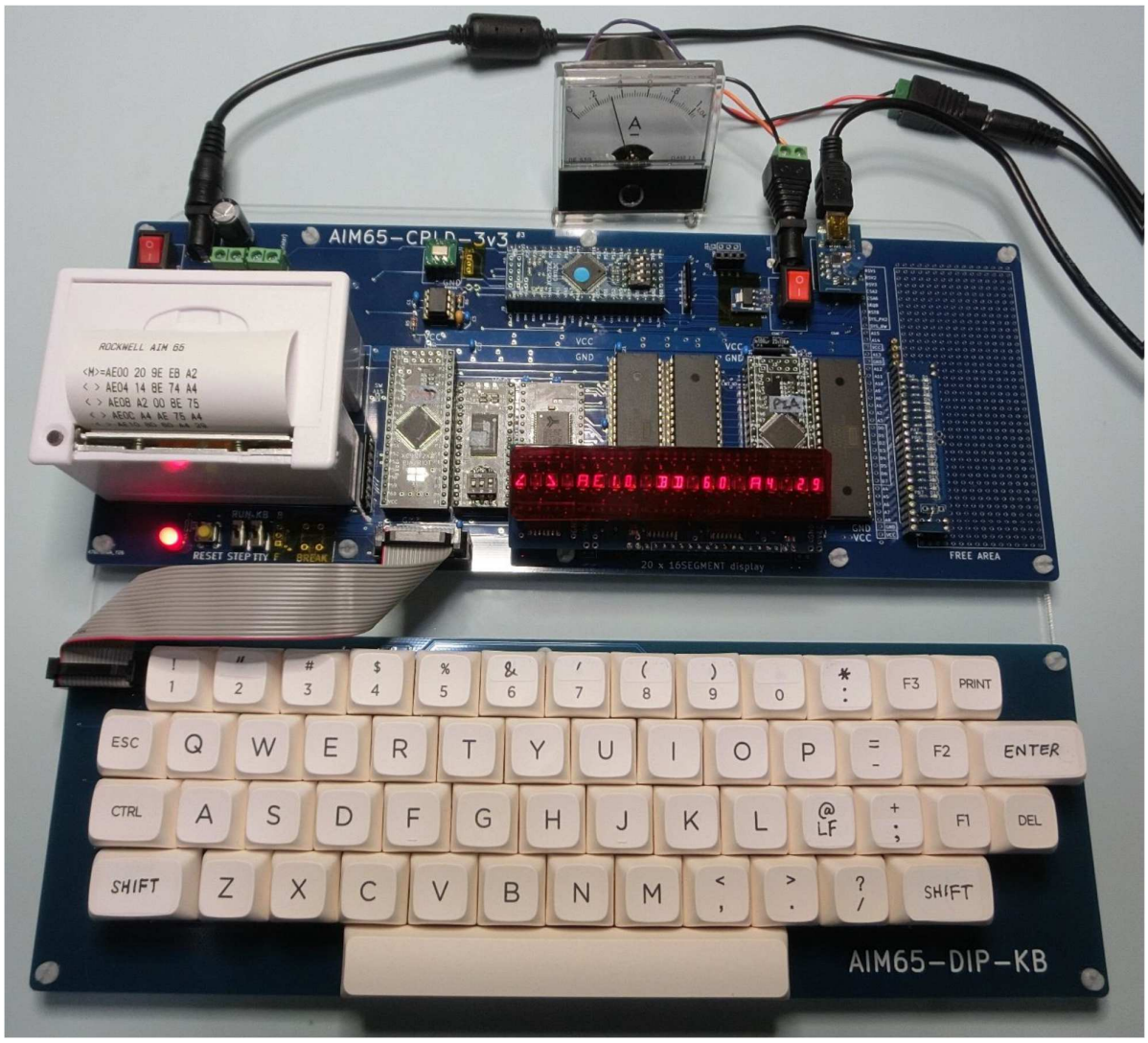
Revival

Retro Computer

AIM 65



March 31, 2023 Rev. 0.5 (Japanese)
April 26, 2023 Rev. 0.2 (English)

The circuit diagrams in this document are based on the board computer AIM 65 USER'S GUIDE and reference card manufactured and sold by Rockwell International Corporation since 1976 with some modifications. AIM 65 As the product is more than 45 years old, we have not obtained the consent of the manufacturer for the preparation of this document.

The contents of this document are subject to change without notice.

The printed circuit board, diagrams and sample code in this manual can be used freely but are provided on "as is" basis without warranties or conditions of any kind, either express or implied, including, without limitation, any warranties or conditions for fitmess for a particular purpose. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with your exercise of permissions under the Creative Commons License.

The AIM 65 design is based upon the material published on the retro computer site http://retro.hansotten.nl.

1 TABLE OF CONTENTS

2 INTRODUCTION

What would it be like to revive a 45-year-old microcomputer? I thought that the production boom of reproduced versions of microcomputers in the early days had passed long ago, but the design information of AIM65 has been published on the Internet, and even now, original parts and replacement parts are still available. Since it is possible to buy these, I decided to give it a try.

3 DESIGN CONSIDERATIONS

The AIM 65 has a clock frequency of 1MHz, PROM/SRAM memory is, and consists of R6502 CPU, R6522 VIA, R6532 RIOT, R6520 PIA (=MC6821). In other words, it is a microcomputer board composed only of the R6502 and its peripheral ICs and has an extremely simple hardware configuration. It is a hardware configuration that says that if you design a microcomputer board using the R6502, please look at this and design it. Anyone can make one if they can buy the parts. It's not easy to create the software, but it's easy to get because there are old ones on the internet that are not subject to copyright. The hurdles for designing/manufacturing a printed circuit board are low both technically and financially if it is a two-layer board.

This document is a design document for reviving the AIM 65 using currently available parts, and half of it is a work memo for myself, so please read it after understanding it. There are many individual parts that have no choice but to use SMDs (Surface Mount Devices), but we created a DIP conversion board for them and mounted through holes on the main board so that anyone can manufacture them with good reproducibility. I got it as a board.

For people who grew up in that era, retro computers remind me of the slow processing speed and inconvenience, and I think that they feel nostalgic and feel nostalgia, but for young people, even if they were restored, that would be too much. The lack of features and slow speed will probably discourage people from actually using it. Even so, the AIM 65 has a teletype interface, and in the past it was possible for individuals to have a teletypes, But if you use the terminal software that runs on today's personal computers, you can comfortably use them even with old microcomputers. Data input/output and program saving/loading are slow.

Since early microcomputers consisted of a CPU, ROM, RAM, and peripheral ICs connected by a bus on a printed circuit board, it is a suitable environment for learning the operation of each component part. Unlike today's MCUs, the microcomputers of that time had their address bus, data bus, and control signals all available at the outside of the IC, so their behavior could be observed with an oscilloscope. Oscilloscopes are completely different from 45 years ago, and you can get high performance and cheap ones.

UARTs and parallel interfaces were common in the past. Although the I2C bus did not exist before, master operation can be achieved by writing software even with an old

microcomputer, so the latest and feature-rich I2C-connected devices can still be connected and used.

The goal of this reproduction edition is to make it a “usable AIM 65” by using the latest text editors and cross compilers on a current PC so that you can feel the old days.

4 THE AIM 65

The AIM 65 is an SBC (Single Board Computer) released in 1976 by Rockwell International with a MOS 6502 CPU. 1976 was the dawn of the microcomputer era, when new microcomputers appeared one after another. NEC's TK-80, KIM-1 and APPLE 1 were all released in 1976. Of course, there were Altair 8800 (1974) and IMSAI 8080 equipped with i8080 before that. However, compared to a microcomputer that inputs binary numbers with a toggle switch and displays hexadecimal characters with a hexadecimal key input and a 7-segment LED, it has a 16-segment LED that can display English characters, a thermal printer, and a full QWERTY keyboard. It can be said that AIM 65 was a very novel machine.

Hardware specifications of the AIM 65

- Full keyboard with QWERTY layout
- 20-digit alphanumeric LED display (16 segments)
- 20-digit thermal printer
- Teletype interface with 20mA current loop
- Dual Cassette Interface
- Equipped with a R6522 VIA chip for expansion
- 4KB RAM
- Equipped with sockets for a total of 20KB (4KB ROM/EPROM chips x 5)
- Although there was a floppy disk controller available, it seems that there was no major DOS such as CP/M for the AIM 65.

AIM65 software specifications

- Machine language monitor program
- (Including memory hexadecimal display and setting, 1-line assembler (mnemonic input), disassembler)
- 2-pass assembler
- 8K real type BASIC interpreter
- FORTH and MATH PACK for FORTH (real number arithmetic package)
- PL/65 compiler
- Others (Instant PASCAL, GWK-BASIC (PC-100 BASIC interpreter sold by Siemens of Germany) are available.

A special feature was that the Users Guide includes circuit diagrams, memory maps, register maps, detailed descriptions of each interface, and a monitor program source list. These materials are open to the public on the Internet, and you can still find out more about them.

This AIM-65 reproduction was created based on the materials on the following website: http://retro.hansotten.nl/6502-sbc/aim-65/

Although it is called an AIM 65 reproduction, the cassette tape interface has not been reproductiond. The VIA R6522 signals and software are left intact, so I think it will be possible in the future. However, if you use a PC serial terminal emulator (e.q. the Teraterm file transfer function or equivalent on your Operating System) instead of TTY's paper tape interface, you can transfer data, so I think there is no need to reproduce the cassette tape interface.
Also, since the same type of thermal printer as the original is not available, a thermal printer unit with UART serial connection is connected. The software has two patches in the MONITOR ROM that call a subroutine that prints the data in the 20 character buffer. The printer driver software was put in the free I/O space of I/O. The printer driver is not special, it is software that only transmits data with 9600bps UART.

This allows the software on the original ROM to run as-is. 8K Real BASIC, FORTH (+ MATHPACK), PL/65, ASSEMBLER, instant PASCAL. Of these, MATHPACK, PL/65, and PASCAL have been confirmed to start, but have not been thoroughly tested. It would be nice to have some sample coding or use cases.
There is no C language self-compiler for the 6502, but there is the cc65 cross-compiler on the PC. With some effort, the AIM 65 can also run C programs.

4.1 Specification of the AIM 65

Let's have a look at the hardware and circuit diagram of the AIM 65.

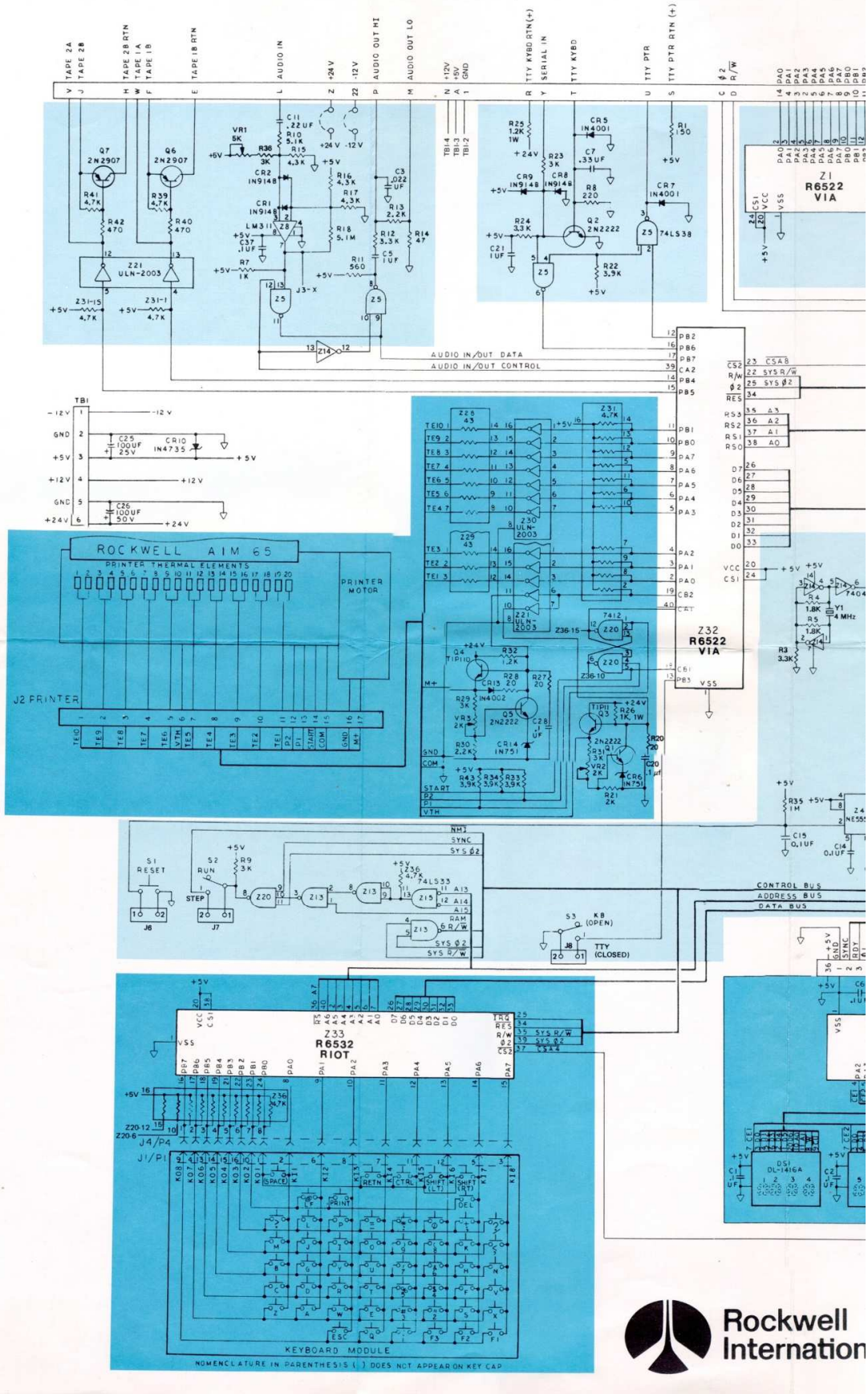
The next figure is a schematic poster delivered with the AIM 65 (approximately 60x45cm in size), including the main board and keyboard of AIM 65.

EPROM 4 KB x 5, SRAM 1 KB, 4 KB even if all 8 are installed in sockets, R6520 PIA (equivalent to MC6821) for 16 segment LED control, thermal printer and cassette tape interface.

The R6522 VIA is installed in the system, and the R6532 RIOT is installed for the keyboard interface. In addition, R6522 VIA for user expansion can be installed and used freely by the user.

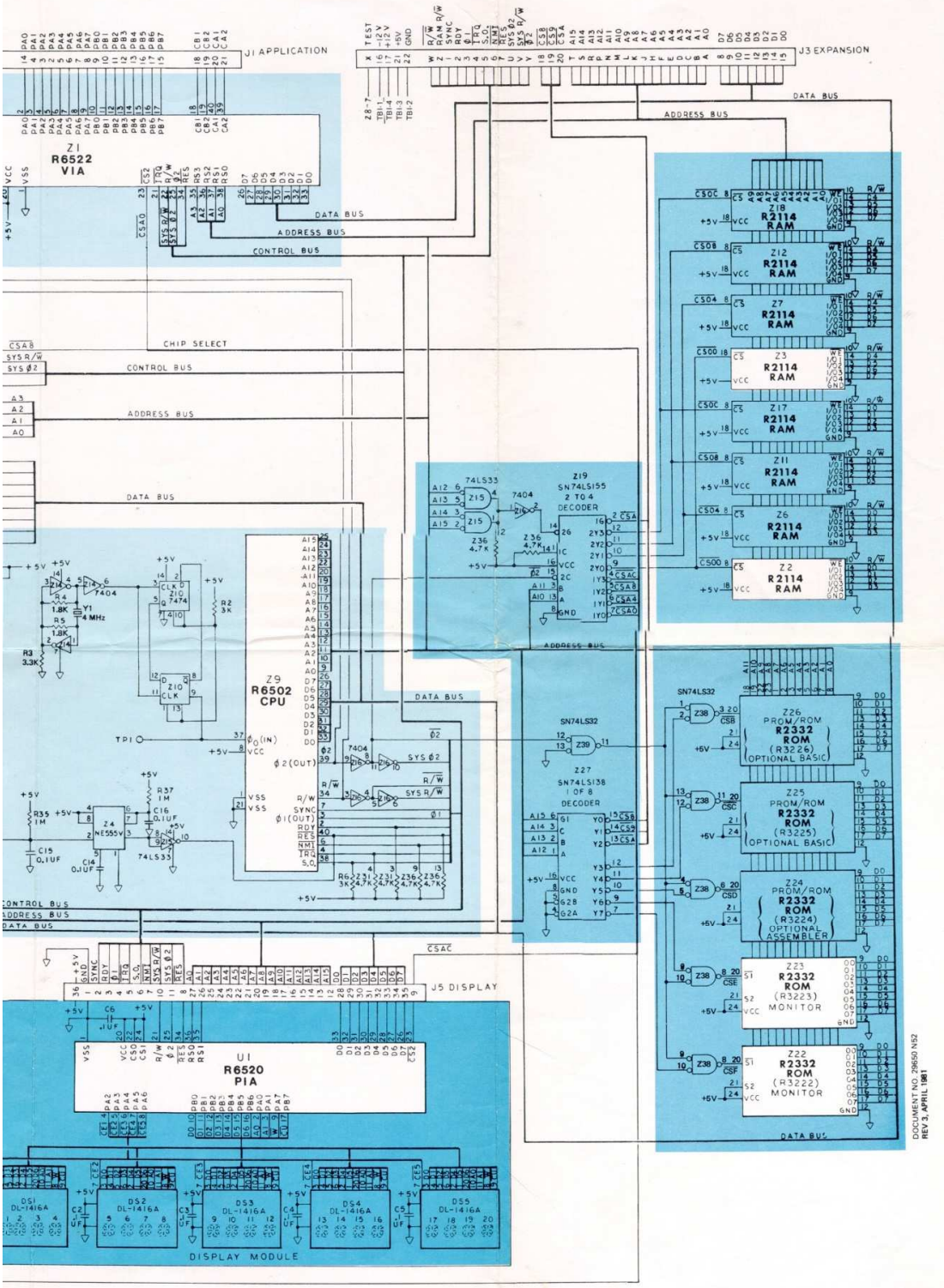
Among these parts, the CPU R6502 and VIA R6522 were CMOS products from Rockwell and equivalent products from other companies, so there are still reused or new products on the market. The troublesome one is the R6532 RIOT, an IC with built-in 128 bytes of RAM, 8 bit GPIO x2 ports, and timer x1. I think it was convenient to create a system with a minimum configuration in the era when memory was expensive, but it is difficult to use it because only a small RAM capacity appears on the memory map. In addition, the R6532A (2MHz version) is NMOS and consumes a large amount of current, here are no high-speed versions that exceed the 2MHz version, and there is no CMOS version. It becomes an obstacle when you want to make a high-speed version of the AIM 65.

Rockwell AIM 65 Advanced In



Rockwell
Internation

ed Interactive Microcomputer

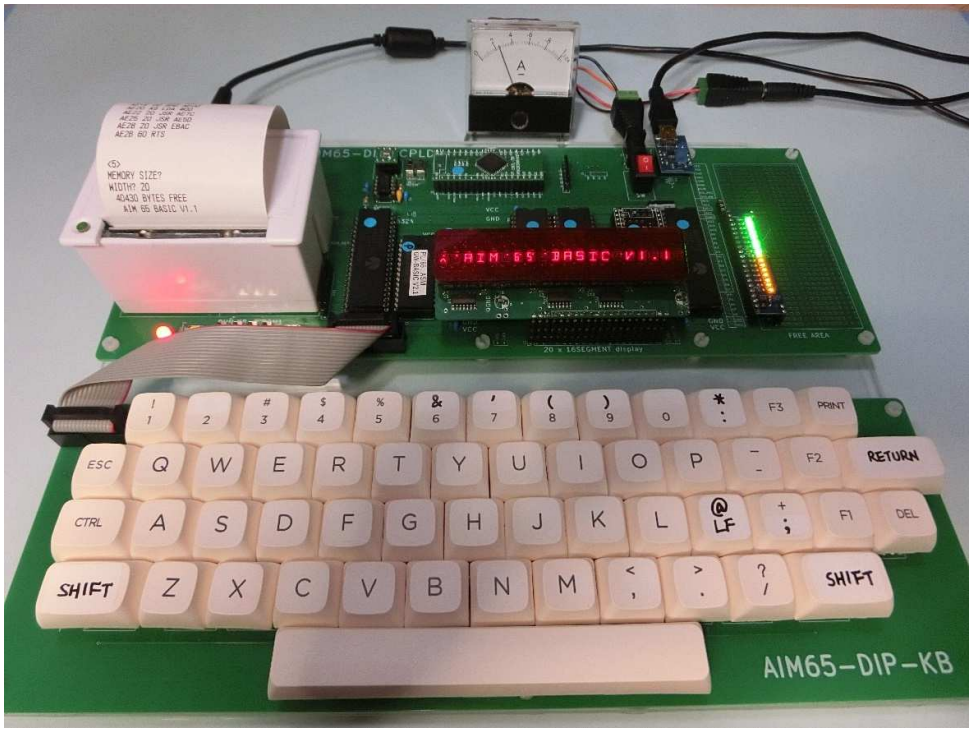


ckwell
ernational

ROCKWELL INTERNATIONAL CORPORATION
ELECTRONIC DEVICES DIVISION
4311 JAMBOREE ROAD, NEWPORT BEACH, CA 92660
Attn: MC501-300

5 THE AIM65-DIP-CPLD

The AIM65-DIP-CPLD was produced with the idea of cloning a CMOS version that can operate at high speed and consumes less power. You can use R65C02P2 for CPU, R65C22P2 for VIA, and HD63B21 for PIA, but RIOT R6532A is still an NMOS product. With this, we could only have products up to 2MHz, and the upper limit of the operating speed was 2MHz. In addition, PIA has replaced it with a CPLD that operates at 5V to add a serial TX pin for the printer.



Reproduction AIM65-DIP-CPLD with 5V operation

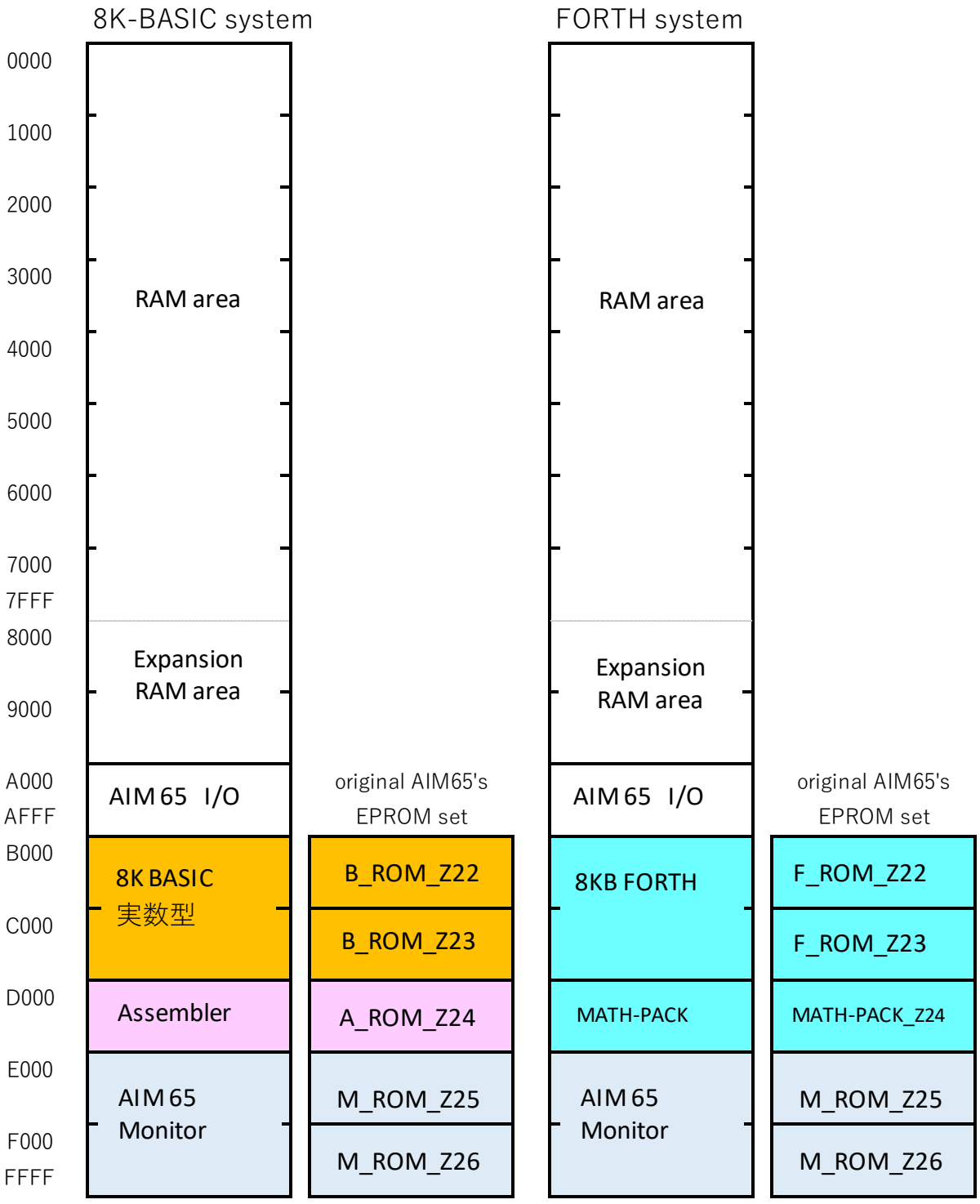
After that, in order to increase speed a printer interface was created that does not depend on the system clock.
A memory map switching function was created with a CPLD with a relatively large number of macrocells, leading to the creation of the 3.3V system AIM65-CPLD-3v3.
In this document, we will explain the 3.3V reproduction AIM65-CPLD-3v3 using the Western Design Center W65C02S CPU and W65C22S VIA.

First, let's have a look at the memory maps.

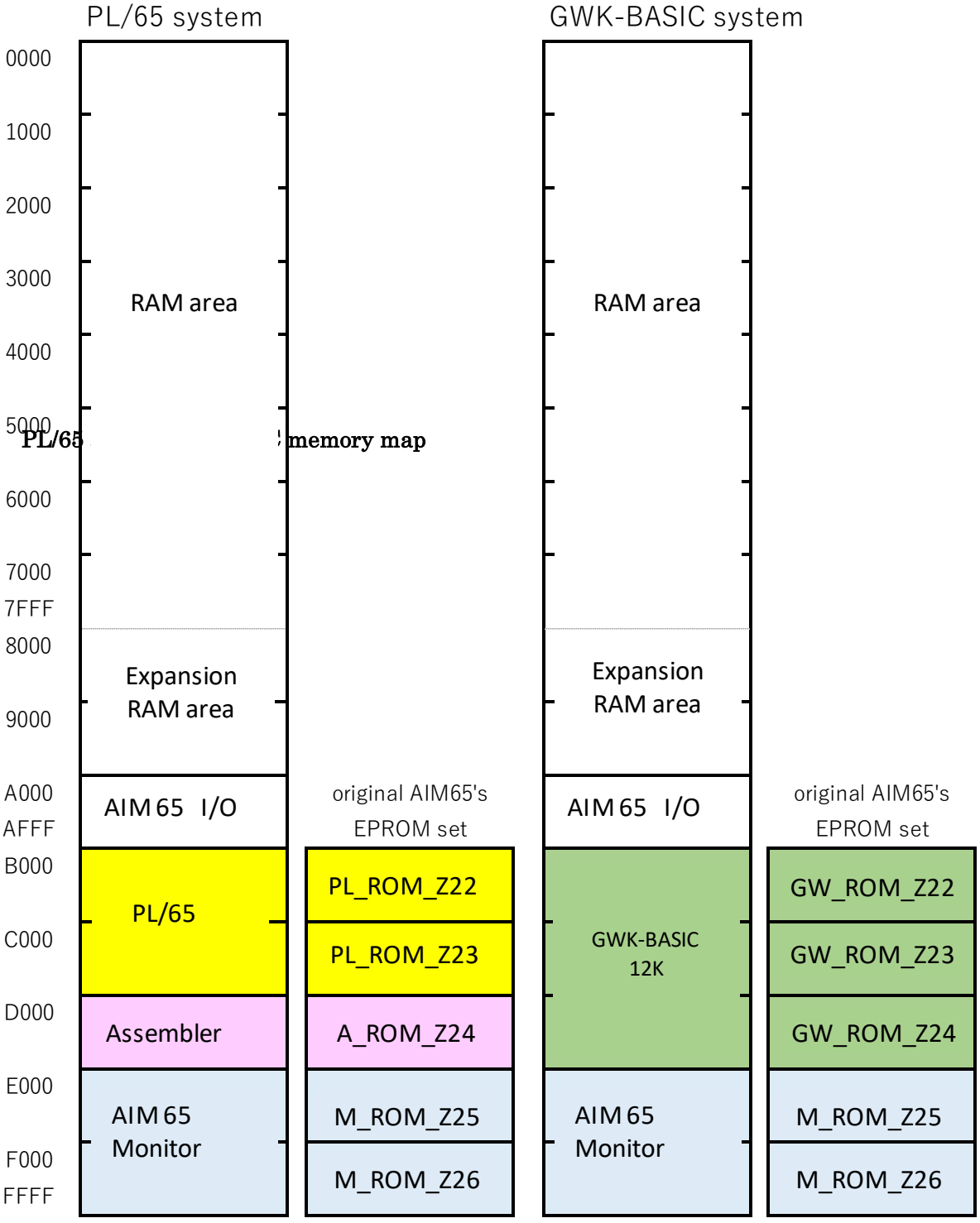
5.1 Memory Map

Reading from the user's manual and software manual, the AIM 65 can be used with the following memory maps.

BASIC and FORTH memory map

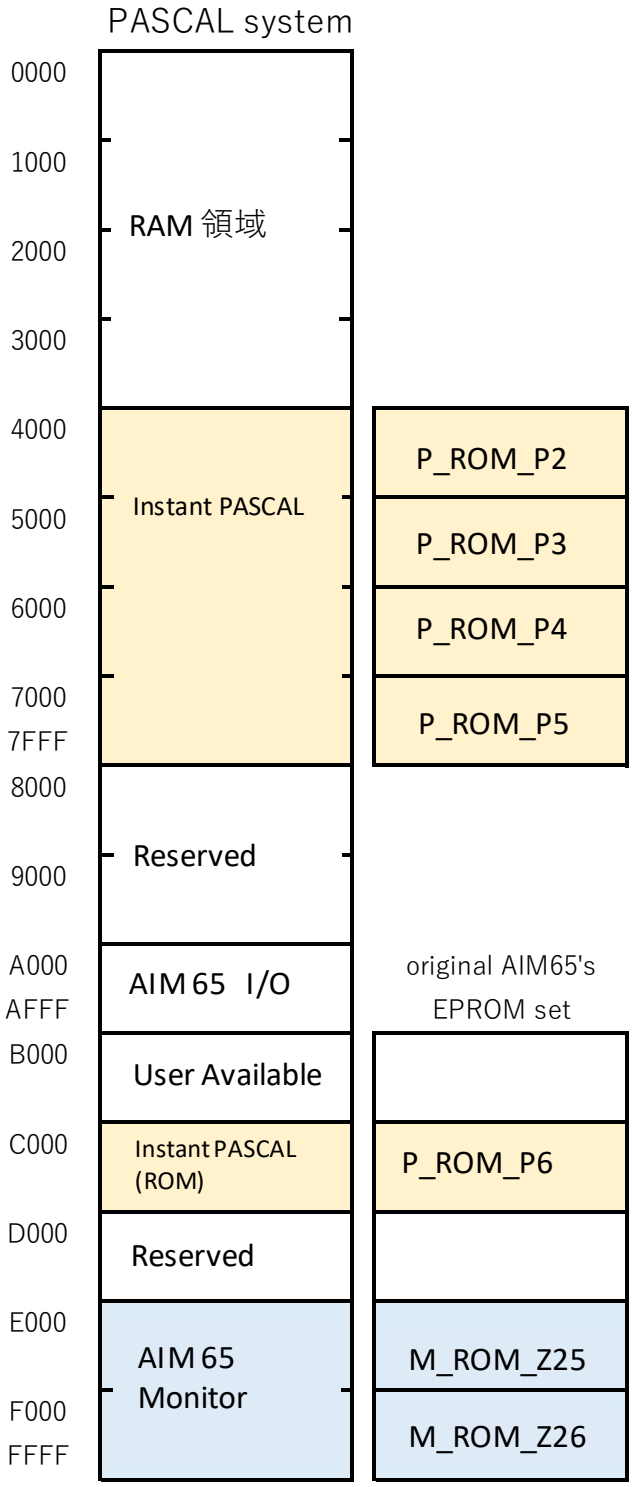


PL/65 and GWK-Basic memory map



PL/65 memory map

Memory map for Instant PASCAL



Switching between memory maps

The original AIM65 was equipped with five 4KB EPROMs (total 20KB), and by replacing these EPROMs, multiple programming languages could be switched and used. Of course, when replacing, it is necessary to turn off the power and replace the EPROM.

In the reproduced version AIM65-CPLD-3V3, multiple programming languages can be switched and used by setting the DIP switch on the board, which is equipped with 512KB capacity FLASH memory. BASIC+ASM, FORTH+MATH, PL/65+ASM can be selected by setting the DIP switch on the FLASH memory board.
Only Instant PASCAL uses a different memory area, so in addition to selecting the DIP switch on the FLASH memory board, it is necessary to switch $4000 to $7FFF to the ROM area by setting the DIP switch on the control CPLD board. If you use the RAM area instead of the ROM, you must load the Pascal software into the RAM in advance. You can switch while power is supplied, but after the switch you need to press the RESET button.

Some notes about the software

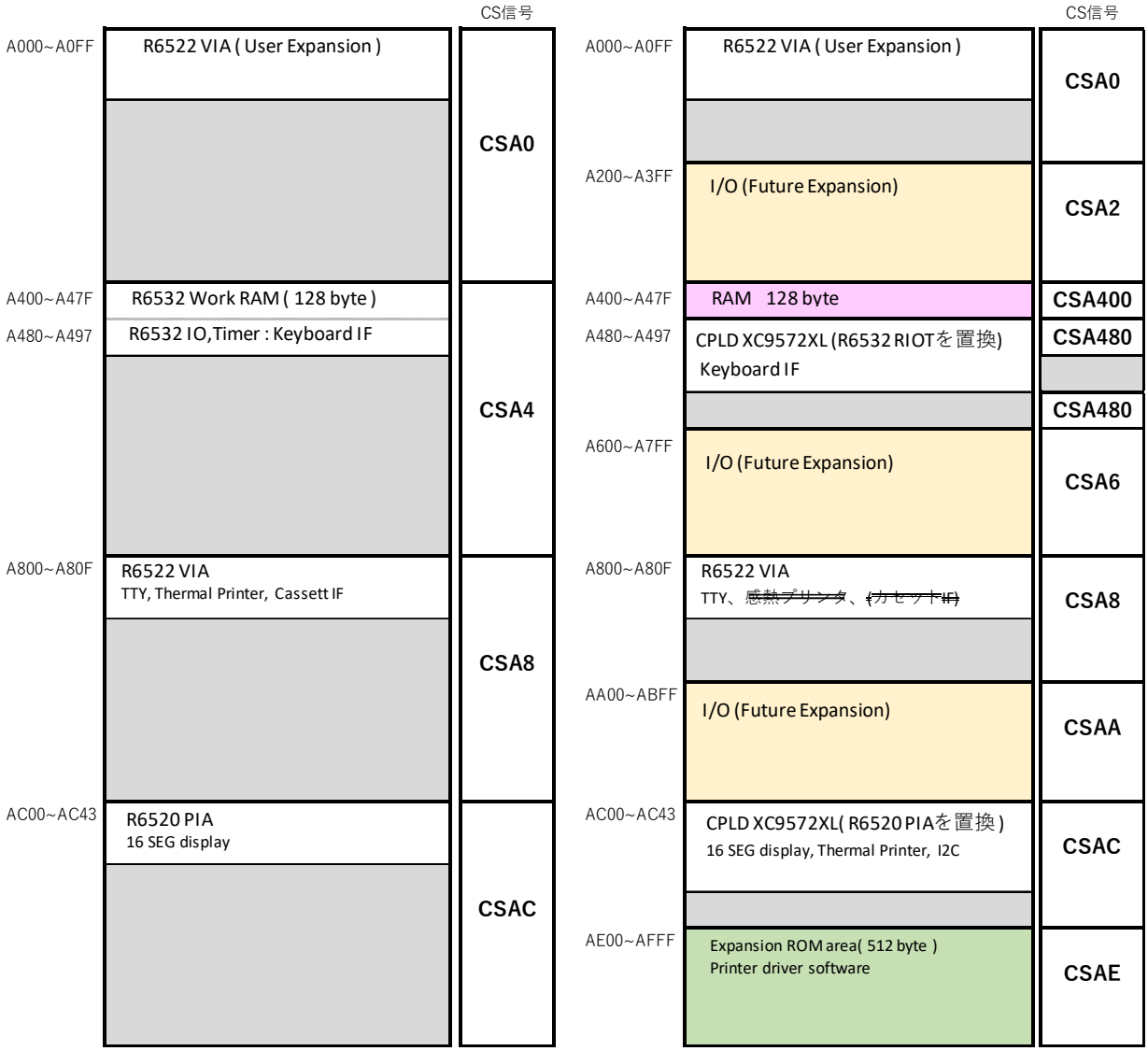
- BASIC and FORTH can be used normally. I can't help but feel that the specifications are outdated or that the functions are lacking.
- MATHPACK, which is an extended word of FORTH, can be started by pressing the N key and can also display words, but I have not obtained a manual and do not know how to use it. If there is a manual, it seems that you can make a complex number calculator.
- PL/65 was able to start, and I tried to compile a program but it has not possible to create a program. There is a manual for PL/65, but it seems to contain quite a few errors.
- Instant PASCAL was able to start. but it's not working at all yet. The manual is a low-quality scan, a good quality manual will be welcome.

5.2 I/O map

In the AIM 65, the I/O area is allocated to $A000-AFFF.
In the original version

- CSA0 : for user extension - VIA R6522
- CSA4 : Keyboard Interface, Cassette Timer - RIOT R6532
- CSA8 : TTY, Thermal Printer, Cassette IF - VIA R6522
- CSAC : 20 digits x 16 segments LED – PIA R6520

Original AIM 65 I/O map Figure　　I/O map of AIM65-CPLD-3V3



In the reproduction AIM 65, by subdividing each area and decoding the addresses, new peripheral devices can be allocated. CSA0, CSA8, and CSAC have half the address range of the original, but software compatibility is maintained.

CSA4 is divided into two, CSA400 and CSA480, and CSA400 is mapped to SRAM. When access to the RAM in the R6532 RIOT is detected by the address decoder built into the control CPLD, and the chip select is switched to the main SRAM. In other words, out of KB mode and TTY mode, if only TTY mode is used, operation is possible without RIOT.
The RIOT's Timer is used to check the startup of the thermal printer when the power is turned on and to measure the pulse width of the cassette interface.

Since the exact same thermal printer as the original is not available it is replaced it with a thermal printer that can be controlled by a UART TX signal. Therefore, when replacing the PIA assigned to CSAC with a CPLD that operates at 3.3V, I also built in a UART-TX and connected the thermal printer to it.

A software patch is also required to replace the thermal printer, but the 512-byte area of CSAE is allocated to the ROM or RAM area instead of I/O, and the driver for the thermal printer is arranged. The 512 bytes of CSAE operate as a RAM area during software development, and a DIP-SW is placed on the control CPLD board so that it can be switched to the ROM area after development is completed.

In the AIM65-CPLD-3V3, all address decoding is performed by the control CPLD, so by rewriting the contents of the CPLD, I/O can be placed at any address without hardware changes.

5.3 Power supply voltages

The reproduction AIM 65 (AIM65-CPLD-3V3) basically operates at 3.3V. The 16-segment LED operates at 5V, but the control signal is TTL level (VIH = 2.0V), so the PIA side can be interfaced with a 3.3V CMOS output. Thermal printers require a 9V or 12V power supply, but can also be driven at 3.3V CMOS levels because the UART control signals are at TTL levels.

3.3V operation products must be used for the oscillator module, reset generation circuit (NE555), CPU, ROM, RAM, VIA, PIA, RIOT, address decode and other control circuits.

Therefore I have chosen for:

- CPU: W65C02S
- VIA : W65C22S
- PIA : Replaced by CPLD XC9572XL (Xilinx)
- RIOT : Replaced with CPLD XC9572XL (Xilinx)
- Clock division, address decoding, bus control: CPLD XC9572XL (Xilinx)

5.4 Clocks

The system clock is called PH2 (Phase 2 Clock). In early NMOS and CMOS CPUs, the logic was designed using PH1 and PH2, which are two-phase non-overlapping clocks that utilize the characteristics of MOS transistors. And there was a remnant of that in the printed circuit board design. Currently, logic design with a single-phase clock is common, the name of PHI2 remains.

In the original AIM65 the system clock is PH2 and VIA, PIA and RIOT peripherals are also running at PH2. In AIM65 REPRODUCTION (AIM65-CPLD-3V3), the system clock is PH2 and named SYS_PH2. AIM65-CPLD-3V3 also introduces SL_PH2 (called slow PH2, which is a clock obtained by dividing PH2 by 4) to operate the peripherals of VIA, PIA, and RIOT at 1/4 the clock of CPU, ROM, and RAM. (It is also possible to operate with SYS_PH2 = SL_PH2 at low speed by setting the DIP switch of the CPLD for control).

AIM65-CPLD-3V3 uses Western Design Center's W65C02S and can operate at 3.3V, 8MHz. The CPU and ROM/RAM are running at 8MHz, and peripheral devices at 2MHz. If you change the oscillation module to 12MHz or 16MHz for experiments, you can operate at 12MHz/3MHz or 16MHz/4MHz. In fact, I usually do my development work at 16MHz/4MHz. Experiment with this overdrive at your own risk.

	AIM 65 Original product	AIM 65 reproduction AIM65-CPLD-3V3
SYSTEMCLOCK **SYS_PH2**	1MHz	8 MHz
SLOW CLOCK **SL_PH2**		2MHz

5.5 About the configuration of the control logic part

As a method of configuring a clock generation circuit, a read/write signal generation circuit, and an address decoder can be made with

1) individual TTL/CMOS logic
2) Burned into GAL (PLD)
3) Built into CPLD

There are other methods such as an FPGA.

For the size of the AIM 65, it's probably time to burn everything, including the CPU, into an FPGA or emulate it with a high-performance CPU, but what made me decide to make a reproduction was, "It seems that parts are still available" was the trigger, so I would like to take emulation with FPGA and high-performance CPU as a next step.

I decided to use a 3.3V operation board and create a control circuit with a CPLD so that I could extend the lifetime even a little in the future considering flexibility and performance. Still, 3.3V CPLDs (such as the XC9572XL) are already in the older technology category.

5.6 AIM65-CPLD-3v3 main board schematic and board layout

The next figure shows the circuit diagram of the main board of the AIM65-CPLD-3V3.

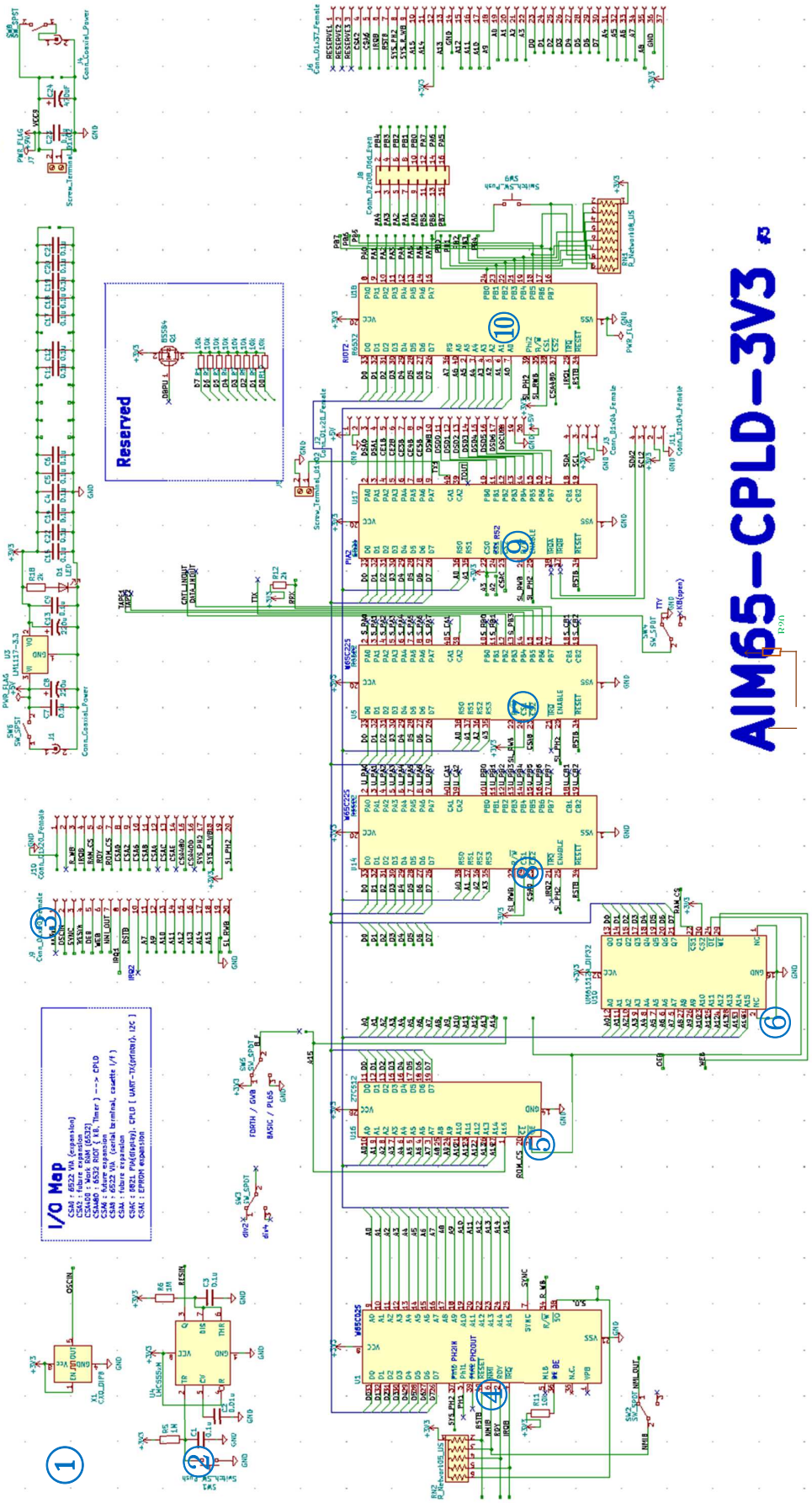
The main parts consist of:

- ① Oscillation module - 8MHz, 12MHz, 16MHz, etc. can be exchanged by socket mounting.
- ② Reset circuit - using NE555
- (3) CPLD for system control (XC9572XL clock division, address decoding, wait circuit)
- ④ CPU W65C02S
- ⑤ ROM SST39LF040 or SST39VF040 (Microchip Technology FLASH memory)
- ⑥ RAM AS7C34096A-10 (ALLIANCE) (AS6C4008-55PCN is also acceptable up to 8MHz)
- ⑦ Peripheral chip W65C22S system VIA
- ⑧ Peripheral chip W65C22S VIA for user expansion
- ⑨ Peripheral chip XC9572XL CPLD (replacement of R6520 PIA 3.3V compatible, printer IF)
- ⑩ Peripheral chip XC9572XL CPLD (replacement of R6532 RIOT, 3.3V compatible)

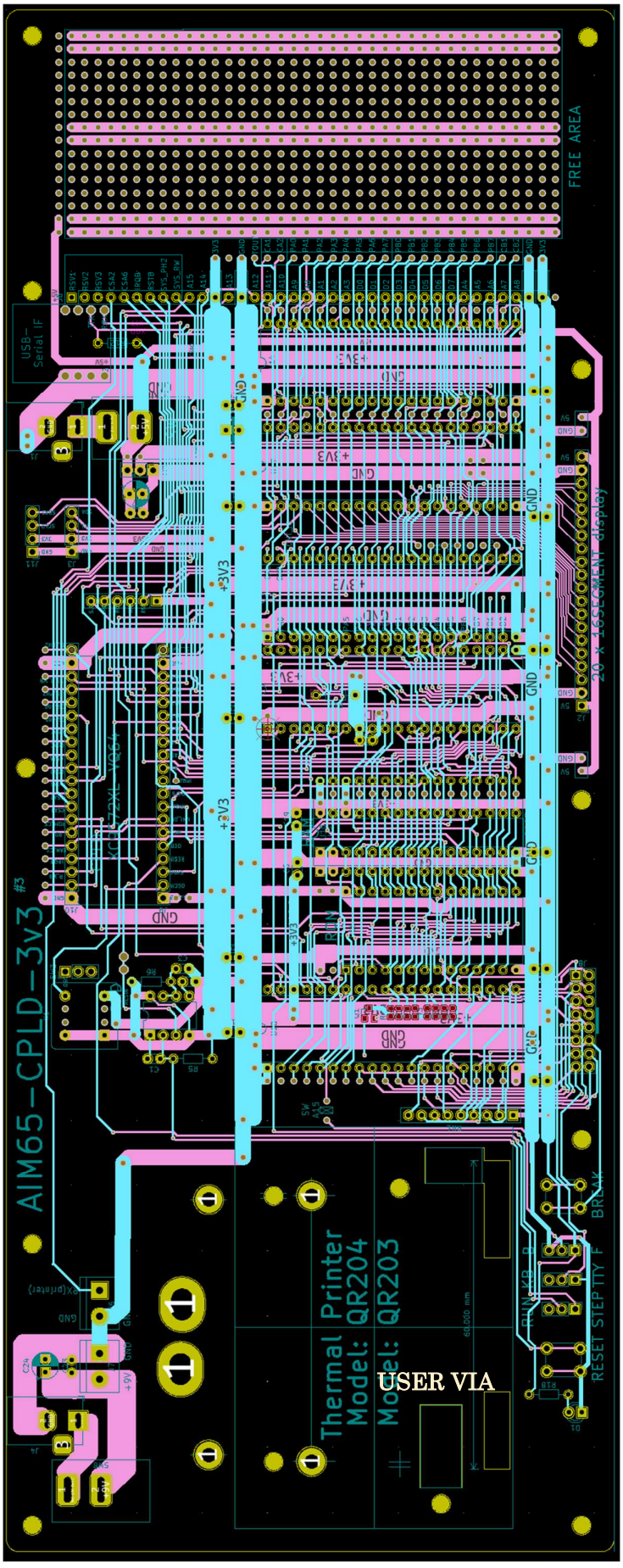
Please refer to the table for details.

