Hitch-Hacker's Guide to the Atari Digital Vector Generator

Philip Pemberton
http://www.philpem.me.uk/

December 4, 2018

1 Introduction

Over the past few years, interest in the emulation of video games has been increasing rapidly. Modern computer hardware has allowed arcade games like Asteroids and Star Wars to be accurately simulated on a commonly-available home computer system. Although Asteroids was one of the first games to be emulated, very little documentation exists on the hardware, beyond the memory maps and notes in the MAME source code.

One of the more unusual parts of the Asteroids hardware is the Digital Vector Generator, shown on Sheet 2A of the Asteroids schematic set. Outside of Atari, very little documentation exists on it. In this article, I will explain how the DVG works, and how to write code for it.

This article is intended to supplement the notes included in the Asteroids schematics, not replace them. It also concentrates more on the programming side of the DVG, rather than attempting to explain the complexities of the DVG circuitry.

2 Thanks and Acknowledgements

Thanks are due to the following people, who assisted (either directly or indirectly) in the creation of this document:

- **Jed Margolin** (`http://www.jmargolin.com`) — Wrote the article "The Secret Lives of Vector Generators", which explains some of the inner workings of the Atari AVG and DVG. He also explained (in an e-mail) how the DVG represents coordinates.
- **Howard Delman** (`http://www.rawbw.com/~delman/`) — Designer of the DVG. Answered a few technical questions on the DVG hardware.
- **Chris Pile** (`http://web.archive.org/*/http://members.tripod.com/asteroids/`) — Documented part of the DVG, and published his notes online (filename: `roidinfo.zip`). Note that the original site has ceased to exist, hence the link to the Web Archive mirror of the same page.

- **Eric Smith** (`http://www.brouhaha.com/~eric/` — Released the source code to the VECSIM emulator, which was used to get a basic overview of the DVG instruction set.

1. Change log

- **18 September 2006**: Added documentation on the '00h' instruction opcode.

3 Documentation conventions

1. Number formats

Format	Type	Examples
`h` suffix	Hexadecimal (base 16) number	`02h`, `FDh`
`d` suffix	Decimal (base 10) number	`10d`, `93d`
`b` suffix	Binary (base 2) number	`0100b`, `0111 0101b`

2. Typesetting conventions

Format	Type	Examples
`Typewriter`	Part designation, signal, instruction code	`DMALD`, `DVX[11..0]`

4 So what is the DVG?

The Digital Vector Generator — or DVG — is a custom-designed CPU, built entirely from small-scale TTL ICs. It has an architecture totally unlike that of any CPU that existed at the time, and was designed specifically to drive vector-beam monitors. This was done because no CPU available at the time Asteroids was designed had enough power to manage game logic and draw vectors at the same time. Offloading the task of drawing the display on to the DVG allows the CPU to be dedicated to the task of running the game logic.

The DVG features:

- 12-bit program counter, with 13-bit address space
- Four-level stack
- State machine microsequencer with eight micro-instructions
- Vector timer
- Two 12-bit binary rate multipliers to vary the timing relationship between the X and Y vectors
- 16-level brightness control
- 1024x1024 display resolution

5 Coordinate System

Absolute coordinates in DVG terms are expressed as binary numbers from 0 to 1023 in the form `(x, y)`, where `(0, 0)` is the top-left of the screen area, `(511, 511)` is the centre of the screen, and `(1023, 1023)` is the bottom-right of the screen area.

Relative coordinates are expressed in sign-magnitude form, and are processed using binary normalisation to ensure that the vectors they represent are drawn as fast as possible. Sign-magnitude form stores the absolute value of the number in the least significant bits, and the sign in the most significant bit. The sign bit on the DVG is high when a number is negative. As a consequence of this, there are two ways to represent zero. Relative coordinates are used by all the vector drawing instructions.

Absolute coordinates are expressed in two's complement form. When the most significant bit is set, the number is taken to be negative. To convert a negative number back into its absolute value, invert all the bits in the word and add one to the result. Absolute coordinates are only used by the `LABS` instruction.

