Atari 2600: Stella Console Hardware & Combat Sample Game Software

Joe Decuir
jdecuir@nwlink.com
alumnus of Atari & Amiga

Agenda

- Requirements for the 2600
- Console architecture
- TIA chip architecture
- System programming model
- Combat cartridge architecture
- Display kernel detail
- Vertical blank game play

Atari before Stella, 1975

- Founded in 1972, on coin-op Pong
- Developed more complex coin-op games
- Coin-op games migrated from random logic to microprocessor based design
- Atari's first successful home game was Pong, implemented in a random logic chip
- Atari recognized the challenge of bringing more complex games homes

Requirements for Stella

- Atari management had a clear vision for the product: provide a means to bring successful Arcade games home.
- We had to hit a $200 max retail price for the console, for Christmas of 1977.
- Expected product life: 3 years (e.g. to 1979)
- Non-goals: be an expandable personal computer.

Implementation Choices

- From coin-op games, there were two obvious ways to architect the system:
  - non-programmable random logic
  - programmable (uP) with screen bit-map
- Fatal flaws in both:
  - random logic would be slow development, and not re-usable
  - bit maps were expensive

Design choice: split objects and playfield

- The targeted games had distinct images:
  - large static images (playfield)
  - small moving images.
- Adequate playfield could be done with low resolution: 40x24 = 120 bytes, 960 bits
- Adequate moving objects could be done with a pair of 8 byte squares and some additional bits (e.g. 2 "missiles" & a "ball")

Design choice: soft vertical

- 960 + 64 + 64 was still a lot of static RAM cells in 1976 technology.
- ROM is the cheapest form of memory
- A preferred implementation would compute pointers into ROM.
- A fast enough microprocessor could do it.
- The MOS Technology 6502 could do the job

Why was the 6502 so good?

- Process speed: depletion load pullup transistors were much faster and smaller than enhancement pullups (e.g. 6800).
- Architecture speed:
  - little-endian addresses pipelined instructions
  - indexed-indirect and indirect-indexed instructions allowed use of zero page as an array of fast memory pointers.

Design decisions

- Our target coin op games were two player action (Tank - Combat), sports (Basketball) and paddle (Pong -Video Olympics) .
- We decided that we needed:
  - 2 8-bit motion objects (P0, P1)
  - 3 1-bit motion objects (M0, M1, Ball)
  - 20 or 40 bits of low resolution playfield

Hardware Software tradeoffs: Motion control

- The easy way to make these motion objects would require a binary horizontal counter, and 5 8-bit position registers and comparators. We thought this would be huge.
- The cheap way was to use dynamic polynomial counters, running in parallel. Motion in implemented with resets and motion vectors.

Motion control, continued

- To appease the programmers, I generated a ‘Compute Horizontal Reset’ CHRST utility.
- Called with object index in X, position in A:
  - computes a loop count (15 clocks)
  - computes a residual motion vector (+/-7)
  - waits for sync, loops, resets and writes motion
- For the programmers, this was good enough

Motion control, epilog

An alternative that we considered too late:

- keep the polynomial horizontal counter
- replace the separate object counters and motion registers with simple position latches and comparators
- use a 160-byte look up table in cartridge ROM to map binary horizontal positions to polynomial counter values.

Other TIA chip features

- 4 7-bit palette registers
- 15 collision detection latches
- 2 channel sound system
  - variable prescaler
  - 4+5 bit polynomial counters
  - volume registers
- trigger and potentiometer input ports
- trigger input could be used for light pens or light guns.

Stella Graphics

- Fundamental pixel resolution is 1 color burst clock (280nsec, 160/line) by 1 line.
- Motion objects are 1, 2, 4 or 8 clocks/bit.
- Motion objects may be replicated in hardware.
- Playfield is 4 clocks per bit.
- Playfield bits are either repeated or reflected in hardware.

