROP_FD
Default value	-	3	-	3	-	3

PROP_FD Propagation segment

PH1_FD Phase 1 segment

PH2_FD Phase 2 segment

3.5 ALC_PRESC

3.5.1 ALC

Type: Read

Bit offset	7-5	4-0
Name	Reserved	ALC_VALUE
Default value	-	0

ALC_VALUE Arbitration lost capture value.

3.5.2 BRP

Type: Read / Write

Baud rate Prescaler register for Nominal bit time. Specifies time quanta duration and synchronization jump width

Bit offset	23-22	21-16
Name	SJW	BRP
Default value	2	10

BRP Baud rate Prescaler

SJW Synchronization jump width

3.5.3 BRP_FD

Type: Read / Write

Baud rate Prescaler register for Data bit time. Specifies time quanta duration and synchronization jump width.

Bit offset	31-30	29-24
Name	SJW_FD	BRP_FD
Default value	2	3

BRP_FD Baud rate Prescaler

SJW_FD Synchronization jump width

3.6 ERROR_TH

3.6.1 EWL

Type: Read / Write

Error warning limit register. When error warning limit is reached interrupt can be called. Error warning limit by default (96)indicates heavily disturbed bus.

Bit offset	8-0
Name	EWL
Default value	96

EWL Error warning limit

3.6.2 ERP

Type: Read / Write

Error passive limit. When one of error counters (RXC/TXC) reaches this value it changes Fault confinement state to error passive.

Bit offset	15-9
Name	ERP_LIMIT
Default value	128

ERP_LIMIT Error passive limit

3.6.3 FAULT STATE

Type: Read

Fault confinement state of the node. This state can be manipulated by writing into registers RXC/TXC and ERP_LIMIT of ERP register. When these counters are set Fault confinement state changes automatically.

Bit offset	31-19	18	17	16
Name	Reserved	BOF	ERP	ERA
Default value	-	0	0	1

ERA Error active

ERP Error passive

BOF Bus off

3.7 ERROR_COUNTERS

3.7.1 RXC/CTR_PRES

Type: Read / Write

When reading from register bit meaning is according to RXC register. When writing to this register bit meaning is according to CTR_PRES.

RXC:

Bit offset	15-0
Name	RXC_VAL
Default value	0

RXC_VAL Receive error counter. This register determines fault confinement state of the device.

CTR_PRES:

Bit offset	15-11	10	9	8-0
Name	Reserved	PRX	PTX	CTR_PRES_VAL
Default value	-	0	0	0

CTR_PRES_VAL Value to which preset the error counter

PTX Preset Transmit error counter into value in CTR_PRES_VAL. This bit is automatically erased after write.

PRX Preset Receive error counter into value in CTR_PRES_VAL. This bit is automatically erased after write.

3.7.2 TXC

Type: Read

Bit offset	31-16
Name	TXC_VAL
Default value	0

TXC_VAL Transmit error counter. This register determines fault confinement state of the device.

3.8 ERROR_COUNTERS_SPECIAL

Special error counters does not influence fault confinement state of CAN node but enable comparison of error rates in both Bit times. These registers can be set to zero. These register are increased by one when any errors appear.

3.8.1 ERR_NORM/CTR_PRES_S

Type: Read/Write

When reading from register bit meaning is according to ERR_NORM register. When writing to this register bit meaning is according to CTR_PRES_SPECIAL.

ERR_NORM:

Bit offset	15-0
Name	ERR_NORM_VAL
Default value	0

ERR_NORM_VAL Number of errors in Nominal bit time.

CTR_PRES_SPECIAL:

Bit offset	15-13	12	11	10-0
Name	Reserved	EFD	ENORM	Reserved
Default value	-	0	0	-

ENORM Erase error counter for Nominal bit time. This bit is set to zero after counter is erased.

EFD Erase error counter for Data bit time. This bit is set to zero after counter is erased.

3.8.2 ERR_FD

Type: Read

Bit offset	31-16
Name	ERR_FD_VAL
Default value	0

ERR_NORM_VAL

Number of errors in Data bit time.

3.9 FILTER_X_MASK

Type: Read / Write

Bit mask for acceptance filters. Filters A,B,C are available. The identifier format is same as transmitted and received identifier format. BASE Identifier is 11 LSB, Identifier extension is on bits 28-12!

Bit offset	31-29	28-0
Name	Reserved	BIT_MASK_X_VAL
Default value	-	0

BIT_MASK_X_VAL Bit mask for acceptance filters. Logic 1 indicates this bit of Income identifier is compared with the same bit in FILTER_X_VALUE. Logic 0 indicates this bit is not compared.

3.10 FILTER_X_VALUE

Type: Read / Write

Bit value for acceptance filters. Filters A,B,C are available. The identifier format is same as transmitted and received identifier format. BASE Identifier is 11 LSB, Identifier extension is on bits 28-12!

Bit offset	31-29	28-0
Name	Reserved	BIT_VAL_X_VAL
Default value	-	0

BIT_VAL_X_VAL Bit Value for acceptance filters to be compared with income identifier. Only bits set in according FILTER_X_MASK register are compared.

3.11 FILTER_RAN_LOW

Type: Read / Write

Range filter lower threshold. Note that 29 bit value of range threshold is not same format as transmitted and received identifier! In TX_DATA_4 (transmitted identifier) BASE Identifier is at 11 LSB bits and Extension at bits 28-12. However actual decimal value of identifier is that BASE Identifier is at MSB bits and 18 LSB bits is identifier extension. Binary value of actual value of identifier should be written into this register!

Bit offset	31-29	28-0
Name	Reserved	BIT_RAN_LOW_VAL
Default value	-	0

BIT_RAN_LOW_VAL Low threshold value

3.12 FILTER_RAN_HIGH

Type: Read / Write

Range filter upper threshold. Note that 29 bit value of range threshold is not same format as transmitted and received identifier! In TX_DATA_4 (transmitted identifier) BASE Identifier is at 11 LSB bits and Extension at bits 28-12. However actual decimal value of identifier is that BASE Identifier is at MSB bits and 18 LSB bits is identifier extension. Binary value of actual value of identifier should be written into this register!

Bit offset	31-29	28-0
Name	Reserved	BIT_RAN_HIGH_VAL
Default value	-	0

BIT_RAN_LOW_VAL High threshold value

3.13 FILTER_CONTROL

Type: Read / Write

Every filter can be set to accept only selected frame types. Every bit active in logic 1.

