Async/HDLC Port

Technical Manual

Systemyde International Corporation

Disclaimer

Systemyde International Corporation reserves the right to make changes at any time, without notice, to improve design or performance and provide the best product possible. Systemyde International Corporation makes no warrant for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make any commitment to update the information contained herein.

Systemyde International Corporation products are not authorized for use in life support devices or systems unless a specific written agreement pertaining to such use is executed between the manufacturer and the President of Systemyde International Corporation. Nothing contained herein shall be construed as a recommendation to use any product in violation of existing patents, copyrights or other rights of third parties. No license is granted by implication or otherwise under any patent, patent rights or other rights, of Systemyde International Corporation. All trademarks are trademarks of their respective companies.

Every effort has been made to ensure the accuracy of the information contain herein. If you find errors or inconsistencies please bring them to our attention. In all cases, however, the Verilog HDL source code for the hdlc_top design defines “proper operation”.

Copyright © 2010, Systemyde International Corporation. All rights reserved.

Index

Features 3
Bus Interface 5
Async/HDLC Receiver 17
  Interface 21
Async/HDLC Transmitter 25
  Interface 28
Digital Phase-locked Loop 31
  Interface 34
IRDA Encode/Decode 37
  Interface 39
Encoder/Decoder 41
  Interface 43
Top Level Verilog 47

Revision History

Date	Description	Page(s)

Features

hdlc_top

General:

- 4 byte FIFOs for both receive and transmit
- Provision for 8-, 16- or 32-bit bus interface
- Provision for expanded FIFO depth
- Bus interface module (Rabbit 4000 compatible) easily replaced with custom interface
- Bus interface module includes interrupt and DMA requests
- Bus interface module includes 15-bit Baud Rate Generator
- FIFO overrun and underrun reporting

Async mode

- 7 or 8 bits/character
- Optional Even, Odd, Mark or Space parity generation and checking
- Optional 9th bit for Address/Data multiprocessor operation
- 16x or 8x oversampling
- Flase start bit rejection
- Optional IRDA encode and decode

HDLC mode

- Automatic Flag generation and checking
- Automatic Abort generation and checking
- Automatic zero-insertion and deletion
- Automatic CRC generation and checking
- 16- or 32-bit CRC polynomial
- Flag Search command for receiver to stop frame reception
- Send Abort command for transmitter to interrupt frame transmission
- Abort-on-underrun option for transmitter
- Optional DPLL for clock recovery
- Optional IRDA encode and decode
- Optional data encode/decode: NRZ, NRZI, Biphase-Level, Biphase-Mark/Space
- Received frame status (with byte count) eliminates real-time checking of frame status
- Frame status strobe to implement external frame status FIFO

Bus Interface

rab_if

The following register set is included in the **rab_if** module, which is the default Rabbit 4000-compatible bus interface. This bus interface can be used as-is, modified to suit your own requirements, or completely replaced. In any case, all of the serial features will still be available.

This bus interface is uses a byte-wide data bus and a 4-bit address bus, with separate read and write strobes. Timing on the bus interface is similar to that used in the industry-standard APB bus.

This interface provides input multiplexers for the serial data and clocks, combined interrupt requests and separate receive and transmit DMA requests. Also included in a baud rate generator.

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
Serial Port Data Register	SDR	0x8	R/W	xxxxxxxx
Serial Port Address Register	SAR	0x9	W	xxxxxxxx
Serial Port Long Stop Register	SLR	0xA	W	xxxxxxxx
Serial Port Status Register	SSR	0xB	R	0xx00000
Serial Port Control Register	SCR	0xC	R/W	00000000
Serial Port Extended Register	SER	0xD	R/W	00000000
Serial Port Divider Low Register	SDLR	0xE	R/W	xxxxxxxx
Serial Port Divider High Register	SDHR	0xF	R/W	0xxxxxxx

Register Descriptions

Serial Port Data Register		(SDR) (Address = 0x8)
Bit(s)	**Value**	**Description**
7:0	Read	Returns the contents of the receive buffer.
	Write	Loads the transmit buffer with a data byte for transmission.

Serial Port Address Register		(SAR) (Address = 0x9)
Bit(s)	**Value**	**Description**
7:0	Read	Returns the contents of the receive buffer.
	Write	Loads the transmit buffer with an address byte, marked with a “zero” address bit, for transmission. In HDLC mode, the last byte of a frame must be written to this register to enable subsequent CRC and closing Flag transmission.

Serial Port Long Stop Register		(SLR) (Address = 0xA)
Bit(s)	**Value**	**Description**
7:0	Read	Returns the contents of the receive buffer.
	Write	Loads the transmit buffer with an address byte, marked with a “one” address bit, for transmission.

Serial Port Status Register		(SSR) (Address = 0xB)
Bit(s)	**Value**	**Description (Async mode only)**
7	0	The receive data register is empty
	1	There is a byte in the receive buffer. The Serial Port will request an interrupt while this bit is set. The interrupt is cleared when the receive buffer is empty.
6	0	The byte in the receive buffer is data, received with a valid Stop bit.
	1	The byte in the receive buffer is an address, or a byte with a framing error. If an address bit is not expected, and the data in the buffer is all zeros, this is a Break.
5	0	The receive buffer was not overrun.
	1	The receive buffer was overrun. This bit is cleared by reading the receive buffer.
4	0	The byte in the receive buffer has no parity error (or was not checked for parity).
	1	The byte in the receive buffer had a parity error.
3	0	The transmit buffer is empty.
	1	The transmit buffer is not empty. The Serial Port will request an interrupt when the transmitter takes a byte from the transmit buffer. Transmit interrupts are cleared when the transmit buffer is written, or any value is written to this register.
2	0	The transmitter is idle.
	1	The transmitter is sending a byte. An interrupt is generated when the transmitter clears this bit, which occurs only if the transmitter is ready to start sending another byte but the transmit buffer is empty.
1:0	00	These bits are read/write but are always ignored in async mode.

Serial Port Status Register		(SSR) (Address = 0xB)
Bit(s)	**Value**	**Description (HDLC mode only)**
7	0	The receive data register is empty
	1	There is a byte in the receive buffer. The Serial Port will request an interrupt while this bit is set. The interrupt is cleared when the receive buffer is empty.
6,4	00	The byte in the receive buffer is data.
	01	The byte in the receive buffer was followed by an Abort.
	10	The byte in the receive buffer is the last in the frame, with valid CRC.
	11	The byte in the receive buffer is the last in the frame, with a CRC error.
5	0	The receive buffer was not overrun.
	1	The receive buffer was overrun. This bit is cleared by reading the receive buffer.
3	0	The transmit buffer is empty.
	1	The transmit buffer is not empty. The Serial Port will request an interrupt when the transmitter takes a byte from the transmit buffer, unless the byte is marked as the last in the frame. Transmit interrupts are cleared when the transmit buffer is written, or any value (which will be ignored) is written to this register.
2:1	00	Transmit interrupt due to buffer empty condition.
	01	Transmitter finished sending CRC. An interrupt is generated at the end of CRC transmission. Data written in response to this interrupt will cause only one Flag to be transmitted between frames, and no interrupt will be generated by this Flag.
	10	Transmitter finished sending an Abort. An interrupt is generated at the end of an Abort transmission.
	11	The transmitter finished sending a closing Flag. Data written in response to this interrupt will cause at least two Flags to be transmitted between frames.
0	0	The byte in the receiver buffer is 8 bits.
	1	The byte in the receiver buffer is less than 8 bits.

Serial Port Control Register	(SCR)	(Address = 0xC)
Bit(s)	**Value**	**Description**
7:6	00	No operation. These bits are ignored in the Async mode.
	01	In HDLC mode, force receiver in Flag Search mode.
	10	No operation.
	11	In HDLC mode, transmit an Abort pattern.
5:4	00	Input bus bit 2 is used for data (and optional clock) input.
	01	Input bus bit 1 is used for data (and optional clock) input.
	10	Input bus bit 0 is used for data (and optional clock) input.
	11	Disable the receiver data input. Clocks from clock input bus bit 0.
3:2	00	Async mode with 8 bits per character.
	01	Async mode with 7 bits per character. In this mode the most significant bit of a byte is ignored for transmit, and is always zero in receive data.
	10	HDLC mode with external clock. The external clocks are supplied via external inputs.
	11	HDLC mode with internal clock. The clock is 16X the data rate, and the DPLL is used to recover the receive clock. If necessary, the receiver and transmitter clocks can be output via parallel port pins.
1:0	00	The Serial Port interrupt is disabled.
	01	The Serial Port uses Interrupt Priority 1.
	10	The Serial Port uses Interrupt Priority 2.
	11	The Serial Port uses Interrupt Priority 3.

Serial Port Extended Register	(SER)	(Address = 0xD)
Bit(s)	**Value**	**Description (Async mode only)**
7:5	000	Disable parity generation and checking.
	001	This bit combination is reserved and should not be used.
	010	This bit combination is reserved and should not be used.
	011	This bit combination is reserved and should not be used.
	100	Enable parity generation and checking with even parity.
	101	Enable parity generation and checking with odd parity.
	110	Enable parity generation and checking with Space (always zero) parity.
	111	Enable parity generation and checking with Mark (always one) parity.
4	0	Normal async data encoding.
	1	Enable RZI coding (3/16ths bit cell IRDA-compliant).
3	0	Normal Break operation. This option should be selected when address bits are expected.
	1	Fast Break termination. At the end of Break a dummy character is written to the buffer, and the receiver can start character assembly after one bit time.
2	0	Async clock is 16X data rate.
	1	Async clock is 8X data rate.
1	0	Continue character assembly during Break to allow timing the Break condition.
	1	Inhibit character assembly during Break. One character (all zeros, with framing error) at start and one character (garbage) at completion.
0		This bit is ignored in async mode.

