DL635E

System-Clock Slave Modul

1. FUNKTION

1.1. Datenblatt

1.1.1. Anwendung

System-Slave zur Synchronisation von Experimentdaten (Events).

1.1.2. Daten

Parameter	Wert	Dimension
Registertiefe	32	bit
Zeitauflösung (Master)	1	ms
Zeitbereich (Master)	ca. 49	Tage
Serielle Ausgänge	..4	NIM
Transferrate	1	Mbps
Steuerausgänge	..4	NIM

1.1.3. Besonderheiten

Serieller Eingang zur Synchronisation mit Master.

Serielle Ausgänge zur Synchronisation von weiteren Slave-Modulen.

Ausgänge programmierbar.

1.1.4. Aufbau

DL600-Submodul

1.1.5. Stromversorgung

Spannung	Strom	Leistung
+5V	380mA	1,9W
Gesamt		1,9W

1.2. Blockdiagramm

1.3. Beschreibung

Das Modul besitzt einen 16 MHz Quartzoszillator, der nach einem Vorteiler (/16) eine Clock mit der Periode von 1µs liefert. Diese dient als Zeitbasis für die serielle Kommunikation mit 1Mbps.

Die Information wird über den Eingang in serieller Form (1 Startbit, D0..D31 Datenbits, Stopbit) mit 1Mbps eingelesen. Ein 32-Bit Register fungiert hier als Schieberegister und kann nicht direkt ausgelesen werden.

Für eine unabhängige Auslese wird das Register normalerweise nach jedem kompletten Datenwort in ein 32Bit Register umgeladen und kann von dort fehlerfrei ausgelesen werden. Da das Umladen auch zwischen zwei (16Bit) Zugriffen vom Rechner erfolgen kann, muß dieses in diesem Fall ausdrücklich vor dem Lesen gesperrt und anschließend wieder freigegeben werden.

2. BETRIEB

2.1. Konfigurierung

2.1.1. Inputs

Der Eingang (oberste Lemobuchse) dient zum Empfangen der seriellen Informationen.

Die rote LED zeigt bei Input von einem Clock-Master im Sekundentakt die Funktion des Zeit-Zählers an!

2.1.2. Outputs

Die Ausgänge (4 untere Lemobuchsen) in NIM können in verschiedener und unabhängiger Weise programmiert werden:

1) Serieller Datenstrom zur Synchronisation von weiteren Slave-Modulen.

2) Ausgabe eines beliebigen Datenbits D31..D0 des Registers.

3) statische Ausgabe.

Die grünen Leuchtdioden zeigen die Pegel der Ausgänge (leuchtet bei NIM=16mA).

2.2. Bedienung

Keine manuellen Bedienungselemente.

2.3. Programmierung

2.3.1. Initialisierung

Bei Sys-Reset werden ALLE Bits (Modes) auf Null gesetzt.

2.3.2. Speicherbelegung

Alle Adressen sind als Offset zur Modul-Adresse definiert. Die Zählweise hier ist Byte-orientiert.

D.h. für VME-Module (DL600) sind die Adressen wie angegeben zur Modul-Adresse zu addieren, z.B:

$F200 = DL600 Modul-Basisadresse (Short IO; Hex Address Switch=$F2);

$0040 = DL635D Submodul-Adresse (Modul 2);

$000A = Subadresse Select Mode Output 3 (siehe Tabelle)

=======

$F24A = Adresse für Funktion

	D15 HighByte **D7**	LowByte **D0**	
R.$0000	D15..	..D0	Auslese Register Low Word

W=Write, R=Read, SH=Short, STD=Standard, EXT=Extended, $=HEX, ss=Address-Switches;

Die Bedeutung der Datenbits D5..D0 bei Select Mode Output 1..4 ist wie folgt:

D5	D4	D3	D2	D1	D0	Bedeutung
0	0	x	x	x	x	Ausgang = Serieller Eingang (Default)
0	1	x	x	x	s	Ausgang = s (1= NIM aktiv)
1	b	b	b	b	b	Ausgang = Datenbit Db (b=0..31)

2.3.3. Beispiel

```
const
    VMESubModuleBase = $F240;
    RegisterLow = VMESubModuleBase+$0;
    RegisterHigh = VMESubModuleBase+$2;
    Inhibit = VMESubModuleBase+$4;
    Mode1 = VMESubModuleBase+$6;
    Mode2 = VMESubModuleBase+$8;
    Mode3 = VMESubModuleBase+$A;
    Mode4 = VMESubModuleBase+$C;
    Reset = VMESubModuleBase+$E;
begin
    Reset^ := 0;        {Daten irrelevant, setzt alle ModeBits auf 0}

    Mode1^ := 0;                {Ausgang1 auf Seriell Out zum Treiben eines weiteren Moduls gesetzt}
    Mode2^ := 32 + 9;           {Ausgang2 gibt Bit9 des Registers aus}
    Mode3^ := 16 + 0;           {Ausgang3 auf statisch NIM=inaktiv gesetzt}
    Mode4^ := 16 + 1;           {Ausgang4 auf statisch MIM=aktiv gesetzt}

    Inhibit^ := 1;      {Disable Update}
    Register := (RegisterHigh^ shl 16) + RegisterLow^; {Einlesen des Registers}
    Inhibit^ := 0;      {Freigabe Update}
end;
```