Bit offset	3,7,11,15	2,6,10,14	1,5,9,13	0,4,8,12
Name	FD_EXT_FRAME	FD_FRAME	EXT_FRAME	BASIC_FRAME
Default value	1,0,0,0	1,0,0,0	1,0,0,0	1,0,0,0

BASIC_FRAME If CAN Basic Frame should be accepted by filter (Bit 0 - Filter A, Bit 4 - Filter B, Bit 8 - Filter C, Bit 12 - Range filter)

EXT_FRAME If CAN Extended Frame should be accepted by filter (Bit 1 - Filter A, Bit 5 - Filter B, Bit 9 - Filter C, Bit 13 - Range filter)

FD_FRAME If FD CAN Basic Frame should be accepted by filter (Bit 2 - Filter A, Bit 6 - Filter B, Bit 10 - Filter C, Bit 14 - Range filter)

FD_EXT_FRAME If CAN FD Extended Frame should be accepted by filter (Bit 3 - Filter A, Bit 7 - Filter B, Bit 11 - Filter C, Bit 15 - Range filter)

3.14 RX_INFO_1

Information register 1 about FIFO receive buffer.

Type: Read

3.14.1 RX_STATUS

Bit offset	7-2	1	0
Name	Reserved	RX_FULL	RX_EMPTY
Default value	-	0	1

RX_FULL Receive buffer is full.

RX_EMPTY Receive buffer is empty. Since buffer is FIFO like it can be not empty neither full!

3.14.2 RX_MC

Register with number of CAN Messages in FIFO buffer. Note that one message takes several 32 bit words in buffer memory.

Bit offset	15-8
Name	RX_MC_VALUE
Default value	0

RX_MC_VALUE Receive buffer message count value.

3.14.3 RX_MF

Bit offset	23-16
Name	RX_MF_VALUE
Default value	Buffer size

RX_MF_VALUE Number of free (32 bit) words in RX Buffer

3.15 RX_INFO_2

Type: Read

3.15.1 RX_BUFF_SIZE

Bit offset	7-0
Name	RX_BUFF_SIZE_VALUE
Default value	Depends on buffer size set before synthesis

RX_BUFF_SIZE_VALUE Size of receive buffer. This parameter is configurable before synthesis.

3.15.2 RX_WPP

Bit offset	15-8
Name	RX_WPP_VALUE
Default value	0

RX_WPP_VALUE Write pointer position in receive buffer. When new message is stored write pointer is increased accordingly.

3.15.3 RX_RPP

Bit offset	23-16
Name	RX_RPP_VALUE
Default value	0

RX_RPP_VALUE Read pointer position in receive buffer. When RX_DATA register is read read pointer is increased accordingly.

3.16 RX_DATA

Bit offset	31-0
Name	RX_DATA_VALUE
Default value	0

Type: Read

RX_DATA_VALUE Receive buffer data at read pointer position. Refer to Receive buffer description for Message format! It is the same as TX Buffer message format. By reading data from this register read_pointer is automatically increased (If not all data were read). Next Read from this register then returns next word of message.

3.17 TRV_DELAY

Type: Read

Bit offset	15-0
Name	TRV_DELAY_VALUE
Default value	0

TRV_DELAY_VALUE When sending CAN FD Frame with bit rate shift, transceiver delay is measured to apply secondary sampling point for bit error detection during transmission. After measurement is done (after EDL bit) it can be read from this register. Value in this register is valid until start of next message. It is recommended to set bit rate shift interrupt and read this value directly after bit rate is shifted in interrupt handling. This register can be used for transceiver TXD to RXD delay verification.

3.18 TX_STATUS

Type: Read

Bit offset	31-2	1	0
Name	Reserved	TXT_2_EMPTY	TXT_1_EMPTY
Default value	-	1	1

TXT_1_EMPTY Active when Transmit buffer 1 is empty.

TXT_2_EMPTY Active when Transmit buffer 2 is empty.

3.19 TX_SETTINGS

Type: Read / Write - Partially automatically erased

This register enables inserting the message to be transmitted (in registers TX_DATA_1 to TX_DATA_20) into transcieve buffers. All bits are active in logic 1.

Bit offset	31-4	3	2	1	0
Name	Reserved	TXT_1_COMMIT	TXT_2_COMMIT	TXT_1_ALLOW	TXT_2_ALLOW
Default value	-	0	0	1	1

TXT_2_ALLOW Allow transmitting messages from TXT buffer 2.

TXT_1_ALLOW Allow transmitting messages from TXT buffer 1.

TXT_2_COMMIT When active value is written into this bit, message in registers TX_DATA_1 to TX_DATA_20 are inserted into Transmit buffer 2. Afterward this value is automatically erased.

TXT_1_COMMIT When active value is written into this bit, message in registers TX_DATA_1 to TX_DATA_20 are inserted into Transmit buffer 1. Afterward this value is automatically erased.

3.20 TX_DATA_X

Type: Write

Registers TX_DATA_1 to TX_DATA_20 contain message to be inserted into transmit buffers. The data in these registers must have following format in order to be properly sent:

Register name	Bit offset											
	31-11	10	9	8	7	9	5	4	3	2	1	0
TX_DATA_1	Reserved	Reserved	BRS	TBF	FR_TYPE	ID_TYPE	RTR	Reserved	DLC			
TX_DATA_2	TS_VAL(63:32)											
TX_DATA_3	TS_VAL(31:0)											
TX_DATA_4	ID_EXT	ID_BASE										
TX_DATA_5	DATA_1 to DATA_4											
TX_DATA_6	DATA_5 to DATA_8											
...	...											
TX_DATA_20	DATA_61 to DATA_64											

DLC Data length code of frame to transcieve according to [2].

RTR Remote transmission request. Logic 1 in this bit means that RTR frame will be sent. DLC value is then sent only when RTRP bit of MODE register is set. Otherwise DLC value has no meaning when this bit is set and zero length DLC is send. Note that RTR frames are valid only CAN format message.

ID_TYPE Logic 1 in this bit means that frame with extended identifier will be sent, otherwise base identifier will be sent.

FR_TYPE Logic 1 in this bit means that CAN FD frame will be sent. Otherwise CAN frame will be sent

TBF Time based format. This bit should be always set to logic 1.