Main boar

circuit dia



Main board
layout diagram



Main board
parts layout

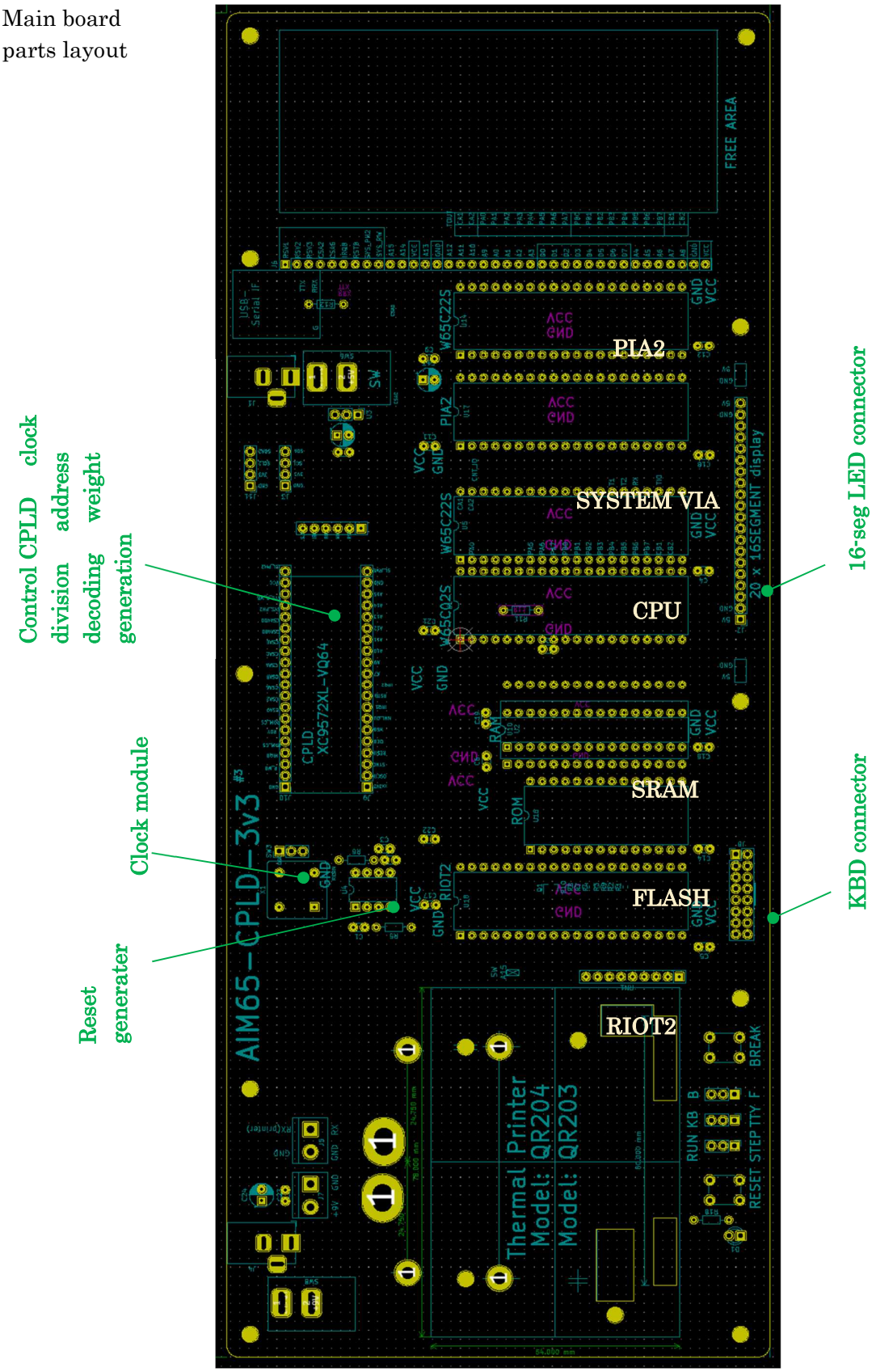


Table AIM65-CPLD- 3V3 main board parts list (1/2)

Reference	Value	Note	Footprint
C13	220u	Power smoothing	Capacitor_THT:CP_Radial_D5.0mm_P2.50mm
C23	0.1u	Bypass printer powersupply	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C24	1000uF	Power smoothing	Capacitor_THT:CP_Radial_D5.0mm_P2.50mm
C1	0.1u	NE555	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C2	0.01u	NE555	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C3	0.1u	NE555	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C6	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C11	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C14	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C15	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C19	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C20	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C21	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C22	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C4	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C5	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C17	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C18	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C9	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C12	0.1u	Bypass	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C7	Unknown	LT1117 Mounted on the module	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C8	Unknown	LT1117 Mounted on the module	Capacitor_THT:CP_Radial_D5.0mm_P2.50mm
D1	LED (Built-in resistor)	Power	LED_THT:LED_D3.0mm
J1	Conn_Coaxial_Power	5V Power supply	Connector_BarrelJack:BarrelJack_Horizontal
J4	Conn_Coaxial_Power	Printer power supply	Connector_BarrelJack:BarrelJack_Horizontal
J11	Conn_01x04_Female	I2C device connector	PinHeader_1x04_P2.54mm_Vertical

J2	Conn_01x20_Female	16-segment LED connection	PinHeader_1x20_P2.54mm_Vertical
J3	Conn_01x04_Female	Reserved for I2C devices	PinHeader_1x04_P2.54mm_Vertical
J5	Screw_Terminal_01x02	Printer TX signal	TerminalBlock:TerminalBlock_bornier-2_P5.08mm
J7	Screw_Terminal_01x02	For printer power connection	TerminalBlock:TerminalBlock_bornier-2_P5.08mm
J6	Conn_01x37_Female	Expansion bus signal	AIM65:PinHeader_1x37_P2.54mm_s_hole
J8	Conn_2x08_Odd_Even	Key_board connector	PinHeader_2x08_P2.54mm_Vertical
J9	Conn_01x20_Female	1/2 for control CPLD	PinHeader_1x20_P2.54mm_Vertical
J10	Conn_01x20_Female	2/2 for control CPLD	PinHeader_1x20_P2.54mm_Vertical
J20	Conn_01x26_Female	For user VIA signals	A through-hole PAD is placed on the layout although it is not shown in the circuit diagram.
R11	10k	Pull-up for BE signal	R_Axial_L3.6mm_D1.6mm_P7.62mm_Horizontal
R12	2k	Pullup for UART_RX	R_Axial_L3.6mm_D1.6mm_P7.62mm_Horizontal
R18	0 Ω	Built-in Resistor	R_Axial_L3.6mm_D1.6mm_P7.62mm_Horizontal
R5	1M	NE555	R_Axial_L3.6mm_D1.6mm_P7.62mm_Horizontal
R6	1M	NE555	R_Axial_L3.6mm_D1.6mm_P7.62mm_Horizontal
R20	8.2k or 9.1k or 10k	For setting KB mode, Not in the circuit diagram but needs to be added	R_Axial_L3.6mm_D1.6mm_P7.62mm_Horizontal
RN1	R_Network08	Resistor pack 9 pin	Resistor_THT:R_Array_SIP9
RN2	R_Network05	Resistor pack 5 pin	Resistor_THT:R_Array_SIP6

Table AIM65-CPLD-3V3 main board parts list (2/2)

Reference	Value	Note	Footprint
SW1	Switch_SW_Push	Tactile switch for reset	Button_Switch_THT:SW_PUSH_6mm
SW2	SW_SPDT	RUN/STEP selector slide switch	PinHeader_1x03_P2.54mm_Vertical
SW4	SW_SPDT	TTY/KB mode changeover slide switch	PinHeader_1x03_P2.54mm_Vertical
SW6	SW_SPST	Main power switch	AIM65:switch_pwr
SW8	SW_SPST	Main power switch	AIM65: switch_pwr
SW3	不要	(SW_SPDT 予備パターン)	PinHeader_1x03_P2.54mm_Vertical
SW5	不要	(SW_SPDT 予備)	PinHeader_1x03_P2.54mm_Vertical
U1	CPU:	W65C02S6TPG-14	AIM65:DIP-40_W15.24mm_s_hole
U5	VIA:	W65C22S6TPG-14, for system	AIM65:DIP-40_W15.24mm_s_hole
U14	VIA:	W65C22S6TPG—14 for user	AIM65:DIP-40_W15.24mm_s_hole
U18	RIOT2:	XC9572XL-VQ64, DIP40 conversion board	AIM65:DIP-40_W15.24mm_s_hole
U17	PIA2: DISP,PTR	XC9572XL-VQ64, DIP40 conversion board	AIM65:DIP-40_W15.24mm_s_hole
U16	FLASH.	SST39VF040/ 39LF040, DIP32 conversion board	AIM65:DIP-32_W15.24mm_s_hole
U10	SRAM.	AS7C34096A-10 DIP32 conversion board	AIM65:DIP-32_W17.78mm_smallpad
U3	LM1117-3.3	Mounted in modules	PinHeader_1x03_P2.54mm_Vertical
U4	LMC555xM,	3.3V compatible product	Package_DIP:DIP-8_W7.62mm
X1	CXO_DIP8	oscillatier module	Oscillator:Oscillator_DIP-8, DIPconversion board
	USB serial	USB serial conversion module (using CH340E)	
	main board	PCB AIM65-CPLD-3V3	
	plastic screws	M3x4mm 10 本	
	plastic nut	M3 x 6mm 10 ケ	

Parts required to operate the AIM65-CPLD-3V3

Reference	Product name	Note	
	16-seg LED	HPDL-1414 x 5 Module mounted	
	Printer	QR20 orQR203	
	keyboard	QWERTY keyboard (including screws, spacers and connection cables)	
	AC adapter 5V	AC adapter 5V, 2A	
	AC adapter 9V	AC adapter 9V, 2A	
	USB cable, mini-B	USB cable, mini-B	

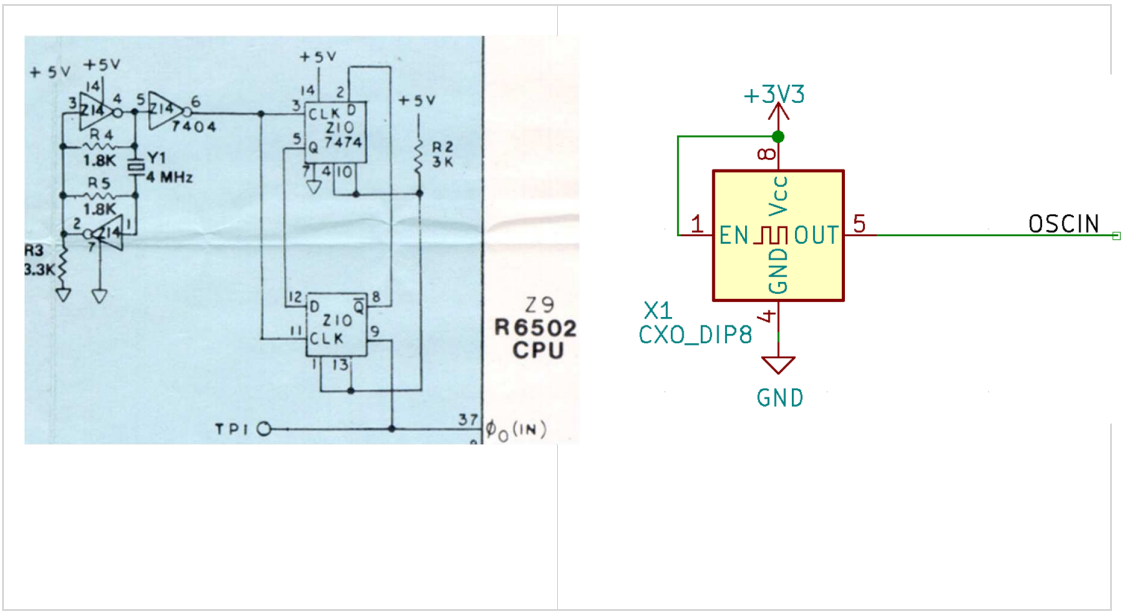
Parts without shading are essential parts for the minimum configuration.

Light green shaded parts are required for the basic configuration

Light orange shading is for extension to FULL system

This section explains the configuration of each part of the main board circuit diagram in the figures.

5.7 Clock generation circuit



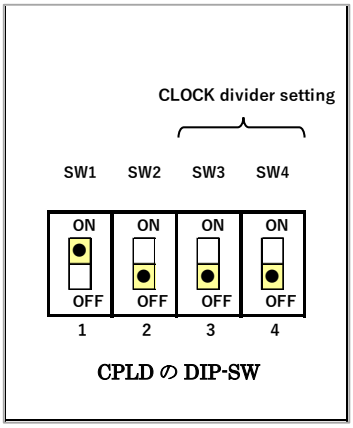
In the original AIM 65, a clock with a duty of 50% is created by dividing the output of the oscillator by the inverter of the crystal oscillator and a Schmitt trigger by a D flipflop. After inputting to Φ0 of the CPU, the CPU outputs as Φ1 and Φ2 (PH2). PH2 is 1MHz.

In the AIM65-CPLD-3V3, the output OSCIN of the crystal oscillator module is input to the control CPLD, and the CPLD generates a clock based on the division setting of the DIP-SW.

Operation clock setting (when using 8MHz oscillation module)

SW3 CKSEL1	SW4 CKSEL0	分周
OFF	OFF	div1
OFF	ON	div2
ON	OFF	div4
ON	ON	div8

SYS_PH2	SL_PH2
8MHz	2MHz
4MHz	1MHz
2MHz	2MHz
1MHz	1MHz

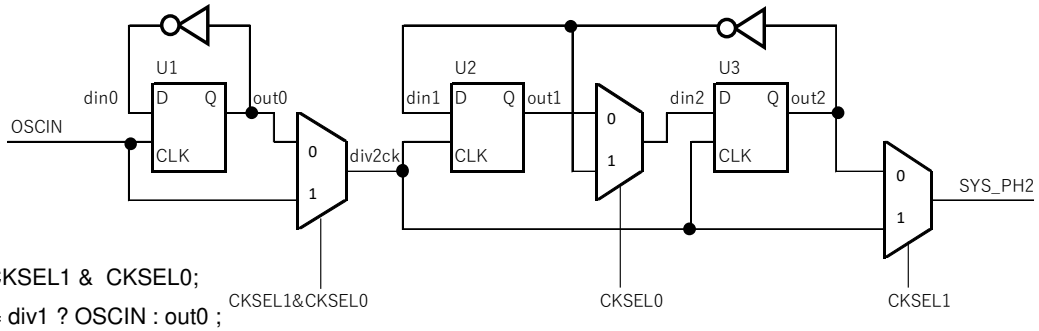


Clock division

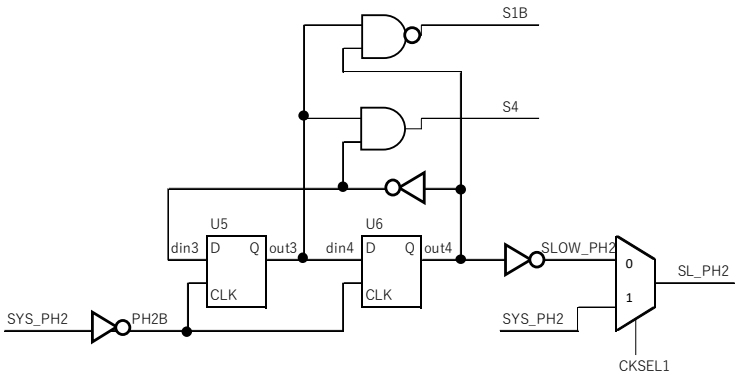
A Verilog description of the control CPLD clock division and an equivalent logic circuit diagram are shown. It's not very elegant description, but please forgive me. The actual circuit will depend on the compilation and optimization of the ISE tool, so please consider it as a reference diagram. SYS_PH2 is the system clock, SL_PH2 is the slow clock, and defines states (S1, S2, S3, S4) for wait generation.

Verilog-HDL description of clock division and equivalent circuit

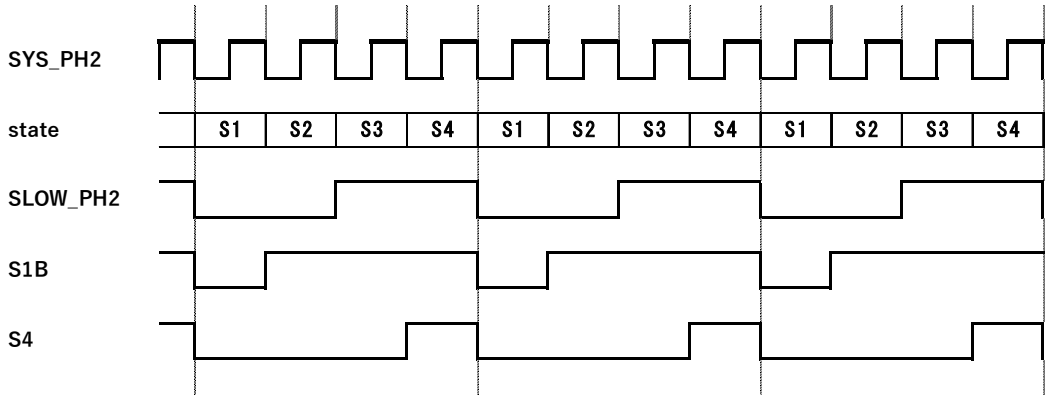
```
wire din0, din1, din2, div2ck, out0, out1, out2;
wire din3, din4, out3, out4, S4, S1B;
wire div1;
```




```
assign div1 = CKSEL1 &  CKSEL0;
assign div2ck = div1 ? OSCIN : out0 ;
assign din0 = ~out0 ;
assign din1 = ~out2;
assign din2 = CKSEL0 ? ~out2 :  out1 ;
DFF U1( .D(din0), .CLK(OSCIN),  .Q(out0) );
DFF U2( .D(din1), .CLK(div2ck), .Q(out1) );
DFF U3( .D(din2), .CLK(div2ck), .Q(out2) );
assign SYS_PH2 = CKSEL1 ? div2ck : out2
assign SYS_PH22 = SYS_PH2;
assign PH2B = ~SYS_PH2;
assign din3 = ~out4;
assign din4 = out3;
assign S4 = out3 & ~out4;
assign S1B = ~( out3 & out4 );
DFF U5( .D(din3), .CLK(PH2B), .Q(out3) );
DFF U6( .D(din4), .CLK(PH2B), .Q(out4) );
assign SLOW_PH2 = ~out4;
assign SL_PH2 =  CKSEL1 ? SLOW_PH2 : SYS_PH2 ;
assign SL_PH22 = SL_PH2;
```

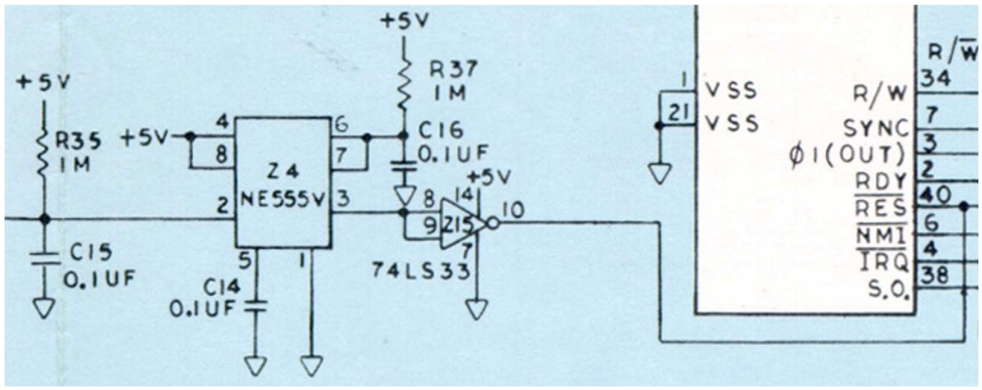


System clock generated by the clock divider

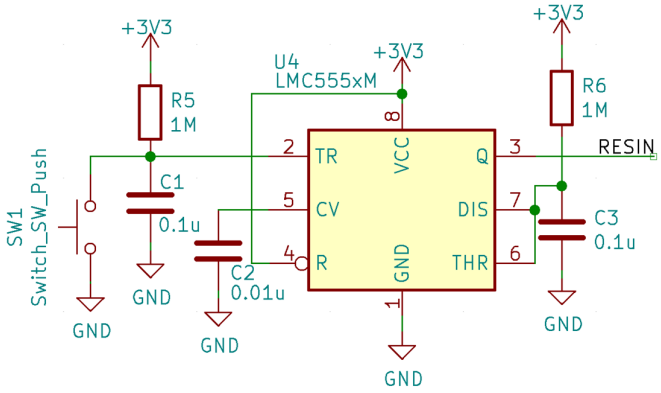


5.8 Reset circuit

Reset circuit of AIM65



AIM65-CPLD-3V3 reset pulse generator circuit



The reset pulse generator circuit of AIM65-CPLD-3V3 is almost identical to that of the original AIM 65. Generates a reset pulse whose time is determined by the time constant τ of R6 and C3 of the 555 timer IC. Same as the original design, 1M x 0.1u = 0.1 seconds. In the reproduction AIM 65, the reset pulse generated by the NE555 is passed through the D-Flipflop in the CPLD, synchronized with the slow clock SL-PH2, and then distributed. This is because when you observe the waveform with an oscilloscope during debugging, if the reset pulse is asynchronous to the clock, the waveform will be distorted and you will not be able to observe a constant waveform. Polarity reversal of the RESIN pulse is also performed within the CPLD.

The Verilog description after receiving the reset signal (RESIN) in the CPLD is shown below. The RESIN input is synchronized to SL_PH2 through DFF and output to the RESB pin.

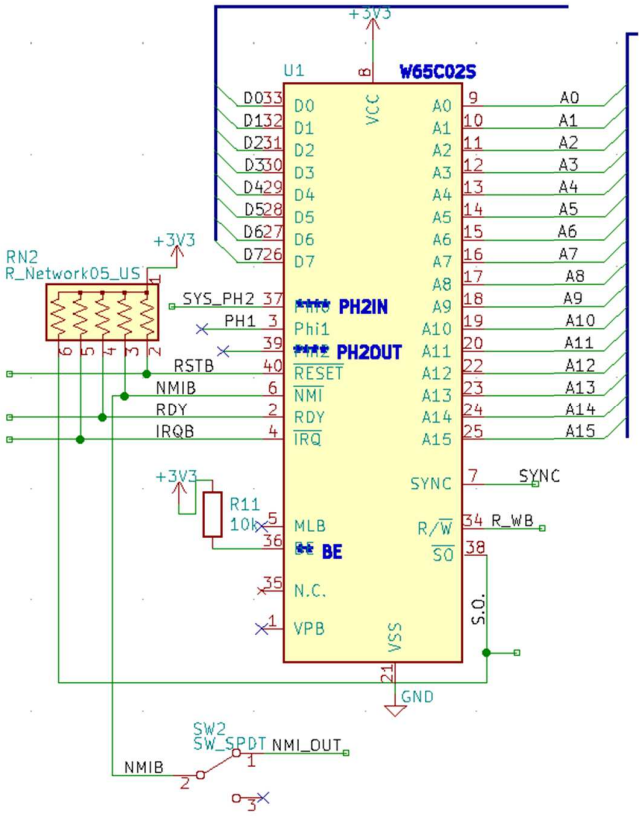
Processing of reset signal RESIN in CPLD (Verilog-HDL description))

```
wire sync_RES;
DFF U4( .D(RESIN), .CLK(SL_PH2), .Q(sync_RES) );
assign RESB = ~sync_RES;;
```

5.9 Around the CPU

The CPU is the W65C02S from Western Design Center.

CPU circuit



The W65C02S has the following features.

- The signal level is CMOS. Considering coexistence with past devices (NMOS products and TTL), it is difficult to use a 5V power supply, but on the contrary, it is easier to transition to 3.3V.
- All AC timings of W65C02S are specified for PH2IN. PH2OUT and PH1OUT are not used in AIM65-CPLD-3V3 because they exist only for the purpose of maintaining compatibility with past designs in the low-speed region.
- RDY, IRQ, and MNI are input signals for the CPU, but in the past, peripheral devices on the output side made input signals to the CPU using wired OR and pull-up with open-drain outputs. However, the open-drain type is not suitable for high-speed operation, so I think it is better to drive the high side as well. The RDY signal is especially important because it must be controlled in half-clock increments. In fact, the W65C02S RDY signal is a bi-directional signal, and the CPU outputs "LOW" while the WAI instruction is being executed. However, AIM65, which started from the era of NMOS R6502, does not use the

WAI instruction, and AIM65-CPLD-3V3 push-pull drives the RDY signal with CPLD to increase speed.

- MLB (Memory Lock: additional signal) is a signal that controls not to pass the right to use the bus during read-modify-write operations of memory and I/O, but it is not used in AIM65-CPLD-3V3.
- BE (bus enable) is fixed to HIGH by a pull-up. When this signal is "LOW" input, the CPU asynchronously sets the address bus, data bus, and R/W output to HIGH-Z. Although it is not used in AIM65-CPLD-3V3, if you want to access memory or I/O devices using the DMA function in the future, set the RDY input to "LOW" to wait for the CPU and then "BE". LOW" to get the bus right

5.10 Control CPLD logic

The next figures show the circuit diagram and layout of the control CPLD board. Verilog-HDL description of CPLD is shown in LIST○○ to LIST○○.

The control CPLD is in charge of clock division, address decoding, ROM/RAM access signal generation, and wait generation. The generation logic of each signal is in the form of text that is almost the same as the circuit diagram by the structure description of Verilog-HDL. It is compiled with the Xilinx (currently AMD) ISE tool (Note 1) and burned into the XC9572XL-VQG64.

The clock division specification and reset signal have already been explained, but the address decoding, wait generation, MNI generation, and IRQ processing will be explained in comparison with the original AIM65.

Interrupt signal IRQ

IRQ1B is connected to RIOT's IRQ, but since IRQ is not generated by RIOT2's CPLD, it is currently only a pull-up (R9).

IRQ2B is a pin for connecting to IRQ of USER VIA, but it is disconnected on the main board before connection, and only a pull-up (R19) is connected. When using the USER VIA interrupt, it is necessary to make a jumper connection on the main board.

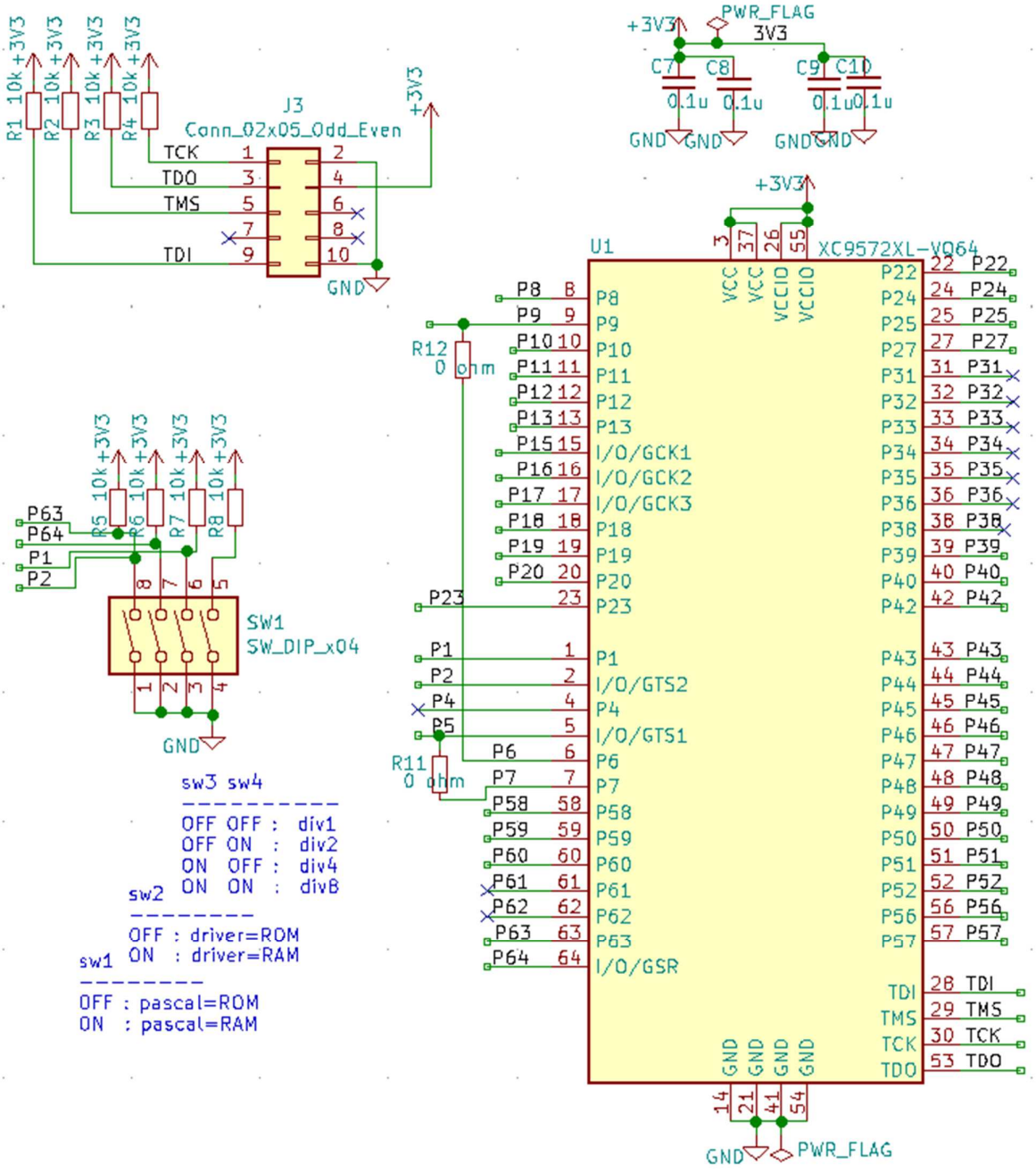
IRQ1B and IRQ2B are ANDed (negative logic OR) and output to IRQB (P24), which is connected to the CPU, but, as mentioned above, currently no interrupts are generated.

Verilog-HDL description of IRQ

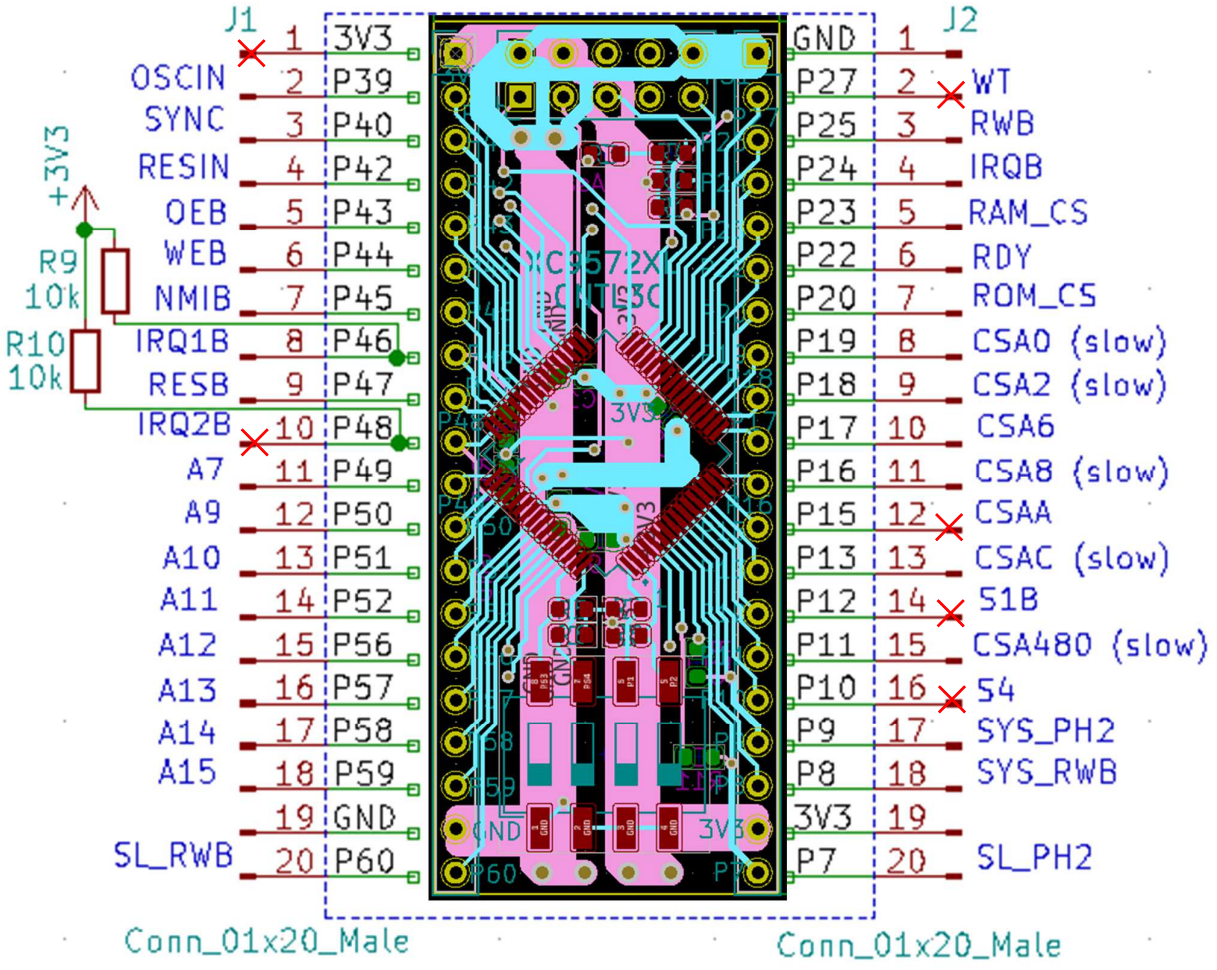
```
assign IRQB = IRQ1B & IRQ2B ;
```

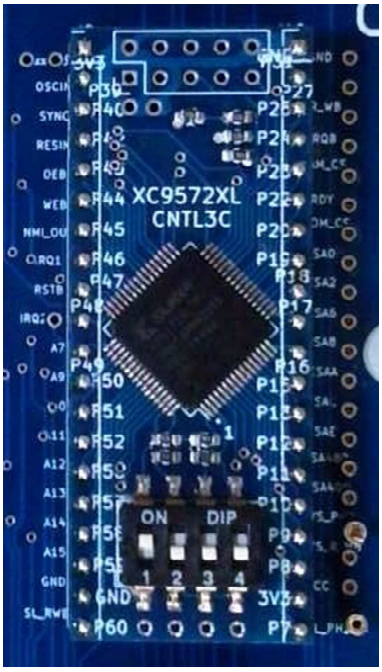
(Note 1) There were no new releases of Xilinx ISE after 2013, it has been migrated to the Vivado Design Suite. It is said that the tools are organized/integrated and the memory usage is more efficient, but I got stuck in various ways when installing ISE WebPack Edition (free version), and finally I was not able to use it. I'm not in the mood to move to Vivado any time soon. I plan to migrate when I need to use the next new PC.

Control CPLD board circuit diagram



Control CPLD DIP conversion board connection diagram and layout diagram





✕ :メインボード上で接続されていないことを表しています。

Improving the drive capability of the system clock

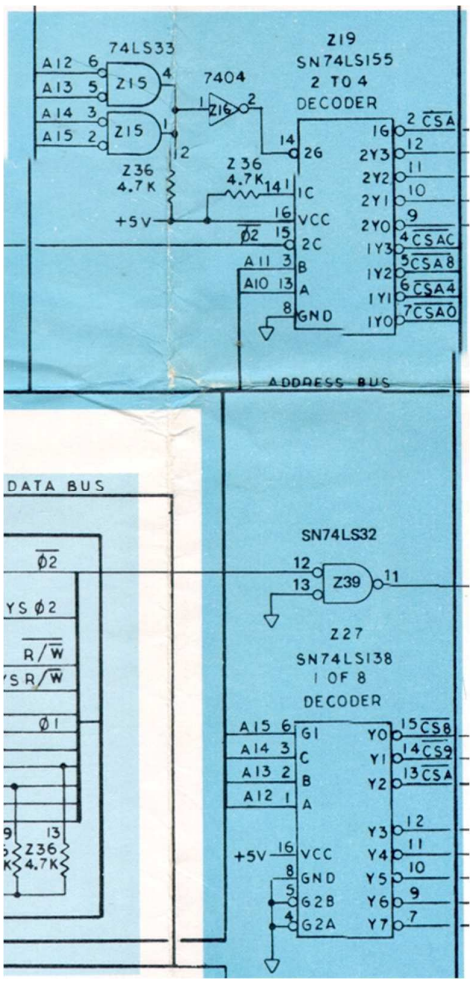
In order to increase the drive capability of SYS_PH2 (P9) and SL_PH2 (P7), I tried outputting the same signal to SYS_PH22 (P6) and SL_PH22 (P5) and driving them in parallel connection, but there was no change (the upper frequency limit was the same), so I didn't connect them in parallel. Regarding the drive capacity of the output terminal, I initially thought that the drive capacity could be specified in the CPLD's .ucf file, but I could not find a way to do it.

Peripheral device address decoding

The address map of AIM65-CPLD-3V3 is shown in the figures. Also, the I/O map is as shown in the figure.

Figure ○○ shows the address decoding circuit of the original AIM 65 peripheral device.

Original AIM65 Peripheral I/O Address Decoder



The AIM65-CPLD-3V3 address decoder is written so that the original AIM65 peripheral devices can be used unchanged in software. That is, the CSA0, CSA4, CSA8, and CSAC addresses match the original (the range is half of the original).

CSA2, CSA6, CSAA and CSAE are areas for expansion. Of these, 512 bytes of CSAE are mapped to FLASH or RAM as an area for arranging driver software for thermal printers.。

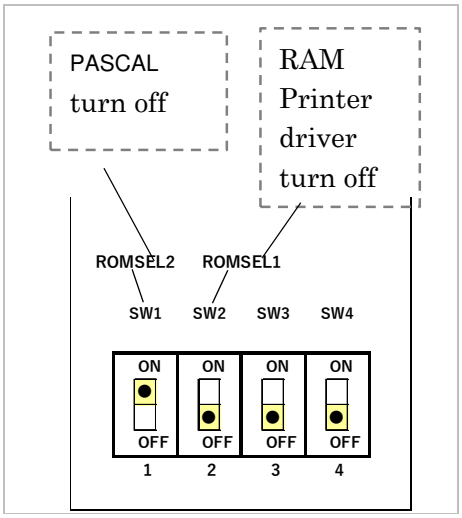
0000 - 9FFF RAM ,

A000 - AFFF I/O

B000 - FFFF ROM

Exceptions are:

- CSA400 (A400 - A47F) is switched to the RAM area (RIOT's internal RAM is not used).
- CSAE (AE00 - AEFF) should be RAM or ROM area (selected by SW2 of DIP-SW).
- 4000 - 7FFF of the RAM area can be switched to ROM for PASCAL execution (selected by SW1 of DIP-SW).



Verilog-HDL description of address decoder

```
1   // ************** address decoder *********************************
2
3   assign IO_AREA = A15 & ~A14 & A13 & ~A12 ; // adr A000~AFFF
4   assign CSA0 = IO_AREA & ~A11 & ~A10 & ~A9 ;
5   assign CSA2 = IO_AREA & ~A11 & ~A10 &  A9 ;
6   assign CSA4 = IO_AREA & ~A11 &  A10 & ~A9 ;
7   assign CSA6 = IO_AREA & ~A11 &  A10 &  A9 ;
8   assign CSA8 = IO_AREA &  A11 & ~A10 & ~A9 ;
9   assign CSAA = IO_AREA &  A11 & ~A10 &  A9 ;
10  assign CSAC = IO_AREA &  A11 &  A10 & ~A9 ;
11  assign CSAE = IO_AREA &  A11 &  A10 &  A9 ;
12  assign CSA400 = CSA4 & ~A7 ;
13  assign CSA480 = CSA4 &  A7 ;
14  assign WAIT_AREA = CSA0 | CSA2 | CSA480 | CSA8 | CSAC ;
15
16  assign PASCAL_en = ROMSEL2 ;
17  assign PDRV_en   = ROMSEL1 ;
18  assign RAM_CS = ( ~A15 & ~A14 ) | ( ~A15 &  A14 & ~PASCAL_en ) |
19              (  A15 & ~A14 & ~A13 ) | CSA400 | ( CSAE & ~PDRV_en ) ;
20  assign ROM_CS = ( A15 & A14 ) | ( A15 & A13 & A12 ) | ( ~A15 & A14 & PASCAL_en ) |
21              ( CSAE & PDRV_en ) ;
22
23  assign CSA0_X    = CKSEL1 ? ~SLOW_CSA0 : ~CSA0 ;
24  assign CSA2_X    = CKSEL1 ? ~SLOW_CSA2 : ~CSA2 ;
25  assign CSA480_X  = CKSEL1 ? ~SLOW_CSA480 : ~CSA480 ;
26  assign CSA8_X    = CKSEL1 ? ~SLOW_CSA8 : ~CSA8 ;
27  assign CSAC_X    = CKSEL1 ? ~SLOW_CSAC : ~CSAC ;
28  assign CSA6_X    = ~CSA6 ;
29  assign CSAA_X    = ~CSAA ;
30  assign CSAE_X    = ~CSAE ;
31  assign RAM_CS_X  = ~RAM_CS ;
32  assign ROM_CS_X = ~ROM_CS ;
```

CSA0 to CSAE are described in positive logic so that it is easy to think in your head, and are converted to negative logic by adding _X to the part that outputs to the external terminal.

Wait generation logic

Figure XX shows the Verilog description and equivalent circuit for RDY (reversal of WAIT) generation. The structure is described almost as it is from the circuit diagram. As soon as the address is decoded and determined to be WAIT_AREA, RDY is withdrawn so that the clock division state S1 falls, and then RDY=1 is transmitted in the S4 state.

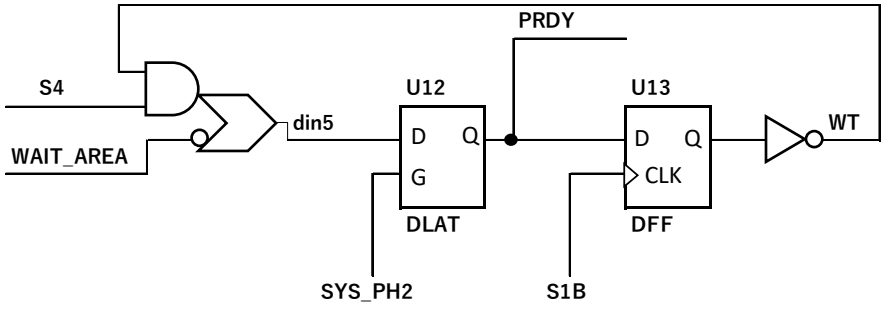
In HDL description and logic synthesis, it is recommended not to use through latches as much as possible, but I wanted to use the time until the last fall of PH2 to decode the address and generate a wait within one clock cycle. I used a slew latch.

RDY must have 15ns setup to the falling edge of SYS_PH2 input to the CPU (tPCS).

When CKSEL1 is "0", RDY is always "1". Then, SL_PH2=SYS_PH2, and the CPU and peripheral devices operate with the same clock.

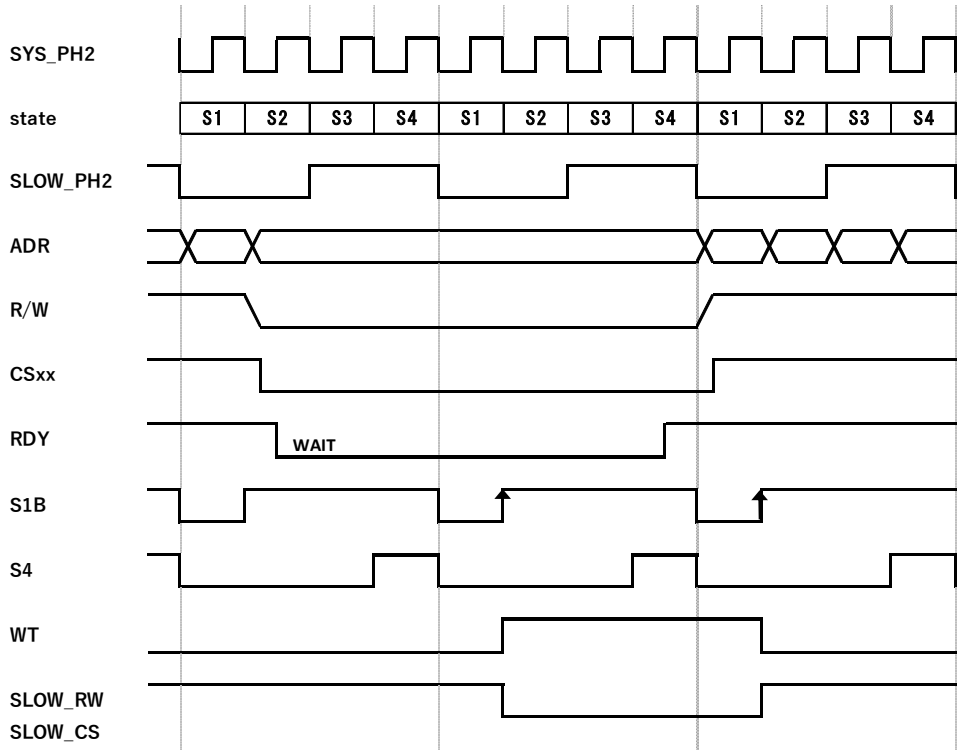
RDY-generated Verilog-HDL description and equivalent circuit

```
// ******* RDY (WAIT) generator ******************************

wire din5, din6, out5, out6, PRDY, WT;
assign din5 = ~WAIT_AREA | ( S4 & WT );
DLAT U12( .D( din5 ), .G( SYS_PH2 ), .Q( PRDY ) );
// assign din6 = PRDY | S1B ;
assign RDY = CKSEL1 ? PRDY : 1'b1 ;
DFF U13( .D( PRDY ), .CLK( S1B ), .Q( out6 ) );
assign WT = ~out6;
```

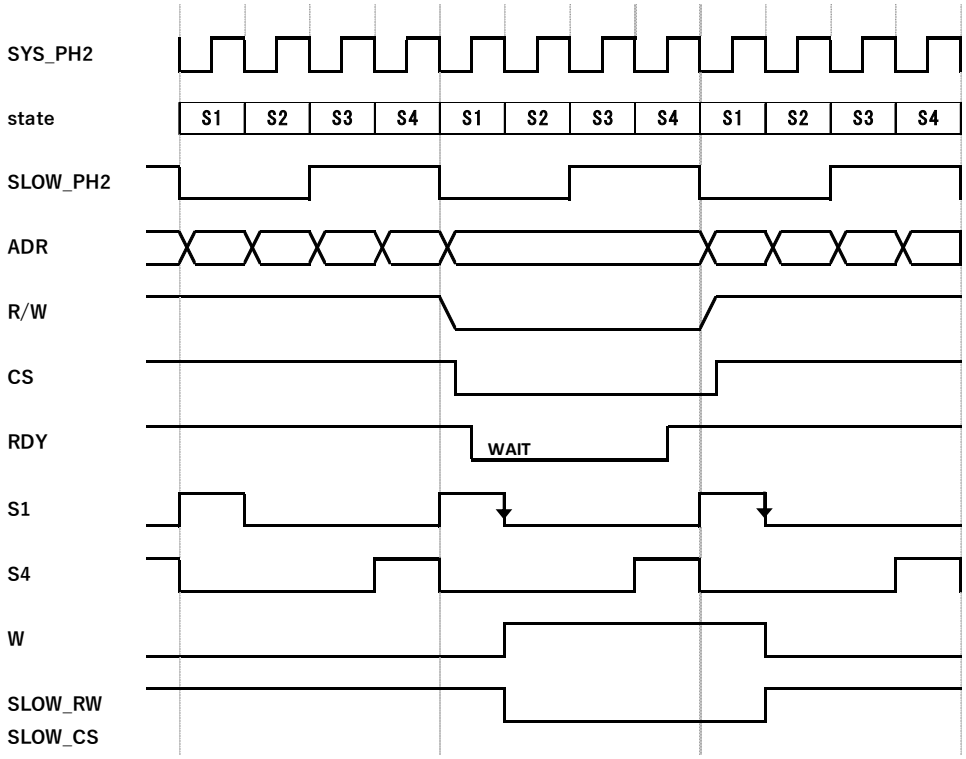


Timing charts for wait generation are shown in the next figure.
Peripheral device access must be synchronized to the slow clock. Therefore, the number of waits for the CPU varies between 3 and 6 waits depending on where the clock division state is when the bus access is started.

Wait generation timing 1. When access starts at S2, 6 waitstates



Wait generation timing 2. When access starts at S1, 3 waitstates



Although the order of the story is reversed, I will explain how the clock of the peripheral device was set to 1/4 of the clock of the CPU.

Western Design Center's CPU W65C02S6 is guaranteed to operate at 3.3V and 8MHz. The peripheral device VIA W65C22S6 is also guaranteed to operate at 3.3V. If it is well designed, it may be possible to design a system with no wait for the entire system, but when I tried a prototype with the AIM65-CPLD-3V3, I could only operate normally up to 6MHz with a single clock. .

Looking at the AC specs of the W65C02S CPU and W65C22S VIA, I think the timing is tough because the tADS (address setup time) of the CPU is 40ns at the maximum, and the CS and ADR of the peripheral chip VIA are fixed before the rise of PHI2. AC specifications (tACR) that require In the case of static memory (FLASH or SRAM), if a memory with a fast tACC (access time) is used, data can be finalized by the next falling edge of PHI2. Even if you do, you may not make it in time. I should have investigated the cause of not operating at high speed in depth, but I quickly decided to delay the peripheral device access and operation clock. If the peripheral device clock is divided by 2 instead of by 4, it may be possible to operate at high speed in the total system, so I will try it separately.

