Using the Microprocessor and UTOPIA Interface of the CY7C955

Introduction

The CY7C955 is an ATM transceiver (PHY layer) which encapsulates and de-encapsulates an ATM cell from a serial SONET data stream. ATM data is exchanged between the ATM layer and the CY7C955 (PHY layer) through the UTOPIA interface. The UTOPIA flow control behavior of the CY7C955 can be programmed to suit different needs.

The CY7C955 provides the user with flexibility to change some of the values in the SONET overhead. Also, the device will keep statistics on error conditions as well as ATM cells sent and received. To access these statistic values and the overhead parameters, the CY7C955 is equipped with 56 internal registers which can be accessed through the microprocessor interface. The CY7C955 can also be programmed through the microprocessor interface to generate interrupt on error conditions.

The first part of this application note will concentrate on the operation and design consideration of the UTOPIA interface. The second part of the application note will discuss interrupt enabling and handling. The last section will show the procedure on how to program the CY7C955 to work in an STM-1 network.

UTOPIA Interface

The UTOPIA interface is defined by the ATM forum for moving data between the physical layer and ATM layer in the ATM protocol stack. The signals used by the UTOPIA interface are shown in *Figure 1*. The CY7C955's UTOPIA interface is compliant with the UTOPIA level 1 standard. From this point on, "transmit side" describes the interface where data flows from the ATM layer to the CY7C955; and "receive side" describes the interface where data flows from the CY7C955 to the ATM layer.

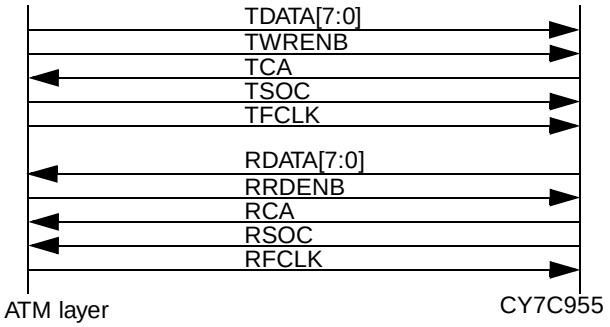

Figure 1.

The UTOPIA specification defines two types of handshaking between the ATM layer and PHY layer; octet-level handshaking and cell-level handshaking. The CY7C955 implements octet-level and cell-level handshaking scheme.

UTOPIA Transmit

On the transmit side, the period of time which the PHY layer can store data is called the transmit window. The PHY layer stores data from TDATA[7:0] on the LOW-to-HIGH transition of TFCLK, when TWRENB is asserted and the transmit window is open. The start of a transmit window is defined by the CY7C955 asserting TCA. TCA is asserted when the transmit FIFO is not full and at least one complete cell can be written in. TCA deassertion timing is programmable to provide enough time for higher layer devices to aware of the FIFO full situation before the transmit FIFO is actually full or it can be used to control the depth of the transmit FIFO. In some applications where the latency through the Transmit ATM PHY interface, i.e. the CY7C955, must be minimized, therefore control of the depth of the FIFO is important. The FIFODP[1:0] bits in the Transmit ATM Cell FIFO Control Register, address 0x63, specifies at which FIFO depth TCA is deasserted. The depth that can be selected is shown in *Table 1*. TCA will not be asserted again until a complete cell is clocked out of the transmit FIFO, therefore the current cell being clocked out is accounted for one cell in the FIFO. For this reason, the user is not advised to set the depth to one cell, because if a cell is written to the FIFO only when TCA is asserted, the next cell being transmitted is not queued up for transmission while the current cell is being transmitted, therefore at least half of the available bandwidth is lost to idle cells depending on the UTOPIA bus clock rate. It is important to observe that the programming of the FIFODP[1:0] only affects the TCA behavior, subsequent writes to the transmit FIFO will still be stored until the FIFO is four cells full.

Table 1. FIFO Depth Selection

FIFODP[1]	FIFODP[0]	FIFO Depth
0	0	4 cells
0	1	3 cells
1	0	2 cells
1	1	1 cell

In addition to being able to control the FIFO depth, the CY7C955 can also be programmed to deassert TCA when the FIFO is 4 bytes from N cell full, where N is determined by FIFODP[1:0]. This feature is controlled by the TCALEVEL0 bit of the TACP Control Register. TCA is default to deassert within 4 bytes from N cell full, it can be programmed to deassert until the last byte of the N^{th} cell is written to the FIFO by setting the FCALEVEL0 bit to a logic one. Such "early warning" flag is useful for not causing the transmit FIFO to overrun and lose data. The polarity of TCA is default to active HIGH; by setting the TCAINV bit of the Master Configuration Register (0x01) the polarity of TCA is changed to active LOW.