BRS Bit rate shift. Logic 1 in this bit means that bit rate will be shifted for DATA and CRC field during message transfer. This bit is valid only for CAN FD frames, it has no meaning for CAN frames.

TS_VAL Time stamp value when controller should attempt to start message transmission. If bus is Idle then transmission will start within next bit time. Otherwise it will start as soon as bus is idle. If message should be transmitted immediately all zeroes should be written into these two registers.

ID_BASE Identifier base.

ID_EXT Identifier extension. Note that MSB bit is sent first. If extended frame is sent then value in register TX_DATA_4 is not binary value of whole identifier. Whole identifier should be then converted to binary number and 11 MSBs should be written to bits 10:0. 18 LSBs should be written to bits 28:11!! Other bits should be zero.

DATA_X Data to be transmitted. The amount of transcieved data is given by DLC code. If data length is shorter than 64 than remaining data bytes have no meaning. Data are transmitted MSB first.

3.21 RX_COUNTER

Type: Read / Write

Bit offset	31-0
Name	RX_COUNTER_VALUE
Default value	0

RX_COUNTER_VALUE Counter for received messages to enable bus traffic measurement.

3.22 TX_COUNTER

Type: Read / Write

Bit offset	31-0
Name	TX_COUNTER_VALUE
Default value	0

TX_COUNTER_VALUE Counter for transcieved messages to enable bus traffic measurement.

3.23 LOG_TRIG_CONFIG

Type: Read / Write

Register for configuration of event logging triggering conditions. If Event logger is in Ready state and any of triggering conditions appear it starts recording the events on the bus (moves to Running state). Logic 1 in each bit means this triggering condition is valid.

Bit offset	7	6	5	4	3	2	1	0
Name	T_USRW	T_BRS	T_ERR	T_OVL	T_TRV	T_REV	T_ARBL	T_SOF
Default value	0	0	0	0	0	0	0	0

Bit offset	15	14	13	12	11	10	9	8
Name	T_ERPC	T_EWLR	T_ACKNR	T_ACKR	T_CRCS	T_DATS	T_CTRS	T_ARBS
Default value	0	0	0	0	0	0	0	0

Bit offset	31-18	17	16
Name	Reserved	T_RES	T_TRS
Default value	0	0	0

T_SOF Trigger on Start of frame field appears

T_ARBL Trigger on arbitration was lost

T_REV Trigger on valid message received

T_TRV Trigger on valid message transmitted.

T_ERR Trigger on error appeared

T_USR When logic 1 is written into this bit event logging is triggered immediately

T_BRS Trigger when bit rate is shifted

T_OVL Trigger when Overload frame is transmitted

T_ARBS Trigger on Arbitration filed starts

T_CTRS Trigger on Control field starts

T_DATS Trigger on Data field starts

T_CRCS Trigger on CRC field starts

T_ACKR Trigger on acknowledge received in ACK slot

T_ACKNR Trigger on acknowledge not received in ACK slot

T_EWLR Trigger on Error warning limit reached

T_ERPC Trigger on Fault confinement state changed

T_TRS Trigger when unit starts transmitting new message.

T_RES Trigger when unit starts receiving new message.

3.24 LOG_CAPT_CONFIG

Type: Read / Write

Register for configuring which events to capture by event logger into logger FIFO memory when event logger is running.

Bit offset	7	6	5	4	3	2	1	0
Name	C_ARBS	C_BRS	C_ERR	C_OVL	C_TRV	C_REV	C_ARBL	C_SOF
Default value	0	0	0	0	0	0	0	0

Bit offset	15	14	13	12	11	10	9	8
Name	C_TRS	C_ERC	C_EWR	C_ACKNR	C_ACKR	C_CRCS	C_DATS	C_CTRS
Default value	0	0	0	0	0	0	0	0

Bit offset	31-18	20	19	18	17	16
Name	Reserved	C_DESTUFF	C_STUFF	C_SYNE	C_RES	C_TRS
Default value	0	0	0	0	0	0

C_SOF Capture when Start of frame field appears

C_ARBL Capture when arbitration was lost

C_REV Capture when valid message received

C_TRV Capture when valid message transmitted.

C_ERR Capture when error appeared

C_OVL Capture when error appeared

C_BRS Capture when bit rate is shifted

C_ARBS Capture when Overload frame is transmitted

C_CTRS Capture when Control field starts

C_DATS Capture when Data field starts

C_CRCS Capture when CRC field starts

C_ACKR Capture when Acknowledge was received in ACK Slot

C_ACKNR Capture when Acknowledge was not received in ACK Slot

C_EWR Capture when Error warning limit is reached

C_ERC Capture when Fault confinement state is changed

C_TRS Capture when unit starts transmitting.

C_TRS Capture when transcieve of message started

C_RES Capture when receive of message started

C_SYNE Capture when synchronization edge was detected (recessive to dominant edge)

C_STUFF Capture when Stuff bit was inserted (transceiver only, one fixed stuff bit before CRC sequence is not captured)

C_DESTUFF Capture when received bit is destuffed (receiver and transceiver, one fixed stuff bit before CRC sequence is not captured)

3.25 LOG_STATUS

Type: Read

3.25.1 LOG_STAT

Bit offset	15-8	7-3	2	1	0
Name	LOG_SIZE	Reserved	LOG_RUN	LOG_RDY	LOG_CFG
Default value	Configurable	-	0	0	1

LOG_CFG Event logger is in Config state

LOG_RDY Event logger is in Ready state

LOG_RUN Event logger is in Running state

LOG_SIZE Size of event logger.

3.25.2 LOG_WPP

Bit offset	23-16
Name	LOG_WPP_VAL
Default value	0

LOG_WPP_VAL Logger write pointer position

3.25.3 LOG_RPP

Bit offset	31-24
Name	LOG_RPP_VAL
Default value	0

LOG_RPP_VAL Logger read pointer position

3.26 LOG_COMMAND

Type: Write - Automatically erased

Register for controlling the state machine of Event logger and read pointer position. Every bit is active in logic 1.

Bit offset	7-4	3	2	1	0
Name	Reserved	LOG_DOWN	LOG_UP	LOG_ABT	LOG_STR
Default value	-	0	0	0	0

LOG_STR Start event logging. Move from Config State to Ready state. Has no effect in Ready state or Running state

LOG_ABT Abort event logging. Move from Ready State or Running State to Config State.

