Internal Technical Report

DSP programs for delay setting in GMRT Mark IV correlator

Prepared by : D. Anish Roshi[1] and Sylvain Alliot[2]
Date: 20-12-99
[1]NCRA, Pune, [2]Presently at NFRA, Netherlands

This is a report on the DSP code used for setting delays in mark IV correlator. The main program is dly_set.dsp, which communicates with the host computer. The two interrupt service routines (ISR) used for delay setting are the dly_stai.dsp and dly_ffti.dsp. These two ISRs are evoked at the occurrence of "sta" and "fft" interrupts respectively. Header files required for these routines are also included in the report. All these routines are modified version of the programs used in mark III correlator. For further details refer internal technical report R00126.

[redacted] DELAY-H

```
.CONST DLY_NSE = 0Xffff;
.CONST DLY_INC = 0X0102;
.CONST DLYMSK1 = 0X07ff;
.CONST DLYMSK2 = 0X1800;
.CONST stacount = 0x200a;
.CONST staflg = 0x200b;
.CONST noofcmd = 0x200c;
.CONST ldflag  = 0x200d;
.CONST bankad  = 0x200e;
.CONST hstacnt  = 0x200f;
.CONST packet_len = 0x2010;
.var/seg=int_dm/ram/abs=0x3800 dpc_adr[10],dly_adr[10];
.var/seg=int_dm/ram/circ dlyval1[10];
.var/seg=int_dm/ram/circ dlyval2[10];
.var/seg=int_dm/ram/circ dlyadr[10];
.var/seg=int_dm/ram/circ dlyinc[128];
{.var/seg=int_dm/ram tststa;
.var/seg=int_dm/ram ldflag;}
.var/seg=int_dm/ram inbuf[256];
```

DLYDPC.H

```
--- ADDRESSES ----------------------------------------}
CONST  CRDSEL          = 0X3000;

CONST  DSPIO           = 0X3100;
CONST  STWR            = 0X3200;

CONST  LNKSEL1_RD      = 0X3300;
CONST  LNKSEL1_WR      = 0X3301;
CONST  LNK1_STAT_IN    = 0X3302;
CONST  LNK1_STAT_OUT   = 0X3303;

CONST  LNKSEL2_RD      = 0X3400;
CONST  LNKSEL2_WR      = 0X3401;
CONST  LNK2_STAT_IN    = 0X3402;
CONST  LNK2_STAT_OUT   = 0X3403;

CONST  TSTRAM          = 0X3500;

CONST  byte_l   = 0X2000;
CONST  byte_m   = 0X2001;
CONST  byte_u   = 0X2002;
CONST  pm_sel_flag = 0X2003;
ONST   PACK_LEN = 0X2008;
ONST   DESTN_ADR = 0X2009;
--- DATA ---------------------------------------------}
ONST   DLYSEL0   = 0X7FFE;
ONST   DPCSEL0A = 0X7FFD;   {DSPSEL = 0}
ONST   DPCSEL0B = 0XFFFD;   {DSPSEL = 1}
ONST   DLYSEL1   = 0X7FFB;

ONST   DLYSEL2   = 0X7FF7;
ONST   DPCSEL1A = 0X7FEF;
ONST   DPCSEL1B = 0XFFEF;
ONST   DLYSEL3   = 0X7FDF;

ONST   DLYSEL4   = 0X7FBF;
ONST   DPCSEL2A = 0X7F7F;
ONST   DPCSEL2B = 0XFF7F;
ONST   DLYSEL5   = 0X7EFF;

NST    DLYSEL6   = 0X7DFF;
NST    DPCSEL3A = 0X7BFF;
NST    DPCSEL3B = 0XFBFF;
NST    DLYSEL7   = 0X77FF;

NST    DLYSEL8   = 0X6FFF;
NST    DPCSEL4A = 0X5FFF;
NST    DPCSEL4B = 0XDFFF;
NST    DLYSEL9   = 0X3FFF;
```

DELAY DLY_SET.DSP

```
{Program to enable IRQ1 & IRQ2 and to set the delay by
  reading the delay values for the two memory banks from
  a file in the PC}
{A constant delay can be set by changing the mr2 reg
  value and addr in i3. The new delay will be set in
  the isr sta6.dsp (IRQ1) and dly counter is incremented in
  the isr fft6.dsp (IRQ2).}

{Delay have to be sent only once. Free04 is kept high through
out for tstdata to syn with stainit.
