



Intellivision Development, Back-In-The-Day

Contents

1	Introduction	3
2	Overall Process	4
3	APh Technological Consulting.....	5
3.1	Host Hardware and Operating System.....	5
3.2	Development Tools	6
3.3	Test Harnesses.....	8
3.4	Tight Finances.....	16
4	Mattel Electronics.....	17
4.1	Host Hardware and Operating System.....	17
4.2	Development Tools	18
4.3	Test Harnesses.....	23
4.4	EPROM and T-Cards.....	32
4.5	Documentation.....	36
4.6	Tight Code.....	36
4.7	Use of High-Level Languages.....	37
4.8	Minkoff Measure	37
4.9	Reverse Engineering.....	38
5	Third Party Development.....	39
5.1	Activision / Cheshire Engineering.....	39
5.2	Atari	39
5.3	Imagic	39
5.4	INTV	40
5.5	Technology Associates	40
6	References	44
6.1	Intellivisionaries Podcasts	44
6.2	Video Interviews.....	45
6.3	Audio Interviews.....	46

1 Introduction

This document is an attempt to collate information about the tools and techniques that were used to develop software and games for the Mattel Intellivision in the 1970s and 1980s. It is largely based on contemporary documentation and the recollections of developers of the time. Clearly, a description constructed nearly 40 years after the event will never be comprehensive and, as it is based in part on individuals' memories, it may be inaccurate or contradictory in places.

A large number of developer interviews are recorded and published in the excellent [Intellivisionaries Podcast](#) (thanks to Paul, Rick and William). A list of episodes and timecodes where development tools or methods are discussed can be found in the references at the end of this document. Additional details obtained from videos of panel discussions at retro gaming exhibitions are also referenced. Finally, a number of documents from the Blue Sky Ranger archive have been made available as part of research into Intellivision history by two UCI Irvine academics.

This version of the document includes information from Intellivisionaries up to episode 38 and comments from the [equivalent Atari Age thread](#) up to June 2020. If you have further information, corrections or insights that should be included in this document, please contact "decle" at the Atari Age forums.

The inspiration for creating this document was [an image of Patrick Aubry](#) working on Intellivision Fireman at Nice Ideas in 1984, shown in Figure 1. In the picture, Patrick appears to be using a [CIT-101 terminal](#), a DEC VT100 clone, presumably to write some code on a remote machine. On the shelf behind him a "Blue Whale" Intellivision Keyboard Component / Master Component combination is being used to test the current state of the game on a portable television.

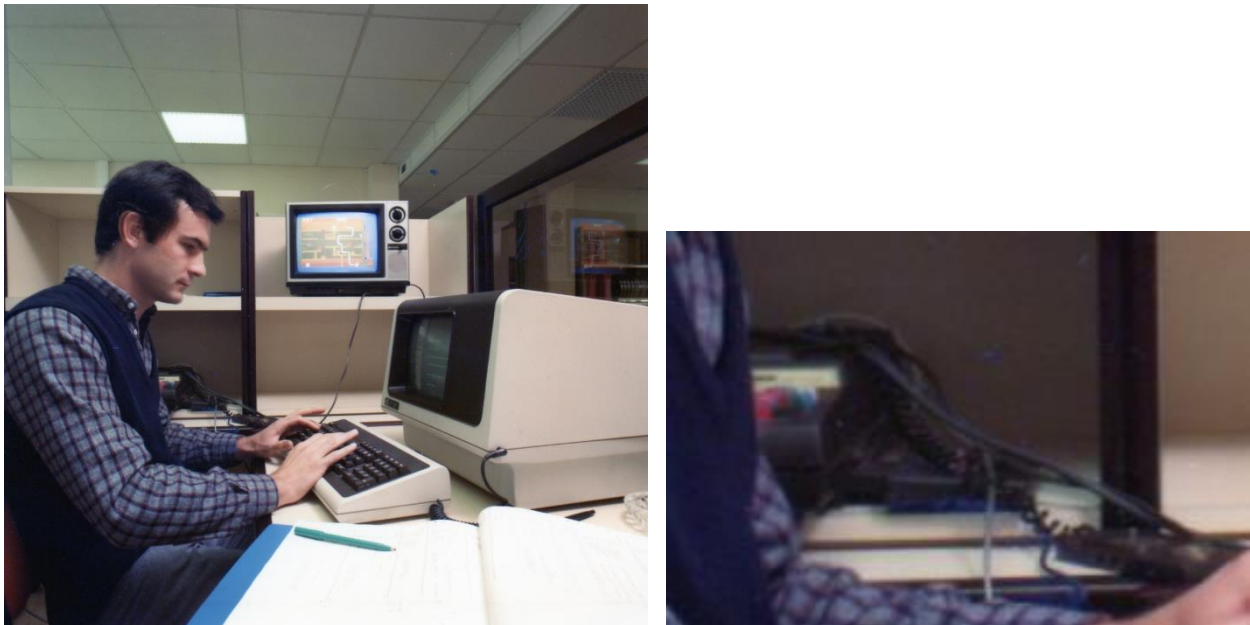


Figure 1: Patrick Aubry Working on Intellivision Fireman at Nice Ideas

So, what do we know about how these and other tools were used to develop the games we have enjoyed for the last 40 years?

2 Overall Process

Perhaps the most succinct description of early development work comes from Keith Robinson and is to be found on [page 5 of issue one of Retro magazine](#):

"For Keith Robinson: Can you describe the computer system (hardware, compilers, etc.) that you used to develop the Intellivision console games? - William McEvoy

Keith responded: The first Intellivision games, developed for Mattel by APh (an outside consulting firm) in Pasadena, were done on PDP-11s.

There were eight PDP-11s (TSX-1, TSX-2, etc.) on the programming floor, each supporting about three to five programmers. Each programmer had an Intellivision console and a television in his or her cubicle, plus a dumb terminal (VT100 compatibles) connected to the Andromedas.

Programming was in the assembly language of the General Instruments CP1610 microprocessor, the heart of the Intellivision. This code was compiled and linked by software custom written by APh.

The object code was then downloaded into a custom piece of hardware (also designed by APh) called a Magus board. The Magus board was mostly memory, connected by a parallel cable to the programmer's Intellivision in place of a cartridge. In this way, within a minute or two of completing a new bit of code, the programmer could compile it, link it, then download it to the Intellivision to see the results on the TV."

This description tallies with much of the setup seen in the picture of Patrick. Development was fundamentally a two-phase process. Software was written in CP-1610 assembler on a shared machine, typically described as a DEC PDP-11. These shared machines were accessed with terminals, like the one Patrick is using. The source code was cross-assembled to CP-1610 machine code on the PDP-11 and the resulting object code downloaded to an Intellivision for evaluation using a test harness.

In Intellivisionaries episode 31 Mark Urbaniec suggests that a typical code, compile, download and test development cycle would take about 15 minutes, with only the first few minutes being spent amending source code.

This broad setup and process is confirmed by numerous Mattel and APh developers, and also seems to have been the fundamental way of working at third party software houses. The following sections examine development at each of the main developers.

3 APh Technological Consulting

The initial development of Intellivision software, including the construction of the EXEC and the first tranche of games, was undertaken by APh Technological Consulting in Pasadena. APh continued to work creating Intellivision games under contract to Mattel up until the console was cancelled in 1984.



3.1 Host Hardware and Operating System

APh appear to have used a mixture of PDP-11 minicomputers and at least one DECSYSTEM-20 mainframe as the hosts for their development.

According to David Rolfe and John Sohl [V2], [V5], [A2] APh used a [DECSYSTEM-20 mainframe](#), also known as a PDP-20. This machine stored the developer's assembly code and the tools required to construct, assemble and test it. David describes the system as running TOPS-10, the PDP-10 operating system. This was possible as the PDP-20 had essentially the same hardware as the earlier PDP-10.

In contrast to these recollections in Intellivisionaries episode 10 Tom Loughry, and later in episode 17 Kimo Yap, refer to the host they used at APh as being a PDP-11. Kimo further clarifies this [on AtariAge](#), describing the host as having an LSI-11 CPU and running RT-11. At the CGE 2K4 panel [A1] Peter Kaminsky named the development host he used while developing Frog Bog as being a PDP-10, but then went on to describe it as a small minicomputer, suggesting it was actually a PDP-11.

The timing of the use of the various machines is not well understood, although the available evidence suggests that a mixture of PDP-11s and the PDP-20 were used for Intellivision development throughout the period. There is circumstantial evidence on [page 98 of the DECUS abstracts](#) that David Rolfe was using a DECSYSTEM-10 mainframe on APh projects from as early as June 1977, see Figure 2.

```
REV, Version: 2(4), June 1977 10-289  
Author: David Rolfe, APH Technological Consulting, Pasadena, CA  
Source Language: MACRO-10 Memory Required: 1K + 2K or, 1K + 3K Core Keywords: File-Handling  
  
Abstract: REV is the product of an attempt to produce the ultimate file manipulation program. Functions are provided to let the user copy, rename, list, type, and delete files by using simple commands. In
```

Figure 2: DECUS entry for David Rolfe's Program REV

Also, the header of a printout of the Keyboard Component tool MERT made in late January 1982 by David (aka TALKIN-HORSE) originates from a machine running TOPS-20, as shown in Figure 3.

```
*START* Job MERT Req #30 for TALKIN-HORSE Date 25-Jan-82 18:57:37 Monitor: APhID, TOPS-20 Monitor 4(32)  
File PS: CLOCAL-DOCUMENTATION\MERT.DOC.5, created: 10-Dec-81 14:22:32  
printed: 25-Jan-82 18:57:52  
Job parameters: Request created: 25-Jan-82 18:57:35 Page limit: 135 Forms: NORMAL Account: APh  
File parameters: Copy: 1 of 1 Spacing: SINGLE File format: ASCII Print mode: ASCII
```

Figure 3: TOPS-20 MERT Printout Header From 1982

David left APH later in 1982 when he joined Cheshire Engineering. It is likely that John Sohl saw the DECSYSTEM-20 mainframe while he was based at the APH offices in late 1980. Kimo Yap only worked at APH over the summer of 1978 when he suggests he used a PDP-11. Finally, like David Rolfe, Tom Loughry worked at APH between 1980 and 1982, before also leaving for Cheshire Engineering. During this time, he states he also used a PDP-11.

3.2 Development Tools

3.2.1 CP-1610 Assembler

It is clear from Keith's description that the fundamental Intellivision development tool was a CP-1610 cross assembler running on the host machine. It is interesting that both Keith Robinson and Kimo Yap suggest that APH wrote their own toolchain, given that General Instrument, the manufacturers of the CP-1610, [supplied one that ran on a PDP-11](#). However, despite the close way in which Mattel and General Instrument worked when developing the Intellivision, this is what happened. The names of the respective assemblers (APH - MX1600, GI - S16XAL), and some of the details of their supported assembly language syntaxes are different.

The reason for the use of the MX1600 assembler becomes clear in an interview with David Rolfe in episode 7 of Intellivisionaries. David explains that APH had an existing and mature cross-platform assembler framework and simulation system. Along with the MX1600 cross-assembler, this toolchain included a linker (LX1600) and a symbolic debugger (PX1600). This technology was the core competitive advantage of APH's business, and it formed the basis of the APH development platform.

The use of APH's range of MX cross-assemblers and LX cross-linkers was described in the document "[Your Friend the Assembler](#)". The introduction to this document implies that the assembler is written in a low level language, and it has been [suggested by Joe Zbiciak](#) that the APH toolset may have been written in the language BLISS. BLISS was a [systems programming language](#) developed for the DEC range of mini-computers. It faded into obscurity following the introduction of C. Further credence is added to this hypothesis by [some APH code](#) obtained by Frank Palazzolo. This program was used to construct Hamming error correction codes for the Keyboard Component tape interface, and appears to be written in a variant of BLISS.

3.2.2 Text Editor

The other important tool for code development is a text or program editor. In the BSR informal video chat [V2] John Sohl mentions that when working at APH his main development environment was [Emacs](#). He noted that repeatedly forking, or pushing, the Emacs process consumed resources and caused problems for his colleagues also working on the host.

3.2.3 Pixel Editor

In the CGE 2K4 panel Peter Kaminsky [A1] briefly describes a piece of hardware used by APH to construct graphics. The design comprised a large number of buttons, similar to the Mattel Graphics Development System described in Section 4.2.4.

3.2.4 MERT, GERT and MODIT

MERT, GERT and MODIT were tools developed by APH to help in the construction of Intellivision Keyboard Component tapes. Software like Jack LaLanne and Conversational French that use the PicSE framework consist of a series of small programs and data files that must be built and placed

at the appropriate point of a Keyboard Component Tape. This is important as it is necessary to load program and data fragments to the correct memory location in a timely manner, such that they are synchronised with any pre-recorded audio. This structure becomes apparent when comparing the definition of this schedule with the running software. Such a comparison is possible for the Keyboard Component Demonstration Cassette where [a MERT schedule is available](#) along with contemporaneous [video of the software in operation](#). Within the video, sections like one containing Space Battle, the Keyboard Component and Conversational French starting at 2:50, can be seen to track blocks within the MERT file, as shown in Figure 4.

```

:***** *****
:***** SPACE BATTLE *****
:***** *****
DATA<$8540> NOCONT $9C00 $BFFF,
             BSDEM 3, ;SPACE BATTLE DEMO
             0

RIB          /TIME:220          ;<<<< SPACE BATTLE >>>>
             REF          MASPIC
             REF          KEYBRD
             REF          FRDEMO

:***** *****
:***** KEYBOARD COMPONENT, FRENCH *****
:***** *****
DATA<$8540> CONT,
             SAC00 ,SBFFF,
    
```

Figure 4: MERT / GERT Script Sections

Each of these MERT blocks represent one or more pieces of code or data that are loaded in order and executed in sync with the audio commentary. The effect of this synchronisation can be clearly seen in the introduction to the benefits of Jack LaLanne’s workout at 4:50 in the video, see Figure 5, where each benefit appears in time with the narrator’s comment.



Figure 5: Jack LaLanne’s Benefits

The format of the MERT schedule suggests that the Keyboard Component targeted audio and software synchronisation with an accuracy of 0.1 sec. If this was achieved reliably it would suggest that [reasonable quality](#) lip-synced animation would have been possible.

At least two versions of MERT and GERT were written by APH, with the second version being documented in [“Your Friends MERT and GERT”](#).

MERT and GERT could be used together with a third program MODIT, to build the two data tracks on Keyboard Component tapes. MERT and GERT used the same script of commands defining the contents and structure of the tape, an example fragment of which is shown in Figure 6.

```

; DEMO CASSETTE MERGE FILE
;
; HEADER/DATA AREA ALLOCATION.
;   2 CHUNKS OF HEADERS
;   1 CHUNK LITERAL DATA:
;
;   BIDECL THIS RECORD NUMBER
;   BIDECL LOWE BOUND ON FOLLOWING RECORD
;   BIDECL UPPE BOUND ON FOLLOWING RECORD
;   BIDECL <ZERO TO STOP AFTER RECORD, NONZERO TO CONCATENATE>
;   BIDECL PROG ADDRESS, CHUNK TO WAIT FOR
;   (MA HAVE UP TO 5 SUCH ENTRIES)
;   BIDECL 0,0
;
; DEFINITIONS OF FILES USED IN DEMO CASSETTE:
;
; ROOT: DEF\ /P:$8560-$87FF EMAP50,P1MAP,ROOT \
;
; PICSE: DEF\ /P:$800-$9BFF /V:$9300-$9BFF EMAP50,P1MAP,PICSE,PSDATA
; PSINT,SCREEN,PSOBJ,PSCODE \
; PUTCOD,MOTION \
;
; BCURT=$9C00
; CURTAN: DEF\ /P:$9C00-$9DFF EMAP50,P1MAP,CTLMAP
; CURTAN \
;
; BINTR=$9E00
; INTRO: DEF\ /P:$9E00-$A3FF /Z:$015D-$01EF EMAP50,P1MAP,CTLMAP
; DSTART,TIMINX,DSOUND,SPARK \
;
; BMAST=$A400
; MASTER: DEF\ /P:$A400-$ABFF EMAP50,P1MAP,CTLMAP
; MASTER \
;
; BBDEM=$AC00
; BDEMO: DEF\ /P:$A00-$BFFF /Z:$015D-$01EF EMAP50,CTLMAP
; BASBLD,KEYS,BDEMO,FIELD
; DEFENS,WHEEL,POSIT,OFFENS
; BASATO,BASPCT,BASSND \

```

Figure 6: Example MERT / GERT Command Script

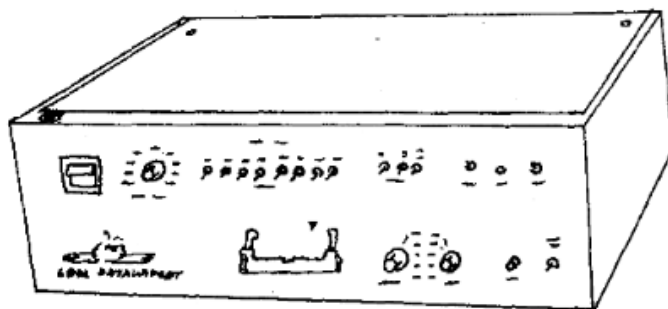
MERT (MERge for Tape) was first used to ensure that the object files listed in the MRT script were up to date. GERT was then used to consolidate the various elements defined in the MRT file into two MOD files, one for the pre-recorded data, the other for the home recorded data track. The final step in the construction of a Keyboard Component tape was to feed the two MOD files to the program MODIT which would drive modulator hardware and write the data to tape.

