][-Vision
Streaming video on an Apple][

Kris Kennaway
KansasFest 2019

Demo

- //gs pretending to be //e
- i.e. 1 MHz, 64k, extended 80-column card
- Standard Apple][speaker, not //gs sound
- Audio out with amplified speaker for presentation
- Uthernet II - streaming from my laptop
- For convenience, loading player from CFFA3000
    - but not relying on these
- Should work on everything going back to original 64KB Apple][
    - but untested (yet!)

Demo

Double Hi-Res

Credits: https://www.youtube.com/watch?v=rXONcuozpvw

5-bit audio from a 1-bit speaker

- Apple][speaker knows how to tick (move in/out), once
- Old technique going back to ~1990
- Pulse Width Modulation: precisely control the duty cycle of 1-bit speaker cone by allowing it to move during a fraction of each time interval
  - e.g. 14 cycles out of every 73.
- By varying this duty cycle you can modulate a constant "carrier wave" (1MHz/73 = 14.3KHz in this case) with lower frequencies, and simulate higher audio bit depth.
  - e.g. with $32=2^5$ possible duty cycles at 73 cycles this gives 5-bit audio at 14.3KHz.
- This frequency is lower than ideal (Michael Mahon's dac522 player uses a non-audible 22KHz carrier) but not too bad on the built-in speaker
  - at least to my 41-year old ears ;)

Audio playback - in code

```
TICK = $C030 ; tick speaker

; tick speaker 12 cycles apart
and pad to 73 cycles (14.3KHz)
op_tick_12:
    STA TICK ; 4 cycles
    ; wait 12 cycles
    NOP ; 2 cycles
    NOP ; 2
    NOP ; 2
    NOP ; 2
    STA TICK ; 4 cycles
```

```
    ; wait 54 cycles
    NOP
    NOP
    NOP
    ; [...24 more NOPs...]
@D:
    JMP somewhere ; 3 cycles
```

Audio playback - in code

- ...look at all those wasted CPU cycles o_O
- In a RAM-based audio player this is where you’d have to deal with stepping through memory, figuring out where to jump next, etc.
- Probably willing to trade extra CPU cycles for more efficient use of memory.
- dac522 gets about 4 seconds of playback from 48KB

Uthernet II ethernet card

- I bought an Uthernet II
  - now what can I do with it?
- W5100 ethernet controller
- Auto-incrementing memory pointer into 8KB of onboard TCP read socket buffer
  - every time you read the same Apple II I/O location it steps through the 8KB of internal memory on the W5100
- Fastest possible way to get data into 6502, short of DMA

From audio to video

- The audio decoding is the most timing-critical part
  - hang off this chassis
- Let’s fill in those wasted cycles by reading bytes from the TCP socket
  - Ignore TCP stream management for now

Step 1: Self-modify to work out what to do next

```
; tick speaker 12 cycles apart and pad
to 73 cycles
op_tick_12:
    STA TICK ; 4 cycles
    ; tick again 12 cycles later
    NOP ; 2 cycles
    NOP ; 2
    NOP ; 2
    NOP ; 2
    STA TICK ; 4
```

```
    ; wait 54 cycles
    NOP
    NOP
    NOP
    ; [...18 more NOPs ]
    ; ask TCP stream where to go next
    LDA TCP_DATA; 4 cycles
    STA @D+2 ; 4 cycles
    LDA TCP_DATA ; 4 cycles
    STA @D+1 ; 4 cycles
@D:
    JMP somewhere ; address self-modified
```

Step 2: Video is about storing bytes to memory

```
; tick speaker 12 cycles apart and pad
to 73 cycles
op_tick_12:
    STA TICK ; 4 cycles
    ; tick again 12 cycles later
    ; ask TCP stream which byte value
    ; to store
    LDA TCP_DATA ; screen content value
in A register
    NOP ; 2
    NOP ; 2
    STA TICK ; 4
```

```
    ; wait 54 cycles
    NOP
    NOP
    NOP
    ; [...18 more NOPs ]
    ; ask TCP stream where to go next
    LDA TCP_DATA ; 4 cycles
    STA @D+2 ; 4 cycles
    LDA TCP_DATA ; 4 cycles
    STA @D+1 ; 4 cycles
@D:
    JMP somewhere ; 3
```

