A TERMINAL INTERFACE, PRINTER INTERFACE, AND BACKGROUND PRINTING FOR AN MC68000-BASED SYSTEM USING THE MC68681 DUART

Prepared by
Kyle Harper
Advanced Microcomputer Applications Engineering
Microprocessor Division
Motorola Inc.
Austin, Texas

INTRODUCTION

Very efficient terminal and printer I/O can be achieved in an MC68000-based system using only the MC68681 dual universal receiver transmitter (DUART) and an RS-232 interface driver chip set. As an extra bonus, a dual-tasking scheme can be easily implemented using the counter/timer on-chip the MC68681 to generate periodic time-slice interrupts to the MC68000. This allows the MC68000 to appear to be executing two tasks simultaneously. Typically, one of the tasks would be a printing task so that printing can be done as a “background” task to something else being executed by the MC68000.

In this Application Note, a complete MC68000/MC68681 interface and a dual-task sample application is presented. It begins with a description of the MC68681 operation and programming for this application. This is followed by a description of the MC68000/MC68681 hardware interface. Finally, the software required for the application is presented. It includes the routines required to initialize and drive the MC68681 serial channels and counter, and the software required to implement the dual-tasking scheme. The software also includes two sample task routines. One continually monitors a terminal (attached to DUART channel A) for incoming characters, assembles them into a character string in an input buffer, then places the string in a print queue. The other task continually monitors the print queue for character strings destined to be printed and sends them to the printer (attached to DUART channel B).

MC68681 OPERATION AND PROGRAMMING

The MC68681 DUART is a communications device that provides two independent full-duplex asynchronous receiver/transmitter channels, a 6-bit parallel input port, an 8-bit parallel output port, and a 16-bit counter/timer in a single package. Also, the MC68681 can be programmed to generate interrupts upon any of the following conditions:

Channel A Transmitter Ready
Channel A Receiver Ready
Channel A Change-in-Break
Channel B Transmitter Ready
Channel B Receiver Ready
Channel B Change-in-Break
Counter/Timer Ready
Input Port Change-of-State

Channels A and B of the MC68681 can operate in four different modes: normal, automatic echo, local loopback, and remote loopback. A channel operating in normal mode allows full-duplex communication. A channel operating in automatic-echo mode operates exactly as in normal mode, but automatically re-transmits any received data. Local loopback and remote loopback modes are diagnostic modes that can be used to verify correct operation of a channel.
The MC68681 has a 6-bit parallel input port and an 8-bit parallel output port. Each of the inputs and outputs can be used as general-purpose inputs and outputs. However, each has programmable alternate functions, as shown below:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Programmable Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP0</td>
<td>Channel A Clear-to-Send Input</td>
</tr>
<tr>
<td>IP1</td>
<td>Channel B Clear-to-Send Input</td>
</tr>
<tr>
<td>IP2</td>
<td>Channel B Receiver External Clock Input or Counter/Timer External Clock Input</td>
</tr>
<tr>
<td>IP3</td>
<td>Channel A Transmitter External Clock Input</td>
</tr>
<tr>
<td>IP4</td>
<td>Channel A Receiver External Clock Input</td>
</tr>
<tr>
<td>IP5</td>
<td>Channel B Transmitter External Clock Input</td>
</tr>
<tr>
<td>OP0</td>
<td>Channel A Request-to-Send Output</td>
</tr>
<tr>
<td>OP1</td>
<td>Channel B Request-to-Send Output</td>
</tr>
<tr>
<td>OP2</td>
<td>Channel A Transmitter Clock Output or Channel A Receiver Clock Output</td>
</tr>
<tr>
<td>OP3</td>
<td>Counter/Timer Output or Channel B Transmitter Clock Output or Channel B Receiver Clock Output</td>
</tr>
<tr>
<td>OP4</td>
<td>Channel A Receiver-Ready or Buffer-Full Interrupt Output</td>
</tr>
<tr>
<td>OP5</td>
<td>Channel B Receiver-Ready or Buffer-Full Interrupt Output</td>
</tr>
<tr>
<td>OP6</td>
<td>Channel A Transmitter-Ready Interrupt Output</td>
</tr>
<tr>
<td>OP7</td>
<td>Channel B Transmitter-Ready Interrupt Output</td>
</tr>
</tbody>
</table>

Finally, the MC68681 has a 16-bit programmable counter/timer that can be used to measure elapsed time between events, or to generate periodic interrupts. It can be programmed to operate as a free-running timer (cannot be stopped and started) or as a counter (can be stopped and started).

This application will use the normal, automatic-echo, and local loopback modes, and will utilize two of the MC68681 interrupt sources: the channel A change-in-break IRQ and the counter/timer IRQ. Also, one of the output port pins and one of the input port pins will be used as RTS/CTS handshake lines. In this application, a terminal will be attached to DUART channel A and will be programmed to transmit and receive at 9600 baud with seven bits/character, even parity, and two stop bits. The channel will be programmed to operate in automatic-echo mode so that the character typed at the terminal keyboard will appear on the CRT screen. So that the channel receiver FIFO is not overrun, channel A will be programmed to use the receiver RTS/CTS handshake protocol. This protocol works as follows: the receiver RTS output is connected to the CTS input of the terminal. So long as the receiver has room in its FIFO for another character, the receiver will assert RTS. If the FIFO becomes full, the receiver will negate RTS. When the FIFO once again has room for another character, it will automatically re-assert RTS. Assuming that the terminal will not transmit a character unless it sees CTS asserted, receiver overrun will not occur. Finally, the BREAK key will be used as an abort button, so that the user can exit to the monitor (or operating system) at any time. Channel A will, therefore, be programmed to generate an interrupt to the MC68000 when it receives a BREAK character from the terminal.

A printer will be attached to DUART channel B and the channel will be programmed to operate in normal mode, transmit at 300 baud with seven bits/character, even parity, and one stop bit. So that the channel does not send characters to the printer faster than the printer can handle them, channel B will be programmed to use the transmitter RTS/CTS handshake protocol. This protocol works as follows: when channel B needs to send a character to the printer, it will assert RTS and then wait for the printer to assert CTS before transmitting the character.

The MC68681 counter/timer will be programmed to generate the time-slice interrupts to the MC68000 required for dual-tasking. The counter/timer must be able to be stopped and re-started; therefore, it is programmed to operate in counter mode. After initializing the counter registers with the count value, the counter will be started. When the counter reaches terminal count, it will generate an interrupt to the MC68000. The MC68000 will then stop the counter, clear the interrupt, swap tasks being executed, and start the counter again. When the counter is started again, it will be re-initialized using the value found in the counter registers.

**INTERFACE HARDWARE**

The hardware required to interface the MC68681 to the MC68000 is minimal, as shown by the schematic in Figure 1. The RESET, R/W, and DTACK lines are connected directly between and MC68681 and the MC68000. Address lines A5-A23 are routed through address decode logic and used to generate the MC68681 chip select. Address lines A1-A4 are tied to the MC68681 register select pins RS1-RS4. The MC68681 data bus pins, D0-D7 are connected to the MC68000 lower data bus lines, D0-D7. Typically, the MC68681 would be attached to the lower data bus because the MC68681 must supply an interrupt vector number to the MC68000 on D0-D7 during IACK cycles. However, if the MC68681 will not be generating interrupts, it could just as easily be attached to the upper data bus. The MC68681 IRQ line must be encoded by the SN74LS148 to give the IRQ a priority level required by the MC68000 on its IPL0-IPL2 lines. Also, the MC68000 A1-A3 lines must be decoded during IACK cycles by the SN74LS138 to generate IACK back to the MC68681. Using the SN74LS148 as the IRQ encoder and the SN74LS138 as the IACK decoder provides full support of the MC68000 seven interrupt levels. The MC68681 requires only one interrupt level. For this application, interrupt level four has been arbitrarily chosen. This leaves the other six levels for future system expansion.

The two channels are connected to the external devices via RS-232 drivers and DB-25 connectors. Because this application uses the OP0 and OP1 lines as the RTSA and CTSA handshake lines, respectively, they too are routed via the RS-232 drivers to their respective connectors.

Finally, a 3.6864 MHz crystal is connected between the MC68681 X1/CLK and X2 pins. The crystal is required for the built-in baud rate generator. The 15 pF and 5 pF shunt capacitors must also be connected between the crystal and ground as shown to insure proper operation of the baud rate generator.