Human Input Requirements

- We needed console controls:
  - Game select, and start switches
  - Options: handicaps, color/monochrome
- We needed various types of game controls:
  - For TANK, etc: a joystick with a "fire" button
  - For PONG: a dual analog potentiometer
  - For Driving: a rotary control
  - For head games: a keyboard

HID implementation

- One power switch
- 5 bits of console parallel I/O, not scanned
- 5 + 5 bits of game control I/O, not scanned
  - 2 bits in TIA, 8 bits in parallel ports
- 4 bits of potentiometer input, in TIA

Memory

- Three choices:
  - Dynamic RAM (multiple supplies, refresh logic)
  - Static RAM (simple to use, expensive)
  - Static RAM built into a combo chip
- Decision:
  - take the off-the-shelf 6530 combo chip
  - delete the 1K byte ROM
  - double the RAM to 128 bytes

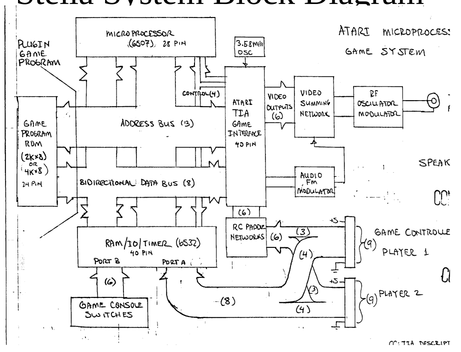
Stella System

- TIA video chip (see below)
- 6502-based processor, "6507":
  - 13 bit address, no interrupts, RDY line
  - 1.2 MHz
- 6532 combo
  - 128 bytes of RAM (all mapped into zero page)
  - 16 bits of parallel I/O (joysticks and panel)
  - timer (interrupt not used)
- cartridge slot for 2K or 4K ROMs (24 pins)
- 2 game control ports

TIA Register Map: 00-0A

• 00:0	Vertical Sync
• 00:1	Vertical Blank
• 02	Wait for Horizontal Sync
• 03	Reset Horizontal sync (testing)
• 04-05	Number and size of P0/M0, P1/M1
• 06-09	Color/lum registers
• 0A	Playfield controls

Stella System Block Diagram



TIA Register Map: 0B-1F

• 0B-0C:3	Player reflect bits
• 0D-0F	Playfield graphics (7-4; 7-0; 7-0)
• 10-14	Horizontal reset, all 5 objects
• 15-16	Audio control
• 17-18	Audio frequency
• 19-1A	Audio volume
• 1B-1C	Player graphics (8 bits)
• 1D-1F	Missile/ball enable (1 bit each)

TIA Register Map: 20-3F

• 20-24	Horizontal motion registers (7-4)
• 25-27	Vertical delay: P0, P1, Ball
• 28-29	Reset Missiles to Players
• 2A	Horizontal Motion strobe
• 2B	Horizontal motion clear
• 2C	Clear collision latches
• 30-37	Collision detect latches
• 38-3D	4 pot inputs, 2 trigger inputs

Combat Game Design

- General architecture
- Display generation
- Game play
- Sounds

Combat Game Architecture

- The code has three components:
  - Game play code
  - Graphics display code
  - Graphics tables
- The rest of the system has:
  - 128 bytes of RAM: variables and stack
  - TIA: graphics, sound, inputs
  - 6532 parallel I/O and timer

General Stella Game timing

- In Vertical Blank:
  - detect collisions and control inputs
  - decide new game conditions
  - computer new game graphics pointers
- In Display, for each line or two:
  - step graphics pointers
  - fetch graphics
  - wait for horizontal blank, and write graphics

Combat Main loop

- VCNTRL: generate vertical sync
- GSGRCK: game select and reset
- LDSTEL: load Stella (TIA) registers
- CHKSW: read the joystick switches
- COLIS: Detect and process object collisions
- STPMPL: Move players and other objects
- ROT: generate & rotate object graphics
- SCROT: generate score graphics
- VOUT: display the game