W65C02S CPU AC characteristics (excerpt from data sheet)

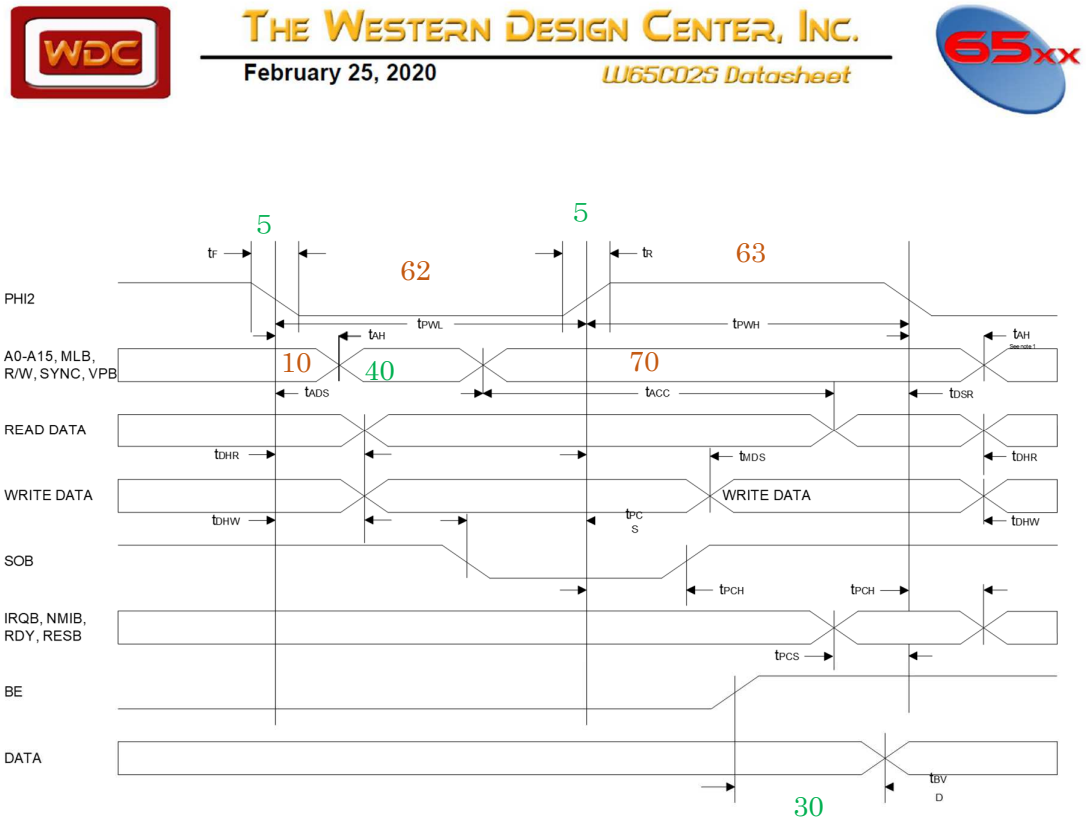


Figure 6-3 General Timing Diagram

W65C22S VIA AC characteristics (excerpt from the data sheet)

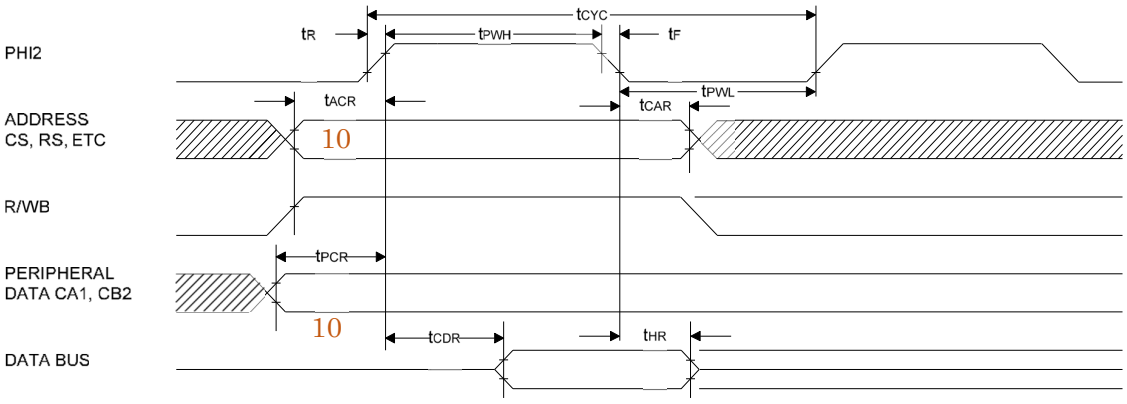


Figure 4-1 Read Timing

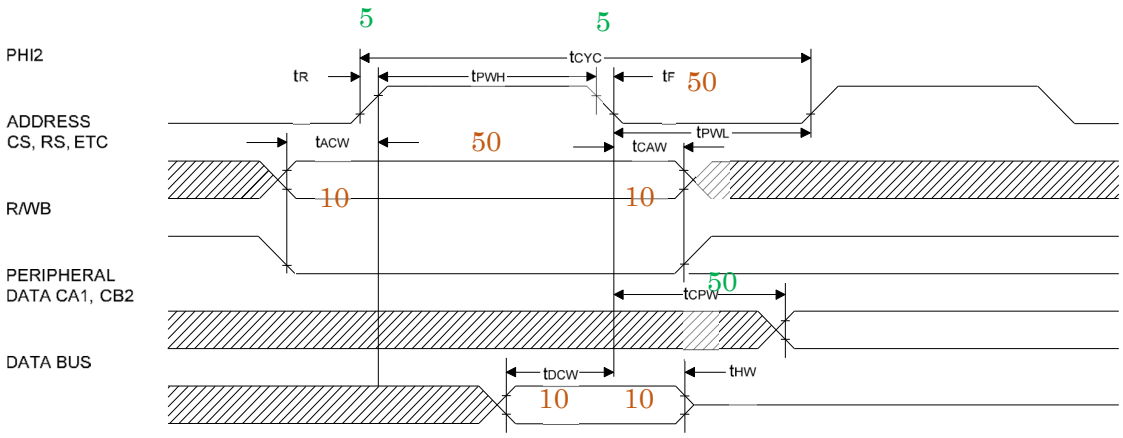
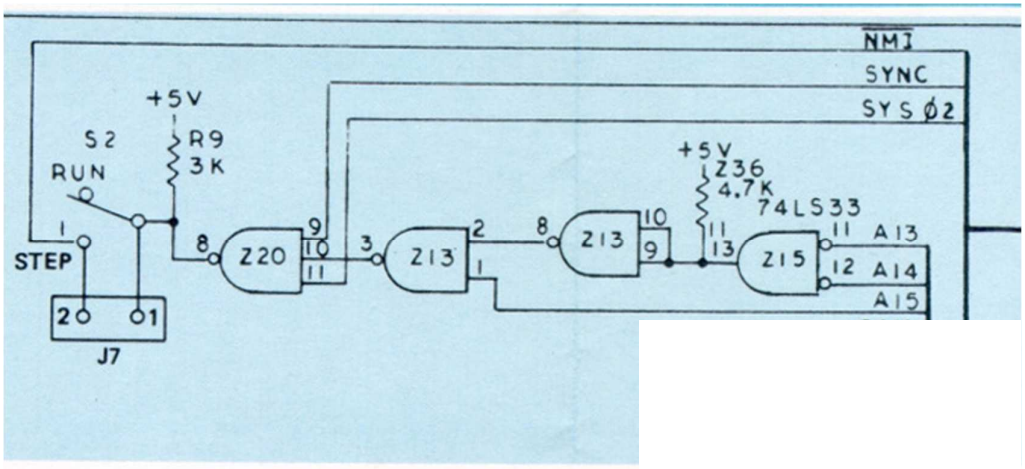


Figure 4-2 Write Timing

5.11 NMI generation circuit

The AIM65 monitor program includes debugging routines that support stepping and setting breakpoints. The NMI of the CPU is used exclusively for debugging functions. It is designed on the premise that the code to be debugged is in the RAM area (0000~9FFF), and NMI does not occur when executing code in EPROM. NMI is generated at the timing of access to the RAM area and instruction code access (SYNC).

NMI generation circuit for step execution in the AIM 65



The AIM65-CPLD-3V3 also incorporates the same NMI generation logic into the CPLD.

```
assign NMI = SYNC & SYS_PH2 & ( ~A15 | ( ~A14 & ~A13 ) ) ;
assign NMIB = ~NMI ;
```

This feature is not useful in BASIC or FORTH interpreters, since they are mostly unaware of the assembler code. Also, I don't write complex assembler code that requires stepping through debugging, and I rarely use this feature.

LISTING Control CPLD XC9572XL-VQG64 Verilog-HDL description (1/4)

```
1    // ******** XC9572XL-VQ64  : 3.3V  device *********************
2    // ******** AIM65-CPLD-3V3 ***********************************
3    // **** printer driver PROM $AE00 ******************************
4    //
5    // CSA0   : User VIA W65C22S
6    // CSA2   : reserved
7    // CSA400 : Work RAM
8    // CSA480 : RIOT CPLD ( R6532A replaced )
9    // CSA6   : reserved
10   // CSA8   : VIA W65C22S
11   // CSAA   : reserced
12   // CSAC   : PIA CPLD ( W65C20 replaced )
13   // CSAE   : RAM or PROM for printer driver )
14   //
15   // ROMSEL2, ROMSEL1
16   //    SW1, SW2     ROM area select
17   //    ------------------------------------------------
18   //         ON  --  PDRV_RAM
19   //         OFF --  PDRV_PROM
20   //    ON       --  PASCAL, RAM
21   //    OFF      --  PASCAL, PROM
22   //
23   // CKSEL1, CKSEL0
24   //    SW3,SW4            osc=8MHz osc=10MHz osc=12MHz
25   //    ------------------------------------------------------------
26   //    ON  ON   -- div8     1MHz   1.25MHz   1.5MHz
27   //    ON  OFF  -- div4     2MHz   2.5MHz      3MHz
28   //    OFF ON   -- div2     4MHz     5MHz      6MHz
29   //    OFF OFF  -- div1     8MHz  ( 10MHz ) ( 12MHz )reserved
30
31   module AIM65_control_4D(
32   OSCIN, PH2, RWB, SYNC, RESIN, CKSEL0, CKSEL1,
33   A7,A9,A10,A11,A12,A13,A14,A15,
34   ROMSEL1,ROMSEL2,IRQ1B,IRQ2B,
35   PH0,RESB,IRQB,RDY,
36   RAM_CS_X,ROM_CS_X,
37   CSA0_X,CSA2_X,CSA6_X,CSA8_X,CSAA_X,CSAC_X,CSAE_X,
38   CSA480_X,SYS_PH2,SYS_RWB,OEB,WEB,NMIB,SL_PH2,SL_RWB,
39   SYS_PH22,SL_PH22,
40   S1B, S4, WT
41   );
42
43   input  OSCIN;
44   input  PH2;
45   input  RWB;
46   input  SYNC;
47   input  RESIN;
48   input  CKSEL0;
49   input  CKSEL1;
50   input  A7,A9,A10,A11,A12,A13,A14,A15;
51   input  ROMSEL1;
52   input  ROMSEL2;
53   input  IRQ1B;
54   input  IRQ2B;
```

```
55    output PHO;
56    output RESB;
57    output IRQB;
58    output RDY;
59    output RAM_CS_X;
60    output ROM_CS_X;
```

LISTING Control CPLD XC9572XL-VQG64 Verilog-HDL description (2/4)

```
61
62    output CSA0_X;
63    output CSA2_X;
64    output CSA480_X;
65    output CSA6_X;
66    output CSA8_X;
67    output CSAA_X;
68    output CSAC_X;
69    output CSAE_X;
70    output SYS_PH2;
71    output SYS_PH22;
72    output SYS_RWB;
73    output SL_PH2;
74    output SL_PH22;
75    output SL_RWB;
76    output OEB;
77    output WEB;
78    output NMIB;
79    output S1B;
80    output S4;
81    output WT;
82    wire   PASCAL_en;
83    wire   PDRV_en;
84    wire   SLOW_PH2;
85    wire   SLOW_RWB;
86    wire   SLOW_CSA0;
87    wire   SLOW_CSA2;
88    wire   SLOW_CSA480;
89    wire   SLOW_CSA8;
90    wire   SLOW_CSAC;
91    wire   WAIT_AREA;
92
93    // ************** address decoder *********************************
94
95    assign IO_AREA = A15 & ~A14 & A13 & ~A12 ; // adr A000~AFFF
96    assign CSA0 = IO_AREA & ~A11 & ~A10 & ~A9 ;
97    assign CSA2 = IO_AREA & ~A11 & ~A10 &  A9 ;
98    assign CSA4 = IO_AREA & ~A11 &  A10 & ~A9 ;
99    assign CSA6 = IO_AREA & ~A11 &  A10 &  A9 ;
100   assign CSA8 = IO_AREA &  A11 & ~A10 & ~A9 ;
101   assign CSAA = IO_AREA &  A11 & ~A10 &  A9 ;
102   assign CSAC = IO_AREA &  A11 &  A10 & ~A9 ;
103   assign CSAE = IO_AREA &  A11 &  A10 &  A9 ;
104   assign CSA400 = CSA4 & ~A7 ;
105   assign CSA480 = CSA4 &  A7 ;
105   assign WAIT_AREA = CSA0 | CSA2 | CSA480 | CSA8 | CSAC ;
```

```
107
108  assign PASCAL_en = ROMSEL2 ;
109  assign PDRV_en   = ROMSEL1 ;
110  assign RAM_CS = ( ~A15 & ~A14 ) | ( ~A15 &  A14 & ~PASCAL_en ) |
111                  (  A15 & ~A14 & ~A13 ) | CSA400 | ( CSAE & ~PDRV_en ) ;
112  assign ROM_CS = ( A15 & A14 ) | ( A15 & A13 & A12 ) | ( ~A15 & A14 & PASCAL_en ) |
113                  ( CSAE & PDRV_en ) ;
114
115  assign CSA0_X    = CKSEL1 ? ~SLOW_CSA0 : ~CSA0 ;
116  assign CSA2_X    = CKSEL1 ? ~SLOW_CSA2 : ~CSA2 ;
117  assign CSA480_X  = CKSEL1 ? ~SLOW_CSA480 : ~CSA480 ;
118  assign CSA8_X    = CKSEL1 ? ~SLOW_CSA8 : ~CSA8 ;
119  assign CSAC_X    = CKSEL1 ? ~SLOW_CSAC : ~CSAC ;
120  assign CSA6_X    = ~CSA6 ;
121  assign CSAA_X    = ~CSAA ;
122  assign CSAE_X    = ~CSAE ;
123  assign RAM_CS_X  = ~RAM_CS ;
     assign ROM_CS_X = ~ROM_CS ;
```

LISTING Control CPLD XC9572XL-VQG64 Verilog-HDL description (3/4)

```
124
125  // ************************************************************
126
127  assign IRQB = IRQ1B & IRQ2B ;
128
129  // *********** CLOCK DIVIDER ***********************************
130
131  wire din0, din1, din2, div2ck, out0, out1, out2;
132  wire din3, din4, out3, out4, S4, S1B;
133  wire div1;
134
135  assign div1 = CKSEL1 &  CKSEL0;
136  assign div2ck = div1 ? OSCIN : out0 ;
137  assign din0 = ~out0 ;
138  assign din1 = ~out2;
139  assign din2 = CKSEL0 ? ~out2 :  out1 ;
140  DFF U1( .D(din0), .CLK(OSCIN),  .Q(out0) );
141  DFF U2( .D(din1), .CLK(div2ck), .Q(out1) );
142  DFF U3( .D(din2), .CLK(div2ck), .Q(out2) );
143  assign SYS_PH2 = CKSEL1 ? div2ck : out2 ;
144  assign SYS_PH22 = SYS_PH2;
145  assign PH2B = ~SYS_PH2;
146  assign din3 = ~out4;
147  assign din4 = out3;
148  assign S4 = out3 & ~out4;
149  assign S1B = ~( out3 & out4 );
150  DFF U5( .D(din3), .CLK(PH2B), .Q(out3) );
151  DFF U6( .D(din4), .CLK(PH2B), .Q(out4) );
152  assign SLOW_PH2 = ~out4;
153  assign SL_PH2 =  CKSEL1 ? SLOW_PH2 : SYS_PH2 ;
154  assign SL_PH22 = SL_PH2;
155
156  // ************************************************************
157
158  DFF U7( .D(SYS_RWB), .CLK(S1B), .Q(SLOW_RWB) );
159
160  DFF  U8( .D(CSA0),    .CLK(S1B), .Q(SLOW_CSA0) );
161  DFF U14( .D(CSA2),    .CLK(S1B), .Q(SLOW_CSA2) );
162  DFF  U9( .D(CSA480),  .CLK(S1B), .Q(SLOW_CSA480) );
163  DFF U10( .D(CSA8),   .CLK(S1B), .Q(SLOW_CSA8) );
164  DFF U11( .D(CSAC),   .CLK(S1B), .Q(SLOW_CSAC) );
165
166  // ******* RDY (WAIT) generator ****************************
167
168  wire din5, din6, out5, out6, PRDY, WT;
169  assign din5 = ~WAIT_AREA | ( S4 & WT );
170  DLAT U12( .D( din5 ), .G( SYS_PH2 ), .Q( PRDY ) );
171  // assign din6 = PRDY | S1B ;
172  assign RDY = CKSEL1 ? PRDY : 1'b1 ;
173  DFF U13( .D( PRDY ), .CLK( S1B ), .Q( out6 ) );
174  assign WT = ~out6;
175
176  // ************************************************************
```

```
177
178  wire sync_RES;
179  DFF U4( .D(RESIN), .CLK(SL_PH2), .Q(sync_RES) );
180  assign RESB = ~sync_RES;
181
182  assign SYS_RWB = RWB;                           // *******
183  assign SL_RWB = CKSEL1 ? SLOW_RWB : RWB;        // *******
184  assign OEB = ~( SYS_PH2 & RWB );
185  assign WEB = ~( SYS_PH2 & ~RWB );
186
187  assign NMI = SYNC & SYS_PH2 & ( ~A15 | ( ~A14 & ~A13 ) ) ;
188  assign NMIB = ~NMI ;
189
190  endmodule
191
```

LISTING Control CPLD XC9572XL-VQG64 Verilog-HDL description (4/4)

```
192  module DLAT( D, G, Q );
193    input D, G;
194    output Q;
195
196    assign Q = G ? D : Q;
197  endmodule
198
199  module DFF( D, CLK, Q );
200    input D;
201    input CLK;
202    output Q;
203    reg Q;
204  always @(posedge CLK)
205    Q <= D ;
206  endmodule
```

LIST○○ 制御用 CPLD XC9572XL-VQG64 の.ucf ファイルの記述

```
1    //**** XC9572XL-VQ64 PIN_ASSIGN *****
2
3    NET "OSCIN"     LOC = "P39";
4    NET "SYNC"      LOC = "P40";
5    NET "RESIN"     LOC = "P42";
6    NET "OEB"       LOC = "P43";
7    NET "WEB"       LOC = "P44";
8    NET "NMIB"      LOC = "P45";
9    NET "IRQ1B"     LOC = "P46";
10   NET "RESB"      LOC = "P47";
11   NET "IRQ2B"     LOC = "P48";
12   NET "A7"        LOC = "P49";
13   NET "A9"        LOC = "P50";
14   NET "A10"       LOC = "P51";
15   NET "A11"       LOC = "P52";
16   NET "A12"       LOC = "P56";
17   NET "A13"       LOC = "P57";
18   NET "A14"       LOC = "P58";
19   NET "A15"       LOC = "P59";
20   NET "SL_RWB"    LOC = "P60";
21
22   NET "WT"        LOC = "P27";
23   NET "RWB"       LOC = "P25";
24   NET "IRQB"      LOC = "P24";
25   NET "RAM_CS_X"  LOC = "P23";
26   NET "RDY"       LOC = "P22";
27   NET "ROM_CS_X"  LOC = "P20";
28   NET "CSA0_X"    LOC = "P19";  // SL
29   NET "CSA2_X"    LOC = "P18";  // SL
30   NET "CSA6_X"    LOC = "P17";
```

```
40    NET "ROMSEL2"   LOC = "P63";
41    NET "ROMSEL1"   LOC = "P64";
42    NET "CKSEL1"    LOC = "P1";
43    NET "CKSEL0"    LOC = "P2";
44
45    NET "SYS_PH22"  LOC = "P6";
46    NET "SL_PH22"   LOC = "P5";
```

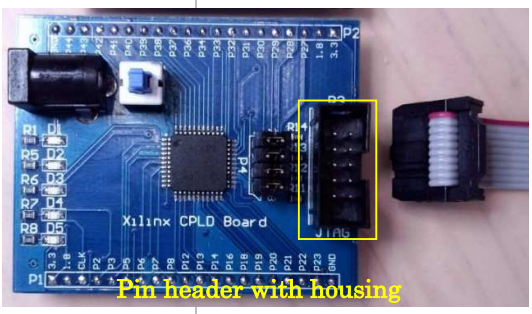
```
31  NET "CSA8_X"    LOC = "P16";  // SL
32  NET "CSAA_X"    LOC = "P15";
33  NET "CSAC_X"    LOC = "P13";  // SL
34  NET "S1B"       LOC = "P12";
35  NET "CSA480_X"  LOC = "P11";  // SL
36  NET "S4"        LOC = "P10";
37  NET "SYS_PH2"   LOC = "P9";
38  NET "SYS_RWB"   LOC = "P8";
39  NET "SL_PH2"    LOC = "P7";
```

.ucf describes the correspondence between the I/O signals in the top layer of Verilog and the pin numbers of the CPLD in this file.

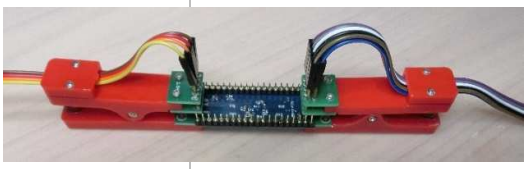
5.12 [Extra] A convenient CPLD writing clip

A pin header with a housing is often used for writing CPLDs. By fixing the position of the signal, it is convenient because there is no misconnection of the signal when connecting to the JTAG programmer. However, when the pin header is mounted on the printed circuit board, it suddenly becomes cluttered. A clip-type writer is used there. This clip uses a test pin (pogo pin) with a spring in each terminal, and the connection is completed simply by pinching the printed circuit board with the clip. If the JTAG signal PAD is placed at a fixed position on the printed board with the CPLD, there is no need to solder the pin header to the printed board. I use this clip not only for CPLDs, but also for FLASH writing for MCUs like MSP430 and STM32 (the number and placement of signals varies).

In the case of Xilinx JTAG programmers, there is no power supply function, and another clip is used only for power connection (although some programmers seem to be able to supply power).



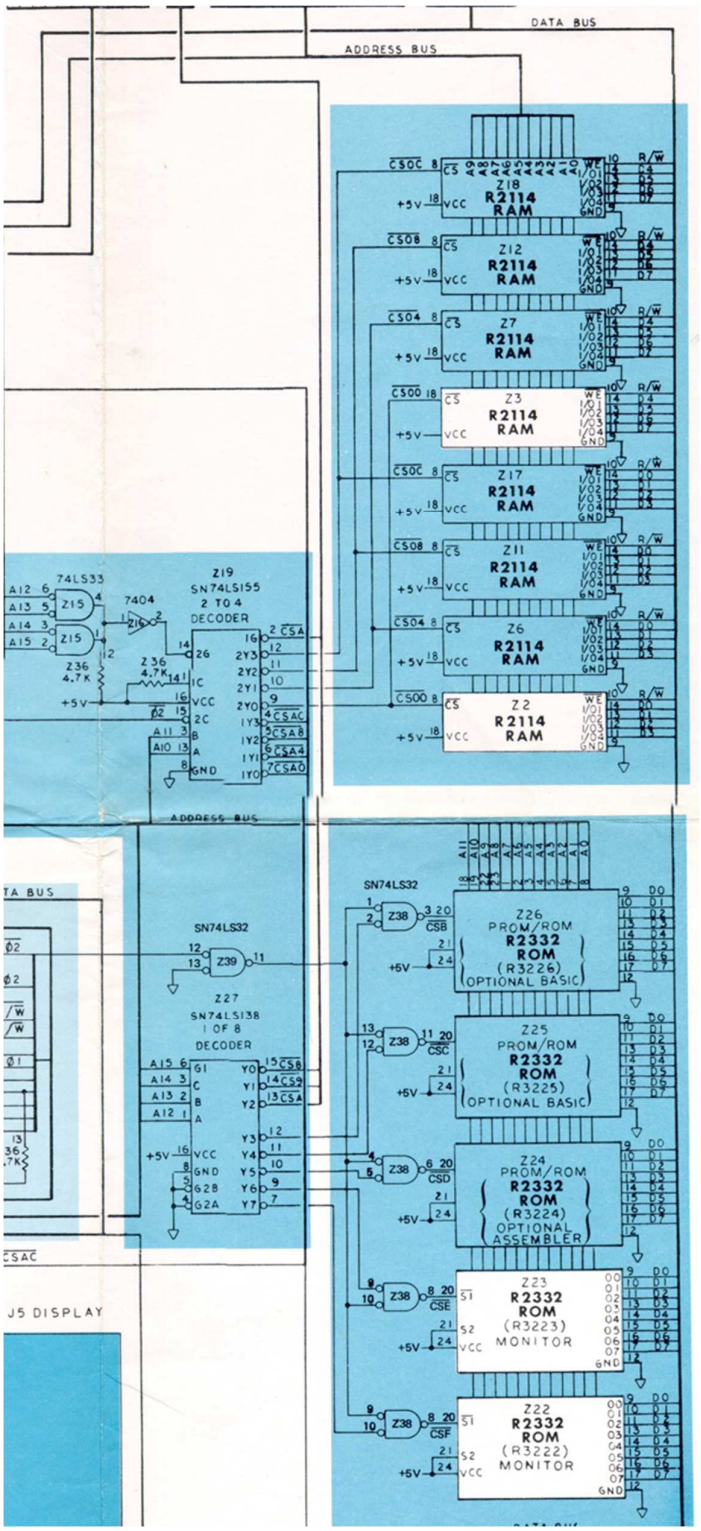
Pin header with housing



5.13 PROM/SRAM section

The ROM/RAM part matches the original AIM65 only in the address map, and the parts used are completely changed. However, since it is static memory, there is no change in the method of signal generation.

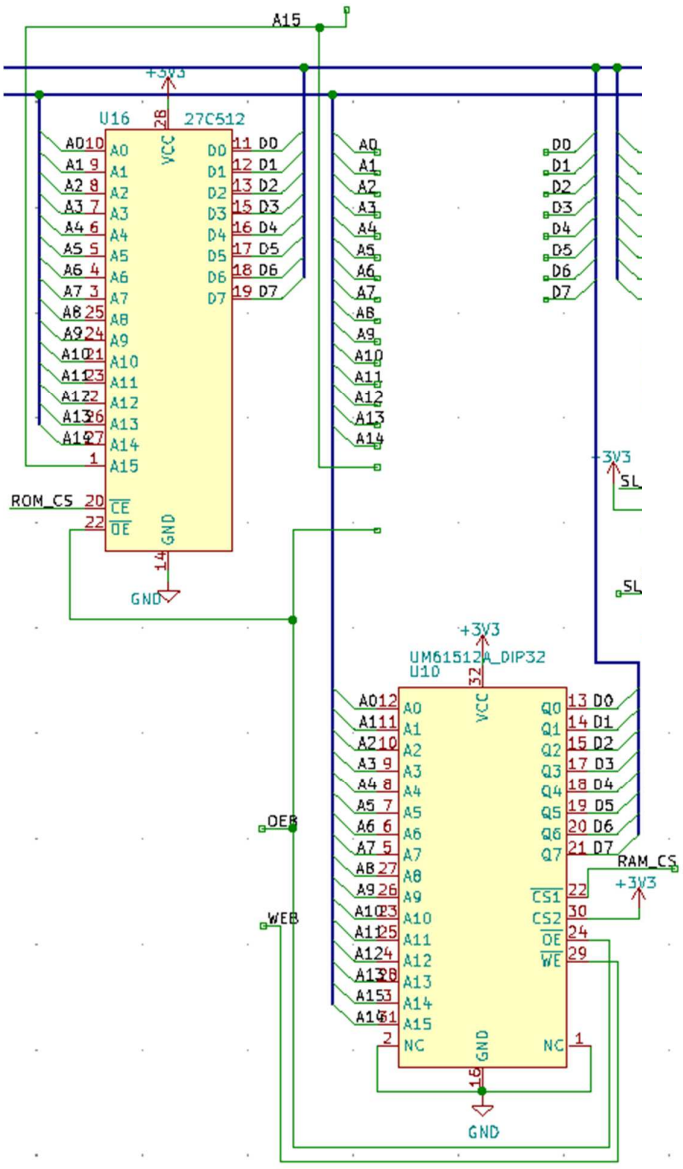
AIM65 ROM/RAM and address decoder



The ROM/RAM of the AIM65-CPLD-3V3 has a terminal layout that matches W27C512 for FLASH and matches UM61512A and AS6C4008 for SRAM so that the transition from the 5V version reproduction goes relatively smoothly. Please note that A15 and A14 of the CPU are connected to A14 and A15 of the SRAM in the 3.3V version as well as the 5V version. This is because it was easier to lay out the EPROM 27C512 and SRAM UM61512A in the 5V version.

In the circuit diagram (schematic data) of AIM65-CPLD-3V3, 27C512 and UM61512A are used for ROM/RAM symbols to match the terminal arrangement to these when creating a FRASH/SRAM DIP conversion board. is. The actual RAM used is AS7C34096A instead of UM61512A and the ROM is SST39LF040 (FLASH) instead of W27C512 (EPROM).

ROM/RAM connection on the main board



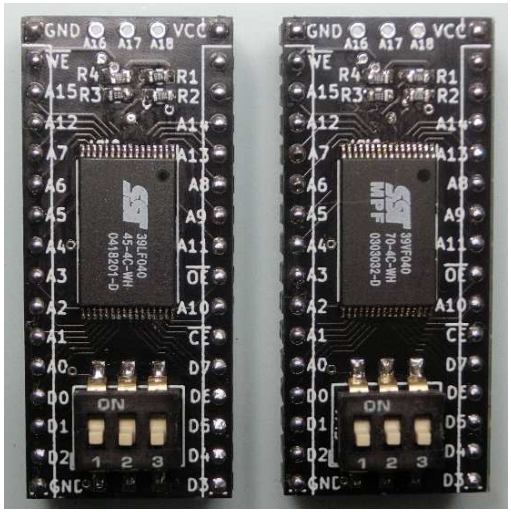
5.14 Flash memory board

FLASH memory is the SST39LF040 or SST39VF040.

The 3pin to 30pin of the DIP conversion board of the FLASH memory matches the pin arrangement of 1pin to 28pin of Winbond W27C512 of 5V operation. It has a capacity of 512KB, but A16, A17, and A18 are fixed by DIP SW and can store 64KB x 8 patterns.

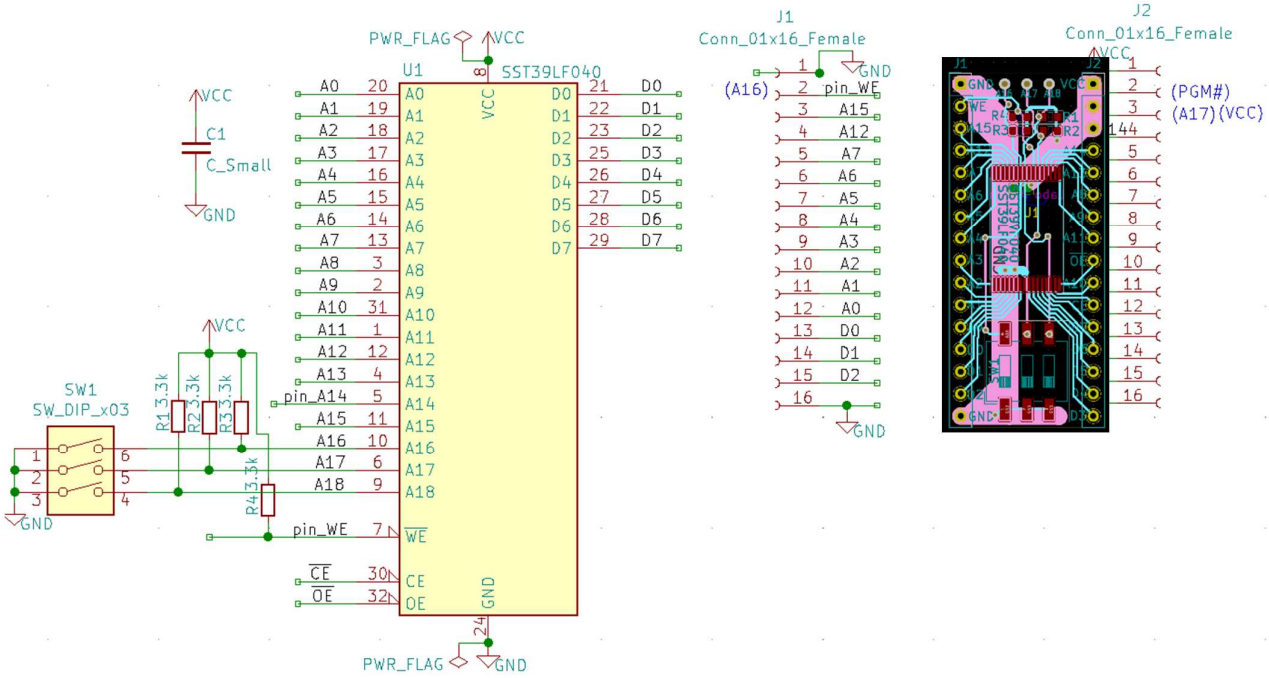
I used my own FLASH writer to write SST39LF040. However, the procedure is not automated and is difficult to use universally. I would like to explain the W27C512 EPROM writer and eraser used in the 5V system on another occasion.

FLASH memory DIP conversion board



SST39LF040 SST39VF040

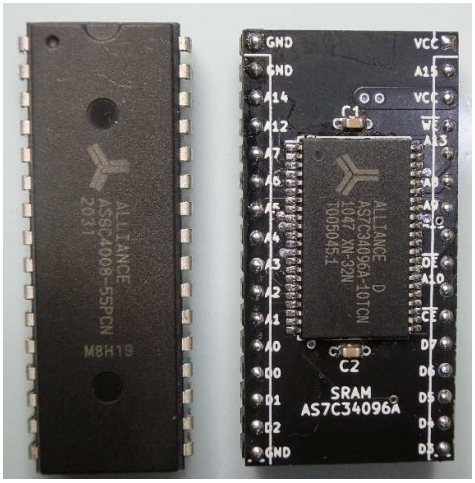
FLASH SST39LF040 DIP conversion board circuit diagram



5.15 SRAM

ALLIANCE's AS7C34096A-10 (access time 10ns product) is used for SRAM. The capacity is 512KB (4Mbit), but only 64KB of it is used. Also, ALLIANCE's AS6C4008-55 can operate from 2.7V to 5.5V and can also be used at 3.3V, but its specs are up to 8MHz. It seems to work normally even with a CPU clock of 12MHz, but please use at 12MHz at your own risk. Basically, we recommend using AS7C34096A-10. In this case, it works experimentally up to CPU clock 16MHz, but please use at 12MHz and 16MHz at your own risk.

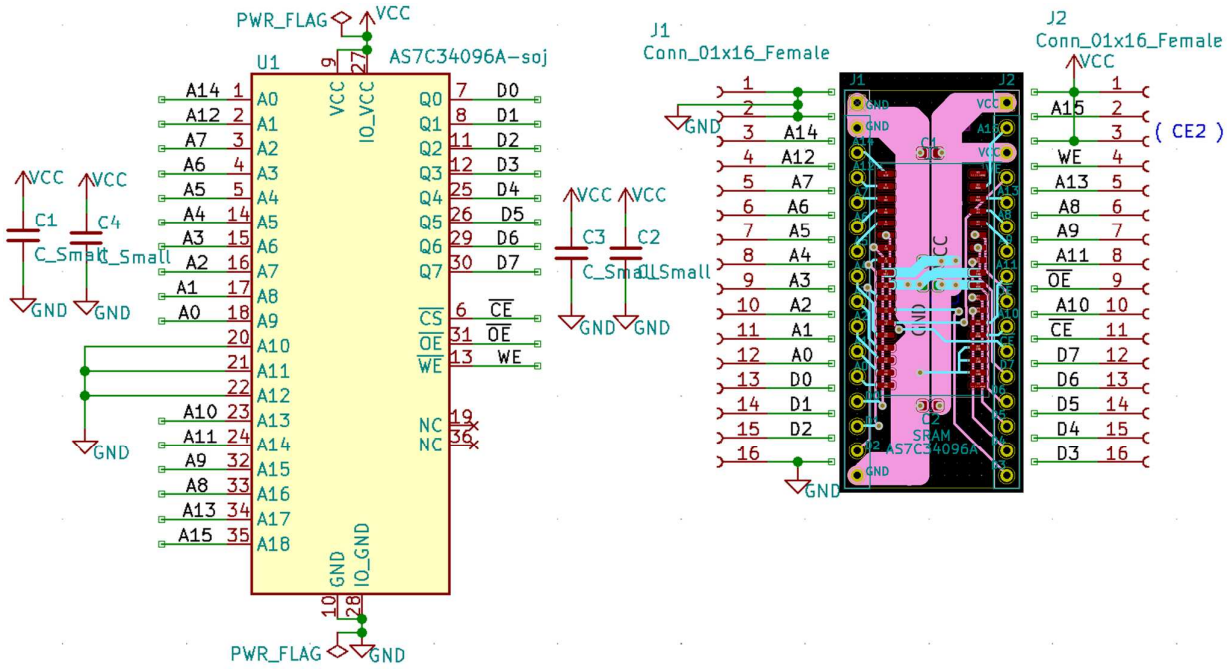
SRAM DIP conversion board



AS6C4008-55 AS7C34096A-10

SRAM AS7C34096A DIP conversion board circuit diagram

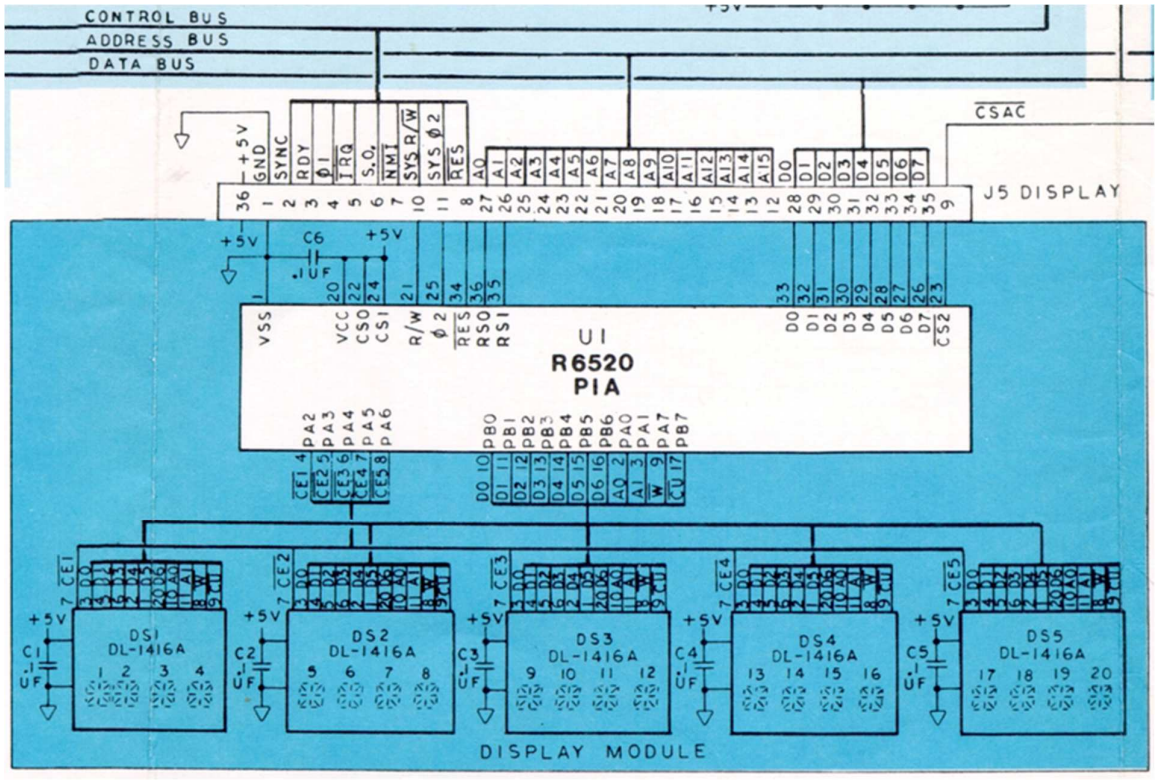
The SRAM DIP conversion board matches the pin arrangement with AS6C4008 and UM61512A。



Foot Print is prepared for C3 and C4, but they are not used.

5.16 16 segment LED display

Original AIM 65 16segment LED circuit diagram



Both PIA GPIO port A and port B are used as simple output ports to control LEDs. CA1, CA2, CB1, and CB2 are not used, nor are interrupts.

The 16-segment LED used in the original AIM-65 is part number DL-1416A, but it is almost impossible to obtain now. I searched for a part with the same function, but could not find it. For HPDL-1414, it seems that reused/recycled products are on the market for similar parts.

The size of HPDL-1414 is about 2/3 of DL-1416A, so it is convenient to make it small.

For HP DL-1414, if you search for "hpdl1414" on AMAZON or AliExpress, reused/recycled products will be found. There is one from about 500 yen. For 20 characters (5 pieces), it costs 3000 yen including shipping, so I can't say it's cheap.

The DL-1414 does not support the cursor display function compared to the DL-1416A, and the control signals are different accordingly. However, when I actually converted the signal and used it, I got the same display as the original AIM 65 without any software changes. The cursor when running BASIC and FORTH is ^, not the type that lights up all 16 segments. From this, it seems that ^ is displayed by software without using the cursor function of DL-1416A.

Like the DL-1416A, the DL-1414 is an old device and has the drawback of large current consumption. If you connect an amperemeter to AIM65-CPLD-3V3 and use it, you can clearly see that the current increases as the number of characters displayed on the 16 segment LED increases. Each asterisk * increases by about 12mA.

16-segment LED circuit diagram

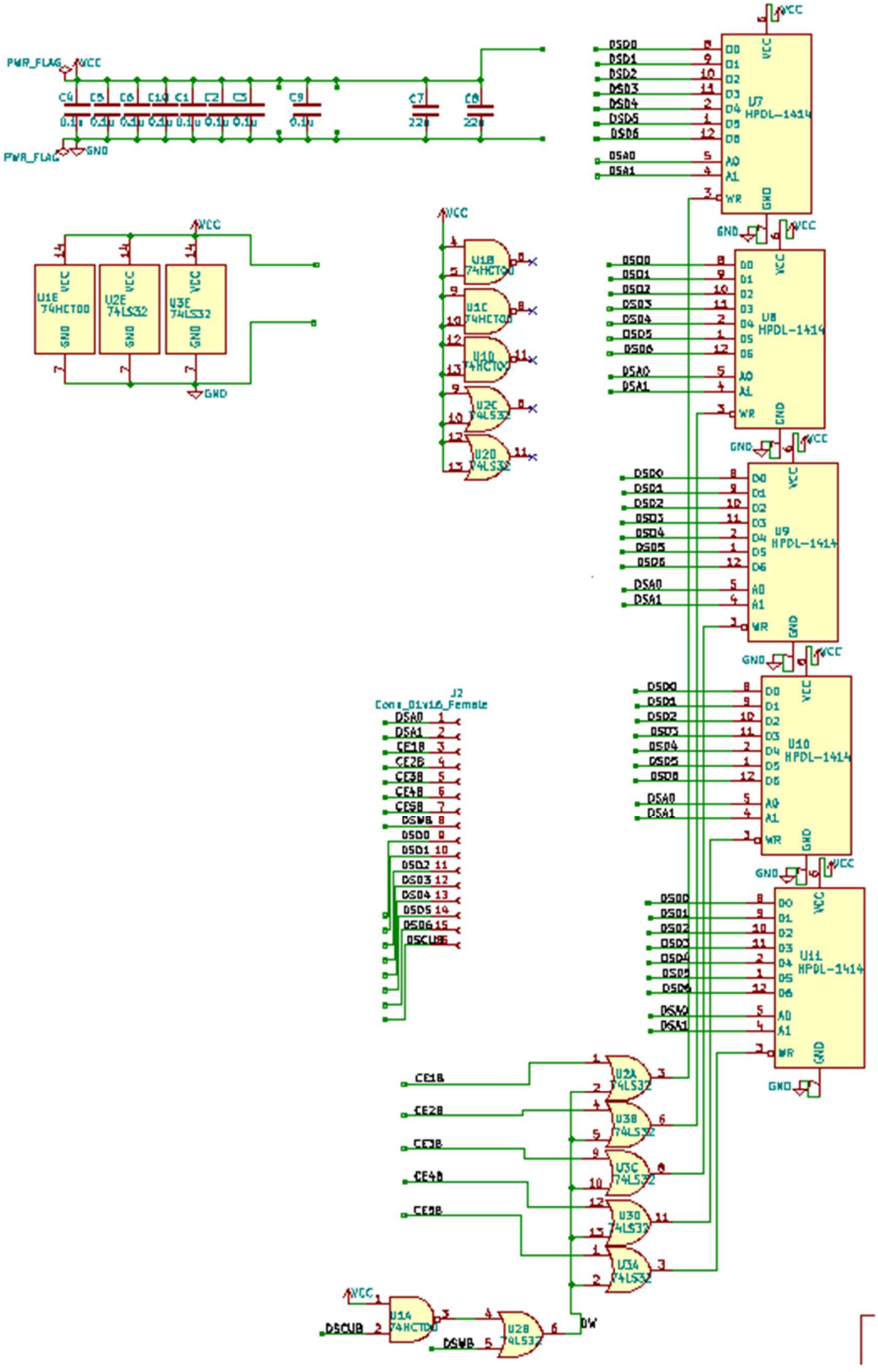
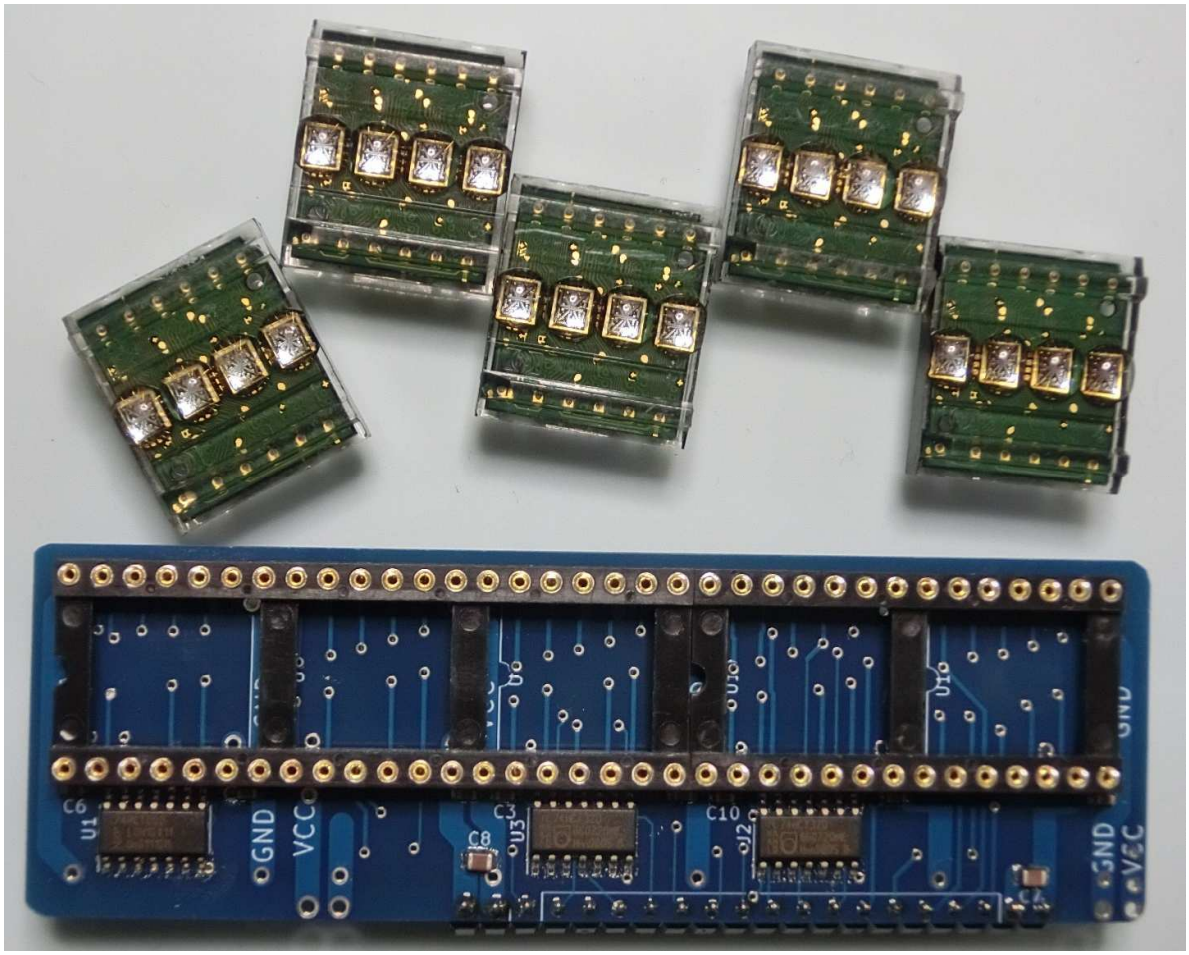


Photo 16-Segment LED

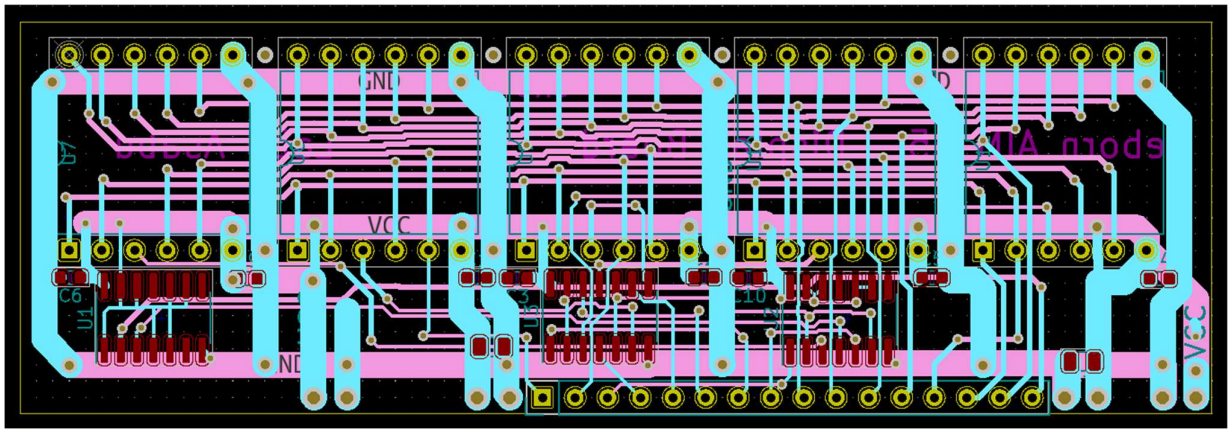


Schematic and board layout

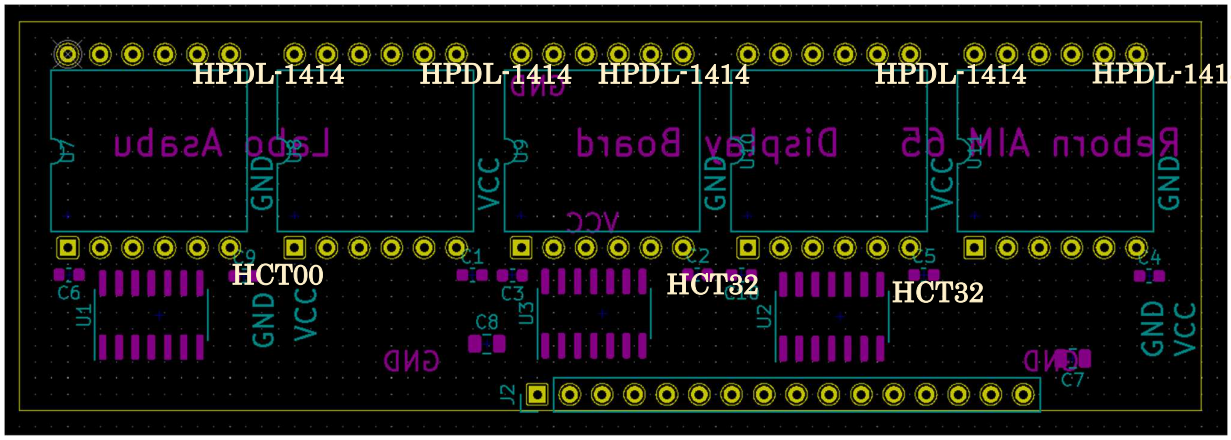
The next figure shows the circuit diagram of the 16segment LED board. The power supply of HPDL-1414 is 5V and the 74HCT00 and 74HCT32 on the LED board also operate at 5V. Since the control signal is TTL level, the output of the 3.3V operation PIO (CPLD) on the main board is directly connected.