UTOPIA Receive

Similarly on the receive side, the read window is considered open when the RRDENB signal is asserted, which means that the ATM layer is ready to accept any data from the CY7C955. RDATA[7:0], RXPRTY and RSOC are updated on the rising edge of RFCLK when the receive window is open. The CY7C955 asserts RCA when at least one complete cell is available in the receive FIFO. By default, the CY7C955 will deassert RCA when there is no cell data left in the receive FIFO. Like the TCA signal, RCA can be programmed to deassert when there are only 4 bytes left in the receive FIFO by clearing the RCALEVEL0 bit of the Receive ATM Cell Processor (RACP) Control Register (0x59). The active state of the RCA signal is default to be active high; by setting the RCAINV bit of the Master Configuration Register (0x01), the polarity of RCA is changed to active Low. Unlike the TCA signal, the depth of the receive FIFO cannot be controlled.

Microprocessor Interface

The CY7C955 internal registers can be accessed through the microprocessor interface. To access the internal registers, the management entity, e.g., the Segmentation And Reassembly (SAR) unit, microprocessor, or micro-controller, will provide the address of the internal register to be accessed, and then data is put onto the 8-bit data bus. The internal register structure of the CY7C955 can be used to control all function modes, to report status and statistics, and to enable as well as to acknowledge interrupts. In the following section ALE usage and two software issues: Interrupt servicing and SONET protocol conversion, will be discussed.

Internal Address Latch

The CY7C955 is equipped with an internal address latch for multiplex address/data bus, which is very common on low cost, low pin count microprocessor and micro-controller. When utilizing a multiplexed address/data bus, the ALE signal is used to latched in the address. The address latch is transparent when ALE is asserted HIGH, the address is latched on the falling edge of ALE. When the address bus and data bus are not shared, then ALE can be left unconnected because there is an integral pull-up resistor at the pin.

Servicing Interrupts

The microprocessor interface includes an interrupt output (INTB) which can be enabled for certain error conditions. All interrupts are disabled after the CY7C955 is reset. To enable an interrupt, the corresponding interrupt register bit has to be set. *Table 2* summarizes the interrupts that can be enabled and the corresponding registers to program. After an interrupt is asserted the corresponding interrupt status register has to be read to clear the interrupt. Table 3 summarizes the interrupt status registers for each interrupt type. The first step to acknowledge an interrupt is to read the Master Interrupt Status Register (0x02), which will give a high-level indication of which type of interrupt is pending. This read access will clear the first level interrupt bits. After defining which type of interrupt is pending, then the individual interrupt status registers can be accessed to acknowledge the interrupt. Appendix A is a program fragment which illustrates how to acknowledge pending interrupts. It is provided as a reference only.

Table 2. Interrupt Enable Summary

Interrupt	Description	Register Name and Bit	Address
LCDE	Change in Loss of Cell Delineation	Master Control Register[7]	0x05
TROOLE	Transmit Reference Out of Lock	Transmit Clock Synthesis Control Register[1]	0x06
RDOOLE	Receive Reference Out of Lock	Receive Clock Synthesis Control Register[1]	0x07
TXFIFOE	Transmit FIFO overflow or misplaced TSOC	TACP Control and Status Register[7]	0x60
TXPRTYPE	Transmit Parity error	TACP FIFO Control Register[6]	0x63
RXFIFOE	Receive FIFO overflow	RACP Interrupt Register[5]	0x51
HCSE	HCS error from receive cell	RACP Interrupt Register[6]	0x51
OOCDE	Out of Cell Delineation	RACP Interrupt Register[7]	0x51
PSLE	Change in Path Signal Label	RPOP Interrupt Enable Register[7]	0x33
PLOPE	Loss of Pointer	RPOP Interrupt Enable Register[5]	0x33
PAISE	Path Alarm Indication Signal	RPOP Interrupt Enable Register[3]	0x33
PRDIE	Path Remote Defect Indication	RPOP Interrupt Enable Register[2]	0x33
PBIPEE	Path BIP-8 error	RPOP Interrupt Enable Register[1]	0x33
PFEBEE	Path Far End Block Error	RPOP Interrupt Enable and Status Register[0]	0x33
LFEBEE	Line Far End Block Error	RLOP Interrupt Enable and Status Register[7]	0x19
LBIPEE	Line BIP-24 error	RLOP Interrupt Enable Status Register[6]	0x19
LAISE	Line Alarm Indication Signal	RLOP Interrupt Enable Status Register[5]	0x19
LRDIE	Line Remote Defect Indication	RLOP Interrupt Enable Status Register[4]	0x19
SBIPEE	Section BIP-24 error	RSOP Control and Interrupt Enable Register[3]	0x10
SLOSE	Loss of Signal	RSOP Control and Interrupt Enable Register[2]	0x10
SLOFE	Loss of Frame	RSOP Control and Interrupt Enable Register[1]	0x10
SOOFE	Out of Frame	RSOP Control and Interrupt Enable Register[0]	0x10

