NTNU – Trondheim
Norwegian University of
Science and Technology

HDLC Module Design Description

Karianne Krokan Kragseth

November 15, 2017

1 HDLC

High-level Data Link Control (HDLC) is a data communication protocol, which provides bit-oriented code-transparent data transmission.

1.1 HDLC Frames

In Table 1.1 we see the structure of the HDLC frame. The FCS (Frame Check Sequence) bits are generated inside the HDLC module. Address, control and information does not need to be any specific values, but the modules expect them to be byte aligned.

Flag	Address	Control	Information	FCS	Flag
8 bits	8 or more bits	8 bits	Variable length, n * 8 bits	16 bits	8 bits

Table 1.1: HDLC Frame

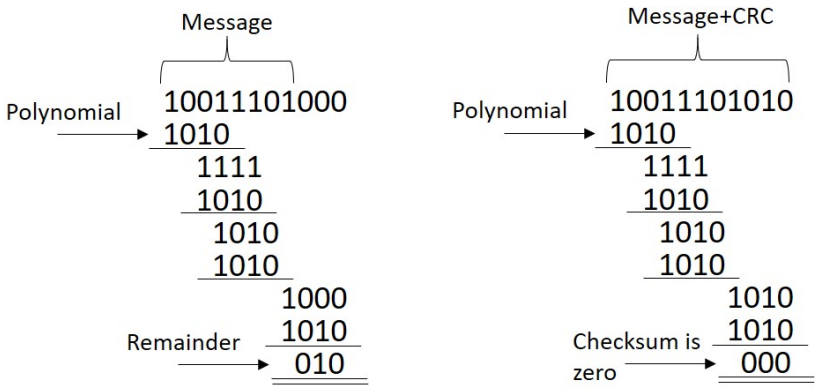
1.2 HDLC Behavior

A frame should start and end with the flag sequence, "0111_1110". The frame is received/transmitted one bit at a time, starting with the least significant bit of the first byte. The FCS should be calculated for the entire length of the frame, excluding the flag bits. The FCS is calculated with the Cycle Redundancy Check(CRC) method. In order to terminate a frame, an abort pattern, "1111_1110", can be generated. If an abort pattern is observed, the current frame should be discarded and ignored. In order to avoid the data in the information field to accidental generate a flag sequence or an abort pattern, a zero should be inserted whenever 5 continuous ones are being transmitted. On the receiving side the information should be examined, and whenever 5 continues ones are received, the following zero should be removed.

1.3 Cycle Redundancy Check(CRC)

CRC calculation is done by dividing the message by a predetermined polynomial, P, used by both sender and receiver, where the remainder after the division is the error correction bits. The polynomial used in this implementation is $P = X^{16} + X^{15} + X^2 + 1$, which is 1 bit larger than the CRC bits and referred to as 0x8005. The CRC calculation is done with bitwise XOR, as the calculation is done with modulo-2 arithmetic without carries or borrows. An example of how the calculation is done is shown in figure 1.1. Here the CRC is calculated for an 8 bit message using a 4 bit polynomial, and the message including CRC is shown to have a zero remainder.

CRC implemented in hardware is done with a shift-register and XOR-gates. This register is initialized to zeros before calculation. Figure 1.2 shows how this is done for the polynomial 0x8005. The CRC bits (the shift register), is found after a whole message has been shifted into the register. When calculating the CRC at the sender side, 16 bits of zeros is added to the message. There is no error at the receiving side if the register only contains zeros after having received a message plus CRC bits.



(a) Generating CRC. (b) Checking CRC.

Figure 1.1: CRC calcualtion example.

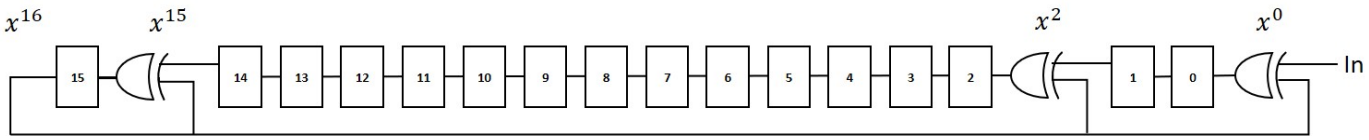


Figure 1.2: Hardware implementation of CRC calculation.