As an ingenuity in the board layout, we placed through holes in the middle of each LED so that 40-pin DIP and 28-pin DIP sockets can be mounted.

16 Segment LED board layout



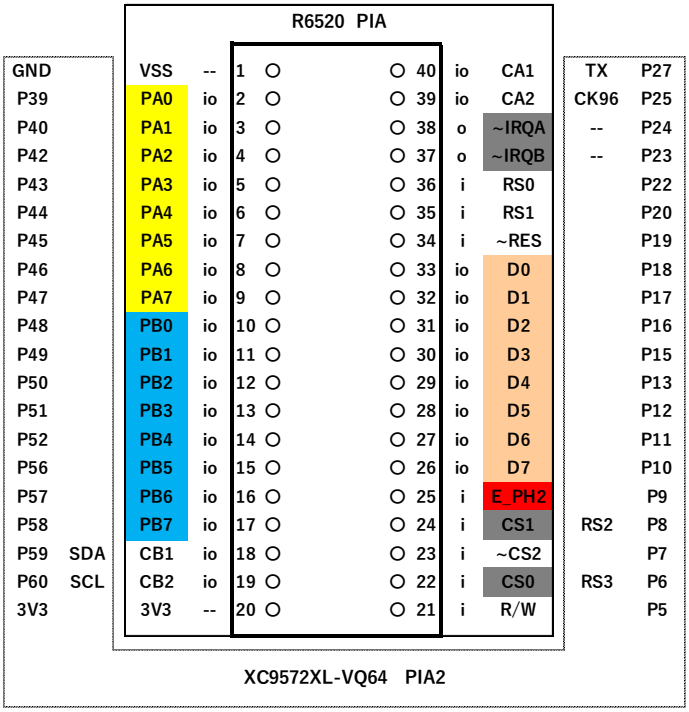
16 Segment LED parts layout



5.17 Replace PIA with CPLD

The original AIM 65 uses an R6520 PIA to control a 16 segment LED. Since the reproduction AIM65-CPLD-3V3 operates at 3.3V, I decided to replace it with the XC9572XL-VQ64. Attach Verilog-HDL description of CPLD (XC9572XL_VQ64_PIA_3)

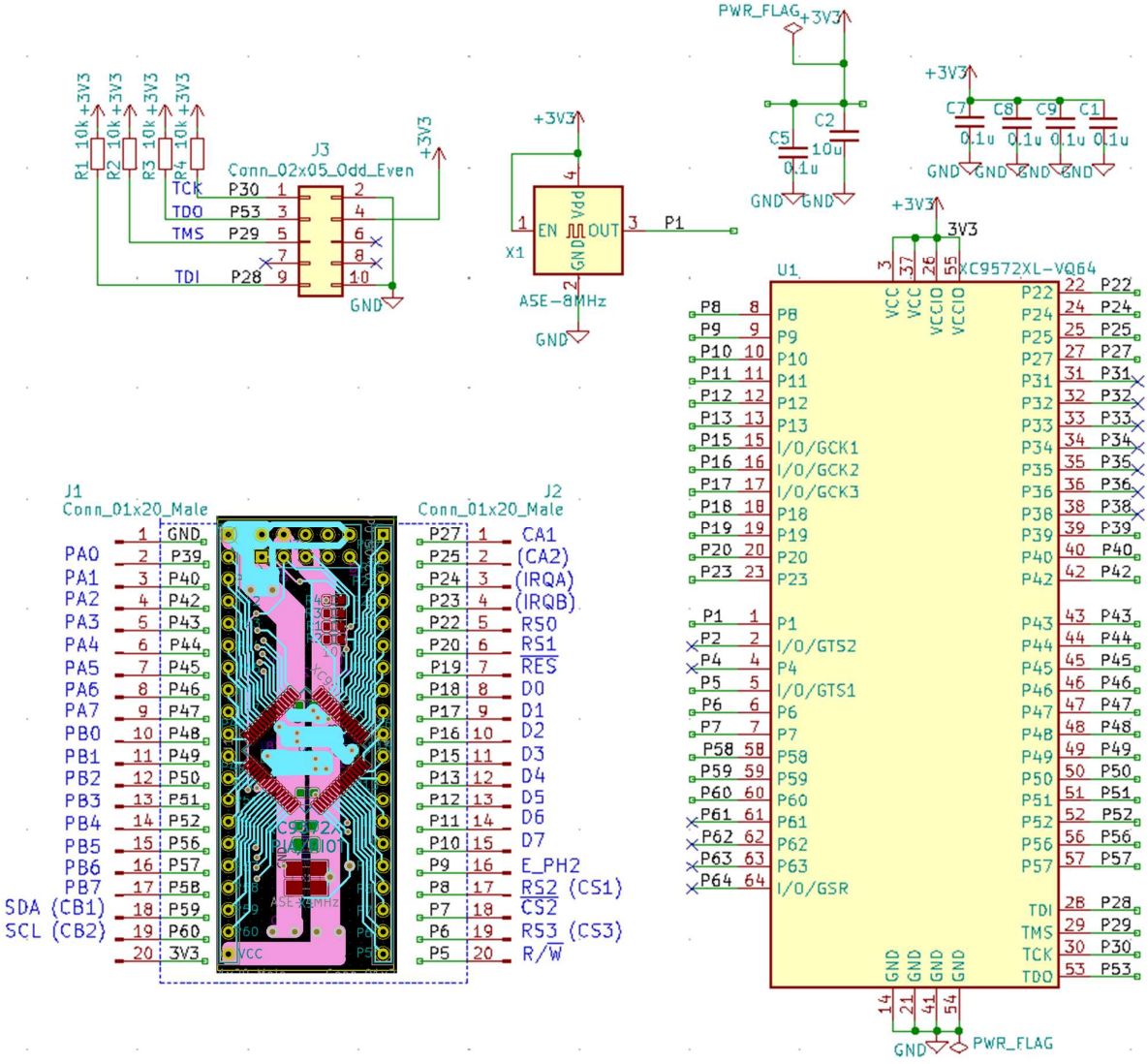
XC9572XL-VQ64 PIA2 pin arrangement



First of all, regarding the terminal arrangement of the CPLD (hereafter referred to as PIA2), some of the terminals that were not used in the original AIM 65 have been changed in order to add functions.

- The CS1 and CS0 pins have been changed to RS2 and RS3 for register address expansion to add built-in resources. However, RS3 is not used due to lack of CPLD macrocells.
- Allocated TX for thermal printer to CA1.
- CA2 is a test terminal. Outputs UART-TX baud rate clock CK96 (9615Hz) for thermal printers.
- CB1 and CB2 are assigned to the extended I2C SDA and SCL pins.
- IRQA and IRQB terminals are NC. I wanted to output a signal, but I could not add the function due to lack of CPLD macrocells.
- Equipped with an 8MHz oscillation module for baud rate generation on the DIP conversion board. The DIP conversion board is common to PIA2 and RIOT2. When using as RIOT2, it is not necessary to install an oscillation module. In the future, it will be useful to have an oscillator module when implementing timers for RIOT.

PIA2 DIP conversion board circuit diagram and layout diagram



PIA2 register allocation and address map

PIA2 register allocation

ポートA ADR = $AC00

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ビット	D7	D6	D5	D4	D3	D2	D1	D0
属性	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初期値	不定	不定	不定	不定	不定	不定	不定	不定

ポートB ADR = $AC02

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ビット	D7	D6	D5	D4	D3	D2	D1	D0
属性	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初期値	不定	不定	不定	不定	不定	不定	不定	不定

TXDレジスタ ADR = $AC04

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ビット	D7	D6	D5	D4	D3	D2	D1	D0
属性	W	W	W	W	W	W	W	W
初期値	不定							

TXステータスレジスタ ADR = $AC05

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ビット	BUSYF	MSBF	8M_STOP	---	---	---	SDA	SCL
属性	R	R/W	R/W	"0"	"0"	"0"	R/W	R/W
初期値	"0"	"0"	"0"				"1"	"1"

Register address of PIA and PIA2

RS2	RS1	RS0	PIA Register	PIA2 Register
O	0	0	Port Data A / DDRA	Port Data A (Port A)
0	0	1	CRA	--
0	1	0	Port Data B / DDRB	Port Data B (Port B)
0	1	1	CRB	--
1	0	0	--	TXD register
1	0	1	--	TX status, I2C register
1	1	0	--	--
1	1	1	--	--

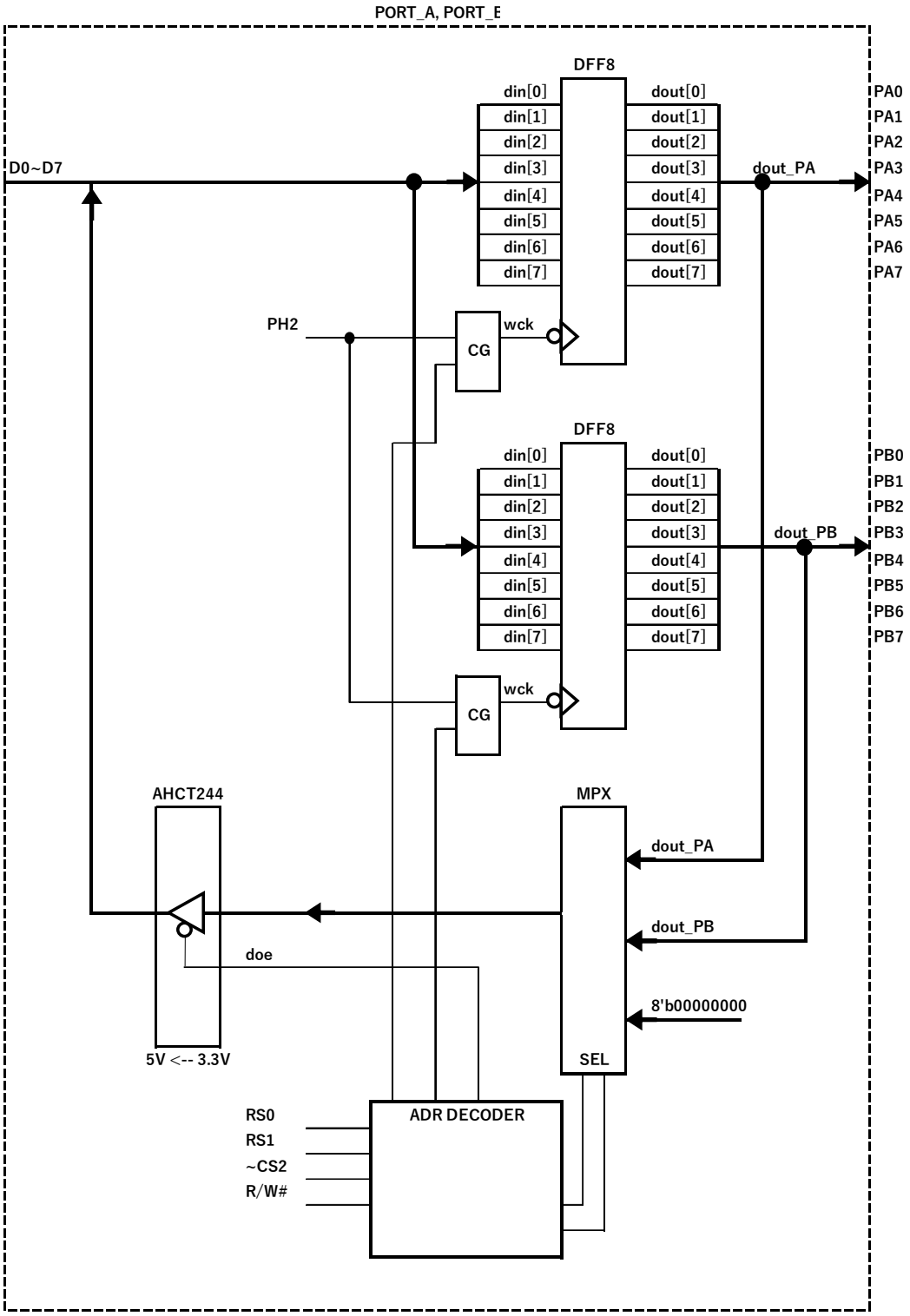
In the original AIM 65, PIA is used as a simple 8 bit output port. CA1, CA2, CB1, CB2, IRQA, IRQB are not used. Both port A and port B are set to output all bits. Therefore, when replacing PIA with CPLD, control registers CRA, CRB and direction registers DDRA, DDRB were not implemented. Since the contents of the ROM have not been changed, accesses to these register addresses occur when AIM 65 is running, but they are all ignored and only accesses to the data registers of port A and port B are valid.

On the other hand, add RS2 (register select 2) signal for additional functions, add TXD register and TX status register for UART-TX of thermal printer, and add I2C port (SDA, SCL)Added.

How to use port A and port B

Just specify the address of port A and port B and store the data. If you want to change some of the 8 bits of the output port, you need to read, modify, and write (read from the port, process it, and write it back to the port).

Verilog-HDL description and equivalent circuit of port A and port B

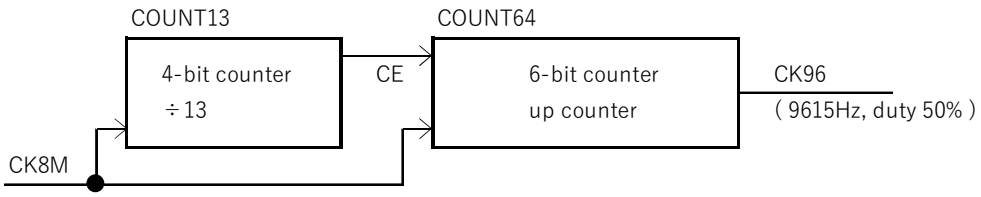

PORT_A, PORT_E
DFF8
din[0]
din[1]
din[2]
din[3]
din[4]
din[5]
din[6]
din[7]
dout[0]
dout[1]
dout[2]
dout[3]
dout[4]
dout[5]
dout[6]
dout[7]
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
D0~D7
dout_PA
PH2
CG
wck
DFF8
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
dout_PB
AHCT244
MPX
doe
8'b00000000
5V <-- 3.3V
SEL
RS0
RS1
~CS2
R/W#
ADR DECODER

5.18 How to use UART-TX for the printer

UART is always performing parallel-to-serial conversion. From the perspective of the CPU, all you have to do is confirm that the BUSYF (busy flag) in the TX status register is 0 and write data to the TXD register. Parallel-to-serial conversion is normally LSB first and bit0 is output first. This is enough to output ASCII code, but when outputting a bitmap image, you may want to output with MSB first, so assign MSBF to bit6 of the status register and write "1" to Enabled parallel serial conversion in First. I planned to stop the clock when the printer is not in use, but I could not add logic due to lack of macrocells.

Verilog-HDL description and equivalent circuit of UART-TX for printer

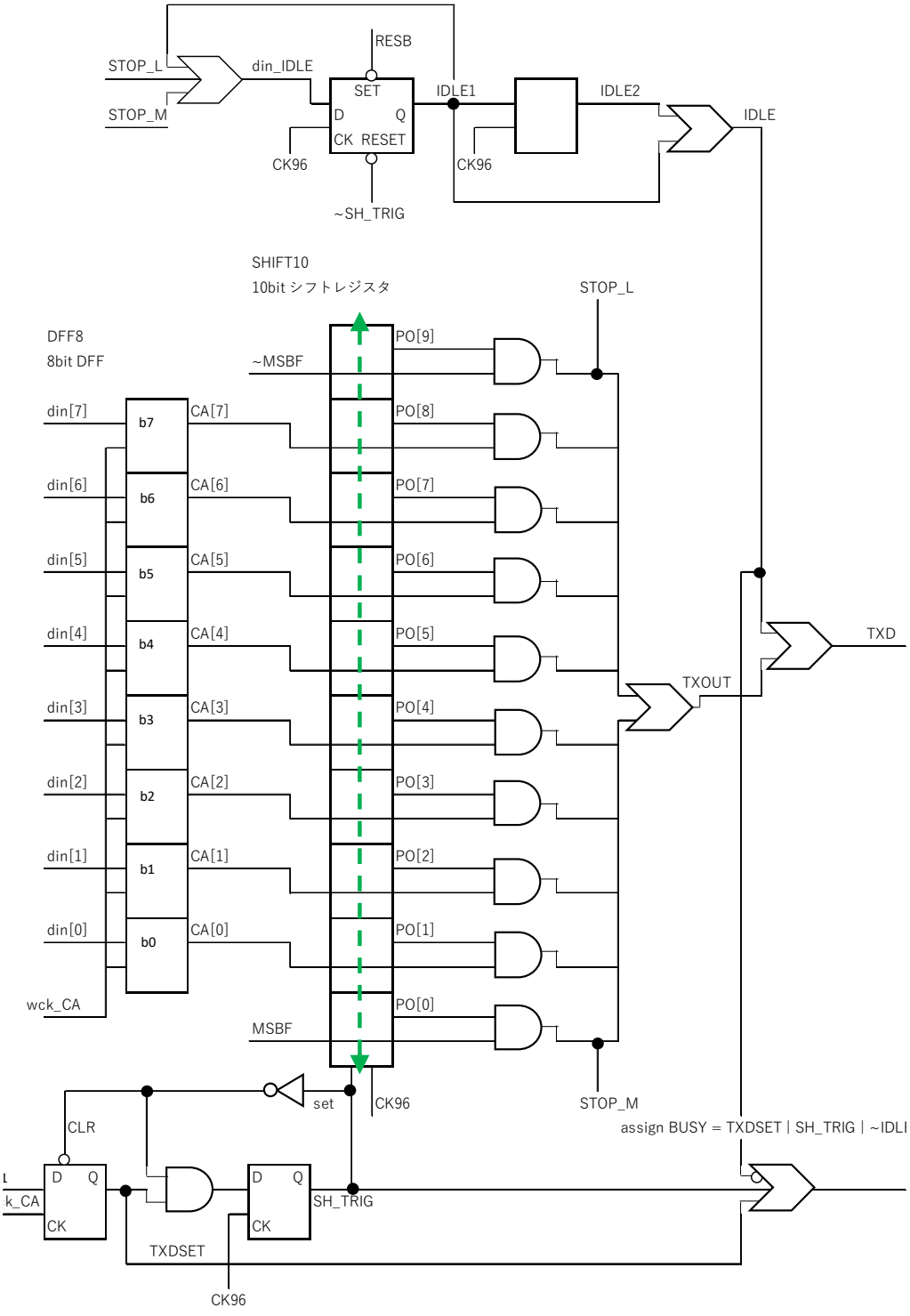
Prescaler for baud rate generation




```
module COUNT13 (CK8M, CE)
input CK8M;
output CE;
reg[3:0] CNTR;
reg CE;

always @(posedge CK8M)
begin CNTR <= CNTR+1'b1; end

always @ (CNTR)
begin
  if (CNTR == 4'b1100 )
    CE <= 1'b1;
    CNTR <= 4'b0000;
  else
    CE <= 1'b0;
end
endmodule
```

```
module COUNT64 (CK8M, CE, CK96)
input CK8M;
input CE;
output CK96;
reg[5:0] CNTR6;
reg CK96;

always @(posedge CK8M)
begin
  if (CE == 1'b1 ) CNTR6 <= CNTR6+1'b1;
end

always @ (CNTR6)
begin
  if (CNTR6 == 6'b011111 )
    CK96 <= 1'b1;
  else if (CNTR6 == 6'b111111 )
    CK96 <= 1'b0;
end
endmodule
```

Parallel-to-serial conversion circuit



Parallel-to-serial conversion Verilog description

```
DFF8 U7( .din(din), .wck(wck_CA), .dout(CA) );
DFF_async_reset  U8( .D(1'b1), .CLK(wck_CA), .RSTB( ~SH_TRIG ), .Q(TXDSET) );
DFF_async_reset U11( .D(din[6]), .CLK(wck_CB), .RSTB(RESB), .Q(MSBF) );
DFF_async_set   U18( .D(din[1]), .CLK(wck_CB), .STB(RESB), .Q(SD) );
DFF_async_set   U19( .D(din[0]), .CLK(wck_CB), .STB(RESB), .Q(SC) );
DFF  U9( .D( TXDSET&~SH_TRIG ), .CLK(CK96), .Q(SH_TRIG) );

assign STOP_L =  ~MSBF & PO[9];
assign STOP_M =   MSBF & PO[0];
assign din_IDLE =  STOP_L | STOP_M | IDLE1 ;
assign IDLE = IDLE1 | IDLE2;
DFF_async_set_reset U10( .D(din_IDLE), .CLK(CK96), .STB(RESB), .RSTB(~SH_TRIG), .Q(IDLE1) );
DFF U21( .D(IDLE1), .CLK(CK96), .Q(IDLE2) );

SHIFT10 U12( .CK(CK96), .MSBF(MSBF), .TRIG(SH_TRIG), .PO(PO) );

assign TXOUT =
| { STOP_L ,( CA[7] & PO[8] ),( CA[6] & PO[7] ),( CA[5] & PO[6] ),( CA[4] & PO[5] ),
   ( CA[3] & PO[4] ),( CA[2] & PO[3] ),( CA[1] & PO[2] ),( CA[0] & PO[1]), STOP_M  };
assign TXD = TXOUT | IDLE;
assign BUSY = TXDSET | SH_TRIG | ~IDLE ;

assign dout_CB  = { BUSY, MSBF, 4'b000,  SDA, SCL };
```

submodule

```
module DFF_async_reset( D, CLK, RSTB, Q );
  input D;
  input CLK;
  input RSTB;
  output Q;
  reg Q;
always @( negedge CLK or negedge RSTB )
begin
  if ( ~RSTB ) Q <= 1b0;
  else  Q <= D ;
end
endmodule
```

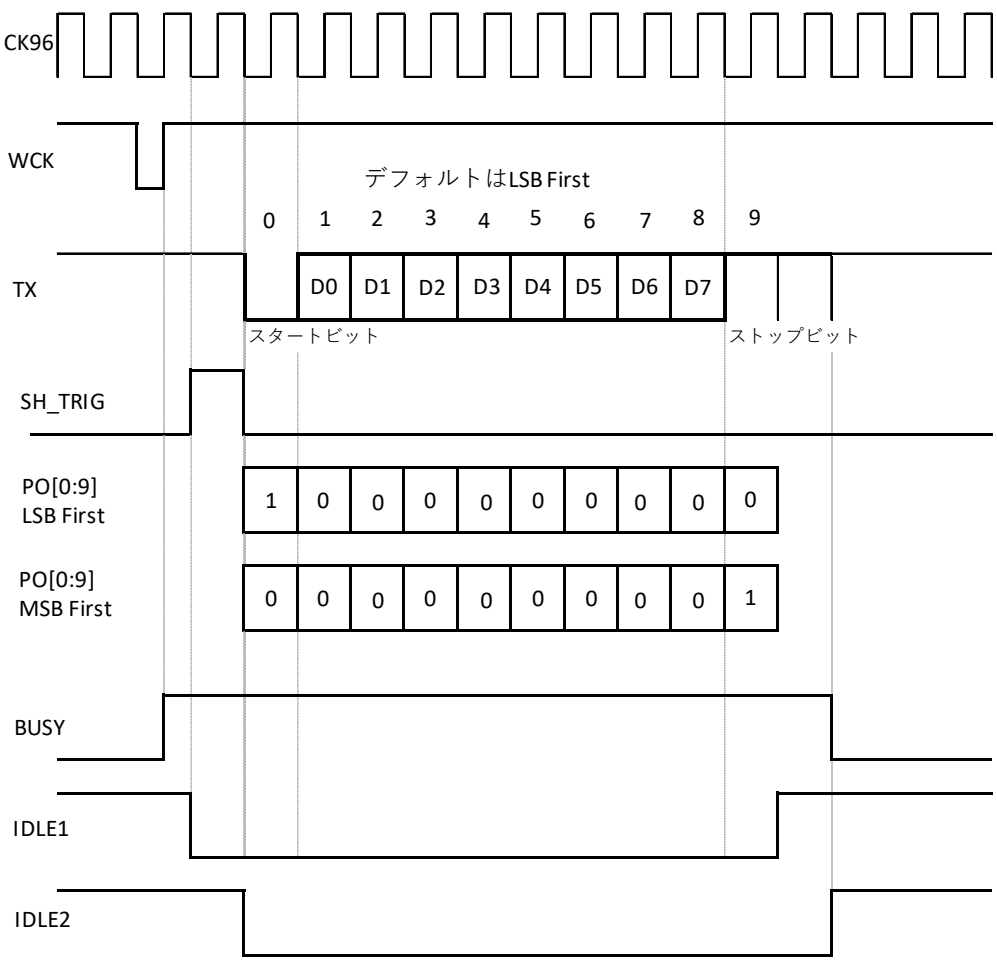
```
module DFF_async_set_reset( D, CLK, STB, RSTB, Q );
  input D;
  input CLK;
  input STB;
  input RSTB;
  output Q;
  reg Q;
always @( negedge CLK or negedge STB or negedge
RSTB )
begin
  if ( ~STB ) Q <= 1'b1;
  else if ( ~RSTB ) Q <= 1'b0;
  else  Q <= D ;
end
endmodule
```

```
module DFF_async_set( D, CLK, STB, Q );
  input D;
  input CLK;
  input STB;
  output Q;
  reg Q;
always @( negedge CLK or negedge STB )
begin
  if ( ~STB ) Q <= 1b1;
  else  Q <= D ;
end
endmodule
```

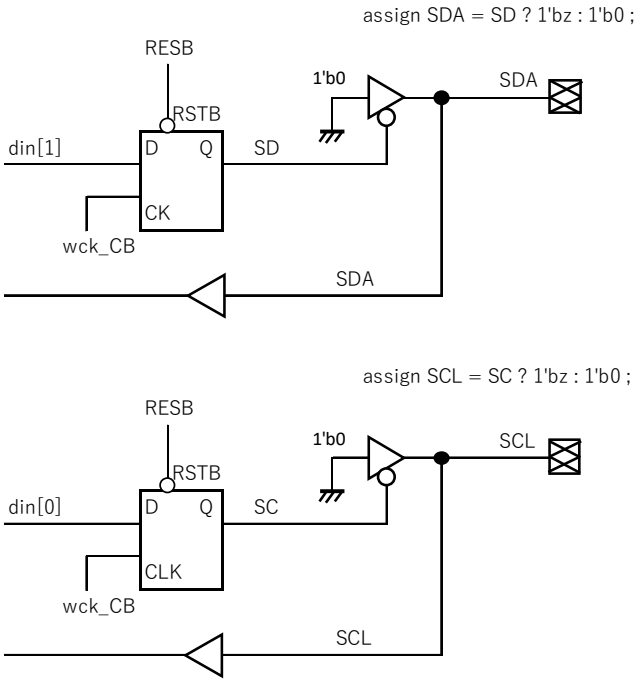
```
module DFF8( din, wck, dout );  // 8bit D-FF
  input  wck ;
  input  [7:0] din;
  output [7:0] dout;

  reg    [7:0] dout;

  always @( negedge wck )
              dout <= din ;
endmodule
```

```
module SHIFT10(CK, MSBF, TRIG, PO);
  input CK;
  input TRIG;
  input MSBF;
  output [9:0] PO;
  reg [9:0] PO;
always @( negedge CK )
begin
  if ( TRIG )
    if ( ~MSBF ) PO <= 10'b0_0000_0000_1;
    else         PO <= 10'b1_0000_0000_0;
  else
    if ( ~MSBF ) PO <= ( PO << 1);
    else         PO <= ( PO >> 1);
end
endmodule
```

Parallel-to-serial conversion time chart



I2C control pin




```
assign CB_sel  = cs &  RWB &  RS2 & ~RS1 &  RS0;  // $AC05 read
assign CB_we   = cs & ~RWB &  RS2 & ~RS1 &  RS0;  // $AC05 write

CG U15( .en( CB_we  ), .ck( PH2 ), .gck( wck_CB ) );

DFF_async_set   U18( .D(din[1]), .CLK(wck_CB), .STB(RESB), .Q(SD) );
DFF_async_set   U19( .D(din[0]), .CLK(wck_CB), .STB(RESB), .Q(SC) );
```

```
wire  SD, SC;
assign D = doe ? dout : 8'bz;
assign din = D;
assign SDA = SD ? 1'bz : 1'b0 ;
assign SCL = SC ? 1'bz : 1'b0 ;
```

PIA2 has added two pairs of Dual Flipflops and pseudo open drain terminals for I2C control with two signals, SDA and SCL. I wanted to assign an independent address to each 1-bit register of SDA and SCL, but due to lack of macrocells, I assigned them to bit1 and bit0 of the TX status register. Can be used as an I2C master device by controlling with software bit-banging method。

Listing Verilog-HDL description of XC9572XL_VQ64_PIA_3 (1/4)

```
1   //////////////////////////////////////////////////////////////////////////////////
2   // Company: Labo Asabu
3   // Create Date:  NOV-24/2022
4   // Design Name: XC9572XL_PIA_3
5   // Target Devices: XC9572XL-VQ64,
6   // Revision:
7   //////////////////////////////////////////////////////////////////////////////////
8   // adr decode
9   //   000 : PA[7:0]                                          : $AC00
10  //   001 : control_A  dummy , read=$00                      : $AC01
11  //   010 : PB[7:0]                                          : $AC02
12  //   011 : control_B  dummy , read=$00                      : $AC03
13  //   100 : CA[7:0] ; UART_TXD                               : $AC04
14  //   101 : CB[7:0] ; { SCL,SDA,3'b000,STOP8M,MSBF,BUSY_F }  : $AC05
15
16  module XC9572XL_VQ64_PIA_3 (
17    PH2, RWB, RESB,  RS0, RS1, RS2, CS2X,
18    D, PA, PB, TXD, SDA, SCL, OSC_8M, CK96 // , SH_TRIG, BUSY, CE
19  );
20
21  input  PH2;
22  input  RWB;
23  input  RESB;
24  input  RS2, RS1, RS0;
25  input  CS2X;
26  input  OSC_8M;
27  inout  [7:0] D;
28  output [7:0] PA;
29  output [7:0] PB;
30  inout  SDA;
31  inout  SCL;
32  output TXD;
33  output CK96;
34  // output SH_TRIG;
35  // output BUSY;
36  // output CE;
37
38  wire  [7:0] din;
39  wire  [7:0] dout;
40  wire  doe;
41  wire  SD, SC;
42  assign D = doe ? dout : 8'bz;
43  assign din = D;
44  assign SDA = SD ? 1'bz : 1'b0 ;
45  assign SCL = SC ? 1'bz : 1'b0 ;
46
47  assign cs = ~CS2X ;
48  wire PA_sel, PB_sel, CB_sel;
49  wire wck_PA, wck_PB, wck_CA, wck_CB; //write clock
50
51  assign PA_sel  = cs &  RWB & ~RS2 & ~RS1 & ~RS0;  // $AC00 read
52  assign PB_sel  = cs &  RWB & ~RS2 &  RS1 & ~RS0;  // $AC02 read
53  // assign CA_sel  = cs &  RWB &  RS2 & ~RS1 & ~RS0;  // $AC04 read
```

```
54   assign CB_sel  = cs &  RWB &  RS2 & ~RS1 &  RS0;  // $AC05 read
55
56   assign PA_we   = cs & ~RWB & ~RS2 & ~RS1 & ~RS0;  // $AC00 write
57   assign PB_we   = cs & ~RWB & ~RS2 &  RS1 & ~RS0;  // $AC02 write
58   assign CA_we   = cs & ~RWB &  RS2 & ~RS1 & ~RS0;  // $AC04 write
59   assign CB_we   = cs & ~RWB &  RS2 & ~RS1 &  RS0;  // $AC05 write
60
61   wire  [7:0] dout_PA;
62   wire  [7:0] dout_PB;
63   wire  [7:0] CA;
64   wire  [9:0] PO;
65   wire  [7:0] dout_CB;
```

Verilog-HDL description of XC9572XL_VQ64_PIA_3 (2/4

```
66   wire  CK8M;
67   wire  IDLE, IDLE1, IDLE2;
68   wire  CE;   // Count Enable
69   wire  CK96;
70   wire  MSBF;
71   // wire  STOP8M;
72   wire  TXDSET, din_IDLE, SH_TRIG, BUSY;
73
74   assign PA = dout_PA;
75   assign PB = dout_PB;
76   assign doe = cs & RWB & PH2;
77
78   CG  U4( .en( PA_we  ), .ck( PH2 ), .gck( wck_PA ) );
79   CG  U5( .en( PB_we  ), .ck( PH2 ), .gck( wck_PB ) );
80   CG  U6( .en( CA_we  ), .ck( PH2 ), .gck( wck_CA ) );
81   CG U15( .en( CB_we  ), .ck( PH2 ), .gck( wck_CB ) );
82
83   DFF8 U1( .din(din), .wck(wck_PA), .dout(dout_PA) );
84   DFF8 U2( .din(din), .wck(wck_PB), .dout(dout_PB) );
85   DFF8 U7( .din(din), .wck(wck_CA), .dout(CA) );
86   DFF_async_reset  U8( .D(1'b1), .CLK(wck_CA), .RSTB( ~SH_TRIG ), .Q(TXDSET) );
87   // DFF U11( .D(din[6]), .CLK(wck_CB), .Q(MSBF) );
88   DFF_async_reset U11( .D(din[6]), .CLK(wck_CB), .RSTB(RESB), .Q(MSBF) );
89   // DFF_async_reset U20( .D(din[5]), .CLK(wck_CB), .RSTB(RESB), .Q(STOP8M) );
90   DFF_async_set   U18( .D(din[1]), .CLK(wck_CB), .STB(RESB), .Q(SD) );
91   DFF_async_set   U19( .D(din[0]), .CLK(wck_CB), .STB(RESB), .Q(SC) );
92   DFF  U9( .D( TXDSET&~SH_TRIG ), .CLK(CK96), .Q(SH_TRIG) );
93
94   assign STOP_L =  ~MSBF & PO[9];
95   assign STOP_M =   MSBF & PO[0];
96   assign din_IDLE =  STOP_L | STOP_M | IDLE1 ;
97   assign IDLE = IDLE1 | IDLE2;
98   DFF_async_set_reset U10( .D(din_IDLE), .CLK(CK96), .STB(RESB), .RSTB(~SH_TRIG), .Q(IDLE1) );
99   DFF U21( .D(IDLE1), .CLK(CK96), .Q(IDLE2) );
100
101  SHIFT10 U12( .CK(CK96), .MSBF(MSBF), .TRIG(SH_TRIG), .PO(PO) );
102
103  assign TXOUT =
104  | { STOP_L ,( CA[7] & PO[8] ),( CA[6] & PO[7] ),( CA[5] & PO[6] ),( CA[4] & PO[5] ),
105      ( CA[3] & PO[4] ),( CA[2] & PO[3] ),( CA[1] & PO[2] ),( CA[0] & PO[1]), STOP_M  };
```

```
106  assign TXD = TXOUT | IDLE;
107  assign BUSY = TXDSET | SH_TRIG | ~IDLE ;
108
109  assign dout_CB  = { BUSY, MSBF, 4'b000,  SDA, SCL };
110
111  MPX  U3( .a(dout_PA), .b(dout_PB), .c(dout_CB),
112           .asel(PA_sel), .bsel(PB_sel), .csel(CB_sel),
113           .mpxout(dout) );
114
115  COUNT13  U13( .CK8M(CK8M), .CE(CE) );
116  COUNT64  U14( .CK8M(CK8M), .CE(CE), .CK96(CK96) );
117
118  assign CK8M = OSC_8M;
119
120  endmodule
121
122  module CG( en, ck, gck );  // Clock Gating cell
123    input en, ck;
124    output gck;
125    wire A, G, Q;
126
127    assign G = ck;
128    assign A = en;
129    assign Q = G ? A : Q;
130    assign gck = Q & G & A;
131  endmodule
```

Verilog-HDL description of XC9572XL_VQ64_PIA_3 (3/4

```
132
133  module DFF8( din, wck, dout );  // 8bit D-FF
134    input  wck ;
135    input  [7:0] din;
136    output [7:0] dout;
137    reg    [7:0] dout;
138
139    always @( negedge wck )
140           dout <= din ;
141  endmodule
142
143  module DFF( D, CLK, Q );
144    input D;
145    input CLK;
146    output Q;
147    reg Q;
148  always @( negedge CLK )
149    Q <= D ;
150  endmodule
151
152  module DFF_async_reset( D, CLK, RSTB, Q );
153    input D;
154    input CLK;
155    input RSTB;
```

```
156    output Q;
157    reg Q;
158  always @( negedge CLK or negedge RSTB )
159  begin
160    if ( ~RSTB ) Q <= 1'b0;
161    else  Q <= D ;
162  end
163  endmodule
164
165  module DFF_async_set( D, CLK, STB, Q );
166    input D;
167    input CLK;
168    input STB;
169    output Q;
170    reg Q;
171  always @( negedge CLK or negedge STB )
172  begin
173    if ( ~STB ) Q <= 1'b1;
174    else  Q <= D ;
175  end
176  endmodule
177
178  module DFF_async_set_reset( D, CLK, STB, RSTB, Q );
179    input D;
180    input CLK;
181    input STB;
182    input RSTB;
183    output Q;
184    reg Q;
185  always @( negedge CLK or negedge STB or negedge RSTB )
186  begin
187    if ( ~STB ) Q <= 1'b1;
188    else if ( ~RSTB ) Q <= 1'b0;
189    else  Q <= D ;
190  end
191  endmodule
192
```

Verilog-HDL description of LIST○○ XC9572XL_VQ64_PIA_3 (4/4

```
193  module SHIFT10(CK, MSBF, TRIG, PO);
194    input CK;
195    input TRIG;
196    input MSBF;
197    output [9:0] PO;
198    reg [9:0] PO;
199  always @( negedge CK )
200  begin
201    if ( TRIG )
202       if ( ~MSBF ) PO <= 10'b0_0000_0000_1;
203       else         PO <= 10'b1_0000_0000_0;
204    else
205       if ( ~MSBF ) PO <= ( PO << 1);
206       else         PO <= ( PO >> 1);
207  end
208  endmodule
209
210  module MPX( a, b, c, asel, bsel, csel, mpxout );  //
211    input [7:0] a, b, c;
212    input asel, bsel, csel;
213    output [7:0] mpxout;
214    reg [7:0] mpxout;
215
216    always @(a or b or c or asel or bsel or csel)
217    if(asel) mpxout = a;
218    else if(bsel) mpxout = b;
219    else if(csel) mpxout = c;
220    else mpxout = 8'b00000000;
221  endmodule
222
223  module COUNT13( CK8M, CE );
224  input CK8M;
225  output CE;
226  reg[3:0] CNTR;
227  reg CE;
228  always @(posedge CK8M)
229    if (CNTR == 4'b1100 )
230      begin CE <= 1'b1; CNTR <= 4'b0000; end
231    else
232      begin CE <= 1'b0; CNTR <= CNTR + 1'b1; end
233  endmodule
234
235  module COUNT64( CK8M, CE, CK96 );
236  input CK8M;
237  input CE;
238  output CK96;
239  reg[5:0] CNTR6;
240  reg CK96;
241  always @(posedge CK8M)
242  begin
243    if (CE == 1'b1 ) CNTR6 <= CNTR6+1'b1;
```

```
244     else if (CNTR6 == 6'b011111 ) CK96 <= 1'b1;
245     else if (CNTR6 == 6'b111111 ) CK96 <= 1'b0;
246   end
247   endmodule
```

Description of XC9572XL_VQ64_PIA_3 .ucf file

```
1    NET "PA[0]"     LOC = "P39";
2    NET "PA[1]"     LOC = "P40";
3    NET "PA[2]"     LOC = "P42";
4    NET "PA[3]"     LOC = "P43";
5    NET "PA[4]"     LOC = "P44";
6    NET "PA[5]"     LOC = "P45";
7    NET "PA[6]"     LOC = "P46";
8    NET "PA[7]"     LOC = "P47";
9    NET "PB[0]"     LOC = "P48";
10   NET "PB[1]"     LOC = "P49";
11   NET "PB[2]"     LOC = "P50";
12   NET "PB[3]"     LOC = "P51";
13   NET "PB[4]"     LOC = "P52";
14   NET "PB[5]"     LOC = "P56";
15   NET "PB[6]"     LOC = "P57";
16   NET "PB[7]"     LOC = "P58";
17   NET "SDA"       LOC = "P59";
18   NET "SCL"       LOC = "P60";
19
20   NET "TXD"       LOC = "P27";
21   NET "CK96"      LOC = "P25";  // internal signal
22   // NET "SH_TRIG"   LOC = "P24";  // internal signal
23   // NET "BUSY"      LOC = "P23";  // internal signal
24   NET "RS0"       LOC = "P22";
25   NET "RS1"       LOC = "P20";
26   NET "RESB"      LOC = "P19";
27   NET "D[0]"      LOC = "P18";
28   NET "D[1]"      LOC = "P17";
29   NET "D[2]"      LOC = "P16";
30   NET "D[3]"      LOC = "P15";
31   NET "D[4]"      LOC = "P13";
32   NET "D[5]"      LOC = "P12";
33   NET "D[6]"      LOC = "P11";
34   NET "D[7]"      LOC = "P10";
35   NET "PH2"       LOC = "P9";
36   NET "RS2"       LOC = "P8";
37   NET "CS2X"      LOC = "P7";
38   // NET "RS3"       LOC = "P6";
39   NET "RWB"       LOC = "P5";
40
41   NET "OSC_8M"    LOC = "P1";   // internal signal
42   // NET "CE"        LOC = "P64";  // internal signal
```

5.19 The thermal printer

A thermal printer identical to the original AIM 65 is not available. Therefore, I decided to use a printer that can be controlled by serial (UART) with AIM65-CPLD-3V3. There are many types of 58mm wide paper available for receipt printing or label printing on the Internet. They support TTL level input, and all can receive at speeds of 9600bps.

If you have 9600bps UART output, you can print to a thermal printer, so you can print not only in KB mode of AIM 65 but also in TTY mode by connecting 9600bps UART output to a printer. Unfortunately, the AIM 65 TTY mode output is 7-bit data, and even if I connect it as it is, some of the characters are garbled, and I couldn't get a completely correct output.

Regarding actual thermal printers, the above three types of thermal printers found on the net seem to be usable with AIM65. Of these, the QR701 was not adopted due to its large size, but the control method looks almost the same as the QR204 and QR203. I don't know where the official Datasheets corresponding to these are. Adafruit

https://cdn-learn.adafruit.com/downloads/pdf/mini-thermal-receipt-printer.pdf

https://www.sparkfun.com/datasheets/Components/General/A2-user%20manual-1.pdf

Since there is an instruction manual for the thermal printer, I decided to control it referring to this.

The commands seem to be somewhat common, and some can be used as is. I decided to use only what I could actually try and use. To check the fonts that can be printed, it is quick and easy to start in test mode. If you turn on the power while pressing the paper feed switch, a test print will be performed, so you can check the fonts that can be printed.

QR204

This printer can print 32 characters/line. However, AIM65's KB mode operation uses only 20 characters/line. After actually using it, I think it is suitable for AIM65 REPRODUCTION because the QR204 is in a box, it is compact, there are no exposed parts inside, it is relatively quiet during operation, and the printing is beautiful. I made QR204 the standard for AIM65-CPLD-3V3 because it looks good on a printed circuit board.

QR203

It is one of the options because the price is cheaper than QR204. However, the problem is that it is noisy. I think the original AIM65 was also quite noisy with exposed mechanics...

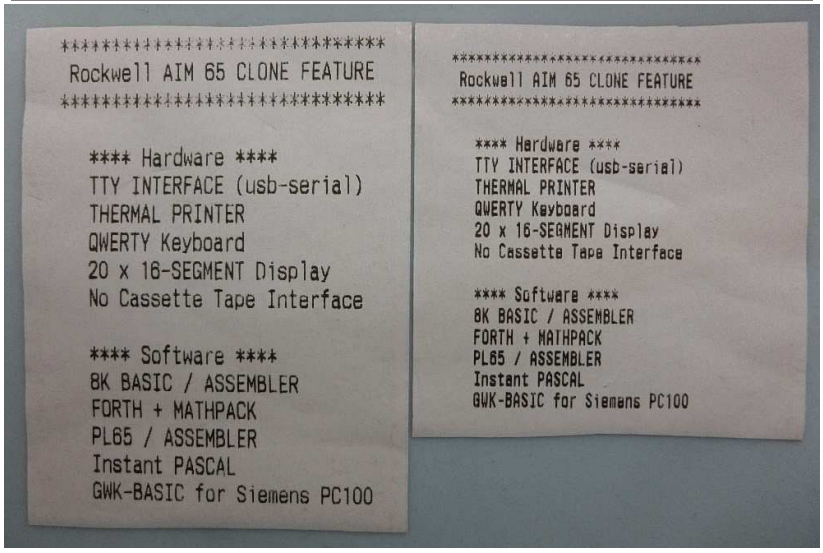
The external size is slightly larger than QR204.

The control method is also slightly different from QR204. A line feed will not occur if only a CR (line feed = carriage return, hexadecimal 0D) is sent when no character data has been sent. Therefore, in the printer driver used for patching, a line feed is generated by sending $0D after sending a space ($20) according to QR203.

The nice thing about the QR203 is that you can use a small 9x17 font in addition to the 12x24 font. In that case, 42 characters/line can be printed. And line spacing can also be adjusted by commands to match small fonts. By default, only 32 characters/line can be printed, but if you use FONT-B, you can print up to 42 characters/line. The KB mode of the original AIM65 is 20 characters/line, so more than double the display is possible. I feel that the printing is a little thinner than QR204. It looks better if you select bold (BOLD_ON)。

Print example with QR203

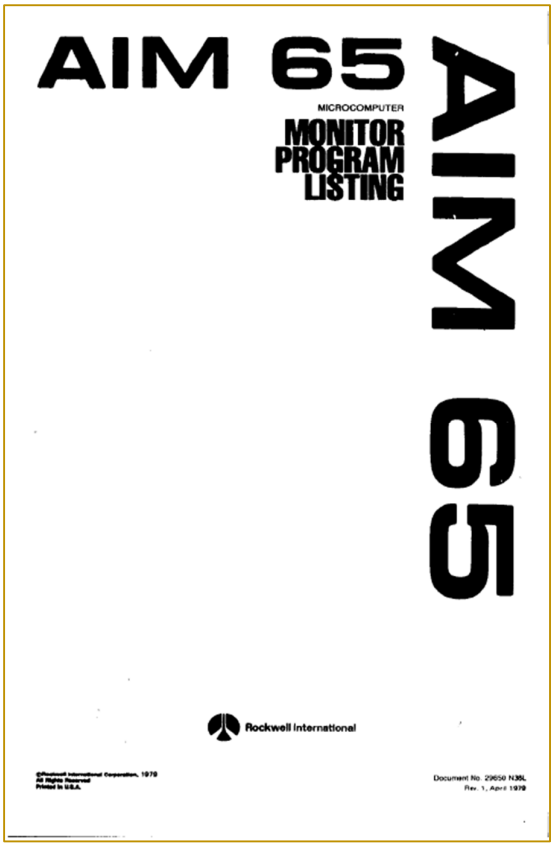
BOLD__ON	BOLD__ON FONT-B (9x17)



The QR204 cannot use 9x24 small fonts. If you really want to use 42 characters/line, I think it's better to put up with the noise and use QR203. Both can receive data at 9600bps, so it's faster than the original AIM 65. For example, even if you execute FORTH's VLIST (output a list of words), it will end in a blink of an eye. Paper is consumed quickly.

5.20 Patching of thermal printer control part

The thermal printer installed in AIM65-CPLD-3V3 is controlled by 9600bps serial interface of UART.



The original AIM 65 has a buffer of 20 characters, and outputs the contents of the buffer to the thermal printer when the buffer is full while outputting to the 16-segment LED, or when a CR (carriage return) is output.
There are also cases where only the LF (line feed) is controlled independently of the character buffer. Therefore, I looked for the patch application point from the MONITOR PROGRAM LISTING and applied the patch application to the following two entry points. It is assumed that the printer driver routines are located at addresses $AE00 and $AE2C of the reproduction AIM 65. The QR204 and QR203 are printers that can print 32 characters per line, but the AIM 65 KB mode only uses up to 20 characters.

(1) Location where the contents of the 20-character buffer (one line) are output to the thermal printer

Label and assembly notation	Address	Code		Patch	Code
IPS0 JSR PINT	F04A	20 CB F0	→	JSR $AE00	20 00 AE

(2) Where line feed is output

Label and assembly notation	Address	Code		Patch	Code
IP00 LDA #PRST+PS12+MON	F050	A9 C1 8D	→	JSR $AE2C	20 2C AE

Serial printers can be powered on/off at any time. In the original AIM 65, printing is executed based on the ON/OFF flag of the printer, but in the reproduction AIM65-CPLD-3V3, the program operates assuming that the printer is always ON. If the printer power is ON, it will print, and if the power is OFF, it will not print.