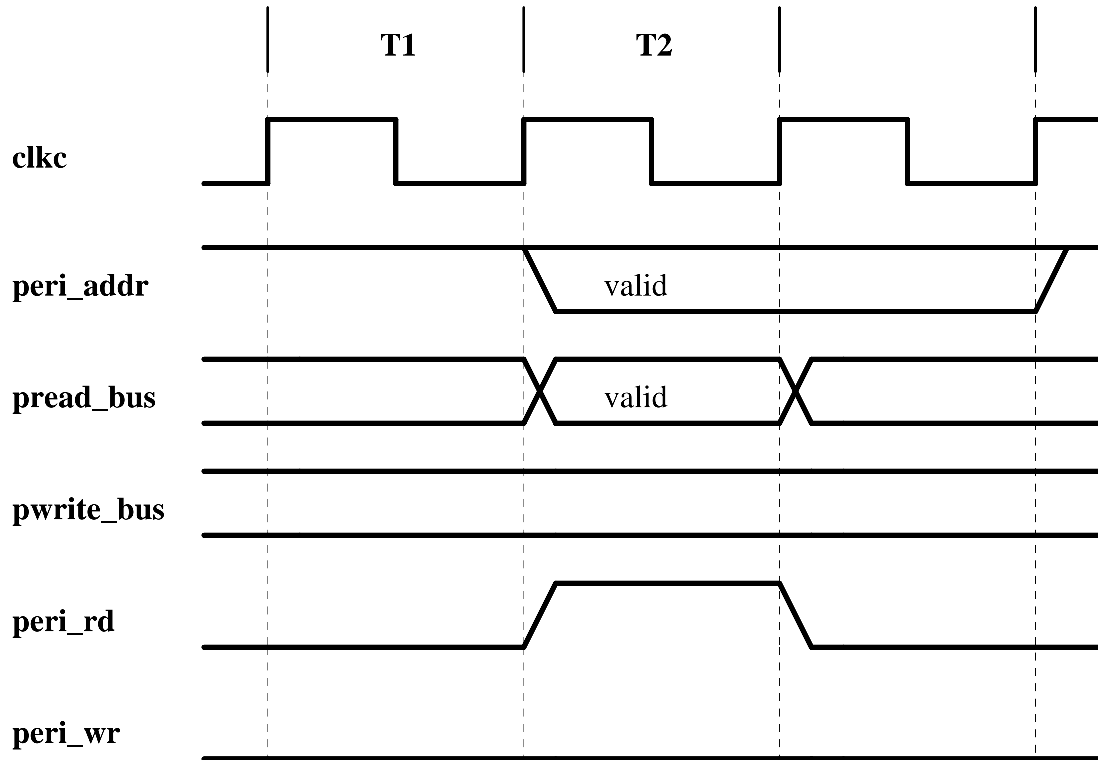
Serial Port Extended Register	(SER)	(Address = 0xD)
Bit(s)	**Value**	**Description (HDLC mode only)**
7:5	000	NRZ data encoding for HDLC receiver and transmitter.
	010	NRZI data encoding for HDLC receiver and transmitter.
	100	Biphase-Level (Manchester) data encoding for HDLC receiver and transmitter.
	110	Biphase-Space data encoding for HDLC receiver and transmitter.
	111	Biphase-Mark data encoding for HDLC receiver and transmitter.
4	0	Normal HDLC data encoding.
	1	Enable RZI coding (1/4th bit cell IRDA-compliant). This mode can only be used with internal clock and NRZ data encoding.
3	0	Idle line condition is Flags.
	1	Idle line condition is all ones.
2	0	Transmit Flag on underrun.
	1	Transmit Abort on underrun.
1	0	Separate HDLC external recieve and transmit clocks.
	1	Combined HDLC external and transmit clock, from transmit clock pin.
0		This bit is ignored in HDLC mode.

Serial Port Divider Low Register	(SDLR)	(Address = 0xE)
Bit(s)	**Value**	**Description**
7:0		Eight LSBs of the divider that generates the serial clock for this channel. This divider is not used unless the MSB of the corresponding SDHR is set to one.

Serial Port Divider High Register	(SDHR)	(Address = 0xF)
Bit(s)	**Value**	**Description**
7	0	Disable the serial port divider, and use the external timer input to clock the serial port.
	1	Enable the serial port divider, and use its output to clock the serial port. The serial port divider counts modulo n+1 and is clocked by the peripheral clock.
6:0		Seven MSBs of the divider that generates the serial clock for this channel.

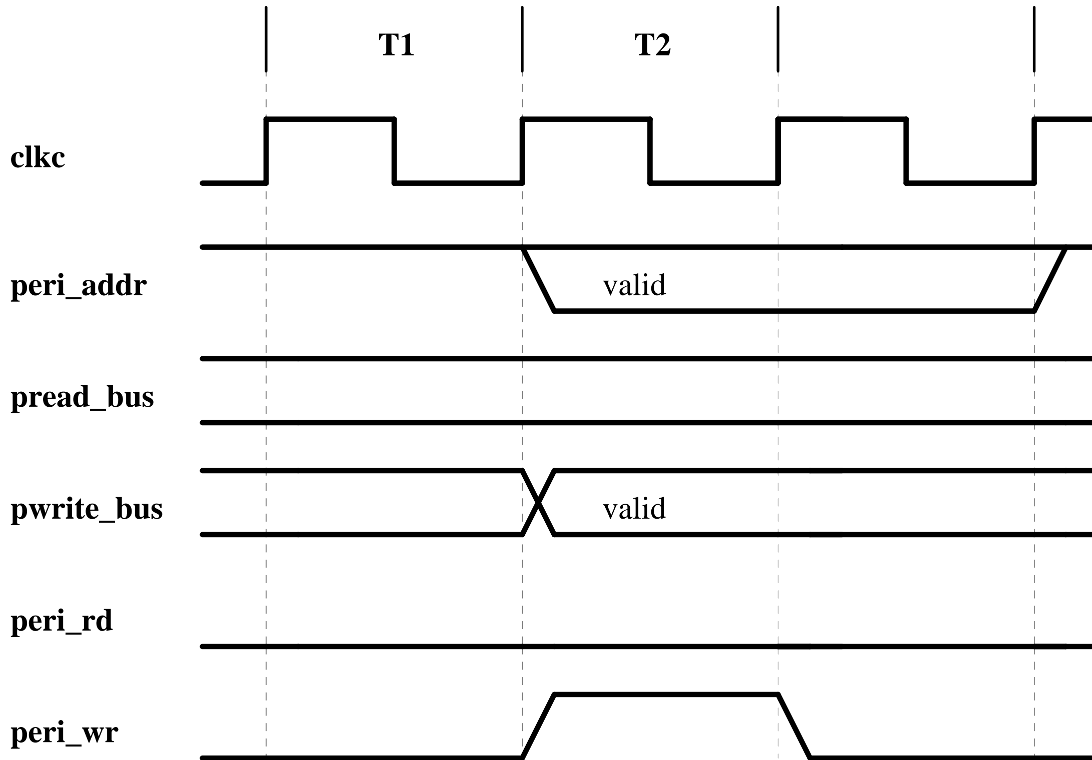
Read Transaction

Read transactions two clock cycles in length. The timing shown below is for the standard version that uses gated clocks for the control register write. If flip-flops are to be used for the control registers the peri_addr signals need one clock setup time instead of the one clock hold time shown.



Write Transaction

Write transactions two clock cycles in length. The timing shown below is for the standard version that uses gated clocks for the control register write. If flip-flops are to be used for the control registers the peri_addr and pwrite_bus signal need one clock setup time instead of the one clock hold time shown.



Top level interface

The Verilog code below shows the top-level interface when using the default bus interface.

```
module hdlc_top (dreq_rxx, dreq_txx, dreq2_txx, serx_iclk, serx_int, serx_rbus, serx_txd,
                 sreq_rxx, clkp, peri_addr, pwrite_bus, resetb, rt_sync, serx_rclki,
                 serx_rd, serx_rxdi, serx_tclki, serx_test, serx_wr);

  input          clkp;          /* main peripheral clock                                    */
  input          resetb;        /* internal reset                                           */
  input          rt_sync;       /* receiver/transmitter clock enable                        */
  input          serx_rd;       /* serial port peripheral read strobe                       */
  input          serx_test;     /* serial port x test mode                                  */
  input          serx_wr;       /* serial port peripheral write strobe                      */
  input   [2:0]  serx_rclki;    /* serial port external receive clock (hdlc mode only)      */
  input   [2:0]  serx_rxdi;     /* receiver data input                                      */
  input   [2:0]  serx_tclki;    /* serial port external transmit clock (hdlc mode only)     */
  input   [3:0]  peri_addr;     /* internal peripheral address bus                          */
  input   [7:0]  pwrite_bus;    /* internal peripheral write bus                            */
  output         dreq_rxx;      /* dma request for receiver                                 */
  output         dreq_txx;      /* dma request for transmitter                              */
  output         dreq2_txx;     /* dma request2 for transmitter                             */
  output         serx_txd;      /* transmitter data output                                  */
  output         sreq_rxx;      /* special dma request                                      */
  output  [1:0]  serx_iclk;     /* serial port clock outputs (hdlc mode only)               */
  output  [3:1]  serx_int;      /* serial port interrupt request                            */
  output  [7:0]  serx_rbus;     /* serial port peripheral read bus                          */
```

Async/HDLC Receiver

hdlc_rx

The **hdlc_rx** module is the async/hdlc receiver. This receiver contains four bytes of buffering, which allows for connection to an 8-, 16- or 32-bit bus or external FIFO. Buffer-full signals for each byte in the buffer, along with byte, word and long read strobes simplify the interface to an external bus or FIFO. Status is buffered along with each byte, and a separate HDLC frame status output bus allows frame status to be buffered separately from the data.

In Async mode the clock can be either sixteen (the default) or eight times the data rate. In HDLC mode the clock is sixteen times the data rate. Thus the maximum data rate is the peripheral clock frequency divided by eight in Async mode and divided by sixteen in HDLC mode. With the external clock option in HDLC mode the maximum rate for the bit rate is **clkp**/5 because of the synchronization logic in the clock and data paths.

In Async mode the port can send and receive seven or eight bits and has the option of recognizing an additional address bit. A status bit distinguishes normal data from “address” data. This status bit is set to one if a “zero” address bit is received. In non-address bit applications, this indicates a framing error. This status bit can also indicate a received break, if the accompanying data is all zeros (this is the definition of break).

HDLC mode encapsulates data within opening and closing Flags, and sixteen or thirty-two bits of CRC precedes the closing Flag. All information between the opening and closing Flag is "zero-stuffed". That is, if five consecutive ones occur, independent of byte boundaries, a zero is automatically inserted by the transmitter and automatically deleted by the receiver. This allows a Flag byte (07Eh) to be unique within the serial bit stream.