3.3 Test Harnesses

Having written and assembled the source code of a game on the PDP-20 or PDP-11, it was downloaded to an Intellivision for testing. Rather than use a test cartridge containing EPROMs, development testing was conducted using a dedicated harness that mimicked a cartridge using RAM. This sped up the development cycle as the process of writing the revised game code to the harness was significantly faster than erasing and blowing EPROMs.

APH's test harness, which was known as a Datawidget or Widget for short, was developed by Shal Farley, Chris Lee and Will McCown. Very few contemporary images of the Datawidget are known to exist. A sketched representation of a Datawidget appeared in an advert on page 3 of a 1980 issue of [The California Tech](#), as shown in Figure 7.

THE TOOLS OF TOMORROW . . .



The DATAWIDGET™ is a self-contained hardware/software debugging aid, created for designers who take their microprocessors seriously. Its elegantly simple design lets you concentrate on your circuit without complicating your life. A concise, powerful Debug Monitor helps you pinpoint your hardware or software problem quickly, but without adding quirks of its own.

Datawidgets are now available in two models (6502 and 6800/6802), and more are being developed.

The MX CROSS-ASSEMBLERS, introduced in 1976, offer a level of convenience and power seldom found in the microcomputer world.

Through extensive usage and simulation, these assemblers have been optimized for quick execution, yielding short turn-around times.

They are now available for ten different microprocessor families (1802, 6502, 6800, 6805, 8048, 8080, Z-80, F8, PIC1650, COP444), in versions that run under RT-11 or TOPS-10 operating systems.

. . . . Are The Tools of Today

You can learn about microprocessors, and these "tools of tomorrow," in Caltech's three-term Microprocessor series: CS 112, CS 114, CS 121. First class is Wednesday, October 1. See the Registrar today!

Figure 7: APh Datawidget Advert from [The California Tech September 25th 1980](#)

It is interesting to note that this advert does not identify the CP-1600 family of microprocessors as a target for the Widget. However, it does state that it could be used with both RT-11 and TOPS-10, confirming that it was compatible with both the PDP-11 and DECSYSTEM-10 / DECSYSTEM-20.

Half of a Datawidget can also be seen to the right of an image of Hal Finney working at APh, which was [posted on Twitter](#) in 2020 and is reproduced in Figure 8.



Figure 8: Hal Finney Working at APh

In 2020 Shal Farley provided some images of a Datawidget, as shown in Figure 9 and Figure 10. These images confirm previous descriptions of the unit by Blue Sky Rangers. For example, at the CGE 2K4 panel [A2] Bill Fisher and John Sohl describe the Widget as a white box with blue sides and toggle switches along the front, and in Intellivisionaries episode 26 Rick Koenig and Ray Kaestner describe it as being a blue box.



Figure 9: APh 6502 Datawidget Front Panel

This Datawidget was targeted at 6502 development rather than the CP-1600 of the Intellivision. Although such 6502 Datawidgets were used for Intellivision Keyboard Component development by APh, this unit is believed to have been intended for Atari 2600 work. A sticker on the front of the unit indicates that it is the property of Mattel Electronics, although it seems Mattel never took possession of it.

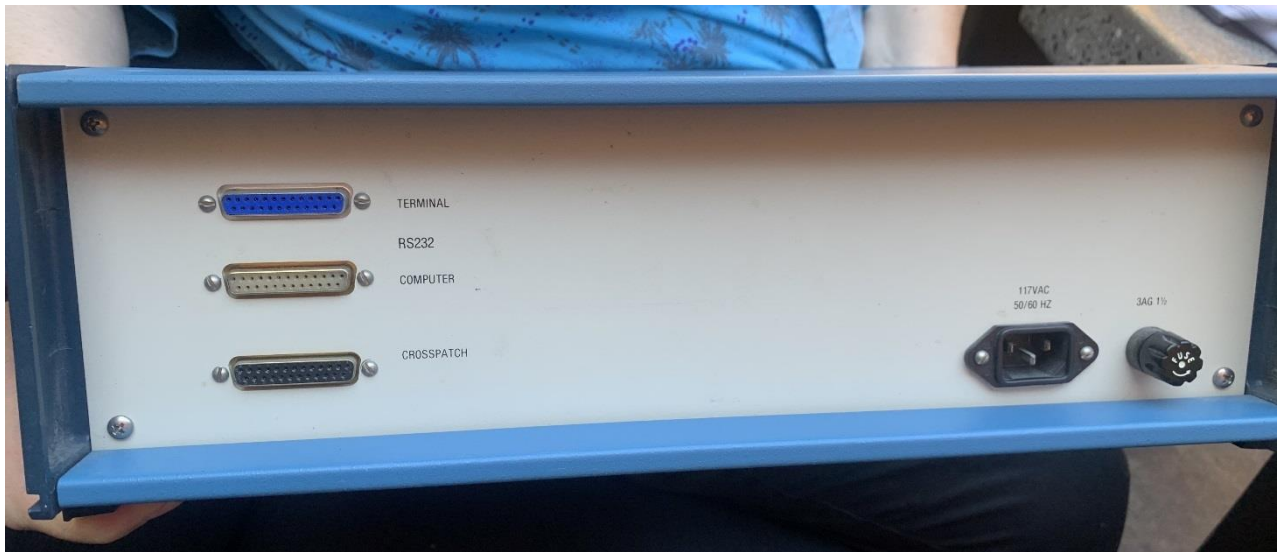


Figure 10: APh 6502 Datawidget Rear Panel

David Rolfe describes the Widget as a serial device that sat between the developer’s terminal and the host machine. This description has subsequently been confirmed by a copy of the [APh Datawidget User Guide](#) discovered in the Blue Sky Ranger’s document archive.

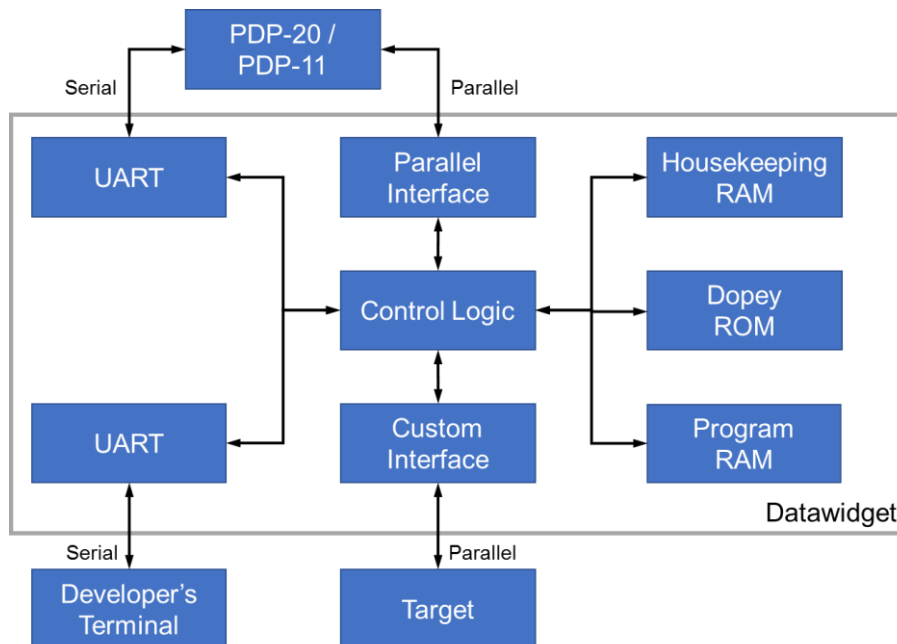


Figure 11: APh Datawidget Architecture

The architecture of the Datawidget is summarised in Figure 11. From its position between the developer’s terminal and the host machine the Widget controlled communication between the computer and terminal. Using control codes the developer could communicate with the Widget, or direct commands to the computer. In a similar way, tools run on the host machine to download programs to the Widget could control whether this communication was also forwarded to the developer’s terminal.

The Datawidget contained serial interfaces to communicate with the host PDP-11 and developer's terminal. It communicated with the target system through a parallel port connector on the front of the unit, and contained an additional parallel interface which was required when the PX1600 Crosspatch symbolic debugger was used on the host machine.

Looking inside the 6502 Datawidget, it can be seen to be a semi-custom design, as shown in Figure 12. In this image the Widget's front panel is at the bottom and a power supply is located to the left of the case. A motherboard occupies the right two thirds of the case. The right half of this motherboard is a regular PCB containing 8Kx8-bit RAM, 3Kx8-bit EPROM, two MC6850 UARTS to handle communication with the developer's terminal and host computer, and a single MC6821 PIA to manage parallel communication with the host.

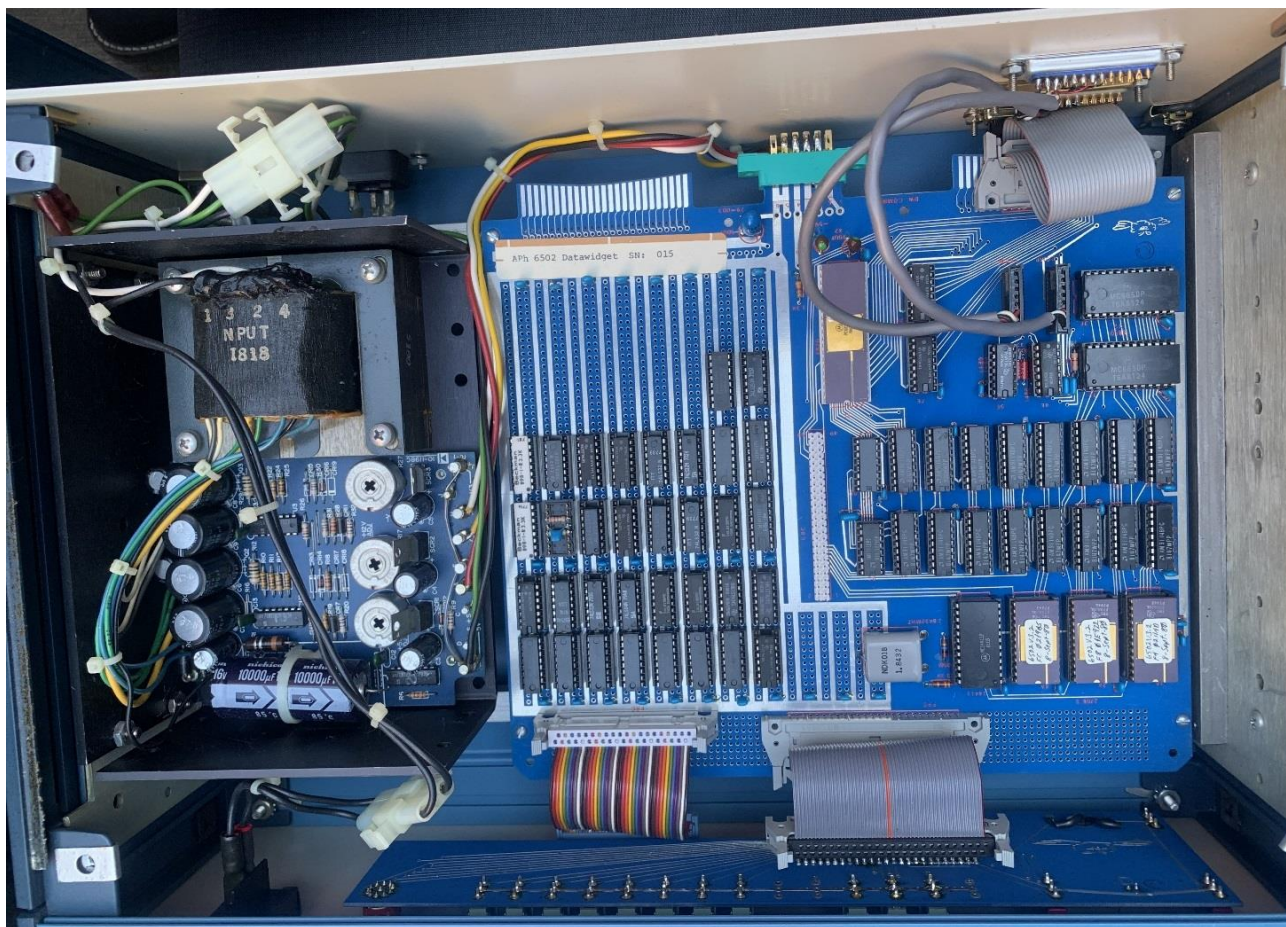


Figure 12: APH 6502 Datawidget Internal

The left-hand side of the motherboard, pictured in the centre of the case, contains a custom section, identified by the regular vertical silver tracks in the image. This part of the Widget uses 7400 logic to interface the standard part of the Datawidget with the particular target system. It contains the glue logic that maps the Widget's common components into the target system's memory map. The incoming target system address and data buses are connected through the rainbow ribbon cable at the bottom of the image and grey parallel cable on the front panel of the case. Shal indicates that this custom section was hand manufactured using wire-wrap techniques. This semi-custom motherboard design represents a later design iteration, with early Datawidgets being implemented completely in wire-wrap.

The CP-1610 version of the Datawidget used for Master Component development, see Figure 13, has the same architecture as the 6502 version shown above. However, it differed in a number of important areas.

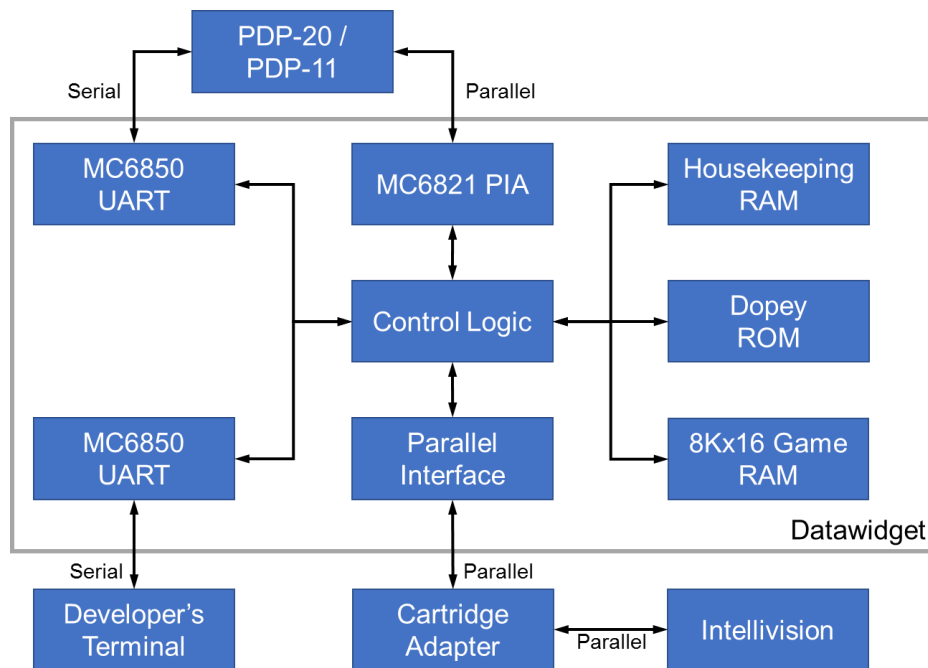


Figure 13: APh CP-1610 Datawidget

As the CP-1610 is a 16-bit CPU the corresponding Widget required 16-bit program and housekeeping RAM and Dopey was stored on two parallel banks of 8-bit EPROMs. This means that it was not compatible with the standard section of the motherboard seen above, and Shal believes that it is likely that all CP-1610 Datawidgets were fully custom, wire-wrap models. The CP-1610 Widget also provided the ability to switch its program memory between 10-bits and 16-bits wide, a feature not seen on the 8-bit variants. The Widget was attached to the Intellivision through a cartridge adapter plugged into the Master Component cartridge slot. This was unusual for Datawidgets, which normally connected to the target system using a clip that attached to the CPU. However, despite this more robust means of connection Shal suggests that this target system connection was a weak point in the design and getting reliable signals into the Widget could be troublesome. He also stated that this problem was quite common when using other in-circuit debuggers of the time.

The basic CP-1610 Widget configuration contained 8K of 16-bit RAM for storing test programs, however, owing to the way in which this memory was aligned to 8K page boundaries, only 4K of this could be used for Intellivision development. The block select knob on the left of the front panel allowed the 8K of RAM to start at one of \$0000, \$2000, \$4000, \$6000, etc. To get RAM to reside at the Intellivision cartridge starting address of \$5000, as required by the EXEC, necessitated a Datawidget start address of \$4000. The Datawidget setup then advised disabling the first 4K of RAM between \$4000 and \$5000 by flipping the left four switches in the centre of the front panel down, as described in item 8 on page 41 of the Datawidget User Guide. This restriction is also noted on pages 3 and 5 of the [APh Intellivision memory map description](#), reproduced in Figure 14, with “normal” Widgets only capable of supporting development of 4K games.

Development configurations

1) Oversize Cartridge Development Configuration - 8K, 12K, 16K with Ram

5000-5FFF	in normal widget	10/16 bit	4K
6000-6FFF	in extension	10/16 bit	4K
C000-DFFF	in extension	10/16 bit	4K
4400-4FFF	in extension overflow	10/16 bit	3K
4000-43FF	in extension <u>RAM</u>	8/16 bit	1K

2) Pseudo Cassette Development

8000-8FFF	in extension	10/16 bit	16K
widget will override gram/grom!			