**INTERFACE SOFTWARE**

The interface software required for this application is flowcharted in Figure 2 and is listed at the end of this Application Note. The routines can be broken down into three categories: the DUART initialization routines, the I/O driver routines, and the interrupt handling routines. The DUART initialization routines consist of DINIT, CHCKH, and CTRCHK. DINIT is the DUART initialization routine, and is called at system initialization time. After DINIT initializes the DUART channels and counter, it checks channel A, channel B, and the counter for operational errors. Before
DINIT is called, the calling routine must allocate three words on the system stack. Upon return to the calling routine, DINIT will pass back three status words on the system stack that reflect the operation of channel A, channel B, and the counter. If DINIT finds no errors in channel A, it will enable the channel A receiver and transmitter. Likewise, if DINIT finds no errors in channel B, it will enable the channel B transmitter. CHCHK and CTRCHK are routines that are called by DINIT to perform the actual checks. CHCHK checks a channel for proper operation. DINIT calls CHCHK twice: the first time to check channel A and the second time to check channel B. After placing the channel in local loop-back mode, CHCHK checks the channel for the following errors: transmitter never ready, receiver never ready, framing error, parity error, and incorrect character received. CTRCHK checks the counter for proper operation by verifying that the counter interrupts the MC68000 properly after reaching terminal count.

The I/O driver routines consist of INCH, OUTCH, and POUTCH. INCH is the terminal input character routine. INCH gets a character from the channel A receiver and places it in the lower byte of register D0. OUTCH is the terminal output character routine. OUTCH sends the character in the lower byte of register D0 to the channel A transmitter. POUTCH is the printer output character routine. POUTCH sends the character in the lower byte of register D0 to the channel B transmitter.

The interrupt handling routines consist of DIRQ and CIRQ. DIRQ is the DUART interrupt handling routine. After the DUART generates an interrupt, the MC68000 begins executing DIRQ. DIRQ determines whether the interrupt was caused by the counter or a channel A change-in-break. If the interrupt was caused by the counter, DIRQ causes the MC68000 to swap tasks being executed. This process is discussed in a later section. If the interrupt was caused by a channel A change-in-break interrupt (beginning of break), DIRQ clears the interrupt source, waits for the next change-in-break condition interrupt (end of break), clears the interrupt source again and then returns from exception processing to the system monitor. CIRQ is used instead of DIRQ as the DUART interrupt handling routine when CTRCHK is executing. When the counter generates an interrupt during execution of CTRCHK, CIRQ sets the carry bit in the status register, thus informing CTRCHK that the counter interrupt was generated correctly.

DUAL-TASKING SOFTWARE

The dual-tasking software required for this application is flowcharted in Figure 3 and is listed at the end of this Application Note. The routines can be broken down in two categories: the routines that facilitate dual-tasking and the two sample tasks themselves. The routines that facilitate dual-tasking consist of SWPTSKS and TSKINIT.

SWPTSKS is the task swapping routine executed when DIRQ determines that the counter generated an interrupt. SWPTSKS “swaps out” the task currently being executed with the task that is currently dormant. The “swap” process works as follows: the counter interrupt causes the MC68000 to begin exception processing. During exception processing the MC68000 stacks the active task program counter and status register on the active task system stack, then executes DIRQ. DIRQ determines that the interrupt was caused by the counter and branches to SWPTSKS. SWPTSKS stops the counter, then saves the active task register contents and user stack pointer on the active task system stack. After saving this information on the active task system stack, SWPTSKS swaps out the active task system stack pointer with the dormant task system stack pointer (stored in a reserved memory location). SWPTSKS then pulls the dormant task user stack pointer and register contents off the dormant task system stack (this information was placed on the dormant system stack by a previous task swap operation), and restarts the counter. Finally, because the dormant task status register contents and program counter are now at the top of the dormant task system stack, the MC68000 will return from exception where the dormant task had been interrupted, thereby re-activating it.

TSKINIT is the task initialization routine. It initializes the DUART by calling DINIT, then checks for operational errors in the two channels and the counter. If errors are found in either of the channels or the counter, TSKINIT prints the appropriate error messages to a “command console” then stops. If no errors are found, TSKINIT then initializes the print task as the initial dormant task. The initialization procedure works like this: the dormant task system stack pointer is initialized. The start address of the print task is stacked on the system stack, then an initial status register content is stacked. This is the order in which the MC68000 requires information to be stacked when returning from exception. Next, the print task initial register contents and user stack pointer are stacked on the system stack. This is the order in which SWPTSKS requires information to be stacked to perform its task swap operation. After initializing the print task as the dormant task, TSKINIT initializes the input task user and system stack pointers, starts the counter, then begins execution of the input task.

The two sample tasks given in this Application Note are INPTTSK and PRNTTSK. The tasks work together to perform two typical I/O operations: character string input from a terminal and character string output to a printer. Because I/O hardware is character-oriented and not string-oriented, character string I/O must be transformed into character I/O by using buffers and queues. Character string input is accomplished through the use of an input buffer. Characters are placed in this buffer as they come in from the terminal. When the carriage return character is received and placed in the buffer, the string has been completely assembled and is moved elsewhere so that another one can be assembled.

Character string printing is accomplished through the use of a print buffer and a print queue. For efficient character string printing, the print buffer should be capable of holding more than one character string. This is because the MC68000 can supply strings to be printed much faster than the printer can print them. A multiple-string print buffer allows the MC68000 to “queue” character strings bound for the printer, then go on to more important things, rather than acting as a slave to the printer. The print queue is required to determine where the next string arriving at the buffer will go and where the next string departing from the buffer can be found. Print “tags” indicating that there are character strings in the print buffer are placed in this queue. The queue has an input and output pointer, and acts in a first-in-first-out manner. Thus, strings in the print buffer will be sent to the printer in the order that their print tags arrived at the print queue.

For this application, a character string is terminated by a carriage return, and maximum string length is set by the constant CSLNTH. CSLNTH is used to define the width of the input buffer and the width of the print buffer. The print queue length is set by the constant PQLNTH. PQLNTH is
used to define the length of the print queue and the length of the print buffer. Both CSLNTH and PQLNTH must be assigned values that are powers of two and can have a maximum value of 256. Because maximum string length is 256 bytes, the print tags need only be a byte value.

When a character string is to be sent to the print buffer, it must be moved into the print buffer and an associated print tag placed in the print queue. When a character string is to be sent to the printer, it must be taken from the print buffer and its associated print tag removed from the print queue.

INPTTSK continually monitors the terminal attached to DUART channel A for incoming characters, assembles them into a character string in the input buffer, then queues the string in the print buffer. INPTTSK consists of two routines: ISTRG and QSTRG. ISTRG is the routine that assembles characters received from the terminal (via the INCH routine) into a character string in the input buffer. QSTRG is the routine that queues the character string in the print buffer. QSTRG first checks the status of the print queue. If the queue is full, QSTRG will wait until there is room in the queue for a print tag. If the queue is not full, QSTRG will move the character string into the print buffer and place a print tag in the print queue.

PRNTTSK continually monitors the print queue for print tags. If it finds a print tag in the queue, PRNTTSK prints the string and removes the tag from the queue. PRNTTSK consists of two routines: RSTRG and PSTRG. RSTRG is the routine that releases a character string from the print buffer, and sends it to the printer via the PSTRG routine. RSTRG checks the status of the print queue. If it is empty, RSTRG will wait until a print tag appears in the queue. If the queue is not empty, RSTRG will call routine PSTRG, then remove the print tag from the print queue. PSTRG is the routine that sends a character string to the printer character-by-character (via the POUTCH routine).

**SUMMARY**

The frequency at which the MC68000 swaps between tasks is directly determined by the frequency at which the DUART counter generates interrupts. This is determined by the count value placed in the upper and lower counter registers. The main concern in determining the count value is making sure that the task-swapping is transparent to the user sitting at the terminal. That is, he must not be aware that he does not have the attention of the system all the time.

The system on which this application was developed performed well with the count value set at $0073. With the counter clock source programmed to be the 3.6864 MHz crystal divided-by-sixteen, this count value causes an interrupt to occur approximately every 500 microseconds.

Also, this Application Note presents the interface required for efficient poll-driven serial I/O using the MC68681 DUART. If you wish to modify this interface to support interrupt-driven I/O, no changes in the hardware are required. Only software modifications need to be made.
FIGURE 2 — MC68681 Interface Software Flowcharts (Sheet 1 of 6)
CHCHK

Place Channel in Local Loopback Mode

Enable Channel's TX. Clear Channel Status Word

TXCHK

Is Transmitter Ready?

N  Waited Too Long?

N  Y  Set TX-Never-Ready Flag

SNDCHR  Y  Send Character to Transmitter

RXCHK

Has Receiver Got the Char.?

N  Waited Too Long?

