DUSCC initialization procedures AN419

Author: Debra Ibarra Revised by: A. Kazmi

INTRODUCTION

The Philips Semiconductors Dual Universal Serial Communications Controller (DUSCC) has forty-eight programmable registers many of which have triple functions that vary with the protocol selected. This application note contains basic initialization instructions and examples Including: description of the input clock circuit; how to use interrupts with status; how to use the digital phase locked loop (DPLL); how to initialize the counter/timer as a system down counter and as a receive character counter. Also included are software examples for the asynchronous local loop back mode, for two synchronous modes (HDLC and BISYNC), and DMA handling.

X2/IDCN PIN

Initially, the X2/IDCN pin is internally programmed as an input pin (used with a crystal) by the assertion of RESET. If an external clock is used as the timing source in place of a crystal, it is necessary to perform the following:

1. If the X2/IDCN is used as an IDCN output, write PCRA[7] = 1 as soon as possible. This allows the IDCN to be used as a daisy chain without delay.
2. If X2/IDCN is not used as the IDCN output, the X2/IDCN pin must be grounded, and leave PCRA[7] = 0.

X1/CLK SOURCES

The DUSCC must have a clock source connected to the X1 input at all times. It can be supplied by a crystal between the X1 and X2 pin, or by driving an external clock Into the X1/CLK input. The frequency must be between 2.0 and 16.0MHz for correct device operation; 14.7456MHz is the nominal frequency which is used to obtain the standard baud rates listed for the internal baud rate generator.

X1/X2 Crystal

The DUSCC oscillator circuitry consists of an inverting amplifier and a feedback resistor which are used to implement a Pierce oscillator. This circuitry will cause the crystal attached between the X1 and X2 pins to go into anti-resonant (parallel) operation. So, while a number of crystal and capacitor combinations will work, obtaining a parallel calibrated crystal and adjusting the external capacitor values until the total circuit capacitance matches the capacitance specified for the crystal will result in the most accurate frequency value. Using two 24pF capacitors, one from X1 to ground and one from X2 to ground, and the parallel crystal recommended below will give accurate, reliable results. The frequency will vary slightly depending on the amount of stray capacitance in the individual circuit, but will typically be off no more than 0.01%. The frequency can be adjusted by trimming the external capacitors; larger capacitors lower the oscillator's frequency and smaller ones raise it.

A source for the 14.7456MHz crystal is: Saronix, Palo Alto, CA. Request part number NYP147-20.

Externally Driven Clock

The most important point in using an external source to drive the X1/CLK input is to meet the V_{IH} specification and the minimum high and low times of 25ns. Also, when driving a clock into X1, be sure to ground the X2 pin if it is not programmed to be IDCN out.

INTERRUPT STATUS

The DUSCC is equipped with an interrupt vector register (IVR) that can be modified. On power up, the IVR is initialized to a fixed value. If ICR[2] = 1, the status registers will modify the interrupt vector. If the vector is modified, ICR[3] must be programmed to modify either vector bits [2:0] or [4:2]. ICR[7:6] dictates which channel will have the highest priority or whether interrupt priorities are interleaved between channel A and channel B.

If a modified vector is used, the three modified bits in the vector will reflect the interrupting condition. For example, if channel B transmitter is ready, the modified bits = 101. Interrupts can be programmed to occur from either the RSR, the TRSR, the ICTSR registers, or the TXRDY and RXRDY conditions. These registers allow either transmit, receive, input, or C/T status bits to set the appropriate GSR bit and force an interrupt.

EXAMPLES – Option: Channel A in Synchronous Mode

Assume channel A is in the COP mode with the receiver sampling off the DPLL output which is programmed to use FM0 encoding. The user chooses to monitor the following items: EOM detect RSRA[7], CRC errors RSRA[1:0], and check for DPLL errors TRSRA[3].

Because an underrun of the transmitter, or an overrun of the receiver can be a problem, program OMRA[4:3]=00. Writing IERA = 79H, will enable all of the above conditions to interrupt as soon as ICR[1] is set, and the transmitter and receiver are enabled. The status of these bits is reflected in GSR[3:0].

Option: Channel B in Asynchronous Mode

Assume channel B is initialized to check for framing and parity errors (RSRB[1:0]) and the counter/timer is to be used as a system down counter. For the user to know when the C/T reaches zero count (ICTSRB[6]), set CTCRB[7] so that the C/T can generate an interrupt. RXRDY can be set to activate when the FIFO fills (OMRB[3] = 1). However, the receiver could be overrun if the baud rate is high or the CPU is latent in reading the receive FIFO. Writing IERB= 51H, will enable all of the above conditions when ICR[0] is set, and the transmitter and receiver are enabled. The status of these bits is reflected in GSR[7:4]

NOTE: Because the synchronous channel has more of a tendency to underrun, channel A has interleaved priority. This means that channel A takes priority when events of equal weight occur in both channels simultaneously; therefore, ICR = 87H.

The interrupts are activated when the transmitter and receiver are enabled. Although some of these interrupts reset themselves when the interrupt routine services the DUSCC, it is safer to reset the status bits that caused the interrupt by writing 1 to each status bit that is set. Resetting the bits in the RSR, TRSR, and ICTSR registers, automatically resets the corresponding bits in the general status register (GSR). If TXRDY or RXRDY causes the interrupt, these can be cleared by writing/reading data to/from Tx/RxFIFOs.