Step 3: Images have redundancy, and errors are OK

```
op_tick_12_page_20:
    STA TICK ; 4 cycles
    LDA TCP_DATA ; load content byte
    ; store same content byte at 4
    ; offsets within memory page $20

    ; load first page offset into Y
    LDY TCP_DATA
    STA TICK ; 4
    STA $2000,Y ; store content byte
    LDY TCP_DATA ; load second offset
    STA $2000,Y ; store content byte
    LDY TCP_DATA ; load third offset
    STA $2000,Y ; store content byte
    LDY TCP_DATA ; load fourth offset
    STA $2000,Y ; store content byte

     ; 6 cycles left over
    NOP
    NOP
    NOP
    ; ask TCP stream where to go next
    LDA TCP_DATA ; 4 cycles
    STA @D+2 ; 4 cycles
    LDA TCP_DATA ; 4 cycles
    STA @D+1 ; 4 cycles
@D:
    JMP somewhere ; 3
```

Summary: Audio + video player opcodes

- We have enough spare cycles to store a single arbitrary byte value at 4 arbitrary offsets on page $20, while ticking the speaker 73 cycles apart
- What about other HiRes screen 1 pages ($21..$3F)? Just copy this 31 times
- Supporting other speaker duty cycles 4, 6, …, 66 - same basic idea (although fiddly to rearrange things to tick at exactly the right cycle counts)
  - and for 2 of them I could only get them off-by-one cycle
- Somewhat magically, these 32*32=1024 player opcode variants (barely) fit in 64K main memory together with ProDOS
  - few KB to spare in odd corners
  - 6 leftover cycles are necessary to allow rearrangements, and to reduce memory by JMP’ing to common code sequences.

Is it enough?

- Can store 4 screen bytes every 73 cycles
- ~57000 byte stores/second
- 40*192=7680 bytes on hires screen
- so about 7.5 complete screen refreshes/second
- should be (barely) enough for reasonable frame rates, assuming we can manage the errors.

Enough details, let’s watch another video

(HiRes, 24 FPS)

Credits: https://www.youtube.com/watch?v=9INZ_Rnr7Jc

Video player - recap

- TCP byte stream is steering CPU to spray a sequence of bytes at 4 offsets on a memory page within HiRes screen page 1
- ...while also steering speaker cone to produce 5-bit audio.
- No conditional 6502 opcodes so far, playback is completely deterministic
  - also only uses 2 of 3 6502 registers, convenient to maintain X=0
- But we can only read 8KB of data before we fall off of the end of the TCP socket buffer

Slow path

- Need to periodically manage TCP socket buffer
- Want to leave some free space so socket buffer can refill in the background while we read from it
- 2KB is good compromise
- Server can cause client to perform TCP buffer management when exactly 2KB has been read from socket buffer
  - move socket read pointer, ACK TCP stream, check that at least 2KB still in socket buffer
- Need to keep ticking speaker every 34+39 cycles during slow path to minimize audio artifacts
  - 34 cycles is the “baseline” speaker duty cycle
- Slow path fits in 2x73 cycles i.e. 2 "neutral" audio frames every 292
  - 0.7% noise/overhead

While we're in here

- What about Double HiRes?

While we're in here

- What about Double HiRes?
- Requires flipping a single soft-switch to steer writes between HiRes screen pages in MAIN memory or AUX memory

While we're in here

- What about Double HiRes?
- Requires flipping a single soft-switch to steer writes between HiRes screen pages in MAIN memory or AUX memory
- Since that has a bit of overhead, we'd probably want to only flip back and forth periodically

While we're in here

- What about Double HiRes?
- Requires flipping a single soft-switch to steer writes between HiRes screen pages in MAIN memory or AUX memory
- Since that has a bit of overhead, we'd probably want to only flip back and forth periodically
- ...so let's do it in the slow path

While we're in here

- What about Double HiRes?
- Requires flipping a single soft-switch to steer writes between HiRes screen pages in MAIN memory or AUX memory
- Since that has a bit of overhead, we'd probably want to only flip back and forth periodically
- ...so let's do it in the slow path
- read a TCP byte and self-modify to interpret this as a soft-switch address to toggle.

Double Hi-Res support in 3 instructions

```
op_ack: ; slow-path - manage TCP buffers
    ; [...]
    ; allow flip-flopping the PAGE1/PAGE2 soft switches to
    ; steer subsequent writes to MAIN/AUX screen memory
    LDA TCP_DATA ; ask TCP stream which soft switch to flip
    STA @D+1
@D:
    STA $C0FF ; flip the soft switch (low-byte is modified)
```

- By adding 3 instructions to the player loop (plus some initialization, and timing fixups) we can support DHGR video playback.
- Gives some visual interlacing but reasonable quality at 2KB frame size

Now what?