3. FERTIGUNG

3.1. Mechanik

3.1.1. Frontplatte

3.1.2. Gehäuse

3.1.3. …

3.2. Elektronik

3.2.1. Schaltbild

3.2.2. Bestückungsplan

3.2.3. Stücklisten

3.2.4. Platinenunterlagen

3.2.5. Jumper

J1.2-3; J2.2-3; J3.2-3: Lemobuchsen als Ausgänge

J21 On : Clock auf Pin 79;

Alle weiteren Jumper Off;

3.2.6. Abel-File

```
Module      DL635E;

Title 'ClockSlave vwa090796 V3.0'
" This is a Clock Slave module, which receives 32 Bits (time) information serially
" with 1MBPS.
" The data can be read out:
" For consistent readout set MUpdate to High (disables update of register), read data
" and then clear MUpdate again.
" Four outputs (W,X,Y,Z) can be programmed independently to:
" 1) output the serial datastream to further modules
" 2) output an arbitrary databit from the received info (0..31)
" 3) output a programmable state

Declarations

    DataIn                  PIN 27;
    Out1..Out4              PIN 11,12,9,10;
    TP1..TP8                PIN 36,34,23,21,7,5,117,115;
    Clock                   PIN 79;
    !SysReset               PIN 82;
    !RD,!WR                 PIN 81,80;
    A4..A1                  PIN 83..86;
    D15..D0                 PIN 72,70,68,66,64,62,57,59,71,69,67,65,63,61,58,60;
    MUpdate                 NODE isType 'reg_d,buffer';
    MUpdateSync             NODE isType 'reg_d,buffer';
    S1..S0                  NODE isType 'reg_d,buffer';
    B4..B0                  NODE isType 'reg_d,buffer';
    T3..T0                  NODE istype 'reg_d,buffer';
    C31..C0                 NODE istype 'reg_d,buffer';
    R31..R0                 NODE istype 'reg_d,buffer';
    W5..W0                  NODE istype 'reg_d,buffer';
```

```
    X5..X0                          NODE istype 'reg_d,buffer';
    Y5..Y0                          NODE istype 'reg_d,buffer';
    Z5..Z0                          NODE istype 'reg_d,buffer';
    WCmp4..WCmp0                    NODE istype 'xor';
    XCmp4..XCmp0                    NODE istype 'xor';
    YCmp4..YCmp0                    NODE istype 'xor';
    ZCmp4..ZCmp0                NODE istype 'xor';
    WBit, XBit, YBit, ZBit          NODE istype 'reg_d,buffer';

    Addr                            = [A4..A1];
    Data                            = [D15..D0];
    DataNib                 = [D5..D0];
    Timer                           = [T3..T0];
    Counter                 = [C31..C0];
    CounterS                        = [C30..C0,.X.];
    Register                = [R31..R0];
    RegisterL                       = [R15..R0];
    RegisterH                       = [R31..R16];
    Bits                            = [B4..B0];
    Status                          = [S1..S0];
    WOut                            = [W5..W0];
    WSel                            = [W4..W0];
    XOut                            = [X5..X0];
    XSel                            = [X4..X0];
    YOut                            = [Y5..Y0];
    YSel                            = [Y4..Y0];
    ZOut                            = [Z5..Z0];
    ZSel                            = [Z4..Z0];
    WCmp                            = [WCmp4..WCmp0];
    XCmp                            = [XCmp4..XCmp0];
    YCmp                            = [YCmp4..YCmp0];
    ZCmp                            = [ZCmp4..ZCmp0];
    SIDLE                           = 0;
    SSTART                          = 1;
    SDATA                   = 2;
    SSTOP                   = 3;

    RCounterL                       = (Addr==0) & RD;
    RCounterH                       = (Addr==1) & RD;
    WUpdate                         = (Addr==2) & WR;
    WWSel                   = (Addr==3) & WR;
    RWSel                           = (Addr==3) & RD;
    WXSel                           = (Addr==4) & WR;
    RXSel                           = (Addr==4) & RD;
    WYSel                           = (Addr==5) & WR;
    RYSel                           = (Addr==5) & RD;
    WZSel                           = (Addr==6) & WR;
    RZSel                           = (Addr==6) & RD;
    MReset                  = (Addr==7) & WR;
    SReset                          = SysReset # MReset;

