The IntelliCart!

The IntelliVision Cartridge Emulator

from

Chad Schell

of

Schell's Electronics

License and Warranty Information

LICENSE TO USE. Schell's Electronics grants to you a non-exclusive license for the use of the accompanying Intellicart and the software contained therein. The Intellicart software is copyrighted, and title to the software and all intellectual property rights are retained by its author. You may not make any copies of the cartridge software or hardware.

LIMITED WARRANTY. Schell's Electronics warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the Intellicart will be free from defects in materials and workmanship under normal use.

All software distributed with the Intellicart, but not actually contained in the Intellicart is provided AS-IS with absolutely no warranty of any kind.

To obtain warranty service, contact Schell's Electronics via email at support@schells.com. If your Intellicart requires service, you will be required to ship the Intellicart to Schell's Electronics, and you will pay the initial shipping charges. Schell's Electronics will pay any return shipping charges if applicable.

THIS WARRANTY IS EXPRESSLY MADE IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

TO THE EXTENT NOT PROHIBITED BY APPLICABLE LAW, SCHELL'S ELECTRONICS LIABILITY IS LIMITED TO, AT SCHELL'S ELECTRONICS OPTION, EITHER THE REPAIR OR REPLACEMENT OF THE INTELLICART OR THE REFUND OF THE PURCHASE PRICE, AND SHALL IN NO EVENT INCLUDE INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING FROM OR RELATED TO THE USE OR MISUSE OF THE INTELLICART, EVEN IF SCHELL'S ELECTRONICS HAS BEEN

ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL SCHELL'S ELECTRONICS LIABILITY TO YOU, TORT (INCLUDING NEGLIGENCE) OR OTHERWISE, EXCEED THE AMOUNT PAID BY YOU FOR THE INTELLICART UNDER THIS AGREEMENT. The above limitations will apply even if the above stated warranty fails of its essential purpose.

This agreement is effective until terminated. You may terminate this agreement at any time by returning the Intellicart and documentation to Schell's Electronics. This agreement will terminate immediately without notice from Schell's Electronics if you fail to comply with any provision of this agreement. Upon termination you must return the Intellicart and documentation to Schell's Electronics.

Any action related to this agreement will be covered by California law and the controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

If any provision of this agreement is held to be unenforceable, this agreement will remain in effect with the provision omitted, unless omission of the provision would frustrate the intent of the parties, in which case this agreement will immediately terminate.

This agreement is the entire agreement between you and Schell's Electronics relating to the Intellicart. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any confliction or additional terms of any quote, order acknowledgement, or other communication between the parties relating to its subject matter during the term of this agreement. No modification of this agreement will be binding unless in writing and signed by an authorized representative of each party.

The Intellicart!

Thank you for purchasing an Intellicart, the Intellivision cartridge emulator.

Getting Started:

The following is a quick introduction on how to use the Intellicart to play games on an Intellivision. Detailed specifications on the use of the Intellicart for the creation of new games can be found in the programmers section. Details on the communications format to the Intellicart are provided in the Intellicart Interface section.

Requirements:

Intellicart
Intellivision (Any model)
Computer with RS-232 Compatible Serial Port (The standard serial port on PCs.)
Binary Files (ROM images, the games)
Intellicart download software

Using the Intellicart

Connect the DB9 serial connector of the Intellicart to your computer's RS-232 compatible serial port. Depending on the type of serial port connector on your computer, you may require an additional adapter. (It is recommended that you only connect or disconnect the Intellicart from your computer while the computer's power is OFF.)

Note: The DB9 connector is the large connector at the end of the gray cable, NOT the round connector at the end of the black cable. The black cable provides access to the +5 volt and ground lines of the Intellivision. It is present to allow the possible future creation of an audio to serial adapter to allow the Intellicart to be loaded from a CD player or other audio source. It is not used at this time. (Do not short the center portion of this connector to ground, doing so might damage your Intellivision.)

MAKE SURE THE INTELLIVISION IS OFF. NEVER INSERT OR REMOVE THE INTELLICART FROM THE INTELLIVISION WHILE THE INTELLIVISION IS ON. DOING SO MIGHT RESULT IN DAMAGE TO THE INTELLICART. THIS IS TRUE FOR ALL MODELS OF THE INTELLIVISION.

Insert the Intellicart into the Intellivision's cartridge slot. (Or the cartridge slot of the Intellivoice or ECS if you have these accessories.)

Turn on the Intellivision. The Intellicart title screen should appear. If it does not, press the reset button on the Intellivision.

Start the Intellicart software on your computer.

Select the ROM image you wish to download (the game you wish to play) and download it to the Intellicart.

If the download was successful, the Intellicart will reset the Intellivision automatically and the ROM will start just as if you had inserted the actual cartridge into your Intellivision and pressed reset. If there is an error during the download, and error message will be displayed on the screen. (See error messages section of this manual.)

The Intellicart will now act as a cartridge containing the ROM image downloaded until the Intellivision is turned off. ONCE THE INTELLIVISION'S POWER IS TURNED OFF, THE ROM IMAGE WILL BE LOST FROM THE MEMORY OF THE INTELLICART.

If you wish to download another ROM image into the Intellicart, turn off the Intellivision and then turn it back on. The Intellicart title screen will appear and you can now download the new ROM image.

Error Messages

Error messages on the Intellicart are unfortunately all interrelated, as an error in communication can show up in a number of ways. If you are using software known to work with the Intellicart, and you get

any error, chances are you can fix it by simply repeating the download.

BAUD ERROR An error occurred while attempting to determine the baud rate of the download. The first remedy is to repeat the download to rule out a one time communication error. If BAUD ERROR persists, try a different baud rate. The Intellicart supports ONLY the following baud rates: 2400, 4800, 9600, 14400, 19200, 38400, and 57600. If switching to another baud rate does not solve the problem contact support@schells.com.

CRC ERROR The CRC of the downloaded file did not match that computed by the Intellicart. This means there was a serial transmission error, which the Intellicart has trapped to prevent bugs from occurring during game play. Repeat the download. If the problem persists contact support@schells.com.