12-6-98}

{Packet communication is included 06-8-98}

{.....................................................................}

.MODULE/ABS=0x800/RAM   set_delay;
.include  <dlydpc.h>;
.include  <delay.h>;


{.....................................................................}
{ alocate bank pointers, init tables, enable interupts                }
{.....................................................................}
initdly: ifc = 0x0024;          {clear all pending interrupts}
        ifc = 0x0000;
        ax0 = 0x1249;
        dm(0x3ffe) = ax0;
        ax0 = 0x3f00;           { read mode}
        dm(stwr) = ax0;

        { init delay adress banks ..................................}
        i0 = ^dly_adr; m0 = 1; l0 = 0;
        i4 = ^dlyadr; m4 = 1; l4 = %dlyadr; {dlyadr[10] is a circular bu

        cntr = %dlyadr;         {Copy the content of dly_adr to dlyadr}
        do swap0 until CE;
          ar = dm(i0,m0);
          dm(i4,m4) = ar;
swap0:  nop;

        { Fill the array dlyval1 & dlyval2 with the delay values of zero
...}
        { modified with value of 1 ...}
        i1 = ^dlyval1; m1 = 1; l1 = %dlyval1;
        i2 = ^dlyval2; m2 = 1; l2 = %dlyval2;
        cntr = %dlyval1;        { clear delay banks }
        do return0 until CE;
           ar = 0x1a03;
           dm(i1,m1) = ar;
           ar = 0x1a03;
           dm(i2,m2) = ar;
        return0 : nop;
```

```
        ena sec_reg;     {enable secondary reg prepare for stacount}
           sr0 = DLYMSK1;     {DLYMSK1 = 0x07ff, used in FFT ISR}
           ay1 = DLYMSK2;     {DLYMSK2 = 0x1800, used in FFT ISR}
        dis sec_reg;

        { init adress pointers and modifiers in ISRs ..............}
        i0 = crdsel; m0 = 0; l0 = 0; {put address of crdsel in i0.
                                      This is to use some multifunction
                                      in the FFT ISR.}
        i4 = dspio; m4 = 0; l4 = 0; {put address of dspio in i4.
                                       This is to use some multifunctio
n
                                       in the FFT ISR.}
        i5 = ^dlyinc; m5 = 1; l5 = %dlyinc; {put the dlyinc[128] in i5
                                       This is used in FFT ISR.}
        l6=%dlyadr; m6=1;           { adress pointer in the FFT ISR }

        { init flags and temp bnak adress ..........................}
        ar = ^dlyval1;           {Initialise bankad }
        dm(bankad) = ar;
        ar = 0x0;                {Initialise ldflag with value of 0}
        dm(ldflag) = ar;
        dm(stacount)=ar;
        ar=8;
        dm(hstacnt) = ar;        { for debugging }

        { enable interupts .........................................}
        icntl = 0x06;    {Make irqs edge trigger; disable nesting}

{..............................................................}
{ main loop ***************************************************  }
{..............................................................}
gtpckt: jump get_packt;          {get the packets}
after_packet :
        ar = dm(packet_len);     {check packet is a sensible one}
        ar = pass ar;
        if ne jump cmdintrp;     {if sensible call command interpreter}
        jump gtpckt;             {get the next packet }
getout: rts;                     {back to boot }

..............................................................}
 command selection                                            }
..............................................................}
cmdintrp: i3 = ^inbuf;           { read packet header }
        m3 = 1;
        l3 = 0;
        cntr = dm(noofcmd);      { number of commands }
        do loopcmd until ce;
           ay0 = 10;
           ax0  = dm(i3,m3);     { read command }
           ar = ax0 - ay0;
           if ne jump reset_cmd;
```

```
            { new delay values ..........................................}
