NATIONAL RADIO ASTRONOMY OBSERVATORY
Green Bank, West Virginia

300-FOOT CONTROL COMPUTER MEMO NO. 17

DA/CP IMPLEMENTATION OF THE 10-ms SIDEREAL CLOCK COUNTER

R. Fisher

August 28, 1985

DA/CP Implementation of the 10-ms Sidereal Clock Counter

27 August 1985
R. Fisher

Part of the sidereal time keeping scheme in the 300-ft control computer system is a 10-ms pulse counter which keeps track of elapsed time since the system's master clock was last read. Servicing this rapid stream of interrupts would impose a considerable timing burden on the main processor, so the DA/CP's own software has been modified to include this counter function and to pass the counter value to the cpu upon request.

External interrupts to the MASSCOMP computer must go through the DA/CP. These can be generated by the I/O modules in the DA/CP or can come from or go through the DA/CP clock module. All of these interrupts enter the DA/CP on the STD bus with one interrupt line for each I/O slot. Slot 0 (the one next to the clock module) has the highest interrupt priority, and higher numbered slots have successively lower priorities. For this application we will have an externally generated 100 Hz pulse train applied to the "S" input of clk0 of the clock module which in turn will generate an interrupt every 10 ms on the slot 0 STD interrupt line. This slot has no I/O module plugged into it which would normally use this interrupt line. For test purposes, the clock module can be programmed to generate a 100 Hz pulse train on the slot0 interrupt line to simulate the external generator.

Each I/O module has a block of DA/CP program code associated with it to control the flow of data and control between the module and the memory in the host computer. Also, associated with every DA/CP slot, including empty ones, are small blocks of code which allow the host cpu to effectively talk directly to and receive data and interrupts directly from each slot in a mode called the buswindow. Before this modification of the DA/CP code the host cpu could receive the 10-ms interrupts directly from slot 0 and keep track of them by incrementing a counter in its own memory. The modified DA/CP code intercepts the 10-ms pulses and increments a DA/CP register counter, and when the host cpu asks for data from slot 0 in the buswindow mode the new code causes the DA/CP to send the counter value as if it had come from an I/O module in this slot. When the counter is read it is reset to zero to keep it from overflowing.

As of this writing some consideration is being given to the possibility of accessing the 10-ms counter from the code (handlers) for other I/O modules in the DA/CP, and the counter word size may be increased to 24 or 32 bits, so the details of the DA/CP code accompanying this note may have been changed by the time you read this, but the general idea should be the same.

Figures 1 and 2 show the logic flow associated with the counter incrementing in response to clock interrupts and the access of the counter value by the host cpu. The first program listing labelled "buswnclk.asd" shows the modified buswindow handler code. The changes are marked in the left margin. One of the three new constants (slot0_rd_mask=0x2e001e) defines the I/O address which the host cpu must read to get the counter

value. In this case the address is (1E hex = 30 decimal). The counter is kept in a DA/CP register "clock_counter". The piece of code under ORIGIN(7) tests the cpu read request for the assigned address and, if a match is found, branches to the new code which puts the counter value in the driver's data word and resets the counter.

ORIGIN(960) is the entry point for the slot 0 interrupt. The register SD is the STD data register to which a command word is written to clear the clock interrupt, SA is the address register for this command word, and QR is a general purpose register used as a dummy here. The operation BLEQUW means "branch if less than or equal to zero" and refers to the preceding CoMPare. The NOP is for timing delay.

The second program listing shows how the DA/CP clock may be used from a C program. The procedure "start_clock" contains the MASSCOMP supplied DA/CP routine references to set up the internal clock or route the external pulses to the STD bus and to open the path for buswindow access to the counter. The routine "sidereal_time" contains the read statement "mrwnrd" for the counter value which ends up in "dacp_counter". The references "mrwnion" and "mrclkintgat" in "reset_clock" enable AST's from the driver and interrupts to the DA/CP from the STD bus, respectively. If the counter has not been read for some time, and it is in danger of overflowing, the DA/CP and its driver generate an AST whose service routine has been defined to be "inc_ten_ms_count" by the open-window call at the end of "start_clock". (See the description of the "mrwnopn" routine in the DA/CP applications programming manual.)