LOG_UP Move read pointer one position up.

LOG_DOWN Move read pointer one position down.

3.27 LOG_CAPT_EVENT_1

Type: Read

Upper 32 bits of Event logger memory at read pointer position.

Bit offset	31-0
Name	EVENT_TIME_STAMP(47:16)
Default value	0

EVENT_TIME_STAMP(47:16) Bits 47 to 16 of time stamp when event occurred. MSB 16 bits from 64 bit time stamp are not captured due to saving capacity after synthesis. 48 bits create enough of resolution to distinguish between events.

3.28 LOG_CAPT_EVENT_2

Type: Read

3.28.1 EVENT_TS(15:0)

Bit offset	31-16
Name	EVENT_TIME_STAMP(15:0)
Default value	0

EVENT_TIME_STAMP(15:0) Bits 15 to 0 of time stamp when event occurred.

Event Type Name	EVENT_DETAILS value	EVENT_TYPE
Start of frame	0x00	0x01
Arbitration lost	0x00	0x02
Message reception valid	0x00	0x03
Message transmission valid	0x00	0x04
Overload frame transcieved	0x00	0x05
Error appeared	ERR_DATA	0x06
Bit rate was shifted	BRS_DATA	0x07
Arbitration field started	0x00	0x08
Control field started	0x00	0x09
Data field started	0x00	0x0A
CRC field started	0x00	0x0B
Acknowledge recieved	0x00	0x0C
Acknowledge not recieved	0x00	0x0D
Error warning limit reached	0x00	0x0E
Fault confinement state changed	0x00	0x0F
Transcieve started	0x00	0x10
Recieve started	0x00	0x11
Synchronisation edge appeared	SYNC_DATA	0x12
Stuff bit was inserted	STUFF_DATA	0x13
Bit was destuffed	DESTUFF_DATA	0x14
Data overrun appeared	0x00	0x15

Table 3: Event details register

3.28.2 EVENT_INFO

Bit offset	15-8	7-0
Name	EVENT_DETAILS	EVENT_TYPE
Default value	0	0

EVENT_DETAILS Details of event which appeared. Refer to 3 for bit meanings.

EVENT_TYPE Type of event which was captured. Refer to 3 for bit meanings

Bit offset	15	14	13	12	11	10	9	8
ERR_DATA	Reserved			FRM_ERR	ACK_ERR	CRC_ERR	ST_ERR	BIT_ERR
BRS_DATA	Reserved						S_DOWN	S_UP
SYNC_DATA	Reserved		SYNC_TYPE		IS_PH2	IS_PH1	IS_PROP	IS_SYNC
STUFF_DATA	Reserved				F_STUFF	STUFF_LENGTH		
DESTUFF_DATA	Reserved				F_DESTUFF	DESTUFF_LENGTH		

FRM_ERR Form error was captured

ACK_ERR Acknowledge error was captured

CRC_ERR CRC error was captured

ST_ERR Bit stuffing error was captured

BIT_ERR Bit error was captured

S_UP Bit rate was shifted from Nominal to Data

S_DOWN Bit rate was shifted from Data to Nominal

IS_SYNC Synchronization edge appeared during Synchronization segment of Bit time

IS_PROP Synchronization edge appeared during Propagation segment of Bit time

IS_PH1 Synchronization edge appeared during Phase 1 segment of Bit time

IS_PH2 Synchronization edge appeared during Phase 2 segment of Bit time

STUFF_LENGTH Amount of same consecutive bits after which stuff bit is inserted

DESTUFF_LENGTH Amount of same consecutive bits after which stuff bit is discarded

F_STUFF Whenever Fixed bit stuffing method was used (CRC field of FD Frames)

F_DESTUFF Whenever Fixed bit destuffing method was used (CRC field of FD Frames)

3.29 YOLO_REG

Type: Read

Register for fun :)

Bit offset	31-0
Name	YOLO_VAL
Default value	0xDEADBEEF

YOLO_VAL What else could be in this register??

4. RTL Simulation

Whole circuit functionality was verified by RTL tests. No code coverage software was used for verification. Two types of test-benches were made. Basic test-benches are verifying just main functionality of every circuit with known inputs (result observed just visually). Second group of test-benches generates random data send it on the bus by one node and detects it by another node.

Note that if construct "if generate" is used for conditional synthesis it is impossible to add signals from conditionally added component by script in ModelSim!

4.1 Basic test-benches

Following basic test-benches were implemented:

intMan_tb.vhd Test-bench for interrupt manager. Interrupt sources are set and according signals are put to active state (logic 1). Interrupt and Interrupt vector proper functionality is verified.

protocolControl_tb1.vhd Test-bench where four different frame formats are put to input of protocolControl module. Data is then transmitted on the CAN bus.

CRC_tb.vhd Test-bench for calculating all CRC value for serial data stream with known correct value of CRC.

bitStuffing_v2_tb.vhd Verification of bit destuffing functionality. Recursive bit stuffing method is verified. Fixed bit stuffing is verified and various stuff lengths are verified.

bitDeStuffing_tb.vhd Verification of bit destuffing functionality and stuff error detection

prescaler_v3_tb.vhd Verification of Nominal bit time and Data bit time clock generation. Verification of proper hard synchronization and resynchronization

busSync_tb_edge.vhd Bus synchronization simple test-bench detecting the edge on input data with known sample signal.

busSync_tb_sample.vhd Test-bench simulating bit sequence before bit rate switch. Transceiver delay is measured, then bit rate is shifted and bit Error output with secondary sampling is observed.

rxBuffer_tb.vhd Simple test-bench storing messages into RX Buffer. Verification of data overrun, buffer status signals. Data are then read from the buffer.

txArbitrator_tb.vhd Test-bench for selecting the data from TXT Buffers. Verification of propagation when time stamp of frame is reached.

messageFilter_tb.vhd Verification of mask filters and range filters. Input frame format verification.

txtBuffer_tb.vhd Transceive buffer status verification. Frame is stored and propagated to output.

4.2 Constraint random test-benches

Two test-benches verifying the functionality with constrained random verification were implemented. Every test-bench have many tests which can be activated before compilation.