COUNTER/TIMER

Each channel of the DUSCC contains a counter/timer that can be used as either a down counter, a delay timer, an external event counter, a TX or RX character counter, a bit length measurement tool, or a timer to generate almost any baud rate. The C/T can be clocked from a number of sources.

Table 1 gives the maximum frequency generated by the C/T with various inputs (prescaler = 1).

The C/T must be loaded with a minimum count of N = 2 which will divide the maximum timer output frequency to one-half if the pulsed mode is programmed, or to one-fourth if the square mode is

Table 1. Maximum Frequency Generated by C/T

Clk Source	Maximum Outputs			Notes
	Pulsed	Square		
RTXC or TRXC	2	1	MHz	TRXC/RTXC = 4MHz
X1/CLK divide by 4	2	1	MHz	X1/CLK = 16MHz
RXBRG or TXBRG (16X)	307.2	153.6	kHz	RX/TX = 38.4K

Table 2. Encoding Schemes

Encoding	Where Sampled	How Decoded
NRZ	Center of bit time	Low = 0, high = 1
NRZI	Center of bit time	Transition since cell = 0, no transition from last cell = 1
FM0	1st and 3rd 1/4 of bit time	Cell halves dissimilar = 0, cell halves same = 1
FM1	1st and 3rd 1/4 of bit time	Cell halves same = 0, cell halves dissimilar = 1
Manchester	Center of bit time	Low to high transition = 0, high to low transition = 1

selected. (Pulsed mode forces the output low for N = 1 periods and then high for one period; square mode forces the output low for N periods and high for N periods.) The C/T is controlled by five registers. CTPRH and CTPRL hold the preset counter value that will be loaded when a start counter command is issued. CTH and CTL hold the current value contained in the C/T and can be read at any time. For best results, stop the C/T prior to reading the CTH and CTL. The CTCR register controls the C/T clock source, prescaler, type of output (pulse or square wave) and the direction taken when the C/T counts down and passes zero detect.

Counter/Timer Used as a Down Counter

The following procedures initialize the C/T as a system down counter, set to interrupt the CPU every 2ms:

1. Using (X1/CLK)/4, the typical X1 rate is 14.7456MHz (t = 67.82ns)
2. Divide the X1 rate by 4 ⇒ 3.6864MHz (t = 271.27ns).
3. The following equation determines the preset count:

$$\text{Count value} = \frac{\text{source freq}}{\text{desired freq} \times 2}$$

(for square wave output)

$$\text{Count value} = \frac{\text{source freq}}{\text{desired freq} \times 2}$$

(for on shot pulse output)

$$\text{Count value} = \frac{3.6864 \times 10^{6}}{(\frac{1}{4} \times 10^{-3}) \times 2}$$

$$= 7372.8 \text{ (dec)}$$

$$= \text{ICCD (hex)}$$

1. Write CTCR = 82H (prescale = 1, X1/4, enable INT (CTCR161 = 0, preset loaded on zero)
2. Write CTPRH = 1CH (preset high value)
3. Write CTPRL = CDH (preset low value)
4. Write ICR = 01 or 02 (enable master interrupt (A or B))
5. Write CCR = 83H (transfer preset value)
6. Write CCR = 80H (issue start C/T command)

After the interrupt occurs:

7. Write ICTSR = 40H (reset C/T status)
8. Wait for next interrupt, preset value automatically reloaded.

Counter/Timer Used as a Character Counter

Assume that characters are being received in the asynchronous mode and the user chooses to accumulate these characters in RAM in blocks of 2048 bytes each. The preset value will be 2048 or 800H. Each time the count reaches zero, the counter/timer interrupts the CPU. The only difference between the C/T as a down counter, and the C/T as a character counter, is that CTCR = 06 which assigns the C/T clock source to be the receive character pulse. The CTPR is loaded with the block count of 800H. Because this particular example is repetitive, the user resets the C/T interrupt (ICTSR = 40H) after each block count has been reached. Since CTCR16] = 0, the preset value will automatically reload on zero detect.

DIGITAL PHASE LOCKED LOOP (DPLL)

The DPLL is designed to be used with the transmission of synchronous data without using a separate transmit clock line. Data is sampled by a 32X clock searching for a transition on data entering the receiver. Once the enter search mode command is issued, the DPLL will start to synchronize on the first data transition. The user must ensure that RXD has no spurious noise which could cause the DPLL to begin to synchronize on the wrong starting edge. Since sampling takes place in parts of the bit cell, the FM mode has the greatest potential for this error.

If noise should trigger the DPLL, it will set TRSR[3]=1, and the DPLL will enter into a search mode automatically.

Table 2 describes five different encoding schemes and where in the bit cell the DPLL will sample data entering the receiver. For reference, see DPLL Waveforms in the DUSCC data sheet.

The DPLL maximum clock frequency is 250KHz. This value can be achieved by using the X1/CLK at 16MHz (divide by 64). A slower speed of 125KHz can be derived by using an external TXC or RXC (TXC or RXC set at 4MHz divided by 32). For complete details of how the DPLL samples a data stream and corrects its clocking edges, refer to the DUSCC data sheet on NRZI and FM mode operation.