Table 3. Interrupt Acknowledge Reference Summary

Interrupt	Register Name and Bit	Address	Corresponding Interrupt Enable
LCDI	Master Interrupt Register[6]	0x02	LCDE
TROOLI	Master Interrupt Register[7]	0x02	TROOLE
RDOOLI	Master Interrupt Register[5]	0x02	RDOOLE
TACPI	Master Interrupt Register[4]	0x02	TXFIFOE, TXPRTYE
RACPI	Master Interrupt Register[3]	0x02	RXFIFOE, HCSE, OOCDE
PPOPI	Master Interrupt Register[2]	0x02	PSLE, PLOPE, PAISE, PRDIE, PBIPEE,PFEBEE
RLOPI	Master Interrupt Register[1]	0x02	LFEBEE, LBIPEE, LAISE, LRDIE
RSOPI	Master Interrupt Register[0]	0x02	SBIPEE, SLOSE, SLOFE, SOOFE
TXFOVRI	TACP Control and Status Register[7]	0x60	TXFIFOE
TSOCI	TACP Control and Status Register[6]	0x60	TXFIFOE
TXPRTYPI	TACP FIFO Control Register[4]	0x63	TXPRTYPE
RXFOVRI	RACP Interrupt Register[1]	0x51	RXFIFOE
CHCSI	RACP Interrupt Register[3]	0x51	HCSE
UHCSI	RACP Interrupt Register[2]	0x51	HCSE
OOCDI	RACP Interrupt Register[4]	0x51	OOCDE
PSLI	RPOP Interrupt Status Register[7]	0x31	PSLE
PLOPI	RPOP Interrupt Status Register[5]	0x31	PLOPE
PAISI	RPOP Interrupt Status Register[3]	0x31	PAISE
PRDII	RPOP Interrupt Status Register[2]	0x31	PRDIE
PBIPEI	RPOP Interrupt Status Register[1]	0x31	PBIPEE
PFEBEI	RPOP Interrupt Status Register[0]	0x31	PFEBEE
LFEBEI	RLOP Interrupt Enable and Status Register[3]	0x19	LFEBEE
LBIPEI	RLOP Interrupt Enable and Status Register[2]	0x19	LBIPEE
LAISI	RLOP Interrupt Enable and Status Register[1]	0x19	LAISE
LRDII	RLOP Interrupt Enable and Status Register[0]	0x19	LRDIE
SBIPEI	RSOP Interrupt Status Register[6]	0x11	SBIPEE
SLOSI	RSOP Interrupt Status Register[5]	0x11	SLOSE
SLOFI	RSOP Interrupt Status Register[4]	0x11	SLOFE
SOOFI	RSOP Interrupt Status Register[3]	0x11	SOOFE

STM-1 operation

The European SDH standard starts at the 155-Mbps rate called STM-1. The frame format of STM-1 is exactly the same as STS3-c, but with one subtle difference, which are the S[1:0] bits in the H1 byte of the Line Overhead. The S[1:0] bits must be set to 0x2. The following steps must be performed to change the S[1:0].

Note: The instruction used below: OUT(ADDR, VALUE); symbolizes that the VALUE indicated is written to the register ADDRessed.

1. Clear the FIXPTR bit in the Master Control Register (Address 0x05) to allow the TPOP Arbitrary Payload Pointer Registers (Address 0x45, 0x46) to be programmed.

   OUT(REG_05, 0x20);

2. Set TPOP Arbitrary Payload Pointer LSB Register (Address 0x45) to the value 0x0A. Which is the LSB of 0x20A (522 decimal), the fixed pointer value for starting the SPE at the first byte of the next SPE; which also is the default value for locked frame structure.