6 Reset Sequence

When the DVG's $\overline{\text{RESET}}$ input goes active, the following actions are performed by the DVG circuitry at roughly the same time:

- Instruction latch `F7` and Y offset latch `H7` are cleared to zero
- State flip-flop `D8` is cleared to zero
- `HALT` flip-flop `A9` is set; `HALT` and $\overline{\texttt{HALT}}$ go active

The DVG will hold in state 0 until the 6502 asserts the $\overline{\texttt{DMAGO}}$ line. This resets the `HALT` flip-flop, causing `HALT` and $\overline{\texttt{HALT}}$ to be deasserted, thus releasing the halt state and allowing the state machine to begin executing opcodes from the vector instruction memory.

Once the DVG is released from the halt state, it will execute a `DMALD` micro-instruction, which loads the program counter from `DVY[11..0]`. As DVY has been cleared to zero during reset, the program counter will be set to zero.

After the program counter has been reset, the state machine will load the instruction latch, then the `DVY` latch. The program counter will be incremented after the `DVY` latch has been loaded. If an opcode for a two-word instruction has been loaded into the instruction latch, the `SCALE` and `DVX` latches will also be loaded from the vector program memory, and the program counter will be incremented again.

7 Opcodes

VCTR — Opcodes 00h – 09h — Draw Long Vector

Word 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP[3..0]				0	Y_S	Y[9..0]									

Word 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Z[3..0]				0	X_S	X[9..0]									

Operands:

OP[3..0] Opcode nibble.
Y_S Y axis sign bit.
Y[9..0] Y axis position.
X_S X axis sign bit.
X[9..0] X axis position.
Z[3..0] Vector intensity.

Draws a vector from the current X and Y position to the position specified by `X[9..0]` and `Y[9..0]`, with the intensity `Z[3..0]`. X and Y are binary normalised, and the opcode value specifies the number of bits they have been shifted.

The following table illustrates how the length divisor and bit shift count relate to the opcode value:

Opcode	Divisor	Bits shifted
09h	1	0
08h	2	1
07h	4	2
06h	8	3
05h	16	4
04h	32	5
03h	64	6
02h	128	7
01h	256	8
00h	512	9

For example, if a vector is drawn using opcode 09h, it will be drawn at the exact length specified in the VCTR instruction. If that same vector is drawn again using opcode 08h, it will be drawn half as long. The same rule applies to all the other VCTR opcodes.

LABS

Opcode 0Ah
Load Absolute

Word 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP[3..0]				0	Y_S	Y[9..0]									

Word 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SF[3..0]				0	X_S	X[9..0]									

Operands:

OP[3..0] Opcode nibble.
Y_S Y axis sign bit.
Y[9..0] Y axis position.
X_S X axis sign bit.
X[9..0] X axis position.
SF[3..0] Global scale factor.

Sets the X and Y position counters to the values stored in `X[9..0]` and `Y[9..0]` respectively, and sets the global scale factor to the value stored in `SF[3:0]`.

The global scale factor is applied to all vectors drawn after the LABS instruction is executed, and decodes as follows:

Global scale	Scale factor	
1111b	/ 2	(divide by 2)
1110b	/ 4	(divide by 4)
1101b	/ 8	(divide by 8)
1100b	/ 16	(divide by 16)
1011b	/ 32	(divide by 32)
1010b	/ 64	(divide by 64)
1001b	/ 128	(divide by 128)
1000b	/ 256?	**(unconfirmed)**
0000b	Zero?	**(unconfirmed)**
0001b	* 2	(multiply by 2)
0010b	* 4	(multiply by 4)
0011b	* 8	(multiply by 8)
0100b	* 16	(multiply by 16)
0101b	* 32	(multiply by 32)
0110b	* 64	(multiply by 64)
0111b	* 128	(multiply by 128)

When a vector is drawn, the global scale factor is used to provide additional scaling. In the case of Asteroids, this means that only one asteroid image has to be stored in memory; the global scale factor allows that image to be expanded or contracted as required.

HALT — Opcode 0Bh — Halt

Word 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP[3..0]				0	0	0	0	0	0	0	0	0	0	0	0

Operands:

OP[3..0] Opcode nibble.

Halts the vector generator and blanks the screen.

JSRL — Opcode 0Ch — Jump to subroutine

Word 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP[3..0]				A[11..0]											

Operands:

OP[3..0] Opcode nibble.
A[11..0] Target address.

Pushes the current value of the program counter onto the stack and sets the program counter to the address stored in `A[11..0]`.

