*Zentrallabor für Elektronik*

**Berichte zum**

**Datenerfassungssystem**

**für physikalische Experimente**

---

# ATRAP

# PCI HOTLink™ - Auslesesystem

W. Erven, H. Lövenich, R. Nellen, K. Zwoll

Forschungszentrum Jülich
Zentrallabor für Elektronik

Abteilung Experimentsteuerung und Kommunikation

Version 1.1
Letzte Bearbeitung: 24. Oktober 2003

# Änderungshistorie

| Version/ Datum | Art der Änderung |
|---|---|
| V1.0 Jul 1999 | erste Version |
| V1.1 Oct 2003 | Überarbeitetes Hardware Design |

# Inhaltsverzeichnis

# 1 Allgemeines

Für die Datenübertragung zwischen Front-End und CSC wird das optische HOTLink™ System verwendet, wobei eine universelle PCI Karte zur Verfügung steht. Die Übertragung erfolgt dabei über ein handelsübliches Glasfaser-Kabel. Die HOTLink™ Transmitter/Receiver der Firma CYPRESS übertragen 8 Bit Daten mit einem speziellen Protokoll. Als optischer Wandler wird ein *Optical Data Link* Baustein der Firma Siemens eingesetzt. Die Netto-Datenrate beträgt 24MByte pro Sekunde.

Das STROBE/TRIGGER Signal wird mit einer eigenen Glasfiberleitung übertragen. Die Ablaufsteuerung mit Generierung der ReadOut Clock wird in einem CPLD (AMD MACH® Baustein) auf der Opto Koppelkarte vorgenommen.

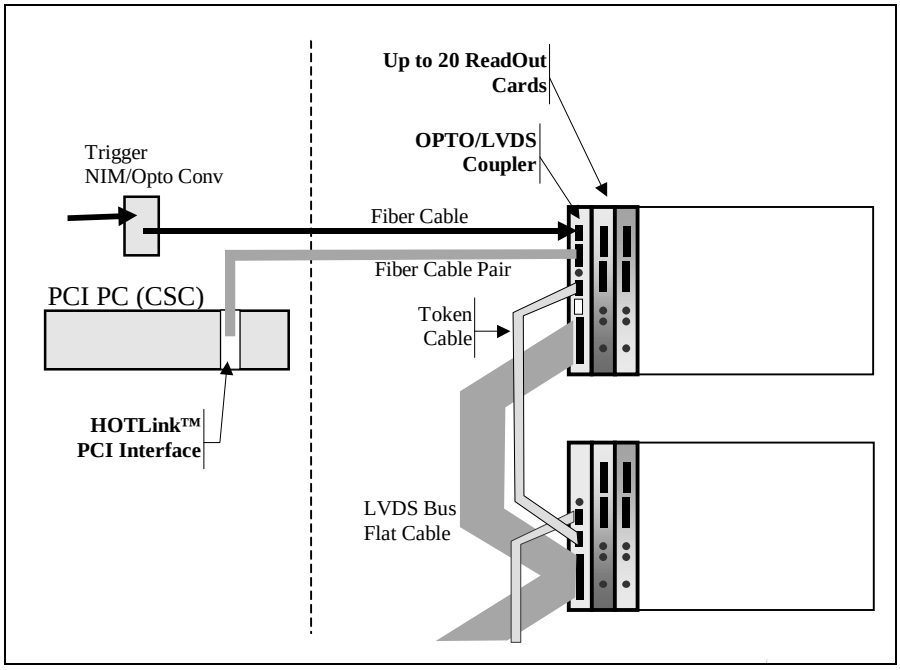


**Abbildung 1 Gesamtsystem**

Die Koppel-Karten werden durch ein Flachbandkabel miteinander verbunden.

- Das STROBE Signal kann durch entsprechende Verdrahtung bei Bedarf jeder ReadOut Karte einzeln zugeführt werden, wobei die Verzögerung durch die Kabellänge bestimmt wird. In diesem Fall wäre eine zusätzliche Karte für die Erzeugung von 19 STROBE Signalen nötig, die neben der Koppelkarte plaziert wird.