Both the CCITT polynomial ($x^{16}+x^{12}+x^5+1$) and Ethernet polynomial ($x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$) are available, with the generator and checker preset to all ones.

Receive operation is essentially automatic. Each byte is marked with status to indicate end-of-frame, short frame and CRC error, and a separate frame status is available with the overall status for the frame. This makes it possible to transfer all of the data in a received fram without checking the status byte-by-byte. This frame status, which includes a byte count, remains valid until the end of the next received frame.

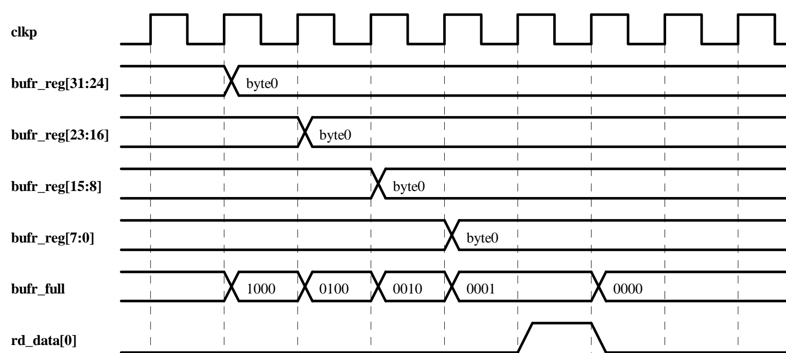
The receiver automatically synchronizes on Flag bytes and presets the CRC checker appropriately. If the current receive frame is not needed (because of an address mismatch, for example) a Flag Search command is available. This command forces the receiver to ignore the incoming data stream until another Flag is received.

Not separate receive interrupt signal is generated by the receiver. Rather, external logic can use **bufr-full** status and **bufr_add** (EOF) status to create a receive interrupt.

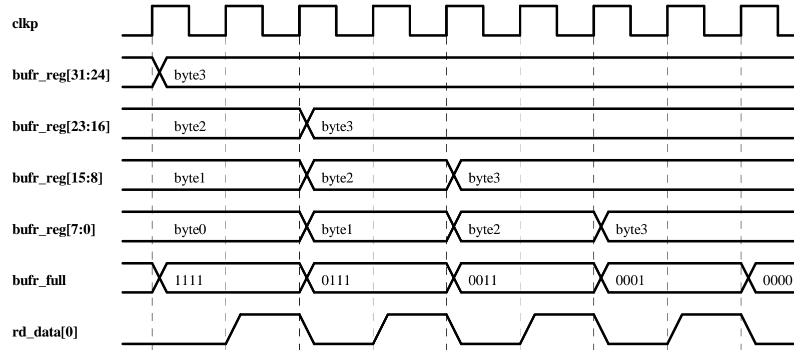
The receiver can receive frames of any bit length. If the last "byte" in the frame is not eight bits, the receiver sets a status flag that is buffered along with this last byte. Software can then use the table below to determine the number of valid data bits in this last "byte". Note that the receiver transfers all bits between the opening and closing Flags, except for the inserted zeros, to the receive data buffer.

Last byte bit pattern	Valid data bits
bbbbbbb0	7
bbbbbb01	6
bbbbb011	5
bbbb0111	4
bbb01111	3
bb011111	2
b0111111	1

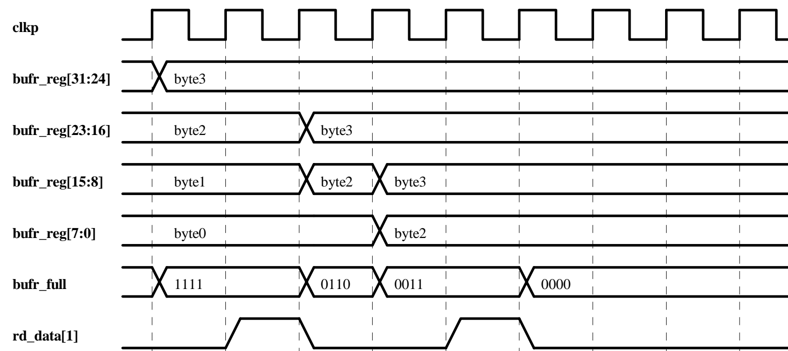
As mentioned previously, data flows through the buffer. The diagram below shows a byte being written to the buffer by the receiver and then rippling through before being read. Note that the **rd_data** signal can be one clock cycle earlier. It is shown delayed for clarity.



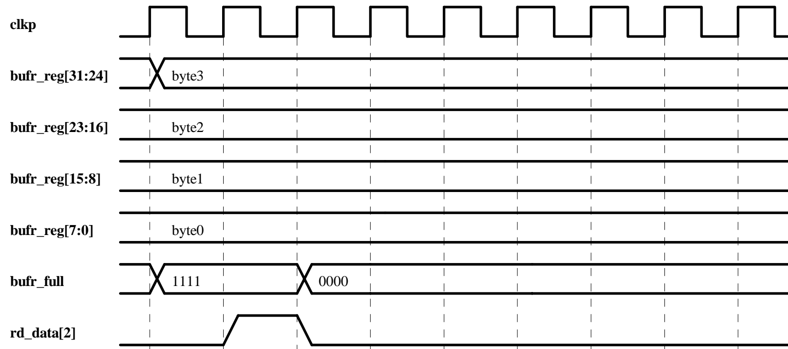
The diagram below shows a byte being written to the buffer, filling the buffer. The buffer is then emptied using four byte reads. Byte reads can have as little as one clock cycle between successive reads.



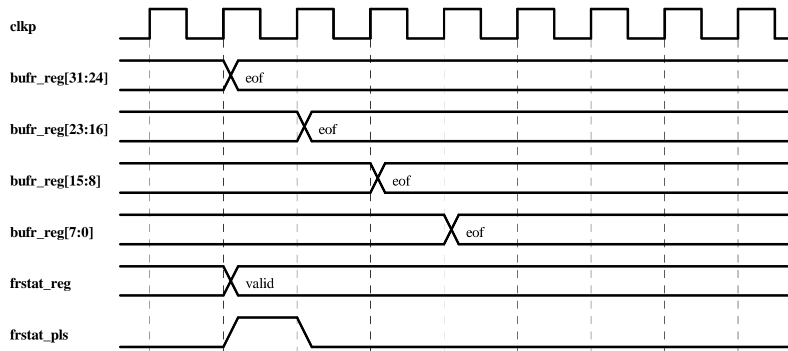
The diagram below shows a byte being written to the buffer, filling the buffer. The buffer is then emptied using two word reads. Word reads can have as little as one clock cycle between successive reads.



The diagram below shows a byte being written to the buffer, filling the buffer. The buffer is then emptied using a long read.



The diagram below shows the timing for the write of the last byte in an HDLC frame and the frame status update.



Interface

The interface signals for the **hdlc_rx** module are detailed below. All inputs except for the reset are sampled by the rising edge of the clock and all outputs change in response to the rising edge of the clock.

asyn_fast (input, active-High) The Async Fast control signal selects the clock divide ratio for Async mode and is ignored in HDLC mode. Low selects divide-by-16, while High selects divide-by-8.

bits7 (input, active-High) The 7 Bits/Character control signal selects the number of data bits per character for Async mode and is ignored in HDLC mode. Low selects 8 data bits/character, while High selects 7 data bits/character.

brk_fast (input, active-High) The Fast Break control signal selects the timing at the end of a received Break condition in Async mode and is ignored in HDLC mode. Low enables the receiver to internally force the Break to terminate on a byte boundary, while High enables the receiver to terminate the Break on any bit boundary. Selecting the byte boundary termination can lead to a lost character in the case when a Break does not end on a byte boundary and is immediately followed by data, so the **brk_fast** signal should usually be High.

brk_spec (input, active-High) The Special Break control signal selects the receiver operation during a received Break condition in Async mode and is ignored in HDLC mode. Low enables the receiver to assemble bytes and transfer them to the buffer during the Break condition, while High disables character assembly during the Break. Enabling character assembly during Break allows the duration of the Break condition to be calculated by counting the number of characters transferred.

bufr_add (output, 4-bit bus) The Address Tag bits accompany data in the data buffer, tagging each byte with the state of any corresponding Address bit (the first bit following the last data bit or the parity bit) in Async mode and marking the last byte in a frame (when High) in HDLC mode.

bufr_bit (output, 4-bit bus) The Unaligned Data Tag bits accompany data in the data buffer, indicating if the last byte in a frame does not contain eight bits. This bit is only valid for the data marked as the last in an HDLC frame. Low signals that the last data contains eight valid bits, while High signals that less than eight bits are valid. The table below shows which bits are valid in the buffer.

Last byte bit pattern	Valid data bits
bbbbbbb0	7
bbbbbb01	6
bbbbb011	5
bbbb0111	4
bbb01111	3
bb011111	2
b0111111	1

bufr_crc (output, 4-bit bus) The CRC Tag bits accompany data in the data buffer, tagging each byte with the result of the parity check (if any) in Async mode and the result of the CRC calculation in HDLC mode. Low signals no error and High signals error. For HDLC this bit should only be used for the byte tagged as the last in the frame.

bufr_full (output, 4-bit bus) The Full Tag bits accompany data in the data buffer, tagging each byte as full or empty. Low signals empty and High signals full. Note that data ripples through the buffer. Refer to the timing diagrams below for details. These bits should be used to create interrupt requests DMA requests, and write signals for an external FIFO.

bufr_ovr (output, 4-bit bus) The Overrun Tag bits accompany data in the data buffer, tagging each byte with the buffer overrun status. This status will be set when the receiver writes to the buffer when the buffer is full, resulting in the loss of data.

bufr_reg (output, 32-bit bus) This is the receive data buffer. Bytes ripple through the FIFO from most-significant byte to least-significant byte. A Byte read removes the **bufr_reg[7:0]** from the buffer, while moving any other bytes in the buffer one byte to the right. A Word read removes the **bufr_reg[15:0]** from the buffer while moving the remaining bytes one byte to the right. These bytes will move one more byte to the right on the next clock cycle. A Long read removes all four bytes from the buffer.

clkp (input, active-High) The Peripheral Clock connects to all flip-flops in the module.