RTSL — Opcode 0Dh — Return from subroutine

Word 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP[3..0]				0	0	0	0	0	0	0	0	0	0	0	0

Operands:

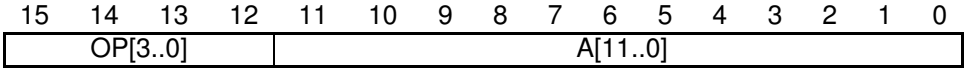
OP[3..0] Opcode nibble.

Pulls an address off of the stack and loads it into the program counter.

JMPL

Opcode 0Eh
Jump

Word 0



Operands:

OP[3..0] Opcode nibble.
A[11..0] Target address.

Sets the program counter to the address stored in `A[11..0]` but, unlike `JSRL`, does not push the return address on to the stack.

SVEC — Opcode 0Fh — Draw Short Vector

Word 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP[3..0]				SF0	Y_S	Y[1..0]		Z[3..0]				SF1	X_S	X[1..0]	

Operands:

- **OP[3..0]** Opcode nibble.
- **Y[1..0]** Y delta.
- **Y_S** Y sign.
- **X[1..0]** X delta.
- **X_S** X sign.
- **Z[3..0]** Vector intensity.
- **SF[1..0]** Scale factor.

Draws a vector from the current X and Y position to the position specified by `X[1..0]` and `Y[1..0]`, with the intensity `Z[3..0]`. This is a variant of the VCTR instruction that is usually used for drawing very small vectors where absolute accuracy is not required, for example text or numbers.

This instruction performs the same function as a VCTR instruction, but in a more compact (and slightly faster) way. `X[1..0]` and `Y[1..0]` in an SVEC instruction only contain bits 8 and 9 (the two least significant bits of the high byte) of the X and Y addresses. The scale factor decodes as follows:

Scale bits	Divisor	Bits shifted
00b (00d)	128	7
01b (01d)	64	6
10b (02d)	32	5
11b (03d)	16	4

This means that the input coordinate values and scale factors decode to the following vector lengths:

Scale	Input			
	00b	01b	10b	11b
00b (divide by 128)	**0**	**2**	**4**	**6**
01b (divide by 64)	**0**	**4**	**8**	**12**
10b (divide by 32)	**0**	**8**	**16**	**24**
11b (divide by 16)	**0**	**16**	**32**	**48**

8 State Diagram

A state diagram for the DVG is shown in Figure 1.

VCTR (0-9)	0 9 D C F E A 1 2 d
LABS (A)	0 9 D C F E B d
HALT (B)	0 9 D C 8 B 0
JSR (C)	0 9 D C 8 9 d
RTS (D)	0 9 D C 9 d
JMP (E)	0 9 D C 9 d
SVEC (F)	0 9 D C A 1 2 d

d = new DMALD; branch to next opcode

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
VCTR	00	01	0D	0D	05	06	07	0D	09	0D	01	00	0F	0C	0A	0E
VCTR	00	01	0D	0D	05	06	07	0D	09	0D	01	00	0F	0C	0A	0E
VCTR	00	01	0D	0D	05	06	07	0D	09	0D	01	0D	0F	0C	0B	0E
VCTR	00	01	0D	0D	05	06	07	0D	0B	0D	01	00	08	0C	0A	0E
VCTR	00	01	0D	0D	05	06	07	0D	09	0D	01	00	08	0C	0A	0E
VCTR	00	01	0D	0D	05	06	07	0D	09	0D	01	00	09	0C	0A	0E
VCTR	00	01	0D	0D	05	06	07	0D	09	0D	01	00	09	0C	0A	0E
VCTR	00	01	0D	0D	05	06	07	0D	09	0D	01	00	0A	0C	0A	0E
VCTR	09	02	0D	0D	05	06	07	0D	09	0D	01	00	0F	0C	0A	0E
VCTR	09	02	0D	0D	05	06	07	0D	09	0D	01	00	0F	0C	0A	0E
LABS	09	02	0D	0D	05	06	07	0D	09	0D	01	0D	0F	0C	0B	0E
HALT	09	02	0D	0D	05	06	07	0D	0B	0D	01	00	08	0C	0A	0E
JSR	09	02	0D	0D	05	06	07	0D	09	0D	01	00	08	0C	0A	0E
RTS	09	02	0D	0D	05	06	07	0D	09	0D	01	00	09	0C	0A	0E
JMP	09	02	0D	0D	05	06	07	0D	09	0D	01	00	09	0C	0A	0E
SVEC	09	02	0D	0D	05	06	07	0D	09	0D	01	00	0A	0C	0A	0E