# 2 Die Koppelkarte

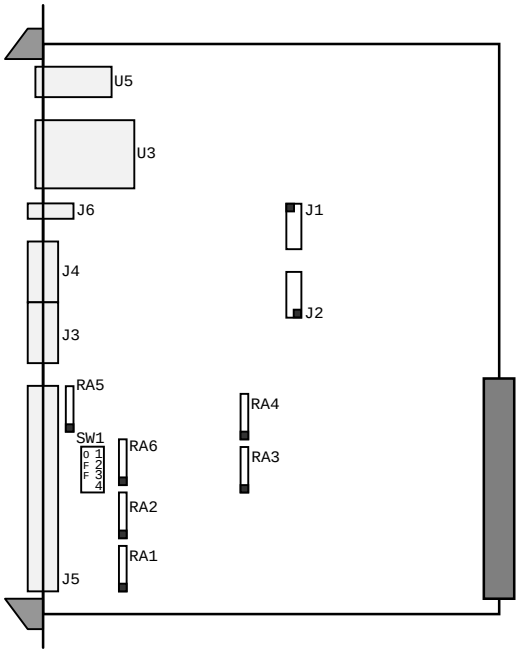


**Abbildung 2 Koppelkarte**

Die Koppelkarte nimmt immer den linken Steckerplatz im Überrahmen ein (siehe **Abbildung 1**). Sie hat zwei Ausführungsformen, die sich in der Bestückung unterscheiden. Die Haupt-Koppelkarte ist voll bestückt und über Glasfaser-Kabeln mit dem Rechner und der Trigger-Logik verbunden. Die Ablauf-Steuerung arbeitet auf einem LVDS Bus, der über einem 50 poligen Flachbandkabel bis zu 14 weitere Überrahmen mit LVDS Koppelkarten verbinden kann. Bei der zweiten Ausführungsform, die LVDS Koppelkarte, ist die Rechner-Kopplung nicht bestückt. Sie ist von außen durch das Fehlen der Stecker **U5** und **U3** erkenntlich. Der Stecker **J3** braucht auf der Master Koppelkarte nicht bestückt zu sein.

Es folgt eine tabellarische Erläuterung der wichtigsten Elemente.

**J1** MACH® Programmierstecker

**J2** Debugstecker

**U5** einzel Glasfaser-Stecker für das Trigger/Strobe Signal

**U3** zweifach Glasfaser-Stecker für die Verbindung zum HOTLink™ Interface im Rechner (CSC).

**J6** LEMO Buchse mit zwei Adern (LVDS Signal) auf der das Trigger/Strobe Signal liegt. Bei Bedarf wird das Trigger/Strobe Signal in eine beliebige ReadOut-Karte auf Buchse **X1** eingespeist. Von dort verbreitet sich das Signal in beiden Richtungen zu den anderen ReadOut Karten. Durch die Länge des Kabels kann eine Laufzeitanpassung erfolgen.

**J4** ist ein 10 poliger Pfostenstecker für den Eingang des Token-Signals. Dieser Stecker wird durch ein 10 poliges Flachbandkabel mit dem Stecker J3 der nächsten LVDS Koppelkarte verbunden. Dadurch wird der Zugriff auf den einzelnen Überrahmen (Spalten) gesteuert.

**J3** ist ein 10 poliger Pfostenstecker für den Ausgang des Token-Signals. Dieser Stecker ist auf der Haupt-Koppelkarte nicht bestückt. Das Token-Signal geht immer vom letzten Überrahmen aus, der nach einem Reset den Token besitzt.

**J5** 50 poliger Pfostenstecker für den LVDS Daten-Bus. Alle Koppelkarten werden durch ein 50 poliges Flachbandkabel miteinander verbunden.

**RA1** bis **RA6**

sind steckbare Abschlußwiderstände für den LVDS Bus. Sie müssen auf der letzten Koppelkarte gesteckt sein. Wenn nur ein Überrahmen vorhanden ist, müssen sie auch dort auf der Haupt-Koppelkarte gesteckt sein. Der Typ des Widerstand-Arrays ist **4Y1014** (4 x 100Ω).

**SW1** Dual Inline Schalter mit 4 Schalter. Position rechts ist geschlossen.

**1/2** Rückführung der (ReadOut) Clock auf dem LVDS Bus. Diese beiden Schalter müssen auf der letzten Koppelkarte geschlossen sein. Also dort, wo auch die Abschlußwiderstände gesteckt sind. Wenn keine Slave Koppelkarten angeschlossen sind, müssen die Schalter auf der Master Koppelkarte geschlossen sein.

**3** Trigger/Strobe Einspeisung. Wenn dieser Schalter offern ist (linke Position) wird das Strobe-Signal von der Koppelkarte direkt an die ReadOut-Karten weitergeleitet. Eine Kabelverbindung von **J6** zu einer ReadOut-Karte darf dann nicht vorhanden sein. Anders wenn der Schalter geschlossen ist, muß das Strobe-Signal über diese Kabelverbindung zugeführt werden.

**4** einzelne Strobe-Zuführungen. Dieser Schalter muß immer geschlossen sein. Wenn dieser Schalter offen ist, muß das Strobe-Signal jeder ReadOut-Karte auf **X1** zugeführt werden. In diesem Fall ist eine zusätzliche Karte, die 19 Strobe-Signale liefert, erforderlich. Z.Zt. ist diese Karte noch nicht geplant.

