SECTION III

GRAPHICS GENERATION SOFTWARE

The graphics generation process is structured on three levels of software. A typical application will use routines from all three levels. These are the chip driver level, the table level and the object level. Figure 3-1 shows the program flow, software structure and its relationship with the outside world.

3.1 Chip Driver Level

The graphics hardware consists of the VDP and 16K VRAM. The VDP has eight write-only control registers and one read-only status register. The chip driver level software interfaces with the VDP registers and VRAM through the VDP. For detailed configuration of the registers, refer to the TMS 9928A VDP Data Manual.

The chip driver level software consists of six subroutines:

READ_VRAM, WRITE_VRAM, READ_REGISTER, WRITE_REGISTER, FILL_VRAM and MODE_1. The first five routines allow programs to access the VDP registers and transfer information to and from VRAM blocks. The sixth routine, MODE_1, initializes the VDP into a standard configuration.

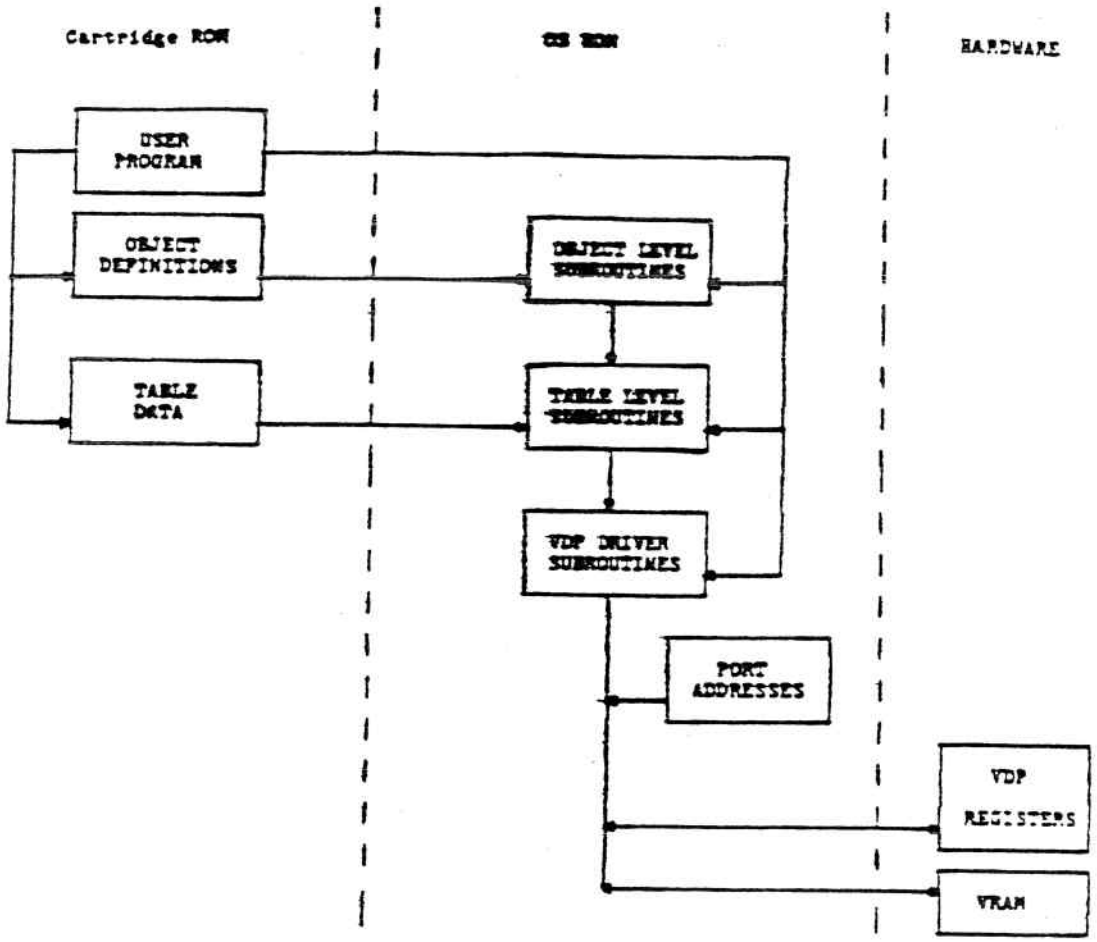

Figure 3-1
OS Graphics Software/VDP Interface

3.1.1. READ_VRAM

Calling Sequence:

```
LD    HL, BUFFER
LD    DE, SRCE
LD    BC, COUNT
CALL  READ_VRAM
```

Description:

READ_VRAM reads COUNT bytes from VRAM starting at SRCE and puts them in BUFFER.

Parameters:

BUFFER	This is the starting address of a CRAM buffer which is to receive the data read from VRAM.
SRCE	VRAM starting address to be read from.