OVERFLOW ERROR The receive buffer in the Intellicart overflowed. This error most likely indicates a problem in the communication between the Intellicart and the Intellivision. Turn off the Intellivision, remove the Intellicart and then reinsert it. Make sure it is inserted all the way into the cartridge slot. Turn the Intellivision back on and retry the download. If the problem persists contact support@schells.com.

TIMEOUT ERROR The next byte of serial data was not received in a timely fashion. This can be caused by almost any serial error. Check the connection between the Intellicart and the computer and try the download again. If the problem persists contact support@schells.com.

BAD FORMAT This error indicates that the ROM image downloaded to the Intellicart is not in the proper format, however, it could be caused by a serial transmission error. Repeat the download. If the problem persists try a different ROM image. If the problem is present using another ROM image, contact support@schells.com.

Troubleshooting:

Problem: The Intellicart title screen does not appear when I insert the Intellicart into the Intellivision and turn on the Intellivision.

Suggested Action: Press the reset button on the Intellivision. If the title screen still does not appear turn off the Intellivision, and ensure that the Intellicart is fully inserted into the cartridge slot. Turn the Intellivision back on and press the reset button. If the title screen still does not appear, test that the Intellivision is functioning correctly by inserting a standard Intellivision cartridge into the Intellivision. If that cartridge also fails to load the error is in the Intellivision and not in the Intellicart. If the Intellivision is functioning correctly and the Intellicart still fails to load, contact support@schells.com.

Problem: Chess, Triple Challenge, and Land Battle do not work correctly on the Intellicart.

Solution: This is a known limitation of the Intellicart. It only supports 16 bit RAM, while those games require 8 bit RAM.

Problem: The Intellicart title screen appears but nothing happens when I download a ROM image from the computer.

Suggested Action: SOMETHING should happen when a ROM is downloaded to the Intellicart. Even if the image downloaded is in the wrong format, at least an error message should appear. If the Load Image message does not change (it should change to Loading) check the connection between the serial port and the Intellicart. Ensure that the software is writing to the correct serial port. If a second serial port is available try switching the serial port used by the Intellicart. If there is still no response to downloads contact support@schells.com.

Problem: When I download a ROM image, the Load Image message changes to Loading, but nothing else happens, or after a download a blank or corrupted screen appears.

Suggested Action: After the Intellicart download software on the computer indicates the transfer is complete, press the reset button on the Intellivision. This error will most likely occur when some other transmission error occurred. The Intellicart uses a hardware reset of the intellivision in these situations, and it was determined in testing of the Intellicart that some Intellivisions have difficulty accepting a hardware reset issued from the cartridge port.

Problem: I received an error while downloading a ROM image. I've pushed reset but the error message does not go away.

Suggested Action: If a download error occurs, the Intellicart will display an error message on the television screen. The message will remain on the screen until a new download is started or the Intellivision is turned off. Resetting the Intellivision has no effect on the error display. There is no need to clear an error before attempting another download. Simply restart the download from the computer. If the error persists, see the error messages section for explanations of what various errors mean.

Programmer's Section

This section contains the necessary information to take advantage of the advanced features of the Intellicart for new programs. It does not cover basic information on the actual programming of the Intellivision, only those aspects which relate to the Intellicart itself.

The following conventions will be used in this section.

All references to memory addresses are 16 bit addresses.

0xYY Represents a hexadecimal value.

Two Memory maps exist, it is important to understand what each one refers to.

Intellivision Memory Map: This refers to the memory map

as seen by programs running on the Intellivision. The Intellivision can address 64K of 16 bit RAM or ROM, from 0x0000-0xFFFF.

Intellicart Memory Map: This refers to the addressing of the actual 64K of 16 bit RAM on the Intellicart itself, also from 0x0000-0xFFFF. By default it mirrors the Intellivision memory map, so a read or write to 0xXXXX will address the same space in the Intellicart RAM and the Intellivision address space. However, bankswitching can change this relationship, so that a window in Intellivision space will point to a different location in the Intellicart RAM.

<X...Y> represents a range of bits from X to Y inclusive. X and Y can range from 0, representing the least significant bit to 15, representing the most significant bit.
Examples:

<15:12> Represents bits 15,14,13, and 12, the highest nibble of the 2 byte word

<0> Represents bit 0, the least significant bit

Address <15:8> would represent the high byte of a 16 bit address.

To load an image to the Intellicart, a binary (.bin) file containing the actual Intellivision machine code of your program is required. Typically a configuration (.cfg) file is also required to inform the Intellicart how to behave and where to load your code. These files are read by a binary to ROM file conversion program to produce a file that is suitable for download to the Intellicart. More information on the format of the ROM file can be found in the Intellicart Download Protocol section of this manual. The Intellicart download programs supplied with the Intellicart have the binary to ROM conversion function built into them.

The Intellicart has 64K x 16 bits of static RAM onboard, all of which is available for use by programs. Access to this RAM is controlled in 2K segments by three enable bits to be explained later. To access the entire 64K of ram, bankswitching is required to work around the memory and devices contained in the Intellivision memory map. With bankswitching disabled, the memory map of the Intellicart exactly overlaps the memo-

ry map of the Intellivision. Thus a reference to location 0x5000 would reference that location in the Intellicart and the Intellivision. Care must be taken not to instruct the Intellicart to respond to addresses used by the Intellivision, as this will create bus contention that will most likely cause the Intellivision to crash.

Enable Bits:
A set of three enable bits are used to control the response of the Intellicart to reads from and writes to each 2K segment in the Intellivision memory map. These enable bits must be set when the ROM image is downloaded to the Intellicart. They cannot be modified from within the Intellivision program itself.

The bits are stored in lookup tables in the Internal memory of the Intellicart (not accessible to Intellivision programs.) As a programmer, to set these bits one creates a configuration (.cfg) file to be distributed with the ROM image. This file tells the Intellicart binary to ROM conversion code (built into the download programs available for use with the Intellicart) what enable bits to set for the program to run correctly.

The three enable bits correspond to the following actions. Setting a bit to 1 enables that action, and setting it to 0 disables that action.