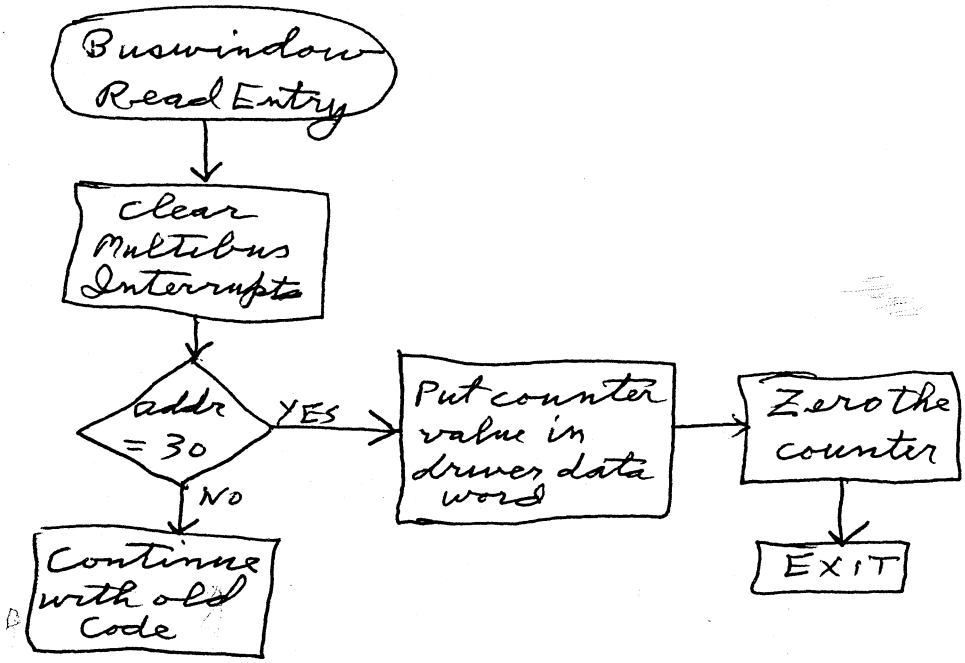

Figure 1

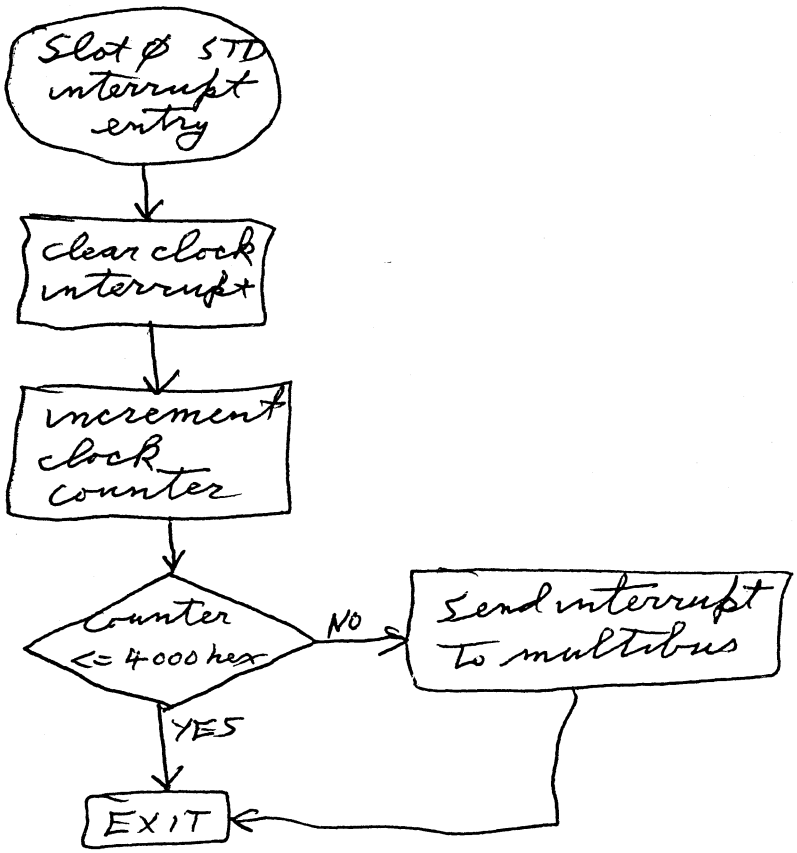

Figure 2

```
; @(#)buswindow.asd     1.1 (MASSCOMP) 10/23/84
;
BEGIN_MODULE
;       ****    Bus Window Mode IPC Handler  ****
;
;
;Revision History
;24-Sep-82       Created
;30-Oct-82       Final Tweaks
;04-Nov-82       Disable Clock Interrupts on Start Up
;10-Dec-82       Change Clock address to FF00 in memory space
;04-Jan_83       Add Register with constant 4
;11-Feb-83       Add Free Pool
;16-Feb-83       Fix Start Code so Fifo Enables correctly
;25-Mar-83       Modify syntax for new assembler
;28-Mar-83       declare "idle" global
;                remove "PSECT" from "bwcode"
;04-apr-83       change bus window interrupt enable/disable to
;                get mask from "bwadr"
;13-Apr-83       add handler table output operations
;08-sep-83       D. Cane broke out all waits into callable routine
;23-aug-85       R. Fisher added patches to implement clk0 intr counter
;*************************************************************************
; this is the default load image for the ipc that i will call its
;operating system. it contains bus window mode for each std interrupt,
;the idle loop, the subroutine return code and the fifin interrupt
;handler. it also has the code for handler entry points


;*************************************************************************
; these are global constants for use by any ipc device handler

intr_clr=0xf800
fi_ov_set==0x201
init_cr=0x241   ;reset fov, enable fi interrupts, disable slot intrpts
clk_wr_csr==0x1eff00
intr_no_stat0==0x600080
intr_no_stat1==0x600088
intr_no_stat2==0x600090
intr_no_stat3==0x600098
intr_no_stat4==0x6000a0
intr_no_stat5==0x6000a8
intr_no_stat6==0x6000b0
intr_no_stat7==0x6000b8
intr_no_stat_ex0==0x60003c
intr_no_stat_ex1==0x600034
intr_stat0==0x600082
intr_stat1==0x60008a
intr_stat2==0x600092
intr_stat3==0x60009a
intr_stat4==0x6000a2
intr_stat5==0x6000aa
intr_stat6==0x6000b2
intr_stat7==0x6000ba
intr_stat_ex0==0x60003e
intr_stat_ex0==0x600036
```

```
;*****************************************************
;Constants associated with 10-ms clock counter in slot 0

slot0_rd_mask=0x2e001e  ;slot0 read address (30 decimal)
intr0_clr=0x008000      ;clk0 interrupt clear
clk_wr_csr4==0x1eff04   ;clk0-3 command address

CONFIGURATION CONFIGURATION_NUMBER, dacp_start, entry_pt, bwcmd

BEGIN_MODULE
PARAMETER
DECLARE bwcmd,DS,GLOBAL,0x000000        ;bus window command cmdpatch
DECLARE bwadr,DS,GLOBAL,0x000000        ;bus window address
DECLARE bwdat,DS,GLOBAL,0x000000        ;bus window data
END_PARAMETER
DECLARE entry_pt,DS,GLOBAL              ;INT0 entry point

ENTRY bw_rd_wr, bw_rd_wr, 0, 0, 0, 0, 0
ENTRY bw_ints, bw_interrupts, 0, 0, 0, 0, 0
STD -1, 0, 0, 0, 0

DECLARE tagvec,DS,LOCAL                 ;temp. loc. for jump thru fifin tag
DECLARE lit_4,REG,GLOBAL                ;will be loaded with constant 4
DECLARE clock_counter,REG,STATIC        ;counter to be used with 10ms sidereal
                                        ;  pulses (300' cntl comp)

HANDLER 0, 0, 0