N  Y  Set RX-Never-Ready Flag

A  Y  B

FIGURE 2 — MC68681 Interface Software Flowcharts (Sheet 2 of 6)
Disable Channel's Transmitter

Restore Channel to Original Mode

Return

Set Framing Error Flag

Set Parity Error Flag

Get Character from Receiver

Same as Character Transmitted?

Set Incorrect Character Flag

Set Framing Error Flag

Set Parity Error Flag

Have Framing Error?

Have Parity Error?

N

Y

N

Y

FIGURE 2 — MC68881 Interface Software Flowcharts (Sheet 3 of 6)
FIGURE 2 — MC68681 Interface Software Flowcharts (Sheet 4 of 6)
ABRK1

Was IRQ Caused by Counter?

Y

Go to SWPTSKS

N

ABRK1

Was IRQ Caused by the Beginning of a BREAK?

Y

Clear Change in BREAK Status Bit

N

ABRK1

Has the End-of-BREAK IRQ Arrived Yet?

Y

Clear Change in BREAK Status Bit

N

Remove BREAK Character from Receiver FIFO

Replace Return Address on System Stack with Monitor Warm Start Address

DIRQ

CIRQ

Was IRQ Caused by Counter?

Y

Stop Counter

N

Set Carry Flag

CIRQR

RTE

INCH

INCH

Does Chan A RX Have a Char.?

Y

Place Character in DO

N

Return

FIGURE 2 — MC68681 Interface Software Flowcharts (Sheet 5 of 6)
OUTCH

Send Character in D0 to Chan A TX

Was Character a Carriage Return?

N

Y

OUTCHI

Is Chan A TX Ready?

N

Y

Send A Line Feed Character to Chan A TX

OUTCHR

Return

POUTC

Send Character in D0 to Chan B TX

Was Character a Carriage Return?

N

Y

POUTCII

Is Chan B TX Ready?

N

Y

Send A Line Feed Character to Chan B TX

POUTCRI

Return

FIGURE 2 — MC68681 Interface Software Flowcharts (Sheet 6 of 6)
FIGURE 3 — Dual-Tasking Software Flowchart (Sheet 1 of 5)
FIGURE 3 — Dual-Tasking Software Flowcharts (Sheet 2 of 5)
Initialize Input Buffer Pointer

GETCHR

Call INCH

Is Character a Backspace ?

Y

N

Place Character in Input Buffer

Bump Input Buffer Pointer

Was Character a Carriage Return ?

Y

N

Return

Decrement Buffer Pointer

Is Pointer at Beginning of Buffer ?
SWPTS

SWPTS

Stop Counter

Save Active Task Register Contents on System Stack

Save Active Task USP on System Stack

Swap Out Active Task SSP With Dormant Task SSP

Restore Dormant Task USP

Restore Dormant Task Register Contents

Start Counter

RTE

TSKINIT

TSKINIT

Call DINIT

CTRERRS

Any Counter Errors?

Print Counter Error Message(s)

Y

N

CHBERRS

Any Channel B Errors?

Print Channel B Error Message(s)

Y

N

CHAERRS

Any Channel A Errors?

Print Channel A Error Message(s)

Y

N

Were There Any Errors?

Stop

Y

N

C

FIGURE 3 — Dual-Tasking Software Flowcharts (Sheet 4 of 5)
FIGURE 3 — Dual-Tasking Software Flowcharts (Sheet 5 of 5)
**DUART685**

OCT  FRDIS,FRS,FRS

**DUART685** - ROUTINES REQUIRED FOR A 68000-BASED SYSTEM TO PERFORM

**ASYNCHRONOUS SERIAL I/O & DUAL TASK EXECUTION USING**

**A 68631 DUART.**

THE FOLLOWING ROUTINES ALLOW A 68000'S EXECUTION TIME TO

BE SPLIT BETWEEN TWO TASKS:

- **TSKINIT** - ROUTINE TO INITIALIZE THE TWO TASKS
- **INPTSK** - SAMPLE INPUT TASK
- **PRUTSK** - SAMPLE PRINT TASK
- **SWPTSK** - ROUTINE TO SWAP BETWEEN TASKS

**INPTSK** CONTINUALLY MONITORS A TERMINAL CONNECTED

TO THE DUART'S CHANNEL A FOR INCOMING CHARACTER STRINGS.

PRINTSK SENDS CHARACTER STRINGS TO A PRINTER CONNECTED

TO THE DUART'S CHANNEL B.

**THE TIME-SLICING INTERRUPT THAT FACILITATES THE DUAL-TASKING**

**IS GENERATED BY THE DUART'S COUNTER.**

**THE FOLLOWING ROUTINES PERFORM THE I/O OPERATIONS:**

- **PSRGR** - SUBROUTINE TO PLACE A CHAR STRING IN PRINT QUEUE
- **PSTRG** - SUBROUTINE TO REMOVE A CHAR STRING FROM PRINT QUEUE
- **STRG** - SUBROUTINE TO GET A CHAR STRING FROM TERMINAL
- **PSPTO** - SUBROUTINE TO SEND A CHAR STRING TO PRINTER
- **DINIT** - SUBROUTINE TO INIT CHECK & ENABLE DUART
- **CMCK** - SUBROUTINE TO CHECK CHANNEL OPERATION
- **CTCK** - SUBROUTINE TO CHECK COUNTER OPERATION
- **CRQ** - DUART INTERRUPT HANDLER USED DURING CTCK
- **INCH** - SUBROUTINE TO INPUT CHARACTER FROM TERMINAL
- **OUTQ** - SUBROUTINE TO OUTPUT CHARACTER TO TERMINAL
- **POUTQ** - SUBROUTINE TO OUTPUT CHARACTER TO PRINTER
- **VIRQ** - DUART INTERRUPT HANDLER

**AUTHOR** - KYLE HARPER

**DATE** - APRIL 9, 1984

**VERSION** - 4

**SYSTEM ADDRESSES**

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DUART</th>
<th>EQU</th>
<th>$000001</th>
</tr>
</thead>
<tbody>
<tr>
<td>00F0001</td>
<td>CHANA</td>
<td>EQU</td>
<td>DUART+0</td>
</tr>
<tr>
<td>00F0001</td>
<td>MR1A</td>
<td>EQU</td>
<td>DUART+0</td>
</tr>
<tr>
<td>00F0001</td>
<td>MR2A</td>
<td>EQU</td>
<td>DUART+0</td>
</tr>
<tr>
<td>00F0001</td>
<td>SRA</td>
<td>EQU</td>
<td>DUART+0</td>
</tr>
<tr>
<td>00F0001</td>
<td>CSRA</td>
<td>EQU</td>
<td>DUART+2</td>
</tr>
<tr>
<td>00F0002</td>
<td>CRA</td>
<td>EQU</td>
<td>DUART+4</td>
</tr>
<tr>
<td>00F0003</td>
<td>RBA</td>
<td>EQU</td>
<td>DUART+6</td>
</tr>
<tr>
<td>00F0004</td>
<td>TBA</td>
<td>EQU</td>
<td>DUART+6</td>
</tr>
<tr>
<td>00F0005</td>
<td>IPCR</td>
<td>EQU</td>
<td>DUART+8</td>
</tr>
<tr>
<td>00F0006</td>
<td>ACR</td>
<td>EQU</td>
<td>DUART+8</td>
</tr>
</tbody>
</table>