crc32_en (input, active-High) The CRC32 Enable control signal selects the CRC polynomial to use in HDLC mode and is ignored in Async mode. Low selects the CCITT polynomial ($x^{16}+x^{12}+x^5+1$), while High selects the Ethernet polynomial ($x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$).

frstat_pls (output, active-High) The Frame Status Pulse signal goes High for one clock cycle at the end of a received frame to indicate that the frame status bus is valid. This pulse can be used to write the frame status to a status FIFO is necessary.

frstat_reg (output, 18-bit bus) The Frame Status bus holds the status information for the most-recently received frame, and is valid until the end of the next frame. Bit 17 is set if there was an overrun anywhere during the frame. Bit 16 is set if the frame ended with an Abort. Bit 15 is set if the frame had a CRC error. Bit 14 is set if the last byte in the frame is not a full eight bits. Bits 13-0 contain the byte count for the frame. This is the number of bytes written to the FIFO by the receiver, which will also be the number of bytes read from the FIFO in the absense of an overrun.

hdlc_mode (input, active-High) The HDLC Mode control signal selects the operating mode for the receiver. Low selects Async, while High selects HDLC.

par_en (input, active-High). The Enable Parity control signal enables the parity checker in Async mode and is ignored in HDLC mode. Parity checking automatically adds one bit to the selected charater length. Low disables parity checking and High enables parity checking.

par_sel (input, 2-bit bus) The Parity Select signals select the type of parity to check for in Async mode and are ignored in HDLC mode. Bit combination 00 selects even parity, 01 selects odd parity, 10 selects Space (always zero) parity and 11 selects Mark (always High) parity. These signals are ignored if parity is not enabled.

rd_data (input, 3-bit bus) The Read Data signals remove data from the buffer. Only one **rd_data** signal may be active at a time, and then only for one clock cycle. Because the data ripples through the buffer only byte reads every clock cycle are supported. Word reads can occur no faster than every other clock cycle, and long reads must obviously wait for the entire buffer to fill. Bit 0 is the byte read strobe, bit 1 is the word read strobe, and bit 2 is the long read strobe.

resetb (input, active-Low) The Master Reset signal is used to initialize most of the flip-flops in the design.

rx_hunt (input, active-High) The Enter Hunt signal is used to force the receiver to halt frame assembly and wait for a Flag in HDLC mode and is not used in Async mode. This signal should be active (High) for one clock cycle, and is usually used to halt reception in the case of an address mismatch.

rx_sync (input, active-High) The Receive Sync signal is the “clock” for the receiver. This signal is active at the data rate for HDLC mode and at 8x or 16x the data rate

in Async mode. If High all the time, the HDLC data rate is the **clkp** rate and the Async data rate is either **clkp**/16 or **clkp**/8. Internally the **rx_sync** signal is used as a clock-enable for flip-flops operating at the data rate.

rxd_asyn (input) The Async Receive Data signal is the data input in Async mode and is ignored in HDLC mode. The **rxd_asyn** signal is sampled in the center of the bit cell in both 8x and 16x mode.

rxd_syn (input) The Synchronous Receive Data signal is the data input in HDLC mode and is ignored in Async mode. The **rxd_syn** signal is sampled whenever the **rx_sync** signal is active. A separate data input for HDLC is used to allow for an external data decoder in the signal path.

serx_test (input, active-High) The Serial Test signal is used only for testing, and modifies the frame length counter to allow shorter test times. Low is normal mode and High is test mode.

Async/HDLC Transmitter

hdlc_tx

The **hdlc_tx** module is the Async/HDLC transmitter. This transmitter contains four bytes of buffering, which allows for connection to an 8-, 16- or 32-bit bus or external FIFO. Buffer-empty signals for each byte in the buffer, along with byte, word and long write strobes simplify the interface to an external bus or FIFO. Status is buffered along with each byte, to carry address/data information in Async mode and End-of-Frame information in HDLC mode.

In Async mode the clock can be either sixteen (the default) or eight times the data rate. In HDLC mode the clock is sixteen times the data rate. Thus the maximum data rate is the peripheral clock frequency divided by eight in Async mode and divided by sixteen in HDLC mode. With the external clock option in HDLC mode the maximum rate for the bit rate is **clkp**/5 because of the synchronization logic in the clock and data paths.

In Async mode the port can send and receive seven or eight bits and has the option of appending and recognizing an additional address bit. On transmit, the address bit is automatically appended to the data when the data is tagged with special status signals. Writing the data with **tag_addr** signal active appends a “zero” address bit to the data, while writing the data with the **tag_long** signal active appends an “one” address bit to the data. The address bit is followed by a normal stop bit. This status information is buffer along with the data.

HDLC mode encapsulates data within opening and closing Flags, and sixteen or thirty-two bits of CRC precedes the closing Flag. All information between the opening and closing Flag is "zero-stuffed". That is, if five consecutive ones occur, independent of byte boundaries, a zero is automatically inserted by the transmitter and automatically deleted by the receiver. This allows a Flag byte (07Eh) to be unique within the serial bit stream.

Both the CCITT polynomial ($x^{16}+x^{12}+x^{5}+1$) and Ethernet polynomial ($x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$) are available, with the generator preset to all ones.

Transmit operation is essentially automatic. In the transmitter, the CRC generator is preset and the opening Flag is transmitted automatically after the first byte is written to the transmitter buffer, and CRC and the closing flag are transmitted after the byte that is written to the buffer with the **tag_addr** signal active. If no CRC is required, writing the last byte of

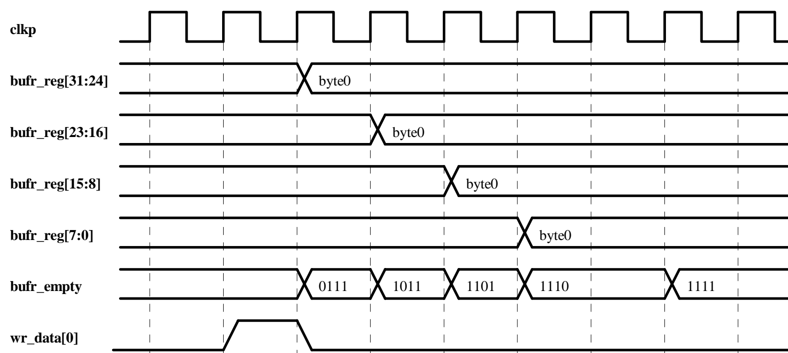
the frame with the **tag_long** signal active automatically appends a closing flag after the tagged byte. If the transmitter underflows, either an Abort or a Flag will be transmitted, under program control.

A command is available to send the Abort pattern (seven consecutive ones) if a transmit frame needs to be aborted prematurely. The Abort command takes effect on the next byte boundary, and causes the transmission of an FEh (a zero followed by seven ones), after which the transmitter will send the idle line condition. The Abort command also purges the transmit FIFO. The idle line condition may be either Flags or all ones.

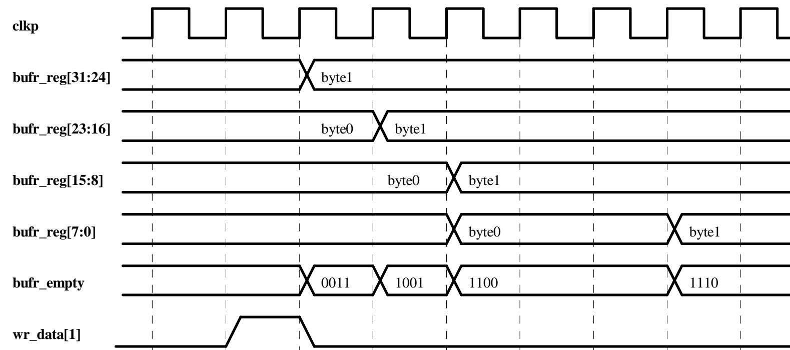
An interrupt is generated every time a byte is removed from the transmitter buffer. The transmitter also generates an interrupt at the end of CRC transmission, at the end of the transmission of an Abort sequence, and at the end of the transmission of a closing Flag.

The transmitter is not capable of sending an arbitrary number of bits, but only a multiple of bytes. Thus and idle line will always be a multiply of eigth bit times, irrespective of whether using Mark idle or Flag idle.

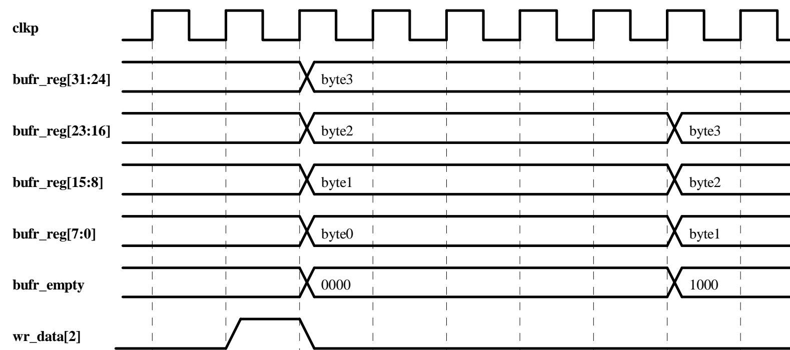
As mentioned previously, data ripples through the buffer. The diagram below shows a byte being written to the buffer and then rippling through before being taken by the transmitter. Date can be removed from the buffer by the transmitter as soon as one clock cycle after entering the least-significant byte of the buffer.



The diagram below shows word data being written to the buffer and rippling through before the first byte is taken by the transmitter.



The diagram below shows long data being written to the buffer and then the first byte being taken by the transmitter.