;*****************************************************************
; this is the code for the "service" interrupts that ARE( part the ipc
;hardwARE(; idle loop, subroutines, fifo in interrupt, etc.....

ORIGIN(0)                               ;idle loop interrupt location
idle::
INT     JMP idle                        ;idle loop must leave DS free
ORIGIN(1)                               ;subroutine interrupt location
        RESUME                          ;pop stack to subroutine
ORIGIN(2)                               ;mbus 0 interrupt location
        JMP entry_pt                    ;jump to handler entry point
ORIGIN(3)                               ;fifo in interrupt handler
        MOV FI,tagvec                   ;get tag for jump to service
        JMP tagvec                      ;go to tag defined code
ORIGIN(5)                               ;ex 0 interrupt location
        JMP ex0loc                      ;handle external interrupt 0
ORIGIN(6)                               ;ex 1 interrupt location
        JMP ex1loc                      ;handle external interrupt 1

;*****************************************************************
; bus window read, write and reset code

ORIGIN(7)                               ;mbus 1 interrupt location
bwcode:
        MOV bwdat,SD                    ;loading data does no harm
        MOV bwadr,SA                    ;do the command
        JMP bwcmd                       ;command determines length
bw_rd_wr:
```

```
        MOV #4,CR                       ;clear the multibus interrupt
        MOV SA,QR
        CMP QR,#slot0_rd_mask           ;is this a word read req from slot0?
        BEQLG get_counter               ;go to special slot0 patch to get
                                        ;     10 ms counter
        jsr wait3
        MOV SD,bwdat                    ;fast read & get data
INT     CLR SA                          ;clear STD BUS reset

;*************************************************************************
; bus window mode interrupt enable/disable code

bw_interrupts:
        MOV #4,CR                       ;clear multibus interrupt
        MOV bwadr,CR                    ;enable/disable STD interrupt
        jmp idle

bw_intr_send:
        MOV QR,FO                       ;send multibus interrupt
INT     MOV QR,QR                       ;nop
;*************************************************************************
;start up code

dacp_start:
        JMP ipc_start1                  ;load next address into pc
ipc_start1:
        MOV #intr_clr,SD                ;disable ex0 interrupt
        MOV #clk_wr_csr,SA,QR           ;on the clock module
        MOV #init_cr,CR                 ;1 enb. fifin int. & clear ovflow
                                        ; disable all interrupts
        jsr wait4
        ADD #4,QR                       ;5 prepARE( to clear ex1
        MOV #intr_clr,SD                ;set to clear ex1
        MOV QR,SA                       ;clear ex1
        MOV #4,lit_4                    ;3 load constant register
        jsr wait5
INT     MOV QR,SD                       ;and terminate the write