1. Respond to Read Accesses (ROM emulation)
2. Respond to Write Accesses (RAM emulation)
3. Bankswitch enable (explained later)

To properly emulate RAM, both the Read and Write enable bits should be set.

The binary to ROM conversion program functions as follows.
If no configuration file is supplied, the program constructs a ROM image with one of the default configurations shown in Table 1. If your binary file does not match one of these program sizes and memory maps, you must use a configuration file.

Table 1: Default ROM sizes and mapping.

File Size (16 bit Words)	READ ENABLED ADDRESSES
1K	0x5000-0x5FFF
2K	0x5000-0x6FFF
4K	0x5000-0x6FFF,0xD000-0xDFFF
8K	0x5000-0x6FFF,0xD000-0xDFFF, 0xF000-0xFFFF

Configuration file syntax:
The configuration file consists of commands followed by addresses to which those commands should apply. The commands have two purposes. They tell the Intellicart how much data to expect in the download and where to place that data, and they tell the Intellicart how to respond to read and write commands for 2K segments of the Intellivision memory map. The default behavior is NOT to respond to any commands.
The syntax of the file was built to be compatible with the configuration files found on the Intellivision Lives CD ROM.

Anything after a semicolon (;) on a line is a comment, and is ignored.

All numbers represent hexadecimal values, and the units are 2-byte words (16 bits.)

For instructions that involve loads to the Intellicart, [mapping] and [preload], the values to the left of the equal sign represent offsets into the binary (.bin) file itself, not into the Intellicart or Intellivision memory maps. These offsets must be sequential and continuous. (i.e. if the last [mapping] command was for 0x0000-0x0FFF, the next [mapping] or [preload] command must start at 0x1000.)

The Intellicart is designed to manipulate only the upper 8 bits of the Intellivision address space. For this reason all addresses specified in the configuration file must span complete 256 word segments. For example, 0x5000-0x50FF is acceptable, but 0x5000-0x5010 is not acceptable. Starting values must all be of the format 0xyy00 and ending values must all be of the format 0xzzFF.

The Intellicart also operates primarily on 2K word boundaries, like 0x5000-0x57FF or 0x5800-0x5FFF. To work around small memory holes in the Intellivision memory map, finer access control is available, but only one segment can be activated in each 2K word segment. For example, you could inform the Intellicart to respond to read accesses in the range 0x5000-0x53FF, or 0x5400-0x57FF, or even 0x5400-0x55FF, but NOT to a split area such as 0x5000-0x51FF AND 0x5400-0x55FF at the same time. To use this finer access control simply specify the desired start and end values in the configuration file just as you would any other address.

Now onto the instructions.

[mapping] - ROM emulation: The [mapping] command is used to instruct the Intellicart to emulate ROM over the specified address range in the Intellivision memory map. It also instructs the Intellicart to expect data to be downloaded to fill that area in its own memory. The format of the command is as follows

```
[mapping]
$aaaa - $bbbb = $cccc
      ...
$aaaa - $bbbb = $cccc
```

Where [aaaa] is the starting offset in the binary file, [bbbb] is the stop offset in the binary file, and [cccc] is the starting location to place the data in the Intellicart memory map.
As many segments as are required can follow the [mapping] command, with each line having the same syntax.

Example: To map the first 0x1000 words of a binary file to 0x5000-0x5FFF in the Intellivision memory map, and to inform the Intellicart to respond to read commands over the same range, the following command would be used.

```
[mapping]
$0000 - $0FFF = $5000
```

[memattr] - RAM emulation: The [memattr] command is used to instruct the Intellicart to emulate RAM over the specified address range in the Intellivision memory map. It does not instruct the Intellicart to load any data into this RAM. The [memattr] command can be applied to areas already configured by other commands to enable write access to those areas. The format of the command is

```
[memattr]
$aaaa - $bbbb = RAM
      ...
$aaaa - $bbbb = RAM
```

Where [aaaa] and [bbbb] are the start and end range in the Intellivision memory map where RAM should be available.

Example: To activate a block of 16 bit RAM at from 0xD000-0xDFFF in the Intellicart memory map, the following command would be used.

```
[memattr]
$D000 - $DFFF = RAM
```

The above two commands are the only commands required for non-bankswitching programs. The next two commands are only useful if bankswitching is going to be used.

[bankswitch] - Bankswitch enable: The [bankswitch] command tells the Intellicart to activate bank switching over the specified address range in the Intellivision memory map. It also informs the Intellicart to respond to read accesses over this range, but does NOT inform the Intellicart to expect data to be loaded into its memory. The format of the command is

```
[bankswitch]
$aaaa - $bbbb
      ...
$aaaa - $bbbb
```

Where [aaaa] and [bbbb] are the start and end range respectively to activate bankswitching. It is not possible to have bankswitched and non-bankswitched areas with the same 2K word segment.

For example, to enable bankswitching of 0x6000-0x67FF in the Intellivision memory map, the following command would be used

```
[bankswitch]
$6000 - $67FF
```

[preload] - Preloading of Intellicart memory: The reason for using bankswitching is to gain access to parts of the Intellicart memory map that would otherwise not be accessible due to conflicts in the corresponding Intellivision memory map locations. Obviously the ability to load data into that space makes access to it more useful. The [preload] command informs the Intellicart to download data to an area of its own memory map, just as in the [mapping] command, but does NOT set the area to respond to read accesses, as this would cause bus contention in the conflicting areas. The format of the command is

```
[preload]
$aaaa - $bbbb = $cccc
      ...
$aaaa - $bbbb = $cccc
```

Where the values have the same meaning as in the [mapping] command.

Bankswitching

To access all 64K of RAM on the Intellicart, bankswitching is required. To use bankswitching, at least one 2K segment in the Intellivision memory map must have bankswitching enabled. Enabling bankswitching on a 2K segment allows you to point that 2K segment of the Intellivision memory map to point to ANY location in the Intellicart memory map.

WARNING: DO NOT BANKSWITCH YOUR STARTING BANK.