# 3 Die ReadOut Karte

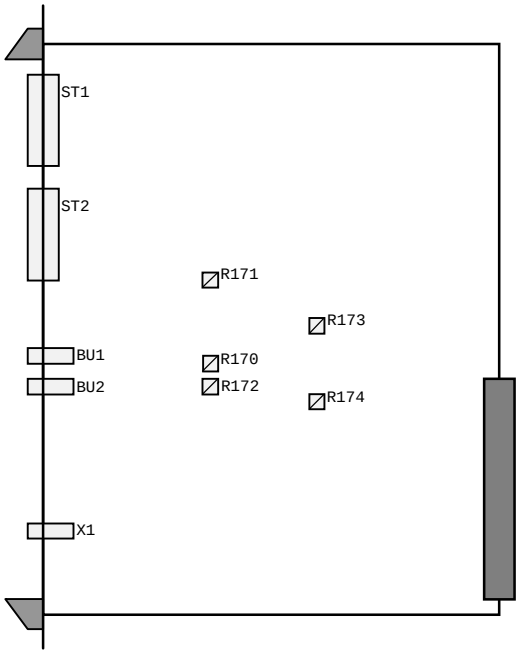


**Abbildung 3 ReadOut Karte**

**ST1** 50Ω Eingänge für die Kanäle 1 bis 8.

**ST2** 50Ω Eingänge für die Kanäle 9 bis 16.

**BU1** LEMO Ausgang. Summen-Signal von den 16 Eingangs-Signale, die direkt am Stecker liegen.

**BU2** LEMO Ausgang für multiplizitäts-Signal. Dieses Signal wird hinter den Diskriminatoren gebildet und liefert für jeden Kanal, der ein Ausgangs-Signal erzeugt, 500mV. Bei Belastung mit 50Ω die Hälfte.

**X1** 2 polige LEMO Buchse für den Strobe-Eingang. Diese Buchse ist isoliert angebracht. Wenn eine Verbindung von der Buchse J6 der Koppelkarte hergestellt wird, wird das Strobe-Signal von dieser Karte aus auf die anderen Karten verteilt.

**R170** bis **R172**
Potis zur Einstellung der Schwellen für das Summen-Signal.
**R170** => Kanal 1,2,3,9,10,11
**R171** => Kanal 12,13,14,15,16
**R172** => Kanal 4,5,6,7,8

**R173** und **R174**
Potis zur Einstellung der Schwellen für das Multiplizitäts-Signal.
**R173** => Kanal 1 bis 8
**R174** => Kanal 9 bis 16

# 4 Datenerfassung mit dem HOTLink™ PCI Interface

Das HOTLink™ PCI Interface wurde im ZEL entwickelt und ist im internen Bericht "***HOTLink™ PCI Interface***" beschrieben. Mit diesem Interface können Datenbyte's über ein Glasfaser-Kabel transparent übertragen werden, wobei die Übertragung auch im DMA Mode erfolgen kann.

## 4.1 Das Sendeprotokoll

Jedes Byte wird von der Opto Koppelkarte interpretiert. Dabei werden 6 Bit ausgewertet, wie in folgender Tabelle dargestellt ist.

| Bit | Wert | Bedeutung |
|---|---|---|
| 7 6 5 4 3 2 1 0 | | |
| - - 0 0 S CTL[3] | 0x00-0x0F | S = Stop 50 MHz Clock für RAL111 Shiftregister<br>CTL = Steuerwort für die RAL111 Control Signale, siehe unten |
| - - 0 1 XDN[4] | 0x10-0x1F | Datenregister für DAC und TEST Werte<br>XDN[0] unterdrückt das externe STROBE Signal |
| - - 1 - - 0 0 0 | 0x20 | ReadOut Clock Impuls, für die Übergabe von DAC und TEST Werte |
| - - 1 - - 0 0 1 | 0x21 | STROBE Impuls für Readouttest |
| - - 1 - - 0 1 0 | 0x22 | Statusbyte anfordern |
| - - 1 - - 0 1 1 | 0x23 | Softwarestart für ReadOut (anstelle STROBE) |
| - - 1 - - 1 0 0 | 0x24 | ReadOut für Selbsttest |
| - - 1 - - 1 0 1 | 0x25 | Normales ReadOut aktivieren, Start erfolgt mit nächster STROBE Rückflanke |
| - - 1 - - 1 1 - | 0x26 | Test ReadOut aktivieren, Start erfolgt mit nächster STROBE Vorderflanke |

**Tabelle 1 Sendeprotokoll**

### 4.1.1 Control Byte

Das Control Byte wird dadurch erkannt, daß Bit 4 und 5 null sind.

***S*** bit 3 (Stop), dieses Signal steuert die 50 MHz Clock für das Shift Register der RAL111 Chip's. Wenn dieses Bit gesetzt ist, wird die Clock angehalten. Beim automatischen Readout wird dieses Signal von der Ablaufsteuerung vorgegeben. Dieses Signal wird auch als Steuersignal beim Laden der RAL111 DAC/TEST Register verwendet (siehe **Tabelle 2**).

