OPERATIONS MANUAL
PCM-COM8

NOTE: *This manual has been designed and created for use as part of the WinSystems' Technical Manuals CD and/or the WinSystems' website. If this manual or any portion of the manual is downloaded, copied or emailed, the links to additional information (i.e. software, cable drawings) will be inoperable.*

REVISION HISTORY

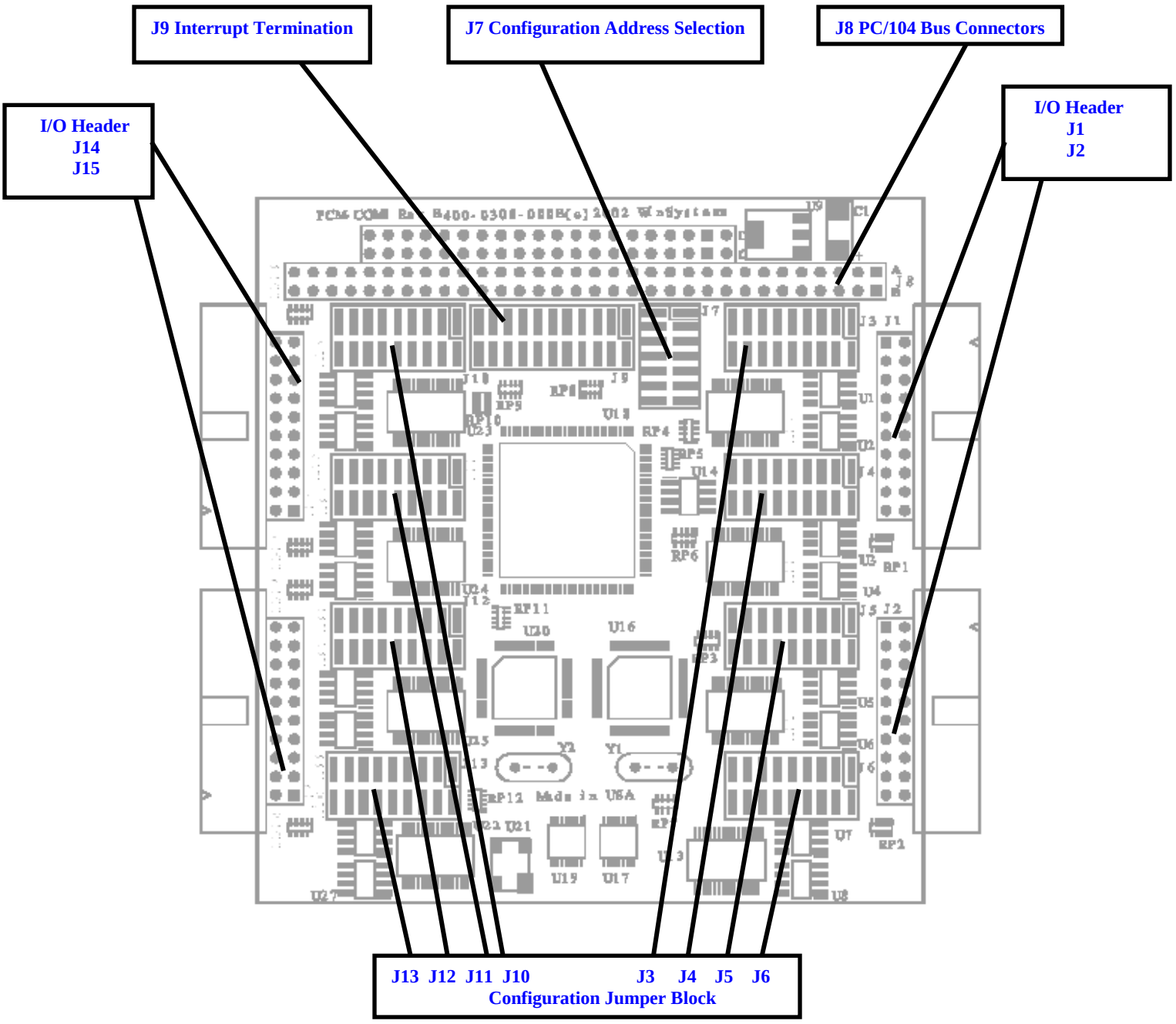
P/N 403-0308-000

ECO Number	Date Code	Rev Level
ORIGINATED	021101	B
03-41	030514	B1
03-60	030604	B2

TABLE OF CONTENTS

Visual Index – Quick Reference

For the convenience of the user, a copy of the Visual Index has been provided with direct links to connector and jumper configuration data.



1 GENERAL INFORMATION

1.1 Features

- Eight independently addressable COM channels
- Asynchronous data rates to 115Kbps
- UART addresses and interrupts are software configurable
- 1K Bit EEPROM stores configuration data
- 16C554 compatible UARTS with 128 byte TX/RX FIFOs
- Individual selection of RS-232/RS-422/RS-485 modes per channel
- Auto TX and Echoed RS-485 capability
- Jumper selectable termination available per channel
- +5 Volt only operation
- Extended temperature -40°C to +85ºC operating range
- Full complement of Modem control signals
- Supports individual or shared interrupts on a per channel basis

1.2 General Description

The PCM-COM8 is a PC/104 compliant 8-channel asynchronous communications card. It is populated with two EXAR 16C854 Quad UARTs. A custom bus-interface chip provides for software configuration of the UART addresses and interrupt assignments. Its on-chip state machine controls storage and retrieval of configuration settings to or from the on board 1Kbit serial EEPROM. The board is capable of loading customized configuration settings from the EEPROM at powerup without the need for software intervention. User selectable jumpers allow for channel by channel configuration for RS-232, RS-422 and RS-485 operating modes. An 8-bit interrupt status register allows software to "at a glance" determine which ports require servicing when operating in a shared interrupt environment. The deep 128 byte transmit and receive FIFOs along with the interrupt sharing capabilities allow even a modest processor to adequately handle the 8 ports on a continuous basis even at higher baud rates.

1.3 Specifications

1.3.1 Electrical

Bus Interface : PC/104 8-bit or 16-Bit expansion bus

VCC: +5V ± 5% 125 mA RS-232 Mode, typical in idle state
350 mA RS-422 Mode, typical with all transmitters enabled and TX/RX termination enabled

1.3.2 Mechanical

Dimensions : 4.1 X 3.8 X .60 inches (without expansion modules or cables)

PC-Board : FR-4 Epoxy Glass with 2 signal layers and 2 power planes with screened component legend, and plated through holes.

Connectors :

Serial I/O : RA type IDH-20LP-SR3-TR
PC/104 : Teka type PC220

Jumpers : 2mm Jumpers. Samtec type 2SN-BK-G

1.3.3 Environmental

Operating Temperature : -40° to +85° C

Non-Condensing Relative Humidity : 5 to 95%

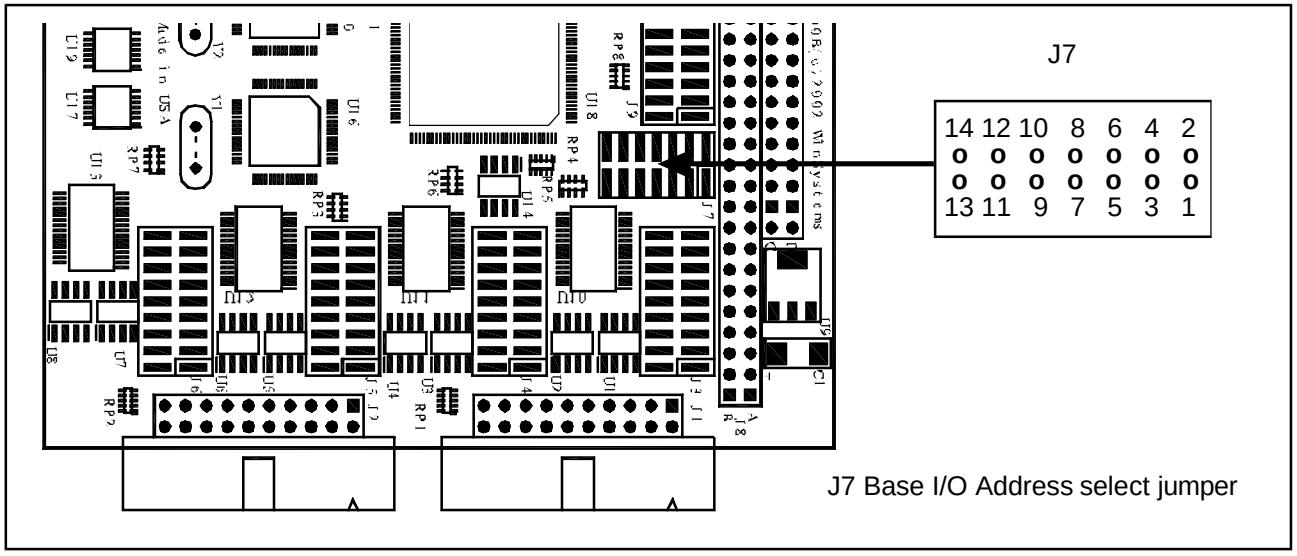
2 PCM-COM8 Technical Reference

2.1 Introduction

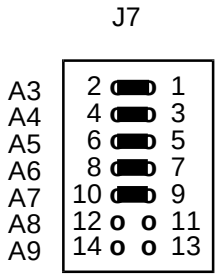
This section of the manual is intended to provide the necessary information to configure the PCM-COM8 for the desired mode(s) of operation. WinSystems maintains a Technical Support Group to help answer questions regarding configuration and programming of the board. For answers to questions not adequately addressed in this manual, contact Technical Support at (817) 274-7553 between 8AM and 5PM Central Time. Appendix D contains the complete reprint of the XR16C854 Datasheet and is provided to the programmer as a source for all UART register programming information.

2.2 Configuration Address Selection

The PCM-COM8 uses 8 I/O addresses for configuration and control of the COM port interface. The individual COM port addresses are software programmable but it is necessary to have control registers at a known location. Jumper block J7 selects the I/O address of the configuration registers. Also note that each enabled COM port uses 8 additional I/O addresses.



The base I/O address is selected by placing or removing jumpers from the J7 jumper block according to the binary address desired. A missing jumper is a '1' and an installed jumper is a '0'. The example below shows the default setting of 300H (11 0000 0XXX).



Example base address of 300H

2.3 Configuration Registers

The PCM-COM8 allows for software configuration of all 8 UART base addresses as well as their IRQ assignments. There is also a direct interface to the 1Kbit onboard EEPROM that can be used to store and retrieve configuration settings. The register and bit definitions are shown in the following tables.

The use of the first three registers is very straight forward and allows applicationsoftware to determine the current UART addresses or interrupts and to change the current mappings as desired. Registers at offsets 4 through 7 control the interface to the EEPROM. See Section 2.7 'EEPROM Programming Interface' for details.

BASE PORT + 0 - INDEX REGISTER (R/W)

Bits 7 - 3 N/A - always reads 0
Bits 2 - 0 Index Value (UART numbers 0 to 7) This specifies the UART for which the next two registers address.

BASE PORT + 1 - BASE ADDRESS REGISTER (R/W)

Bit 7 Enable/Disable UART (1 = enable)
Bit 6 - 0 Upper 7 bits of the desired 10-bit I/O Address (i.e. For 300h value would be 60h)

BASE PORT + 2 - IRQ ASSIGNMENT REGISTER (R/W)

Bits 7 - 4 N/A - Always reads 0
Bits 3 - 0 IRQ assignment 0 to 15. IRQ assigned to 0 is the same as disabled.

NOTE : Not all IRQs are available on the PC/104 Bus. IRQs 1,8,13 are typically not available.

BASE PORT + 3 - INTERRUPT ID REGISTER (RO)

Bit 7	UART 8 has interrupt pending
Bit 6	UART 7 has interrupt pending
Bit 5	UART 6 has interrupt pending
Bit 4	UART 5 has interrupt pending
Bit 3	UART 4 has interrupt pending
Bit 2	UART 3 has interrupt pending
Bit 1	UART 2 has interrupt pending
Bit 0	UART 1 has interrupt pending

BASE PORT + 4 - EEPROM COMMAND REGISTER (R/W)

Bits 7 - 0 EEPROM operating command

BASE PORT + 5 - EEPROM HIGH DATA REGISTER (R/W)

Bits 7 - 0 Upper 8 Bits of EEPROM read/write data (Hold Base Address of UART)

BASE PORT + 6 - EEPROM LOW DATA REGISTER (R/W)

Bits 7 - 0 Lower 8 Bits of EEPROM read/write data (Holds IRQ Setting)

BASE PORT + 7 - COMMAND REGISTER (WO)

Bits 7 - 2 N/A

Bit 1 Read Multiple. Starts state machine to load config registers from EEPROM

Bit 0 Command Start. Writing a 1 to this bit starts the command specified in BASE+4

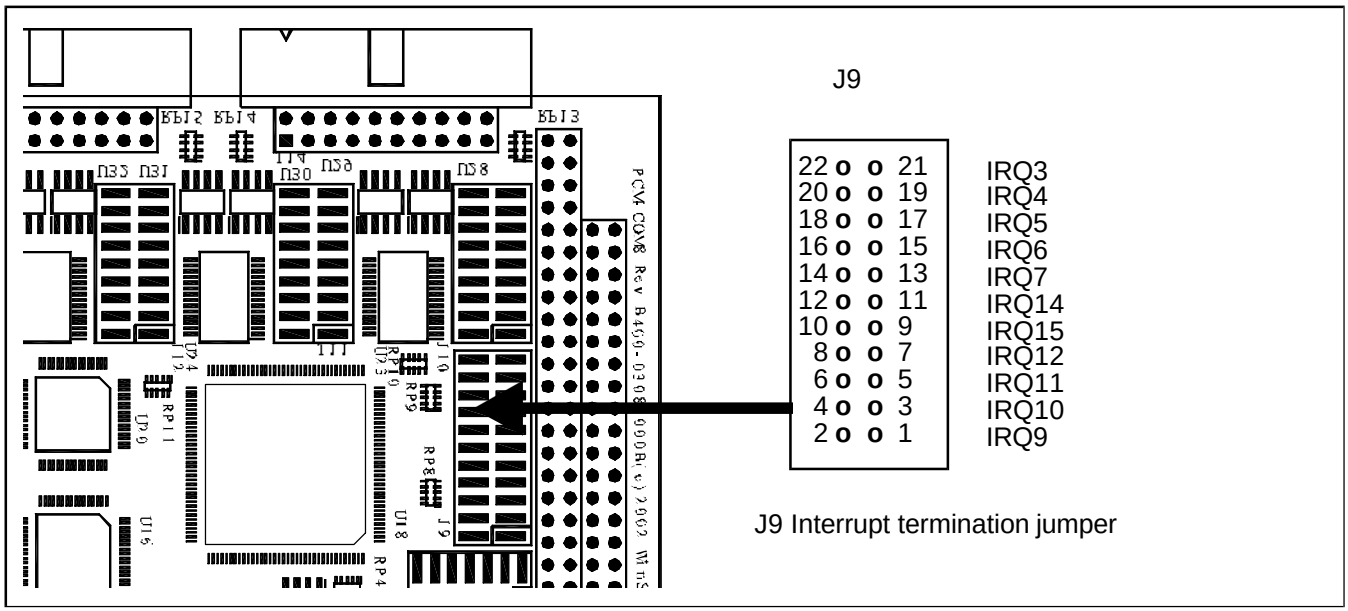
BASE PORT + 7 - STATUS REGISTER (RO)

Bit 7 1 = Not Busy, 0 = Busy

Bit 6 1 = Transfer in progress . When this bit is set all other registers are unavailable.

2.4 Interrupt Termination

Even though the PCM-COM8 is fully software configurable, due to the nature in which its interrupts are driven, a terminating resistor MUST be enabled for all interrupts actually used on the board. A jumper placed on the J9 jumper block corresponding to the desired IRQ places a 1K OHM resistor between the interrupt line and ground.



2.5 RS-232/RS-422/RS-485 Mode Select

Each of the 8 serial channels on the PCM-COM8 can be individually configured for any one of a number of operating modes, including :

1. RS-232 Mode
2. RS-422 Mode with RTS transmitter enable
3. RS-422 Mode with auto transmitter enable
4. RS-485 Mode with RTS transmitter enable
5. RS-485 Mode with RTS transmitter enable and echo back
6. RS-485 Mode with auto transmitter enable
7. RS-485 Mode with auto transmitter enable and echo back

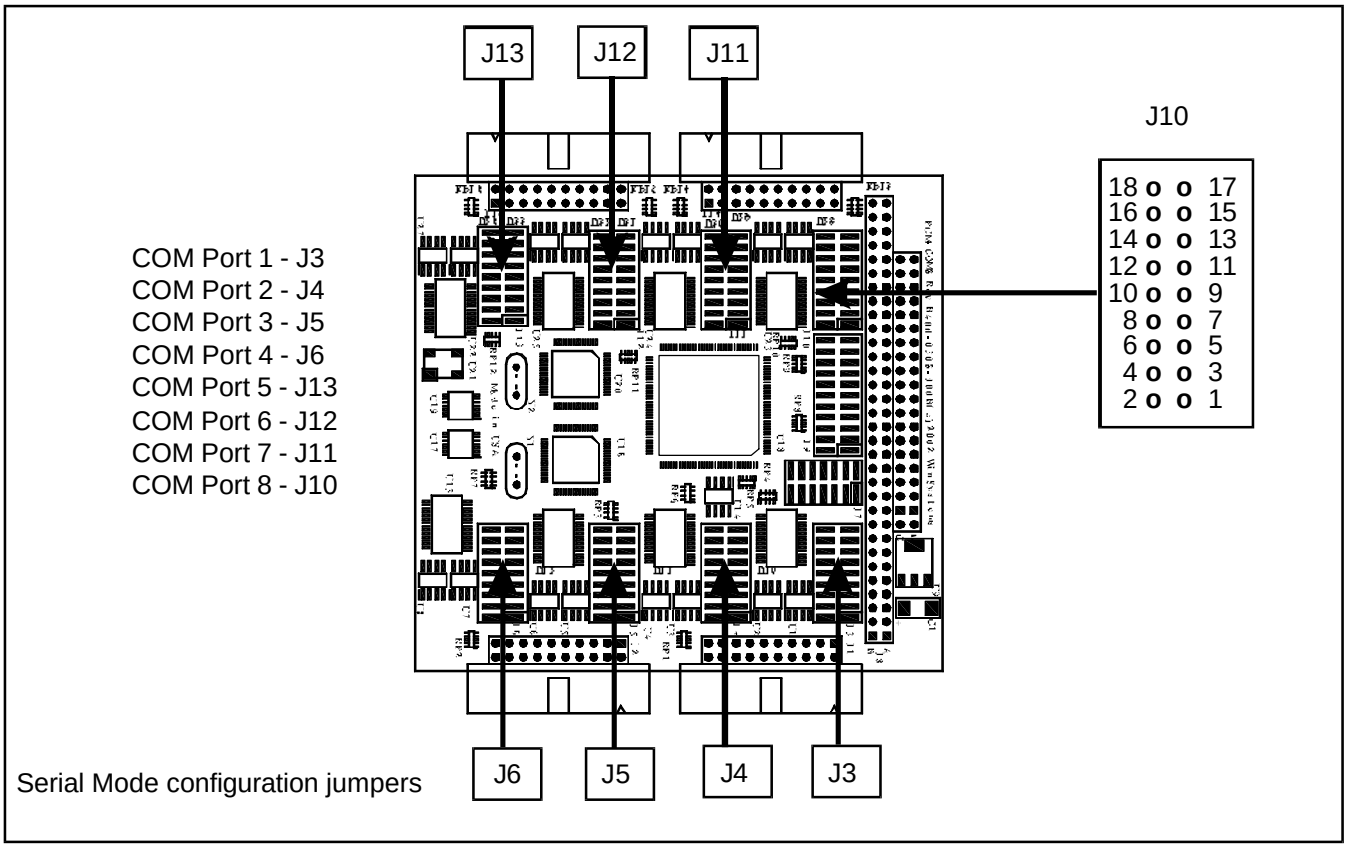
Modes 2, 4, and 5 require the RTS bit in the MCR (Bit 1) be set in order to Transmit
Modes 4 and 5 Require that RTS in the MCR (Bit 1) be deasserted in order to receive.

Each of the RS-422/RS-485 modes also allows for jumper selection of transmit and/or receive termination resistor(s). There is a 18 pin configuration jumper for each of the 8 ports that allows the user to select the operating mode and its optional features and termination. The Jumper numbers and corresponding port numbers are shown on the following page. There are three choices for termination when RS-422 or RS-485 modes are used.

TX(100) - Places a 100 Ohm resister across the TX+/TX- pair

RX(100) - Places a 100 Ohm resistor across the RX+/RX- pair

TX/RX(300) - Places a 100 Ohm Resistor from +5V to TX/RX+, a 100 Ohm resistor from TX/RX- to ground and a 100 Ohm resistor between TX/RX+ and TX/RX--



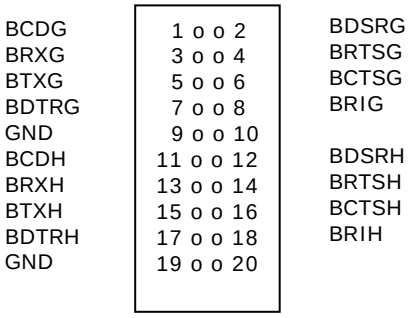
Each channel is configured using its specific jumper block as shown above. The table below shows the appropriate jumpering for the various modes.

Mode #	Description	Jumpers	Termination		
			TX (100)	RX (100)	TX/RX (300)
1	RS-232	1-2	N/A	N/A	N/A
2	RS-422 RTS Enable	3-4, 9-10	11-12	17-18	11-12 13-14 15-16
3	RS-422 Auto Enable	3-5, 9-10 (One node must use TX/RX 300 Termination)	N/A	17-18	11-12 13-14 15-16
4	RS-485 RTS Enable	3-4, 7-8	11-12	N/A	11-12 13-14 15-16
5	RS-485 RTS Enable with Echo-Back	3-4, 8-6	11-12	N/A	11-12 13-14 15-16
6	RS-485 Auto Enable	3-5, 7-8 (One node must use TX/RX 300 Termination)	N/A	N/A	11-12 13-14 15-16
7	RS-485 Auto Enable with Echo-Back	3-5, 8-6 (One node must use TX/RX 300 Termination)	N/A	N/A	11-12 13-14 15-16

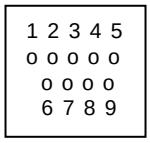
2.6 Serial Port I/O Pin Definitions

The PCM-COM8 terminates its serial ports into four 20-pin IDC style .100" connectors. Each connector supports two serial ports. J1 is for ports 1 and 2, J2 is for ports 3 and 4, J14 is for ports 5 and 6, and J15 is for ports 7 and 8. WinSystems has available a pre-terminated cable (CBL-173-1) that connects from the 20-pin IDC female to two 9-pin D-Sub male connectors. Users wishing to create their own cables may use the cable drawings as a guide. When using the standard WinSystems cables, the 9-pin D-Sub pin definitions are dependent upon the operating mode of the channel. Shown below are the pin definitions for each of the three major operating modes.

J1, J2, J14, J15



DB9 Male



RS-232 Modes	RS-422 Modes	RS-485 Modes
1. CD	1. N/A	1. N/A
2. RX	2. TX+	2. TX/TR+
3. TX	3. TX-	3. TX/TR-
4. DTR	4. N/A	4. N/A
5. GND	5. GND	5. GND
6. DSR	6. RX+	6. N/A
7. RTS	7. RX-	7. N/A
8. CTS	8. N/A	8. N/A
9. RI	9. N/A	9. N/A

2.7 EEPROM Programming Interface

The PCM- COM8 has an onboard 1 Kbit serial EEPROM. At powerup the onboard state machine loads eight 16- bit words from the EEPROM starting at address 0 into the configuration space for the eight UARTS, setting their base addresses and interrupt assignments from the loaded data. A DOS configuration program, config.exe, is supplied by WinSystems to allow the user to reprogram the defaults. It may be desirable for application code to be able to access and store configuration data in the EPROM directly. This section will briefly document the steps necessary to directly access the EEPROM.

To read a 16- bit value from the EEPROM

1. Load the address (0-63) of the desired value in BASE+4 with Bit 7 set
2. Start the read operation by writing a 1 to BASE+7
3. Delay approximately 5mS
4. Poll BASE+7. When Bit 7 is set and Bit 6 is clear, the operation is complete
5. Extract the 16- bit data value. BASE+5 holds the upper 8 bits, BASE+6 holds the lower 8 bits

To Enable Writes to the EEPROM

1. Load BASE+4 with 30H
2. Start the operation by writing 1 to BASE+7
3. Delay approximately 5mS
4. Poll BASE+7. When Bit 7 is set and Bit 6 is clear, the operation is complete

To Write a 16-bit value to the EEPROM

1. Load the address (0-63) of the destination location to BASE+4 with Bit 6 set.
2. Load BASE+5 with the upper 8 bits of the 16-bit data value
3. Load BASE+6 with the lower 8 bits of the 16-bit data value
4. Start the operations by writing a 1 to BASE+7
5. Delay approximately 5mS
6. Poll BASE+7. When Bit 7 is set and Bit 6 is clear, the operation is complete

To Disable Writes to the EEPROM

1. Load BASE+4 with 0
2. Start the operation by writing 1 to BASE+7
3. Delay approximately 5mS
4. Poll BASE+7. When Bit 7 is set and Bit 6 is clear, the operation is complete

To Load Configuration Data from EEPROM into Configuration Registers

1. Load the address (0-63) of the first value in BASE+4 with Bit 7 set
2. Start the configuration load operation by writing a 3 to BASE+7
3. Delay approximately 5mS
4. Poll BASE+7. When Bit 7 is set and Bit 6 is clear, the operation is complete

EEPROM ACCESS NOTES :

1. By default, Writes are disabled to the EEPROM. The enable sequence given above must be followed before Writes will be allowed. It is advisable that the "Disable Writes to EEPROM" sequence be accomplished after updating EEPROM data. That will help avoid inadvertent Writes or corruption of the EEPROM data. The WinSystems *config.exe* program disables EEPROM Writes before exiting.

2. There is no command to store all of the current config data to EEPROM. It must be read from the configuration space and written to the EEPROM word-by-word.

3. At powerup the data beginning at address 0 will be loaded into the configuration space. It is possible to store other configurations starting at any address, but they must be loaded under program control.

4. If desired, the 56 words of data not used for powerup configuration (starting at word 8), could be used for storage of any arbitrary information.

5. Refer to the source code examples in Section 3 for actual working program examples.

2.8 PC/104 Bus Connectors

The PCM-COM8 plugs into the PC/104 bus using the connectors at J8. The PC/104 bus pin definitions are shown here for reference.