4.2.1 core_top_tb1.vhd

This test-bench creates two instances of core_top (CAN Core) and generates random data, frame format, frame type, identifier type and bit rate shift. Then randomly generated data are sent by one node and received by another node. When there is mismatch between the transmitted data and received data, DATA_MISMATCH signal is set. Within this testbench these tests are availiable:

tran_rec_test Transcieve-recieve test.

arbit_test_1 Arbitration test. Data are generated and sent by both nodes. Arbitration procedure is verified this way.

error_test Error test forcing incorrect value on the bus in different frame fields. Error passive and error active state testing.

Note that only one test can be executed at a time.

4.2.2 top_level_2_tb.vhd

Tesbench implements whole system developed in [5]. EMIF decoder,output multiplexor and TimeStamp generator components are used. This testbench instantiates two CAN Nodes in slots 1 and 2. Bus level with configurable Transceiver delay is created with recessive and dominant values. Following tests (configurable before compilation), are available:

check_system_test Test which reads out identifiers of Output Multiplexor and Time Stamp generator. If any of identifiers are not read correctly test fails.

rec_buf_test Test for reception of frames. Random frames are inserted into node 2 and send on the bus. Data are sent on the bus and received by node 1. Then data are read out from RX_Buffer of node 1 and compared with transmitted data. Amount of CAN messages to test is specified by "rec_buf_dur" constant.

data_over_run_test Testing of Data Overrun functionality. Two long frames (64 bytes) are sent by node 2 and received by node 1. Second frame is discarded (note that RX buffer size of Node 1 has to be set to 32, otherwise frame wont be discarded) and overrun flag is set. Test fails if data overrun flag is not read after reception of second frame.

tx_arbitrator_test Test for sending the frame in specific value of time stamp. Prescaler of Time stamp generator is set and frame is inserted (time stamp value 500 used). Then status of the node is checked in cycle. Test fails when data were transmitted earlier than time stamp value 500 or when data are not being transmitted after time stamp value 500 was reached.

message_counters_test Bus traffic TX counter is read, then frame is transmitted and TX counter is read again. Test fails if value read after the transmission is not higher by one than previous value.

filter_test Filter C and Range filter are enabled and three frames are sent on the bus by node 2. First one does not pass any of enabled filters. Second one pass Range filter and third one pass Filter C. After each frame number of messages is read from RX_STATUS register.Test fails when frame was stored which was not supposed to pass the filters or vice versa.

fault_conf_test Error counters are set by write and fault confinement state is then read from fault confinement register. Test fails if read state does match expected one.

event_logger_test Test for capturing events by event logger. Start of frame is captured and any event is set as trigger. Afterward 16 messages are sent and event logger state is checked if running. When logging is finished one event is read out and checked if correct Event type value was recorded. Test fails if write pointer between the writes is not at expected position or when recorded event type is not Start of frame.

fd_support_test Test for forbidding Flexible data-rate support of CAN node. FD Message is transmitted (by node 2) and then error counter is checked if receiver sent an error frame due to form error in EDL bit. Test fails if error counter was not incremented.

overload_test Dominant level is forced on the bus level during overload condition. Output of this test is only visual. Transmission of overload frame must be observed.

arbitration_test Two messages are sent by both node at the same time with different identifier. Arbitration procedure is observed just visually. Test fails if arbitration lost capture value read out after the arbitration is not correct.

interrupt_test Transmit and receive interrupts are tested for synthesized nodes. Test fails when interrupt source was not set in interrupt vector.

5. Synthesis and testing

Whole design is synthesized as part of system developed in [5]. ALTERA FPGA EP4CE55F23C8N with 55856 LUTs is used as target device.

5.1 Design size

Design size depends on configuration constants (size of buffers and event logger). Synthesis was performed several times and various results were obtained. Settings for balanced Synthesis and Fitting were used. Table 4 shows these results. Note that this design contains also EMIF Decoder, Output Multiplexor and Time Stamp generator components described in [5].

RX Buffer size	Logger size	TX Buffer size	Overall design size (LUTs)
16	0 (not used)	Basic size	8 743
32	0 (not used)	Basic size	9 978
64	0 (not used)	Basic size	12 394
128	0 (not used)	Basic size	16 824
256	0 (not used)	Basic size	27 137
16	0 (not used)	FD size	10 296
16	8	FD size	11 477
16	16	FD size	12 441
16	64	FD size	19 621
16	256	FD size	37 631

Table 4: Design size in FPGA

Based on Table 4 simple equation for size of the design was estimated (equation is just estimation, design size might change rapidly with different synthesis, optimization settings or different technology):

$$N \approx 1500 + a * [6000 + 76 * b + c * 1550 + d * (104 * e + 1047)]$$

Where:

$$N - Design\ size$$
$$a - Number\ of\ CAN\ controllers\ in\ the\ design$$
$$b - Recieve\ buffer\ size$$
$$c = 0\ when\ isFDSize = false;\ 1\ when\ isFDSize = true$$
$$d = 0\ when\ useLogger = false;\ 1\ when\ useLogger = true$$
$$e = Logger\ size$$

Note that this equation is valid for system developed in [5]. Without adding 1500 LUTs (EMIF decoder, Output multiplexor, Time Stamp generator) the equation is valid when only CAN Controller is synthesized. In order to predict if the design will fit into the device at least 5% error of calculated value should be considered. Considering at least 10 % of the device should be always free, the overall calculated size shouldn't exceed 85% of device capacity.

5.2 Timing analysis

TimeQuest timing analyzer was used for timing analysis. Various maximal clk_sys frequencies were reached with different buffer sizes and amount of CAN controllers (from 58 to 70 MHz). Note that maximal circuit frequency depends on actual place and route at the FPGA or ASIC. However for EP4CE55F23C8N device it can be said that maximal frequency of clk_sys to be used is 50 MHz (reserve of 8 MHz is kept).

5.3 FPGA Tests

The Core functionality is just partially verified in real hardware. Test platform developed in [5] is used for this purpose. CAN Core is mapped into memory of Texas Instruments processor and accessed via C program. CANoe program for transmitting on CAN bus was used to communicate with the controller. Speed of 1 MBit was verified with 25 MHz system clock and Prescaler value set to 1. Arbitration mechanism and synchronization mechanism were verified via Signal Tap II logic analyzer. Basic and Extended frames were verified as well as RTR frames.