The bankswitch offset tables that control where a bankswitched window points are NOT guaranteed to be initialized to any value when your program starts. The Intellicart does not monitor cart resets so it cannot control the state the tables are in after a reset. If you bankswitch your starting code window, it will quite likely point to something other than your starting code, causing the program to fail.

For a 2K segment that has bankswitching enabled, the Intellicart intercepts all accesses to those addresses and redirects them. The redirection is based on the following algorithm.

```
Old Address = 0xYYZZ
New Address = 0xXXZZ
```

Where 0xXX = (0xYY & 0x07) + (Bankswitch offset), and & represents bitwise logical and.

The bankswitch offset is an 8 bit value stored in a table in the Intellicart's internal memory (not directly accessible to Intellivision programs.) This table can be accessed by writing to the low byte of locations 0x0040-0x005F (see Table 2 for mapping), It cannot be read.

Table 2: Bankswitch Offset Loading Addresses

Intellivision Memory Range	Bankswitch offset table address	Intellivision Memory Range	Bankswitch offset table address
0x0000-0x07FF	0x0040	0x0800-0x0FFF	0x0050
0x1000-0x17FF	0x0041	0x1800-0x1FFF	0x0051
0x2000-0x27FF	0x0042	0x2800-0x2FFF	0x0052
0x3000-0x37FF	0x0043	0x3800-0x3FFF	0x0053
0x4000-0x47FF	0x0044	0x4800-0x4FFF	0x0054
0x5000-0x57FF	0x0045	0x5800-0x5FFF	0x0055
0x6000-0x67FF	0x0046	0x6800-0x6FFF	0x0056
0x7000-0x77FF	0x0047	0x7800-0x7FFF	0x0057
0x8000-0x87FF	0x0048	0x8800-0x8FFF	0x0058
0x9000-0x97FF	0x0049	0x9800-0x9FFF	0x0059
0xA000-0xA7FF	0x004A	0xA800-0xAFFF	0x005A
0xB000-0xB7FF	0x004B	0xB800-0xBFFF	0x005B
0xC000-0xC7FF	0x004C	0xC800-0xCFFF	0x005C
0xD000-0xD7FF	0x004D	0xD800-0xDFFF	0x005D
0xE000-0xE7FF	0x004E	0xE800-0xEFFF	0x005E
0xF000-0xF7FF	0x004F	0xF800-0xFFFF	0x005F

Example 1: To remap 0x6000-0x67FF in the Intellivision memory map to point to 0x1000-0x17FF in the Intellicart memory map, do the following.

Set the bankswitch enable bit for 0x6000-0x67FF when the ROM is downloaded.

From the Intellivision program, write 0x0010 to address 0x0046.

Example 2: To remap 0x5800-0x5FFF in the Intellivision memory map to point to 0x2200-0x29FF in the Intellicart memory map, do the following:

Set the bankswitch enable bit for 0x5800-0x5FFF when the ROM is downloaded.

Within the Intellivision program write 0x0022 to address 0x0055.

Example 3: To remap 0x5000-0x5FFF in the Intellivision memory map to point to 0x2000-0x2FFF in the Intellicart memory map, do the following.

Set the bankswitch enable bits for both 0x5000-0x57FF and 0x5800-0x5FFF when the ROM image is downloaded.

Within the Intellivision program write 0x0020 to address 0x0045 and 0x0028 to address 0x0055.

Note as in example two that ANY 8 bit value can be stored in the bankswitch offset table, allowing one to use sliding bankswitch memory windows.

Access control to the bankswitched area is set by the enable bits for the original window, regardless of where the window currently points in Intellicart address space. So if you create a ROM location in Intellivision address space 0x5000-0x57FF, and a bankswitched RAM location at 0x6000-0x67FF in Intellivision address space, you can point that window to 0x5000-0x57FF in Intellicart address space

and actually overwrite the ROM image stored there.

Remember, the bankswitch offset table can be modified by the Intellivision program, but the actual enabling or disabling of bank switching for a given window in the Intellivision address space can only be set at the time the ROM image is downloaded.

Intellicart Interface Section:

The Intellicart hardware is designed to receive RS-232 standard serial data signals at 2400, 4800, 9600, 19200, 38400, or 57600 baud. It uses only the transmit (from the computer) and ground lines, and does not use software or hardware flow control. All of the necessary formatting and error detection signals are embedded in the ROM file format described below. Once the ROM file is formed, it should be sent directly to the Intellicart as a raw binary file transfer with No parity, eight (8) data bits, and one (1) stop bit. A raw binary transfer means sending the bytes of the ROM file unmodified directly to the serial port.

Two cables are available from the Intellicart. The first has a DB9 connector and is used for serial communications. Two pins of this connector are used. Pin 3 is the Intellicart serial receive line, and Pin 5 is ground. The second cable provides access to the Intellivision's +5 volt power supply and ground to power any future communication adapters, such as audio to serial conversion. The DC power connector provided is center positive, with internal diameter of 2.5mm, and outside diameter of 5.5mm.

The following text describes the protocol used to send a binary file to the Intellicart. This conversion is performed by the download programs provided with the Intellicart, but the format is presented here so that interface software can be written for platforms not currently supported. At the end of this section, C code will be presented which is a complete binary to ROM file converter. (A ROM file is a file in the format the Intellicart expects to receive data.)

Intellicart Download Protocol:

1. Send Byte 0xA8, used for auto baud rate detection
2. Send number of ROM segments
   This is the number of non-contiguous ROM segments that will follow. For example, a cart that occupied memory from 0x5000-0x6FFF and 0xD000-0xDFFF would send the number 0x02. (The actually binary value, not its ASCII equivalent.)
3. Send the ones complement of the number of ROM segments.
4. Send high byte of start address of first segment, 0x50 for above example.
5. Send high byte of stop address of first segment, 0x6F for above example.
6. Send the ROM segment itself.
7. Send the high byte of the CRC-16 for the first ROM segment.
8. Send the low byte of the CRC-16 for the first ROM segment.
9. Repeat steps 4-8 for each additional ROM segment.
10. Send the Enable Tables defined later.
11. Send high byte of CRC-16 for the tables.
12. Send low byte of CRC-16 for the tables.
13. Done

Enable Tables:

Access Table - Instructs the Intellicart whether or not to respond to read and/or write accesses in each of the 2K word segments of the Intellivision memory map. Also sets whether or not bankswitching is enabled for each 2K window.