               call newdly;      { option cmd=10 update delay table }
               jump loopcmd;
reset_cmd:  ay0 = 7;
            ar = ax0 - ay0;
            if ne jump dlyin_cmd;
            { reset stacount value ......................................}
               call rststa;      { option cmd=7 }
               jump loopcmd;
dlyin_cmd:  ay0 = 9;
            ar = ax0 - ay0;
            if ne jump exitintr_cmd;
            { fill the delay table ......................................}
               imask = 0x0000; {Enable IRQ1 & IRQ2}
               call dlyinfil;    { option cmd=9 }
               imask = 0x0004;   { enable IRQ1 only }
               idle;             { wait for 1st STA interrupt }
               imask = 0x0024;   { now enable IRQ1 & IRQ2 }
               jump loopcmd;
exitintr_cmd: ay0 = 6;
            ar = ax0 - ay0;
            if ne jump loopcmd;
            { exit software .............................................}
               jump getout;{ option cmd=0x06 }
loopcmd:    jump gtpckt;                    {get the next packet }

{.....................................................................}
{ update delay table and then shift the bank                          }
{.....................................................................}
newdly:  ay0 = dm(ldflag);
         ar = PASS ay0;
         if ne jump newdly;

         i2 = ^dlyval1;
         ay0 = dm(bankad);
         ar = ^dlyval1;
         ar = ar - ay0;     {check which data bank should be used}
         if NE jump tst1;
         i2 = ^dlyval2;

tst1:    si = dm(i3,m3);    {Get the stacount (2bytes) specified by the ho
st}
         sr0 = dm(i3,m3);   {at which the newdelay has to be set.}
         sr = sr or lshift si by 8 (lo);
         dm(hstacnt) = sr0;

         cntr = %dlyval1;    {Get the 10 delay values (2 bytes each).}
         do return1 until CE;
           si = dm(i3,m3);
           sr0 = dm(i3,m3);
           sr = sr or lshift si by 8 (lo);
         return1: dm(i2,m2) = sr0;
```

```
        dm(bankad) = i2;    {switch adress bank, i2 is back to nominal po
sition}

        { Set the ldflag signifying arrival of new delay values}
        ar = 1;
        dm(ldflag) = ar;
rts;

{.....................................................................}
{ reset stainit                                                       }
{.....................................................................}
rststa: ar = 0;
        dm(stacount) = ar;
rts;

{.....................................................................}
{ Fill dlyinc array                                                   }
{.....................................................................}
dlyinfil:si = dm(i3,m3); {get delay increment}
        sr0 = dm(i3,m3);
        sr = sr or lshift si by 8 (lo);
        i5 = ^dlyinc;
        m5 = 1;
        l5 = %dlyinc;
        cntr = %dlyinc;
        do return2 until CE;
 return2: dm(i5,m5) = sr0;

        ay1 = dm(i3,m3); {nth  value at which 2 should be added }
        ar = ay1 - 1;
        ay0 = ^dlyinc;
        ar = ar + ay0;
        i5 = ar;
        m5 = ay1;
        l5 = %dlyinc;

        si = dm(i3,m3); {get the loop length}
        cntr = si;
        ay0 = 2;
        do return3 until CE;
          ar = dm(i5,m4); { m4=0 }
          ar = ar + ay0;
return3: dm(i5,m5) = ar;

        i5 = ^dlyinc;
        m5 = 1;
        l5 = %dlyinc;
        rts;

{.....................................................................}
{ exchange of data with host                                          }
{.....................................................................}
```

```
        {  Handshake for C012 communication:
           All communication originates from the host.
           First byte of each packet should contain length (maximum 255
)
           Packet length of 0 or 1 are ignored.
