ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Brandywine Communications / JXI2 PMC-SyncClock32-UNIV* at our website: *Click HERE*

PMC-SYNCCLOCK32-UNIV
Synchronized Clock with 32 bit PCI 3.3V or 5V Signaling Interface
Operation & Maintenance Manual
13 JUNE 2005
Document No. 8004200
Module S/N 32531 H/W Rev Level:B6
Options: -TAG3IRS422-HQ-100PPSPW422-PTTIIN-NORDEN2-ITR-CONFCOAT

jxi2, Inc.

10432 Brian Mooney Ave.
El Paso
TX 79935
USA

phone: 915-856-0241
fax: 915-856-0245

Manual Addendum for -HQ -100PPSPW422-PTTIIN-NORDEN2 options

The PMC-SYNCCLOCK32 module shipped with this manual has the -HQ, -100PPSPW422, -PTTIIN, and –NORDEN2 options installed.

The -HQRS422 option provides for use of Havequick II serial and 1 PPS time reference per ICD-GPS-060A, The input pin assignments are:

1PPS /GND:	1PPS: J3 pin 14 /GND: J3 pin 32
serial HavequickII /GND:	HQII: J3 pin 17 /GND J3 pin 35

The –100PPSPW422 output option provides 2 100PPS RS422 outputs. The pulse asserts (+ side goes high, - side goes low) on 10 millisecond boundaries of the on-board clock. The pulse de-asserts when the digits programmed by the user into the Match_Usec10_Usec1_Port register and the Match_Msec1_Usec100_Port register match the 1 millisecond through 1 microsecond digits of the on-board clock. So, the pulse width may be varied by changing the match times from Match_Msec1_Usec100_Port=00, Match_Usec10_Usec1_Port=0x01 for a pulse width of 1 microsecond to from Match_Msec1_Usec100_Port=0x99, Match_Usec10_Usec1_Port=0x99 for a pulse width of 1 microsecond for a pulse width of 9999 microseconds. The 100PPS pin assignments are:

100PPS +/- side output 1	+:J3 pin 27	-: J3 pin 9
100PPS +/- side output 2	+:J3 pin 8	-: J3 pin 26

The –PTTIIN option provides for use of PTTI serial (50 bps) and 1 PPS time reference per ICD-GPS-060B, at RS422 levels terminated on-board in 100 ohms. The input pin assignments are:

1PPS +/- side:	+: J3 pin 2	-: J3 pin 20
serial PTTI (50 bps) +/- side:	+: J3 pin 6	-: J3 pin 24

The –NORDEN2 option providies for selection of Havequick II or PTTI 50 bps serial messages as the time message using dual port RAM location DP_GPS_Status (see gpsdef.h). If bit 0 is CLEARED, Havequick II input will be used. If bit 1 is SET serial 50 pps PTTI input will be used. Default on power-on or reset is CLEARED.

-TAG3IRS422 Option Manual addendum

The –TAG3IRS422 option provides a total of 3 external event time tags with RS422 inputs level terminated in 100 ohms compared with only one single ended TTL level input on the standard PMC-SYNCCLOCK32. Each of the 3 external time tags has its' own polarity (↑ or ↓ edge) and its' own interrupt enable. Each external time tag has 2 individually addressable 32 time registers and a control/status register
that also contains the low order BCD digit of the captured time (100 ns digit) in bits <7:4>.

External event time tag J3 pin assignments

The J3 pin assignments for the 8 external event time tag inputs are:

External time tag 0 +/- side:	+: J3 pin 7	-: J3 pin 25
External time tag 1 +/- side:	+: J3 pin 4	-: J3 pin 22
External time tag 2 +/- side:	+: P4 pin 49	-: P4 pin 51

Programming external event data input (see t8iregs.h)

The `Ext_Ready` and `Ext_Polarity` bits of the `Status_Port` and the `Ext_Intr_Enb` bit of the `Control_Port` are "0" because separate byte width status/control registers are implemented with the -TAG8I .

Whenever an external time tag occurs, the `Ext_Ready` bit in the corresponding `Ext<x>_Ctl_Sts_Port` will be set. The bit is cleared when the first of the two longwords of time tag data is read from `Ext<x>Sec10_Usec1_Port` or `Ext<x>Year1_Min1_Port`. If the `Ext_Intr_Enb` bit of `Ext<x>_Ctl_Sts_Port` is set when an event occurs or is simulated, an interrupt request will occur.

To select rising/falling edge time tag polarity for external event input <x>:
Clear (rising edge) or set (falling edge) the `Ext_Pol` bit in `Ext<x>_Ctl_Sts_Port`.
To enable interrupts for external event input <x>:
Set (enable) or clear (disable) the `Ext_Intr_Enb` bit in `Ext<x>_Ctl_Sts_Port`.
To simulate external event input <x>:
Write a 1 in `Trigger_Sim_Ext_Time_Tag` bit in `Ext<x>_Sts_Ctl_Port`.
The `Trigger_Sim_Ext_Time_Tag` bit is write-only. It will read back as 0.
To read the time tag data for external event <x>:
Read the two long word data words from `Ext<x>_Sec10_Usec1_Port` and the `Ext<x>_Year1_Min1_Port` and (if desired) the 100 nsec digit from bits <7:4> of `Ext<x>_Ctl_Sts_Port`.

The time tag data must be read within 90 milliseconds of the occurrence of the external event for the data validity of the year..100msec bits to be assured.

The `t8imono.c`, and `t3itintr.c` demo programs in the appendix demonstrate use of the –TAG3I features.

```
// t8imono.c 4sep04 jck mask off hi nibl of Extn_Ctl_Sts_Port in Rdy test
/* t8imono.c 17jul01 jck convert to TAG8I (individual data regs) */
/* t8mono.c 1apr99 JCK test EXT0..EXT7 time latching by monotonicity */
/* Works with -TAG8I option ONLY. */
/* **** tie all EXTTAG inputs hi or low**** or to MAT (J3 pin 6) */
#include <i86.h>
#include <stdio.h>
#include <conio.h>
#include "t8iregs.h"
union REGS regs;
unsigned long tlo,thi,oldlo,oldhi,oldestlo,oldesthi,subsys_id_vid;
int BASE,instance=0,any_9050_10b5,any_9030_10b5,i,j,charcnt,MAXCHANNEL=7;
char ready;
main()
{
/* USE PCI BIOS CALLS TO DETERMINE MEM BASE ADDRESS */
do  {
    regs.h.ah=0x0b1;
    regs.h.al=0x02;
    regs.w.cx=0x9030;
    regs.w.dx=0x10b5;/* query if device id 9030/vendor id 10b5 is present */
    regs.w.si=instance;     /* search for 0th, 1st etc. */
    int386(0x1a,&regs,&regs);
    printf("9030/10b5 present if 0: %2x bus#: %2x dev/funct: %2x flags: %4x \n",
             regs.h.ah,regs.h.bh,regs.h.bl,regs.w.cflag);
    any_9030_10b5=regs.h.ah;
    if (any_9030_10b5==0)
            {
            regs.h.ah=0x0b1;
            regs.h.al=0x0a;
            regs.w.di=0x2c;  /* query Subsystem ID/ Subsystem Vendor ID*/
            int386(0x1a,&regs,&regs);
            printf("Read Subsystem id/vendor id. OK if 0: %2x Subsystem ID/VID: %8x flags: %4x \
              regs.h.ah,regs.x.ecx,regs.w.cflag);
            subsys_id_vid=regs.x.ecx;
            regs.h.ah=0x0b1;
            regs.h.al=0x0a;
            regs.w.di=0x18;  /* query local address space 0 */
            int386(0x1a,&regs,&regs);
            printf("9030 Read PCI base address. OK if 0: %2x base addr: %8x flags: %4x \n",
              regs.h.ah,regs.x.ecx,regs.w.cflag);
            instance++;      /* if wrong subsys_id_vid, try next */
           }
    } while ((any_9030_10b5==0) && (subsys_id_vid!=0x104210b5));
if ((subsys_id_vid!=0x104210b5) || (any_9030_10b5!=0) )
       { printf ("SYNCCLOCK with PLX PCI9030 not detected on PCI bus\n");
        instance=0;
        do
                {
                regs.h.ah=0x0b1;
                regs.h.al=0x02;
                regs.w.cx=0x9050;
                regs.w.dx=0x10b5;/* query if device id 9050/vendor id 10b5 is present */
                regs.w.si=instance;     /* search for 0th, 1st etc. */
                int386(0x1a,&regs,&regs);
                printf("9050/10b5 present if 0: %2x bus#: %2x dev/funct: %2x flags: %4x \n",
                regs.h.ah,regs.h.bh,regs.h.bl,regs.w.cflag);
                any_9050_10b5=regs.h.ah;
                if (any_9050_10b5==0)
                        {
                        regs.h.ah=0x0b1;
                        regs.h.al=0x0a;
                        regs.w.di=0x2c;  /* query Subsystem ID/ Subsystem Vendor ID*/
                        int386(0x1a,&regs,&regs);
                        printf("Read Subsystem id/vendor id. OK if 0: %2x Subsystem ID/VID: %8x
                        regs.h.ah,regs.x.ecx,regs.w.cflag);
                        subsys_id_vid=regs.x.ecx;
                        regs.h.ah=0x0b1;
                        regs.h.al=0x0a;
                        regs.w.di=0x18;  /* query local address space 0 */
                        int386(0x1a,&regs,&regs);
                        printf("9050 Read PCI base address. OK if 0: %2x base addr: %8x flags: %
                        regs.h.ah,regs.x.ecx,regs.w.cflag);
                        instance++;      /* if wrong subsys_id_vid, try next */
                        }
                } while ((any_9050_10b5==0) && (subsys_id_vid!=0x104210b5));
```

```
        if ((subsys_id_vid!=0x104210b5) || (any_9050_10b5!=0) )
              { printf ("SYNCCLOCK with PLX PCI9050 not detected on PCI bus \n");
               return (0);}
       }
if ((regs.x.ecx&0x01)!=0) {printf(" IO MAPPED - SHOULD BE MEMORY MAPPED \n");return(0);}
BASE=regs.x.ecx&0xfffffffe;
    regs.h.ah=0x08;
    regs.h.al=0x00;
    regs.w.bx=(BASE>>16)&0x0ffff; /*hi 16 bits of phys. addr */
    regs.w.cx=BASE&0x0ffff; /*lo 16 bits of phys addr */
    regs.w.si=0;        /* hi 16 bits of region size */
    regs.w.di=0x20;     /* lo 16 bits of region size */
    int386(0x31,&regs,&regs);    /* map phys addr into cpu addr space */
BASE=(((long)(regs.w.bx))<<16)+(((long)regs.w.cx)&0x0ffff);
printf ("mapped BASE=%08x\n",BASE);
/* IF BOARD IS FOUND, CONTINUE WITH T8IMONO */
*(volatile unsigned char*)(BASE+Match_Msec1_Usec100_Port)=0xff;  /* prevent match toggle */
*(volatile unsigned char*)(BASE+Match_Usec10_Usec1_Port)=0xff;
for (i=0;i<=MAXCHANNEL;i++) /* clear ext evt ready for all 8 ext events*/
      {tlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(8*i));
      }
while (1)
 {
  for (i=0;i<=MAXCHANNEL;i++)
       {
       *(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(4*i))=Trigger_Sim_Ext_Time_Tag;
       ready=*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(4*i));
       if ((ready&0x0f)!=1)
          printf("Ready flag error: Ext%02x_Ctl_Sts_Port low nibble is %02x after sim evt should
                 ,i,ready&0x0f);
       tlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(8*i));
       thi=*(volatile unsigned long*)(BASE+Ext0_Year1_Min1_Port+(8*i));
       ready=*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(4*i));
       if ((ready&0x0f)!=0)
              printf("Ready flag error: Ext%02x_Ctl_Sts_Port is %02x after sim evt should be 0
                 ,i,ready);
       if (  (thi<oldhi) || ((thi==oldhi) && (tlo<=oldlo) ) )
              {printf ("%8x %8x old%8x %8x oldest%8x %8x ch#%01x"
              ,thi,tlo,oldhi,oldlo,oldesthi,oldestlo,i);
              charcnt=0;}
       if ( thi > oldhi )
              {for (j=0;j<=charcnt;j++) printf ("*");
               for (j=0;j<40;j++) printf("\b*"); /* force buffer flush */
               printf ("\r");
               charcnt++;
               if (charcnt >=60){ printf ("\n"); charcnt=0;}
              }
       oldesthi=oldhi;
       oldestlo=oldlo;
       oldhi=thi;
       oldlo=tlo;
      }
 }
 printf (" completed ");
return (0);
}
```

```
//T3ITINTR.C 7JUL06 JCK evt 2 to evt 0 spacing changed to 6 msec
//T3ITINTR.C 11JUN05 JCK FOR -TAG3I (NORDEN) CHANGE MAXCHANNEL:7->2 waitcnt max 10000->100000
/* t8itintr.c 12mar03 JCK BCD version CONNECT TSTOUT<7..0> to EXT<7:0> */
/*b37t8iintr.c 26jun02 JCK. CONNECT TSTOUT<7..0> to EXT<7:0>          */
/*      P8 <21.48,47,45,43,42,41,25> to J3 pin <26,25,24,23,22,21,19,3> */
/*     TST<7..0> generate sliding 1 moving LSB to MSB, so EVTn 1 msec after EVTn-1 */
/* t8iintr.c 20jul01 JCK convert -TAG8 to -TAG8I                       */
/* t8intr.c 2apr99 JCK tests pci-sync32-tag8 intr generation capability */
/* connect all ext time tag inputs hi,lo, or to mat to prevent unwanted tags */
/* THIS DEMO CODE MAY HAVE PROBLEMS WORKING WITH IRQ LEVELS 3 or 5     */
typedef unsigned char byte;
#include "t8iregs.h"  /* offsets for PCI-SYNCCLOCK regs + flag definitions */
#include "picdef.h"  /* addresses for host x86 interrupt ctlrs & vectors */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <dos.h>
#include <conio.h>
#include <i86.h>
union REGS regs;
unsigned long intcount=0,subsys_id_vid;
int BASE,instance=0,any_9050_10b5,any_9030_10b5;
void   interrupt pc_int(void);
void   setvects(void);
void   resvects(void);
void   i_enable(void);
void   i_disable(void);
void (__interrupt __far *oldvec)()=0;
int bin (int value);
volatile long int c;
volatile char readIMR;
volatile unsigned long int busec,oldbusec,oldthi,oldtlo,tlo,thi,waitcnt;
int xnsec100,i,j,ready70,charcnt,MAXCHANNEL=2,error=0,expectedchannel;
int errcnt=0;errchan[10000],intflag=0,passcount=0;
unsigned char Extsts[64];
char IMRMASK,IMRENB[16]={0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
                  0xfe,0xfd,0xfb,0xf7,0xef,0xff,0xff,0x7f};
char VECTOR,IRQVEC[16]={0,0,0,0,0,0,0,0,0x70,0x0A,0x72,0x73,0x74,0,0,0x77};
main()
{/* USE PCI BIOS CALLS TO DETERMINE MEM BASE ADDRESS */
do  {
    regs.h.ah=0x0b1;
    regs.h.al=0x02;
    regs.w.cx=0x9030;
    regs.w.dx=0x10b5;/* query if device id 9030/vendor id 10b5 is present */
    regs.w.si=instance;     /* search for 0th, 1st etc. */
    int386(0x1a,&regs,&regs);
    printf("9030/10b5 present if 0: %2x bus#: %2x dev/funct: %2x flags: %4x \n",
            regs.h.ah,regs.h.bh,regs.h.bl,regs.w.cflag);
    any_9030_10b5=regs.h.ah;
    if (any_9030_10b5==0)
           {
           regs.h.ah=0x0b1;
           regs.h.al=0x0a;
           regs.w.di=0x2c;  /* query Subsystem ID/ Subsystem Vendor ID*/
           int386(0x1a,&regs,&regs);
           printf("Read Subsystem id/vendor id. OK if 0: %2x Subsystem ID/VID: %8x flags: %4x \
             regs.h.ah,regs.x.ecx,regs.w.cflag);
           subsys_id_vid=regs.x.ecx;
           regs.h.ah=0x0b1;
           regs.h.al=0x0a;
           regs.w.di=0x18;  /* query local address space 0 */
           int386(0x1a,&regs,&regs);
           printf("9030 Read PCI base address. OK if 0: %2x base addr: %8x flags: %4x \n",
             regs.h.ah,regs.x.ecx,regs.w.cflag);
           instance++;      /* if wrong subsys_id_vid, try next */
          }
    } while ((any_9030_10b5==0) && (subsys_id_vid!=0x104210b5));
if ((subsys_id_vid!=0x104210b5) || (any_9030_10b5!=0) )
      { printf ("SYNCCLOCK with PLX PCI9030 not detected on PCI bus\n");
       instance=0;
       do
              {
              regs.h.ah=0x0b1;
```

```
                regs.h.al=0x02;
                regs.w.cx=0x9050;
                regs.w.dx=0x10b5;/* query if device id 9050/vendor id 10b5 is present */
                regs.w.si=instance;     /* search for 0th, 1st etc. */
                int386(0x1a,&regs,&regs);
                printf("9050/10b5 present if 0: %2x bus#: %2x dev/funct: %2x flags: %4x \n",
                regs.h.ah,regs.h.bh,regs.h.bl,regs.w.cflag);
                any_9050_10b5=regs.h.ah;
                if (any_9050_10b5==0)
                        {
                        regs.h.ah=0x0b1;
                        regs.h.al=0x0a;
                        regs.w.di=0x2c;  /* query Subsystem ID/ Subsystem Vendor ID*/
                        int386(0x1a,&regs,&regs);
                        printf("Read Subsystem id/vendor id. OK if 0: %2x Subsystem ID/VID: %8x
                        regs.h.ah,regs.x.ecx,regs.w.cflag);
                        subsys_id_vid=regs.x.ecx;
                        regs.h.ah=0x0b1;
                        regs.h.al=0x0a;
                        regs.w.di=0x18;  /* query local address space 0 */
                        int386(0x1a,&regs,&regs);
                        printf("9050 Read PCI base address. OK if 0: %2x base addr: %8x flags: %
                        regs.h.ah,regs.x.ecx,regs.w.cflag);
                        instance++;      /* if wrong subsys_id_vid, try next */
                        }
                } while ((any_9050_10b5==0) && (subsys_id_vid!=0x104210b5));
        if ((subsys_id_vid!=0x104210b5) || (any_9050_10b5!=0) )
                { printf ("SYNCCLOCK with PLX PCI9050 not detected on PCI bus \n");
                return (0);}
        }
if ((regs.x.ecx&0x01)!=0) {printf(" IO MAPPED - SHOULD BE MEMORY MAPPED \n");return(0);}
BASE=regs.x.ecx&0xfffffffe;
regs.h.ah=0x0b1;
regs.h.al=0x0a;
regs.w.di=0x3c;  /* query latency,grant,interrupt pin,interrupt line */
int386(0x1a,&regs,&regs);
printf("Read PCI config reg 3C. OK if 0: %02x config reg3c: %08x flags: %04x \n",
regs.h.ah,regs.x.ecx,regs.w.cflag);
IMRMASK=IMRENB[regs.x.ecx&0x0f];
VECTOR=IRQVEC[regs.x.ecx&0x0f];
printf ("IRQLINE: %02x IMRMASK: %02x IRQVEC[IRQLEVEL]: %02x \n",
       regs.x.ecx&0x0f,IMRMASK, IRQVEC[regs.x.ecx&0x0f]);
     regs.h.ah=0x08;
     regs.h.al=0x00;
     regs.w.bx=(BASE>>16)&0x0ffff; /*hi 16 bits of phys. addr */
     regs.w.cx=BASE&0x0ffff; /*lo 16 bits of phys addr */
     regs.w.si=0;          /* hi 16 bits of region size */
     regs.w.di=0x20;       /* lo 16 bits of region size */
     int386(0x31,&regs,&regs);     /* map phys addr into cpu addr space */
BASE=(((long)(regs.w.bx))<<16)+(((long)regs.w.cx)&0x0ffff);
printf ("mapped BASE=%08x\n",BASE);
*(volatile unsigned char*)(BASE+Control_Port)=0;                    /* disable hb,ramfifo,mat,cmd rd
*(volatile unsigned char*)(BASE+Match_Usec10_Usec1_Port)=0x00 ;
*(volatile unsigned char*)(BASE+Match_Msec1_Usec100_Port)=0x10 ; // 100pps pulse on norden2 will
sleep(1);
// POLL all MAXCHANNEL+1 event inputs to make sure they are occuring
for (i=0;i<=MAXCHANNEL;i++) /* clear ext evt ready for all 8 ext events*/
       {*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(j*4))=0;
        printf (" check channel %d for event input present \n",i);
        tlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(8*i)); // clear Ext_Ready
        while ((*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(i*4)) & Ext_Ready)==0) {;}
       }

printf (" Ext evt 0-2 external event inputs have been detected \n");
setvects();
i_enable();
while (errcnt<20)
{
intcount=0;
for (i=0;i<=MAXCHANNEL;i++) /* clear ext evt ready for all 8 ext events*/
       {tlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(8*i));}
/* while waiting for event 7 clear evt 0..6 ready status */
      if (MAXCHANNEL>0)
         do
         {for (i=0;i<=MAXCHANNEL-1;i++) tlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Por
         while ( ( *(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(MAXCHANNEL*4))&Ext_Ready )
/* read time of event 7 and init oldbusec */
oldtlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(MAXCHANNEL*8));
oldthi=*(volatile unsigned long*)(BASE+Ext0_Year1_Min1_Port +(MAXCHANNEL*8));
oldbusec=bin((oldthi    )&0x0ff)*60000000
```

```
        +bin((oldtlo>>24)&0x0ff)*1000000
        +bin((oldtlo>>16)&0x0ff)*10000
        +bin((oldtlo>>8 )&0x0ff)*100
        +bin((oldtlo    )&0x0ff)*1;
for (i=0;i<=MAXCHANNEL;i++) tlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(8*i));
for (j=0;j<=MAXCHANNEL;j++)
      if ( (*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(j*4))&Ext_Ready)!=0  )
         {printf(" Initial Ext%01x_Ctl_Sts_Port= %02x should be 00 \n",j,
             *(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(j*4)));error=1;}
if (error) return (0);
readIMR = inp(IMR) & IMRMASK;  /* mask bit of "0" enables interrupt */
outp(IMR, readIMR) ;
while ( (((intcount+1)%10000)!=0) && (errcnt<20))
  {
  expectedchannel=intcount%(MAXCHANNEL+1);
  for (j=0;j<=MAXCHANNEL;j++) *(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(j*4))=Ext_Intr_
  waitcnt=0;
  do {readIMR = inp(IMR) & IMRMASK;  /* mask bit of "0" enables interrupt */
      outp(IMR, readIMR) ;
      waitcnt++;} while ((waitcnt<100000) & (intflag==0));
 intflag=0;
               /* &&  (   (*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(expectedchannel*4)
                &&  ( (Extsts[expectedchannel]&Ext_Ready)==0)
                   ); */
   if (waitcnt>=100000)
       {printf(" Ext%01x_Ctl_Sts_Port= %02x should be 01 \n",intcount%8,Extsts[expectedchannel]);
        return(0);}
   for (j=0;j<=MAXCHANNEL;j++)
       {
       if ( (j!=expectedchannel) && ((Extsts[j]&Ext_Ready)!=0)  )errchan[errcnt++]=j;
       Extsts[j]=0;
       }
       busec=bin((thi    )&0x0ff)*60000000
            +bin((tlo>>24)&0x0ff)*1000000
            +bin((tlo>>16)&0x0ff)*10000
            +bin((tlo>>8 )&0x0ff)*100
            +bin((tlo    )&0x0ff)*1;
     if (
        ((expectedchannel!=0) && (  ( (busec-oldbusec)>1001) || ( (busec-oldbusec)<999 ) ))
        ||
        ((expectedchannel==0) && (  ( (busec-oldbusec)>6001) || ( (busec-oldbusec)<5999 ) ))
        )
        {
       printf ("tlo=%08lx oldtlo=%08x delta:%d intcount:%d \n",tlo,oldtlo,busec-oldbusec,intcoun
       return(0);
        }
  oldtlo=tlo;
  oldbusec=busec;
  intcount++;
  }
for (i=0;i<=MAXCHANNEL;i++) *(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(i*4))=0; /* disab
printf ("intcount=%lu errcnt=%d passcount=%d \n",intcount,errcnt,passcount);
passcount++;
if (errcnt!=0)
        {
        for (i=0;i<errcnt;i++) printf("%01x ",errchan[i]);
        printf ("\n");
        }
for (i=0;i<1000000;i++) j=*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port); /* delay for print
}
resvects();     /* example doesn't call resvects & i_disable */
i_disable();    /* but user code should call them */
return(0);
}
/*  this is the actual interrupt handler code.
It is entered via a VECTORED interrupt.
When the ISA-SYNCCLOCK asserts the configured IRQ signal ,
the bus IRQ signal drives a pin on a 8259 priority interrupt controller on
the host CPU board.

(Actually, the priority interrupt controller is probably embedded inside a
big LSI glue chip - but it still looks like an 8259 to the programmer and
the bus).

The 8259 then in turn - if the interrupt input level is enabled -
will try to interrupt the host x86 processor. When the x86 accepts the
interrupt the 8259 will supply an interrupt vector (VEC_9 in this example)
The x86 will then pick up a FAR pointer from the interrupt vector location
and use it as the address for entering the interrupt routine. */
```

```
void    __interrupt __far pc_int(void) /* Handle external event interrupts */
{   int j,discard;
/* _disable();    */      /* causes compiler to EMIT x86 "CLI" op code */
    intflag=1; /* notify bkgnd program that pci-sync32 ext intr occured */
    for (j=0;j<=MAXCHANNEL;j++) Extsts[j]=*(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(j*4
    tlo=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(8*expectedchannel));
    thi=*(volatile unsigned long*)(BASE+Ext0_Year1_Min1_Port +(8*expectedchannel));
    for (j=0;j<=MAXCHANNEL;j++) *(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(j*4))=0; /* d
    for (j=0;j<=MAXCHANNEL;j++)
        discard=*(volatile unsigned long*)(BASE+Ext0_Sec10_Usec1_Port+(8*j));  /* read selected
    outp(ICR, EOI); /* Signal end of hardware interrupt */
    outp(ICR0_7, EOI); /* Signal end of hardware interrupt */
         _enable();          /* causes compiler to EMIT x86 "STI" op code */
}
void   setvects() /* Change intr vector to point to PC-SYNCCLOCK interrrupt code */
{       if ( !oldvec) {
                oldvec = _dos_getvect(VECTOR); /* save old vector */
                _dos_setvect(VECTOR, pc_int); /* set new vector */
        }
}
void   resvects(void) /* Uninstall our vectors before exiting the program */
{       if (oldvec) _dos_setvect(VECTOR, oldvec);
}

void   i_enable() /* Enable PC-SYNCCLOCK interrupts */
{   _disable();         /* causes compiler to EMIT x86 "CLI" op code */
    c = inp(IMR) & IMRMASK;  /* mask bit of "0" enables interrupt */
   /* outp(IMR, c); */
    _enable();          /* causes compiler to EMIT x86 "STI" op code */
}
void   i_disable(void) /* Turn off PCI-SYNCCLOCK interrupts */
{   int                c;
    _disable();         /* causes compiler to EMIT x86 "CLI" op code */
    c = inp(IMR) | ~IMRMASK; /* set mask bit to "1" to disable */
    outp(IMR, c);
    for (j=0;j<=MAXCHANNEL;j++) *(volatile unsigned char*)(BASE+Ext0_Ctl_Sts_Port+(j*4))=0;
    _enable();          /* causes compiler to EMIT x86 "STI" op code */
}
int bin (int value)
{return (((((value&0x0f0)/16)*10) + (value&0x0f))&0x0ff  );}
```