2 Implementation

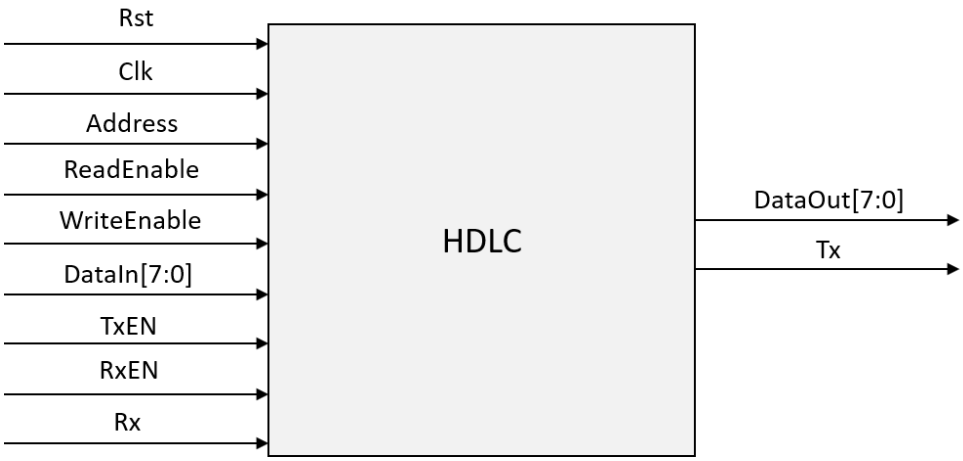


Figure 2.1: Block diagram of HDLC module.

A block diagram of the HDLC can be seen in figure 2.1. This include clock, reset, an address interface, enable TX, enable RX, RX input and TX output. The address interface is used for communicating with the module by reading from/writing to a set of registers. The RX input is used when receiving a frame, and the TX output when transmitting.

2.1 HDLC Signals

Table 2.1 lists the input/output signals of the HDLC module, while table 2.2 lists the internal signals.

Input/output signals:			
Rst	input	logic	Reset
Clk	input	logic	Clock
Tx	output	logic	Tx serial line
Rx	input	logic	Rx serial line
TxEN	input	logic	Tx serial enable
RxEN	input	logic	Rx serial enable
Address	input	logic [2:0]	Address for read/write
WriteEnable	input	logic	Write enable
ReadEnable	input	logic	Read enable
DataIn	input	logic [31:0]	Data input
DataOut	output	logic [31:0]	Data output

Table 2.1: Input/output signals to the HDLC module

Internal signals:		
Tx_ValidFrame	logic	Transmitting valid TX frame
Tx_AbortedTrans	logic	TX transmission is aborted
Tx_WriteFCS	logic	Transmit FCS byte
Tx_InitZero	logic	Initialize zero insertion with two first bytes of TX buffer
Tx_StartFCS	logic	Start FCS calculation
Tx_RdBuff	logic	Read new byte from TX buffer
Tx_NewByte	logic	New byte is loaded to be transmitted
Tx_FCSDone	logic	Finished calculating FCS bytes
Tx_Data	logic [7:0]	Next TX byte to be transmitted
Tx_Done	logic	TX buffer can be written
Tx_Full	logic	TX buffer is full
Tx_DataAvail	logic	Data is available in TX buffer
Tx_FrameSize	logic [7:0]	Number of bytes in TX buffer
Tx_DataArray	logic [127:0][7:0]	TX buffer
Tx_DataOutBuff	logic [7:0]	Next TX byte in buffer to be transmitted
Tx_WrBuff	logic	Write new byte to TX buffer
Tx_Enable	logic	Start transmission of TX buffer
Tx_DataInBuff	logic [7:0]	Data from write operation
Tx_AbortFrame	logic	Abort current TX frame
Rx_ValidFrame	logic	Receiving valid frame
Rx_WrBuff	logic	Write RX byte to buffer
Rx_EoF	logic	Whole RX frame has been received, end of frame
Rx_AbortSignal	logic	RX frame was aborted
Rx_StartZeroDetect	logic	Start detecting inserted zeros
Rx_FrameError	logic	Error in received RX frame
Rx_StartFCS	logic	Start FCS calculation
Rx_StopFCS	logic	Stop FCS calculation
Rx_Data	logic [7:0]	Received RX byte
Rx_NewByte	logic	New RX byte has been received
Rx_FlagDetect	logic	Flag detected
Rx_AbortDetect	logic	Abort detected
RxD	logic	Delayed RX bit, 9 clock cycles
Rx_FCSerr	logic	FCS error
Rx_Ready	logic	RX Buffer ready to be read
Rx_FrameSize	logic [7:0]	Number of bytes received (minus FCS bytes)
Rx_Overflow	logic	RX buffer is full
Rx_DataBuffOut	logic	Data read from RX buffer
Rx_FCSen	logic	FCS error detection enabled
Rx_RdBuff	logic	Read byte from RX buffer
Rx_Drop	logic	Drop received RX frame