An example of how to initialize the receiver sampling data with the DPLL using a 38.4KHz rate in the NRZI mode (BRG clock source) follows.

1. Write RTR = 6FH; (DPLL (BRG), BRG = 38.4K)
2. Write CCR = C3H; (Set DPLL mode = NRZI)
3. Write CCR = 40H; (Reset receiver)
4. Write CCR = C0H; (DPLL enter search mode)

ASYNCHRONOUS MODE

Regardless of the channel connection used in CMR2[7:6], the basic initialization in asynchronous mode is the same. Note that whenever a change is made in the channel mode registers, the transmitter and receiver should first be disabled. Also, before any change is made in the transmit parameter register (TPR) or the transmit timing register (TTR), the transmitter should be disabled.

After a change(s) has been completed, the transmitter should be reset; then enabled.

If changes are made in either the receive parameter register (RPR) or the receive timing register (RTR), the receiver should be disabled. After a change(s) has been completed, the receiver should be reset; then enabled.

The software illustration (Figure 1) demonstrates the minimum number of registers needed to program the DUSCC for asynchronous transmission. This routine is also an example of how the integrity of the system can be checked in local loop back by writing, reading, and verifying 256 characters through each channel. Since the comments in the software examples are extensive, there are no flow charts provided in this application note. If a reset pulse greater than 1.2µs is asserted on the reset input after V_{DD} has stabilized, most registers will be reset to zero. Many of these registers can be disregarded during initialization. The registers that must be programmed are the channel mode registers (CMR1/2), channel command registers (CCR), and the parameter and timing registers (TPR, TTR, RPR, RTR) for both the transmitter and receiver.

SYNCHRONOUS PROTOCOLS

The SCN68562 will work with almost any synchronous protocol at rates exceeding maximum speeds recommended by CCITT standards (USA = 1.5MHz; Europe = 2.04MHz). While asynchronous transmission frames are sent on a character by character basis (including, start, stop, and parity bits), synchronous transmission requires a protocol that works with a predefined frame. Within each transmission frame there exists four main parts:

1. Synchronizing characters
2. A header defining addresses and/or control information
3. The physical data stream
4. 4. Block character check (BCC)

Most synchronous transmissions fall into two main categories, bit-oriented protocols (BOP) and character-oriented protocols (COP). This document describes the two most widely used protocols, an HDLC sample for BOP, and a BISYNC sample for COP.

The SCN68562 handles many of the operations necessary to send and receive a BOP or COP frame. CMR1[2:0] allows the user to select either BOP or COP with options. The synchronous/secondary address registers (S1R, S2R) are used to initialize synchronizing characters (COP) or the frame address (BOP). TPR[7:4] defines underrun control, fill pattern during an idle transmit state, and when TEOM is to be transmitted. RPR[7:3] defines which SYN patterns should be stripped, should the frame check sequence be sent to the RXFIFO, and what action the receiver should take after EOM is received or an overrun occurs. These bits also inform the receiver when external synchronous pulses will be used to synchronize received data, and if parity is to be stripped before assembly.

BOP TRANSMISSION

The software example in Figure 2 demonstrates how the user can initialize the SCN68562 for transmitting with the HDLC protocol using the BOP mode. In this example the channel is set up as a secondary station (CMR1 = 01) using an eight-bit address mode without extended control. This means that an eight-bit address must follow the opening flags of each frame. The address identifies the receiver which is to be the data destination. If this channel was programmed to be a primary (master) station, no address comparison would be necessary. For simplification, CMR2 is set up so that no FCS character is issued when the end of message command is executed. The transmitter parameter register is set to issue ABORT (FF) followed by FLAGs (7E) if the transmitter is underrun. Also, the idle state of TXD is defined as FLAGs (7E) between frames. The receive parameter register is programmed with the overrun mode set to hunt. This will force the receiver to terminate the frame when the RXFIFO and RX shift register are full, and then hunt for flags. Finally, S1R is loaded with the receiver compare address.

The start of message (SOM or TSOM) command must be issued to send out the opening synchronous flags. This is followed by the receiver address and data loaded into the TXFIFO. The frame is ended when the CPU sends an EOM or TSOM command to the command register. This must be done just before the last character is loaded into the TXFIFO. The EOM command is then appended to the next character written into the TXFIFO. After this character is shifted out, the DUSCC will issue the FCS (if programmed to do so) followed by a closing flag.

Note also that the receiver cannot use the BRG directly, unless the asynchronous mode is programmed. Therefore, (RTRA = 6F), the DPLL (at 32X), is selected. In this case the transmitter will not know what encoding scheme to use until the set NRZI command (CCRA = C3) is sent to initialize the DPLL. Immediately before reception starts, the DPLL is turned on with an enter search mode command (CCRA = C0). Since TXD is tied to RXD in this example, the receiver will look for the first byte, after the opening flag, to be an address. The address is compared to the value in S1R, and if a match occurs, the address and all other data are loaded into the RXFIFO.

COP TRANSMISSION

An example using the COP mode follows (see Figure [3]). This example is set up almost identical to BOP. The key character-oriented protocol differences are the synchronize and control characters. Before transmission begins, SYN1 and SYN2 characters are loaded into S1R and S2R. These characters are issued with the TSOM command to open the COP frame. If an underrun occurs in the transmitter, or a frame ends, TXD is filled with SYNC1/SYNC2 characters (TPRA = E3). None of the synchronized characters are transferred to the receive holding register if SYN stripping is invoked (RPRA = 83).