VCNTRL (F032-F053)

- Count the frame
- Clear motion registers
- Three blank lines
- Three lines of vertical sync
- Set timer for rest of vertical blank

GSGRCK (F157-F1F1)

- Check game control switches
- If game done and timeout, do attract mode
- If game reset, clear and reset game
- If game select:
  - step game number
  - configure options

LDSTEL (F572-F5BC)

- Derive indexes into ROM game tables from game number
- Copy data from ROM game tables into TIA
- Write color registers, from game or from attract mode
- Implement invisibility if required by game option

CHKSW (F2DA-F40F)

- Lock out controls for spin mode
- Check difficulty switches (handicaps)
- If rotation, update rotation index
- If enabled, controls steer missiles, too
- If fire buttons, initiate missile flight
- Determine motor sounds
  - MOTORS (F410-F443)

COLIS (F444-F524)

- Scan selected hardware collision detect bits
- If a missile hits opponent player:
  - Scoring
  - Changes in motion: spin, RECOIL
- If a player or missile hits playfield
  - Bounce/change direction
  - Implement RECOIL if a tank is in a wall

STPMPL (F214-F2A8)

- Determine which objects need to move
- Inputs: 4+4 bit motion vectors
- Apply motion vectors to horizontal and vertical addresses
  - Fastest objects step every frame, slower don't
  - Jet/biplane games wrap around vertical and horizontal

ROT (F2A9-F2D9)

- Input: 4 bit rotation index: 22.5 degree steps
- Conditionally enable graphics reflection
- Compute the ROM graphics table address
- Fill the output table in RAM (OTTBL)

SCROT (F1F2-F253)

- Inputs: 4 BCD score digits
- Compute pointers to ROM table addresses for each digit
- ROM tables are 3x5 bit images for simple images of digits 0-9 plus blank

VOUT (F054-F156)

- Main components:
  - Wait on the 6532 timer until the end of vertical blank
  - Implement horizontal motion for all 5 objects
  - Display score digits
  - Display playfield and objects

Horizontal motion

- For each moving object:
  - Given the horizontal position (0-159)
  - Compute a loop count for a wait loop, mod 15
  - Compute the horizontal motion step, -7 to +7
  - Wait for horizontal sync
  - Run the wait loop
  - Reset the object motion counter
  - Write the horizontal motion register
- Write HMOVE after all registers set up

Display the Score digits

- For 6 horizontal line pairs, run a display kernel:
- Step the score digit indexes on alternate lines
- For each line:
  - Use score digit index values to compile two bytes of score graphics
  - Write the first score to PF1
  - Wait to mid-screen
  - Write the second score to PF1
- Exit to display the game

Display the Game

- For pairs of horizontal lines:
- Compute indexes to playfield:
  - move 2.5 bytes from ROM tables
  - playfields are vertically reflected in software
- For each object that is on, copy graphics
  - For 8 bit objects, copy graphics from RAM
  - For 1 bit objects, enable/disable
- Use Wait-for-sync, and write graphics in horizontal blank

Sound Control Registers

- Hardware generates sounds
- Two channels:
  - Audio control: 4 bits (next slide)
  - Audio frequency: 5 bits (divide by 1- 32)
  - Audio volume: 4 bits (0-15)
- The base frequency = 30kHz (2 x hsync)

Audio Control Registers

- 0, B: set to 1, modulate with volume
- 1: 4 bit polynomial counter
- 2: div by 15 -> 4 bit poly counter
- 3: 5 bit poly counter clocks 4 bit poly counter
- 4, 5: divide by 2
- 6, A: divide by 31
- 7: 5 bit poly counter, divide by 2
- 8: 9 bit poly counter (white noise)
- 9: 5 bit poly counter
- C, D: divide by 6
- E: divide by 93
- 5 bit poly counter divided by 6