contents of DVG PROM Atari part 034602

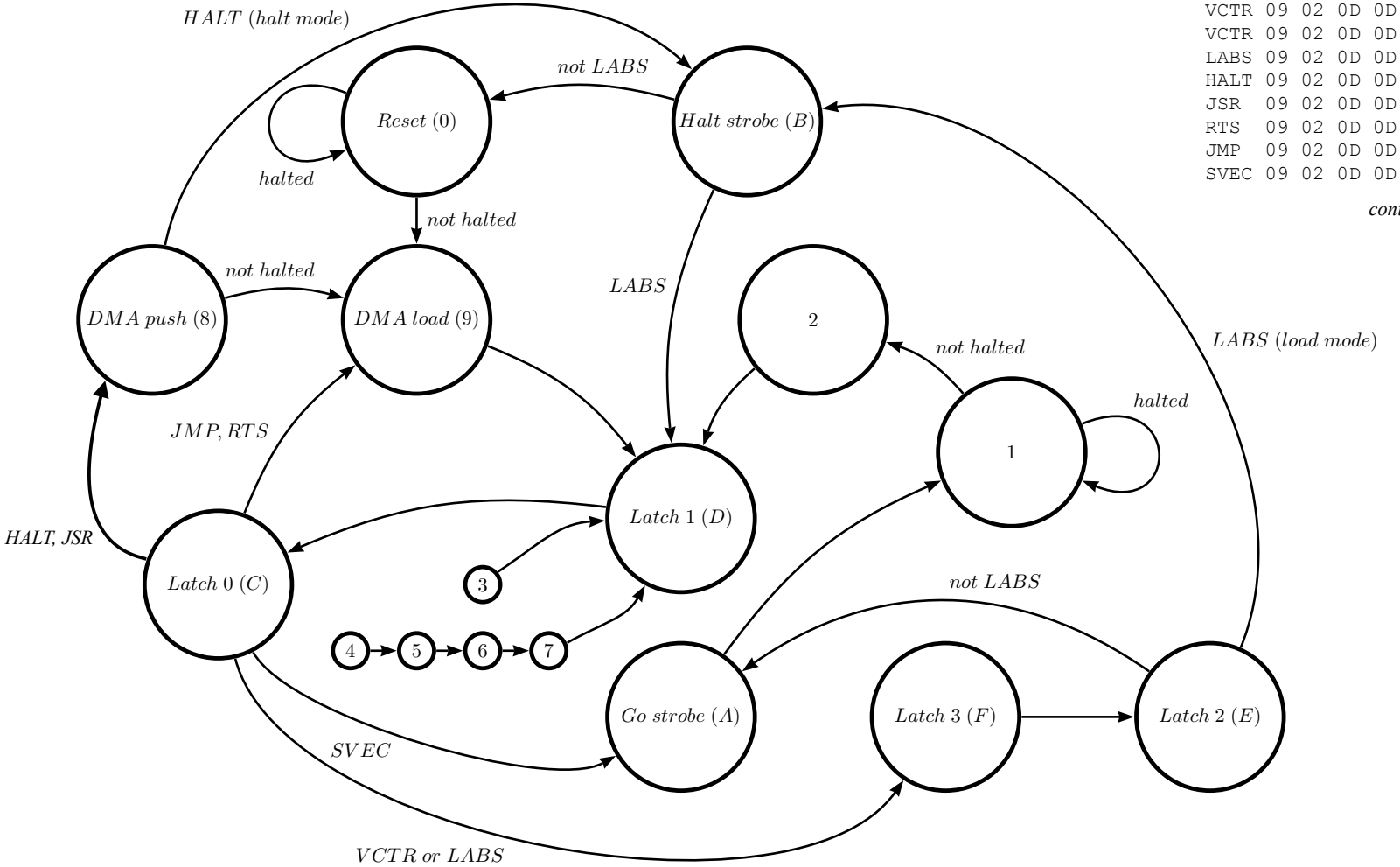


Figure 1: State transition diagram for the Atari DVG