10Base-T Interface

Features

Full support for 10Base-T, full- or half-duplex

Includes 802.3.28 auto-negotiation function

Sixteen bytes of buffering for transmit, thirty-two bytes for receive

No external PHY required - only two LVDS receivers plus passive components needed

Includes filters for both physical and multicast address match

DMA interface for both transmit and receive

Standard byte-wide interface

Description

The network port implements all of the required digital elements of the 10BASE-T standard and will normally be used with two channels of the DMA controller. The receiver contains thirty-two bytes of buffering and the transmitter contain sixteen bytes of buffering. The network port can operate in either half-duplex or full-duplex mode, selected via Auto-negotiation.

The network port transmitter automatically precedes the transmit data with a preamble and start-frame-delimiter, and appends CRC and the end-frame-delimiter after the last byte. Frame transmission starts automatically once the transmit FIFO is full and any interframe gap time or back-off time has expired. Transmission is aborted if a collision is detected and is retried up to sixteen times, using the standard random back-off time algorithm. Detection of a collision causes the transmitter to send a 32-bit “jam” pattern, of all ones, to guarantee that all receivers in the network recognize the collision. The transmitter uses the ten most-significant bits of the CRC checker, starting with bit 22 and increasing, to generate the initial seed for the back-off algorithm. Collisions that occur later than one slot time (512 bit times) are reported as late collisions but are otherwise treated identically to “normal” collisions. If transmission is not successful after sixteen attempts, the transmitter halts and reports the failure via an interrupt. The transmitter guarantees the 9.6uS inter-frame gap and implements the fair access algorithm within the inter-frame gap. The transmitter automatically sends link test pulses, even while otherwise disabled, every 16.0mS. The transmitter contains a jabber timer, which automatically disables the transmitter after 26.2mS of continuous transmission. This error condition generates an interrupt and must be answered by resetting the network port. The corresponding DMA channel is automatically halted by this error condition and must be restarted after the network port has been reset.

The network port receiver uses the received preamble to synchronize to the phase of the incoming frame and then waits for the start-frame-delimiter. Character assembly begins at this point and each byte is transferred to the receive FIFO. However, no interrupt or DMA request will occur until after the first six bytes of the frame have been received and checked for an address match. The receiver can receive frames independent of the address (promiscuous mode) or frames with a physical address match, a broadcast address match, or a multicast address match. A physical address match requires the received frame address to be a physical address (lsb of the address is zero) that matches every bit of the programmed receive address. A broadcast address match requires that all forty-eight bits of the received frame address be “ones”. A multicast address match requires the received frame address to be a multicast address (lsb of the address is one) and a match in the multicast address filter. The multicast address filter uses the six most significant bits of the CRC calculated on the receive address as an index into a 64-by-1 bit table written under program control. A one in the corresponding table entry constitutes a multicast address match as far as the network port is concerned.

Normal DMA transfers of data begin once an address match occurs, and continue until the end-frame-delimiter is recognized or the line goes idle because of a collision. The network receiver calculates the CRC across the entire frame in parallel with character assembly and reports the result when the end-frame-delimiter is recognized. Normally frames with bad CRC are discarded. The receiver also reports misaligned end-frame-delimiters (those that do not occur on byte boundaries).

To help with handling high-level protocols such as TCP/IP, the network port receiver accumulates a 16-bit checksum across the entire received frame except for the first fourteen bytes. The first fourteen bytes are the destination address field (six bytes), source address field (six bytes) and the frame length field (two bytes), which are not part of the TCP/IP payload. This checksum is initialized to all zeros during the address compare time, and then each pair of bytes is added to the checksum, with the carry from the previous add carried to the following add. The first-received byte adds to the lower byte of the checksum and the second-received byte adds to the upper byte of the checksum. In the case of an odd length frame, the second-received byte value is filled with zeros for the 16-bit add. The checksum at the end of the frame is transferred a holding register so that it can be read by software.

The network port implements the NLP receive link integrity test state machine, which requires link integrity pulses to be detected at certain intervals in the absence of other network activity.

The network port implements the auto-negotiation algorithm to determine half-duplex or full-duplex operation. This feature can be disabled or commanded to execute under software control, in addition to its normal automatic operation.

Interface

```
module ethr_top (clk10k_reg, clk10r_reg, clk10t_reg, clk20_raw, dabrt_erxx, dabrt_etxx,
                 dhalt_etxx, dreq_erxx, dreq_etxx, dreq2_etxx, ethx_int, ethx_rbus, ethx_txd,
                 link_actb, sreq_erxx, clk10k, clk10r, clk10t, clk20, clke, clkp, ethx_in,
                 ethx_rd, ethx_test, ethx_wr, lnkauto_test, peri_addr, pwrite_bus, resetb);

  input         clk10k;        /* slow clock (102.4uS period)                              */
  input         clk10r;        /* 10MHz receive clock                                      */
  input         clk10t;        /* 10MHz transmit clock                                     */
  input         clk20;         /* ethernet 20MHz clock                                     */
  input         clke;          /* dedicated input 20MHz clock                              */
  input         clkp;          /* main peripheral clock                                    */
  input         ethx_rd;       /* ethernet peripheral read strobe                          */
  input         ethx_test;     /* ethernet port x test mode                                */
  input         ethx_wr;       /* ethernet peripheral write strobe                         */
  input         lnkauto_test;  /* link autoneg test mode                                   */
  input         resetb;        /* internal reset                                           */
  input   [1:0] ethx_in;       /* ethernet input bus                                       */
  input   [5:0] peri_addr;     /* internal peripheral address bus                          */
  input   [7:0] pwrite_bus;    /* ethernet peripheral write bus                            */
  output        clk10k_reg;    /* slow clock (102.4uS period) predrive                     */
  output        clk10r_reg;    /* 10MHz receive clock predrive                             */
  output        clk10t_reg;    /* 10MHz transmit clock predrive                            */
  output        clk20_raw;     /* clk20 predrive                                           */
  output        dabrt_erxx;    /* dma abort for receiver                                   */
  output        dabrt_etxx;    /* dma abort for transmitter                                */
  output        dhalt_etxx;    /* dma halt for transmitter                                 */
  output        dreq_erxx;     /* dma request for receiver                                 */
  output        dreq_etxx;     /* dma request for transmitter                              */
  output        dreq2_etxx;    /* dma request2 for transmitter                             */
  output        sreq_erxx;     /* special dma request                                      */
  output  [3:0] ethx_txd;      /* tx data output                                           */
  output  [1:0] link_actb;     /* led status control                                       */
  output  [3:1] ethx_int;      /* ethernet interrupt request                               */
  output  [7:0] ethx_rbus;     /* ethernet peripheral read bus                             */
```