Since CANoe does not support CAN with Flexible Data-rate functionality , FD frames and bit rate shifting functionality wasn't verified in the real hardware (communication with reference controller). This functionality was verified in RTL tests where random CAN FD frame was generated send on the bus and received by another node. However there is still possibility that error was compensated in the controller, since one implementation is used for transmittion and reception.

Transmission with FD phase was verified with oscilloscope (no acknowledge received).

6. Future work and improvements

As mentioned earlier next version of the CAN FD controller can implement RX Buffer and Event logger as Optionally inferred RAM Memory.

6.1 Missing RTL tests

Since no code coverage software was used there are many RTL tests which were not implemented. Mainly tests with randomized input needs to be implemented. The following list names few:

- Event logger test - Properly verify capturing of all events, triggering on all events. Reveal all possible conflicts if more than one event type is captured.
- Arbitration lost test - Verify arbitration lost position functionality with randomly generated identifiers distributed evenly across the all fields of arbitration field.
- Message filter test - Verify functionality of all Message filters as well as range filters. Verify random identifier and frame types as well as combination with range identifier.
- Interrupt test - Proper interrupt test for all possible sources of interrupts.
- Randomized RX Buffer test - Storing random data into RX Buffer and reading it out. Comparison with reference model (also in VHDL).
- Randomized TX Arbitrator test - Verify all types of frames and identifiers to be transcieved in different times. Verify conflict behaviour (same transmittion time, same identifier)
- prescaler_v3.vhd test - Verify functionality of hard Synchronisation and resynchronisation by creating tesbench with randomly distributed synchronisation edges.
- Internal loopback test - Test functionality of internall loopback mode as well as bus monitoring mode.

6.1 Missing hardware tests

Due to missing hardware Flexible Data-rate functionality was verified only in testbenches (core_top_tb, top_level_2_tb). Verification by communication with reference controller is needed. Bit Rate switching functionality as well as fixed bit Stuffing functionality needs to be verified with reference controller. Also CRC computation for FD Frames needs to be verified with reference controller. It is reccomended to generate random identifiers, data and frame types for transmittion and perform automatic tests. E.g. data are generated by PC program sent via RS232 to reference controller, then

reference controller sends the data on the bus and CAN FP IP function receives these data. As next data are read by Texas insruments processor ([5]) and sent over ethernet to PC program and checked for consistence.

As part of [5] firmware which was written in Code Composer studio needs to be verified. More functions for manipulating filters, event logger need to be written. Simple API with firmware functions needs to be created to enable manipulation with C functions instead of registers. This API would contain functions for performing tests on CAN bus as well as functions for manipulation with Output Multiplexor and TimeStamp generator.

As part of hardware test's functionality of Minimal Timing (Maximal Bit-rate) needs to be verified.

Appendix A - Driving bus signals

Index	Width	Name	Destination unit	Signal description
0-5	6	drv_tq_nbt	Prescaler	Time quantum length, Nominal bit time
6-11	6	drv_tq_dbt	Prescaler	Time quantum length, Nominal bit time
17-12	8	drv_prs_nbt	Prescaler	Propagation segment length , Nominal bit time
18-23	6	drv_ph1_nbt	Prescaler	Phase 1 segment length, Nominal bit time
24-29	6	drv_ph2_nbt	Prescaler	Phase 2 segment length, Nominal bit time
30-33	4	drv_prs_dbt	Prescaler	Propagation segment length , Data bit time
34-37	4	drv_ph1_dbt	Prescaler	Phase 1 segment length, Data bit time
38-41	4	drv_ph2_dbt	Prescaler	Phase 2 segment length, Data bit time
42-45	4	drv_sjw_nbt	Prescaler	Synchronisation jump width, Nominal bit time
46-49	4	drv_sjw_dbt	Prescaler	Synchronisation jump width, Data bit time
50-60	11	reserved	-	-
61-80	20	reserved	-	-
81-109	29	drv_filter_A_mask	Message filter	Mask for filter A
110-113	4	drv_filter_A_ctrl	Message filter	Allowed frames for filter A
114-142	29	drv_filter_A_bits	Message filter	Bits to compare for filter A
171-143	29	drv_filter_B_mask	Message filter	Mask for filter B
175-172	4	drv_filter_B_ctrl	Message filter	Allowed frames for filter B
204-176	29	drv_filter_B_bits	Message filter	Bits to compare for filter B
205-233	29	drv_filter_C_mask	Message filter	Mask for filter C
234-237	4	drv_filter_C_ctrl	Message filter	Allowed frames for filter C
238-266	29	drv_filter_C_bits	Message filter	Bits to compare for filter C
267-270	4	drv_filter_ran_ctrl	Message filter	Allowed frames for range filter
271-299	29	drv_filter_ran_lo_th	Message filter	Low range treshold for range filter
300-328	29	drv_filter_ran_hi_th	Message filter	High range treshold for range filter
329	1	drv_filter_ena	Message filter	Enable applying message filters.
330-349	29	reserved	-	
350	1	drv_erase_rx	RX Buffer	Erase recieved buffer
351	1	reserved	-	
352	1	drv_read_start	RX Buffer	Move to next word in recieve buffer
353	1	drv_clr_ovr	RX Buffer	Clear Overrun flag
351-355	3	reserved	-	
356	1	drv_erase_txt1	TXT Buffer 1	Erase message in TXT 1 buffer
357	1	drv_store_txt1	TXT Buffer 1	Store message in registers into TXT 1 buffer