Manual Addendum for -HQ -100PPSPW422-PTTIIN-NORDEN2 options

The PMC-SYNCCLOCK32 module shipped with this manual has the -HQ, -100PPSPW422, -PTTIIN, and –NORDEN2 options installed.

The -HQRS422 option provides for use of Havequick II serial and 1 PPS time reference per ICD-GPS-060A, The input pin assignments are:

1PPS /GND:	1PPS: J3 pin 14 /GND: J3 pin 32
serial HavequickII /GND:	HQII: J3 pin 17 /GND J3 pin 35

The –100PPSPW422 output option provides 2 100PPS RS422 outputs. The pulse asserts (+ side goes high, - side goes low) on 10 millisecond boundaries of the on-board clock. The pulse de-asserts when the digits programmed by the user into the Match_Usec10_Usec1_Port register and the Match_Msec1_Usec100_Port register match the 1 millisecond through 1 microsecond digits of the on-board clock. So, the pulse width may be varied by changing the match times from Match_Msec1_Usec100_Port=00, Match_Usec10_Usec1_Port=0x01 for a pulse width of 1 microsecond to from Match_Msec1_Usec100_Port=0x99, Match_Usec10_Usec1_Port=0x99 for a pulse width of 1 microsecond for a pulse width of 9999 microseconds. The 100PPS pin assignments are:

100PPS +/- side output 1	+:J3 pin 9	-: J3 pin 27
100PPS +/- side output 2	+:J3 pin 8	-: J3 pin 26

The –PTTIIN option provides for use of PTTI serial (50 bps) and 1 PPS time reference per ICD-GPS-060B, at RS422 levels terminated on-board in 100 ohms. The input pin assignments are:

1PPS +/- side:	+: J3 pin 2	-: J3 pin 20
serial PTTI (50 bps) +/- side:	+: J3 pin 6	-: J3 pin 24

The –NORDEN2 option providies for selection of Havequick II or PTTI 50 bps serial messages as the time message using dual port RAM location DP_GPS_Status (see gpsdef.h). If bit 0 is CLEARED, Havequick II input will be used. If bit 1 is SET serial 50 pps PTTI input will be used. Default on power-on or reset is CLEARED.

Manual Addendum –ITR-CONFCOAT options

The PMC-SYNCCLOCK32 module shipped with this manual has the –ITR and -CONFCOAT options installed.

The –ITR option uses integrated circuits that are rated –40C to +85C.

The –CONFCOAT option is conformal coating.

INTRODUCTION 4

SPECIFICATIONS 5

Physical: 5

Inputs & Outputs (see addenda at front of manual for custom options) 5

Bus Interface (see addenda at front of manual for special options): 6

PCI Configuration Registers 7

INPUT/OUTPUT CONNECTIONS 8

Bezel J1, J2 (SMB): 8

Front Panel J3 (IEEE 1284 type jack): (see addenda at front of manual for custom options) 8

PMC Bus Connector (P1/P2): 9

PMC bus P4: (see manual front addendum for custom options) 10

CONFIGURATION, INSTALLATION & OPERATION 11

Configuration: 11
Modulated Code Input Termination (P6) 11
External Oscillator Input Termination (P7) 11
1 PPS 50 ohm termination (P8) 11

Installation 11
PCI Address Verification (Power PC VME boards) 11
PCI Address/IRQ Configuration Verification (X86 systems) 12
Determining Dual Port Ram Time Code Propagation Delay Setting 12
Determining Dual Port Ram 1PPS Propagation Delay Setting 12

Operation 12
Have Quick 12

LED Status Indication 13

PROGRAMMING 14

Introduction 14

PCI Configuration Registers 14

Accessing PMC-SYNCCLOCK in Windows environment 15
Program Development in Windows environment 15
Reading Current Time (BCD Format) (non -BIN option) 16
Multiple Threads Reading Current Time (BCD Format) 16

Reading Current Time (-BIN option only) 16
Reading External Time Tags (non -BIN option) 17
Reading External Binary Time Tags (-BIN option only) 18
Simulating External Time Tags 18
Heartbeat Pulses and Interrupts 19
Dual Port RAM for Supplementary Status/Control 19
Accessing Dual Port RAM 19
Sending Commands through the Dual Port RAM. 20
Setting Leap Year Control in Dual Port RAM 20
Selecting 1PPS or Time Code Reference Priority in Dual Port RAM 20
Setting Time Code Propagation Delay Polarity in Dual Port RAM 21
Setting PPS Propagation Delay in Dual Port RAM 21
Specifying Modulated Input Code Format in Dual Port RAM 21
Specifying Leap Second Day in Dual Port RAM 21
Setting Time Code Propagation Delay in Dual Port Ram 22
Non-volatile Propagation Delay Settings 22
Setting PPS Propagation Delay in Dual Port Ram 22
Setting Major Time in Dual Port RAM 22
Setting Year in Dual Port RAM 22
Changing Heartbeat, Lowrate and Rate2 23
GPS Satellite Tracking and Navigation Data (except Have-Quick receivers) 24
Differential Data Acknowlege Count (-DF option with Motorola Oncore Only) 25
GPS Satellite Data (Have-Quick receivers) 25

THEORY OF OPERATION **26**

Time Code Stripping **26**
Input Signal Coupling 26
Zero Crossing Detector 26
Voltage Divider 27
Peak Detector 27
Burst Decoder 27

FPGA Clock **27**
200ms .. 8 Year Bits 27
Disciplining 27
Clock conditioning 28
Modulated code generation 28
Error Measurement 28
Match Times 28
3.0 MHz frequency generation 29
Heartbeat Rates 29

Microcomputer **29**
D/A frequency control (DTCXO option) 29
GPS receiver serial interface 29
HAVE QUICK interface option 29

PCI Bus Interface **29**

BOARD LAYOUT **31**

SCHEMATIC DRAWINGS 32

PARTS LIST 36

PROGRAM LISTINGS 38

Introduction

The PMC-SYNCCLOCK32-UNIV is a digital clock with a PMC (mezzanine PCI) bus interface that will automatically synchronize to time reference signals. The "UNIV" designates its' ability to function in systems with 3.3V or 5V PCI bus signaling levels. It is software compatible with older PMC-SYNCCLOCK32-UNIV models that were 5V signaling only. Programs running on the PMC host can read the clock time directly from longword registers. In many applications, no other user programming is needed. The advantages of the PMC-SYNCCLOCK32's digital clock over the host system "real time" clock include better resolution (100 nsec vs. msec or worse), easier access (single C statement vs. DOS system call) and - most importantly - stability and synchronization to Universal Coordinated Time by locking to time reference signals.

The time reference signals can be standardized modulated "time code" signals (IRIG-B, IRIG-A, NASA-36, 2137 or XR3). Or, the reference can be a 1 PPS time pulse from an external GPS or LORAN receiver. Even though the time reference occurs as slowly as once per second, the on-board clock maintains specified accuracy and resolution by using a phase locked 10 MHz crystal oscillator. The user program can specify the propagation delay from the time code source to the PMC or the 1 PPS source to the PMC-SYNCCLOCK32-UNIV input and the PMC-SYNCCLOCK32-UNIV will automatically adjust the clock to compensate for the delay. A redundancy feature allows BOTH a 1 PPS signal and a time code to be used as references. The user can specify which reference is preferred. If the preferred reference is invalid, the secondary reference is used until the primary becomes valid again. A user-supplied military GPS receiver (the "Plugger" for example) with "Have Quick" output may be used as the time reference (-HQ option)

In addition to the registers that latch the time when a user program reads the clock, a second set of time registers is used to accurately capture the time of external "time tag" pulses (with 100 nanosecond resolution). Because external pulses may occur at unpredictable times a PCI bus interrupt may be enabled by the user to signal the occurrence of an external event and cause the captured time to be stored in host memory, With modern host processor speeds, external pulse spacing as low as a few μseconds may be supported.

Two user selected pulse rates (one from 1.5 M PPS to 45.77 PPS and one configurable 50 PPS to .00153 PPS or 1.5 M PPS to 45.77 PPS) are locked to the clock frequency. The 1.5 M PPS to 45.77 PPS rate can cause "heartbeat" interrupts to periodically wake up host processes. The user also has direct access to a comparator for generating "match" pulses or toggles at precise times.

The PMC-SYNCCLOCK32-UNIV is designed so that the most common user applications require the least programming. For example, reading current time can be accomplished with a single C statement. Data and parameters that need fast access and update are assigned individual registers directly accessible over the PMC bus, while a much larger number of infrequently (if ever) accessed status and control parameters are accessed by a Dual Port Ram occupying only two PMC bus addresses.

For Windows environments (NT 4.0, 2000, 98, XP), drivers and access for all registers is provided using a virtual address pointer to the base address of the registers. This software links to a library provided by PLX Technology who makes the PCI9030 bridge device that sits between the PCI bus and the local bus to which the clock registers are connected. Use of the PLX software is subject to the following restriction:

Copyright ©2001 PLX Technology, Inc. PLX Technology Inc. licenses this software under specific terms and conditions. Use of any of the software or derivatives thereof in any product without a PLX Technology chip is strictly prohibited.

The PMC-SYNCCLOCK32-UNIV uses a PLX Technology chip, so the licensing requirement is satisfied. Sample "C" source Windows programs and definition files are supplied to speed programming operations. Users who require interrupt support in Windows must purchase a development seat of WinIntr software. WinIntr includes the PLX SDKPro software package along with sample and demo applications using interrupts in the Windows environment.

SPECIFICATIONS

Physical:

Dimensions:	74 mm X 149 mm ("Single CMC")
Power:	+5V ±5% .12A typ, .15A max
	+3.3V±5% .12A max
	+12V± 5%: .06A typ .10A max
	-12V± 5%: .025A typ .05A max
Fabrication:	1.68mm±.2mm (.062 in±008in)
	FR4, 6 layer
Temperature:	0 to 70C operating, -40C to 85C storage
	-40C to +85C operating with –ITR option
Humidity:	0 to 99% non-condensing

Inputs & Outputs (see addenda at front of manual for custom options)

Inputs:	
Multipin Input/output connector:	J3 Hi Density 36 pin IEEE1284 type. Mating connector is Molex 52316-3611. See input/output pin assignment section for recommendations on mating pre-assembled cables
Coaxial time code input/output connectors	Right angle SMB
Code input types:	Modulated IRIG-A,B,NASA36.XR3,2137 and IRIG-G optional
Code input amplitude:	250 mV to 10V pk/pk
Code input modulation ratio:	2:1 to 4:1
Code input frequency error:	100PPM maximum
Code input impedance:	>10K ohm
Code Sync accuracy:	1µsec
1PPS Sync accuracy:	100 nsec
External event pulse input:	↑ or ↓ edge TTL
Minimum event to event spacing (host interrupt mode):	Limited only by host speed
1PPS input:	TTL level positive edge
Have Quick input:	per ICD-GPS-060
Outputs:	
Modulated IRIG-B (option) :	2.5V±10% pk/pk into 600Ω
Match:	TTL pulse
Status (Red LED):	Flashes coded patterns
Sync Indicator (Green LED):	Lights if SYNCOK status is set

Bus Interface (see addenda at front of manual for special options):

Addressing: PCI memory space (8 contiguous longwords)
Interrupt levels: INTA

Register Assignments:

Offset from Base Addr	Bits Bit 0 is LSB	Read Usage	Write Usage
0	<31:0>	Zero latency 40 sec .. 1 μsec BCD time (BCD) μsec<31:0> for -BIN option	Read Only
4	<31:0>	Zero latency 8 year .. 1 minute BCD time. For -BIN option: <31:22>=0,<21:14> binary year of century, <13:5> binary day of year, <4:0> = μsec<36:32>	Read Only
8	<31:0>	External time tag 40 sec .. 1 μsec BCD time. For -BIN option , μsec<31:0>	Read Only
C	<31:0>	External time tag 8 year .. 1 minute BCD time. For -BIN option: <31:22>=0,<21:14> binary year of century, <13:5> binary day of year, <4:0> = μsec<36:32>	Read Only
10	0	1=External time tag data ready	"1" simulates external time tag
	1	1=Command Response Ready	Read Only
	2	n/u	n/u
	3	Match Flag	"1" clears match flag
	4	Heartbeat Flag	Read Only
	5	External Event Polarity 0=↑, 1=↓edge	Same as read
	6	In-Sync Flag	Read Only
11	0	External time tag data ready interrupt enable	1= enable, 0=disable
	1	CMD response ready interrupt enable	1 = enable, 0=disable
	2	N/U	N/U
	3	Time Match flag interrupt enable	1 = enable, 0=disable
	4	Heartbeat pulse interrupt enable	1 = enable, 0=disable
	5	Time Match Extended toggle enable	1 = enable, 0=disable
	6	n/u	n/u
	7	n/u	n/u
14	<7:0>	Write Only	Match Time 10 μsec, 1 μsec digits
15	<7:0>	Write Only	Match Time 1 msec, 100 μsec digits
1A	<7:0>	Write Only	Dual Port RAM Address
1E	<7:0>	Dual Port RAM Data	Same as read

PCI Configuration Registers

Default values in *italics* EEPROM initialization values in **bold**

PCI CFG register address	Bits <31:24>	Bits <23:16>	Bits <15:8>	Bits <7:0>	PCI Writeable	EEPROM Writeable
00h	Device ID			Vendor ID	N	Y
	9030			**10B5**		
04h	Status			Command	Y	N
	0002			*0000*		
08h	Class Code			Rev. ID	N	Y(<31:8>)
	0680			**00***01*		
0ch	BIST	Header Type	Latency Timer	Cache Line	Y (<7:0>)	N
	00	*00*	*00*	*00*		
10h	PCI Base Addr 0 for Memory Mapped Local Bus Config Registers				Y	N
	00	*00*	*00*	*00*		
14h	PCI Base Address 1 for I/O Mapped Local Bus Config Registers				Y	N
	00	*00*	*00*	*00*		
18h	PCI Base Address 2 for Local Bus Address Space 0				Y	N
	00	*00*	*00*	*00*		
1ch	PCI Base Address 3 for Local Bus Address Space 1 (not used)				Y	N
	00	*00*	*00*	*00*		
20h	PCI Base Address 4 for Local Bus Address Space 2 (not used)				Y	N
	00	*00*	*00*	*00*		
24h	PCI Base Address 5 for Local Bus Address Space 3 (not used)				Y	N
	00	*00*	*00*	*00*		
28h	Cardbus CIS Pointer (Not Supported)				N	N
	00	*00*	*00*	*00*		
2ch	Subsystem ID		Subsystem Vendor ID		N	Y
	1042		**10B5**			
30h	PCI Base Address for Local Expansion ROM (not used)				Y	N
	00	*00*	*00*	*00*		
34h	reserved				N	N
38h	reserved				N	N
3C	Max Latency	Min Grant	Interrupt Pin	Interrupt	Y (<7:0>)	Y (<15:8>)
	00	00	01	00		

INPUT/OUTPUT CONNECTIONS

Bezel J1, J2 (SMB):

J1 is used for modulated time code input.
J2 (optional) is used for modulated time code output (option).

Front Panel J3 (IEEE 1284 type jack): (see addenda at front of manual for custom options)

Pin 1: Time Code Input ("+" side if balanced input)
Pin 2: + Side of RS422 1 PPS input from External GPS Receiver
Pin 3: Ground
Pin 4: + Side of RS422 Serial Data Output to External GPS Receiver
Pin 5: + Side of RS422 Serial Data Input from External GPS Receiver
Pin 6: + Side of RS422 Serial RTCM104 Serial Differential Data (-DF option only)
Pin 7: + Side of RS422 DC level shift time code input (-DCCODE options only)
Pin 8: + Side of RS422 DC level shift time code output
Pin 9: + Side of Auxiliary RS422 output (special options only)
Pin 10: Time tag input
Pin 11: CLK0: Low rate (50 PPS .. .00153 PPS) user programmable pulse output
Pin 12: CLK2: High rate (1.5 M PPS to 45.7 PPS) user programmable pulse output
Pin 13: CLK1: HEARTBEAT user programmable pulse output
Pin 14: Have Quick 1 PPS in. Have Quick 1 PPS in. P8 configures 50 ohm termination. Diode clamped thru 75ohms to +5V.(If using with PLUGGER, connect to pin 6 of Plugger)
Pin 16: Optional modulated time code output
Pin 17: Have Quick Serial Data In (option). (If using Plugger, connect to pin 7 of Plugger.)
Pin 18: External oscillator sine wave input (option)
Pin 19: Time Code Input Ground (or "-" side if balanced input)
Pin 20: - Side of RS422 1 PPS input from External GPS Receiver
Pin 21: Ground
Pin 22: - Side of RS422 Serial Data Output to External GPS Receiver
Pin 23: - Side of RS422 Serial Data Input from External GPS Receiver
Pin 24: - Side of RS422 Serial RTCM104 Serial Differential Data (-DF option only)
Pin 25: - Side of RS422 DC level shift time code input (-DCCODE options only)
Pin 26: + Side of RS422 DC level shift time code output
Pin 27: + Side of Auxiliary RS422 output (special options only)
Pin 29: "REF" unprotected TTL level 1PPS input. Use of pin 14 is recommended.
Pin 31: "MATCH" pulse output
Pin 32: Ground
Pin 34: Ground
Pin 36: Ground
Unassigned pins are reserved

Because the mating connector for J3 requires expensive tooling, most users needing J3 interface by using prefabricated cable assemblies that mate with J3 and breaking out wire pairs for their application.
L-COM has cable assemblies (CSM84CC series) with mating connectors at each end that may be cut in half to make two mating cables. The pin assignments on J3 line up with CSM84CC twisted pair assignments. Each row of the table below corresponds to one wire pair:

Pinouts for L-COM CSM84CC-1M,-2M,-3M,-5M,-10M cables:

J3 signal	wire color	J3 pin	J3 pin	wire color	J3 signal
CODEIN+	BLU	1	19	BLU/WHT	CODEIN-
PPSIN+	YEL	2	20	YEL/BLK	PPSIN-
GND	BRN	3	21	BRN/WHT	GND
GPSRXI-	RED	4	22	RED/BLK	GPSRXI-
GPSRXO+	GRN	5	23	GRN/BLK	GPSRXO-
RTCMIN+	ORG	6	24	ORG/BLK	RTCMIN-
DCCODEIN+	PNK/RED	7	25	PNK/BLU	DCCODEIN-
DCCODEO UT+	GRY/RED	8	26	GRY/BLU	DCCODEOUT-
TXDA+	LTGR/BLU	9	27	GRN/WHT	TXDA-
EXTEVT	YEL/RED	10	28	YEL/BLU	SPARE196
CLK0	RED/BLU	11	29	RED/WHT	REF
CLK2	WHT/RED	12	30	WHT/BLU	SPARE202
HBPULSEL	LTGRN	13	31	GRN/BLK	MAT
PTTI1PPS	PINK	14	32	PNK/BLK	GND
SPARE95	GRY	15	33	GRY/BLK	SPARE172
CODEOUT	VIO	16	34	VIO/WHT	GND
HAVEQSER	LTBLU	17	35	LTBLU/BLK	SPARE179
SINECLK	WHT	18	36	WHT/BLK	GND

PMC Bus Connector (P1/P2):

(per PMC specification)

PMC bus P4: (see manual front addendum for custom options)

Pin 1: CLK1: HEARTBEAT user programmable pulse output
Pin 3: CLK0: Low rate (50 PPS .. .00153 PPS) user programmable pulse output
Pin 5: CLK2: High rate (1.5 M PPS to 45.7 PPS) user programmable pulse output Pin 6:
Pin 7: Time tag input TTL level
Pin 9: "REF" unprotected TTL level 1PPS input.
DO NOT CONNECT ICD-060 (Havequick) 1PPS SIGNAL TO PIN 9. (See pin 60).
Pin 11: Time tag 5 input TTL level (-TAG8I OR -FIF8 option only)
Pin 12: Time Code Input Ground (or "-" side if balanced input)
Pin 13: "MATCH" pulse output
Pin 15: Time tag 1 input TTL level (-TAG8I OR -FIF8 option only)
Pin 17: Optional external 10 MHz input
Pin 19: Time tag 4 input TTL level (-TAG8I OR -FIF8 option only)
Pin 20: Time Code Input ("+" side if balanced input)
Pin 21: Time tag 6 input TTL level (-TAG8I OR -FIF8 option only)
Pin 25: Have Quick Serial Data In (-HQ option). (If using PLGR, connect to pin 7 of PLGR.)
Pin 27: Ground
Pin 33: Time tag 3 input TTL level (-TAG8I OR -FIF8 option only)
Pin 34: Ground
Pin 36: Low source impedance pulse output A. Defaults to DC level shift IRIG-B +5/0V levels.
Pin 44: Low source impedance pulse output A. Defaults to 1PPS 20μsec width +5/0V levels.
Pin 45: Time tag 2 input TTL level (-TAG8I OR -FIF8 option only)
Pin 48: Modulated time code output (-BOUT or -AOUT or –GOUT option)
Pin 49: + Side of RS422 I/O A direction and use is option dependent
Pin 50: Ground
Pin 51: - Side of RS422 I/O A direction and use is option dependent
Pin 55: Time tag 7 input TTL level (-TAG8I OR -FIF8 option only)
Pin 57: + Side of RS422 I/O C direction and use is option dependent
Pin 58: 51 ohm termination resistor to ground
Pin 59: - Side of RS422 I/O C direction and use is option dependent
Pin 60: PTTI 1PPS input. Jumper to pin 58 for 50 ohm termination
Pin 61: + Side of RS422 I/O D direction and use is option dependent
Pin 62: + Side of RS422 I/O B direction and use is option dependent
Pin 63: - Side of RS422 I/O D direction and use is option dependent
Pin 64: - Side of RS422 I/O B direction and use is option dependent

CONFIGURATION, INSTALLATION & OPERATION

Configuration:

The user should check that the configuration of PMC-SYNCCLOCK32-UNIV matches the configuration that is required by the user application. Factory default configurations are shown below.

