Switching control unit for data communication via RS232 AN040

INTRODUCTION

In many laboratories and institutions one can still find many PCs or similar computer equipment which are not connected via a network. There is often a need for occasional communication and data exchange in these cases and this application note describes a inexpensive solution, allowing intercommunications of up to eight stations.

The specific goal of this design was to connect six PCs and two PLD-Programmers in such a way that any two of them could communicate. Each of these devices has a serial interface (RS 232), which may be connected to another device in a null modem configuration by connecting theTrD (Transmit Data) and RcD (Receive Date) lines of the two selected stations. With the addition of readily available software for the PCs that handshakes using a XON/XOFF protocol, the design realizes a digital switching network.

The main part of the developed design represents a MUX-DEMUX-circuit realizing the primary switching network. One Multiplexer for each direction switches the TrD line of an activated station to an internal crosspoint and a Demultiplexer connects it to the RcD wire of the corresponding station. The choice of the stations to be connected is set up by the user via mechanical switches, so that an additional priority encoder has to guarantee the activation of only two stations at a time. In order to display the actual status of the switching network, two decoders are added which can drive 7-Segment-Displays directly and indicate the actual connection.

Figure 1 shows the basic interconnection to be realized by the switching network and the structure of the complete design is shown in Figure 2. Beside the Line Buffers (DS 232) the complete design was to be implemented using just one programmable logic component. A Philips Semiconductor's PLHS501 proved to be an excellent choice for this design.