**BASE ADDRESS OF 68631 DUART**

**CHANNEL A BASE ADDRESS**

**MODE REGISTER 1A**

**MODE REGISTER 2A**

**STATUS REGISTER A**

**CLOCK-SELECT REGISTER A**

**COMMAND REGISTER A**

**RECEIVER BUFFER A**

**TRANSMITTER BUFFER A**

**INPUT PORT CHANGE REGISTER**

** AUXILIARY CONTROL REGISTER**
MOTOROLA M68000 ASSEMBLY LANGUAGE

**Source Code**

```
00:000000  ISR  EQU  DUART+10  INTERRUPT STATUS REGISTER
01:000000  IMR  EQU  DUART+10  INTERRUPT MASK REGISTER
02:000000  CM5B  EQU  DUART+12  CURRENT COUNTER/TIMER MOST SIGNIFICANT BYTE
03:000000  CTRU  EQU  DUART+12  COUNTER/TIMER UPPER REGISTER
04:000000  CMTL  EQU  DUART+14  CURRENT COUNTER/TIMER LEAST SIGNIFICANT BYTE
05:000000  CTRL  EQU  DUART+14  COUNTER/TIMER LOWER REGISTER
06:000000  CHAB  EQU  DUART+16  CHANNEL B BASE ADDRESS
07:000000  KTB  EQU  DUART+16  MODE REGISTER 1B
08:000000  KTB  EQU  DUART+16  MODE REGISTER 1B
09:000000  KTB  EQU  DUART+16  MODE REGISTER 2B
10:000000  KTB  EQU  DUART+16  MODE REGISTER 2B
11:000000  KTB  EQU  DUART+18  STATUS REGISTER B
12:000000  KTB  EQU  DUART+16  CLOCK SELECT REGISTER A
13:000000  KTB  EQU  DUART+16  COMMAND REGISTER A
14:000000  RBB  EQU  DUART+22  RECEIVER BUFFER B
15:000000  TBB  EQU  DUART+22  TRANSMITTER BUFFER A
16:000000  IP  EQU  DUART+24  INTERRUPT VECTOR REGISTER
17:000000  XCPC  EQU  DUART+26  INPUT PORT (UNLATCHED)
18:000000  STC  EQU  DUART+28  OUTPUT PORT CONFIGURATION REGISTER
19:000000  BTST  EQU  DUART+28  START COUNTER COMMAND
20:000000  BTST  EQU  DUART+28  OUTPUT PORT REGISTER BIT SET COMMAND
21:000000  BTST  EQU  DUART+30  STOP-COUNTER COMMAND
22:000000  BTST  EQU  DUART+30  OUTPUT PORT REGISTER BIT CLEAR COMMAND
23:000000  50  EQU  $003800  INPUT TASK'S USER STACK AREA
24:000000  50  EQU  $004000  INPUT TASK'S SYSTEM STACK AREA
25:000000  50  EQU  $004000  INPUT TASK'S SYSTEM STACK AREA
26:000000  50  EQU  $005000  PRINT TASK'S USER STACK AREA
27:000000  50  EQU  $005000  PRINT TASK'S SYSTEM STACK AREA
28:000000  50  EQU  $000000  MONITOR TASK'S START ADDRESS
29:000000  50  EQU  $000000  MONITOR TASK'S START ADDRESS
30:000000  50  EQU  $000000  MONITOR TASK'S START ADDRESS
31:000000  50  EQU  $000000  MONITOR TASK'S START ADDRESS
```

**Documentation**

```
* CONSTANTS

* CSNLTH EQU 128 CHARACTER STRING LENGTH IN BYTES (MAX=256)
* PLCNLTH EQU 256 PRINT QUEUE LENGTH IN BYTES (MAX=256)
* CSLMK EQU CSNLTH+1 CHARACTER STRING LENGTH MASK
* PCLMK EQU PLCNLTH+1 PRINT QUEUE LENGTH MASK
* TXCNT EQU $FFFF TX WAIT LOOP COUNT (MAX=$FFFF)
* RXCNT EQU $FFFF RX WAIT LOOP COUNT (MAX=$FFFF)
* IRCNT EQU $FFFF IRQ WAIT LOOP COUNT (MAX=$FFFF)
* IQMSK EQU $CC IQ MASK: ALLOWS CHANNEL A BREAK, & COUNTER IRQ
* CR EQU $00 ASCII CARRIAGE RETURN
* LF EQU $0A ASCII LINE FEED
* SF EQU $08 ASCII BACKSPACE
* ORG EQU $0C2000
```

**Notes**

- TSKINIT - ROUTINE TO INITIALIZE THE TWO TASKS TO BE EXECUTED BY THE M68000.
- TSKINIT INITIALIZES & CHECKS THE DUART CHANNELS & COUNTER, ENABLES THE CHANNELS, INITIALIZES THE PRINT TASK AS THE DORMANT TASK.

STARTS THE COUNTER, THEN BEGINS EXECUTION OF THE INPUT TASK.

ALLOCATE STACK SPACE FOR STATUS WORDS

INITIALIZE & CHECK UART

PULL STATUS WORDS OFF STACK

COUNTER ERROR(S)?

YES, PRINT COUNTER ERROR MESSAGE

IS IT TX NEVER READY?

NO, SKIP NEXT PART

YES, PRINT TX-NEVER-READY MESSAGE

IS IT RX NEVER READY?

NO, SKIP NEXT PART

YES, PRINT RX-NEVER-READY MESSAGE

IS IT A FRAMING ERROR?

NO, SKIP NEXT PART

YES, PRINT FRAMING-ERROR MESSAGE

IS IT A PARITY ERROR?

NO, SKIP NEXT PART

YES, PRINT PARITY-ERROR MESSAGE

IS IT A BAD CHARACTER?

NO, SKIP NEXT PART

YES, PRINT BAD-CHARACTER MESSAGE

CHANNEL B ERROR(S)?

NO, SKIP NEXT PART
MOTOROLA M68000 ASM VERSION 1.30SYS : 5.APPOINT .DU6T68S.SA 04/12/34 15:14:00

DU6T68S

176 00020BE 48F8256B LEA CHAMS2A5 YES, PRINT RX-NEVER-READY MESSAGE
177 00020092 4DE0029 LEA LCHAMS2(A5),A6
178 0002096 6138 BSR PRTMSG
179
180 0002098 0B2002BT CST #2/2 IS IT A FRAMING ERROR?
181 000209C 670A BEQ CHAMS5A5 NO, SKIP NEXT PART
182 0002098 4F8F259CH CHAMS5A5 YES, PRINT FRAMING-ERROR MESSAGE
183 0002096 4A00010 LEA LCHAMS2(A5),A6
184 00020A6 6138 BSR PRTMSG
185
186 00020A8 0B200003 CST #3/2 IS IT A PARITY ERROR?
187 00020A4 670A LEA CHAMS5A5 NO, SKIP NEXT PART
188 00020AE 4F8F25B0 LEA LCHAMS4A5, AYES, PRINT PARITY-ERROR MESSAGE
189 00020B2 4DE0001C BEQ CHAMS4A5
190 00020B6 6118 BSR PRTMSG
191
192 00020BA 0B20004 CST #4/2 IS IT A BAD CHARACTER?
193 00020B8 675C BEQ INPTSK NO, SKIP NEXT PART
194 00020B8 4F8F25CC LEA LCHAMS5A5, AYES, PRINT BAD-CHARACTER MESSAGE
195 00020BC 4DE0002C LEA LCHAMS4A5, A
196 00020C6 6108 BSR PRTMSG
197
198 00020CE 8041 ERECH OR.W 01/00 WERE THERE ANY ERRORS?
199 00020CA 8042 OR.W 02/00
200 00020CC 670A LEA IINITTSK1 NO, CONTINUE WITH DEMO
201 00020C8 60FE BEQ * YES, STOP.
202
203 0002008 103C20F3 PRTMSG MOVE.B #243,07 PRINT MESSAGE TO SCREEN
204 0002004 4E5E TRAP #14
205 0002006 4E75 RTS
206
207 *
208 * INITIALIZE PRINT TASK (PRNTSSK) AS DORMANT TASK/ INITIALIZE
209 *
210 *
211 *
212 *
213 *
214
215 00020DE 2E7C00050D0 INITSK1 MOVE.L #PSSPA7 INIT PRINT TASK'S SYSTEM STACK POINTER
216 00020DF 2F3C0002122 MOVE.L #PRNTTSK-(A7) INIT PRINT TASK'S PROGRAM COUNTER
217 00020DE 3F3C2300 MOVE.W #63200-(A7) INIT PRINT TASK'S STACK REGISTER:IPJT4-7
218 00020E9 700E MOVEQ.L #14,00 INIT PRINT TASK'S REGISTERS
219 00020EA 42A7 INITSK2 CLR.L -A7 INIT TASK'S USER STACK POINTER
220 00020EC 51CBFFFC DBRA D0.INITSK2 SAVE PRINT TASK'S SYSTEM STACK POINTER
221 00020F0 2F3C0004800 MOVE.L #PUSPA7 SAVE PRINT TASK'S SYSTEM STACK POINTER
222 00020F6 21CF7000 MOVE.L A7,DTSTKSSP
223 00020F4 42387084 CBR.B PGIN INIT PRINT QUEUE INPUT POINTER
225 00020F4 42387085 CBR.B POUT INIT PRINT QUEUE OUTPUT POINTER
226
227 0002102 2E7C0003800 MOVE.L #ISP,A7 INIT TASK'S USER STACK POINTER
228 0002108 4E67 MOVE.L #47,USP INIT TASK'S USER STACK POINTER
229 000210A 2E7C0004000 MOVE.L #ISP,A7 INIT TASK'S SYSTEM STACK POINTER
230 0002110 46FC2300 MOVE.W #63200-SR
231 0002114 4A39000010 TST.B STRC START COUNTER
* INPTTSK - TASK THAT CONTINUALLY CHECKS TERMINAL FOR INCOMING CHARACTER
* STRINGS. WHEN THE COMPLETE CHARACTER STRING HAS BEEN RECEIVED,
* INPTTSK SUBMITS THE STRING TO THE PRINT QUEUE.