Table 2.2: Internal signals in the HDLC module

2.2 Register Interface

The *AddressIF* module contains 5 registers which allows access to operation status information and enables control of the HDLC module for users. The registers are presented in table 2.3, 2.4, 2.5, 2.6 and 2.7, with their addresses and indications which bits are readable and writeable.

Bit:	7 - 5	4	3	2	1	0
Field:	N/A	Tx_Full	Tx_AbortedTrans	Tx_AbortFrame	Tx_Enable	Tx_Done
Reset:	0x0	0	0	0	0	0
R/W:	RO	RO	RO	WO	WO	RO

Table 2.3: TX status/control register (Tx_SC), address=*0x0*

Bit:	7 - 0
Field:	Tx data byte
Reset:	0x0
R/W:	WO

Table 2.4: TX data buffer (Tx_Buff), address=*0x1*

Bit:	7 - 6	5	4	3	2	1	0
Field:	N/A	Rx_FCSen	Rx_Overflow	Rx_AbortSignal	Rx_FrameError	Rx_Drop	Rx_Ready
Reset:	0x0	0	0	0	0	0	0
R/W:	RO	WO	RO	RO	RO	WO	RO

Table 2.5: RX status/control register (Rx_SC), address=*0x2*

Bit:	7 - 0
Field:	Rx data byte
Reset:	0x0
R/W:	RO

Table 2.6: RX data buffer (Rx_Buff), address=*0x3*

Bit:	7 - 0
Field:	Rx frame length
Reset:	0x0
R/W:	RO

Table 2.7: RX frame length (Rx_Len), , address=*0x4*

These registers can be read or written by using the address interface. An example is presented in figure 2.2. When reading a register, done by setting *ReadEnable*, the output data is available on the same clock cycle. The value of the input data will be written to a register on the next clock cycle, which is done by setting *WriteEnable*.

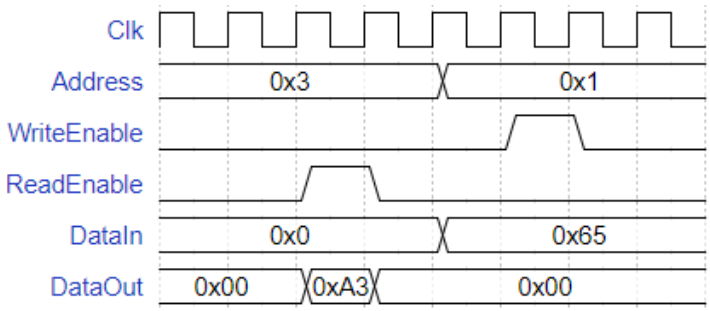


Figure 2.2: Wave diagram of read write operation using *AddressIF*.

2.3 Receive Design

Figure 2.3 shows a block diagram of the receive implementation, which consists of the sub-modules *RxController*, *RxChannel*, *RxFCS* and *RxBuff* and an address interface *AddressIF*. Signals *Clk*, *Rst* and *RxEN* are not included in the figure. Before and after frames, i.e., when it's is not operating, it should only receives ones, which will keep the RX logic in idle state.