The remaining differences occur with special control characters that define the beginning and ending of text, the header field, and various acknowledgements. A few of these commands can be issued through the command register, but the majority must be part of the data stream.

DUSCC WITH DMA CONTROLS

See Figures 4 and 5 for software examples. The schematic (Figure 6) defines the typical handshake lines necessary for the DUSCC to interface with the SCB68430, Direct Memory Access Interface (DMAI). The DUSCC will interface directly with any of the 68K family of DMA controllers (68430, 68440, 68450).

The DUSCC is shown in the half duplex single address configuration. The request pin (RTXDRQAN) initializes the DMA cycle for both the transmitter and the receiver in this mode. Once the controller has finished arbitration for the bus, acknowledgment is given to the DUSCC on the RTXDAKA input. After acknowledgment is made, RDYN is driven low by the DTACKN output. DTACKN from the DUSCC must be isolated from system DTACKN when the DMA controller has control of the bus. After each character is transferred to the transmitter, or when data is read from the receiver, DTCN is asserted. When transfer count has been reached, DONEN is asserted, ending the DMA.

DMA SOFTWARE

Two simple software programs are provided. Both are initialized with channel A in half duplex single address mode and channel B in normal polled mode. The comments in each sample explain the basic initialization.

The first software sample (Figure 4) is a transmit DMA. TXDA is tied to RXDB by a wire. The CPU loads the RAM with 256 characters and then initializes the the DMA controller. As the transmitter becomes ready and asserts IRQN, the DMA controller will acknowledge by loading the TxFIFO. After instructing the DMAI to start, the CPU reads the GSR looking for the receiver to become ready. Once RXRDY = 1, the character is read and compared against the expected character that was transmitted. The DMA transfer rate in burst mode is faster than the time needed for the 8MHz 68000 to test for and read the received characters. Therefore, flow control is used via RTS and CTS. In this case, the RxB controls RTS/CTS which controls TxA.

The second software example (Figure 5) is a receive DMA. RXDA is tied to TXDB with a wire. This example is the reverse of the previous software routine. The CPU loads 256 characters into the TXB FIFO as it becomes ready. The DMA controller reads RXA FIFO and stores the received data in RAM. When the count is complete, the CPU reads the RAM and verifies whether the data transferred is correct. No flow control is needed since the DMA controller can read the received characters faster than the RXA FIFO can fill up.

NOTE: Please note that the following examples are used for illustration purposes only. User must verify their software in an actual system use.

Basic asynchronous initialization for DUSCC (local loopback mode). This routine will send 256 characters to both channel A and B. The looped back characters are read and verified by the CPU.

```
            Begin
            ;
            ;   SETUP   MOVE.B  #07,        CMR1A     ;NO PARITY, ASYN MODE
                        MOVE.B  #07,        CMR1B
                        MOVE.B  #$B8,       CMR2A     ;LOCAL LOOPBACK, POLLED/INT
                        MOVE.B  #$B8,       CMR2B
                        MOVE.B  #$73,       TPRA      ;1 STOP BIT, 8 BIT CHAR
                        MOVE.B  #$73,       TPRB
                        MOVE.B  #$3E,       TTRA      ;TX=BRG CLK, 19.2K BAUD
                        MOVE.B  #$3E,       TTRB
                        MOVE.B  #03,        RPRA      ;8 BIT RX CHAR
                        MOVE.B  #03,        RPRB
                        MOVE.B  #$2E,       RTRA,     ;RX=BRG CLK, 19.2K BAUD
                        MOVE.B  #$2E,       RTRB
                        MOVE.B  #0,         CCRA      ;RESET TX
                        MOVE.B  #0,         CCRB
                        MOVE.B  #$40,       CCRA      ;RESET RX
                        MOVE.B  #$40,       CCRB
                        MOVE.B  #02,        CCRA      ;ENABLE TX
                        MOVE.B  #02,        CCRB
                        MOVE.B  #$42,       CCRA      ;ENABLE RX
                        MOVE.B  #$42,       CCRB
            ;
            ;           WRITE/READ/VERIFY DATA IN LOCAL LOOPBACK MODE
            ;
                        CLR.L   D6                    ;CLR D6
                        MOVE.L  #GSR,A0               ;A0=POINTER TO STATUS REG
            AGAIN:      MOVE.B  D6,         TXFIFA    ;LOAD TX FIFO (A & B)
                        MOVE.B  D6,         TXFIFB
            CHA:        MOVE.B  [A0],       D2        ;CHECK FOR RXA READY
                        BTST    #0,         D2
                        BEQ     CHA                   ; WAIT FOR RX TO COME READY
                        MOVE.B  RXFIFA,     D0        ;READ RX FIFO A
                        CMP.B   D0,         D6        ;DOES RXA CHAR=TXA CHAR?
                        BEQ     CHB                   ;IF YES, THEN CHECK RXB
                        MOVE.W  #1,         D7        ;NO, THEN FLAG ERROR AND
            STOP        TRAP    #15
            CHB:        MOVE.B  RXFIFB,     D1        ;IS TX CHAR = RX CHAR (CHAN B)
                        CMP.B   D1,         D6
                        BEQ     INCD6                 ;YES, THEN INC D6
                        MOVE.W  #2          D7        ;NO, THEN FLAG ERROR AND
            STOP        TRAP    #15
            INCD6:      ADDI.B  #1,         D6        ;INC D6
                        CMPI.B  #00         D6        ;HAS D6 ROLLED OVER TO 00?
                        BNE     AGAIN                 ;IF NO, KEEP GOING 0–FF
                        MOVE.W  #0,         D7        ;IF YES, CLR ERROR AND STOP
                        TRAP    #15
            ;
                        END     SETUP
```