***CTL*** bit 2..0, diese drei Signale steuern zusammen mit dem Signal ***S*** den Betriebsmodus der RAL111 Chip's. Folgende Tabelle zeigt die Codierung mit den Zuordnungen der RAL111 Signale.

| CTL S 2 1 0 | Read Out | DAC Load | Test Load | Test Enable | Reset | Funktion |
|---|---|---|---|---|---|---|
| 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | Idle |
| 0 0 0 1 | 0 | 0 | 0 | 1 | 0 | Test Readout, all one |
| 0 0 1 0 | 0 | 0 | 1 | 0 | 0 | Test Readout mit Test Register |
| 1 0 1 0 | 0 | 0 | 1 | 0 | 0 | Load Test Register |
| 0 0 1 1 | 0 | 0 | 1 | 1 | 0 | Test Readout, all zero |
| 1 1 0 0 | 0 | 1 | 0 | 0 | 0 | Load DAC Register |
| 0 1 0 1 | 1 | 0 | 0 | 0 | 0 | Readout |
| 0 1 1 1 | 0 | 0 | 0 | 0 | 1 | Reset |

**Tabelle 2 Control Byte**

Die grau unterlegten Zeilen haben im Normalfall keine Bedeutung. Die betreffenden "Test Readout" funktionieren nicht richtig und der "Readout" wird von der Ablaufsteuerung gesetzt. Nur für die Fehlersuche könnten die betreffenden Funktionen von Bedeutung sein.

### 4.1.2 Das Daten Byte

Das Daten-Byte wird dadurch erkannt, daß Bit5/4 gleich 0/1 sind. Es wird zum Laden der RAL111 DAC- und TEST-Register benötigt. Das erste Bit, ***XDN[0]***, hat noch eine weitere Bedeutung. Wenn dieses Bit gesetzt ist, wird die Weiterleitung des externen Strobe Signals verhindert. Dadurch können Störungen durch das externe Strobe Signal beim Selbsttest verhindert werden. Für den normalen ReadOut Betrieb muß dieses Bit null sein.

## 4.2 Das Empfangsprotokoll

Die Opto Koppelkarte sendet ein oder mehrere Bytes bei folgenden Anlässen.

1. Statusanforderung: 1 Byte
2. ReadOut Clock Impuls: 1 Byte
3. Readout Start: Readout Daten

Das Format der empfangenen Bytes ist in folgender Tabelle dargestellt.

| Bit | Bedeutung |
|---|---|
| **7 6 5 4 3 2 1 0** | |
| 0 0 1 1 X[4] | Dieses Datenbyte wird nach einem ReadOut Clock Impuls übertragen |
| 0 0 1 0 - a t r<br><br>0 0 0<br>0 0 1<br>1 0 1<br>0 1 1<br>1 1 1 | Statusbyte, a: ReadOut Daten werden gerade übertragen, t: Test ReadOut, r: ReadOut ist aktiviert und wartet auf Start (Strobe).<br>Idle<br>ReadOut wartet auf Start<br>ReadOut Daten werden übertragen<br>Test ReadOut wartet auf Start<br>Test ReadOut Daten werden übertragen |
| R 1 0 0 X[4] | ReadOut Byte, X ist Drahtnummer in der aktuellen Reihe |
| R 1 1 0 X[4] | ReadOut Byte, X ist Drahtnummer in der aktuellen Reihe, die Reihe wird anschließend inkrementiert (row++;) |
| R 1 1 1 - - - - | Null Byte, die Reihe wird inkrementiert (row++;) |
| 0 1 0 1 - - - 0 | Null Byte, Reihe wird 0, Spalte wird inkrementiert (row=0; col++;) |
| 0 1 0 1 - - - 1 | Registermarke beim Test ReadOut, (row=0; col=0; regm++;) |
| 0 0 0 0 - - - - | ReadOut Endekennung |

**Tabelle 3 Empfangsprotokoll**

Das Bit "R" ist das Row Bit 0 und dient nur zur Kontrolle des Row Counters im Treiber.

## 4.3 Betrieb mit mehreren Rahmen

Wenn am LVDS BUS ein oder mehrere Rahmen angschlossen sind, daß die Token Leitungen richtig gesteckt sind (siehe **Abbildung 1**) und auf der letzten Karte die Abschlußwiderstände gesteckt sind und nur hier die Schieber 1 und 2 des Schalters SW1 geschlossen sind (siehe Seite **2**). Nach einem Reset Control Byte (Sendebyte = 0x07) hat die letzte Controller Karte (Spalte 0) den Token. Der Token wird um eine Karte weitergeschaltet, wenn das Control Byte von 0x0A (Load Teset) oder von 0x0C (Load DAC) auf null zurückgesetzt wird.