Interface

The interface signals for the **hdlc_tx** module are detailed below. All inputs except for the reset are sampled by the rising edge of the clock and all outputs except for the transmit data change in response to the rising edge of the clock.

asyn_fast (input, active-High) The Async Fast control signal selects the clock divide ratio for Async mode and is ignored in HDLC mode. Low selects divide-by-16, while High selects divide-by-8.

bits7 (input, active-High) The 7 Bits/Character control signal selects the number of data bits per character for Async mode and is ignored in HDLC mode. Low selects 8 data bits/character, while High selects 7 data bits/character.

bufr_empty (input, 4-bit bus) The Empty Tag bits accompany data in the data buffer, tagging each byte as full or empty. Low signals full and High signals empty. Note that data ripples through the buffer. Refer to the timing diagrams below for details. These bits should be used to create interrupt requests DMA requests, and read signals for an external FIFO.

clkp (input, active-High) The Peripheral Clock connects to all flip-flops in the module.

clr_int (input, active-High) The Clear Interrupt signal is used to clear the **txint_reg** status when no further data is to be written. This signal should go active for one clock cycle to clear the transmit interrupt.

crc32_en (input, active-High) The CRC32 Enable control signal selects the CRC polynomial to use in HDLC mode and is ignored in Async mode. Low selects the CCITT polynomial ($x^{16}+x^{12}+x^5+1$), while High selects the Ethernet polynomial ($x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$).

hdlc_mode (input, active-High) The HDLC Mode control signal selects the operating mode for the transmitter. Low selects Async, while High selects HDLC.

irda_mode (input, active-High) The IRDA Mode control signal enables special operation for the transmitter in HDLC mode and is ignored in Async mode. Low selects normal mode, with one opening Flag in a frame, while High selects the special case of two opening flags in a frame.

mrk_idl (input, active-High) The Mark Idle control signal enables special operation for the transmitter in HDLC mode and is ignored in Async mode. Low selects

normal mode, with continuous Flags transmitted between frames, while High selects the special case of continuous ones transmitted between frames.

par_en (input, active-High). The Enable Parity control signal enables the parity generatorer in Async mode and is ignored in HDLC mode. Parity generation automatically adds one bit to the selected charater length. Low disables parity generation and High enables parity generation.

par_sel (input, 2-bit bus) The Parity Select signals select the type of parity to generate in Async mode and are ignored in HDLC mode. Bit combination 00 selects even parity, 01 selects odd parity, 10 selects Space (always zero) parity and 11 selects Mark (always High) parity. These signals are ignored if parity is not enabled.

pwrite_bus (input, 32-bit bus) This is the data for the transmit data buffer. Bytes ripple through the FIFO from most-significant byte to least-significant byte. A Byte write transfer **pwrite_bus[7:0]** the msbyte of the buffer, which then ripples to the right. A Word write **pwrite_bus[15:0]** the two msbytes of the buffer which then ripple to the right. A Long write fills all four bytes of the buffer. The right-most byte of the buffer is transmitted first.

resetb (input, active-Low) The Master Reset signal is used to initialize most of the flip-flops in the design.

serx_tsync (output, active-High) The Serial Tx Sync signal is active for one clock cycle at the start of each transmit bit. This signal is used by an IRDA encoder to trigger the monostable multivibrator that does the IRDA encoding.

tag_addr (input, 4-bit bus) The Address Tag bits accompany data in the data buffer, marking a byte to be transmitted with a Zero between the last data bit or the parity bit and the Stop bit in Async mode and marking the last byte in a frame in HDLC mode. Low has no effect, while High marks the byte as “address” or “EOF”.

tag_long (input, 4-bit bus) The Long Stop Tag bits accompany data in the data buffer, marking a byte to be transmitted with a One between the last data bit or the parity bit and the Stop bit in Async mode. These signal are not used in HDLC mode. Low has not effect, while High markes the byte as “data”. When not sending an address/data bit, these signals may be tied High to always send two stop bits in Async mode.

tx_abrt (input, active-High) The Transmit Abort signal is used to force the transmitter to halt frame transmission and send a Abort (0x7F) in HDLC mode and is not used in Async mode. This signal should be active (High) for one clock cycle,

and is usually used to halt transmission because of some higher-level protocol error.

tx_data (output) The Transmit Data signal is the serial output from the transmitter.

tx_status (output, 2-bit bus) The Transmit Status signals reports the state of the transmitter. In Async mode only bit 1 is used: Low signals that the transmitter is idle and High signals that the transmitter is sending a byte. In HDLC mode bit combination 00 signals that the transmit interrupt is due to a buffer empty condition; bit combination 01 signals that the CRC transmission is complete; bit combination 10 signals that the Abort transmission is complete; and bit combination 11 signals that transmission of the closing Flag is complete. The **txint_reg** signal is set by any change on **tx_status**.

tx_sync (input, active-High) The Transmit Sync signal is the "clock" for the transmitter. This signal is active at the data rate for HDLC mode and at 8x or 16x the data rate in Async mode. If High all the time, the HDLC data rate is the **clkp** rate and the Async data rate is either **clkp**/16 or **clkp**/8. Internally the **tx_sync** signal is used as a clock-enable for flip-flops operating at the data rate.

txint_reg (output, active-High) The Transmit Interrupt signal is setwhenever a byte is removed from the transmit buffer, and when there is any change of state on a **tx_status** signal. The normal response to a transmit interrupt is to write more data to the buffer, but tha interrupt can also be cleared via the clr_int signal if no data is being transferred to the buffer.

urun_abrt (input, active-High). The Underrun Abort control signal is ignored in Async mode and is used in HDLC mode to enable the transmitter to automatically send an Abort if the transmitter underruns.

wr_data (input, 3-bit bus) The Write Data signals load data to the buffer. Only one **wr_data** signal may be active at a time, and then only for one clock cycle. Because the data ripples through the buffer only byte reads every clock cycle are supported. Word reads can occur no faster than every other clock cycle, and long reads must obviously wait for the entire buffer to fill. Bit 0 is the byte write strobe, bit 1 is the word write strobe, and bit 2 is the long write strobe.

Digital Phase-locked Loop

dpll_top

The **dpll_top** module is a digital phase-locked loop (DPLL) to recover the clock from an encoded HDLC serial data stream. It is clocked at 16x the bit rate, and generates a clock enable suitable for decoding the receive data. A fixed /16 clock enable output suitable for clocking the transmit data is also provided.

The DPLL is just a divide-by-16 counter that uses the timing of the transitions on the receive data stream to adjust its count. The DPLL adjusts the count so that the output of the DPLL will be properly placed in the bit cells to sample the receive data.

To work properly, then, transitions are required in the receive data stream. NRZ data encoding does not guarantee transitions in all cases (a long string of zeros for example), but other data encodings do. NRZI guarantees transitions because of the HDLC inserted zeros, and the Biphase encodings all have at least one transition per bit cell.

The DPLL counter normally counts by sixteen, but if a transition occurs earlier or later than expected the count will be modified during the next count cycle. If the transition occurs earlier than expected, it means that the bit cell boundaries are early with respect to the DPLL-tracked bit cell boundaries, so the count is shortened, either by one or two counts. If the transition occurs later than expected, it means that the bit cell boundaries are late with respect to the DPLL-tracked bit cell boundaries, so the count is lengthened, either by one or two counts. The decision to adjust by one or by two depends on how far off the DPLL-tracked bit cell boundaries are. This tracking allows for minor differences in the transmit and receive clock frequencies.

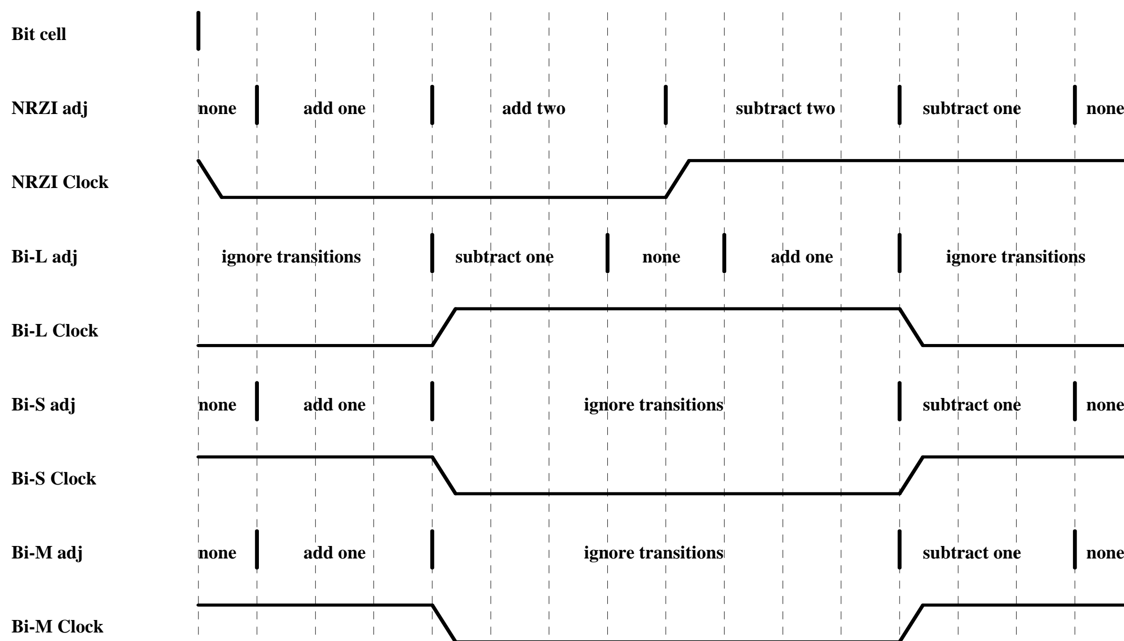
With NRZ and NRZI data encoding, the DPLL counter runs continuously, and adjusts after every receive data transition. Since NRZ encoding does not guarantee a minimum density of transitions, the difference between the sending data rate and the DPLL output clock rate must be very small, and depends on the longest possible run of zeros in the received frame. NRZI encoding guarantees at least one transition every six bits (with the inserted zeros). Since the DPLL can adjust by two counts every bit cell, the maximum difference between the sending data rate and the DPLL output clock rate is 1/48 (~2%).

With Biphase data encoding (either -Level, -Mark or -Space), the DPLL runs only as long as transitions are present in the receive data stream. Two consecutive missed transitions causes the DPLL to halt operation and wait for the next available transition. This mode of operation is necessary because it is possible for the DPLL to lock onto the optional transi-

tions in the receive data stream. Since they are optional, they will eventually not be present and the DPLL can attempt to lock onto the required transitions. Since the DPLL can adjust by one count every bit cell, the maximum difference between the sending data rate and the DPLL output clock rate is 1/16 (~6%).

With Biphase data encoding the DPLL is designed to work in multiple-access conditions where there may not be Flags on an idle line. The DPLL will properly generate an output clock based on the first transition in the leading zero of an opening Flag. Similarly, only the completion of the closing Flag is necessary for the DPLL to provide the extra two clocks to the receiver to properly assemble the data. In Biphase-Level mode, this means the transition that defines the last zero of the closing Flag. In Biphase-Mark and Biphase-Space modes this means the transition that defines the end of the last zero of the closing Flag.