GND	D0	C0	GND
MEMCS16*	D1	C1	SBHE*
IOCS16*	D2	C2	LA23
IRQ10	D3	C3	LA22
IRQ11	D4	C4	LA21
IRQ12	D5	C5	LA20
IRQ15	D6	C6	LA19
IRQ14	D7	C7	LA18
DACK0*	D8	C8	LA17
DRQ0	D9	C9	MEMR*
DACK5*	D10	C10	MEMW*
DRQ5	D11	C11	SD8
DACK6*	D12	C12	SD9
DRQ6	D13	C13	SD10
DACK7*	D14	C14	SD11
DRQ7	D15	C15	SD12
+5V	D16	C16	SD13
MASTER*	D17	C17	SD14
GND	D18	C18	SD15
GND	D19	C19	KEY

IOCHK*	A1	B1	GND
SD7	A2	B2	RESET
SD6	A3	B3	+5V
SD5	A4	B4	IRQ9
SD4	A5	B5	-5V
SD3	A6	B6	DRQ2
SD2	A7	B7	-12V
SD1	A8	B8	SRDY
SD0	A9	B9	+12V
IOCHRDY	A10	B10	KEY
AEN	A11	B11	SMEMW*
SA19	A12	B12	SMEMR*
SA18	A13	B13	IOW*
SA17	A14	B14	IOR*
SA16	A15	B15	DACK3*
SA15	A16	B16	DRQ3
SA14	A17	B17	DACK1*
SA13	A18	B18	DRQ1
SA12	A19	B19	REFRESH*
SA11	A20	B20	BCLK
SA10	A21	B21	IRQ7
SA9	A22	B22	IRQ6
SA8	A23	B23	IRQ5
SA7	A24	B24	IRQ4
SA6	A25	B25	IRQ3
SA5	A26	B26	DACK2*
SA4	A27	B27	TC
SA3	A28	B28	BALE
SA2	A29	B29	+5V
SA1	A30	B30	OSC
SA0	A31	B31	GND
GND	A32	B32	GND

2.9 Jumper/Connector Summary

Jumper/ Connector	Description	Page Number
J1	PORT1/PORT2 I/O Header	2-6
J2	PORT3/PORT4 I/O Header	2-6
J3	PORT1 Configuration Jumper Block	2-5
J4	PORT2 Configuration Jumper Block	2-5
J5	PORT3 Configuration Jumper Block	2-5
J6	PORT4 Configuration Jumper Block	2-5
J7	Base I/O Address Select jumper	2-1
J8	PC/104 Bus Connector	2-8
J9	IRQ Termination jumper block	2-3
J10	PORT8 Configuration Jumper Block	2-5
J11	PORT7 Configuration Jumper Block	2-5
J12	PORT6 Configuration Jumper Block	2-5
J13	PORT5 Configuration Jumper Block	2-5
J14	PORT7/PORT8 I/O Header	2-6
J15	PORT5/PORT6 I/O Header	2-6

3 PCM-COM8 Software Examples

3.1 Introduction

This section gives some application code examples of programming some of the advanced capabilities and features of the PCM-COM8. The samples are provided by WinSystems on an "as-is" basis and there is no warranty or declaration of fitness of purpose associated with these examples. All of the accompanying programs and source code are Copyright 2002 by WinSystems Inc. All rights reserved. See the header files for allowed usage, disclosure, and incorporation into application code.

There are two classes of programs and sample code in this section. The first portion gives examples of accessing the configuration registers and the direct access of the onboard EEPROM.

The second section is a set of C functions written in the DOS environment that illustrates the initialization, data handling, and interrupt handling, for all 8 serial ports. Individual and shared interrupts are supported. Data-flow handshaking, RS-422, and RS-485 are not supported in these samples.

3.2 Configuration Program Examples

3.2.1 SHOWCOMS.EXE

SHOWCOMS.EXE is an extremely simple DOS application that simply reads out the current configuration from the configuration registers and prints the results to the screen. It is invoked at the DOS command prompt with :

```
showcoms 300
```

where the 300 is replaced with the actual hex address that the board is jumpered for. This program serves as a confirmation of the current settings of the ports. From a programming standpoint examining the source code for showcoms.exe shows how the configuration registers are read and decoded into meaningful values.

3.2.2 INIT.EXE

INIT.EXE is another even simpler DOS program that simply executes the configuration load command which reloads all of the configuration registers with the values stored in EEPROM. This program is provided primarily as a programming example of how to load all of the configuration registers at once from the EEPROM. It can be executed at the DOS prompt with :

```
init 300
```

As with the previous and all subsequent examples the I/O address on the command line should be replaced with the one for which the board is actually jumpered.

3.2.3 CONFIG.EXE

CONFIG.EXE is the grand daddy of the configuration examples. This useful program is how, without writing a bit of code, that the user can reprogram the EEPROM for their desired defaults that will be reloaded at each powerup automatically. The program is invoked at the DOS prompt with :

```
config 300 config.txt
```

As with the previous programs, the actual I/O port set by the jumpers on the board is used on the command line. The second argument is the name of an ASCII file containing the port I/O address, interrupt, and enable status instructions. Each line is read by the program and then the entire configuration is stored to the EEPROM at Address 0. At the next powerup the board will autoload the new configuration. Here is a sample of the config.txt file :

```
0 308 15 Y
1 310 15 Y
2 318 15 Y
3 320 15 Y
4 328 15 Y
5 330 15 Y
6 338 15 Y
7 340 15 Y
```

The first item in a line is the port number starting with 0 for port 1. The second item is the I/O address from 0 to 400H. The program expects the value to be in Hex. The third item is the IRQ assignment from 0 to 15. These values are in decimal. Setting an IRQ of 0 effectively disables interrupts from that port. The board is fully capable of hardware sharing of interrupts and will do so on command as shown above with all of the ports sharing IRQ3. The software however, must be cognizant of this sharing and deal with the interrupts appropriately. Lastly a 'Y' or 'N' character designates whether the port is to be enabled or not. A 'Y' signals that it is to be enabled. A disabled port uses no I/O space or IRQ resources. Extra lines or white space in the configuration file can confuse the elementary parsing done by *CONFIG.EXE* so the best course is simply to copy and carefully edit configuration files using text editors that do not embed any formatting info into the file (pure ASCII). This program, when examined at the source level, combines nearly all of the elements needed to programmatically reconfigure the board.

3.3 Serial I/O Examples

The *COM8IO.C* and *COM8IO.H* files are intended to demonstrate the deep FIFO capability of the EXAR 16C854 quad UARTs, the abili ty of soft ware to auto- configure for the I/O port and in ter rupt assignments and to demonstrate how shared interrupts should be handled on a multi-port board such as the COM8. These files do not comprise a comprehensive driver although for a large number of users utilizing RS-232, they probably could be used as-is. Customer written programs could be linked directly to the COM8 I/O rou tines con tained in these files. Be cause of the deep 128 byte FIFO there is no implementation of any type of flow-control. There is also no support for alternate communications modes such as RS-422 or RS-485. The main functions in the COM8IO.C file will be briefly described and any special features or caveats will be enumerated. This discussion should be followed with the source code listing in hand.

3.3.1 COM8IO.C

This file begins with a number of data and structure definitions. All of the COM port operating parameters, as well as buffer pointers, are stored in an array of eight structures named *comm_struct[]*.

Refer to the structure definition in COM8IO.H for particulars on this structure. If additional features such as RS-422 or RS-485 support were to be added, flags and parameters, could be added to *comm_struct[]*.

To make the code more readable, rather than an array of functions for the primary ISRs ,since there's a fixed number (8) we just de clare 8 unique ISR func tions, all of which will then vec tor to common handler code.

The first real func tion is *com8_init()* which has no re turn value and takes no ar gu ments. This would be the first func tion that any pro gram de sir ing to use the other func tions MUST call. This func tion reads the configuration registers on the PCM-COM8 and fills in *comm_struct[]* with the I/O port and interrupt assignment val ues. It also de fines a de fault baud rate of 38400. There is no ac tual UART initialization effected in this function.

An application after calling *com8_init()* would then need to fill in any of the other parameters in *comm_struct[]* that are desired such as baud rate or transmit or receive FIFO levels.

The next function that an application would need to call before performing I/O on a COM port is *com_open()*. This function takes as an argument the COM port number 0-7 and returns 0 on success or sev eral non- zero val ues on fail ure. Once this func tion has been called, the port is ready for I/O. Note that this func tion can ini tial ize and util ize the en tire 128 byte FI FO and will build an in terupt shar ing mask to al low the IS Rs to de ter mine what other ports are shar ing this in ter rupt. This is one of the long est and most com plex func tions in the ex am ple. Note that the ini tiali za tion is hard wired to an 8 bit word with 1 stop bit and no parity.

Once an application is finished using one of the ports, it is recommended to call *com_close()*. This function take the port number as an argument and re turns 0 on suc cess. Ports closed prop erly will reinstate any previous interrupt handlers, and shutdown the port from generating any more interrupts.

The next thing in the source code is the eight primary interrupt service routines *isr_1()* to *isr_8()*. These eight ISRs simply call a com mon han dler, *common_isr()*, with the pri mary port number as signed

to that interrupt as an argument. The primary purpose of the *common_isr()* function is to poll the *int_id* register to determine which UARTs have interrupts pending and call *handler()* with the port number needing service as an argument. The *common_isr()* function continues to poll and dispatch interrupts until the *int_id* register comes up 0, at which time it re-arms the interrupt controller and exits. The *handle()* function does the meat of the work for an interrupt. Only one port at a time is serviced within *handle()*. The first task is to examine the UARTs IIR register (offset 2) and determine what is the highest priority interrupt source needing service. Each possible cause for an interrupt is examined by the 'case' statement and handled, in most cases, with inline code. Like the outer loop in *common_isr()* the *handler()* function does not return until ALL interrupts currently pending in the UART being serviced have been handled. This is the key to handling shared interrupts. It is imperative that all sources and all UARTs have been serviced before exiting the ISR or a lockout (no more interrupts serviced) will occur.

The functions that follow are, more or less, ordinary serial I/O functions that you might expect. *com_putch()* takes two arguments, the port number and the character to send. *com_puts()* also requires two arguments, the port number, and a pointer to a string to be sent. Note that this function will NOT send an arbitrary buffer of characters and assumes that there is a null terminated string at the pointer's address.

The next function *com_check()* is very useful for interrogating whether any characters are currently waiting in the receive buffer. It takes a single argument specifying the COM port number and returns 0 if there are currently no characters in the receive buffer or 1, if there are 1 or more characters available.

The function *com_getch()* takes the same argument of port number and returns an available character or -1 if a time-out occurred waiting for a character. There are a couple of other functions in the file that are of limited general utility and can be examined by the programmer as desired.

3.3.2 COM8TEST.C

This file is actually a little test program written to test some of the functionality of the sample I/O routines present in *com8io.c* . It has undergone a number of iterations and it is presented in its most recent form when it was used to test Transmit and Receive FIFO operation. The program inits all eight ports to 38400 baud with the Transmit and Receive FIFO levels set and then sends a test string out Port 5. It then goes into a loop waiting for incoming characters on any of the eight ports or a keyboard input which terminates the program. On terminating, the program prints some statistics about the number of characters that were sent and received by each channel as well as the number of transmit and receive interrupts that were serviced. If the program is started, and then immediately terminated, it should show that on channel 5 a fairly large number of characters were sent with only a few interrupts required to send them. This program was compiled with the Borland C/C++ version 3.1 and 5.1 compilers withthe command line :

```
bcc com8test.c com8io.c
```

A precompiled version of *com8test.exe* is also present on the accompanying diskette.

APPENDIX A

Example Programs Source Listings

```
/* showcoms.c Copyright 2002 WinSystems Inc. All rights reserved */
/*********************************************************************
*
*   Name : showcoms.com
*
*   Project : PCM-COM8
*
*   Date    : 10/17/02
*
*   Revision : 1.00
*
*   Author  : Steve Mottin
*
*********************************************************************
*
*   Revision History
*
*   Date        Rev     Description
*   --------    -----   ------------------------------------------
*   10/17/02    1.00    Initial version
*
*********************************************************************
*/

#include <stdio.h>
#include <dos.h>

int com8read(int index, int port);

unsigned base_port;

#define ADDRESS     1
#define IRQ_ASSIGN  2

main(int argc, char *argv[])
{
unsigned base_address[8];
unsigned irq_assign[8];
int x;

    if(argc != 2)
    {
        printf("usage : showcoms address\n");
        exit(1);
    }

    sscanf(argv[1],"%x",&base_port);

    printf("\n\nPCM-COM8 Control port at %04X\n\n",base_port);

    printf("COM Port    Base Address    IRQ Number      Enabled\n");
    for(x=0; x<=7; x++)
    {
        printf("    %d\t\t",x);
```

```
        base_address[x] = com8read(x,ADDRESS);
        irq_assign[x] = com8read(x,IRQ_ASSIGN);
        printf("%04X\t\t",(base_address[x] &0x7f) << 3);

        if((irq_assign[x] & 0x0f) != 0)
            printf("%d\t\t",irq_assign[x] & 0x0f);
        else
            printf("N/A\t\t");

        if(base_address[x] & 0x80)
            printf("Y\n");
        else
            printf("N\n");
    }
}

int com8read(int index, int port)
{
    outportb(base_port, index);
    return(inportb(base_port+port));
}
```

```
/* config.c */

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

unsigned read_eeprom(int address);
void write_eeprom(int address, unsigned value);
void enable_write(void);
void disable_write(void);
void load_config(int address);


unsigned base_address;

char buffer[80];


main(int argc, char *argv[])
{
int x;
unsigned value;
FILE *fp;
unsigned address;
unsigned irq;
unsigned temp;
char enable;

    if(argc != 3)
    {
        printf("only %d arguments given\n",argc);
        printf("usage : program base_address program_file");
        exit(1);
    }

    sscanf(argv[1],"%x",&base_address);

    fp = fopen(argv[2],"r");

    if(fp == NULL)
    {
        printf("Can't open program file - Aborting\n");
        exit(1);
    }

    enable_write();

    while(fgets(buffer,70,fp) != NULL)
    {
        if(buffer[0] == '\x0a')
            break;
        sscanf(buffer,"%d %x %d %c",&address, &value, &irq, &enable);
        if(address > 7)
        {
```

```
            printf("Bad port number specified %d - Aborting\n",address);
            exit(1);
        }

        if((value <=0) || (value >= 0x400))
        {
            printf("Bad I/O Address specified %04x - Aborting\n",value);
            exit(2);
        }

        if((irq <=0) || (irq > 15))
        {
           printf("Bad IRQ specified %d - Aborting\n",irq);
           exit(3);
        }

        printf("Port %d Address %04X IRQ %d Enable = %c\n",address,value,irq,enable);
        temp = value >> 3;
        temp = temp << 8;

        if(enable == 'y' || enable == 'Y')
            temp = temp | 0x8000;

        temp = temp | (irq & 0x0f);

        write_eeprom(address,temp);

    }

    fclose(fp);
    disable_write();
    load_config(0);

}

/* Load up the config registers from the EEPROM starting from address */

void load_config(int address)
{
unsigned retry;

    outportb(base_address+4,address | 0x80);
    outportb(base_address + 7, 3);  /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address + 7) & 0xc0) == 0x80)
            break;
    }
    if(retry == 0)
```

```
        printf("Load Config - Timeout Error\n");

}


void enable_write(void)
{
unsigned retry;

    outportb(base_address+4,0x30);    /* Write out the command */


    outportb(base_address+7,1);     /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("enable write - timeout error\n");

}

void disable_write(void)
{
unsigned retry;

    outportb(base_address+4,0x00);    /* Write out the command */


    outportb(base_address+7,1);     /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("disable write - timeout error\n");

}
```

```
void write_eeprom(int address, unsigned value)
{
unsigned retry;

    outportb(base_address+4,address | 0x40);    /* Write out the command */

    outportb(base_address+5,value >> 8);
    outportb(base_address+6,value & 0xff);

    outportb(base_address+7,1);     /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("write eeprom - timeout error\n");

}

unsigned read_eeprom(int address)
{
unsigned value;
unsigned retry;

    outportb(base_address+4,address | 0x80);    /* Write out the command */
    outportb(base_address+7,1);         /* Start the transfer */

    delay(5);

    value = 0;
    retry = 50000;
    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("read eeprom - timeout error\n");

    value = inportb(base_address +5);
    value = value << 8;
    value = value | inportb(base_address + 6);


    return(value);
}
```

```
/* init.c */

#include <stdio.h>
#include <dos.h>

unsigned read_eeprom(int address);
void write_eeprom(int address, unsigned value);
void enable_write(void);
void disable_write(void);
void load_config(int address);


unsigned base_address;

main(int argc, char *argv[])
{
int x;
unsigned value;

    if(argc != 2)
    {
        printf("usage : init base_address");
        exit(1);
    }

    sscanf(argv[1],"%x",&base_address);

    load_config(0);

}

/* Load up the config registers from the EEPROM starting from address */

void load_config(int address)
{
unsigned retry;

    outportb(base_address+4,address | 0x80);
    outportb(base_address + 7, 3);  /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address + 7) & 0xc0) == 0x80)
            break;
    }
    if(retry == 0)
        printf("Load Config - Timeout Error\n");

}
```

```
void enable_write(void)
{
unsigned retry;

    outportb(base_address+4,0x30);    /* Write out the command */


    outportb(base_address+7,1);     /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("enable write - timeout error\n");

}

void disable_write(void)
{
unsigned retry;

    outportb(base_address+4,0x00);    /* Write out the command */


    outportb(base_address+7,1);     /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("disable write - timeout error\n");

}


void write_eeprom(int address, unsigned value)
{
unsigned retry;
```

```
    outportb(base_address+4,address | 0x40);    /* Write out the command */

    outportb(base_address+5,value >> 8);
    outportb(base_address+6,value & 0xff);

    outportb(base_address+7,1);     /* Start the command */

    delay(5);

    retry = 50000;

    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("write eeprom - timeout error\n");

}

unsigned read_eeprom(int address)
{
unsigned value;
unsigned retry;

    outportb(base_address+4,address | 0x80);    /* Write out the command */
    outportb(base_address+7,1);         /* Start the transfer */

    delay(5);

    value = 0;
    retry = 50000;
    while(retry--)
    {
        if((inportb(base_address +7) & 0xc0) == 0x80)
            break;

    }
    if(retry == 0)
        printf("read eeprom - timeout error\n");

    value = inportb(base_address +5);
    value = value << 8;
    value = value | inportb(base_address + 6);


    return(value);
}
```

```
/* COM8IO.C Copyright 2002 WinSystems Inc. All rights Reserved */
/*****************************************************************************
*
*   Name        : com8io.c
*
*   Project     : WinSystems PCM-COM8 Demonstrataion Code
*
*   Revision    : 1.00
*
*   Author      : Steve Mottin
*
*****************************************************************************
*
*   Revision History :
*
*   Date        Revision    Description
*   -------     --------    ------------------------------------------
*   10/30/02    1.00        Original
*
*****************************************************************************
*
*                   COPYRIGHT NOTICE AND DISCLAIMER
*
*   This source file is Copyright 2002 by WinSystems Inc, Arlington
*   Texas. All rights reserved except as provided below.
*
*   This file is intended to demonstrate possible usages and programming
*   of the PCM-COM8 board. It has NOT been tested extensively in a
*   multitude of environments and it is the users sole responsibility
*   to determine fitness of purpose. This file is provided 'as-is'
*   and WinSystems makes no warranty as to usability of these routines.
*
*   Users of the WinSystems PCM-COM8 may use any or all of the functions
*   in this file for incorporation in their application code utilizing the
*   PCM-COM8 without license or permission.
*
*   Other disclosure or use of this code is strictly prohibited.
*   Refer to the WinSystems Hardware/Software Sales Agreement for other
*   terms and conditions that may apply
*
*****************************************************************************
*/

#include <dos.h>
#include "com8io.h"

struct com_port comm_struct[8];

int cflag = 0;
int irq_masks[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

int intflag=0;

int int_id = 0;
```

```
int com8_base = 0;

void interrupt isr_0();
void interrupt isr_1();
void interrupt isr_2();
void interrupt isr_3();
void interrupt isr_4();
void interrupt isr_5();
void interrupt isr_6();
void interrupt isr_7();

void handle(int port_num);
void common_isr(int port_num);

/* ==================== Start of COM Functions ========================== */

void com8_init(unsigned base_port)
{
int x;
unsigned base_address;
unsigned irq_assign;

    com8_base = base_port;
    int_id = com8_base + 3;

    /* Read in all of the base_address and irq_assigns from the chip */

    for(x=0; x<8; x++)
    {
        comm_struct[x].int_num = 0;
        comm_struct[x].uart_base = 0;
        comm_struct[x].open_flag = 0;

        outportb(com8_base,x);  /* Set the index */
        base_address = inportb(com8_base + 1);
        irq_assign = inportb(com8_base + 2);

        if((base_address & 0x80) == 0)
            base_address = 0;

        base_address = (base_address & 0x7f) << 3;
        irq_assign = irq_assign & 0x7f;

        if((base_address < 0x100) || (base_address > 0x400))
        {
            base_address = 0;
            irq_assign = 0;
            continue;
        }

        if((irq_assign == 0) || (irq_assign > 15))
            continue;

        comm_struct[x].int_num = irq_assign;
```

```
        comm_struct[x].uart_base = base_address;
        comm_struct[x].baud_rate = 38400L;

    }
}


int com_open(int port_num)
{
unsigned base_port;
unsigned baud_divisor;
int int_number;
unsigned mask_val;
int temp;


    if(comm_struct[port_num].open_flag == OPEN_MAGIC)
        return 3;

    if(port_num > 7)
        return 2;
    if(comm_struct[port_num].uart_base == 0)
        return 1;


    int_number = comm_struct[port_num].int_num;

    if(int_number < 2 || int_number > 15)
        return 2;

    /* Set up the baud rate */

    baud_divisor = (unsigned) (1843200L / (comm_struct[port_num].baud_rate * 16L));

    base_port = comm_struct[port_num].uart_base;

    outportb(base_port+3,inportb(base_port + 3) | 0x80);
    outportb(base_port+1, baud_divisor >> 8);
    outportb(base_port, baud_divisor & 0xff);
    outportb(base_port+3,inportb(base_port+3) & 0x7f);


    /* Check whether a vector is already in place for the specified
        IRQ. If so just add our port to the mask value
    */

    mask_val = 1 << port_num;

    if(irq_masks[int_number])    /* if non-zero */
        irq_masks[int_number] |= mask_val;
    else
    {
        irq_masks[int_number] |= mask_val;
```

```
/* Install the interrupt vector */

    if(int_number > 7)
        int_number = int_number + 0x68;
    else
        int_number = int_number + 8;

    comm_struct[port_num].old_isr = getvect(int_number);

    disable();

    switch(port_num)
    {
        case 0:
            setvect(int_number,isr_0);
            break;
        case 1:
            setvect(int_number,isr_1);
            break;
        case 2:
            setvect(int_number,isr_2);
            break;
        case 3:
            setvect(int_number,isr_3);
            break;
        case 4:
            setvect(int_number,isr_4);
            break;
        case 5:
            setvect(int_number,isr_5);
            break;
        case 6:
            setvect(int_number,isr_6);
            break;
        case 7:
            setvect(int_number,isr_7);
            break;
    }

}

comm_struct[port_num].in_ptr = 0;
comm_struct[port_num].out_ptr = 0;
comm_struct[port_num].rx_int_count = 0;
comm_struct[port_num].rx_char_count = 0;

comm_struct[port_num].buff_cnt = 0;
comm_struct[port_num].tx_head = 0;
comm_struct[port_num].tx_tail = 0;
comm_struct[port_num].tx_int_count = 0;
comm_struct[port_num].tx_char_count = 0;

outportb(base_port+3,3);    /* 8-bits, 1 stop, no parity */
outportb(base_port+4,0x0b);    /* Set RTS, DTR, OUT2 */
```

```
 inportb(base_port);
 inportb(base_port);
 inportb(base_port);
 inportb(base_port);


 /* Check for and enable FIFO's if present */

outportb(base_port + 2,0x0);        /* Reset transmit and recieve FIFOs */
outportb(base_port + 2,0x1);        /* Reset transmit and recieve FIFOs */

temp = inportb(base_port+3);         /* Get current value */

outportb(base_port+3,0xbf);           /* Turns on Enhanced registers */

outportb(base_port+2,0x10);  /* Enable enhanced bits */

outportb(base_port+1,0x30);  /* Select RX FIFO table 'D' */
outportb(base_port+0,comm_struct[port_num].rx_fifo_count);

outportb(base_port+1,0xb0);  /* Select TX FIFO table 'D' */
outportb(base_port+0,comm_struct[port_num].tx_fifo_count);

if(comm_struct[port_num].tx_fifo_count > 0)
     comm_struct[port_num].tx_fifo_max = 128 - comm_struct[port_num].tx_fifo_count;
else
     comm_struct[port_num].tx_fifo_max = 0;

outportb(base_port+3,temp);  /* Restore original value */

outportb(base_port + 2,0xa1);
outportb(base_port + 2,0xa7);  /* Reset all FIFO counters to 0 */

 /* End of FIFO CODE */

 int_number = comm_struct[port_num].int_num;

 if(int_number < 8)
 {
     mask_val = 1 << int_number;
     outportb(0x21,inportb(0x21) & ~mask_val);
 }
 else
 {
     mask_val = 1 << (int_number - 8);
     outportb(0xa1,inportb(0xa1) & ~mask_val);
 }

 comm_struct[port_num].open_flag = OPEN_MAGIC;


 enable();
```

```
    outportb(base_port+1,3);    /* Enable TX/RX interrupts  */

    return 0;
}

int com_close(int port_num)
{
struct com_port *ptr;
unsigned int_number;
unsigned mask_val;

    if(comm_struct[port_num].open_flag != OPEN_MAGIC)
        return(-1);

    ptr = &comm_struct[port_num];

    int_number = ptr->int_num;

    mask_val = 1 << port_num;

    mask_val = ~mask_val;

    irq_masks[int_number] &= mask_val;  /* Clear the bit associated with us */

    if(irq_masks[int_number] == 0)  /* If we're the last one using this IRQ */
    {
        disable();

        if(int_number < 8)
        {
            mask_val = 1 << int_number;
            outportb(0x21,inportb(0x21) | mask_val);
        }
        else
        {
            mask_val = 1 << (int_number - 8);
            outportb(0xa1,inportb(0xa1) | mask_val);
        }

        if(int_number > 7)
            int_number = int_number + 0x68;
        else
            int_number = int_number + 8;

        setvect(int_number,ptr->old_isr);

        enable();
    }

    return 0;
}

void interrupt isr_0(void)
{
```

```
        common_isr(0);
}

void interrupt isr_1(void)
{
    common_isr(1);
}

void interrupt isr_2(void)
{
    common_isr(2);
}

void interrupt isr_3(void)
{
    common_isr(3);
}

void interrupt isr_4(void)
{
    common_isr(4);
}

void interrupt isr_5(void)
{
    common_isr(5);
}

void interrupt isr_6(void)
{
    common_isr(6);
}

void interrupt isr_7(void)
{
    common_isr(7);
}


void common_isr(int port_num)
{
struct com_port *ptr;
unsigned pending_irqs;

    outportb(0x1ed,1);

    enable();       /* Must allow for nesting */

    ++intflag;

    ptr = &comm_struct[port_num];

    pending_irqs = inportb(int_id);
    pending_irqs &= irq_masks[ptr->int_num];
```

```
    while(pending_irqs)
    {
        if(pending_irqs & 1)
            handle(0);
        if(pending_irqs & 2)
            handle(1);
        if(pending_irqs & 4)
            handle(2);
        if(pending_irqs & 8)
            handle(3);
        if(pending_irqs & 0x10)
            handle(4);
        if(pending_irqs & 0x20)
            handle(5);
        if(pending_irqs & 0x40)
            handle(6);
        if(pending_irqs & 0x80)
            handle(7);

        pending_irqs = inportb(int_id);
        pending_irqs &= irq_masks[ptr->int_num];
    }
    if(ptr->int_num > 8)
        outportb(0xa0,0x20);

    outportb(0x1ed,0);
    outportb(0x20,0x20);
}


void handle(int port_num)
{
struct com_port *ptr;
int stat;
int x;

    ptr = &comm_struct[port_num];

    while(1)
    {
        stat = inportb(ptr->uart_base + 2);
        if(stat  & 1)
            break;

        stat = (stat >> 1) & 3;
        switch(stat)
        {
            case 0:     /* Modem Status Interrupt */
                inportb(ptr->uart_base + 6);
                break;
            case 1:     /* TX Holding Register Empty */
                if(ptr->tx_tail == ptr->tx_head)
                {
```

```
                   ptr->tx_int_count++;
                }
                else
                {
                    ptr->tx_int_count++;

                    if((inportb(ptr->uart_base + 5) & 0x20) && (ptr->tx_tail != ptr->
tx_head))
                    {
                        for(x=0; x < ptr->tx_fifo_max; x++)
                        {
                            ptr->tx_char_count++;
                            outportb(ptr->uart_base,ptr->tx_buffer[ptr->tx_head++]);
                            if(ptr->tx_head == BUFFER_SIZE)
                                ptr->tx_head =0;

                            if(ptr->tx_head == ptr->tx_tail)
                                break;
                        }
                    }

                }
                break;

            case 2:      /* Receive Data Ready */
                ptr->rx_int_count++;
                while(inportb(ptr->uart_base + 5) & 0x01)
                {
                        ptr->rx_char_count++;
                        ptr->buffer[ptr->in_ptr++] = inportb(ptr->uart_base);
                        ptr->buff_cnt++;
                        if(ptr->in_ptr == BUFFER_SIZE)
                            ptr->in_ptr = 0;
                }
                break;
            case 3:      /* Line status interrupt */
                inportb(ptr->uart_base + 5);
                break;
        }
    }
}

int com_putch(int com_port, char c)
{
struct com_port *ptr;
long retry;

    if(comm_struct[com_port].open_flag != OPEN_MAGIC)
        return(1);

    ptr = &comm_struct[com_port];

    disable();
```

```
    if((ptr->tx_head != ptr->tx_tail) || ((inportb(ptr->uart_base +5) & 0x20) == 0))
    {
        ptr->tx_buffer[ptr->tx_tail++] = c;
        if(ptr->tx_tail == BUFFER_SIZE)
            ptr->tx_tail = 0;
    }
    else
    {
        ptr->tx_char_count++;
        outportb(ptr->uart_base,c);         /* Force feed the UART */
    }
    enable();
    return 0;

}


int com_puts(int com_port, char *str)
{
struct com_port *ptr;

    if(comm_struct[com_port].open_flag != OPEN_MAGIC)
        return(1);

    ptr = &comm_struct[com_port];

    disable();
    while(*str)
    {
        ptr->tx_buffer[ptr->tx_tail++] = *str++;
        if(ptr->tx_tail == BUFFER_SIZE)
            ptr->tx_tail = 0;
    }

    if(inportb(ptr->uart_base + 5) & 0x20)
    {
        ptr->tx_char_count++;
        outportb(ptr->uart_base,ptr->tx_buffer[ptr->tx_head++]);
        if(ptr->tx_head == BUFFER_SIZE)
            ptr->tx_head =0;
    }
    enable();

    return 0;
}

int com_check(int com_port)
{
struct com_port *ptr;

    if(comm_struct[com_port].open_flag != OPEN_MAGIC)
        return(0);

    ptr = &comm_struct[com_port];
```

```
    if(ptr->in_ptr == ptr->out_ptr)
        return 0;
    else
        return 1;
}

int com_getch(int com_port)
{
struct com_port *ptr;
int c;
long retry;

    if(comm_struct[com_port].open_flag != OPEN_MAGIC)
        return(-1);

    ptr = &comm_struct[com_port];

    retry = 100000l;

    while(retry--)
    {
        if(com_check(com_port))
            break;
    }

    if(retry == 0)
        return -1;

    disable();
    c = ptr->buffer[ptr->out_ptr++];
    if(ptr->out_ptr == BUFFER_SIZE)
        ptr->out_ptr = 0;
    enable();

    return(c & 0xff);
}

int com_gets(int com_port, char *str)
{
int c;
int count;

    count = 0;
    while(1)
    {
        c = com_getch(com_port);
        if(c == -1)
            return(1);
        if(c == '\r' && count == 0)
            continue;
        if(c == '\r')
            break;
        if(c != '\n')
```

```
        {
            *str++ = c;
            count++;
        }
    }
    *str = '\0';
    return 0;
}


int com_read(int com_port)
{
struct com_port *ptr;

    if(comm_struct[com_port].open_flag != OPEN_MAGIC)
        return(-1);

    ptr = &comm_struct[com_port];

    if(inportb(ptr->uart_base+5) & 1)
        return(inportb(ptr->uart_base) & 0xff);
    else
        return -1;
}

int com_flush(int port_num)
{
struct com_port *ptr;
int x;

    if(comm_struct[port_num].open_flag != OPEN_MAGIC)
        return(1);

    ptr = &comm_struct[port_num];
    disable();
    ptr->in_ptr = 0;
    ptr->out_ptr = 0;
    ptr->buff_cnt = 0;

    for(x=0; x < BUFFER_SIZE; x++)
        ptr->buffer[x] = 0;
    enable();
    while(com_read(port_num) != -1)
        ;

    return 0;
}
```

```
/*****************************************************************************
*
*   Name        : com8io.h
*
*   Project     : WinSystems PCM-COM8 Demonstrataion Code
*
*   Revision    : 1.00
*
*   Author      : Steve Mottin
*
*****************************************************************************
*
*   Revision History :
*
*   Date        Revision    Description
*   -------     --------    ------------------------------------------
*   10/30/02    1.00        Original
*
*****************************************************************************
*
*                   COPYRIGHT NOTICE AND DISCLAIMER
*
*   This source file is Copyright 2002 by WinSystems Inc, Arlington
*   Texas. All rights reserved except as provided below.
*
*   This file is intended to demonstrate possible usages and programming
*   of the PCM-COM8 board. It has NOT been tested extensively in a
*   multitude of environments and it is the users sole responsibility
*   to determine fitness of purpose. This file is provided 'as-is'
*   and WinSystems makes no warranty as to usability of these routines.
*
*   Users of the WinSystems PCM-COM8 may use any or all of the functions
*   in this file for incorporation in their application code utilizing the
*   PCM-COM8 without license or permission.
*
*   Other disclosure or use of this code is strictly prohibited.
*   Refer to the WinSystems Hardware/Software Sales Agreement for other
*   terms and conditions that may apply
*
*****************************************************************************
*/

#define BUFFER_SIZE 1024

#define OPEN_MAGIC 0x2795

struct com_port {

        unsigned open_flag;
        unsigned uart_base;
        unsigned int_num;
        long baud_rate;
        void interrupt (*old_isr)();
        int in_ptr;
```

```
        int out_ptr;
        int rx_fifo_count;
        unsigned rx_int_count;
        unsigned rx_char_count;
        int tx_head;
        int tx_tail;
        int tx_fifo_count;
        int tx_fifo_max;
        unsigned tx_int_count;
        unsigned tx_char_count;
        unsigned buff_cnt;
        char tx_buffer[BUFFER_SIZE];
        char buffer[BUFFER_SIZE];
        };


extern struct com_port comm_struct[8];
extern int cflag;
extern int com_flag[];
extern int intflag;

void com8_init(unsigned base_port);
int com_open(int port_num);
int com_close(int port_num);
int com_putch(int com_port, char c);
int com_puts(int com_port, char *str);
int com_check(int com_port);
int com_flush(int port_num);
int com_read(int com_port);
int com_gets(int com_port, char *str);
int com_getch(int com_port);
```