# 5 Programmierung

Im folgenden Headerfile sind alle wesentlichen Typen und Konstanten aufgeführt. Weitere Informationen zum PCI HOTLink Interface können der entsprechenden Beschreibung entnommen werden.

```
//========================== HOTLink Interface ============================
//
//--------- PCI Device ID
//
#define HOTLINK_DEVICE_ID     0x0006
//
//--------- Device Register Region 1
//
typedef struct {
   u_long   creg;          // Control Register
   u_long   screg;         // Status/Control Register
   union {                 // Data Output Register
      u_char   byte;       //    send 1 Byte
      u_short  word;       //    send 2 Byte
      u_long   dword;      //    send 4 Byte
   } dreg;
} DEV_REGS;
//
//--------- Control Register
//
#define REC_ENABLE      0x01
#define LOOPBACK        0x20
//
//--------- Status/Control Register
//
#define INPUT_COUNTER   0x03        // RO
#define CLEAR_FIFO      0x01        // WO clear FIFO and Input Counter
#define FLUSH_INPUT     0x02        // WO

#define CLR_INT_STATES  0xFF000000L // Interrupt states
#define IS_START_DATA   0x01000000L
#define IS_END_DATA     0x02000000L
#define IS_FIFO_FULL    0x04000000L
#define IS_REC_ERROR    0x08000000L
#define IS_CARRIER      0x10000000L
//
//========================== ReadOut Opto Coupler ==========================
//
//--------- Control Byte (b5/b4 = 00)
//
#define CTL_TST_ONE     0x1   // Test, set all channel at STROBE
#define CTL_LD_TST      0x2   // Load test pattern, set channel as pattern
#define CTL_TST_ZERO    0x3   // Test, set non channel at STROBE
#define CTL_LD_DAC      0x4   // Load DAC register
#define CTL_ROUT        0x5
#define CTL_RESET       0x7   // Reset RAL111 and Coupler
#define STP_FCLK        0x8   // Bit ID, stop 50MHz input clock
//
//--------- Data Byte (b5/b4 = 01)
//
#define SET_DREG        0x10  // data register for load (DAC, TEST)
//
//--------- Command Byte (b5 = 1)
//
#define CLOCK           0x20  // send clock pulse
#define STROBE          0x21  // send strobe pulse
#define STATE_REQ       0x22  // request state
#define ROUT_RUN        0x23  // start readout (still activated)
#define ROUT_DIRECT     0x24  // run readout
#define ROUT_ACT_NORM   0x25  // activate normal readout
#define ROUT_ACT_TEST   0x26  // activate test(total) readout
//
//--------- Receive Bytes
//
#define RxDATA          0x30  // Received Data ID
#define RxRO_DATA       0x40  // Bit ID for ReadOut Data
#define RxRO_NEXT       0x20  // Bit ID to increment the ROW
#define RxRO_NULL       0x10  // Bit ID for non Data
//
//--------- constants for ReadOut
//
#define REGS         64       // # register of RAL111
#define MAX_ROWS     64       // # of rows (# of Boards or RAL111) per column
```

**Listing 1 Header File**

## 5.1 Initialisierung

In der Initialisierung werden unter anderem Pointer auf PCI Regionen und PCI Register initialisiert. Wenn 2 oder 4 Byte gesendet werden (*dev_regs->dreg.word* bzw. *dev_regs->dreg.dword*), wird zuerst das niederwertige Byte gesendet.

```
DEV_REGS *dev_regs;     // HOTLink device registers


   if (pci_attach(&region, HOTLINK_DEVICE_ID, 0)) return 1;

   hotl_ok=0;
   if (pci_ok == 2) {
      hotl_ok=1;
      dev_regs=pci_ptr(&region, 1, 0);
      if (!dev_regs) hotl_ok=0;
   }

   return 0;
}
```

## 5.2 Generelles Reset

```
//
//-------------------------- gen_reset -----------------------------------
//
int gen_reset(void)
{
   region.amcc_regs->mcsr =0;
   dev_regs->dreg.byte=CTL_RESET;
   dev_regs->dreg.byte=0;
   dev_regs->screg=INT_STATES|CLEAR_FIFO;
   region.amcc_regs->mcsr=AD2FIFO_RST |FIFO2AD_RST;
   dev_regs->creg =REC_ENABLE;
   if (!(dev_regs->screg &CARRIER)) return E_NO_CARRIER;
//
//--- it need 2 times to clear FIFO's
//
   dev_regs->screg=INT_STATES|CLEAR_FIFO;
   region.amcc_regs->mcsr=AD2FIFO_RST |FIFO2AD_RST;
   region.amcc_regs->mwar =p_dma_buf;
   region.amcc_regs->mwtc =MAX_DMA_LEN;
   region.amcc_regs->mcsr =MEMWR_ENBL;
   return 0;
}
```

# 6 Laden und Test der DAC Register

Die Schwellen für die Diskriminatoren können für jeden Kanal individuell eingestellt werden. Für die Erzeugung der analogen Schwellenwerte wird der serielle DAC **MAX528** von der Firma MAXIM verwendet (siehe auch Datenblatt). Diese Chips haben 8 Kanäle, d.h. auf jeder ReadOut Karte mit 16 Kanälen befinden sich zwei **MAX528**.