   OUT(REG_45, 0x0A);

3. Set TPOP Arbitrary Payload Pointer LSB Register (Address 0x46) to the value 0x9A. Which sets the most significant 2 bit is the fixed pointer to 0x2, completing the value 0x20A. Also, this leaves the NDF unchanged, and setting the S[1:0] to 0x2.

   OUT(REG_46, 0x9A);

4. Set the PLD bit of the TPOP Pointer Control Register to one. This will tell the CY7C955 to load the values written to the TPOP Arbitrary Payload Pointer Registers into the internal operating register. The PLD bit will be cleared after the values are loaded into the internal registers.

   OUT(REG_41, 0x10);

Conclusion

This application note presented a detailed description of the features available on the UTOPIA interface of the CY7C955. It also described the usage of the internal address latch of the microprocessor interface. Furthermore, it discussed the interrupt structure and interrupt handling. Finally, it illustrated the steps to configure the CY7C955 to operate in STM-1 mode.

Appendix A. C Code Fragment for Interrupt Handling

```
 This is a sample Interrupt handling routine for the
 CY7C955. Primarily, this routine will read in
 the Master Interrupt Register value, then determine
 which interrupt registers to access in order to isolate
 the corresponding interrupt. After the generated interrupt
 is identified, then the appropriate user defined handling
 procedures are called.
*/

#include <stdio.h>

/* Constants for register bit masking */
#define TROOLI0x80
#define TROOLV0x04
#define LCDI0x40
#define LCDV0x40
#define RDOOLI0x20
#define RDOOLV0x08
#define TACPI0x10
#define TXFOVRI0x20
#define TSOCI0x40
#define TXPRTYI0x10
#define RACPI 0x08
#define OOCDI0x10
#define CHCSI0x08
#define UHCSI0x04
#define RXFOVRI0x02
#define RPOPI0x04
#define RLOPI0x02
#define PSLI0x80
#define LOPI0x20
#define PAISI0x08
#define PRDII0x04
#define PBIPEI0x02
#define PFEBEI0x01
#define PLOPI0x20
#define LAISI0x02
#define LRDII0x01
#define LBIPEI0x04
#define LFEBEI0x08
#define RSOPI0x01
#define SBIPEI0x10
#define LOSI0x20
#define LOFI0x10
#define OOFI0x08
#define LOSV0x04
#define LOFV0x02
#define OOFV0x01

/* function prototypes*/
int inbyte(int address, int* value){};
int report_tx_clk_outoflock();
int report_tx_clk_inlock();
int report_no_lcd();
int report_lcd();
int report_rx_clk_outoflock();
```

Appendix A. C Code Fragment for Interrupt Handling (continued)

```
int report_rx_clk_inlock();
int report_txfifo_overrun();
int report_tsoc_error();
int report_txprty_error();
int report_racp_chst();
int report_correct_HEC();
int report_uncorrect_HEC();
int report_rxfifo_overrun();
int report_psl_chg();
int report_lop_chg();
int report_pais_chg();
int report_prdi_chg();
int report_pbipe();
int report_pfebe();
int report_lais_chg();
int report_lrdi_chg();
int report_lbipe();
int report_lfebe();
int report_sbipe();
int report_los();
int report_not_los();
int report_lof();
int report_not_lof();
int report_oof();
int report_not_oof();

int main (){
        Interrupt_Handling();
}

int Interrupt_Handling() {
        int master_int_reg;
        int tx_clk_cntl_reg;
        int master_cntl_reg;
        int rx_clk_cntl_reg;
        int tacp_cntl_reg;
        int tacp_fifo_reg;
        int racp_int_reg;
        int rpop_int_reg;
        int rlop_int_reg;
        int rsop_int_reg;

        /*Read in Master Interrupt Register.*/
        inbyte (0x02, &master_int_reg);
        /*This inbyte(ADDR, &variable); instruction
          reads in the register at ADDR and stores
          its value into variable.
        */


        /*Compare mask out TROOLI bit, if it is a
          one then access the corresponding register
          and act accordingly.
        */
        if (master_int_reg & TROOLI){
                inbyte (0x06, &tx_clk_cntl_reg);
                if ( tx_clk_cntl_reg & TROOLV){
```