Format:

16 Bytes treated as 32 4-bit nibbles each holding permission for a given 2K region.

Intellivision address bits <15:12> index into table. (Zero relative.) Upper nibble contains permissions for upper 2K, lower nibble for lower 2K.

If the nibble is clear Intellicart ignores this address space.
If bit zero of a nibble is set Intellicart responds to read commands (ROM/RAM).
If bit one of a nibble is set Intellicart responds to write commands. (RAM)
If bit three of a nibble is set Intellicart allows this window to be bankswitched.
Bit two is unused.

Example: C array format, ROM at 0x5000-0x5FFF, RAM at 0xD000-0xD7FF

```
unsigned char read_write[16];
     int i;
     for(i = 0; i < 16; i++)
          read_write[i] = 0;
     /*Set read permission for 0x5000-0x57FF and
     0x5800-0x5FFF
     First 1 sets permission for 0x5800-0x5FFF
     Second 1 sets permission for 0x5000-0x57FF*/
     read_write[5] = 0x11;
     /*Set read/write permission for 0xD000-0xD7FF
     A value of 3 in a nibble is read/write (both
     lower bits set.)
     Only enable the LOWER 2K in this example*/
     read_write[0xD] = 0x03;
```

End of Example

Fine Address Restriction Table - Used to restricted access to a smaller portion inside a 2K window, for working around memory holes in the Intellivision, predominantly the STIC registers.
Format:
32 Bytes treated as 64 4-bit nibbles.
All lower 2K windows (0x0000-0x07FF … 0xF000-0xF7FF) are in the first 32 bytes.
All upper 2K windows (0x0800-0x0FFF … 0xF800-0xFFFF) are in the second 32 bytes.

Intellivision address bit 11 (zero relative) determines which half

of the table.
ADDR <11> == 0 —-> table[0] to table[15]
ADDR <11> == 1 —-> table[16] to table[31].

Intellivision address bits <15:12> index into the appropriate table half. Each byte has the following format: [0xxx 0yyy], where 0xxx is the 3 bit LOWER bound (yes, placed in the upper nibble) and 0yyy is the 3 bit UPPER bound. The bounds are inclusive, i.e. 1-7 only restricts access to 0.

Note, for the upper half of the table, xxx and yyy are still limited to be between 0 and 7. For example, in the lower 2K 0x5xxx block, a value of [0001 0003] would restrict access to 0x5100-0x53FF. In the upper 2K block, the same byte would restrict access to 0x5900-0x5B00.

Example: Restrict RAM from Access Table example to only occupy 0xD000-0xD3FF

```
unsigned char fine_restrict[32];
/*Note the default value for this table must be
0x07 to allow access to the entire range opened
by the Enable table.*/
for(i =0; i < 32; i++)
     fine_restrict[i] = [0x07];
/*Limit range 0xD000-0xD7FF to be 0xD000-0xD3FF
Note this is a lower 2K boundary, so it used the
first half of the table.*/
fine_restrict[0xD] = [0x03];
```

End of Example

CRC-16 Format:

The Intellicart CRC calculations can be performed using the following table and algorithms, presented in C code examples. For a standardized reference to the algorithm used, here is the algorithm described in the Rocksoft Model Format:

Name : “Intellicart CRC-16”
Width : 16
Poly : 0x1021
Init : 0xFFFF
RefIn : False
RefOut : False
XorOut : 0000

For more on the Rocksoft Model visit the website at the following URL http://www.repairfaq.org/filipg/LINK/F_crc_v3.html.

Table driven Intellicart CRC-16 algorithm:

```
unsigned short crctable[256] =
{
 0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
 0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
 0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
 0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
 0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
 0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
 0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
 0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
 0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
 0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
 0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
 0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
 0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
 0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
 0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
 0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
 0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
 0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
 0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
 0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
 0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
 0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
 0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
 0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
 0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
 0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
 0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
 0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
 0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
 0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
 0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
 0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
```

```
};

/*Intialize the CRC to the starting value*/
void crc_init(unsigned short *reg)
{
      *reg = 0xFFFF;
}

/*Update the CRC calculation with the latest data*/
void crc_calc(unsigned short *reg, unsigned char x)
{
      *reg = (*reg<<8) ^ crctable[(*reg >> 8) ^ x];
}



/*This code would be used to implement the CRC algorithm, say
in main()*/
void crc_calculation_sample
{
      /*Variable to store the CRC as it is calculated*/
      unsigned short crc;

      /*Sample Data array, assume this was generated by reading
      the ROM section from a .bin file, and that the number of
      bytes of data is data_length*/

      unsigned char data[];
      unsigned int data_length;

      /*A counter variable*/
      unsigned int i;

      /*Initialize the CRC this would be done at the start of
      each ROM segment and at the start of the Enable Tables*/
      crc_init(&crc);

      /*Compute the CRC for the data segment*/
      for(i=0; i < data_length; i++)
            crc_calc(&crc,data[i]);

      /*After this loop, the value stored in CRC is correct,
      and would be sent to the Intellicart after the data[] was
      sent, represented by the dummy function write_data().*/

      write_data(data[]);
      write_crc(outfile, crc);
}
```

Binary to ROM file converter code

Below is example C code to perform binary file to ROM file conversion. It assumes that the binary file has the extension .bin and that any additional information necessary is provided in a configuration file of the same base name as the .bin file, but with the .bin extension replaced with the extension .cfg. The configuration file must be stored in the same directory as the binary file. It creates a ROM file with the same base name as the .bin file, but replaces the .bin extension with .rom. This ROM file is ready for binary download to the Intellicart.

This code was specifically developed for a PC with a 32 bit operating system, but should be easy to port to other platforms. The configuration file parser included is of a VERY basic nature, and is not robust to even small changes in the configuration file format and is included only as a starting point.