APPENDIX B

EXAR 16C854 Datasheet

XR16C854

QUAD UART WITH 128 BYTE TX and RX FIFO and AUTO RTS/CTS FLOW CONTROL

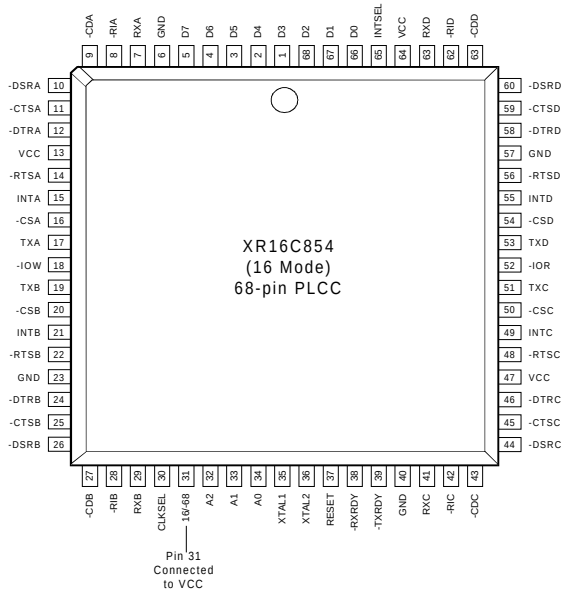
November-1999

DESCRIPTION

The XR16C854 *[1] (854) is a Universal Asynchronous Receiver and Transmitter (UART) with a dual foot print interface compatible to ST16C654/654D, ST16C554/554D and ST68C554. The 854 is an enhanced quad UART each with 128 bytes of transmit and receive FIFOs, transmit and receive FIFO counters and trigger levels, automatic hardware and software flow control, and data rates of up to 2.0 Mbps. Each UART has a set of registers that provide to the user with operating status and control, receiver error indications, and modem serial interface controls. Selectable interrupt polarity provides flexibility to meet design requirements. An internal loopback capability allows onboard diagnostics. The 854 is available in 64 pin TQFP, 68 pin PLCC, and 100 pin QFP packages. The 64 pin package offers the 16 mode interface which is compatible with the industry standard ST16C554 and ST16C654. The 68 and 100 pin packages offer an additional 68 mode interface which allows easy integration with Motorola processors. The XR16C854CV (64 pin) offers three state interrupt output while the XR16C854DV provides continuous interrupt output. The 64 pin devices do not offer TXRDY/RXRDY A-D status outputs or the prescaler clock selection option pin, CLKSEL. The 100 pin package provides additional FIFO status outputs (-TXRDY and -RXRDY A-D), separate infrared transmit data outputs (IRTX A-D) and channel C external clock input (CHCCLK). The 854 combines the 16 and 68 interface modes of previous ST16C554/654 and ST68C554/654 series in a single integrated chip.

FEATURES

- Pin compatible with the industry standard ST16C554/654, ST68C554/654 and TI's TL16C554FN and TL16C754FN
- Four enhanced UARTs, each provides:
  - Control and status register set
  - Data rates of up to 2.0 Mbps
  - 128 byte of TX and RX FIFO
  - Programmable FIFO interrupt trigger level
  - TX and RX FIFO level counter
  - Automatic RTS/CTS flow control with hysteresis
  - Automatic software Xon/Xoff flow control
  - Software selectable Baud Rate Generator prescalable clock rates of 1X or 4X
  - Standard modem serial interface or wireless infrared IrDA v1.0 encoder/decoder interface
- 100-QFP packages offer extras: TX and RX FIFO status outputs, concurrent IrDA TX outputs and external clock input for channel C
- Sleep mode (200µA typical)
- 3.3V and 5.0V supply operation



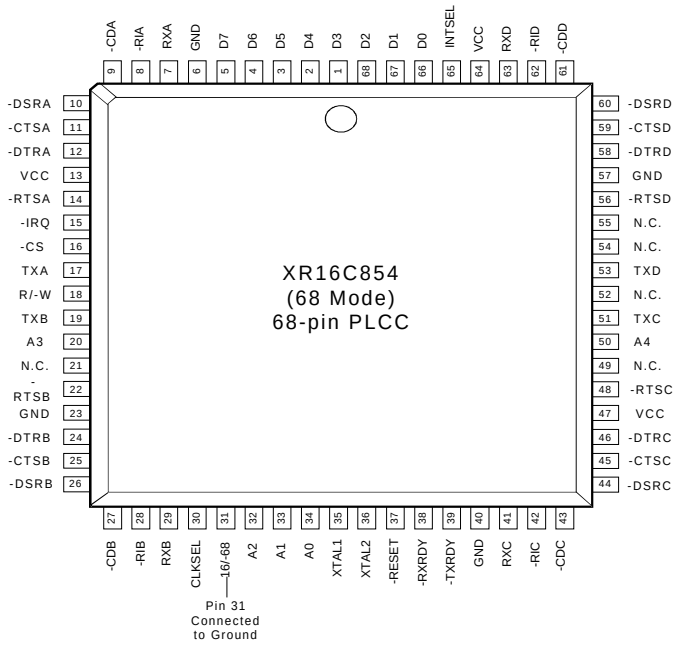
ORDERING INFORMATION

Part number	Pins	Package	Operating temperature
XR16C854CJ	68	PLCC	0° C to + 70° C
XR16C854CV	64	TQFP	0° C to + 70° C
XR16C854DCV	64	TQFP	0° C to + 70° C
XR16C854CQ	100	QFP	0° C to + 70° C

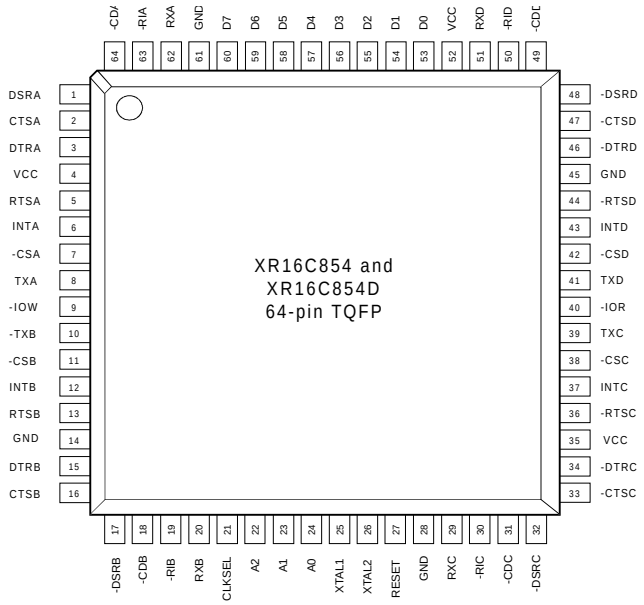
Part number	Pins	Package	Operating temperature
XR16C854IJ	68	PLCC	-40° C to + 85° C
XR16C854IV	64	TQFP	-40° C to + 85° C
XR16C854DIV	64	TQFP	-40° C to + 85° C
XR16C854IQ	100	QFP	-40° C to + 85° C

Note *1: Covered by U.S. patent #5,649,122 and patent pending.

EXAR Corporation, 48720 Kato Road, Fremont, CA 94538 • (510) 668-7000 • FAX (510) 668-7017 • www.exar.com

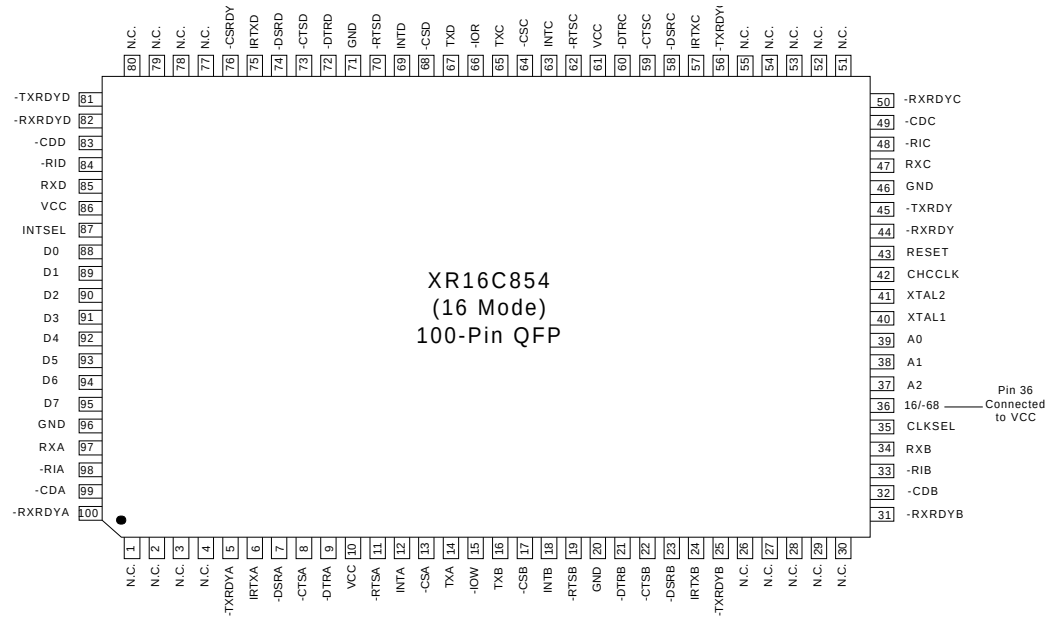


68-Pin PLCC Package in 68 Mode Bus Interface

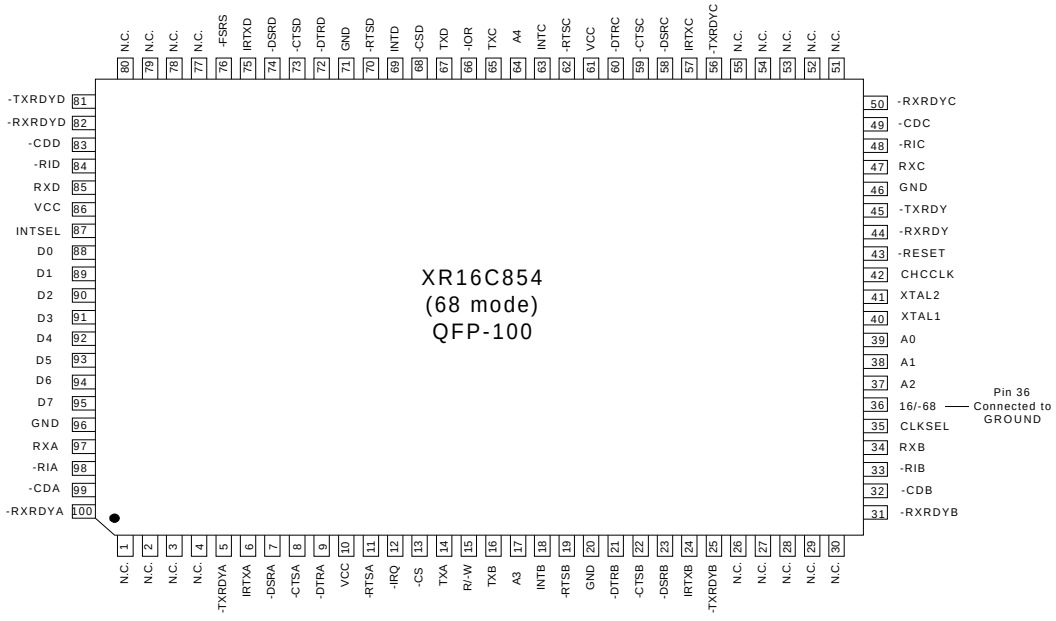


64-Pin TQFP Package in 16 Mode Bus Interface (68 mode not available)

Figure 1, Package and Pin Descriptions



100-Pin QFP Package in 16 Mode Bus Interface



100-Pin QFP Package in 68 Mode bus Interface

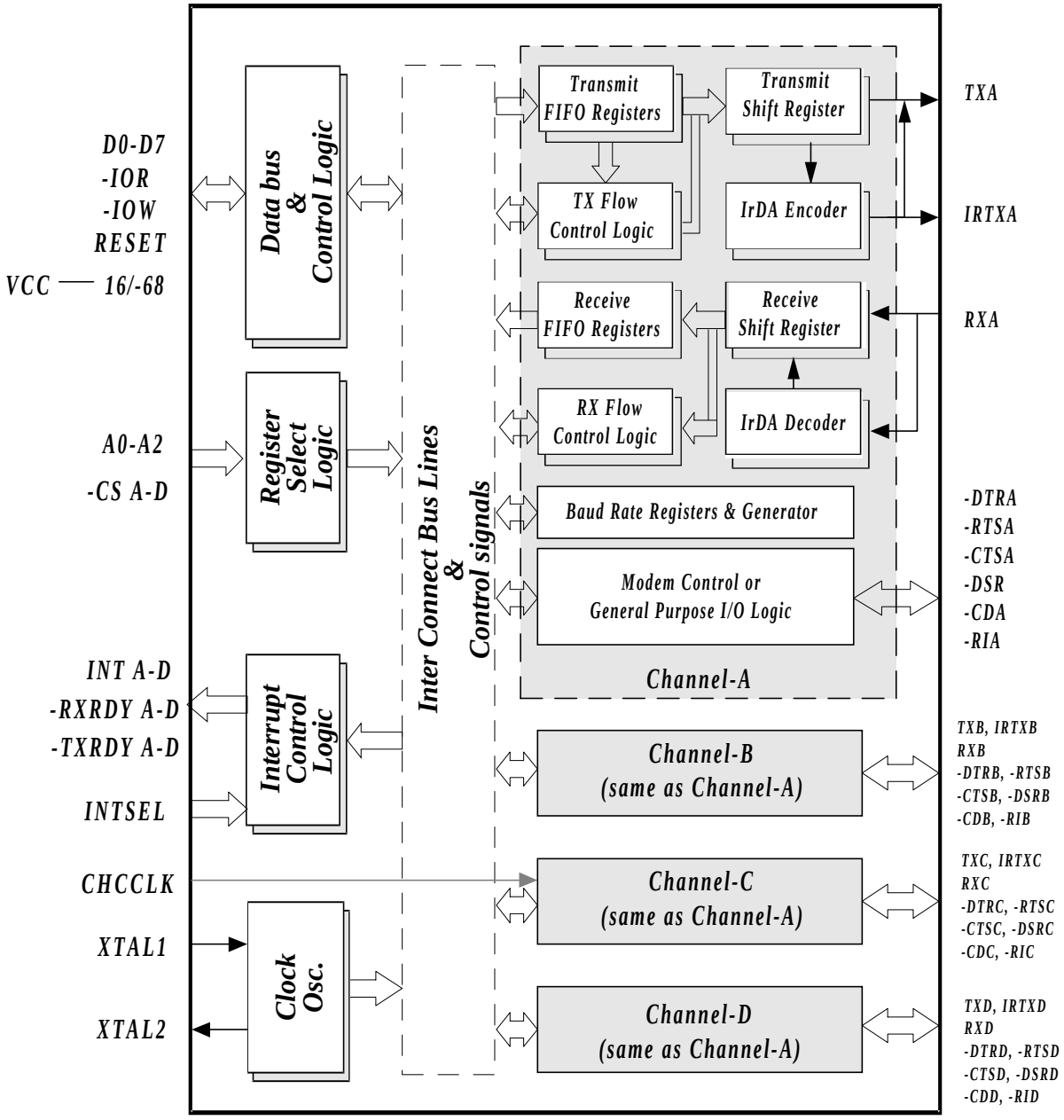


Figure 2, Block Diagram 16 Mode

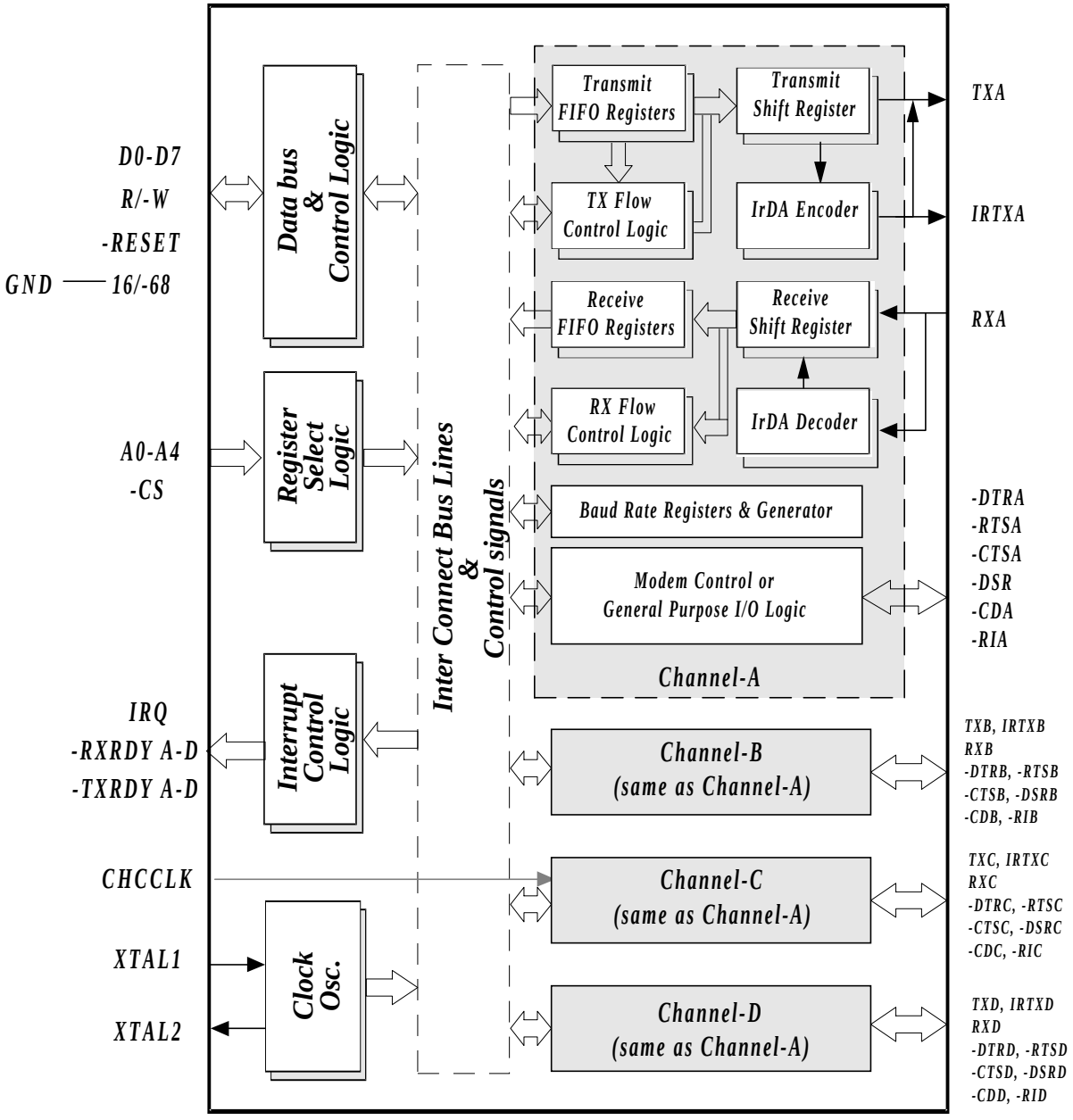


Figure 3, Block Diagram 68 Mode

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
16/-68	31	36	-	I	16/68 Interface Mode Select (input with internal pull-up). This input selects the 16 (Intel) or 68 (Motorola) bus operating mode. The functions of -IOR, -IOW, INT A-D, and -CS A-D are reassigned with the logical state of this pin. When this pin is a logic 1, the 16 mode interface is selected. When this pin is a logic 0, the 68 mode interface is selected. With this pin at logic 0 for 68 mode, -IOW is reassigned to R/-W, RESET is reassigned to -RESET, -IOR is not used, and INT A-D(s) are internally connected in wire-or'ed configuration to become IRQ output. This pin is not available on 64 pin packages which operate in the 16 mode only.
A0	34	39	24	I	Address-0 Select Bit. Internal registers address selection in 16 and 68 modes.
A1	33	38	23	I	Address-1 Select Bit. Internal registers address selection in 16 and 68 modes.
A2	32	37	22	I	Address-2 Select Bit. Internal registers address selection in 16 and 68 modes.
A3-A4	20,50	17,64	-	I	Address 3-4 Select Bits. - 68 mode only. These pins are used to address or select individual UART's (providing -CS is a logic 0). In the 16 mode, these pins are reassigned as chip selects, see -CSB and -CSC. These pins are not available on 64 pin packages which operate in the 16 mode only.
CLKSEL	30	35	21	I	Clock Select (input with internal pull-up). The pin selects the clock prescaler of 1X or 4X. The 1X clock is selected when CLKSEL is a logic 1 (connected to VCC) or the 4X is selected when CLKSEL is a logic 0 (connected to GND). MCR bit-7 can override the state of this pin following a reset or initialization (see MCR bit-7).
-CS	16	13	-	I	Chip Select. (active low) - 68 mode interface only. All four UARTs (A-D) are enabled when the -CS pin is active. A logic zero transition starts the internal read cycle to retrieve the content of a register pointed by address bits A0 and A2 while a logic one transition writes the data byte on the bus to

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
					the internal UART register. Individual UART channel is selected by address bits A3 and A4. When 16 mode is selected on 68/100 pin devices, this pin functions as -CSA, see definition under -CS A-B. This pin is not available on 64 pin packages which operate in 16 mode only.
-CS A-B -CS C-D	16,20 50,54	13,17 64,68	7,11 38,42	I	Chip Select A, B, C, D (active low) - 16 mode only. These pins enable the 854 for bus operation. They enable data transfers between the controling processor and the XR16C854 for the channel(s) addressed. Individual UART A, B, C and D are addressed by providing a logic 0 on the respective -CS A-D pin. When 68 mode is selected, the functions of these pins are re-assigned. The 68 mode functions are described under their respective name/pin headings, and -CSD pin should be connected to logic 1.
-FSRS	-	76	-	I	FIFO Status Register Select (active low, input with internal pull-up) - 100 pin QFP package only. The content of the FSTAT register is place on the data bus when this pin becomes active. D0-D3 are inverted logic states of -TXRDY A-D pins, and D4-D7 are inverted logic states of -RXRDY A-D pins. Address line is not required when reading this status register.
D0-D2 D3-D7	66-68 1-5	88-90 91-95	53-55 56-60	I/O	Data Bus (Bi-directional). These pins are the eight bit, three state data bus for transferring information to or from the controlling CPU. D0 is the least significant bit and the first data bit in a transmit or receive serial data stream.
INT A-B INT C-D	15,21 49,55	12,18 63,69	6,12 37,43	O	Interrupt A, B, C, D (active high) - 16 mode only. These pins provide individual channel interrupts, INT A-D, and are enabled with MCR bit-3 set to a logic 1. Individual interrupt is enabled in each interrupt enable register (IER) in the UART. Interrupt conditions include: receiver errors, receiver FIFO/buffer data ready, transmit FIFO/buffer empty, or when a serial input status flag is detected. In the 68 mode, the functions of these pins are reassigned. 68 mode functions are described under the their respective name/pin headings.