Modulated Code Input Termination (P6)

Installation of a push on jumper on P6 will terminate the modulated code input with a 51 ohm resistor. Factory default configuration is NOT installed.

External Oscillator Input Termination (P7)

If XO1 is not installed, an external 10 MHZ oscillator input should be connected to J3 or P4. Installation of a push on jumper on P7 will terminate the external oscillator input with a 51 ohm resistor. P7 jumper should not be installed if XO1 is installed.

1 PPS 50 ohm termination (P8)

Installation of a push on jumper on P8 will terminate the 1 PPS (J3 pin 14 or P4 pin 60) input with a 51 ohm resistor. Or, P4 pin 60 may be connected to P4 pin 58 for termination.Factory default configuration is not terminated.

Installation

Double check and record configuration settings before installing the PMC-SYNCCLOCK32. The PMC host system should be powered down before the PMC-SYNCCLOCK32-UNIV is added. Plug the PMC-SYNCCLOCK32-UNIV into the appropriate location in the PMC host.

Attach inputs and outputs before applying power .
Apply power.
Observe LED1 (red) flashing pattern and verify appropriate status per operation notes below.

PCI Address Verification (Power PC VME boards)

In Power PC based systems with a PMC slot, the boot ROM should set PMC-SYNCCLOCK32-UNIV configuration register 18 (PCI base address 2 /Local Bus Address Space 0) to a base PCI bus memory space address in the PCI address space. The base address of the PCI memory space in Power PC address space should be added to this to obtain the base address of the PMC-SYNCCLOCK32-UNIV in Power PC address space. The base address in POWER-PC address space is added to the offsets defined in the `regdef.h` file to generate PCI bus addresses corresponding to PMC-SYNCCLOCK32-UNIV registers.

Example: The MVME1600-001 assigns POWER PC address range 80810000 to 808100FF to the 256 byte PMC slot PCI bus contiguration registers. So, configuration register 18 is located at POWER PC address 80810018. When the PMC-SYNCCLOCK32-UNIV is installed in a MVME1600-001, say location 80810018 contains "00020101" as the PCI memory space base address assigned to the PMC-SYNCCLOCK32. "00020101" is in reversed byte order because the

PMC-SYNCCLOCK32-UNIV is Big Endian while the PCI bus is little Endian. So, the PCI memory space base address is 01010200. Now, the PCI I/O address range 0000000-3EFFFFFis mapped to MVME 1600-001 address space C0000000-FEFFFFF.
So in this example, the PMC-SYNCCLOCK32-UNIV base address is C1010200 in MVME1600-001 POWER-PC address space. A "MD C1010200" debugger command will display time of day in two 32 bit words (byte reversed because of Big Endian/Little Endian shuffling).

PCI Address/IRQ Configuration Verification (X86 systems)

In x86 based systems with a PCI bus, the PCI BIOS will set PMC-SYNCCLOCK32-UNIV configuration register 18 (PCI base address 2 /Local Bus Address Space 0) to a base PCI bus I/O address in the PCI address space . This value is added to the offsets defined in the `regdef.h` file to generate PCI bus addresses corresponding to PMC-SYNCCLOCK32-UNIV registers.

Determining Dual Port Ram Time Code Propagation Delay Setting

For achieving μsecond accuracy with time code inputs, it is necessary for the user to determine the propagation delay correction for each PMC-SYNCCLOCK32-UNIV installation. The correct values for propagation delay correction can be estimated base on cable lengths and propagation speed of the cable used, but the effects of capacitive loading usually require that the 10s and units of μsecond digits and below be determined experimentally. Using an approximate value, observe a J3 output that should be "on-time" (for example the factory default heartbeat rate on J3 pin 8 is 1000 PPS ↓ edge on-time). Using an oscilloscope, compare the on-time edge with a reference, for example the 1 PPS output of a portable standard or GPS receiver. Another method (if the source is fairly close) is to run **identical length** cables from a pulse output of the time code source and from the pulse output of the PMC -SYNCCLOCK32 to the oscilloscope. Adjust the propagation delay settings until the pulses are coincident. Once set, the values are valid unless any changes are made in the connections between the time code source and the PMC-SYNCCLOCK32.

Determining Dual Port Ram 1PPS Propagation Delay Setting

For 100 nsec accuracy with the on-board GPS antenna or with an external 1PPS input, it is necessary for the user to set the propagation delay correction for each PMC-SYNCCLOCK32-UNIV installation. For an external 1 PPS input use a procedure similar to determining the Code Propagation Delay setting. Adjust the value for any source 1 PPS error. For on-board GPS receiver, you can simply set a value based on the antenna cable length.

Operation

The PMC-SYNCCLOCK32-UNIV will start operation automatically when the PCI bus RST# signal goes low to high. No manual operator intervention is possible or required.

Have Quick

If using a user supplied GPS receiver with HaveQuick output, be sure the receiver setup is configured for Continuous operation with serial output set to STANDARD, 1 PPS set to UTC, and HAVEQUICK set to ON. If your application is a fixed location, setting the GPS receiver to fixed position mode will improve reliability of receiver operation because only a single GPS satellite needs to be in view.

LED Status Indication

LED1 (red LED) flashes echo the data in the `DP_Extd_Sts` dual port RAM location. The LSB (`DP_Extd_Sts_Nosync`) is sent first. "0" bits are short flashes, "1" bits are long flashes. Trailing short flashes are deleted. For example, if `DP_Extd_Sts` is 1C, the pattern is short short long long long.

red LED	Short	Long
First flash	Synchronization Verified	Synchronization NOT Verified
Second flash	Time Code Input Readable	Time Code Input NOT Readable
Third flash	1 PPS Verified	1 PPS NOT Verified
Fourth flash	Major Time has been SET after last JAM synchronization	Major Time has NOT been set after last jam synchronization
Fifth flash	Year valid	Year invalid

LED2 (green LED) lights when synchronization has been verified during the last 5 seconds. If a good time code or GPS signal is applied the green LED should light a few seconds later and remain lit after a few more seconds.

PROGRAMMING

Introduction

The PMC-SYNCCLOCK32-UNIV is designed so that the most common user applications require the least programming. For example, reading 64 bits of current time can be accomplished with two C statements. Data and parameters that need fast access and update are assigned individual registers directly accessible over the PCI bus, while a much larger number of infrequently (if ever) accessed status and control parameters are accessed by a Dual Port Ram occupying only two PCI bus addresses. The `DPDEF.H` file contains the Dual Port RAM assignments. The `REGDEF.H` file contains the offset addresses and bit assignments for registers.

Note that all accesses to the PMC-SYNCCLOCK32-UNIV are relative to a BASE address, which varies depending on the user environment.

Windows (NT 4.0, 2000, 98, XP) environments:
Drivers are furnished with the board. The `SelectSyncclock` subroutine provides users a user space virtual address pointer to the base of the PMC-SYNCCLOCK timing registers. An application for automatically setting the system time periodically to the SYNCCLOCK time is also supplied.
Users who require interrupt support in Windows must purchase a development seat of WinIntr software. WinIntr includes the PLX SDKPro software package along with sample and demo applications using interrupts in the Windows environment.

The PMC-SYNCCLOCK power-on configuration EEPROM has a bit that selects whether the PMC-SYNCCLOCK registers are in memory or I/O space. Some older PMC-SYNCCLOCK units had EEPROMS programmed to request that the PMC-SYNCCLOCK registers be in I/O space. The I/O space EEPROMS must be replaced with memory mapped EEPROM. Or, a DOS utility from PLX can be used to reprogram the EEPROM.

Sample programs in the appendices show using the `SelectSyncclock` for:

1. Reading Current Time: (`curtime.c`)
2. Reading Dual Port RAM (`dpdump.c`)
3. Using the Match register to generate Match pulses at precise times: (`mattst.c`)
4. Using the Dual Port RAM to initialize Major Time for timing using only 1 PPS time reference (`setmajor.c`).
5. Using the Dual Port RAM to specify a Heartbeat Rate (`heart.c`).

PCI Configuration Registers

The host system BIOS or operating system will normally detect the presence of the PMC-SYNCCLOCK32-UNIV and assign a base PCI bus memory address. In Windows environments, the user does not need to access the PCI configuration registers. For non Windows/Linux environments, the base address may be determined by:

1. User programs using PCI BIOS calls (in a 32 bit DOS environment).
2. User OS calls (non "x86" environment).

Accessing PMC-SYNCCLOCK in Windows environment

In a Windows environment, a thread calls `SelectSyncclock` once to get the virtual base address of the PMC-SYNCCLOCK in user space:

```
        unsigned long BASE=-1;
         HANDLE Handle;
 if (SelectSyncclock(&Handle,&BASE)!=1)
          {printf("*ERROR* Could not get PMC-SYNCCLOCK base address"};
       /* if successful BASE = base virtual address of PMC-SYNCCLOCK registers */
```

The SelectSyncclock subroutine links to `PlxApi.lib` for calls to `PlxApi.dll` to get a user space virtual base address that points to the PMC-SYNCclock time registers. `SelectSyncclock` checks the subsystem vendor ID and subsystem ID to be sure that the SYNCCLOCK board is being accessed. The `&Handle` parameter is only used for applications that use interrupts from the PMC-SYNCCLOCK board.

Before exiting, the thread should release the virtual base address window with the following statement:

```
DeselectSyncclock(&Handle);
```

Program Development in Windows environment

Visual C++ V6.0 is the programming environment for using `SelectSyncclock` and `PlxApi.lib/PlxApi.dll`. The following hints/suggestions apply:

1. In Project Settings C/C++:
   Precompiled Headers should be set to:Not using precompiled headers.

2. 4 different versions of `SelectSyncclock` are supplied:
   A. `\Release\c\SelectSyncclockc.lib and .obj` is supplied for use with C (user .c source file) projects with Build Set Active Configuration set for Win32 Release
   A. `\Release\cpp\SelectSyncclockcpp.lib and .obj` is supplied for use with C++ (user .cpp source file) projects with Build Set Active Configuration set for Win32 Release
   A. `\Debug\c\SelectSyncclockc.lib and .obj` is supplied for use with C (user .c source file) projects with Build Set Active Configuration set for Win32 Debug
   A. `\Debug\cpp\SelectSyncclockcpp.lib and .obj` is supplied for use with C++ (user .cpp source file) projects with Build Set Active Configuration set for Win32 Debug

To select the correct library, on the C/C++ menu bar, select Project then select Add to Project then select Files and then browse for *.lib files and click to select the correct library.

3. 2 different versions of `PlxApi.lib/PlxApi.dll` are supplied:
   A. `\Release\PlxApi\PlxApi.lib` and `\Release\PlxApi\PlxApi.dll` are supplied for use with projects with Build Set Active Configuration set for Win32 Release
   B `\Debug\PlxApi\PlxApi.lib` and `\Debug\PlxApi\PlxApi.dll` are supplied for use with projects with Build Set Active Configuration set for Win32 Debug

To select the correct library, on the C/C++ menu bar, select Project then select Add to Project then select Files and then browse for *.lib files and click to select the correct library.

4. Use the preamble of one of the included sample programs as a starting point.

Reading Current Time (BCD Format) (non -BIN option)

The first read operation (from register `BASE+Sec10_Usec1_Port` latches all the other current time bits at same time. The clock continues to run and the LATCHED data for all the other digits may be read across the PCI bus by a following read. So, `BASE+Sec10_Usec1_Port` must always be read before the other latched time ports are read - even if the user is uninterested in the low order digits. The high order latched time data from `BASE+Year1_Min1_Port` then may be read or may be ignored to reduce the amount of stored data or to speed up time reading. The 1 μsecond through 100 Millisecond bits remain latched until `BASE+Sec10_Usec1_Port` is read again. The 200 millisecond through 8 years bits remain latched for at least 90 milliseconds after `BASE+Sec10_Usec1_Port` is read.

Example: (see `curtime.c` sample code) Reads all 8 bytes of current time with two C statement that takes about 2 usec to execute on a 350 MHz AMD K6 / Windows 98.

```
timelo=*(volatile unsigned long*)(BASE+Sec10_Usec1_Port);
timehi=*(volatile unsigned long*)(BASE+Year1_Min1_Port);
```

Multiple Threads Reading Current Time (BCD Format)

The first read operation (from register `BASE+Sec10_Usec1_Port` latches the high order time digits in `BASE+Year1_Min1_Port`. Some applications my have multiple threads reading current time, leading to a possibility of the following scenario:

1. Say for example that the time is years (ls digit):3 years: 012 days 21 hours: **45** minutes 59 seconds 999123 microseconds. Thread A reads `timelo` from `BASE+Sec10_Usec1_Port (0x59999123)`, but is suspended before reading `BASE+Year1_Min1_Port.`
2. A new minute occurs. Time is now 3 years 012 days 22 hours 46 minutes 0 seconds
3. Thread B reads `BASE+Sec10_Usec1_Port` causing new `BASE+Year1_Min1_Port` value to be latched
4. Some time later, Thread A reads `timehi from BASE+Year1_Min1_Port` which now contains 3 years 012 days 21 hours **46** minutes. When combined with the `timelo` value stored before thread A was suspended, the resulting time is about 1 second fast.

The fastest solution is for each thread to read both time ports twice and
1. Compare the `BASE+Year1_Min1_Port` values read.

A. If the second `BASE+Year1_Min1_Port` value read is higher, then the first set of values should be used (in case the scenario happened between the 2 register reads for the second set).

B. If the second `BASE+Year1_Min1_Port` value read is equal, use the second set in case the scenario occurred between the 2 register reads of the first set.

C, If the second `BASE+Year1_Min1_Port` value read is lower, then an end-of-decade rollover has happened and the second set should be used.

See `setsystemtime.c` for an example using this procedure

Another solution is to use a `mutex` , but the re-reading solution is probably faster.

Reading Current Time (-BIN option only)

The same C statements used for a PMC-SYNCCLOCK with the standard BCD format is used for binary format. (See "Reading Current Time (BCD Format) (non -BIN option)") However, the data format of the 2 32 bit time registers is different:

Example: Reads all 8 bytes of current binary time with two C statements that takes about 2 usec to execute on a 350 MHz AMD K6 / Windows 98

```
/* read 2**31 (msb) to 2**0 usec of day bits first */
timelo=*(volatile unsigned long*)(BASE+Sec10_Usec1_Port);
/* then read
   BCD year of century (2**21..2**14 bits
   binary day of year (2**13..2**5 bits)
   2**36..2**32 usec of day (2**4..2**0 bits)
   2**31..2**22 bits of exttimehi are not defined */
timehi=*(volatile unsigned long*)(BASE+Year1_Min1_Port);
```

The same considerations for multiple threads accessing current time apply for –BIN format as for the standard BCD format. The potential scenario can only occur every 4096 seconds with –BIN format instead of every 60 seconds with BCD format.

Reading Status from Status_Port

The `Status_Port` is read with a single C statement:

```
Status=*(volatile unsigned char*)(BASE+Status_Port);
```

Status bit 0: `Ext_Ready` bit (READ ONLY) will be "1" if an external time tag pulse has occurred since the last read of any of the External Time Tag data registers. It will be cleared when any of the External Time Tag data registers are read across the PCI bus. The programmer can simulate the occurrence of an external time tag pulse by writing a 1 to this bit, which will cause the current time to be latched in the `BASE+Ext_Sec10_Usec1` and `BASE+Ext_Year1_Min1` registers.

Status bit 1: The `Response_Ready` bit is set when the on-board microprocessor responds to commands (commands are writes to Dual Port RAM location `DP_Command`).

Status bit 3: The `Match` bit (READ ONLY) toggles when the time programmed into `Match_Usec10_Usec1_Port` and `Match_Msec1_Usec100_Port` registers match the clock time. If the `Time Match Extended` bit in the `Control Port` is set (enabled), the dual port RAM locations `DP_Match_SS` and `DP_Match_ht` must match the current seconds and 10^5 and 10^4 μsecond digits (or be 0xFF if don't cares). **When Time Match Extended is enabled, the desire match time must be specified at least 10 milliseconds before the match is to occur**. The `Match` latch can be cleared by writing `Reset_Match` to the `Status_Port`.

Status bit 4: The `Heartbeat` bit (READ ONLY) is set when a heartbeat pulse occurs at the HBPULSEL output of J3. The heartbeat is generated by dividing a 3 MHz (or 100PPS for alternate P13 configuration) signal by a user programmed divisor of 2 to 65535 (default divisor is 3000 for 1K PPS heartbeat). The `Heartbeat` latch is cleared by writing `Reset_Heartbeat` to `BASE+Status_Port`.

Status bit 5 The `Ext_Polarity` status bit (READ/WRITE) will be 0 for rising edge external time tags and 1 for falling edge external time tags.

Status bit 6 The `Sync_OK` status bit (READ ONLY) will be "1" if the PCI clock is synchronized to a time reference input. More extensive time synchronization status information can be read from the dual port RAM `DP_Extd_Sts` location.

Reading External Time Tags (non -BIN option)

The simplest and fastest way to read external time tags uses the PMC-SYNCCLOCK32-UNIV external time tag latches which may be read directly across the PCI bus:

Example: Reads all 8 bytes of external BCD time tag with two C statements that take about 2 usec to execute on a 350 MHz AMD K6 / Windows 98. A third C statement can be used to read 100s of nanoseconds digit.

```
exttimelo=*(volatile unsigned long*)(BASE+Ext_Sec10_Usec1_Port);
exttimehi=*(volatile unsigned long*)(BASE+Ext_Year1_Min1_Port);
exttime100ns=*(volatile unsigned char*)(BASE+Ext_100ns_Port);
```

When any of the external time tag latch ports are read, the Ext_Ready flag in BASE+Status_Port is cleared. The 100 Nsec through 100 Millisecond bits remain latched until another external time tag occurs. **The 200 millisecond through 8 years bits remain latched for at least 90 milliseconds after the external time tag occurs, so the external time tag data must be read within 90 milliseconds after the time tag occurs.**

Reading External Binary Time Tags (-BIN option only)

The simplest and fastest way to read external time tags uses the PMC-SYNCCLOCK32-UNIV external time tag latches which may be read directly across the PCI bus:

An example reads External Time Tag data with 3 C statements:

```
/* read 2**31 (msb) to 2**0 usec of day bits first */
exttimelo=*(volatile unsigned long*)(BASE+Ext_Sec10_Usec1_Port);
/* then read
   BCD year of century (2**21..2**14 bits
   binary day of year (2**13..2**5 bits)
   2**36..2**32 usec of day (2**4..2**0 bits)
   2**31..2**22 bits of exttimehi are not defined
   BCD 100s of nsec digit can also be read*/
exttimehi=*(volatile unsigned long*)(BASE+Ext_Year1_Min1_Port);
exttime100ns=*(volatile unsigned char*)(BASE+Ext_100ns_Port);
```

The external time tag latches can be read in any order and the user can select any or all to read. When any of the external time tag latch ports are read, the Ext_Ready flag in Status_Port is cleared. The 100 Nsec through 2**18 μsec bits remain latched until another external time tag occurs. The 2**19 μsec through 2**7 years bits remain latched for at least 90 milliseconds after the external time tag occurs.

Simulating External Time Tags

For program testing purposes the user can simulate the occurrence of an external time tag by writing Trigger_Sim_Ext_Time_Tag to Status_Port.

Example:

```
*(volatile unsigned char*)(BASE+Status_Port)=Trigger_Sim_Ext_Time_Tag ;
```

Time Match Pulses/Toggles

The Match bit of Status_Port toggles when the time programmed into the Match_Usec10_Port and Match_Msec1_Usec100_Port registers (and, if Matchext_Enb in Control_Port is set the DP_Match_SS and DP_Match_ht dual port RAM locations) match the clock time. The Match bit is also output to I/O connectors for use in triggering or gating other equipment. The user may clear the Match bit by writing Reset_Match to Status_Port. Depending on the application it may be desirable to clear Match right away (for example if the user is trying to generate a set of closely spaced pulses. In other applications (for example generating a logic gate enabling a A/D converter) Match may be allowed to remain set for longer times until another clock value is matched to toggle it off. Before writing new match register settings, the user should write 0xFF (which cannot be matched because time is in BCD format) into both match ports to avoid false matches. DP_Match_SS and DP_Match_ht are don't cares when set to 0xFF or Matchext_Enb bit of Control_Port is 0 (disabled). Note that the Time

Match only applies to 1 μsec through 8 msec bits when Matchext_Enb bit of Control_Port is 0 (disabled) and It is the user's responsibility to check for correct upcoming 10 millisecond interval before writing the Match time registers. The Control_Port register controls interrupt enable when match times occur. Users who require interrupt support in Windows must purchase a development seat of WinIntr software. WinIntr includes the PLX SDKPro software package along with sample and demo applications using interrupts in the Windows environment. See the mattst.c demo program for using Match with Matchext_Enb disabled and matssht.c for using Match with Matchext_enb enabled.