3) Huge single chunk

C000-FFFF	in extension	10/16 bit	16K
-----------	--------------	-----------	-----

4) Alternate oversize Development Configuration

Normal widget wherever desired

4000-43FF	in Extension - <u>RAM</u>	8/16 bit	1K
4400-7FFF	in Extension	10/16 bit	15K

Figure 14: APh Datawidget Memory Configurations

The same document also hints at the existence of a Widget “extension”, used for larger cartridges, or those requiring additional RAM. From this description, an extended Widget would seem to have housed an additional 12K of 16-bit RAM. This enabled the extended Widget to be configured to mimic up to 16K of 8, 10 or 16-bit ROM and up to 1K of 8 or 16-bit RAM. There would also seem to have been some flexibility around the placement of game memory within the Intellivision’s memory map, with at least 7 configurations being described. The nature of this extension and whether it was an APh only enhancement is unknown. It is known that Mattel identified a workaround for debugging games of up to 8K in size using a standard Widget. This was achieved by relocating game code from \$6000 to \$4000 and making use of the first 4K of Datawidget RAM. This, along with various constraints associated with the technique, was documented in [an internal memo](#) by David Stifel in October 1982. Further, there are strong suggestions that an early 5K prototype of Space Spartans written by Brian Dougherty and [shared by Steve Roney](#) in 2020 was [configured in this way](#) for debugging on the Datawidget.

In addition to the RAM used to store the game code under test, the Widget claimed 8K of memory for its firmware, peripherals and internal housekeeping. This memory was placed between \$E000 and \$FFFF in the Intellivision’s memory map. This 8K block is known to contain a number of things, as detailed in Figure 15.

Address	Purpose
\$4000-\$4FFF	4Kx16-bit Program RAM, typically disabled
\$5000-\$5FFF	4Kx16-bit Program RAM
\$E200	Single step interrupt trigger (Datawidget Users Guide p36)
\$E410-\$E411	Terminal MC6850 UART (Datawidget Users Guide p37)
\$E440-\$E441	Host computer MC6850 UART (Datawidget Users Guide p37)
\$E600-\$E6FF	256 words of housekeeping RAM (Datawidget Users Guide p37) \$E6C0-\$E6FF – Reset stack
\$E700-\$E7FF	256 words of housekeeping RAM (Your Friend the EXEC p54)
\$F000-\$F7FF	Possible Crosspatch EPROM (Your Friend the EXEC p54)
\$F800-\$FFFF	Dopey EPROM (Your Friend the EXEC p54) The following entry points are identified on p39 of the Datawidget User Guide: \$FFE0 – EXIT program, enter Dopey entry point \$FFE2 – SUSPEND program, enter Dopey \$FFE4 – INLMES – write string to developer terminal \$FFE6 – TTYIN – Read 7bit character from terminal \$FFE8 – TTYIN8 – Read 8bit character from terminal \$FFEA – TTYOUT – Write a character to the terminal \$FFEC – DIGOUT – Write a hex character to the terminal \$FFEE – BYTEOUT – Write two hex characters to the terminal \$FFF0 – HEXOUT – Write four hex characters to the terminal

Figure 15: APh Datawidget Memory Map

The firmware within the Widget contained a diagnostic ROM, named Dopey, which is mentioned in the BSR informal video chat [V1] and the CGE 2K4 panel discussion [A2]. Dopey was initially designed to be very simple and easy to implement, hence its name. It is important to note that because the Datawidget did not contain its own CPU, Dopey would be written for, and run on, the CPU of the target system. Further, because the Widget's peripherals might have to be mapped to different addresses in particular target platforms, Dopey was likely to have to be tweaked when deploying to a new product. Dopey's simplicity allowed APh to target their development tools to a new platform quickly and cheaply. David Rolfe suggests a Dopey for a new system could be written in a few hours.

According to David Rolfe, when running on the Intellivision's CP-1610 CPU, Dopey would listen for and then execute one of four very simple commands:

- Poke a memory address with a value
- Peek the value at a specified address and report it to the host
- Insert a breakpoint at a specified address
- Execute code from a specified address until a breakpoint is hit

It seems this minimal framework was flexible enough to enable the construction of sophisticated interactive debugging capabilities between the host and Intellivision. Dopey permits the developer to upload a program to the game RAM at 9600 baud using a binary file format that was transmitted by running the program LOAD on the developer's terminal. The Datawidget can then protect memory ranges from an errant program, preventing it from overwriting itself. Once uploaded, the developer can view and change memory contents. It is also possible to view and change the contents of each of the CPU's registers. The developer can set, review and remove

breakpoints, which will halt a program if the execution reaches predetermined addresses. Dopey's breakpoints are implemented by recording the program instructions at the required address and replacing them with a jump into Dopey's code. This process is described briefly by David in Intellivisionaries Episode 7. Once game code execution has been started, it would continue until a breakpoint was reached. When this occurred, the machine state was evaluated and reported to the host. At this point, Dopey would regain control of the Intellivision and wait for the next command. From here the developer could single step instructions, observing the results on the CPU's registers as described by John Sohl in the CGE 2K7 panel [V5]. In addition to running the program from a known address, it was also possible to run a specific subroutine to its completion. During PRGE 2014 [V8] Keith Robinson briefly described the same breakpoint and single step features available to APH. Shal Farley explains that Dopey was inspired by the DEC [Online Debugging Tool \(ODT\)](#) and takes its command syntax from it.

Dopey also contained a kernel for the Crosspatch symbolic debugger, which probably explains why it claimed 8K of memory, despite its relative simplicity. Crosspatch will have allowed a developer to follow program execution in the source code displayed on their terminal. This is a very powerful debugging feature. To achieve this required the use of a parallel interface linking the Datawidget to the host machine. In the CGE 2K7 panel [V5] John Sohl suggests that the debugging tools in use by APH were superior to those sold to Mattel. During CGE 2K4 [A2] he seems to imply that he is not referring to Dopey, which Mattel did have use of, but a more advanced APH debugging system which he believes was called Merlin. It is plausible that John was referring to Crosspatch, the symbolic debugger, which he could have seen in operation when he was working on Astrosmash at APH's offices. The binary that provided the server side of Crosspatch was named PX1600. Again, Shal Farley indicates that Crosspatch was inspired by DEC's [Dynamic Debugging Technique \(DDT\)](#) which extended ODT and enabled symbolic debugging on the DECSYSTEM-10 and DECSYSTEM-20 mainframes.

Within Appendix A of Your Friend the EXEC a memory map for the Master Component is provided. This details the section from \$E000-\$FFFF as shown in Figure 16. Interestingly this suggests that at least two variants of the CP-1610 Datawidget were produced, distinguished by their serial number. These seem to have differed in both their memory map, and therefore, Dopey.

E000-FFFF		&160000	DATAWIDGET (when attached)
S/N 0-99	S/N 100+		
E000-EFFF			[reserved]
F000-F5FF	E000-E5FF		Read/Write Sensitive Locations
F600-F7FF	E600-E7FF		RAM (for kernal or Dopey)
	E800-EFFF		[reserved]
F800-FBFF	F000-F7FF		Expansion ROM
FC00-FFFF	F800-FFFF		Kernal

Figure 16: Your Friend the EXEC Memory Map

3.4 Tight Finances

In Intellivisionaries episode 17 Kimo Yap has described how APH made use of a payphone, rather than a regular business line in order to obtain unlimited fixed price local calls. This would have dramatically cut the cost of accessing local third-party servers.

4 Mattel Electronics

The canonical Intellivision development systems most frequently described by the BSRs, are those used at Mattel Electronics. According to John Sohl in the BSR informal video chat [V2] these systems started to be installed at Mattel in January 1981. All development prior to this, including his work on Astrosmash, was undertaken at APH's offices in Pasadena.

4.1 Host Hardware and Operating System

Most engineers describe the system that hosted Mattel development as being a [DEC PDP-11](#). In Intellivisionaries episode 31 Mark Urbaniec identifies it as a PDP-11/45 or PDP-11/70. In Keith's description documented in Retro magazine, he refers to the host machines at various points as being PDP-11s, TSX-1 and TSX-2, and finally Andromedas. The moniker TSX-1 and TSX-2 appear to have been the names of specific PDP-11 machines. Where the TSX prefix seems to refer to the [TSX-Plus operating system](#). This was a multi-user extension, written by S&H Computer Systems, to the single user [RT-11 real time operating system](#), created by DEC and which was widely used on the PDP-11. Mike Winans confirms the use of TSX in episode 29 of the Intellivisionaries podcast, although he suggests that the machines were PDP-8s, rather than PDP-11s. However, this is unlikely as TSX-Plus was not released for the earlier PDP-8.

Returning to Keith's description, the use of the term Andromeda is a reference to [Andromeda Systems INC](#), a [DEC supplier](#) from Canoga Park CA about 30 miles from Mattel in Hawthorn. Keith Robinson confirms that Mattel used the cheaper Andromeda PDP-11 clones, like those pictured in Figure 17, rather than DEC branded hardware in the BSR informal video chat [V3]. In episode 7 of Intellivisionaries Eddie Dombrower, describes the PDP-11 as having 8" disks with a capacity of approximately 1MB. This is confirmed by Dave Akers [in a comment on Atari Age](#).

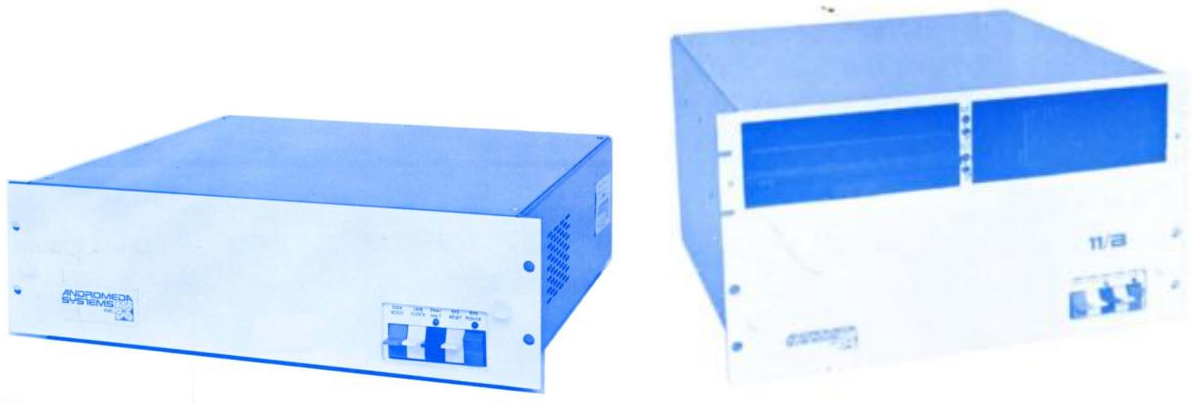


Figure 17: Andromeda Systems Inc 11/A23 and 11/B Machines

In the CGE 2K7 panel video [V6] Keith Robinson and David Warhol discuss the difficulties of retrieving game source code from these floppy disks at the outset of work at INTV. It would seem that the 8" drives used Andromeda controller cards that employed a proprietary format when used in double density mode. Page 12 of "[A Source Handbook for Digital Equipment Corporation LSI-11 Compatible Products](#)" describes 2 such controllers, the FDC11-A and FDC11-B. These drove the Shugart SA-800-2 and SA-850 single and double sided 8" double density drives respectively. It also notes that the FDC11-B / SA-850 combination is only generally compatible

when used in single sided mode. Page 2 of the [Shugart SA-850 manual](#) indicates that in double density mode this drive is capable of storing 1MB once formatted.

No other mention has been found of other storage media, such as hard disks, tape drives or cartridge systems. Therefore, it is not clear if floppy disks were the primary storage on the machines or not. It is also unknown if the use of TSX-Plus would have been practical without a hard drive.

At PRGE 2014 [V7] Keith Robinson states that later Mattel host machines were [DEC VAXs](#) running UNIX, rather than PDP-11s. Keith explains at CGE 2010 [V13] that two VAXs, named Ken and Barbie, were used to support new engineers, as development scaled up in late 1982 and early 1983. He also suggests that these machines were official DEC products, rather than clones like the Andromeda PDP-11s. Steve Roney has subsequently [confirmed that the VAXs were introduced in mid/late 1982](#) and also linked the introduction of the Blue Whale test harnesses to them.

Therefore, it seems very likely the host systems for Intellivision development at Mattel up until late 1982 were PDP-11 clones which probably ran TSX-Plus atop RT-11. After this, developers transitioned to using Unix running on DEC VAXs during the final year of Mattel Electronics.

Finally, some [Mattel Technical Bulletins](#) have been found within the Blue Sky Rangers document archive that suggest an HP-1000 machine was used to [capture and process speech data for Intellivoice games](#). No further information is known about the capabilities of these machines.

4.2 Development Tools

4.2.1 CP-1610 Assembler

As noted above, Mattel made use of the APh development tool chain, including the CP-1610 cross assembler when writing code on the PDP-11. However, the [commissioning of the VAX hosts](#) in early 1983 saw the introduction of a toolchain based on Nuvatec assemblers. [Nuvatec](#) were a product development firm based in Illinois who supplied the 6502 and Z80 cross-assemblers that were used on the VAX. They also contributed to the development of the Colecovision and wrote a number of early games for the system. Interestingly the Nuvatec assembler CP-1610 syntax was different to the APh toolchain, requiring the use of tools to translate source code.

4.2.2 Text Editor

In Intellivisionaries Episode 29 Mike Winans describes the use of the [TECO editor](#) and macro language. TECO has a long history dating back to the PDP-1 and is a direct ancestor of the desktop environment (and sometime text editor) Emacs that was used by APh.

4.2.3 Source Code Management

It is interesting that no mention is made of source code management tools. Although applications like SCCS were written in the early 1970s and were ported to UNIX on the PDP-11, it does not appear that a version control system was available for RT-11. Indeed, in Intellivisionaries episode 28 Steve Roney and Bill Fisher discuss the partitioning of source code in order to work collaboratively on Space Spartans. Similarly, at PRGE 2018 Ray Kaestner explains how the development of He-Man was partitioned with the Wind Raider and fireball screens being

completed by Rick Koenig and himself respectively [V15]. Again, this segmented approach significantly reduced the need for formal source code management.

4.2.4 Graphics Development System

Mattel created a couple of tools to specifically help with the construction of graphics. The first, the Intellivision Graphics Development System is [briefly described by Dave Akers](#). This was a workstation and piece of hardware used to construct Intellivision screens and is shown in an image provided by David James, reproduced in Figure 18.



Figure 18: Mattel Graphics Development System (image courtesy of David James)

The system's control panel consisted of an 20x12 grid of buttons on the left used to position cards on the BACKTAB. To the right of these a second grid of 8x16 buttons was used to define background cards and MOBS. To the far right a column of 16 buttons allowed selection of colours. The purpose of an additional row of buttons above the BACKTAB is not known. The resulting screens were shown on a monitor above the control panel. A terminal and PDP-11 compatible computer sat on a pedestal to the left of the control panel. No Intellivision console is visible in the image, and it is not known whether the control panel interacted with the mini-computer; or how the image presented on the screen was generated.

4.2.5 Mr Color

Access to the Graphics Development System was limited and it was superseded by a native Intellivision application called Mr Color. According to Russ Haft in Intellivisionaries episode 19, Mr Color was written by Eric Wells, who did much of the early graphics work at Mattel. In episode 16 Connie Goldman describes how graphics designers used the Intellivision controllers to move around the screen, toggling the state of pixels to construct an animation or background image. Importantly, as described in a [Mattel Technical Bulletin](#), Mr Color provided a means to dump and restore the state of work, allowing projects to be conducted over several sessions. This was done using the programs dcolor and ucolor that ran on the PDP-11 or [VAX](#) host machine.

Keith Robinson suggests in Intellivisionaries episode 9 that it is possible to get a feeling for the features of Mr Color, as its core is used to drive the graphics development within the ECS title Game Factory. Although not released by Mattel, this can be found on the Intellivision Rocks compilation.

4.2.6 Dr Color

An updated version of Mr Color, [named Dr Color](#), was written for use on both the PDP-11 and VAX hosts. Whilst it is not known how this version differed from Mr Color, a document describing feature requests, suggest that key enhancements may have been.

- Addition of support for the STIC's Color Stack mode.
- Greater flexibility over the use of GRAM increasing the number of background tiles that could be defined and allowing background sequencing.
- Scrolling of pixels within GRAM cards.
- Linking of multiple MOBs.
- Extension of MOB animation sequences to more than 16 frames.

4.2.7 Mr Sound

Inspired by the success of Mr Color, Daniel Bass and Andy Sells [developed Mr Sound](#), an audio equivalent. Mr Sound was an audio editor that ran on the Master Component, it allowed developers to interactively construct and test sound effects. It was designed to create effects that could be played back by the EXEC's built in sound tracker, as described in Supplement 1 of Your Friend the EXEC. The EXEC sound infrastructure was used to play the iconic Intellivision sounds, such as the crowd cheer. Like Mr Color, effects could be loaded and saved to the PDP-11 host machine, allowing effects to be developed across multiple sessions. Once the developer was happy with an effect, it was uploaded to the PDP-11 where it was converted into CP-1610 assembler source, ready for inclusion in game code.