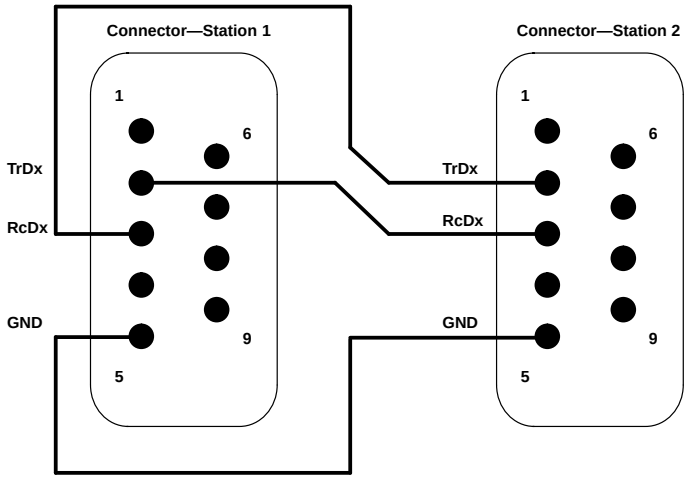

Figure 1. Interconnection via RS232 for a Simple XON/XOFF Protocol

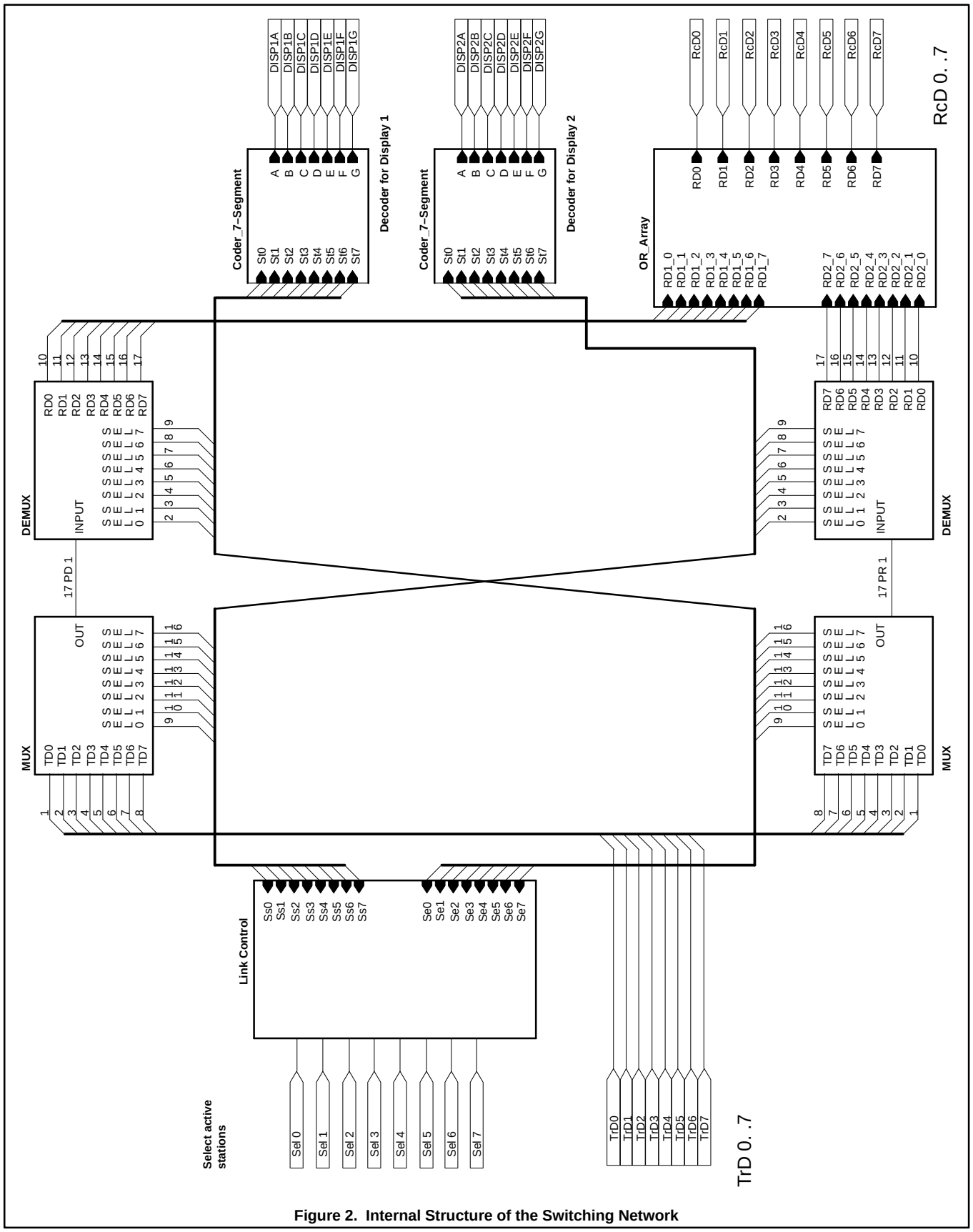

Figure 2. Internal Structure of the Switching Network

Switching control unit for data communication via RS232 AN040

BASIC OPERATION

Link Control and Priority Encoder

The first module of the design is the link control unit. As inputs it has eight selection signals indicating the choice of two stations which are to be connected. This primary selection will be converted to internal control signals for the switching network by the link control. As outputs there are 2x8 signals (8 for each direction of data flow) controlling the multiplexer and demultiplexer directly.

In addition to the basic function, the link control has to ensure that just two stations are connected with each other, even if more are activated by the switches. In order to fulfill this constraint there has been implemented an implicit priority encoder satisfying the following rules:

- if just two stations are chosen by the switches (the normal case) then an interconnect of these stations will be established;
- if only one switch is set active, the transmission signal of this station is directed to the Receive line of the same station (self test);
- if no switch is activated no interconnection is set up at all;
- if more then two switches were set to the 'Active' level just the two stations with the highest and lowest number (Station 0, Station 7 – highest priority) are interconnected with each other.

Realizing such privileges the appropriate function of the switching network is guaranteed in a way, that never more then two stations are linked together.

Multiplexer

Two multiplexers are the first part of the internal switching network. Directly controlled by the output of the link control module the multiplexers have the task to switch the transmission line of a selected station to an internal crosspoint. Since both directions of data flow are to be supported, two multiplexers exist which link the TrD lines to the crosspoints PD and PR respectively.

Demultiplexer

The second part of the digital switching network is formed by two demultiplexers. While the multiplexers have to link the active transmitter to central crosspoints, the demultiplexers connect these internal nodes with the RcD line of the corresponding counter station. The links to be established are as well controlled by the link control module and in the result each set of multiplexer/demultiplexers realizes an interconnection between any TrD and RcD signal lines. Since two of these sets are contained, both directions of data flow can be satisfied and a logical OR-operation of both demultiplexer outputs completes the switching network.

Status Display

A display was added to the basic function to show the actual interconnection. Since the original task was to connect eight stations with each other two numerical displays were used to display just the number of the two active ones. In order to accomplish this function two identical decoders were created. The decoders read directly the eight control signals from the link control module and in correspondence to the active line they drive the displays with numbers 1 to 8. For the case that no station is selected, the displays will blank.

THE PML-ARCHITECTURE

Circuits with a PML-structure represent an architecture that can replace all typical PLDs. In contrast to PAL and PLA circuits which use AND and OR gates, PML is composed of just one programmable logic array using only NAND gates. The outputs of each gate folds back upon itself and all other NAND gates. Inputs, outputs and bi-directional pins are available in the same way as they are in other PLDs.

The general PML-architecture is shown in Figure 3. It is possible to realize any logical function block with this architecture. An example of implementing a D-type Latch is shown in Figure 4. Efficient design implementations require a certain design style and software that includes an appropriate simulator, optimizer and compiler. Such tools are contained in the PLD-Development Software SNAP, which can be used to implement designs within a short period of time for PML as well as for other programmable logic devices.

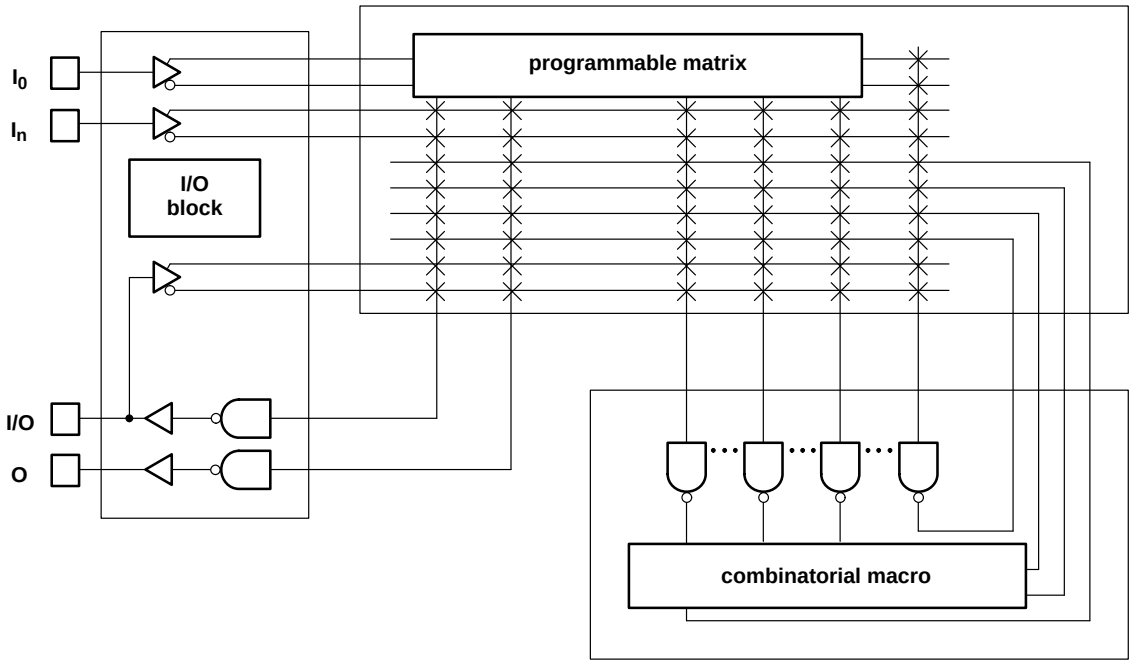

Figure 3. The Basic Architecture of PML-Circuits

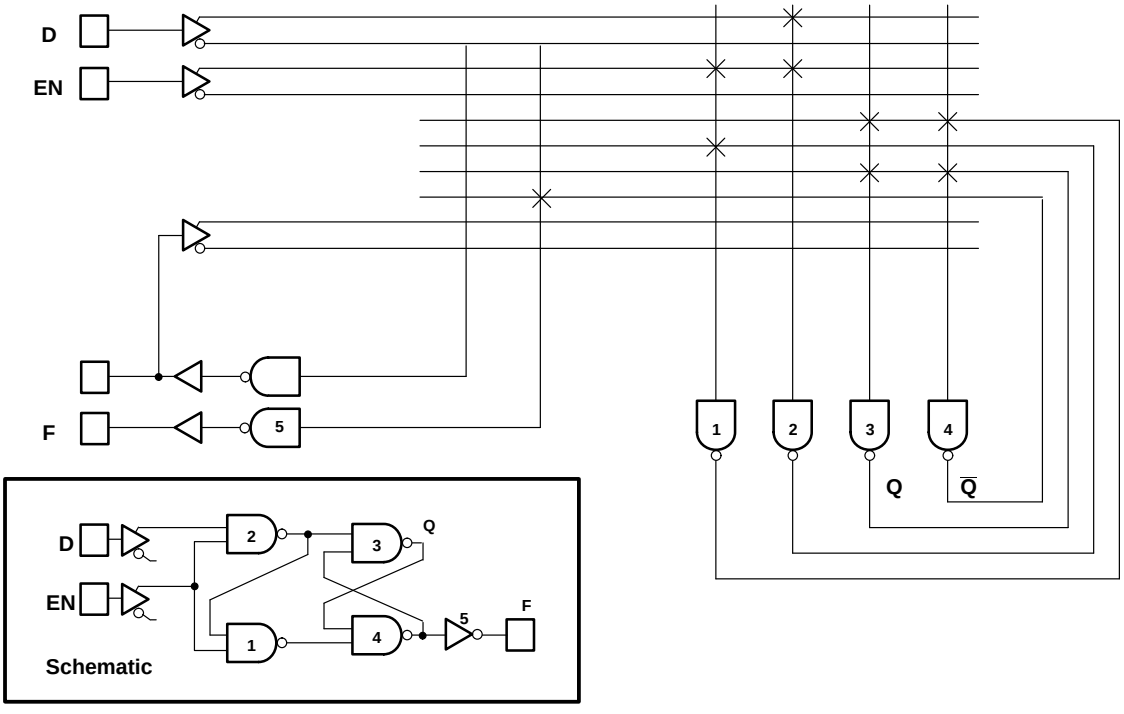

Figure 4. Realization of a D-Latch by a PML-Architecture

Switching control unit for data communication via RS232 AN040

A DESIGN TOOL FOR PML-CIRCUITS (SNAP)

Design implementation for PML as well as of other programmable logic ICs is supported by the Software tool SNAP. This program package, offered by Philips Semiconductors, guides the user through the complete design process beginning with the design description up to the automatic generation of final programming data and a corresponding test vector file. For the initial description of designs several entry tools are available. First there is the opportunity to create a network description via a schematic editor (e.g. OrCAD SDT). Alternatively SNAP defines a specific HDL (Hardware Description Language) and by using this method, a design description may be given in terms of Boolean equations, truth tables or FSM (Finite State Machines) syntax.

Whichever entry method is used becomes converted into an internal network description in a following step. The internal network format is similar to the EDIF-format and it corresponds with the data formats used by Philips Design Station for ASIC development. So it is also possible with SNAP to import designs given as EDIF-network descriptions and moreover the internal format keeps open the choice of a PLD or a Gate Array design implementation. Finally SNAP contains a minimzer module can optimize the Boolean function thereby increasing the quality of the design implementation, sometimes significantly.

After a description has been created, the point of interest is to verify the correct operation of the design. For this purpose SNAP has included an easy to use digital simulator. The simulator, LESIM3, is contained in SNAP as well as in the Philips Design Station which has been used for ASIC-development for many years.

Basically LESIM3 supports the simulation of abstract network descriptions as well as of concrete circuit models. The simulator either assumes a constant, propagation delay value for a given, bare network or it can consider the real timing relations if the model for a certain IC is available. The stimuli for a design simulation can be generated by an interactive, graphical wave form entry or a textual stimuli description. The graphical entry is very easy to handle and leads quickly to appropriate stimuli for smaller designs. The other choice, the textual notation of stimuli, bases on an own simulation control language (SCL) of LESIM3 and it allows the compact description of sophisticated sequences for extensive design verification. This opportunity serves more the needs of experienced designers and in correlation with fault simulator, test pattern generator, model generator and other options it makes possible all kinds of functional simulation and test.

The goal of design development software is the generation of ICs realizing an application specific function. Since the initial design description in SNAP is device independent, one task is to select a specific component. Even the choice of a PLD is made easy by SNAP, since all available devices are listed directly on the screen together with their most relevant data. After the designer has decided which PLD to use, there is only left the task of specifying pinning of the device. An initial pin assignment is suggested by SNAP automatically, which can be revised in an interactive process directly on the graphical presentation of the device.

If all assignments are done, the ultimate design step is the compiler run. Automatically the compiler tries to map the given network on the selected PLD-architecture and in case of success a programmer file is generated (the format of programmer file follows JEDEC No.3A standard). This file can be downloaded onto a device programmer directly from the SNAP-shell, so that the IC-implementation can be completed immediately by programming the physical device. If on the other hand the compiler fails in its run the appeared problems will be listed in a status file and in this way the designer gets hints how to modify the design for a successful compilation. Especially in these cases SNAP has an essential advantage, since the SNAP-shell with its clear structure (Figure 5) allows additional design iterations quickly and furthermore automated.

DEVELOPMENT OF THE SWITCHING CONTROL UNIT WITH SNAP

Design Description

The general functionality of the intended switching network has been outlined already in section two. There, the complete design has been subdivided into basic modules, of which the logical function can easily be described by boolean equations. Therefore it makes sense to use the ‘Equations’ entry of SNAP for design description and activating this module from the SNAP-shell an editor makes it possible to fill in a script of an empty HDL-file. Using the Equation-entry any design definition has to follow the specific HDL-syntax of SNAP and Figure 6 shows this description for the complete design.

Referring to the listing (Figure 6) the basic structure of HDL-files can be demonstrated. The first section ‘@PINLIST’ serves for the definition of primary inputs, outputs and bi-directional pins of a design. Just the names of the ports and the port types are fixed here. After declaring the I/O-part the section ‘@LOGIC EQUATIONS’ contains the definition of the designs functionality. In this part the output is to specify in terms of boolean equations, in which inputs, intermediate signals and logical operators can be used. The basic operators are AND (’*’), OR (’+’), NOT (’/’) and EXOR (’:+:’), but more abstract operations are available too.

Within a HDL-file additional sections can be used for describing truth tables and finite state machines, but since they are not necessary for the switching network they are not explained here. Finally it should be mentioned that any project file in SNAP gets a specific file name extension upon completion of each development stage, so that the created Equation-file is marked with ‘.EQN’ and the project file is named as NullMod.Eqn.

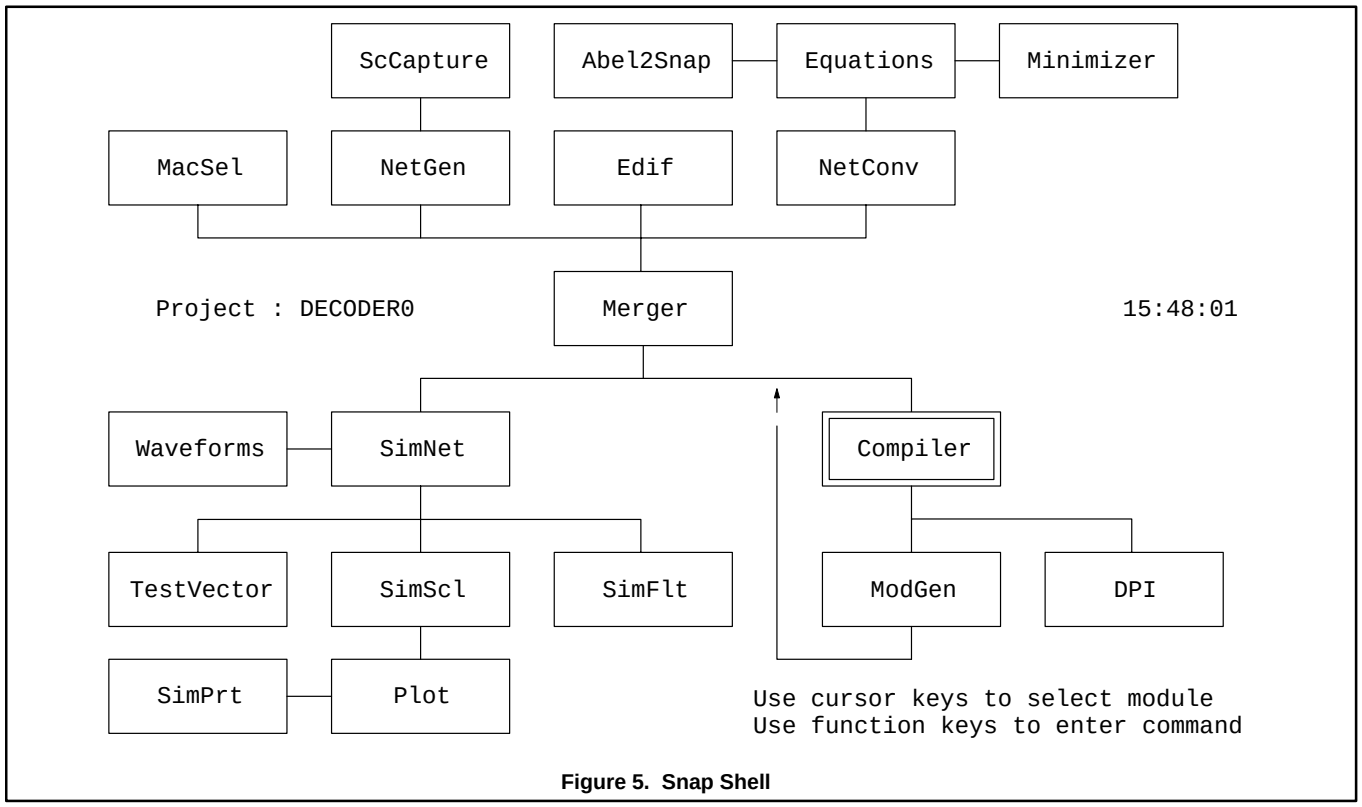

Figure 5. Snap Shell

```
@PINLIST
Sel0       I;
Sel1       I;
Sel2       I;
Sel3       I;
Sel4       I;
Sel5       I;
Sel6       I;
Sel7       I;
TrD0       I;
TrD1       I;
TrD2       I;
TrD3       I;
TrD4       I;
TrD5       I;
TrD6       I;
TrD7       I;
RcD0       O;
RcD1       O;
RcD2       O;
RcD3       O;
RcD4       O;
RcD5       O;
RcD6       O;
RcD7       O;
Disp1A     O;
Disp1B     O;
Disp1C     O;
Disp1D     O;
Disp1E     O;
Disp1F     O;
Disp1G     O;
Disp2A     O;
Disp2B     O;
Disp2C     O;
Disp2D     O;
Disp2E     O;
Disp2F     O;
Disp2G     O;

@LOGIC EQUATIONS

Ss0 = Sel0 ;
Ss1 = /Sel0 * Sel1;
Ss2 = /Sel0 * /Sel1 * Sel2;
Ss3 = /Sel0 * /Sel1 * /Sel2 * Sel3;
Ss4 = /Sel0 * /Sel1 * /Sel2 * /Sel3 * Sel4;
Ss5 = /Sel0 * /Sel1 * /Sel2 * /Sel3 * /Sel4 * Sel5;
Ss6 = /Sel0 * /Sel1 * /Sel2 * /Sel3 * /Sel4 * /Sel5 * Sel6;
Ss7 = /Sel0 * /Sel1 * /Sel2 * /Sel3 * /Sel4 * /Sel5 * /Sel6 * Sel7;

PD =       TrD0 * Ss0 + TrD1 * Ss1 + TrD2 * Ss2 + TrD3 * Ss3 +
           TrD4 * Ss4 + TrD5 * Ss5 + TrD6 * Ss6 + TrD7 * Ss7 ;

Se7 = Sel7 ;
Se6 = /Sel7 * Sel6 ;
Se5 = /Sel7 * /Sel6 * Sel5 ;
Se4 = /Sel7 * /Sel6 * /Sel5 * Sel4 ;
Se3 = /Sel7 * /Sel6 * /Sel5 * /Sel4 * Sel3 ;
Se2 = /Sel7 * /Sel6 * /Sel5 * /Sel4 * /Sel3 * Sel2 ;
Se1 = /Sel7 * /Sel6 * /Sel5 * /Sel4 * /Sel3 * /Sel2 * Sel1 ;
Se0 = /Sel7 * /Sel6 * /Sel5 * /Sel4 * /Sel3 * /Sel2 * /Sel1 * Sel0;

PR =       TrD7 * Se7 + TrD6 * Se6 + TrD5 * Se5 + TrD4 * Se4 +
           TrD3 * Se3 + TrD2 * Se2 + TrD1 * Se1 + TrD0 * Se0;
```

Figure 6. The Complete Design Description for the Switching Network Circuit (1 of 2)

```
RcD7 = PD * Se7 + PR * Ss7 ;
RcD6 = PD * Se6 + PR * Ss6 ;
RcD5 = PD * Se5 + PR * Ss5 ;
RcD4 = PD * Se4 + PR * Ss4 ;
RcD3 = PD * Se3 + PR * Ss3 ;
RcD2 = PD * Se2 + PR * Ss2 ;
RcD1 = PD * Se1 + PR * Ss1 ;
RcD0 = PD * Se0 + PR * Ss0 ;

Disp1G = /( Se2 + Se3 + Se4 + Se5 + Se6 );
Disp1D = /( Se0 + Se2 + Se3+ Se5 + Se6 );
Disp1E = /( Se0 + Se2+ Se6 );
Disp1F = /( Se0 + Se4 + Se5 + Se6 );
Disp1A = /( Se0+ Se2 + Se3+ Se5 + Se6 + Se7 );
Disp1B = /( Se0 + Se1 + Se2 + Se3 + Se4+ Se7 );
Disp1C = /( Se0 + Se1 + Se3 + Se4 + Se5 + Se6 + Se7 );

Disp2G = /( Ss2 + Ss3 + Ss4 + Ss5 + Ss6 );
Disp2D = /( Ss0 + Ss2 + Ss3 + Ss5 + Ss6 );
Disp2E = /( Ss0 + Ss2 + Ss6 );
Disp2F = /( Ss0+ Ss4 + Ss5 + Ss6 );
Disp2A = /( Ss0 + Ss2 + Ss3+ Ss5 + Ss6 + Ss7 );
Disp2B = /( Ss0 + Ss1 + Ss2 + Ss3 + Ss4 + Ss7 );
Disp2C = /( Ss0 + Ss1 + Ss3 + Ss4 + Ss5 + Ss6 + Ss7);
```

Figure 6. The Complete Design Description for the Switching Network Circuit (2 of 2)

Generation of an Internal Netlist

As already mentioned, each design description within SNAP becomes converted into an internal netlist format first. In general this step is carried out automatically just by activating the corresponding program module and for the actual design this means to start the ‘Net Converter’ and the ‘Merger’ in succession. The network converter translates the given HDL-file into an EDIF like netlist description, while the merger has no essential meaning for this design. It is rather important for larger, hierarchical designs, where this module has to merge several function blocks into one, flat netlist.

Regardless of design complexity, the merger has to be executed for each design and the out coming netlist file with the name NullMod.Net (project name with extension ‘.NET’) forms the bases for the following simulation and for the compiler. That’s why a part of the resulting netlist is shown in Figure 7 for demonstration of this important, internal file format.

```
*****************************************************
* Output of Merger                Version 1.60    *
* Date: 10/ 1/1993                Time: 14:26:44  *
*****************************************************
* Input File Name:                NULLMOD.MAC   *
* Netlist File Name :             NULLMOD.NET    *
*****************************************************
*
NETSTART
*
SS1 AN2 I(N48_1,Sel1) O(SS1)
B48_1 INV I(Sel0) O(N48_1)
SS2 AN3 I(N49_1,Sel2,N48_1) O(SS2)
B49_1 INV I(Sel1) O(N49_1)
SS3 AN4 I(N49_1,N50_1,Sel3,N48_1) O(SS3)
B50_1 INV I(Sel2) O(N50_1)
                      #
                      #
                      #
Disp2B NO6 I(Sel0,SS1,SS2,SS3,SS4,SS7) O(Disp2B)
Disp2C NO7 I(Sel0,SS1,SS3,SS4,SS5,SS6,SS7) O(Disp2C)
*
NETEND
*
NETIN Sel0,Sel1,Sel2,Sel3,Sel4,Sel5,Sel6,Sel7,TrD0,TrD1,TrD2,
#TrD3,TrD4,TrD5,TrD6, TrD7
NETOUT RcD0,RcD1,RcD2,RcD3,RcD4,RcD5,RcD6,RcD7,
#Disp1C,Disp1B,Disp1A,Disp1F,Disp1E,Disp1D,Disp1G,
#Disp1C,Disp2B,Disp2A,Disp2F,Disp2E,Disp2D,Disp2G
*
```

Figure 7. The Netlist File NullMod.Net

Switching control unit for data communication via RS232 AN040

Design Simulation

The simulation of a design is an important step in the development of each circuit . Beside the bare functional simulation, the actual device timing relations, the testability, and last but not least an appropriate test pattern needs to be generated. All these functions can be realized by SNAP, even if only the basic procedure is demonstrated here.

Working with SNAP the first step is to convert the netlist into a more compact format easier to read for the simulator. This will be accomplished by starting the program module 'SIMNET' and as a result a binary file with the name NullMod.Bin becomes created. Additionally a stimuli file NullMod.Scl will automatically be generated too. These automatically generated stimuli are quite arbitrary and so it is advisable to modify them by the module 'WAVEFORMS' or by the available text editor. Are the binary design file and the stimuli file present the real simulator 'SIMSCL' can be run and the simulation results are stored in a project file ending with '.RES' (NullMod.Res for the switching network). These results can either be visualized graphically on the screen by 'PLOT' or printed out by the program module 'SIMPRT'.

A complete simulation takes a good deal of work for any design and it can become quite extensive. That's why only a small part of the stimuli file for the switching network shall be shown here for demonstration (Figure 8) and Figure 9 shows the corresponding section of the graphical simulator output.

```
************************************************************
* Output of Updsirn              Version 1.00          *
* Date: 10/01/93                 Time: 14:27:02        *
************************************************************
* Input File Name:               NULLMOD.NET    *
* Output File Name:              NULLMOD.SCL    *
************************************************************

P Sel0, Sel1, Sel2, Sel3, Sel4, Se0, Se1, Se2, Se3
PC0
S 0  (500) Sel0
S 0 (1000) Sel1
S 0 (1700) Sel2
S 0 (2500) Sel3
S 0 (2800) Sel4
S 0 (3100) Sel5
S 0 (3500) Sel6
S 0 (4000) Sel7

SU time = 5000
```

Figure 8. A Section of the Simulator Stimuli for the Verification of the Design

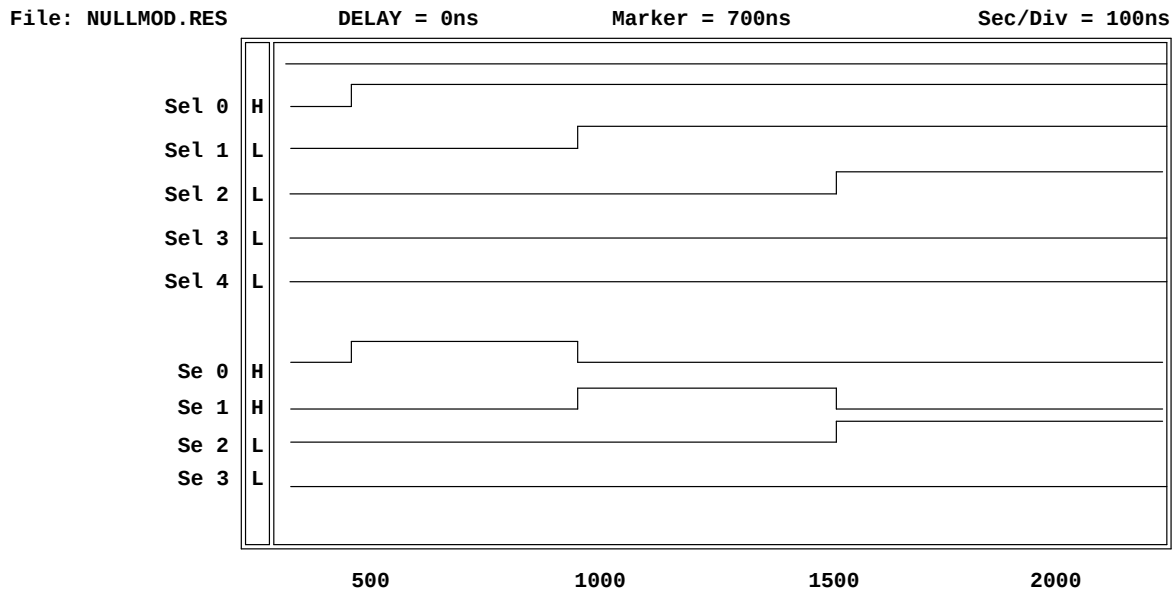

Figure 9. Graphical Presentation of Simulator Output

The Compiler

The final aim of the design process is to generate a data file in order to customize programmable logic devices with the functionality of the developed design. The programming data format generated by SNAP follows the international standard JEDEC No. 3A, which is widely accepted and used in industry and research.

The compiler block of SNAP has the task to map a given, verified netlist onto a certain device architecture and to convert the result into a JEDEC file readable by a device programmer. In order to do so a specific device should be chosen and a pin out selected. For this design, a PLHS501 is chosen within the program module 'DEVSELECT' and the Pinning needs to be entered with 'PINSELECT' in correspondence to that of Figure 10. After running these two programs, the compiler can be started which fits the design into the chosen device. In case of problems during the compilation an error file (with extension '.ERR') will be generated reporting errors, warnings and information about necessary modifications to the initial design. But for the design of the switching network the compiler finishes its work with success resulting in a JEDEC file named NullMod.Jed. This file can now directly be used to program a PLHS501 or it can be modeled for a final timing simulation.

PROGRAMMING OF THE PLHS501

Among other alternatives the UP2000 device programmer represents an interesting, efficient tool for the development of programmable logic especially in laboratories and smaller enterprises. Manufactured for use in PC environments it consists of an expansion board for PCs (8-Bit Slot), an adapter box, several adapters for mounting various package layouts, and corresponding programmer software. This device programmer and associated software supports all of the programmable logic devices that are contained within the SNAP library, so that the UP2000 is the ideal companion for the SNAP development system. The programmer software is mostly self explanatory and with some skill it can directly be included into the SNAP shell. Using the JEDEC file from SNAP and a UP2000 to program a PLHS501 revealed no problems at all. The programming of the design of the switching network into the PLHS501 was finished within one minute.

SUMMARY

The design of a digital switching network as described here was implemented within just one Philips PLHS501. Adding some buffer/drivers and the necessary switches and display elements completed the design to a working application. It was tested within a laboratory of 8 PCs, which were interconnected by the switching network via their standard serial interfaces. By changing the switches, any of the computers could be interconnected and using communication link programs such as LapLink or Norton Commander, a simple data transfer method was realized.

Switching control unit for data communication via RS232 AN040

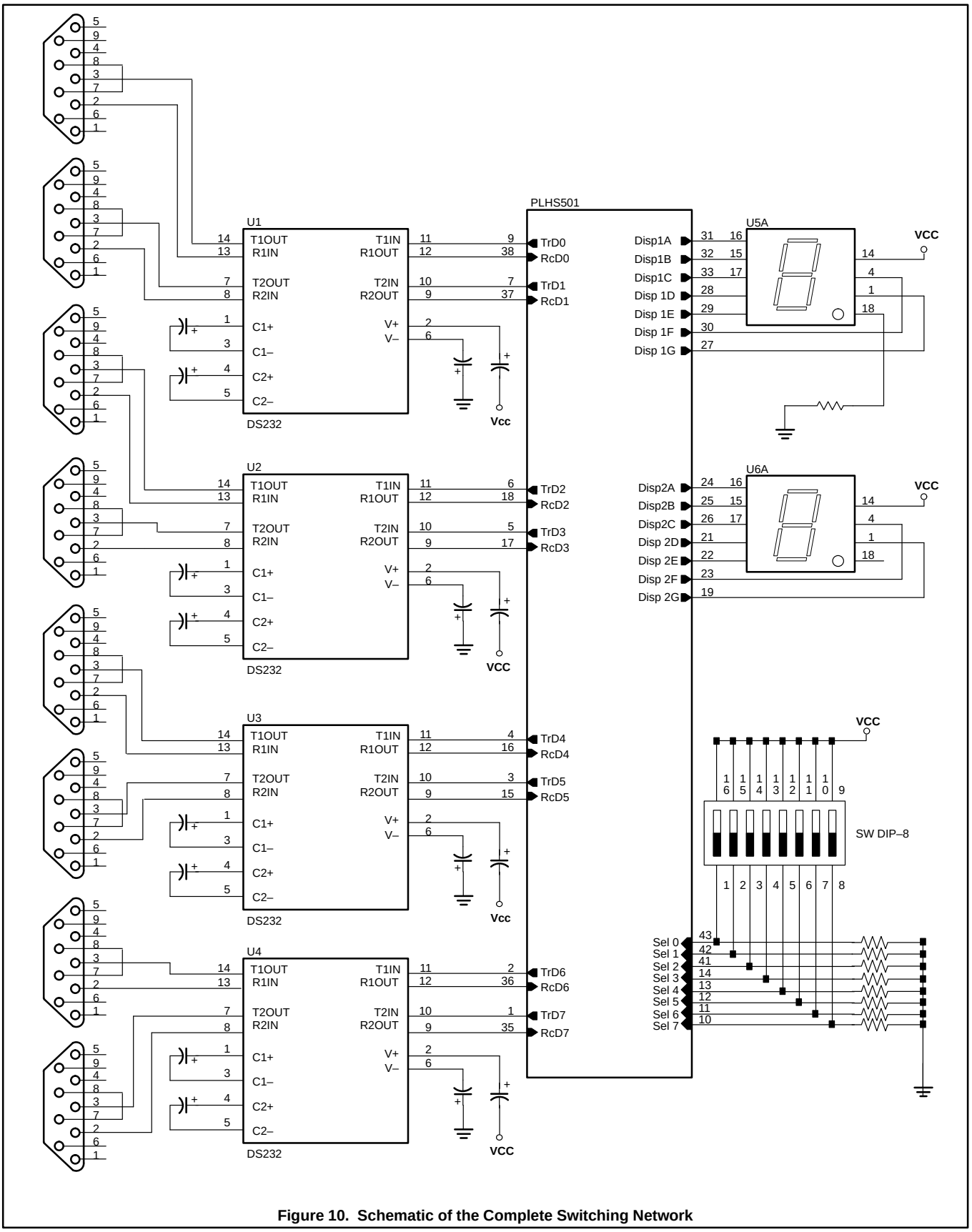

Figure 10. Schematic of the Complete Switching Network