Heartbeat Pulses and Interrupts

The Heartbeat bit of the Status_Port will be set when a heartbeat pulse is output at the HBPULSEL (pin 8) of connector J3. HBPULSEL is driven by the output pulses from a 82C54 programmable counter driven at 3 MHz or 100 PPS as configured by P13. The 3 MHz or 100 PPS is divided by a user controllable divisor of 2 to 65535 (default 3000) to control the heartbeat rate. The Control_Port register controls interrupt enable when HBPULSEL pulses occur. Users who require interrupt support in Windows must purchase a development seat of WinIntr software. WinIntr includes the PLX SDKPro software package along with sample and demo applications using interrupts in the Windows environment.

The heartbeat rate may be changed using the dual port RAM.

Dual Port RAM for Supplementary Status/Control

A 256 byte dual port RAM on the PMC-SYNCCLOCK32-UNIV shares access between the microprocessor in the PMC-SYNCCLOCK32-UNIV and the host PCI computer. The first 128 locations are READ ONLY from the user program. The last 128 locations are READ/WRITE from the user program are initialized after reset by the on-board processor. The dual port address assignments are listed in the DPDEF.H listing in the appendices.

Accessing Dual Port RAM

To reduce the number of bus addresses needed for the dual port RAM, it is accessed over the PCI bus in a two step operation. First, the user write the dual port address byte to Dual_Port_Address_Port. Within 1 μsec, the current contents of the dual port address selected are available in Dual_Port_Data_Port. To change the current contents of the selected address, the user program writes the new data to the Dual_Port_Data_Port. A short delay (two Status_Port reads works well) should follow any dual port access. To allow the on-board microprocessor enough bus cycles for its' clock time synchronizing function, dual port RAM accesses should not exceed 100 per millisecond. Users should make sure that multiple threads accessing dual port RAM do not interfere with each other. A mutex is useful for ensuring exclusive access.

Example:

```
 /* Specify dual port address */
*(volatile unsigned char*)(BASE+Dual_Port_Address_Port)=DP_Control0;
 dpdata=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data read
 dpdata=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PMC-SYNCCLOCK
 dpdata=*(volatile unsigned char*)(BASE+Dual_Port_Data_Port);//get current contents
 /* write new contents */
 *(volatile unsigned char*)(BASE+Dual_Data_Port)=dpdata |Control_CodePriority) ;
 dpdata=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data write
 dpdata=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PMC-SYNCCLOCK
```

Users should make sure that multiple threads accessing dual port RAM do not interfere with each other. A mutex is useful for ensuring exclusive access.

Sending Commands through the Dual Port RAM.

The DP_Command location in the dual port RAM is used to send commands to the PMC-SYNCCLOCK32. Illegal commands will be echoed as 0xFF. Legal commands are shown in dpdef.h. Some examples are:

Command_Set_Major copies time from dual port major time locations to clock

Command_Set_Years copies time from dual port years locations to PMC-SYNCCLOCK clock (To avoid incorrect clock settings when major time or year has only been partially initialized, the commands signal from the user to the PCI that the major time or year is ready to use.)

Command_Set_RAM_FIFO (-RAMFIFO option only) tells the PMC-SYNCCLOCK32-UNIV to use RAM_FIFO mode for External Time Tags

Command_Reset_RAM_FIFO (-RAMFIFO option only) tells the PMC-SYNCCLOCK32-UNIV to stop using RAM_FIFO mode (-RAMFIFO option) for External Time Tags.

Command_Clear_RAM_FIFO (-RAMFIFO option only) tells the PMC-SYNCCLOCK32-UNIV to set the External RAM FIFO (-RAMFIFO option) to empty.

Setting Leap Year Control in Dual Port RAM

The DP_Control_Leapyear bit of dual port RAM address DP_Control_0 tells the PMC-SYNCCLOCK if the current year is a leap year. Because IRIG and NASA36 time code inputs do not contain year information, the PMC-SYNCCLOCK requires the user to tell the PMC-SYNCCLOCK if the current year is a leap year. If an on-board GPS receiver is used as the time reference, the leap year information is determined automatically from the year data provided by the GPS receiver. Leap second information is only used by the PMC-SYNCCLOCK at 365 days 23 hours 59 minutes 59 seconds (or 60 seconds if leap second day =365) to determine if the next day should be 366 (leap year control =1) or 001 (leap year control =0). The DP_Control_Leapyear flag is initialized to 0 when the PMC-SYNCCLOCK does power-on reset. If set incorrectly, the PMC-SYNCCLOCK will detect the correct new day about half a time code frame into the new day (.5 seconds for IRIG-B, NASA36, 2137 or .05 seconds for IRIG-A, .005 seconds for IRIG-G) and set the onboard clock time to the correct value. The DP_Control_Leapyear bit is only used if the year in the DP_Year10_Year1 and DP_Year1000_Year100 locations is invalid. If they are valid, the PMC-SYNCCLOCK32-UNIV can automatically determine leap year status.

If the only reference input to the PMC-SYNCCLOCK is a 1 PPS pulse it is **VERY IMPORTANT** to maintain the correct value of the DP_Control_Leapyear bit or to set the year in the DP_Year1000_Year100 and DP_Year10_Year1 locations. If correct values are not maintained, the PMC-SYNCCLOCK may make an incorrect decision on the day value following Julian day 365.

Selecting 1PPS or Time Code Reference Priority in Dual Port RAM

The PMC-SYNCCLOCK32-UNIV allows redundant use of a time code input and a 1 PPS (or GPS receiver) input simultaneously as reference. The DP_Control0_CodePriority bit of the DP_Control0 dual port RAM location will be "1" if the time code is the primary reference and "0" if the 1PPS is the primary reference. Default is for 1PPS (GPS) to be the primary reference.

Setting Time Code Propagation Delay Polarity in Dual Port RAM

The PMC-SYNCCLOCK allows negative propagation delay correction for use in installations where the time code "on-time" position is ahead of on time. Setting the DP_Control0_NegCodePropD bit in the DP_Control0 specifies that the DP_CodePropDel digits specify a NEGATIVE propagation delay correction. DP_Control0_NegCodePropD is cleared on power on or reset.

Setting PPS Propagation Delay in Dual Port RAM

The PMC-SYNCCLOCK32-UNIV allows negative propagation delay correction for use in installations where the 1 PPS "on-time" position is ahead of on time. Setting the DP_Control0_NegPPSPropD bit in the DP_Control0 specifies that the DP_CodePropDel digits specify a NEGATIVE propagation delay correction. DP_Control0_NegPPSPropD is cleared on power on or reset.

Specifying Modulated Input Code Format in Dual Port RAM

The PMC-SYNCCLOCK32-UNIV allows the user to specify the format of the modulated input time code in the DP_Code_Select dual port RAM address by writing the following constants defined in DPDEF.H. Default is IRIG-B. If only GPS or 1 PPS is used as a reference, the DP_Code_Select setting is irrelevant.

IRIG-B:	CodeSelect_IRIGB	
NASA36:	CodeSelect_NASA36	
IRIG-A:	Code_Select_IRIGA	
2137:	Code_Select_2137	(requires -R2137 option support)
XR3:	Code_Select_XR3	(requires -RXR3 option support)
IRIG-G	Code_Select_IRIGG	(requires -RG or –GIN option support)
IRIG-E	Code_Select_IRIGE	(requires -RE option support)

Specifying Leap Second Day in Dual Port RAM

UTC time (Universal Coordinated Time) has an extra second added periodically (about once every 18 months) to keep UTC time aligned with astronomical time. The added second (called a LEAP SECOND) occurs at the end of a specified day so that the last second of that day is at 23 hours 59 minutes 60 seconds. In order for the PMC-SYNCCLOCK clock to maintain correct UTC time, the PMC-SYNCCLOCK must know by the PRECEDING second if a given second is a leap second. GPS satellites broadcast warning of upcoming leap seconds. PMC-SYNCCLOCK units with GPS receivers will initialize the Leap Second Day locations if the selected GPS receiver allows the PMC-SYNCCLOCK to decode it. The standard modulated time codes do not provide leap second advisory information, so the user is allowed to set a value based on user program support.

The specified day is usually June 30 or December 31. It is usually announced several months in advance. 10s and 1s of the Julian date are specified in dual port RAM locations DP_Leap_SecDay10Day1 (for 10s and units) and DP_Leap_SecDay1000Day100 for 100s of days. Power on and reset initializes both to FF. For example, say a leap second is to occur on Dec. 31 in a non-leap year. Then the user should set DP_Leap_SecDay10Day1 = 0x65 and DP_Leap_SecDay1000Day100=0x03.

If the Leap Second Day is not set and a leap second occurs, starting with the leap second the PMC-SYNCCLOCK32-UNIV will be 1 second fast until:

1. If using modulated time code input: about 1/2 way into the next modulated time code field (.5 sec for IRIG-B, NASA36, XR3 ,2137, .05 sec for IRIG-A, .005 second for IRIG-G)

2. If using PPS reference without time code until a Set Major Time command is issued by the user with the correct time of the most recent PPS pulse set correctly.

Setting Time Code Propagation Delay in Dual Port Ram

DP_CodePropDel_ms100ms10 specifies time code propagation delay 100ms and 10 ms digit.
DP_CodePropDel_ms1us100 specifies time code propagation delay 1ms and 100 μs digits.
DP_CodePropDel_us10us1 specifies time code propagation delay 10μs and 1 μs digits.
DP_CodePropDel_ns100ns10 specified time code propagation delay 100 ns and 10 ns digits.
Power on and reset default setting are 0 for all digits except for units with the -NVP option.

Non-volatile Propagation Delay Settings

Units with the -NVP (non-volatile propagation delay setting) will set the propagation delay to the values automatically stored at the last POWER DOWN. New settings will be stored automatically at POWER DOWN. If using the -NVP option, the user should do a "sanity check" by reading the values after power up from the dual port RAM and checking them for reasonable values.

Setting PPS Propagation Delay in Dual Port Ram

For achieving μsecond accuracy it is necessary for the user to set the propagation delay correction for each PMC-SYNCCLOCK32-UNIV installation. The correct values for propagation delay correction can be estimated base on cable lengths and propagation speed of the cable used, but the effects of capacitive loading usually require that the 10s and units of μsecond digits and below be determined experimentally. Refer to the installation section of the manual.

DP_PPS_PropDel-ms100ms10 specifies PPS pulse propagation delay 100ms and 10 ms digit.
DP_PPS_PropDel_ms1us100 specifies PPS pulse propagation delay 1ms and 100 μs digits.
DP_PPS_PropDel_us10us1 specifies PPS pulse propagation delay 10μs and 1 μs digits.
DP_PPS_PropDel_ns100ns10 specifies PPS pulse propagation delay 100 ns and 10 ns digits.
Power on and reset default setting are 0 for all digits except for units with the -NVP option. The -NVP option will retain the propagation delay settings at the last POWER OFF.

For achieving best accuracy it is necessary for the user to set the propagation delay correction for each PMC-SYNCCLOCK installation. Refer to the installation section of the manual.

Setting Major Time in Dual Port RAM

The user MUST initialize Major Time if the 1 PPS input to the PMC-SYNCCLOCK is the sole timing reference and the seconds, minutes etc. clock times must be correct.. If there is a time code input, the PMC-SYNCCLOCK can determine major time from the time code. If the -GPS option is used, the PMC-SYNCCLOCK can determine major time from the GPS receiver serial output. To set major time, the user sets the DP_Major_Time_s10s1 byte for seconds, through DP_Major_Time_d1000d100 for 100s day digit. See setmajor.c sample program.

Setting Year in Dual Port RAM

With a time code or PPS reference input, the user can set DP_Year10Year1 and DP_Year1000Year100 for the clock year (10 & 1 digits) to be correct and to allow the PMC-SYNCCLOCK to automatically detect leap years. If a GPS receiver option is used, the PMC-SYNCCLOCK will determine year from the GPS receiver serial output.

Changing Heartbeat, Lowrate and Rate2

The CLK0 signal (J3 pin 7), the Heartbeat signal (J3 pin 8), and the CLK2 signal (J3 pin 5) are generated by three sections (0,1, and 2 respectively) of a programmable 82C54 clock IC. Each section divides an input signal by a programmable divisor of 2-65535. The input signal for section 0 is a 100 PPS pulse synchronized to the clock. The input signal for sections 2 is a 3 MHz pulse rate also synchronized to the clock. The input for section 1 is configured by P13 to be either 100 PPS or 3 MHz. Because all inputs are synchronized to the clock, they share the same frequency disciplining. So the generated clock outputs will track the reference input like the main clock does. The factory default divisor for CLK0 is 100, so CLK0 will be 1PPS. The factory default divider for section 1(HEARTBEAT) is 3000, so HEARTBEAT will be 1000 PPS. The factory default for CLK2 divider is 39, so RATE2 will be 76.923 KHZ.
To change the settings from the factory default, the user can change the `DP_Ctr2_ctl ,, DP_Ctr0_lmsb` values in dual port RAM. A `Command_Set_Ctr0`, `Command_Set_Ctr1` or `Command_Set_Ctr2` command can be issued to cause the changed parameters to be changed immediately. or, a `Command_Rejam` command can cause all counters to be set to the new parameters at the start of the next second. The rejam method is used when it is important that a rate not only be synchronized in frequency to the clock, but also with a specific phase. Rejam will force the clock and the counters to resynchronize to the input signal at the start of the next second.

Each of the 3 counters has a control byte (`DP_Ctr0_ctl, DP_Ctr1_ctl, DP_Ctr2_ctl`) and a 16 bit divide count split into a most significant byte (`DP_Ctr0_msb, DP_Ctr1_msb, DP_Ctr2_msb`) and a low order byte (`DP_Ctr0_lsb, DP_Ctr1_lsb, DP_Ctr2_lsb`).

Counter Control Byte Bit Assignments

Bit <0>

Bit value	Mnemonic	Description
0	DP_ctl_bin	binary count
1	DP_ctl_bcd	BCD count

Bit <3:1>

Bit value	Mnemonic	Description
000	DP_ctl_mode0	"Stop at Terminal Count". Seldom used. Causes output to go low when Command_Set_Ctrx or Command_Rejam command is issued and to remain low until counter reaches 0. Then output goes high and stays high.
001	DP_ctl_mode1	"Hardware Re-triggerable One-Shot". Seldom used. Output goes low when jam sync occurs and stays low until count=0. Then goes hi until next jam sync.
010	DP_ctl_mode2	"Divide by N Counter". Out is high until count=0, then low for one CLKx pulse.
011	DP_ctl_mode3	"Square Wave". Out is high for first half of count and low for second half.
100	DP_ctl_mode4	"Software Trigger". Seldom used. Output is high until count=0, then low for one CLKx pulse. Sequence is restarted by writing new count.

101	DP_ctl_mode5	"Hardware Triggered Strobe" . Seldom used. Output is high until jam sync. Then after count=0 output goes low one CLKx cycle and then goes high again.

Bit <5:4> should always be 11 ("DP_ctl_rw")
Bit<7:6>:

Bit value	Mnemonic	Description
00	DP_Ctr0_ctl_sel	Select counter 0
01	DP_Ctr1_ctl_sel	Select counter 1
11	DP_Ctr2_ctl_sel	Select counter 2

GPS Satellite Tracking and Navigation Data (except Have-Quick receivers)

The GPSDEF.H file defines the dual port RAM locations used by the -GM options to pass navigation data and status to the user. Locations are provided for latitude, longitude, speed & heading over ground data and for navigation status and the number of satellites tracked. The data is updated once per second. Latitude & longitude data is available in both BCD format (DP_GPS_Lat_MinEM34 .. DP_GPS_Lon_Deg32) and in raw binary format. Because the raw binary format does not lose any accuracy in firmware conversion to BCD, it should be used when resolution better than one arc second is required for the application. The raw binary format is GPS receiver type dependant:

For the Motorola Oncore (GT,UT,M12) series, the raw binary format is 32 bit twos complement. The value of the LSB is 1 milliarcsecond. Negative values indicate S latitude or W latitude. **See gpsbin.c for example of using binary format data.**

Preventing/Correcting Data Change While Reading Position

Once per second the PMC-SYNCCLOCK firmware takes about 50 milliseconds to receive and convert binary position data arriving from the receiver. The firmware waits until latitude and longitude are received and converted and then copys the converted data in a burst to the dual port RAM. Even though the burst copy only takes a few microseconds, every second there is a small possibility that some dual port position byte reads could mix data from two different seconds. Even for static (non-moving) applications this could result in wrong data. For example, suppose the latitude is sometimes N 50 degrees 0.00 minutes and sometimes 49 degrees 59.9999 minutes. If data from two different seconds was mixed the user might get 49 degrees 0.00 minutes or 50 degrees 59.9999 minutes. To prevent wrong data the user can:

1. Read a set of lat/lon dual port data and then read a second set. If both sets agree the data is OK. If they don't agree, read a third time. The third set will be OK.

2. As part of an initialization procedure, detect (approximately) when in the second the dual port data is updated. You can determined when the data is updated by doing repeated dual port RAM position reads until a change is detected. The change will always occur at the same point in the second give or take 10 milliseconds. By reading time from the PMC-SYNCCLOCK32, the user program can avoid dual port position reads during the update times.

3. Test dual port RAM location DP_GPS_UpdateFlag for non 0 (on-board firmware sets non 0 after copying new position data into dual port RAM). New data is available when non 0. After reading data, write 0 into dual port RAM location DP_GPS_UpdateFlag so that it will not be non-0 until next set of new data is written by firmware.

Differential Data Acknowlege Count (-DF option with Motorola Oncore Only)

The Motorola Oncore GPS receiver sends the on-board Z80 microprocessor an acknowledge message whenever a valid RTCM104 message has been successfully received. The Z80 firmware will increment the dual port RAM `DP_GPS_RTCM_Ack` location whenever an acknowlegement is received from the Oncore. The user can periodically read the `DP_GPS_RTCM_Ack` value to verify that valid RTCM104 data is being received. See `gpsbin.c` for an example. The variations in latitude and longitude should be much smaller when valid RTCM104 data is received.

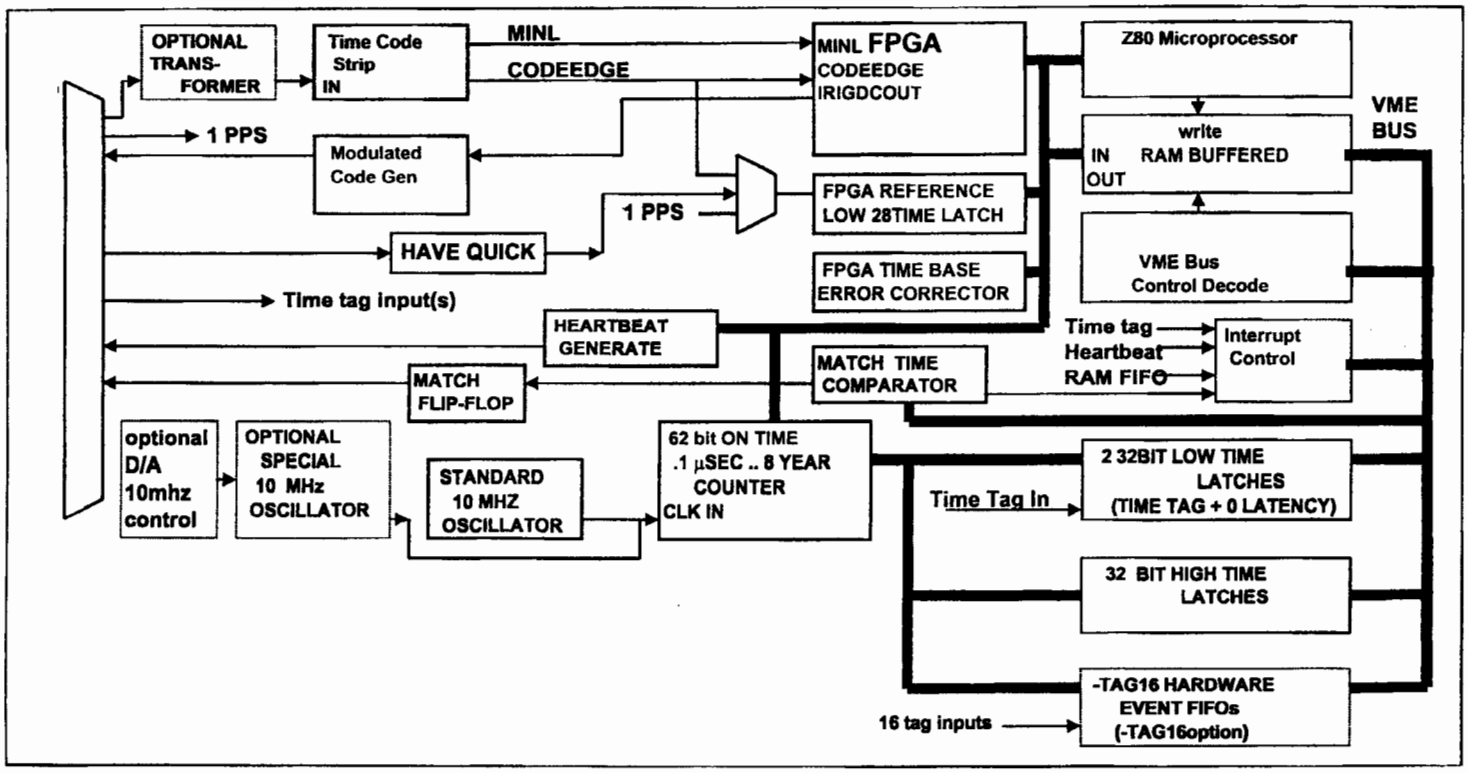
GPS Satellite Data (Have-Quick receivers)

GPS receiver Time Figure-of-Merit may be accessed through dual port RAM locations `DP_HQ_TFOM` (defined in `gpsdef.h`) . The data is updated once per second. The TFOM meaning is:

<0	No information
9	1PPS accuracy no better than 10 milliseconds
8	1 PPS accuracy 1 to 10 milliseconds
7	1 PPS accuracy 100 μsec to 1 millisecond
6	1 PPS accuracy 10 μsec to 100 μsec
5	1 PPS accuracy 1 μsec to 10 μsec
4	1 PPS accuracy .1 μsec to 1 μsec (typical for Plugger)
3	1 PPS accuracy .01 μsec to .1 μsec
2	1 PPS accuracy .001 μsec to .01 μsec
1	1 PPS accuracy better than 1 nsec
0	"Normal" (used when HQ receiver cannot compute actual TFOM)

Because the Have Quick signal does not contain navigation information, only time and TFOM are available.

Theory of Operation



Time Code Stripping

Time code stripping uses a zero crossing detector combined with a voltage divider and a negative peak detector to produce a sampling trigger and a high amplitude cycle pulse to send to a burst decoder in the FPGA device.

Input Signal Coupling

If the "X" option is implemented, transformer T1 couples the isolated CODEIN+ and CODEIN- inputs to the CODEIN and board ground. If no T option, CODEIN+ connects by jumper JPT1A to CODEIN and CODEIN- connects by jumper JPT1B to board ground. Capacitor C1 translates the input signal from + and - voltages around ground to + and - relative to the +5 volts through R19. Voltage follower U4 provides a high input impedance.