To run the program, use the command line and supply as the argument the base name of the binary file, without the .bin extensions.

Start of Code:

==========

```
/*
Intellicart Binary to ROM file conversion Program
Chad Schell - 6/19/2000

This code may be freely distributed and modified for use in the
creation of new programs
for communication with the Intellicart.

Use of this code for other purposes in ok, but it's pretty
clunky and you can probably find a
better set of code to start from.
*/


#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/* Declare structire used by parser for storing configuration
file information */

typedef struct ROM_SETUP
{
   unsigned int Number_Sections;
   unsigned short Boundaries[40];
   unsigned char Enable_Table[16];
   Unsigned char Fine_Range[32];
} ROM_SETUP;

/*Function Declarations*/
void write_tables(FILE *outfile, ROM_SETUP *setup);
void write_crc(FILE *outfile, unsigned short crc);
void crc_init(unsigned short *reg);
void crc_calc(unsigned short *reg, unsigned char x);
void create_file_names(char *romname, char *cfgname, char *outname);
int Parse_Config_File(FILE *Cfg, ROM_SETUP *setup);
int Create_ROM_SEND_FILE(char *romname, char *outname);
void ROM_TableUpdate(ROM_SETUP *setup, unsigned short
start_addr, unsigned short stop_addr, unsigned char type);

unsigned short crctable[256] =
{
 0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
 0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
 0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
 0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
```

```
 0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
 0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
 0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
 0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
 0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
 0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
 0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
 0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
 0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
 0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
 0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
 0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
 0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
 0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
 0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
 0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
 0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
 0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
 0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
 0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
 0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
 0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
 0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
 0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
 0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
 0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
 0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
 0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};

/*Main's only purpose is to read the command line and call the
rest of the code*/
int main(int argc, char *argv[])
{
   char romname[200],outname[200];

/*Must supply a binary file name, and nothing else.*/

   if(argc < 2) exit(1);
   if(argc > 2) exit(1);

/*Generate binary file name - add .bin extension*/
   strcpy(romname,argv[1]);
   strcat(romname,".bin");

/*Create the ROM file*/
   return Create_ROM_Send_File(romname,outname);
}
```

```
/*The bulk of the code. Performs the actual conversion of the
binary file to the ROM file*/
int Create_ROM_Send_File(char *romname, char *outname)
{
   /*Mapping represents that ROM sections were declared in a
configuration file*/
   /*A zero means that they were not declared*/
   int mapping = 0;
   char cfgname[200];
   FILE *romfile, *configfile, *outfile;
   ROM_SETUP setup;
   unsigned int i;
   int j;
   unsigned char temp;
   int segment_size;
   long filesize;

   unsigned short crc;


   /*Initlialize Enable Tables*/
   for(i = 0; i < 16; i++)
      setup.Enable_Table[i] = 0;
   for(i = 0; i < 32; i++)
   {
      setup.Fine_Range[i] = 0x07;
   }

   /*Create the configuration and ROM file names*/
   create_file_names(romname, cfgname, outname);


   /*Open the binary file - abort if it doesn't exist*/
   if((romfile = fopen(romname,"rb"))== NULL)
   {
      return(1);
   }

   /*Open the ROM file for writing - overwrite any existing ROM
file of the same name*/
   if((outfile = fopen(outname,"wb"))==NULL)
   {
      return(1);
   }

   /*This is necessary for auto baud rate detection*/
   fputc(0xA8,outfile);

   /*Is there a configuration file?*/
   if((configfile = fopen(cfgname,"r"))!=NULL)
```

```
    {
        /*Yes - parse it*/
        mapping = Parse_Config_File(configfile, &setup);
        fclose(configfile);
    }

    /*Did a configuration file with ROM segments exist?*/
    if(mapping != 0)
    {
        /*Yes - copy those segments from the binary file to the
ROM file*/
        /*Write the number of ROM segments that will be in the
ROM file*/
        fputc(setup.Number_Sections,outfile);
        /*Write it's one's complement for error detection*/
        fputc(~setup.Number_Sections,outfile);

        /*Write the ROM segments one a time*/
        for(i = 0; i < setup.Number_Sections; i++)
        {
            /*Initialize the CRC for each new ROM segment*/
            crc_init(&crc);

            /*Write the start and stop address of the ROM seg-
ment, and update the CRC*/
            temp = setup.Boundaries[i*2] / 0x100;
            crc_calc(&crc,temp);
            fputc(temp, outfile);
            temp = setup.Boundaries[i*2+1] / 0x100;
            crc_calc(&crc,temp);
            fputc(temp,outfile);

            /*Determine the size of the ROM segment, copy it to
the ROM file, and update
            the CRC with each new byte of data*/
            segment_size = 2*( setup.Boundaries[i*2+1]-
setup.Boundaries[i*2]+1);
            for(j = 0; j < segment_size; j++)
            {