           Second byte of each packet (length > 1) should contain  no o
f
           commands to be executed.
           Rest of the packet contains arguments, size depending on len
gth.

           Upto 256 commands can be recognised.
           i.e. all one byte commands are recognized.
           The following are the valid commands (0 <= x <= 255)

             255              ==>   Unconditional end of packet executi
on
             6                ==>   exit soft
             7                ==>   Reset sta_count
             9                ==>   delay increment values update
             10               ==>   delay update

           Only the first byte in each packet (length) is acknowledged
           in the protocol.  Protocol restricts maximum packet length
           to 255 bytes.

           Any number of command to be processed can be sent at once, bu
t
           it is recommended to sent only three or 4 command at a time.
        }

get_packt: nop;
        ax0 = 0x3f00;       { read mode}
        dm(stwr) = ax0;
        call b_in;
        dm(packet_len)=sr0;
        af = pass sr0;
        if eq rts;
        af = af - 1;        { Subtract the pcklen byte count}
        if eq rts;
        call b_in;          { get the number of commands}
        dm(noofcmd) = sr0;
        ar = af - 1;
        i7 = ^inbuf ;       { inbuf contains the packet as a byte-stream
}
        m7 = 1;
        ay0=ar;
        af = pass 0;
        beg_pack: call b_in ;  { loop equivalent to do while but not usi
ng PC}
          dm(i7,m7) = sr0 ;
        r_pkt:  af = sr0 + af ;  { contains the summation }
          ay0=ar;
```

```
          ar=ay0-1;
        if ne jump beg_pack;
        end_pkt: sr0 = dm(packet_len);
        af = sr0 + af;
        sr0 = dm(noofcmd);
        af = sr0 + af;
        call b_in;          { getting check sum}
        si = sr0 ;
        call b_in ;
        sr = sr or lshift si by 8 (lo) ;
        ar = sr0 XOR af ;    { checking whether check sums are matching}
        if eq jump no_err ;    {    jump no_err; }
             af = pass 0 ;
             ar = 0;
             dm(packet_len) = ar;

no_err : ar = pass af;      { reply checksum to host }
        sr0 = ar;
        call b_out ;
        sr = lshift sr0 by -8 (lo) ;
        call b_out ;

sendsta:sr0 = dm(stacount); { reply stacount to host }
        call b_out;
        sr = lshift sr0 by -8 (lo) ;
        call b_out;
        jump after_packet;  { back to main subroutine  }


{.....................................................................}
{ control card read mode                                              }
{.....................................................................}
readmode: ax0 = 0x3f00;             { read mode}
        dm(stwr) = ax0;
rts;

{.....................................................................}
{ control card write mode                                             }
{.....................................................................}
writmode: ax0 = 0x3300;             { write mode}
        dm(stwr) = ax0;
rts;

{.....................................................................}
{ get a byte                                                          }
{.....................................................................}
b_in:   nop;                        { get a byte}
wait_rx:  if not flag_in jump wait_rx;
        mr1 = ar;
        mr2 = ay0;
        ar = dm(lnksel1_rd);
        ay0 = 0x00ff;
        ar  = ar AND ay0;
```

```
        sr0 = ar;
        ar = mr1;
        ay0 = mr2;
rts;

{.....................................................................}
{ send a byte                                                         }
{.....................................................................}
b_out:  nop;
        mr1 = ar;    {Save AR}
        mr2 = ay0;   {Save AY0}
        ay0 = 0x0001;
        ax0 = 0x3f00;           { read mode}
        dm(stwr) = ax0;
wait_tx : sr1 = dm(lnk1_stat_out);
        ar = sr1 and ay0;       { mask other bits }
        ar = ar - ay0;
        if ne jump wait_tx;
        ax0 = 0x3300;           { write mode}
        dm(stwr) = ax0;

        dm(lnksel1_wr) = sr0;     { send back to the master }

        ar = mr1;
        ay0 = mr2;

        rts;

{.....................................................................}
{ for debugging, blink led                                            }
{.....................................................................}
blink:   set flag_out;
        cntr=1000;
        toggle flag_out;
        do blk until ce;
           cntr=500;
           do blkbis until ce;
           nop;
        blkbis: nop;
        blk:nop;

        cntr=1000;
        do blk1 until ce;
           cntr=500;
           do blk1bis until ce;
           toggle flag_out;
           nop;
        blk1bis: nop;
        blk1:nop;

       jump blink;
rts;

.ENDMOD;
```

DLY_FFTI.DSP

```
{ ..................................................................