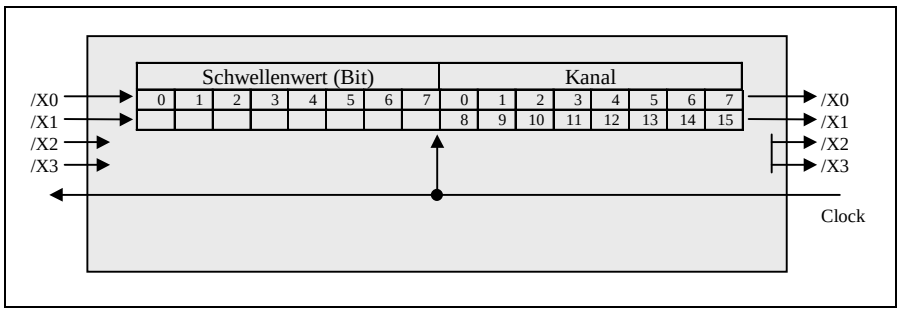


**Abbildung 4**

Die beiden DAC's befinden sich in Bit 0 (**X0**) und Bit 1 (**X1**) des seriellen Daten-Kanals. Bit 2 und 3 des 4 Bit breiten Daten-Kanals werden nicht dabei nicht benutzt und mit dem konstanten Wert 1 zurückgelesen. Es ist zubeachten, daß die Datenbit's auf dem seriellen Daten-Kanal "*low active*" sind und nicht invertiert werden, wenn sie in die DAC's eingeschrieben bzw. ausgelesen werden. D.h. die Software muß die Daten invertieren.

Für das Laden mit Überprüfung muß man sich vorstellen, daß alle DAC Register in einer Spalte zu einem Schieberegister zusammengeschaltet sind. Die folgende Abbildung zeigt das logische Blockschaltbild beim Beschreiben der DAC Register.

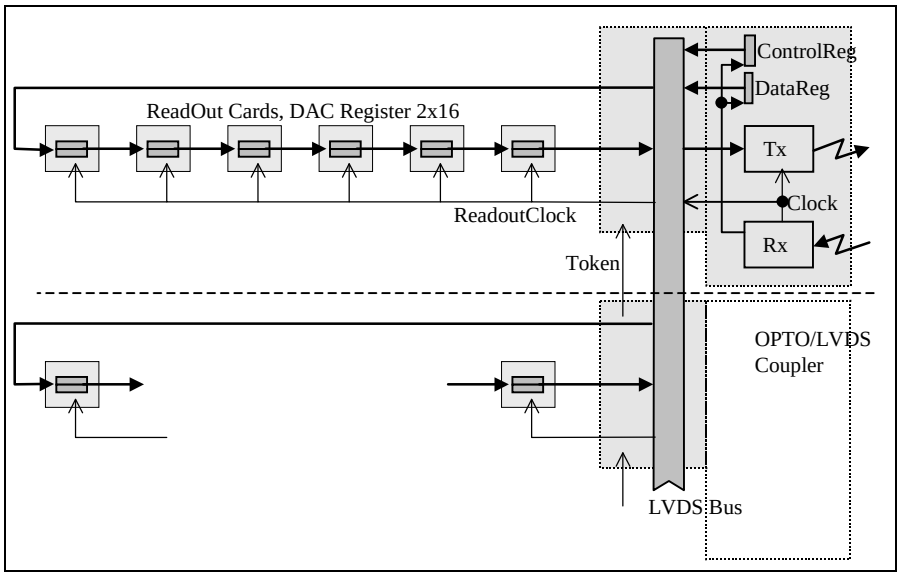


**Abbildung 5**

Mit jedem Readout Clock Impuls werden Bit 0 und Bit 1 vom Datenregister in den beiden linken DAC Chips geschrieben. Alle Bit's werden dabei um eine Stelle nach rechts geschoben, wobei die beiden letzten Bit's über die HOTLink Opto Kopplung zum Rechner gesendet werden. Da jeder DAC Chip 16 Bit hat, ist die Gesamtzahl der Clock Impulse 16 mal Anzahl der ReadOut Karten, um alle Bit's zu beschreiben. Die Anzahl kann auch mit einem entsprechenden Programm festgestellt werden.