Index	Width	Name	Destination unit	Signal description
358	1	drv_erase_txt2	TXT Buffer 2	Erase message in TXT 2 buffer
359	1	drv_store_txt2	TXT Buffer 2	Store message in registers into TXT2 buffer
360	1	reserved	-	
361	1	drv_allow_txt1	TXT Arbitrator	Allow sending messages from TXT1 buffer
362	1	drv_allow_txt2	TXT Arbitrator	Allow sending messages from TXT2 buffer
363-365	3	reserved	-	
366	1	drv_write_tx	TX Buffer	Signal not used
367	1	drv_write_rx	TX Buffer	Signal not used
368-371	4	reserved	-	
372	1	drv_sam	Bus Synchron.	Tripple sampling for slow speeds
373-375	4	reserved	-	
376	1	drv_bus_err_int_ena	Interrupt manager	Enable Bus error interrupt
377	1	drv_arb_lst_int_ena	Interrupt manager	Enable Arbitration lost interrupt
378	1	drv_err_pas_int_ena	Interrupt manager	Enable Fault confinement state changed interrupt
379	1	drv_wake_int_ena	Interrupt manager	Signal not used
380	1	drv_dov_int_ena	Interrupt manager	Enable Data overrun interrupt
381	1	drv_err_war_int_ena	Interrupt manager	Enable Error warning limit reached interrupt
382	1	drv_tx_int_ena	Interrupt manager	Enable sucessfull transcieve interrupt
383	1	drv_rx_int_ena	Interrupt manager	Enable logging finished interrupt
384	1	drv_log_fin_int_ena	Interrupt manager	Enable sucessfull recieve interrupt
385	1	drv_brs_int_ena	Interrupt manager	Enable bit rate shift interrutpt
386	1	drv_rx_full_int_ena	Interrupt manager	Enable interrupt when recieve buffer is full
387	1	drv_int_vect_erase	Interrupt manager	Command to erase interrupt vector
388-399	13	reserved	-	-
400-407	8	drv_ewl	Fault confinement	Error warning limit (by standard 96)
408-415	8	drv_erp	Fault confinement	Error passive treshold (by standard 128)
424-416	8	drv_ctr_val	Fault confinement	Value for presetting error counter
428-425	8	drv_ctr_sel	Fault confinement	Control signals, which counters to preset
459-429	31	reserved	-	-
460	1	drv_CAN_fd_ena	Protocol control	Enable recieve of CAN FD frames
461	1	drv_rtr_pref	Protocol control	RTR preffered behaviour
462-464	3	reserved	-	-
465	1	drv_retr_lim_ena	Protocol control	Retransmittion limit of errornous frames is enabled
466-469	4	drv_retr_th	Protocol control	Retransmittion treshold
470	1	drv_bus_mon_ena	Protocol control	Bus monitoring mode
471	1	drv_self_test_ena	Protocol control	Self Test mode
472	1	drv_abort_tran	Protocol control	Immediately abort actual transmittion
473	1	drv_set_rx_ctr	CAN Core	Preset sucessfully recieved messages counter
474	1	drv_set_tx_ctr	CAN Core	Preset sucessfully transcieved messages counter
475-506	32	drv_set_ctr_val	CAN Core	Value for presetting RX and TX counter
507	1	drv_ack_forb	Protocol control	Acknowledge sending is forbidden
508	1	drv_int_loopback_ena	CAN Core	Internal loopback is enabled
509-519	11	reserved	-	-
520-551	32	drv_trig_config_data	Event logger	Signal is not used

Index	Width	Name	Destination unit	Signal description
552	1	drv_trig_sof	Event logger	Trigger on Start of frame
553	1	drv_trig_arb_lost	Event logger	Trigger on Arbitration lost
554	1	drv_trig_rec_valid	Event logger	Trigger on sucesfull recieve
555	1	drv_trig_tran_valid	Event logger	Trigger on sucesfull transcieve
556	1	drv_trig_ovl	Event logger	Trigger on overload frame transcieved
557	1	drv_trig_error	Event logger	Trigger on error appeared
558	1	drv_trig_brs	Event logger	Trigger on bit rate shifted
559	1	drv_trig_user_write	Event logger	Trigger by logic 1 in this signal
560	1	drv_trig_arb_start	Event logger	Trigger on Arbitration field start
561	1	drv_trig_contr_start	Event logger	Trigger on Control field start
562	1	drv_trig_data_start	Event logger	Trigger on Data field start
563	1	drv_trig_crc_start	Event logger	Trigger on CRC field start
564	1	drv_trig_ack_rec	Event logger	Trigger on acknowledge recieved
565	1	drv_trig_ack_n_rec	Event logger	Trigger on acknowledge was not recieved
566	1	drv_trig_ewl_reached	Event logger	Trigger on error warning limit was reached
567	1	drv_trig_erp_changed	Event logger	Trigger on error passive state changed
568	1	drv_trig_tran_start	Event logger	Trigger on transmittion started
569	1	drv_trig_rec_start	Event logger	Trigger on reception started
570-579	10	reserved	-	-
580	1	drv_cap_sof	Event logger	Capture Start of Frame
581	1	drv_cap_arb_lost	Event logger	Capture Arbitration lost
582	1	drv_cap_rec_valid	Event logger	Capture that message was recieved valid
583	1	drv_cap_tran_valid	Event logger	Capture that message was transcieved valid
584	1	drv_cap_ovl	Event logger	Capture when overload frame is transmitted
585	1	crv_cap_error	Event logger	Capture when error appears
586	1	drv_cap_brs	Event logger	Capture when bit rate is shifted
587	1	drv_cap_arb_start	Event logger	Capture when Arbitration field starts
588	1	drv_cap_contr_start	Event logger	Capture when Control field starts
589	1	drv_cap_data_start	Event logger	Capture when Data field starts
590	1	drv_cap_crc_start	Event logger	Capture when CRC field starts
591	1	drv_cap_ack_rec	Event logger	Capture when Acknowledge was recieved
592	1	drv_cap_ack_n_rec	Event logger	Capture when Acknowledge was not recieved
593	1	drv_cap_ewl_reached	Event logger	Capture when Error warning limit was reached
594	1	drv_cap_erp_changed	Event logger	Capture when Fault confinement state has changed
595	1	drv_cap_tran_start	Event logger	Capture when Transmittion starts
596	1	drv_sap_rec_start	Event logger	Capture when reception starts
597	1	drv_cap_sync_edge	Event logger	Capture that Synchronisation edge appeared
598	1	drv_cap_stuffed	Event logger	Capture that stuff bit was inserted
599	1	drv_cap_destuffed	Event logger	Capture that bit was destuffed from stream
600	1	drv_cap_ovr	Event logger	Capture that data overrun appeared
601-609	9	reserved	-	-
610	1	drv_log_cmd_str	Event logger	Command to start capturing
611	1	drv_log_cmd_abt	Event logger	Command to abort capturing
612	1	drv_log_cmd_up	Event logger	Command to move read pointer up
613	1	drv_log_cmd_down	Event logger	Command to move read pointer down