240 0000211A 610000B6 INPTTSK BSR.L ISTPG INPUT STRING FROM CHANNEL A
241 0000211E 612E BSR QSTPG SUBMIT STRING TO PRINT QUEUE
242 00002120 62F8 BRA INPTTSK

* PRNTTSK - TASK THAT CONTINUALLY CHECKS PRINTER QUEUE FOR STRINGS TO BE
* PRINTED. WHEN A STRING IS TO BE PRINTED, PRNTTSK WILL SEND THE
* STRING FROM THE PRINT BUFFER TO THE PRINTER, IF NO STRINGS NEED
* TO BE PRINTED, PRNTTSK WILL CONTINUE CHECKING QUEUE FOR STRINGS
* TO BE PRINTED.

252 00002122 6172 PRNTTSK BSR RSTPG RELEASE STRING FROM PRINT QUEUE
253 00002124 63FC BRA PRNTTSK CHECK QUEUE FOR ANOTHER PRINT TAG

* SWPTS KS - ROUTINE TO SWAP TASKS BEING EXECUTED BY THE 68000.
* SWPTS KS SWAPS BETWEEN TWO TASKS BY EXCHANGING THE
* SYSTEM STACK POINTER, REGISTER CONTENTS, USER STACK POINTER,
* STATUS REGISTER, & PROGRAM COUNTER OF ONE TASK TO THAT OF THE OTHER

259 ENTRY CONDITIONS:
* DRMNT TASK'S SSP IN DTSPSSP.
* ACTIVE TASK'S SSP IN A7.
* SSP+0 - ACTIVE TASK'S STATUS REGISTER CONTENTS.
* SSP+2 - ACTIVE TASK'S PROGRAM COUNTER CONTENTS.

260 EXIT CONDITIONS:
* NEW DRMNT TASK'S SSP IN DTSPSSP.
* NEW ACTIVE TASK'S SSP IN A7.
* SSP+0 - NEW ACTIVE TASK'S STATUS REGISTER CONTENTS
* SSP+2 - NEW ACTIVE TASK'S PROGRAM COUNTER CONTENTS

265 00002126 4A3900000F SWPTS KS TST.B STPC STOP COUNTER
266 0000212C 487FFF MOVEM.L A0-A6/00-D7-A7 SAVE ACTIVE TASK'S REGISTER CONTENTS
267 00002130 466E MOVE.L USP-A6 SAVE ACTIVE TASK'S USER STACK POINTER
268 00002132 2F0E MOVE.L A6+P-A7

269 00002134 4007 LEAL.L (A7)+A6 SAVE TEMP COPY OF ACTIVE TASK'S SSP
270 00002136 2E780000 MOVE.L DTSPSSP,A7 GET DRMNT TASK'S SYSTEM STACK POINTER
271 0000213A 21CE0000 MOVE.L A6+DTSPSSP SAVE ACTIVE TASK'S SYSTEM STACK POINTER

272 0000213E 2C5F MOVE.L (A7)+A6 GET DRMNT TASK'S USER STACK POINTER
273 00002140 466E MOVE.L A6+USP
274 00002142 40F7FFF MOVE.L (A7)+00-D7/A0-A6 GET DRMNT TASK'S REGISTER CONTENTS

275 00002146 4A39000010 TST.B STKC START COUNTER
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>292</td>
<td>0000214C 4E73</td>
<td>RTE RETURN FROM EXCEPTION TO NEW ACTIVE TASK</td>
</tr>
<tr>
<td>293</td>
<td></td>
<td></td>
</tr>
<tr>
<td>294</td>
<td></td>
<td></td>
</tr>
<tr>
<td>295</td>
<td>00002152 4242</td>
<td>QSTRG SUBROUTINE TO SUBMIT A CHARACTER STRING TO PRINT QUEUE</td>
</tr>
<tr>
<td>296</td>
<td></td>
<td>QSTRG CHECKS THE STATUS OF THE PRINT QUEUE. IF IT IS</td>
</tr>
<tr>
<td>297</td>
<td></td>
<td>FULL, QSTRG WILL WAIT UNTIL THERE IS ROOM IN THE QUEUE FOR</td>
</tr>
<tr>
<td>298</td>
<td></td>
<td>A TAG. IF THE QUEUE IS NOT FULL, QSTRG WILL MOVE THE CHARACTER</td>
</tr>
<tr>
<td>299</td>
<td></td>
<td>STRING INTO THE PRINT BUFFER &amp; PLACE A PRINT TAG IN THE PRINT</td>
</tr>
<tr>
<td>300</td>
<td></td>
<td>QUEUE.</td>
</tr>
<tr>
<td>301</td>
<td></td>
<td>A PRINT TAG IS A BYTE CONTAINING THE LENGTH OF THE STRING TO BE</td>
</tr>
<tr>
<td>302</td>
<td></td>
<td>PRINTED.</td>
</tr>
<tr>
<td>303</td>
<td></td>
<td>ENTRY CONDITIONS:</td>
</tr>
<tr>
<td>304</td>
<td></td>
<td>AO CONTAINS STRING'S START ADDRESS.</td>
</tr>
<tr>
<td>305</td>
<td></td>
<td>D1 CONTAINS STRING'S LENGTH (MAX = 256 CHARACTERS).</td>
</tr>
<tr>
<td>306</td>
<td></td>
<td>EXIT CONDITIONS:</td>
</tr>
<tr>
<td>307</td>
<td></td>
<td>CHARACTER STRING MOVED INTO PRINT BUFFER.</td>
</tr>
<tr>
<td>308</td>
<td></td>
<td>PRINT TAG PLACED IN PRINT QUEUE.</td>
</tr>
<tr>
<td>309</td>
<td></td>
<td>ALL REGISTERS UNALTERED.</td>
</tr>
<tr>
<td>310</td>
<td></td>
<td></td>
</tr>
<tr>
<td>311</td>
<td>0000214E 48E7F0CD</td>
<td>QSTRG MOVEL A0-A1/D0-D3-/(A7) SUBROUTINE USES REGS A0-A1/D0-D4</td>
</tr>
<tr>
<td>312</td>
<td></td>
<td>CLR.W D2</td>
</tr>
<tr>
<td>313</td>
<td></td>
<td>MOVE.B PPIN/D2</td>
</tr>
<tr>
<td>314</td>
<td></td>
<td>ADDL.B #1/D2</td>
</tr>
<tr>
<td>315</td>
<td></td>
<td>CMP.B PDOUT/D2</td>
</tr>
<tr>
<td>316</td>
<td></td>
<td>BEQ QSTRG1</td>
</tr>
<tr>
<td>317</td>
<td></td>
<td>LEA.L PRBUP/A1</td>
</tr>
<tr>
<td>318</td>
<td></td>
<td>CLR.L D3</td>
</tr>
<tr>
<td>319</td>
<td></td>
<td>MOVE.W D2/D3</td>
</tr>
<tr>
<td>320</td>
<td></td>
<td>MUL.W #CSLNTH/D3</td>
</tr>
<tr>
<td>321</td>
<td></td>
<td>LEA (D1/D3,L)/A1</td>
</tr>
<tr>
<td>322</td>
<td></td>
<td>CLR.W DO</td>
</tr>
<tr>
<td>323</td>
<td></td>
<td>MOVE.B DO/00</td>
</tr>
<tr>
<td>324</td>
<td></td>
<td>SUB.L #1/00</td>
</tr>
<tr>
<td>325</td>
<td></td>
<td>AND.L #CSLNTH/D0</td>
</tr>
<tr>
<td>326</td>
<td></td>
<td>QSTRG2 MOVEB (A0)+/(A1)+</td>
</tr>
<tr>
<td>327</td>
<td></td>
<td>DERA D0/QSTRG2</td>
</tr>
<tr>
<td>328</td>
<td></td>
<td>LEA.L PDQS/A1</td>
</tr>
<tr>
<td>329</td>
<td></td>
<td>MOVE.B D1/(A1/D0.W)</td>
</tr>
<tr>
<td>330</td>
<td></td>
<td>LEA.L PDQS/A1</td>
</tr>
<tr>
<td>331</td>
<td></td>
<td>MOVE.B D1/(A1/D0.W)</td>
</tr>
<tr>
<td>332</td>
<td></td>
<td>MOVEL (A7)+/(A0-A1/D0-D3)</td>
</tr>
<tr>
<td>333</td>
<td></td>
<td>RTS</td>
</tr>
<tr>
<td>334</td>
<td></td>
<td>RSTRG - SUBROUTINE TO RELEASE A CHARACTER STRING FROM PRINT QUEUE.</td>
</tr>
</tbody>
</table>
MOTOROLA M68000 ASM VERSION 1.30SYS: 5.APPNOTE.DUART68S.SA 04/12/84 15:14:00