..
  ISR for delay update
  6/8/98
  DLY_FFTISR.DSP
  IRQ2
  adress declaration and initialisation in DLY_SET.DSP
  (code optimised for minimum stack use )
  ..................................................................
..}

.MODULE/ABS=0x3000/RAM  dlyfft;
.include  <dlydpc.h>;
.include  <delay.h>;

   ena sec_reg;

   ay0 = dm(i5,m5);  {dlyinc}

   i6 = ^dlyadr;
   ax0 = dm(i6,m6);  {dly card address}

   ax1 = dm(i1,m1);  {get first dly value and increment i1}

   dm(i0,m0) = ax0, af= ax1 + ay0;     {Add dlyinc & Card select}
   dm(i4,m4) = ax1, af = sr0 and af;   {mask (DLYMSK1) af & write dly
                                          val to dspio}
   ar = ax1 and ay1, ax1 = dm(i1,m0); {extract oneclk dly (DLYMSK2) &
                                          get next dly value}
   ar = ar or af, ax0 = dm(i6,m6);     {complete the dly val update &}
   dm(i1,m1) = ar;            {write the updated the dly val &}

   dm(i0,m0) = ax0, af= ax1 + ay0;     {optimised code that avoid the use
 of PC}
   dm(i4,m4) = ax1, af = sr0 and af;
   ar = ax1 and ay1, ax1 = dm(i1,m0);
   ar = ar or af, ax0 = dm(i6,m6);
   dm(i1,m1) = ar;


   dm(i0,m0) = ax0, af= ax1 + ay0;
   dm(i4,m4) = ax1, af = sr0 and af;
   ar = ax1 and ay1, ax1 = dm(i1,m0);
   ar = ar or af, ax0 = dm(i6,m6);
   dm(i1,m1) = ar;


   dm(i0,m0) = ax0, af= ax1 + ay0;
   dm(i4,m4) = ax1, af = sr0 and af;
   ar = ax1 and ay1, ax1 = dm(i1,m0);
   ar = ar or af, ax0 = dm(i6,m6);
   dm(i1,m1) = ar;
```

```
    dm(i0,m0) = ax0, af= ax1 + ay0;
    dm(i4,m4) = ax1, af = sr0 and af;
    ar = ax1 and ay1, ax1 = dm(i1,m0);
    ar = ar or af, ax0 = dm(i6,m6);
    dm(i1,m1) = ar;


    dm(i0,m0) = ax0, af= ax1 + ay0;
    dm(i4,m4) = ax1, af = sr0 and af;
    ar = ax1 and ay1, ax1 = dm(i1,m0);
    ar = ar or af, ax0 = dm(i6,m6);
    dm(i1,m1) = ar;


    dm(i0,m0) = ax0, af= ax1 + ay0;
    dm(i4,m4) = ax1, af = sr0 and af;
    ar = ax1 and ay1, ax1 = dm(i1,m0);
    ar = ar or af, ax0 = dm(i6,m6);
    dm(i1,m1) = ar;

    dm(i0,m0) = ax0, af= ax1 + ay0;
    dm(i4,m4) = ax1, af = sr0 and af;
    ar = ax1 and ay1, ax1 = dm(i1,m0);
    ar = ar or af, ax0 = dm(i6,m6);
    dm(i1,m1) = ar;

    dm(i0,m0) = ax0, af= ax1 + ay0;
    dm(i4,m4) = ax1, af = sr0 and af;
    ar = ax1 and ay1, ax1 = dm(i1,m0);
    ar = ar or af, ax0 = dm(i6,m6);
    dm(i1,m1) = ar;

    dm(i0,m0) = ax0, af= ax1 + ay0;
    dm(i4,m4) = ax1, af = sr0 and af;
    ar = ax1 and ay1, ax1 = dm(i1,m0);
    ar = ar or af, ax0 = dm(i6,m6);
    dm(i1,m1) = ar;

  dm(i0,m0) = DLY_NSE;
  dis sec_reg;

  rti;
.endmod;
```

DLY_STAZ.DSP

```
{ ..........................................................