The figure below shows the adjustment ranges and output clock for the different modes of operation of the DPLL. Each mode of operation will be described in turn.



With NRZ and NRZI encoding all transitions occur on bit-cell boundaries and the data should be sampled in the middle of the bit cell. If a transition occurs after the expected bit-cell boundary (but before the midpoint) the DPLL needs to lengthen the count to line up the bit-cell boundaries. This corresponds to the “add one” and “add two” regions shown. If a transition occurs before the bit cell boundary (but after the midpoint) the DPLL needs to shorten the count to line up the bit-cell boundaries. This corresponds to the “subtract one”

and “subtract two” regions shown. The DPLL makes no adjustment if the bit-cell boundaries are lined up within one count of the divide-by-sixteen counter. The regions that adjust the count by two allow the DPLL to synchronize faster to the data stream when starting up.

With Biphase-Level encoding there is a guaranteed “clock” transition at the center of every bit cell and optional “data” transitions at the bit cell boundaries. The DPLL only uses the clock transitions to track the bit cell boundaries, by ignoring all transitions occurring outside a window around the center of the bit cell. This window is half a bit-cell wide. Additionally, because the clock transitions are guaranteed, the DPLL requires that they always be present. If no transition is found in the window around the center of the bit cell for two successive bit cells the DPLL is not in lock and immediately enters the search mode. Search mode assumes that the next transition seen is a clock transition and immediately synchronizes to this transition. No clock output is provided to the receiver during the search operation. Decoding Biphase-Level data requires that the data be sampled at either the quarter or three-quarter point in the bit cell. The DPLL here uses the quarter point to sample the data.

Biphase-Mark and Biphase-Space encoding are identical as far as the DPLL is concerned, and are similar to Biphase-Level. The primary difference is the placement of the clock and data transitions. With these encodings the clock transitions are at the bit-cell boundary and the data transitions are at the center of the bit cell, and the DPLL operation is adjusted accordingly. Decoding Biphase-Mark or Biphase-Space encoding requires that the data be sampled by both edges of the recovered receive clock.

Interface

The interface signals for the **dpll_top** module are detailed below. All inputs except for the reset are sampled by the rising edge of the clock and all outputs change in response to the rising edge of the clock.

clkp (input, active-High) The Peripheral Clock connects to all flip-flops in the module.

dec_mode (input, 3-bit bus) The Decode Mode bus selects the type of data encoding that the DPLL will expect for the serial data according to the table below. Any bit combinations not shown are invalid.

dec_mode	Data encoding
000	NRZ
010	NRZI
100	Biphase-Level
110	Biphase-Space
111	Biphase-Mark

dpll_en (input, active-High) The DPLL Enable control signal enables the DPLL. While disabled the **dpll_rxmain**, **dpll_rxmid**, **dpll_txmain** and **dpll_txmid** signals are all Low. The **dpll_tclk** signal holds the last value and the **dpll_rclk** signal is Low unless the **dec_mode** is selecting biphase-mark or biphase-space.

dpll_rclk (output) The DPLL Receive Clock is a square wave (in the absense of adjustments) sutiable for external use with the receive data. Refer to the table below for the phase of this signal relative to the bit cell.

dpll_rxmain (output) The DPLL Main Rx Clock is a one clock cycle pulse at the appropriate main sampling point for the receive data. This corresponds to the rising edge of the **dpll_rclk** signal. The table below shows the location within the bit cell for the various dpll output signals.

dec_mode	Data encoding	dpll_rxmain	dpll_rxmid	dpll_txmain	dpll_txmid
000	NRZ	center	none	boundary	center
010	NRZI	center	none	boundary	center
100	Biphase-Level	1/4 point	none	boundary	center
110	Biphase-Space	3/4 point	1/4 point	boundary	center
111	Biphase-Mark	3/4 point	1/4 point	boundary	center

dpll_rxmid (output) The DPLL Middle Rx Clock is a one clock cycle pulse at the appropriate main sampling point for the receive data with Biphase-Space and Biphase-Mark encoding. These encoding methods require two samples of the data per bit cell to decode.

dpll_tclk (output) The DPLL Transmit Clock is a square wave sutiable for external use with the transmit data.

dpll_txmain (output) The DPLL Main Tx Clock is a one clock cycle pulse at the falling edge of the transmit clock for use by the transmitter.

dpll_txmid (output) The DPLL Middle Tx Clock is a one clock cycle pulse at the rising edge of the transmit clock for use by the transmitter.

resetb (input, active-Low) The Master Reset signal is used to initialize most of the flip-flops in the design.

serx_rxd (input) The Serial Receive Data signal is the encoded serial data input.

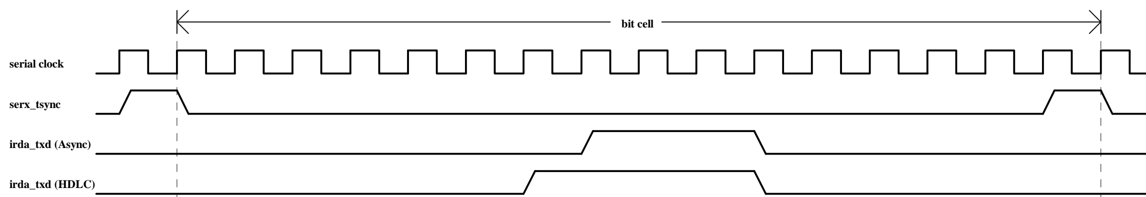
tr_sync (input, active-High) The Transmit/Receive Sync signal is the “clock” for the DPLL. This signal is pulses High for one clock cycle at 16x the data rate. If High all the time, data rate is the **clkp** rate divided by 16.

IRDA Encode/Decode

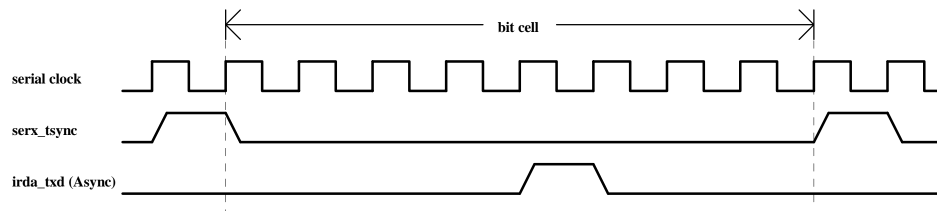
irda_top

The **irda_top** module decodes IRDA receive data into NRZ and encodes NRZ transmit data into IRDA transmit data. It is clocked at 16x the bit rate when used with HDLC and either 8x or 16x the bit rate for Async. Enabling the IRDA-compliant encode/decode modifies the transmitter in HDLC mode so that there are always two opening Flags transmitted.

The IRDA encoder sends an active-High pulse for a zero and no pulse for a one. In the asynchronous 16x mode this pulse is 3/16ths of a bit cell wide, while in the asynchronous 8x mode it is 1/8th of a bit cell wide. In HDLC mode the pulse is 1/4th of a bit cell wide. The diagram below shows the details of the timing for 16x Async and HDLC. Note that the encoded introduces a few **clkp** cycles of delay into the signal.

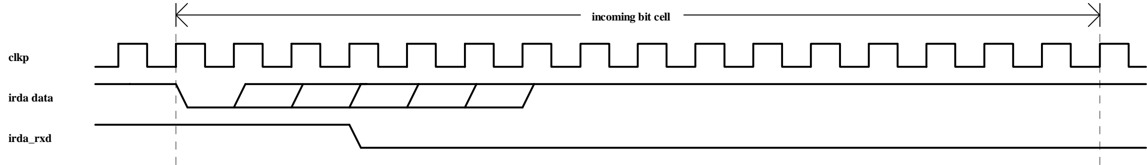


The diagrams below shows the details of the timing for the 8x Async case.



The IRDA decoder watches for active-Low pulses, which are stretched to one bit time wide to recreate the normal NRZ serial waveform for the receiver. The diagram below

shows the details of the timing for the receive data. The relative timing is identical for Async (8x and 16x) and HDLC modes.



Interface

The interface signals for the **irda_top** module are detailed below. All inputs except for the reset are sampled by the rising edge of the clock and all outputs change in response to the rising edge of the clock.

asyn_fast (input, active-High) The Async Fast control signal selects the clock divide ratio for Async mode and is ignored in HDLC mode. Low selects divide-by-16, while High selects divide-by-8.

clkp (input, active-High) The Peripheral Clock connects to all flip-flops in the module.

hdlc_mode (input, active-High) The HDLC Mode control signal selects the operating mode for the encoder and decoder. Low selects Async, while High selects HDLC.

irda_mode (input, active-High) The IRDA Mode control signal enables the encoder and decoder. Low disables the encoder and decoder while High enables them both.

irda_rxd (output) The IRDA Receive Data signal is the decoded receive data.

irda_txd (output) The IRDA Transmit Data signal is the encoded transmit data.

resetb (input, active-Low) The Master Reset signal is used to initialize most of the flip-flops in the design.

serx_rdat (input) The Serial Receive Data signal is the encoded serial data input.

serx_tsync (input, active-High) The Serial Tx Sync signal is active for one clock cycle at the start of each transmit bit. This signal is used by the encoder to trigger the monostable multivibrator that does the IRDA encoding.

tr_sync (input, active-High) The Transmit/Receive Sync signal is the “clock” for the encoder/decoder. This signal is pulses High for one clock cycle at 16x the data rate. If High all the time, data rate is the **clkp** rate divided by 16.