Mr Sound allowed the developer to construct effects by viewing, inserting, modifying and deleting calls to the various SCODE macros used by the EXEC sound tracker. The parameters associated with each macro, used to choose channels, note periods etc., could also be changed. Any of up to 10 sound effects stored in the Master Component's memory could be previewed at any time and uploaded to the PDP-11 host. Two versions of Mr Sound were to be developed, judging by [Daniel Bass' 1983 performance review](#). The first for the Magus was due for delivery at the end of March 1983, the second compatible with the Keyboard Component test harness was to be delivered in mid-April. Like many of the development tools, although initially developed to run on the PDP-11, the programs to upload and download data to Mr Sound were [converted to](#)

[also run on the VAX](#) in 1983. Unfortunately, unlike Mr Color, Mr Sound did not make its way into Game Factory.

4.2.8 Sound T

Referenced in the instructions for Mr Sound (see above), Sound T appears to have been a precursor tool written by Bill Goodrich. No further information is known about its capabilities.

4.2.9 Bijan Jalali's Magus Debugger

Although the APH Widget discussed in Section 3.2.4 provided debugging facilities, these units seem to have been in short supply at Mattel [A2]. There is also strong evidence that the more common Magus test boards did not contain a Dopey style debugger, as will be discussed in Section 4.3.1. As a consequence, developers at Mattel found debugging time consuming and difficult. According to [memos sent in early 1982](#) Bijan Jalali seems to have been responsible for the development of a debugger for use with Magus. This appears to have been in [testing in late March](#) and was released in [late April / early May 1982](#). The debugger added the ability to inspect and change register values, set and execute to a breakpoint and single step instructions. The developer interacted with the debugger using their terminal, and it used similar commands to the Datawidget. In order to communicate with the debugger over the parallel connection to the Magus, the developer had to run the program DEBUG on their host machine. It is believed that this would capture debugger commands and translate them into requests placed in the memory shared by the host and Magus. A small 281 decl monitor program linked in with the code under test would then read and process requests, returning results to DEBUG via the shared memory. Like the Datawidget, breakpoint features seem to have been implemented using the replacement of program instructions in RAM.

Although the features of this debugger would seem to be a significant step forward, during testing [a memo from Don Daglow](#) notes a number of perceived shortcomings. Specifically, although small, linking the debugger prevented testing of games close to the 8K size limit of the Magus at that time. The fact that breakpoints were implemented with JMP instructions placed a constraint on their use in some circumstances, and finally synchronised access to timing sensitive STIC and GRAM locations was not supported.

The take-up of Bijan's debugger are not totally unclear. However, updates to the Magus download software do not mention its existence, in contrast to the later Blue Whale documentation; and in 1983 Rick Koenig wrote another debugger. The reasons that Bijan's debugger did not seem to gain widespread adoption are not known.

4.2.10 Rick Koenig's Native Debugger

In the two to three weeks between working on Motocross and Masters of the Universe, Rick Koenig wrote a native Intellivision debugger that could be linked with a game. Whether this work was independent of Bijan Jalali's earlier Magus debugger or built upon its foundations is not known. The key difference between Bijan's and Rick's work is that the later Koenig debugger dispensed with the use of the developer's terminal as a means of communicating with the debugger. Instead commands were entered on the Intellivision controllers and the results displayed on the game screen. In Intellivisionaries episode 26 Rick explains how the debugger was activated using the hand controller buttons and then enabled the usual tasks of state inspection, setting of break points and single stepping through code. Like Bijan's implementation

Rick’s debugger does not provide synchronised reading and writing to timing sensitive GRAM and STIC memory addresses. However, unlike all other debuggers in use at Mattel, Rick’s code had the capability to disassemble CP-1610 object code, if only one instruction at a time.

Another key difference between Rick’s debugger and other tools is the manner in which breakpoints were implemented. Rather than the debugger replacing instructions to set breakpoints during testing sessions, they had to be pre-compiled into the game code. The debugger could choose which breakpoints were active, but not their location in memory. This change will have addressed the problems associated with replacing instructions with JMPs, as used in Bijan’s debugger, and noted in Don Daglow’s memo.

JoeZ highlighted that the code for Rick’s debugger can still be found embedded in all World Championship Baseball ROMs. This is evidenced by the fact that a list of assembler mnemonics is visible if JzIntv is used to examine the memory starting at \$DA98, as shown in Figure 19.

```

> m da98 100
DA98: 0010* 022C 000A 02B0 02B4 02B1 02B7 0050 # .....P
DAA0: 004F 0050 0020 0050 0055 0053 0048 0043 # .O.P...P.U.S.H.C
DAA8: 004C 0052 0020 0054 0053 0054 0020 004E # .L.R...T.S.T...N
DAB0: 004F 0050 0020 0048 004C 0054 0020 0053 # .O.P...H.L.T...S
DAB8: 0044 0042 0044 0045 0049 0053 0020 0044 # .D.B.D.E.I.S...D
DAC0: 0049 0053 0020 004A 0055 004D 0050 0054 # .I.S...J.U.M.P.T
DAC8: 0043 0049 0020 0043 004C 0052 0043 0053 # .C.I...C.L.R.C.S
DAD0: 0045 0054 0043 0049 004E 0043 0020 0044 # .E.T.C.I.N.C...D
DAD8: 0045 0043 0020 0043 004F 004D 0020 004E # .E.C...C.O.M...N
DAE0: 0045 0047 0020 0041 0044 0043 0020 0047 # .E.G...A.D.C...G
DAE8: 0053 0057 0044 0052 0053 0057 0044 0053 # .S.W.D.R.S.W.D.S
DAF0: 0057 0041 0050 0044 0053 0057 0050 0053 # .W.A.P.D.S.W.P.S
DAF8: 004C 004C 0020 0052 004C 0043 0020 0053 # .L.L...R.L.C...S
DB00: 004C 004C 0043 0053 004C 0052 0020 0053 # .L.L.C.S.L.R...S
DB08: 0041 0052 0020 0052 0052 0043 0020 0053 # .A.R...R.R.C...S
DB10: 0041 0052 0043 004D 004F 0056 0020 0041 # .A.R.C.M.O.V...A
DB18: 0044 0044 0020 0053 0055 0042 0020 0043 # .D.D...S.U.B...C
DB20: 004D 0050 0020 0041 004E 0044 0020 0058 # .M.P...A.N.D...X
DB28: 004F 0052 0020 0042 0052 0041 0020 0042 # .O.R...B.R.A...B
DB30: 0043 0053 0020 0042 0056 0053 0020 0042 # .C.S...B.V.S...B
DB38: 0050 004C 0020 0042 0045 0051 0020 0042 # .P.L...B.E.Q...B
DB40: 004C 0054 0020 0042 004C 0045 0020 0042 # .L.T...B.L.E...B
DB48: 0055 0053 0043 0052 0045 0054 004E 0042 # .U.S.C.R.E.T.N.B
DB50: 0043 0043 0020 0042 0056 0043 0020 0042 # .C.C...B.V.C...B
DB58: 004D 0049 0020 0042 004E 0045 0020 0042 # .M.I...B.N.E...B
DB60: 0047 0045 0020 0042 0047 0054 0020 0042 # .G.E...B.G.T...B
DB68: 0045 0053 0043 0053 0049 004E 0020 004A # .E.S.C.S.I.N...J
DB70: 0020 0020 0020 004A 0045 0020 0020 004A # .....J.E....J
DB78: 0044 0020 0020 004A 0053 0052 0020 004A # .D....J.S.R...J
DB80: 0053 0052 0045 004A 0053 0052 0044 0275 # .S.R.E.J.S.R.D.u
DB88: 0088 0040 0064 0064 03B8 000F 0004 01D8 # .....d.d.....
DB90: 03A7 0088 0040 03B8 000F 0004 01D8 03A7 # .....
> █
    
```

Figure 19: Rick Koenig’s Debugger Mnemonic List within World Championship Baseball

Using the code within World Championship Baseball, Rick Koenig’s debugger was isolated, reverse engineered and documented in 2019. A full write up of the results, including a ROM that demonstrates the use of the debugger, can be found on [Atari Age](#). An example of the output shown on the main debugger screen is illustrated in Figure 20.

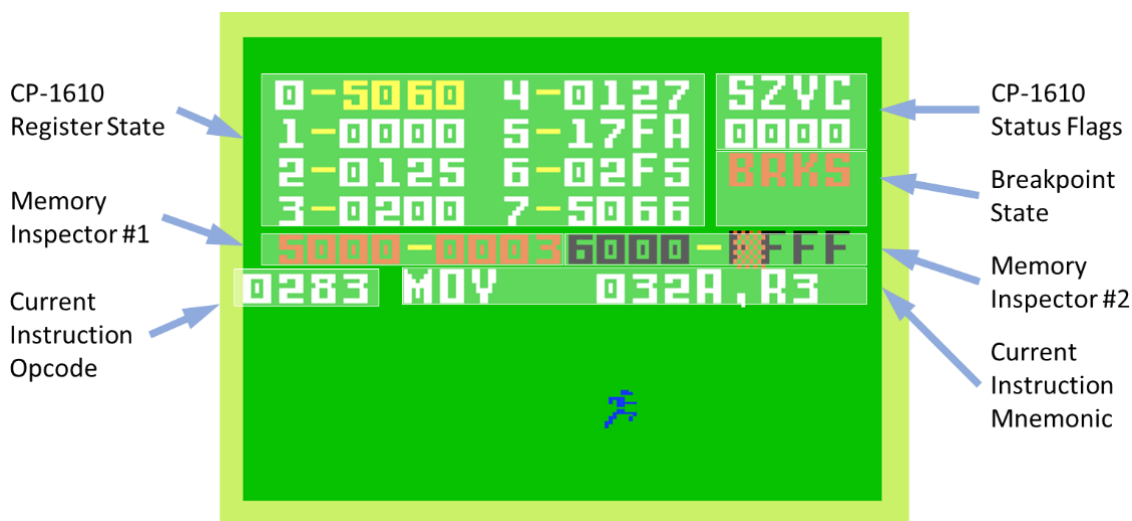


Figure 20: Rick Koenig's Debugger

4.3 Test Harnesses

As already noted, Mattel had access to a small number of APH Datawidgets; during the CGE 2K4 panel [A2] Bill Fisher explains that a single Widget was available as a shared resource. The limited number of Datawidgets available at Mattel was confirmed by both Gabriel Baum and David Warhol at PRGE 2018 [V14]. It also appears that Mattel viewed the design of the Widget as being marginal, leading to it being rather fragile. Mattel expressed concerns about releasing it to third parties, as detailed on [page 4 of a memo on PapalIntellivision](#). This, along with the short supply of devices, may have resulted in Mattel seeking other test harnesses for their use.

4.3.1 Magus / MAGUS / MEGAS / Megus

The majority of early Mattel testing was done on boards [named Magus](#), also designed and manufactured by APH. It would seem that the Magus was not simply a renaming of the APH Datawidget, but a rather different design. Although the Intellivision Lives site names this board Magus, other Mattel references appear to call it [MAGUS](#), [MEGAS](#) or Megus, suggesting the name may have been an acronym. For consistency this document will continue to use the name Magus. Glen Hightower has suggested that the Magus was originally requested by Mattel marketing as a means of demonstrating Intellivision Keyboard Component software. Like the Widget, relatively little public information is available on the Magus boards and no images are known to exist of it. On the BSR website they are described as being "hand built" although whether this refers to PCB component placement and soldering, or that a prototyping technology such as wire wrap was used, is not clear. The fact that one of Don Daglow's complaints regarding the use of the Magus was its "habitually exposed and entangled wiring" (see Figure 21) might also suggest the use of a prototyping manufacturing technique.

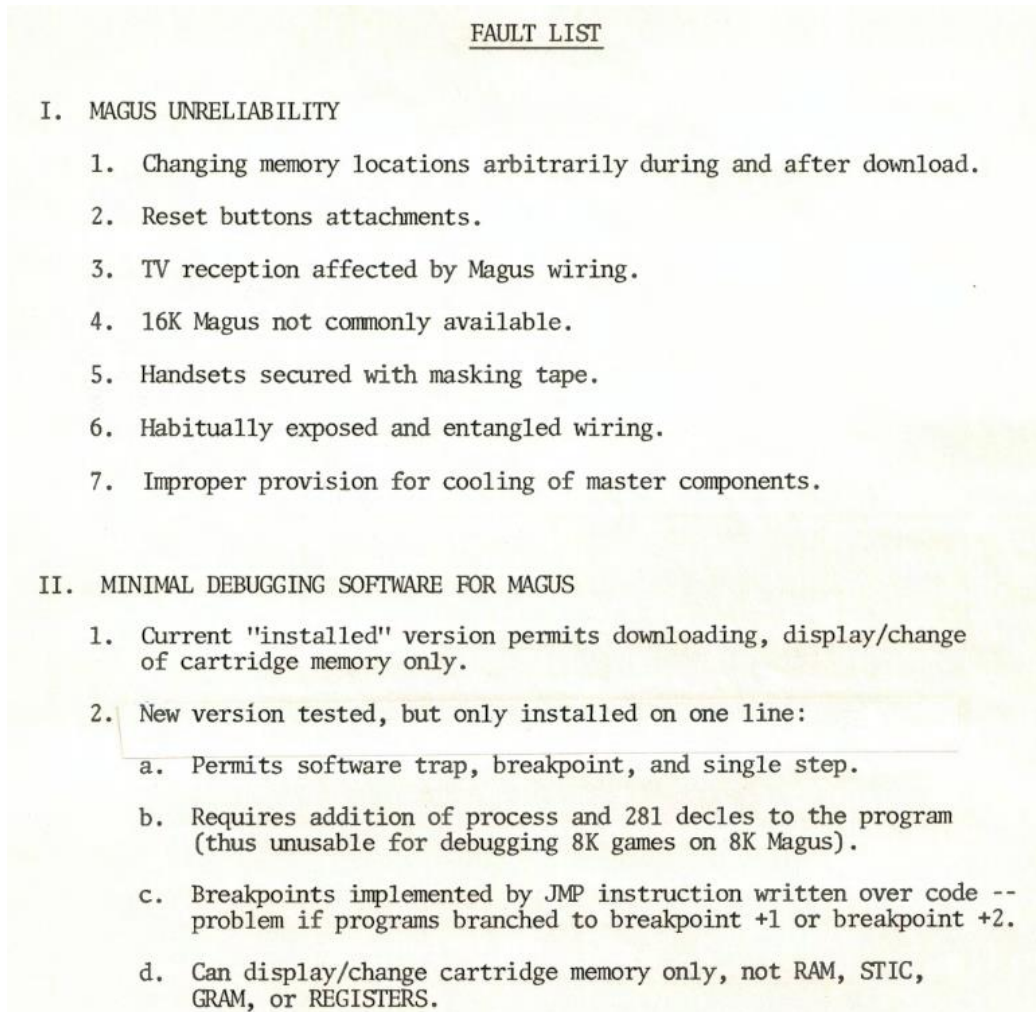


Figure 21: Magus Development Faults

It is known that the Magus was connected to the PDP-11 host using a parallel interface, and a [Technical Bulletin describing its use](#) suggests that this was terminated with a [DIO11 card](#) within the PDP-11, leading to an architecture similar to that shown in Figure 22.

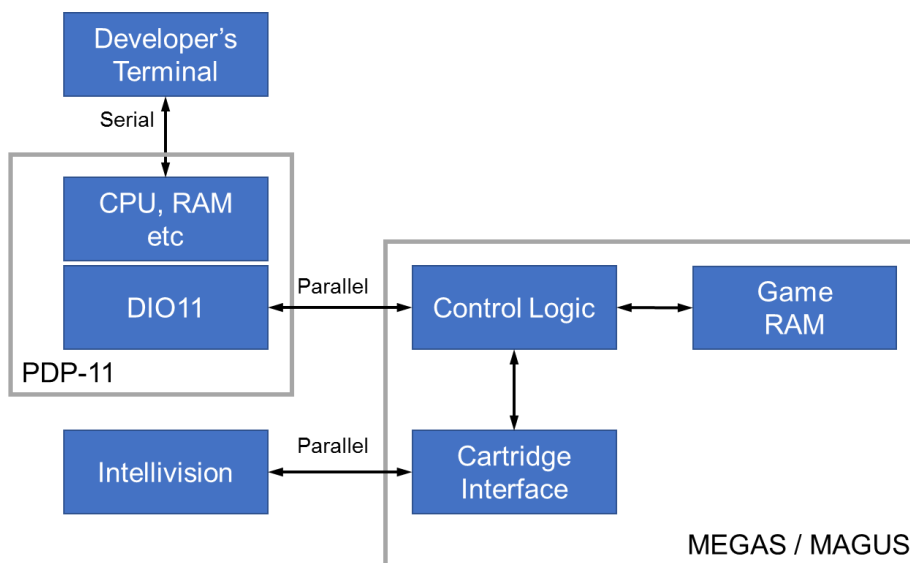


Figure 22: Mattel Magus Architecture

It is not clear how much, if any, of the Magus hardware was physically located in the PDP-11 rather than being on a card in the developer's cubicle. During the CGE 2010 panel [V13] David Warhol makes it clear that the Magus or "ROM emulator" was a two-piece design with cards located in a PDP-11 at the end of each row of cubicles. These cards were connected to an Intellivision in each developer's workstation by a long parallel cable. David suggests that each PDP-11 supported several Magus boards. This is confirmed in a [Technical Bulletin](#), which indicates that up to seven Magus harnesses could be connected to a single PDP-11 running TSX+. Initially the Magus seems to have contained up to 8K decles of RAM for game code. This would seem to have increased to 16K by Mach 1982, although a memo from the development team suggests the larger model was in short supply at this point (see Figure 21).