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
INTSEL	65	87	-	I	Interrupt Select. (active high, input with internal pull-down) - 16 mode only. When 16 mode is selected, this pin can be used in conjunction with MCR bit-3 to enable or disable the three state interrupts on INT A-D pins or override MCR bit-3 and force continuous interrupts. Interrupt outputs are enabled continuously by making this pin a logic 1. Making this pin a logic 0 allows MCR bit-3 to control the three state drivers to the interrupt output pins. In this mode, MCR bit-3 is set to a logic 1 to enable the continuous output. See MCR bit-3 description for full detail. This pin must be at logic 0 in the 68 mode. Due to pin limitations on 64 pin packages, this pin is not available. To cover this limitation, two 64 pin TQFP package versions are offered. The XR16C854D operates in the continuous interrupt enable mode by bonded this pin to VCC internally. The XR16C854 operates with MCR bit-3 software control by bonding this pin to GND.
-IOR	52	66	40	I	Input/Output Read. (active low strobe) - 16 mode only. A logic 0 transition on this pin will load the contents of an Internal register defined by address bits A0-A2 onto the XR16C854 data bus (D0-D7) for access by an external CPU. This pin should be tied to logic 1 in 68 mode operation.
-IOW	18	15	9	I	Input/Output Write. (active low strobe) - 16 mode only. A logic 0 transition on this pin will transfer the contents of the data bus (D0-D7) from the external CPU to an internal register that is defined by address bits A0/A2. When 68 mode is selected (68/100 pin devices), this pin functions as R/-W, see definition under R/W.
-IRQ	15	12	-	O	Interrupt Request (active low, open drain) - 68 mode only. The interrupts from UART channels A-D are logically wire-or'ed to function as a single IRQ interrupt output. The INTSEL input pin must be connected to logic 0 for -IRQ to function properly. This pin transitions to a logic 0 if enabled by the Interrupt Enable Register (IER) whenever a UART channel requires service. Individual channel interrupt status can be determined by addressing each channel through its associ-

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
					ated internal register, using -CS and A3-A4. An external pull-up resistor of 5-10K ohms typical must be connected between this pin and VCC.
IRTX A-B IRTX C-D	- -	6,24 57,75	- -	O	Infrared Transmit Data Output - 100 pin packages only. These pins provide separate infrared IrDA v1.0 encoded TX data outputs for UART channel A-D. The serial infrared data IRTX A-D is transmitted via these pins with added start, stop and parity bits. The IRTX signal will be a logic 0 during reset, idle (no data), or when the transmitter is disabled. MCR bit-6 selects the standard modem or infrared interface. Caution, this pin is a logic 1 after power up and prior initialization. .
CHCCLK	-	42	-	I	Channel C Clock Input - 100 pin QFP package only. This input provides the clock for UART channel C. An external 16X baud clock or the crystal oscillator's output, XTAL2, must be connected to this pin for normal operation. This input may also be used with MIDI (Musical Instrument Digital Interface) applications when an external MIDI clock is provided.
RESET -RESET	37	43	27	I	Chip Reset. In the 16 mode, a logic 1 on this pin will reset the internal registers and all the outputs. The UART transmitter output and the receiver input will be disabled (logic 1) during reset time. (See XR16C854 External Reset Conditions for initialization details.) When in 68 mode (16/-68 pin=0), this pin functions similarly but, as an inverted reset signal, -RESET.
R/-W	18	15	-	I	Read/Write Strobe (active low) - 68 mode only. This input determines the data bus operation. A logic one sets for a read operation while a logic 0 sets for a write operation.
-RXRDY	38	44	-	O	Receive FIFO Ready (active low) - 68 mode in the 68 and 100 pin packages only. The -RXRDY is a logically wire-or'ed status of all four receive channel FIFOs, RXRDY A-D. A logic 0 indicates receive data ready status, i.e. a RHR is full or a FIFO has one or more receive data characters available for unloading. This pin goes to a logic 1 when the FIFO/RHR are empty or when there is no more character available in any of the FIFO or RHR. The

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
					100 pin chip-sets provide both the combined wire “or’ed” output and individual channel RXRDY-A-D outputs. RXRDY A-D is discussed in a following paragraph. For 64/68 pin packages, individual channel RX status is read by examining individual internal registers via -CS and A0-A4 pin functions.
-RXRDY A-B -RXRDY C-D	- -	100,31 50,82	-	O	Receive FIFO Ready A-D (active low) - 100 pin package only. This function provides the receive FIFO/RHR status for individual receive channels (A-D). A logic 0 indicates there is receive data to read/unload, i.e., receive ready status with one or more RX characters available in the FIFO/RHR. This pin is a logic 1 when the FIFO/RHR is empty or when the programmed RX FIFO trigger level has not been reached.
-TXRDY	39	45	-	O	Transmit FIFO Ready (active low) - 68 mode in the 68 and 100 pin packages only. The -TXRDY output is a logically wire-or'ed status of all four transmit channel FIFOs, TXRDY A-D. A logic 0 indicates transmit buffers ready status, i.e., at least one location is empty and available in one of the TX channels (A-D) FIFOs. This pin goes to a logic 1 when all four channels have no more empty locations in the TX FIFO or THR. The 100 pin chip-sets provide both the combined wire-or’ed output and individual channel TXRDY-A-D outputs. TXRDY A-D is discussed in a following paragraph. For 64/68 pin packages, individual channel TX status can be read by examining individual internal registers via -CS and A0-A4 pin functions.
-TXRDY A-B -TXRDY C-D	- -	5,25 56,81	-	O	Transmit FIFO Ready A-D (active low) - 100 pin package only. These outputs provide the transmit FIFO/THR status for individual transmit channels (A-D). As such, an individual channel’s -TXRDY A-D buffer ready status is indicated by a logic 0, i.e., at least one location is available for data in the FIFO. This pin goes to a logic 1 when there is no empty locations in the FIFO.
XTAL1	35	40	25	I	Crystal or External Clock Input Functions as a crystal input or as an external clock input. A crystal can be connected between this pin and XTAL2 to form an internal oscillator circuit (see figure 4). Alternatively, an

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
					external clock can be connected to this pin to provide custom data rates (see Baud Rate Generator Programming and optional CHCCLK).
XTAL2	36	41	26	O	Crystal Oscillator or Buffered Clock Output Crystal oscillator output or buffered clock output. (See also XTAL1).
-CD A-B -CD C-D	9,27 43,61	99,32 49,83	64,18 31,49	I	Carrier Detect A-D (active low inputs) These inputs are associated with individual UART channels A through D. A logic 0 on this pin indicates that a carrier has been detected by the modem for that channel. These pins may be used as general purpose inputs when not used as CD signals.
-CTS A-B -CTS C-D	11,25 45,59	8,22 59,73	2,16 33,47	I	Clear to Send A-D (active low inputs) These inputs are associated with individual UART channels, A through D. A logic 0 on the -CTS pin indicates the modem or data set is ready to accept transmit data from the 854. Status can be tested by reading MSR bit-4. This pin only affects the transmit and receive operations when Auto CTS function is enabled via the Enhanced Feature Register (EFR) bit-7, for hardware flow control operation. These pins may be used as general purpose inputs when not used as CTS signals.
-DSR A-B -DSR C-D	10,26 44,60	7,23 58,74	1,17 32,48	I	Data Set Ready A-D (active low inputs) These inputs are associated with individual UART channels, A through D. A logic 0 on this pin indicates the modem or data set is powered-on and is ready for data exchange with the UART. This pin has no effect on the UART's transmit or receive operation. These pins may be used as general purpose inputs when not used as DSR signals.
-DTR A-B -DTR C-D	12,24 46,58	9,21 60,72	3,15 34,46	O	Data Terminal Ready A-D (active low inputs) These inputs are associated with individual UART channels, A through D. A logic 0 on this pin indicates that the 854 is powered-on and ready. This pin can be controlled via the modem control register. Writing a logic 1 to MCR bit-0 will set the -DTR output to logic 0, enabling the modem. This pin will

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
					be a logic 1 after writing a logic 0 to MCR bit-0, after a reset or during loopback mode. This pin has no effect on the UART's transmit or receive operation. These pins may be used as general purpose inputs when not used as DTR signals.
-RI A-B -RI C-D	8,28 42,62	98,33 48,84	63,19 30,50	I	Ring Indicator A-D (active low inputs) These inputs are associated with individual UART channels, A through D. A logic 0 on this pin indicates the modem has received a ringing signal from the telephone line. A logic 1 transition on this input pin will generate an interrupt. These pins may be used as general purpose inputs when not used as RI signals.
-RTS A-B -RTS C-D	14,22 48,56	11,19 62,70	5,13 36,44	O	Request to Send A-D (active low outputs) These outputs are associated with individual UART channels, A through D. A logic 0 on the -RTS pin indicates the transmitter has data ready and waiting to send. Writing a logic 1 in the modem control register (MCR bit-1) will set this pin to a logic 0 indicating data is available. This pin is a logic 1 after a reset and during loopback mode. This pin only affects the transmit and receive operations when Auto RTS function is enabled via the Enhanced Feature Register (EFR) bit-6, for hardware flow control operation. These pins may be used as general purpose outputs when not used as RTS signals.
RX/IRRX A-B RX/IRRX C-D	7,29 41,63	97,34 47,85	62,20 29,51	I	Receive Data / RX/IRRX A-D. These inputs are associated with individual channel's serial receive data to the XR16C854. Two user selectable interface options are available. The first option supports the standard modem interface. The second option provides an Infrared decoder interface, see figures 2/3. When interfacing to a modem interface, the receive signal must idle at logic 1, "marking", during normal operation and reset. The idle state, "marking", for the Infrared decoder interface is a logic 0. MCR bit-6 selects the standard modem or infrared interface. During the local loopback mode, the RX input pin is disabled and TX data is internally connected to the UART RX Input, internally.
TX/IRTX A-B TX/IRTX C-D	17,19 51,53	14,16 65,67	8,10 39,41	O	Transmit Data A-D. These outputs are associated with individual serial transmit

SYMBOL DESCRIPTION

Symbol	Pin 68	Pin 100	Pin 64	Signal type	Pin Description
					channel data from the 854. Two user selectable interface options are available. The first user option supports a standard modem interface. The second option provides an Infrared encoder interface, see figures 2/3. When using the standard modem interface, the TX signal will be a logic 1 during reset, idle (no data), or when the transmitter is disabled. The inactive state (no data) for the Infrared encoder/ decoder interface is a Logic 0. MCR bit-6 selects the standard modem or infrared interface. During the local loopback mode, the TX input pin is disabled and TX data is internally connected to the UART RX Input.
VCC VCC	13 47,64	10 61,86	4,21 35,52	PWR	Power supply inputs.
GND GND	6,23 40,57	96,20 46,71	14,28 45,61	PWR	Signal and power ground.

GENERAL DESCRIPTION

The XR16C854 (854) provides serial asynchronous receive data synchronization, parallel-to-serial and serial-to-parallel data conversions for both the transmitter and receiver sections. These functions are necessary for converting the serial data stream into parallel data that is required with digital data systems. Synchronization for the serial data stream is accomplished by adding start and stops bits to the transmit data to form a data character (character orientated protocol). Data integrity is insured by attaching a parity bit to the data character. The parity bit is checked by the receiver for any transmission bit errors. The electronic circuitry to provide all these functions is fairly complex especially when manufactured on a single integrated silicon chip. The XR16C854 represents such an integration with greatly enhanced features. The 854 is fabricated with an advanced CMOS process to achieve low drain power and high speed requirements.

The 854 is an upward solution that provides 128 bytes of transmit and receive FIFO memory, instead of 64 bytes provided in ST16C654, 16 bytes provided in the 16/68C554, or none in the 16/68C454. The 854 is designed to work with high speed modems and shared network environments, that require fast data processing time. Increased performance is realized in the 854 by the larger transmit and receive FIFO's. This allows the external processor to handle more networking tasks within a given time. For example, the ST16C554 with a 16 byte FIFO, unloads 16 bytes of receive data in 1.53 ms (This example uses a character length of 11 bits, including start/stop bits at 115.2Kbps). This means the external CPU will have to service the receive FIFO at 1.53 ms intervals. However with the 128 byte FIFO in the 854, the data buffer will not require unloading/loading for 12.2 ms. This increases the service interval giving the external CPU additional time for other applications and reducing the overall UART interrupt servicing time. In addition, the 4 selectable levels of FIFO trigger interrupt and automatic hardware/software flow control is uniquely provided for maximum data throughput performance especially when operating in a multichannel environment. The combination of the above greatly reduces the bandwidth requirement of the external controlling CPU, increases performance, and reduces power consumption.

The 854 combines the package interface modes of the 16C554/654 and 68/C554/654 series on a single integrated chip. The 16 mode interface is designed to operate with the Intel type of microprocessor bus while the 68 mode is intended to operate with Motorola, and other popular microprocessors. Following a reset, the 854 is downward compatible with the ST16C454/ ST68C454 or the ST68C454/ST68C554 dependent on the state of the interface mode selection pin, 16/-68.

The 854 is capable of operating up to 1.5Mbps with a 24 MHz crystal or with an external clock up to 40MHz. With a crystal of 14.7464 MHz and through a software option, the user can select data rates up to 460.8Kbps or 921.6Kbps, 8 times faster than the 16C554.

The rich feature set of the 854 is available through internal registers. Automatic hardware/software flow control, selectable transmit and receive FIFO trigger levels, selectable TX and RX baud rates, infrared encoder/decoder interface, modem interface controls, and a sleep mode are all standard features. MCR bit-5 provides a facility for turning off (Xon) software flow control with any incoming (RX) character. In the 16 mode INTSEL and MCR bit-3 can be configured to provide a software controlled or continuous interrupt capability. Due of pin limitations for the 64 pin 854 this feature is offered by two different QFP packages. The XR16C854DCV operates in the continuous interrupt enable mode by bonded INTSEL to VCC internally. The XR16C854CV operates in conjunction with MCR bit-3 by bonding INTSEL to GND internally.

The 68 and 100 pin XR16C854 packages offer a clock prescaler select pin to allow system/board designers to preset the default baud rate table on power up. The CLKSEL pin selects the 1X or 4X prescaler for the baud rate generator. It can then be overridden following initialization by MCR bit-7.

The 100 pin packages offer several enhances features. These features include an CHCCLK clock input, an internal FIFO monitor register, and separate IrDA TX outputs. The CHCCLK must be connected to the XTAL2 pin for normal operation or to external MIDI (Music Instrument Digital Interface) oscillator for MIDI applications. A separate register is provided for monitoring the real time status of the FIFO signals -TXRDY and -RXRDY for each of the four UART channels (A-D).

This reduces polling time involved in accessing individual channels. The 100 pin QFP package also offers, four separate IrDA (Infrared Data Association Standard) outputs for Infrared applications. These outputs are provided in addition to the standard asynchronous modem data outputs.

FUNCTIONAL DESCRIPTIONS

Interface Options

Two user interface modes are selectable for the 854 package. These interface modes are designated as the “16 mode” and the “68 mode.” This nomenclature corresponds to the early 16C554/654 and 68C554/654 package interfaces respectively.

The 16 Mode Interface

The 16 mode configures the package interface pins for connection as a standard 16 series (Intel) device and operates similar to the standard CPU interface available on the 16C554/654. In the 16 mode (pin 16/-68 logic 1) each UART is selected with individual chip select (CSx) pins as shown in Table 1 below.

-CSA	-CSB	-CSC	-CSD	UART CHANNEL
1	1	1	1	None
0	1	1	1	A
1	0	1	1	B
1	1	0	1	C
1	1	1	0	D

Table 1, Serial Port Channel Selection, 16 Mode Interface.

The 68 Mode Interface

The 68 mode configures the package interface pins for connection with Motorola, and other popular microprocessor bus types. The interface operates similar to the 68C554/654. In this mode the 854 decodes two additional addresses, A3-A4 to select one of the four UART ports. The A3-A4 address decode function is used only when in the 68 mode (16/-68 logic 0), and is shown in Table 2 below.

-CS	A4	A3	UART CHANNEL
1	N/A	N/A	None
0	0	0	A
0	0	1	B
0	1	0	C
0	1	1	D

Table 2, Serial Port Channel Selection, 68 Mode Interface.

Internal Registers

Each UART in the 854 provides a total of 21 internal registers for monitoring and control. Thesegresisters are shown in Table 3 on next page are similar to those already available in the standard 16C554/16C654. These registers function as data holding registers (THR/RHR), interrupt status and control registers (IER/ISR), a FIFO control register (FCR), line status and control registers (LCR/LSR), modem status and control registers (MCR/MSR), programmable data rate (clock) control registers (DLL/DLM), and a general purpose scratched pad register (SPR). Beyond the general 16C554/654 features and capabilities, the 854 added enhanced feature registers (EFR, Xon/Xoff-1, Xon/Xoff-2, FCTR, TRG, EMSR) that provides on board hardware/software flow control. Register functions are more fully described in the following paragraphs.

ADDRESS			BUS OPERATION	
A2	**A1**	**A0**	**READ ONLY**	**WRITE ONLY**
General Registers are accessible only when LCR is not 0xBF.				
0	0	0	Receive Holding Register	Transmit Holding Register
0	0	1	Interrupt Enable Register	Interrupt Enable Register
0	1	0	Interrupt Status Register	FIFO Control Register
0	1	1	Line Control Register	Line Control Register
1	0	0	Modem Control Register	Modem Control Register
1	0	1	Line Status Registe	N.A.
1	1	0	Modem Status Register	N.A.
1	1	1	Scratch Pad Register	Scratch Pad Register
1	1	1	FIFO Level Counter(see note 1)	Enhanced Mode Select Register(see note 1)
Baud Rate Divisor Registers are accessible only when LCR bit-7 is logic 1 and not 0xBF.				
0	0	0	LSB of Divisor Latch	LSB of Divisor Latch
0	0	1	MSB of Divisor Latch	MSB of Divisor Latch
Enhanced Registers are accessible only when LCR is set to 0xBF.				
0	0	0	FIFO Level Counter	FIFO Trigger Level
0	0	1	Feature Control Register	Feature Control Register
0	1	0	Enhanced Feature Register	Enhanced Feature Register
1	0	0	Xon-1 Word	Xon-1 Word
1	0	1	Xon-2 Word	Xon-2 Word
1	1	0	Xoff-1 Word	Xoff-1 Word
1	1	1	Xoff-2 Word	Xoff-2 Word
Channel A-D FIFO Status Register is accessible only when -FSRS pin is active.				
X	X	X	RXRDY A-D and TXRDY A-D	N.A.

Note 1: FIFO Level Counter and Enhanced Mode Select Register are accessible only when FTCR bit-6=1. When FTCR bit-6=0, the Scratchpad Registe is available.

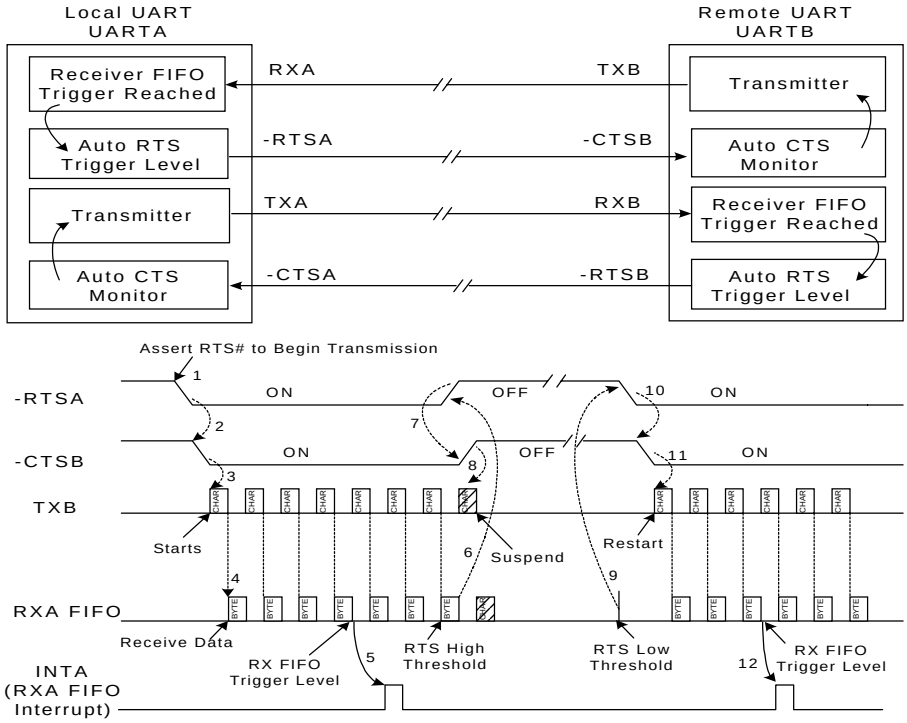
Table 3, Internal Registers Summary

Hardware (RTS/CTS) Flow Control Operation

Automatic hardware or RTS and CTS flow control is used to prevent data overrun to the local receiver FIFO and remote receiver FIFO. The -RTS output pin is used to request remote unit to suspend/restart data transmission while the -CTS input pin is monitored to suspend/restart local transmitter. The auto RTS and auto CTS flow control features are individually selected to fit specific application requirement and enabled through EFR bit-6 and 7. The auto RTS function must be started by asserting -RTS pin (MCR bit-1=1 after it is enabled. The figure below shows how it works.

Two interrupts associated with RTS and CTS flow control have been added to give indication when -RTS pin or -CTS pin is de-asserted during operation. The RTS and CTS interrupts must be first enabled by EFR bit-4, and then enabled individually by IER bit-6 and 7.

Automatic hardware flow control is selected by setting bits 6 (RTS) and 7 (CTS) of the EFR register to logic 1. If CTS# pin transitions from logic 0 to logic 1 indicting a flow control request, ISR bit-5 will be set to logic 1 (if enabled via IER bit 6-7), and the UART will suspend TX transmissions as soon as the stop bit of the character in process is shifted out. Transmission is resumed after the -CTS input returns to logic 0, indicating more data may be sent.



The local UART (UARTA) starts data transfer by asserting -RTSA (1). -RTSA is normally connected to -CTSB (2) of remote UART (UARTB). -CTSB allows its transmitter to send data (3). TXB data arrives and fills UART-A receive FIFO (4). When RXA data fills up to its receive FIFO trigger level, UARTA activates its RXA data ready interrupt (5) and continues to receive and put data into its FIFO. If interrupt service latency is long and data is not being unloaded, UARTA monitors its receive data fill level to match the upper threshold of RTS delay and de-assert -RTSA (6). -CTSB follows (7) and request UARTB transmitter to suspend data transer. UART-B stops or finishes sending the data bits in its transmit shift register (8). When receive FIFO data in UARTA is unloaded to match the lower threshold of RTS delay (9), UARTA re-assert -RTSA (10) -CTSB recognizes the change (11) and restarts its transmitter and data flow again until next RX trigger (12). This same event applies tothe reverse direction when UARTA sends data to UARTB with -RTSB and -CTSA controlling the data flow.

The 854 offer programmable receive FIFO level flow control trigger hysteresis while maintaining compatibility to ST16C654. The hysteresis level is programmable from 0x00 to 0x7F to provide the user for best data throughput optimization.

With the Auto RTS function enabled, -RTS pin will go to logic 1 turning RTS off when the RX FIFO level has reached the upper hysteresis limit that is set to be several bytes of data above the RX FIFO trigger level. This delay action of suspending remote transmitter effectively keeps data coming hence increases data throughput. The -RTS pin will return to a logic 0 setting RTS on when the RX FIFO level reaches the lower hysteresis limit. The receiver continues to accept data until receive FIFO gets completely full. The Auto RTS function must be started by asserting -RTS pin to logic 0 (RTS On). See EMSR bit 4 and 5 for a complete programming table for the hysteresis level.

Software Flow Control

When software flow control is enabled, the 854 compares one or two sequential receive data characters with the programmed Xon or Xoff-1,2 character value(s). If receive character(s) (RX) match the programmed values, the 854 will halt transmission (TX) as soon as the current character(s) has completed transmission. When a match occurs, the receive ready (if enabled via Xoff IER bit-5) flag will be set and the interrupt output pin (if receive interrupt is enabled) will be activated. Following a suspension due to a match of the Xoff characters values, the 854 will monitor the receive data stream for a match to the Xon-1,2 character value(s). If a match is found, the 854 will resume operation and clear the flags (ISR bit-4).

The 854 offers a special Xon mode via MCR bit-5. The initialized default setting of MCR bit-5 is a logic 0. In this state Xoff and Xon will operate as defined above. Setting MCR bit-5 to a logic 1 sets a special operational mode for the Xon function. In this case Xoff operates normally however, transmission (Xon) will resume with the next character received, i.e., a match is declared simply by the receipt of an incoming (RX) character.

Reset initially sets the contents of the Xon/Xoff 8-bit flow control registers to a logic 0. Following reset the user can write any Xon/Xoff value desired for software flow control. Different conditions can be set to detect Xon/Xoff characters and suspend/resume transmissions. When double Xon/Xoff characters are selected, the 854 compares two consecutive receive characters with two software flow control values (Xon1, Xon2, Xoff1, Xoff2) and controls TX transmissions accordingly. Under the above described flow control mechanisms, flow control characters are not placed (stacked) in the user accessible RX data buffer or FIFO.

In the event that the receive buffer is overfilling and flow control needs to be executed, the 854 automatically sends an Xoff message (when enabled) via the serial TX output to the remote modem. The 854 sends the Xoff-1,2 characters two character times after received data passes the programmed FIFO trigger level. To clear this condition, the 854 will transmit the programmed Xon-1,2 characters as soon as receive data drops below the next lower programmed trigger level or programmed RTS hysteresis level.

Special Feature Software Flow Control

A special feature is provided to detect a character when bit-5 is set in the Enhanced Feature Register (EFR). When the character is detected, it will be placed on the user accessible data stack along with normal incoming RX data. This condition is selected in conjunction with EFR bits 0-3. Note that software flow control should be turned off when using this special mode by setting EFR bit 0-3 to a logic 0.

The 854 compares each incoming receive character with Xoff-2 data. If a match exists, the received data will be transferred to FIFO and ISR bit-4 will be set to indicate detection of special character (see Figure 9). Although the Internal Register Table shows each X-Register with eight bits of character information, the actual number of bits is dependent on the programmed word length. Line Control Register (LCR) bits 0-1 defines the number of character bits, i.e., either 5 bits, 6 bits, 7 bits, or 8 bits. The word length selected by LCR bits 0-1 also determines the number of bits that will be used for the special character comparison. Bit-0 in the X-registers corresponds with the LSB bit for the receive character.

Xon Any Feature

A special feature is provided to return the Xoff flow control to the inactive state following its activation. In this mode any RX character received will return the Xoff flow control to the inactive state so that transmissions may be resumed with a remote buffer. This feature is more fully defined in the Software Flow Control section.

Device Identification

The XR16C854 provides a Device Identification and Device Revision code to distinguish the part from others.

To read the identification number from the part, it is required to set the baud rate generator divisor, LCR bit-7, to logic 1 and then set the baud rate generator DLL and DLM registers to 0x00. Now, reading the content of the DLM will provide 0x14 for XR16C854 part and reading the content of the DLL will provide the revision of the part; for example, a reading of 0x01 means revision A.

FIFO Interrupts, Trigger Levels and Time-out

The 128 byte transmit and receive data FIFO's are enabled by the FIFO Control Register (FCR) bit-0. The receive FIFO is actually 11 bit wide to hold 3 error bits for each character received. The 854 provides independent FIFO interrupt trigger control for both receiver and transmitter. The transmit and receive trigger levels are set to 1 following a reset to be compatible to ST16C554. The user must activate EFR bit-4 to a logic 1 before setting the transmit trigger levels. The receiver FIFO section includes a receive time-out interrupt function to ensure data is delivered to the external processor. This occurs whenever the receive trigger level has not reached. The time-out delay is about 4 characters period.