COUNT Number of bytes to be read from VRAM.

Side Effects:

- Destroys AF, BC, DE and HL.
- Cancels any previously initiated VDP operations.

3.1.2 WRITE_VRAM

Calling Sequence:

```
LD    HL, BUFFER
LD    DE, DEST
LD    BC, COUNT
CALL  WRITE_VRAM
```

Description:

WRITE_VRAM takes COUNT bytes from BUFFER and sends them through the VDP to VRAM. The starting address in VRAM for the write operation is given as DEST.

Parameters:

BUFFER	This is the starting address of a buffer where data to be sent to the VDP is located.

DEST	This is the VRAM address where the data is to be sent.
COUNT	This is the number of bytes that are to be transferred to VRAM. Count should be either less than 256 (100H) or even multiples of 256. (Ref. ColecoVision Bulletin No. 0002).

Side Effects:

- Destroys AF, BC, DE and HL.
- Cancels any previously initiated VDP operations.

3.1.3 READ_REGISTER

Calling Sequence:

```
CALL  READ_REGISTER
```

Description:

READ_REGISTER reads and returns the contents of the VDP status register in the accumulator. This value should be stored at VDP_STATUS_BYTE in CRAM. The information in this register can only be guaranteed valid during the vertical retrace time.

Return value:

Returns the contents of the VDP status register which has the following form (see VDP manual for further details):

Bit 7	Bit 6	Bit 5	Bits 4..0
Interrupt	Fifth Sprite	Coincidence	Fifth Sprite No.

Figure 3-2

VDP Status Register

Side Effects:

This routine has no effect at all in the processor memory or register space. However, a status read has a significant side effect to the VDP.

It acts as an interrupt acknowledge operation, i.e., it clears the interrupt flag and enables further generation of interrupts.

Thise side effect must be treated with care for two reasons. First of all, as is pointed out in the VDP manual, asynchronous reads may cause the interrupt flag in the status register to be reset before it is detected; this may cause problems in systems that expect to perform synchronization using the interrupt flag.

The second reason concerns interrupts which halt the execution of routines while they are accessing VRAM. In order to re-enable interrupts, a service routine must read the status register. However, to prevent the NMI from re-interrupting the service routine, the user should avoid reading the status register until all of its work is done. A defer interrupt routine, DEF_INT, has been developed to assist the user in handling this situation. Refer to ColecoVision Bulletin No. 0010 for additional information.

3.1.4 WRITE_REGISTER

Calling Sequence:

```
LD    B, REGISTER
LD    C, VALUE
CALL  WRITE_REGISTER
```

Description:

WRITE_REGISTER takes VALUE and writes it to the VDP register numbered REGISTER.

WRITE_REGISTER also maintains two bytes in CRAM starting at address VDP_MODE_WORD. The first is intended to duplicate the current contents of VDP Register 0, and the second to duplicate Register 1. When writing to a register using WRITE_REGISTER, the appropriate half of VDP_MODE_WORD is updated.

Parameters:

REGISTER	This is the VDP register number (0 - 7) to be written.
VALUE	This is the value to be written to REGISTER.

Side Effects:

- Destroys the AF register pair.

3.1.5 FILL_VRAM

Calling Sequence:

```
LD    HL, ADDRESS
LD    DE, COUNT
LD    A, VALUE
CALL  FILL_VRAM
```

Description:

FILL_VRAM writes COUNT copies of VALUE to VRAM starting at ADDRESS.

Parameters:

ADDRESS	VRAM address to start fill operation.
COUNT	Number of bytes to fill.

VALUE 8-bit value to fill with.

Side Effects:

- Destroys AF and DE.
- Cancels all previously initiated VRAM operations.

Calls to other OS routines:

- READ_REGISTER (Ref. Sec. 3.1.3)

3.1.6 MODE_1

Calling Sequence:

```
CALL  MODE_1
```

Description:

MODE_1 sets the VDP to graphics mode 1 and sprite size 0. It also uses the INIT_TABLE routine to define the VRAM table addresses as follows:

- Sprite Generator Table	- 3800H
- Patter Color Table	- 2000H
- Sprite Attribute Table	- 1B00H
- Pattern Name Table	- 1800H
- Pattern Generator Table	- 0000H

When MODE_1 returns, the screen is blanked and the backdrop plane color is set to black.

Side Effects:

- Destroys AF, BC and HL.

Calls to other OS routines:

- WRITE_REGISTER
- INIT_TABLE

3.2 Table Level

The VDP requires various table areas within VRAM to operate. These tables are interrelated, each controlling its own aspect of the graphics generation process. The table level software provides routines which will read or write VRAM with respect to these table areas. The routines also provide the capability of reading and writing entire tables entries or sections of these entries up to and including the whole table. This level also has special functions which were found helpful.