The same memo describes other issues with using the Magus, including its lack of reliability and limited debugging facilities. The new version of the Magus software identified in the memo is believed to be Bijan Jalali's debugger, detailed in Section 4.2.9, which was released in April of the same year.

Sometime before March 1983, the 16K Magus seems to have been superseded by a model with 20K decles of RAM mapped in two blocks from \$5000 - \$6FFF and from \$D000 - \$FFFF, and by mid-1983, [larger capacity Magus boards were introduced](#) capable of supporting games of 24K and 32K decles. In addition to the blocks identified above these supported the address range \$8000 - \$BFFF. The software used to download game code to the Magus was also updated at this time to improve error detection.

As already noted, John Sohl has suggested that the debugging tools used by APH were superior to those sold to Mattel. During the BSR informal video chat [V4] David Warhol goes further, stating that the initial work at Mattel had no debugger support. This view might be supported by Keith Robinson in the CGE 2K4 panel [A2] when he contrasts the methodical Widget based approach to debugging, with that associated with more intuitive developers. Together, this suggests that initially the Magus was only a RAM cartridge and did not contain a debugger. Again, this is corroborated by the [Magus technical notes](#), which suggest that as late as summer 1983, other than downloading game code, the DLM program only supported inspecting and altering memory values. If this is correct, it would have made testing significantly more challenging and explains why Bijan Jalali and Rick Koenig developed their debuggers (Sections 4.2.9 and 4.2.10).

4.3.2 Blue Whale

It would seem that production of the Magus interfaces became a bottleneck as Mattel scaled up development during 1982. In Intellivisionaries episode 16 David Warhol describes how on joining Mattel, although he had terminal access to develop software, he had to steal time on a colleague's test harness. It was realised that the cancelled "Blue Whale" Keyboard Component could be repurposed to act as the test harness in place of the Magus. Until recently, details of the resulting Blue Whale development harness were thin on the ground. It is not known who came up with the idea to use Keyboard Components as development environments, or who did the work to convert them. Although [Intellivision Lives states](#) that these development systems were known as "Black Whales", it should be noted that Steve Roney and the other BSRs [do not recognise the use of this term](#). This view is corroborated by [Daniel Bass' 1983 review](#).

In Intellivisionaries Episode 5, Keith Robinson describes the primary modification to the Keyboard Component as being the addition of a serial port to enable a VAX host to download games. This ties up with the distinction with the Magus' parallel connection, made on the Intellivision Lives Keyboard Component page.

Early in 2018 Daniel Bass, author of Loco-Motion, posted images of a pair of development boards he liberated from Mattel Electronics when it closed, as shown in Figure 23.

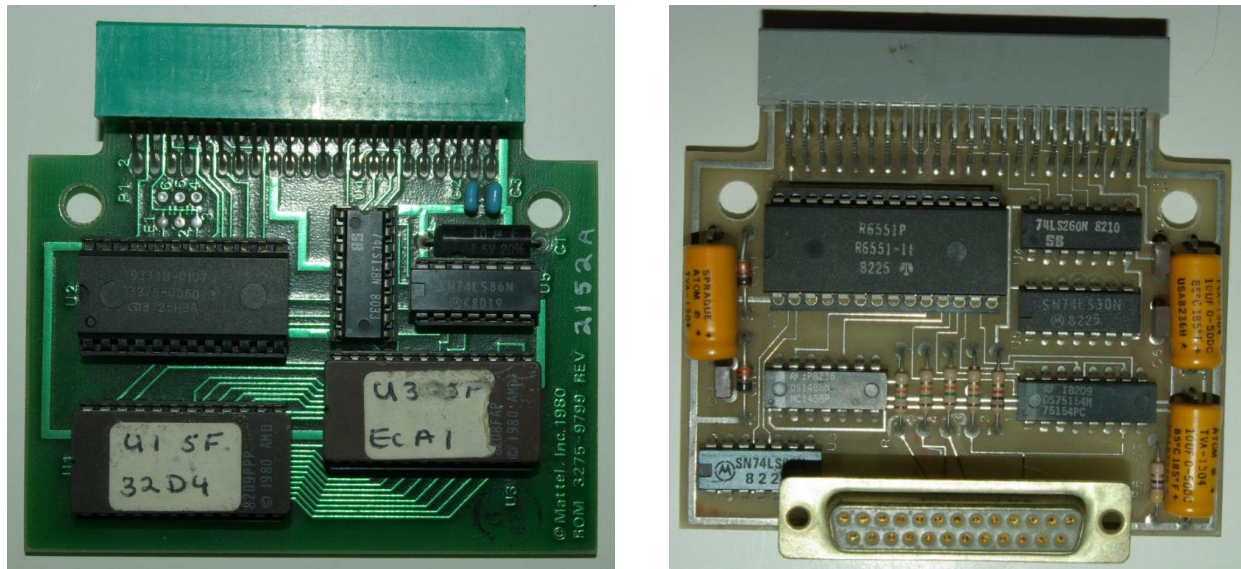


Figure 23: Mattel Blue Whale Development Kit Boards

These were identified as being a variation of the Keyboard Component Microsoft BASIC cartridge (left) and a serial cartridge with a Rockwell R6551 UART (right). Daniel very kindly lent both cartridges to JoeZ who did an extensive reverse engineering exercise on them, aided by FrankP. [The results of this work](#) are well worth taking the time to read. Joe's analysis suggests that the Blue Whale conversion probably consisted of three parts and a minor tweak to the Master Component. No change was required to the Keyboard Component itself.

The conclusions of Joe and Frank's work were confirmed in 2019 when a set of [Mattel Technical Bulletins](#) describing the introduction and evolution of the Blue Whale development system was found in the BSR document archive. The overall architecture of the development system is shown in Figure 24 .

The developer's terminal [sits between the host computer and Blue Whale test harness](#), connected to both by RS-232 serial interfaces. The need for two serial connections limited the choice of terminals that could be used and it seems that Mattel standardised on the use of the CIT-101 with an optional auxiliary serial port fitted. Therefore, the picture of Patrick Aubry's workspace would seem to be typical of a Blue Whale development setup. This architecture means that inputs, in terms of keystrokes; and output from the host or Blue Whale, must be routed by the terminal to one or more of the terminal screen, the host computer and the Blue Whale. This routing could be done manually from the terminal keyboard or automatically using control codes embedded in escape sequences.

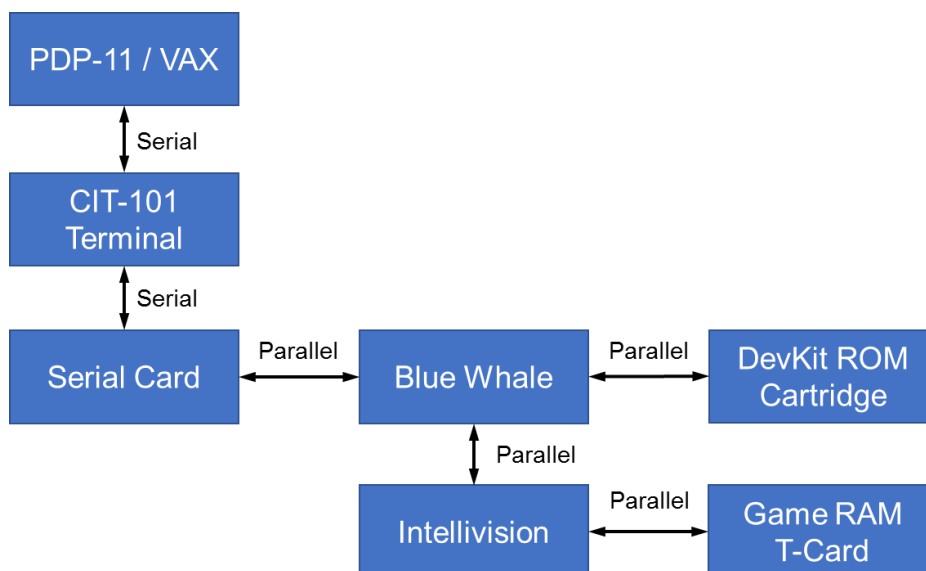


Figure 24: Blue Whale Development System Architecture

The CIT-101 was connected to the Blue Whale using the development kit serial board, which plugged into one of the Keyboard Component's 6502 cartridge ports. The ROM board plugged into the second 6502 cartridge port and contains two pieces of software. There is no technical reason for this hardware to be split across two boards. Presumably Mattel made use of surplus BASIC cartridge boards to save costs, rather than build a single integrated solution.

The final component in the development kit was a CP-1610 T-Card loaded with 8K decles of RAM. This can be seen plugged into the side of the Keyboard Component being used by Patrick in Figure 25.

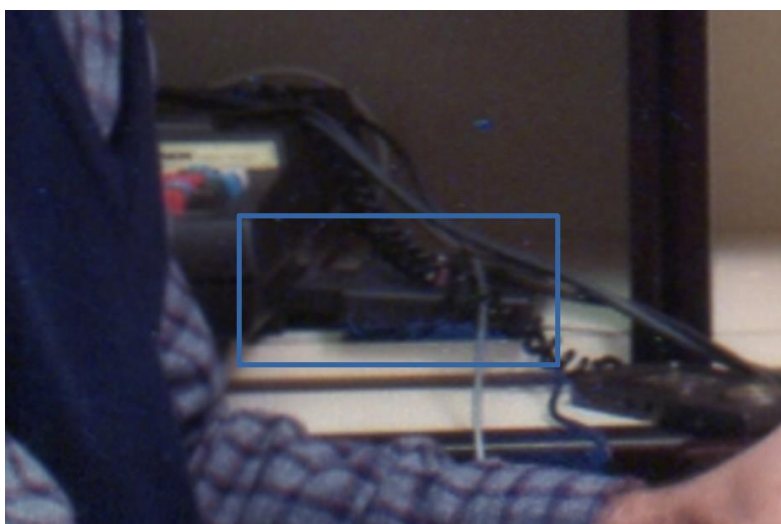


Figure 25: T-Card in use with a Blue Whale Development Kit Board

The development kit ROM board contains a 4K monitor program for the Keyboard Component 6502 processor. This manages communication between the host, Keyboard Component and Master Component using the serial board and shared Keyboard Component RAM. The ROM board also has 4K bytes (2K decles) worth of code that is loaded into the shared CP-1610 memory space by the Keyboard Component's 6502 processor during initialisation. This CP-1610 code

contains a monitor and debugger for the Master Component. Daniel's board identifies the software as version 5f of the 1610 Development System.

The [Intellivision Lives website](#) implies that within the Blue Whale some of the Keyboard Component shared RAM is moved to replicate the memory map of a standard Intellivision cartridge. However, there is no evidence that this is the case. Further, the way in which the 6502 monitor initialises the Master Component memory between \$5000-\$6FFF supports the fact that this RAM was not shared with the Keyboard Component, but instead was located on a T-Card. Adopting this approach of adding more RAM, rather than changing the addressing of shared memory would have simplified the conversion process.

The one small change to the Master Component was to link the !PCIT pin to the !INTR pin on the CP-1610. The implementation of breakpoints by the development kit relies on software interrupts. However, the CP-1610 SIN instructions that generate these just strobe the !PCIT pin, which is not connected to anything on a standard Intellivision. Linking !PCIT to !INTR would close this loop and cause the SIN instruction to raise an interrupt that could be caught by the development kit. Testing suggests that this change requires a small circuit to extend the !PCIT signal long enough for it to trigger an interrupt. The circuit used to do this is not known, however, functionally equivalent circuits are easy to construct.

From the developer's perspective the harness was controlled using their terminal session on the PDP-11 or VAX. They could download code, using one of a number of programs running on the host machine. Initially the program WHALE.SAV was used for this purpose, this was later superseded by ORCA.SAV on the PDP-11 and whale on the VAX; and finally nuwhale, which was supported on both VAX and PDP-11. The developer could use the Blue Whale to manage breakpoints, single step through instructions, inspect and alter both memory and registers from their terminal, much as we do today using JzIntv. It was not possible for the development system to disassemble memory contents or highlight the current execution point in the program source when single stepping, as a modern debugger does.

The [Mattel technical bulletins](#) identify Daniel Bass' development kit as dating from the first half of 1983. These updates also provide a partial history of the timing and features of the development kit software as it evolved. They also confirm that use of PDP-11 hosts with Blue Whale development kits started in October 1982 and continued throughout 1983, with VAX host support being added in January of 1983. This information is summarised in Figure 26.

Version	Approx. Date	Addition Features
1	Oct 82	Program upload and execution
2	After Oct 82	Memory examination and alteration Higher speed program transfers
3	Unknown	Unknown
4d	Before Jan 83	<i>The following features were present in version 4d. It is not known which were carried over from version 3:</i> Set, review, remove and run to breakpoints Single Step / Trace execution Move a block of memory from one address to another Generate memory checksum Search memory for a specific pattern Zero all program memory space
4d VAX	Jan 83	VAX support
5f	Jan 83	Register state examination and alteration Program interruption on ESC key 9600 baud communication speed
6g	Sep 83	Paged ROM support through setting active pages Support for non-standard interrupt service routines Change of interruption key from ESC to TAB for improved compatibility Improved communications reliability (may also increase speed)

Figure 26: Evolution of Blue Whale Development Kit Software

A replica of a Blue Whale development kit has been constructed based on the board designs and software provided by Daniel Bass. Details of its implementation and a video of it in operation [can be found here](#). It should be noted that it is now known that the architecture of the system presented in the video is incorrect, as it is centred on the host computer, rather than the developer's terminal.

Despite its relative sophistication David Warhol suggested that the Blue Whale was a step backward from the Magus harness in Intellivisionaries episode 16. This is primarily because the serial communication between the host and Blue Whale increased the time to download a game by a factor of 5 or more. As a consequence, iterative development cycles were significantly slower. Examining the protocol used to send game code to the Blue Whale JoeZ has established that the successful transfer of a 16K game would take 40-45 seconds over the 9600 baud link. It should be noted that the serial link between the host and Blue Whale does not make use of flow control. Therefore, it may have been necessary to throttle transfers, further increasing the download time, to prevent the host flooding the Blue Whale and corrupting the game. It is interesting that download speed is not a criticism levelled at the APh Datawidget, which also used a 9600 baud serial link for communication. Despite its use of a binary format to send data, it is difficult to see how transmitting a game to the Datawidget would have been significantly faster than the Blue Whale.

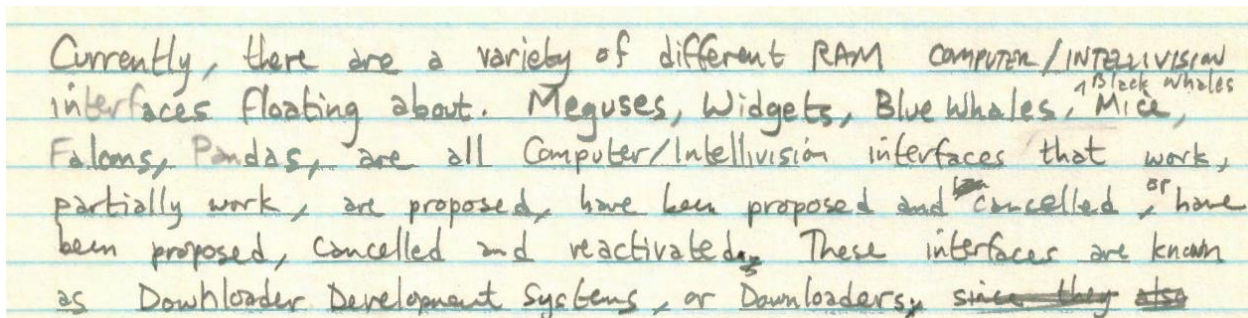
A fivefold slowdown suggests that a similar Magus download took less than 10 seconds. Despite this loss of performance, the switch to serial communication between the VAX / PDP-11 host and the Blue Whale can perhaps be explained by improved reliability and additional cost savings. Moving away from a proprietary parallel communication mechanism to the more standard RS-

232 interface may have simplified the host end of the link, making the development systems cheaper and potentially easing the migration to using VAXs for development work.

Overall, like the APH Datawidget, the Blue Whale seems to have been a powerful development tool that would have been a significant help when testing and debugging games.

4.3.3 Comparison of Mattel Harnesses

It is clear from the description above that there were several Intellivision development systems available to Mattel developers at different times, a fact highlighted by Keith Robinson in a draft update to Your Friend the EXEC, see Figure 27.



Currently, there are a variety of different RAM COMPUTER/INTELLIVISION interfaces floating about. Meguses, Widgets, Blue Whales, ^{15k} Mice, Falcons, Pandas, are all Computer/Intellivision interfaces that work, partially work, are proposed, have been proposed and cancelled, or have been proposed, cancelled and reactivated. These interfaces are known as Downloader Development Systems, or Downloaders, since they also

Figure 27: Mattel Testing Harnesses Listed by Keith Robinson

Keith goes on to list the advantages and disadvantages of each system as shown in Figure 28.