    STimer                          = [1,1,1,1]; "wait for 16 clocks
    STimerH                         = [0,1,1,1]; "wait for 8 clocks
    NumBits                         = [1,1,1,1,1];

Equations

"   TP1                 = Timer==0;
"   TP2                 = Bits==0;
    TP3                 = R9;     "LED red
    TP4                 = !Out1; "LED green
    TP5                 = !Out2; "LED green
    TP6                 = !Out3; "LED green
    TP7                 = !Out4; "LED green
"   TP8                 = MUpdate;

    Status.ACLR         = SReset;
```

```
Status.CLK              = Clock;

MUpdate.ACLR  = SReset;
MUpdate.CLK             = !WUpdate;
MUpdate                 := D0;

MUpdateSync.ACLR        = SReset;
MUpdateSync.CLK         = Clock;
MUpdateSync             := MUpdate;

Bits.ACLR               = SReset;
Bits.CLK                = Clock;

Timer.ACLR              = SReset;
Timer.CLK               = Clock;

Counter.ACLR            = SReset;
Counter.CLK             = Clock;

Register.ACLR           = SReset;
Register.CLK            = Clock;

WOut.ACLR               = SReset;
WOut.CLK                = !WWSel;
WOut                    := DataNib;

XOut.ACLR               = SReset;
XOut.CLK                = !WXSel;
XOut                    := DataNib;

YOut.ACLR               = SReset;
YOut.CLK                = !WYSel;
YOut                    := DataNib;

ZOut.ACLR               = SReset;
ZOut.CLK                = !WZSel;
ZOut                    := DataNib;

WBit.ACLR               = SReset;
WBit.CLK                = Clock;
when W5 then
        Out1=WBit;
else
        when W4 then
                Out1=!W0
        else
                Out1= DataIn;

XBit.ACLR               = SReset;
XBit.CLK                = Clock;
when X5 then
        Out2=XBit;
else
        when X4 then
                Out2=!X0
        else
                Out2= DataIn;

YBit.ACLR               = SReset;
YBit.CLK                = Clock;
when Y5 then
        Out3=YBit;
else
        when Y4 then
                Out3=!Y0
        else
                Out3= DataIn;
```

```
    ZBit.ACLR           = SReset;
    ZBit.CLK            = Clock;
    when Z5 then
        Out4=ZBit;
    else
        when Z4 then
            Out4=!Z0
        else
            Out4= DataIn;

    Data                = RegisterL & RCounterL;
    Data                = RegisterH & RCounterH;
    Data.OE         = RCounterL # RCounterH;

    WCmp  = Bits $ !WSel;
    XCmp  = Bits $ !XSel;
    YCmp  = Bits $ !YSel;
    ZCmp  = Bits $ !ZSel;

    when (Status==SDATA) & (Timer==0) then {
        when (WCmp==0) then WBit:=DataIn else WBit:=WBit;
        when (XCmp==0) then XBit:=DataIn else XBit:=XBit;
        when (YCmp==0) then YBit:=DataIn else YBit:=YBit;
        when (ZCmp==0) then ZBit:=DataIn else ZBit:=ZBit;}
    else {
        WBit:=WBit;
        XBit:=XBit;
        YBit:=YBit;
        ZBit:=ZBit;}

State_Diagram Status;

    STATE SIDLE:
        Register:=Register;
        If DataIn then SIDLE else SSTART with Timer:=STimerH;

    STATE SSTART:
        Register:=Register;
        If DataIn then
            SIDLE
        else    If Timer==0 then SDATA with {
                    Timer:=STimer;
                    Bits:=NumBits;}
            else SSTART with Timer:=Timer-1;

    STATE SDATA:
        Register:=Register;
        if Timer==0 then
            if Bits==0 then SSTOP with {
                C31:= DataIn;
                CounterS:=Counter;
                Timer:=STimer;}
            else SDATA with {
                Bits:=Bits-1;
                Timer:=STimer;
                C31 := DataIn;
                CounterS:=Counter;}
        else SDATA with {
            Bits:=Bits;
            Timer:=Timer-1;
            Counter:=Counter;}

    STATE SSTOP:
        Counter:=Counter;
        if Timer==0 then SIDLE with
            when MUpdateSync then Register:=Register else Register:=Counter
```

```
        else SSTOP with {
                Timer:=Timer-1;
                Register:=Register;}

    End
```

4. TEST

4.1. Aufbau

4.2. Ergebnisse

4.3. …

5. MODIFIKATION

5.1. Version

5.2. …

6. ANHANG

6.1. Bausteinunterlagen

6.2. …