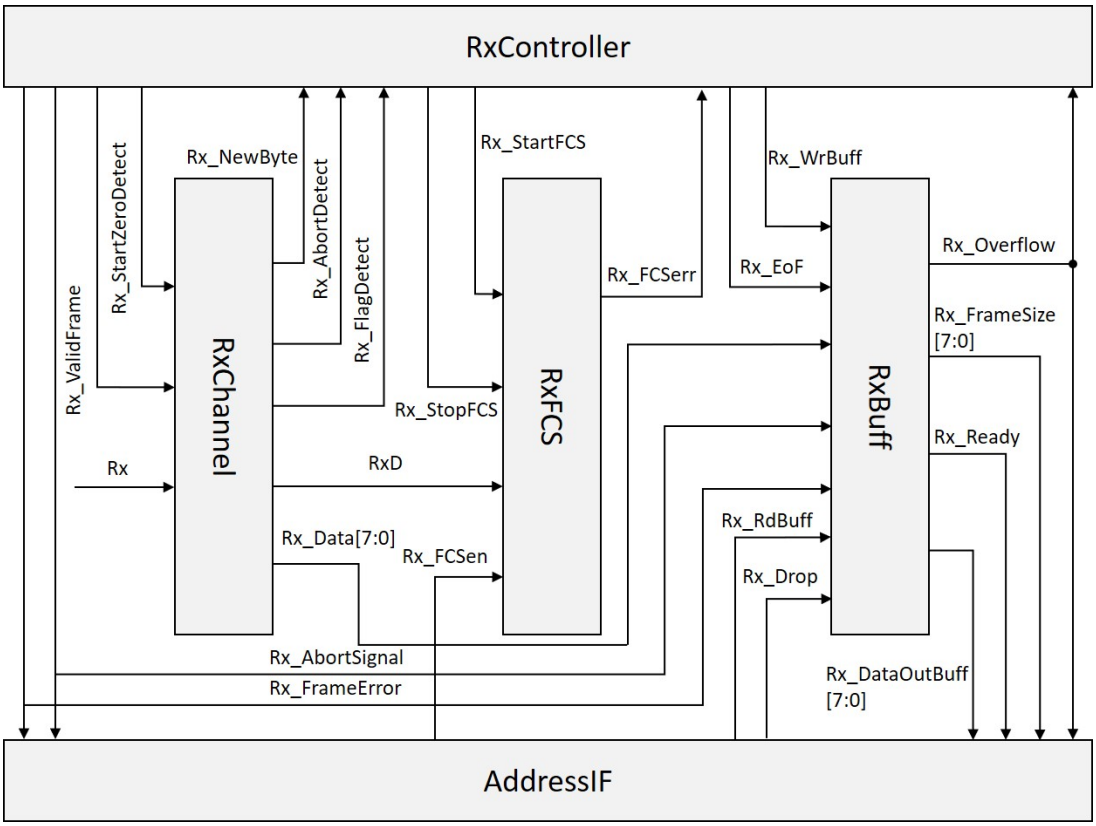


Figure 2.3: Block diagram of RX logic.

RxChannel

The first part of receive chain is *RxChannel*. The RX input is monitored, and whenever the flag sequence or abort pattern is recognized, it signals the controller. *RxChannel* is also responsible for removing inserted zeros, when 5 consecutive ones is received, and collecting the received data into bytes.

RxController

RxController is responsible for controlling the operation of *RxFCS* and *RxBuff*, based on information from *RxChannel*. When a flag is detected, it signals to start FCS calculation, and at each received byte, it signals for the RX buffer to store it. At an end flag or abort pattern it signals the FCS calculation to stop, and signals end of frame for the RX buffer to be ready to be read. It also controls when the zero removal logic should operate in *RxChannel*.

RxFCS

RxFCS checks for CRC errors by continuously calculating from a delayed RX input from *RxChannel*. The calculation is stopped when a whole frame has been received. *FCSerr* is signalled if an error is discovered from the CRC calculation.