The major difference between the table level and the chip driver level is that the applications programmer is no longer required to manipulate VRAM addresses on the table level. Instead, each of the VRAM tables is assigned a number or table code as listed in Table 3-1.

Table Name	Code
Sprite attribute table	0
Sprite generator table	1
Pattern name table	2
Pattern generator table	3
Pattern color table	4

Table 3-1

VRAM Table Code

When an applications program needs to operate on a table, only a table code needs to be passed to the applicable table processing the routine.

Furthermore, in graphics mode 1 and graphics mode 2, which are supported by the OS graphics software, the tables have more or less fixed shapes. The entry numbers and bytes per entry for each of the five tables, as well as their boundaries, is given in Table 3-2.

TABLE CODE	MODE(S)	ENTRIES	BYTES/ ENTRY	HEX EQUIVALENTS
0	1 & 2	32	4	80H
1	1 & 2	256	8	800H
2	1 & 2	768	1	400H
3	1	256	8	800H
3	2	768	8	800H
4	1	32	1	40H
4	2	768	8	2000H

Table 3-2
Table Entries and Boundaries

The table management software takes advantage of this regularity by letting application programs address table entries as integral entities. Let us take, for example, the task of getting the 14th sprite attribute entry from VRAM. In terms of the chip driver software, the task appears as follows:

- Get sprite attribute table address.
- Calculate offset into table (14 * table row length).
- Add offset to address.
- Read one table entry (4 bytes) from VRAM at off set + attribute table address.

On the other hand, when using the table level software, the task is now reduced to the following:

- Give offset into table (14).
- Give table code.
- Give item count (1).
- Call GET_VRAM (places the desired bytes at a user-defined area).

In a video program that requires accessing the sprite attribute table frequently (for example, an action-oriented game), the table level method constitutes a significant savings in cartridge code.

Software in the table level may be further subdivided into three groups of routines as follows:

- Table Managers
- Table-oriented Graphics Routines
- Sprite Reordering Software

3.2.1 Table Managers

There are three routines in this group: INIT_TABLE, GET_VRAM and PUT_VRAM. As the names imply, they deal with table initialization, getting data from tables and placing data into tables, respectively.

Table initialization is a very simple operation which involves assigning a base address to a table. The base addresses are "saved" for later use by GET_VRAM and PUT_VRAM for address calculations and remain fixed until they are reinitialized. GET_VRAM and PUT_VRAM both take a table code, an entry number, as well as an element count and a buffer address in CRAM as parameters when they perform their respective transfers of information between CRAM and VRAM.

3.2.1.1 INIT_TABLE

Calling Sequence:

```
LD    A, TABLE_CODE
LD    HL, ADDRESS
CALL  INIT_TABLE
```

Description:

INIT_TABLE takes a table code and a VRAM address at which that table is to reside, and initializes the VDP base address register for the given table. It also stores the unconverted form of the address in an array called VRAM_ADDR_TABLE for later use in address arithmetic. This address is stored at VRAM_ADDR_TABLE [TABLE_ CODE].

INIT_TABLE makes use of the current graphics mode in determining the actual value written to the base address register in some cases. It determines the graphics mode

by looking at the VDP_MODE_WORD. Thus, it is imperative that the graphics mode be set up using WRITE_ REGISTER before INIT_TABLE.

Parameters:

TABLE_CODE	Number of the table to be initialized. TABLE_CODE must be one of the legal table codes defined in (Table 3-1).
ADDRESS	Intended VRAM address of table. Each table has its own boundary defined by the table base address in the VDP control register. The user should refer to Table 3-2 for the proper table boundary.

Side Effects:

- Destroys AF, BC, HL, IX and IY.

Calls to other OS routines:

- REG_WRITE

3.2.1.2 GET_VRAM

Calling Sequence:

```
LD    A, TABLE_CODE
LD    DE, START_INDEX
LD    HL, DATA
LD    IY, COUNT
CALL  GET_VRAM
```

Description:

GET_VRAM reads into the CRAM buffer DATA, COUNT entries from the table specified by TABLE_CODE, which starts at the table entry number START_INDEX.

GET_VRAM uses the VDP_MODE_WORD and VRAM_ADDR_TABLE to calculate VRAM addresses and byte counts. It is imperative, before calling GET_VRAM, that the graphics mode be initialized using WRITE_REGISTER, and that the table being accessed be initialized using INIT_TABLE.

Parameters:

TABLE_CODE — VRAM table code (Table 3-1) to be read.

START_INDEX — START_INDEX is a two-byte number that indicates the starting entry of the table.

The range of START_INDEX is table dependent. However, no boundary checking is done; therefore, if an index is given that is outside the range of the table, but still a legal VRAM address, the specified number of "entries" will be extracted from that location in VRAM.

Both the pattern generator and the color tables in graphics mode 2 are 768 entries long and they are