Listing MONITOR PROGRAM patching locations

```
F000                *=$F000
F000                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
F000                ;OUTPUT ACC TO PRINTER SUBROUTINE
F000                ;PRINTS ON 21RST CHAR OR WHEN <CR>
F000                ;IT WILL PUT IT ON BUBFFER BUT WONT PRINT IF
F000                ;PRIFLG=0
F000  48            OUTPRI PHA             ;SAVE CHR TO BE OUTPUT
F001  20 9E EB             JSR PHXY        ;SAVE X
F004  C9 0D                CMP #CR         ;SEE IF CR
F006  F0 07                BEQ OUT01       ;YES SO PRINT THE BUFF
F008  AE 16 A4             LDX CURPOS      ;PTR TO NEXT POS IN BUFF
F00B  E0 14                CPX #20         ;SEE IF BUFF FULL
F00D  D0 16                BNE OUT04       ;NOT FULL SO RETURN
F00F                ;<CR> SO FILL REST OF BUFFER WITH BLANKS
F00F  48            OUT01  PHA
F010  A9 00                LDA #0          ;CURPOS = 0
F012  AE 16 A4             LDX CURPOS      ;SEE IF ANYTHING IN BUFFER
F015  8D 16 A4             STA CURPOS
F018  20 38 F0             JSR OUTPR       ;CLEAR PRIBUF TO THE RIGHT
F01B                ;BUFFER FILLED SO PRINT IT
F01B  20 45 F0             JSR IPST        ;START THE PRINT
F01E  A2 00                LDX #0          ;STORE CHR IN BUFF (FIRST LOC)
F020  68                   PLA             ;GET IT
F021  C9 0D                CMP #CR         ;DONT STORE IF <CR>
F023  F0 0E                BEQ OUT05
F025  9D 60 A4      OUT04  STA IBUFM,X     ;STORE CHR IN BUFF
F028  EE 16 A4             INC CURPOS      ;INCR BUFF PNTR
F02B  E8                   INX
F02C  29 80                AND #$80

AIM65_monitor_list_TASM.txt  Page 34

F02E  D0 03                BNE OUT05       ;DONT CLR IF MSB=1
F030  20 38 F0             JSR OUTPR       ;CLEAR PRIBUFF TO THE RIGHT
F033  20 AC EB      OUT05  JSR PLXY        ;RESTORE REGS
F036  68                   PLA
F037  60                   RTS
F038  A9 20         OUTPR  LDA #' '        ;FILL REST OF BUFF WITH BLANKS
F03A  E0 14         OUTPR1 CPX #20         ;SEE IF END OF BUFF
F03C  F0 06                BEQ OUTPR2
F03E  9D 60 A4             STA IBUFM,X     ;NO SO STORE BLANK
F041  E8                   INX             ;INCR BUFF PNTR
F042  10 F6                BPL OUTPR1
F044  60            OUTPR2 RTS
F045
F045                ;SUB TO OUTPUT BUFFER, 70 DOTS (10 DOTS AT
F045                ;A TIME BY 7 ROWS) FOR EACH LINE OF PRINTING
F045  2C 11 A4      IPST   BIT PRIFLG      ;PRINT FLG ON ?
F048  10 2E                BPL IP04
F04A  20 CB F0      IPS0   JSR PINT        ;INITIALIZE VALUES          JSR $AE00
F04D  20 E3 F0             JSR IPSU        ;SET UP FIRS OUTPUT PATTERN
F050  A9 C1         IP00   LDA #PRST+SP12+MON ;TURN MOTOR ON           JSR $AE2C
F052  8D 0C A8             STA PCR
```

```
2417  F055  20  A0  FF          JSR PAT23       ;TIME OUT ?
2418  F058  D0  0C              BNE IPO2        ;NO, START SIGNAL RECEIVED
2419  F05A  20  A0  FF          JSR PAT23       ;YES, TRY AGAIN
2420  F05D  D0  07              BNE IPO2        ;OK
2421  F05F  4C  79  F0          JMP PRIERR      ;TWO TIME OUTS - ERROR
2422  F062  EA                  NOP
2423  F063  EA                  NOP
2424  F064  EA                  NOP
2425  F065  EA                  NOP
2426  F066  20  87  F0   IPO2   JSR PRNDOT      ;STRB P1=1 PRINT DOTS (1.7MSEC)
2427  F069  20  87  F0          JSR PRNDOT      ;STRB P2=1 PRINT DOTS (1.7MSEC)
2428  F06C              ;CHECK FOR 90, WHEN 70 PRNDOT WILL OUTPUT ZEROS
```

Listing MONITOR PROGRAM patching points (continued)

```
2429  F06C  AD  77  A4          LDA IDOT
2430  F06F  C9  5A              CMP #90
2431  F071  90  F3              BCC IP02      ;L.T. 90 THEN GO STROB P1
2432  F073  A9  E1        IP03  LDA #PRST+SP12+MOFF ;TURN MOTOR OFF
2433  F075  8D  0C  A8          STA PCR
2434  F078  60             IP04  RTS
2435  F079
2436  F079  20  44  EB    PRIERR JSR CLR       ;CLEAR PRI PNTR
2437  F07C  20  B1  FE           JSR PATCH5    ;TURN PRI OFF
2438  F07F  A0  3B               LDY #M12-M1
2439  F081  20  AF  E7          JSR KEP
2440  F084  4C  A1  E1          JMP COMIN     ;BACK WHERE SUBR WAS CALLED
2441  F087
```

Listing EPROM CODE patching location

```
F000 : 48 20 9E EB C9 0D F0 07 AE 16 A4 E0 14 D0 16 48
F010 : A9 00 AE 16 A4 8D 16 A4 20 38 F0 20 45 F0 A2 00
F020 : 68 C9 0D F0 0E 9D 60 A4 EE 16 A4 E8 29 80 D0 03
F030 : 20 38 F0 20 AC EB 68 60 A9 20 E0 14 F0 06 9D 60
F040 : A4 E8 10 F6 60 2C 11 A4 10 2E 20 CB F0 20 E3 F0
F050 : A9 C1 8D 0C A8 20 A0 FF D0 0C 20 A0 FF D0 07 4C
F060 : 79 F0 EA EA EA EA 20 87 F0 20 87 F0 AD 77 A4 C9
F070 : 5A 90 F3 A9 E1 8D 0C A8 60 20 44 EB 20 B1 FE A0
F080 : 3B 20 AF E7 4C A1 E1 A9 00 8D 01 A8 AD 0D A8 29
```

20 00 AE

20 2C AE

Writing Printer Drivers

Of the I/O area $A000 to $AFFF, the 512 bytes from $AE00 to $AEFF are used to allocate additional software such as printer driver routines.

The serial printer receives ASCII code via 9600bps UART RX signal, stores it in the buffer, and prints it when CR (carriage return) code is received.. The UART-TX macro built into the PIA2 can parallel-serial convert and transmit at 9600bps by writing an 8-bit code to the TXD register. Therefore, the printer control program is not an exaggerated driver routine, and can be controlled by simply checking the BUSY flag of the UART-TX macro and writing the ASCII code to the TXD register.

The following listing shows the printer control program
UATX is a routine that outputs the ASCII code held in the A register to TXD.
PRTOUT is a routine that prints the contents of a 20-character buffer. Calling UATX (20 times) for each character。

LFOUT is a line feed routine. Called when she presses the LF key on the QWERTY keyboard. In the case of the QR204 printer, just receiving the CR code $0D will feed the line after printing, but the QR203 will print CR($0D) when there is no character data in the buffer.

Listing Printer control program

```
1     ; ****************************************
2     ; *  THERMAL PRINTER ROUTINE ( UART-TX )   *
3     ; ****************************************
4     ;
5     TXD    = $AC04 ; TXD REG ADDRESS (CPLD PIA2)
6     BUSYF  = $AC05 ; BUSY FLAG( CPLD PIA2 BIT7 )
7     IBUFM  = $A460 ; PRINT BUFFER (20 BYTES)
8     PHXY   = $EB9E ; PUSH X,Y ROUTINE
9     PLXY   = $EBAC ; PULL X,Y ROUTINE
10    CHRCNT = $A474 ; VARIABLE FOR CHARACTER COUNT
11    WORKX  = $A475 ;
12    ;
13    ; ENTRY ADDRESS
14      *=$AE00
15    ; OUTPUT BUFFER( 20 BYTES )
16    ; TO THERMAL PRINTER
17    PRTOUT JSR PHXY     ; PUSH X,Y
18           LDX #20      ; PRINT 1 LINE
19           STX CHRCNT
20           LDX #0
21           STX WORKX
22    B1     LDX WORKX
23           LDA IBUFM,X
24           AND #$7F
25           JSR UATX     ; TX ONE BYTE
26           INC WORKX
27           DEC CHRCNT
28           BNE B1
29           LDA #$0D     ; OUTPUT CARIDGE RETURN
30           JSR UATX     ;
31           JSR PLXY     ; PULL X,Y
32           RTS;
33      *=$AE2C
34    LFOUT  JSR PHXY     ; PUSH X,Y
35           LDA #$20     ; OUTPUT SPACE
36           JSR UATX
37           LDA #$0D     ; OUTPUT CARIDGE RETURN
38           JSR UATX
39           JSR PLXY
40           RTS
41
42    UATX   TAY          ; TX ONE BYTE
43    UA1    LDA BUSYF    ; WAIT FOR BUSY FLAG=0
44           ROL A        ;
45           BCS UA1      ;
46           STY TXD      ;
47           RTS;
48
```

5.21 Power supply for printer

My recommendation is the following AC adapter with selectable voltage. I think most thermal printers operate on 9V, but once I ordered a QR204, it was sent with a 12V power supply. If you have a voltage switching type AC adapter, it can also support 12V.

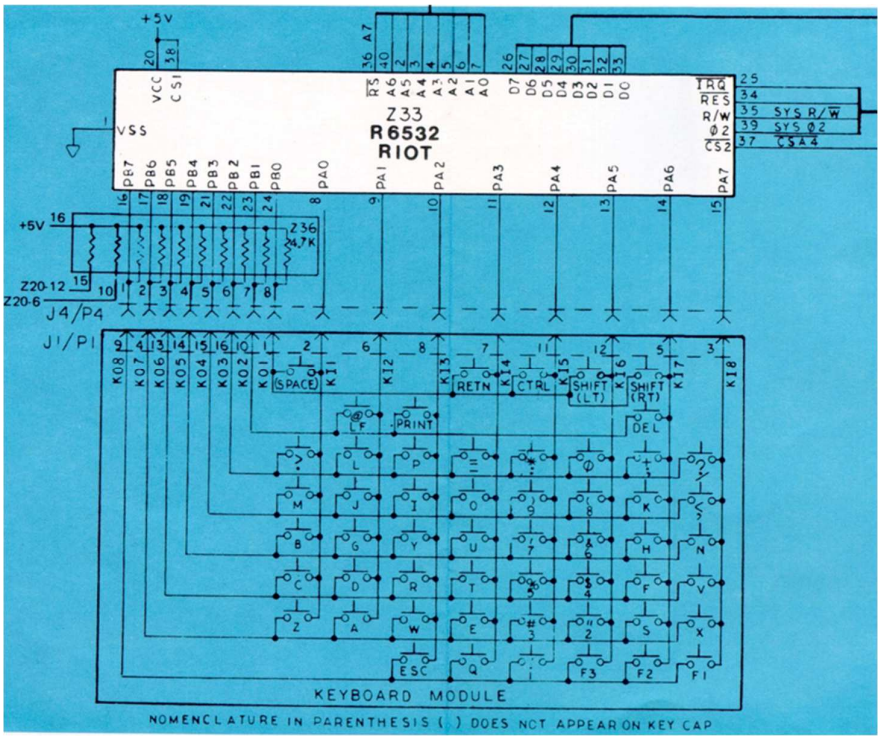
	
SoulBay 3V~12V, 2A	BOLWEO 40W 3V 4.5V 5V 6V 7.5V 9V 12V 3A

A power capacity of 2A is required. Chinese products for 2A and 3A are sold at low prices. However, even if the ones shown below are listed as 2A or 3A, the capacity is small and they do not work properly. I suspected that it was caused by the large current fluctuation of the printer, so I increased the smoothing capacitor to 1000uF, but it did not work at all.

Example of an AC adapter that did not

5.22 Keyboard interface



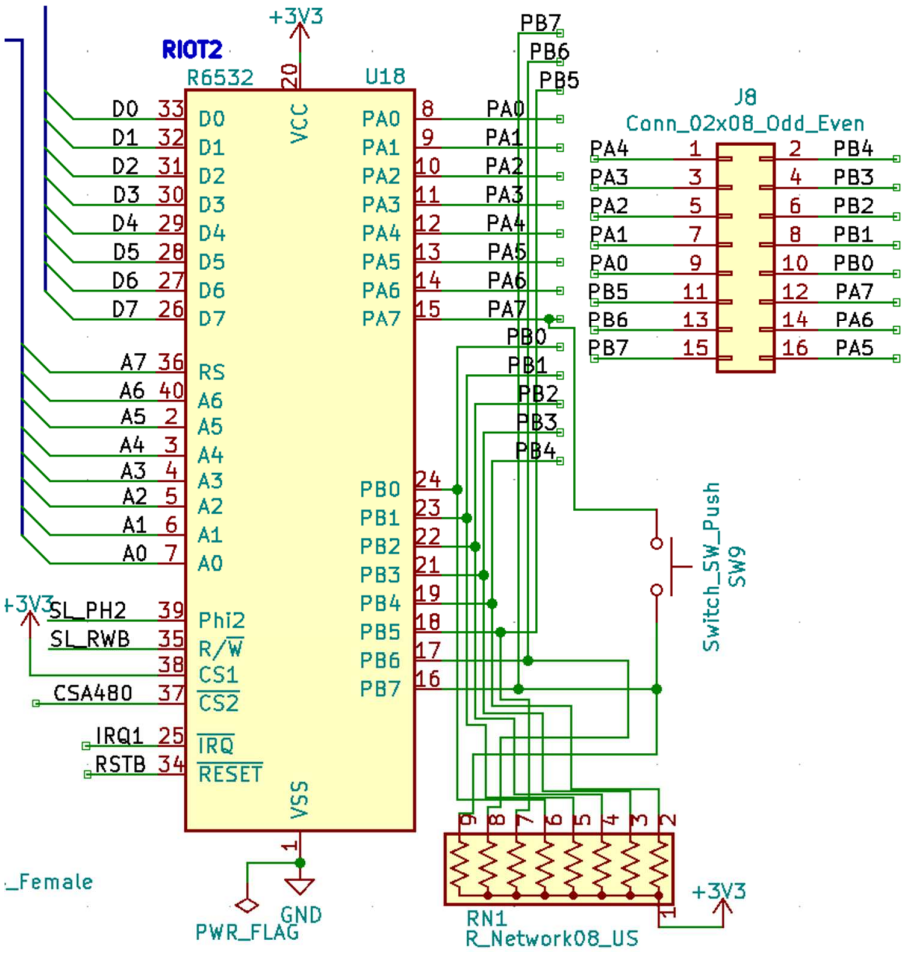
The AIM 65 keyboard interface circuit uses the R6532 RIOT's port A as output and port B as input to form a key matrix, which is scanned by software.

The keyboard control circuit of AIM65-CPLD-3V3 is the same as the original, only the operating voltage is different. Enabled when KB mode is selected with the KB---TTY switch on the mainboard. If the TTY mode is selected and characters are input/output with TeraTerm on the PC via the USB serial interface, the main keyboard is unnecessary. However, it seems that the BREAK operation in BASIC is useless unless it is the F1 key on the keyboard of the main unit. Correspondence between main keyboard and TTY keyboard in AIM-65 USER'S GUIDE、

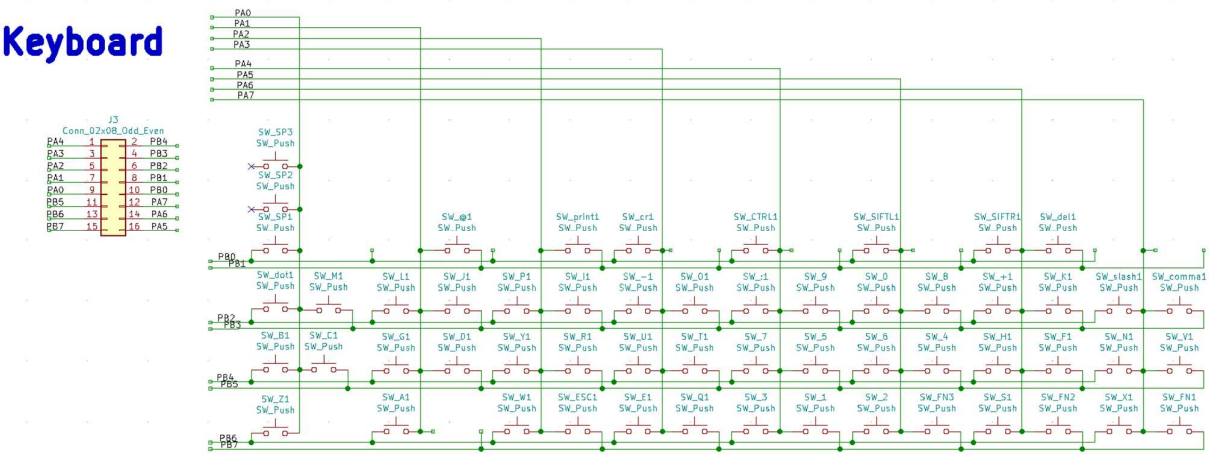
F1 [
F2]
F3 ^
However, I was not able to cause a BREAK with the [key while running a BASIC program using TeraTerm. For this reason, by implementing a tact switch corresponding to the F1 key on the main board of the AIM-65 reproduction, we designed the board so that BREAK can be generated without a QWERTY keyboard. On the other hand, if the contents of the memory have not been rewritten due to program runaway, press the reset button, set the baud rate with the DEL key, and then re-enter BASIC with the monitor command 6 key to enter the BASIC command

input state. Since it can be returned, even if there is no BREAK key, it will not be a big disadvantage。

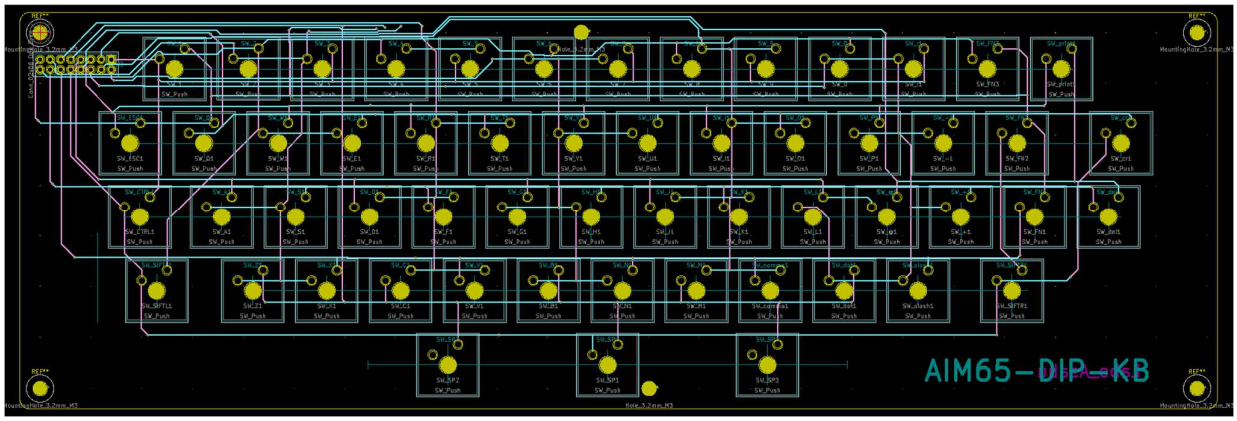
図〇〇 AIM65-CPLD-3V3 RIOT まわりの回路図



図〇〇 AIM65-DIP-KB 回路図



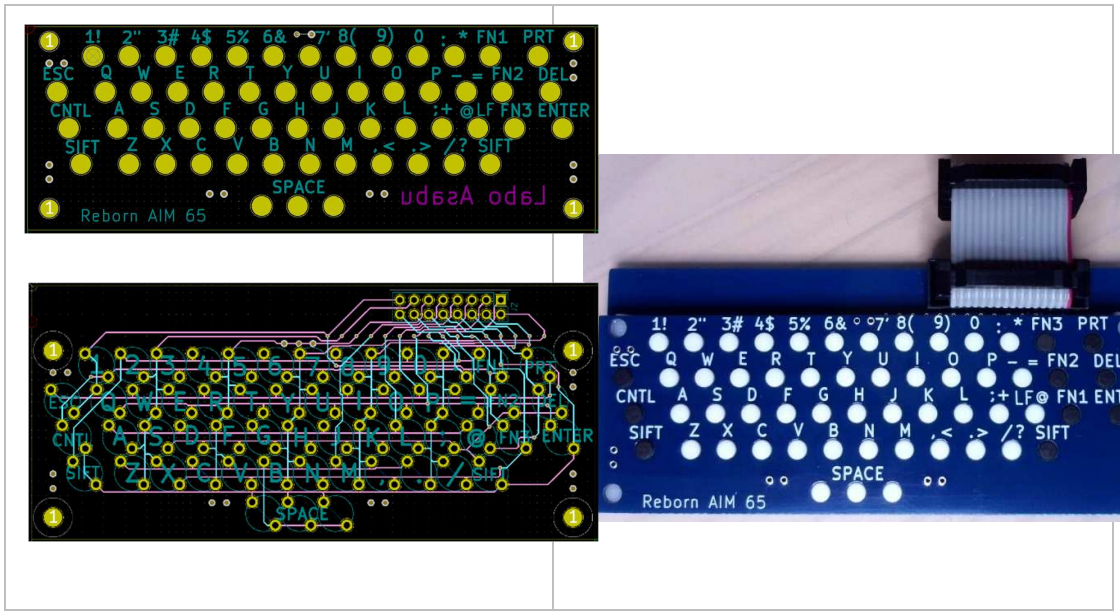
AIM65-DIP-KB board layout and keyboard photo after assembly



The figures show the layout of the printed circuit board of the keyboard. Schema data of AIM65-DIP-KBD is put into the layout as it is. I designed a printed circuit board, actually purchased key switches and keycaps, and created a QWERTY keyboard. However, I couldn't find a keycap that perfectly matched the keyboard for AIM 65. In addition, it seems that there is no place that undertakes custom-made production in small quantities and at low cost. For those who need a keyboard, I can provide a printed circuit board, so please select the appropriate one from the various types of key switches and key caps sold for PCs.

A QWERTY keyboard is necessary when creating a formal replica of the AIM 65, but TTY mode with a wider display area is easier to use when creating programs for actual use. In the reproduction version, KB mode is no longer necessary because you can print to a thermal printer simply by writing ASCII data to the UART register without using KB mode.

I also created a compact type MINI keyboard for those who rarely use it but need a keyboard.

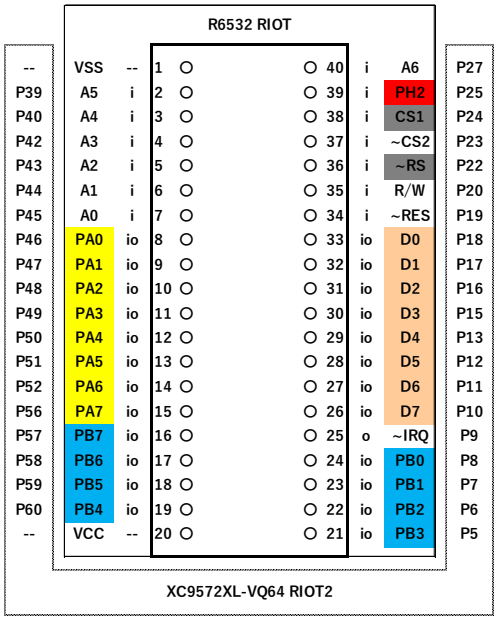


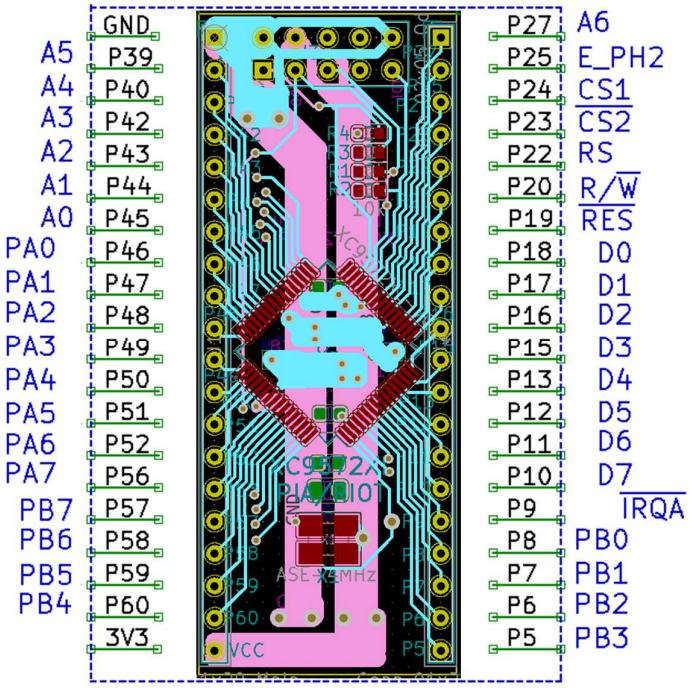
When creating the MINI keyboard, I choose tactile keys that are not only small but also increase the mounting density. If the mounting density is increased, there will be no space for silk-printing the key name. So I made another printed board just for the purpose of silk-printing the key name. If the height of the tactile key cap is slightly short and the plate is covered, it will not be possible to increase the pushing depth. I had no choice but to thin the printed board down to 0.8mm. Thanks to this, the cover sideboard costs more to manufacture than the base printed board.

RIOT2 Verilog-HDL description and equivalent circuit

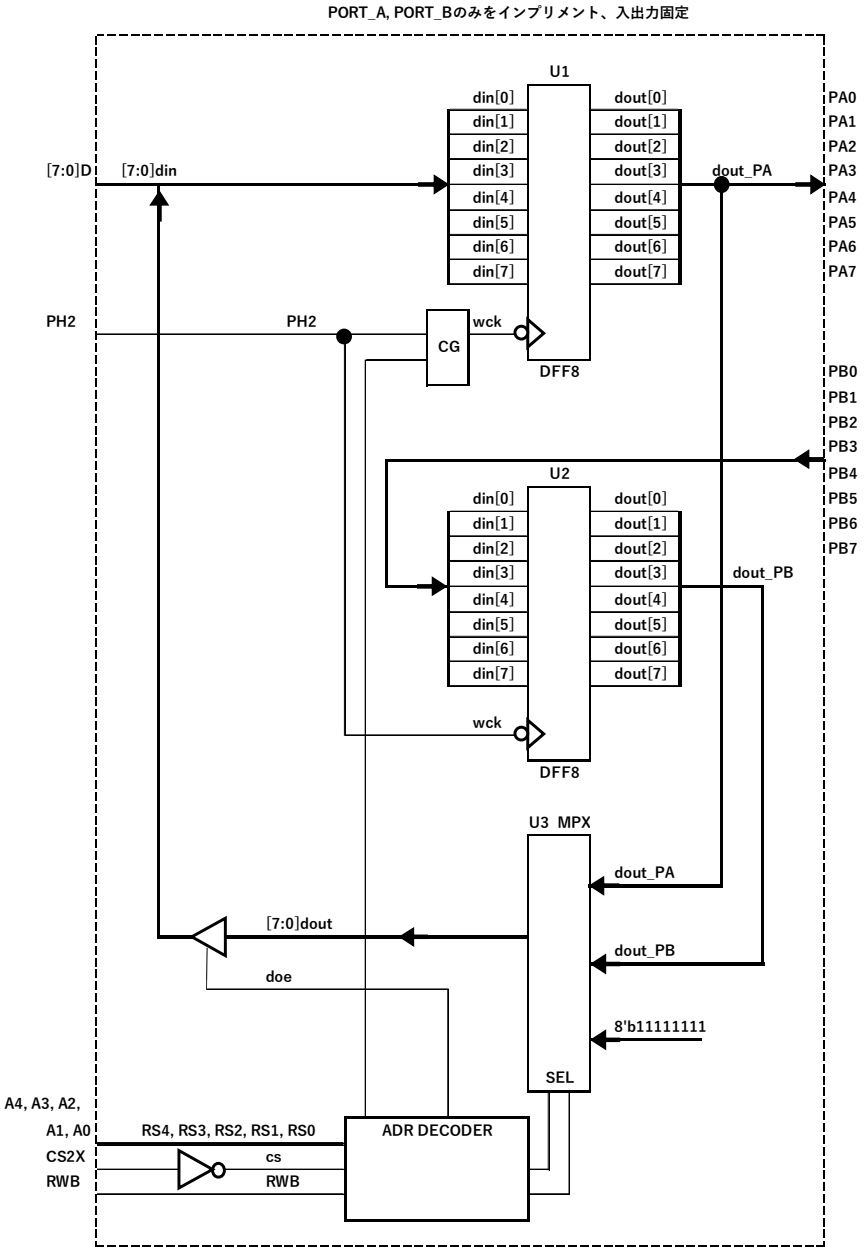
The RIOT2 DIP conversion board is the same as the PIA2 DIP conversion board. RIOT and PIA have the same power supply terminal position, so both can be handled by changing the logic to write to the CPLD. Furthermore, if a CPLD with a large number of macrocells becomes available, a conversion board with common VIA can be made.

RIOT2 terminal arrangement and DIP conversion board layout diagram





A RIOT should have a built-in timer for cassette interface in addition to port A and port B, but currently it only has port A and port B for keyboard interface. Moreover, port A is implemented with a fixed output and port B with a fixed input.



Verilog-HDL description listing XC9572XL_VQ64_RIOT_2 (1/2)

```
1    ////////////////////////////////////////////////////////////////////////////////
2    // Company: Labo Asabu
3    // Create Date: DEC-10/2022
4    // Design Name: XC9572XL_VQ64_RIOT_2
5    // Target Devices: XC9572XL-VQ64,
6    // Revision:
7    ////////////////////////////////////////////////////////////////////////////////
8    // adr decode
9    //   00000 : PA[7:0] output                                    : $A480
10   //   00001 : control_A  dummy , read=$FF  (NOT IMPLEMENTED)    : $A481
11   //   00010 : PB[7:0] input                                     : $A482
12   //   00011 : control_B  dummy , read=$FF  (NOT IMPLEMENTED)    : $A483
13
14   //   00100 : RD TIMER,         disable int.                    : $A484
15   //   00101 : RD INT_FLAG, &CLEAR                               : $A485
16   //   00110 : RD TIMER,         disable int.                    : $A486
17   //   00111 : RD INT_FLAG, &CLEAR                               : $A487
18
19   //   00100 : WR      neg_edge, disable_int. (NOT USED)         : $A484
20   //   00101 : WR      pos_edge, disable_int. (NOT USED)         : $A485
21   //   00110 : WR      neg_edge, enable_int.  (NOT USED)         : $A486
22   //   00111 : WR      pos_edge, enable_int.  (NOT USED)         : $A487
23
24   //   01000 :                                (NOT USED)         : $A488
25   //   01001 :                                (NOT USED)         : $A489
26   //   01010 :                                (NOT USED)         : $A48A
27   //   01011 :                                (NOT USED)         : $A48B
28   //   01100 : RD TIMER          enable int.                     : $A48C
29   //   01101 : RD INT_FLAG, &CLEAR                               : $A48D
30   //   01110 : RD TIMER          enable int.                     : $A48E
31   //   01111 : RD INT_FLAG, &CLEAR                               : $A48F
32
33   //   10000 :                                (NOT USED)         : $A490
34   //   10001 :                                (NOT USED)         : $A491
35   //   10010 :                                (NOT USED)         : $A492
36   //   10011 :                                (NOT USED)         : $A493
37   //   10100 : WR TIMER  div1    disable int.                    : $A494
38   //   10101 : WR TIMER  div8    disable int.                    : $A495
39   //   10110 : WR TIMER  div64   disable int.                    : $A496
40   //   10111 : WR TIMER  div1024 disable int.                    : $A497
41
42   //   11000 :                                (NOT USED)         : $A498
43   //   11001 :                                (NOT USED)         : $A499
44   //   11010 :                                (NOT USED)         : $A49A
45   //   11011 :                                (NOT USED)         : $A49B
46   //   11100 : WR TIMER  div1    enable int.                     : $A49C
47   //   11101 : WR TIMER  div8    enable int.                     : $A49D
48   //   11110 : WR TIMER  div64   enable int.                     : $A49E
49   //   11111 : WR TIMER  div1024 enable int.                     : $A49F
50
51
52   module XC9572XL_VQ64_RIOT(
53     PH2, RWB,  A0, A1, A2, A3, A4, CS2X,
```

```
54      D, PA, PB
55    );
56
57    input  PH2;
58    input  RWB;
59    // input  RESB;
60    input  A4, A3, A2, A1, A0;
61    input  CS2X;
62    inout  [7:0] D;
63    output [7:0] PA;
64    input  [7:0] PB;
65
```

Verilog-HDL description listing XC9572XL_VQ64_RIOT_2 (2/2)

```
66    wire  [7:0] din;
67    wire  [7:0] dout;
68    wire  doe;
69    assign D = doe ? dout : 8'bz;
70    assign din = D;
71
72    assign cs = ~CS2X ;
73    wire PA_sel, PB_sel;
74    wire wck_PA;   //write clock
75
76    assign PA_sel = cs &  RWB & ~A4 & ~A3 & ~A2 & ~A1 & ~A0 ;
77    assign PB_sel = cs &  RWB & ~A4 & ~A3 & ~A2 &  A1 & ~A0 ;
78    assign PA_we  = cs & ~RWB & ~A4 & ~A3 & ~A2 & ~A1 & ~A0 ;
79    // assign PB_we  = cs & ~RWB & ~RS2 &  RS1 & ~RS0;
80
81    wire  [7:0] dout_PA;
82    wire  [7:0] dout_PB;
83
84    assign PA = dout_PA;
85    assign doe = cs & RWB & PH2;
86
87    CG U4( .en( PA_we ), .ck( PH2 ), .gck( wck_PA ) );
88    // CG U5( .en( PB_we ), .ck( PH2 ), .gck( wck_PB ) );
89
90    DFF8 U1( .din(din), .wck(wck_PA), .dout(dout_PA) );
91    DFF8 U2( .din(PB), .wck(PH2), .dout(dout_PB) );
92
93    MPX  U3( .a(dout_PA), .b(dout_PB), .asel(PA_sel), .bsel(PB_sel), .mpxout(dout) );
94
95    endmodule
96
97    module CG( en, ck, gck );  // Clock Gating cell
98      input en, ck;
99      output gck;
100     wire A, G, Q;
101
102     assign G = ck;
103     assign A = en;
104     assign Q = G ? A : Q;
105     assign gck = Q & G & A;
```

```
106  endmodule
107
108  module DFF8( din, wck, dout );  // 8bit D-FF
109    input  wck ;
110    input  [7:0] din;
111    output [7:0] dout;
112    reg    [7:0] dout;
113
114    always @( negedge wck )
115           dout <= din ;
116  endmodule
117
118  module MPX( a, b, asel, bsel, mpxout );  //
119    input [7:0] a, b;
120    input asel, bsel;
121    output [7:0] mpxout;
122    reg [7:0] mpxout;
123
124    always @(a or b or asel or bsel )
125    if(asel) mpxout = a;
126    else if(bsel) mpxout = b;
127  //  else mpxout = 8'b00000000;
128    else mpxout = 8'b11111111;
129  endmodule
```

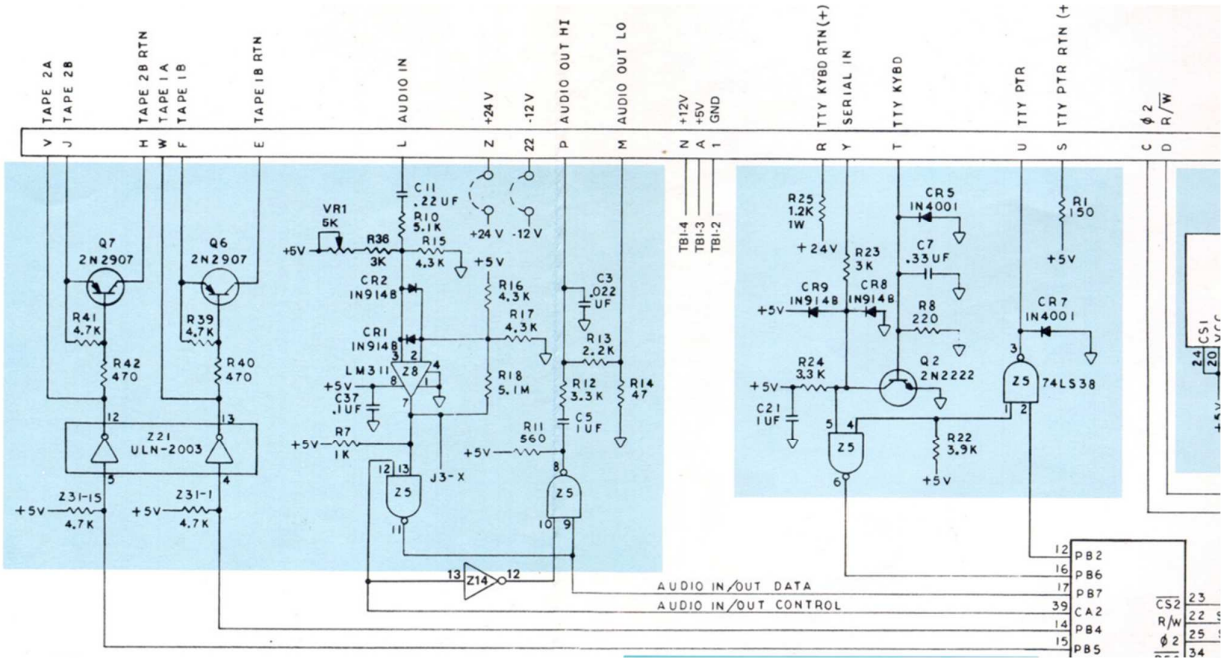
Description of .ucf file for C9572XL_VQ64_RIOT_2

```
1    NET "A5"        LOC = "P39";
2    NET "A4"        LOC = "P40";
3    NET "A3"        LOC = "P42";
4    NET "A2"        LOC = "P43";
5    NET "A1"        LOC = "P44";
6    NET "A0"        LOC = "P45";
7    NET "PA[0]"     LOC = "P46";
8    NET "PA[1]"     LOC = "P47";
9    NET "PA[2]"     LOC = "P48";
10   NET "PA[3]"     LOC = "P49";
11   NET "PA[4]"     LOC = "P50";
12   NET "PA[5]"     LOC = "P51";
13   NET "PA[6]"     LOC = "P52";
14   NET "PA[7]"     LOC = "P56";
15   NET "PB[7]"     LOC = "P57";
16   NET "PB[6]"     LOC = "P58";
17   NET "PB[5]"     LOC = "P59";
18   NET "PB[4]"     LOC = "P60";
19
20   NET "A6"        LOC = "P27";
21   NET "PH2"       LOC = "P25";
22   // NET "xxx"      LOC = "P24";
23   NET "CS2X"      LOC = "P23";
24   // NET "xxx"      LOC = "P22";
25   NET "RWB"       LOC = "P20";
26   NET "RESB"      LOC = "P19";
27   NET "D[0]"      LOC = "P18";
28   NET "D[1]"      LOC = "P17";
29   NET "D[2]"      LOC = "P16";
30   NET "D[3]"      LOC = "P15";
31   NET "D[4]"      LOC = "P13";
32   NET "D[5]"      LOC = "P12";
33   NET "D[6]"      LOC = "P11";
34   NET "D[7]"      LOC = "P10";
35   // NET "IRQB"      LOC = "P9";
36   NET "PB[0]"     LOC = "P8";
37   NET "PB[1]"     LOC = "P7";
38   NET "PB[2]"     LOC = "P6";
39   NET "PB[3]"     LOC = "P5";
```

5.23 Cassette Tape Interface

The AIM65-CPLD-3V3 does not have a cassette tape interface.

Original AIM65 cassette tape interface circuit



R6522 VIA

Z5: 74LS38 NAND (Open collector))

ULN2003 Transistor Array 50V,500mA

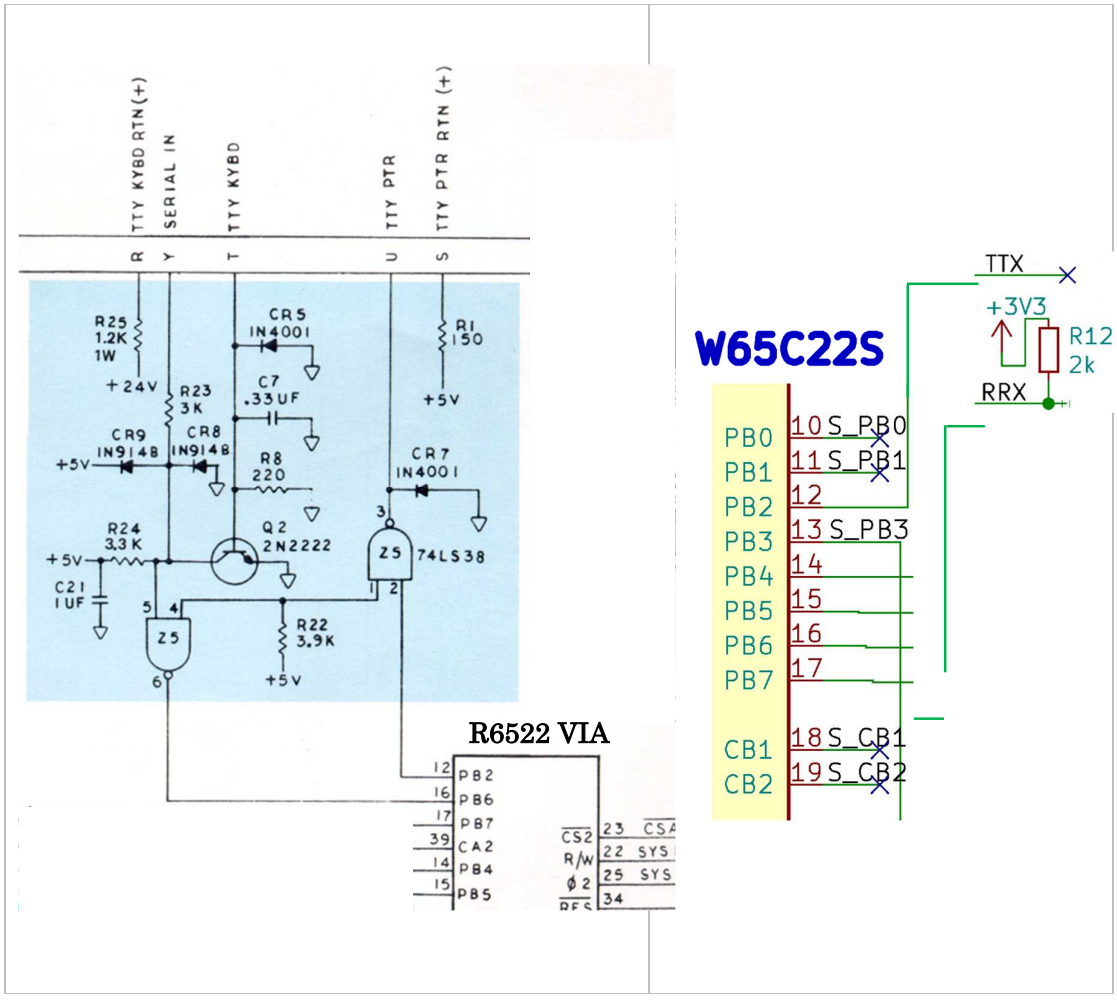
NPN Darlington transistor array

The original AIM 65 cassette tape interface is shown in the diagram above. In addition to this circuit, VIA's built-in timer and RIOT's built-in timer are required. The 5V version reproduction can be equipped with the R6532A RIOT, so if you add the above circuit, you can use the software as it is and use the cassette tape. Since 3.3V version RIOT does not exist in 3.3V version reproduction, it is necessary to describe RIOT timer in CPLD. Restoring the cassette tape interface and increasing the CPU clock to 8MHz does not mean that the cassette interface can also be eight times faster, so the RIOT timer has not yet been described in Verilog-HDL.

5.24 Teletype interface

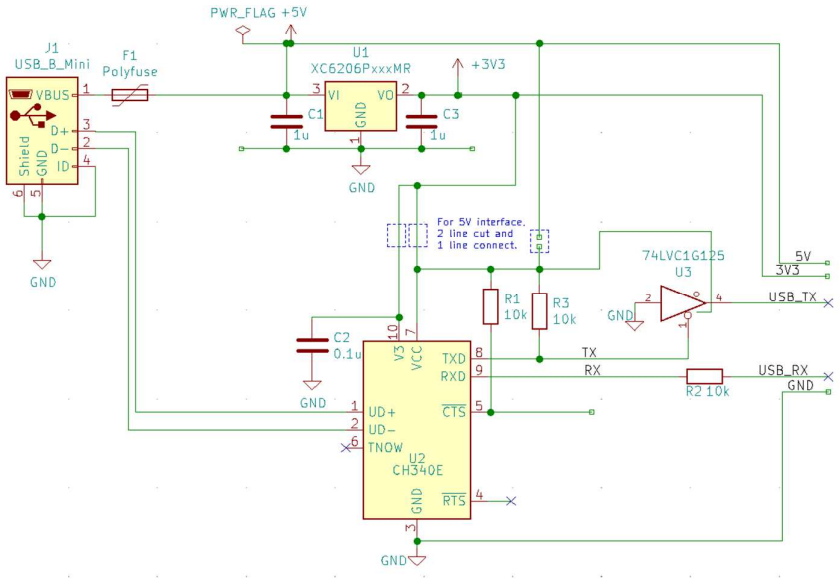
The original AIM 65 had a TTY interface with a 20mA current loop. The reproduction version AIM65-CPLD-3V3 does not have a current loop circuit, and the VIA PB2 and PB6 pins are connected to the USB serial conversion board as TTL level UART signals.

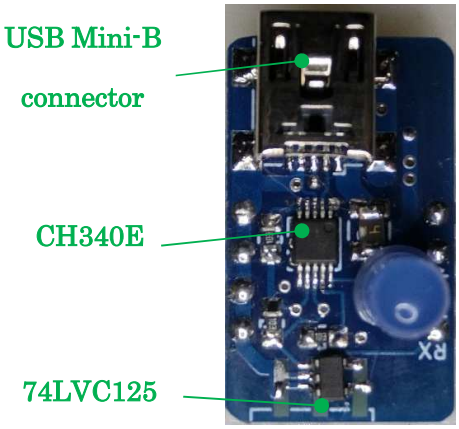
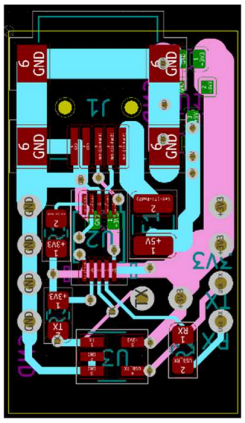
TTY interface



USB serial conversion board

USB serial conversion board circuit diagram



USB serial conversion board	Layout diagram of USB serial conversion board
	

The USB serial conversion board is a board that converts the UART signal on the microcomputer side to USB and connects it to a PC. Various types of USB serial conversion boards are sold as general products, and any of them can be used as long as the signal level is TTL or 3.3V CMOS level. However, in order to make it easier to use for AIM65-CPLD-3V3, I made a new USB serial conversion board using CH340E.

1. How to make power-on reset work properly.

As for the order of turning on the power of AIM65-CPLD-3V3, first start up the PC, connect the USB serial conversion board to the PC, set the communication parameters, and then turn on the power of AIM65-CPLD-3V3. is normal. However, the "H" level of the TX signal of the USB serial conversion board is transmitted to the VIA terminal of the AIM 65, and the voltage is applied halfway to the entire board due to power supply leakage (the power indicator LED lights up faintly). To do). If you turn on the AIM65-CPLD-3V3 in this state, the power-on reset may not work correctly, and you will have to press the reset button every time you turn on the power. This makes me feel uncomfortable, so I used a 74LVC1G125 tri-state buffer as a pseudo-open drain for the TX signal so that it doesn't drive the "H" level. It is not suitable for high-speed operation, but there is no problem if it is about 9600bps. Of course, the RX on the microcomputer side connected to TX needs to be pulled up.

2. Added LED to indicate data transfer status

AIM65-CPLD-3V3 does not have cassette IF. Instead, use the TTY's ability to read from paper tape. At that time, the L command of the monitor, the LOAD command of BASIC, and the SOURCE command of FORTH do not produce sound like the cassette IF, and it is difficult to determine the end of the file transfer. As a countermeasure, the LED blinks while the AIM65 receives data.). If the LED is completely off, data reception is complete.

When loading MOS Technology papertape format data using the monitor's L command using Teraterm's file transfer function, add 6 blank lines at the end of the data, i.e. send 6 CR ($0D) automatically to the L command will end. It is the same as typing ⏎ (Enter) on the keyboard six times when finished.

With BASIC's LOAD command, after transferring a file with Teraterm, input Cntl-Z from the keyboard to terminate LOAD (it should have been written somewhere in the manual).

If you place FINIS at the end of the program file when executing the SOURCE command of FORTH and reading the program by file transfer, the SOURCE command will end after reading FINIS.

6 PRODUCTION

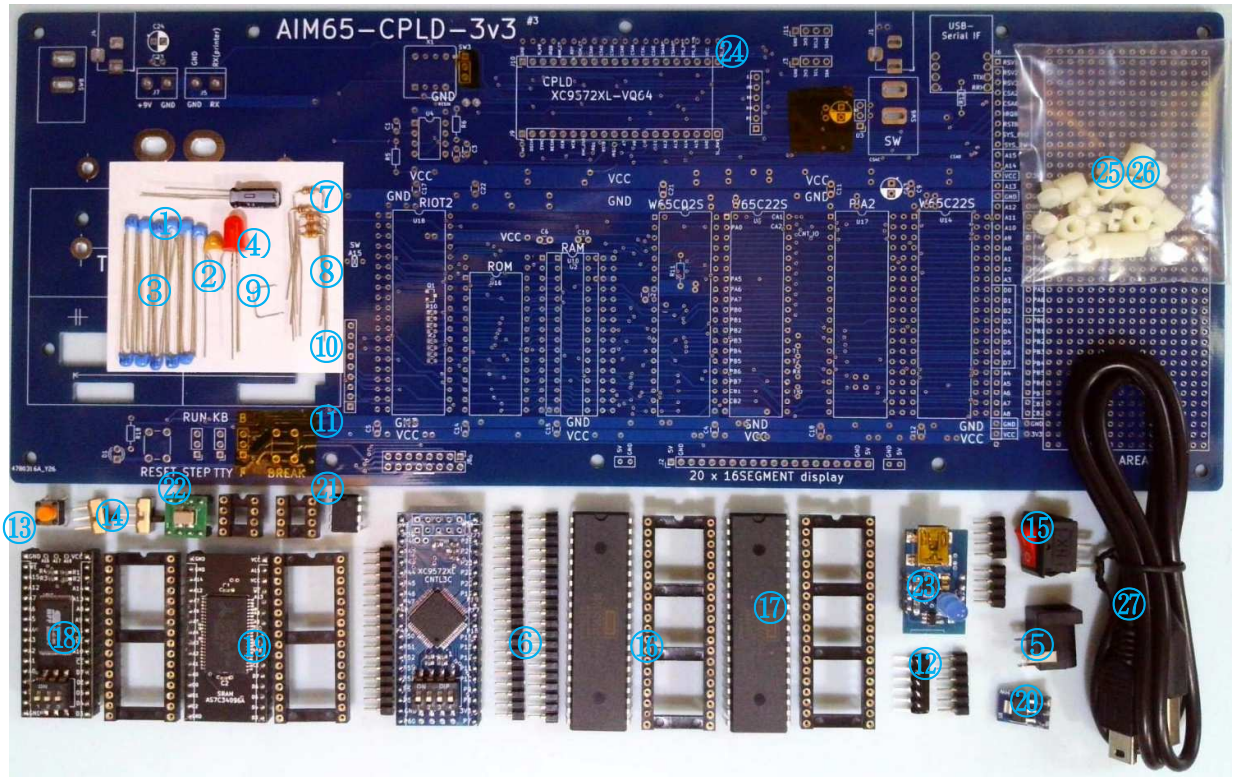
The procedures for making the board from minimal to full configuration is described in this chapter.