- So now we have a player that is capable of playing a (D)HGR video stream with multiplexed audio.

Now what?

- So now we have a player that is capable of playing a (D)HGR video stream with multiplexed audio.
- How do we produce one?

Now what?

- So now we have a player that is capable of playing a (D)HGR video stream with multiplexed audio.
- How do we produce one?
- How do Apple II colour graphics work, anyway?

Now what?

- So now we have a player that is capable of playing a (D)HGR video stream with multiplexed audio.
- How do we produce one?
- How do Apple II colour graphics work, anyway?
- Let’s see what Woz had to say...

Woz explains Apple II colour graphics

(HiRes, 30 FPS)

Credits: https://www.youtube.com/watch?v=uCRijF7lxzI

Woz explains Apple II colour graphics

(HiRes, 30 FPS)

Credits: https://www.youtube.com/watch?v=uCRijF7lxzI

(4 days without sleep, folks)

Crash course in Apple II colour graphics

- signals from colour TVs go up and down, up and down
  - at a certain speed
- 4 little 0-1 bits, circling around
- going up and down at a different time, then red would become blue
- 16 patterns of 1’s and 0’s, become different shades

Crash course in Apple II colour graphics

- signals from colour TVs go up and down, up and down
  - at a certain speed
- 4 little 0-1 bits, circling around
- going up and down at a different time, then red would become blue
- 16 patterns of 1’s and 0’s, become different shades
- **(Come to my lightning talk tomorrow)**

Video transcoder

- Demultiplex input sequence of video frames and audio stream
- Downsample audio to 14.7KHz (=44.1KHz / 3) and normalize to 5-bit range
    - 2.8% slower playback at 14.3Khz but better downsampling quality
- Bill Buckels' BMP2DHR to turn video frame into "ground truth" Apple II memory representation
- Compute colour artifact representation of memory frame

Video transcoder

- Choose sequence of (content, page, offsets) to minimize error between old and new image frame
    - this is the main computational step
- Multiplex audio and video into sequence of player opcodes
- Compile opcodes into byte stream tailored for particular version of player, i.e. known memory addresses
- Insert slow path opcodes every 2KB

Frame error minimization

- Choose `(content, page, 4 offsets)` such that they minimize the **perceptual** difference between current screen content and target content
- Priority order - resolve the largest differences first
- Residual errors accumulate to next frame
- Prioritizes large differences between frames
- If we don’t get to resolving fine detail this frame, we’ll be more likely to next frame (if diff is still there)

Perceptual distance

- Some colours are perceptually more similar than others
    - e.g. lower visual error introduced by substituting a nearby colour or leaving with “wrong” colour
- CIE2000 perceptual colour difference metric
- Compute strings of coloured pixels that are influenced by storing a given byte
- Damerau-Levenshtein edit distance between two such pixel strings
    - measures how many **pixel colour changes** and **transpositions** are needed to turn one string into the other
    - i.e. also accounts for shifting groups of pixels left/right (less perceptual difference)
- Precompute edit distance between all possible source and target pairs
    - i.e. taking into account the visual pixel colour changes introduced by storing all possible bytes against all possible backgrounds.
- Python (NumPy), ~5x-10x slower than real-time

Future work: video quality

- BMP2DHR + encoding errors gives an approximation of an approximation of the true image
  - Also BMP2DHR is completely unaware of artifact colours?
- Should get better results by directly optimizing against original image frame
- Can optimize directly for the unusual constraints of video player (4 offset stores of the same content byte)
- Take artifact colours into account
- Better control of spatial error diffusion

Caveat: likely more computationally expensive

Future work: audio, player

Audio quality

- with better signal processing should be possible to reduce audio artifacts

Player

- Player currently hard-codes IP address (should be ~easy to fix) and Uthernet II slot (harder to fix - need to patch thousands of memory addresses)
- Would be nice to have a file selector UI
  - i.e. bidirectional TCP communication with server to enumerate and select available files
- Playback seek controls would require real-time encoding
  - i.e. rewriting in higher-performance language

https://github.com/KrisKennaway/ii-vision

kris.kennaway@gmail.com