Zero Crossing Detector

The + side of C1 is the modulated signal input referenced to +5 volts. It is applied to the + side of a LM339 (U5) comparator section at pin U5-5. The - side of the comparator (U5-4) is connected to +5. A 3.3K pull-up resistor pulls up U5-2 whenever U5-5 is HIGHER than U5-4. The open collector output of U5-2 pulls low when U5-5 is LOWER than U5-4. So a +5V/0V square wave in phase with the analog input signal appears at U5-2. Of course there are switching transients showing up as false transitions near the correct zero crossing points that must be dealt with in subsequent processing.

Voltage Divider

The analog reference signal swinging + and - around +5 V is applied to one end of voltage divider R5 and R6. The other end of the divider is +5 volts. Because R6 is about twice the value of R5, the peak to peak AC voltage at the R5/R6 junction will be 2/3 the peak to peak AC voltage at C1 + side - but still centered on +5 V.

Peak Detector

Because the 2/3 scaled voltage at R5/R6 junction connects to comparator section U5-9 "+" side input, if U5-9 value is ever LESS than the comparator output voltage U5-14 then U5-14 will be driven lower. This makes U5-14 follow the NEGATIVE peak of the 2/3 scaled signal. A long time constant R7/C4 combination slowly increases the U5-14 voltage if the input signal amplitude decreases. The U5-14 level is the ideal comparison level to separate low and high amplitude input signal cycles. Because "low" amplitude cycles are 1/3 the amplitude of "high" amplitude cycles the 2/3 point is halfway between low and high. U22-5 is applied to U5-10 and compared with the full amplitude signal at U5-11. Whenever the full amplitude signal instantaneous signal is less than the U5-14 level the U5-13 signal will be pulled low, generating negative TTL pulses at about 15% duty cycle.

Burst Decoder

The FPGA includes a precision digital programmable time retriggerable one-shot which triggers on the negative going edge of the zero crossing comparator. The one-shot rearms when the zero crossing comparator output is high. So, the one-shot generates an edge a programmable time delay (about 25 μsec for IRIG-A and 250 μsec for other codes) after the zero crossing negative transition. Noise transitions that have a LOW time less than the programmable time are rejected for noise immunity. By using a counter to generate delay, the temperature varying errors of RC based approaches are avoided. Each delayed edge shifts in the amplitude comparator (U5-13) data for the corresponding cycle of the input code. Logic inside the FPGA detects the beginning of each burst (a burst starts when a large cycle follows a small cycle) and transfers the high/low amplitude information for each cycle of the burst to the microprocessor.

FPGA Clock

The FPGA contains a conventional BCD clock (optional binary) that counts the 10 MHz crystal oscillator pulses. The hardware counter covers the 100 ns bit through the 100 ms bit.

200ms .. 8 Year Bits

Higher order bits (200 ms .. 8 years) are maintained in two sets of latches. One set of latches contains the high order bits for "even" 100 millisecond intervals and the other set contains the high order bits for "odd" intervals. By latching the state of the 100 millisecond counter bit the rest of the high order bits are selected. The high order latches are updated in a leap-frog fashion. Just before the end of a ODD 100 millisecond interval, the EVEN latches are updated to contain correct data for the upcoming even interval. And, just before the end of the EVEN interval, the ODD latches are updated. This approach allows the user to have plenty of time (at least 100 msec) to retrieve time measurements and also allows economical implementation of high order bits.

Disciplining

Disciplining is achieved by periodic advance/retard techniques as used in conventional packaged and board level time code generators. Optionally a voltage controlled oscillator may be installed

either at XO1 or on a daughterboard. When a voltage controlled oscillator is used, a D/A converter is used for fine control of frequency after approximate synchronization is established using the retard/advance circuit. The D/A is a 16 bit 0/4V range.

Clock conditioning

Because telecommunications applications usually require that the disciplined oscillator generate a sine wave output, a simple circuit is used to convert the sine wave into a square wave, as well as to protect the PMC-SYNCCLOCK32-UNIV against transients if an external oscillator is used. C28 AC couples the sine wave and R13/R14 provide hysterisis. CR2 and CR3 provide protection.

Modulated code generation

Modulated time code output uses an Exar data book circuit (XR2206 U6) to generate an amplitude modulated sine wave. C7 and RP3-3/6 in series with adjustment pot R8 determine the center frequency (1 KHZ for IRIG-B, NASA36 and 2137, 10 KHZ for IRIG-A, 100KHZ for IRIG-G, 250 Hz for XR3).
The DC frequency control current established by R8/RP3-3/6 is augmented by current through RP3-2/7. When the XR2206 is locked to the FPGA clock, the IADJ signal will be a 50% duty cycle 2 KHZ (20 KHZ for IRIG-A, 200KHz for IRIG-G, 500 Hz for XR3) square wave. If the XR2206 is running fast, the square wave will have a lower duty cycle, reducing the augmented current and reducing the XR2206 frequency. If the XR2206 is running fast, the square wave duty cycle will increase, raising the XR2206 frequency. RP3-1/8 and C6 filter out the AC component of the square wave leaving a slowly varying DC voltage at C6 for XR2206 frequency control. The sine wave output at U6 pin 2 is amplitude modulated using the AM input pin 1. The 0/5V IRIGDCOUTL signal is LOW for large amplitude bursts and HIGH for small amplitude bursts. The 0/5 is scaled by voltage divider R10/R11 to 2.5/5 V. Trimmer pot R3 allows adjustment of the modulated sine wave symmetry. Because the modulated sine wave output is riding on a +6V DC signal, it is AC coupled to an output driver U4 by capacitor C8 and drain resistor R4. The time code output at the input to driver U4 is 2.5V pk/pk. U4 uses a non-inverting feedback configuration for a gain of 1X. U4 is capable of driving 50 ohm loads, and uses series resistor R43 to set a source resistance of 51 ohms. When terminated at the receiving system in 50 ohms, a 1.25V pk/pk value will be received.

Error Measurement

The FPGA contains internal "reference" latches for capturing the time of the clock when the one-shot sampling pulses occur (or one sample per burst for high speed codes like IRIG-A). The on-board microprocessor uses the captured times to estimate the difference between the clock time and the modulated code time. If the time code input has priority for error correction, the difference information from the burst capture is used for disciplining. The PMC-SYNCCLOCK32-UNIV firmware can also capture the time of a reference 1PPS input. By sharing the reference latches between the between time code and 1 PPS pulses, redundant operation is possible.

Match Times

The FPGA contains two 8 bit comparators directly accessible across the PCI bus to allow the user to toggle the MATCH flip flop when the user programmed times match the clock time. Direct bus access and an associated interrupt allow user's to generate very tightly spaced pulses at arbitrary times that cannot be achieved in conventional approaches which maintain the comparators under microcomputer control.

3.0 MHz frequency generation

3.0 MHz synchronous to the main clock is output by the FPGA for use in generating "modem" rates 24000 Hz, 2400 Hz etc. that are not easily derived from 10 MHz or 5 MHz. The 3 MHz is disciplined by the same mechanisms used for the FPGA clock, so it will be synchronous to the FPGA clock.

Heartbeat Rates

A conventional Intel architecture 82C54 triple 16 bit counter (U11) is used to generate three user controlled rates / frequencies. The rates and frequencies will track the on-board clock because they are generated from the same disciplining clock frequencies.

Microcomputer

The microcomputer section is a conventional Z80 microprocessor (U10), 32K X 8 EPROM/PLD/8KX8 RAM (U26), and triple programmable counter (U11). The dual port RAM accessed by the host program is maintained in part of U26 by FPGA (U3) control. The microprocessor runs at 5 MHz derived from XO1 (10 MHz) divided by 2 inside U26. The FPGA U3 generates Z80 interrupts for code reading, disciplining, external time tag processing (if user enabled), and ODD.EVEN latch updating and dual port RAM support.

D/A frequency control (DTCXO option)

16 bit D/A U21 is controlled by sending data bits to it in serial accompanied by a SERCLK clock signal. The data bits come directly from the Z80 data bus and the SERCLK signal is generated by decoding an I/O address in FPGA (U3). The SERLDL signal transfers the data to the D/A converter and is also generated by I/O address decoding in U3.

GPS receiver serial interface

A Z80 SIO (U12) allows 4800 baud or 9600 baud serial bi-directional communication between an external GPS receiver and the Z80 microcomputer using channel A of the two channel SIO. The 32X baud rate clock is generated by a divider implemented in FPGA device U3 by dividing 3 MHZ or 10MHZ. Channel B of the SIO is configured for 4800 baud reception of RTCM104 format serial differential GPS signals for cases where the Motorola VP Oncore GPS receiver is used. When the external GPS receiver option (-XGPS) is selected, the U13 and U15 RS422 driver/receiver Ics are used to map the serial I/O levels from RS-422 to TTL.

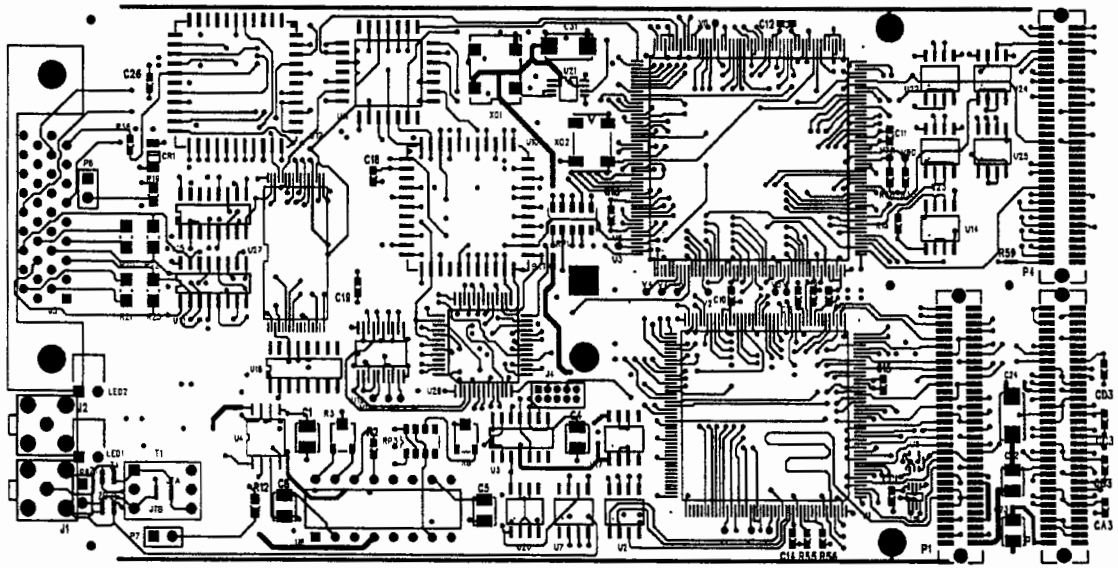
HAVE QUICK interface option

The HAVE QUICK GPS receiver interface receives the 1 PPS (PTTI1PPS) and serial data (PTTI) from a user furnished GPS receiver (example: PLUGGER). The 1PPS is clamped by R9 and CR1 to TTL levels and used as the 1 PPS reference by FPGA U3. The PTTI data is input to the RXDB pin of U12. Raw HaveQuick serial Manchester data is sampled at 5.4KHz by U12 and converted by firmware into binary data.

PCI Bus Interface

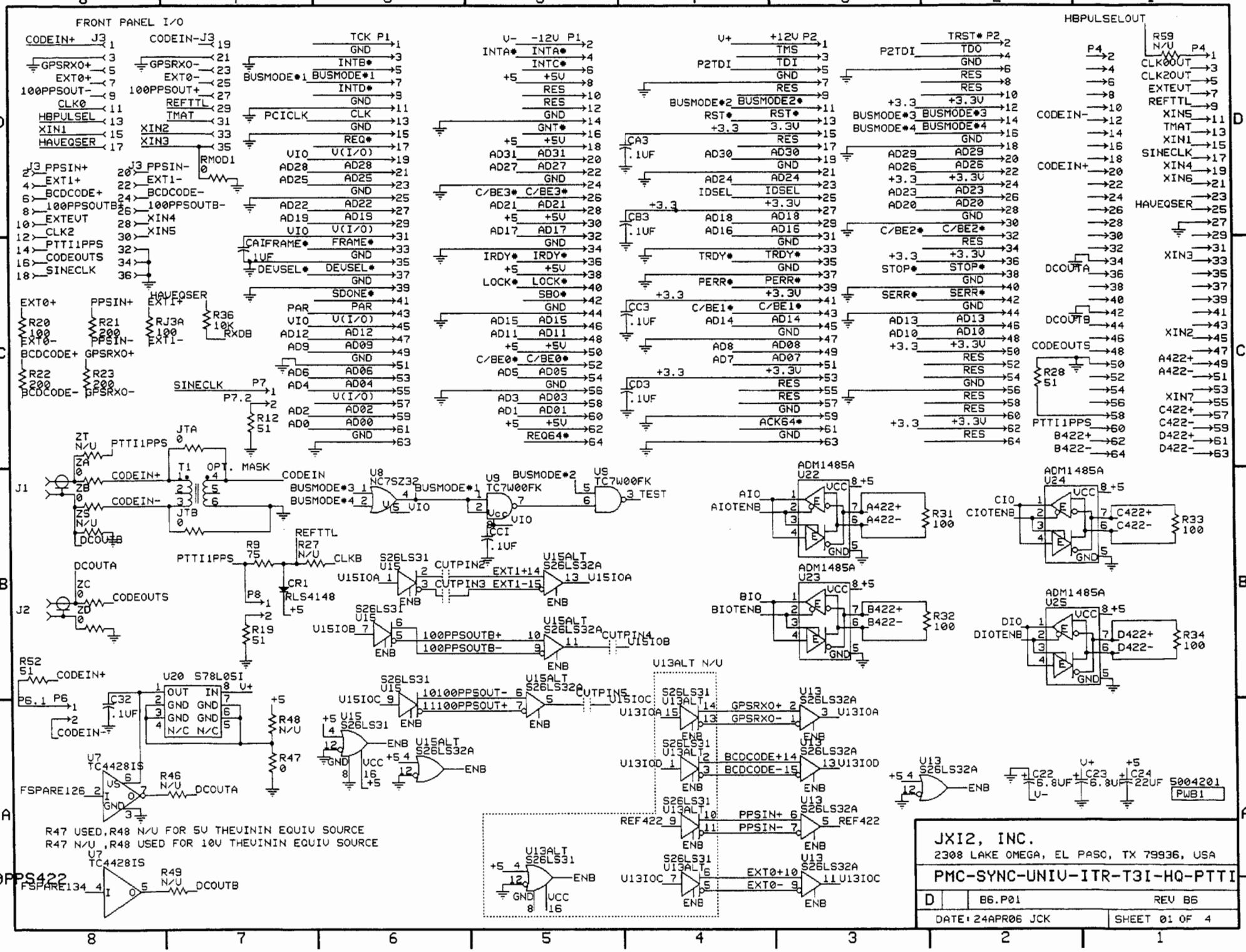
The PCI bus interface uses an off-the-shelf ASIC (PLX PCI9030) U1 to bridge from the PCI bus to a 32 bit local bus (LAD<31:0>, LA<5:2>, LBE<3:0>/L,CS0,LRDY/,ADS/,LINTI1,LWR/,LW/R/). FPGA Device U3 implements the registers the user reads and writes. A user access is decoded

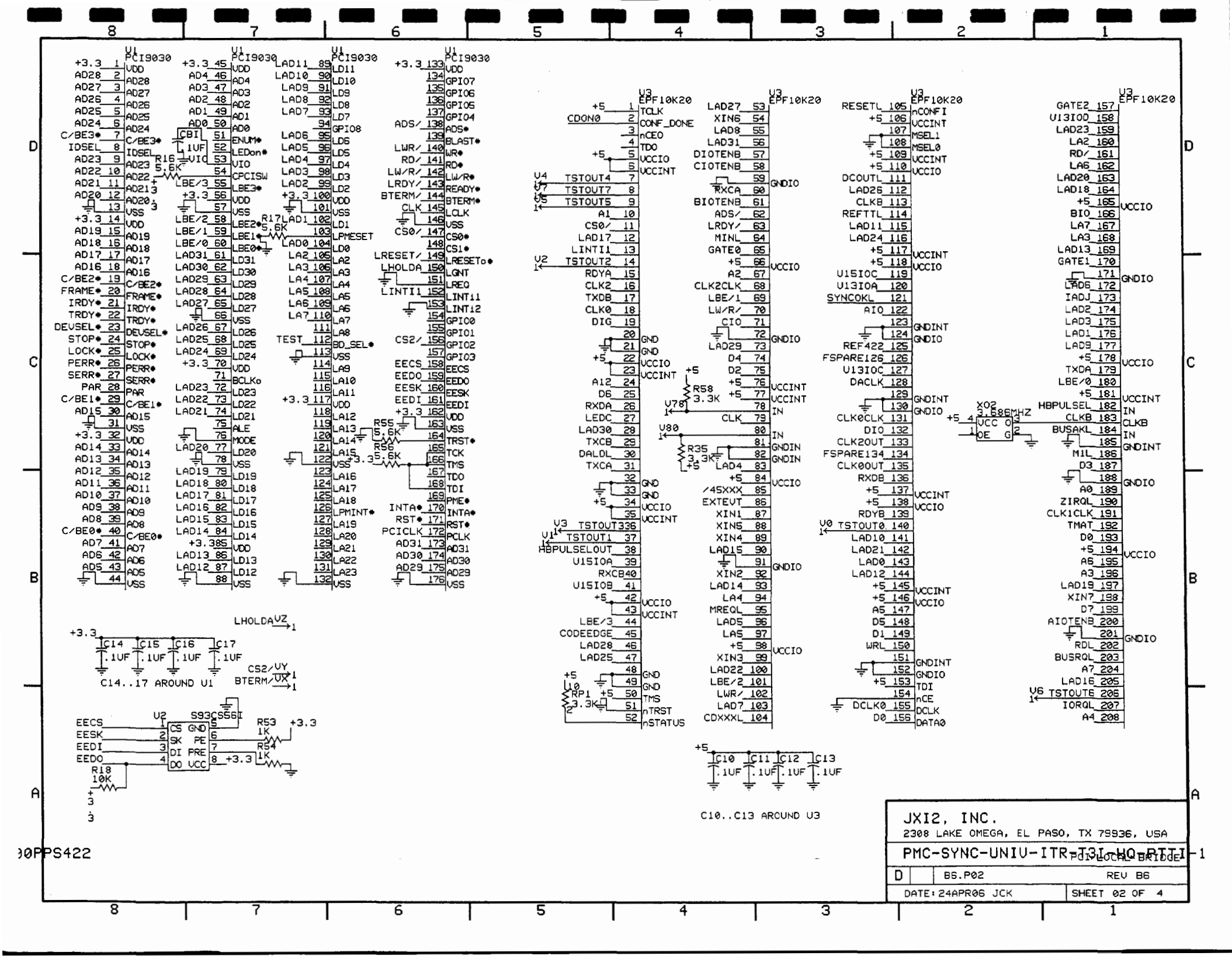
by U1 which immediately terminates with a PCI bus specification V2.1 retry to release the PCI bus. Then U1 completes the access over the local bus and completes the user access with a PCI retry cycle.