6.1 Minimal configuration

1) Confirm that all parts are present..

Once confirmed, solder the parts with the lowest height first.

Parts drawing of the minimum configuration



Reference	Note		個数	Correspondens to numbers in figure
C13	220u		1	①
C2	0.01u		1	②
C1, C3, C6, C4, C11, C14, C15, C19, C20, C21, C22,		0.1u	11	③
D1	LED (with builtin resistor)		1	④
J1	Conn_Coaxial_Power		1	⑤
J9, J10	Conn_01x20_Connector fot CPLD		2	⑥
R11	10k		1	⑦
R12	2k		1	⑧
R18	0 Ω		1	⑨
R5, R6	1M		2	⑩
R20	8.2k		1	⑪
RN2	Resistor 5 pack 6 pin		1	⑫
SW1	Tactile switch for reset		1	⑬
SW2, SW4	Slide switch		2	⑭
SW6	Main power switch		1	⑮
U1	CPU: W65C02S6TPG-14		1	⑯

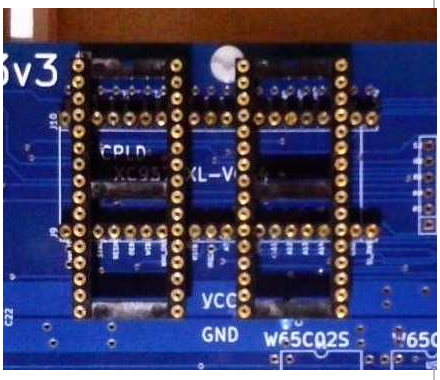
U5	VIA: W65C22S6TPG-14 system	1	⑰
U16	FLASH,. DIP32 conversion board	1	⑱
U10	SRAM, DIP32 conversion board	1	⑲
U3	LM1117-3.3 module	1	⑳
U4	NE555N 3.3V	1	㉑
X1	Oscillator	1	㉒
no number	USB serial conversion module (using CH340E)	1	㉓
no number	Printed circuit board AIM65-CPLD-3V3	1	㉔
no number	plastic screws M3x4mm	10	㉕
no number	plastic nut M3 x 6mm	10	㉖
no number	USB cable, mini-B	1	㉗

2) Attach a 4mm plastic screw and a 6mm plastic nut to the printed circuit board.
If you use a 3mm thick acrylic board as the baseboard, we assume that you will use 6mm plastic screws for the screws on the baseboard side. If the thickness of the printed circuit board of the main board is 1.6mm and the thickness of the acrylic board is 3mm, the upper and lower screws (4mm and 6mm) fit in the 6mm nuts, which is convenient.

3) Solder the resistors and capacitors.
It is convenient to attach R11(10k) and R12(2k) to the back side of the board so that they do not interfere with other parts.
R20 (8.2k) is not shown in the circuit diagram, but is soldered between the KB/TTY selector switch and 3V3 on the back of the board. It is not necessary when using only in TTY mode, but it is necessary when switching to KB mode.

4) Implement sockets.
AIM65-CPLD-3V3 is based on inserting a round pin header into a round pin socket. Please note that normal size pin headers cannot be inserted.

It is recommended to fix the pitch of the two SIP sockets with the DIP sockets and mount them before the DIP sockets. It can be arranged perpendicular to the pitch between pins and the board. This is because the SIP socket may tilt or have an incorrect pin-to-pin pitch during mounting. Check the verticality before soldering, such as temporarily fixing the SIP socket for the collective resistor with masking tape.



As long as the position and direction of the DIP socket match, the pitch on both sides is fixed and it can be mounted vertically on the board surface.

The width (pin spacing) on both sides of the SRAM AS7C34096A DIP conversion board is 2.54mm wider than the normal DIP32, so add an additional SIP socket to the right outside of the DIP32 socket.

5) Solder the switch and power connector

Check the continuity between 3.3V and GND to confirm that there is no continuity (the current flows momentarily because the capacitor is attached).

6) Solder the LED and AMS1117 LDO module.
Be careful with the polarity of the LED. Only the side of the GND terminal is flat.
Soldering is finished at this point.

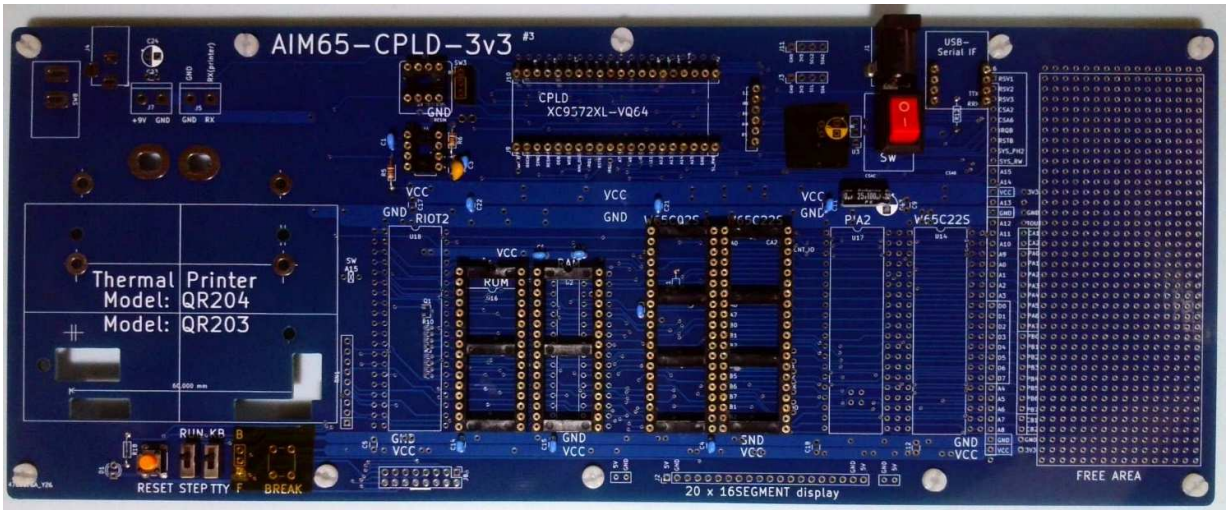
Connect the 5V power supply (AC adapter) and turn on the power switch. Confirm that the power confirmation LED lights up. Also, measure the voltage between 3.3V and GND with a tester and confirm that 3.3V is output.

If you have a multimeter, connect it in series with the AC adapter and check that the needle swings slightly (about 10mA).

7) Plug in the oscillation module, turn on the power, and observe the clock waveform of the OSCIN signal. It is OK if you can observe the 8MHz square wave correctly.
At this time, the current is ~mA.

8) Plug in the control CPLD board and turn on the power.
Observe the waveforms of SYS_PH2 and SL_PH2, and if the waveforms of 8MHz and 2MHz are observed, it is OK.
At this time, the current is ~mA.

9) Turn off the power once and insert other parts into the sockets.
The minimal system is now complete.



10) Teraterm connection, see the chapter on using Teraterm.

11) System startup

The supply current when operating with the minimum configuration CPU/ROM/RAM: 16MHz, peripheral: 4MHz is about 70mA when starting BASIC and waiting for a command.
It is about 80mA without a big change even when Mandelbrot's ASCII art program is running.

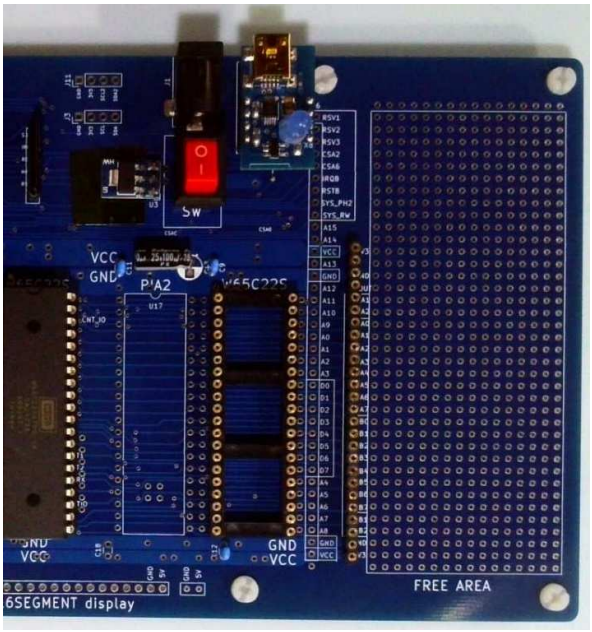
6.2 Production of base with VIA system--

1) Implementation of VIA for user extension

C9, C12	0.1u	bypass capacitor
J20	Conn_01x26_Female	Connector for user VIA signals (not shown in schematic)
U14	VIA:	W65C22S6TPG-14, for user extension

Socket mount VIA W65C22S for user expansion. Along with that, we will implement a SIP socket next to the FREE AREA for VIA interface signal connection.

This allows you to place speakers and LEDs in the FREE AREA for demonstrations using user VIA.



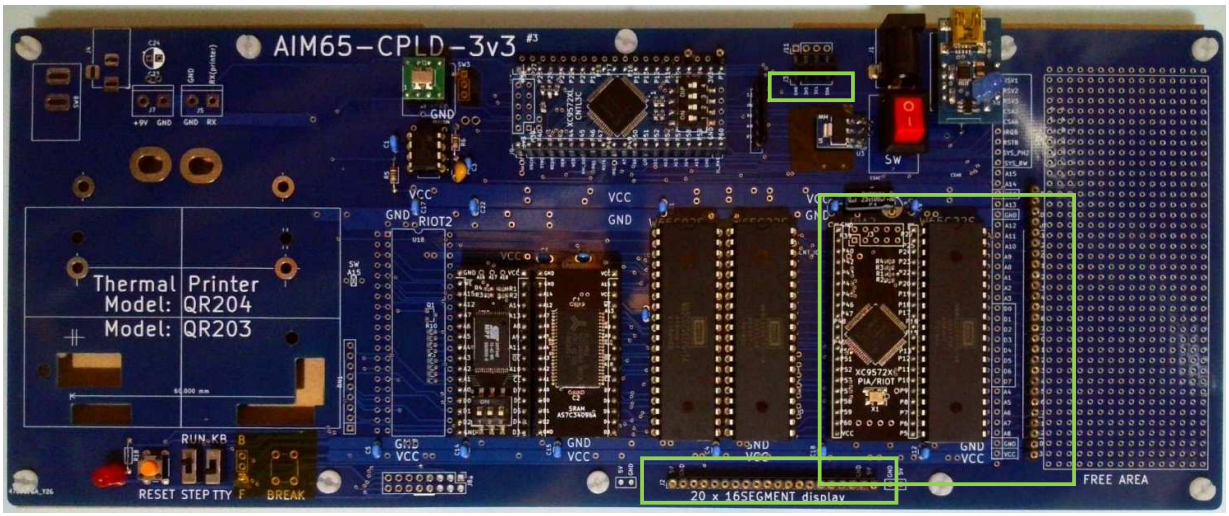
6.3 Production of a full system

1) Implementation of PIA2

C5	0.1u	bypass capacitor	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C17	0.1u	bypass capacitor	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C18	0.1u	bypass capacitor	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
U17	PIA2: DISP,PTR	XC9572XL-VQ64, DIP40 conversion board	AIM65:DIP-40_W15.24mm_s_hole

2) Implementation of 16-seg LED module and I2C socket

J2	Conn_01x20_Female	For 16-segment LED connection	PinHeader_1x20_P2.54mm_Vertical
* *	16-seg LED	HPDL-1414 x 5 modules	
J11	Conn_01x04_Female	I2C device connector	PinHeader_1x04_P2.54mm_Vertical
J3	Conn_01x04_Female	Reserved for I2C devices	PinHeader_1x04_P2.54mm_Vertical

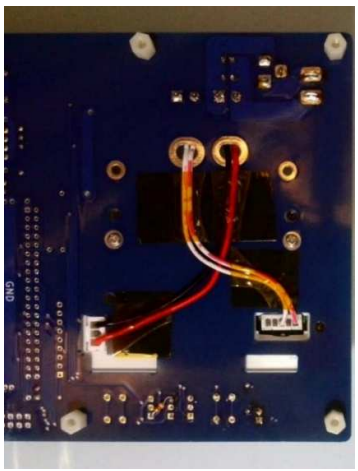


3) Printer implementation

C23	0.1u	Bypass capacitor for printer power supply	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C24	1000uF	Smoothing of printer power supply	Capacitor_THT:CP_Radial_D5.0mm_P2.50mm
J4	Conn_Coaxial_Power	Smoothing of printer power supply	Connector_BarrelJack:BarrelJack_Horizontal
J5	Screw_Terminal_01x02	Printer TX signal	TerminalBlock:TerminalBlock_bornier-2_P5.08mm
J7	Screw_Terminal_01x02	For printer power connection	TerminalBlock:TerminalBlock_bornier-2_P5.08mm
SW8	SW_SPST	printer power switch	AIM 65:power switch
* *	thermal printer	QR204 or QR203	

After actually setting up the printer, I found that the printed characters could not be seen immediately because the printer box was in the way. As a receipt printer, you can read the printed content by executing LF (line feed) after printing a batch, but as a line printer, you want to read the content while printing. Therefore, in the AIM 65 reproduction, when installing the printer, it is installed after additional processing.

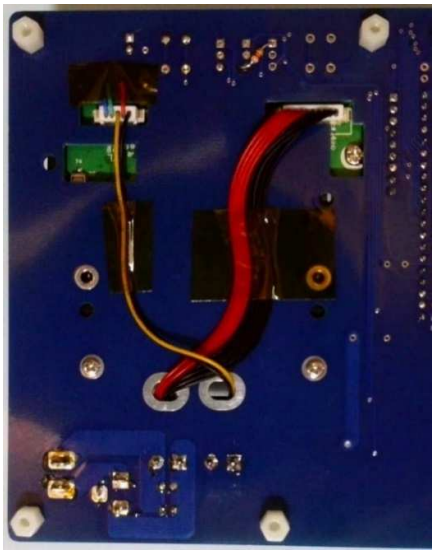
QR204



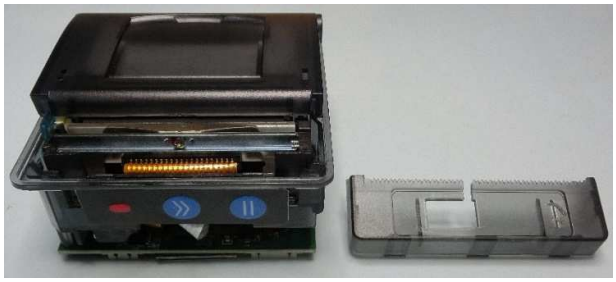
Cut the front part of the top of the printer with a drill and saw.

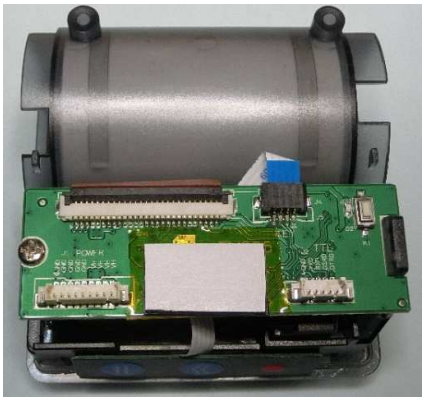
Drill two holes on the bottom and attach it to the board using tapping screws。

QR203



Remove the upper front cover, move the control switch to the front, and attach it with double-sided tape.。



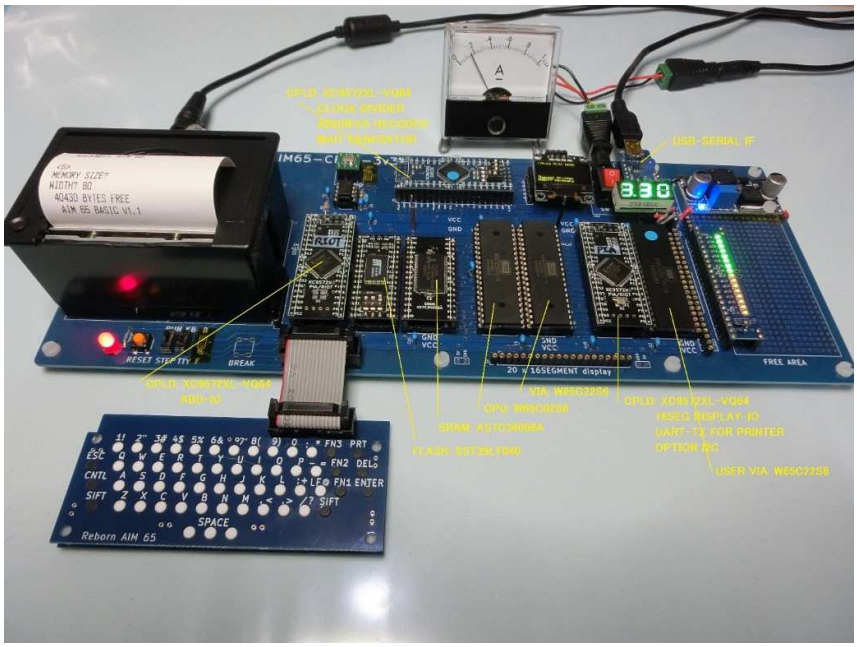


Attach it to the board with tapping screws using the two rear cylindrical holes. 。

4) Connecting a QWERTY keyboard

Install the following parts。

U18	RIOT2:	XC9572XL-VQ64, DIP40 conversion board	AIM65:DIP-40_W15.24mm_s_hole
RN1	R_Network08	Resistorpack 8 elements, 9 pins	Resistor_THT:R_Array_SIP9
J8	Conn_2x08_Odd_Even	Keyboard connector	PinHeader_2x08_P2.54mm_Vertical
* *	keyboard	QWERTY keyboard (including screws, spacers and cables)	
J6	Conn_01x37_Female	Expansion bus signal	AIM 65:PinHeader_1x37_P2.54mm_s_hole



7 OPERATION

This chapter explains how to operate the AIM65-DPLD-3V3 by giving examples of various demonstration programs, which are difficult to read from the original AIM65 English manual. In the explanation of the demo program, I added a link to the video (YouTube video) to help you understand. The video quality and sound quality are not good due to lack of skill in production. I would like to replace it in the future.

7.1 TeraTerm settings

Setting the terminal software Tera Term is essential for using AIM65-DPLD-3V3. Since the AIM65 TTY interface does not have flow control, it requires a wait setting on the terminal side in addition to the baud rate setting. Below is a summary of Tera Term's settings that I've tried and found to work properly. Frequencies in the table refer to SL_PH2, not SYS_PH2 values。

Tera Term settings

	When AIM 65 operates at 2MHz	When AIM 65 operates at 1MHz
Communication speed (baud rate)	2400 bps	1200 bps
Data	7bit	7bit
send delay	25ms to 100ms/行	50ms to 200ms/行
Wait when copying and pasting	BASIC about 200ms OK TEXT EDITOR 100ms OK FORTH 1500ms/line *1	400ms/行
Line feed code	CR	CR
key to send DEL	DEL, BS	DEL, BS

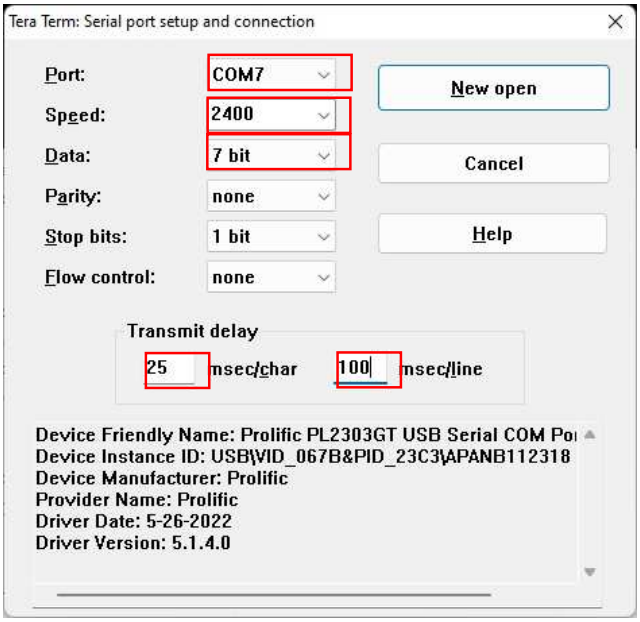
	When AIM 65 operates at 4MHz	When AIM 65 operates at 6MHz
Communication speed (baud rate)	4800 bps	9600bps
Data	7bit	7bit
send delay	B: 10ms/word, 100ms/line F: 20ms/word, 150ms/line	
Wait when copying and pasting	BASIC about 200ms/line TEXT EDITOR 100ms/line FORTH 200ms/line	
Line feed code	CR	CR
key to send DEL	DEL, BS	DEL, BS

*1) In the case of FORTH, if there is a compilation error (e.g. NOT UNIQUE) in the middle, it takes time to display the message. , it will be dropped in time and further errors will occur.

Note that the AIM 65 only accepts uppercase letters, so you need to set the keyboard to uppercase mode by pressing Caps Lock.

Tera Term setting screens

[Settings] → [Serial Port]

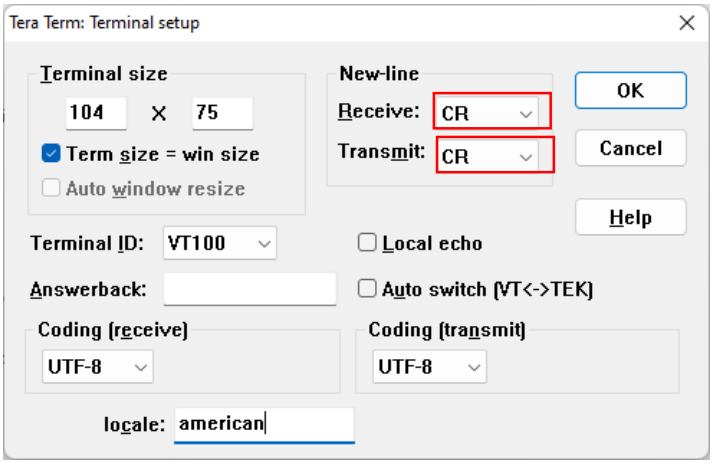


The port number varies depending on the PC used, so select the port to which the USB serial interface is assigned.

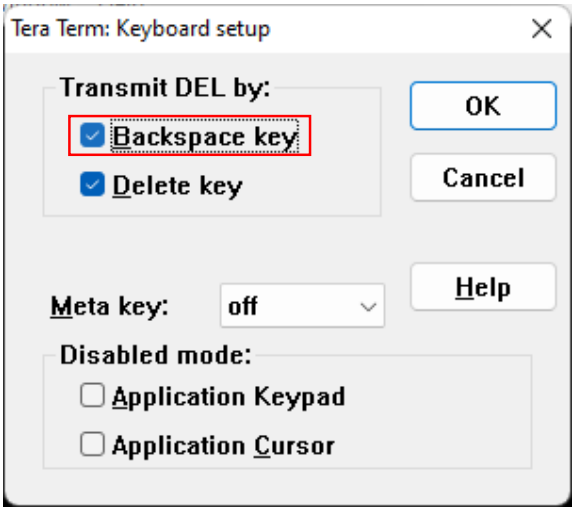
Data is 7bit. If 8bit is selected, the characters will be garbled.

AIM65 does not have flow control such as CTS/RTS, so you have to add an appropriate transmission delay.

[Settings] → [Terminal]

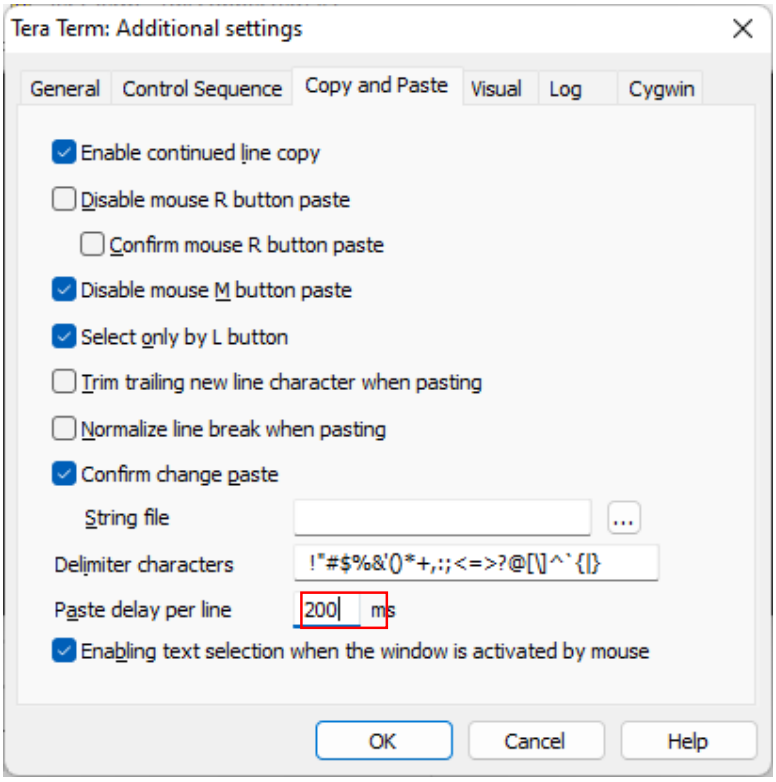


[Settings] → [Keyboard]]



AIM 65 doesn't have BS key. It is set so that DEL is also sent when the BS key is pressed.

[Settings] → [Other Settings] → [Copy and Paste]



7.2 [Extra] Teletype ASR33 and Video Terminal VT100

About the ASR33, which is synonymous with teletype

https://ja.wikipedia.org/wiki/ASR-33

The ASR-33 was a teletype terminal manufactured by Teletype, which was widely used as a computer terminal in the 1960s and 1970s. "Teletype" is a trademark of Teletype, Inc., but the company's terminals have become so popular that people often refer to these terminals as "teletype terminals" or "TTY terminals."

The model 33 type terminal was able to transmit and receive ASCII code (characters are uppercase only) at 110bps through a 20mA current loop interface. Printing is done on roll paper using a typewriter. If there was support from the operating system side, it was possible to switch to the paper tape using an appropriate escape sequence, so it was possible to load and save programs using the paper tape as if it were an external recording device.

ASR-33

https://commons.wikimedia.org/w/index.php?curid=4259050

Since the data transfer speed is 110 bps, you can see that it was dauntingly slow.

ASCII 2 characters represent 1 byte, and considering that 1 byte is 8bit (7bit code?), the speed is equivalent to 7 bytes/second.

A video of downloading and starting 4K integer BASIC from TTY paper tape to Altair8800 is on Youtube: https://www.youtube.com/watch?v=qv5b1Xowxdk&t=1s

Although it is uploaded, part of the download part is omitted as expected. Just downloading 4K-BASIC should take more than 10 minutes.

And it seems that the loud noise during operation was not something that could be used in private homes.

7.3 About the VT100 video terminal

https://ja.wikipedia.org/wiki/VT100

> The VT100 is a video display terminal developed and manufactured by Digital Equipment Corporation (DEC).
>
> VT100 is the de facto standard for video display terminals, and many terminal emulators emulate the operation of VT100.
>
> It appeared in August 1978 as a successor to the VT52 and was mass-produced. It was the first DEC terminal to use the commercially available microprocessor Intel 8080. As an option, it was possible to connect an external printer and add VRAM (AVO = Advanced Video Option).
>
> For the first time, DEC introduced "graphic effects" such as blinking, bold display, reverse display, and underline display, and the display was selectable between 80 digits and 132 digits. A 132-column by 24-line display requires additional VRAM. With the introduction of additional character sets, it is also possible to draw forms on the screen.
>
> All terminal settings were interactively possible on the VT100, and the setting data was stored in the terminal's internal non-volatile memory.
>
> It connects to the host system over a serial line and communicates using ANSI standard ASCII character codes and control sequences.
>
> The VT100 family control sequences conform to the ECMA-48 standard, ISO/IEC 6429 and ANSI X3.64, also known as ANSI escape codes. The VT100 wasn't the first X3.64-based terminal, Heath put out a microprocessor-based video display terminal that implemented a subset of his ANSI X3.64 proposal stage standard.

From https://ja.wikipedia.org/wiki/VT100

7.4 Obtaining software

The software installed in the FLASH memory of AIM65-CPLD-3V3 is all from Hans Otten's Retro Computing page, except for patching the thermal printer.

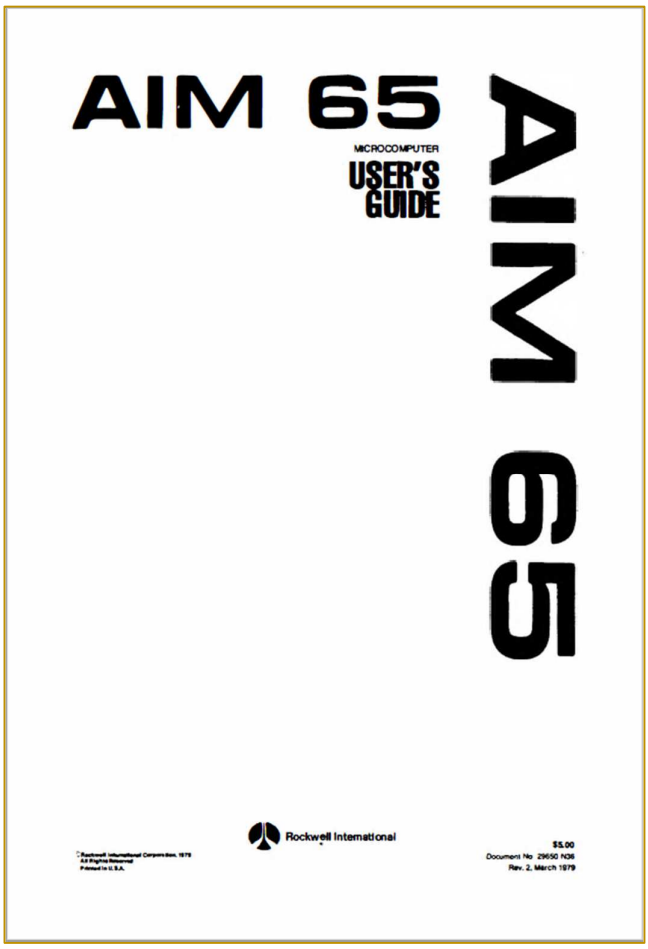
http://retro.hansotten.nl/6502-sbc/aim-65/

I used the binary data or HEX data provided.

If you already have an environment where you can use AIM65-CPLD-3V3, you can output the contents of FLASH memory to Teraterm with the monitor's D (dump) command. The formats on the website are not unified, but if you use the D command, it will be unified to the MOS-Technology papertape format, so it seems easy to handle.

7.5 Functions of the monitor program

The functionality of the monitor program is explained in the AIM 65 USER'S GUIDE.

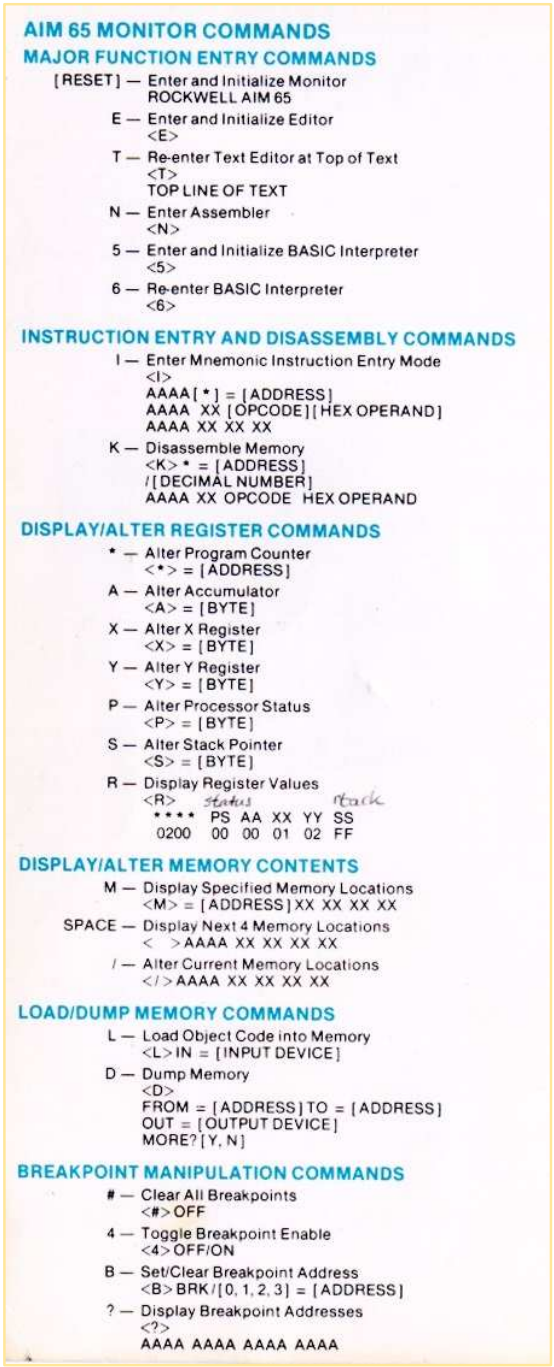


- Memory operation (Hexadecimal input and LED display)
- Dump and load memory contents
- Device specification at LOAD
- T : Cassette tape
- K : Cassette tape, KIM-1 format
- L : Paper tape
- U : user-defined device
- Device specification at DUMP
    - ⏎ or SPACE : AIM65 Display / PRINTER
- Output to thermal printer when using KB
- When using a teletype, use a teletype printer
- T: cassette tape
- K: Cassette tape (KIM-1 format)
- L: paper tape puncher

- U: User-defined device
- X: dummy output

- One-line assembler
- Disassembler
- Text editor

図〇〇 AIM65 MONITOR COMMANDS(summary sheet より)



AIM 65 MONITOR COMMANDS

MAJOR FUNCTION ENTRY COMMANDS

[RESET] — Enter and Initialize Monitor
ROCKWELL AIM 65

E — Enter and Initialize Editor
<E>

T — Re-enter Text Editor at Top of Text
<T>
TOP LINE OF TEXT

N — Enter Assembler
<N>

5 — Enter and Initialize BASIC Interpreter
<5>

6 — Re-enter BASIC Interpreter
<6>

INSTRUCTION ENTRY AND DISASSEMBLY COMMANDS

I — Enter Mnemonic Instruction Entry Mode
<I>
AAAA [*] = [ADDRESS]
AAAA XX [OPCODE] [HEX OPERAND]
AAAA XX XX XX

K — Disassemble Memory
<K> * = [ADDRESS]
/ [DECIMAL NUMBER]
AAAA XX OPCODE HEX OPERAND

DISPLAY/ALTER REGISTER COMMANDS

* — Alter Program Counter
<*> = [ADDRESS]

A — Alter Accumulator
<A> = [BYTE]

X — Alter X Register
<X> = [BYTE]

Y — Alter Y Register
<Y> = [BYTE]

P — Alter Processor Status
<P> = [BYTE]

S — Alter Stack Pointer
<S> = [BYTE]

R — Display Register Values
<R>
**** PS AA XX YY SS
0200 00 00 01 02 FF

DISPLAY/ALTER MEMORY CONTENTS

M — Display Specified Memory Locations
<M> = [ADDRESS] XX XX XX XX

SPACE — Display Next 4 Memory Locations
< >AAAA XX XX XX XX

/ — Alter Current Memory Locations
</> AAAA XX XX XX XX

LOAD/DUMP MEMORY COMMANDS

L — Load Object Code into Memory
<L> IN = [INPUT DEVICE]

D — Dump Memory
<D>
FROM = [ADDRESS] TO = [ADDRESS]
OUT = [OUTPUT DEVICE]
MORE? [Y, N]

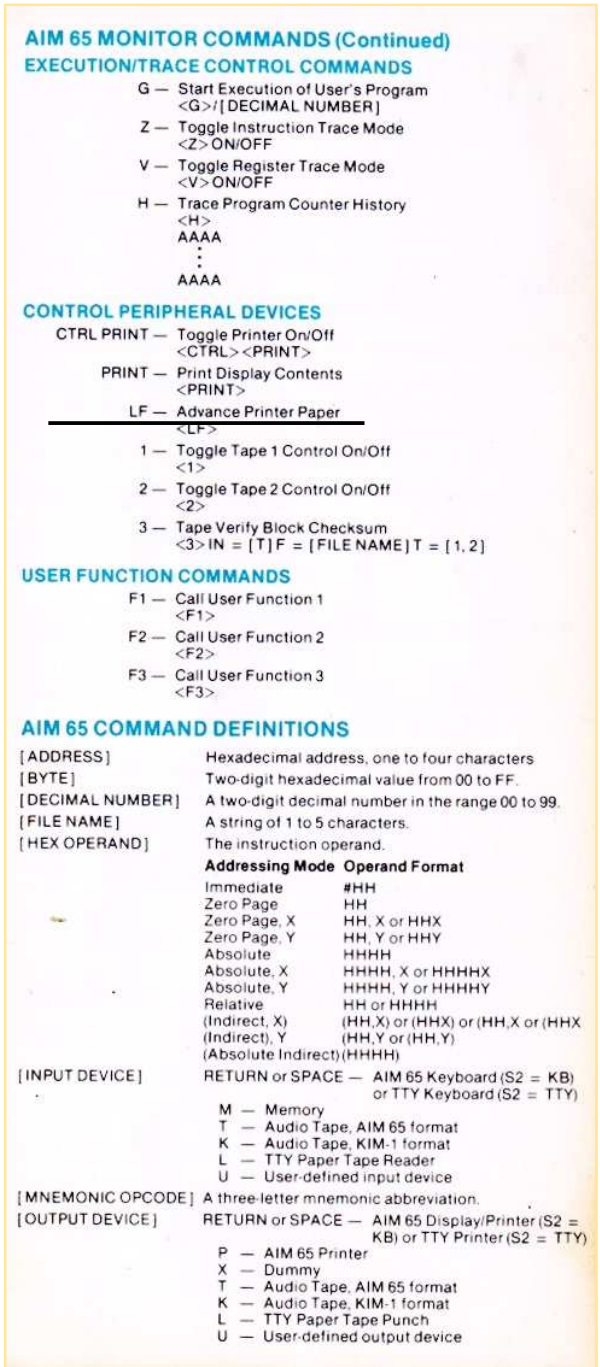
BREAKPOINT MANIPULATION COMMANDS

— Clear All Breakpoints
<#> OFF

4 — Toggle Breakpoint Enable
<4> OFF/ON

B — Set/Clear Breakpoint Address
<B> BRK/ [0, 1, 2, 3] = [ADDRESS]

? — Display Breakpoint Addresses
<?>
AAAA AAAA AAAA AAAA



AIM 65 MONITOR COMMANDS (Continued)

EXECUTION/TRACE CONTROL COMMANDS

G — Start Execution of User's Program
<G>/ [DECIMAL NUMBER]

Z — Toggle Instruction Trace Mode
<Z> ON/OFF

V — Toggle Register Trace Mode
<V> ON/OFF

H — Trace Program Counter History
<H>
AAAA
:
AAAA

CONTROL PERIPHERAL DEVICES

CTRL PRINT — Toggle Printer On/Off
<CTRL><PRINT>

PRINT — Print Display Contents
<PRINT>

LF — Advance Printer Paper
<LF>

1 — Toggle Tape 1 Control On/Off
<1>

2 — Toggle Tape 2 Control On/Off
<2>

3 — Tape Verify Block Checksum
<3> IN = [T] F = [FILE NAME] T = [1, 2]

USER FUNCTION COMMANDS

F1 — Call User Function 1
<F1>

F2 — Call User Function 2
<F2>

F3 — Call User Function 3
<F3>

AIM 65 COMMAND DEFINITIONS

[ADDRESS] Hexadecimal address, one to four characters

[BYTE] Two-digit hexadecimal value from 00 to FF.

[DECIMAL NUMBER] A two-digit decimal number in the range 00 to 99.

[FILE NAME] A string of 1 to 5 characters.

[HEX OPERAND] The instruction operand.

Addressing Mode	Operand Format
Immediate	#HH
Zero Page	HH
Zero Page, X	HH, X or HHX
Zero Page, Y	HH, Y or HHY
Absolute	HHHH
Absolute, X	HHHH, X or HHHHX
Absolute, Y	HHHH, Y or HHHHY
Relative	HH or HHHH
(Indirect, X)	(HH,X) or (HHX) or (HH,X or (HHX
(Indirect), Y	(HH,Y or (HH,Y)
(Absolute Indirect)	(HHHH)

[INPUT DEVICE] RETURN or SPACE — AIM 65 Keyboard (S2 = KB) or TTY Keyboard (S2 = TTY)
M — Memory
T — Audio Tape, AIM 65 format
K — Audio Tape, KIM-1 format
L — TTY Paper Tape Reader
U — User-defined input device

[MNEMONIC OPCODE] A three-letter mnemonic abbreviation.

[OUTPUT DEVICE] RETURN or SPACE — AIM 65 Display/Printer (S2 = KB) or TTY Printer (S2 = TTY)
P — AIM 65 Printer
X — Dummy
T — Audio Tape, AIM 65 format
K — Audio Tape, KIM-1 format
L — TTY Paper Tape Punch
U — User-defined output device

7.6 MOS Technology papertape file format

MOS Technology papertape Format is a file format for uploading and downloading binary files between a PC and microcontrollers and evaluation boards. It is similar but not identical to Intel's HEX format and Motorola's S-record format.

Data Record

Each line consists of 5 fields. A line always begins with a semicolon ; and prints the length field, the ADDRESS field, the data field, and the checksum field

;	Length	Address	Data	Checksum	CRLF

Length	A 2-character (1-byte) field that specifies the number of data bytes in the record. Usually 24 or less.
Address	A 2-byte address (big endian) that specifies where the data in the record should be loaded into memory.
Data	Data fields contain executable code, memory loadable data, or descriptive information to transfer.
Checksum	The checksum is a 2-byte field that represents the least significant 2 bytes of the sum of the values represented by the character pairs (big endian) that make up the length, address, and data fields of the record.

Last Record

The data length of the last line is zero and the data line count is printed in the address field. The checksum is the regular checksum.

When I execute the dump command to TTY with THE AIM 65, it starts with ; and the data length is 18 (24 in decimal),

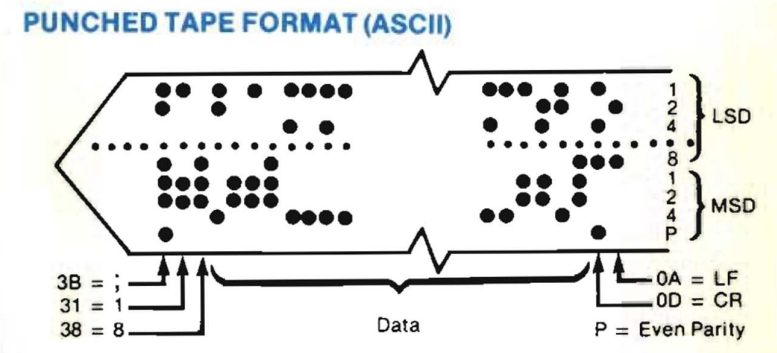
It is confirmed that the 4-digit start address, hexadecimal data, and 4-digit checksum are output in this order. Since 1 byte of data is output as 2 ASCII characters, the file size is usually 2.5 times larger than binary data. When displayed on a TTY, one line is output with 59 characters.

AIM65 monitor dump format

```
<D>
FROM=B000 TO=B100
OUT=L

;18B0004CA3CE4C7FB2FEBED1C05DB65BB5FFBA66B7BBB9D9BDEFB910FA
;18B01813B813B7EBB696B730B6F6B640B7A9B75BB6B9B786BF6BC50F42
;18B03047E89EB6F0C062C5A8B884B6BBB480B4ACB964B478C90BCA1022
;18B04897C90300BDC0DEC075CC96CD29C7F1CCD2CDD9CD22CEBB000FCF
;18B0604CC5BAC4A3C1EBC4C9C42AC43EC46AC475C479A8C57991C51063
;18B0787B69C77B50C87F7ECC5041BD463EBD7DB7CC5A9BBC646EBD0E16
;18B090454EC4464FD24E4558D4444154C1494E5055D44449CD52450B70
;18B0A841C44C45D4474F54CF5255CE49C6524553544F52C5474F530BA4
;18B0C055C25245545552CE5245CD53544FD04FCE4E554CCC5741490BE2
;18B0D8D44C4F41C4534156C54445C6504F4BC55052494ED4434F4E0BAE
;11B0F0D44C4953D4434C4541D24745D44E45D7540946
MORE?N;00000C000C
```

By the way, the punch format recorded by the TTY paper tape puncher is as follows.

Paper tape punch format

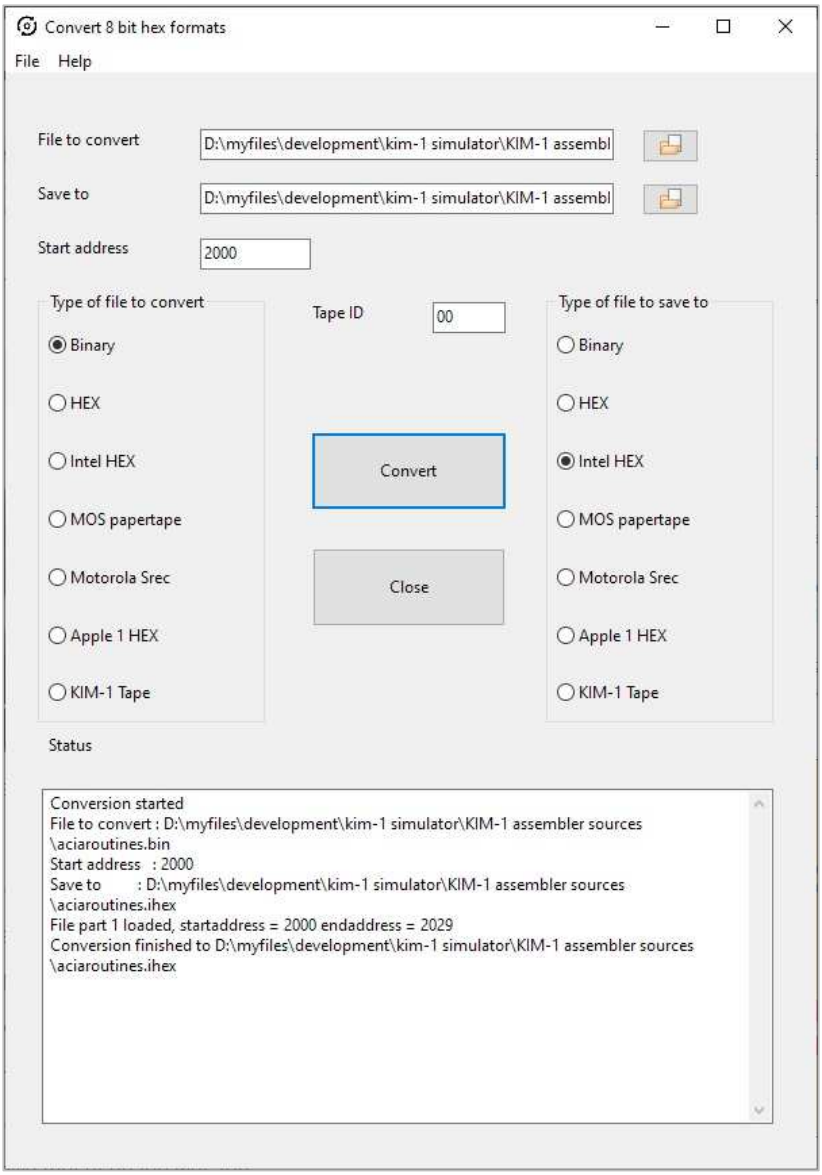


7.7 [Extra] Convert 8 bit hex formats

Use the "Convert 8 bit hex formats" tool at http://retro.hansotten.nl/my-retro-toolchain/pc-tools/ to convert between hexadecimal formats such as .

- MOS Technology papertape file format
- Intel HEX
- Motorola Srec
-

AIM 65 uses the MOS Technology papertape format, so conversion is useful when using other tools such as cross-compilers.



Binary data such as monitor program, BASIC, assembler, and FORTH can be output and displayed on the terminal by starting the AIM 65 and specifying the address range and then executing the D command, so if you save this as a text file on your PC You can reuse it whenever you need it.

If this dump list can be loaded quickly with the L command, AIM65 can be configured with RAM instead of ROM. However, the current 4800bps without flow control takes too long to load and is not practical.

In the future, we would like to add an option to dump and load data on AIM65 at high speed.

Using the assembler

Enter source in text editor

Here is an example of executing the assembler with the source in memory (input with a text editor). I am using the printer control program for the material.

1) Start Teraterm on your PC, start a text editor in another window, and open a source file.

2) AIM65 launches a text editor from Teraterm with the E command. Addresses $1000 to $1FFF are specified as the working area of the editor. After that, IN= will ask you how to input, so enter ⏎ and select input from the keyboard. In this state, by copying and pasting the source from the text editor on the PC to Teraterm, characters are input to the editor instead of input from the keyboard.

When you're done typing, type ⏎ to exit input mode. Then use the T command to move to the beginning of the text (TOP), then use the Q command to exit the editor.

Blank lines (lines with no characters and only CR) when entering text will terminate the editor's input mode, so it is necessary to remove blank lines from the source file.

Text editor in Teraterm

```
<E>EDITOR
FROM=1000⏎  TO=1FFF⏎

IN=⏎*;
*****************************************
*; *  THERMAL PRINTER ROUTINE ( UART-TX )   *
*; *****************************************
*;
*TXD    = $AC04 ; TXD REG ADDRESS (CPLD PIA2)
*BUSYF  = $AC05 ; BUSY FLAG( CPLD PIA2 BIT7 )
*IBUFM  = $A460 ; PRINT BUFFER (20 BYTES)
*PHXY   = $EB9E ; PUSH X,Y ROUTINE
*PLXY   = $EBAC ; PULL X,Y ROUTINE
*CHRCNT = $A474 ; VARIABLE FOR CHARACTER COUNT
*WORKX  = $A475 ;
*;
*; ENTRY ADDRESS
*  *=$AE00
*; OUTPUT BUFFER( 20 BYTES )
*; TO THERMAL PRINTER
*PRTOUT JSR PHXY     ; PUSH X,Y
*       LDX #20      ; PRINT 1 LINE
*       STX CHRCNT
*       LDX #0
*       STX WORKX
*B1     LDX WORKX
*       LDA IBUFM,X
*       AND #$7F
*       JSR UATX     ; TX ONE BYTE
*       INC WORKX
*       DEC CHRCNT
*       BNE B1
*       LDA #$0D     ; OUTPUT CARIDGE RETURN
*       JSR UATX     ;
*       JSR PLXY     ; PULL X,Y
*       RTS
*;
*  *=$AE2C
*LFOUT  JSR PHXY     ; PUSH X,Y
*       LDA #$20     ; OUTPUT SPACE
*       JSR UATX
*       LDA #$0D     ; OUTPUT CARIDGE RETURN
*       JSR UATX
*       JSR PLXY
*       RTS
*;
*UATX   TAY          ; TX ONE BYTE
*UA1    LDA BUSYF    ; WAIT FOR BUSY FLAG=0
*       ROL A        ;
*       BCS UA1      ;
*       STY TXD      ;
*       RTS
```

```
*;
*⏎
⏎
END
=<T>
; ****************************************
=<Q>
```

7.8 The Assembler

The NEXT figure shows an example of executing the assembler. After editing the source code in the text editor, invoke the assembler with the N command. At this time, the work area of the assembler is specified. But here we specify $2000 to $2FFF so that it does not overlap with the work area of the text editor.