Seven interrupts are provided by the UART to indicate activities in the UART. These interrupts are enabled in IER register bits 0-7. Following a reset the transmitter interrupt is disabled. Upon enabling the transmit empty interrupt, the 854 will issue an interrupt to indicate that transmit holding register is empty. This interrupt must be serviced prior to continuing operations. The receiver provides 2 interrupts for receive data ready and a receive data time-out function. The receive data time-out interrupt is caused when the number of data bytes in the FIFO has not reached the trigger level and the receiver has been idle (no incoming data, FIFO not accessed) for about four character period. This ensure data are passed on to the CPU. The LSR register provides the current singular highest priority interrupt only for transmitter and receiver status and errors. There is the MSR interrupt that indicates the change of state on the modem input pins. It should be noted that CTS and RTS interrupts have the lowest interrupt priority. A condition can exist where a higher priority interrupt may mask the lower priority CTS/RTS interrupt(s). Only after servicing the higher pending interrupt will the lower priority CTS/ RTS interrupt(s) be reflected in the status register. Servicing the interrupt, without investigating further interrupt conditions, can result in data errors.

When two interrupt conditions have the same priority, it is important to service these interrupts correctly. Receive data ready and receive time-out have the same interrupt priority (when enabled by IER bit-0). The receiver issues an interrupt after the number of characters have reached the programmed trigger level. In this case, the FIFO may hold more characters than the programmed trigger level. Following the removal of a data byte, the user should recheck LSR bit-0 for additional characters. A receive time-out will not occur if the receive FIFO is empty. The time out counter is reset at the center of each stop bit received or each time the receive holding register (RHR) is read. The actual time out value is T (**T**ime out length in bits) = 4 X P (**P**rogrammed word length) + 12. To convert the time out value to a character value, the user has to consider the complete word length, including data information length, start bit, parity bit, and the size of stop bit, i.e., 1X, 1.5X, or 2X bit times.

Example -A:
If the user programs a word length of 7, with no parity and one stop bit, the time out will be:
T = 4 X 7(programmed word length) +12 = 40 bit times. The character time will be equal to 40 / 9 = 4.4 characters, or as shown in the fully worked out example: T = [(programmed word length = 7) + (stop bit = 1) + (start bit = 1) = 9]. 40 (bit times divided by 9) = 4.4 characters.

Example -B:
If the user programs the word length = 7, with parity and one stop bit, the time out will be:
T = 4 X 7 (programmed word length) + 12 = 40 bit times.
Character time = 40 / 10 [(programmed word length = 7) + (parity = 1) + (stop bit = 1) + (start bit = 1)] = 4 characters.
In the 16 mode for 68/100 pin packages, the system/board designer can optionally provide software controlled three state interrupt operation. This is accomplished by INTSEL and MCR bit-3. When INTSEL interface pin is left open or made a logic 0, MCR bit-3 controls the three state interrupt outputs, INT A-D. When INTSEL is a logic 1, MCR bit-3 has no effect on the INT A-D outputs and the package operates with interrupt outputs enabled continuously.

Programmable Baud Rate Generator (BRG)

The 854 supports high speed modem technologies that have increased input data rates by employing data compression schemes typically of 4 to 1 ratio. For example a 56Kbps modem that employs data compression may require a 230.42Kbps of DTE input data rate. A 128.0Kbps ISDN modem that supports data compression may need an input data rate of 460.8Kbps. The 854 can support a standard data rate of 921.6Kbps.

A single baud rate generator is provided for the transmitter and receiver, allowing independent TX/RX channel control. The programmable Baud Rate Generator is capable of accepting a clock up to 24 MHz from the on-chip oscillator circuit for 1.5 Mbps data rate, or up a 32 MHz external clock on pin XTAL1 for up to 2.0 Mbps typical data rate. The internal clock oscillator is designed to use an industry standard microprocessor crystal (parallel resonant with 10-22 pF load) connected externally between the XTAL1 and XTAL2 pins (see Figure 4). Alternatively, an external clock can be connected to the XTAL1 pin to clock the internal baud rate generator for standard or custom rates. (see Baud Rate Generator Programming and Figure 7 for data rate performance curves on the AC Characteristics page).

The 854 has a clock prescaler that divides the crystal or external clock by 1 or 4 . This clock feeds the input of the BRG. The generator divides this input clock by any divisor from 1 to 2^{16} -1. Further division of this clock provides two table rates to support low and high data rate applications using the same clock frequency. After a hardware reset, the 854 defaults to a baud data rate table acccording to the state of the CLKSEL. pin. A logic 1 on CLKSEL will set the 1X (divide by 1) clock table, whereas, a logic 0 will set the 4X (divide by 4) clock table. Following the baud clock rate selection during initialization, the rate tables can be changed by the internal register, MCR bit-7. Setting MCR bit-7 to a logic 1 when CLKSEL is a logic 1 provides an additional divide by 4 whereas, setting MCR bit-7 to a logic 0 only divides by 1. (See Table 4 and Figure 5). Customized Baud Rates can be achieved by selecting the proper divisor values for the MSB and LSB sections of baud rate generator.

Programming the Baud Rate Generator Registers DLM (MSB) and DLL (LSB) provides a user capability for selecting the desired final baud rate. The example in Table 4 below, shows the two selectable baud rate tables available when using a 7.3728 MHz crystal.

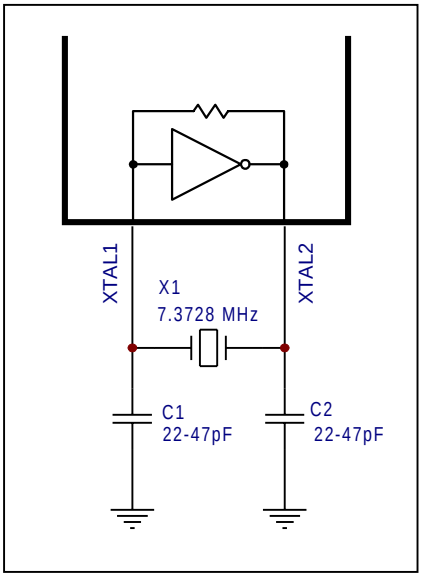
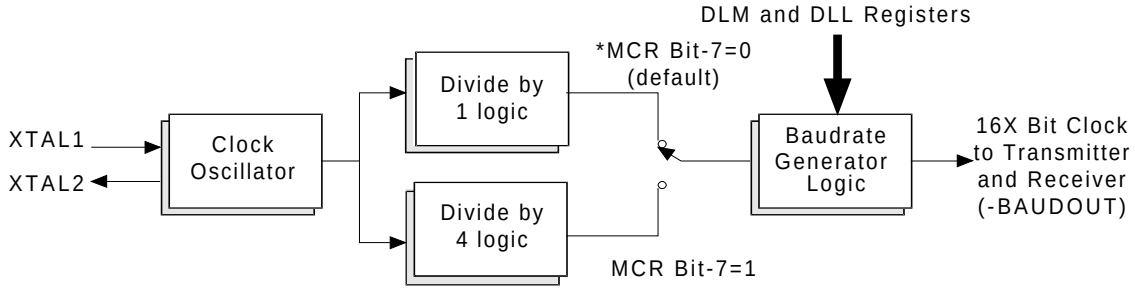


Figure 4, Crystal oscillator connection

Output Baud Rate MCR BIT-7=1	Output Baud Rate MCR Bit-7=0	User 16 x Clock Divisor (Decimal)	User 16 x Clock Divisor (HEX)	DLM Program Value (HEX)	DLL Program Value (HEX)
50	200	2304	900	09	00
300	1200	384	180	01	80
600	2400	192	C0	00	C0
1200	4800	96	60	00	60
2400	9600	48	30	00	30
4800	19.2K	24	18	00	18
9600	38.4k	12	0C	00	0C
19.2k	76.8k	6	06	00	06
38.4k	153.6k	3	03	00	03
57.6k	230.4k	2	02	00	02
115.2k	460.8k	1	01	00	01

Table 4, Baud Rate Generator Programming Table with a 7.3728 MHz Crystal or External Clock



*Note: After power up or reset, the selection made by CLKSEL pin may be overriden with MCR bit-7.

Figure 5, Baud Rate Generator Circuitry

DMA Operation

The 854 FIFO trigger level provides additional flexibility to the user for block mode operation. LSR bits 5-6 provide an indication when the transmitter is empty or has an empty location(s). The user can optionally operate the transmit and receive FIFO's in the DMA mode (FCR bit-3). When the transmit and receive FIFO's are enabled and the DMA mode is deactivated (DMA Mode "0"), the 854 activates the interrupt output pin for each data transmit or receive operation. When DMA mode is activated (DMA Mode "1"), the user takes the advantage of block mode operation by loading or unloading the FIFO in a block sequence determined by the preset trigger level. In this mode, the 854 sets the interrupt output pin when characters in the transmit FIFO's are below the transmit trigger level, or the characters in the receive FIFO's are above the receive trigger level.

Sleep Mode

The 854 is designed to operate with low power consumption. A special sleep mode is included to further reduce power consumption when the chip is not being used. With EFR bit-4 and IER bit-4 enabled (set to a logic 1), the 854 enters the sleep mode but resumes normal operation when a start bit is detected, a change of state on any of the modem input pins RX, -RI, -CTS, -DSR, -CD, or transmit data is provided by the user. If the sleep mode is enabled and the 854 is awakened by one of the conditions described above, it will return to the sleep mode automatically after the last character is transmitted or read by the user. In any case, the sleep mode will not be entered while an interrupt(s) is pending. The 854 will stay in the sleep mode of operation until it is disabled by setting IER bit-4 to a logic 0.

Loopback Mode

The internal loopback capability allows onboard diagnostics. In the loopback mode the normal modem interface pins are disconnected and re-configured for loopback internally. MCR register bits 0-3 are used for controlling loopback diagnostic testing. In the loopback mode OP1 and OP2 in the MCR register (bits 3/2) control the modem -RI and -CD inputs respectively. MCR signals -DTR and -RTS (bits 0-1) are used to control the modem -CTS and -DSR inputs respectively. The transmitter output (TX) and the receiver input (RX) are disconnected from their associated interface pins, and instead are connected together internally (See Figure 6). The -CTS, -DSR, -CD, and -RI are disconnected from their normal modem control inputs pins, and instead are connected internally to -DTR, -RTS, -OP1 and -OP2. Loopback test data is entered into the transmit holding register via the user data bus interface, D0-D7. The transmit UART serializes the data and passes the serial data to the receive UART via the internal loopback connection. The receive UART converts the serial data back into parallel data that is then made available at the user data interface, D0-D7. The user optionally compares the received data to the initial transmitted data for verifying error free operation of the UART TX/RX circuits.

In this mode, the receiver and transmitter interrupts are fully operational. The Modem Control Interrupts are also operational. However, the interrupts can only be read using lower four bits of the Modem Control Register (MCR bits 0-3) instead of the four Modem Status Register bits 4-7. The interrupts are still controlled by the IER.

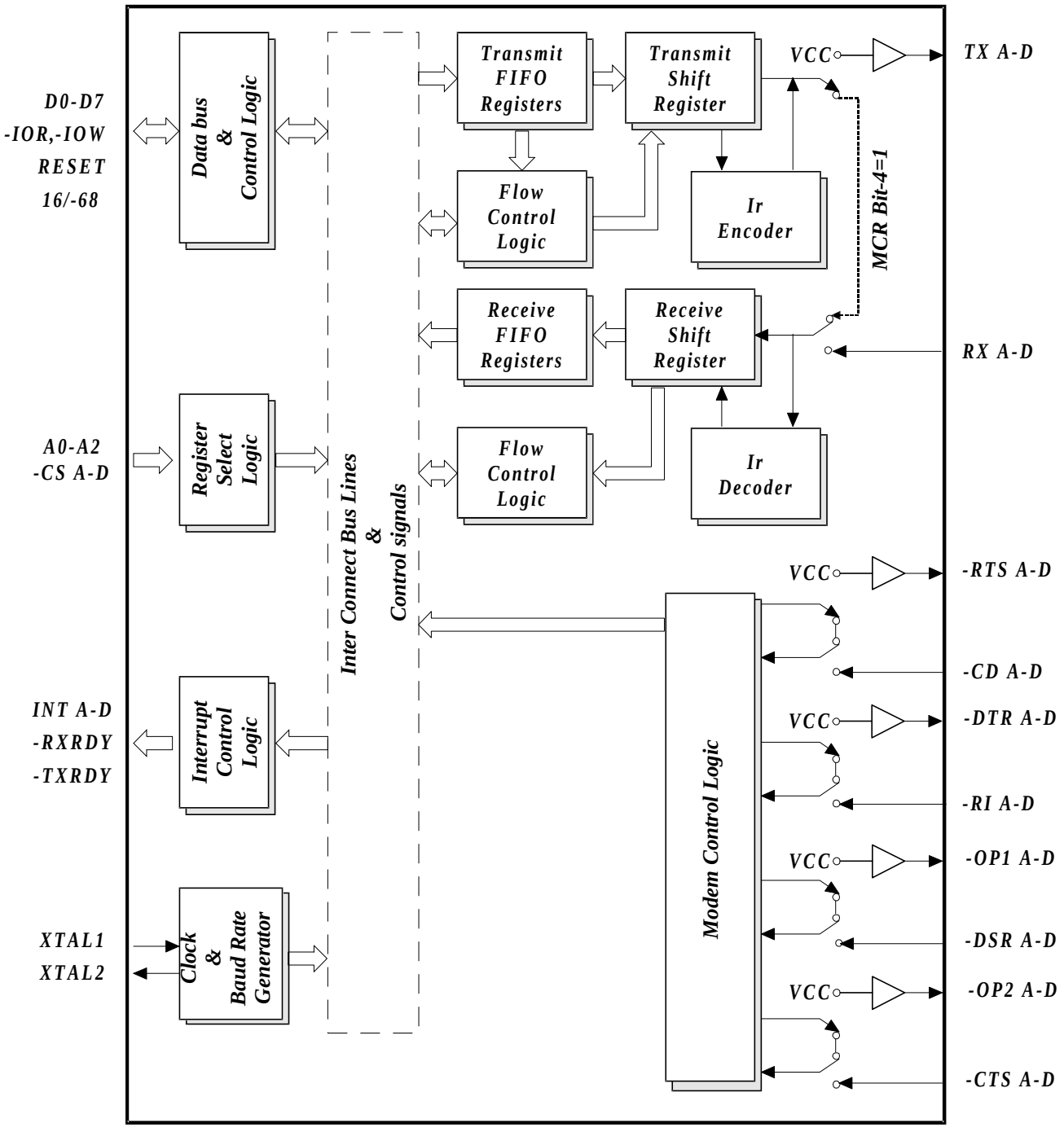


Figure 6, Internal Loop Back Mode Diagram, shown for channel A/B/C/D.

Internal Registers Descriptions

The following table delineates the assigned bit functions for the 854 internal registers. The assigned bit functions are more fully defined in the following paragraphs.

Bus Read or Write	Address A2 A1 A0	Register [Default] Note *2	BIT-7	BIT-6	BIT-5	BIT-4	BIT-3	BIT-2	BIT-1	BIT-0
General Registers are accessible only when LCR bit-7 is not 0xBF. Shaded bits are available only when they are enabled by EFR bit-4.										
Read	0 0 0	RHR [XX]	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
Write	0 0 0	THR [XX]	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
Read / Write	0 0 1	IER [00]	0/ -CTS interrupt	0/ -RTS interrupt	0/ Xoff interrupt	0/ Sleep mode	Modem status interrupt interrupt	Receive line status	Transmit holding register	Receive holding register
Write	0 1 0	FCR [00]	RCVR trigger (MSB)	RCVR trigger (LSB)	0/TX trigger (MSB)	0/TX trigger (LSB)	DMA mode select	XMIT FIFO reset	RCVR FIFO reset	FIFO enable
Read	0 1 0	ISR [01]	0/ FIFOs enabled	0/ FIFOs enabled	0/Int Status Bit-5	0/Int Status Bit-4.	Int Status Bit-3	Int Status Bit-2	Int Status Bit-1	Int status Bit-0
Read / Write	0 1 1	LCR [00]	BRG Prescaler enable	Set Break	Set parity	Even parity	Parity enable	Stop bits	Word length bit-1	Word length bit-0
Read / Write	1 0 0	MCR [00]	Clock select enable	0/ IRRT Any	0/ Xon	Loop back INT	(-OP2) 3-state	(-OP1)	-RTS	-DTR
Read	1 0 1	LSR [60]	0/ FIFO error	TSR empty	THR empty	RX Break	RX Framing error	RX Parity error	RX Overrun error	Receive data ready
Read	1 1 0	MSR [00]	-CD	-RI	-DSR	-CTS -CD	Delta -RI	Delta -DSR	Delta -CTS	Delta
Read / Write	1 1 1	SPR [FF]	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
Read	1 1 1	FLVL [00] (note 3)	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
Write	1 1 1	EMSR [00] (note 3)	Reserved	Reserved	RTS Hyst bit-5	RTS Hyst bit-4	Reserved	Reserved	FIFO Count bit-1	FIFO Count bit-0
Baud Rate Gen. Registers are accessible only when LCR bit-7 is set to logic 1 and not 0xBF.										
Read / Write	0 0 0	DLL [00]	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
Read / Write	0 0 1	DLM [00]	Bit-15	Bit-14	Bit-13	Bit-12	Bit-11	Bit-10	Bit-9	Bit-8

Internal Registers Descriptions continues

Bus Read or Write	Address A2 A1 A0	Register [Default] Note *2	BIT-7	BIT-6	BIT-5	BIT-4	BIT-3	BIT-2	BIT-1	BIT-0
Enhanced Registers are accessible only when LCR register is 0xBF.										
Read / Write	0 0 0	TRG [00]	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
Read / Write	0 0 1	FCTR [00]	Rx/Tx Mode	SPR Swap	Trig Table Bit-1	Trig Table Bit-0	Auto RS485 control	IrRx Inv.	RTS Hyst Bit-1	RTS Hyst Bit-0
Read / Write	0 1 0	EFR [00]	Auto -CTS	Auto -RTS select	Special Char. Bits 4-7,	Enable IER Control ISR, FCR Bits 4-5, MCR Bits 5-7	Cont-3 Tx,Rx Control	Cont-2 Tx,Rx Control	Cont-1 Tx,Rx Control	Cont-0 Tx,Rx
Read / Write	1 0 0 1 0 1 1 1 0 1 1 1	Xon-1 [00] Xon-2 [00] Xoff-1 [00] Xoff-2 [00]	Bit-7 Bit-15 Bit-7 Bit-15	Bit-6 Bit-14 Bit-6 Bit-14	Bit-5 Bit-13 Bit-5 Bit-13	Bit-4 Bit-12 Bit-4 Bit-12	Bit-3 Bit-11 Bit-3 Bit-11	Bit-2 Bit-10 Bit-2 Bit-10	Bit-1 Bit-9 Bit-1 Bit-9	Bit-0 Bit-8 Bit-0 Bit-8

FIFO Status Register is only selected by the -FSRS pin, no A0-A2 address lines required.										
Read only	X X X	FSTAT	RXRDY-D Ready	RXRDY-C Ready	RXRDY-B Ready	RXRDY-A Ready	TXRDY-D Ready	TXRDY-C Ready	TXRDY-B Ready	TXRDY-A Ready

Note 2: The value between the square brackets represents the register's initialized HEX value.

Note 3: The FLVL and EMSR registers are only accessible when FTCR register bit-6 is set to logic 1, in this case, SPR register would not be available.

Transmit (THR) and Receive (RHR) Holding Registers

The serial transmitter section consists of an 8-bit Transmit Hold Register (THR) and Transmit Shift Register (TSR). The status of the THR is provided in the Line Status Register (LSR). Writing to the THR transfers the contents of the data bus (D7-D0) to the THR. The THR empty flag in the LSR register will be set to a logic 1 when the last character in the THR is transferred to the TSR. LSR bit-6 will be set when the last character has been shifted out of TSR.

The serial receive section also contains an 8-bit Receive Holding Register, RHR. Receive data is removed from the 854 and receive FIFO by reading the RHR register. The receive section provides a mechanism to prevent false starts. On the falling edge of a start or false start bit, an internal receiver counter starts counting clocks at 16x clock rate. After 7 1/2 clocks the start bit time should be shifted to the center of the start bit. At this time the start bit is sampled and if it is still a logic 0 it is validated. Evaluating the start bit in this manner prevents the receiver from assembling a false character. Receiver status codes will be posted in the LSR.

Interrupt Enable Register (IER)

The Interrupt Enable Register (IER) masks the interrupts from receiver ready, transmitter empty, line status and modem status registers. These interrupts would normally be seen on the INT A-D output pins in the 16 mode, or on WIRE-OR'ed IRQ output pin, in the 68 mode.

IER versus Receive FIFO Interrupt Mode Operation

When the receive FIFO (FCR BIT-0 = a logic 1) and receive interrupts (IER BIT-0 = logic 1) are enabled, the receive interrupts and register status will reflect the following:

A) The receive data available interrupts are issued to the external CPU when the FIFO has reached the programmed trigger level. It will be cleared when the FIFO drops below the programmed trigger level.

B) FIFO status will also be reflected in the user accessible ISR register when the FIFO trigger level is reached. Both the ISR register status bit and the interrupt will be cleared when the FIFO drops below the trigger level.

C) The data ready bit (LSR BIT-0) is set as soon as a character is transferred from the shift register to the receive FIFO. It is reset when the FIFO is empty.

IER versus Receive/Transmit FIFO Polled Mode Operation

When FCR BIT-0 equals a logic 1; resetting IER bits 0-3 enables the 854 in the FIFO polled mode of operation. Since the receiver and transmitter have separate bits in the LSR either or both can be used in the polled mode by selecting respective transmit or receive control bit(s).

A) LSR BIT-0 indicates there is data in RHR/RX FIFO.

B) LSR BIT 1-4 provides the type of receive data errors encountered, if any.

C) LSR BIT-5 indicates when THR empty.

D) LSR BIT-6 indicates when both the transmit FIFO and TSR are empty.

E) LSR BIT-7 indicates the sum of errors in the RX FIFO.

IER BIT-0: RHR Interrupt Enable
This interrupt will be issued when RHR is full or when receive data in the receive FIFO have reached the programmed trigger level and operating in DMA mode 1.
Logic 0 = Disable the receiver ready interrupt. (normal default condition)
Logic 1 = Enable the receiver ready interrupt.

IER BIT-1: THR Interrupt Enable
This interrupt is associated with bit-5 in the LSR register. An interrupt is issued whenever the THR becomes empty or when data in the FIFO falls below the programmed trigger level and operating in DMA mode 1.
Logic 0 = Disable the transmitter empty or not full interrupt. (normal default condition)
Logic 1 = Enable the transmitter empty or not full interrupt.

IER BIT-2: Receive Line Status Interrupt Enable
Any of the LSR register bits 1,2,3 or 4 becomes active

will generate an interrupt to inform the host controller about the error status of the current data byte in FIFO.
Logic 0 = Disable the receiver line status interrupt. (normal default condition)
Logic 1 = Enable the receiver line status interrupt.

IER BIT-3: Modem Status Interrupt Enable
Logic 0 = Disable the modem status register interrupt. (normal default condition)
Logic 1 = Enable the modem status register interrupt.

IER BIT -4: Sleep Mode Enable (requires EFR bit-4 =1)
Logic 0 = Disable sleep mode. (normal default condition)
Logic 1 = Enable sleep mode. See Sleep Mode section for details.

IER BIT-5: Xoff Interrupt Enable (requires EFR bit-4=1)
Logic 0 = Disable the software flow control, receive Xoff interrupt. (normal default condition)
Logic 1 = Enable the software flow control, receive Xoff interrupt. See Software Flow Control section for details.

IER BIT-6: -RTS Output Interrupt Enable (requires EFR bit-4=1)
Logic 0 = Disable the RTS interrupt. (normal default condition)
Logic 1 = Enable the RTS interrupt. The 854 issues an interrupt when the RTS pin transitions from a logic 0 to a logic 1.

IER BIT-7: -CTS Input Interrupt Enable (requires EFR bit-4=1)
Logic 0 = Disable the CTS interrupt. (normal default condition)
Logic 1 = Enable the CTS interrupt. The 854 issues an interrupt when CTS pin transitions from a logic 0 to a logic 1.

FIFO Control Register (FCR)

This register is used to enable the FIFO's, clear the FIFO's, set the transmit/receive FIFO trigger levels, and select the DMA mode. The DMA, and FIFO modes are defined as follows:

DMA MODE
Mode 0 Set the interrupt for each character transmit or receive operation, and is similar to the ST16C454 mode. Transmit Ready (-TXRDY) will go to a logic 0 whenever the Transmit Holding Register (THR) becomes empty. Receive Ready (-RXRDY) will go to a logic 0 whenever the Receive Holding Register (RHR) has a character.

Mode 1 Set the interrupt to the FIFO trigger level for data block transfer. The transmit interrupt is set when the transmit FIFO falls below the programmed trigger level. -TXRDY remains at logic 0 as long as there is one empty location. The receive interrupt is set when the receive FIFO fills up to the programmed trigger level. However the FIFO continues to fill regardless of the programmed level until the FIFO gets completely full. -RXRDY remains at logic 0 as long as the FIFO fill level is above the programmed trigger level. RXRDY time-out still generates an interrupt whenever data does not reaches the trigger level. Also, if the host does not load data into the TX FIFO up to the programmed trigger level, it will generates an empty interrupt when becoming empty.

FCR BIT-0: TX and RX FIFO Enable
Logic 0 = Disable the transmit and receive FIFO. (normal default condition)
Logic 1 = Enable the transmit and receive FIFO. This bit must be set to logic 1 when other FCR bits are written or they will not be programmed.

FCR BIT-1: RX FIFO Reset
This bit is only active when FCR bit-0 is active.
Logic 0 = No FIFO receive reset. (normal default condition)
Logic 1 = Reset the receive FIFO pointers and FIFO counter logic (the receive shift register is not cleared or altered). This bit will return to a logic 0 after resetting the FIFO.

FCR BIT-2: TX FIFO Reset
This bit is only active when FCR bit-0 is active.
Logic 0 = No FIFO transmit reset. (normal default condition)
Logic 1 = Reset the transmit FIFO pointers and FIFO counter logic (the transmit shift register is not cleared or altered). This bit will return to a logic 0 after resetting the FIFO.

FCR BIT-3: DMA Mode Select
Logic 0 = Set DMA mode "0". (normal default condition)
Logic 1 = Set DMA mode "1."

Transmit operation in mode "0":
When the 854 is in the ST16C450 mode (FIFO's disabled, FCR bit-0 = logic 0) or in the FIFO mode (FIFO's enabled, FCR bit-0 = logic 1, FCR bit-3 = logic 0) and when there are no characters in the transmit FIFO or transmit holding register, the -TXRDY pin will be a logic 0. Once active the -TXRDY pin will go to a logic 1 after the first character is loaded into the transmit holding register.

Receive operation in mode "0":
When the 854 is in mode "0" (FCR bit-0 = logic 0) or in the FIFO mode (FCR bit-0 = logic 1, FCR bit-3 = logic 0) and there is at least one character in the receive FIFO, the -RXRDY pin will be a logic 0. Once active the -RXRDY pin will go to a logic 1 when there are no more characters in the receiver.

Transmit operation in mode "1":
When the 854 is in FIFO mode (FCR bit-0 = logic 1, FCR bit-3 = logic 1), the -TXRDY pin will be a logic 1 when the transmit FIFO is completely full. It will be a logic 0 if one or more FIFO locations are empty.

Receive operation in mode "1":
When the 854 is in FIFO mode (FCR bit-0 = logic 1, FCR bit-3 = logic 1) and the trigger level has been reached, or a Receive Time Out has occurred, the -RXRDY pin will go to a logic 0. Once activated, it will go to a logic 1 after there are no more characters in the FIFO.

FCR BIT 4-5: TX Trigger Select
(logic 0 or cleared is the default condition, TX trigger level = none)
The FCTR Bits 4-5 are associated with these 2 bits by selecting one of the four tables. Four user selectable trigger levels in 4 tables are supported for compatibility reasons. These bits set the trigger level for the transmit FIFO interrupt. The 854 will issue a transmit empty interrupt when the number of characters in the FIFO falls below the selected trigger level, or when it gets empty in case that the FIFO did not get filled over the trigger level.