RxBuff

The received bits are handled and stored into bytes in the RX buffer in *RxBuff*. This sub-module is responsible for signaling, through the address interface, when the buffer is ready to be read and if there is an overflow. The frame size and the RX buffer is read from this sub-module when using the address interface.

2.4 Transmit Design

The transmit implementation consist of the sub-modules *TxController*, *TxChannel*, *TxFCS* and *TxBuff* and an address interface *AddressIF*, as shown in figure 2.4. Signals *Clk*, *Rst* and *TxEN* are not included in the figure. When not transmitting a frame, only ones are transmitted.

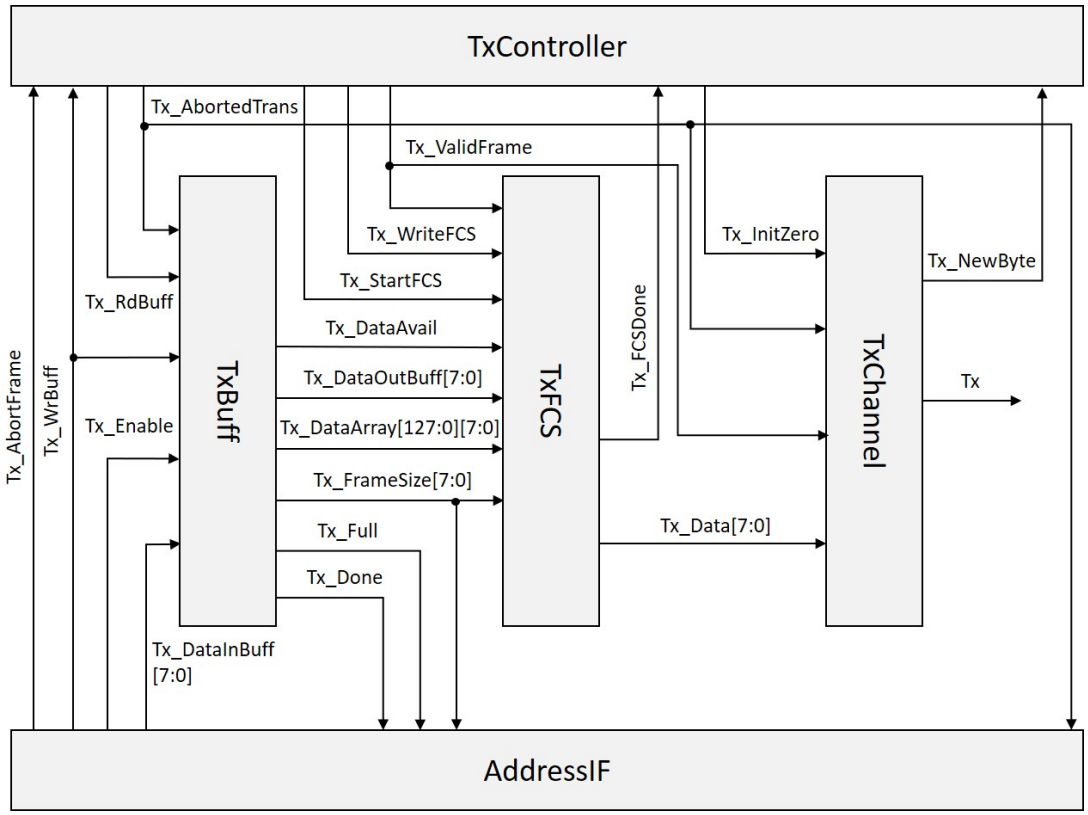


Figure 2.4: Block diagram of TX logic.

TxBuff

TxBuff is responsible for storing the TX buffer, written through *AddressIF*. This sub-module contains logic for storing bytes in the buffer, and reading from it. It also signals through the address interface if it's full and when it's ready to be written.