IN= asks for the source file input device, so specify M (memory)

LIST? asks whether or not there is list output, so answer with Y, and enter L to specify TTY for LIST-OUT=. OBJ? will ask you if you want to output the object, so answer with N. This is done to prevent the assembler output list and object output list from being mixed and output to the TTY when Y is specified. Only the hexadecimal list is not output to the TTY, but the converted machine code output is written to memory (address $AE00 in this example).

Since the assembler is processed at high speed when the source code is stored in memory, execute the assembler again with the conditions of LIST?N, OBJ?Y to obtain the object list as shown in the figure. . This will output to TTY in the same format as the AIM 65 monitor memory dump list, so you can save this AIM 65 dump format to a text file on your PC, burn it to ROM, or load it to RAM.

By the way, the figure is the result of disassembling the code in memory generated by the assembler and outputting it as a list. Start the disassembler with the K command, enter the start address, and specify the number of lines in two decimal digits.

Assembler execution screen (Part 1: Execute with LIST?Y, OBJ?N)

```
FROM=2000 TO=2FFF
IN=M
LIST?Y
LIST-OUT=L

OBJ?N
PASS 1

PASS 2


==0000
; ****************************************
; *  THERMAL PRINTER ROUTINE ( UART-TX )   *
; ****************************************
;
==0000 TXD
       = $AC04
; TXD REG ADDRESS (CPLD PIA2)
==0000 BUSYF
       = $AC05
; BUSY FLAG( CPLD PIA2 BIT7 )
==0000 IBUFM
       = $A460
; PRINT BUFFER (20 BYTES)
==0000 PHXY
       = $EB9E
; PUSH X,Y ROUTINE
==0000 PLXY
       = $EBAC
; PULL X,Y ROUTINE
==0000 CHRCNT
       = $A474
; VARIABLE FOR CHARACTER COUNT
==0000 WORKX
       = $A475
;
;
; ENTRY ADDRESS
==0000
       *=$AE00
==AE00
; OUTPUT BUFFER( 20 BYTES )
; TO THERMAL PRINTER
==AE00 PRTOUT
209EEB JSR PHXY
; PUSH X,Y
A214   LDX #20
; PRINT 1 LINE
8E74A4 STX CHRCNT
A200   LDX #0
8E75A4 STX WORKX
==AE0D B1
AE75A4 LDX WORKX
==AE1E
D0ED   BNE B1
A90D   LDA #$0D
; OUTPUT CARIDGE RETURN
2038AE JSR UATX
;
20ACEB JSR PLXY
; PULL X,Y
60     RTS
;
==AE29
       *=$AE2C
==AE2C LFOUT
209EEB JSR PHXY
; PUSH X,Y
A90D   LDA #$20
; OUTPUT SPACE
2038AE JSR UATX
20AC   LDA #$0D
; OUTPUT CARIDGE RETURN
EB60A8 JSR UATX
AD05AC JSR PLXY
==AE3C
2A     RTS
;
==AE3D UATX
B0     TAY
; TX ONE BYTE
==AE3E UA1
FA8C04 LDA BUSYF
; WAIT FOR BUSY FLAG=0
AC     ROL A
;
60FF   BCS UA1
;
FFFFFF STY TXD
;
FF     RTS
;
;
 ERRORS= 0000
```

```
BD60A4 LDA IBUFM,X
297F   AND #$7F
2038AE JSR UATX
; TX ONE BYTE
EE75A4 INC WORKX
CE74A4 DEC CHRCNT
```

Assembler execution screen (Part 2: Execute with LIST?N, OBJ?Y)

```
<N>
ASSEMBLER
FROM=2000 TO=2FFF
IN=M
LIST?N
LIST-OUT=L

OBJ?Y
OBJ-OUT=L


PASS 1

PASS 2


;18AE00209EEBA2148E74A4A2008E75A4AE75A4BD60A4297F203DAE0C4F
;11AE18EE75A4CE74A4D0EDA90D203DAE20ACEB600A59
;18AE2C209EEBA920203DAEA90D203DAE20ACEB60A8AD05AC2AB0FA0C21
 ERRORS= 0000
;04AE448C04AC600292
;0000050005
```

Execution result of disassembler

```
<K>*=AE00
/30
 AE00 20 JSR EB9E
 AE03 A2 LDX #14
 AE05 8E STX A474
 AE08 A2 LDX #00
 AE0A 8E STX A475
 AE0D AE LDX A475
 AE10 BD LDA A460,X
 AE13 29 AND #7F
 AE15 20 JSR AE38
 AE18 EE INC A475
 AE1B CE DEC A474
 AE1E D0 BNE AE0D
 AE20 A9 LDA #0D

 AE22 20 JSR AE38

 AE25 20 JSR EBAC

 AE28 60 RTS

 AE29 FF ???
```

```
AE2A FF ???
AE2B FF ???
AE2C 20 JSR EB9E
AE2F A9 LDA #0D
AE31 20 JSR AE38
AE34 20 JSR EBAC
AE37 60 RTS
AE38 A8 TAY
AE39 AD LDA AC05
AE3C 2A ROL .A
AE3D B0 BCS AE39
AE3F 8C STY AC04
AE42 60 RTS
```

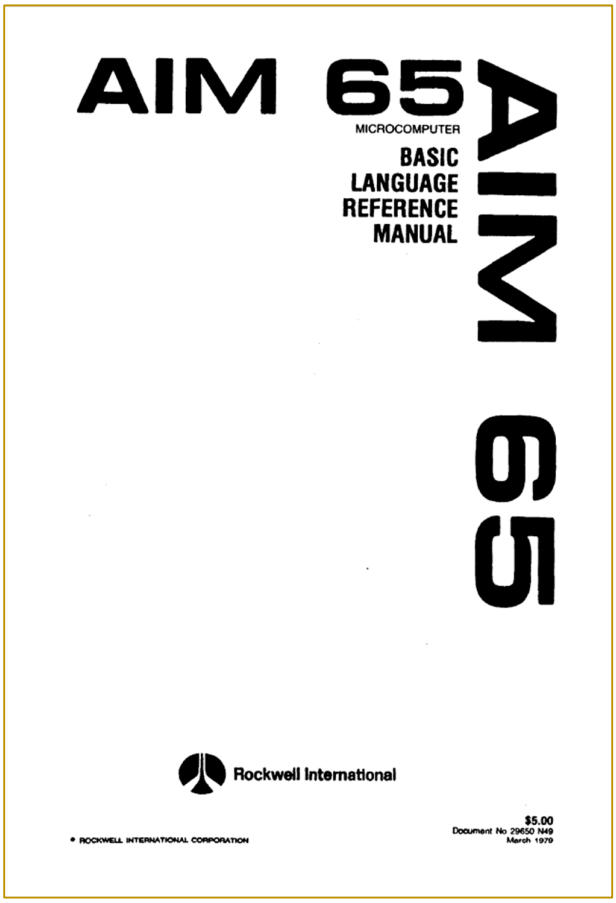
Assembler error codes

01	Undefined Symbol
02	Label Previously Defined or Forward Reference to Page 0 Symbol
03	Illegal or Missing Opcode
04	Address Not Valid
05	Accumulator Mode Not Allowed
06	Forward Reference to Page Zero
07	Ran off End of Line
08	Label Does Not Begin with Alphabetic Character
09	Label Greater Than Six Characters
10	Label or Opcode Contains Not Alphanumeric
11	Forward Reference in Equate
12	Invalid Index --- Must Be X or Y
13	Invalid Expression
14	Undefined Assembler Directive
15	Invalid Page 0 Operand
16	--
17	Relative Branch Out of Range
18	Illegal Operand Type for This Instruction
19	Out of Bounds on Indirect Addressing
20	A, X, Y, S and P are Reserved Labels
21	Program Counter Negative – Reset to 0

7.9 BASIC

BASIC is started with the 5 command in the monitor

BASIC manual cover



About the SAVE and LOAD commands in BASIC

If you specify L (printer) as the output device with the SAVE command when using teletype, the result on the Teraterm screen is the same as executing the LIST command.

Therefore, if you want to save a BASIC program, it seems better to output it to the TTY with the LIST command, copy and paste the Teraterm screen output to a text editor, and then save it to a file.

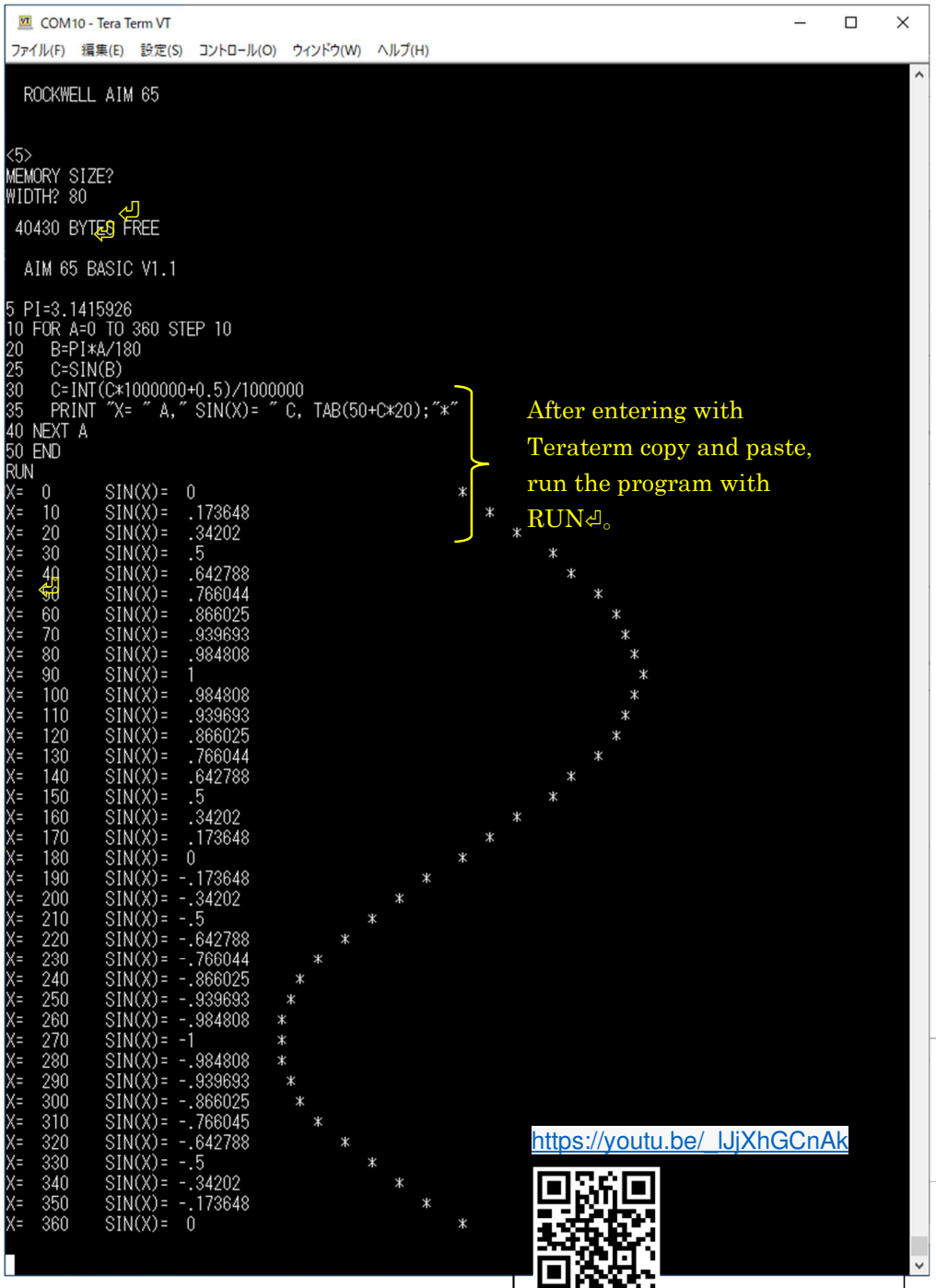
If you specify L as the device in the LOAD command, it will wait for data from the teletype side, so you can use Teraterm's file transfer function to transfer files to the AIM 65. When using the teletype, the paper tape reader operates, so you can see the progress of loading, but Teraterm does not display the progress of transmission, so it is difficult to determine whether the transfer has finished. After a sufficient amount of time has passed, if you enter Ctrl-Z from the keyboard on the Teraterm screen, AIM 65 will judge that the file transfer has ended and the LOAD command will end.

In the case of BASIC, it is easier to copy and paste the program list from the text editor on the PC to the Teraterm screen on the command input screen of the interpreter than to execute the LOAD command, so you can check the progress on the screen. Considering that the terminal is 2400bps and the wait time must be set for each line, the speed is a little slower than the LOAD command, but if the AIM 65 is clocked to about 8MHz, it will not be a problem.

BASIC program examples

LIST 0 1 BASIC sample code #1 SIN function

```
1     5 PI=3.1415926
2     10 FOR A=0 TO 360 STEP 10
3     20   B=PI*A/180
4     25   C=SIN(B)
5     30   C=INT(C*1000000+0.5)/1000000
6     35   PRINT "X= " A," SIN(X)= " C, TAB(50+C*20);"*"
7     40 NEXT A
8     50 END
```



```
5     50 A=CA
6     60 B=CB
7     70 FOR I=0 TO 15
8     80 T=A*A-B*B+CA
9     90 B=2*A*B+CB
10    100 A=T
11    110 IF (A*A+B*B)>4 THEN GOTO 200
12    120 NEXT I
13    130 PRINT " ";
14    140 GOTO 210
15    200 IF I>9 THEN I=I+7
16    205 PRINT CHR$(48+I);
17    210 NEXT X
18    220 PRINT
19    230 NEXT Y
20
21    231 PRINT"OK"
```

ASCII ART (Mandelbrot set) execution screen

The listing shows an ASCII ART program that displays the Mandelbrot set calculation to ASCII strings.
231 PRINT"OK" on the 21st line is for automatically measuring the execution time with a Teraterm macro and has nothing to do with the Mandelbrot calculation.

- CPU clock: 16MHz
- Peripheral clock: 4MHz
- Teraterm settings: 4800bps, wait: 20ms/character, 200ms/line

As a result, the processing time was 36 seconds. Originally, if it is 16MHz, the processing time should be able to be shortened. AIM65-CPLD-3V3 has a peripheral clock that is 1/4 of the CPU clock, and it takes up to 7 waits due to timing loss at the break of the clock. It is processing and it takes time to output characters. It seems that it can be improved by stopping output to 16seg display and using serial with flow control and buffered UART.

LIST 7 3 BASIC SAMPLE CODE #3 LED-BLINK USING POKE

POKE usage example, LED_BLINK

```
10 POKE 40962,255 : REM $A002=DDRB
20 POKE 40963,255 : REM $A003=DDRA
30 FOR A=0 TO 7
32 POKE 40975,170 : REM $A00F=PORT_A $AA
34 POKE 40960,170 : REM $A000=PORT_B $AA
36 GOSUB 200
38 POKE 40975,85 : REM $A00F=PORT_A $55
40 POKE 40960,85 : REM $A000=PORT_B $55
42 GOSUB 200
44 NEXT A
50 FOR A=0 TO 7
52 POKE 40975,204 : REM $A00F=PORT_A $CC
54 POKE 40960,204 : REM $A000=PORT_B $CC
56 GOSUB 200
58 POKE 40975,51  : REM $A00F=PORT_A $33
60 POKE 40960,51  : REM $A000=PORT_B $33
62 GOSUB 200
64 NEXT A
70 FOR A=0 TO 7
72 POKE 40975,240 : REM $A00F=PORT_A $F0
74 POKE 40960,240 : REM $A000=PORT_B $F0
76 GOSUB 200
78 POKE 40975,15  : REM $A00F=PORT_A $0F
80 POKE 40960,15  : REM $A000=PORT_B $0F
82 GOSUB 200
84 NEXT A
90 FOR A=0 TO 7
92 POKE 40975,255  : REM $A00F=PORT_A $FF
94 POKE 40960,0    : REM $A000=PORT_B $00
96 GOSUB 200
98 POKE 40975,0    : REM $A00F=PORT_A $00
100 POKE 40960,255 : REM $A000=PORT_B $FF
102 GOSUB 200
104 NEXT A

106 END

200 FOR K=1 TO 1000
210 J=J+K
215 NEXT
220 RETURN
```

Click here for video:
https://youtu.be/_lJjXhGCnAk

This program is a demonstration program that blinks the LEDs connected to PORT_A and PORT_B of VIA for user expansion with the POKE instruction of BASIC.

LIST 7 4 BASIC sample code #4 Character output to printer using PEEK and POKE

Usage example of PEEK, POKE, character output to printer

```
1     4 TXD=44036   : REM TXD $AC04 TXD
2     6 BUSYF=44037 : REM BUSYF $AC05 ( BUSY_FLAG : bit7 )
3     10 FOR A=1 TO 3
4     12 GOSUB 100
5     14 POKE TXD,72 : REM H
6     16 GOSUB 100
7     18 POKE TXD,69 : REM E
8     20 GOSUB 100
9     22 POKE TXD,76 : REM L
10    24 GOSUB 100
11    26 POKE TXD,76 : REM L
12    28 GOSUB 100
13    30 POKE TXD,79 : REM O
14    32 GOSUB 100
15    34 POKE TXD,44 : REM ,
16    36 GOSUB 100
17    38 POKE TXD,32 : REM SPACE
18    40 GOSUB 100
19    42 POKE TXD,87 : REM W
20    44 GOSUB 100
21    46 POKE TXD,79 : REM O
22    48 GOSUB 100
23    50 POKE TXD,82 : REM R
24    52 GOSUB 100
25    54 POKE TXD,76 : REM L
26    56 GOSUB 100
27    58 POKE TXD,68 : REM D
28    60 GOSUB 100
29    62 POKE TXD,33  : REM !
30    64 GOSUB 100
31    66 POKE TXD,13  : REM CR
32    68 NEXT A
33    70 GOSUB 100
34    72 POKE TXD,13  : REM CR
35    74 GOSUB 100
36    76 POKE TXD,13  : REM CR
37    78 END
38
39    100 F=PEEK(BUSYF) AND 128
40    REM 105 PRINT F
41    110 IF F>0 GOTO 100
42    120 RETURN
```

While checking the BUSY flag of the UART-TX for printer with the PEEK command, output the character code to the TXD register with the POKE command. in the whole program

7.10 FORTH

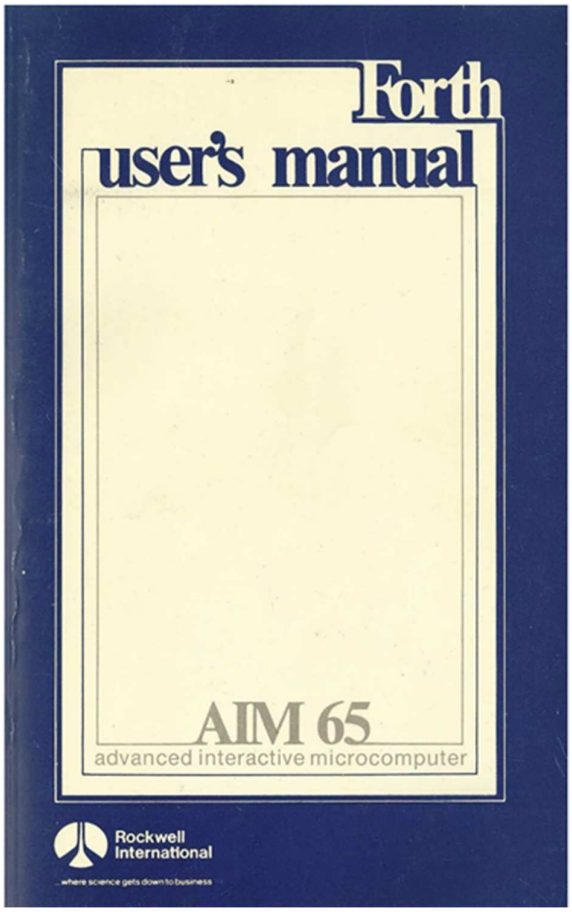
Starting FORTH

By pressing the 5 key in the monitor

```
 ROCKWELL AIM 65

<5>
 AIM 65 FORTH V1.3
```

is displayed, and the FORTH interactive input state is entered. After this, enter the words of FORTH, that is, the program, and proceed with confirmation while executing. If you define an unexpected infinite loop while creating a program and control from the keyboard becomes ineffective, you may have no choice but to press the reset button. Then you can reset and press the 6 key while the monitor program is running to re-enter FORTH. Unless the program has gone out of control and rewritten the memory contents, the program remains and development can be continued.

AIM 65 Forth user's manual cover



AIM 65's FORTH is a fairly old system, so there are times when you can't use functions that you can use now and take for granted.
Below is a list of items that require attention in FORTH for AIM65.

- This is the system before the oldest standard FORTH-79.
- Only capital letters can be used as FORTH words. Lowercase letters can also be used for character data. However, of course, when outputting to the 16SEGMENT LED, it is displayed as blanks and strange symbols.
- The STRING word is not defined by default, and you have to add it yourself by referring to the example described in the manual.
    - There is no word for memory bit manipulation (bit set CBIS!, bit clear CBIC!, bit invert CXOR!, etc.) like Mecrisp (FORTH for MSP430 and Cortex-M). It seems that there is no choice but to load the value to the data stack once, process it with AND, OR, XOR, and then write it back with the store.
- Prefixes ($FFFF, %01100101, #256, etc.) cannot be used in the description of immediate values, so if you want to switch between hexadecimal and decimal input, change the BASE. .
- NOT means to invert TRUE/FALSE, not to invert all bits of data. All bit inversion of 16bit data will be processed by "FFFF XOR" instead of NOT. . ・ How is the shift operation performed? Even if other FORTH systems don't have words for shift operations, the words 2* and 2/ exist and can be used to shift 1 bit. AIM65 seems to have no choice but to substitute the following.
- SHL(left shift by 1 bit) ===> 2 * or substitute with DUP + .
- SHR(right shift by 1 bit) ===> 2 / substitute
- Line comment \ (backslash) cannot be used.
- In FORTH of AIM65, when defining a word, it must be within 60 characters per line.
- Lines with more than 60 characters are forcibly truncated and treated as input for the next line. If the length is likely to exceed 60 characters, the continuation of the program must be defined with a newline.

Although there are various restrictions, it functions as FORTH, so programs can be built.

FORTH program example

Definition of LIST

The AIM-65 FORTH has a standard VLIST word that can output the currently defined word (vocabulary) list to the terminal. However, I don't want to use it because it takes a long time to output because it doesn't fit on one screen because it breaks a line for each word.

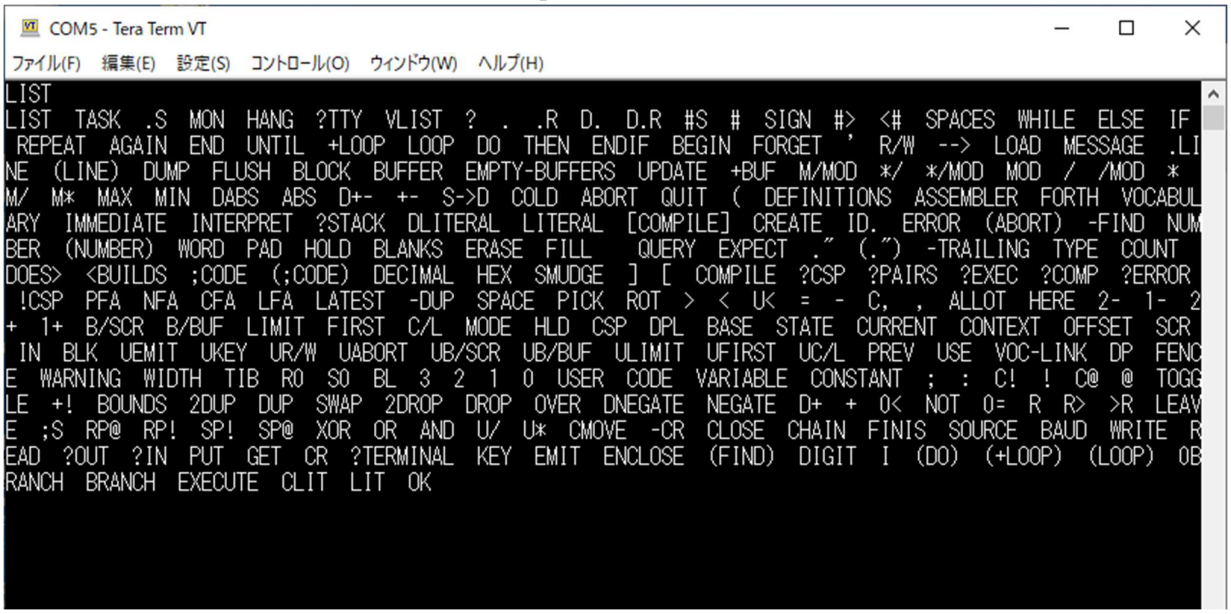
So let's define the word "LIST" using 5.5.3 Forth Word Handling Example from the Forth user's manual.

LIST 7 2 FORTH sample code "LIST" definition

```
1     : LIST ( -- )
2     CR LATEST
3     BEGIN
4       DUP ID. SPACE PFA LFA @
5       DUP 0=
6     UNTIL
      ;
```

Input this from the terminal or copy and paste from the editor on the PC with TeraTerm and compile it as a FORTH word. A subsequent LIST can output a list of currently defined words.

FORTH LIST command execution example



LIST 7 2 FORTH sample code "DEMO_LED1"

```
: LIST ( -- )
CR LATEST
BEGIN
  DUP ID. SPACE PFA LFA @
  DUP 0=
UNTIL
;
( R6522 VIA REGISTER ADRESS )
HEX
A000 CONSTANT UDRB
A001 CONSTANT UDRAH
A002 CONSTANT UDDRB
A003 CONSTANT UDDRA
A004 CONSTANT UT1L
A005 CONSTANT UT1CH
A006 CONSTANT UT1LL
A007 CONSTANT UT1LH
A008 CONSTANT UT2L
A009 CONSTANT UT2H
A00A CONSTANT USR
A00B CONSTANT UACR
A00C CONSTANT UPCR
A00D CONSTANT UIFR
A00E CONSTANT UIER
A00F CONSTANT UDRA
```

```
: INIT_VIA ( -- )
FF UDDRA C!  ( PORT_A OUTPUT )
FF UDDRB C!  ( PORT_B OUTPUT )
0 UDRA C!    ( ALL LED ON )
0 UDRB C!    ( ALL LED ON )
;
2000 VARIABLE DD
DECIMAL
: DELAY DD @ 0 DO LOOP ;
HEX
: DEMO_LED1 ( -- )
INIT_VIA
CR ." DEMO : LED BLINK  $AA_$55 "
30 0 DO
  AA DUP UDRA C! UDRB C! DELAY
  55 DUP UDRA C! UDRB C! DELAY
LOOP
;
FINIS
```

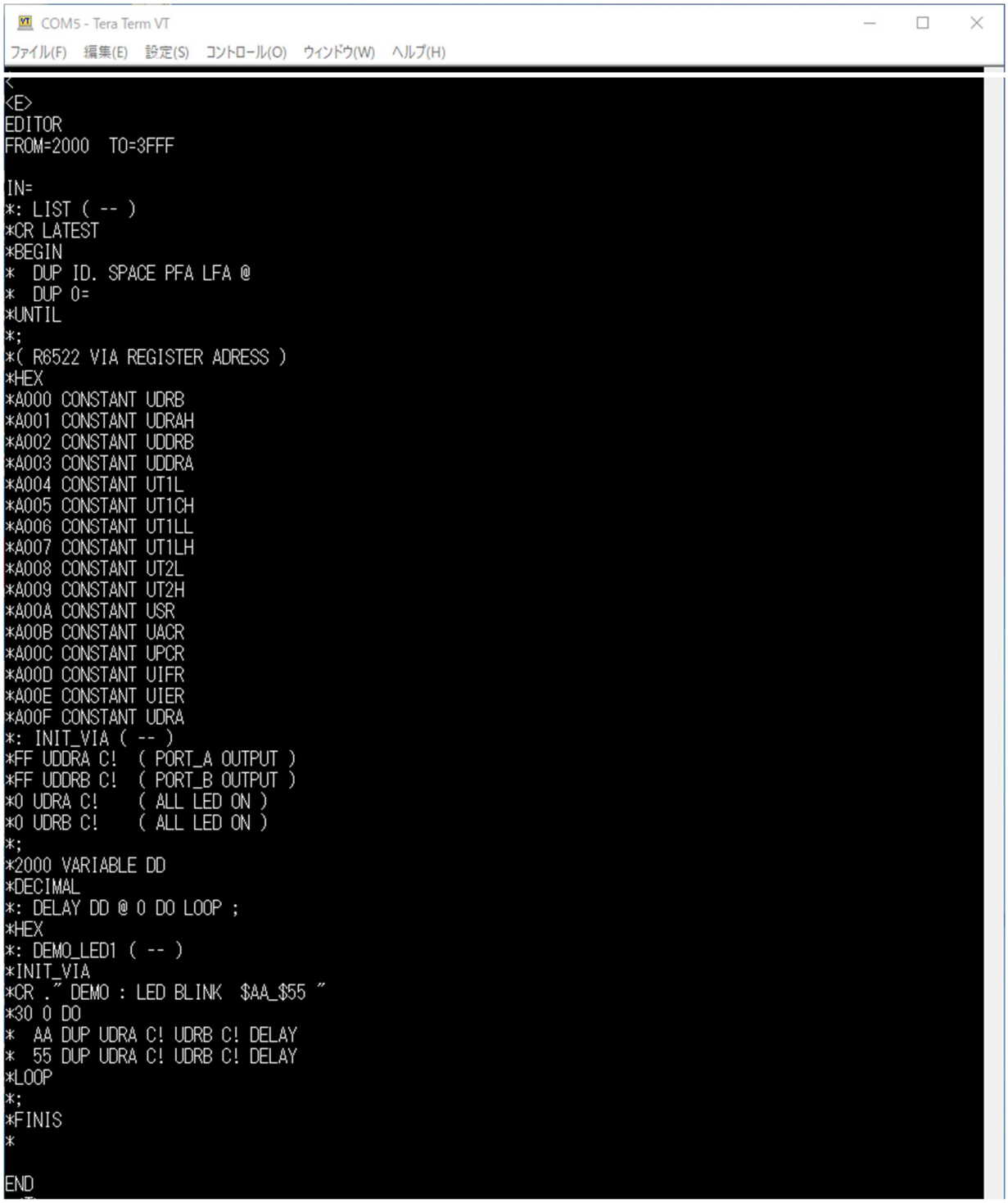
Much easier than controlling with BASIC. BASIC has PEEK and POKE, but they cannot be used in hexadecimal notation, so they can hardly be used for programs that require setting to peripheral IC registers. In FORTH, if you want to set data D1 to address ADR1, define D1 and ADR1 in advance.

D1 ADR1 C! (C! stores byte data on the data stack)

just run

For program input in FORTH, input line by line by interactive input, and define words by incremental compilation. If input from the keyboard is troublesome, copy and paste from a text editor to Teraterm. FORTH's word definition is bottom-up, so once an error occurs, all subsequent input using that word will fail. While the accuracy of the program is low, it is better to enter the copy and paste units in small increments.

If the program has already been debugged, it can be read all at once using the SOURCE command. In that case, FINIS must be placed at the end of the source code.



Invoke the TEXT EDITOR with the E command and enter FROM=2000⏎ TO=3FFF⏎ IN=⏎ to specify the editor's working area and input from the terminal. After that, by copying and pasting the program source from the editor on the PC to TeraTerm, it is input to the TEXT EDITOR of AIM-65. If you set the wait time for each line of TeraTerm to about 100ms/line, reading will be completed in a relatively short time.

At the end of the program, enter FINIS⏎⏎ (enter twice) to complete the input. Position the input pointer at the beginning with the T command, then exit the TEXT EDITOR (press Q).

Then start FORTH and type SOURCE⏎ IN=M⏎ to load the source program from memory and display OK when finished. The method of reading the source program from memory and compiling finishes in a fairly short time (several seconds). If you run the LIST command, you can see that the word DEMO_LED1 has been added.

Register Number	RS Coding				Register Desig.	Register/Description	
	RS3	RS2	RS1	RS0		Write (R/$\overline{W}$ = L)	Read (R/$\overline{W}$ = H)
0	0	0	0	0	ORB/IRB	Output Register B	Input Register B
1	0	0	0	1	ORA/IRA	Output Register A	Input Register A
2	0	0	1	0	DDRB	Data Direction Register B	
3	0	0	1	1	DDRA	Data Direction Register A	
4	0	1	0	0	T1C-L	T1 Low-Order Latches	T1 Low-Order Counter
5	0	1	0	1	T1C-H	T1 High-Order Counter	
6	0	1	1	0	T1L-L	T1 Low-Order Latches	
7	0	1	1	1	T1L-H	T1 High-Order Latches	
8	1	0	0	0	T2C-L	T2 Low-Order Latches	T2 Low-Order Counter
9	1	0	0	1	T2C-H	T2 High-Order Counter	
10	1	0	1	0	SR	Shift Register	
11	1	0	1	1	ACR	Auxiliary Control Register	
12	1	1	0	0	PCR	Peripheral Control Register	
13	1	1	0	1	IFR	Interrupt Flag Register	
14	1	1	1	0	IER	Interrupt Enable Register	
15	1	1	1	1	ORA/IRA	Output Register A*	Input Register A*

NOTE: *Same as Register 1 except no handshake.

Address Allocation of R6522 VIA for User Expansion

Address.	Register name	Explanation
A000	UDRB	Data register B
A001	UDRAH	Data register A, with handshake control
A002	UDDRB	DIR register B
A003	UDDRA	DIR register A
A004	UT1C-L	Write : T1 Low- Oder Latches Read : T1 Low-Oder Counter
A005	UT1C-H	T1 High-Oder Counter
A006	UT1L-L	T1 Low-Oder Latches
A007	UT1L-H	T1 High -Oder Latches
A008	UT2C-L	Write : T1 Low- Oder Latches Read : T1 Low-Oder Counter
A009	UT2C-H	T2 High-Oder Counter
A00A	USR	shift register
A00B	UACR	Auxiliary control register
A00C	UPCR	Peripheral control register
A00D	UIFR	interrupt flag register
A00E	UIER	interrupt enable register
A00F	UDRA	data register A, does not affect handshake

LIST 7 3 FORTH sample code UART_PRT_QR203

We introduce a demo program that outputs characters to a thermal printer. AIM 65 FORTH has old specifications, and words that handle character strings are not defined. However, APPENDIX I of the AIM 65 FORTH MANUAL shows an extended example of words that handle STRING, so I used it as is to create a program that defines a character string and outputs it to the printer. Also, the QR203 printer has a function to print small fonts of 9x24 dots, so it is also defined.

```
      : LIST ( -- )
1     CR LATEST
2     BEGIN
3       DUP ID. SPACE PFA LFA @
4       DUP 0=
5     UNTIL
6     ;
7     ( R6522 VIA REGISTER ADRESS )
8     HEX
9     A000 CONSTANT UDRB
10    A001 CONSTANT UDRAH
11    A002 CONSTANT UDDRB
12    A003 CONSTANT UDDRA
13    A004 CONSTANT UT1L
14    A005 CONSTANT UT1CH
15    A006 CONSTANT UT1LL
16    A007 CONSTANT UT1LH
17    A008 CONSTANT UT2L
18    A009 CONSTANT UT2H
19    A00A CONSTANT USR
20    A00B CONSTANT UACR
21    A00C CONSTANT UPCR
22    A00D CONSTANT UIFR
23    A00E CONSTANT UIER
24    A00F CONSTANT UDRA
25    : INIT_VIA ( -- )
26    FF UDDRA C!  ( PORT_A OUTPUT )
27    FF UDDRB C!  ( PORT_B OUTPUT )
28    0 UDRA C!    ( ALL LED ON )
29    0 UDRB C!    ( ALL LED ON )
30    ;
31    2000 VARIABLE DD
32    DECIMAL
33    : DELAY DD @ 0 DO LOOP ;
34    HEX
35    : DEMO_LED1 ( -- )
36    INIT_VIA
37    CR ." DEMO : LED BLINK  $AA_$55 "
38    30 0 DO
39      AA DUP UDRA C! UDRB C! DELAY
40      55 DUP UDRA C! UDRB C! DELAY
41    LOOP
42    ;
43    FINIS
```

BUZZER (MELODY) DEMO

Here is a demo that uses the R6522 VIA timer to ring the speaker.。

Click here for video

https://youtu.be/a6Vgk7XXLG0

LIST○○ BUZZER DEMO

```
1     : LIST ( -- )
2       CR LATEST
3       BEGIN
4         DUP ID. SPACE PFA LFA @
5         DUP 0=
6       UNTIL ;
7
8     ( R6522 VIA REGISTER ADRESS )
9     HEX
10    A000 CONSTANT UDRB  ( PORT. B  )
11    A001 CONSTANT UDRAH ( PORT. A HANDSHAKE )
12    A002 CONSTANT UDDRB ( DIR. B  )
13    A003 CONSTANT UDDRA ( DIR. A  )
14    A004 CONSTANT UT1L  ( T.1   )
15    A005 CONSTANT UT1CH ( T.1   )
16    A006 CONSTANT UT1LL ( T.1   )
17    A007 CONSTANT UT1LH ( T.1   )
18    A008 CONSTANT UT2L  ( T.2   )
19    A009 CONSTANT UT2H  ( T.2   )
20    A00A CONSTANT USR   (       )
21    A00B CONSTANT UACR  (       )
22    A00C CONSTANT UPCR  (       )
23    A00D CONSTANT UIFR  (       )
24    A00E CONSTANT UIER  (       )
25    A00F CONSTANT UDRA  ( PORT. A )

      : INIT-VIA-BUZZER ( -- )
      00 UDRA C!  ( PORT-A INPUT )
      E0 UDDRB C! ( PORT-B PB7,PB6,PB5 OUTPUT )
      00 UDRA C!  ( ALL 0 )
      80 UDRB C!  ( PB7=1, PB6=0, PB5=0 )
      UACR C@ 1F AND UACR C! ( T1_STOP,
      T2_ONE_SHOT_MODE )
      A0 UIER C! ( T2.INT ENABLE )
      ;
      9C40 VARIABLE T ( $9C40 = 40000 -->
      20ms )
      6 VARIABLE TEMPO
      8 VARIABLE N  ( 8x6x20ms = 0.96s )

      HEX
      : T_MEAS ( -- )
        N @ TEMPO @ * 0 DO
          T @ UT2L ! ( T2L/T2H WRITE --> 20ms )
          BEGIN UIFR C@ 20 AND UNTIL ( WAIT
      T2.IRQ )
```

```
26
27    : SO_   ( L -- ) 9F5 SWAP BUZZER ;
28    : SO#_  ( L -- ) 965 SWAP BUZZER ;
29    : RA_   ( L -- ) 8DE SWAP BUZZER ;
30    : RA#_  ( L -- ) 85F SWAP BUZZER ;
31    : SI_   ( L -- ) 7E6 SWAP BUZZER ;
32    : DU    ( L -- ) 775 SWAP BUZZER ;
33    : DU#   ( L -- ) 709 SWAP BUZZER ;
34    : RE    ( L -- ) 6A4 SWAP BUZZER ;
35    : RE#   ( L -- ) 645 SWAP BUZZER ;
36    : MI    ( L -- ) 5EA SWAP BUZZER ;
37    : FA    ( L -- ) 595 SWAP BUZZER ;
38    : FA#   ( L -- ) 545 SWAP BUZZER ;
39    : SO    ( L -- ) 4F9 SWAP BUZZER ;
40    : SO#   ( L -- ) 4B1 SWAP BUZZER ;
41    : RA    ( L -- ) 46E SWAP BUZZER ;
42    : RA#   ( L -- ) 42E SWAP BUZZER ;
43    : SI    ( L -- ) 3F2 SWAP BUZZER ;
44    : DU~   ( L -- ) 3B9 SWAP BUZZER ;
      : DU#~  ( L -- ) 383 SWAP BUZZER ;
      : RE~   ( L -- ) 351 SWAP BUZZER ;
      : RE#~  ( L -- ) 321 SWAP BUZZER ;
      : MI~   ( L -- ) 2F4 SWAP BUZZER ;
      : FA~   ( L -- ) 2C9 SWAP BUZZER ;
      : FA#~  ( L -- ) 2A1 SWAP BUZZER ;
      : SO~   ( L -- ) 27B SWAP BUZZER ;
      : SO#~  ( L -- ) 257 SWAP BUZZER ;
      : RA~   ( L -- ) 236 SWAP BUZZER ;
      : RA#~  ( L -- ) 216 SWAP BUZZER ;
      : SI~   ( L -- ) 1F8 SWAP BUZZER ;
      : DU~~  ( L -- ) 1DB SWAP BUZZER ;
      : DU#~~ ( L -- ) 1C0 SWAP BUZZER ;
      : RE~~  ( L -- ) 1A7 SWAP BUZZER ;
      : RE#~~ ( L -- ) 18F SWAP BUZZER ;

      : WAIT8 ( -- ) ." " 8 N ! T_MEAS ;  ( 960ms
      WAIT )
      : WAIT4 ( -- ) ." " 4 N ! T_MEAS ;  ( 480ms
      WAIT )
      : WAIT2 ( -- ) ." " 2 N ! T_MEAS ;  ( 240ms
      WAIT )
      : WAIT1 ( -- ) ." " 1 N ! T_MEAS ;  ( 120ms
      WAIT )
      : GAP  5 0 DO LOOP ;
```

```
    UT2L C@ DROP ( T2.IRQ CLEAR )
  LOOP
;
INIT-VIA-BUZZER

: BUZZER ( TONE DURATION -- )
  ." *" N !
  CO UACR C!
  UT1L ! T_MEAS
  0 UACR C! ( T1.STOP )
;
```

```
DECIMAL
: ROBINSON ( -- )
8 RA 2 DU 2 FA 2 RA GAP 2 RA 4 SI 2 DU~ 8 SI WAIT1
8 SI 2 MI 2 RE~ 2 SI 2 SI 4 DU~ 2 RE~ 8 DU~ WAIT1
6 MI~ 2 RA GAP 4 RA 4 MI~ 6 RE~ 2 SO GAP 4 SO GAP GAP
2 SO 2 FA 2 MI 4 RA GAP 8 RA
;
: NAMONAKI ( -- )
3 SI 3 RA 2 SO GAP 6 SO GAP 6 SO~
4 RA~ 4 FA#~ 2 SO~ 2 FA#~ 4 RE~ WAIT2
2 SO 6 DU~ 6 RE~ 4 MI~ 4 SI 2 DU~ 4 SI 2 RA
GAP 2 RA 2 SO 4 SO WAIT2
2 SO 4 FA# 2 SO GAP 2 SO 4 RA 4 SO
4 FA# 4 SO 10 MI~ 2 RE~ 2 SI 10 RE~
3 SI 3 RA 2 SO GAP 6 SO GAP 6 SO~ 4 RA~
4 FA#~ 2 SO~ 2 FA#~ 4 RE~ WAIT2
2 SO 6 DU~ 6 RE~ 4 MI~ 4 SI 2 DU~ 4 SI 2 RA
GAP 2 RA 2 SO GAP 3 SO 3 RA 2 SI GAP 4 SI WAIT4
3 DU#~ 3 RE~ 3 MI~ GAP 3 MI~
3 FA#~ 2 SO~ GAP 12 SO~ WAIT1
2 SO 4 DU~ 2 SI 4 SO 4 FA# 8 SO
;
: AONO ( -- )
GAP 8 SO#~ 8 RA~ 10 DU#~~ 4 SI~ 2 DU#~~ 6 SI~
2 RA~ 4 SO#~ 2 RA~ 16 SO#~
GAP 2 SO#~ 4 SO#~ 4 RA#~ 4 DU~~ 4 SO#~ 4 FA#~ 2 SO#~ 10 MI~
GAP 4 DU#~ 4 MI~ 4 RA~ 2 SO#~ 4 RA~ 6 SI~ 4 SO#~ 4 FA#~
GAP 4 SO#~ 4 RA~ 10 DU#~~ 4 SI~ 2 DU#~~ 6 SI~
2 RA~ 4 SO#~ 2 RA~ 16 SO#~
GAP 2 SO#~ 4 RE#~~ 4 DU~~ 4 SO#~ 2 FA#~
2 MI~ 4 MI~ 4 SO#~ 4 RE#~~ 16 DU#~~
GAP 4 MI~ 4 RE#~ 4 FA#~ 4 DU#~~ 12 SI~
GAP 4 SO#~ 4 RA~ 12 SI~ 2 MI~ GAP 8 MI~
GAP 4 SO#~ 4 RA~ 12 SI~ 4 MI~ GAP 6 MI~
GAP 2 RA~ 26 SO#~ 4 FA#~ 24 MI~
;
: LEMON ( -- )
2 DU~ 2 RE~ 4 MI~ 2 DU~ 6 RA 4 RE~ 4 SI 2 SO
6 MI 4 SI 4 RA 2 SO 6 DU 4 SO 8 MI WAIT4
2 RE 2 MI 8 FA 4 DU~ 2 SI  2 DU~ 8 SO 4 FA
2 MI 2 FA 8 FA# 4 DU~ 2 SI 2 RA 8 SO# WAIT4
2 DU~ 2 RE~ 4 MI~ 2 DU~ 6 RA 4 RE~ 4 SI 2 SO
6 MI 4 SI 4 RA 2 SO 6 DU 4 SO 8 MI WAIT4
2 RE 2 MI 8 FA 4 SO 2 FA 2 SO 4 MI 4 SO 4 DU~
4 MI~ 2 RE~ 6 RE~ 2 RE~ 4 DU~ 2 DU~ 8 DU~  WAIT8
6 RA 2 SI 4 DU~ 2 SI 2 RA 4 SO 4 MI~ 8 MI~
6 SI 2 MI~ 4 FA~ 2 MI~ 2 RE~ 4 DU~ 4 RE~ 8 SO
6 FA 2 SO 4 RA 2 SO 2 FA 4 MI 4 DU~ 4 DU~ 4 DU~
8 SI 4 RA 4 SI 8 DU~ GAP
2 RE~ 2 MI~ 2 RE~ 2 DU~ 2 RA 6 DU~
2 MI~ 6 SO~ 2 RE~ 2 DU~ WAIT4
2 RE~ 2 MI~ 2 RE~ 2 DU~ 2 RA 6 DU~
2 MI~ 6 SO~ 2 RE~ 2 DU~ WAIT4
```

```
2 RE~ 2 MI~ 2 RE~ 2 DU~ 2 RA 6 DU~
2 MI~ 4 SO~ 2 SO~ 2 RA~ 6 SO~ GAP
2 SO~ 6 DU~~ GAP 2 SI~ 4 SO~ 2 MI~
2 MI~ 4 SO~ 2 RE~ GAP 8 RE~ GAP
2 RE~ 2 MI~ 2 RE~ 2 DU~ 2 RA 6 DU~
2 MI~ 6 SO~ 2 RE~ 6 DU~ GAP
2 DU~ 2 DU~ 2 RE~ 2 MI~ 2 FA~ 6 MI~ 2 RE~ 6 SI 8 DU~ WAIT4
2 DU~ 2 SI 4 RA 4 SI 4 DU~ 4 RE~
4 DU~ 4 SO 2 MI 4 SO GAP 2 SO
2 RA 2 DU~ 4 RE~ 2 SI 6 DU~ GAP 8 DU~ WAIT4
2 DU~ 2 SI 4 RA 4 SI 4 DU~ 4 RE~ 4 DU~ 4 SO 2 DU~ 6 RE~
2 DU~ 6 FA~ 2 RE~ 6 DU~ GAP 8 DU~
;
```

```
: DEMO-MELODY ( -- )
INIT-VIA-BUZZER
CR ." ROBINSON BY SPITZ : "
  40000 T ! ROBINSON
  WAIT8 WAIT8
CR ." NAMONAKI-UTA BY MR.CHILDREN : "
  40000 T ! NAMONAKI
  WAIT8 WAIT8
CR ." AONO_REQUIEM BY CHITOSE HAJIME : "
  35000 T ! AONO
  WAIT8 WAIT8
CR ." LEMON BY KENSI YONEZU : "
  28000 T ! LEMON
  WAIT8 WAIT8
;
```

Writing I2C Drivers in FORTH

I2C is a convenient interface standard that allows data to be exchanged by synchronous communication using two signals, SDA and SCL. However, when the AIM-65 was released, there was no I2C interface standard and no I2C compliant devices existed. Now that the I2C standard has been clarified, if one-to-one communication is performed with a master driver, the method of controlling GPIO by software can be used sufficiently. By having an I2C interface, it is possible to control E2PROM, calendar IC, AD converter, small display device (LCD, OLED), etc.