Transmit Trigger Table-A
Default setting after reset. Compatible to ST16C550 and ST16C554.

BIT-5	BIT-4	FIFO trigger level
X	X	Zero (default)

Transmit Trigger Table-B
Compatible to ST16C650A

BIT-5	BIT-4	FIFO trigger level
0	0	16
0	1	8
1	0	24
1	1	30

Transmit Trigger Table-C
Compatible to ST16C654

BIT-5	BIT-4	FIFO trigger level
0	0	8
0	1	16
1	0	32
1	1	56

Transmit Trigger Table-D
Compatible to XR16C850, XR16C2850/2852

BIT-5	BIT-4	FIFO trigger level
X	X	User programmable Trigger levels

FCR BIT 6-7: (logic 0 or cleared is the default condition, RX trigger level =8)
These bits are used to set the trigger level for the receiver FIFO interrupt. The FCTR Bits 4-5 selects one of the following table.

Receive Trigger Table-A
Default setting after reset. Compatible to ST16C550 and ST16C554

BIT-7	BIT-6	FIFO trigger level
0	0	1 (default)
0	1	4
1	0	8
1	1	14

Receive Trigger Table-B
Compatible to ST16C650A

BIT-7	BIT-6	FIFO trigger level
0	0	8
0	1	16
1	0	24
1	1	28

Receive Trigger Table-C
Compatible to ST16C654

BIT-7	BIT-6	FIFO trigger level
0	0	8
0	1	16
1	0	56
1	1	60

Receive Trigger Table--D
Compatible to XR16C850 and XR16C2850/2852

BIT-7	BIT-6	FIFO trigger level
X	X	User programmable Trigger levels

Interrupt Status Register (ISR)

The 854 provides multiple levels of prioritized interrupts to minimize external software interaction. The Interrupt Status Register (ISR) provides the user with six interrupt status bits. Performing a read cycle on the ISR will give the user the highest pending interrupt level to be serviced, others are queue up for next service. No other interrupts are acknowledged until the pending interrupt is serviced. Whenever the interrupt status register is read, the interrupt status is cleared. However, it should be noted that only the current pending interrupt is cleared by the read. A lower level interrupt may be seen after re-reading the interrupt status bits. The Interrupt Source Table, Table 5, shows the data values (bit 0-5) for the six prioritized interrupt levels and the interrupt sources associated with each of these interrupt levels.

Interrupt generation:

- LSR is by any of the LSR bits 1, 2, 3 and 4.
- RXRDY is by LSR bit-0.
- RXRDY Time-out is by the 4-char delay timer.
- TXRDY is by LSR bit-5 (or bit-6 in auto RS485 control).
- MSR is by any of the MSR bits, 0, 1, 2 and 3.
- Receive Xoff/Special char. is by the detection of a Xoff or Special character.
- CTS is by the change of state on the input pin, and RTS is when auto RTS flow control is enabled and the receiver changes the state of the output pin.

Interrupt clearing:

- LSR interrupt is cleared by a read to the LSR register.
- RXRDY and RXRDY time out are cleared by a read to the LSR register.
- TXRDY interrupt is cleared by a read to the ISR register.
- MSR interrupt is cleared by a read to the MSR register.
- Xoff or Special character interrupt is cleared by a read to the ISR register.
- RTS and CTS status change interrupts are cleared by a read to the MSR register.

Priority Level	[ISR Register Status Bits] Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0	Source of the interrupt
1	0	0	0	1	1	0	LSR (Receiver Line Status Register)
2	0	0	0	1	0	0	RXRDY (Received Data Ready)
2	0	0	1	1	0	0	RXRDY (Receive Data time out)
3	0	0	0	0	1	0	TXRDY (Transmitter Holding Register Empty)
4	0	0	0	0	0	0	MSR (Modem Status Register)
5	0	1	0	0	0	0	RXRDY (Received Xoff signal)/Special character
6	1	0	0	0	0	0	CTS, RTS change of state
X	0	0	0	0	0	1	None (default)

Table 5, Interrupt Source Table.

ISR BIT-0: Interrupt Status
Logic 0 = An interrupt is pending and the ISR contents may be used as a pointer to the appropriate interrupt service routine.
Logic 1 = No interrupt pending. (default condition)

ISR BIT 1-3: Interrupt Status
These bits indicate the source for a pending interrupt at interrupt priority levels 1, 2, and 3 (See Interrupt Source Table).

ISR BIT 4-5: Interrupt Status
These bits are enabled when EFR bit-4 is set to a logic 1. ISR bit-4 indicates that matching Xoff character(s) have been detected. Note that once set to a logic 1, the ISR bit-4 will stay a logic 1 until a Xon character is received. ISR bit-5 indicates that CTS or RTS has changed state.

ISR BIT 6-7: FIFO Enable Status
These bits are set to a logic 0 when the FIFO's are disabled. They are set to a logic 1 when the FIFO's are enabled.

Line Control Register (LCR)

The Line Control Register is used to specify the asynchronous data communication format. The word length, the number of stop bits, and the parity are selected by writing the appropriate bits in this register.

LCR BIT 0-1: TX and RX Word Length Select
These two bits specify the word length to be transmitted or received.

BIT-1	BIT-0	Word length
0	0	5 (default)
0	1	6
1	0	7
1	1	8

LCR BIT-2: TX and RX Stop-bit Length Select
The length of stop bit is specified by this bit in conjunction with the programmed word length.

BIT-2	Word length	Stop bit length (Bit time(s))
0	5,6,7,8	1 (default)
1	5	1-1/2
1	6,7,8	2

LCR BIT-3: TX and RX Parity Select
Parity or no parity can be selected via this bit.
Logic 0 = No parity.
Logic 1 = A parity bit is generated during the transmission while the receiver checks for parity error of the data byte received.

LCR BIT-4: TX and RX Parity Select
If the parity bit is enabled with LCR bit-3 set to a logic 1, LCR BIT-4 selects the even or odd parity format. Logic 0 = ODD Parity is generated by forcing an odd number of logic 1's in the transmitted data. The receiver must be programmed to check the same format. (normal default condition)
Logic 1 = EVEN Parity is generated by forcing an even the number of logic 1's in the transmitted. The receiver must be programmed to check the same format.

LCR BIT-5: TX and RX Parity Select
If the parity bit is enabled, LCR BIT-5 selects the forced parity format.
LCR BIT-5 = logic 0, parity is not forced. (normal default condition)
LCR BIT-5 = logic 1 and LCR BIT-4 = logic 0, parity bit is forced to a logical 1 for the transmit and receive data.
LCR BIT-5 = logic 1 and LCR BIT-4 = logic 1, parity bit is forced to a logical 0 for the transmit and receive data.

LCR Bit-5	LCR Bit-4	LCR Bit-3	Parity selection
X	X	0	No parity
0	0	1	Odd parity
0	1	1	Even parity
1	0	1	Force parity "1"
1	1	1	Forced parity "0"

LCR BIT-6: Transmit Break Enable
When enabled the Break control bit causes a break condition to be transmitted (the TX output is forced to a logic 0 state). This condition exists until disabled by setting LCR bit-6 to a logic 0.
Logic 0 = No TX break condition. (normal default condition)
Logic 1 = Forces the transmitter output (TX) to a logic 0 for alerting the remote receiver to a line break condition.

LCR BIT-7: Baud Rate Divisors Enable
The internal baud rate counter latch and Enhance Feature mode enable.
Logic 0 = Divisor latch disabled. (normal default condition)
Logic 1 = Divisor latch and enhanced feature register enabled.

Modem Control Register (MCR)

This register controls the interface with the modem or a peripheral device.

MCR BIT-0: -DTR Pins Control
Logic 0 = Force -DTR output to a logic 1. (normal default condition)
Logic 1 = Force -DTR output to a logic 0.

MCR BIT-1: -RTS Pins Control
Logic 0 = Force -RTS output to a logic 1. (normal default condition)
Logic 1 = Force -RTS output to a logic 0.
Automatic RTS may be used for hardware flow control by enabling EFR bit-6 (See EFR bit-6).

MCR BIT-2:
The -OP1 output pin is not available in the 854.
This bit is used in the loopback mode only. In the loopback mode this bit is use to write the state of the modem -RI interface signal via -OP1.

MCR BIT-3: Interrupt A-D Output Select
The -OP2 output pin is not available in the 854.
Select interrupt INT A-D outputs to operate in continuous or three-state mode. This function is associated with the INTSEL input, see below table for details. This bit is also used to control the -CD signal during loopback mode.
Logic 0 = Forces INT A-D outputs to the three state operation in the 16 mode. (normal default condition). During loopback mode, it sets -OP2 (-CD) internally to a logic 1.
Logic 1 = Forces the INT A-D outputs to continuous operation in the 16 mode. During loopback mode, it sets -OP2 (-CD) internally to a logic 0.

INTSEL Pin*	INT A-D Outputs in 16 Mode
= 1	Continuous
= 0	if MCR bit-3=0, then it's three-state if MCR bit-3=1, then it's continuous

*INTSEL must be set to logic zero for 68 mode.

MCR BIT-4: Internal Loopback Enable
Logic 0 = Disable loopback mode. (normal default condition)
Logic 1 = Enable local loopback mode, see loopback section and figure 6.

MCR BIT-5: Xon-Any Enable
Logic 0 = Disable Xon-Any function (for 16C550 compatibility). (normal default condition)
Logic 1 = Enable Xon-Any function. In this mode any RX character received will enable Xon.

MCR BIT-6: Infrared Encoder/Decoder Enable
Logic 0 = Enable the standard modem receive and transmit input/output interface. (normal default condition)
Logic 1 = Enable infrared IrDA receive and transmit inputs/outputs. While in this mode, the TX/RX output/Inputs are routed to the infrared encoder/decoder. The data input and output levels will conform to the IrDA infrared interface requirement. As such, while in this mode the infrared TX output will be a logic 0 during idle data conditions.

MCR BIT-7: Clock Prescaler Select
Logic 0 = Divide by one. The input clock from the crystal or external clock is fed directly to the Programmable Baud Rate Generator (BRG) without further modification, i.e., divide by one. (normal, default condition)
Logic 1 = Divide by four. The prescaler divides the input clock from the crystal or external clock by four and feeds it to the Programmable Baud Rate Generator.

Line Status Register (LSR)

This register provides the status of data transfers between the 854 and the CPU.

LSR BIT-0: Receive Data Ready Indicator
Logic 0 = No data in receive holding register or FIFO. (normal default condition)
Logic 1 = Data has been received and is saved in the receive holding register or FIFO.

LSR BIT-1: Receiver Overrun Flag
Logic 0 = No overrun error. (normal default condition)
Logic 1 = Overrun error. A data overrun error occurred in the receive shift register. This happens when additional data arrives while the FIFO is full. In this case the previous data in the shift register is overwritten. Note that under this condition the data byte in the receive shift register is not transferred into the FIFO, therefore the data in the FIFO is not corrupted by the error.

LSR BIT-2: Receive Data Parity Error Flag
Logic 0 = No parity error. (normal default condition)
Logic 1 = Parity error. The receive character does not have correct parity information and is suspect. In the FIFO mode, this error is associated with the character at the top of the FIFO.

LSR BIT-3: Receive Data Framing Error Flag
Logic 0 = No framing error. (normal default condition)
Logic 1 = Framing error. The receive character did not have a valid stop bit(s). In the FIFO mode this error is associated with the character at the top of the FIFO.

LSR BIT-4: Receive Break Flag
Logic 0 = No break condition. (normal default condition)
Logic 1 = The receiver received a break signal (RX was a logic 0 for one character frame time). In the FIFO mode, only one break character is loaded into the FIFO.

LSR BIT-5: Transmit Holding Register Empty Flag
This bit is the Transmit Holding Register Empty indicator. This bit indicates that the transmitter is ready to accept a new character for transmission. In addition, this bit causes the UART to issue an interrupt to CPU when the THR interrupt enable is set. The THR bit is set to a logic 1 when the last data byte is transferred from the transmit holding register to the transmit shift register. The bit is reset to logic 0 concurrently with the data loading to the transmit holding register by the CPU. In the FIFO mode this bit is set when the transmit FIFO is empty; it is cleared when at least 1 byte is written to the transmit FIFO.

LSR BIT-6: Transmit Shift Register Empty Flag
This bit is the Transmit Shift Register Empty indicator. This bit is set to a logic 1 whenever the transmitter goes idle. It is reset to logic 0 whenever either the THR or TSR contains a data character. In the FIFO mode this bit is set to one whenever the transmit FIFO and transmit shift register are both empty.

LSR BIT-7: Receive FIFO Data Error Flag

Logic 0 = No FIFO error. (normal default condition)
Logic 1 = An indicator for the sum of all error bits in the RX FIFO. At least one parity error, framing error or break indication is in the FIFO data. This bit clears when there is no more error in the FIFO.

Modem Status Register (MSR)

This register provides the current state of the control interface signals from the modem, or other peripheral device that the 854 is connected to. Four bits of this register are used to indicate the changed information. These bits are set to a logic 1 whenever a control input from the modem changes state. These bits are set to a logic 0 whenever the CPU reads this register.

MSR BIT-0: Delta -CTS Input Flag
Logic 0 = No -CTS Change (normal default condition)
Logic 1 = The -CTS input to the 854 has changed state since the last time it was read. A Modem Status Interrupt will be generated.

MSR BIT-1: Delta -DSR Input Flag
Logic 0 = No -DSR Change. (normal default condition)
Logic 1 = The -DSR input to the 854 has changed state since the last time it was read. A modem Status Interrupt will be generated.

MSR BIT-2: Delta -RI Input Flag
Logic 0 = No -RI Change. (normal default condition)
Logic 1 = The -RI input to the 854 has changed from a logic 0 to a logic 1. A modem Status Interrupt will be generated.

MSR BIT-3: Delta -CD Input Flag
Logic 0 = No -CD Change. (normal default condition)
Logic 1 = Indicates that the -CD input to the has changed state since the last time it was read. A modem Status Interrupt will be generated.

MSR BIT-4: CTS Input Status
-CTS functions as hardware flow control signal input if it is enabled by EFR bit-7. Auto -CTS flow control (when enabled) allows the starting and stopping the transmissions based on the external modem -CTS signal. A logic 1 at the -CTS pin will stop 854 transmissions as soon as current character has finished transmission.

Normally MSR bit-4 bit is the compliment of the -CTS input. However in the loopback mode, this bit is equivalent to the RTS bit in the MCR register.

MSR BIT-5: DSR Input Status
DSR (active high, logical 1). Normally this bit is the compliment of the -DSR input. In the loopback mode, this bit is equivalent to the DTR bit in the MCR register.

MSR BIT-6: RI Input Status
RI (active high, logical 1). Normally this bit is the compliment of the -RI input. In the loopback mode this bit is equivalent to the OP1 bit in the MCR register.

MSR BIT-7: CD Input Status
CD (active high, logical 1). Normally this bit is the compliment of the -CD input. In the loopback mode this bit is equivalent to the OP2 bit in the MCR register.

Scratch Pad Register (SPR)

This is a temporary data register for user data. The SPR is set to 0xFF after power up and upon a reset. Also, see FCTR bit-6 for additional functions detail associated with registers FLVL and EMSR.

Baud Rate Divisor Registers (DLL and DLM)

The Baud Rate Generator (BRG) is a 16-bit counter that generates the data rate for the transmitter and receiver. The rate is programmed through registers DLL and DLM which are only accessible when LCR bit-7 is set to logic 1 and not 0xBF. See Table 4 for programming selection.

Enhanced Feature Register (EFR)

Enhanced features are enabled or disabled using this register. Bit 0-3 provide single or dual consecutive character software flow control selection (see Table 6). When the Xon1 and Xon2 and/or Xoff1 and Xoff2 modes are selected, the double 8-bit words are concatenated into two sequential characters. Please note that whenever changing the TX or RX flow control bits, always reset all bits back to logic 0 (disable) before programming a new setting.

EFR BIT 0-3: Software Flow Control Select
Combinations of software flow control can be selected by programming these bits.

EFR bit-3 Cont-3	EFR bit-2 Cont-2	EFR bit-1 Cont-1	EFR bit-0 Cont-0	TX and RX Software Flow Control
0	0	0	0	No TX and RX flow control (default)
0 1 0 1	0 0 1 1	X X X X	X X X X	No transmit flow control Transmit Xon1/Xoff1 Transmit Xon2/Xoff2 Transmit Xon1 and Xon2/Xoff1 and Xoff2
X X X	X X X	0 1 0	0 0 1	No receive flow control Receiver compares Xon1/Xoff1 Receiver compares Xon2/Xoff2
1 0	0 1	1 1	1 1	Transmit Xon1/ Xoff1, Receiver compares Xon1 or Xon2, Xoff1 or Xoff2 Transmit Xon2/Xoff2, Receiver compares Xon1 or Xon2, Xoff1 or Xoff2
1 0	1 0	1 1	1 1	Transmit Xon1 and Xon2/Xoff1 and Xoff2, Receiver compares Xon1 and Xon2/Xoff1 and Xoff2 No transmit flow control, Receiver compares Xon1 and Xon2/Xoff1 and Xoff2

Table 6, Software Flow Control Functions

EFR BIT-4: Enhanced Function Bits Enable
Enhanced function control bit. The content of the IER bits 4-7, ISR bits 4-5, FCR bits 4-5, and MCR bits 5-7 can be modified and latched. After modifying any bits in the enhanced registers, EFR bit-4 can be set to a logic 0 to latch the new values. This feature prevents existing software from altering or overwriting the 854 enhanced functions.

Logic 0 = disable/latch enhanced features. IER bits 4-7, ISR bits 4-5, FCR bits 4-5, and MCR bits 5-7 are saved to retain the user settings, then IER bits 4-7, ISR bits 4-5, FCR bits 4-5, and MCR bits 5-7 are initialized to the default values shown in the Internal Resister Table. After a reset, the IER bits 4-7, ISR bits 4-5, FCR bits 4-5, and MCR bits 5-7 are set to a logic 0 to be compatible with ST16C554 mode. (normal default condition).
Logic 1 = Enables the enhanced functions. When this bit is set to a logic 1 all enhanced features of the 854 are enabled.

EFR BIT-5: Special Character Detect Enable
Logic 0 = Special Character Detect Disabled. (normal default condition)
Logic 1 = Special Character Detect Enabled. The 854 compares each incoming receive character with Xoff-2 data. If a match exists, the received data will be transferred to FIFO and ISR bit-4 will be set to indicate detection of special character. Bit-0 corresponds with the LSB bit for the receive character. When this feature is enabled, the normal software flow control must be disabled (EFR bits 0-3 must be set to a logic 0).

EFR BIT-6: Auto RTS Flow Control Enable
Automatic RTS may be used for hardware flow control by enabling EFR bit-6. When Auto RTS is selected, an interrupt will be generated when the receive FIFO is filled to the programmed trigger level and -RTS will go to a logic 1at the next trigger level. -RTS will return to a logic 0 when data is unloaded below the next lower trigger level (Programmed trigger level -1). The -RTS output must be asserted (logic 0) before the auto RTS takes effect. -RTS will function as a general purpose output

when hardware flow control is disabled.
0 = Automatic RTS flow control is disabled. (normal default condition)
1 = Enable Automatic RTS flow control.

EFR bit-7: Auto CTS Flow Control Enable
Automatic CTS Flow Control.
Logic 0 = Automatic CTS flow control is disabled. (normal default condition)
Logic 1 = Enable Automatic CTS flow control. Transmission will stop when -CTS goes to a logical 1. Transmission will resume when the -CTS pin returns to a logical 0.

FEATURE CONTROL REGISTER (FCR)

This register controls the XR16C854 new functions that are not available on ST16C554 or ST16C654.

FCTR BIT-0 and 1: RTS Trigger Level Select
User selectable auto -RTS trigger delay timer for hardware flow control application. After reset, these bits are set to logic 0 selecting the next FIFO trigger level for hardware flow control. These 2 bits are associated with EMSR register bit 4 and 5 for more level control, see EMSR bit 4 and 5 for complete detail.

FCTR BIT-2: RX Input Select
0 = Select RX input as encoded IrDA data.
1 = Select RX input as active high encoded IrDA data.

FCTR BIT-3: Auto RS485 Enable
The auto RS485 is not available in 854, see XR16C864. However, it still selects/changes the TX empty interrupt from THR to TSR.
0 = Standard ST16C550 mode. Transmitter generates interrupt when transmit holding register (THR) becomes empty. Transmit Shift Register (TSR) may still be shifting data bit out.
1 = Enable Auto RS485 half duplex direction control and change the transmit interrupt to transmit shift register (TSR) empty. Transmit empty interrupt is generated when the transmitter shift register becomes empty (and changes -OP2 A-D output pin to logic 1 with one bit time delay for receive mode. The -OP2 output will change back to logic 0 for transmit again when the TX FIFO is loaded with a data byte. The change to logic 0 occurs prior shifting the start-bit bit out).

FCTR BIT-4 and 5: TX and RX FIFO Trigger Table Select
These 2 bits select the transmit and receive FIFO trigger table A, B ,C or D. When table A, B, or C is selected the auto RTS flow control trigger level is set to "next level" for compatibility to ST16C550/650 series. -RTS triggers on the next level of the RX FIFO trigger level, in another word, one level above and one level below. See Tables A-C in FCR bit 4-5 and FCR bit 6-7, i.e. if Table A is used on the receiver with RX FIFO trigger level set to 8 bytes, -RTS output will de-asserts at 14 and re-asserts at 4.

FCTR Bit-5	FCTR Bit-4	TX and RX FIFO Trigger Table
0	0	TX and RX Trigger Table-A (default)
0	1	TX and RX Trigger Table-B
1	0	TX and RX Trigger Table-C
1	1	TX and RX Trigger Table-D

FCTR BIT-6: Register Address 0x111 Select
Scratch Pad (SPR) register or FLVL and EMSR registers select for address location 0x111.
0 = Scratch Pad register (SPR) is selected as a general read and write register, ST16C554 compatible mode.
1 = Changes the register function at address location 0x07. It switches to FIFO Level Count (FLVL) Register for bus read operation and Enhanced Mode Select Register for bus write operation. The FLVL indicates the number of data bytes in the transmit or receive data FIFO. The SPR is not available in this configuration.

FCTR BIT-7: TX/RX FIFO Trigger Register Select
Programmable trigger register select. This bit is associated with the TRG register which actually sets the FIFO trigger levels.
0 = Select programmable receive FIFO trigger level register.
1 = Select programmable transmit FIFO trigger level register.

TRIGGER LEVEL REGISTER (TRG)

This is the user programmable transmit /receive FIFO trigger level register. The register is associated with the FCTR bit-7 which selects TX or RX FIFO trigger register.

TRG BIT 0-7: Write only. FIFO Trigger Level
The value written to this register sets the TX or RX FIFO trigger level from 0x00 (zero) to 0x80 (128). The FIFO trigger level generates an interrupt whenever the transmit FIFO level falls below its preset trigger level while the RX FIFO generates an interrupt as soon as incoming data fills up to its preset trigger level.

TRG BIT 0-7: Read only.
Transmit / receive FIFO level byte count. It gives an indication of the number of characters present in the transmit or receive FIFO. A simpler way to read the FIFO level count is through register FLVL instead.

FIFO LEVEL REGISTER (FLVL)

The FIFO Level Byte Count Register is read only and accessible only when FCTR bit-6 has been set to logic 1. The user can take advantage of the FIFO level byte counter for faster data loading to the transmit FIFO and unloading data from the receiver FIFO. Hence, reduces CPU bandwidth requirement.

FLVL BIT 0-7: Read only. FIFO Level Counter
This register provides the number of data bytes in the RX or TX FIFO which is determined by EMSR bit-0 and 1. The returned byte count ranges from 0x00 (zero) to 0x80 (128).

ENHANCED MODE SELECT REGISTER (ESMR)

This is a write only register and accessible only when FCTR bit-6 is set to logic 1.

EMSR BIT-0 and 1: Write only. TX or RX FIFO Level Counter Select
EMSR bit-0 and 1 select which FIFO byte count for FLVL register to present when reading the byte count. It can be transmit only, receive only, or alternate receive and transmit byte count after each read operation. In the alternate mode, it presents the receive count first and the transmit count on the second read operation, and repeats thereafter. In this case, user's software must remember the state for the byte count.

Bit-1	Bit-0	TX/RX/ALT FIFO Level Count
0	0	Receive FIFO level count (deafult)
0	1	Transmit FIFO level count
1	0	Receive FIFO level count
1	1	Alternate RX/TX FIFO level count

EMSR BIT 2 and 3
Reserved for XR16C864 use.

EMSR BIT-4 and 5 - Write only. Auto RTS Flow Control Hysteresis
These bits select the auto RTS flow control hysteresis and are associated with FCTR bit 0 and 1, and only valid when TX and RX Trigger Table-D is selected (FCTR bit-4 and 5 are set to logic 1). The RTS hysteresis is reference to the RX FIFO trigger level. Below table shows the 16 selectable hysteresis levels.

EMSR Bit-5	**EMSR Bit-4**	**FCTR Bit-1**	**FCTR Bit-0**	**RTS Hysteresis** (characters)
0	0	0	0	Next level (default)
0	0	0	1	+/- 4
0	0	1	0	+/- 6
0	0	1	1	+/- 8
0	1	0	0	+/- 8
0	1	0	1	+/- 16
0	1	1	0	+/- 24
0	1	1	1	+/- 32
1	1	0	0	+/- 40
1	1	0	1	+/- 44
1	1	1	0	+/- 48
1	1	1	1	+/- 52
1	0	0	0	+/- 12
1	0	0	1	+/- 20
1	0	1	0	+/- 28
1	0	1	1	+/- 36

EMSR BIT 6 and 7:
Reserved for future use.

FIFO Status Register (FSTAT)

This register is applicable only to the 100 pin XR16C854. The FIFO Status Register provides a status indication for each of the transmit and receive FIFO. These status bits are complementary of the individual -TXRDY A-D and -RXRDY A-D outputs. Each channel has its own 128 bytes of TX and RX FIFO. A bit associated with each UART channel is asserted when any of the TX FIFO becomes empty, or any RX FIFO becomes full, or any FIFO level has reached the programmed trigger level. Their behavior changes between DMA mode 0 and 1. See Internal Registers Description.

BIT 0-3: TX FIFO channel A-D Status
DMA Mode 0
0 = The transmit FIFO is not empty, data in FIFO. One or more locations is available for transmit data.
1 = The transmit FIFO is completely empty, no data.

DMA Mode 1
0 = The transmit FIFO is completely full and will not accept any more data.
1 = The transmit FIFO is not full, some data in FIFO . One or more locations is available for more data.

BIT 4-7: RX FIFO Channel A-D Status
DMA Mode 0
0 = The receiver, RHR/FIFO is empty.
1 = The receiver, RHR/FIFO is not empty.

DMA Mode 1
0 = The receive FIFO is empty or has not reached the programmed trigger level.
1 = The receive FIFO has reached the programmed trigger level or a time-out has occurred.

XR16C854 EXTERNAL RESET CONDITIONS

REGISTERS	RESET STATE
RHR	Bits 0-7 = 0xXX
THR	Bits 0-7 = 0xXX
IER	Bits 0-7 = 0x00
FCR	Bits 0-7 = 0x00
ISR	Bits 0-7 = 0x01
LCR	Bits 0-7 = 0x00
MCR	Bits 0-7 = 0x00
LSR	Bits 0-7 = 0x60
MSR	Bits 0-3 = logic 0 Bits 4-7 = logic levels of the inputs
SPR	Bits 0-7 = 0xFF
FLVL	Bits 0-7 = 0x00
EMSR	Bits 0-7 = 0x00
DLL	Bits 0-7 = 0xXX
DLM	Bits 0-7 = 0xXX
TRG	Bits 0-7 = 0x00
FCTR	Bits 0-7 = 0x00
EFR	Bits 0-7 = 0x00
Xon-1	Bits 0-7 = 0x00
Xon-2	Bits 0-7 = 0x00
Xoff-1	Bits 0-7 = 0x00
Xoff-2	Bits 0-7 = 0x00
FSTAT	Bits 0-7 = 0x00

SIGNALS	RESET STATE
TX A-D	Logic 1
IRTX A-D	Logic 0
INT A-D	Logic 0
-IRQ	Logic 1 (68 mode, INTSEL=0)
-RXRDY A-D	Logic 1
-TXRDY A-D	Logic 0
-RTS A-D	Logic 1
-DTR A-D	Logic 1

ABSOLUTE MAXIMUM RATINGS

Supply range	7 Volts
Voltage at any pin	GND - 0.3 V to VCC +0.3 V
Operating temperature	-40° C to +85° C
Storage temperature	-65° C to 150° C
Package dissipation	500 mW

Typical Package Thermal Resistance	68-PLCC	64-TQFP	100-QFP
Theta-ja	43	70	45
Theta-jc	17	14	12

DC ELECTRICAL CHARACTERISTICS

T_A=0° - 70°C (-40° - +85°C for Industrial grade packages), Vcc=3.3 - 5.0 V ± 10% unless otherwise specified.