DUART68S

350     RSTRG CHECKS THE STATUS OF THE PRINT QUEUE. IF THE QUEUE IS
351     EMPTY, RSTRG WILL WAIT UNTIL A PRINT TAG APPEARS IN THE QUEUE.
352     A PRINT TAG IS A BYTE containing THE LENGTH OF THE STRING TO
353     BE PRINTED.
354     IF THE PRINT QUEUE IS NOT EMPTY, RSTRG WILL SEND THE STRING
355     FROM THE PRINT BUFFER TO THE PRINTER, THEN PULL THE TAG FROM THE
356     PRINT QUEUE.
357
358     ENTRY CONDITIONS:
359
360     (NONE)
361
362     EXIT CONDITIONS:
363
364     CHARACTER STRING IS SENT FROM THE PRINT BUFFER
365     TO CHANNEL B.
366     PRINT TAG IS REMOVED FROM PRINT QUEUE.
367     ALL REGISTERS UNALJTED.
368
369
370
371 00002196 48E7C0C0 RSTRG MOVEL 20-D0/A0-A1,-(A7) SUBROUTINE USES REGS D0, D1, A0, & A1
372
373 0000219A 4260 CLRW 00
374 0000219C 10387085
375 000021A0 8038704
376 000021A4 67FA RSTRG1
377
378 000021A6 41F87186 LEAL 
379 000021AA 4281 LEAL 
380 000021AC 3200 MOVE.W 00
381 000021AE C2FC0080 MULU.W #CSLNTH,01
382 000021B2 41F31800 LEAL 0(A0,D1,L),A0
383
384 000021B6 43F87086 LEAL 
385 000021BA 4241 CLRL 01
386 000021BC 12310000 MOVE.B 0(A0,A0,W),01
387
388 000021CO 6142 BS R PSTRG
389
390 000021C2 5200 ADDQ.B #1,D0 BUMP PRINT QUEUE OUTPUT POINTER
391 000021C4 020000FF ADDI.B #POLMK,DO (KEEP POINTER WITHIN QUEUE BOUNDS)
392 000021C8 11C07085 MOVE.B CO,PQOUT
393
394 000021CC 4C0F0303 MOVEL.L (A7)+,D0,D1/A0-A1 RESTORE REGISTER CONTENTS
395 000021DD 4E75 RTS
396
397     RISTR - ROUTINE TO INPUT A CHARACTER STRING FROM THE TERMINAL & PLACE
398     IT IN INPUT BUFFER.
399     A CHARACTER STRING CAN BE A MAXIMUM OF 256 CHARACTERS LONG
400     (AS DEFINED BY THE CSLNTH), $ ENDS WITH CARRIAGE RETURN CHARACTER.
401     IF A BACKSPACE IS RECEIVED, RISTR WILL DECREMENT THE INPUT
402     BUFFER POINTER UNLESS POINTER IS AT FIRST POSITION IN BUFFER.
403
404     ENTRY CONDITIONS:
405
406     (NONE)
MOTOROLA M68000 ASM VERSION 1.30SYS : 5.APPNOTE.DUART685.5A 04/12/84 15:14:00
DUART685

0002100 4BE78000  ISTRG  MOVEM.L  DO,-(A7)  SUBROUTINE USES REGISTERS DO

0002104 4F1F7004  LEA.L  INBUF,A0  GET BASE ADDRESS OF INPUT BUFFER

0002108 4241  CLR.W 01  INIT INPUT BUFFER POINTER

000210C 61000182  GETCHAR  BSR.L  INCH  GET CHARACTER FROM CHANNEL A

000210E 0C000008  BSMK  CMP.B  #BS:/DO  IS IT A BACKSPACE CHARACTER?

0002114 6608  BNE  PUTCHAR  NO, SKIP NEXT PART

0002116 4401  TST.B 01  YES, ARE WE AT BEGINNING OF BUFFER?

0002118 67F2  BEQ  GETCHAR  YES, DO NOT DECREMENT POINTER

000211A 5301  SUB.B  #1,01  NO, DECREMENT BUFFER POINTER

000211C 600E  BRA  GETCHAR  THEN GET NEXT CHARACTER

0002120 1101000  PUTCHAR  MOVE.B  DO,(A0,DO,01.W)  PUT CHARACTER IN BUFFER,

0002124 5201  ADDQ.B  #1,01  BUMP BUFFER POINTER

0002126 0201007F  ANDQ.B  #CSLM5K,01  (KEEP IT WITHIN STRING LENGTH BOUNDS)

0002128 0C000000  CMP.B  #CR,DO  WAS IT A CARRIAGE RETURN?

000212A 660E  BNE  GETCHAR  NO, GET NEXT CHAR

000212C 40001  MOVM.L  (A7),-DO  YES, RESTORE REGISTER CONTENTS & RETURN

0002200 447E0001  RTS

*  *  PSTRG - ROUTINE TO SEND A CHARACTER STRING TO THE PRINTER.

*  *  ENTRY CONDITIONS:

*  *  A0 CONTAINS STRING'S START ADDRESS.

*  *  D1 CONTAINS STRING'S LENGTH (MAX = 256 CHARACTERS).

*  *  EXIT CONDITIONS:

*  *  CHARACTER STRING IS SENT TO PRINTER VIA CHANNEL B.

*  *  ALL REGISTERS ARE UNALTEDER.

0002204 40E7C080  PSTRG  MOVEM.L  A0,DO,-D1,-(A7)  SUBROUTINE USES REGS A0,DO,D1

0002208 5301  SUBQ.B  #1,01  INIT CHARACTER COUNT FROM STRING LENGTH

000220A 0201007F  ANDQ.B  #CSLM5K,01  (KEEP IT WITHIN STRING LENGTH BOUNDS)

000220C 1018  MOVEM.L  (A0,DO,01)  GET CHAR OF STRING TO BE PRINTED

0002210 6100018A  BSR.L  POUCH  PRINT CHARACTER

0002214 51C9FF8  DBRA  D1,PSTRG1  WAS IT THE LAST CHARACTER OF STRING?

0002218 4C0F0103  MOVEM.L  (A7),+A0,DO-D1  YES, RESTORE REGISTER CONTENTS

000221C 4E75  RTS
DINIT - QUART INITIALIZATION ROUTINE.

AFTER INITIALIZING THE QUART'S CHANNELS & COUNTER FOR OPERATION, DINIT CHECKS CHANNEL A, CHANNEL B, & THE COUNTER FOR OPERATIONAL ERRORS.

ENTRY CONDITIONS:

ALLOCATE THREE WORDS ON SYSTEM STACK BEFORE CALLING.

EXIT CONDITIONS:

THREE STATUS WORDS ARE PLACED ON THE SYSTEM STACK.

THE STATUS WORDS' FORMATS ARE AS FOLLOWS:

<table>
<thead>
<tr>
<th>WORD</th>
<th>BIT</th>
<th>STATUS (1=ERROR, 0=NO ERROR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7</td>
<td>0</td>
<td>CHAN A TRANSMITTER NEVER READY</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RECEIVER NEVER READY</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>FRAMING ERROR</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>PARITY ERROR</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>INCORRECT CHARACTER RECEIVED</td>
</tr>
<tr>
<td></td>
<td>5-15</td>
<td>NOT USED</td>
</tr>
<tr>
<td>A7</td>
<td>2</td>
<td>CHAN B TRANSMITTER NEVER READY</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RECEIVER NEVER READY</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>FRAMING ERROR</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>PARITY ERROR</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>INCORRECT CHARACTER RECEIVED</td>
</tr>
<tr>
<td></td>
<td>5-15</td>
<td>NOT USED</td>
</tr>
<tr>
<td>A7</td>
<td>4</td>
<td>COUNTER IRQ NEVER RECEIVED</td>
</tr>
<tr>
<td></td>
<td>1-15</td>
<td>NOT USED</td>
</tr>
</tbody>
</table>

IF NO ERRORS ARE FOUND IN CHAN A, DINIT WILL ENABLE A'S RX.
IF NO ERRORS ARE FOUND IN CHAN B, DINIT WILL ENABLE B'S TX.
THE COUNTER WILL NOT BE RUNNING.
ALL REGISTER CONTENTS ARE UNALTEDERED.