Figure 1. Minimum Number of Registers Needed to Program the DUSCC

This file sets up channel A to transmit and receive four BOP characters in the HDLC format. TXDA is tied to RXDA by a wire. Note that the end-of-message command is issued to the CCR before the last character is loaded in the transmitters.

```
Begin
;
                MOVE.B  #1,         CMR1A     ;BOP SECONDARY MODE, 8 BIT ADR, 8 BIT CONTROL
                MOVE.B  #$38,       CMR2A     ;POLLED/INT MODE, NORMAL MODE, NO FCS
                MOVE.B  #$3F,       TTRA      ;TX=38.4K BAUD
                MOVE.B  #$6F,       RTRA      ;RX=38.4K, DPLL X32 FROM BRG
                MOVE.B  #$E3,       TPRA      ;TX 8 BIT/CHAR,UNDERRUN=ABORT–FLAG,IDLE=FLAGS
                MOVE.B  #3,         RPRA      ;RX 8 BIT/CHAR,OVERRUN=HUNT,NO FCS OR EXT SYNC
                MOVE.B  #$F7,       OMRA      ;TXRDY=EMPTY, RXRDY=NOT EMPTY, NO RESID CHAR
                MOVE.B  #$FF,       S1RA      ;RX ADDRESS=HEX FF (SETUP FOR 8 BIT ADDRESS)
                MOVE.B  #0,         CCRA      ;RESET TX MOVE.B #$40,CCRA ; RESET TX
                MOVE.B  #2,         CCRA      ;ENABLE TX MOVE.B #$42,CCRA ; ENABLE RX
                MOVE.B  #$C3,       CCRA      ;SET NRZ MODE FOR DPLL
                MOVE.B  #$C0,       CCRA      ;ENTER SEARCH MODE
;
                BSR TXRDY                     ;WAIT FOR TXRDY
                MOVE.B  #4,         CCRA      ;TRANSMIT START OF MESSAGE
                MOVE.B  #$FF,       TXFIFA    ;TRANSMIT HEX FF
                MOVE.B  #0,         TXFIFA    ;TRANSMIT HEX 0
                MOVE.B  #$AA,       TXFIFA    ;TRANSMIT HEX AA
                MOVE.B  #6,         CCRA      ;TRANSMIT END OF MESSAGE
                MOVE.B  #$55,       TXFIFA    ;TRANSMIT HEX 55
;
                BSR RXRDY                     ;WAIT FOR RXRDY
                MOVE.B  RXFIFA,     D0        ;READ FIRST CHAR
;
                BSR RXRDY                     WAIT FOR RXRDY
                MOVE.B  RXFIFA,     D1        ;READ SECOND CHAR
;
                BSR RXRDY                     ;WAIT FOR RXRDY
                MOVE.B  RXFIFA,     D2        ;READ THIRD CHAR
;
                BSR RXRDY                     ;WAIT FOR RXRDY
                MOVE.B  RXFIFA,     D3        ;READ FOURTH CHAR
                TRAP    #15                   ;END TEST
;-------------------------------------------------------------------------------
;                                   SUBROUTINES
;
TXRDY:          MOVE.B  GSR,        D6        ;MOVE GSR TO D6
                BTST    #1,         D6        ;TEST GSR1
                BEQ     TXRDY                 ;IF GSR1=0 LOOP
                RTS
;
RXRDY:          MOVE.B  GSR,        D6        ;MOVE GSR TO D6
                BTST    #0,         D6        ;TEST GSR0
                BEQ     RXRDY                 ;IF GSR0=0 LOOP
RTS
;               END
```