;*************************************************************************
;clk0 counter fetch

get_counter:
        MOV clock_counter,bwdat         ;get the current counter value
        CLR clock_counter               ;reset the counter
INT     CLR SA                          ;clear STD bus reset and get next intr

;*************************************************************************
;ipc interrupt handing code

ex0loc:
        MOV #intr_no_stat_ex0,QR,CR     ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ex1loc:
        MOV #intr_no_stat_ex1,QR,CR     ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(512)                             ;slot a interrupt location
```

```
        MOV #intr_no_stat0,QR,CR        ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(576)                             ;slot b interrupt location
        MOV #intr_no_stat1,QR,CR        ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(640)                             ;slot c interrupt location
        MOV #intr_no_stat2,QR,CR        ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(704)                             ;slot d interrupt location
        MOV #intr_no_stat3,QR,CR        ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(768)                             ;slot e interrupt location
        MOV #intr_no_stat4,QR,CR        ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(832)                             ;slot f interrupt location
        MOV #intr_no_stat5,QR,CR        ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(896)                             ;slot g interrupt location
        MOV #intr_no_stat6,QR,CR        ;disable ipc interrupt
        JMP bw_intr_send                ;go to common interrupt service
ORIGIN(960)                             ;slot h interrupt location
        MOV #intr0_clr,SD               ;clear clk0 interrupt
        MOV #clk_wr_csr4,SA
        INC clock_counter               ;add one to elapsed time counter
        CMP clock_counter,#0x004000     ;is counter getting full?
        NOP
        BLEQUW toidle                   ;if not reenable intr
        MOV QR,SD                       ;complete intr clr xfer
        MOV #intr_no_stat7,QR           ;load interrupt I.D.
        JMP bw_intr_send                ;go to common interrupt service
toidle:
        MOV QR,SD                       ;complete intr clr xfer
        JMP idle
;*************************************************************************
psect
wait5:: mov qr,qr
wait4:: mov qr,qr
wait3:: mov qr,qr
wait2:: mov qr,qr
wait1:: rts

psect
quit5:: mov qr,qr
quit4:: mov qr,qr
quit3:: mov qr,qr
quit2:: mov qr,qr
quit1:: int clr sd
END_MODULE

;The following constants are defined for use by all IPC ucode.

enable_intr==0xc0       ;enable selected interrupt in CR
disable_intr==0x80      ;disable selected interrupt in CR

io_rd_B==0x2d0000       ;read STDL onto low and mid byte of inbus
io_rd_W==0x2e0000       ;read word to inbus
```

```
mem_rd_B==0x0d0000      ;read STDL onto low and mid byte of inbus
mem_rd_W==0x0e0000      ;read word to inbus