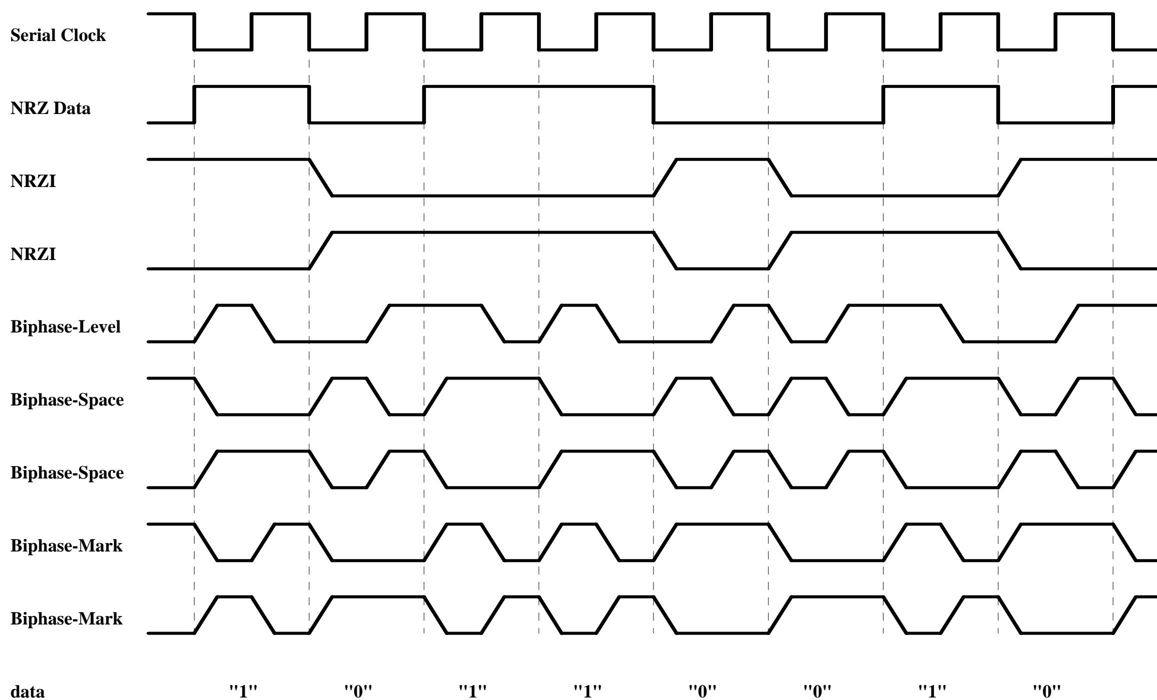
tx_data (input) The Transmit Data signal is the serial output from the transmitter.

Encoder/Decoder

endec_top

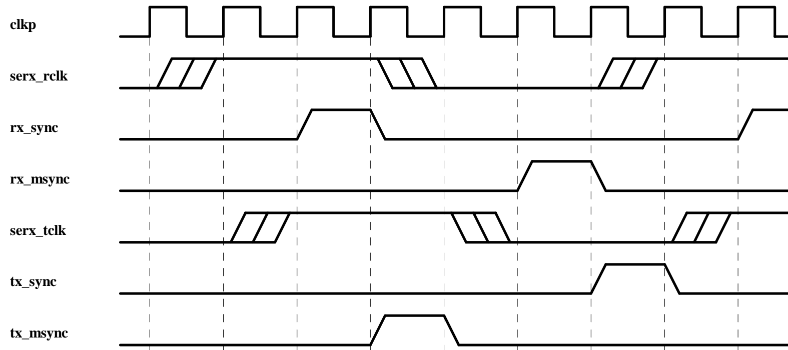
The **endec_top** module encodes and decodes the serial data. It also contains the synchronization logic required for the serial data and clock signal to cross into the **clkp** clock domain.

Several types of data encoding are available in the HDLC mode. In addition to the normal NRZ, they are NRZI, Biphase-Level (Manchester), Biphase-Space (FM0) and Biphase-Mark (FM1). Examples of these encodings are shown in the Figure below. Note that in NRZI, Biphase-Space and Biphase-Mark the signal level does not convey information. Rather it is the placement of the transitions that determine the data. In Biphase-Level it is the polarity of the transition that determines the data.



The serial clock signals from outside the hdlc_top module are all synchronized to the clkp clock domain in the **endec_top** module. These signals are the external receive clock

(**serx_rclk**) and the external transmit clock (**serx_tclk**). The diagram below shows the timing for the synchronizers. Because of this synchronization, the data rate when using data encoding and decoding is limited to **clkp**/4 for synchronous serial clocks and **clkp**/5 for asynchronous serial clocks.



The tranmsit data out of the **endec_top** module in the case of HDLC mode with an external clock uses the actual external transmit clock (**serx_tclk**) to time the transmit data. However the decoder, in the same case, does not use the external clock to sample the receive data. This can be done externally to **hdlc_top** if an edge-synchronous sampling point for the receive data is required.

Interface

The interface signals for the **endec_top** module are detailed below. All inputs except for the reset are sampled by the rising edge of the clock and almost all outputs change in response to the rising edge of the clock.

clkp (input, active-High) The Peripheral Clock connects to all flip-flops in the module.

comclk_en (input, active-High) The Common Clock Enable control signal enables the receive clock to be sourced from the transmit clock input (HDLC mode with external clock). Low enables the receive clock to be sourced from the **serx_rclk** signal, while High enables the receive clock to be sources from the **serx_tclk** signal.

dec_mode (input, 3-bit bus) The Decode Mode bus selects the type of data encoding expected for the serial receive data, according to the table below. Any bit combinations not shown are invalid.

dec_mode	Data encoding
000	NRZ
010	NRZI
100	Biphase-Level
110	Biphase-Space
111	Biphase-Mark

enc_mode (input, 3-bit bus) The Encode Mode bus selects the type of data encoding to be performed for the serial transmit data, according to the table below. Any bit combinations not shown are invalid.

enc_mode	Data encoding
000	NRZ
010	NRZI
100	Biphase-Level
110	Biphase-Space
111	Biphase-Mark

dpll_en (input, active-High) The DPLL Enable control signal enables the encoder and decoder to use the DPLL outputs as clocks for the data in HDLC mode. Low enables the external clocks for HDLC while HIgh enables the DPLL output clocks.

dpll_rxmain (output) The DPLL Main Rx Clock is a one clock cycle pulse at the appropriate main sampling point for the receive data. The table below shows the location within the bit cell for the various DPLL output signals.

dec_mode	Data encoding	dpll_rxmain	dpll_rxmid	dpll_txmain	dpll_txmid
000	NRZ	center	none	boundary	center
010	NRZI	center	none	boundary	center
100	Biphase-Level	1/4 point	none	boundary	center
110	Biphase-Space	3/4 point	1/4 point	boundary	center
111	Biphase-Mark	3/4 point	1/4 point	boundary	center

dpll_rxmid (output) The DPLL Middle Rx Clock is a one clock cycle pulse at the appropriate main sampling point for the receive data with Biphase-Space and Biphase-Mark encoding. These encoding methods require two samples of the data per bit cell to decode.

dpll_txmain (output) The DPLL Main Tx Clock is a one clock cycle pulse at the falling edge of the transmit clock for use by the transmitter.

dpll_txmid (output) The DPLL Middle Tx Clock is a one clock cycle pulse at the rising edge of the transmit clock for use by the transmitter.

hdlc_mode (input, active-High) The HDLC Mode control signal selects the operating mode for the encoder and decoder. Low selects Async, while High selects HDLC.

resetb (input, active-Low) The Master Reset signal is used to initialize most of the flip-flops in the design.

rx_sync (output, active-High) The Receive Sync signal is the “clock” for the receiver. This signal is active at the data rate for HDLC mode and at 8x or 16x the data rate in Async mode. If High all the time, the HDLC data rate is the **clkp** rate and the Async data rate is either **clkp**/16 or **clkp**/8.

rxd_syn (output) The Synchronous Receive Data signal is the decoded serial data output in HDLC mode and is undefined in Async mode.

serx_rxd (input) The Serial Receive Data signal is the encoded serial data input.

serx_rclk (input) The Serial Receive Clock signal is the external receive clock for use in HDLC mode.

serx_tclk (input) The Serial Transmit Clock signal is the external transmit clock for use in HDLC mode.

serx_tdat (output) The Serial Transmit Data signal is the external transmit clock for use in HDLC mode.

tr_sync (input, active-High) The Transmit/Receive Sync signal is the “clock” for the encoder and decoder. This signal is pulses High for one clock cycle at 16x the data rate. If High all the time, data rate is the **clkp** rate divided by 16.

tx_data (input) The Transmit Data signal is the serial output from the transmitter.

tx_sync (input, active-High) The Transmit Sync signal is the “clock” for the transmitter. This signal is active at the data rate for HDLC mode and at 8x or 16x the data rate in Async mode. If High all the time, the HDLC data rate is the **clkp** rate and the Async data rate is either **clkp**/16 or **clkp**/8.