                fread(&temp,1,1,romfile);
                fputc(temp, outfile);
                crc_calc(&crc,temp);
            }
            /*Write the CRC for this segment*/
            write_crc(outfile, crc);
        }
        /*Write the enable tables - these were created while
parsing the configuration file*/
        write_tables(outfile, &setup);
    }
```

```
    else
    {
        /*No ROM segments were defined in the configuration file.
Attempt to fit the binary
        file to one of the standard cartridge layouts. Update
the Enable tables accordingly*/

        /*Determine file size*/
        fseek(romfile,0,SEEK_END);
        filesize = ftell(romfile);
        fseek(romfile,0,SEEK_SET);
        if(filesize == 0x2000) /*4K Word ROM*/
        {
            fputc(1,outfile);
            fputc((unsigned char)(~1),outfile);
            crc_init(&crc);
            fputc(0x50,outfile);
            crc_calc(&crc,0x50);
            fputc(0x5F,outfile);
            crc_calc(&crc,0x5F);
            for(j = 0; j < 8192; j++)
            {
                fread(&temp,1,1,romfile);
                fputc(temp, outfile);
                crc_calc(&crc,temp);
            }
            write_crc(outfile, crc);
            /*4K Word carts occupy from 0x5000-0x5FFF*/
            ROM_Table_Update(&setup, 0x5000, 0x5FFF, 0x01);
            write_tables(outfile, &setup);
        }
        if(filesize == 0x4000) /*8K Word ROM*/
        {
            fputc(1,outfile);
            fputc((unsigned char)(~1),outfile);
            crc_init(&crc);
            fputc(0x50,outfile);
            crc_calc(&crc,0x50);
            fputc(0x6F,outfile);
            crc_calc(&crc,0x6F);
            for(j = 0; j < 16384; j++)
            {
                fread(&temp,1,1,romfile);
                fputc(temp, outfile);
                crc_calc(&crc,temp);
            }
            write_crc(outfile, crc);
            ROM_Table_Update(&setup, 0x5000, 0x6FFF, 0x01);
            write_tables(outfile, &setup);
```

```
}
if(filesize == 0x6000) /*12K Word ROM*/
{
    fputc(2,outfile);
    fputc((unsigned char)(~2),outfile);
    crc_init(&crc);
    fputc(0x50,outfile);
    crc_calc(&crc,0x50);
    fputc(0x6F,outfile);
    crc_calc(&crc,0x6F);
    for(j = 0; j < 16384; j++)
    {
        fread(&temp,1,1,romfile);
        fputc(temp, outfile);
        crc_calc(&crc,temp);
    }
    write_crc(outfile, crc);
    ROM_Table_Update(&setup, 0x5000, 0x6FFF, 0x01);
    crc_init(&crc);
    fputc(0xD0,outfile);
    crc_calc(&crc,0xD0);
    fputc(0xDF,outfile);
    crc_calc(&crc,0xDF);
    for(j = 0; j < 8192; j++)
    {
        fread(&temp,1,1,romfile);
        fputc(temp, outfile);
        crc_calc(&crc,temp);
    }
    write_crc(outfile, crc);
    ROM_Table_Update(&setup, 0xD000, 0xDFFF, 0x01);
    write_tables(outfile, &setup);
}
if(filesize == 0x8000) /*16K Word ROM*/
{
    fputc(3,outfile);
    fputc((unsigned char)(~3),outfile);
    crc_init(&crc);
    fputc(0x50,outfile);
    crc_calc(&crc,0x50);
    fputc(0x6F,outfile);
    crc_calc(&crc,0x6F);
    for(j = 0; j < 16384; j++)
    {
        fread(&temp,1,1,romfile);
        fputc(temp, outfile);
        crc_calc(&crc,temp);
    }
    write_crc(outfile, crc);
    ROM_Table_Update(&setup, 0x5000, 0x6FFF, 0x01);
```

```
            crc_init(&crc);
            fputc(0xD0,outfile);
            crc_calc(&crc,0xD0);
            fputc(0xDF,outfile);
            crc_calc(&crc,0xDF);
            for(j = 0; j < 8192; j++)
            {
               fread(&temp,1,1,romfile);
               fputc(temp, outfile);
               crc_calc(&crc,temp);
            }
            write_crc(outfile, crc);
            ROM_Table_Update(&setup, 0xD000, 0xDFFF, 0x01);

            crc_init(&crc);
            fputc(0xF0,outfile);
            crc_calc(&crc,0xF0);
            fputc(0xFF,outfile);
            crc_calc(&crc,0xFF);
            for(j = 0; j < 8192; j++)
            {
               fread(&temp,1,1,romfile);
               fputc(temp, outfile);
               crc_calc(&crc,temp);
            }
            write_crc(outfile, crc);
            ROM_Table_Update(&setup, 0xF000, 0xFFFF, 0x01);
            write_tables(outfile, &setup);
         }

         /*No ROM segments were defined in a configuration file,
so if it was not a standard
         file size, there is an error with the binary file*/
         if(( (filesize == 0x2000) || (filesize == 0x4000) ||
(filesize == 0x6000)
            || (filesize == 0x8000) ) == 0)
         {
            printf("Bad File size\n");fflush(stdout);
            fclose(romfile);
            fclose(outfile);
            return(1);
         }
      }
      /*Close the files*/
      fclose(romfile);
      fclose(outfile);

      /*Done*/
      return 0;
```

```
}

/*This routine writes the CRC value to the ROM file in the
proper format*/
void write_crc(FILE *outfile, unsigned short crc)
{
    int temp;
    temp = (crc & 0xFF00) >> 8;
    fputc(temp, outfile);
    temp = (crc & 0xFF);
    fputc(temp,outfile);
    return;
}

/*This routine writes the Enable tables to the ROM file in the
proper format.
It also calculates their CRC and writes that to the ROM file as
well.*/
void write_tables(FILE *outfile, ROM_SETUP *setup)
{
    unsigned short crc;
    int j;
    crc_init(&crc);
    for(j = 0; j < 16; j ++)
    {
        crc_calc(&crc,setup->Enable_Table[j]);
    }
    fwrite(setup->Enable_Table,1,16,outfile);
    for(j = 0; j < 32; j ++)
    {
        crc_calc(&crc,setup->Fine_Range[j]);
    }
    fwrite(setup->Fine_Range,1,32,outfile);
    write_crc(outfile, crc);
}

/*A VERY simple parser to read configuration files - NOT robust
at all.
Spacing in commands must EXACTLY match the patterns specified*/
int Parse_Config_File(FILE *Cfg, ROM_SETUP *setup)
{
//  int i,j,min;
    char line[200];
    unsigned short rom1,rom2,addr;
    int mapping = 0;
    setup->Number_Sections = 0;

    while(fgets(line,200,Cfg)!=NULL)
    {
        /*[mapping] – defines ROM segments, sets read access to
```

```
those segments
       Addresses must follow in the format $xxxx - $yyyy =
$zzzz
       No comment only lines can appear between [mapping] and