io_wr_H==0x380000       ;write mid byte onto STDL & STDH
io_wr_L==0x350000       ;write low byte onto STDL & STDH
io_wr_W==0x3e0000       ;write word to STDL & STDH
mem_wr_H==0x180000      ;write mid byte onto STDL & STDH
mem_wr_L==0x150000      ;write low byte onto STDL & STDH
mem_wr_W==0x1e0000      ;write word to STDL & STDH
fast_std==0x400000      ;fast write mode bit

rd_byte==0x800000       ;multibus byte read
rd_word==0x900000       ;multibus word read
rd_long==0xb00000       ;multibus longword read
wr_byte==0xc00000       ;multibus byte write
wr_word==0xd00000       ;multibus word write
wr_long==0xf00000       ;multibus longword write
loop_back==0x200000     ;loop back
ipc_intr==0x600000      ;multibus interrupt
```

```
/*
**      This is a demonstration of the use of the 10 ms sidereal time
**  counter in the DA/CP.  The DA/CP buswindow code has been modified to
**  intercept interrupts from the clock module and increment a counter
**  in the DA/CP on each interrupt.  This counter may be read from a
**  program in the host computer.  The DA/CP counter is only 15 bits wide
**  so when it is read it is reset to zero, and the host program must
**  keep its own elapsed time counter to which each reading is added.
**  If the DA/CP counter is not read after about 2min 40sec the DA/CP
**  will interrupt the host with an AST to a service routine which is
**  expected to read the DA/CP counter and update its own counter.
**      The program is in file /users/staff/rick/Asdfiles/clockdemo.c.
**  Until the DA/CP code is permanently modified you will have to load
**  the patched code by running the shell script
**                /users/staff/rick/Asdfiles/ld
**  This will load only the clock and buswindow code so the other DA/CP
**  modules will be inoperative.  To compile this code use
**                cc clockdemo.c -lmr -lm
**
**                                        R.Fisher 26 Aug. 1985
*/

#include <fcntl.h>
#include "/usr/include/mr.h"

int clockpn;              /* dacp clock0 path number */
int ten_ms_counter;       /* variable containing elapsed time in ms */

main()

   {
   int ext_int = 0;         /* assume internal clock time generator */
   int lst;

   start_clock(ext_int);    /* set up internal clock rate or route external
                               10 ms clock pulses to STD interrupt line */
   reset_clock();           /* zero elapsed time counter (ten_ms_counter)
                               and enable clock interrupts */
   while(1)
      {
      lst = sidereal_time();  /* gets current elapsed time in 10 ms unite */

      display(lst);           /* demonstration routine to show elapsed time */

      sleep(10);
      }
   }

/*
**  Return the current elapsed sidereal time
*/

sidereal_time()

   {
   short dacp_counter;
```

```
      /* read the dacp counter and reset it  */
   mrwnrd(clockpn,30,2,&dacp_counter);
   return(ten_ms_counter = ten_ms_counter+dacp_counter);
   }

/*
**  This routine displays elapsed time in hh:mm:ss.ss format
*/

display(lst)

   int lst;
   {
   int h,m,s,f1,f2;

   f1 = (lst/10) % 10;