Top-level Verilog

hdlc_top

The Verilog code for the **hdlc_top** module is shown below to illustrate how the individual modules connect.

```
/*****************************************************************************************/
/**                                                                                     **/
/** COPYRIGHT (C) 2010, Systemyde International Corporation, ALL RIGHTS RESERVED         **/
/**                                                                                     **/
/** async/hdlc                                                       Rev 0.0  05/25/2010 **/
/**                                                                                     **/
/*****************************************************************************************/
module hdlc_top (dreq_rxx, dreq_txx, dreq2_txx, serx_iclk, serx_int, serx_rbus, serx_txd,
                 sreq_rxx, clkp, peri_addr, pwrite_bus, resetb, rt_sync, serx_rclki,
                 serx_rd, serx_rxdi, serx_tclki, serx_test, serx_wr);

  input         clkp;         /* main peripheral clock                                    */
  input         resetb;       /* internal reset                                           */
  input         rt_sync;      /* receiver/transmitter clock enable                        */
  input         serx_rd;      /* serial port peripheral read strobe                       */
  input         serx_test;    /* serial port x test mode                                  */
  input         serx_wr;      /* serial port peripheral write strobe                      */
  input   [2:0] serx_rclki;   /* serial port external receive clock (hdlc mode only)      */
  input   [2:0] serx_rxdi;    /* receiver data input                                      */
  input   [2:0] serx_tclki;   /* serial port external transmit clock (hdlc mode only)     */
  input   [3:0] peri_addr;    /* internal peripheral address bus                          */
  input   [7:0] pwrite_bus;   /* internal peripheral write bus                            */
  output        dreq_rxx;     /* dma request for receiver                                 */
  output        dreq_txx;     /* dma request for transmitter                              */
  output        dreq2_txx;    /* dma request2 for transmitter                             */
  output        serx_txd;     /* transmitter data output                                  */
  output        sreq_rxx;     /* special dma request                                      */
  output  [1:0] serx_iclk;    /* serial port clock outputs (hdlc mode only)               */
  output  [3:1] serx_int;     /* serial port interrupt request                            */
  output  [7:0] serx_rbus;    /* serial port peripheral read bus                          */

  /***************************************************************************************/
  /*                                                                                     */
  /* signal declarations                                                                 */
  /*                                                                                     */
  /***************************************************************************************/
  wire       asyn_fast;                           /* async 8x enable               */
  wire       bits7;                               /* seven data bit mode           */
  wire       brk_fast;                            /* fast break term enable        */
  wire       brk_spec;                            /* disable data wr during brk    */
  wire       clr_int;                             /* clear interrupt (w/o data)    */
  wire       comclk_en;                           /* common rx/tx clock enable     */
  wire       dpll_en;                             /* dpll enable                   */
  wire       dpll_rxmain;                         /* dpll rx main sample pulse     */
  wire       dpll_rxmid;                          /* dpll rx min sample pulse      */
  wire       dpll_txmain;                         /* dpll tx main sample pulse     */
  wire       dpll_txmid;                          /* dpll tx min sample pulse      */
  wire       dreq_rxx;                            /* dma request for receiver      */
  wire       dreq_txx;                            /* dma request for transmitter   */
  wire       dreq2_txx;                           /* dma request2 for transmitter  */
  wire       hdlc_mode;                           /* hdlc mode                     */
  wire       irda_mode;                           /* enable irda encode/decode     */
  wire       irda_rxd;                            /* irda-decoded rx data          */
  wire       irda_txd;                            /* irda-encoded tx data          */
  wire       mrk_idl;                             /* mark idle enable (hdlc)       */
```

```
wire        par_en;                                        /* parity enable              */
wire        rd_data;                                       /* read data register         */
wire        rx_hunt;                                       /* rcvr hunt command          */
wire        rx_sync;                                       /* internal rx clock          */
wire        rxd_syn;                                       /* decoded rx data            */
wire        ser_rxd;                                       /* rcvr data input (enabled)  */
wire        serx_rclk;                                     /* ext rx clock (hdlc)        */
wire        serx_rdat;                                     /* receiver pin input         */
wire        serx_tclk;                                     /* ext tx clock (hdlc)        */
wire        serx_tdat;                                     /* encoded tx data            */
wire        serx_tsync;                                    /* transmitter sync output    */
wire        serx_txd;                                      /* transmitter data output    */
wire        sreq_rxx;                                      /* dma special request        */
wire        tag_addr;                                      /* "address" byte or eof tag  */
wire        tag_long;                                      /* "long stop" tag            */
wire        tr_sync;                                       /* internal clock             */
wire        tx_abrt;                                       /* xmtr abort command         */
wire        tx_data;                                       /* xmtr raw data out          */
wire        tx_sync;                                       /* internal tx clock          */
wire        txint_reg;                                     /* xmtr int req               */
wire        urun_abrt;                                     /* abort on underrun (hdlc0   */
wire        wr_data;                                       /* data write                 */
wire  [1:0] par_sel;                                       /* parity type select         */
wire  [1:0] serx_iclk;                                     /* internal clock output      */
wire  [1:0] tx_status;                                     /* xmtr status                */
wire  [2:0] dec_mode;                                      /* data decode mode           */
wire  [3:0] bufr_add;                                      /* rcvr buffer is special     */
wire  [3:0] bufr_bit;                                      /* rcvr buffer short (hdlc)   */
wire  [3:0] bufr_crc;                                      /* rcvr crc err (hdlc)        */
wire  [3:0] bufr_empty;                                    /* xmtr buffer empty          */
wire  [3:0] bufr_full;                                     /* rcvr buffer full           */
wire  [3:0] bufr_ovr;                                      /* rcvr buffer overrun        */
wire  [3:1] serx_int;                                      /* interrupt request bus      */
wire  [7:0] serx_rbus;                                     /* read data bus              */
wire [31:0] bufr_reg;                                      /* rcvr buffer reg            */

/******************************************************************************************/
/*                                                                                        */
/* rabbit interface                                                                       */
/*                                                                                        */
/******************************************************************************************/
rab_if RABIF    ( .asyn_fast(asyn_fast), .bits7(bits7), .brk_fast(brk_fast),
                  .brk_spec(brk_spec), .clr_int(clr_int), .comclk_en(comclk_en),
                  .dec_mode(dec_mode), .dpll_en(dpll_en), .dreq_rxx(dreq_rxx),
                  .dreq_txx(dreq_txx), .dreq2_txx(dreq2_txx), .hdlc_mode(hdlc_mode),
                  .irda_mode(irda_mode), .mrk_idl(mrk_idl), .par_en(par_en),
                  .par_sel(par_sel), .rd_data(rd_data), .rx_hunt(rx_hunt),
                  .serx_int(serx_int), .serx_rbus(serx_rbus), .serx_rclk(serx_rclk),
                  .serx_rdat(serx_rdat), .serx_tclk(serx_tclk), .sreq_rxx(sreq_rxx),
                  .tag_addr(tag_addr), .tag_long(tag_long), .tr_sync(tr_sync),
                  .tx_abrt(tx_abrt), .urun_abrt(urun_abrt), .wr_data(wr_data),
                  .bufr_add(bufr_add[0]), .bufr_bit(bufr_bit[0]), .bufr_crc(bufr_crc[0]),
                  .bufr_empty(bufr_empty), .bufr_full(bufr_full), .bufr_ovr(bufr_ovr[0]),
                  .bufr_reg(bufr_reg[7:0]), .clkp(clkp), .peri_addr(peri_addr),
                  .pwrite_bus(pwrite_bus), .resetb(resetb), .rt_sync(rt_sync),
                  .serx_rclki(serx_rclki), .serx_rd(serx_rd), .serx_rxdi(serx_rxdi),
                  .serx_tclki(serx_tclki), .serx_test(serx_test), .serx_wr(serx_wr),
                  .tx_status(tx_status), .txint_reg(txint_reg) );

/******************************************************************************************/
/*                                                                                        */
/* data encode/decode & clock mux                                                         */
/*                                                                                        */
/******************************************************************************************/
endec_top ENDEC ( .rx_sync(rx_sync), .rxd_syn(rxd_syn), .serx_tdat(serx_tdat),
                  .tx_sync(tx_sync), .clkp(clkp), .comclk_en(comclk_en),
                  .dec_mode(dec_mode), .dpll_en(dpll_en), .dpll_rxmain(dpll_rxmain),
                  .dpll_rxmid(dpll_rxmid), .dpll_txmain(dpll_txmain),
                  .dpll_txmid(dpll_txmid), .enc_mode(dec_mode), .hdlc_mode(hdlc_mode),
                  .resetb(resetb), .ser_rxd(ser_rxd), .serx_rclk(serx_rclk),
                  .serx_tclk(serx_tclk), .tr_sync(tr_sync), .tx_data(tx_data) );

/******************************************************************************************/
/*                                                                                        */
/* digital phase-locked loop                                                              */
/*                                                                                        */
/******************************************************************************************/
dpll_top DPLL   ( .dpll_rclk(serx_iclk[1]), .dpll_rxmain(dpll_rxmain),
```

```
                  .dpll_rxmid(dpll_rxmid), .dpll_tclk(serx_iclk[0]),
                  .dpll_txmain(dpll_txmain), .dpll_txmid(dpll_txmid), .clkp(clkp),
                  .dec_mode(dec_mode), .dpll_en(dpll_en), .resetb(resetb),
                  .ser_rxd(ser_rxd), .tr_sync(tr_sync) );

/*******************************************************************************************/
/*                                                                                         */
/* irda decode and encode                                                                  */
/*                                                                                         */
/*******************************************************************************************/
assign ser_rxd   = (irda_mode) ? irda_rxd : serx_rdat;
assign serx_txd  = (irda_mode) ? irda_txd : serx_tdat;

irda_top IRDA   ( .irda_rxd(irda_rxd), .irda_txd(irda_txd), .asyn_fast(asyn_fast),
                  .clkp(clkp), .hdlc_mode(hdlc_mode), .irda_mode(irda_mode),
                  .resetb(resetb), .serx_rdat(serx_rdat), .serx_tsync(serx_tsync),
                  .tr_sync(tr_sync), .tx_data(tx_data) );

/*******************************************************************************************/
/*                                                                                         */
/* serial receiver                                                                         */
/*                                                                                         */
/*******************************************************************************************/
hdlc_rx RCVR    ( .bufr_add(bufr_add), .bufr_bit(bufr_bit), .bufr_crc(bufr_crc),
                  .bufr_full(bufr_full), .bufr_ovr(bufr_ovr), .bufr_reg(bufr_reg),
                  .frstat_pls(), .frstat_reg(), .asyn_fast(asyn_fast), .bits7(bits7),
                  .brk_fast(brk_fast), .brk_spec(brk_spec), .clkp(clkp), .crc32_en(1'b0),
                  .hdlc_mode(hdlc_mode), .par_en(par_en), .par_sel(par_sel),
                  .rd_data({2'b0, rd_data}), .resetb(resetb), .rx_hunt(rx_hunt),
                  .rx_sync(rx_sync), .rxd_asyn(ser_rxd), .rxd_syn(rxd_syn),
                  .serx_test(serx_test) );

/*******************************************************************************************/
/*                                                                                         */
/* serial transmitter                                                                      */
/*                                                                                         */
/*******************************************************************************************/
hdlc_tx XMTR    ( .bufr_empty(bufr_empty), .serx_tsync(serx_tsync), .tx_data(tx_data),
                  .tx_status(tx_status), .txint_reg(txint_reg), .asyn_fast(asyn_fast),
                  .bits7(bits7), .clkp(clkp), .clr_int(clr_int), .crc32_en(1'b0),
                  .hdlc_mode(hdlc_mode), .irda_mode(irda_mode), .mrk_idl(mrk_idl),
                  .par_en(par_en), .par_sel(par_sel), .pwrite_bus({24'h0, pwrite_bus}),
                  .resetb(resetb), .tag_addr({4{tag_addr}}), .tag_long({4{tag_long}}),
                  .tx_abrt(tx_abrt), .tx_sync(tx_sync), .urun_abrt(urun_abrt),
                  .wr_data({2'b0, wr_data}) );

endmodule
```