* CONSTANTS

00000000 CHASTS EQU 12  STACK OFFSET TO CHAN A STATUS WORD
00000000 CHASTS EQU 14  STACK OFFSET TO CHAN B STATUS WORD
00000010 CTASTS EQU 16  STACK OFFSET TO COUNTER STATUS WORD

DINIT MOVEHL AO/00,-(A7) SUBROUTINE USES REGS AO-44 & D0

* INITIALIZE QUART CHANNELS & COUNTER

0000222 13FC003000F#0  MOVE.B #51D,ACR BRG SET 1, CNTR MODE, CLK SRC: X1/16
0000222 13FC008000F#0  MOVE.B #588,CSPR A: RX & TX AT 9600 BAUD
0000222 13FC006000F#0  MOVE.B #55A,MPYA * RX-RTS, CHN ERR, PARCE PAR, 7 CHAR
```asm
588  ; TRANSmitter NEVER READY
589  ; RECEIVER NEVER READY
590  ; FRAMING ERROR
591  ; PARITY ERROR
592  ; INCORRECT CHARACTER RECEIVED

ENTRY CONDITIONS:
593  ; CHANNEL IS ALREADY CONFIGURED FOR OPERATION, BUT NOT ENABLED
594  ; AND CONTAINS BASE ADDRESS OF DUART CHANNEL.
595  ;
596  ; EXIT CONDITIONS:
597  ; CHANNEL IS RESTORED TO ORIGINAL OPERATING MODE.
598  ; A CHANNEL STATUS WORD IS PLACED IN REGISTER D0.
599  ;
600  ; THE CHANNEL STATUS WORD FORMAT IS AS FOLLOWS:
601  ;     ---
603  ;     0    1    2    3    4    5-15
604  ;     --------------------------
605  ; 0  TRANSMITTER NEVER READY
606  ; 1  RECEIVER NEVER READY
607  ; 2  FRAMING ERROR
608  ; 3  PARITY ERROR
609  ; 4  INCORRECT CHARACTER RECEIVED
610  ; 5-15 (NOT USED)
611  ;
612  ; ALL OTHER REGISTERS ARE UNALTERED.

601  000022C4 48E77000   CHKMK   MOVEM.L D1-D3,-(A7)   SUBROUTINE USES REGS D1-D3
602
603
604
605  000022C8 1610   MOVE.B  (A0),D3   SAVE ORIGINAL MR2x REGISTER CONTENTS
606  000022CA 001000FC   ORI.B   #$00, (A0)   PUT CHANNEL IN LOCAL LOOPBACK MODE 3
607  000022CE 021000AF   ANDI.B   #$00, (A0)   MAKE SURE CTS-TX IS DISABLED FOR CHECK
608  000022D0 117C00050004   MOVE.B   #$05, (A0)   ENABLE CHANNEL'S TX
609  000022D3 4240   CLR.W   DD   CLEAR CHANNEL STATUS WORD
610
611
612
613  000022EA 323CFFFF   MOV.W   #TXCNT,D1   INIT TX WAIT LOOP COUNT
614  000022F2 0B2300200002   TXCHK   BTST.B   #$22, (A0)   WAIT FOR TX TO BECOME READY
615  000022DE 56C9FFFF   DBNE   01, TXCHK   WAITED TOO LONG?
616  000022EA 6626   BNE   SNDCHR   NO, SKIP NEXT PART
617  000022E4 00400001   ORI.W   #$0001, D0   YES, SET TX-NEVER-READY FLAG BUT
618  000022EE 6042   BRA   RSTCM   & SKIP REST OF CHECK
619  000022FC 117C00550006   SNDCHR   MOVE.B   #$55, (A0)   TX IS READY, SEND TEST CHARACTER
620
621
622
623  000022F4 323CFFFF   MOV.W   #RXCNT,D1   INIT RX WAIT LOOP COUNT
624  000022FA 0B2800000002   RXCHK   BTST.B   #$02, (A0)   WAIT FOR RX TO RECEIVE CHARACTER
625  000022FC 55C9FFFF   DBNE   01, RXCHK   WAITED TOO LONG?
```
MOTOROLA M68000 ASM VERSION 1.30SYS: 5.APPNOTE.DUART635.SA 04/12/84 15:14:00

<table>
<thead>
<tr>
<th>Line No.</th>
<th>Address</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>626</td>
<td>00002304</td>
<td>BNE</td>
<td>FRCHK</td>
</tr>
<tr>
<td>627</td>
<td>00002305</td>
<td>ORI.W</td>
<td>#0002,00</td>
</tr>
<tr>
<td>628</td>
<td>00002306</td>
<td>BRA</td>
<td>RSTCHN</td>
</tr>
<tr>
<td>629</td>
<td>00002307</td>
<td>STST.B</td>
<td>#5,2(AD)</td>
</tr>
<tr>
<td>630</td>
<td>00002308</td>
<td>REQ</td>
<td>PRCHK</td>
</tr>
<tr>
<td>631</td>
<td>00002309</td>
<td>ORI.W</td>
<td>#0004,00</td>
</tr>
<tr>
<td>632</td>
<td>00002310</td>
<td>STST.B</td>
<td>#5,2(AD)</td>
</tr>
<tr>
<td>633</td>
<td>00002311</td>
<td>BEQ</td>
<td>CHRMCH</td>
</tr>
<tr>
<td>634</td>
<td>00002312</td>
<td>ORI.W</td>
<td>#1005,00</td>
</tr>
<tr>
<td>635</td>
<td>00002313</td>
<td>CMF.B</td>
<td>#115,02</td>
</tr>
<tr>
<td>636</td>
<td>00002314</td>
<td>BEQ</td>
<td>RSTCHN</td>
</tr>
<tr>
<td>637</td>
<td>00002315</td>
<td>ORI.W</td>
<td>#0010,00</td>
</tr>
<tr>
<td>638</td>
<td>00002316</td>
<td></td>
<td></td>
</tr>
<tr>
<td>639</td>
<td>00002332</td>
<td></td>
<td></td>
</tr>
<tr>
<td>640</td>
<td>1170000A0004</td>
<td>MOVE.B</td>
<td>#00A,4(AD)</td>
</tr>
<tr>
<td>641</td>
<td>00002333</td>
<td>MOVE.B</td>
<td>071,4(AD)</td>
</tr>
<tr>
<td>642</td>
<td>1083</td>
<td></td>
<td></td>
</tr>
<tr>
<td>643</td>
<td>00002338</td>
<td></td>
<td></td>
</tr>
<tr>
<td>644</td>
<td>0400000E</td>
<td>MOVEM.L</td>
<td>(A7)+,D1-03</td>
</tr>
<tr>
<td>645</td>
<td>0000233E</td>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td>646</td>
<td>4075</td>
<td></td>
<td></td>
</tr>
<tr>
<td>647</td>
<td>4076</td>
<td></td>
<td></td>
</tr>
<tr>
<td>648</td>
<td>4077</td>
<td></td>
<td></td>
</tr>
<tr>
<td>649</td>
<td>4078</td>
<td></td>
<td></td>
</tr>
<tr>
<td>650</td>
<td>4079</td>
<td></td>
<td></td>
</tr>
<tr>
<td>651</td>
<td>407A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>652</td>
<td>407B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>653</td>
<td>407C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>654</td>
<td>407D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>655</td>
<td>407E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>656</td>
<td>407F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>657</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>658</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>659</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>660</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>661</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>662</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>663</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>664</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>665</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>666</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>667</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>668</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>669</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>670</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>671</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>672</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>673</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>674</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>675</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>676</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>677</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>678</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>679</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>680</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>681</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>682</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>683</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* CHANNEL CHECK COMPLETE, STACK STATUS WORD & RESTORE
  * CHANNEL TO ORIGINAL MODE OF OPERATION.
  * CTRCHK - COUNTER CHECK ROUTINE.
    * CHECKS UART COUNTER FOR OPERATIONAL ERRORS.
    * AFTER RE-POINTING THE UART'S EXCEPTION VECTOR
    * TO ITS OWN INTERRUPT Handler, CTRCHK STARTS THE
    * COUNTER & WAITS FOR THE COUNTER TO GENERATE AN IRQ.
    * ENTRY CONDITIONS:
      * UART CONFIGURED FOR A COUNTER IRQ (IRIC(3)=1).
      * IRQ VECTOR REGISTER IS ALREADY INITIALIZED.
      * COUNTER UPPER & LOWER REGISTERS ARE ALREADY INITIALIZED.
      * COUNTER IS NOT RUNNING.
    * EXIT CONDITIONS:
      * ORIGINAL UART EXCEPTION VECTOR IS RESTORED.
      * A COUNTER STATUS WORD IS PLACED IN REGISTER 01.
      * THE ERROR STATUS WORD FORMAT IS AS FOLLOWS:
      * BIT | STATUS (1=ERROR, 0=NO ERROR)
      * --- | ------------------------------
      * 0   | COUNTER IRQ NEVER RECEIVED
      * 1-15 | (NOT USED)
      * ALL OTHER REGISTERS ARE UNALTERED.
      * SUBROUTINE USES REG 01