Wenn mehrere Spalten angeschlossen sind, wird der Zugriff auf den einzelnen Spalten durch einen Token gesteuert. Nach einem Reset hat die entfernteste Spalte (Spalte 0) den Token. Wenn im Control Register die Kennung zum Beschreiben der DAC Register zurückgesetzt wird, wird der Token zur nächsten Spalte weitergegeben. Somit kann man nacheinander alle Spalten beschreiben. Ein

entsprechender Ausschnitt aus einem Testprogramm ist in **Listing 2** dargestellt. Um zu prüfen, ob die Daten richtig geladen wurden, müssen die Daten zweimal geladen werden.

```
//
//--------------------------- load_dac_regs ---------------------------------
//
int load_dac_regs(
   CONF_ROMOD  *romod)  // value record
{
   int   ret;
   u_int col;
   u_int row;
   u_int ch0;
   u_int cha;
   u_int ix,iy;
   u_int a0,b0,a1,b1;

   ch0=0;      // ..7
   do {
      if ((ret=gen_reset()) != 0) return ret;

      dev_regs->dreg.byte=STP_FCLK;
      col=0;
      do {
         dev_regs->dreg.byte=STP_FCLK |CTL_LD_DAC; ix=0;
         do {                                     // 2 times to verify data
            row=0; cha=ch0;
            do {
               a0=1<<ch0; b0=romod->column[col].dac[cha];
               a1=a0;     b1=romod->column[col].dac[cha+8];
               iy=8;
               do {
                  iy--;
                  dev_regs->dreg.byte=(SET_DREG |0x3) -(((a0 >>iy) &1) |
                                       (((a1 >>iy) &1) <<1));
                  dev_regs->dreg.word=CLOCK *0x0101;  // makes a clock length of 80ns
               } while (iy);

               iy=8;
               do {
                  iy--;
                  dev_regs->dreg.byte=(SET_DREG |0x3) -(((b0 >>iy) &1) |
                                       (((b1 >>iy) &1) <<1));
                  dev_regs->dreg.word=CLOCK *0x0101;
               } while (iy);

               cha +=16;
            } while (++row < romconf.column[0].rows);

         } while (++ix < 2);

         dev_regs->dreg.byte=STP_FCLK;

         if ((ret=read_fifo(0x8)) != 0) return ret;

         if (dma_len != 32*romod->column[col].rows) {
            printf(
"load DAC col %u: wrong # of clocks, DataBus disturbed? (see DMA Buf)\n",
                    col);
            return E_GEN;
         }

         row=0; cha=ch0;
         ix=16*romod->column[col].rows;
         do {
            a0=1<<ch0; b0=romod->column[col].dac[cha];
            a1=a0;     b1=romod->column[col].dac[cha+8];
            iy=8;
            do {
               iy--;
               if (dma_buf[ix++] !=
                        (RxDATA |0xF) -(((a0 >>iy) &1) |
                                        (((a1 >>iy) &1) <<1))) {
                  printf("DAC read back error\n"); return E_GEN;
               }
            } while (iy);

            iy=8;
            do {
               iy--;
               if (dma_buf[ix++] !=
```

```
                        (RxDATA |0xF) -(((b0 >>iy) &1) |
                                       (((b1 >>iy) &1) <<1))) {
                    printf("DAC read back error\n"); return E_GEN;
                 }
              } while (iy);

              cha +=16;
           } while (++row < romod->column[col].rows);

        } while (++col < romod->columns);

     } while (++ch0 < 8);

     dev_regs->dreg.byte=0;
     return 0;

}
```

**Listing 2 Laden der Diskriminator-Schwellen**

## 6.1 Breite des Clockimpulses

Laut Datenblatt muß die Clockbreite größer oder gleich 80ns sein. Durch die HOTLink™ Clock von 25MHz wird durch das Kommando-Byte *CLOCK* ein Impuls von 40ns erzeugt. Um die auf die doppelte Breite zu kommen, müssen zwei *CLOCK* Kommando-Bytes direkt hintereinander gesendet werden.

# 7 Laden und Test der Pattern Register

Für den Selbsttest besitzt jeder RAL111 Chip (jede Karte) ein 16 Bit Patternregister. In diesem Register ist jedes Bit einem Kanal zugeordnet. Da der Datenbus 4 Bit breit ist, sind die 16 Bit in einem 4 mal 4 Bit Schieberegister angeordnet, wobei die Kanalzuordnung entsprechend der **Abbildung 6** ist. Wenn ein Bit gesetzt ist, wird beim folgenden ReadOut für den entsprechenden Kanal ein Ereignis erzeugt.

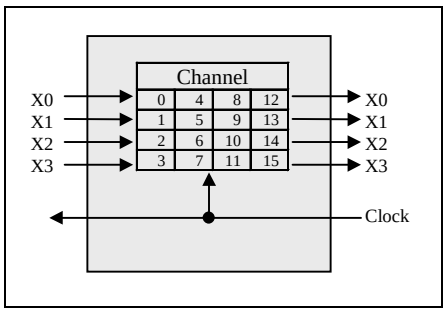


**Abbildung 6 Kanalzuordnung der Pattern-Registers**

Für das Laden mit Überprüfung muß man sich vorstellen, daß alle Pattern-Teilregister mit jeweils 4 Bit in einer Spalte zu einem Schieberegister zusammengeschaltet sind. Die folgende Abbildung zeigt das logische Blockschaltbild beim Beschreiben der Pattern-Register.