Symbol	Parameter	Limits 3.3		Limits 5.0		Units	Conditions
		Min	Max	Min	Max		
V_{ILCK}	Clock input low level	-0.3	0.6	-0.5	0.6	V	
V_{IHCK}	Clock input high level	2.4	VCC	3.0	VCC	V	
V_{IL}	Input low level	-0.3	0.8	-0.5	0.8	V	
V_{IH}	Input high level	2.0		2.2	VCC	V	
V_{OL}	Output low level on all outputs				0.4	V	I_{OL}= 5 mA
V_{OL}	Output low level on all outputs		0.4			V	I_{OL}= 4 mA
V_{OH}	Output high level			2.4		V	I_{OH}= -5 mA
V_{OH}	Output high level	2.0				V	I_{OH}= -1 mA
I_{IL}	Input leakage		±10		±10	µA	
I_{CL}	Clock leakage		±10		±10	µA	
I_{CC}	Avg power supply current		3		6	mA	
I_{CC}	Avg stand by current		100		200	µA	
C_P	Input capacitance		5		5	pF	
R_{IN}	Internal pull-up / pull-down resistance	40	80	40	80	kΩ	

Note: See the Symbol Description Table for a listing of pins having internal input pull-up or pull-down resistor.

AC ELECTRICAL CHARACTERISTICS

T_A=0° - 70°C (-40° - +85°C for Industrial grade packages), Vcc=3.3 - 5.0 V ± 10% unless otherwise specified.

Symbol	Parameter	Limits 3.3		Limits 5.0		Units	Conditions
		Min	Max	Min	Max		
T_{1w},T_{2w}	External clock periods	20		15		ns	
T_{3w}	External clock freq. (40/60% duty cycle)	24 typ.		32 typ.		MHz	see figure 7
	Serial data rate	1.5 typ.		2.0 typ.		Mbps	see figure 7
T_{3w}	On-chip oscillator frequency		8		24	MHz	
T_{6s}	Address setup time	10		5		ns	(see note 4)
T_{7d}	-IOR delay from chip select	10		5		ns	(see note 4)
T_{7w}	-IOR strobe width	35		25		ns	
T_{7h}	Address hold time from -IOR	10		5		ns	(see note 4)
T_{9d}	Read cycle delay	40		30		ns	
T_{12d}	Delay from -IOR to data		35		25	ns	
T_{12h}	Data disable time		25		15	ns	
T_{13d}	-IOW delay from chip select	10		10		ns	(see note 4)
T_{13w}	-IOW strobe width	35		25		ns	
T_{13h}	Address hold time from -IOW	10		5		ns	(see note 4)
T_{15d}	Write cycle delay	40		30		ns	
T_{16s}	Data setup time	10		5		ns	(see note 4)
T_{16h}	Data hold time	10		5		ns	(see note 4)
T_{17d}	Delay from -IOW to output		50		40	ns	100 pF load
T_{18d}	Delay to set interrupt from MODEM input		40		35	ns	100 pF load
T_{19d}	Delay to reset interrupt from -IOR		40		35	ns	100 pF load
T_{20d}	Delay from stop to set interrupt		1		1	RCLK	
T_{21d}	Delay from -IOR to reset interrupt		40		40	ns	100 pF load
T_{22d}	Delay from stop to interrupt		45		40	ns	
T_{23d}	Delay from initial INT reset to transmit start	8	24	8	24	RCLK	
T_{24d}	Delay from -IOW to reset interrupt		45		40	ns	
T_{25d}	Delay from stop to set -RXRDY		1		1	RCLK	
T_{26d}	Delay from -IOR to reset -RXRDY		45		40	ns	
T_{27d}	Delay from -IOW to set -TXRDY		45		40	ns	
T_{28d}	Delay from start to reset -TXRDY		8		8	RCLK	
T_{30s}	Address setup time	10		10		ns	
T_{30w}	Chip select strobe width	40		40		ns	
T_{30h}	Address hold time	15		15		ns	
T_{30d}	Read cycle delay	70		70		ns	
T_{31d}	Delay from -CS to data	15		15		ns	
T_{31h}	Data disable time			15		ns	
T_{32s}	Write strobe setup time	10		10		ns	
T_{32h}	Write strobe hold time	10		10		ns	

AC ELECTRICAL CHARACTERISTICS

T_A=0° - 70°C (-40° - +85°C for Industrial grade packages), Vcc=3.3 - 5.0 V ± 10% unless otherwise specified.

Symbol	Parameter	Limits 3.3		Limits 5.0		Units	Conditions
		Min	Max	Min	Max		
T_{32d}	Write cycle delay	70		70		ns	
T_{33s}	Data setup time	20		15		ns	
T_{33h}	Data hold time	10		10		ns	
T_R	Reset pulse width	40		40		ns	
N	Baud rate divisor	1	2_{16}-1	1	2_{16}-1	RCLK	

Note 4: Refer to application note DAN107 for further explanation.

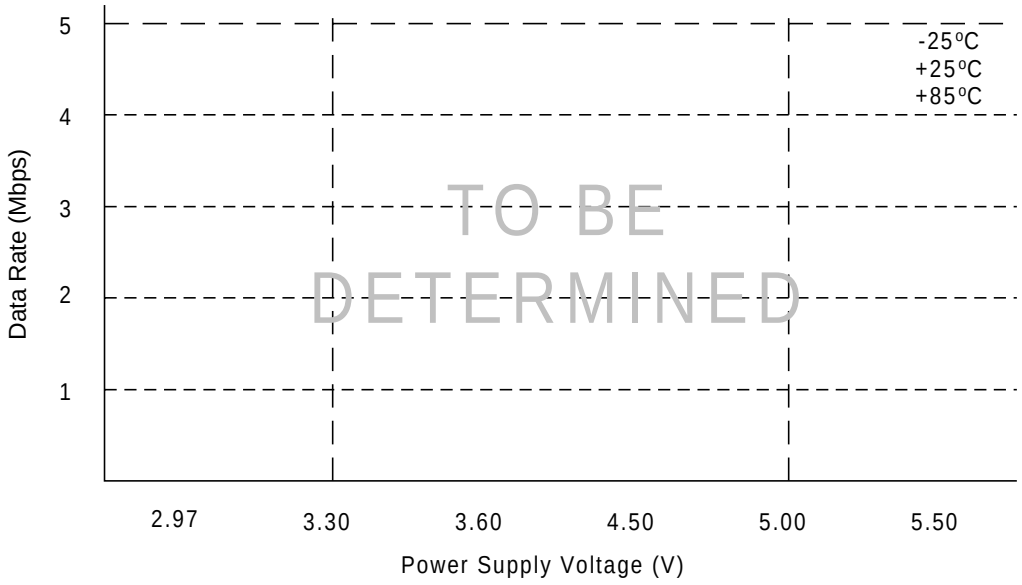
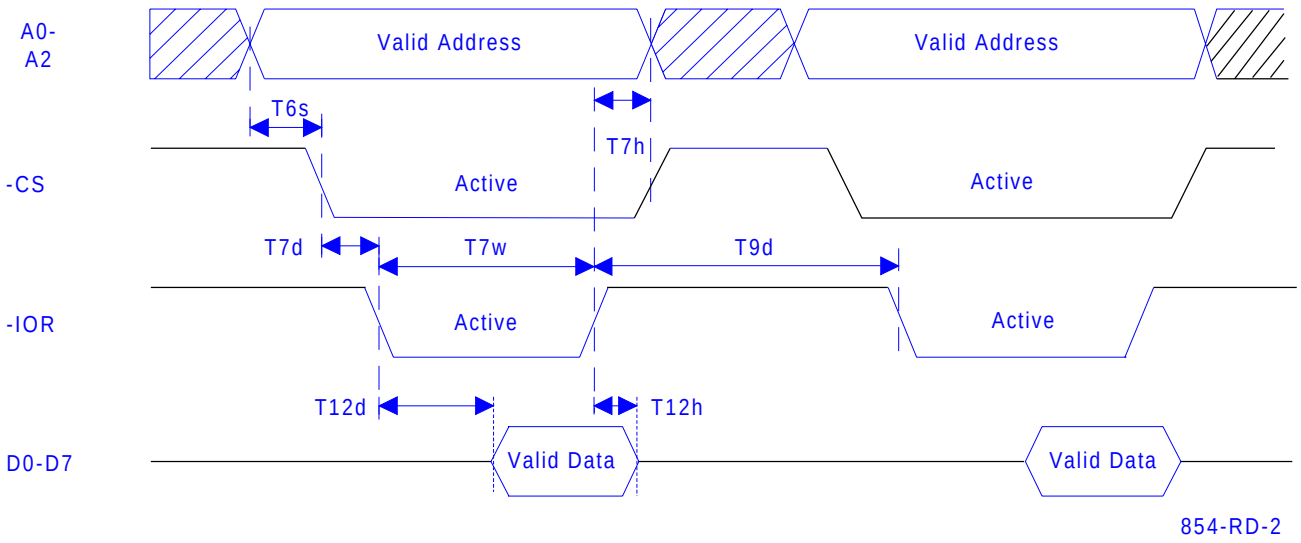
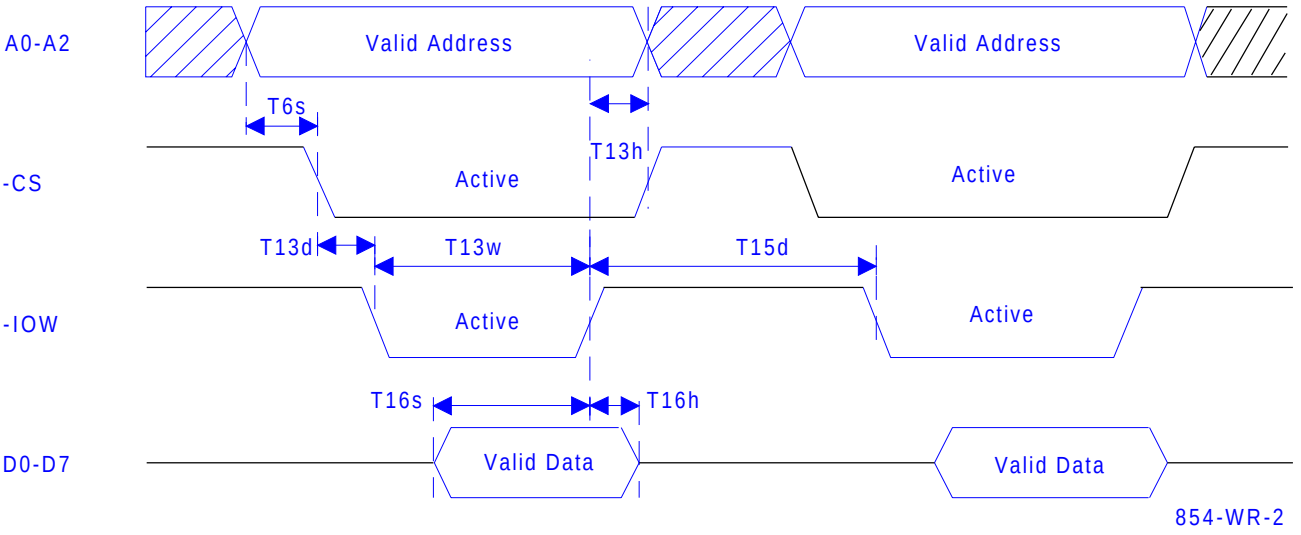


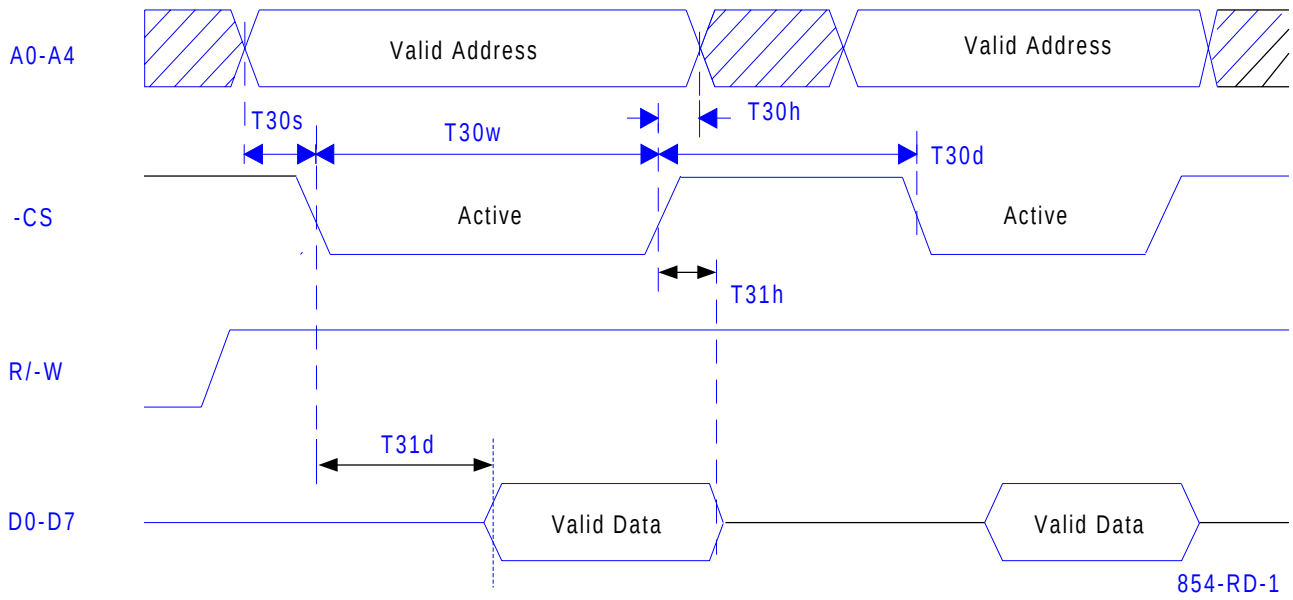
Figure 7 . Data Rate Perfomance Chart



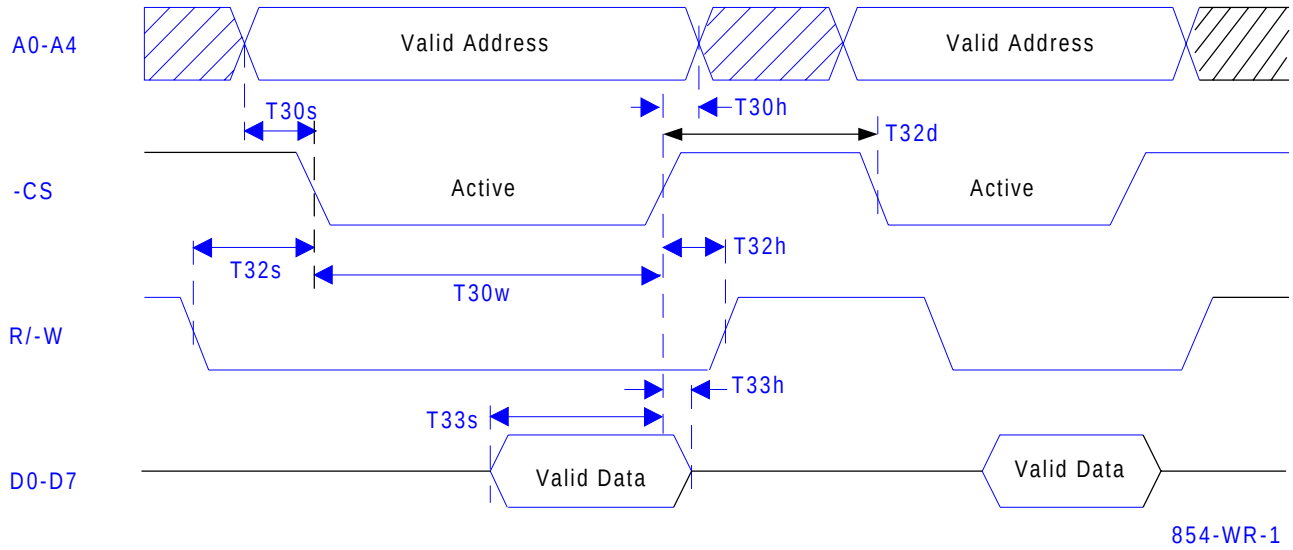
Data bus Read Timing in 16 Mode



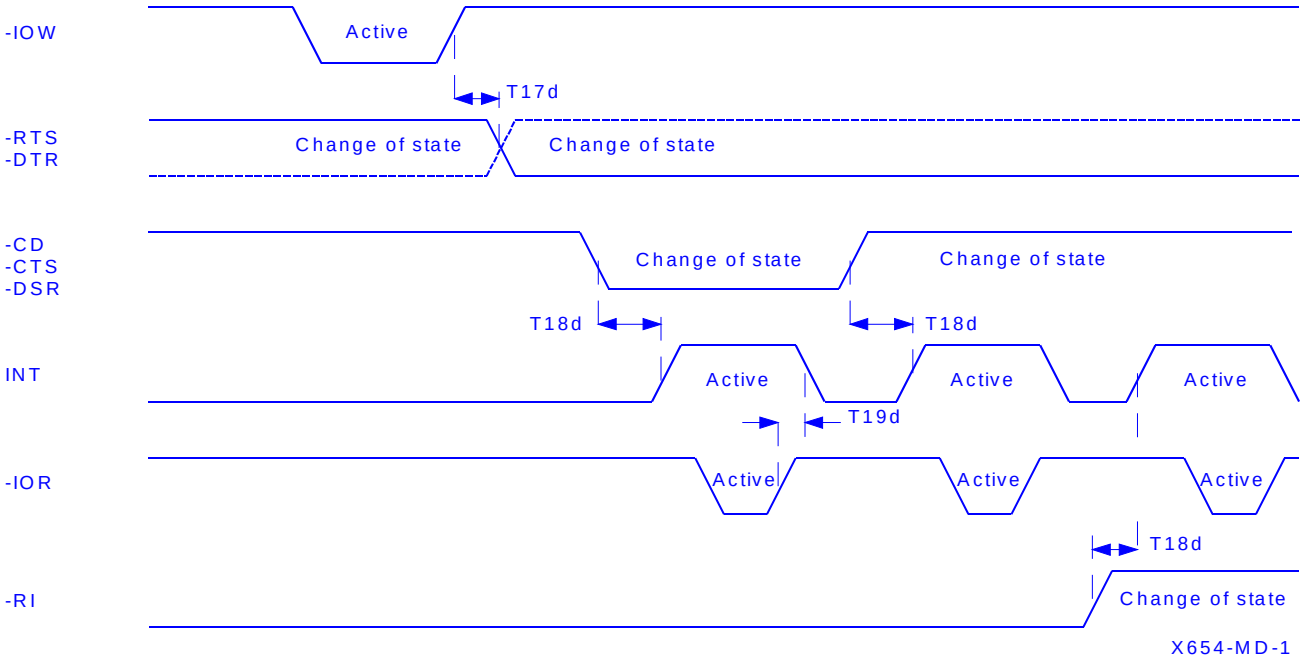
Data Bus Write Timing in 16 Mode



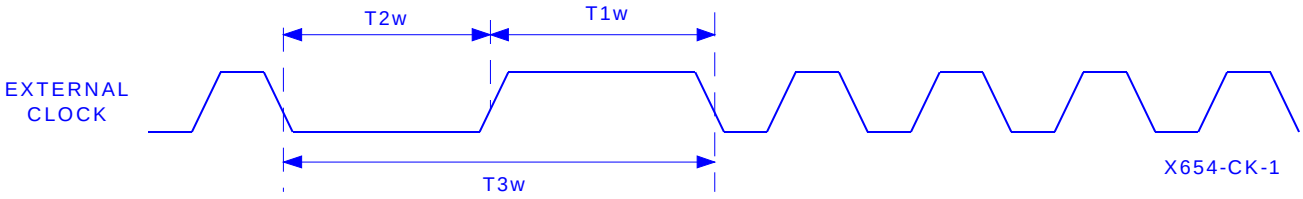
Data Bus Read Timing in 68 Mode



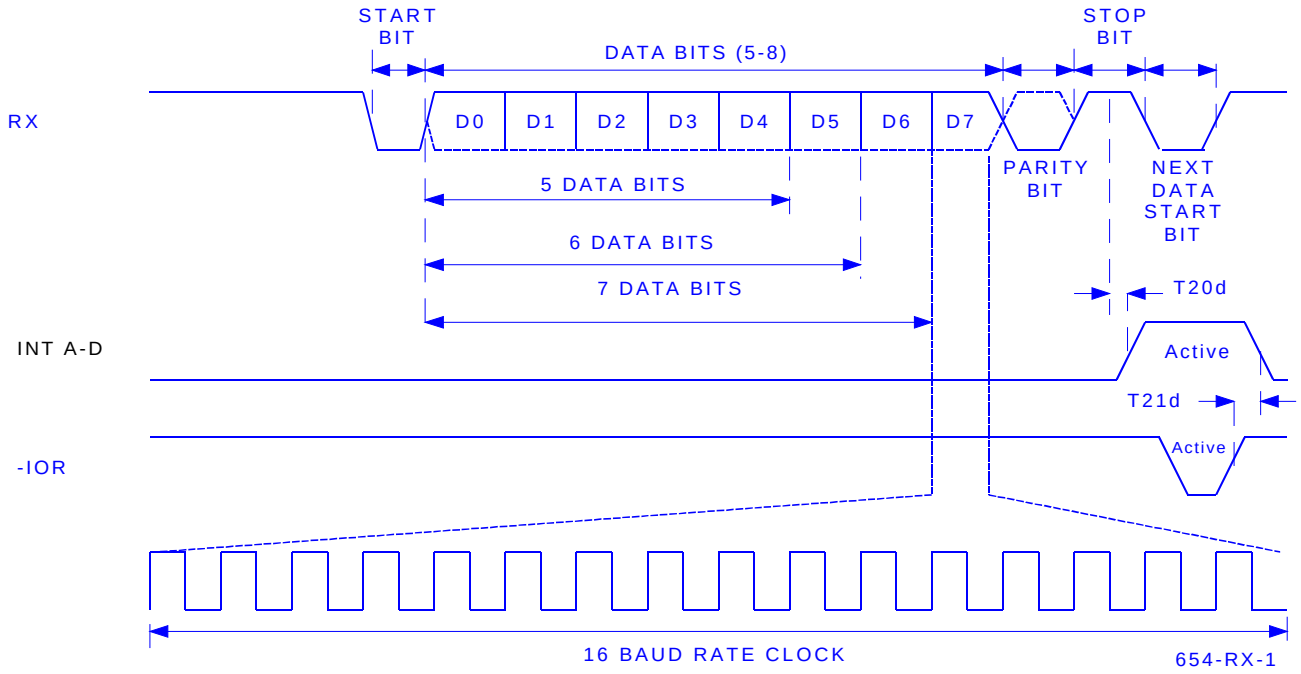
Data Bus Write Timing in 68 Mode



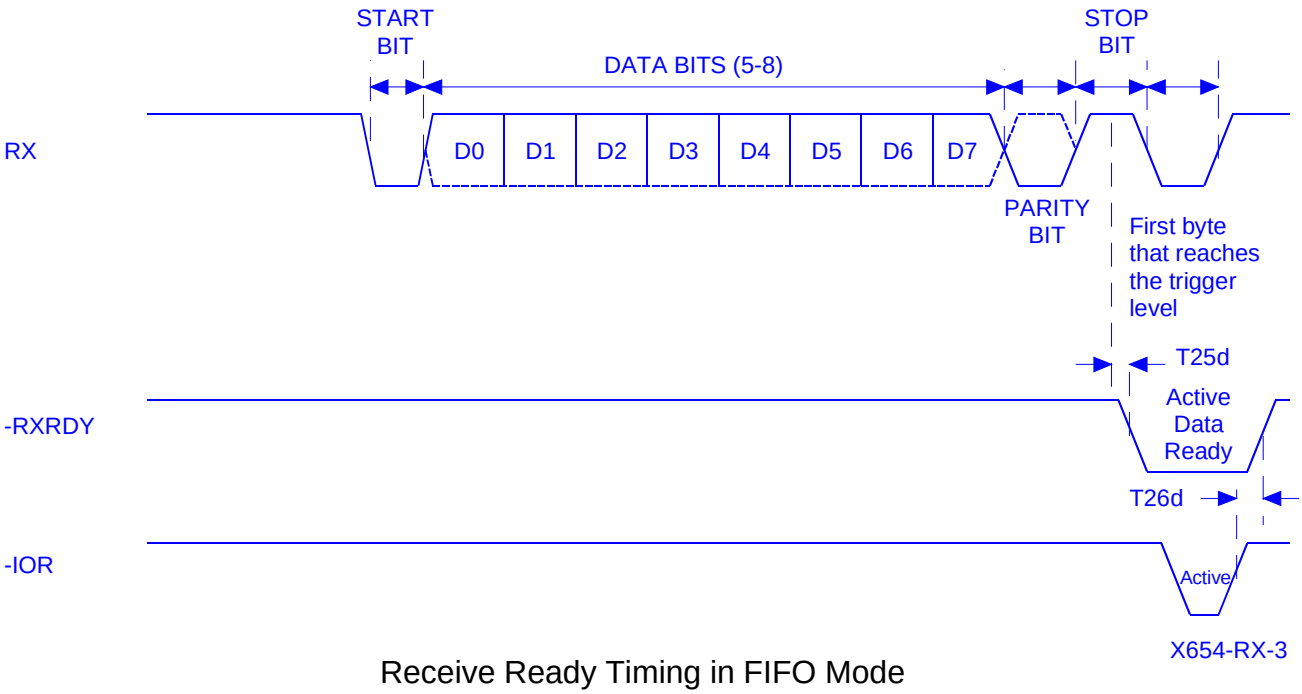
Modem or General Purpose Input/output Timing