Figure 2. Example Using BOP Mode

This file sets up channel A to transmit and receive four BISYNC characters. TXDA is tied to RXDA by a wire.

```
        Begin
        ;
                    MOVE.B  #1,         CMR1A     ;COP BISYNC MODE, EBCDIC
                    MOVE.B  #$3F,       CMR2A     ;POLLED/INT MODE, NORMAL, CCITT PRESET 1'S
                    MOVE.B  #$3F,       TTRA      ;38.4K BAUD
                    MOVE.B  #$6F,       RTRA      ;38.4K, DPLL X32 FROM BRG
                    MOVE.B  #$E3,       TPRA      ;TX=8 BIT/CHAR,UNDERRUN=SYNS, IDLE=SYNS
                    MOVE.B  #$83,       RPRA      ;RX=8 BIT/CHAR, STRIP SYN, NO FCS OR HUNT MODE
                    MOVE.B  #$F7,       OMRA      ;TXRDY=EMPTY, RXRDY=NOT EMPTY, NO RESID CHAR
                    MOVE.B  #$66,       S1RA      ;FIRST SYNC CHARACTER=HEX 66
                    MOVE.B  #$99,       S2RA      ;SECOND SYNC CHARACTER=HEX 99
                    MOVE.B  #0,         CCRA      ;RESET TX
                    MOVE.B  #$40,       CCRA,     ;RESET RX
                    MOVE.B  #$2,        CCRA      ;ENABLE TX
                    MOVE.B  #$42,       CCRA      ;ENABLE RX
                    MOVE.B  #$C3        CCRA      ;SET NRZ MODE FOR DPLL
                    MOVE.B  #$C0        CCRA      ;ENTER SEARCH MODE (DPLL)
        ;
                    BSR TXRDY                     ;WAIT FOR TXRDY
                    MOVE.B  #4,         CCRA      ;TRANSMIT START OF MESSAG
                    MOVE.B  #$0D,       CCRA      ;EXCLUDE FROM CRC
                    MOVE.B  #2,         TXFIFA    ;TRANSMIT STX
                    MOVE.B  #$55,       TXFIFA    ;TRANSMIT TEXT (HEX 55)
                    MOVE.B  #$AA,       TXFIFA    ;TRANSMIT TEXT (HEX AA)
                    MOVE.B  #6,         CCRA      ;TRANSMIT END OF MESSAGE
                    MOVE.B  #3,         TXFIFA    ;TRANSMIT ETX
        ;
                    BSR RXRDY                     WAIT FOR RXRDY
                    MOVE.B  RXFIFA,     D0        ;READ FIRST CHAR
        ;
                    BSR RXRDY                     ;WAIT FOR RXRDY
                    MOVE.B  RXFIFA,     D1        ;READ SECOND CHAR
        ;
                    BSR RXRDY                     ;WAIT FOR RXRDY
                    MOVE.B  RXFIFA,     D2        ;READ THIRD CHAR
        ;
                    BSR     RXRDY                 ;WAIT FOR RXRDY
                    MOVE.B  RXFIFA,     D3        ;READ FOURTH CHAR.
                    TRAP    #15                   ;END TEST
        ; ------------------------------------------------------------------------
        ;                                 SUBROUTINES
        TXRDY:      MOVE.B  GSR,        D6        ;MOVE GSR TO D6
                    BTST    #1,         D6        ;TEST GSR1
                    BEQ     TXRDY                 ;IF GSR1=0 LOOP
                    RTS
        ;
        RXRDY:      MOVE.B  GSR,        D6        ;MOVE GSR TO D6
                    BTST    #0,         D6        ;TEST GSR0 BEQ
                    RXRDY                         ;IF GSR0=0 LOOP
                    RTS
        ;
          END
```

Figure 3. Example Using COP Mode

This example does a DMA transfer of 256 bytes from memory to chan 'A' transmitter. It is set up in normal mode with TXA tied to RXB, and RTSB tied to CTSA by wire. (Receiver 'B' controls RTS, stopping any overruns). Flow control is imposed since the 8MHz CPU is too slow in testing the GSR and reading received characters.

```
SETUP:      MOVE.B  #0,         CCRA      ;RESET REQN(A) BY RESETTING TXA & RXB
            MOVE.B  #$40,       CCRB
            MOVE.B  #0,         PCRA      ;SET UP PIN CONFIGURATION REGISTER
            MOVE.B  #$20,       PCRB      ;RTS ENABLED FOR CHANNEL 'B'
            MOVE.B  #$7,        CMR1A     ;NO PARITY, ASYN MODE
            MOVE.B  #$7,        CMR1B
            MOVE.B  #0,         CMR2A     ;CHAN A = HALF DUPLEX SINGLE ADDRESS DMA
            MOVE.B  #$38,       CMR2B     ;CHAN B = NORMAL, POLLED/INTERRUPT
            MOVE.B  #$77,       TPRA      ;1 STOP BIT, 8-BIT, TX CONTROL BY CTS
            MOVE.B  #$3F,       TTRA      ;TX = BRG CLK, 38.4K BAUD
            MOVE.B  #13,        RPRB      ;RX = 8-BIT CHARACTERS, RXB CONTROLS RTS
            MOVE.B  #2F,        RTRB      ;RX = BRG CLK, 38.4K BAUD
            MOVE.B  #$F0,       OMRA,     ;TXRDY WITH FIFO NOT FULL (CHAN A)
            MOVE.B  #$F1,       OMRB      ;RXRDY WITH FIFO NOT EMPTY (CHAN B), RTS = 1
            MOVE.B  #0,         CCRA      ;RESET TXA & TXB
            MOVE.B  #0,         CCRB
            MOVE.B  #$40,       CCRB      ;RESET RXA & RXB
            MOVE.B  #$40,       CCRA
            MOVE.B  #2,         CCRA      ;ENABLE TXA
            MOVE.B  #$42,       CCRB      ;ENABLE RXB
            CLR.L   D6                    ;CLR D6
            MOVE.L  #GSR,       A0        ;A0=POINTER TO STATUS REG
            MOVE.L  #$77000,    A1        ;A1=MEMORY POINTER
;***************************INITITALIZE MEMORY*************************************************
   AGAIN:   MOVE.W  D6,         [A1]+     ;D6 = DATA VALUE
            ADDI.B  #1,         D6
            CMPI.B  #0,         D6        ;DOES D6=FF ?
            BNE     AGAIN                 ;CONTINUE UNTIL IT DOES
;************************INITIALIZE DMA CONTROLLER TO WRITE TO TRANSMITTER*****************
SETDMA:     MOVE.B  #$B8,       CSR       ;RESET DMAI
            MOVE.B  #$18,       DCR       ;BURST MODE
            MOVE.B  #$12,       OCR       ;MEMORY TO DEVICE TRANSFER, BYTE
            MOVE.B  #1,         MTCH      ;SET COUNTER TO TRANSFER 256 CHAR
            MOVE.B  #0,         MTCL      ;MEMORY ADDRESS = 77000
            MOVE.B  #$7,        MACMH
            MOVE.B  #$70,       MACML
            MOVE.B  #$0,        MACL
            MOVE.W  #$100,      D6
            CLR.L   D1
            MOVE.B  #$B8,       CSR       ;RESET DMAI
;           MOVE.B  #$80,       CCRX      ;DMAI START OPERATION
;*****************************REA/VERIFY DMA PROCESS*****************************************
RXRDYB:     MOVE.B  [A0],       D0        ;CHECK FOR RXB READY
            BTST    #4,         D0
            BEQ     RXRDYB                ;WAIT FOR RX TO COME READY
            MOVE.B  #4,         GSR       ;RESET RXRDY 'B'
            MOVE.B  RXFIFB,     D2
            CMP.B   D1,         D2
            BNE     STOP                  ;IF TX NOT EQUAL TO RX, STOP
            ADDI.B  #1,         D1        ;INC CHARACTER POINTER
            CMP.B   D6,         D1        ;HAVE WE RECEIVED FOUR CHARACTERS?
            BNE     RXRDYB                ;IF NOT, GET THE NEXT RX CHAR
STOP:       TRAP    #15                   ;END TEST
;           END     SETUP
```