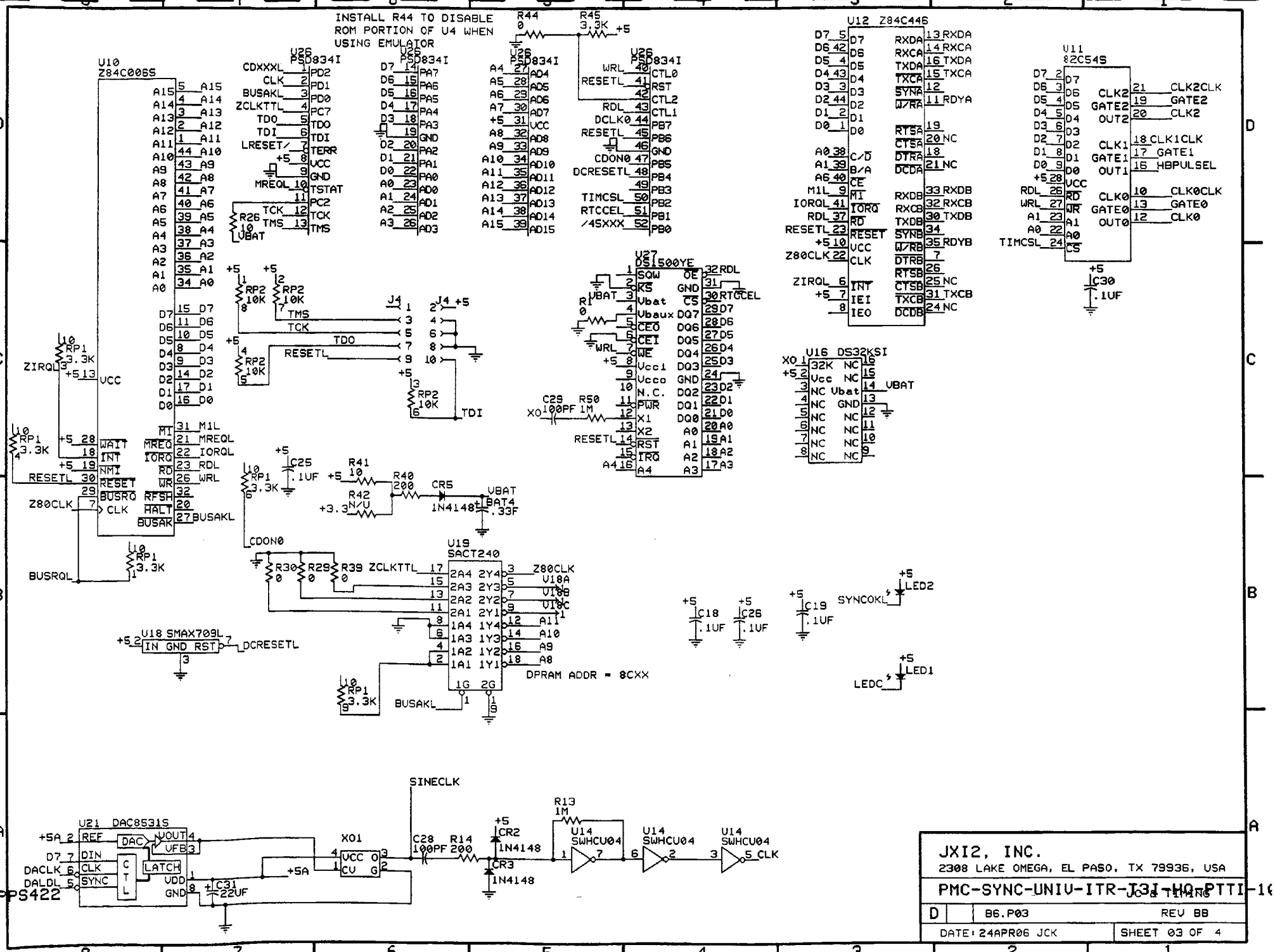


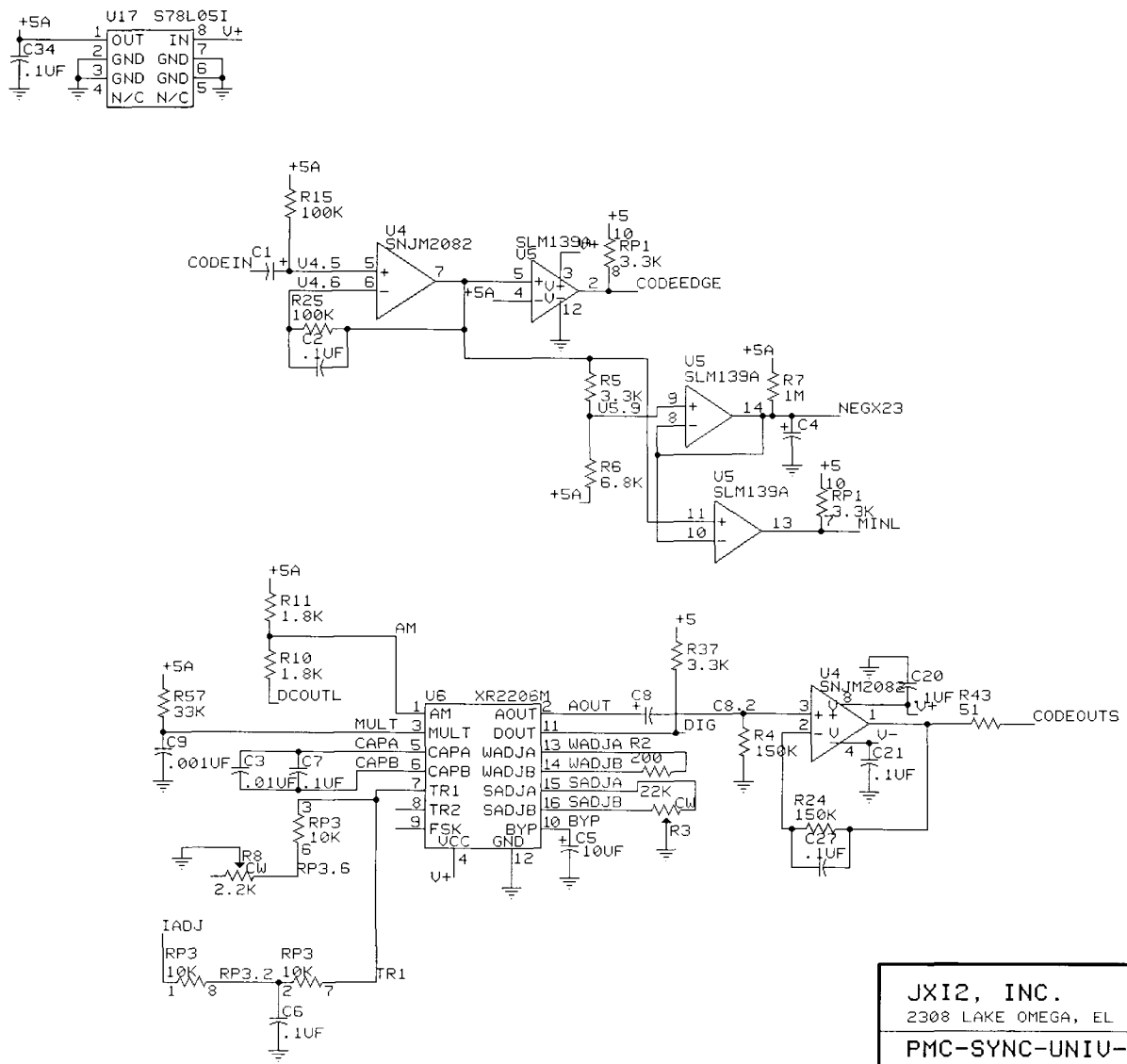
COMPONENT SIDE

PMC-SYNC32-UNIV REV B 6/24/05


FRONT PANEL I/O
CODEIN+ J3
CODEIN-J3
GPSRXO+
GPSRXO-
EXT0+
EXT0-
100PPSOUT-
100PPSOUT+
CLK0
REFTTL
HBPULSEL
TMAT
XIN1
XIN2
HAVEQSER
XIN3
RMOD1
J3 PPSIN+
J3 PPSIN-
EXT1+
EXT1-
BCDCODE+
BCDCODE-
100PPSOUTB+
100PPSOUTB-
EXTEVT
XIN4
CLK2
XIN5
PTTI1PPS
CODEOUTS
SINECLK
TCK P1
INTB*
BUSMODE*1
INTD*
PCICLK
CLK
REQ*
V(I/O)
AD28
AD25
AD22
AD19
FRAME*
CAIFRAME*
.1UF
DEVSEL*
SDONE*
PAR
AD12
AD09
AD06
AD04
AD02
AD00
V- -12V P1
INTA*
INTC*
+5V
RES
GNT*
AD31
AD27
C/BE3*
AD21
AD17
IRDY*
LOCK*
SBO*
AD15
AD11
C/BE0*
AD05
AD03
AD01
REQ64*
V+ +12V P2
TMS
P2TDI
TDI
BUSMODE*2
BUSMODE2*
RST*
3.3V
CA3
AD30
AD24
IDSEL
CB3
AD18
AD16
TRDY*
PERR*
CC3
C/BE1*
AD14
AD08
AD07
CD3
ACK64*
TRST* P2
TDO
BUSMODE*3
BUSMODE2*3
BUSMODE*4
BUSMODE2*4
AD29
AD26
AD23
AD20
C/BE2*
STOP*
SERR*
AD13
AD10
HBPULSELOUT
P4
R59
N/U
CLK0OUT
CLK2OUT
EXTEVT
REFTTL
XIN5
TMAT
XIN1
SINECLK
XIN4
XIN6
HAVEQSER
XIN3
XIN2
A422+
A422-
XIN7
C422+
C422-
D422+
D422-
DCOUTA
DCOUTB
CODEOUTS
PTTI1PPS
B422+
B422-
R20 100 EXT0-
R21 200 PPSIN-
RJ3A 100 EXT1-
R36 10K RXDB
R22 200 BCDCODE-
R23 200 GPSRXO-
R28 51
SINECLK P7
P7.2
R12 51
J1
J2
ZT N/U
ZA 0
ZB 0
ZS N/U
ZC 0
ZD 0
JTA 0
JTB 0
T1
OPT. MASK
CODEIN
U8 NC7SZ32
U9 TC7W00FK
U8 TC7W00FK
TEST
VIO
VCC
CCI .1UF
R9 75
R27 N/U
CLKB
CR1 RLS4148
P8
R19 51
U15 S26LS31
U15ALT S26LS32A
U15IOA
U15IOB
U15IOC
CUTPIN2
CUTPIN3 EXT1+15
EXT1-15
CUTPIN4
CUTPIN5
100PPSOUTB+
100PPSOUTB-
10100PPSOUT-
11100PPSOUT+
ENB
GND
VCC
+5
U13ALT N/U
U13 S26LS32A
U13IOA
U13IOB
U13IOC
U13IOD
GPSRXO+
GPSRXO-
BCDCODE+14
BCDCODE-15
EXT0+10
EXT0-
PPSIN+
PPSIN-
REF422
ADM1485A
U22
U23
U24
U25
AIO
AIOTENB
BIO
BIOTENB
CIO
CIOTENB
DIO
DIOTENB
R31 100
R32 100
R33 100
R34 100
R52 51
U20 S78L05I
OUT
IN
GND
N/C
C32 .1UF
R48 N/U
R47 0
U7 TC4428IS
R46 N/U
R49 N/U
FSPARE126_2
FSPARE134_4
R47 USED,R48 N/U FOR 5V THEVININ EQUIV SOURCE
R47 N/U ,R48 USED FOR 10V THEVININ EQUIV SOURCE
C22 6.8UF
C23 6.8UF
C24 22UF
5004201
PWB1
JXI2, INC.
2308 LAKE OMEGA, EL PASO, TX 79936, USA
PMC-SYNC-UNIV-ITR-T3I-HQ-PTTI-1
D
B6.P01
REV B6
DATE: 24APR06 JCK
SHEET 01 OF 4


U1 PCI9030
U3 EPF10K20
C14..17 AROUND U1
C10..C13 AROUND U3
U2 S93CS66I
JXI2, INC.
2308 LAKE OMEGA, EL PASO, TX 79936, USA
PMC-SYNC-UNIU-ITR-T3I-HQ-PTI
PCI LOCAL BRIDGE
D
B6.P02
REV B6
DATE: 24APR06 JCK
SHEET 02 OF 4
30PPS422


INSTALL R44 TO DISABLE ROM PORTION OF U4 WHEN USING EMULATOR
U10 Z84C0085
U26 PSD834I
U27 DS1500YE
U12 Z84C446
U11 82C545
U16 DS32KSI
U18 SMAX709L
U19 SACT240
U21 DAC8531S
U14 SWHCU04
DPRAM ADDR = 8CXX
JXI2, INC.
2308 LAKE OMEGA, EL PASO, TX 79936, USA
PMC-SYNC-UNIV-ITR-J3I-HQ-PTTI-16
B6.P03
REV BB
DATE: 24APR06 JCK
SHEET 03 OF 4



JXI2, INC.
2308 LAKE OMEGA, EL PASO, TX 79936, USA

PMC-SYNC-UNIV-ITR-T3I-HQ-PTJ-1(

D | B6.P04 | REV B6

DATE: 24APR06 JCK | SHEET 04 OF 4

)0PPS422

QTY	JXN	DESCRIPTION	COMMENTS	REF. DES.
16	0	VIA, COMPONENT SIDE		V0,V1,V2,V3,V4,V5,V6, V7,VX,VY,VZ,V78,V80 V18A,V18B,V18C
1	0000000	MOD.-PIN CUT: U15-2	!	CUTPIN2
1	0000000	MOD.-PIN CUT: U15-3	!	CUTPIN3
1	0000000	MOD.-PIN CUT: U15ALT-11	!	CUTPIN4
1	0000000	MOD.-PIN CUT: U15ALT-5	!	CUTPIN5
1	0030030	IC, XR2206M LIN WAVEFORM GEN -50/+125C	OPT -xOUT	U6
1	0120003	LED, RED T1 (INT. RES)		LED1
1	0120005	LED, GREEN T1 (INT. RES)		LED2
1	0399003	CAPACITOR, .33F 5.5V DOUBLE LAYER CAPACITOR -25C	! MYLAR SHIM	BAT4
1	0400000	TRANSFORMER, 10K:10K	OPT. MASK	T1
2	0700002	CONNECTOR, 1 X 2 .025 SQ HEADER		P6,P8
1	0700002	CONNECTOR, 1 X 2 .025 SQ HEADER	OPT. MASK	P7
1	0799039	CONNECTOR, 36 PIN IEEE1284		J3
1	0799040	CONNECTOR, SMB JACK RT ANGLE PC		J1
1	0799040	CONNECTOR, SMB JACK RT ANGLE PC	OPT.	J2
1	0799078	CONNECTOR, 2 X 5 .05 FEMALE		J4
1	1000301	IC, 74WHCU04 TRIPLE. HC UNBUFFERED CMOS INVER	!1000312	U14
1	1000307	IC, NC7SZ32 2 INPUT OR -40/+85C		U8
1	1000308	IC, TC7W00FK DUAL 2 INPUT NAND -40/+85C		U9
1	1000309	IC, 74ACT240PW -40/+85C DUAL QUAD 3ST. INV.		U19
1	1010000	IC, Z84C00XXVEC Z80 CPU -40/+85C 6,8,10MHZ PLCC4	1010003 OK	U10
1	1010001	IC, Z84C44XXVEC Z80 SIO 6,8,10MHZ -40/+85C		U12
1	1010002	IC, 82C54 - TRI PROG CTR CMOS PLCC28		U11
1	1010007	IC, PCI9030 PCI BUS TARGET I/F		U1
1	1010010	IC, DS1500YEN RTC/32X8 RAM -40/+85C	OPT.	U27
1	1010012	IC, DS32KHZ 32.768KHZ TCXO -40/+85C	OPT.	U16
1	1030005	IC, MAX709LCSA 4.65V RESET SUPERVISOR		U18
1	1030047	IC, TC4428 DUAL INV/NONINV DRIVER -40/+85C		U7
2	1030052	IC, 78L05 5V VOLT REG -40/+85C		U17,U20
1	1030053	IC, LM139AM QUAD VOLT COMP -4C/+85C		U5
1	1030063	IC, SNJM2082 HI DRV/LO OFF DUAL OP AMP -40/+85C		U4
2	1040001	IC, DS26LS32ACM QUAD RS422 RECEIVER		U13,U15ALT
1	1040002	IC, DS26LS31CM QUAD RS422 DRIVER		U15
1	1040002	IC, DS26LS31CM QUAD RS422 DRIVER	! N/U	U13ALT
1	1040014	IC, DAC8531 16 BIT SER DAC V OUT -40/+105C		U21
4	1040021	IC, ADM1485AR RS485 TRANSCVR -40/+85C		U22,U23,U24,U25
1	1090018	IC, EPF10K20 FPGA, 208 PQFP		U3
1	1090029	IC, EEPROM 2K SERIAL -40/+85C		U2
1	1090035	IC, PSD834F2-90MI FLASH/RAM/CPLD -40/+85		U26
3	1100002	DIODE, 1N4148 SILICON SIGNAL		CR2,CR3,CR5
1	1100002	DIODE, RLS4148 SURF. MT. SILICON SIGNAL		CR1
5	1201510	RESISTOR, SURF MT. 51 OHM 1/10W 5% CF		R12,R19,R28,R43,R52
2	1201510	RESISTOR, SURF MT. 51 OHM 1/10W 5% CF	N/U	R46,R49
1	1201750	RESISTOR, SURF MT. 75 OHM 1/10W 5% CF		R9
1	1202100	RESISTOR, SURF MT. 100 OHM 1/10W 5% CF	!	R20
1	1202100	RESISTOR, SURF MT. 100 OHM 1/10W 5% CF	! J3-4/22	RJ3A
1	1210000	RESISTOR, 0 OHM	N/U	R27
4	1212100	RESISTOR, SURF MT. 100 OHM 1/8W 5% CF		R31,R32,R33,R34
4	1212200	RESISTOR, 200 OHM 1/8W 5% CF		R14,R21,R22,R23