   f2 = lst % 10;
   s = lst/100;
   m = (s/60) % 60;
   h = (s/3600) % 24;
   s = s % 60;
   printf("Sidereal time = %d:%d:%d.%d%d\n",h,m,s,f1,f2);
   }

/*
**  Updates the 10ms elapsed time counter if the dacp counter is getting
**  full.  This can happen if the time has not been read for over 2m40s.
**  When the dacp counter gets half full it interrupts the host computer
**  with and AST to this routine.
*/

inc_ten_ms_count()
   {
   short dacp_counter;

      /* read the dacp counter and reset it  */
   mrwnrd(clockpn,30,2,&dacp_counter);
   ten_ms_counter = ten_ms_counter+dacp_counter;
      /* re-enable AST interrupt from dacp */
   mrwnion(clockpn);
   }

/*
**  Set elapsed time counter to zero and start clock
*/

reset_clock()
   {
   short dacp_counter;

   ten_ms_counter = 0;
   mrwnrd(clockpn,30,2,&dacp_counter);  /* zero dacp_counter */
   mrwnion(clockpn);                    /* enable dacp interrupt to cpu */
   mrclkintgat(clockpn,1);              /* enable clock interrupt */
   }
```

```
/*
**  This routine opens the clock path and sets up the internal clock parameters
**  or routes the external pulses.
*/

start_clock(ext_int)

   int ext_int;             /* 0 = internal clock */
   {
   int read_write = 1;      /* allow read only to clock */
   int nearest = 0;         /* pick closest clock frequency to one specified */
   int square = 4;          /* use square clock waveform */
   int low = 0;             /* waveform begins low */
   int ndiv = 1;            /* divisor for external pulse rate */
   int armsw = 1;           /* arm the external clock */
   int nclks = 1;           /* number of clocks to be armed */
   int pnarray[1];          /* array containing clock path nos. */
   int naddrs = 1;          /* number of STD address ranges */
   int adrsarray[3];
   int intpri = 127;        /* AST priority of service routine */

   double trigfreq = 100.*366.25/365.25;
                            /* internal clock sidereal rate in Hz */
   double freturn;          /* internal clock rate actually set */
   double wreturn;          /* internal pulse width actually set */

   adrsarray[0] = 0;        /* dacp clock is treated as an I/O address */
   adrsarray[1] = 30;       /* address start */
   adrsarray[2] = 30;       /* address end */

   mropen(&clockpn,"/dev/dacp0/clk0",read_write);/* open clock path */
   mrclkintgat(clockpn,0);                       /* disable clock interrupts */

   if(!ext_int)
      {
          /* set up internal clock parameters */
      mrclk1(clockpn,nearest,trigfreq,&freturn,square,0.0,&wreturn,low);
          /* start the clock generator */
      pnarray[0] = clockpn;
      mrclkarm(nclks,pnarray);
      printf("Clock freq. requested = %f Hz\n",trigfreq);
      printf("         actually set = %f Hz\n",freturn);
      }
   else
          /* send external 10ms pulses directly to STD bus */
      {
      mrclkbyps(clockpn,ndiv,armsw);
      }

          /* open protected window to slot0 */
   mrwnopn(clockpn,naddrs,adrsarray,inc_ten_ms_count,intpri);
   printf("Clock started\n");
   }
```