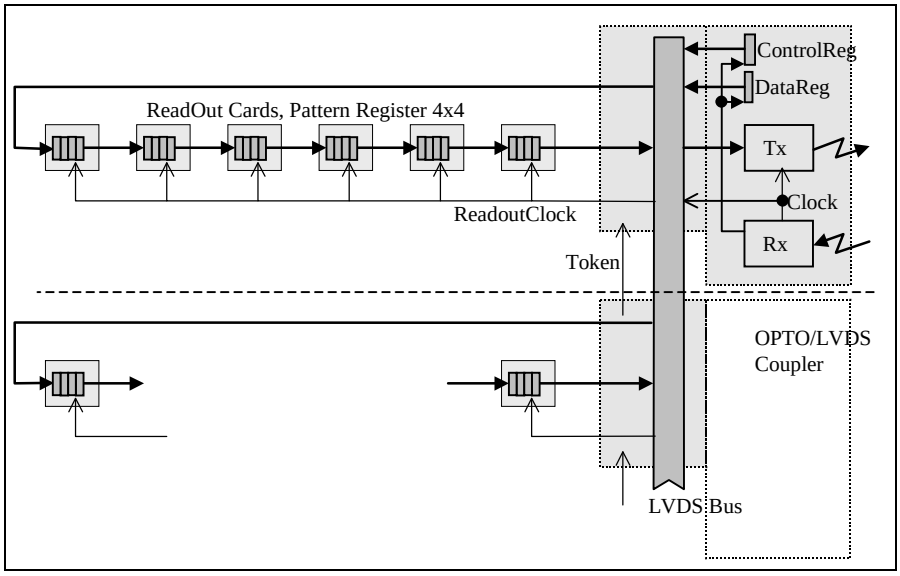


**Abbildung 7**

Mit jedem Readout Clock Impuls wird der Inhalt aus dem Data Register in das linke Pattern-Teilregister geschrieben und der Inhalt vom rechten Pattern-Teilregister wird über die HOTLink Opto Kopplung zum Rechner gesendet. Da auf jeder Readout Karte 1 RAL111 Chip mit 4 Pattern-Teilregistern ist, ist die Gesamtzahl der Pattern-Teilregister in einer Spalte gleich 4 mal Anzahl der Readout Karten. Die Anzahl kann mit einem entsprechenden Programm festgestellt werden.

Wenn mehrere Spalten angeschlossen sind, wird der Zugriff auf den einzelnen Spalten durch einen Token gesteuert. Nach einem Reset hat die entfernteste Spalte (Spalte 0) den Token. Wenn im Control Register die Kennung zum Beschreiben der Pattern-Register zurückgesetzt wird, wird der Token zur nächsten Spalte weitergegeben. Somit kann man nacheinander alle Spalten beschreiben. Der Programmablauf sieht wie folgt aus:

```
//
//------------------------------ load_test_regs ------------------------------
//
int load_test_regs(
   long  ctrl)    // >= 0: load all with this value
                  //   -1: load from config file
{
   int      ret;
   u_int    ix,ux;
   int      col;
   u_int    row;
   u_short  treg;

   if ((ret=gen_reset()) != 0) return ret;

   dev_regs->dreg.byte=STP_FCLK;
   col=0;
   do {
      dev_regs->dreg.byte=STP_FCLK |CTL_LD_TST; ix=0;
      do {                                    // load 2 times to verify data
         row=0;
         do {
            if (ctrl < 0) treg=romconf.column[col].test_reg[row];
            else treg=(u_short)ctrl;

            ux=16;
            do {
               ux -=4;
               dev_regs->dreg.word=(CLOCK<<8) |SET_DREG | ((treg >>ux) &0x0F);
            } while (ux);

         } while (++row < romconf.column[col].rows);
      } while (++ix < 2);

      dev_regs->dreg.byte=STP_FCLK;
      if ((ret=read_fifo(0x8)) != 0) return ret;

      if (dma_len != 8*romconf.column[col].rows) {
         printf(
"load TEST col %u: wrong # of clocks, DataBus disturbed? (see DMA Buf)\n",
                col);
         return E_GEN;
      }

      row=0; ix=4*romconf.column[col].rows;
      do {
         if (ctrl < 0) treg=romconf.column[col].test_reg[row];
         else treg=(u_short)ctrl;

         ux=16;
         do {
            ux -=4;
            if (dma_buf[ix++] != (RxDATA | ((treg >>ux) &0x0F))) {
               printf("TEST read back error\n"); return E_GEN;
            }
         } while (ux > 0);

      } while (++row < romconf.column[col].rows);

      col++;
   } while (col < romconf.columns);

   dev_regs->dreg.byte=0;
   return 0;
}
```