QTY	JXN	DESCRIPTION	COMMENTS	REF. DES.
4	1243560	RESISTOR, 5.60K OHM 1/16W 1% TF		R16,R17,R55,R56
11	1250000	RESISTOR, 0 OHM		R1,ZA,ZB,ZC,ZD,JTA,JTB R29,R30,R39,R47
1	1250000	RESISTOR, 0 OHM	! N/U	R59
2	1250000	RESISTOR, 0 OHM	!N/U	ZS,ZT
1	1251100	RESISTOR, SURF MT. 10 OHM 1/16W 5% CF		R26
1	1251100	RESISTOR, SURF MT. 10 OHM 1/16W 5% TF		R41
1	1251100	RESISTOR, SURF MT. 10 OHM 1/16W 5% TF	! N/U	R42
1	1260000	RESISTOR, 0 OHM	! J3-35/36	RMOD1
2	1260000	RESISTOR, 0 OHM	N/U	R44,R48
2	1262200	RESISTOR, 200 OHM 1/10W 5%		R2,R40
2	1263100	RESISTOR, 1.0K OHM 1/10W 5%		R53,R54
2	1263180	RESISTOR, 1.8K OHM 1/10W 5%		R10,R11
5	1263330	RESISTOR, 3.3K OHM 1/10W 5%		R5,R35,R37,R45,R58
1	1263680	RESISTOR, 6.8K OHM 1/10W 5%		R6
2	1264100	RESISTOR, 10K OHM 1/10W 5%		R18,R36
1	1264330	RESISTOR, 33K OHM 1/10W 5%		R57
2	1265100	RESISTOR, 100K OHM 1/10W 5%		R15,R25
2	1265150	RESISTOR, 150K OHM 1/10W 5%		R4,R24
3	1266100	RESISTOR, 1.0M OHM 1/10W 5%		R7,R13,R50
1	1279322	POTENTIOMETER, 2.2K 3MM SEALED -40/+100C		R8
1	1279422	POTENTIOMETER, 22K 3MM SEALED -40/+100C		R3
2	1282410	RESISTOR ARRAY, 10K OHM SEPARATE 8 PIN SURF MT		RP2,RP3
1	1289333	RESISTOR ARRAY, 3.3K OHM BUSSED 10 PIN SURF MT		RP1
2	1321002	CAPACITOR, 100 PF 5% NPO SURF. MT.		C28,C29
1	1331003	CAPACITOR, .001UF 10% X7R MLC SURF MT		C9
1	1341004	CAPACITOR, .01UF 50V 5% C0G -55/+125C		C3
27	1351002	CAPACITOR, .1 UF 10% X7R SURF. MT.		C2,C6,C10,C11,C12,C13 C14,C15,C16,C17,C18, C19,C20,C21,C25,C26, C27,C30,C32,C34,CA3, CAI,CB3,CBI,CC3,CCI, CD3
1	1351003	CAPACITOR, .1UF 5% 25v C0G		C7
2	1366800	CAPACITOR, 6.8UF 25V TANTALUM SURF. MT.		C22,C23
4	1371000	CAPACITOR, 10UF 16V TANTALUM SURF. MT.		C1,C4,C5,C8
2	1372200	CAPACITOR, 22UF 10% 6V TANTALUM		C24,C31
1	1500001	OSCILLATOR 10.0 MHZ +/-1.5PPM -40/+85C TCVCXO		XO1
1	1500003	XTAL OSC 3.6864 MHZ +/-100PPM -40/+85C	OPT.	XO2
3	1799000	CONNECTOR, 64 PIN PMC MALE		P1,P2,P4
1	5004201	PWB PMC-SYNC-UNIV REV B		PWB1

```
// t8iregs.h 13JUN05 JCK document 100nsec digit ext time tag read
/* T8IREGS.H 17JUL01 JCK correct Ext_Polarity comment: 0=^ 1=V edge*/
/* T8IREGS.H 17JUL01 JCK add -TAG8I definitions */
/* REGDEF.H 26JUN98 JCK RAMFIFO port moved from x16 to x18 */
/* PORT ADDRESS DEFINITIONS FOR PCI-SYNCCLOCK32 */
/* ALL ADDRESSES ARE RELATIVE TO CONFIGURED BASE ADDRESS */
/* RO means Read only. RW means read write. WO means Write only. */
#define Sec10_Usec1_Port        0x00 /* LONG RO latch new time & read ms8..us1 bits*/
#define Year1_Min1_Port         0x04 /* LONG RO read latched hr8..min1 bits*/

#define Status_Port             0x10    /* status & interrupt enables */
#define Response_Ready          2       /* RO Command Response Ready if 1*/
#define RAM_FIFO_Ready          4       /* RO RAM FIFO data ready */
#define Match                   8       /* RO Match register toggles on Match */
#define Reset_Match             8       /* WO bit3= 1: Reset Match reg */
#define Heartbeat               0x10    /* RO Heartbeat pulse detected if 1 */
#define Reset_Heartbeat         0x10    /* WO bit4 = 1: Reset Heartbeat flag*/
#define Sync_OK                 0x40    /* RO In-sync to time reference if 1 */

#define Control_Port            0x11
#define Command_Rdy_Intr_Enb    0x02    /* RW Command Response Ready Intr Enable */
/* the RAM_FIFO_Rdy_Intr_Enb bit is implemented for -RAMFIFO option ONLY  */
#define RAM_FIFO_Rdy_Intr_Enb   0x04    /* RW RAM FIFO Ready Intr Enable */
#define Time_Match_Intr_Enb     0x08    /* RW Time Match Intr Enable   */
#define Heartbeat_Intr_Enb      0x10    /* RW Heartbeat  Intr Enable   */
#define Matchext_Enb            0x20    /* RW Enables Match Extension past 10**3 usec*/

#define Mult_Frac_Port          0x12    /* implemented for -MULFRAC option only */
#define MulFrac2                0x01    /* WO select X2 or /2  */
#define MulFrac4                0x02    /* WO select X4 or /4  */
#define Frac                    0x80    /* WO select fraction speed */
#define Mult                    0x00    /* WO select multiple speed */

#define Match_Usec10_Usec1_Port  0x14    /* WO 10**1,10**0 match usec */

#define Match_Msec1_Usec100_Port 0x15    /* WO 10**3,10**2 match usec */

#define RAM_FIFO_Port            0x18    /* RO RAM FIFO */

#define Dual_Port_Address_Port  0x1A    /* WO Dual Port RAM Address */

#define Dual_Port_Data_Port     0x1E    /* RW Dual Port RAM Data */


#define Ext0_Ctl_Sts_Port       0x21    /* ext. time tag 0 control/status */
#define Ext1_Ctl_Sts_Port       0x25    /* ext. time tag 1 control/status */
#define Ext2_Ctl_Sts_Port       0x29    /* ext. time tag 2 control/status */
#define Ext3_Ctl_Sts_Port       0x2d    /* ext. time tag 3 control/status */
#define Ext4_Ctl_Sts_Port       0x31    /* ext. time tag 4 control/status */
#define Ext5_Ctl_Sts_Port       0x35    /* ext. time tag 5 control/status */
#define Ext6_Ctl_Sts_Port       0x39    /* ext. time tag 6 control/status */
#define Ext7_Ctl_Sts_Port       0x3d    /* ext. time tag 7 control/status */
/* each of ExtN_Ctl_Sts registers has following bits: */
#define Ext_Intr_Enb            0x04    /* RW Ext. Time Tag N Intr Enable   */
#define Ext_Polarity            0x02    /* RW Ext. Time Tag N polarity 0=^ 1=V*/
#define Ext_Ready               0x01    /* RO Ext. Time Tag N Data Ready if 1*/
#define Trigger_Sim_Ext_Time_Tag 0x08   /* WO bit0 = 1: Fake ext. time tag */
// bits <7:4> of Ext<n>_Ctl_Sts_Port contain the 100nsec
// BCD digit of time tag n

/* READING ANY OF THE FOLLOWING PORTS WILL CLEAR Ext_Ready for Time Tag N */
#define Ext0_Sec10_Usec1_Port    0x40 /* LONG RO ext. time tag 0 msec..usec    */
#define Ext0_Year1_Min1_Port     0x44 /* LONG RO ext. time tag 0 10sec ..10msec*/
#define Ext1_Sec10_Usec1_Port    0x48 /* LONG RO ext. time tag 1 msec..usec    */
#define Ext1_Year1_Min1_Port     0x4c /* LONG RO ext. time tag 1 10sec ..10msec*/
#define Ext2_Sec10_Usec1_Port    0x50 /* LONG RO ext. time tag 2 msec..usec    */
#define Ext2_Year1_Min1_Port     0x54 /* LONG RO ext. time tag 2 10sec ..10msec*/
#define Ext3_Sec10_Usec1_Port    0x58 /* LONG RO ext. time tag 3 msec..usec    */
#define Ext3_Year1_Min1_Port     0x5c /* LONG RO ext. time tag 3 10sec ..10msec*/
#define Ext4_Sec10_Usec1_Port    0x60 /* LONG RO ext. time tag 4 msec..usec    */
#define Ext4_Year1_Min1_Port     0x64 /* LONG RO ext. time tag 4 10sec ..10msec*/
#define Ext5_Sec10_Usec1_Port    0x68 /* LONG RO ext. time tag 5 msec..usec    */
#define Ext5_Year1_Min1_Port     0x6c /* LONG RO ext. time tag 5 10sec ..10msec*/
#define Ext6_Sec10_Usec1_Port    0x70 /* LONG RO ext. time tag 6 msec..usec    */
#define Ext6_Year1_Min1_Port     0x74 /* LONG RO ext. time tag 6 10sec ..10msec*/
#define Ext7_Sec10_Usec1_Port    0x78 /* LONG RO ext. time tag 7 msec..usec    */
```

```
#define Ext7_Year1_Min1_Port      0x7c /* LONG RO ext. time tag 7 10sec ..10msec*/
```

```
/* REGDEF.H 5SEP03 JCK FIX COMMENT ON Ext_Polarity: 0=^, 1=V */
/* REGDEF.H 26JUN98 JCK RAMFIFO port moved from x16 to x18 */
/* PORT ADDRESS DEFINITIONS FOR PCI-SYNCCLOCK32 */
/* ALL ADDRESSES ARE RELATIVE TO CONFIGURED BASE ADDRESS */
/* RO means Read only. RW means read write. WO means Write only. */
#define Sec10_Usec1_Port        0x00 /* LONG RO latch new time & read ms8..us1 bits*/
#define Year1_Min1_Port         0x04 /* LONG RO read latched hr8..min1 bits*/
/* READING ANY OF THE FOLLOWING PORTS WILL CLEAR EXT READY FLAG */
#define Ext_Sec10_Usec1_Port    0x08 /* LONG RO ext. time tag msec..usec    */
#define Ext_Year1_Min1_Port     0x0c /* LONG RO ext. time tag 10sec ..10msec*/
#define Ext_100ns_Port          0x13 /* BYTE RO ext. time tag 100ns .. 0000 */

#define Status_Port             0x10    /* status & interrupt enables */
#define Ext_Ready               1       /* RO Ext. Time Tag Data Ready if 1*/
#define Trigger_Sim_Ext_Time_Tag 1      /* WO bit0 = 1: Fake ext. time tag */
#define Response_Ready          2       /* RO Command Response Ready if 1*/
#define RAM_FIFO_Ready          4       /* RO RAM FIFO data ready */
#define Match                   8       /* RO Match register toggles on Match */
#define Reset_Match             8       /* WO bit3= 1: Reset Match reg */
#define Heartbeat               0x10    /* RO Heartbeat pulse detected if 1 */
#define Reset_Heartbeat         0x10    /* WO bit4 = 1: Reset Heartbeat flag*/
#define Ext_Polarity            0x20    /* RW External event input polarity 0=^ 1=V*/
#define Sync_OK                 0x40    /* RO In-sync to time reference if 1 */

#define Control_Port            0x11
#define Ext_Intr_Enb            0x01    /* RW Ext. Time Tag Intr Enable   */
#define Command_Rdy_Intr_Enb    0x02    /* RW Command Response Ready Intr Enable */
/* the RAM_FIFO_Rdy_Intr_Enb bit is implemented for -RAMFIFO option ONLY  */
#define RAM_FIFO_Rdy_Intr_Enb   0x04    /* RW RAM FIFO Ready Intr Enable */
#define Time_Match_Intr_Enb     0x08    /* RW Time Match Intr Enable   */
#define Heartbeat_Intr_Enb      0x10    /* RW Heartbeat  Intr Enable   */
#define Matchext_Enb            0x20    /* RW Enables Match Extension past 10**3 usec*/

#define Mult_Frac_Port          0x12    /* implemented for -MULFRAC option only */
#define MulFrac2                0x01    /* WO select X2 or /2  */
#define MulFrac4                0x02    /* WO select X4 or /4  */
#define Frac                    0x80    /* WO select fraction speed */
#define Mult                    0x00    /* WO select multiple speed */

#define Match_Usec10_Usec1_Port  0x14    /* WO 10**1,10**0 match usec */

#define Match_Msec1_Usec100_Port 0x15    /* WO 10**3,10**2 match usec */

#define RAM_FIFO_Port            0x18    /* RO RAM FIFO */

#define Dual_Port_Address_Port  0x1A    /* WO Dual Port RAM Address */

#define Dual_Port_Data_Port     0x1E    /* RW Dual Port RAM Data */
```

```
/* DPDEF.H 10JAN03 JCK add DP_Control0:IgnorePPS&Ignore_Code definitions */
/* DPDEF.H 6MAY99 JCK change irig-g code select from 0f to 09 */
/* DPDEF.H 3FEB98 JCK add DP_Match_ht & DP_Match_SS */
/* DPDEF.H 6AUG97 JCK fix DP_CodeSelect_NASA36, add DP_CodeSelect_IRIGA, Binary */
/* DPDEF.H 13JAN97 Set_Ctrx & GPS lock/unlock commands added */
/* DPDEF.H 8JAN97 DP_ctl_rw added */
/* DPDEF.H 24DEC96 DP_Ctr2_ctl..DP_Ctr0_msb added */
/* DPDEF.H 31MAY96 */
#define DP_Command 0xff
#define No_op                    0 /* no operation */
#define Command_Set_Major        2 /* Set clock seconds..days to Major seconds..days */
#define Command_Set_Years        4 /* Set clock years to dual port years  */
#define Command_Set_RAM_FIFO     6 /* Set RAM FIFO external time tag mode */
#define Command_Reset_RAM_FIFO   8 /* Reset RAM FIFO external time tag mode */
#define Command_Empty_RAM_FIFO  10 /* Empty RAM FIFO */
#define Command_Set_Ctr0        12 /* Set 82C54 ctr 2 ("lowrate") params */
#define Command_Set_Ctr1        14 /* Set 82C54 ctr 2 ("heartbeat") params */
#define Command_Set_Ctr2        16 /* Set 82C54 ctr 2 ("rate2") params */
#define Command_Rejam           18 /* Re-jam at start of next second */
#define Command_Spfun1          20 /* Reserved for special function #1 */
#define Command_Spfun2          22 /* Reserved for special function #2 */
#define GPS_Readlock            24 /* lock GPS position buffer for read */
#define GPS_Readunlock          26 /* unlock GPS position buffer to allow update*/
/*   DUAL PORT RAM LOCATIONS 0x00..0x7F ARE READ-ONLY */
#define DP_Extd_Sts 0x00 /* Extended status READ ONLY */
#define DP_Extd_Sts_Nosync 0x01 /* Set if NOT in sync */
#define DP_Extd_Sts_Nocode 0x02 /* Set if selected input code NOT decodeable */
#define DP_Extd_Sts_NoPPS  0x04 /* Set if PPS input invalid */
#define DP_Extd_Sts_NoMajT 0x08 /* Set if major time NOT set since jam */
#define DP_Extd_Sts_NoYear 0x10 /* Set if year NOT set */
#define DP_Code_CtlA 0x0f       /* Control field "A" read data */
#define DP_Code_CtlB 0x10       /* Control field "B" read data */
#define DP_Code_CtlC 0x11       /* Control field "C" read data */
#define DP_Code_CtlD 0x12       /* Control field "D" read data */
#define DP_Code_CtlE 0x13       /* Control field "E" read data */
#define DP_Code_CtlF 0x14       /* Control field "F" read data */
#define DP_Code_CtlG 0x15       /* Control field "G" read data */

#define DP_Control0 0x80 /* Dual Port Ram Address for Control Register */
#define DP_Control0_Leapyear     0x01 /* Current year is leap year*/
#define DP_Control0_CodePriority 0x02 /* Time code input has priority over PPS*/
#define DP_Control0_NegCodePropD 0x04 /* code input prop delay setting is - */
#define DP_Control0_NegPPSPropD  0x08 /* PPS input prop delay setting is - */
#define DP_Control0_IgnorePPS    0x40 /* Ignore 1 PPS input reference */
#define DP_Control0_IgnoreCode   0x80 /* Ignore time code input reference */

#define DP_CodeSelect 0x81              /* Time code input select */
#define DP_CodeSelect_IRIGB 0x0b        /* IRIG-B */
#define DP_CodeSelect_IRIGA 0x0a        /* IRIG-A */
#define DP_CodeSelect_NASA36 0x06       /* NASA36 */
#define DP_CodeSelect_2137 0x07         /* 2137   */
#define DP_CodeSelect_XR3 0x03          /* XR3    */
#define DP_CodeSelect_IRIGG 0x09        /* IRIG-G */
#define DP_CodeSelect_IRIGE 0x0e        /* IRIG-E */

#define DP_LeapSec_Day10Day1 0x82       /* Day (10s & 1s) ending in leap sec*/

#define DP_LeapSec_Day1000Day100 0x83   /* Day (0,100s) ending in leap sec*/

#define DP_CodePropDel_ns100ns10  0x84  /* time code prop. delay 100,10 ns */
#define DP_CodePropDel_us10us1    0x85  /* time code prop. delay 10,1 us */
#define DP_CodePropDel_ms1us100   0x86  /* time code prop. delay 1000,100 us */
#define DP_CodePropDel_ms100ms10  0x87  /* time code prop. delay 100,10 ms */

#define DP_PPS_PropDel_ns100ns10  0x88       /* PPS prop. delay 100,10 ns  */
#define DP_PPS_PropDel_us10us1    0x89       /* PPS prop. delay 10,1 us    */
#define DP_PPS_PropDel_ms1us100   0x8a       /* PPS prop. delay 1000,100 us*/
#define DP_PPS_PropDel_ms100ms10  0x8b       /* PPS prop. delay 100,10 ms  */

#define DP_PPS_Time_ns100ns10     0x8c          /* PPS time 100,10ns */
#define DP_PPS_Time_us10us1       0x8d          /* PPS time 10,1us */
#define DP_PPS_Time_ms1us100      0x8e          /* PPS time 1000,100us */
#define DP_PPS_Time_ms100ms10     0x8f          /* PPS time 100,10ms */
#define DP_Major_Time_s10s1       0x90          /* Major time 10,1second */
```

```
#define DP_Major_Time_m10m1          0x91          /* Major time 10,1minute */
#define DP_Major_Time_h10h1          0x92          /* Major time 10,1hour */
#define DP_Major_Time_d10d1          0x93          /* Major time 10,1 day */
#define DP_Major_Time_d1000d10       0x94           /* Major time 0, 100 day */
#define DP_Year10Year1          0x95               /* 10,1 years */
#define DP_Year1000Year100      0x96               /* 1000,100 years */
#define DP_Codebypass           0X97               /* #frames to validate code */

#define DP_Ctr2_ctl             0x98      /* ctr 2 control word */
#define DP_Ctr1_ctl             0x99      /* ctr 1 control word */
#define DP_Ctr0_ctl             0x9A      /* ctr 0 control word */
#define DP_Ctr2_ctl_sel                     0x80  /* ALWAYS used for ctr 2 */
#define DP_Ctr1_ctl_sel                     0x40  /* ALWAYS used for ctr 1 */
#define DP_Ctr0_ctl_sel                     0x00  /* ALWAYS used for ctr 0 */
#define DP_ctl_rw                           0x30  /* ALWAYS used */
#define DP_ctl_mode0                        0x00  /* Ctrx mode 0 select */
#define DP_ctl_mode1                        0x02  /* Ctrx mode 1 select */
#define DP_ctl_mode2                        0x04  /* Ctrx mode 2 select */
#define DP_ctl_mode3                        0x06  /* Ctrx mode 3 select */
#define DP_ctl_mode4                        0x08  /* Ctrx mode 4 select */
#define DP_ctl_mode5                        0x0A  /* Ctrx mode 5 select */
#define DP_ctl_bin                          0x00  /* Ctrx binary mode select */
#define DP_ctl_bcd                          0x01  /* Ctrx bcd mode select */
#define DP_Ctr2_lsb             0x9B      /* ctr 2 count LSB    */
#define DP_Ctr2_msb             0x9C      /* ctr 2 count MSB    */
#define DP_Ctr1_lsb             0x9D      /* ctr 1 count LSB    */
#define DP_Ctr1_msb             0x9E      /* ctr 1 count MSB    */
#define DP_Ctr0_lsb             0x9F      /* ctr 0 count LSB    */
#define DP_Ctr0_msb             0xA0      /* ctr 0 count MSB    */
#define DP_Battery_Holdoff      0xBD      /* hold off time before sync to 1 PPS */
#define DP_Match_ht             0xBE      /* match time .ht (BCD)(VME -EXTMATCH opt)*/
#define DP_Match_SS             0xBF      /* match time ss (BCD)(VME -EXTMATCH opt)*/
#define DP_Match_MM             0xCB      /* -EXTMATCH option: match time mm (BCD)  */
#define DP_Match_HH             0xCC      /* -EXTMATCH option: match time hh (BCD)  */
#define DP_Match_D1D0           0xCD      /* -EXTMATCH option: match time day (10,1 digits) (BCD) */
#define DP_Match_XD2            0xCE      /* -EXTMATCH option: match time don't care, 100 days digit
#define DP_GPS_TFOM_GEN         0xC9      /* 3..0 generated TFOM decade */
#define DP_GPS_4430_Scale_Leap_Ind 0xCA /* see manual STANAG4430 programming */
```

```
/* gpsdef.h 24mar03 add GPS commands for open,short bits in RcvrStatusM12 */
/* gpsdef.h 4mar03  correct antenna ok, open,short bits in RcvrStatusM12 */
/* gpsdef.h 16jan03 add DP_GPS_DOP_Type,DP_GPS_RcvrStatus defs for M12 */
/* gpsdef.h 24nov00 add DP_GPS_PosFracSec */
/* gpsdef.h 24aug00 add GPS_Disable_Pos_Hold_Command */
/* gpsdef.h 12aug00 add DP_GPS_Sawtooth,GPS_Command,GPS cable comp  */
/* gpsdef.h 5jan99 add DP_GPS_CH0..DP_GPS_RcvrStatus */
/* GPSDEF.H 28apr98 DP_GPS_Altbin3124..DP_GPS_Altbin0700 added  */
/* GPSDEF.H 31jul97 DP_GPS_UpdateFlag..DP_GPS_RTCM_Ack  */
/* GPSDEF.H 8jan97 JCK add DP_HQ_TFOM */
/* GPSDEF.H 7jun96 JCK */
#define DP_GPS_Semaphore          0xA3     /* 0=unlocked,1=read lock,-1=wr lock */

#define DP_GPS_Status             0xA4
#define DP_GPS_Status_Sats        0x0F     /* lo order nibble: BIN # sats tracked */
#define DP_GPS_Status_South       0x80     /* Bit 7: 1=South,0=North latitude */
#define DP_GPS_Status_West        0x40     /* Bit 6: 1=West,0=East longitude */
#define DP_GPS_Status_Nav         0x10     /* Bit 4: 1=Navigate, 0=Acquiring */

#define DP_GPS_Lat_MinEM34        0xA5     /* BCD 10-3,10-4 min latitude */
#define DP_GPS_Lat_MinEM12        0xA6     /* BCD 10-1,10-2 min latitude */
#define DP_GPS_Lat_MinE10         0xA7     /* BCD 10+1,10+0 min latitude */
#define DP_GPS_Lat_DegE10         0xA8     /* BCD 10+1,10+0 deg latitude */
#define DP_GPS_Lon_MinEM34        0xA9     /* BCD 10-3,10-4 min longitude */
#define DP_GPS_Lon_MinEM12        0xAA     /* BCD 10-1,10-2 min longitude */
#define DP_GPS_Lon_MinE10         0xAB     /* BCD 10+1,10+0 min longitude */
#define DP_GPS_Lon_DegE10         0xAC     /* BCD 10+1,10+0 deg longitude */
#define DP_GPS_Lon_DegE32         0xAD     /* BCD 10+3,10+2 deg longitude */
#define DP_GPS_SOG0M1             0xAE     /* BCD speed over ground, 1,10-1 m/sec */
#define DP_GPS_SOG21              0xAF     /* BCD speed over ground, 100,10 m/sec */
#define DP_GPS_HOG0M1             0xB0     /* BCD heading over ground, 1,10-1 deg */
#define DP_GPS_HOG21              0xB1     /* BCD heading over ground, 100,10 deg */
#define DP_HQ_TFOM                0xB2     /* HaveQuick Time Figure of Merit */
#define DP_GPS_UpdateFlag         0xB3     /* set non-0 by board when GPS data updated */
#define DP_GPS_Latbin3124         0xB4     /* Oncore: ms byte lat arc milliseconds */
#define DP_GPS_Latbin2316         0xB5     /* Oncore: <23:16> lat arc milliseconds */
#define DP_GPS_Latbin1508         0xB6     /* Oncore: <15:08> lat arc milliseconds */
#define DP_GPS_Latbin0700         0xB7     /* Oncore: ls byte lat arc milliseconds */
#define DP_GPS_Lonbin3124         0xB8     /* Oncore: ms byte lon arc milliseconds */
#define DP_GPS_Lonbin2316         0xB9     /* Oncore: <23:16> lon arc milliseconds */
#define DP_GPS_Lonbin1508         0xBA     /* Oncore: <15:08> lon arc milliseconds */
#define DP_GPS_Lonbin0700         0xBB     /* Oncore: ls byte lon arc milliseconds */
#define DP_GPS_RTCM_Ack           0xBC     /* Oncore: Ack count of RTCM104 messages received */
#define DP_GPS_Altbin3124         0xC0     /* Oncore: ms byte alt above geoid in cm.*/
#define DP_GPS_Altbin2316         0xC1     /* Oncore: <23:16> alt above geoid in cm.*/
#define DP_GPS_Altbin1508         0xC2     /* Oncore: <15:08> alt above geoid in cm.*/
#define DP_GPS_Altbin0700         0xC3     /* Oncore: ls byte alt above geoid in cm.*/

#define DP_GPS_Command            0xD2     /* GPS receiver commands: */
#define GPS_Autosurvey_Command    0x01     /* init auto-survey */
#define GPS_Cable_Length_Command  0x02     /* set cable compensation */
#define GPS_Disable_Pos_Hold_Command 0x03  /* disable position hold mode */
#define GPS_Set_Hold_Position_Command 0x04 /* specify hold position */
#define GPS_Set_1_1_2001_Command  0x05     /* init rcvr clock to 1-1-2001 */
#define GPS_Disable_TRAIM         0x06     /* disable TRAIM              */
#define GPS_Enable_TRAIM          0x07     /* enable TRAIM               */
#define GPS_Set_1_1_2019_Command  0x08     /* init rcvr clock to 1-1-2019 */
#define GPS_Set_1_1_2038_Command  0x09     /* init rcvr clock to 1-1-2038 */
#define GPS_Set_Default_Commmand  0x0A     /* init rcvr settings to default */
#define GPS_Enable_Pos_Hold_Command 0x0B   /* enable position hold mode */


#define DP_GPS_cablcomp3          0xD3     /* cable length compensation 2**31..2**24 ns */
#define DP_GPS_cablcomp2          0xD4     /* cable length compensation 2**23..2**16 ns */
#define DP_GPS_cablcomp1          0xD5     /* cable length compensation 2**15..2**8 ns */
#define DP_GPS_cablcomp0          0xD6     /* cable length compensation 2**7..2**0 ns */
#define DP_GPS_SawtoothErr        0xd7     /* 1PPS pulse sawtooth error */
#define DP_GPS_UTC_Offset         0x18     /* GPS/UTC time offset */
#define DP_GPS_RcvrType           0x19     /* 0=not def., 1=Mot Oncore UT+ */
                                           /* type 2= Mot M12 (Dynamic) */
                                           /* type 5= Mot M12 (Timing) */
#define DP_GPS_ChanDat            0x20     /* old name for DP_GPS_CH0 */
/* DP_GPS_CH0..CH7 valid for units shipped 6 Mar 1999 and later */
#define DP_GPS_CH0              0x20    /* 4 byte channel 0 info */
#define DP_GPS_CH1              0x24    /* 4 byte channel 1 info */
#define DP_GPS_CH2              0x28    /* 4 byte channel 2 info */
```

```
#define DP_GPS_CH3              0x2c    /* 4 byte channel 3 info */
#define DP_GPS_CH4              0x30    /* 4 byte channel 4 info */
#define DP_GPS_CH5              0x34    /* 4 byte channel 5 info */
#define DP_GPS_CH6              0x38    /* 4 byte channel 6 info */
#define DP_GPS_CH7              0x3c    /* 4 byte channel 7 info */
/* DP_GPS_CH8..11 will be 0 for 8 channel receivers */
#define DP_GPS_CH8              0x40    /* 4 byte channel 8 info */
#define DP_GPS_CH9              0x44    /* 4 byte channel 9 info */
#define DP_GPS_CH10             0x48    /* 4 byte channel 10 info */
#define DP_GPS_CH11             0x4c    /* 4 byte channel 11 info */