..
  ISR for delay update
  6/8/98
  DLY_STAISR.DSP
  IRQ1
  adress declaration and initialisation in DLY_SET.DSP
  ..........................................................
..}
.MODULE/ABS=0x3800/RAM  dlysta;
.include  <dlydpc.h>;
.include  <delay.h>;
        toggle flag_out;

        ena sec_reg;
        i5 = ^dlyinc;                   {initialize i5 every stacycle}

        ay0 = dm(stacount);             { increment }
        ar = ay0 + 1;
        dm(stacount) = ar;              {store stacount}

tst2:   ay0 = dm(hstacnt);              {compare with host stacount}
        ar = ar - ay0;                  {warning ar value ! }
        if ne jump tst4;
          ay0 = dm(ldflag);             {test flag new delay }
          ar = pass ay0;
          if eq jump tst4;
{If new delay values are available, switch the memory banks using bankad
}
            ar=0;                           { temporary for debug purpos
e }
            dm(ldflag) = ar;                {First reset ldflag to zero
!}

            ay0 = dm(bankad);
            i1 = ay0;
tst4 :  dis sec_reg;

        rti;
.endmod;
```

```
{
  DPC_CMD.DSP
}

.module/abs=0x800/ram dpc_cmd;
.include        <dlydpc.h>;
.var/seg=int_dm/ram/abs=0x3800 dpc_adr[10],dly_adr[10];

        call get_byte1;   {Get the DPC card address}
        si = mr0;
        sr = lshift si by 1 (LO);

        ay0 = ^dpc_adr;   {Get the base address for DPC}
        ar  = sr0 + ay0;

        i0  = ar;
        l0  = 0;
        m0  = 1;

        ax0 = dm(i0,m0);
        dm(crdsel)=ax0;   {Assert the cardselect with DSPSEL=0}

        call get_byte1;   {Get the Upper Byte for U13 on DPC card}
        sr = lshift mr0 by 8 (LO);
        ay1 = sr0;

        call get_byte1;   {Get the Lower byte for U7}
        ar = mr0 OR ay1;

        dm(dspio) = ar;
        rts;

get_byte1 :
                ax0 = 0x1f00;            { read mode}
                dm(stwr) = ax0;
        wait_rx :
                if not flag_in jump wait_rx;

                mr0 = dm(lnksel1_rd);
                ay0 = 0x00ff;
                ar  = mr0 AND ay0;
                mr0 = ar;

                sr0 = mr0;

                call tx_sr0;

        rts;
{.....................................................}
tx_sr0 :
                mr1 = ar;    {Save AR}
                mr2 = ay0;   {Save AY0}
                ay0 = 0x0001;
```

```
            sr1 = 0x1f00;              { read mode}
            dm(stwr) = sr1;
      wait_tx :
            sr1 = dm(lnk1_stat_out);
            ar = sr1 and ay0;          { mask other bits }
            ar = ar - ay0;
            if ne jump wait_tx;

            sr1 = 0x1300;              { write mode }
            dm(stwr) = sr1;
            dm(lnksel1_wr) = sr0;      { send back to the master }

            ay0 = 0;
            ar = mr1 + ay0;
            ay0 = mr2;
      rts;

.endmod;
```