1. WIDGETS: The oldest downloader system.
ADVANTAGES: GOOD DEBUGGING TOOLS, GOOD DOCUMENTATION *
DISADVANTAGES: SERIAL INPUT SLOWS DOWNLOAD TIME, ~~CANT HANDLE PROGRAMS OVER 8K WITHOUT PUTTING SOME BURNING EFFORTS~~ LIMITED TO 8K PROGRAMS, ~~PROGRAMS OVER 4K MUST BE SPECIALLY LINKED IN~~ PROGRAMS OVER 4K MUST BE SPECIALLY HANDLED.
2. MEGUSES (The spelling of this varies Megus, Megas, Meguses, Megii, Vice President Gabriel "In Ital We Trust" Baum prefers Megus / Meguses, so we'll just stick with that): Downloader currently in widest use.
ADVANTAGES: 16K programs, Parallel input - fast download time, ^{Mr. Color}
DISADVANTAGES: Limited debugging tools (improvements have been promised) and documentation. Doesn't work with the VAX.
3. BLUE WHALES: Modified old keyboard components (we had to do something with them).
ADVANTAGES: Debugging tools close to that of Widgets (and still being improved), 16K programs. (?)
DISADVANTAGES: Serial input slows download time.

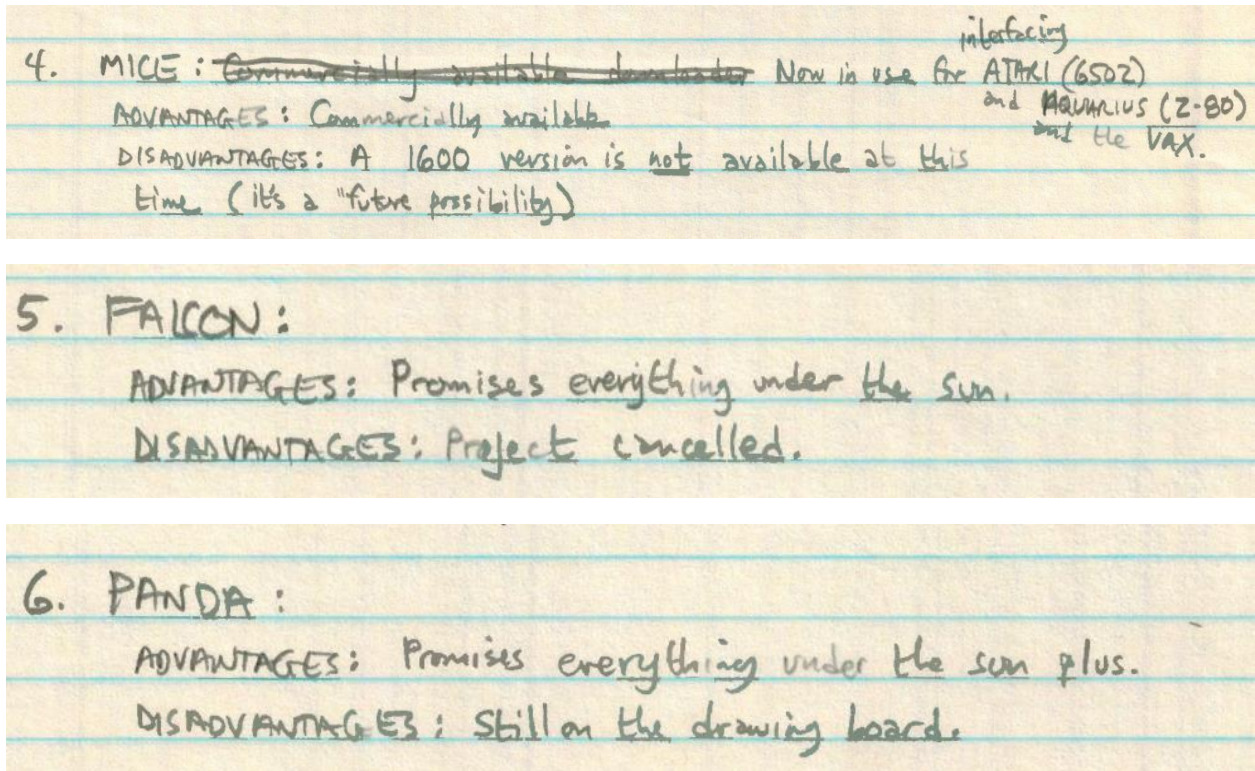


Figure 28: Mattel Testing Harnesses Pros and Cons by Keith Robinson

In addition to the Widget, Magus and Blue Whales identified above, Keith lists the [MicroTek MICE](#) used for Atari 2600 and Aquarius debugging; and two internal Mattel projects, the [Falcon](#) and Panda debuggers. Neither of these latter projects are believed to have made it to production.

The features and capabilities of both the debugging hardware and software used for Intellivision development at Mattel are summarised in Figure 29 and Figure 30.

	Datawidget	Magus	Blue Whale
Mattel introduction	Early 1981	Unknown	Late 1982
Availability	Limited	Good	Good
Connection type	Serial (9600 baud)	Parallel	Serial (9600 baud)
PDP-11 compatible	Y	Y	Y
VAX compatible	Y	.	Y
RAM sizes	8K, 24K (Aph only?)	8K, 16K, 24K, 32K	20K
Game sizes	4K typical 8K with special layout Up to 17K (Aph only?)	4K-32K	4K-20K
Compatible debuggers	Dopey	Bijan Jalali Rick Koenig	Blue Whale

Figure 29: Mattel Debugger Hardware Summary

	Dopey	Magus	Blue Whale	Bijan Jalali	Rick Koenig	JzIntv
Hardware						
Control device	Terminal	Terminal	Terminal	Terminal	Intellivision	Terminal
Compatible harnesses	Datawidget	Magus	Blue Whale	Magus	Magus Blue Whale	.
Register commands						
Show register contents	Y	.	Y	Y	Y	Y
Alter register contents	Y	.	Y	Y	Y	Y
Memory commands						
Inspect address	Y	Y	Y	Y	Y	Y
Alter data	Y	Y	Y	Y	Y	Y
Inspect block	Y	Y	Y	Y	.	Y
Copy block	.	.	Y	.	.	.
Disassemble current instruction	Y	Y
Disassemble block	Y
Execution commands						
Run from current address	Y	Y	Y	Y	Y	Y
Run from arbitrary address	Y	.	Y	.	.	.
Run instruction count	Y
Single step instruction	Y	.	Y	Y	Y	Y
Step instruction count	Y
Step to end of subroutine	Y	.
Step over subroutine	Y	Y
Step to non-EXEC code	Y	.
Run with source-code linked
Breakpoints						
Run to breakpoint	Y	.	Y	Y	Y	Y
Step to breakpoint	Y
Enable / disable breakpoint	Y	.	Y	Y	Y	Y
Set breakpoint address	Y	.	Y	Y	.	Y
List breakpoint addresses	Y	.	Y	Y	.	Y
Watchpoints						
Observe memory reads	Y
Observe memory writes	Y
Set watchpoint address	Y
List watchpoints	Y

Figure 30: Mattel Debugger Software Summary

4.4 EPROM and T-Cards

In the final stages of testing, software was transferred to EPROM based test cartridges known as T-Cards. They acquired this name because of their characteristic T shape, as shown in Figure 31. The reason for this design was entirely technical. The General Instrument CP-1610 CPU in use on the Intellivision has a multiplexed address and data bus, with rather unusual control signals. To make use of cheaper, standard EPROM chips it was necessary to include glue logic to decode the buses and interface them with standard memory protocols. This logic was not required on production cartridges, as they used General Instrument ROMs that interfaced directly with the proprietary bus.

This glue logic, and the fact that the 10 or 16-bit ROM used by the Intellivision required twice as many 8-bit EPROMs as production GI ROMs, meant that the test cartridges had to be substantially larger than production models. From here, the need to maintain the narrow dimensions of the cartridge slot, and minimise space use, led to the T layout.

The T-Card is not a single product, in fact there are at least three different designs known to exist. The most flexible is believed to be pictured in Figure 31.

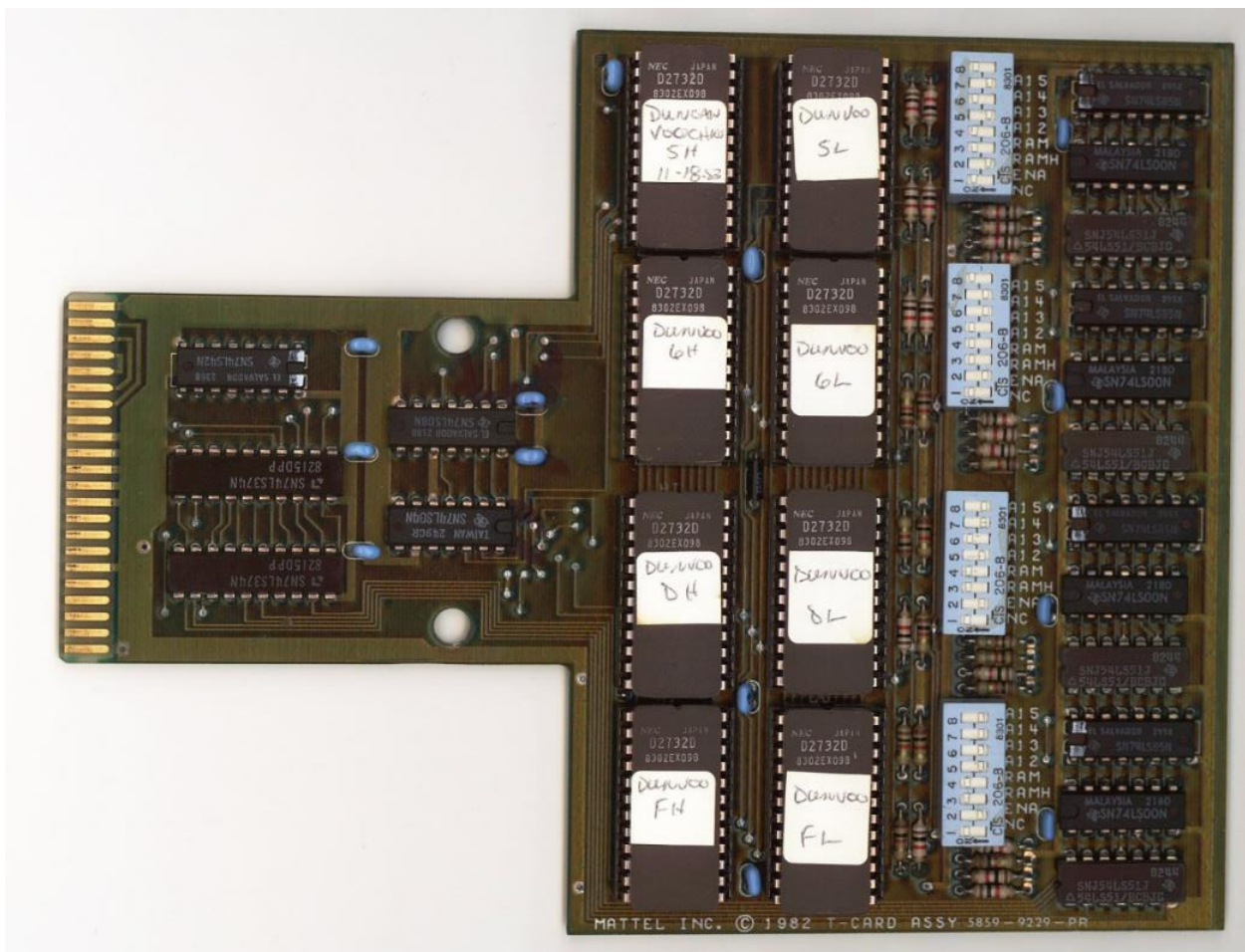


Figure 31: Intellivision 16K ROM / RAM T-Card ([image courtesy of FrankP](#))

As can be seen in the image above, the T-Card can support up to four banks of 4K decles using 2732 EPROMs, giving a maximum of 16K of memory. Each bank can be located at any 4K page boundary in the memory map by setting DIP switches (A12-A15). Alternatively, each bank could be set up to contain 2K decles of RAM using 6116 SRAM chips, giving up to 8K of 16-bit RAM. A full description of the T-Card including circuit schematics can be found in Section 9 of De Re Intellivision. A wide, double page render of which [can be found here](#).

A second T-Card design can be seen in two images on the [PapaIntellivision site](#), reproduced in Figure 32. This design is believed to be an earlier iteration and is much simpler. It can host at most four 2732 EPROM chips giving it a maximum capacity of 8K decles. As it does not appear to have any configuration switches and much simpler glue logic it seems unlikely that it would support the use of RAM or the mapping of its memory to different addresses.

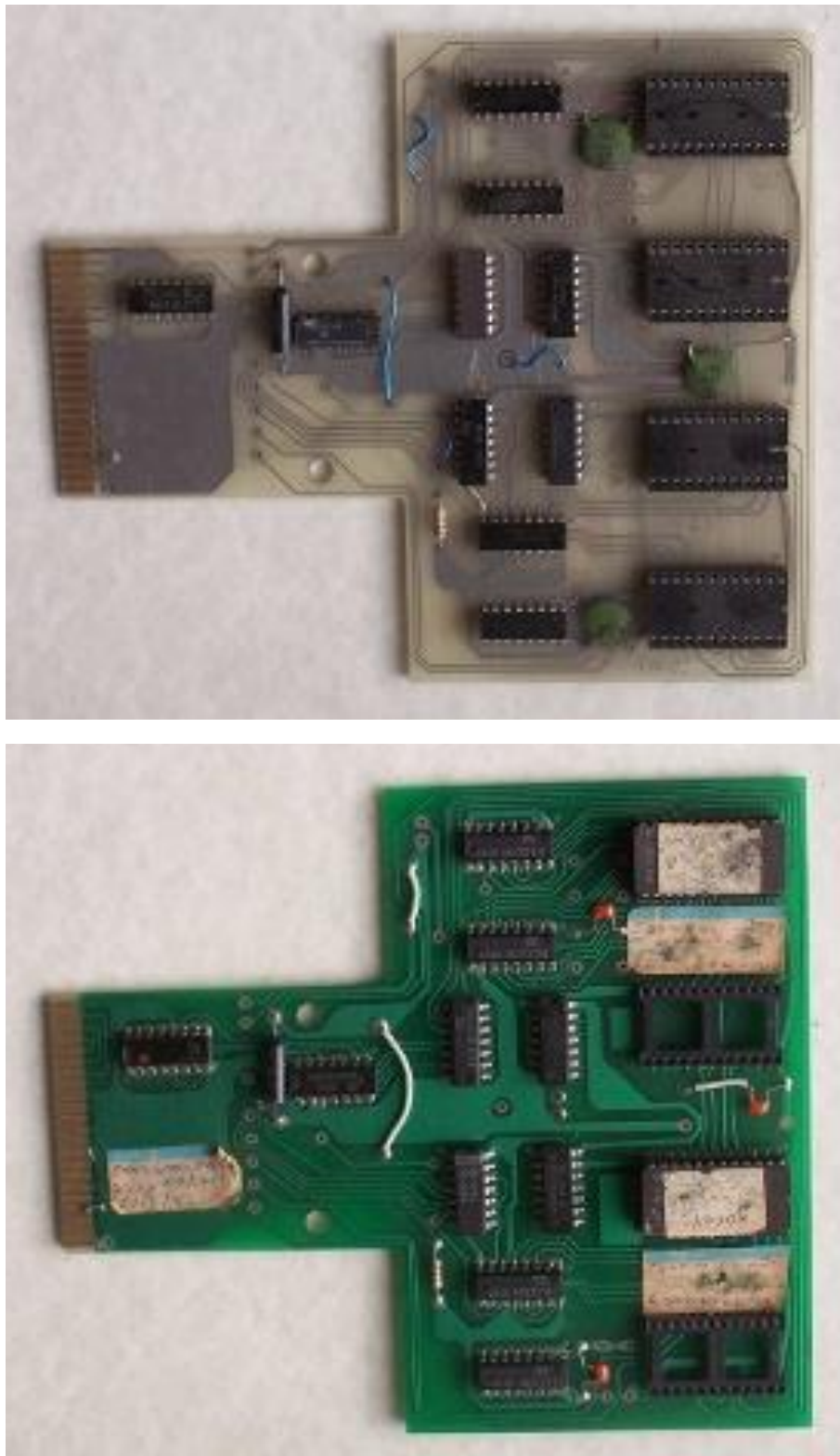


Figure 32: Intellivision 8K ROM T-Cards ([image courtesy of PapaIntellivision](#))

The final design seems to be a later iteration and is shown in Figure 33, although this initially seems to be similar to the PapaIntellivision design its four EPROM chips are larger 28 pin devices. The wiring of these suggests that this design could host 2764 EPROMs giving it a capacity of 16K decles.