Figure 4. DMA Transmit Program

This example does a DMA transfer of 256 bytes from memory to chan 'A' transmitter. It is set up in normal mode with TXB tied to RXA. No flow control is imposed.

```
SETUP:      MOVE.B  #43,          CCRA        ;RESET REQN(A) BY DISABLING RXA
            MOVE.B  #$40,         CCRB        ;DISABLE TXB
            MOVE.B  #7,           CMR1A       ;NO PARITY ASYN MODE
            MOVE.B  #7,           CMR1B
            MOVE.B  #0,           CMR2A       ;CHAN A = HALF DUPLEX SINGLE ADDRESS DMA
            MOVE.B  #$38,         CMR2B       ;CHAN B=NORMAL, POLLED/INTERRUPT
            MOVE.B  #$73,         TPRB        ;1 STOP BIT, 8 BIT CHAR
            MOVE.B  #$3F,         TTRB        ;TX=BRG CLK, 38.4 BAUD
            MOVE.B  #3,           RPRA        ;RX = 8 BIT CHARACTERS
            MOVE.B  #$2F,         RTRA        ;RX = BRG CLK, 38.4K BAUD
            MOVE.B  #$13,         RPRB        ;RXRDY WITH FIFO NOT EMPTY (CHAN A)
            MOVE.B  #$E0,         OMRA        ;RX = BRG CLK, 38.4K BAUD
            MOVE.B  #$E0,         OMRB,       ;TXRDY WITH FIFO NOT FULL (CHAN B)
            MOVE.B  #0,           CCRB        ;RESET TXB
            MOVE.B  #0,           CCRA        ;RESET RXA
            MOVE.B  #2,           CCRB        ;ENABLE TXB
            MOVE.B  #$42,         CCRB        ;ENABLE RXA
            MOVE.B  #GSR,         A0          ;A0 = POINTER TO STATUS REG
            MOVE.B  #RAM,         A1          ;A1 = MEMORY POINTER
;*************************************CLEAR MEMORY*****************************************
AGAIN:      MOVE.W  #0,           [A1]+
            CMPA.L  #RAMEND, A1
            BNE     AGAIN
;**********************INITITALIZE DMA CONTROLLER TO READ RECEIVER*****************************
SETDMA:     MOVE.B  #$B8,         CSR         ;RESET DMAI
            MOVE.B  #$38,         DCR         ;BURST MODE
            CMPI.B  #$92,         OCR         ;DEVICE TO MEMORY TRANSFER, WORD
            MOVE.B  #1,           MTCH        ;SET COUNTER TO TRANSFER 256
CHAR        MOVE.B  #0,           MTCL
            MOVE.B  #7,           MACMH       ;LOCAL RAM ADDRESS = 77000
            MOVE.B  #$70,         MACML
            MOVE.B  #$0,          MACL
            CLR.L   D1
            MOVE.B  #$80,         CCRX        ;DMAI START OPERATION
;*****************************CPU WRITES TO TXB WHILE DMA READS RXA*********************
TXRDYB:     MOVE.B  [A0],         D0
            BTST    #5,           D0          ;CHECK FOR TXB READY
            BEQ     TXRDYB                    ;WAIT FOR TX TO COME READY
            MOVE.B  D1,           TXFIFB
            ADDI.B  #1,           D1          ;INC CHARACTER POINTER
            CMPI.B  #0,           D1          ;STOP AFTER 256 TXB CHAR-
;ACTERS      BNE     TXRDYB                    ;IF NOT, SENT THE NEXT TX CHAR
;*****************************CPU VERIFIES RECEIVED DATA IN MEMORY*****************************************
            CLR.L   D1
            MOVE.L  #RAM,         A1          ;MEM START ADDRESS
VERIFY      MOVE.W  [A1]+,        D2          ;READ NEXT RAM LOCATION
            CMP.B   D1,           D2
            BNE     STOP                      ;STOP IF RX CHAR NOT EQUAL TO TX CHAR
            ADDI.B  #1,           D1
            CMP.L   RAMEND,       A1          ;HAVE WE COMPARED ALL RXA CHAR?
            BNE     VERIFY                    ;IF NOT, KEEP LOOPING
STOP:       TRAP    #15                       ;END TEST
;           END     SETUP
```