TxController

TxController is responsible for starting CRC calculation in *TxFCS* when the TX buffer has been written and the enable signal is set through the address interface. After the calculation has finished it starts transmission. This involves signalling to start reading the TX buffer and signal *TxChannel* to transmit a start flag and start zero insertion, followed by transmission of the read bytes. Whenever a byte has been transmitted, it signals for a new byte to be read from the TX buffer. When the whole TX buffer and both FCS bytes are transmitted, it signals for the end flag to be transmitted.

TxFCS

The CRC is calculated in *TxFCS* before transmission is started. The CRC is calculated based on the signals from *TxBuff*, which provides information on the content of the TX buffer and on how many bytes it contains. During transmission *TxFCS* is responsible for sending data to *TxChannel* for transmission, which is either the data byte from *TxBuff* or the calculated CRC bytes.

TxChannel

The TX output is set in *TxChannel*, where the flag sequence, abort pattern or byte from *TxFCS* is broken down into bits and transmitted. It's also responsible for inserting zeros whenever 5 consecutive ones are transmitted.

2.5 Receive Frame

- *Rx_AbortSignal* should be asserted by the *RxController* when *RxChannel* detects an abort flag. But only if it is detected during a valid frame (Fig. 2.5)

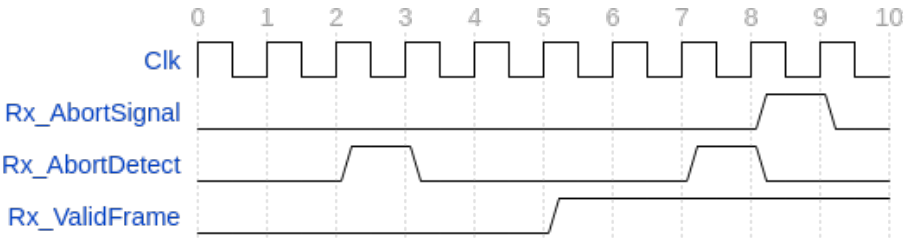


Figure 2.5: Wave diagram of *Rx_AbortSignal* behavior.

- After a whole frame has been received it's recommended to read the *Rx_SC* register to check if everything is in order, with respect to overflow, aborted frame and errors. If the ready bit is set, the RX buffer is ready to be read.
- The *Rx_Len* register should be read to check the number of bytes in the RX buffer. This value is only valid after having received an error free and non-aborted frame, and until a new frame is being received. Maximum size is 126 bytes (128 - 2 FCS bytes).
- The *RxBuff* should be read as long as *Rx_Ready* is set (*Rx_Len* times). Attempting to read this when *Rx_Ready* is not set will only yield zeros.

- If a new frame is being received before the buffer is read, the previous will be discarded.
- A frame can be dropped (discarded) by writing to the drop bit in the *Rx_SC* register.
- After an overflow the *RxBuff* should contain the first 126 bytes of the received data. Attempting to read more than this should result in zeros.

2.6 Transmit Frame

- *Tx_AbortedTrans* should be asserted by the *TxController* module if it receives the *Tx_AbortFrame* signal (Fig. 2.6).

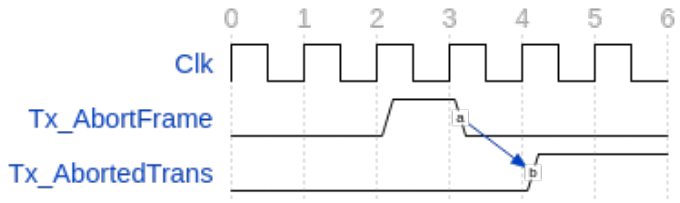


Figure 2.6: Wave diagram of *Tx_AbortedTrans* behavior.

- Check *Tx_Done* bit in *Tx_SC* if TX buffer is ready to be written. After first write this will go low until the whole buffer has been read during transmission. When writing 126 bytes or more, *Tx_Overflow* bit in *Tx_SC* will be set until transmission is started, and the first 126 bytes written will be transmitted.
- Transmit the TX buffer by setting *Tx_Enable* bit in *Tx_SC*.
- The flag sequence is generated after the CRC bytes are calculated, which takes a variable number of clock cycles dependent on the buffer size.