#define DP_GPS_SatID    0 /* offset 0 in 4 byte field for satellite ID */

#define DP_GPS_ChMode   1 /* offset 1 in 4 byte field for channel mode */
/* definition of DP_GPS_ChMode values for ONCORE receivers  */
#define DP_GPS_ChMode_Code_Search  0 /* Code search mode if ChMode=0 */
#define DP_GPS_ChMode_Code_Acquire 1 /* Code acquire mode if ChMode=1 */
#define DP_GPS_ChMode_AGCset 2 /* AGC Set mode if ChMode=2 */
#define DP_GPS_ChMode_PreqAcq 3 /* Pre-acquire mode if ChMode=3 */
#define DP_GPS_ChMode_BitSyncDet 4 /* Bit Sync Detect mode if ChMode=4 */
#define DP_GPS_ChMode_MsgSyncDet 5 /* Msg Sync Detect mode if ChMode=5 */
#define DP_GPS_ChMode_SatTimeAvail 6 /* Sat time avail mode if ChMode=6 */
#define DP_GPS_ChMode_EphemAcq 7 /* Ephemeris acq mode if ChMode=7 */
#define DP_GPS_ChMode_AvailPos 8 /* Available for position mode if ChMode=8 */

#define DP_GPS_ChCNo    2 /* offset 2 in 4 byte field for channel C/no ratio */

#define DP_GPS_ChStatus 3 /* offset 3 n 4 byte field for channel status flags */
/* bit assignments for DP_GPS_ChStatus: for ONCORE GPS receivers*/
#define DP_GPS_ChStatus_PosFix   0x80 /* Using for position fix flag */
#define DP_GPS_ChStatus_SatMom   0x40 /* Satellite momentum alert flag */
#define DP_GPS_ChStatus_SatAntiSpoof 0x20 /* Satellite Anti-Spoof flag */
#define DP_GPS_ChStatus_SatUnhealthy 0x10 /* Satellite unhealthy flag */
#define DP_GPS_ChStatus_SatInaccurate 0x08 /* Satellite inaccurate flag */
#define DP_GPS_ChStatus_Spare        0x04 /* spare flag */
#define DP_GPS_ChStatus_TimeSoln     0x02 /* Using sat for time solution flag */
#define DP_GPS_ChStatus_ParityErr    0x01 /* Parity error flag */
/* bit assignments for DP_GPS_ChStatus: for M12 GPS receivers*/
#define DP_GPS_ChStatusM12_PosFix       0x80 /* Using for position fix flag */
#define DP_GPS_ChStatusM12_SatMom       0x40 /* Satellite momentum alert flag */
#define DP_GPS_ChStatusM12_SatAntiSpoof 0x20 /* Satellite Anti-Spoof flag */
#define DP_GPS_ChStatusM12_SatUnhealthy 0x10 /* Satellite unhealthy flag */
#define DP_GPS_ChStatusM12_SatAccu      0x0f /* <3:0> encode satellite URA*/

/* DP_GPS_DOP_Type..DP_GPS_RcvrStatus defined in units shipped 5jan2000 & later */

#define DP_GPS_DOP_Type   0x50  /* DOP type & antenna connection info */
/* bit assignments for DP_GPS_DOP_Type: (for Oncore receivers)*/
#define DP_GPS_DOP_Type_AntOpen  0x80  /* antenna undercurrent flag */
#define DP_GPS_DOP_Type_AntShort 0x40  /* antenna overcurrent flag */
#define DP_GPS_DOP_Type_AutoSurv 0x20  /* Auto-survey mode flag     */
#define DP_GPS_DOP_Type_Spare4   0x10  /* spare flag     */
#define DP_GPS_DOP_Type_Spare3   0x08  /* spare flag     */
#define DP_GPS_DOP_Type_Spare2   0x04  /* spare flag     */
#define DP_GPS_DOP_Type_Spare1   0x02  /* spare flag     */
#define DP_GPS_DOP_Type_DOP2D    0x01  /* DOP is 2D if set,3D if not set flag */
/* bit assignments for DP_GPS_DOP_Type: (for M12 receivers)*/
#define DP_GPS_DOP_TypeM12_3DFix    0xe0 /* <7:5>=111 =3D FIX */
#define DP_GPS_DOP_TypeM12_2DFix    0xc0 /* <7:5>=110 =2D FIX */
#define DP_GPS_DOP_TypeM12_PropMode 0xa0 /* <7:5>=101 =Propagate Mode*/
#define DP_GPS_DOP_TypeM12_PosnHold 0x80 /* <7:5>=100 =Position Hold */
#define DP_GPS_DOP_TypeM12_AcqSats  0x60 /* <7:5>=011 =Acq. Satellites*/
#define DP_GPS_DOP_TypeM12_BadGeom  0x40 /* <7:5>=010 =Bad Geometry  */
#define DP_GPS_DOP_TypeM12_Reservea 0x20 /* <7:5>=001 =Reserved      */
#define DP_GPS_DOP_TypeM12_Reserveb 0x00 /* <7:5>=000 =Reserved      */
#define DP_GPS_DOP_TypeM12_Reserve4 0x10 /* <4:2>      Reserved      */
#define DP_GPS_DOP_TypeM12_Reserve3 0x08 /* <4:2>      Reserved      */
#define DP_GPS_DOP_TypeM12_Reserve2 0x04 /* <4:2>      Reserved      */
#define DP_GPS_DOP_TypeM12_FasAcqPo 0x02 /* Fast Acquisition Position*/
#define DP_GPS_DOP_TypeM12_FResRAW  0x01 /* Filt Reset RAW GPS Soln  */

#define DP_GPS_RcvrStatus 0x51  /* Receiver status flags */
/* bit assignments for DP_GPS_RcvrStatus: for ONCORE GPS receivers */
#define DP_GPS_RcvrStatus_PosProp  0x80  /* Position propagate mode flag */
#define DP_GPS_RcvrStatus_PoorGeom 0x40 /* Poor geometry (DOP>12) flag */
#define DP_GPS_RcvrStatus_3DFix    0x20 /* 3D position fix       flag */
#define DP_GPS_RcvrStatus_2DFix    0x10 /* 2D position fix       flag */
#define DP_GPS_RcvrStatus_ACQHold  0x08 /* Acquiring sats/PosHold flag */
#define DP_GPS_RcvrStatus_DiffFix  0x04 /* Differential fix      flag */
```

```
#define DP_GPS_RcvrStatus_InsVisib 0x02 /* Insuffient vis sats    flag */
#define DP_GPS_RcvrStatus_BadAlman 0x01 /* Bad Almanac Flag       flag */
/* bit assignments for DP_GPS_RcvrStatus: for M12 GPS receivers */
#define DP_GPS_RcvrStatusM12_Cold      0x80 /* Cold start                  */
#define DP_GPS_RcvrStatusM12_Diff      0x40 /* Differential fix            */
#define DP_GPS_RcvrStatusM12_PosLock   0x20 /* Position Lock               */
#define DP_GPS_RcvrStatusM12_Autosurv  0x10 /* Autosurvey mode             */
#define DP_GPS_RcvrStatusM12_InsVisSat 0x08 /* Insufficient visible sats   */
#define DP_GPS_RcvrStatusM12_AntennaOK 0x00 /* <2:1>=00 Ant DC Current OK  */
#define DP_GPS_RcvrStatusM12_AntShort  0x02 /* <2:1>=01 Ant DC Overcurrent */
#define DP_GPS_RcvrStatusM12_AntOpen   0x04 /* <2:1>=10 Ant DC Undercurrent*/
#define DP_GPS_RcvrStatusM12_CodLocInt 0x01 /* 1=Int,0=Ext Code Location   */

#define DP_GPS_PosFracSec 0x5D   /* most sig byte of pos fix time frac sec */
```

```
/* curtime.c 21feb2003 display current time from pci-syncclock as 2 8digit longword
 * Copyright (c) 2002 PLX Technology, Inc.
 * PLX Technology Inc. licenses this software under specific terms and
 * conditions.  Use of any of the software or derviatives thereof in any
 * product without a PLX Technology chip is strictly prohibited. */
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "\pci\sync\sdktest\regdef.h" // defines SYNCCLOCK addresses relative to BASE
signed char SelectSyncclock(HANDLE *pHandle,unsigned long *VaPciBar);
signed char DeselectSyncclock(HANDLE *pHandle);
void main(void)
{
        HANDLE Handle;
    unsigned long timehi,timelo,BASE=-1;
    printf("\n\n\t\t curtime.c Sample Application 21feb2003\n\n");
    if (SelectSyncclock(&Handle,&BASE)!=1)
                {printf("*ERROR* Could not get handle to PCI-SYNCCLOCK \n");return;}
        while (1)
                {timelo=*(volatile unsigned long*)(BASE+Sec10_Usec1_Port);
                timehi=*(volatile unsigned long*)(BASE+Year1_Min1_Port);
                printf("%08x %08x \n",timehi,timelo);}
        DeselectSyncclock(&Handle);
}
```

```
 /* mattst.cpp 21feb2003 jck tests Match function for PCI-SYNCCLOCK
 * Copyright (c) 2002 PLX Technology, Inc.
 * PLX Technology Inc. licenses this software under specific terms and
 * conditions.  Use of any of the software or derviatives thereof in any
 * product without a PLX Technology chip is strictly prohibited. */
#include <stdio.h>
#include <conio.h>
#include <Windows.h>
#include "\pci\sync\sdktest\regdef.h" // defines register addresses relative to BASE
#include "\pci\sync\sdktest\dpdef.h" // defines dual port RAM addresses
signed char SelectSyncclock(HANDLE *pHandle,unsigned long *VaPciBar);
signed char DeselectSyncclock(HANDLE *pHandle);
void Set_Dual_Port_RAM(unsigned char addr, unsigned char value);
unsigned long BASE=-1,timehi,timelo,xtime,xmsec,xusec,i,match;
HANDLE Handle;
unsigned char matstart,matus0,matus1,matus2,matus3;
void main(void)
{
        printf("\n\n\t\t Mattst Sample Application 21Feb2003\n\n");
    //set BASE=virtual address base of syncclock registers
        if (SelectSyncclock(&Handle,&BASE)!=1)
                {printf("*ERROR* Could not get handle to PCI-SYNCCLOCK \n");return;}
        Set_Dual_Port_RAM (DP_Match_ht,0xff);  // don't care for .ht digits
        Set_Dual_Port_RAM (DP_Match_SS,0xff);  // don't care for ss digits
        //MAKE SURE MATCH DOESN'T TOGGLE WHILE NON-BCD MATCH TIME IS SET
        *(volatile unsigned char*)(BASE+Match_Msec1_Usec100_Port)=0xff; // match time = ffff
        *(volatile unsigned char*)(BASE+Match_Usec10_Usec1_Port)=0xff;
        matstart=*(volatile unsigned char*)(BASE+Status_Port);
        //starting match flag is Match bit
        while ((*(volatile unsigned long*)(BASE+Sec10_Usec1_Port) & 0x0ffff) < 0x09000) {;}
        while ((*(volatile unsigned long*)(BASE+Sec10_Usec1_Port) & 0x0ffff) >= 0x09000) {;}
        if ( ( ((*(volatile unsigned char*)(BASE+Status_Port)) ^matstart) & Match)!=0)
        {printf("Error - Status_Port Match bit toggled while non-BCD match time \n");}
        else
        {
        printf(" OK  - Match bit did not toggle while inhibited \n");
    for (match=9999;match!=9900;match=((match+99)%10000) )
                {
        matus3=(unsigned char)( match/1000);/* get 1000 usec BCD digit */
        matus2=(unsigned char)((match%1000)/100); /* get 100 usec BCD digit */
        matus1=(unsigned char)((match%100)/10);    /* get 10 usec BCD digit */
        matus0=(unsigned char)(match%10);        /* get 1 usec BCD digit */
        // set new BCD match time
                *(volatile unsigned char*)(BASE+Match_Msec1_Usec100_Port)=matus3*16+matus2;
                *(volatile unsigned char*)(BASE+Match_Usec10_Usec1_Port)=matus1*16+matus0;
                *(volatile unsigned char*)(BASE+Status_Port)=Reset_Match;  // Reset Match Status
                while ( !(*(volatile unsigned char*)(BASE+Status_Port)&Match) ){;} //wait for ti
                        xtime=*(volatile unsigned long*)(BASE+Ext_Sec10_Usec1_Port) & 0x0000ffff
                        xmsec=(xtime&0x0f000)/0x1000;
                        xusec=(((xtime&0x0f00)/0x100)*100)+(((xtime&0x0f0)/0x10)*10)+(xtime&0x0f
           if ((xusec != (match %1000)) || (xmsec!= (match/1000)))
           {printf(".%03ld%03ld %04d \n " ,xmsec,xusec,match);}
          }
        printf (" mattst completed \n");
    }
        *(volatile unsigned char*)(BASE+Match_Msec1_Usec100_Port)=0;  // set new BCD match time
        *(volatile unsigned char*)(BASE+Match_Usec10_Usec1_Port)=0;  // for toggle on 10msec bou
        DeselectSyncclock(&Handle); // close device and free up virtual address window
        return;
}
void Set_Dual_Port_RAM(unsigned char addr, unsigned char value)
{
unsigned char status;
*(volatile unsigned char*)(BASE+Dual_Port_Address_Port)=addr;
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data read
status=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
*(volatile unsigned char*)(BASE+Dual_Port_Data_Port)=value;
status=*(volatile unsigned char*)(BASE+Status_Port);
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data write
return ;
}
```

```
/* dpdump.c 21feb2003 jck demonstrate selecting syncclock and reading dual port RAM
 * Copyright (c) 2002 PLX Technology, Inc.
 * PLX Technology Inc. licenses this software under specific terms and
 * conditions.  Use of any of the software or derviatives thereof in any
 * product without a PLX Technology chip is strictly prohibited. */
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "\pci\sync\sdktest\regdef.h" // defines SYNCCLOCK addresses relative to BASE
signed char SelectSyncclock(HANDLE *pHandle,unsigned long *VaPciBar);
signed char DeselectSyncclock(HANDLE *pHandle);
volatile unsigned char Read_Dual_Port_RAM(unsigned char addr);
unsigned long BASE=-1;
void main(void)
{
        HANDLE Handle;
        unsigned char i,j;
        printf("\n\n\t\t Dual Port Ram Access Sample Application 21Feb2003\n\n");
    //set BASE=virtual address base of syncclock registers
        if (SelectSyncclock(&Handle,&BASE)!=1)
                {printf("*ERROR* Could not get handle to PCI-SYNCCLOCK \n");return;}
        for (j=0;j<=15;j++) // display 16 rows with 16 DP RAM contents per row
                {
                printf ("%1x0   ",j); // display DP ram address of first byte of row
                for (i=0;i<=15;i++) printf ("%02x ",Read_Dual_Port_RAM(  (unsigned char)((16*j)+
                printf(" \n");
                }
        // close device and free up virtual address window
        DeselectSyncclock(&Handle);
    return;
}
volatile unsigned char Read_Dual_Port_RAM(unsigned char addr)
{unsigned char response;
// for multi-thread applications, make sure only 1 thread
// at a time accesses dual port RAM (by using mutex or other method)
*(volatile unsigned char*)(BASE+Dual_Port_Address_Port)=addr;
response=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data read
response=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
response=*(volatile unsigned char*)(BASE+Dual_Port_Data_Port);
return response;
}
```

```
/* setmajor.c 22feb2003 jck set SYNCCLOCK major time
 * Copyright (c) 2002 PLX Technology, Inc.
 * PLX Technology Inc. licenses this software under specific terms and
 * conditions.  Use of any of the software or derviatives thereof in any
 * product without a PLX Technology chip is strictly prohibited. */
#include <stdio.h>
#include <conio.h>
#include <Windows.h>
#include "\pci\sync\sdktest\regdef.h" // defines register addresses relative to BASE
#include "\pci\sync\sdktest\dpdef.h" // defines dual port RAM addresses
signed char SelectSyncclock(HANDLE *pHandle,unsigned long *VaPciBar);
signed char DeselectSyncclock(HANDLE *pHandle);
void Set_Dual_Port_RAM(unsigned char addr, unsigned char value);
unsigned long BASE=-1,timehi,timelo,i;
void main(void)
{
        HANDLE Handle;
        printf("\n\n\t\t Setmajor Sample Application 22Feb2003\n\n");
    //set BASE=virtual address base of syncclock registers
        if (SelectSyncclock(&Handle,&BASE)!=1)
                {printf("*ERROR* Could not get handle to PCI-SYNCCLOCK \n");return;}
        Set_Dual_Port_RAM (DP_Major_Time_d1000d10, 0x03);
        Set_Dual_Port_RAM (DP_Major_Time_d10d1,0x65);
        Set_Dual_Port_RAM (DP_Major_Time_h10h1,0x23);
        Set_Dual_Port_RAM (DP_Major_Time_m10m1,0x59);
        Set_Dual_Port_RAM (DP_Major_Time_s10s1,0x55);
        // Command Set Major is used when 1PPS reference is used w/o on-board GPS.
        // the new time will appear at the beginning of the next second.
        // the fractional second time will not jump, only major time.
        Set_Dual_Port_RAM (DP_Command,Command_Set_Major);  /* Set 365 23:59:55 */
        // hint: To set SYNCCLOCK time immediately (actually takes about 100usec)
        //   to DDD HH:MM:ss.000000 (where DDD HH:MM:ss=DP_Major_Time)
        //      use Command_Rejam instead of Command_Set_Major
        // Use Command_Rejam instead of Command_Setmajor when the SYNCCLOCK
        // is free-running with no input reference:
        //Set_Dual_Port_RAM (DP_Command,Command_Rejam);  /* Set 365 23:59:55 */
        Sleep(1); // wait 1 msec for Command_Set_Major to be proceesed by SYNCCLOCK
        for (i=1;i<10;i++)
                {
                timelo=*(volatile unsigned long*)(BASE+Sec10_Usec1_Port);
                timehi=*(volatile unsigned long*)(BASE+Year1_Min1_Port);
                printf("%08lx %08lx \n" ,timehi,timelo);
                Sleep(200); // wait 200 msec
                }
        // close device and free up virtual address window
        DeselectSyncclock(&Handle);
    return;
}
void Set_Dual_Port_RAM(unsigned char addr, unsigned char value)
{
// If multiple threads are accesing dual port RAM, use a mutex or other
// mechanism to ensure that any given thread completes its' Set_Dual_Port_RAM
// call (or Read_Dual_Port_RAM call) before another thread starts DP Ram access
unsigned char status;
*(volatile unsigned char*)(BASE+Dual_Port_Address_Port)=addr;
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data read
status=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
*(volatile unsigned char*)(BASE+Dual_Port_Data_Port)=value;
status=*(volatile unsigned char*)(BASE+Status_Port);
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data write
return ;
}
```

```
/* heart.cpp 22feb2003 jck demo changing SYNCCLOCK heartbeat rate
 * Copyright (c) 2002 PLX Technology, Inc.
 * PLX Technology Inc. licenses this software under specific terms and
 * conditions.  Use of any of the software or derviatives thereof in any
 * product without a PLX Technology chip is strictly prohibited. */
#include <stdio.h>
#include <conio.h>
#include <Windows.h>
#include "\pci\sync\sdktest\regdef.h" // defines register addresses relative to BASE
#include "\pci\sync\sdktest\dpdef.h" // defines dual port RAM addresses
signed char SelectSyncclock(HANDLE *pHandle,unsigned long *VaPciBar);
signed char DeselectSyncclock(HANDLE *pHandle);
void Set_Dual_Port_RAM(unsigned char addr, unsigned char value);
unsigned long BASE=-1,timehi,timelo,i;
void main(void)
{
        HANDLE Handle;
        printf("\n\n\t\t Heart Sample Application 21Feb2003\n\n");
    //set BASE=virtual address base of syncclock registers
        if (SelectSyncclock(&Handle,&BASE)!=1)
                {printf("*ERROR* Could not get handle to PCI-SYNCCLOCK \n");return;}
        // If board is found, set heartbeat rate to 50 PPS
        Set_Dual_Port_RAM (DP_Ctr1_lsb, (3000000/50)%256);
        Set_Dual_Port_RAM (DP_Ctr1_msb, (3000000/50)/256);
        // New rate setting is copied from DP Ram to counter with
        // Command_Rejam (which also aligns with second boundary) or
        // Command_Set_Ctr1 (which just copies to counter).
        Set_Dual_Port_RAM (DP_Command,Command_Rejam);
        // close device and free up virtual address window
        DeselectSyncclock(&Handle);
    return;
}
void Set_Dual_Port_RAM(unsigned char addr, unsigned char value)
{
unsigned char status;
// If multiple threads are accesing dual port RAM, use a mutex or other
// mechanism to ensure that any given thread completes its' Set_Dual_Port_RAM
// call (or Read_Dual_Port_RAM call) before another thread starts DP Ram access
*(volatile unsigned char*)(BASE+Dual_Port_Address_Port)=addr;
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data read
status=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
*(volatile unsigned char*)(BASE+Dual_Port_Data_Port)=value;
status=*(volatile unsigned char*)(BASE+Status_Port);
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data write
return ;
}
```

```
/* gpsbin.c 7mar2003 display GM GPS lat/lon/height data
 * Copyright (c) 2002 PLX Technology, Inc.
 * PLX Technology Inc. licenses this software under specific terms and
 * conditions.  Use of any of the software or derviatives thereof in any
 * product without a PLX Technology chip is strictly prohibited. */
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "\pci\sync\sdktest\regdef.h" // defines SYNCCLOCK addresses relative to BASE
#include "\pci\sync\sdktest\gpsdef.h" // defines SYNCCLOCK addresses relative to BASE
signed char SelectSyncclock(HANDLE *pHandle,unsigned long *VaPciBar);
signed char DeselectSyncclock(HANDLE *pHandle);
unsigned long BASE=-1;
volatile unsigned char Read_Dual_Port_RAM(unsigned char addr);
volatile unsigned char Set_Dual_Port_RAM(unsigned char addr, unsigned char value);
void main(void)
{
        HANDLE Handle;
    int latdeg,latsec,latmin,latmsec,londeg,lonmin,lonsec,lonmsec,status,rtcmack;
        int altmeters,altfrac;
    char EW,NS,NA;
    int long latbin,lonbin,altbin;
    printf("\n\n\t\t -GM option lat/lon/alt Sample Application 7MAR2003\n\n");
    if (SelectSyncclock(&Handle,&BASE)!=1)
                {printf("*ERROR* Could not get handle to PCI-SYNCCLOCK \n");return;}
        while (1)
     {  while ( (Read_Dual_Port_RAM(DP_GPS_UpdateFlag) & 0x0ff)==0) {;} /* wait for update */
        status=Read_Dual_Port_RAM(DP_GPS_Status);
        rtcmack=Read_Dual_Port_RAM(DP_GPS_RTCM_Ack)&0x00ff; // counts RTCM msgs if DF opt
        latbin=Read_Dual_Port_RAM(DP_GPS_Latbin0700)&0x000000ff; // lsb is 1 milliarcsecond
        latbin=latbin+( (long)((Read_Dual_Port_RAM(DP_GPS_Latbin1508)&0x000000ff))<<8);
        latbin=latbin+( (long)((Read_Dual_Port_RAM(DP_GPS_Latbin2316)&0x000000ff))<<16);
        latbin=latbin+( (long)((Read_Dual_Port_RAM(DP_GPS_Latbin3124)&0x000000ff))<<24);
        if (latbin<0) {NS='S';latbin=-latbin;} //negative latbin means S latitude
        else NS='N';
        latdeg=latbin/3600000; //1 degree=60arcmin=3600arcsec=3600000 milliarcseconds
        latmin=(latbin%3600000)/60000; //1 arcmin=60arcsec=60000 milliarcseconds
        latsec=(latbin/1000)%60; // 1 arc sec = 1000 milliarcseconds
        latmsec=latbin%1000;
                // now get longitude
        lonbin=Read_Dual_Port_RAM(DP_GPS_Lonbin0700)&0x000000ff;
        lonbin=lonbin+ ((long)((Read_Dual_Port_RAM(DP_GPS_Lonbin1508)&0x000000ff))<<8);
        lonbin=lonbin+ ((long)((Read_Dual_Port_RAM(DP_GPS_Lonbin2316)&0x000000ff))<<16);
        lonbin=lonbin+ ((long)((Read_Dual_Port_RAM(DP_GPS_Lonbin3124)&0x000000ff))<<24);
        if (lonbin<0) {EW='W';lonbin=-lonbin;} // negative lonbin means W longitude
        else EW='E';
        londeg=(lonbin/3600000); //1 degree=60arcmin=3600arcsec=3600000 milliarcseconds
        lonmin=(lonbin%3600000)/60000; //1 arcmin=60arcsec=60000 milliarcseconds
        lonsec=(lonbin/1000)%60;  // 1 arc sec = 1000 milliarcseconds
        lonmsec=(lonbin%1000);
                // now get altitude above geoid in cm
        altbin=Read_Dual_Port_RAM(DP_GPS_Altbin0700)&0x000000ff;
        altbin=altbin+ ((long)((Read_Dual_Port_RAM(DP_GPS_Altbin1508)&0x000000ff))<<8);
        altbin=altbin+ ((long)((Read_Dual_Port_RAM(DP_GPS_Altbin2316)&0x000000ff))<<16);
        altbin=altbin+ ((long)((Read_Dual_Port_RAM(DP_GPS_Altbin3124)&0x000000ff))<<24);
        altmeters=altbin/100;
                altfrac=altbin%100;
        if (status&DP_GPS_Status_Nav)   NA='N'; else NA='A';
        printf("%c sats:%d RTCM Ack:%03d %c%02d %02d'%02d.%03d\"  %c%03d %02d'%02d.%03d\"",
         NA,status&0x0f,rtcmack,NS,latdeg,latmin,latsec,latmsec,
         EW,londeg,lonmin,lonsec,lonmsec);
                printf("alt: %5d.%03dM\n",altmeters,altfrac);
                Set_Dual_Port_RAM(DP_GPS_UpdateFlag, 0); /* clear for next time */
     }
}
volatile unsigned char Read_Dual_Port_RAM(unsigned char addr)
{unsigned char response;
// for multi-thread applications, make sure only 1 thread
// at a time accesses dual port RAM (by using mutex or other method)
*(volatile unsigned char*)(BASE+Dual_Port_Address_Port)=addr;
response=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data read
response=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
response=*(volatile unsigned char*)(BASE+Dual_Port_Data_Port);
return response;
}
volatile unsigned char Set_Dual_Port_RAM(unsigned char addr, unsigned char value)
{
```

```
char status;
*(volatile unsigned char*)(BASE+Dual_Port_Address_Port)=addr;
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data read
status=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
status=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
*(volatile unsigned char*)(BASE+Dual_Port_Data_Port)=value;
status=*(volatile unsigned char*)(BASE+Status_Port);
status=*(volatile unsigned char*)(BASE+Status_Port); //short delay for data write
status=*(volatile unsigned char*)(BASE+Status_Port); //doing DMA inside PCI-SYNCCLOCK
return status;
}
```

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP