Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2004

MSc THESIS

USB-enabled PDP8 computer

CE-MS-2004-04

Abstract

The PDP-8 family of minicomputers were built by Digital Equipment Corporation between 1965 and 1990. The PDP8 computers were one of the first computers that were affordable by a broader range of customers that contributed in the success of these machines. In 1976, Intersil developed a PDP8 chipset consisting of the IM6100 processor [6], the I/O extension (IM6101) and the UART (IM6402). The PDP8 utilizes a small instruction set. Only eight basic instructions are implemented, which provide enough functionality to compose complete programs.
A single board computing system was designed using the chipset from Intersil and a 80C32 based microcomputer system. The implemented computer is a minimal PDP8 computer with an UART to interface with legacy peripherals. The 80C32 microcomputer system controls the memory and the system state of the PDP8 sub-system. This microcomputer system communicates with a host computer via an USB interface. The USB connection is based upon the PDIUSB12 USB controller from Philips. Using the 80C32 microcomputer system and specially designed host software, executables can be loaded into the PDP8 memory and their execution can be controlled. The whole PDP8 computer is controllable from a special application running on a Microsoft Windows host computer. Borland C++ Builder was used to develop this application.
This thesis will describe the development process and results of building such a system. The design of the PDP8-computer and the PC host sofware will be presented.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

USB-enabled PDP8 computer
Specification, design & host software

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Johan van de Pol
born in Rhenen, The Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

USB-enabled PDP8 computer

by Johan van de Pol

Abstract

The PDP-8 family of minicomputers were built by Digital Equipment Corporation between 1965 and 1990. The PDP8 computers were one of the first computers that were affordable by a broader range of customers that contributed in the success of these machines. In 1976, Intersil developed a PDP8 chipset consisting of the IM6100 processor [6], the I/O extension (IM6101) and the UART (IM6402). The PDP8 utilizes a small instruction set. Only eight basic instructions are implemented, which provide enough functionality to compose complete programs.

A single board computing system was designed using the chipset from Intersil and a 80C32 based microcomputer system. The implemented computer is a minimal PDP8 computer with an UART to interface with legacy peripherals. The 80C32 microcomputer system controls the memory and the system state of the PDP8 sub-system. This microcomputer system communicates with a host computer via an USB interface. The USB connection is based upon the PDIUSB12 USB controller from Philips. Using the 80C32 microcomputer system and specially designed host software, executables can be loaded into the PDP8 memory and their execution can be controlled. The whole PDP8 computer is controllable from a special application running on a Microsoft Windows host computer. Borland C++ Builder was used to develop this application.

This thesis will describe the development process and results of building such a system. The design of the PDP8-computer and the PC host sofware will be presented.

Laboratory : Computer Engineering
Codenumber : CE-MS-2004-04

Committee Members :

Advisor:	Georgi Gaydadjiev, CE, TU Delft
Advisor:	W.L. van der Poel, SE, TU Delft
Chairperson:	Stephan Wong, CE, TU Delft
Member:	Arjan van Genderen, CE, TU Delft

To my parents for their endless love and support.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I am deeply grateful to my advisor Ir. Georgi Gaydadjiev who was so helpful, patient and enthusiastic. His patience, experience and encouragement were valuable sources that have helped me when I got a bit stuck. He really was a great help.

Next, I want to thank prof.dr.ir. W.L. van der Poel for his encouraging and kind help. He provided me with information about the PDP8 machine, where he had worked with in the seventies. I will remember his enthusiasm for a long time.

At last, I want to thank Prof. Stamatis Vassiliadis for his encouraging and kind help during my study in the Computer Engineering group. I also want to thank all the other members in the Computer Engineering group, which I had a great time with.

Johan van de Pol
Delft, The Netherlands
July 5, 2004

1 Introduction

This section will give an introduction to the thesis. Section 1.1 will explain the goals of the project and this thesis. Next, in Section 1.2, a short historical overview of the PDP8 processor is presented. Section 1.3 will present the design trajectory of the entire project. At last, the framework of this thesis is given in Section 1.4.

1.1 The PDP8 project

The PDP8 family of microcomputers were built by Digital Equipment Corporation. These computers have a historical value, because they were the first computers that were affordable by a broader range of customers. A small historical overview of this machine is given in Section 1.2. The PDP8 processors were quite special because of their small instruction set: only eight basic instructions were supported. These eight basic instructions are sufficient to provide enough functionality to compose complete applications. Due to its historical value and the fact that many PDP8 computers are no longer operational, a PDP8 computer that can be controlled from a host computer is designed and created. In particular, the host computer can control and manipulate the memory and the state of the PDP8 computer. This thesis describes a part of the design of this system.

1.2 Historical overview

In 1957, Ken Olson and Harlan Anderson founded Digital Equipment Corporation (DEC), capitalized at $100.000, and 70 percent owned by American Research and Development Corporation. They wanted to call their company Digital Computer Corporation, but the venture capitalists insisted that they avoid the term Computer and hold off on building computers, because of the stereotype that computers were big and expensive, needing a computer center and a large staff.

With facilities in an old woolen mill in Maynard Massachusetts, DEC's first product was a line of transistorized digital "systems modules"; these were plug-in circuit boards with a few logic gates per board. Starting in 1960, DEC finally began to sell computers. Soon after this, there were enough users that DECUS, the Digital Equipment Computer User's Society was founded.

DEC's first computer, the PDP1 had a price label of only $120.000 at a time when other computers sold for over $1.000.000. In addition, DEC quoted prices as low as $85.000 for minimal models. The term computer was also avoided by using the term Programmable Data Processor or PDP for their products. For over a decade, all digital computers sold by DEC were called PDPs.

Table 1.1: Overview of the PDPs

Model	Date	Price	Bits	Number	Comments
PDP1	1960	$120.000	18	50	DEC's first computer
PDP2			24		Never built? Prototype only?
PDP3			36		One built by a customer, not by DEC.
PDP4	1962	$60.000	18	45	Predecessor of the PDP7.
PDP5	1963	$27.000	12	1.000	The ancestor of the PDP8.
PDP6	1964	$300.000	36	23	A big computer.
PDP7	1965	$72.000	18	120	Widely used for real-time control.
PDP8	1965	$18.500	12	≈50.000	The smallest and least expensive PDP.
PDP9	1966	$35.000	18	445	An upgrade of the PDP7.
PDP10	1967	$110.000	36	≈700	A PDP6 followup, great for time-sharing.
PDP11	1970	$10.800	16	>600.000	DEC's first and only 16 bit computer.
PDP12	1969	$27.900	12	725	A PDP8 relative.
PDP13					There was no such machine.
PDP14					A ROM-based programmable controller.
PDP15	1970	$16.500	18	790	A TTL upgrade of the PDP9.
PDP16	1972		8/16		A register-transfer module system.

In the early 1960's, DEC was the only manufacturer of large computers without a leasing plan. IBM, Burroughs, CDC and other computer manufacturers leased most of their machines, and many machines were never offered for outright sale. DEC's cash sales approach led to the growth of third party computer leasing companies such as DELOS.

DEC built a number of different computers under the PDP label, with a huge range of price and performance. The largest of these are fully worthy of large computer centers with big support staffs. Some early DEC computers were not really built by DEC. With the PDP3 and LINC, for example, customers built the machines using DEC parts and facilities. Table 1.1 lists the PDP computers. Each PDP is listed with its introduction date, price at introduction, width of databus, number of sold processors and additional comments [9].

The PDP-8 family of minicomputers were built by Digital Equipment Corporation between 1965 and 1990. The PDP-8 was largely upward compatible with the PDP-5, a machine that was unveiled in 1963. All of these machines were characterized by a 12 bit word, typically 4K words of memory, and simple but powerful instruction sets.

By late 1973, the PDP-8 family was the best selling computer in the world, and it is likely that it was only displaced from this honor by the Apple II (which was displaced by the IBM PC). Most models of the PDP-8 set new records as the least expensive computer on the market at the time of their introduction. The PDP-8 has been described as the model-T of the computer industry because it was the first computer to be mass produced

at a cost that just about anyone could afford.

In 1976 Intersil, offered the first PDP8 processor to occupy a single chip using CMOS technology: the IM6100. DEC verified that it was a PDP8 and began to apply it to a product in the fall of 1976 [6].

1.3 Design trajectory

In order to obtain the targeted result, a design trajectory has to be followed. This trajectory is showed in Figure 1.1. The design trajectory of the PDP8 device and software consists of the following steps:

1. A specification for the design of the PDP8 device and the PC host software has to be defined. In order to acquire the specification, a thorough investigation of the requirements and the design space is needed.
2. After the specification is defined, the hardware is designed. The hardware design consists of the following sub-steps:
   - Design a functional representation of the system, i.e., develop a schematic design according to the specifications.
   - Design a physical representation of the system by converting the schematic design into a Printed Circuit Board (PCB) design.
3. The software for both the PDP8 device and the PC host is developed simultaniously, because both software applications communicate with each other. Therefore, one application can't be tested without the other.
4. Verification of the entire design with the design specifications.

If errors are found in previous steps, the design result of that step will be corrected afterwards.

1.4 Thesis framework

The purpose of this thesis is to present an overview of the research and development work, concerning the PDP8 project, done at the Computer Engineering Laboratory of the department of Electrical Engineering. This work is performed as part of a graduation project in the Computer Engineering Master of Science program at Delft University of Technology. The thesis organization is as follows:

- Chapter 2 discusses the specification of the PDP8 device and the PC host software.
- Chapter 3 presents the schematic design implementation of the PDP8 computer.
- Chapter 4 describes the design and implementation of the PC host software.
- Chapter 5 concludes this thesis and describes the main contributions.

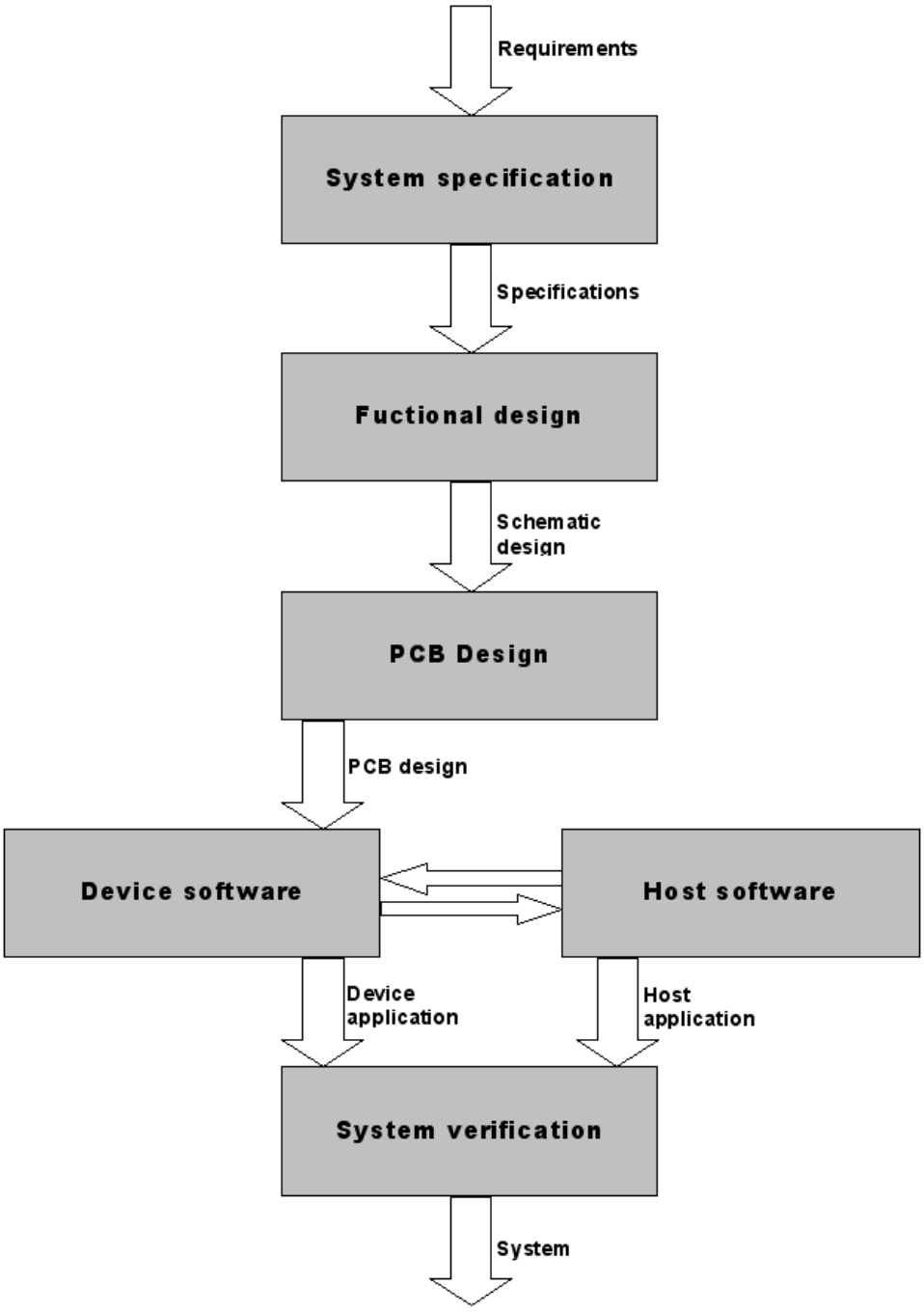


Figure 1.1: Design trajectory

2 Specification

Before starting with producing a system (hardware/software), a clear specification of the system is needed. This specification is needed to be able to check the designs at the end of the whole design process. The next chapter describes the specification of the PDP8 device and the host application.

Section 2.1 describes the specification and the requirements of the PDP8 device and in Section 2.2, the specification and requirements of the host software is given.

2.1 PDP8 device

This section describes the specification of the PDP8 computer device.

2.1.1 Requirements

The PDP8 system must be a complete basic system of a PDP8 machine as it was used in earlier times. This means that the system has to contain at least the following components:

- PDP8 Processor
- 4k words memory space.
- I/O-extension
- UART

The switch register that these machines contained needs also to be implemented and the PDP8 computer should function as it did in the past to have compatibility.

This basic PDP8 machine must be controllable from a host computer. This means that from a program on the host computer, the PDP8 processor must be controllable and a few actions can be performed:

- Start the PDP8 processor
- Stop the PDP8 processor
- Step the PDP8 processor
- Reset the PDP8 processor
- Write a value in the switch register

- Read the content of the switch register

The last requirements have to deal with the RAM of the PDP8. The software on the host computer must have full control over the RAM of the PDP8. This means that the RAM must be shared and the content must be transmitted to the host computer when requested. Data can be received by the device when send from the host computer and stored in the RAM of the PDP8.

2.1.2 Design choices

To let the program communicate with the PDP8 device, a connection type must be chosen. Al lot of possibilities exist for this communication. A few possibilities are listed below:

- PCI card in the computer
- Serial RS232 connection
- Parallel connenction
- USB connection
- FireWire connection
- Wireless connection

An extern device is preferred, because such a device is easier to port from one computer to another computer. Also a connection type that is widely available is preferred. Almost all computers nowadays have a serial, parallel or USB interface and because a wireless connection isn't necessary, only these three connection types are left. The serial interface is becoming more and more old fashioned and disappears from modern computers, especially from laptops. Also the parallel interface is old fashioned which means that only the USB interface is left. This is the reason that for the connection with the host computer, USB is chosen. USB has also the nice property that the device is hot-plugable.

2.2 Host software

This section describes the specification of the host software of the PDP8 device.

2.2.1 Requirements

The software for the host must provide an inteface for controlling the PDP8 processor on the USB device. The host software has to be able to perform a number of tasks. First tasks related to control the PDP8 computer are listed. The host software must be able to:

- start the PDP8 processor.
- stop the PDP8 processor.

- step the PDP8 processor.
- reset the PDP8 processor.
- set the switch register of the PDP8 computer.
- read the value of the switch register and display this value.
- read status information of the PDP8 processor.

Besides the controls to operate the PDP computer, also controls are needed to manipulate the RAM of the PDP8 computer. The host software must be able to:

- write data to a specified location of the PDP8 RAM.
- read data from a specified location of the PDP8 RAM. The read data must be displayed in an appropiate way.
- store data from the pdp8 RAM in a binary file. This file must be in a common format.
- load a binary file in a common format into the RAM of the PDP8 processor.

Furthermore, assembly files written in the PAL format need to be edited and compiled using an extern compiler. When facilities for editing these files are provided by the host program, it would make it easier to work with the program, because less other applications are needed to operate the PDP8 computer. This means that a few more tasks must be handled by the host software. The host software must:

- provide an editor to edit PAL assembly files.
- be able to load a PAL file into the editor.
- be able to save the editor text into a pal file.
- be able to compile the text in the editor and show the error messages of the compiler.
- be able to write the binary output of the compiler to the RAM of the PDP8 processor.

The user interface of the host program should be simple and easy to use. Examining the interface should be enough to understand the working and the possibilities of the program without the need for any documentation. Things like connecting to the device should be done without bothering the user if possible. The user should only be concerned with its task, operating the PDP8 computer, and not with the interface.

2.2.2 Design choices

There are just a few design choices to make that are related to the host program. The choices that have to be made are:

- What kind of interface?
- What programming language?
- Which compiler?

In the remaining part of this section these questions are answered and argumented.

What kind of interface?
The interface may be a command line or a graphical interface. Ofcourse, the graphical interface is much more attractive. A command line interface will never be as easy as a graphical interface might be, because the user must be familiar with the specific commands and their semantics. This means that a command line interface will never meet the requirements.

There exist many types of graphical interfaces. An interface can be build with standard buttons and widgets or with custom build widgets. Using standard widgets gives the program a standard look while custom widgets give the program a more exclusive look. An interface with standard widgets will look more familiar and raises less questions by the user. This is the reason why the interface of the program will be a graphical interface with standard widgets.

What programming language?
What programming language is chosen is an implementation issue and does not really matter for the result of the program functionality. Therefore, C++ is used, because this is the programming language the author is most familiar with.

Which compiler?
The compiler should be a visual C++ compiler. This means that there are in fact only two compilers that can be used: *Microsoft Visual Studio* [5] and *Borland C++ Builder* [3]. Both programs can create user interfaces that look almost the same and therefore it does not matter which one is chosen. Borland C++ Builder was chosen, because of the author previous experiences with this compiler.

3 PDP8 device

The PDP8 device, most times simply called the device, is a device connected to a host computer using an USB interface. The device contains a complete PDP8 computer that is controlled by a second microcomputer system. This chapter describes the PDP8 computer and its implementation.

Section 3.1 will give an overview of the device. Next, in Section 3.2 the PDP8 computer is explained and the implementation is presented. The microcomputer system that has control over the PDP8 computer needs access to the RAM of the PDP8. This means that some kind of multiplexer needs to be implemented. The implementation of this multiplexer is given in Section 3.3. At last in Section 3.4 the Printed Circuit Board (PCB) is presented.

3.1 PDP8 device

The main part of the external device is the PDP8 computer. To provide control over this computer, the device is connected to a host computer using an USB interface as required by the specifications. A microcomputer system, using a 80C32 processor, takes care of the interface between the host and the PDP8 hardware. The RAM of the PDP8 must also be available for the 80C32 processor, but the 80C32 and the PDP8 processor both run independent of each other. They don't share a data and address bus, which means that some kind of multiplexer for switching the RAM is needed. An overview of the hardware as planned is given in Figure 3.1.

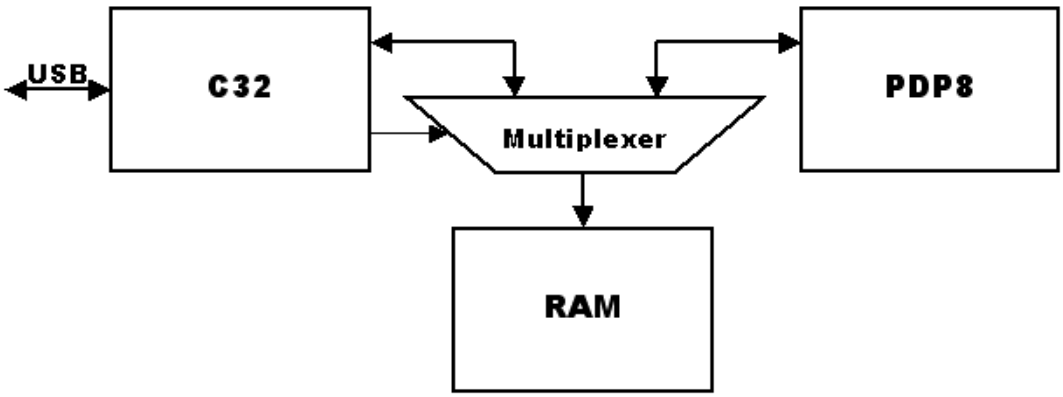


Figure 3.1: Overview of the hardware

The design of the device is split in three parts that can be distinquished in Figure 3.1:

- C32 computer
- PDP8 computer
- Shared RAM with multiplexer

Splitting the design is necessary to keep a good overview of the design. In the next two sections the second and third item will be discussed and the functional design will be presented. The design of the subsystem with the C32 processor will not be discussed in this document.

3.2 PDP8 computer

This section will describe the PDP8 computer. First the processor is described and then the memory, I/O extension and the switch register are presented.

3.2.1 PDP8 processor

The bits in the PDP8 words are counted from left to right. The most significant bit (the left one) is counted as 0 and the least significant bit is counted as 11. This is just the other way around as in most cases where the most significant bit gets the highest count. The result of this difference is that both address and data busses must be cross-connected when linked to the same component.

In the next subsections the processor is described. First the internal structure and the instruction set and at last the wiring of the processor are presented.

3.2.1.1 Internal structure

Figure 3.2 shows the internal structure of the PDP8 processor. The processor has six 12-bit registers, a programmable logic array, an arithmetic and logic unit. Below the internal components are described [8, 4].

Accumulator(AC)
The accumulator is a 12-bit register. Data may be fetched from the memory to the accumulator or stored from the accumulator into the memory. Arithmetic and logical operations involve two operands, one from the accumulator and one from the memory. The result of the operation is stored in the accumulator. The accumulator may be cleared, complemented, tested, incremented or rotated. The accumulator also serves as an input/output register for all programmed data transfers.

Link(L)
The link is a 1-bit register that serves as the high-order extension of the accumulator. A carry out of the ALU complements the content of the link. The link can be cleared,

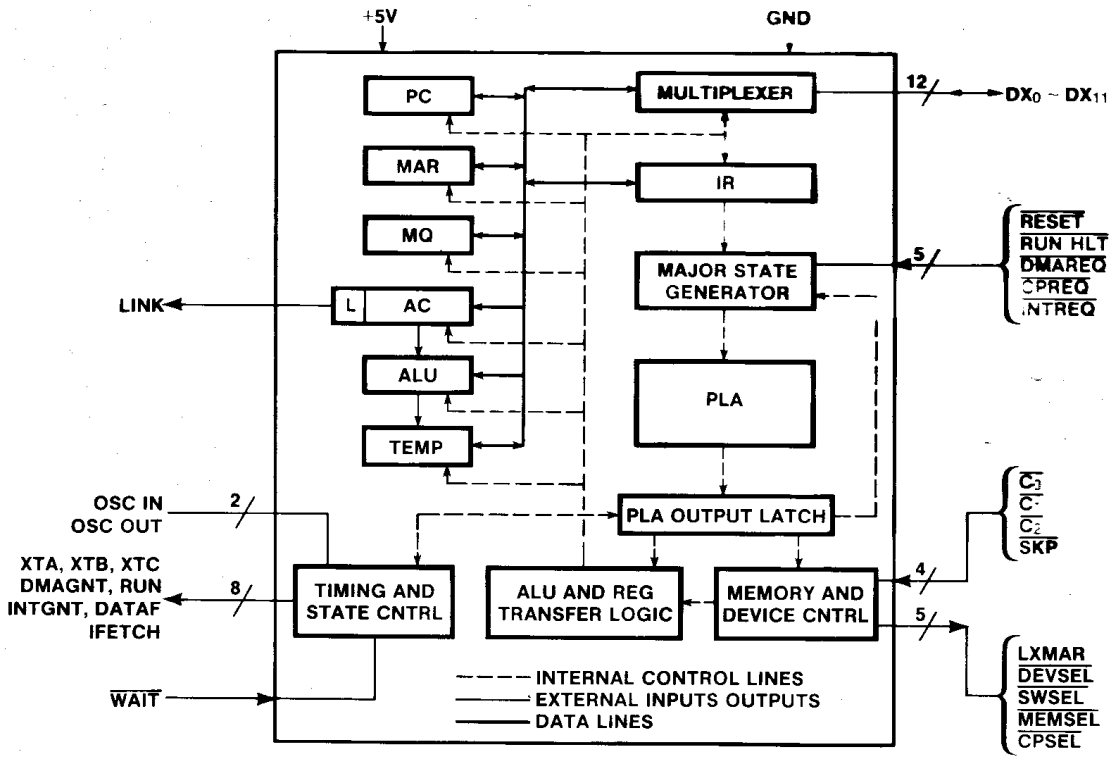


Figure 3.2: The internal structure of the PDP8 processor

set, complemented and tested. It can also be rotated in a chain with the accumulator.

MQ register(MQ)
The MQ register is a 12-bit temporary register which is accessible under program control. The contents of the accumulator may be tranfered to the MQ register. The content of the MQ register can be OR'ed with the accumulator and the result of this operation will be placed in the accumulator. The contents of the accumulator and the MQ register may also be exchanged under program control.

Memory Address Register(MAR)
While the memory is accessed, the 12-bit register MAR contains the memory address that is currently being read or written. The MAR is also used as an internal register for microprogram control during data transfers to and from the memory and peripherals.

Program Couter(PC)
The program counter contains the 12-bit address of the memory location from which the next instruction is fetched. During an instruction fetch, the PC is loaded into the MAR and the PC is incremented by 1. In case of a brach to another memory location, this new address is loaded in the PC. Normally a branch takes place under program control, but a peripheral may also cause the next instruction to be skipped by incrementing the

PC by 1. This skip instruction may be conditional on the state of the accumulator and the link.

Arithmetic and logical unit(ALU)
The ALU performs both arithmetic and logic operations: 2's complement binary addition, logic AND, logic OR and inversion (complement). The ALU can perform a single bit shift in both directions. The instruction set also contains a seperate instruction for a double bit shift, which in fact is implemented as two single bit shifts. The accumulator is always one of the inputs of the ALU, but it may be replaced by the constants all zeros or all ones under program control. The second input may be one of the other registers under internal microprogram control.

Instruction register(IR)
After an instruction fetch, the instruction register contains the instruction that is going to be executed next by the CPU. The instruction register specifies the initial step of the microprogram sequence for each instruction and is also used as an internal register to store temporary data for microprogram control.

Multiplexer(DX)
The 12-bit input/output multiplexer handles data, address and instruction transfers into and out of the CPU, from or into the main memory and peripheral devices on a time-multiplexed basis.
Major state generator and the programmed logic array (PLA)
During an instruction fetch, the instruction to be executed is loaded in the instruction register. The PLA is then used for the correct sequencing of the CPU for the appropriate instruction. After a instruction is completely executed, the major state generator scans the internal priority network and the state of the network decides whether the next instruction is fetched or one of the external request lines need to be serviced.

PLA output latch
The PLA output latch holds the the PLA output, permitting to be pipelined. It fetches the next control sequence while the CPU is still executing the current control sequence.

Memory and device control, ALU and register transfer logic
The memory and device control unit provides external control signals to communicate with peripheral devices (*DEVSEL*), switch register (*SWSEL*), memory (*MEMSEL*) and control panel memory (*CPSEL*). During I/O-instructions, this unit also modifies the PLA outputs depending on the states of the four device control lines (*Skip*, *C0*, *C1* and *C2*). The ALU and register transfer logic provide the control signals for the internal register transfers and ALU operation.

Timing and state control
All timing and state signals are generated internally in the CPU. An external crystal provides the clock frequency of the processor. The CPU divides this frequency by 2 and the internal state control uses this frequency.

3.2.1.2 Instruction set

Instruction words are organized as follows [9, 4]:

00	01	02	03	04	05	06	07	08	09	10	11
opcode			i	z	address						

opcode = the opcode of the instruction (3 bits)
i = the indirect bit (0 = direct, 1 = indirect)
z = the page bit (0 = page zero, 1 = current page)
address = the word in page (7 bits)

The 5 most significant of the 12 bit program counter give the current page and memory addressing is also complicated by the fact that absolute memory locations 8 through 15 are incremented prior to use when used as indirect addresses. These locations are referred as auto-index registers despite the fact that they are in the main memory. These auto-index registers provide a way for very tightly programmed array operations.

The PDP8 instruction set contains 8 basic instructions. These instructions are:

- 000 - AND - logical AND of the operand with the AC
- 001 - TAD - arithmetic addition of the operand and <L,AC> (a 13 bit value)
- 010 - ISZ - increment operand and skip if result is zero
- 011 - DCA - deposit AC in memory and clear AC
- 100 - JMS - jump to subroutine
- 101 - JMP - absolute jump
- 110 - IOT - input/output transfer
- 111 - OPR - microcoded operations.

The ISZ and other skip instructions conditionally skip the next instruction. The ISZ is commonly used to increment a loop counter and skip if done, and it is also used as an general increment instruction, either followed by a NOP or in contexts where it is known that the result will never be zero.

The JMS instruction stores the return address in relative word zero of the subroutine, with execution starting with relative word one. The return of the subroutine is done with an indirect JMP through the return address. Subroutines commonly increment their return addresses to index through inline parameter lists or to perform conditional skips over instructions following the call.

The IOT instruction has the following form:

00	01	02	03	04	05	06	07	08	09	10	11
1	1	0	device						operand		

The IOT instruction specifies one of up to 8 operations on one of 64 devices.

Typically (but not universally), each bit of the operand field invokes an operation. IOT instructions may be used to initiate data break transfers from block devices such as disk or tape. The term "data break" was DEC's preferred term for cycle-stealing direct-memory-access data transfers. Some CPU functions are accessed only by IOT instructions. For example, interrupt enable and disable are IOT instructions.

An interrupt is requested when any device raised its flag. The processor resets all flags and disables interrupts. In effect, an interrupt is a JMS instruction to location zero, with the side effect of disabling interrupts. The interrupt service routine is expected to test the device flags and perform the operations needed to reset them, and then return using ION immediately before the indirect return JMP. The effect of ION is delayed so that interrupts are not enabled until after the JMP instruction.

A wide variety of operations are available through the OPR microcoded instructions. These are divided in groups. Group 1:

```
11101xxxxxxx   - CLA - clear accumulator
1110x1xxxxxx   - CLL - clear the link bit
1110xx1xxxxx   - CMA - ones complement accumulator
1110xxx1xxxx   - CML - complement link bit
1110xxxxxxx1   - IAC - increment <L,AC>
1110xxxx100x   - RAR - rotate <L,AC> right
1110xxxx010x   - RAL - rotate <L,AC> left
1110xxxx101x   - RTR - rotate <L,AC> right twice
1110xxxx011x   - RTL - rotate <L,AC> left twice
```

In general, the above operations can be combined by OR'ing the bit patterns for the desired operations into a single instruction. If none of the bits are set, the result is the NOP instruction. When these operations are combined, they operate top to bottom in the order shown above. The exception to this is that IAC cannot be combined with the rotate operations on some models. Attempts to combine rotate operations with IAC have different effects from one model PDP8 to another. For example, on the PDP-8/E, the rotate code 001 means "swap 6 bit bytes in the accumulator". Previous models of the PDP8 took this to mean something like "shift neither left nor right 2 bits", which is the same as a NOP. Therefore, this combination is not allowed.

Below the second group of microcoded instructions is shown:

```
1111x1xx0xx0   - SMA - skip on accumulator < 0
1111xx1x0xx0   - SZA - skip on accumulator = 0
1111xxx10xx0   - SNL - skip on link ≠ 0
1111x0001xx0   - SKP - skip unconditionally
1111x1xx1xx0   - SPA - skip on accumulator ≥ 0
1111xx1x1xx0   - SNA - skip on accumulator ≠ 0
1111xxx11xx0   - SZL - skip on link = 0
11111xxxxxxx   - CLA - clear accumulator
1111xxxxx1x0   - OSR - or switches with accumulator
1111xxxxxx10   - HLT - halt the processor
```

The above operations may be combined by OR'ing them together, except that there

are two distinct incompatible groups of skip instructions. When combined, SMA, SZA and SNL, skip if one or the other of the indicated conditions are true, while SPA, SNA and SZL skip if all of the indicated conditions are true. When combined, these operate top to bottom in the order shown; thus, the accumulator may be tested and then cleared. Setting the halt bit in a skip instruction is a useful way to set a breakpoint for debugging. If none of the bits are set, the result is an alternative form of a NOP instruction.

A third group of operate microinstructions (with a '1' in the least significant bit) deals with the optional extended arithmetic element (EAE) to allow such things as hardware multiplication and division, 24 bit shift operations, and normalization. These operations involve an additional data register, MQ or multiplier quotient, and a small step count register. On the PDP-8/E and successors, MQ and the instructions for loading and storing it were always present, even when the EAE was absent.

3.2.1.3 Wiring the processor

The available PDP8 processor is the IM6100 developed by Intersil. This processor comes in a 40-pin DIP package. In Appendix A, the pinout of this chip is given together with an explanation of its pins [8]. Wiring the processor for the PDP8 computer is done by the following:

- Direct Memory Acces and the Control Panel are not used. Also the *WAIT* input is not needed, so these input pins are tied high.
- The *Run* output indicates the state of the internal run/halt flipflop. This pin controls a LED as an indication of this state. The pin is also given as an input to the C32 microcomputer. The *RUN* output is buffered by a AND-port. The *RUN/HALT* input of the PDP8 is provided by an output of the C32 processor.
- The crystal of the PDP8 prcessor can have any clock value between 0 and 5 Mhz. 4 Mhz is the typical value, that's why a 4 Mhz crystal is used.
- The four device control lines (*Skip*, *C0*, *C1* and *C2*) are all open collector inputs. Therefore they need a pull-up resistor. *C0* is not used anymore, so this pin can be tied high directly at the Vcc.
- The *RESET* is, just like the *RUN/HALT* input, tied to an outut of the C32 computer. The reset signal is inverted, because the *RESET* of the UART is tied to the same signal, but this signal is active high and the PDP8's reset is active low. The pull-up resistor is used to ensure that on power-up, the reset stays active untill the C32 clears the reset.

The schematic with the wiring of the PDP8 processor is shown in Figure 3.3.

3.2.2 Memory

This section describes the memory of the PDP8 computer. First the memory organization is given, second the timing of the memory interface. The actual wiring of the memory is not given, because it is changed by the RAM multiplexer that is described later in Section 3.3.

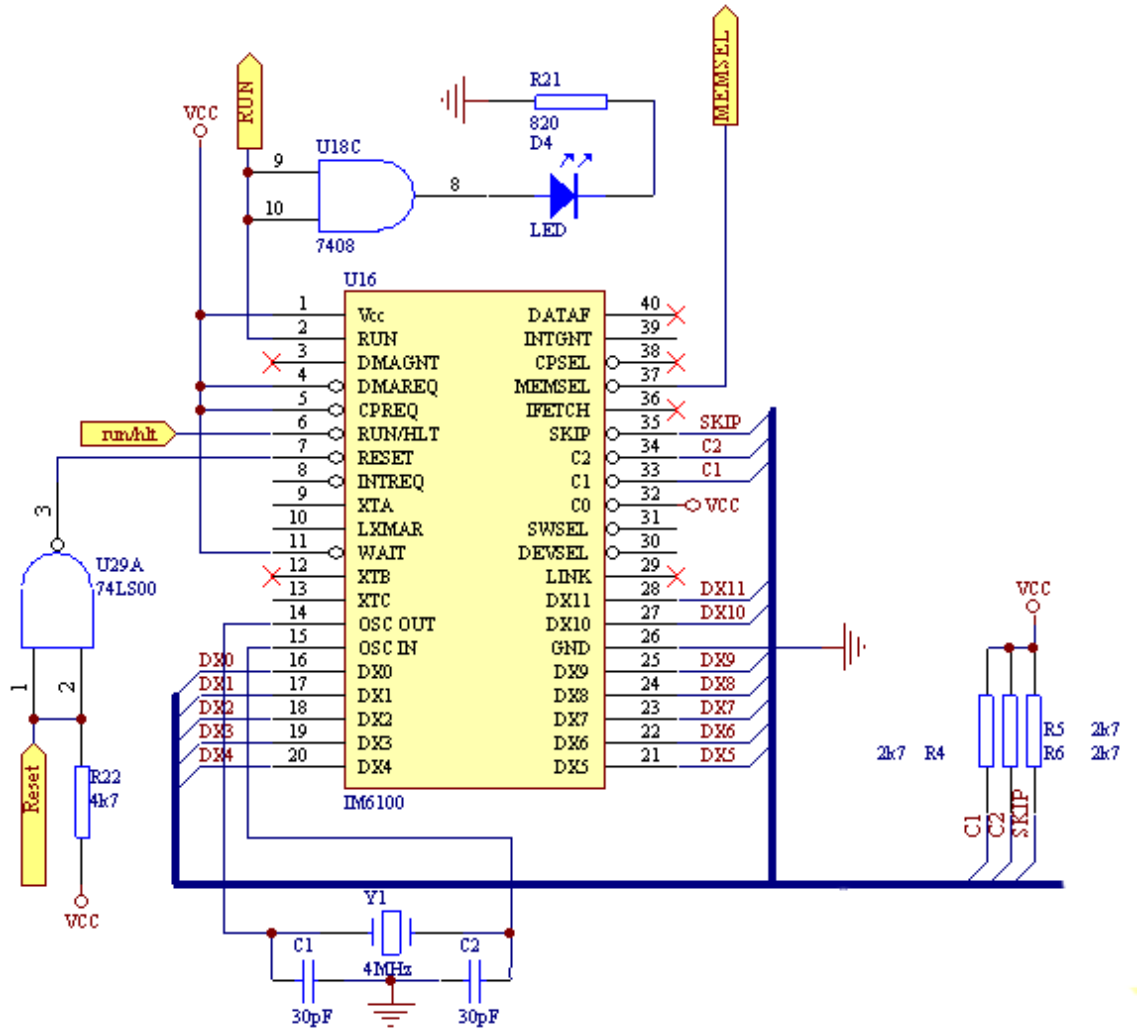


Figure 3.3: The wiring of the PDP8 processor

3.2.2.1 Memory organization

The memory of the PDP8 is divided into pages. The total memory space is divided into fields of 4k words. A maximum of 8 fields can be used if Extended Memory Control hardware is installed. Each field is divided into 32 pages of 128 words. Data contained in page 0 and the current page of the program counter can be accessed directly. All other locations must be accessed indirectly by addressing a memory location that can be directly addressed. This second memory location must contain the address of the memory location needs to be accessed or jumped to. A graphical representation of the memory organization is shown in Figure 3.4.

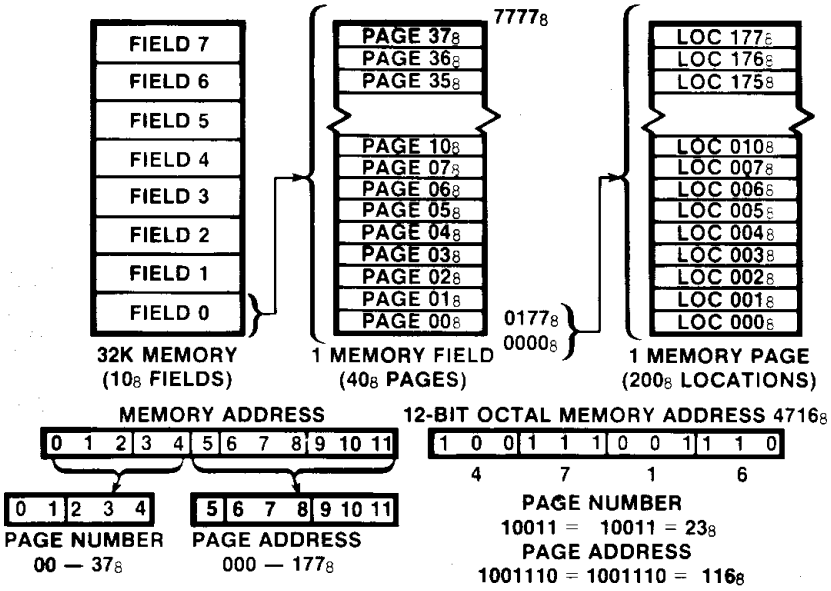


Figure 3.4: The memory organization of the PDP8

Table 3.1: Timing signals required by the RAM and provided by the PDP8

Required by RAM	Provided by PDP8
Chip enable	*MEMSEL*
Output enable	$\overline{XTA}$
Write enable	*XTC*

3.2.2.2 Memory timing signals

The SRAM chips have 3 timing input signals: chip enable, output enable and write enable. The required timing signals for a read and write operation on the RAM is shown in Figure 3.5. The address is latched (using 74HC573 latches) on the negative edge of *LXMAR* and from that moment on, the address is valid on the address bus. *SWSEL* provides the correct timing signal for the chip enable and *XTA* inverted the output enable as can be seen in Figure 3.5. During the read operations, the write enable must stay high. The read operation is controlled by the output enable signal. However, a write operation is, as a result of the timing signals of the PDP8, controlled by the chip select. The timing diagram for a write operation on the RAM is also shown in Figure 3.5. The write enable signal of the PDP8 is *XTC*. This signal is high during a read operation and low during a write operation. In Table 3.1, the timing signals are summarized.

3.2.3 I/O extension

The IM6100 processor provides only a poor I/O interface. Therefore an extension (IM6101), the Parallel Interface Element (PIE, is added to the design. This extension provides a more convenient interface for peripherals. In fact, the extension is often

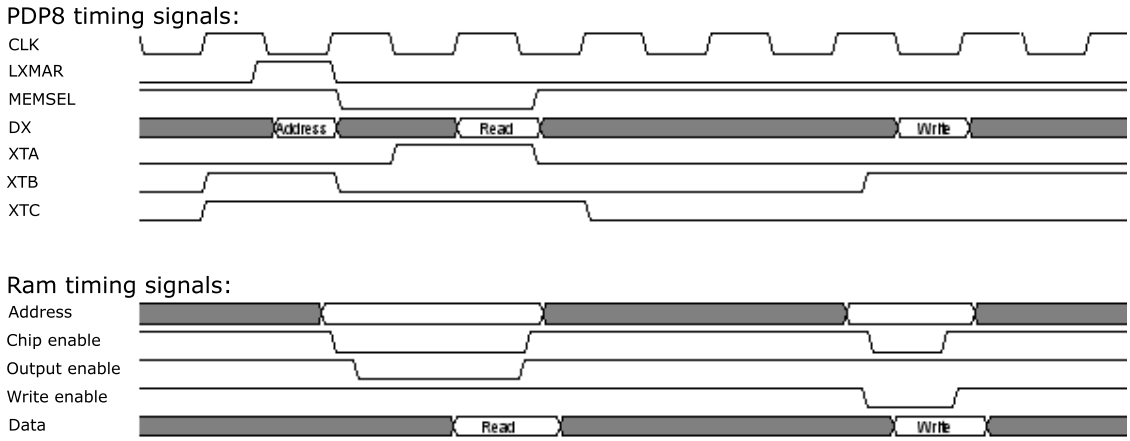


Figure 3.5: Timing of the RAM and the PDP8

seen as a part that is missing in the processor. In the next section the PIE is presented and in the following section one of these peripherals, an UART, is described.

3.2.3.1 PIE (IM6101)

Each I/O instructions contains a six bit device selection code, but only five bits are used as a selection code by the PIE. If the first 5 bits of the selection code match with the selection code of the PIE, then the PIE is selected. The last bit of this selection code is used together with the operand to specify the operation that is to be performed. Data transfers between the data bus of the PDP8 and peripherals are controlled by the PIE via 2 read, 2 write, 4 sense and 4 flag functions. The PIE contains 2 registers, A and B, that program write polarities, sense polarities, sense levels or edges, flag values and interrupt enables.

The PIE comes in a 40-pin DIP package and the pinout of this chip is given in Appendix B. The way the IM6101 connects to the IM6100 is quite simple because pins share the same name. The selection code can be selected using a few dip-switches with pull-up resistors. For the conection is no extra logic needed. Figure 3.6 shows how the PIE is connected to the processor, the switches for the selection code are not shown. This figure also shows the connection with the UART which is described in the next section.

3.2.3.2 UART (IM6402)

The IM6402 is a general purpose UART for interfacing an asynchronous serial data channel to a parallel synchronous data channel. The UART is set up for connecting to a ASR-33 Teletype. A teletype was the standard I/O for PDP8 and other computers in the seventies. The data format consists of 11 bits: 1 start bit, 8 data bits and 2 stop bits. No parity bits are used by the teletype.

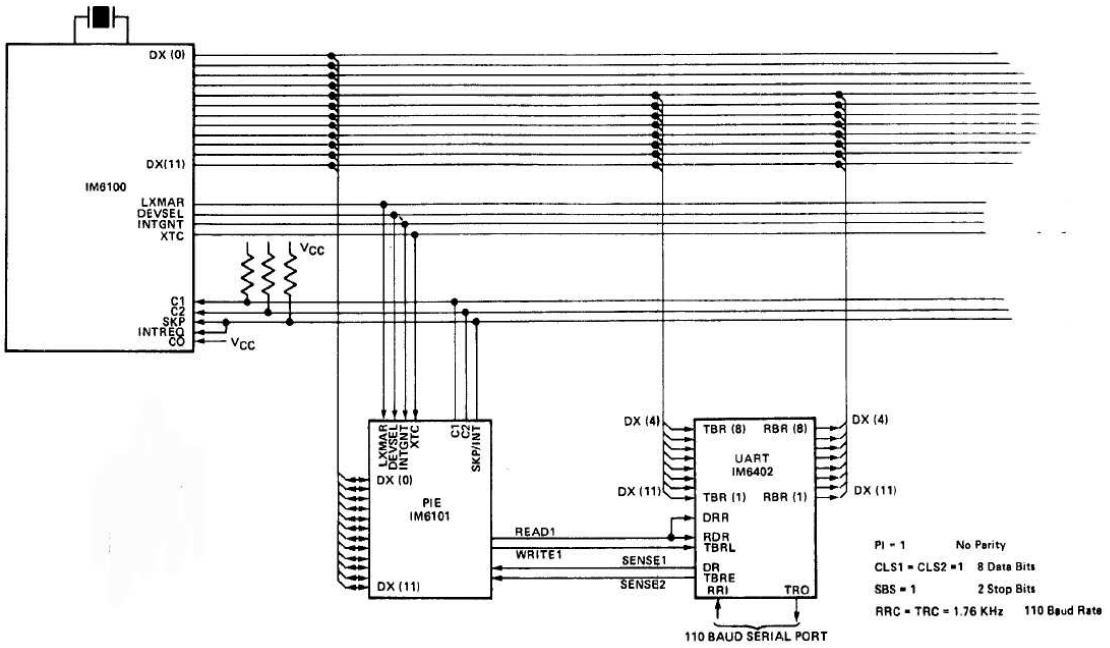


Figure 3.6: The design of the PDP8 computer

The parallel input and output of the UART is connected to the low order 8 bits of the data bus of the PDP8. The UART is connected to the PIE by 1 read, 1 write and 2 sense signals. The clock signal for the baud rate and the reset are received from the C32 sub-system. The connection to the PIE and the processor is shown in Figure 3.6. The serial output of the UART uses TTL level (0 and 5 volt), which means that a converter to RS232 (12 and -12 volt) is needed. The *MAX232* is the converter that is used. This converter needs only a 5 volt supply voltage to generate the output voltages. The RS232 signal output and input are connected to a DB25 serial connector; no other communication or hardware handshaking signals are provided by the UART.

3.2.4 Switch Register

The switch register is a register that is traditionally implemented by twelve switches. This register can be read by the execution of the OSR (Or accumulater with Switch Register) instruction on the PDP8. This instruction ORs the content of the accumulator with the content of the switch register and deposits the result in the accumulator. When the OSR instruction is executed a single signal from the PDP8 processor, *SWSEL*, will instruct the switch register to deposit its contents on the bus of the PDP8 processor.

In our system it must be possible to set the switch register from the host computer to a specific value. Therefore, a electronic switch is needed instead of the physical switches that were used on the original machines. The data for the switch register is send by the host to the device and received by the C32 computer. The C32 will then write the received value into the switch register. To reduce the I/O-pin usage of the C32 processor, the switch register is implemented as a shift register. The C32 computer will shift the

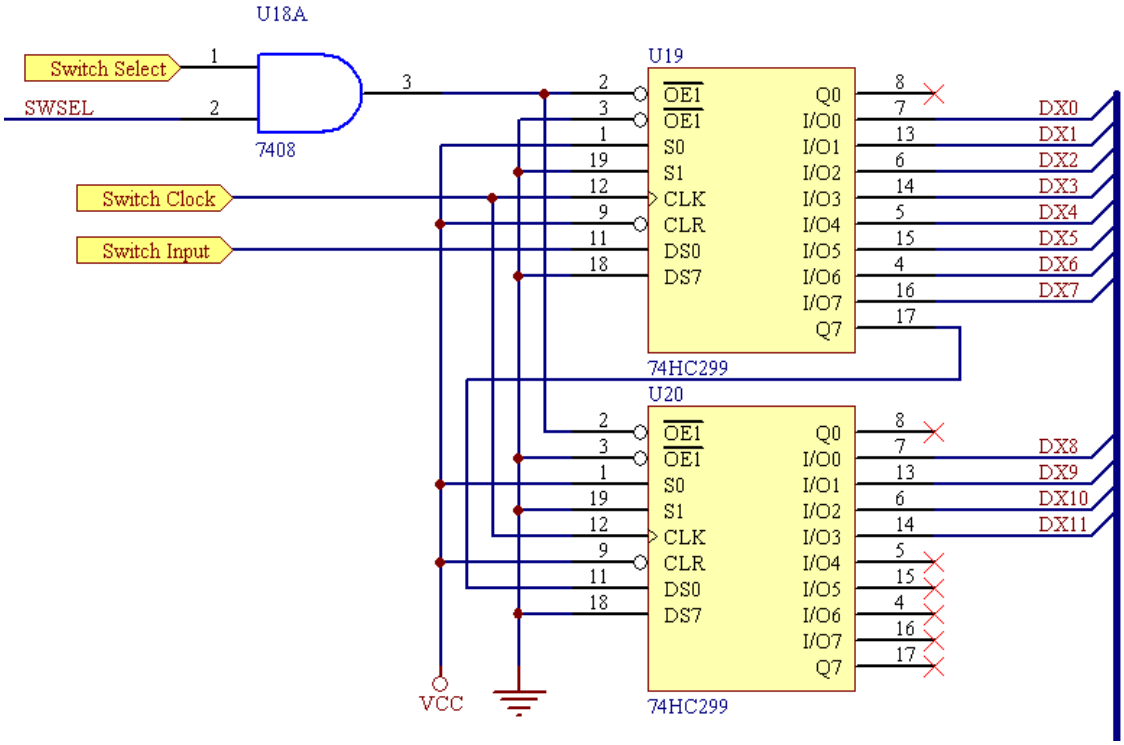


Figure 3.7: The design of the switch register

received value serially into this register. The C32 processor needs only 3 I/O-pins to set the switch register:

- Serial data line (*Switch Input* in Figure 3.7)
- Clock line (*Switch Clock* in Figure 3.7), needed to clock the data serially in.
- Disable line (*Switch Select* in Figure 3.7), needed to disable the switch register during the loading of a new value in order to prevent race conditions.

Switch registers are available in different formats, but not with a size of 12-bit, so two 8-bit registers (type 74HC299) are used (4-bit shift registers come with 2 registers in a single chip and won't reduce the amount of redundant hardware). These registers need to be tri-stated ('0', '1', 'X'), because the output of the shift registers needs to float if the switch register isn't selected by the PDP8. The two shift registers are connected together and construct a 16-bit shift register. The design of the switch register is shown in figure 3.7. The two shift registers form a chain by connecting the serial output (*Q7*, pin 17) of the first register to the serial input (*DS0*, pin 11) of the second shift register. *SWSEL* is an output of the PDP8 processor, telling the switch register to put its contents on the address/data bus.

Table 3.2: Differences with respect to the RAM of the 80C32 and the PDP8

80C32	PDP8
8-bit data bus	12-bit data bus
16-bit address	12-bit address bus
Write/read signals are different signals that give a pulse on activation	Write/read signal is a single signal (high/low)

3.3 RAM multiplexer

The RAM on the board has to be available for both the C32 and PDP8 processor. For the C32, this RAM is just some RAM in the memory space; the C32 does not need this RAM for its normal operation. For the PDP8 however, this RAM is the main memory and it also replaces the ROM normally found in these computers. The PDP8 and the C32 both have to acces the RAM, but a few differences exist between these processors. These differences are summarized in Table 3.2.

The PDP8 is capable of adressing 4k words of memory, because the processor has only 12 address lines. For additional memory, a hardware extension is needed and this hardware is not available. The databus of the PDP8 is 12 bits wide and RAM chips with this format are not available. This means that several RAM chips must be used to build the RAM for the PDP8. Chips containing 4k words are not widely available, so two chips with 8k bytes are used. This gives a memory of 8k x 16 bit. The PDP8 can only address half of the memory and uses only 75 percent of each memory word. The C32 can only address memory of 1 byte width, and thus has to see the RAM as 16k x 8 bit or (if half of the memory is made unadressable) 8K x 8 bit. The part of the RAM that can't be addressed by the PDP8 can be used for other purposes, so the C32 will be able to address the entire RAM. In Figure 3.8, the address spaces of the C32 and the PDP8 are showed. X is the startaddress of the PDP8 RAM in the memory space of the C32. Also can be seen how the same RAM chips are seen by both processors.

Because both processors must have acces to the same RAM, some logic is needed for these chips. Only one processor can have acces to the RAM at the same time, which processor has control over the RAM is controlled by the C32 processor. The RAM switch that has to be designed must switch between the data bus, address bus and all timing and other signals that control the RAM. All these signals are summerized below:

- 13 address bus lines
- 8 data bus lines
- Chip enable of RAM 1
- Chip enable of RAM 2
- Output enable
- Write enable

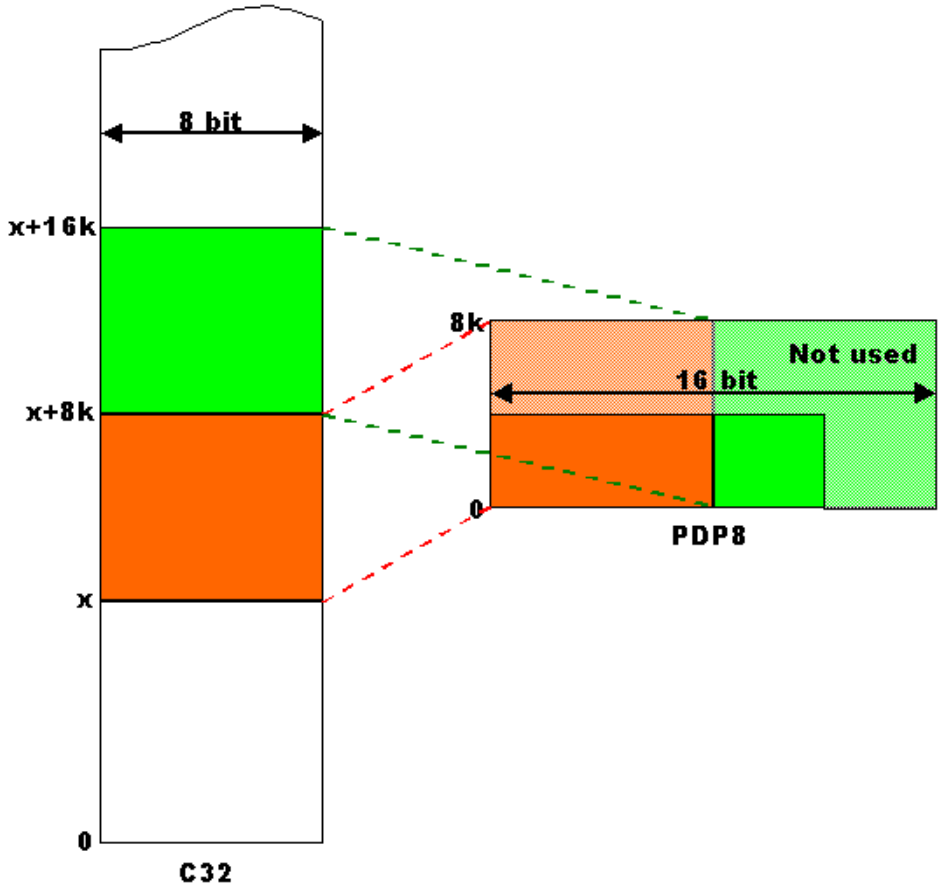


Figure 3.8: Address spaces and shared RAM of the PDP8 and C32 processor

Except for the data bus lines, all these signal are inputs of the RAM. The databus lines are both input and output. To select different inputs, a simple multiplexer (type 74HC157) is needed. For the datalines a simple multiplexer will not be sufficient, because data flows in both directions. Therefore some transmission gates (type 74HC245) are used for switching the datalines of the RAM. Two 8-bit transmission gates are used for cutting of the data bus of the C32. For cutting of the databus of the PDP8, a 12-bit transmission gate would be sufficient. Such 12-bit transmission gate is not available and therefore also two 8-bit transmission gates are used. Figure 3.9 shows how the transmission gates are placed. These transmission gates are tri-stated, which means that, when not enabled, the connected lines will be high-impedance and not be tight to a '0' or '1'. This is important, because otherwise the the databus is disturbed and wouldn't function.

The transmission gates have two control signals: chip enable and direction. The chip enable is determined by the switch and the chip enables of the two RAM chips. If the RAM is switched for the C32, the bottom two transmission gates can be enabled and

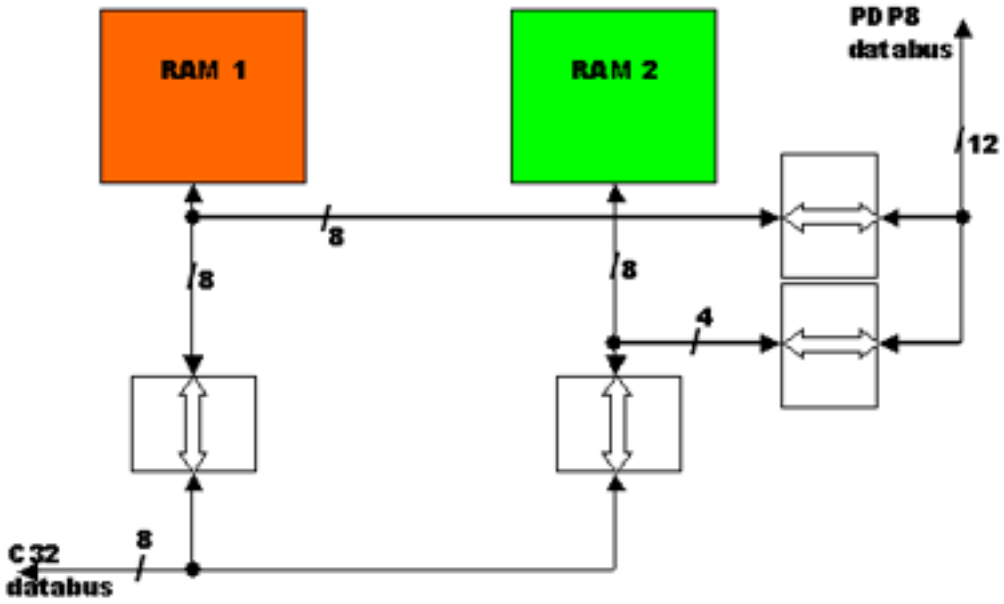


Figure 3.9: Placement of the transmission gates for switching the RAM

the other two are other two are disabled. When the RAM is switched to the PDP8, the situation is reversed and the two bottom transmission gates are disabled and the other two can be enabled. The transmission gates are only enabled when the RAM behind that transmission gate is selected and enabled. The direction of the transmission gates is determined by the direction of the data flow. Only if data flow goes from the RAM to the processor, the direction must be towards the processor. In all other situations the direction is to the RAM , so the data bus isn't disturbed by the output of the transmission gate. The read signals of both processors provide this direction of the transmission gates. If the two RAM chips are selected and enabled, the read signal can tell what the direction of the data flow is.

The C32 and PDP8 processor both have an own set of timing signals. The only difference between those sets is that the C32 computer provides two chip select signals for the two RAM chips and the PDP8 one, because the PDP8 will always select both RAM chips at the same time. Together with the read and write signal, this gives four signals per processor. These two sets of signals are fed into a multiplexer. One of these sets is selected by the switching signal, which is also used by the enabling and disabling of the transmission gates. Also the 12 address bits are fed into multiplexers and the switching signal selects one of the addresses.

3.4 Printed Circuit Board (PCB)

A PCB contains multiple layers stacked on each other with each layer containing a copper pattern. The function of the copper pattern is to provide electrical connections between the components mounted on the PCB. The schematic design is made in *Protel99 SE* from Altium [1]. The schematic design consists of all circuits described above and the C32 sub-system that is not described. Every component in this design gets a small decoupling capacitor (100 nF) between the power and the ground.

When the schematic design is completed and checked on concistency, all components are provided with a footprint and the PCB is generated. Next, the components are placed on the PCB in such a way that the distances between the components that have connection in between are as short as possible. After the placement of all components, the interconnections are routed automatically and these connections are checked on length. Especially the connection of clock lines are checked, because of the capacity of a connections which becomes larger for longer connections. The created PCB is checked on consistency with the schematic design and after succesful completion of this check, Gerber files are created. These Gerber files contain all information about all layers that is needed by the manufacturer of the PCB.

4 Host Software

The host software provides the user control over the PDP8 processor and its memory. Therefore the host software has to communicate with the software in the embedded system. This is realized by an USB connection. The USB protocol is incorporated into the operating system Microsoft Windows, which means that the host software uses of the USB functions included in the operating system. The Host software is written in C++ using Borland C++ Builder, because this package provides an easy way of creating graphical user interfaces using standard widgets.

Section 4.1 will describe how the USB protocol in incorperated in Windows. Next, in Section 4.2 I will describe how the software in the device communicates with the host software. In Section 4.3 the interface is presented and explained. At last, in Section 4.4 the organization of the host software is given.

4.1 USB Protocol and Windows

This section describes USB and the way it can be used as I/O in software for Microsoft Windows.

4.1.1 The Protocol

Initially the USB standard (USB 1.0 and 1.1) defined two speeds of transmission: low speed (1.5Mbps) and full speed (12 Mbps). In 2000, the final specification voor USB 2.0 was released [11]. This high speed (version 2.0) USB has a transmission speed of 480Mbps. For the PDP8 device we use full speed USB (version 1.1). The maximum amount of data that needs to be sent over the USB bus will be 4k words. This happens when the entire memory of the PDP8 is filled. A data package of 40 words (this is explained later) can be send every frame (1 ms). Sending 103 data packages (=4k words) takes about 100 ms. This means that full speed USB is fast enough.

USB makes use of endpoints for data communication. Data channels have at both sides an endpoint and several data channels can be defined for a single device, causing the device to have also several endpoints. Each endpoint can be configured to support one transmission type. The USB Protocol supports 4 types of transmissions. These transmission types with their properties are [10]:

- Isochronous transmission: Isochronous transmission data capacity is assured for 1 frame, but the quality of the data is not guaranteed. This type of transmission is used for applications that require realtime capacity (voice, movie camera, telephony, etc.).

- Interrupted transmission: Small volume data is transmitted at preset periodic times and the quality of the transmitted data is guaranteed. This type of transmission is used for applications where small volumes of data are transmitted regularly (keyboard, mouse, joystick, etc.).
- Bulk transmission: Bulk transmission data is transmitted during idle times between isochronous and interrupted transmissions. The quality of the data is guaranteed. Bulk transmission is used for applications where large volumes of data are transmitted irregularly (keyboard, scanner, digital still camera, storage, etc.).
- Control transmission: This type of transmission is used for the transmission/reception of set-up information such as configuration, address data, etc. The quality of the data is guaranteed.

A device has always one control endpoint. This endpoint is used for control data to set up an USB connection between a device and the host.

The USB protocol makes a distinction between a classes of devices. For each class, the USB data format has a standard. Such a standard has as the advantage that the devices belonging to such a class can use a standard driver for the communication. There exist a lot of class specification, but the most important classes are:

- Communication
- Power Device
- Audio
- IrDA
- Mass Storage
- Printer
- Monitor
- Human Interface Devices

The Human Interface Device (HID) class is most appropiate for the device with the PDP8, because this class specifies devices that send and receive data specified by the descriptors of those devices. This means that this class does not only supports mice and keyboards, but also other kind of devices that use bidirectional communication. The other option for the PDP8 device is to write drivers for this specific product, but this requires more time than writing a few descriptors for the HID device [7]. This is the reason that for the PDP8 device to use the HID class and its drivers. The HID Class uses the control transmission and, if available, the interrupt transmission. Because this is a standard class, standard drivers are used. These drivers will take care of the lower levels of the USB Protocol.

USB devices use descriptors to tell the host what kind of device it is. The most common descriptors are shown in Table 4.1. The last two descriptors are used only for

Table 4.1: Most common descriptors and HID descriptors

Descriptor type	Description
Device descriptor:	the device descriptor includes information such as what USB revision the device complies to, the Product and Vendor IDs used to load the appropriate drivers and the number of possible configurations the device can have.
Configuration descriptors:	the configuration descriptor specifies values such as the amount of power this particular configuration uses, if the device is self or bus powered and the number of interfaces it has.
Interface descriptors:	the interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device.
Endpoint descriptors:	each endpoint descriptor is used to specify the type of transfer, direction, polling interval and maximum packet size for each endpoint. Endpoint zero, the default control endpoint is always assumed to be a control endpoint and as such never has a descriptor.
String descriptors:	String descriptors provide human readable information and are optional.
HID descriptor:	The HID descriptor describes the HID device, HID version it complies to and describes the number and kind of descriptors that follow.
Report descriptor:	The report descriptor tells how the reports send from and to the device look like.

a HID device. The messages send to and received from a HID Device are called reports, the descriptor describing these messages is called the report descriptor.

A device can only have one device descriptor, but can have several configuration descriptors. Each configuration can have multiple interfaces and each interface can have several endpoints. This is illustrated in Figure 4.1. The most simple devices will have only 1 configuration with just 1 interface.

4.1.2 Setting up an USB connection

When a USB device is plugged into the system, the first operation performed is enumerating the device. Enumeration is the process of determining what device has just been

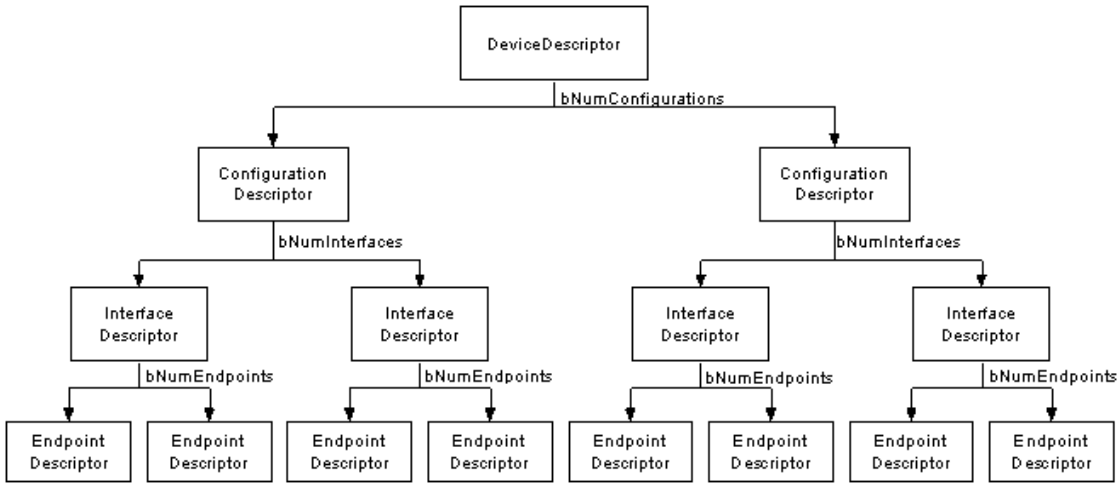


Figure 4.1: Relationship between descriptors

connected to the USB bus and what parameters it requires such as power consumption, number and type of endpoint(s), product class, etc. The host will then assign the device an address and enable a configuration allowing the device to transfer data on the bus. A common Windows enumeration involves the following steps [10]:

1. The host or hub detects the connection of a new device via the device's pull up resistors on the data pair.

2. The host waits for at least 100ms allowing for the plug to be inserted fully and for power to stabilise on the device.

3. Host issues a reset placing the device is the default state. The device may now respond to the default address zero.

4. The MS Windows host asks for the first 64 bytes of the Device Descriptor.

5. After receiving the first 8 bytes of the Device Descriptor, it immediately issues another bus reset.

6. The host now issues a Set Address command, placing the device in the addressed state.

7. The host asks for the entire 18 bytes of the Device Descriptor.

8. It then asks for 9 bytes of the Configuration Descriptor to determine the overall size of this descriptor.

9. The host asks for 255 bytes of the Configuration Descriptor.

10. Host asks for any String Descriptors if they were specified.

The enumeration is completely performed by Microsoft Windows and the device drivers, which can be the standard class drivers if a standard class device is chosen. After its enumeration the device is ready to receive and/or transmit data. In Windows, the file *HID.dll* contains all functions specific for a HID Device. Together with *setupAPI.dll* all functions for searching and connecting a specific USB device are provided and no drivers need to be written.

The first step in finding the correct device is to request a list with all possible devices. This list can be shortened by giving a mask that ensures that only HID devices are returned. This mask can be obtained by using the *HidD_GetHidGuid* funtion from *HID.dll*. This mask is than given as an argument to *SetupDiGetClassDevs*, which returns a list with all devices that fit within the given mask. Device details of the devices in this list can be extracted using the *SetupDiGetDeviceInterfaceDetail* function (before calling this function first *SetupDiEnumDeviceInterfaces* and *SetupDiGetDeviceInterfaceDetail* needs to be called to gain some arguments). Among these device details is the full device path with which it is possible to open a file on the device using the standard *CreateFile* function. This function returns a handle that can be used to ask for the device attributes using *HidD_GetAttributes* from *HID.dll*. These attributes include the Vendor ID, Product ID and Version number and can tell if this handle is the handle for the device that was needed. Retrieving the device details should be done for all items in the list until the wanted device is found.

4.1.3 Reading and Writing

Once the handle is found, reading and writing from and to the USB device is simple. In the report descriptor, the size of the reports is defined. In Windows it is necessary to send a buffer that is one byte longer than the reportsize. This because of the presence of a Report ID that comes in front of the actual report. Writing to the USB device is now as easy as writing to a file by using the *WriteFile* function.

Reading from USB can be done with the *ReadFile* function. *ReadFile* will read from the buffer that contains incoming reports and will first read the report the came in first. When this buffer is too small, the first reports will be dropped and can't be retrieved. In Windows 98 this buffer is as large as 2 reports, in Windows ME and Windows 2000 8 reports and in Windows XP this buffer is as large as 32 reports. To overcome the loss of reports, the buffer can be enlarged using the *HidD_SetNumInputBuffers* function. In Windows98, this buffer can't be enlarged, but in Windows 2000 and Windows XP it can be enlarged to a maximum of 512 reports [2]. Another problem with reading that needs to be taken care of is that if there is no report in the buffer, *ReadFile* will wait until there is one which means that the program is stalled. In practice this means that it is only possible to read from the USB device using *ReadFile* if it is for sure that a report has been received from the device.

4.2 Communication with the device

The communication between our device and the host is messagebased. All messages together form the communication between the host and the device. First of all, all

messages that are needed will be presented. They are as follows:

- Read from PDP8 RAM
- Write to PDP8 RAM
- Get CRC after reading a block of data to PDP8 RAM
- Get CRC after writing a block of data to PDP8 RAM
- Write the Switch Register of the PDP8
- Read the value of the Switch Register from the PDP8 computer
- Read Status information about PDP8
- Start PDP8
- Stop PDP8
- Step PDP8: execution of a single instruction
- Reset PDP8
- Send error to host
- Send message to have data on USB

This gives 13 different message types. During the development, a few more message types were defined for writing information to the RAM and sending debug information to the host. However, these message types are no longer needed, but are still defined in the final program. One extra message type is needed to establish the communication, because the *ReadFile* function will not return until it has actually read data from the USB. If there is no data available, it keeps waitng for the data. Therefore, before checking on the presence of messages from the device, first a message with it's header is send and the device will respond by sending that message back. Now the last package received is the one with this header, which means that it is possible to keep reading from USB until this particular message is found without the problem of stalling the program waiting for data on the USB. This all means that in total 18 message types are defined. The used CRC is a CRC-16 with the following polynominal: $x^{16} + x^{15} + x^2 + 1$.

Of course a few requirements are needed that the communication has to fulfill. We defined the following requirements:

- The communication needs to be simple and robust
- The communication must be easy expandable

The first requirement is wanted to keep the code needed to decode a message simple, which also means a lower chance on errors. The second requirement must ensure that addition of message types is easy and straightforward, without any complications, limitations or obstructions.

Table 4.2: Overview of possible messages from host to device

Message type	Arguments and data	size (bits)	asks for reply?
Read PDP8 RAM	Start (max. 4095)	16	yes
	Length (max. 4096)	16	
Write PDP8 RAM	Start (max. 4095)	16	no
	Length (meax. 40)	8	
	Data	12 (max 40x)	
CRC after write	Start (max. 4095)	16	yes
	Length (max. 4096)	16	
	CRC of written data	16	
Write Switch Register	Data	12	no
Read Switch Register	-	-	yes
Read PDP8 Status	-	-	yes
Start PDP8	-	-	no
Stop PDP8	-	-	no
Step PDP8	-	-	no
Reset PDP8	-	-	no
Ask for reply on USB	-	-	yes

To keep the communication simple and robust we always send messages of the same size. Every message will consist of 64 bytes. This size is the maximum size the USB Controller in the device can accept at once. Otherwise bigger messages will be send in several packages, which means a complication of the device firmware. A message with a standard length of 64 bytes will ensure simpler code for both the device and the host. Larger messages mean that the message need to be transmitted in several packages. Smaller messages are also possible, but with smaller messages more messages may have to be sent to accomplish a single task. The first byte of the message contains the header, which tells what kind of message it is. After this header there may be some arguments or data, dependant of the message type. A header of 1 byte gives us 255 message types, which is much more than the amount of messages required. In fact the first byte is splitted and the first half defines a group of messages en the second half gives a message in that particular group.

A message in general will have a length of 64 bytes and the first byte will be the header. The 63 bytes that are left will be dependent on the message type used. First it

Table 4.3: Overview of possible messages from device to host

Message type	Arguments and data	size (bits)	asks for reply?
Read PDP8 RAM	Start (max. 4095)	16	no
	Length (max. 4096)	16	
CRC after read	Start (max. 4095)	16	no
	Length (max. 4096)	16	
	CRC of read data	16	
CRC after write	Start (max. 4095)	16	no
	Length (max. 4096)	16	
	CRC of host	16	
	CRC of written data	16	
Read Switch Register	Data	12	no
Read PDP8 Status	Data (accumulator)	16	no
	Data (link)	16	
	Data (MQ register)	16	
	Data (Program counter)	16	
	Data (Memory value)	16	
Ask for reply on USB	-	-	no
Error message	0-terminated String	max. 59 bytes	no
Debug message	0-terminated String	max. 59 bytes	no

will contain some arguments and this is followed by data if required. In Table 4.2 are the messages the host can send to the device given with the arguments and data needed. In the first column is the message type, which indicates the first byte, the header. In the second column the arguments and data are shown. The size of each argument or the data is given in the third column. The fourth column tells whether the host can expect a reply from the device, this reply from the device will always have the same message type as the message that caused the reply. This means that the name of a message must always be seen from the side of the host, otherwise some names may not make any sense. In the message all the arguments and data are concatenated and the reamining part of the message is filled with zeros.

The *CRC after write* message is always sent after writing a block of data to check if this block is written correctly. This does not mean that the data is checked after each write message, because writing a block of data can take up to 103 write messages (maximum of writing 4k words with 40 words a message).

A message from the device to the host follows of course the same format as the the messages from host to device. All possible messages the device can send are shown in

Table 4.3. None of the messages needs a reply as can be seen in the table.

During the development, another message type was introduced for reading and writing the RAM. Before implementing the final RAM read and write functions, the RAM was seen as a block of RAM with a length of 16kB and a width of 1 byte instead of a RAM with length of 4k and width of 12 bits. This way of looking at the RAM corresponds more with the physical implementation and was easier to program and to test the hardware. After everything was found to function correctly, the functions for reading and writing the RAM the way as they were seen by the PDP8 processor were implemented. This made the original functions for reading and writing the RAM obsolete. Although these functions are not needed anymore, they are still included in the software of both the host and the device together with the routines written for processing these messages. The messages are almost the same as the messages for reading and writing to the RAM as mentioned above. However, the maximum values for start, length and size of the data differ.

4.3 Graphical user interface

This section describes the development of the graphical user interface.

4.3.1 Borland C++ Builder

The interface of the host program is build with standard widgets from Borland C++ builder 5. Borland creates a class for the interface. This class contains all the buttons, edit-fields, timers and other controls as seperate objects. Each of these object has its own properties that can be accessed just in the same way as members of a class are accessed. Operations on the different objects can be performed by the implemented member functions of the class. Dealing with the interface in this way, where buttons and other objects are inserted by simply dragging the object to the interface, is easy and creating the interface does not take a significant amount of time. Because the inerface can be created quickly, most of the time can be invested in programming the response of an event with a object. These responses or callback functions are implemented as member functions of the interface.

4.3.2 Layout

The program is divided into three main parts. The first part delivers the control of the PDP8 machine. The second part takes care of reading, viewing and writing to the RAM per word. The last part is an editor where files can be edited, created, compiled and written to the PDP8 RAM. The distinction between these three parts is made visible in the program by putting every part on a separate tab sheet.

The first sheet is shown in Figure 4.2 and contains 4 different groups of items. These 4 groups are the following:

- PDP8 Control: this group contains the buttons for controlling the PDP8. Here it is possible to start, stop, step and reset the PDP8.

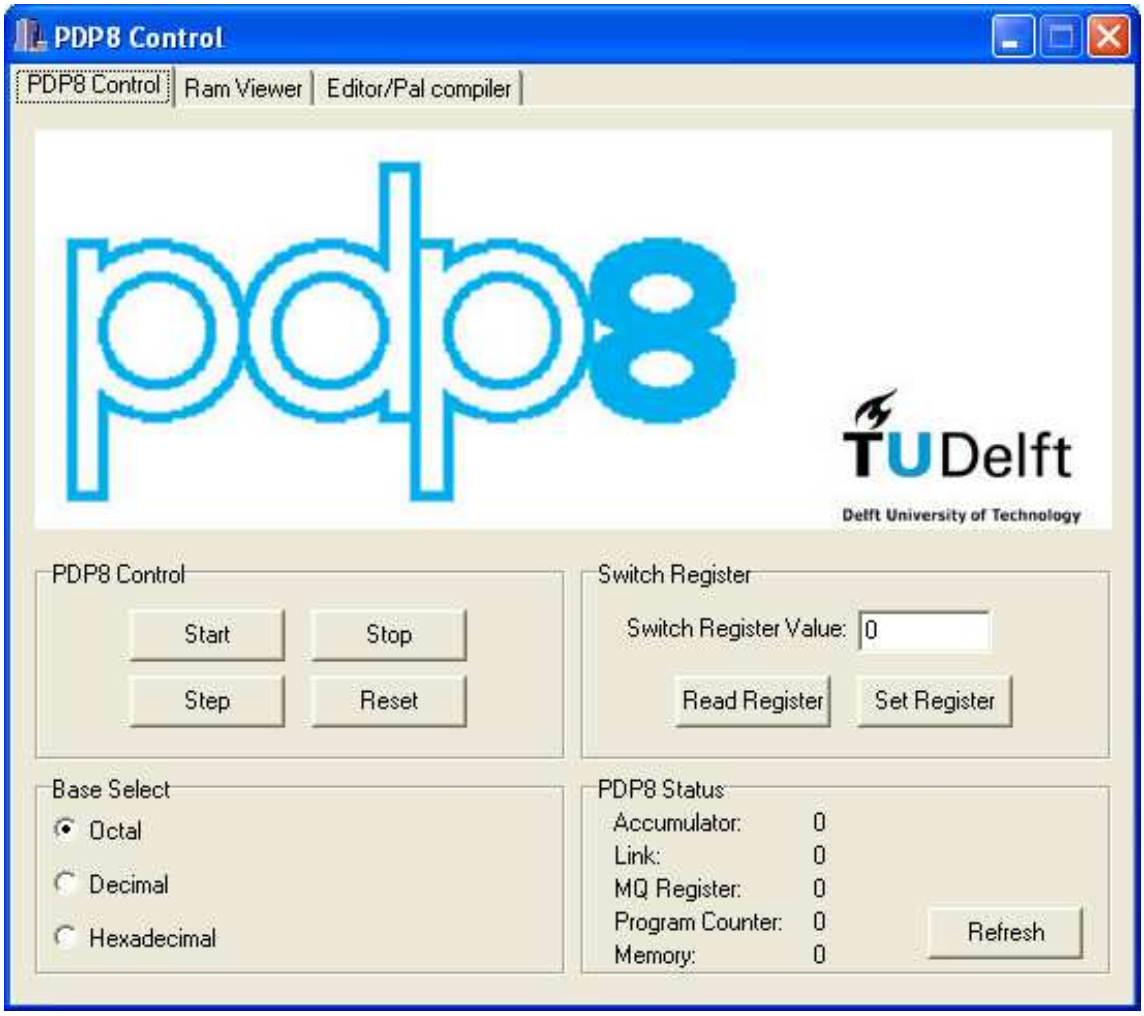


Figure 4.2: Screenshot sheet 1

- Base Select: this group selects the base in which to view and edit the values of the items in the last two groups.
- Switch Register: this group contains the buttons and the edit field to read a value from the switch register or write a value to this register.
- PDP8 Status: this group shows the status of the PDP8 processor, the values of the accumulator, MQ register, link bit, program counter and the value of the memory at the position of the program counter.

The selected base has influence on the values on this sheet only, not on any other sheets. With the controls on this sheet, it is possible to control the PDP8.

The second sheet is shown in Figure 4.3. This sheet provides the means to manipulate and examine the RAM of the PDP8. At the left side the table with the RAM data is shown. In the first column of this table the address of the next data item is shown

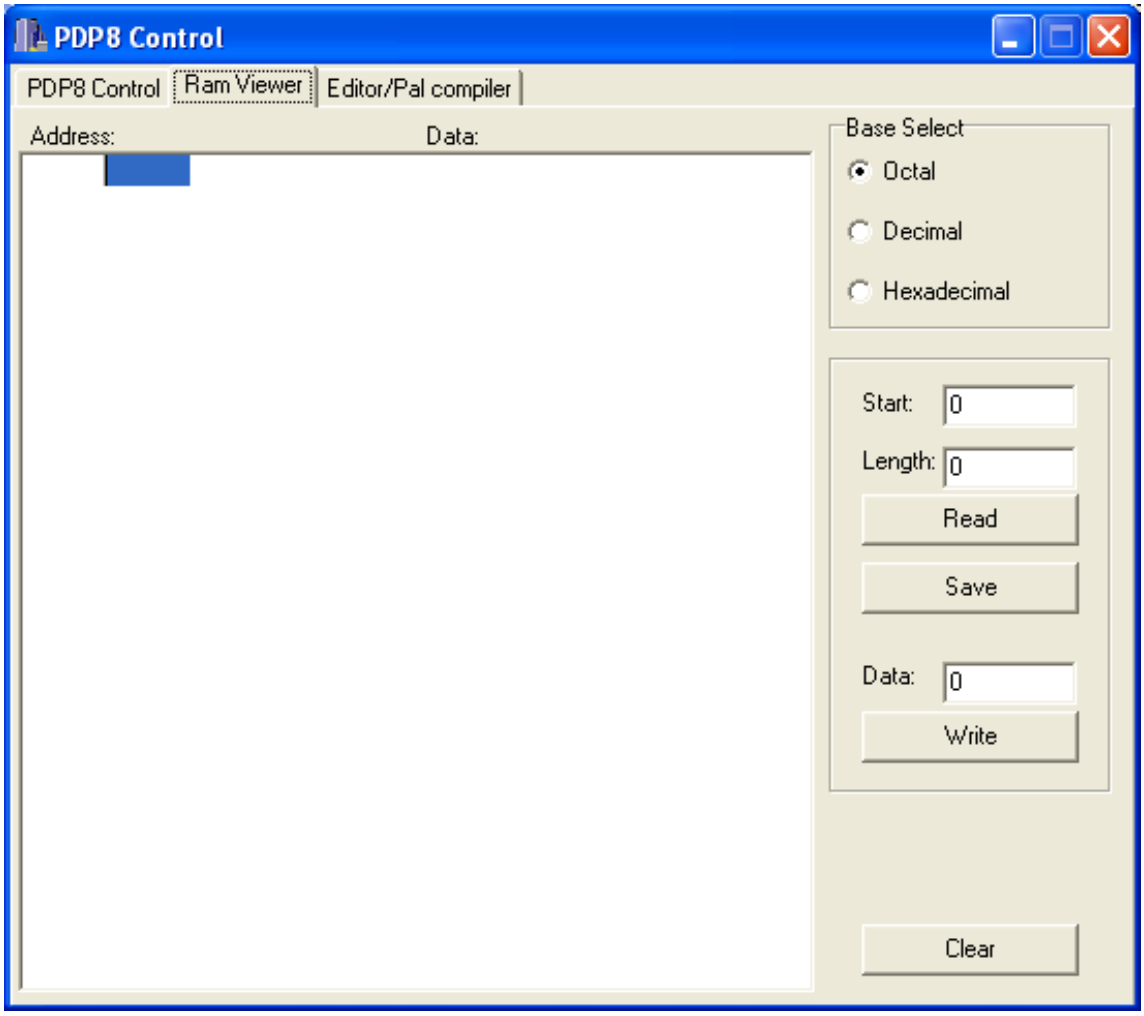


Figure 4.3: Screenshot sheet 2

and each row contains eight data items. At the right side of this sheet the controls for manipulating the RAM are provided. The box at the top selects the base in which the data items on this sheet are shown. Below this box it is possible to specify a start address and a length which to apply the read, write and save operations to. The write operation has one extra box that specifies the datavalue that is to be written to all the selected memory places. At last, in the bottom right corner is a button called clear, which clears the table with the RAM data.

In Figure 4.4, the third sheet is shown. This sheet contains the controls for creating and loading binaries to PDP8's RAM. This sheet contains an editor field where file can be edited in PAL8 assembly format. This means that a box for loading and saving a file in this PAL8 format is present. Binaries in paper tape format are created when compiling this PAL8 file. Therefore it is also possible to load binaries or load them and directly write them to the RAM of the PDP8. The third box on this sheet contains the controls to compile the text in the editor box and write the result to the RAM of the

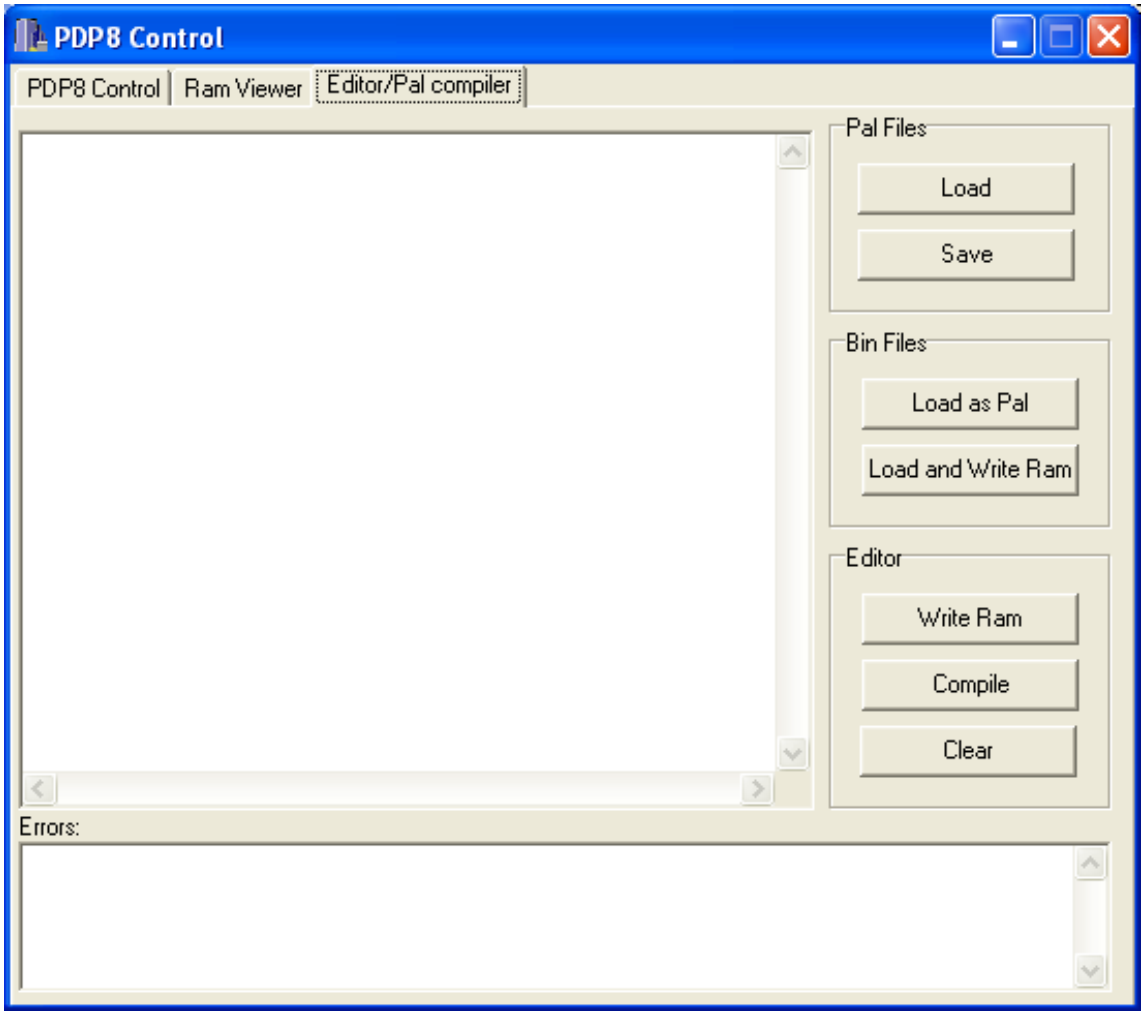


Figure 4.4: Screenshot sheet 3

pdp8. At last, a button is provided for clearing the whole box at once. The compiler shows its output in the errorbox below the editor box.

4.4 Software organization

Action performed on the interface specify what happens. Below, the several different types of used objects and the way that actions belonging to events of these object are discussed.

4.4.1 USB connection

The USB connection is established as earlier explained in Section 4.1.2. For connecting the function *ConnectUSB* is created:

bool *ConnectUSB();*

This function tries to find the PDP8 Device and if it succeeds it tries to connect. The VendorID, ProductID en version number needed to find the device are defined in this function. Connecting means finding the full path of the device in Windows, creating the handle and opening the device using this handle. When trying to connect to the device while the device is already opened, the device will be opened again. Every time the USB connection is needed, the host calls *ConnectUSB*. If this function fails to connect an errorbox is created that asks to verify the physical connection of the device with the host. Then the option is given to retry to connect or cancel the action. If the function succeeds in opening the device, the function returns ***true***, otherwise it returns ***false***.

4.4.2 Base selection group

The interface contains a base selection group on both the first and second sheet. When the base in which numbers are represented is changed, the numbers in the edit-boxes are transformed from the old base to the new base. The function *ConvertBase* takes care of this conversion:

char* *ConvertBase(*
AnsiString str,
int *old_base,*
int *new_base);*

The first argument is in ansistring format, because this is the format in which the data is gathered from the edit boxes. This function builds from the input string a new string. The new string needs to be copied directly after returning from this function, because otherwise the string will be lost.

After clicking on the basegroup, the callback function *OnClick* is called and all values on the current sheet are changed according to the selected base.

4.4.3 Edit boxes

The edit boxes are found on both the first and the second sheet. These boxes can only contain the digits or letters that are that are allowed in the selected base. When the content of the edit-box changes the callback funtion *OnChange* is called. This function analyzes the string in the edit-box and discards all digits and leters that are not allowed in the specified base. This means that when a number digit or letter is entered that is not allowed, this letter is directly removed and an message beep is sounded, indicating an incorrect input.

Also the max number of digits or letters in the number is checked for. This is easy, because the maximum number of letters in a string is a standard property of the edit-box. Each edit-box also can have a maxinum number. To check if the number in the edit-box isn't too large, the callback function *OnExit* is called after leaving the edit-box. If the edit-box containd no string a zero is entered into

the box. This funcion also calculates the value of the string using the function *CalcValue*:

```
int CalcValue(
        AnsiString str,
        int base);
```

If this function returns a value that is too large for that specific edit-box, the value will be changed to the maximum allowed value.

4.4.4 Open and save file dialog

By calling the member function *Execute* of the dialogs, the dialog is launched. The dialog is a standard dialog as it comes with Windows. Properties of the dialog can be set to specify for example file types and overwrite messages. When the dialog closes it returns the button that caused this return, for example the cancel, save or open button. If a file was selected, its full path is saved in one of the properties of the dialog and can be used for opening or saving a file.

4.4.5 Timer

The timer is needed for receiving data from the USB device. Every second, the callback funtion *OnTimer* is launched on a timer event and the first thing that happens is sending the report that asks for a reply from the USB device. Then the callback funtion starts reading from the USB until the reply is found. Every report that is received is processed. A few types of reports can be received (excluding the reply just asked for):

- Read PDP8 RAM: When this report is received, the data is saved into a variable representing the PDP8 RAM. The data is also displayed in the matrix for RAM data.
- CRC after read: This report contains two CRC-16 values. When these values differ, a errorbox is launched, telling that the read operation was unsuccesfull and the received data may be corrupt.
- CRC after write: This report contains two CRC-16 values. When these values differ, a errorbox is launched, telling that the write operation was unsuccesfull.
- Read Switch Register: the value of the switch register is replaced by the value provided by this report.
- Read PDP8 Status: the status information contained in this report is written to the labels on the first sheet.
- Error message: if this report is received, an errorbox is launched with the message that is provided as a string in the report.
- Debug message: this type of report is only used during development and will have no function in the final version.

4.4.6 Buttons

When a button is pressed, the specified *OnClick* callback function is called. All buttons on the first sheet simply send a report to the PDP8 device. The callback funtion first prepares the report and then sends this report the de PDP8 device. If the device sends a reply to this message, this reply is handled by the timer.

The second sheet contains the following buttons.

- The read buttons prepares a report that requests the data from the PDP8 RAM specified by the two edit-boxes above this button.
- The save button saves the RAM data specified by the same two edit boxes from the variable representing the PDP8 RAM. The data is saved in papertape format.
- The write button sends a buffer with to the PDP8 device. The amount of data is specified in the edit-box for the length and determines how many reports are needed. A buffer with the specified length is filled with the data indicated by the data edit-box. With the startaddress in the address edit-box, the more general function *pdp8_ram_write* is called which sends the buffer to the device.
- The clear button in the bodom right corner calls a member function of the RAM data matrix that clears the entire matrix.

For sending a buffer to the PDP8 device, a more general function is used. The prototype of this function is given below:

```
void pdp8_ram_write(
        unsigned short *buffer,
        unsigned int begin_start,
        unsigned int length);
```

The *begin_start* parameter is used to specify at which address the first word in the buffer is to be written in the PDP8 RAM.

The third sheet contains three separate groups of buttons. The first group deals with pal assembly files. The load button calls the open file dialog and on return it opens the selected file in the editor-box. Loading a text file in the editor-box is a standard member function of the box. The save button calls a save file dialog and on return it saves the content of the editor-box to the specified filename. Saving the content is also a standard member function of the box.

The second group on the third sheet deals with loading binary files. A binary can be loaded as a pal assembly file and loaded into the editor-box. First the open file dialog is executed and then the selected file is opened and processed to the pal format that can be understood by the compiler. The other button in this groups opens the binary file and in stead of writing it in pal format in the editor-box, the data is saved into a buffer and *pdp8_ram_write* is called, writing the entire file content directly to the PDP8 device.

The third group of buttons takes care of compiling the content of the editor-box and writing it into the RAM of the PDP8. The write RAM button first saves the

content of the editor-box using the name and path of the file that was opened. If no file was opened, a save file dialog appears and a name can be specified. When the text is saved, the extern compiler is called, using the *system* function. This function is able of executing a DOS-command. This DOS-command is given as a string parameter to the *system* function. The string of the DOS-command looks as follows:

pal.exe <FILENAME> 2>tmp93455.txt

<FILENAME> must be replaced by the full path of the pal inputfile. This path must be a short path name: directories and files must be in their short 8 character, no spaces format (ISO 9660 level 1). The command behind the filename redirects all standard error output to a temporary file. After completion of the compiler, the temporary file is loaded into the error-box and this file is deleted. If after loading the file into the error-box, the error-box stiill is empty then compiling has succeeded and a message is written into the error-box and the created binary file is written to the RAM in the same way as the button that writes a binary direct into the RAM.

The compile button only compiles the text in the editor-box, but does not write it to the RAM of the PDP8 on successful completion. The clear function clears the editor-box by calling a standard member function of this box.

5 Conclusions

In this thesis the design of a fully controllable PDP8 system was presented. The PDP8 system provides a system that is able to run programs from the early days of computer history. This way, the system preserves a historical value in the history of computers. The design trajectory of the PDP8 project was introduced involving: specification definition; schematic design; PCB (Print Circuit Board) design; software design for both the device and the host and verification af the design with the system specifations. Considerations done during the system specification were presented. The system specification was defined, as a result of the requirements and features. Furthermore the schematic design of the PDP8 computer, consisting of creating a functional design conform specifications was presented and the PCB design, consisting of converting the functional design to a physical hardware design, was discussed. At last, the PC host software was presented.

This chapter presents conclusions and highlights the main contributions. This chapter is organized as follows: Section 5.1 summarizes the main conclusions of this thesis and Section 5.2 presents the main contributions.

5.1 Summary

Chapter 2 introduced the system specification, which is required to design a system. There were several requirements for the PDP8 computer design: start the PDP8 processor; stop the PDP8 processor; step the PDP8 processor; reset the PDP8 processor; write a value in the switch register; read the content of the switch register and the RAM of the PDP8 should be completely controllable.

In addition, Chapter 2 introduced the design specification of the PC host software. This reslulted in a list of requirements that are summarized hereafter. Start the PDP8 processor. Stop the PDP8 processor. Step the PDP8 processor. Reset the PDP8 processor. Set the switch register of the PDP8 computer. Read the value of the switch register and display this value. Read status information of the PDP8 processor. Write data to a specified location of the PDP8 RAM. Read data from a specified location of the PDP8 RAM. The read data must be displayed in an appropiate way. Store data from the pdp8 RAM in a binary file. This file must be in a common format. Load a binary file in a common format into the RAM of the PDP8 processor. Provide an editor to edit PAL assembly files. Be able to load a PAL file into the editor. Be able to save the editor text into a pal file. Be able to compile the text in the editor and show the error messages of the compiler. Be able to write the binary output of the compiler to the RAM of the PDP8 processor.

The entire design of the PDP8 computer is presented in Chapter 3. Providing the

RUN/HALT and *RESET* signal inputs of the PDP8 to the C32 sub-system ensures that the first four requirements are fulfilled. The design of the switch register permits the C32 sub-system to write a value in the switch register of the PDP8. However, reading the contents of the switch register is not possible, the switch register can only be changed by the C32 subsystem and not by the PDP8 computer. This means that the value inside the switch register can always be retrieved if the last written value is stored in a memory location of the C32 RAM. Dealing with the switch register in this way fulfills also this requirement.

The last requirement of the PDP8 computer had to deal with the RAM. The RAM should be completely controllable from the host computer. This is accomplished by giving the C32 full control over the RAM. Because the RAM can't be connected to both address and data busses at the same time, a multiplexer is implemented to switch the RAM between the PDP8 computer and the C32 computer and this requirement is also fulfilled.

In Chapter 4 the PC host software design was presented and described. The first seven requirements deal with controlling the processor itself. These controls were all provided on the first sheet of the PC host application. The next three requirements were fulfilled by the controls provided on the second sheet. At last, the reamaining six requirements are all fulfilled by the controls on the third sheet.

5.2 Main contributions

The main contributions of this thesis can be summarized as follows:

- The PDP8 project was initiated and requirements and restrictions for the PDP8 project were defined.
- The design trajectory was presented and was followed in the next stages.
- The specification was created as a result of the requirements.
- The functional design was created conform to the specifications. The features were implemented in the form of a schematic design and the PCB design was created.
- The software for both the PDP8 device and the PC host was created.
- The entire system is checked on consistency with the specifications, which are all fullfilled.

Bibliography

[1] Altium, *Altium.com*, Website: http://www.altium.com, 2004.

[2] Jan Axelson, *Jan axelson's lakeview research*, Website: http://lvr.com, 2004.

[3] Borland, *Borland: Leading provider of technology for software applications*, Website: http://www.borland.com, 2004.

[4] Digital Equipment Corporation, *Small computer handbook*, Digital Press, 1970.

[5] Microsoft Corporation, *Microsoft corporation*, Website: http://www.microsoft.com, 2004.

[6] J. Craig Mudge Gordon C. Bell and John E. McNamara, *Computer engineering, a dec view of hardware system design*, Digital Press, 1978.

[7] John Hyde, *Usb design by example*, Intel Press, 2001.

[8] Intersil, *Datasheet: Im6100 cmos 12 bit microprocessor*, Intersil Inc., 1979.

[9] Douglas W. Jones, *Pdp-8 frequently asked questions*, Website: http://www.faqs.org/faqs/dec-faq/pdp8, 2001.

[10] Craig Peacock, *Beyond logic*, Website: http://www.beyondlogic.org, 2004.

[11] USB.org, *Usb.org - welcome*, Website: http://www.usb.org, 2004.

[12] Chris Ward, *http://www.armory.com/ rstevew/public/pgmrs/eeprom/chrisward/ programmer.html*, Website: http://www.armory.com/ rstevew/Public/Pgmrs/ EEPROM/ChrisWard/programmer.html, 1999.

Pinout IM6100

A

Pin	Symbol	Description
1	Vcc	Supply voltage.
2	RUN	Indication of the runstate of the CPU and may be used to power down external circuitry.
3	DMAGNT	Direct Memory Access Grant - DX lines are 3-stated.
4	$\overline{DMAREQ}$	Direct Memory Access Request - DMA is granted at the end of the current instruction. Upon DMA grant, the CPU suspends program execution until the DMAREQ line is released.
5	$\overline{CPREQ}$	Control Panel Request - A dedicated interruptwhich bypasses the normal device interrupt request structure.
6	$\overline{RUN/HLT}$	Pulsing the RUN/HLT line causes the CPU to alternately run and halt by changing the state of the internal RUN/HLT flip flop.
7	$\overline{RESET}$	Clears the accumulator and loads 7777_8 into the program counter.
8	$\overline{INTREQ}$	Peripharal device interrupt request.
9	XTA	External coded minot cycle timing - Signifies input transfers to the IM6100.
10	LXMAR	The Load External Memory Address Register is used to store memory and peripheral addresses externally.
11	$\overline{WAIT}$	Indicates that peripherals or external memory is not ready to transfer data. The CPU state gets extended as long as WAIT is active. The CPU is in the lowest power state with clovks running.
12	XTB	External coded minot cycle timing - Signifies output transfers from the IM6100.
13	XTC	External coded minot cycle timing - Used in conjunction with the select lines to specify read or write operations.
14	OSC OUT	Crystal input to generate the internal timing (also external clock input).
15	OSC IN	See pin 14 (Also external clock ground).
16	DX0	DataX - multiplexed data in, data out and address lines
17	DX1	See pin 16.
18	DX2	See pin 16.
19	DX3	See pin 16.
20	DX4	See pin 16.
21	DX5	See pin 16.
22	DX6	See pin 16.
23	DX7	See pin 16.
24	DX8	See pin 16.
25	DX9	See pin 16.

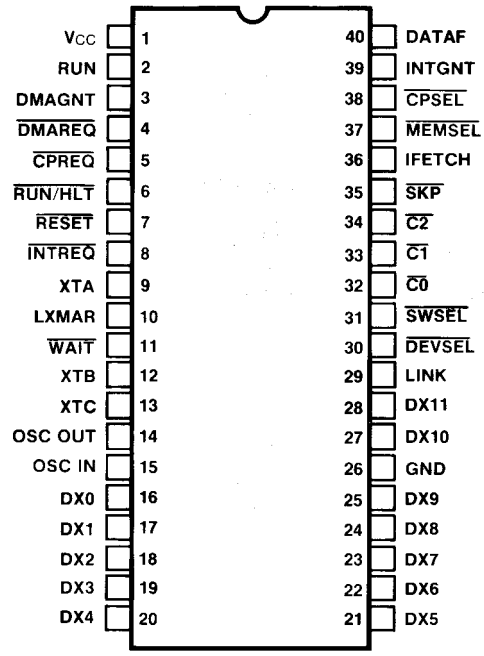


Figure A.1: Pin configuration of the IM6100

Pin	Symbol	Description
26	GND	Ground.
27	DX10	See pin 16.
28	DX11	See pin 16.
29	LINK	Indicates state of link flip/flop
30	$\overline{DEVSEL}$	Device select for I/O transfers.
31	$\overline{SWSEL}$	Switch Register Select for the "or the switch register"-instuction (OSR).
32	$\overline{C0}$	Control line inputfrom the peripheral device during an I/O transfer.
33	$\overline{C1}$	See pin 32.
34	$\overline{C2}$	See pin 32.
35	$\overline{SKP}$	Skips the next sequential instruction if active during an I/O transfer.
36	IFETCH	Instruction fetch cycle.
37	$\overline{MEMSEL}$	Memory Select for memory transfers.
38	$\overline{CPSEL}$	The Control Panel Memory Select becomes active, instead of the MEMSEL, for control panel routines. Signal may be used to distinguish between control panel and main memories
39	INTGNT	Peripheral device interrupt grant.
40	DATAF	Data Field indicates the execute phase of indirectly addressed AND, TAD, ISZ and DCA instructions so that the data transfers are controlled by the data field (DF) and not the instruction field (IF) if extended memory control hardware is used to extend the address space from 4k to maximal 32k words.

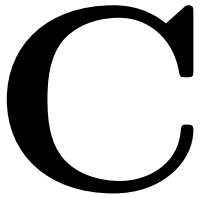
Pinout IM6101

B

Pin	Symbol	Description
1	Vcc	Supply voltage, +5 volts.
2	INTGNT	A high level on INTERRUPT GRANT inhibits recognition of new interrupt requests and allows the priority chain time to uniquely specify a PIE.
3	PRIN	A high level on PRIORITY IN and an interrupt request will select a PIE for vectored interrupt.
4	SENSE4	The SENSE input is controlled by the SL (sense level) and SP (sense polarity) bits of control register B. A high SL level will cause the sense flipflop to be set by a level a low SL level causes the sense flipflop to be set by an edge. A high SP level will cause the sense flipflop to be set by a positive going edge or high level. A high IE (interrupt enable) level generates an interrupt request whenever the sense flipflop is is set.
5	SENSE3	See pin 4.
6	SENSE2	See pin 4.
7	SENSE1	See pin 4.
8	SEL3	Matching SELECT(3-7) inputs with PIE addressing on DX(3-7) during IOT.
9	SEL4	See pin 8.
10	LXMAR	A high positive pulse on LOAD EXTERNAL ADDRESS REGISTER loads address and control data (DX3-11) into the address register.
11	SEL5	See pin 8.
12	SEL6	See pin 8.
13	XTC	The XTC is an timing signal produced by the microprocessor. When XTC is high, a low going pulse on DEVSEL indicates a read operation. When XTC is low, a low going pulse on DEVSEL indicates a write operation.
14	SEL7	See pin 8.
15	DX0	Data transfers between the microprocessor and the PIE take place via these I/O pins.
16	DX1	See pin 15.
17	DX2	See pin 15.
18	DX3	See pin 15.
19	DX4	See pin 15.
20	DX5	See pin 15.
21	DX6	See pin 15.

Pin	Symbol	Description
22	DX7	See pin 15.
23	DX8	See pin 15.
24	DX9	See pin 15.
25	DX10	See pin 15.
26	DX11	See pin 15.
27	GND	Ground.
28	$\overline{DEVSEL}$	The DEVSEL input is a timing signalproduced by the microprocessor during IOT instructions. It is used by the PIE to generate timing for controlling PIE registers and read/write operations.
29	FLAG4	The FLAG outputs reflect the data stored in control register A. Flags (1-4) can be set or reset by changing data in control register A via a write command to the PIE. FLAG1 and FLAG3 can be controlled directly by PIE commands.
30	FLAG3	See pin 29.
31	FLAG2	See pin 29.
32	FLAG1	See pin 29.
33	$\overline{C1}$	The PIE decodes address, control and priority information and asserts outputs C1 and C2 during the IOT instruction cycle to control the type of data transfer. These outputs are open drain for bussing and require a puul-up resistor to Vcc.
34	$\overline{C2}$	See pin 33.
35	READ1	Outputs READ1 and READ2 are used to gate data from a peripheral device onto the data bus for input to the PDP8. Data does not pass through the PIE.
36	WRITE1	Outputs WRITE1 and WRITE2 are used to gate data from the data bus of the PDP8 processor to a peripheral device. Data does not pass through the PIE.
37	READ2	See pin 35.
38	WRITE2	See pin 36.
39	$\overline{SKP/INT}$	The PIE asserts this line low to generate interrupt requests and to signal the processor when sense flipflops are set during skip instructions. The output is open drain.
40	POUT	A high level on PRIORITY OUT indicates no higher priority PIE interrupts requests are outstanding. This output is tied to the PIN input of the next lower priority PIE in the chain.

EEPROM Programmer



The program running on the C32 sub-system is located in an 32kx8 EEPROM (Electronical Eraseble Programmeble ROM). The EEPROM programmer used to program the EEPROM of the C32 computer is a modified version of Chris Ward's programmer [12]. The programmer is controlled from a application running on a host computer. Communication with the programmer goes via the parallel interface. His version was capable of programming 8kx8 and 2kx8 EEPROMs and reading 8kx8, 16kx8 and 32kx8 EPROMs. A few modification have been made to be able to program a 32kx8 EEPROM. The functionality to read EPROMs was removed to keep it simple. In Figure C.1 the modified schematic design of the EEPROM programmer is shown. The shown EEPROM in the schematic is actually a ZIF-socket to be able to easily install and remove a EEPROM in the programmer.

The software of the programmer also needed some modification. A screenshot of the programmer application is given in Figure C.2. In the next paragraphs, the modifications that were needed are discussed.

First of all, the choice between three EEPROM sizes in the program in stead of the five options that were present in the original version is changed.

Second, the ready bit used to verify if the byte is programmed is not present on the largest EEPROM. Therefore, the EEPROM is polled after a write operation until the byte is written. If the byte is not written yet, the most significant bit of the returned data has changed polarity.

At last, the program runs under Microsoft Windows XP. In this version of Windows, Windows 2000 and Windows NT direct access of the parallel port is prohibited. Access to this I/O port can only be done through a driver. *INPUT32* is a function library that contains functions that install a driver when the library is loaded. Using the I/O functions in this library, the parallel port can be accessed 'directly'.

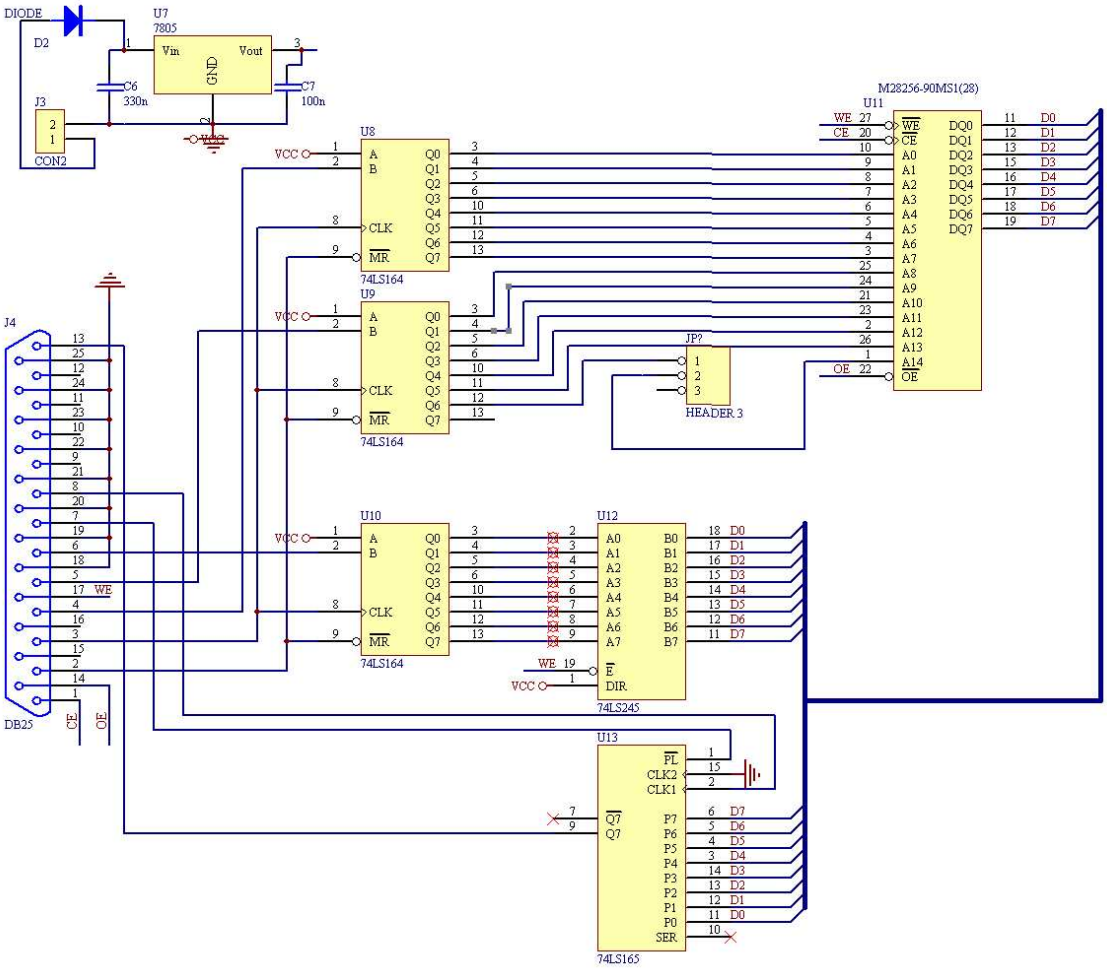


Figure C.1: Schematic design of the EEPROM programmer

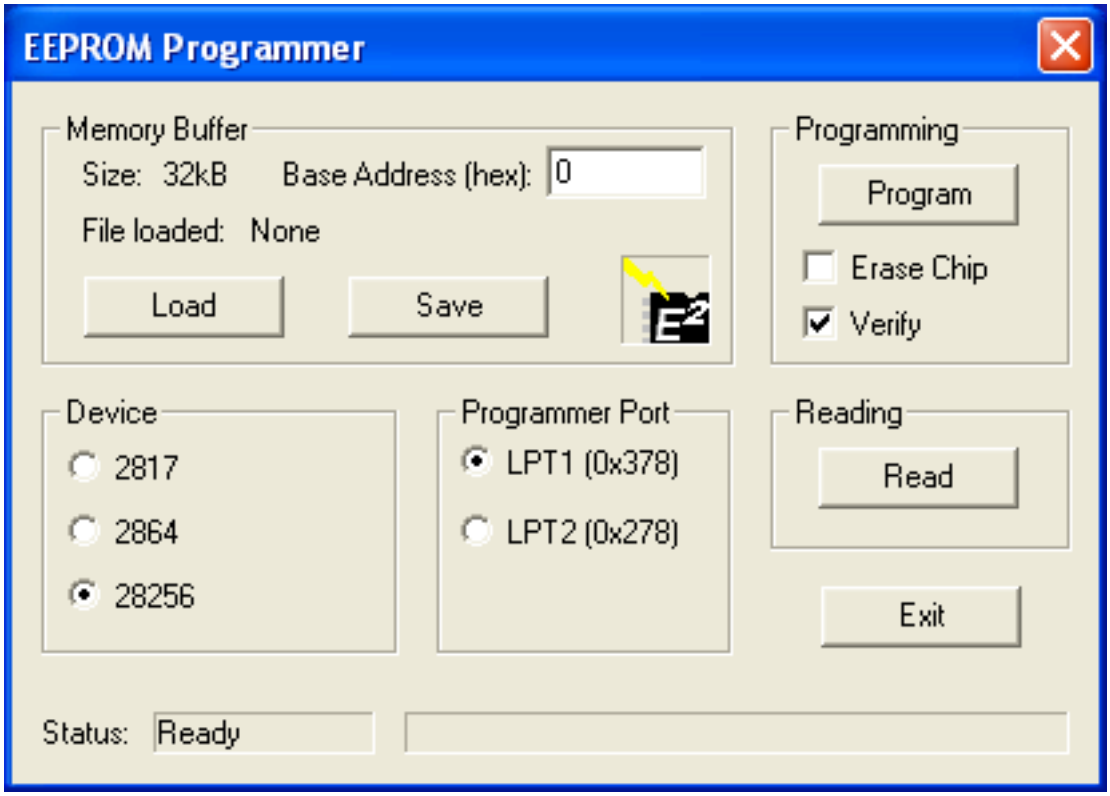


Figure C.2: Screenshot of the EEPROM programmer

Curriculum Vitae

Johan van de Pol was born in Rhenen, The Netherlands, on December 24 1980. He attended the Meerwegen College high school in Amersfoort, from which he graduated in 1999. In that same year, he was admitted to the Electrical Engineering faculty of the Delft University of Technology in the Netherlands.

After receiving his Bachelor of Science degree, he joined the Computer Engineering laboratory, led by professor Stamatis Vassiliadis, to start his MSc graduation project under the supervision of Ir. Georgi Gaydadjiev. His thesis is titled **USB-enabled PDP8 computer**. His research interests include: computer architecture, embedded systems and programming.