Appendix B - Status bus signals

Index	Width	Name	Signal description
0-1	2	stat_OP_State	Operation state
2-5	4	stat_PC_State	Protocol Control state
6	1	stat_arb_lost	Arbitration was lost
7	1	stat_set_trans	Unit is set as transciever from next clock
8	1	stat_set_rec	Unit is set as reciever from next clock
9	1	stat_is_idle	Unit is idle
10-11	2	stat_sp_control	Sample point control
12	1	stat_ssp_reset	Secondary sample point reset
13	1	stat_trv_delay_calib	Transciever delay calibration enabled
14-15	2	stat_sync_control	Synchronisation control
16	1	stat_data_tx	Transcieved data
17	1	stat_data_rx	Recieved data
18	1	stat_bs_enable	Bit Stuffing enable
19	1	stat_fixed_stuff	Fixed stuffing method is applied
20	1	stat_data_halt	Bit was stuffed, transmitting should be halted
21-23	3	stat_bs_length	Bit stuffing length
24	1	stat_stuff_error	Stuff Error appeared
25	1	stat_destuffed	Bit is destufffed, shouldntbe recorded by Protocol control
26	1	stat_bds_ena	Bit destuffing is enabled
27	1	stat_stuff_error_ena	Bit Stuffing error detection enabled
28	1	stat_fixed_destuff	Fixed destuffing method should be used
29-31	3	stat_bds_length	Bit destuffing length
32-60	29	stat_tran_ident	Transcieved identifier
61-64	4	stat_tran_dlc	Transcieved dlc
65	1	stat_tran_is_rtr	Transcieved frame is rtr
66	1	stat_tran_frame_type	Transcieved frame type (normal or FD frame)
67	1	stat_tran_ident_type	Transcieved identifier type (basic or extended)
68	1	stat_tran_data_ack	Acknowledge for TXT buffers, TX data are stored in internal buffer
69	1	stat_tran_brs	Transcieved message should shift bitrate
70	1	stat_frame_store	Command to store input frame for transcieve

Index	Width	Name	Signal description
71-79	9	stat_tx_counter	TX error counter
80	1	reserved	-
81-89	9	stat_rx_counter	RX error counter
90-98	9	stat_error_counter_norm	Error counter for errors appeared in nominal bit rate
99-107	9	stat_error_counter_fd	Error counter for errors appeared in data bit rate
108-109	2	stat_error_state	Fault confinement state
110	1	stat_form_error	Form error appeared
111	1	stat_crc_error	CRC error appeared
112	1	stat_ack_error	Acknowledge error appeared
113	1	stat_unknown_state_error	Protocol control is in undefined state
114	1	stat_bit_stuff_error	Bit or Stuff error appeared
115	1	stat_first_bit_after	Signal not used
116	1	stat_rec_valid	Message was recieved valid
117	1	stat_tran_valid	Message was transcieved valid
118	1	stat_const7	Signal not used
119	1	stat_const14	Signal not used
120	1	stat_transm_error	Signal not used
121-149	29	stat_rec_ident_type	Recieved identifier
150-153	4	stat_rec_dlc	Recieved data length code
154	1	stat_rec_is_rtr	Recieved frame is rtr
155	1	stat_rec_frame_type	Recieved frame type (normal or FD)
156	1	stat_rec_ident_type	Recieved identifier type (basic or extended)
157	1	stat_rec_brs	Recieved frame with bit rate shift
158-178	21	stat_rec_crc	Recieved CRC value
179	1	stat_rec_esi	Recieverd Error state indicator
180	1	stat_crc_ena	CRC calculation is enabled
181	1	stat_tran_trig	Transcieve trigger (in sync segment)
182	1	stat_rec_trig	Recieve trigger (in sample point)
183-187	5	stat_alc	Arbitration lost capture
188-219	32	stat_rx_ctr	Sucesfully recieved message counter
220-251	32	stat_tx_ctr	Sucesfully transcieved message counter
252	1	stat_erp_changed	Error passive state has changed
253	1	stat_ewl_reached	Error warning limit was reached
254	1	stat_err_valid	Error is valid
255	1	stat_ack_recieved_out	Acknowledge was recieved
256	1	stat_bit_error_valid	Bit Error appeared

Bibliography

[1] CAN with Flexible Data-Rate Specification v 1.0, Robert Bosch GmbH, April 2012

[2] CAN 2.0 Protocol standard, Robert Bosch GmbH, Rev 3.0

[3] Controller Area Network - Basics, protocols, chips and applications, Prof. Dr.-Ing. K. Etschberger, 2001

[4] CRC for CAN with flexible data rate (CAN FD) - Whitepaper

[5] Software for Test Platform, DataSheet, Ille Ondrej, Czech Technical University, July 2015

[6] Implementation of unconventional CAN controller in VHDL, Diploma Thesis, Dušan Hamza, Czech technical university, 2013

[7] Robustness of a CAN FD Bus System – About Oscillator Tolerance and Edge Deviations, Dr. Arthur Mutter, Robert Bosch GmbH, 2013

[8] SJA1000 Standalone CAN Controller,Philips Semiconductors, January 2000

[9] TJA1041 High speed CAN transceiver Rev. 06, December 2007, NXP Semiconductors

[10] ModelSim Advanced verification and debugging SE Command Reference, Mentor Graphics, v 6.0 November 2004

[11] VHDL guidlines for synthesis, Siemens semiconductor group

[12] FPGA prakticky - Realizace číslicových systémů pro hradlová pole, Jakub Šťastný, BEN Technická literatura 2011

[13] Číslicové systémy a jazyk VHDL , Jiří Pinker, Martin Poupa, BEN Technická literatura 2006

[14] Understanding Metastability in FPGAs, ALTERA

[15] SAM V71, SMART ARM-based Flash MCU, Preeliminary datasheet, Atmel, February 2015

[16] VHDL Guides , Dr. Jayram, Moorkanikara Nageswaran, Department of Computer Science University of California, http://www.ics.uci.edu/~jmoorkan/vhdlref/

[17] Avalon Interface Specifications,ALTERA March 2015

[18] Methods for Testing the FlexRay Start-up Mechanism, Diploma thesis, Martin Paták ,Czech Technical University in Prague, 2012

[19] TimeQuest Timing Analyzer - Quick Start Tutorial, ALTERA, December 2009

[20] Implementing Inferred RAM, http://quartushelp.altera.com/14.0/mergedProjects/hdl/vhdl/vhdl_pro_ram_inferred.htm