the commands*/

       if(!strncmp(line,"[mapping]",9)) /* mapping line detect-
ed*/
       {
          mapping = 1;
          while(fgets(line,200,Cfg)!=NULL)
          {
             if(line[0] == '$') /*Address command line*/
             {
                sscanf(line,"$%x - $%x =
$%x",&rom1,&rom2,&addr);
                setup->Boundaries[(setup->Number_Sections)*2] =
addr;
                setup->Boundaries[(setup->Number_Sections)*2+1]
= rom2-rom1+addr;
                /*Modify enable tables for read access*/
                ROM_Table_Update(setup,
                   setup->Boundaries[(setup-
>Number_Sections)*2],
                   setup->Boundaries[(setup-
>Number_Sections)*2+1],0x01);
                setup->Number_Sections++;
             }
             else
                break;
          }
       }
       /*[Preload] – same as [mapping] but does not set Enable
tables for read access*/
       if(!strncmp(line,"[preload]",9)) /* preload line detect-
ed*/
       {
          mapping = 1;
          while(fgets(line,200,Cfg)!=NULL)
          {
             if(line[0] == '$') /*Address command line*/
             {
                sscanf(line,"$%x - $%x =
$%x",&rom1,&rom2,&addr);
                setup->Boundaries[(setup->Number_Sections)*2] =
addr;
                setup->Boundaries[(setup->Number_Sections)*2+1]
= rom2-rom1+addr;
                setup->Number_Sections++;
             }
```

```
            else
                break;
        }
    }
    /*[memattr] – used to define RAM areas. Sets read and
write access for these areas
    Addresses must follow in the format $xxxx -$yyyy = $zzzz
    Note the lack of a space after the hyphon. This matches
the existing .cfg files,
    and this is an example of how unrobust this parser is*/
    if(!strncmp(line,"[memattr]",9)) /* memattr line detect-
ed*/
    {
        while(fgets(line,200,Cfg)!=NULL)
        {
            if(line[0] == '$') /*Address command line*/
            {
                sscanf(line,"$%x -$%x = RAM
%d",&rom1,&rom2,&addr);
                /*Modify Tables to enable RAM in this region*/
                /*There is no provision for 8 vs. 16 bit
RAM.*/
                ROM_Table_Update(setup, rom1, rom2, 0x03);
            }
            else
                break;
        }
    }
    /*[bankswitch] – used to enable bankswitching on a given
window,
    also enables read access to that window.
    Addresses must follow in the format $xxxx - $yyyy*/
    if(!strncmp(line,"[bankswitch]",9)) /* bankswitch line
detected*/
    {
        while(fgets(line,200,Cfg)!=NULL)
        {
            if(line[0] == '$') /*ddress command line */
            {
                sscanf(line,"$%x - $%x",&rom1,&rom2);
                /*Load Bankswitch Table with starting address
and modify enable tables*/
                ROM_Table_Update(setup, rom1, rom2, 0x09);
            }
            else
                break;
        }
    }
```

```
    }

    if(mapping == 0)
        return 0;
    else
        return 1;
}

//This code handles the address convolution to place values in
the Enable tables.
void ROM_Table_Update(ROM_SETUP *setup, unsigned short
start_addr,
    unsigned short stop_addr, unsigned char enable_bits)
{
    unsigned short addr;
    unsigned char high_addr, index;
    addr = start_addr;
    while(addr < stop_addr)
    {
        high_addr = (addr & 0xFF00) >> 8;
        index = (high_addr & 0xF0) >> 4;
        if((high_addr & 0x08) == 0)
        {
            setup->Enable_Table[index] |= enable_bits;
        }
        else
        {
            setup->Enable_Table[index] |= (enable_bits << 4);
        }
        addr += 0x0800;
        if(addr == 0) break;
    }
    /*Do Fine Range Restrictions if Necessary*/
    /*Start Address*/
    addr = (start_addr & 0x0700) >> 8;
    if(addr != 0)
    {
        high_addr = (start_addr & 0xFF00) >> 8;
        index = (high_addr & 0xF0) >> 4;
        if((high_addr & 0x08) == 0)
        {
            setup->Fine_Range[index] |= (addr << 4);
        }
        else
        {
            setup->Fine_Range[index+16] |= (addr << 4);
        }
    }
    /*Stop Address*/
    addr = (stop_addr & 0x0700) >> 8;
```

```
    if(addr != 7)
    {
        high_addr = (stop_addr & 0xFF00) >> 8;
        index = (high_addr & 0xF0) >> 4;
        if((high_addr & 0x08) == 0)
        {
            setup->Fine_Range[index] &= 0xF0;
            setup->Fine_Range[index] |= addr;
        }
        else
        {
            setup->Fine_Range[index+16] &= 0xF0;
            setup->Fine_Range[index+16] |= addr;
        }
    }

}

/*Creates the .rom and .cfg filenames from the .bin filename*/
void create_file_names(char *romname, char *cfgname, char *out-
name)
{
    int namesize = strlen(romname);
    strncpy(cfgname,romname,namesize-3);
    strncpy(outname,romname,namesize-3);
    cfgname[namesize-3] = '\0';
    outname[namesize-3] = '\0';
    strcat(cfgname,"cfg");
    strcat(outname,"rom");
}

/*Initializes the CRC to it's starting value*/
void crc_init(unsigned short *reg)
{
    *reg = 0xFFFF;
}

/*Computes the CRC, using the CRC table at the top of this
code*/
void crc_calc(unsigned short *reg, unsigned char x)
{
    *reg = (*reg<<8) ^ crctable[(*reg >> 8) ^ x];
}
```

==========

End of Code.

Legal notices:

Acknowledgements:

Many, many, many thanks to Joe Zbiciak for his help in developing the Intellicart. Without his assistance this project would never have been completed. He provided all of the Intellivision code for working out the bugs in the Intellicart, as well as providing assistance in many other aspects of the development. He deserves much of the credit for making the Intellicart the product that it is.

Thanks to Tim Lindner for formatting this manual, and developing and testing the Mac version of the Intellicart download software.

Thanks to Jason Willis for developing the PC version of the Intellicart download software.

Thanks to Chris Neiman for testing the Intellicart with all of the rare games he owns and plays.

Thanks to Doug Parsons and William Moeller for providing Intellivision background information, encouragement, and sanity checks along the way.