Figure 5. DMA Receive Program

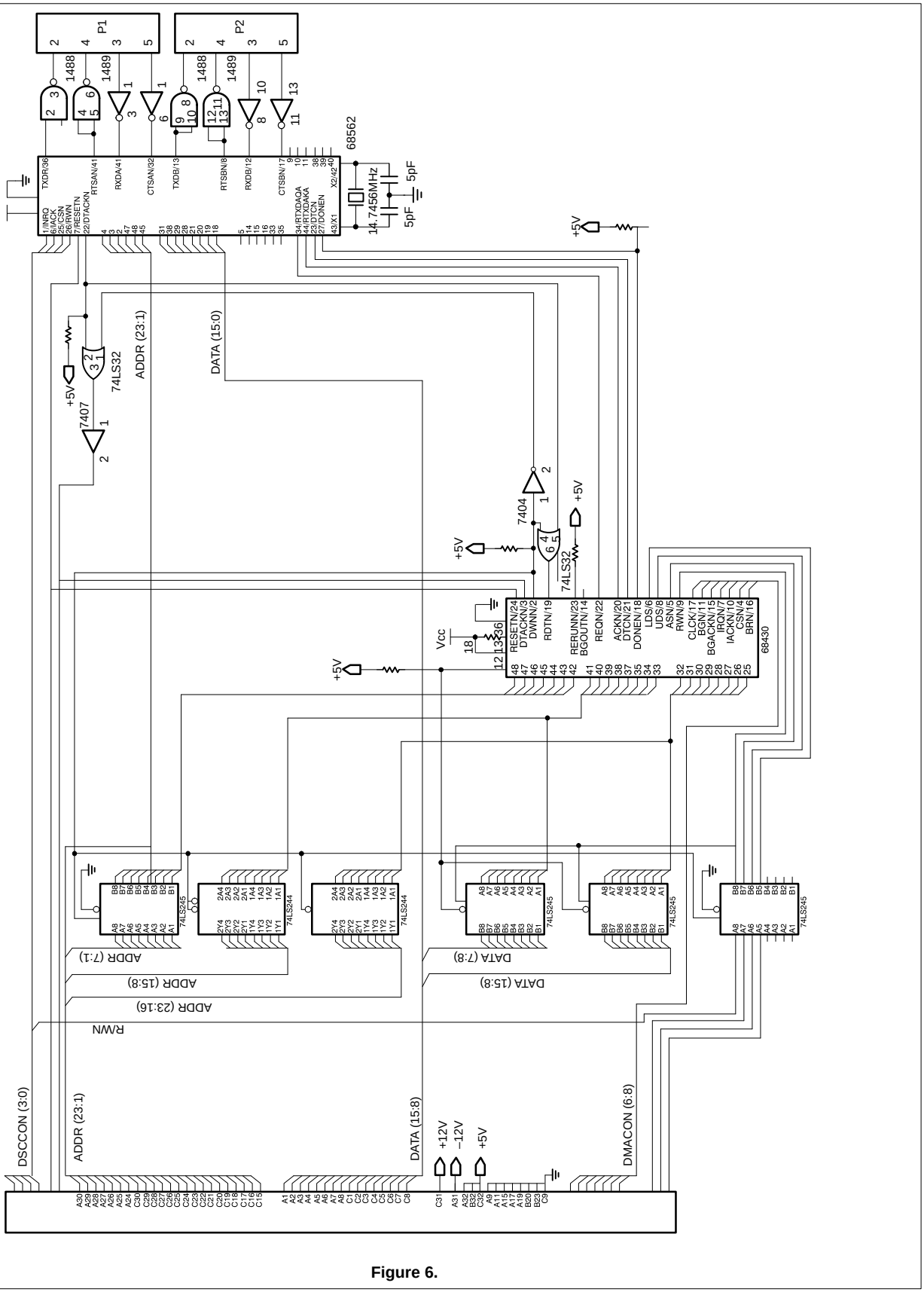
P1
P2
1488
1489
68562
14.7456MHz
5pF
+5V
74LS32
7407
7404
ADDR (23:1)
DATA (15:0)
68430
74LS245
74LS244
ADDR (7:1)
ADDR (15:8)
ADDR (23:16)
DATA (7:8)
DATA (15:8)
R/WN
DSCCON (3:0)
ADDR (23:1)
DATA (15:8)
DMACON (6:8)
+12V
–12V
+5V

Figure 6.