Appendix A. C Code Fragment for Interrupt Handling (continued)

```
                report_tx_clk_outoflock();
        }
        else {
                report_tx_clk_inlock();
        }
}

/*Check for loss of cell delineation.*/
if (master_int_reg & LCDI){
        inbyte (0x05, &master_cntl_reg);
        if (master_cntl_reg & LCDV){
                report_no_lcd();
        }
        else {
                report_lcd();
        }
}

/*Check for receive reference clock lock.*/
if (master_int_reg & RDOOLI){
        inbyte(0x07, &rx_clk_cntl_reg);
        if (rx_clk_cntl_reg & RDOOLV){
                report_rx_clk_outoflock();
        }
        else{
                report_rx_clk_inlock();
        }
}

/*Check for TACP interrupts*/
if (master_int_reg & TACPI){
        inbyte(0x60, &tacp_cntl_reg);

        /*Check for fifo overrun.*/
        if (tacp_cntl_reg & TXFOVRI){
                report_txfifo_overrun();
        }
        /*Check for out of place TSOC*/
        if (tacp_cntl_reg & TSOCI){
                report_tsoc_error();
        }
        /*Check for parity error*/
        inbyte(0x63, &tacp_fifo_reg);
        if (tacp_fifo_reg & TXPRTYI){
                report_txprty_error();
        }
}

/*Check for RACP interrupts*/
if (master_int_reg & RACPI){
        inbyte(0x51, &racp_int_reg);

        /*Check for cell delineation*/
        if (racp_int_reg & OOCDI){
                report_racp_chst();
        }
        /*Check for correctable HCS*/
```

Appendix A. C Code Fragment for Interrupt Handling (continued)

```
        if (racp_int_reg & CHCSI){
                report_correct_HEC();
        }
        /*Check for uncorrectable HCS*/
        if (racp_int_reg & UHCSI){
                report_uncorrect_HEC();
        }
        /*Check for fifo overrun*/
        if (racp_int_reg & RXFOVRI){
                report_rxfifo_overrun();
        }
}

/*Check for RPOP interrupts*/
if (master_int_reg & RPOPI){
        inbyte(0x31, &rpop_int_reg);

        /*Check for Path Signal Label error*/
        if (rpop_int_reg & PSLI){
                report_psl_chg();
        }
        /*Check for Loss of Pointer*/
        if (rpop_int_reg & LOPI){
                report_lop_chg();
        }
        /*Check for Path AISI*/
        if (rpop_int_reg & PAISI){
                report_pais_chg();
        }
        /*Check for Path Remote Defect Indication*/
        if (rpop_int_reg & PRDII){
                report_prdi_chg();
        }
        /*Check for Path BIP-8*/
        if (rpop_int_reg & PBIPEI){
                report_pbipe();
        }
        /*Check for Path FEBE*/
        if (rpop_int_reg & PFEBEI){
                report_pfebe();
        }
}

/*Check for RLOP interrupt*/
if (master_int_reg & RLOPI){
        inbyte(0x19, &rlop_int_reg);

        /*Check for Line AIS*/
        if (rlop_int_reg & LAISI){
                report_lais_chg();
        }
        /*Check for Line RDI*/
        if (rlop_int_reg & LRDII){
                report_lrdi_chg();
        }
```

Appendix A. C Code Fragment for Interrupt Handling (continued)

```
                /*Check for Line BIP-24*/
                if (rlop_int_reg & LBIPEI){
                        report_lbipe();
                }
                /*Check for Line FEBE*/
                if (rlop_int_reg & LFEBEI){
                        report_lfebe();
                }
        }

        /*Check for RSOP*/
        if (master_int_reg & RSOPI){
                inbyte(0x11, &rsop_int_reg);

                /*Check for Section BIP-24*/
                if (rsop_int_reg & SBIPEI){
                        report_sbipe();
                }
                /*Check for Loss Of Signal*/
                if (rsop_int_reg & LOSI){
                        if(rsop_int_reg & LOSV){
                                report_los();
                        }
                        else{
                                report_not_los();
                        }
                }
                /*Check for Loss Of Frame*/
                if (rsop_int_reg & LOFI){
                        if(rsop_int_reg & LOFV){
                                report_lof();
                        }
                        else{
                                report_not_lof();
                        }
                }
                /*Check for Out Of Frame*/
                if (rsop_int_reg & OOFI){
                        if(rsop_int_reg & OOFV){
                                report_oof();
                        }
                        else{
                                report_not_oof();
                        }
                }
        }
}
```