I2C driver routine by listing FORTH (MASTER ONLY)

```
1     ( PIA2 REGISTER ADRESS )
2     ( 000 : PA[7:0]                                : $AC00 )
3     ( 001 : control_A  dummy , read=$00            : $AC01 )
4     ( 010 : PB[7:0]                                : $AC02 )
5     ( 011 : control_B  dummy , read=$00            : $AC03 )
6     ( 100 : TXD ; UART_TXD                         : $AC04 )
7     ( 101 : I2C ; bit0:SCL, bit1:SDA, bit7:BUSY_F : $AC05 )
8
9     HEX
10    AC00 CONSTANT PA    ( PORT A     )
11    AC02 CONSTANT PB    ( PORT B     )
12    AC04 CONSTANT TXD   ( UART TX    )
13    AC05 CONSTANT I2C   ( BIT_0 : SCL, BIT_1 : SDA )
14    : INIT-PIA2-I2C ( -- )
15    I2C C@ 03 OR I2C C!  ( SDA-1, SCL-1 )
16    ;
17    ( ******************************************************* )
18    ( BIT-BANGED I2C DRIVER                                   )
19    ( THIS I2C DRIVER SUPPORTS MASTER-ONLY. CLOCK STRETCHING. )
20    ( THERE HAVE TO BE 1..10K OHM RESISTORS ON SDA AND SCL    )
21    ( TO PULL THEM UP TO IDLE STATE.                          )
22    ( ******************************************************* )
23    ( PIA2   pin18:CB1--SDA   pin19:CB2--SCL  )
24    HEX
25    : SDA-0 ( -- )  I2C C@ FD AND I2C C! ; ( SDA 0_OUT  )
26    : SDA-1 ( -- )  I2C C@ 02 OR  I2C C! ; ( SDA HIGH-Z )
27    : SCL-0 ( -- )  I2C C@ FE AND I2C C! ; ( SCL 0_OUT  )
28    : SCL-1 ( -- )  I2C C@ 01 OR  I2C C!
29                    BEGIN I2C C@ 01 AND UNTIL ;
30                    ( SCL 1_OUT THEN WAIT SCL INPUT =1 )
31                    ( SUPPORT CLOCK STRETCHING )
32    : SDA> ( -- F ) I2C C@ 02 AND IF FF ELSE 00 THEN ;
33      ( 1 BIT READ FROM SDA )
34    : HALF ( -- ) DUP DROP ; ( half-cycle delay )
35    : I2C-START
36          SDA-1 HALF SCL-1 HALF SDA-0 HALF SCL-0 ;
37      ( START CONDITION: DURING SCL_HIGH , CHANGE SDA 1_TO_0 )
38    : I2C-STOP
39          SDA-0 HALF SCL-1 HALF SDA-1 HALF ;
40      ( STOP CONDITION: DURING SCL_LOW, CHANGE SDA 0_TO_1 )
41    : I2C-TX-BIT ( F -- ) ( 1BIT SDA OUT, SCL PULSE GEN )
42          0= IF SDA-0 ELSE SDA-1 THEN HALF SCL-1 HALF SCL-0 ;
43    : I2C-RX-BIT ( -- B ) ( 1BIT SDA_INPUT, SCL PULSE GEN )
44          SDA-1 HALF SCL-1 HALF SDA> SCL-0 ;
45    : I2C-TX ( B -- NAK ) ( SEND ONE BYTE )
46          8 0 DO DUP 80 AND I2C-TX-BIT DUP + LOOP DROP
47          I2C-RX-BIT ;
48    : I2C-RX ( NAK -- B ) ( RECV ONE BYTE THEN SEND ACK/NOACK )
49          0 8 0 DO DUP + I2C-RX-BIT 1 AND + LOOP SWAP
50          I2C-TX-BIT ;
```

The above listing is an example of I2C driver routine by FORTH.

The pins used as SDA and SCL can be GPIO, but here we use two ports, bit1 and bit0, added to $AC05 address in the Verilog description of PIA2.

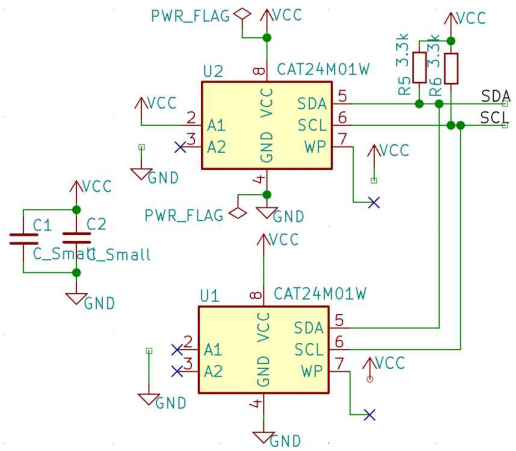
When data is stored in the two ports SDA and SCL, it is output to the pin, and when loaded, the pin level is read. Since the terminal is a pseudo open-drain terminal, writing "0" outputs LOW, and writing "1" outputs high-Z. Since it is pulled up externally, HIGH is transmitted to the other slave side device. Normally, the output value is read by reading, but even if "1" is written, "0" will be read if the other slave side is outputting 0.

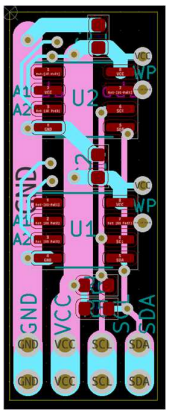
E2PROM write/read demo program with I2C interface

Here is an example of an E2PROM write/read control program with an I2C interface.

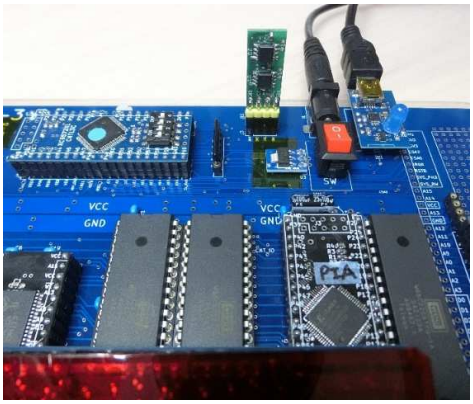
The next figure is a circuit diagram and a layout diagram of an E2PROM board with two CAT24M01s mounted. CAT24M01 has a capacity of 1Mbit (128KB), so two of them will have a capacity of 256K bytes.

E2PROM CAT24M01 board





This E2PROM board is used by connecting to the SDA and SCL terminals added by Verilog description to the PIA2 CPLD.



```
1     HEX
2     ( ******************************************************* )
3     (  E2PROM CAT24M01 read/write demo                        )
4     (  8bit 128k = 1024Mbit  , 256byte/page                   )
5     ( ******************************************************* )
6     ( Frame format for E2P write                              )
7     ( 1st byte  : slave address                               )
8     (      b7-4 : 1010                                        )
9     (      b3   : A2    0 for this board                      )
10    (      b2   : A1    00: BANK0  01: BANK1                  )
11    (      b1   : A16   10: BANK2  11: BANK3                  )
12    (      b0   : R/W#                                        )
13    ( 2nd_byte : E2P addr High  A15-A8                        )
14    ( 3rd_byte : E2P addr Low   A7-A0                         )
15    ( 4th_and_following_data   1 to 256 byte                  )
16    ( ******************************************************* )
17
18    0 VARIABLE BK#     ( 0=BANK0, 1=BANK1, 2=BANK2, 3=BANK3 )
19    : BK ( N -- N ) BK# @ DUP + OR ;
20    A0 CONSTANT SLA_W    ( SLAVE ADDRESS FOR WRITE BANK0 )
21    A1 CONSTANT SLA_R    ( SLAVE ADDRESS FOR READ  BANK0 )
22    100 CONSTANT BSIZE   ( BUFFER SIZE 256 )
23    FF CONSTANT BSIZE-1  (     BSIZE-1 255 )
24    0 VARIABLE BUF-A BSIZE ALLOT
25    0 VARIABLE BUF-B BSIZE ALLOT
26    0 VARIABLE BUF-C BSIZE ALLOT
27    0 VARIABLE BUSY       ( BUSY FLAG )
28
29    : BYTE_WRITE ( BYTE -- )
30    I2C-TX IF ." NO_ACK ERROR" CR THEN ;
31    : BYTE_WRITE_CHECKB ( BYTE -- )
32    I2C-TX IF ." **NOACK** "  -1 BUSY ! THEN ;
33
34    : BUF_ASCEND_FILL ( BUF_ADR -- )
35       BSIZE 0 DO DUP I SWAP C! 1+ LOOP DROP ;
36    : BUF_DECEND_FILL ( BUF_ADR -- )
37       BSIZE-1 + BSIZE 0 DO DUP I SWAP C! 1- LOOP DROP ;
38    : BUF_FF_FILL ( A -- )
39       BSIZE 0 DO DUP FF SWAP C! 1+ LOOP DROP ;
40    : BUF_00_FILL ( A -- )
41       BSIZE 0 DO DUP 00 SWAP C! 1+ LOOP DROP ;
42    : BUF_AA_FILL ( A -- )
43       BSIZE 0 DO DUP AA SWAP C! 1+ LOOP DROP ;
44    : BUF_55_FILL ( A -- )
45       BSIZE 0 DO DUP 55 SWAP C! 1+ LOOP DROP ;
46
47    : BUF. ( BUF_ADR -- )
48      BASE @ >R HEX
49      BSIZE 0 DO
50        DUP C@ U.2 SPACE 1+ I 1+ 10 MOD 0= IF CR THEN
        LOOP DROP CR
        R> BASE !
      ;
```

continuated

```
51    : READY? ( -- READY_FLAG )
52       0 BUSY !
53       I2C-START                  ( START CONDITION          )
54       SLA_W BK BYTE_WRITE_CHECKB ( SLAVE_ADR+W , CHECK BUSY )
55       0 BYTE_WRITE_CHECKB        ( E2P_ADR_H , CHECK BUSY   )
56       0 BYTE_WRITE_CHECKB        ( E2P_ADR_L , CHECK BUSY   )
57       I2C-START
58       SLA_R BK BYTE_WRITE_CHECKB ( SLAVE_ADR+R , CHECK BUSY )
59       1 I2C-RX  DROP             ( DUMMY READ               )
60       I2C-STOP                   ( STOP CONDITION           )
61       BUSY @ NOT
62      ( BUSY @ 0= IF ." READY" 1 ELSE ." BUSY" 0 THEN CR )
63    ;
64
65    : E2P_WRITE ( BUF_ADR E2P_ADR -- ) ( ** BUFFER WRITE ** )
66       BEGIN READY? UNTIL              ( WAIT UNTIL READY   )
67       I2C-START                       ( START CONDITION    )
68       SLA_W BK BYTE_WRITE             ( SLAVE_ADR+W        )
69       DUP 100 / BYTE_WRITE            ( E2P_ADR_H          )
70       FF AND BYTE_WRITE               ( E2P_ADR_L          )
81       BSIZE 0 DO DUP I + C@ BYTE_WRITE LOOP DROP
82       I2C-STOP                        ( STOP CONDITION     )
83    ;
84
85    : E2P_READ ( BUF_ADR E2P_ADR -- )  ( ** BUFFER READ **  )
86     ." R " BEGIN READY? UNTIL         ( WAIT UNTIL READY   )
87       I2C-START                       ( START CONDITION    )
88       SLA_W BK BYTE_WRITE             ( SLAVE_ADR+W        )
89       DUP 100 / BYTE_WRITE            ( E2P_ADR_H          )
90       FF AND BYTE_WRITE               ( E2P_ADR_L          )
91       I2C-START                       ( START CONDITION    )
92       SLA_R BK BYTE_WRITE             ( SLAVE_ADR+R        )
93       BSIZE-1 0 DO DUP I + 0 I2C-RX SWAP C! LOOP
94       BSIZE-1 + 1 I2C-RX SWAP C!    ( LAST READ WITH NOACK )
95       I2C-STOP                        ( STOP CONDITION     )
96    ;
```

```
: TEST ( -- )
INIT-PIA2-I2C
BUF-A BUF_ASCEND_FILL
BUF-B BUF_FF_FILL
BUF-C BUF_DECEND_FILL

CR
." BUFFER A:" CR BUF-A BUF.
." BUFFER B:" CR BUF-B BUF.
." BUFFER C:" CR BUF-C BUF.

BUF-A 0000 E2P_WRITE
BUF-B 0100 E2P_WRITE
BUF-C 0200 E2P_WRITE

BUF-C 0000 E2P_READ
BUF-A 0100 E2P_READ
BUF-B 0200 E2P_READ

CR
." BUFFER A:" CR BUF-A BUF.
." BUFFER B:" CR BUF-B BUF.
." BUFFER C:" CR BUF-C BUF.
;
```

Example of OLED control with I2C connection

In order to output characters to OLED, there are things that must be added in advance other than the I2C driver. One is a word for dealing with STRING which is not defined in FORTH of the AIM 65, and the other is FONT data.

Preparation 1 Definition of STRING WORD

```
( ********** STRING WORDS ********** )
DECIMAL
: SRCH
  DUP BEGIN DUP
  C@ SWAP 1+ SWAP
  0= END SWAP - 1- ;

: STRING
  <BUILDS ABS
  255 MIN 1 MAX DUP
  C,
  0 DO 32 C, LOOP 0 C,
  DOES> 1+ DUP SRCH ;

0 VARIABLE IB
254 ALLOT

: (")
  R COUNT DUP 1+
  R> + >R ;

: "
  34 STATE @ IF
  COMPILE (") WORD
  HERE C@ 1+ ALLOT
  ELSE WORD HERE COUNT
  IB SWAP ROT OVER IB
  SWAP 1+ CMOVE 2DUP
  + 0 SWAP C! THEN ;
  IMMEDIATE

: VAL
  OVER + BL SWAP
  C! 1- NUMBER ;

: STR$
  SWAP OVER DABS
  <# #S SIGN #> ;

: MLEN
  DROP 1- C@ ;

: S!
  DROP DUP 1- C@
  ROT MIN 1 MAX 2DUP
  + 0 SWAP C! CMOVE ;

: MID$
  SWAP >R ROT
  MIN 1 MAX SWAP OVER
  MAX OVER - 1+ SWAP
  R> + 1- SWAP OVER
  SRCH MIN ;

: LEFT$
  >R >R 1 SWAP
  R> R> MID$ ;

: RIGHT$
  >R >R 256

  R> R> MID$ ;

: S+
  ROT >R ROT R>
  SWAP OVER IB SWAP
  CMOVE SWAP OVER +
  255 MIN DUP >R OVER
  - SWAP IB + SWAP
  CMOVE R> 0 OVER IB
  * C! IB SWAP ;

: SUB
  ROT MIN 1 MAX
  CMOVE ;

: S=
  ROT OVER
  = IF 1 SWAP 0 DO
  DROP OVER
  C@ OVER C@ = IF 1+
  SWAP 1+ SWAP 1 ELSE
  0 LEAVE THEN LOOP
  ELSE DROP 0 THEN
  SWAP
  DROP SWAP DROP ;
```

```
    : LEN
      SWAP DROP ;
```

Preparation 2 Prepare FONT data (5x8 dot)

```
1     ( ********** FONT 5X8 ********** )
2     HEX
3     0000 VARIABLE FONT
4           0000 , 0000 ,     ( 32: Space )
5     0000 , 4F00 , 0000 ,    ( 33 !      )
6     0007 , 0007 , 0000 ,    ( 34 ″      )
7     147F , 147F , 1400 ,    ( 35 #      )
8     242A , 7F2A , 1200 ,    ( 36 $      )
9     2313 , 0864 , 6200 ,    ( 37 %      )
10    3649 , 5522 , 5000 ,    ( 38 &      )
11    0005 , 0700 , 0000 ,    ( 39 '      )
12    001C , 2241 , 0000 ,    ( 40        )
13    0041 , 221C , 0000 ,    ( 41        )
14    1408 , 3E08 , 1400 ,    ( 42 *      )
15    0808 , 3E08 , 0800 ,    ( 43 +      )
16    0050 , 3000 , 0000 ,    ( 44 ,      )
17    0808 , 0808 , 0800 ,    ( 45 -      )
18    0060 , 6000 , 0000 ,    ( 46 .      )
19    2010 , 0804 , 0200 ,    ( 47 /      )
20    3E51 , 4945 , 3E00 ,    ( 48 0      )
21    0042 , 7F40 , 0000 ,    ( 49 1      )
22    4261 , 5149 , 4600 ,    ( 50 2      )
23    2141 , 454B , 3100 ,    ( 51 3      )
24    1814 , 127F , 1000 ,    ( 52 4      )
25    2745 , 4545 , 3900 ,    ( 53 5      )
      3C4A , 4949 , 3000 ,    ( 54 6      )
      0171 , 0905 , 0300 ,    ( 55 7      )
      3649 , 4949 , 3600 ,    ( 56 8      )
      0649 , 4929 , 1E00 ,    ( 57 9      )
      0036 , 3600 , 0000 ,    ( 58 :      )
      0056 , 3600 , 0000 ,    ( 59 ;      )
      0814 , 2241 , 0000 ,    ( 60 <      )
      1414 , 1414 , 1400 ,    ( 61 =      )
      0041 , 2214 , 0800 ,    ( 62 >      )
      0201 , 5109 , 0600 ,    ( 63 ?      )
      3249 , 7941 , 3E00 ,    ( 64 @      )
      7E11 , 1111 , 7E00 ,    ( 65 A      )
      7F49 , 4949 , 3600 ,    ( 66 B      )
      3E41 , 4141 , 2200 ,    ( 67 C      )
      7F41 , 4122 , 1C00 ,    ( 68 D      )
      7F49 , 4949 , 4100 ,    ( 69 E      )
      7F09 , 0909 , 0100 ,    ( 70 F      )
      3E41 , 4949 , 7A00 ,    ( 71 G      )
      7F08 , 0808 , 7F00 ,    ( 72 H      )
      0041 , 7F41 , 0000 ,    ( 73 I      )
      2040 , 413F , 0100 ,    ( 74 J      )
      3E41 , 4141 , 3E00 ,    ( 79 O      )
      7F09 , 0909 , 0600 ,    ( 80 P      )
      3E41 , 5121 , 5E00 ,    ( 81 Q      )
      7F09 , 1929 , 4600 ,    ( 82 R      )
      4649 , 4949 , 3100 ,    ( 83 S      )
      0101 , 7F01 , 0100 ,    ( 84 T      )
      3F40 , 4040 , 3F00 ,    ( 85 U      )
      1F20 , 4020 , 1F00 ,    ( 86 V      )
      3F40 , 3840 , 3F00 ,    ( 87 W      )
      6314 , 0814 , 6300 ,    ( 88 X      )
      0708 , 7008 , 0700 ,    ( 89 Y      )
      6151 , 4945 , 4300 ,    ( 90 Z      )
      007F , 4141 , 0000 ,    ( 91 [      )
      0204 , 0810 , 2000 ,    ( 92 ¥      )
      0041 , 417F , 0000 ,    ( 93 ]      )
      0402 , 0102 , 0400 ,    ( 94 ^      )
      4040 , 4040 , 4000 ,    ( 95 _      )
      0102 , 0400 , 0000 ,    ( 96 `      )
      2054 , 5454 , 7800 ,    ( 97  a     )
      7F48 , 4848 , 3000 ,    ( 98  b     )
      3844 , 4444 , 4400 ,    ( 99  c     )
      3048 , 4848 , 7F00 ,    ( 100 d     )
      3854 , 5454 , 5800 ,    ( 101 e     )
      0008 , 7E09 , 0200 ,    ( 102 f     )
      4854 , 5454 , 3C00 ,    ( 103 g     )
      7F08 , 0808 , 7000 ,    ( 104 h     )
      0000 , 7A00 , 0000 ,    ( 105 i     )
      2040 , 403D , 0000 ,    ( 106 j     )
      7F20 , 2844 , 0000 ,    ( 107 k     )
      0041 , 7F40 , 0000 ,    ( 108 l     )
      7C04 , 3804 , 7C00 ,    ( 109 m     )
      7C08 , 0404 , 7800 ,    ( 110 n     )
      3844 , 4444 , 3800 ,    ( 111 o     )
      7C14 , 1414 , 0800 ,    ( 112 p     )
      0814 , 1414 , 7C00 ,    ( 113 q     )
      7C08 , 0404 , 0800 ,    ( 114 r     )
      4854 , 5454 , 2400 ,    ( 115 s     )
      0404 , 3F44 , 2400 ,    ( 116 t     )
      3C40 , 4040 , 3C00 ,    ( 117 u     )
      1C20 , 4020 , 1C00 ,    ( 118 v     )
      3C40 , 3040 , 3C00 ,    ( 119 w     )
      4428 , 1028 , 4400 ,    ( 120 x     )
      0448 , 3008 , 0400 ,    ( 121 y     )
      4464 , 544C , 4400 ,    ( 122 z     )
      0836 , 4141 , 0000 ,    ( 123 {     )
      0000 , 7700 , 0000 ,    ( 124 |     )
```

```
7F08 , 1422 , 4100 ,     ( 75 K      )
7F40 , 4040 , 4000 ,     ( 76 L      )
7F02 , 0C02 , 7F00 ,     ( 77 M      )
7F04 , 0810 , 7F00 ,     ( 78 N      )
```

```
0041 , 4136 , 0800 ,     ( 125 }     )
0402 , 0202 , 0100 ,     ( 126 ~     )
FFFF , FFFF , FF00 ,     ( 127 DEL   )

DECIMAL
( TRANSLATES ASCII TO ADDRESS OF
BITPATTERNS. )
: ASCII>BITPATTERN ( C -- C-ADDR )
  32 MAX 127 MIN
  32 - 6 * FONT +
;
```

Demo program for character output to OLED (1/2)

```
51    HEX
52    ( ********************************************** )
53    (  OLED 128x64 dot demo                          )
54    (  CB1 18pin ==> SDA     CB2 19pin ==> SCL       )
55    ( ********************************************** )
56    78 CONSTANT SLA ( SLAVE ADDRESS       )
57    8 CONSTANT PLM  ( PAGE_LIMIT          )
58    0 VARIABLE OPA  ( OLED_PAGE_ADDRESS   )
59    0 VARIABLE OCA  ( OLED_COLUMN_ADDRESS )
60
61    : OLED_WRITE ( B -- ) I2C-TX IF ." NO_ACK ERROR" CR THEN ;
62    : OLED_PAGE_SET ( PAGE# -- )
63    I2C-START
64    SLA OLED_WRITE ( SLAVE_ADR             )
65    00 OLED_WRITE  ( CMD/DATA CONTINUE BIT )
66    DUP OPA !
67    7 AND B0 + OLED_WRITE ( SET START PAGE )
68    I2C-STOP
69    ;
70    : OLED_COL_SET ( COL# -- )
81    I2C-START
82    SLA OLED_WRITE ( SLAVE_ADR             )
83    00 OLED_WRITE  ( CMD/DATA CONTINUE BIT )
84    DUP OCA ! 6 *
85    DUP 0F AND OLED_WRITE         ( LOWER COL# SET )
86    10 / 0F AND 10 + OLED_WRITE   ( HIGER COL# SET )
87    I2C-STOP
88    ;
89    : OLED_INIT
90    INIT-PIA-I2C
91    I2C-START
92    SLA OLED_WRITE ( SLAVE_ADR             )
93    00 OLED_WRITE  ( CMD/DATA CONTINUE BIT )
94    A8 OLED_WRITE
95    3F OLED_WRITE
96    D3 OLED_WRITE
      00 OLED_WRITE
      40 OLED_WRITE
      A1 OLED_WRITE  ( A0/A1 )
      C8 OLED_WRITE  ( C0/C8 )
      DA OLED_WRITE
      02 OLED_WRITE
      81 OLED_WRITE
      7F OLED_WRITE
      A4 OLED_WRITE
      A6 OLED_WRITE
      D5 OLED_WRITE
      80 OLED_WRITE
      8D OLED_WRITE
      14 OLED_WRITE
      AF OLED_WRITE
```

```
DA OLED_WRITE  ( OLED NO2 ONLY )
12 OLED_WRITE  ( OLED NO2 ONLY )
I2C-STOP
0 OLED_PAGE_SET
0 OLED_COL_SET
;
: OLED_CLEAR ( -- )
0 OLED_COL_SET
PLM 0 DO I OLED_PAGE_SET
  I2C-START
  SLA OLED_WRITE
  40 OLED_WRITE
  80 0 DO 0 OLED_WRITE LOOP
  I2C-STOP
LOOP
0 OLED_PAGE_SET
0 OLED_COL_SET
;
```

Demo program for character output to OLED (2/2)

```
51    : OLED_NL ( -- )  ( NEW_LINE )
52    OPA @ 1+ DUP PLM = IF DROP 0 THEN DUP OPA ! OLED_PAGE_SET
53    0 OLED_COL_SET
54      I2C-START
55      SLA OLED_WRITE
56      40 OLED_WRITE
57      80 0 DO 0 OLED_WRITE LOOP
58      I2C-STOP
59    0 OLED_COL_SET
60    OPA @ OLED_PAGE_SET
61    ;
62    : OLED_EMIT ( CHAR -- )
63      ASCII>BITPATTERN DUP 4 + SWAP DUP 2 + SWAP
64      I2C-START
65      SLA OLED_WRITE
66      40 OLED_WRITE
67      3 0 DO
68        @ DUP 100 /
69        OLED_WRITE OLED_WRITE
70      LOOP
81      I2C-STOP
82    ;
83    : OLED_TYPE ( C-ADR LEN -- )
84    DUP IF OVER + SWAP DO I C@ OLED_EMIT LOOP ELSE 2DROP THEN ;
85
86    : OLED_DEMO1
87            14 0 DO I 20 + OLED_EMIT LOOP
88    OLED_NL 14 0 DO I 34 + OLED_EMIT LOOP
89    OLED_NL 14 0 DO I 48 + OLED_EMIT LOOP
90    OLED_NL 14 0 DO I 5C + OLED_EMIT LOOP
91    OLED_NL 14 0 DO I 70 + OLED_EMIT LOOP
92    OLED_NL 14 0 DO I 20 + OLED_EMIT LOOP
93    OLED_NL 14 0 DO I 34 + OLED_EMIT LOOP
94    OLED_NL 14 0 DO I 48 + OLED_EMIT LOOP
95    ;
96    : OLED_DEMO2
            " 128x64 OLED DEMO    "  OLED_TYPE
      OLED_NL " ==================="  OLED_TYPE
      OLED_NL " ROCKWELL AIM65      "  OLED_TYPE
      OLED_NL " 8MHZ, F:20KB, R:40KB"  OLED_TYPE
      OLED_NL " Serial Terminal IF  "  OLED_TYPE
      OLED_NL " 16SEG x 20 char LED "  OLED_TYPE
      OLED_NL " Thermal Printer     "  OLED_TYPE
      OLED_NL " QWERTY Keyboard     "  OLED_TYPE
      ;
      : DELAY5 ( -- ) 50000 0 DO DUP DROP LOOP ;

      : MAIN_OLED2 ( -- )
      OLED_INIT
      BEGIN
        OLED_CLEAR OLED_DEMO1 DELAY5
        OLED_CLEAR OLED_DEMO2 DELAY5
      AGAIN
```

```
;

: DEMO2 ( -- )
OLED_INIT
BEGIN
  OLED_CLEAR
  OLED_DEMO1 DELAY5
  OLED_CLEAR
  OLED_DEMO2 DELAY5
AGAIN
;
```

7.11 About MATH PACKAGE

The AIM 65 comes with a 4KB ROM MATH PACKAGE (hereafter referred to as MATHPACK) that supports real number operations for FORTH. Invoking the N key invokes FORTH with the MATH PACK word added, enabling floating point arithmetic. In AIM65-CPLD-3V3, MATHPACK is already written to FLASH memory together with FORTH. I also confirmed that it can actually start. However, since there is no manual or sample code for MATHPACK, and no explanation of the format of the data on the data stack, it is not yet usable. If the operation of each word is clear, it seems that a complex number calculator can be written relatively easily.

The words added in MATHPACK are described below.

MATH PACKAGE FORTH WORDS (A65-040)*

FLOATING POINT ARITHMETIC

F+	Adds two floating point numbers.
F–	Subtracts one floating point number from another floating point number.
F*	Multiplies two floating point numbers.
F/	Divides one floating point number by another floating point number.

UTILITY, SIGN AND COMPARISONS

FABS	Takes the absolute value of a floating point number.
INT	Truncates a floating point number to an integer.
SGN	Converts the sign of a floating point number to a floating point number.
FSIGN	Gets a value corresponding to the sign of a floating point number.
FCOMP	Compares the value of a compacted number in memory to a floating point number.

POLYNOMIAL

POLY	Evaluates a polynomial with consecutive exponents.
POLYODD	Evaluates a polynomial with odd exponents.

EXPONENTIAL AND LOGARITHMIC

SQR	Takes the square root of a floating point number.
>	Raises one floating point number to the power of another floating point number.
EXP	Raises the transcendental number e to the power of a floating point number.
LOG	Computes the logarithm to the base 10 (i.e., common log) of a floating point number.
LN	Computes the logarithm to the base e (i.e., natural log) of a floating point number.

*Requires AIM 65 FORTH or RM 65 FORTH be resident.

USER VARIABLE

MIN-WIDTH	Specifies the minimum field width to be output.
DEC-LENGTH	Specifies the number of places to the right of the decimal point to be output.

ASCII/FLOATING POINT CONVERSIONS

FIN	Converts a number in memory from ASCII to floating point format.
FOUT	Converts a number from floating point to ASCII.

FORMAT CONVERSION AND DATA MOVING

M>F	Unpacks the compacted number in memory to floating point.
F>M	Packs the floating point number to compacted format and stores the result in memory.
M>A	Unpacks the floating point number in memory.
S>A	Converts an integer to floating point format.
S>F	Converts an integer to floating point format.
F>S	Converts a number from floating point to an integer.

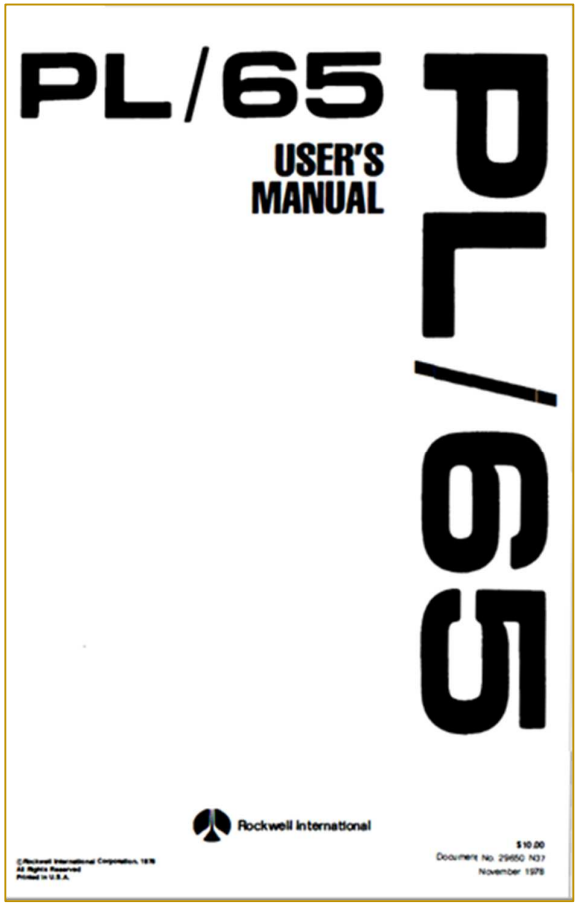
TRIGONOMETRIC AND UNITS CONVERSION

SIN	Calculates the sine of a floating point number (in radians).
COS	Calculates the cosine of a floating point number (in radians).
TAN	Calculates the tangent of a floating point number (in radians).
ARCTAN	Calculates the arc tangent of a floating point number.
DEGREES	Converts a floating point number from radians to degrees.
RADIANS	Converts a floating point number from degrees to radians.

7.12 Booting PL/65

PL/65 is started by pressing the 5 key in the monitor.

```
 ROCKWELL AIM 65

<5>

AIM 65  PL/65  V1.0

IN=M  OUT=L


PASS(1 OR 2)?1
```



A compilation example will be shown separately, but before that you need to know what PL/65 is.

Even if you explain that PL/65 is a programming language and its compiler that is a combination of PL/1 and ALGOL, it won't help you much because you don't know what kind of languages PL/1 and ALGOL are. My first impression of PL/65 from this sentence was that it was "as old as FORTRAN, does not require line numbers, and allows structured programming." What I understood is "a language that allows structured expression instead of writing assembler code for the 8-bit microcomputer R6502", that is, "a high-level assembler for 6502". However, since the output of PL/65 becomes the input source file of the AIM65 assembler, it is still necessary to apply the work to the assembler. It's kind of like an assembler preprocessor. Errors are not displayed in detail, and there are many errors that can be identified only after running the assembler. Considering that it is a compiler for 8-bit microcomputers and a self-compilation environment that existed from 1975 to 1980, it is useless to expect too advanced functions. Still, with the one-board microcomputers of the time, the instruction code and destination address were manually converted into hexadecimal numbers and typed in using a 7-segment LED and hexadecimal keys.

I will list the features of PL/65 again below.

- It is a self-compiler that runs on AIM-65.
- Use the compiler by loading it from the ROM version or from the RM65 disk system.
- The compiler can be started by inserting it into a socket instead of BASIC or FORTH ROM.
- It is a structured language that does not require line numbers.
- Read PL/65 source file and output assembler source
- All program reserved words are in uppercase letters.
- The only data that can be handled is bytes or a one-dimensional array of bytes (the element number always starts from 0).
- Arithmetic operations can only use + and -, *, / and MOD cannot be used.
- Logical operations include .AND, .OR, .EOR, and .XOR (.NOT as a unary operation).
- Assembler code can be written directly using the CODE statement.
- I/O statements such as INPUT and PRINT are not supported, and it seems that there is no other choice but to call the input/output subroutine that AIM-65 originally has as a function of the monitor program.
-

There is a compiler called PL/M-80 provided by Intel as a development environment for the i8080. http://bitsavers.trailing-edge.com/pdf/intel/PLM/9800268B_PLM-80_Programming_Manual_Jan80.pdf

PL/65 is similar to PL/M-80, but I think PL/M-80 was more established as a development environment. I presume that the presence of the PL/M-80 in 1976 prompted Rockwell to develop a similar PL/65.

Here are the statements taken from PL/65 USER'S MANUAL

Declarations DECLARE DEFINE DATA comment Examples of expressions and assignment statements B = C + D - E + 24; Order SHIFT ROTATE CLEAR SET CODE HALT	WAIT STACK (=push) UNSTACK (=pull) INC INCW DEC DECW PULSE ENTRY/EXIT Conditional IF-THEN-ELSE IF-THEN CASE	Loops FOR-TO-BY WHILE Branches GOTO, CALL, RETURN RTI BREAK

PL/65 sample program

I created a program that outputs while adding 1 to the 16-bit LEDs connected to port A and port B of the VIA for user expansion. PL/65 cannot handle 16-bit variables, so 8-bit variables I and J are used, and 16-bit increments are performed in a double loop consisting of a FOR loop and a WHILE loop. Output to port A and port B is executed by assignment to DRA and DRB. The delay at the time of LED display is adjusted by software, and the subroutine is called with the name DELAY (label defined with DELAY:). This will finish PL/65 PASS1, PASS2 and assembler without error.

with the monitor turned on

*=9000⏎

G⏎

/⏎

If you enter and execute the machine code, the incremented value will be output to the LEDs connected to port A and port B of the user extension VIA. However, the output to port A is from 0 to 254, skipping 255.

LISTING of LED BLINK demo program by PL/65

```
1     " *** PL/65 PROGRAM *********** ";
2     " *** LED-BLINK (INCREMENT) *** ";
3     ;
4     ;" R6522 VIA REGISTER ADRESS ";
5     DEF DRB  = $A000;
6     DEF DDRB = $A002;
7     DEF DDRA = $A003;
8     DEF DRA  = $A00F;
9     ;
10    DCL I,J,K,L,M;
11    ;
12    ENTRY $9000;
13    ;
14    DDRA=$FF; "PA7-0 output"
15    DDRB=$FF; "PB7-0 output"
16    DRA=$FF;
17    DRB=$FF;
18    I=0;
19    K=1;
20    WHILE K=1
21    DO;
22      FOR J=0 TO 254
23        BEGIN;
24          DRA=J;
25          DRB=I;
26          CALL DELAY;
27        END;
28      I=I+1;
29    END;
30    DELAY:
31      FOR L=0 TO 50
32      BEGIN;
33        M=L+1;
34      END;
35    RETURN;
36    ;
37    EXIT;
```

The problem is the FOR statement.
The listing is an example that works as described, but on line 22

FOR J=0 TO 254 for example

FOR J=253 TO 255

If you specify 255 as the final value like this, it will not work as intended. It compiles and assembles without error, but in this case it loops around J=255 and then back to J=0. That is, repeat FOR J=0 TO 255 indefinitely after reaching 255. It will not be executed after I=I+1 on the 28th line. It may be a specification of PL/65, but it behaves so much that it seems to be a bug.

The PL/65 manual says

FOR J=5 TO 4 . .. ; does not execute the loop itself at all. And

FOR J=1 TO 1 .....; will be an infinite loop.

There is a comment like this, but there is no note when the upper limit is 255.

It's 8 bit, so at least I want it to work correctly up to 255. So, I understand that PL/65 is about this level.

PL/65 PASS1

```
<5>
AIM 65  PL/65  V1.0
IN=M  OUT=L

PASS(1 OR 2)?1
;" *** PL/65 PROGRAM ********** ";
;" *** LED-BLINK (INCREMENT) *** ";
;;
;;" R6522 VIA REGISTER ADRESS ";
;DEF DRB  = $A000;
;DEF DDRB = $A002;
;DEF DDRA = $A003;
;DEF DRA  = $A00F;
;;
;DCL I,J,K,L,M;
;;
;ENTRY $9000;
;;
;DDRA=$FF; "PA7-0 output"
;DDRB=$FF; "PB7-0 output"
;DRA=$FF;
;DRB=$FF;
;I=0;
;K=1;
;WHILE K=1
;DO;
;  FOR J=0 TO 254
;    BEGIN;
;      DRA=J;
;      DRB=I;
;      CALL DELAY;
;    END;
;  I=I+1;
;END;
;DELAY:
;  FOR L=0 TO 50
;  BEGIN;
;    M=L+1;
;  END;
;RETURN;
;;
;EXIT;

ERRORS= 00
```

PL/65 PASS 2

```
<5>
AIM 65  PL/65  V1.0
IN=M  OUT=L

PASS(1 OR 2)?2
;" *** PL/65 PROGRAM ********** ";
;" *** LED-BLINK (INCREMENT) *** ";
;;
;;" R6522 VIA REGISTER ADRESS ";
;DEF DRB  = $A000;
DRB=$A000
;DEF DDRB = $A002;
DDRB=$A002
;DEF DDRA = $A003;
DDRA=$A003
;DEF DRA  = $A00F;
DRA=$A00F
;;
;DCL I,J,K,L,M;
I  *=*+1
J  *=*+1
K  *=*+1
L  *=*+1
M  *=*+1
;;
;ENTRY $9000;
 *=$9000
R0=0
R1=R0+2
R2=R1+1
R3=R2+1
CLD
LDX #$FF
TXS
;;
;DDRA=$FF; "PA7-0 output"
LDA #$FF
STA DDRA
;DDRB=$FF; "PB7-0 output"
LDA #$FF
STA DDRB
;DRA=$FF;
LDA #$FF
STA DRA
;DRB=$FF;
LDA #$FF
STA DRB
;I=0;
LDA #0
STA I
;DO;
;  FOR J=0 TO 254
LDA #0
STA J
ZZ0003 CMP #254
BEQ *+7
BCC *+5
JMP ZZ0004
;    BEGIN;
;      DRA=J;
LDA J
STA DRA
;      DRB=I;
LDA I
STA DRB
;      CALL DELAY;
JSR DELAY
;    END;
INC J
LDA J
JMP ZZ0003
ZZ0004
;  I=I+1;
LDA I
CLC
ADC #1
STA I
;END;
JMP ZZ0001
ZZ0002
;DELAY:
DELAY
;  FOR L=0 TO 50
LDA #0
STA L
ZZ0005 CMP #50
BEQ *+7
BCC *+5
JMP ZZ0006
;  BEGIN;
;    M=L+1;
LDA L
CLC
ADC #1
STA M
;  END;
INC L
LDA L
JMP ZZ0005
ZZ0006
;RETURN;
RTS
;;
```

```
;K=1;
LDA #1
STA K
;WHILE K=1
ZZ0001 LDA K
CMP #1
BEQ  *+5
JMP ZZ0002
```

```
;EXIT;
.END

ERRORS= 00
```

ASSEMBLER

```
<N>
ASSEMBLER
FROM=2000 TO=2FFF
IN=M
LIST?Y
LIST-OUT=L

OBJ?N
PASS 1

PASS 2


==0000
;" *** PL/65 PROGRAM ********** ";
;" *** LED-BLINK (INCREMENT) *** ";
;;
;;" R6522 VIA REGISTER ADRESS ";
;DEF DRB  = $A000;
==0000 DRB=$A000
;DEF DDRB = $A002;
==0000 DDRB=$A002
;DEF DDRA = $A003;
==0000 DDRA=$A003
;DEF DRA  = $A00F;
==0000 DRA=$A00F
;;
;DCL I,J,K,L,M;
==0000 I
       *=*+1
==0001 J
       *=*+1
==0002 K
       *=*+1
==0003 L
       *=*+1
==0004 M
       *=*+1
;;
;ENTRY $9000;
==0005
       *=$9000
```

```
8502   STA K
==9020
;WHILE K=1
==9020 ZZ0001
A502   LDA K
C901   CMP #1
F003   BEQ *+5
4C5490 JMP ZZ0002
;DO;
;  FOR J=0 TO 254
A900   LDA #0
8501   STA J
==902D ZZ0003
C9FE   CMP #254
F005   BEQ *+7
9003   BCC *+5
4C4A90 JMP ZZ0004
;    BEGIN;
;      DRA=J;
A501   LDA J
8D0FA0 STA DRA
;      DRB=I;
A500   LDA I
==903D
8D00A0 STA DRB
;      CALL DELAY;
205490 JSR DELAY
;    END;
E601   INC J
A501   LDA J
4C2D90 JMP ZZ0003
==904A ZZ0004
;  I=I+1;
A500   LDA I
18     CLC
6901   ADC #1
8500   STA I
;END;
4C2090 JMP ZZ0001
==9054 ZZ0002
;DELAY:
==9054 DELAY
;  FOR L=0 TO 50
```

```
==9000 R0=0
==9000 R1=R0+2
==9000 R2=R1+1
==9000 R3=R2+1
D8     CLD
A2FF   LDX #$FF
9A     TXS
;;
;DDRA=$FF; "PA7-0 output"
A9FF   LDA #$FF
8D03A0 STA DDRA
;DDRB=$FF; "PB7-0 output"
A9FF   LDA #$FF
8D02A0 STA DDRB
;DRA=$FF;
A9FF   LDA #$FF
==9010
8D0FA0 STA DRA
;DRB=$FF;
A9FF   LDA #$FF
8D00A0 STA DRB
;I=0;
A900   LDA #0
8500   STA I
;K=1;
A901   LDA #1
A900   LDA #0
8503   STA L
==9058 ZZ0005
C932   CMP #50
F005   BEQ *+7
9003   BCC *+5
4C6F90 JMP ZZ0006
;  BEGIN;
;    M=L+1;
A503   LDA L
18     CLC
6901   ADC #1
8504   STA M
==9068
;  END;
E603   INC L
A503   LDA L
4C5890 JMP ZZ0005
==906F ZZ0006
;RETURN;
60     RTS
;;
;EXIT;
       .END
 ERRORS= 0000
```

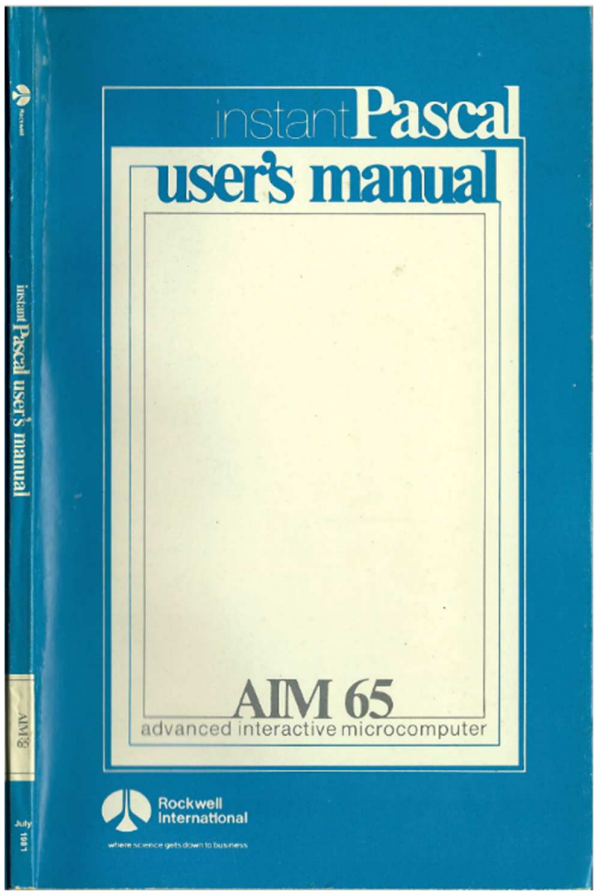
Disassembler listing of the generated machinecode:

```
<K>*=9000
/52
 9000 D8 CLD
 9001 A2 LDX #FF
 9003 9A TXS
 9004 A9 LDA #FF
 9006 8D STA A003
 9009 A9 LDA #FF
 900B 8D STA A002
 900E A9 LDA #FF
 9010 8D STA A00F
 9013 A9 LDA #FF
 9015 8D STA A000
 9018 A9 LDA #00
 901A 85 STA 00
 901C A9 LDA #01
 901E 85 STA 02
 9020 A5 LDA 02
 9022 C9 CMP #01
 9024 F0 BEQ 9029
 9026 4C JMP 9054
 9029 A9 LDA #00
 902B 85 STA 01
 902D C9 CMP #FE
 902F F0 BEQ 9036
 9031 90 BCC 9036
 9033 4C JMP 904A
 9036 A5 LDA 01
 9038 8D STA A00F
 903B A5 LDA 00
 903D 8D STA A000
 9040 20 JSR 9054
 9043 E6 INC 01
 9045 A5 LDA 01
 9047 4C JMP 902D
 904A A5 LDA 00
 904C 18 CLC
 904D 69 ADC #01
 904F 85 STA 00
 9051 4C JMP 9020
 9054 A9 LDA #00
 9056 85 STA 03
 9058 C9 CMP #32
 905A F0 BEQ 9061
 905C 90 BCC 9061
 905E 4C JMP 906F
 9061 A5 LDA 03
 9063 18 CLC
 9064 69 ADC #01
 9066 85 STA 04
 9068 E6 INC 03
 906A A5 LDA 03
 906C 4C JMP 9058
 906F 60 RTS
```

7.13 instant Pascal how to start

PASCAL can be started on AIM65-CPLD-3V3 but I did not actually use it.

AIM65 instant Pascal user's manual cover



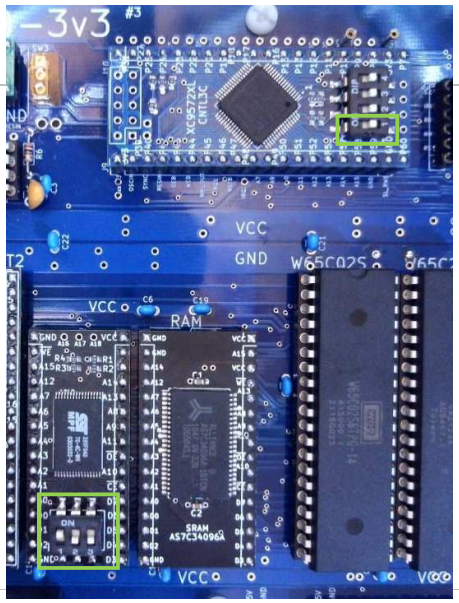
http://retro.hansotten.nl/6502-sbc/aim-65/aim-65/aim-65-software/pascal-for-the-aim-65/

The following settings are required to use Instant Pascal with the AIM65-CPLD-3V3.

(1) Turn ON OFF OFF the DIP SW on the FLASH memory board.

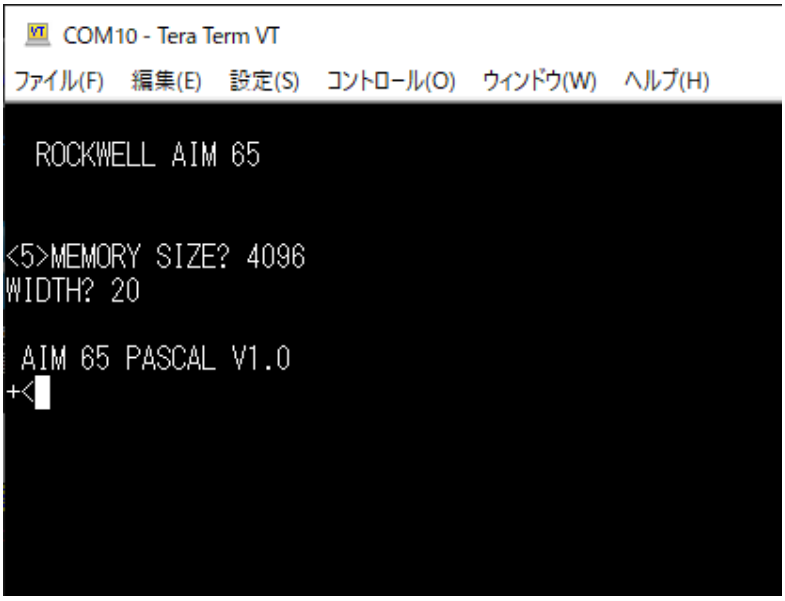
Select Pascal by setting it to this.

(2) Turn off DIP-SW 1 on the control CPLD board and set addresses 4000 to 7FFF in the ROM area.



In this state, turn on the power, set the baud rate with the DEL key, and answer the MEMORY SIZE? and WIDTH? inquiries with ⏎ to display the PASCAL prompt on the teraterm screen. This is the end of the operation check for Pascal.

Instant Pascal startup screen



7.14 [Extra] C Compiler cc65

The AIM 65 does not have a C language self-compiler environment. However, there is a C language cross-compiler for the 6502 that runs on a PC: cc65. It is both a C compiler and an assembler.

Speaking of current PCs, there are 64-bit CPUs with 16 or 24 cores that operate at 4 GHz. In this case, the compiler is determined to have better performance in the cross environment. Since there is such software, AIM65 should also make full use of it.

8 POSTSCRIPT

By cloning the AIM 65, which originally ran at 1MHz, I was able to run it at 16MHz, albeit at an experimental level, and at least I was able to do what I wanted to do. BASIC, FORTH, PL/65, and PASCAL run as they are, so I think I've come to the point where I can play quite a bit. As an impression after creating a reproduction, the AIM 65 has KB mode and TTY mode, but KB mode must be used in the range of 20 characters / line, and it is very inefficient in creating programs. . In that case, TTY mode, which can display multiple lines with 80 characters/line or more, is used, and KB mode is unnecessary. If the thermal printer has one UART-TX channel, it can output characters simply by writing the ASCII code to the TXD register, so it can be used easily even in TTY mode. With TTY mode, you don't need a QWERTY keyboard or a 16-segment x 20-digit LED.

There are other things I want to do about AIM65, such as the following, but I hope that other people will participate in further improvement work, so I would like to conclude the work of AIM 65 revival so far.

- Increase PL/65 sample code. I want a summary and explanation of how to use it.
- Show the sample code of Instant PASCAL. I want a summary and explanation of how to use it.
- Show sample code and demo program of MATHPACK for FORTH.
- ROMize the FORTH demo program and add it to the FLASH memory
- Operate the TTY interface at 9600bps in 8bit mode (original AIM 65 is 7bit).
- Make it possible to connect the output of the above TTY interface to a thermal printer.
- Improve input/output speed by connecting UART (SC16C550) with built-in data buffer
- Port DOS/65 (SD card base).
- Improve speed by using FPGA version or high-performance emulation for CPU core.
- Add a reproduction of the cassette tape interface.

that's all

Revival Retro Computer AIM 65

March 31, 2023 Rev. 0.5 (Japanese)
April 26, 2023 Rev. 0.2 Hans Otten English