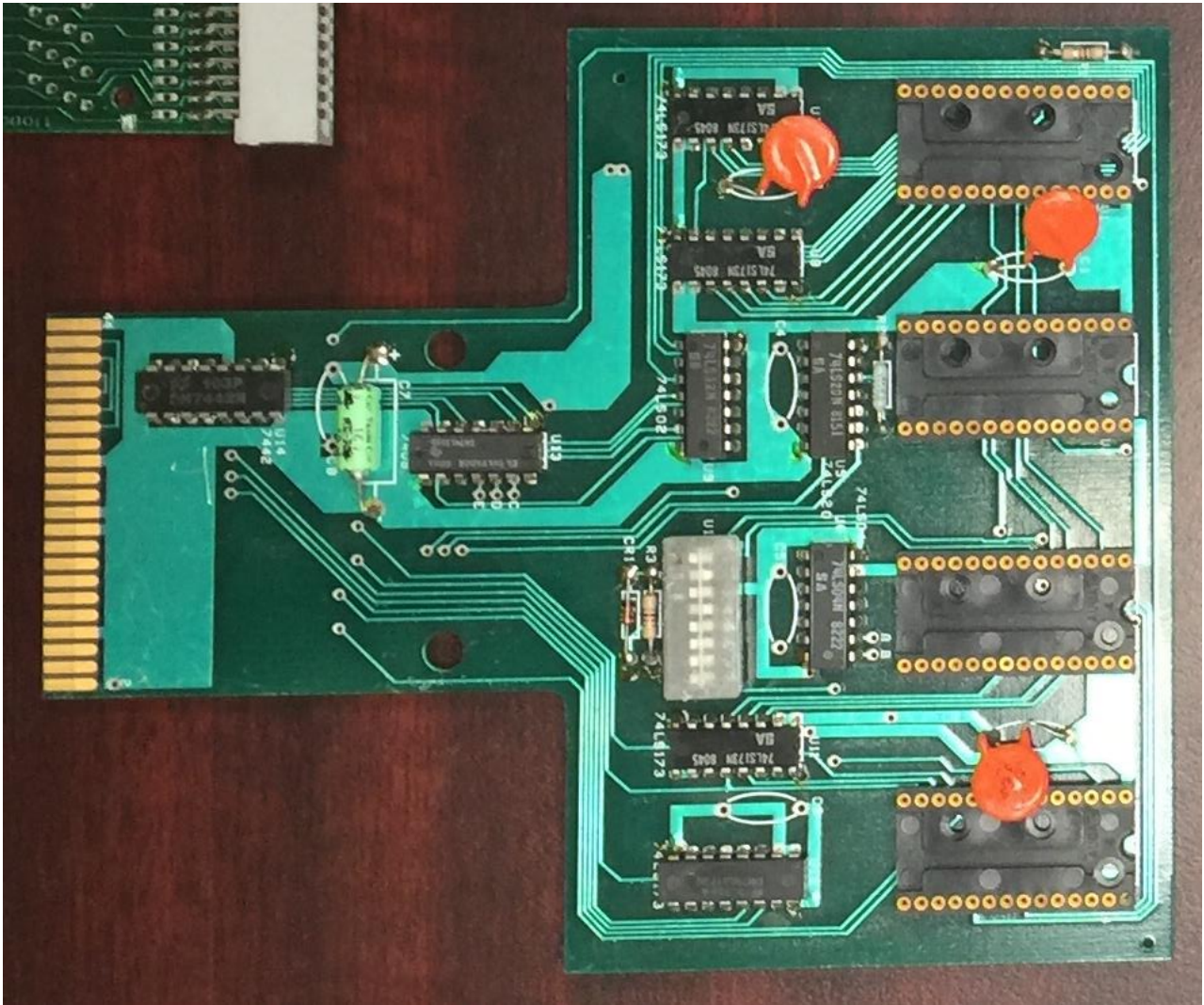


Figure 33: Intellivision 4K ROM T-Cards (image courtesy of Bill Fisher)

The existence of a bank of configuration DIP switches suggests that this board might also support the remapping of memory addresses or use of RAM, although this has not been confirmed.

Finally, it was possible to stack a pair of T-Cards to allow games with more than 16K of code to be run. To do this a Y connector, [as illustrated by Daniel Bass](#) and shown in Figure 34, was used.

The T-Cards seem to have primarily been used for QA testing as Keith Robinson describes at PRGE 2014 [V9] and promotional activities, such as trade shows. According to John Sohl in the BSR informal video chat [V2], this included demonstrating work in progress to Mattel prior to January 1981 when all development work was undertaken at APH.

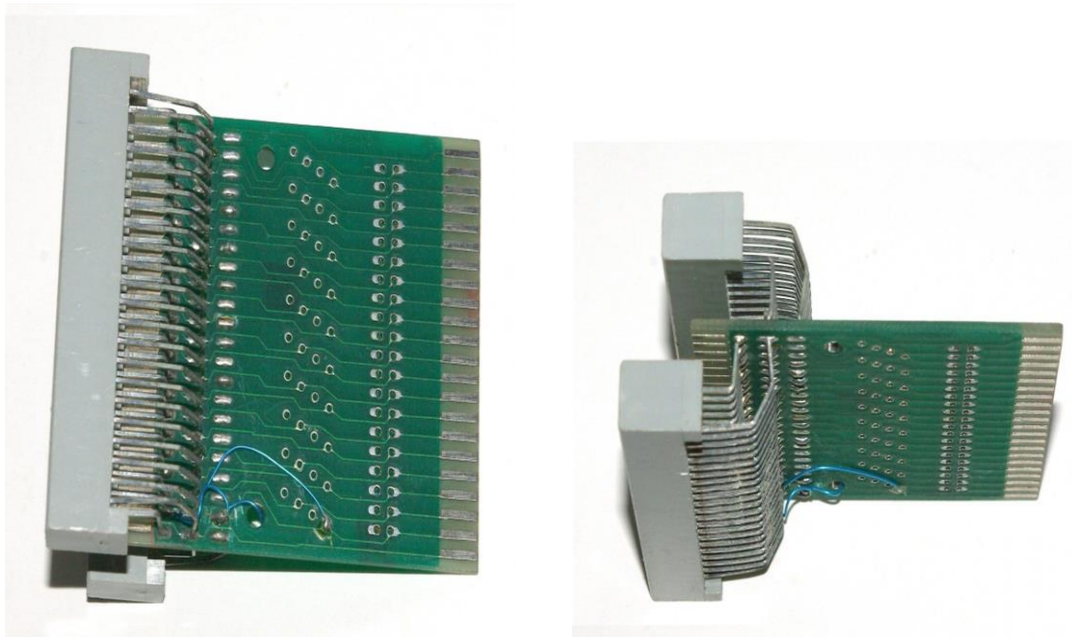


Figure 34: Dual T-Card Y Adapter

4.5 Documentation

The first significant piece of documentation used by the APh and Mattel developers was [Your Friend The EXEC](#), written by David Rolfe as he constructed the EXEC and Baseball. Initially, the developers at Mattel did not have access to the source code of the EXEC, and even later, it was not available as a general resource to developers.

YFTE was complemented by extensive notes constructed by John Sohl. This is briefly described during Classic Game Fest 2016 [V12] and led to John earning a reputation within Mattel for knowing everything about the Intellivision, and the nickname Dr Sohl.

As noted above in 1982 and 1983 the infrastructure teams at Mattel regularly produced [Technical Bulletin memos](#), which were distributed to the Blue Sky Rangers. These described updates to the development tools and infrastructure at the BSR's disposal.

4.6 Tight Code

The focus on writing tight code to extract the most from the limited performance of the CP-1610 is highlighted by Steve Roney and Bill Fisher discussing the use of the EXEC in Intellivisionaries episode 28. Here they describe how developers bypassed the official EXEC entry points in order to save the execution of a few instructions at the cost of corruption of unneeded registers.

At PRGE 2018 Bill Fischer, Steve Roney and David Warhol discussed the importance of minimising memory usage in games, including techniques to limit GRAM card usage through the distortion of images [V16]. They also talked about coding competitions to implement functionally equivalent code in the minimum memory space. David Warhol then goes on to describe how the extensive music found within Thunder Castle was made possible by implementing a compressed music format and new tracker rather than making use of the EXEC routines [V17].

4.7 Use of High-Level Languages

At PRGE 2014 [V7] Keith Robinson briefly describes some initial work undertaken by Steve Roney at the end of the Mattel era to investigate the use of the programming language C for developing games. In the CGE 2010 panel discussion [V9] Keith makes it clear that this work was initially focused on the Aquarius version of Utopia rather than an Intellivision title.

4.8 Minkoff Measure

Several Intellivision developers refer to an interview technique known as the Minkoff Measure. This was instituted by Mike Minkoff following the recruitment of Russ Lieblich who, whilst a talented musician (he wrote the wonderful sudden death tunes at the end of SNAFU), seems to have found programming challenging.

The measure appears to have been what is now a fairly standard “explain what this code fragment does” exercise. It was given to candidates during an interview and they would have to describe the behaviour of the code verbally.

The code fragment seems to have been approximately 10 lines of CP-1610 assembly language lifted from the EXEC or a game (presumably one of Mike’s titles, PBA Bowling or SNAFU). During the Classic Game Fest 2016 panel, Keith Robinson [V11] describes it as being a while loop, with the test at the beginning, which would repeat seven times applying an OR operation to seven of eight input digits. The use of an OR operation is interesting as the CP-1610 had no atomic instruction for this. Together, identifying the logical OR operation from its constituent parts and the fact it was applied to only seven of the eight digits seem to have been the critical points in a candidate making the measure.

The following is a bit of idle speculation. Talking to JoeZ, he suggests that the Minkoff Measure may have been taken from the X_SET_BIT function in the EXEC (starting at \$16E6), which makes use of the X_POW2 subroutine (starting at \$1745). This function sets the n^{th} bit of an address in memory. X_POW2 creates a bit mask by repeatedly shifting the value 1 left. This mask is then OR’d with the current value of the memory location.

Whilst this contains the key features mentioned by Keith, at 19 instructions excluding any register initialisation, it is rather longer than the test recalled by the BSRs. It also includes several PDP-11 / CP-1610 specific idioms, like the use of the R5 calling convention, the inclusion of post-increment indirect addressing and using INCR R7 to skip an instruction. As such, it might have been too domain specific for use in an interview. However, it could have formed the basis of a simplified piece of code that would perhaps be a fairer test for developers not familiar with the idiosyncrasies of the CP-1610 or PDP-11. As noted above, this is speculation on my and Joe’s part, suggestions of alternative Minkoff Measures that meet Keith’s criteria are most welcome.

4.9 Reverse Engineering

In the [Technical Bulletin describing the use of Mr Color](#) on the PDP-11 there is a short description of the tool “spout”. This would upload an 8K decle image of the Intellivision memory from \$5000 to \$6FFF to the PDP-11 host machine. As part of this description the document notes that spout can be used to rip Coleco and Activision ROM images from a cartridge inserted into a Blue Whale. Further, the same document then goes on to describe the specific tool ‘magic’ which will perform a similar task on Imagic cartridges which occupy \$4800 - \$4FFF.

The fact that Mattel were willing to commit to formal documentation of cartridge ripping and reverse engineering tools is interesting. Mattel’s internal attitude toward reverse engineering is made explicit in the following paragraph [found within the bulletin documenting the VAX version of the MICE uploader](#) program. This test harness is believed to have been used for Atari 2600 and Apple II testing.

```
By the way, reverse engineering is quite legal. Infact, Mattel has taken pains to fully document the reverse engineering of one of the Atari produced cartridges so that Atari can't sue us. The way Atari has sued other companies in the past is on the basis of theft of trade secrets, like by hiring ex-Atari employees. So reverse engineer all you want. Just don't copy code line for line and incorporate it in a Mattel game because then there may be copyright problems.
```

Figure 35: Reverse Engineering is Fine, Copying Code, Not So Much

5 Third Party Development

5.1 Activision / Cheshire Engineering

Development of Activision games was undertaken by Cheshire Engineering, an offshoot of APH. In Intellivisionaries episode 6 Tom Loughry mentions that Larry Zwick at Cheshire Engineering produced the tools, including assemblers, for this activity. Cheshire also constructed hardware debugging tools, presumably similar to those in use at APH as the same personnel worked on them.

[Joe Zbiciak reports](#) that when David Rolfe moved to Cheshire, and wrote Beamrider for Activision, the toolchain used employed General Instruments assembler syntax rather than APH's.

5.2 Atari

In Intellivisionaries episode 19 Russ Haft briefly mentions the EXEC equivalent developed at Atari and which formed the foundation of their games. For legal reasons, like all third parties other than INTV, this was written from scratch in a clean environment. The Atari framework was known as the MCP (Master Control Program, a Tron reference) and ran at 60Hz.

5.3 Imagic

Gary Kato suggests that Imagic used a similar two step development process to Mattel also centred on a PDP-11 [in this interview](#). However, in this instance the machine was a DEC PDP-11/23 running UNIX rather than TSX-Plus as seen at Mattel. It seems that Gary wrote Imagic's CP-1610 cross assembler and he still has a copy of the source code. Gary, please can you put this into the public domain, we would love to get it working again in an emulator.

Later, having moved offices to Los Gatos, Imagic migrated development from the PDP-11/23 to a DEC VAX-11/750 running BSD 4.1. It might also be inferred from Gary's comments that Imagic had a bespoke hardware solution for testing.

Gary initially appears to have written his own mini-EXEC, similar to other Intellivision third parties, but subsequently Imagic migrated to using the Mattel EXEC following the commissioning of a clean room reverse engineering project from an external consultancy. Examining the code of Imagic games they all appear to make calls to the EXEC's utility functions, for example FILL_ZERO. The earlier games, such as Demon Attack, make use of the ECS initialisation vector at \$4800 and do not seem to make a call back into the EXEC. As a consequence, they appear to completely bypass the EXEC's initialisation and game loop. This contrasts with later games, like Dracula, that make use of the main EXEC initialisation vector at \$5000 and contain a standard cartridge header. However, these headers do not appear to make use of the full EXEC game loop unlike Mattel games. Therefore, the meaning of the distinction made in Gary's interview is not clear.

A couple of videos showing development at Imagic can be found on YouTube:

<https://www.youtube.com/watch?v=R3x6ldp8oT0>

<https://www.youtube.com/watch?v=qmrZUNIDM2k>

These nicely contrast the difference in development approach taken on Atari and Intellivision games. Whilst in the first video Pat Ransil has a terminal and Intellivision test harness in his office, in the second Dennis Koble goes to a separate lab to program the Atari 2600.

Finally, whilst at Imagic in 1982 Rick Levine wrote a graphics development tool call Da Vinci [as mentioned here](#). No further information is known about it at this time.

5.4 INTV

INTV acquired the Intellivision intellectual property from Mattel and continued Intellivision development. To support this work David Warhol wrote an assembler and linker in C on the PC. In Intellivisionaries episode 16 he describes how this assembler had to be capable of compiling the two syntaxes of CP-1610 assembly language found at Mattel. The two formats were the result of Mattel using both the APh toolchain on PDP-11s and the Nuvatec cross-assemblers when working on the VAX hosts.

Originally David had intended to connect the PC to an Intellivision through a parallel interface to the controller ports and use this to download code to be tested. The Intellivision would run a small bootloader to read the incoming data from the controller ports to a RAM cart. However, David could not get this scheme to work reliably. Instead he employed Scott Robitelle to construct a ROM emulator. This was an expansion card that sat in the PC containing RAM that both the PC and Intellivision could access. The PC was then connected to the Intellivision using a parallel cable to a cartridge shell. As a consequence, the architecture of the INTV development system was very similar to that of the Mattel Magus, and David believes that the results were superior to the development kits available at Mattel.

Both Eric Well's Mr Color (Section 4.2.5) and Rick Koenig's native debugger (Section 4.2.9) were also used for development at INTV.

With the march of time the original 20Hz EXEC in the Intellivision became a hindrance, making games that used it seem sluggish in comparison with the competition. Like other third-party developers INTV developed their own 60Hz EXEC, called the Revised EXEC or REX. This was written by David Warhol and Steve Ettinger and was first used by Super Pro Golf, as described by David and Steve in Intellivisionaries episode 18.

According to Intellivision Lives, musician [David Warhol wrote tools](#) to convert a MIDI recording to Intellivision source code as a means of generating in game music more efficiently. This tool was used to convert an improvisation by Jimmy Martin into the music for Deep Pockets Super Pro Billiards [as reported in the notes for Intellivision in Hi-Fi](#). Unfortunately, INTV folded before the game could be released or the tools put to use on other Intellivision titles.

5.5 Technology Associates

Finally, mention must be made of the work undertaken by Joe Jacobs and Dennis Clark who [managed to construct their own development system](#) based on the PlayCable. The results were good enough for them to land a Mattel contract to port Bump 'N' Jump to the Intellivision. As noted by Keith Robinson at the Classic Game Fest 2016 Intellivision panel [V10], Joe and Dennis are engineers who worked for Jerrold, the manufacturers of the PlayCable.