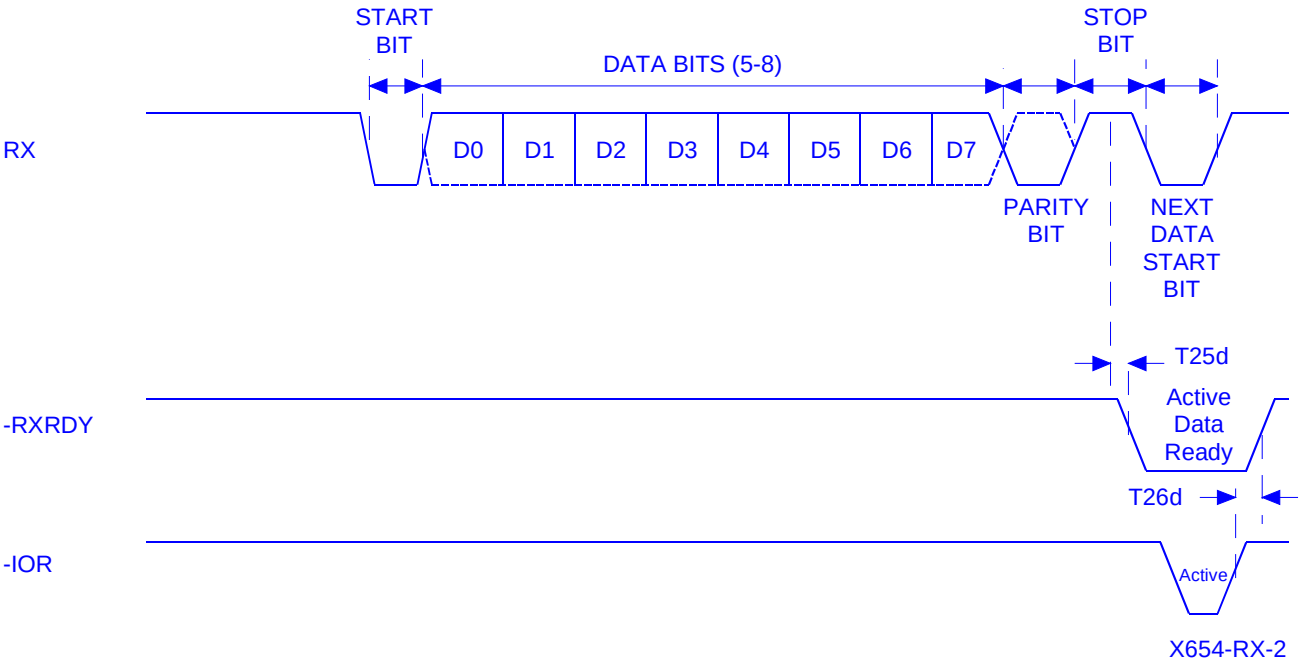
External Clock Timing



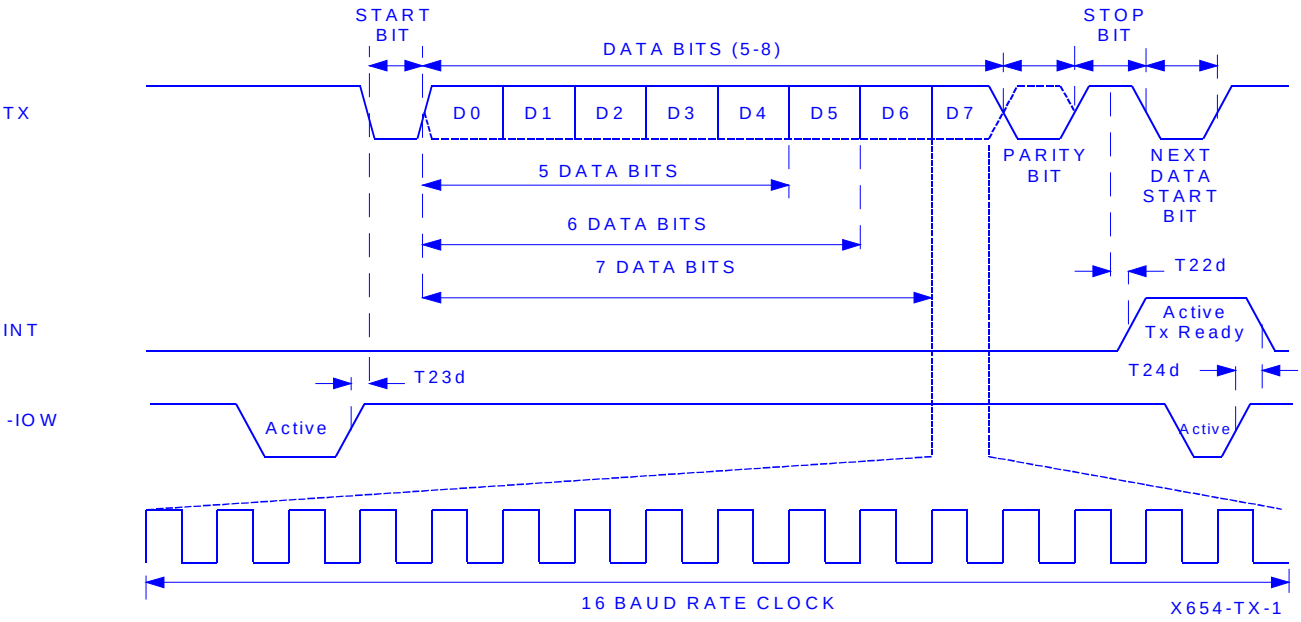
Receiver Timing



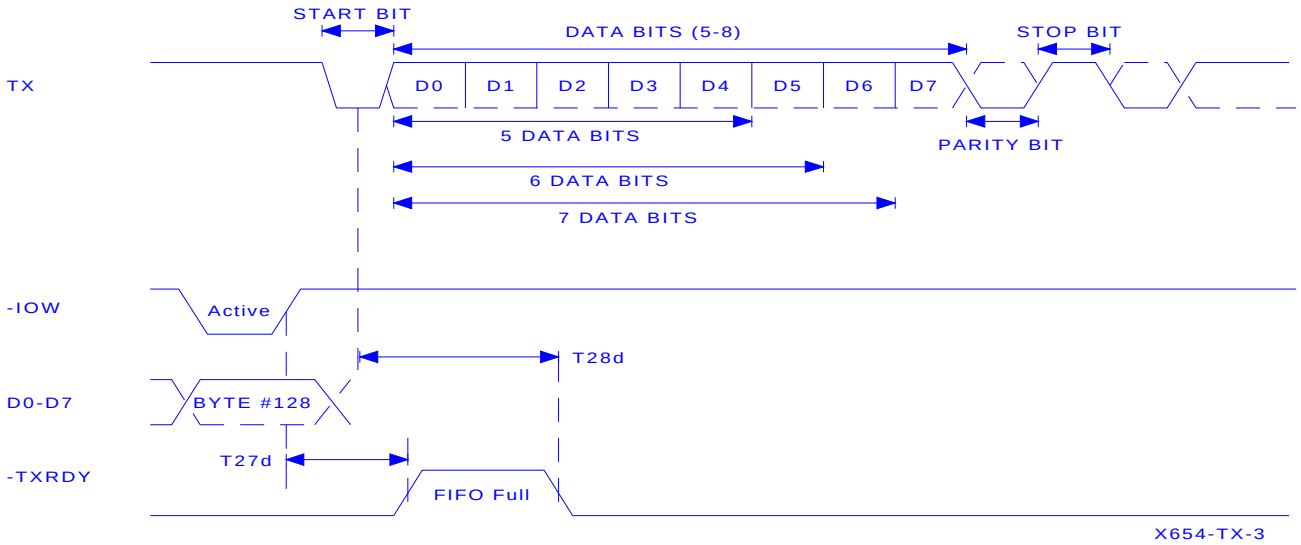
Receive Ready Timing in FIFO Mode



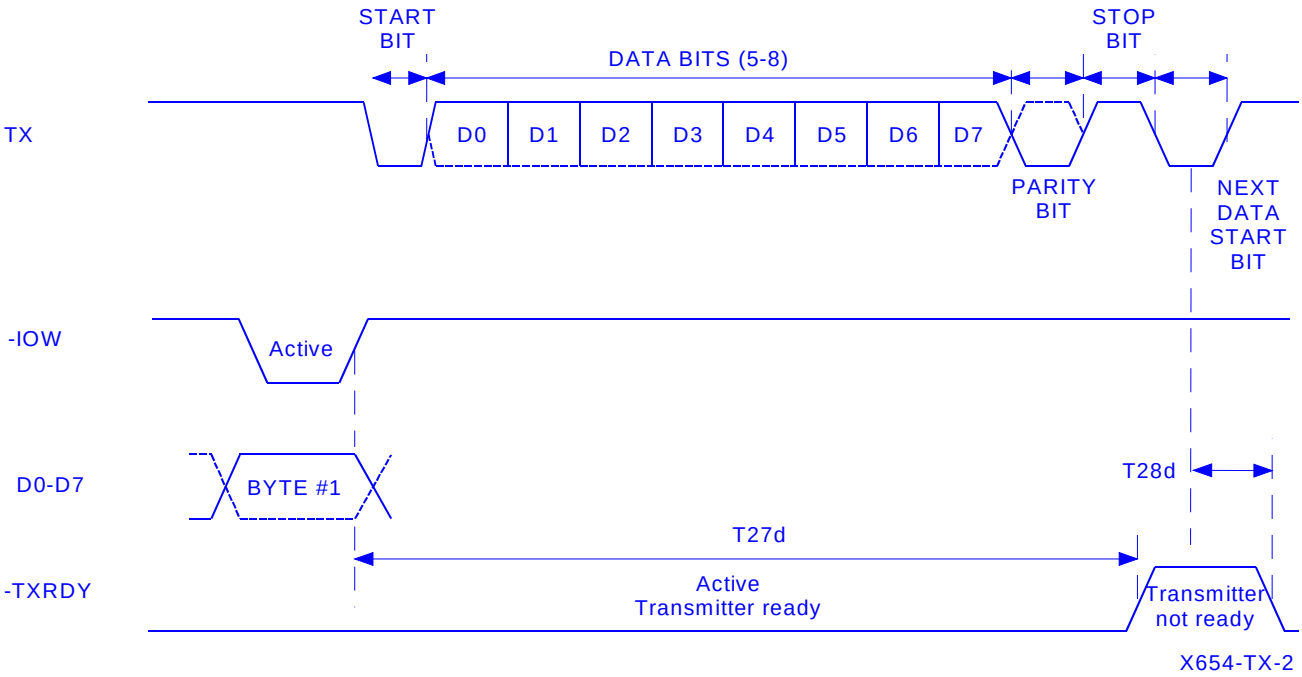
Receive Ready Timing in non FIFO Mode



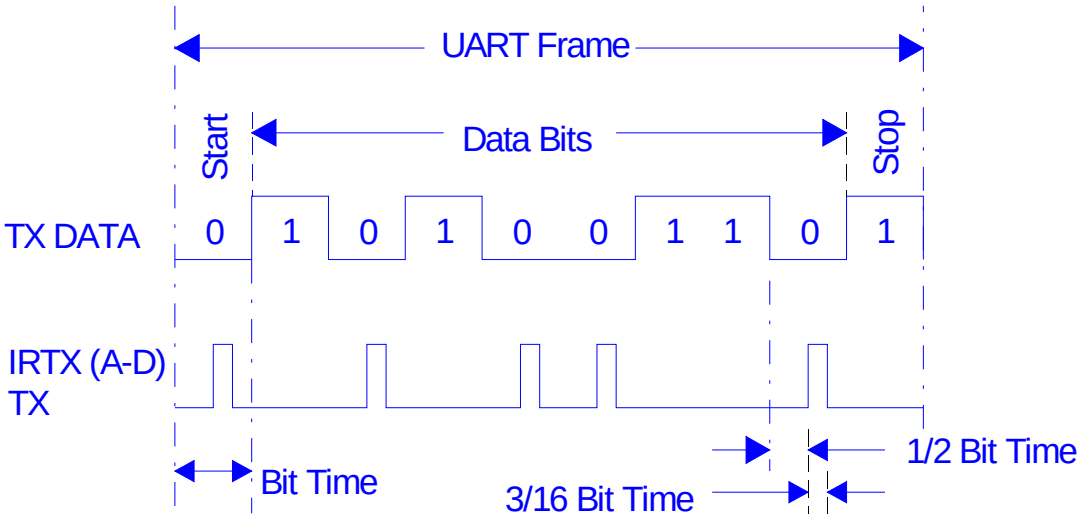
Transmitter Timing



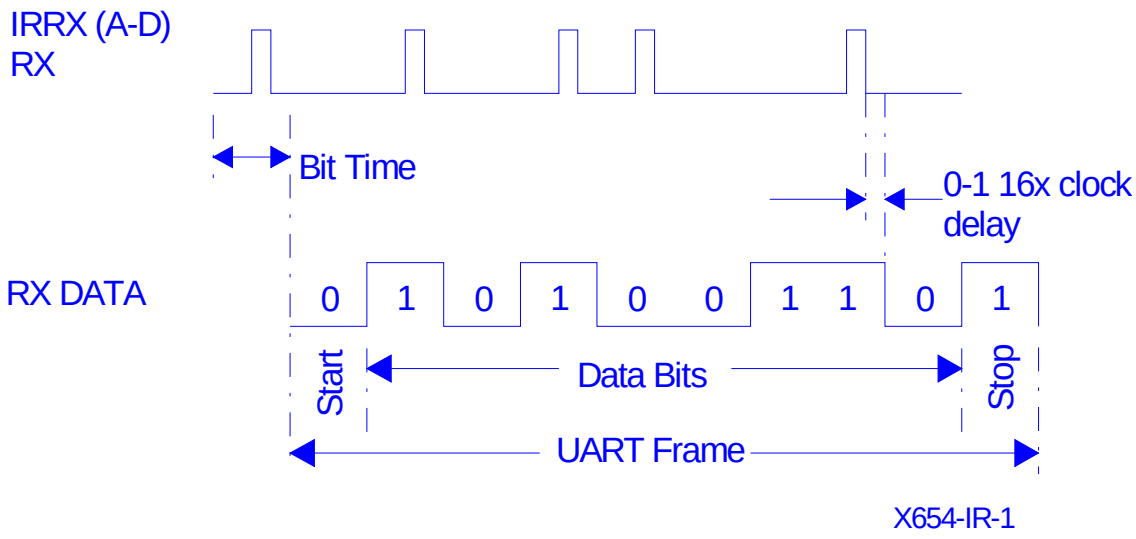
Transmit Ready Timing in FIFO Mode



Transmit Ready Timing in non FIFO Mode



Infrared Transmitter/Encoder Timing

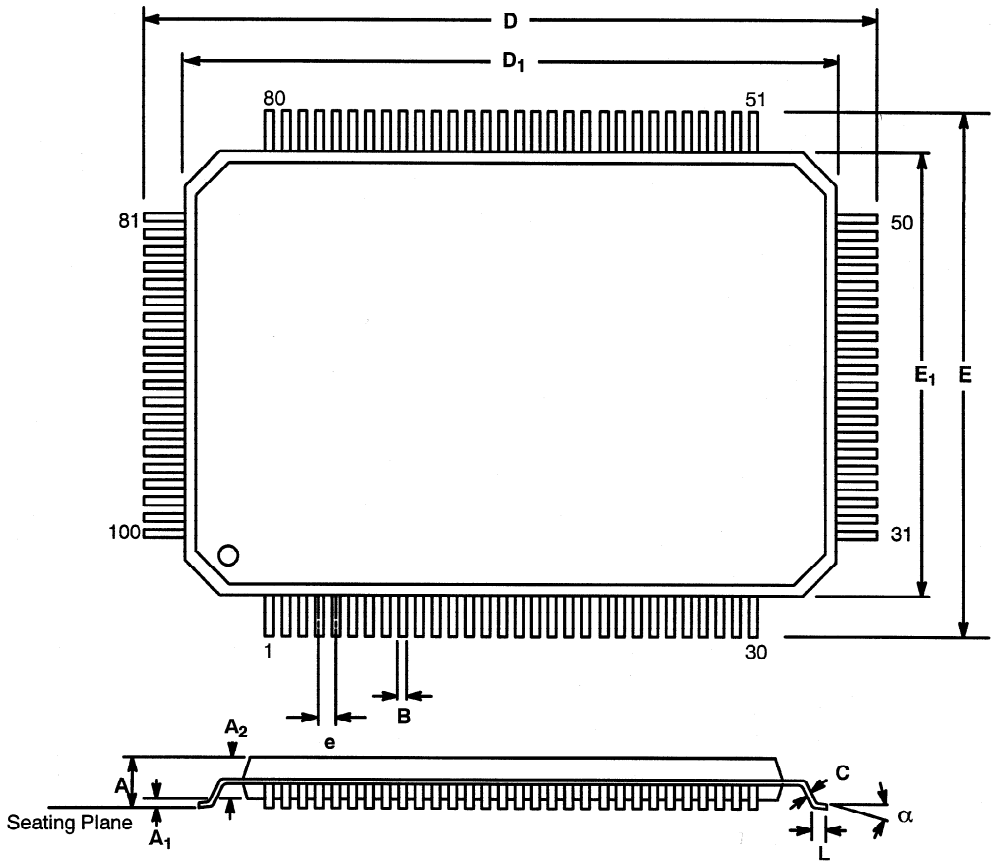


Infrared Receiver/Decoder Timing

Package Dimensions

100 LEAD PLASTIC QUAD FLAT PACK
(14 mm x 20 mm, QFP)

Rev. 2.00



1.95 mm Form

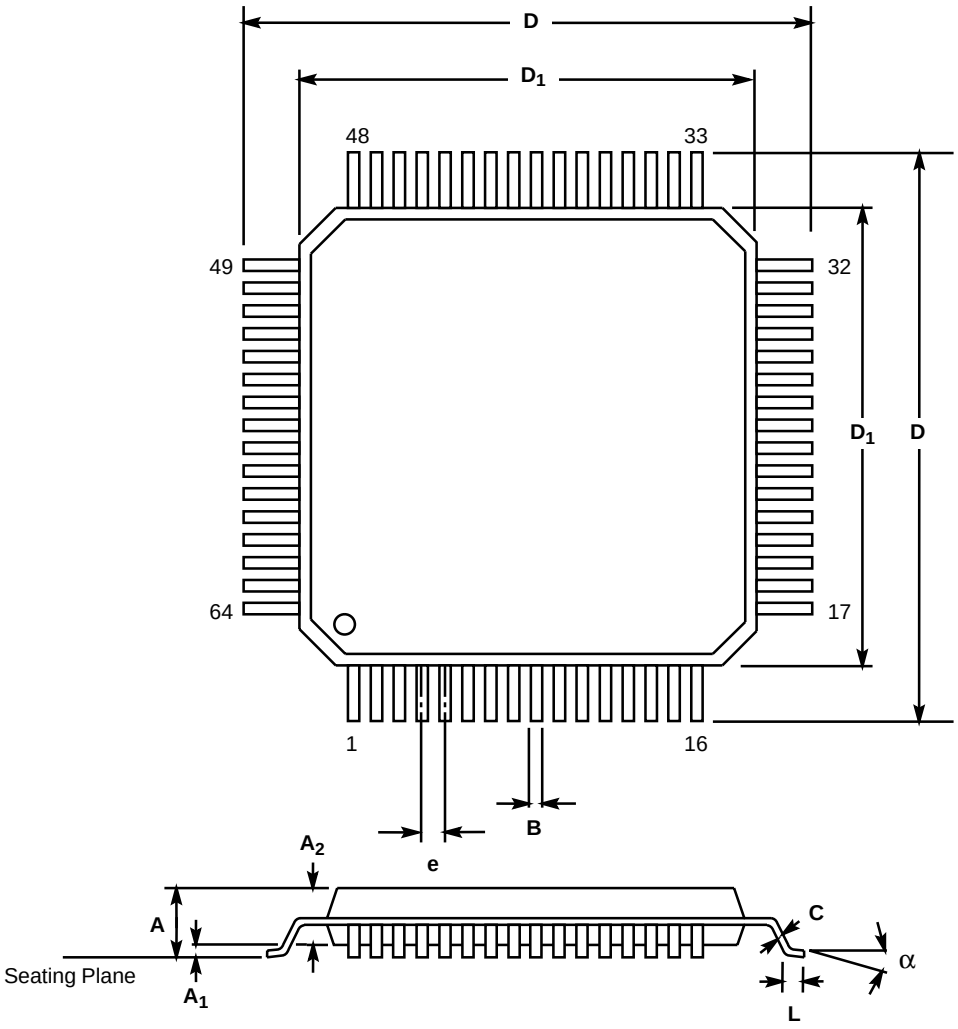
SYMBOL	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.102	0.134	2.60	3.40
A_1	0.002	0.014	0.05	0.35
A_2	0.100	0.120	2.55	3.05
B	0.009	0.015	0.22	0.38
C	0.005	0.009	0.13	0.23
D	0.931	0.951	23.65	24.15
D_1	0.783	0.791	19.90	20.10
E	0.695	0.715	17.65	18.15
E_1	0.547	0.555	13.90	14.10
e	0.0256 BSC		0.65 BSC	
L	0.026	0.037	0.65	0.95
α	0°	7°	0°	7°

Note: The control dimension is the millimeter column

Package Dimensions

64 LEAD THIN QUAD FLAT PACK
(10 x 10 x 1.4 mm, TQFP)

Rev. 2.00



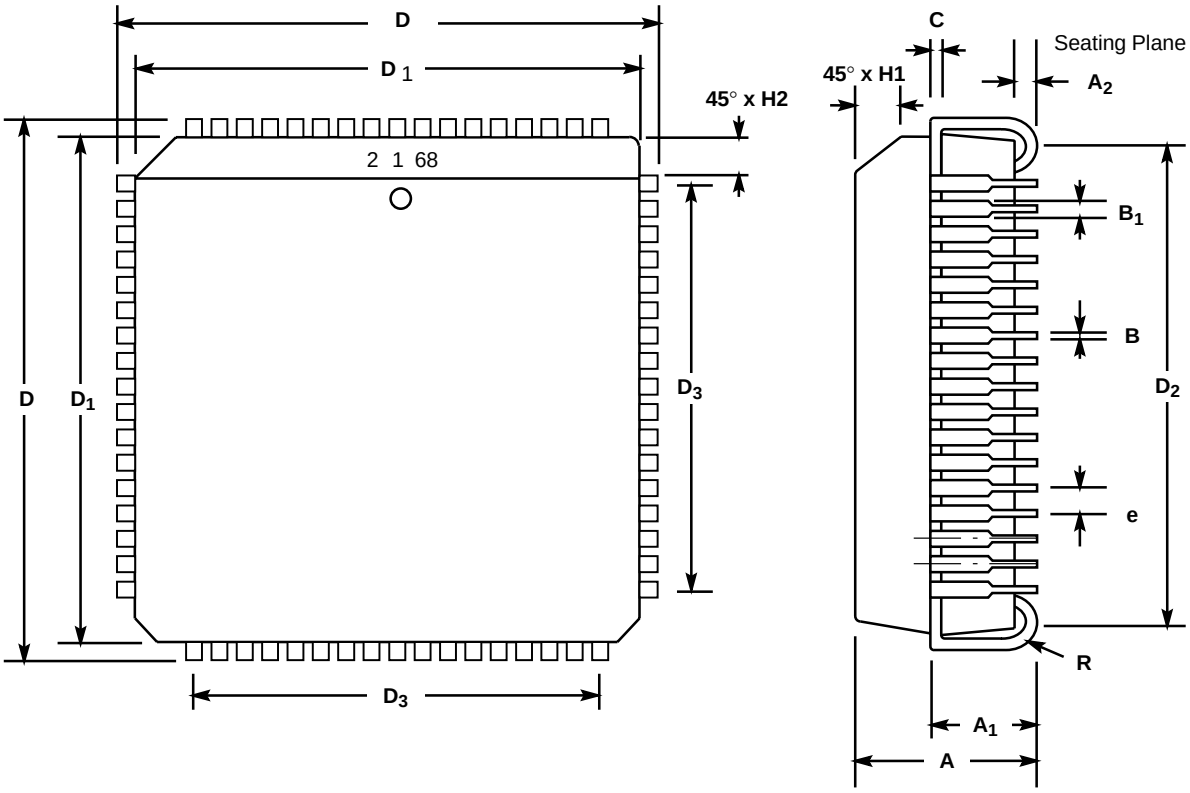
SYMBOL	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.055	0.063	1.40	1.60
A_1	0.002	0.006	0.05	0.15
A_2	0.053	0.057	1.35	1.45
B	0.005	0.009	0.13	0.23
C	0.004	0.008	0.09	0.20
D	0.465	0.480	11.80	12.20
D_1	0.390	0.398	9.90	10.10
e	0.020 BSC		0.50 BSC	
L	0.018	0.030	0.45	0.75
α	0°	7°	0°	7°

Note: The control dimension is the millimeter column

Package Dimensions

68 LEAD PLASTIC LEADED CHIP CARRIER (PLCC)

Rev. 1.00



SYMBOL	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.165	0.200	4.19	5.08
A_1	0.090	0.130	2.29	3.30
A_2	0.020	—.	0.51	—
B	0.013	0.021	0.33	0.53
B_1	0.026	0.032	0.66	0.81
C	0.008	0.013	0.19	0.32
D	0.985	0.995	25.02	25.27
D_1	0.950	0.958	24.13	24.33
D_2	0.890	0.930	22.61	23.62
D_3	0.800 typ.		20.32 typ.	
e	0.050 BSC		1.27 BSC	
H1	0.042	0.056	1.07	1.42
H2	0.042	0.048	1.07	1.22
R	0.025	0.045	0.64	1.14

Note: The control dimension is the inch column

NOTICE

EXAR Corporation reserves the right to make changes to the products contained in this publication in order to improve design, performance or reliability. EXAR Corporation assumes no responsibility for the use of any circuits described herein, conveys no license under any patent or other right, and makes no representation that the circuits are free of patent infringement. Charts and schedules contained here in are only for illustration purposes and may vary depending upon a user's specific application. While the information in this publication has been carefully checked; no responsibility, however, is assumed for in accuracies.

EXAR Corporation does not recommend the use of any of its products in life support applications where the failure or malfunction of the product can reasonably be expected to cause failure of the life support system or to significantly affect its safety or effectiveness. Products are not authorized for use in such applications unless EXAR Corporation receives, in writing, assurances to its satisfaction that: (a) the risk of injury or damage has been minimized; (b) the user assumes all such risks; (c) potential liability of EXAR Corporation is adequately protected under the circumstances.

APPENDIX B

EXAR 16C554 Datasheet Reprint

EXAR 16C554 – Quad UART Datasheet Reprint	EXAR 16C854.pdf

Cable Drawings

Part Number	Description
CBL-173-1	20-pin ribbon to two male 9-pin "D" connector adapter cable

Software Examples

Windows NT/2000/XP Registry changes for using shared interrupts with PCM-COM4A and PCM-COM8	NTCOM4Example HTML
PCM-COM8 Examples Files	pcom8.zip
Using Shared Interrupts with Linux	linux_com4_shared.pdf

Telephone: 817-274-7553 . . Fax: 817-548-1358
http://www.winsystems.com . . E-mail: info@winsystems.com

WARRANTY

WinSystems warrants that for a period of two (2) years from the date of shipment any Products and Software purchased or licensed hereunder which have been developed or manufactured by WinSystems shall be free of any material defects and shall perform substantially in accordance with WinSystems' specifications therefore. With respect to any Products or Software purchased or licensed hereunder which have been developed or manufactured by others, WinSystems shall transfer and assign to Customer any warranty of such manufacturer or developer held by WinSystems, provided that the warranty, if any, may be assigned. The sole obligation of WinSystems for any breach of warranty contained herein shall be, at its option, either (i) to repair or replace at its expense any materially defective Products or Software, or (ii) to take back such Products and Software and refund the Customer the purchase price and any license fees paid for the same. Customer shall pay all freight, duty, broker's fees, insurance changes and other fees and charges for the return of any Products or Software to WinSystems under this warranty. WinSystems shall pay freight and insurance charges for any repaired or replaced Products or Software thereafter delivered to Customer within the United States. All fees and costs for shipment outside of the United States shall be paid by Customer. The foregoing warranty shall not apply to any Products or Software which have been subject to abuse, misuse, vandalism, accidents, alteration, neglect, unauthorized repair or improper installations.

THERE ARE NO WARRANTIES BY WINSYSTEMS EXCEPT AS STATED HEREIN. THERE ARE NO OTHER WARRANTIES EXPRESS OR IMPLIED INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, IN NO EVENT SHALL WINSYSTEMS BE LIABLE FOR CONSEQUENTIAL, INCIDENTAL, OR SPECIAL DAMAGES INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF DATA, PROFITS OR GOODWILL. WINSYSTEMS' MAXIMUM LIABILITY FOR ANY BREACH OF THIS AGREEMENT OR OTHER CLAIM RELATED TO ANY PRODUCTS, SOFTWARE, OR THE SUBJECT MATTER HEREOF, SHALL NOT EXCEED THE PURCHASE PRICE OR LICENSE FEE PAID BY CUSTOMER TO WINSYSTEMS FOR THE PRODUCTS OR SOFTWARE OR PORTION THEREOF TO WHICH SUCH BREACH OR CLAIM PERTAINS.

WARRANTY SERVICE

All products returned to WinSystems must be assigned a Return Material Authorization (RMA) number. To obtain this number, please call or FAX WinSystems' factory in Arlington, Texas and provide the following information:

1. Description and quantity of the product(s) to be returned including its serial number.
2. Reason for the return.
3. Invoice number and date of purchase (if available), and original purchase order number.
4. Name, address, telephone and FAX number of the person making the request.
5. Do not debit WinSystems for the repair. WinSystems does not authorize debits.

After the RMA number is issued, please return the products promptly. Make sure the RMA number is visible on the outside of the shipping package.

The customer must send the product freight prepaid and insured. The product must be enclosed in an anti-static bag to protect it from damage caused by static electricity. Each bag must be completely sealed. Packing material must separate each unit returned and placed as a cushion between the unit(s) and the sides and top of the shipping container. WinSystems is not responsible for any damage to the product due to inadequate packaging or static electricity.