CTRCHK MOVEM.L D1-(A7) SUBROUTINE USES REG 01

582

MOVE.L DIPQVEC-(A7) SAVE ORIGINAL EXCEPTION VECTOR
MOTOROLA M6800C ASM VERSION 1.30SYS : SAPPNOTE .DUART68S.SA 04/12/84 15:14:00
DUART68S

684  00002346 21FC0000237A  MOVE.L  #CIRQ,DIRQVEC  RE-POINT EXCEPTION VECTOR
685
686  00002350 4240  CLRT.W  00  CLEAR COUNTER STATUS WORD
687  00002352 4A39000001  TST.B  0RC  START COUNTER
688
689  00002358 323CFFFF  MOVE.W  #IRQCNT+01  INIT IRQ #BIT LOOP COUNT
690  0000235C 023C00FE  AND.L  #(IR)  CLEAR CARRY BIT
691
692  00002360 55CFFFFF  MTRQ  OBCS  DJWTRQ  WAIT FOR COUNTER IRQ: WAITED TOO LONG?
693
694  00002364 6534  BCS  CTCHKR  NO, SKIP NEXT PART
695  00002366 04000001  ORI.W  #01/00  YES, SET IRO-NEVER-REC'D FLAG BIT
696
697  * COUNTER CHECK COMPLETE, STOP COUNTER, RESTORE ORIGINAL EXCEPTION VECTOR,
698  * & STACK ERROR STATUS WORD.
699
700  0000236A 4A39000001  CTCHKR  TST.E  STPC  STOP COUNTER
701  00002370 215F036C  MOVIL.L  (A7)+,DIRQVEC  RESTORE ORIGINAL EXCEPTION VECTOR
702
703  00002374 4C0F0002  MOVEM.L  (A7)+,01  RESTORE REGISTER CONTENTS
704  00002378 4E75  RTS
705
706
707  * CIRQ - COUNTER CHECK IRQ HANDLING ROUTINE.
708  * DUART IRQ HANDLING ROUTINE USED DURING CTCHKR ONLY.
709
710  * ENTRY CONDITIONS:
711
712  * DUART IRQ.
713
714  * EXIT CONDITIONS:
715
716  * IF COUNTER WAS CAUSE OF DUART IRQ:
717  * COUNTER/TIMER READY BIT CLEARED IN DUART'S ISR,
718  * & CARRY BIT SET.
719
720  * OTHERWISE:
721  * CARRY BIT REMAINS CLEARED.
722
723  0000237A 0839D000300F0  CIRQ  BST.B  #3.ISR  WAS IRQ CAUSED BY COUNTER?
724
725  0000237E 673A  BEQ  CIRQR  NO, SKIP NEXT PART
726  00002384 4A39000001  TST.E  STPC  YES, STOP COUNTER
727  0000238A 00870001  ORI  #0001,(A7)  & SET CARRY BIT OF SR ON STACK
728
729  * CIRQR  RTE
730
731  * INCH - TERMINAL INPUT CHARACTER ROUTINE.
732  * GETS CHARACTER FROM TERMINAL VIA DUART CHANNEL A,
733  * THEN PLACES IT IN DO.
734  * (BECAUSE CHN A IS IN AUTO-ECHO MODE, CHARACTER DOES NOT NEED TO
735  * BE RE-TRANSMITTED BACK TO TERMINAL BY SOFTWARE.)
736
737  * ENTRY CONDITIONS:
738
739  * DUART CHANNEL A RX & TX ENABLED.
740  *  EXIT CONDITIONS:
741  *  
742  *  RECEIVED CHARACTER PLACED IN DO.
743  *  
744  *  ALL OTHER REGISTERS UNALTERED.
745  *  
746  
747  000023A0 0839000000FD INCH BTST.3 #0,SRA WAIT FOR CHAN A'S RX TO GET A CHAR
748  000023A0 0003 BEQ INCH     GET CHARACTER FROM RECEIVER
749  000023A0 10390000007 MOVE.6 R8A,00 RTS
750  000023A0 4E75
751  
752  *  OUTFCH - TERMINAL OUTPUT CHARACTER ROUTINE.
753  *  
754  *  OUTPUTS CHARACTER IN DO TO TERMINAL VIA CHAN A'S TX.
755  *  
756  *  IF CHARACTER IN DO IS A CARRIAGE RETURN, OUTFCH WILL
757  *  OUTPUT BOTH A CARRIAGE RETURN & LINE FEED CHARACTER.
758  *  
759  *  ENTRY CONDITIONS:
760  *  
761  *  DUART CHANNEL A TX ENABLED.
762  *  
763  *  CHARACTER TO BE TRANSMITTED IN DO.
764  *  
765  *  EXIT CONDITIONS:
766  *  
767  *  ALL REGISTERS UNALTERED.
768  *  
769  
770  000023A2 083900002000FD OUTFCH BTST.8 #2,SRA WAIT FOR CHAN A'S TX TO BECOME READY
771  000023A0 67F6 BEQ OUTFCH
772  000023A0 13C00000007 MOVE.6 DO,T8A SEND CHAR TO TRANSMITTER
773  000023A0 0C000000 CMP.6 #CR,DO WAS IT A CARRIAGE RETURN?
774  000023B6 6612 BNE OUTFCH NO, SKIP NEXT PART
775  000023B9 083900020000 OUTFCH1 BTST.8 #2,SRA YES, WAIT FOR TX TO BECOME READY AGAIN
776  000023A0 67F6 BEQ OUTFCH1
777  000023C2 13FC00040000 MOVE.6 #LF,T8A SEND A LINE FEED
778  000023A0 4E75 OUTFCH RTS
779  
780  *  POUTCH - PRINTER OUTPUT CHARACTER ROUTINE.
781  *  
782  *  OUTPUTS CHARACTER IN DO TO PRINTER VIA CHAN B'S TX.
783  *  
784  *  IF CHARACTER IN DO IS A CARRIAGE RETURN, POUTCH WILL
785  *  OUTPUT BOTH A CARRIAGE RETURN & LINE FEED CHARACTER.
786  *  
787  *  ENTRY CONDITIONS:
788  *  
789  *  DUART CHANNEL B TX ENABLED.
790  *  
791  *  CHARACTER TO BE TRANSMITTED IN DO.
792  *  
793  *  EXIT CONDITIONS:
794  *  
795  *  ALL REGISTERS UNALTERED.
796  *  
797  *  CHARACTER SENT TO CHANNEL B TX.
<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000242E</td>
<td>49#2446</td>
<td>LEAF</td>
<td>Break received message</td>
</tr>
<tr>
<td>0002432</td>
<td>49#0007</td>
<td>LEAF</td>
<td>Break received message</td>
</tr>
<tr>
<td>0002435</td>
<td>4E#C0F3</td>
<td>MOVE.B</td>
<td>#2447, C7</td>
</tr>
<tr>
<td>000243A</td>
<td>46#E</td>
<td>TRAP</td>
<td>#14</td>
</tr>
<tr>
<td>000243C</td>
<td>47#000300000000</td>
<td>MOVE.L</td>
<td>Monitor, 2(47), no, exit to monitor</td>
</tr>
</tbody>
</table>

**Break Received Message**

**Counter Error Message**

**Channel B TX never ready message**

**Channel B RX never ready message**

**Channel B framing error message**

**Channel B parity error message**

**Channel B incorrect char rec'd message**

**Channel A TX never ready message**

**Channel A RX never ready message**

**Channel A framing error message**

**Channel A parity error message**

**Channel A incorrect char rec'd message**
LCHAMS05 EQU --CHAMS05

DORMANT TASK'S SYSTEM STACK POINTER

INPUT BUFFER

PRINT QUEUE INPUT POINTER

PRINT QUEUE OUTPUT POINTER

PRINT QUEUE

PRINT BUFFER

EXCEPTION VECTOR TABLE ENTRIES

DIRQ EXCEPTION VECTOR

END

***** TOTAL ERRORS 0--
***** TOTAL WARNINGS 0--
Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola and (M) are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Employment Opportunity/ Affirmative Action Employer.