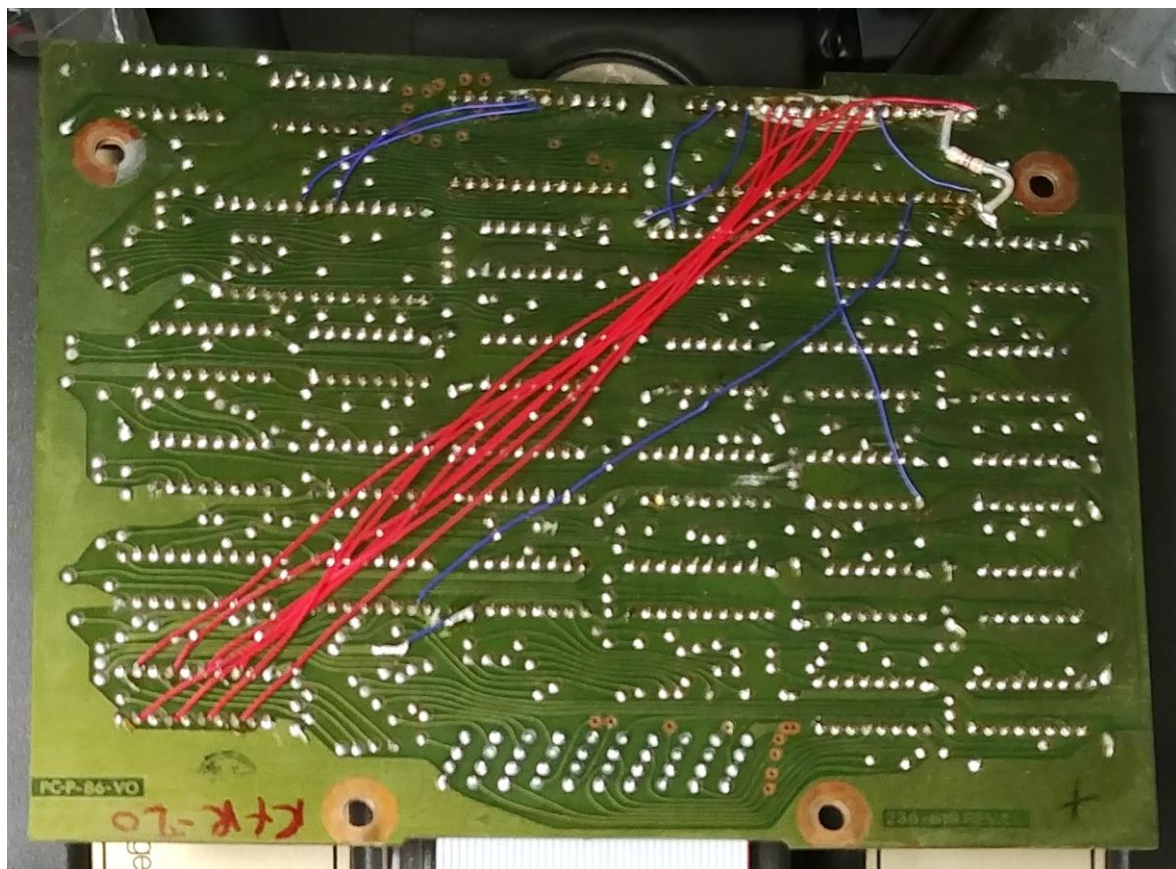
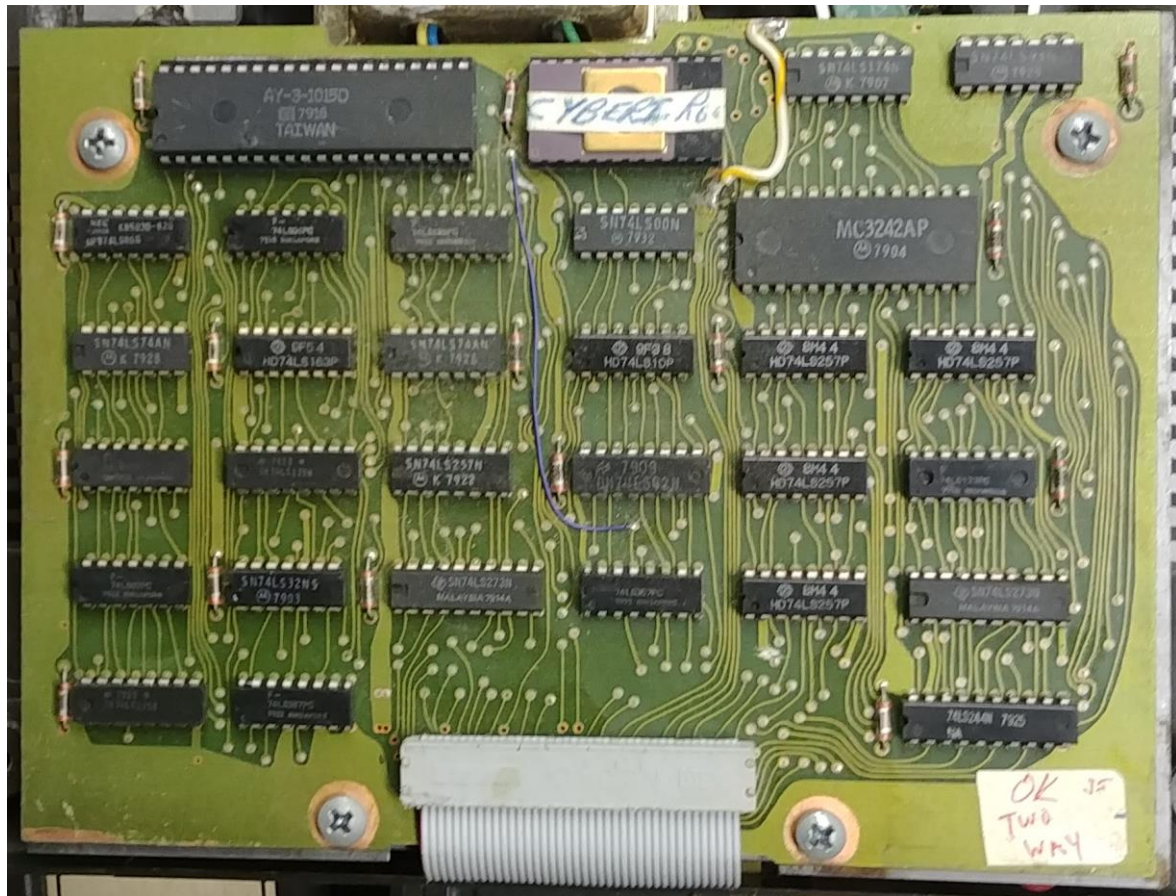


Figure 36: Technology Associates Development PlayCable Adapter Logic Board (image courtesy of Joe Jacobs)

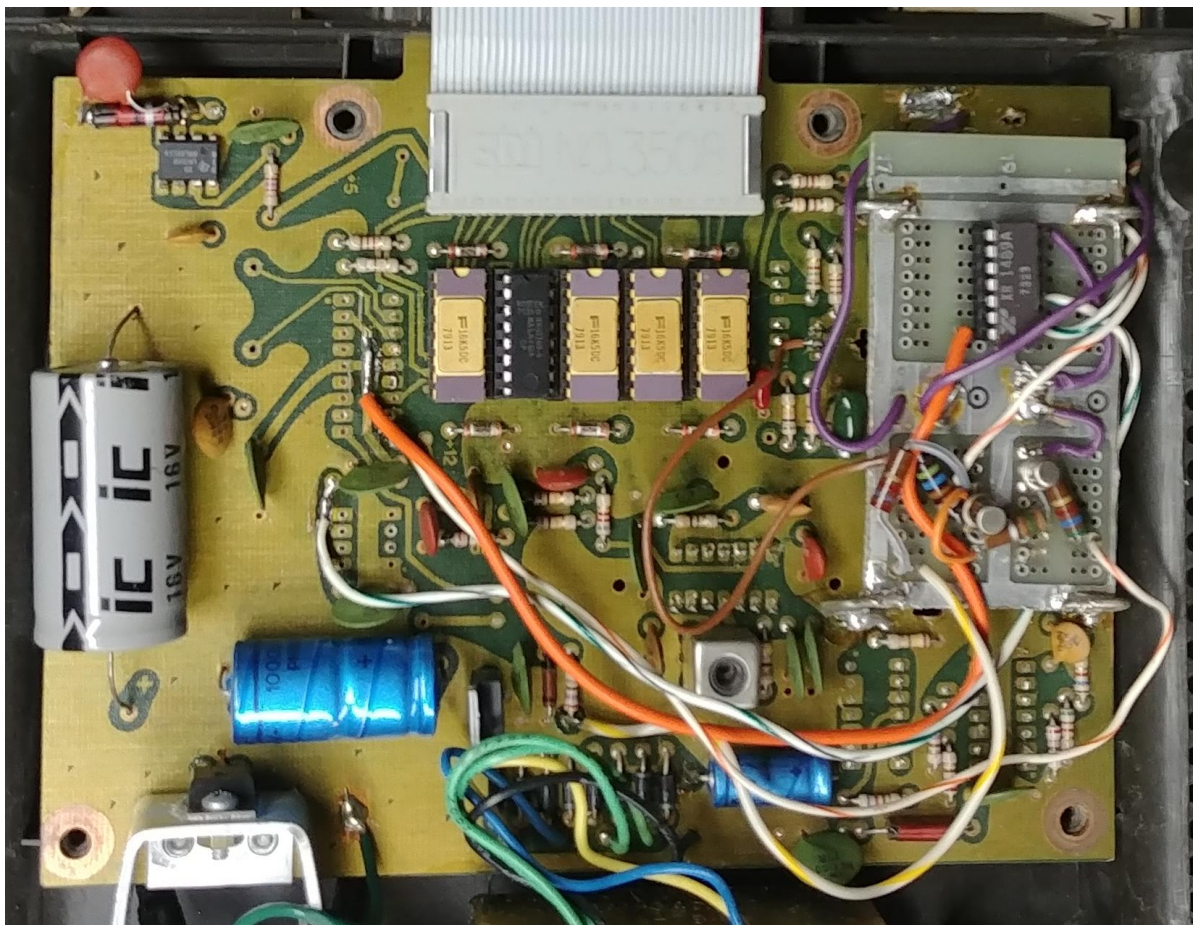


Figure 37: Technology Associates Development PlayCable Adapter Receiver Board (image courtesy of Joe Jacobs)

It appears that there was a bit of a history of PlayCable hacking within the Jerrold engineering department. This was focused on the earlier Jerrold branded model, which had a main logic board built of standard parts, rather than a single, consolidated ASIC of the later General Instrument branded units. Joe and Dennis took this hacking to the next level, patching the digital board of a Jerrold PlayCable, see Figure 36, and stripping out much of the receiver circuit, as shown in Figure 37.

These changes allowed the PlayCable to be directly connected to a PDP-11 compatible computer, which both Joe and Dennis possessed, using an RS-232 serial link. This alteration opened up two-way communication between the PlayCable and PDP-11 using standard protocols, rather than the PlayCable's more esoteric format. The ROM found on the PlayCable digital board, which normally contained the bootstrap firmware to load the PlayCable menu, was replaced with a larger, 4K x 8 EPROM, shown in the middle of Figure 38.

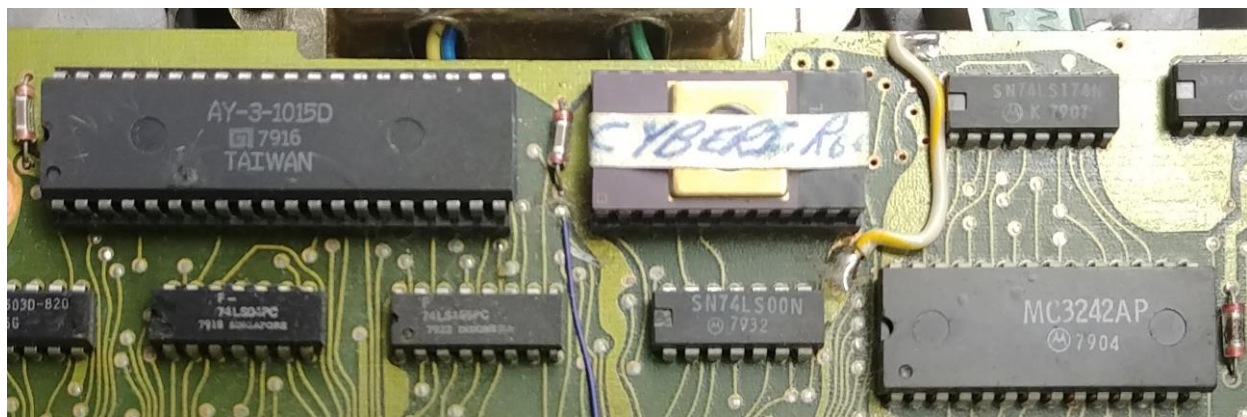


Figure 38: Development PlayCable Digital Board (image courtesy of Joe Jacobs)

Without access to the ROM and serial UART (top left in Figure 38), as provided by the older Jerrold digital board, these changes would not have been possible. Joe's adaptations allowed Dennis to construct a debugger, of up to 2K decles in size with similar features to the Blue Whale development kit. Using this debugger, it was possible to load and run code from the PDP-11, add and trigger breakpoints and single step code, much like the developers at Mattel. Just like the Blue Whale development kit the breakpoint and single stepping features implemented by Dennis required the use of modified Master Components. In this instance the all-important interrupts were triggered by accessing specific memory addresses, rather than issuing a software interrupt instruction. When the trigger address was accessed, Joe's "vector" add-on board spotted it on the CP-1610 bus and signalled an interrupt. The resulting harnesses could be used to test games of up to 8K decles in size using the RAM resident in the PlayCable to host the program under test. More technical details regarding development system constructed by Joe and Dennis, including reverse engineered schematics, can be found in the [PlayCable Technical Summary](#).

Joe and Dennis constructed these homebrew development systems with the intention of landing jobs with Mattel to program Intellivision games. They realised that they would need a demonstration to showcase their capabilities and catch the eye of Mattel's managers. Joe and Dennis used their personal PDP-11s and homebrew test harnesses to put together a version of PAC-MAN, which they named Clone-Man, and sent a video of it to Don Daglow. This eventually led to their developing Intellivision Bump 'N' Jump for Mattel. However, that is another, longer story...

6 References

6.1 Intellivisionaries Podcasts

Many of the developers interviewed on the Intellivisionaries podcast describe or reference the tools and methods of development they used. The following table identifies the episodes and time codes where these early development techniques are discussed:

Episode	Time	Interviewee	Content
5	2:50	Keith Robinson	Use of keyboard component for development
6	3:40	Tom Loughry	Cheshire Engineering development
7	2:54	David Rolfe	Aph framework and test harnesses
7	5:32	Eddie Dombrower	PDP-11 description
9	3:16	Keith Robinson	Game Factory and relationship with Mr Color
10	4:00	Tom Loughry	PDP-11 description
16	3:16	David Warhol	Initial development and lack of harnesses
16	3:46	David Warhol	INTV development
16	3:46	Connie Goldman	Use of Mr Color
17	3:35	Kimo Yap	PDP-11 and cross compilation
17	3:47	Kimo Yap	Aph use of a payphone for modem calls
18	4:08	David Warhol	Development of REX
19	4:25	Russ Haft	PDP-10 host
19	4:37	Russ Haft	Use of Mr Color
19	4:53	Russ Haft	Development of the Atari EXEC
26	2:45	Ray Kaestner / Rick Koenig	Use of Keyboard Component and debugger
26	3:10	Ray Kaestner / Rick Koenig	INTV PC based development and debugger
28	3:18	Steve Roney / Bill Fisher	Collaborative development and code management
28	3:25	Steve Roney / Bill Fisher	Use of alternate EXEC entry points
29	2:22	Mike Winans	PDP-11, TSX and the TECO editor
31	4:19	Mark Urbaniec	Development description

6.2 Video Interviews

The following references are to video interviews and panels conducted with Intellivision developers

Id	URL	Time	Interviewee	Content
V1	https://www.youtube.com/watch?v=Oil2XnGvb4M	0:24	David Rolfe John Sohl	Debugging tools
V2	https://www.youtube.com/watch?v=Oil2XnGvb4M	0:31	David Rolfe John Sohl	APh development
V3	https://www.youtube.com/watch?v=Oil2XnGvb4M	0:33	Keith Robinson	Andromeda hardware
V4	https://www.youtube.com/watch?v=Oil2XnGvb4M	2:45	David Warhol	Lack of debugger at Mattel
V5	https://www.youtube.com/watch?v=Vk98eyzFnFU	0:20	John Sohl	APh PDP-20 and debugging
V6	https://www.youtube.com/watch?v=Vk98eyzFnFU	0:47	Keith Robinson David Warhol	PDP-11 floppy drives
V7	https://www.youtube.com/watch?v=63pkqzodpqq	0:56	Keith Robinson	Use of C and DEC VAX
V8	https://www.youtube.com/watch?v=63pkqzodpqq	1:10	Keith Robinson	QA use of T-Cards and taping. Mattel use of Datawidget
V9	https://www.youtube.com/watch?v=zq40a_w55yl	0:05	Keith Robinson	Use of C
V10	https://www.youtube.com/watch?v=QqwmZ1DBXyU	0:21	Keith Robinson	Bump 'n Jump development
V11	https://www.youtube.com/watch?v=QqwmZ1DBXyU	0:49	Keith Robinson	Minkoff Measure
V12	https://www.youtube.com/watch?v=QqwmZ1DBXyU	0:54	Keith Robinson	John Sohl's documentation
V13	https://www.youtube.com/watch?v=yZlcqvlGfGY	0:04	David Warhol	Magus design, use of DEC VAX
V14	https://www.youtube.com/watch?v=hfEwvm3bR9M	16:40	Gabriel Baum Bill Fischer David Warhol	Datawidget development
V15	https://www.youtube.com/watch?v=hfEwvm3bR9M	26:50	Ray Kaestner Bill Fischer Steve Roney	Joint development
V16	https://www.youtube.com/watch?v=hfEwvm3bR9M	42:40	Bill Fischer Steve Roney David Warhol	Code compression
V17	https://www.youtube.com/watch?v=hfEwvm3bR9M	43:30	David Warhol	Thunder Castle music

6.3 Audio Interviews

The following references are to audio interviews and panels conducted with Intellivision developers

Id	URL	Time	Interviewee	Content
A1	http://www.digitpress.com/cge/2k4_mp3/hi/intellivision%20panel%20at%20cge2k4.mp3	0:08	Peter Kaminsky	APh PDP-10 / 11, Pixel Editor
A2	http://www.digitpress.com/cge/2k4_mp3/hi/intellivision%20panel%20at%20cge2k4.mp3	0:30	Bill Fisher John Sohl Keith Robinson	Mattel use of Datawidget, Merlin, Debugging process