Finish Him!

Reversing Midway Arcade Audio Compression

exide

REcon 2015, Montreal

Introduction

- About Me
  - IT Monkey (Consultant) by day
  - Hardware Hacker by night
- Interests
  - Designing & reversing embedded systems
  - IC security & failure analysis
  - Arcade platforms
  - Automotive stuff
- Contact
  - Email: exide31337@yahoo.com

Overview of DCS

- *Digital Compression System (DCS)* sound system
- Developed by Williams Electronics for Williams pinballs & casino slot machines and Midway coin-op arcade games
- Architecture provides 6 channels of 16-bit audio
  - Independent control over the volume, looping, & playback of each
- Playback anything from a short sound effect to voice-overs, to several minute long music tracks
- 1st pinball game to use DCS was Indiana Jones (1993)
- 1st arcade game to use DCS was Mortal Kombat II (1993)

Source: http://www.planetarypinball.com/mm5/Williams/archives/willy3.htm

Motivation

- As a kid, I remembered MKII & follow-on games sounding way more atmospheric & realistic than anything else at the time
- Wanted to understand the ADSP-2100 series of DSPs, and reverse the HW & code
- MAME does a fantastic job of emulating DSP, but executes binary DSP code blob as-is
  - Get no insight into how the DCS algorithm works
- Did some Googling, searching forums & Usenet
  - Found a decent Usenet post (circa 1995) from Williams, covering high-level system architecture
  - Mostly found other people asking about how it works
- Chance to learn DSP reversing, signal processing, & audio compression techniques

Variants of DCS

- DCS1 ROM-based mono
  - ADSP2105 DSP w/ single DAC
- DCS-95
  - Revised version of above for WPC-95 pinball system
- DCS2 ROM-based stereo
  - ADSP2104 DSP w/ dual DACs
- DCS2 RAM-based stereo
  - ADSP2115 DSP w/ dual DACs
- DCS2 RAM-based multi-channel
  - ADSP2181 DSP w/ up to 6 DACs

Source: https://en.wikipedia.org/wiki/Digital_Compression_System

History of Arcade Audio

- Techniques
  - Analog
  - Sound generators
    - BurgerTime (1982) – GI sound generator IC
  - FM synthesis (ie, AdLib/SoundBlaster)
    - Jackal (1986) – Yamaha FM IC
  - Sample-based (PCM, ADPCM, etc.)
    - Out Run (1986) – Yamaha FM IC + PCM sample IC
    - TMNT4 (1991) – Yamaha FM IC + ADPCM sample IC
    - Mortal Kombat (1992) – Yamaha FM IC + ADPCM sample IC
    - Super SFII Turbo (1994) – Yamaha FM IC + QSound sample IC
    - SFIII 3rd Strike (1999) – 16-channel 8-bit sample IC
  - Perceptual-based audio compression (DCS)
    - MKII (1993) – Analog Devices DSP
  - Modern PC-based (WAV, OGG, MP3, etc.)








How DCS Works

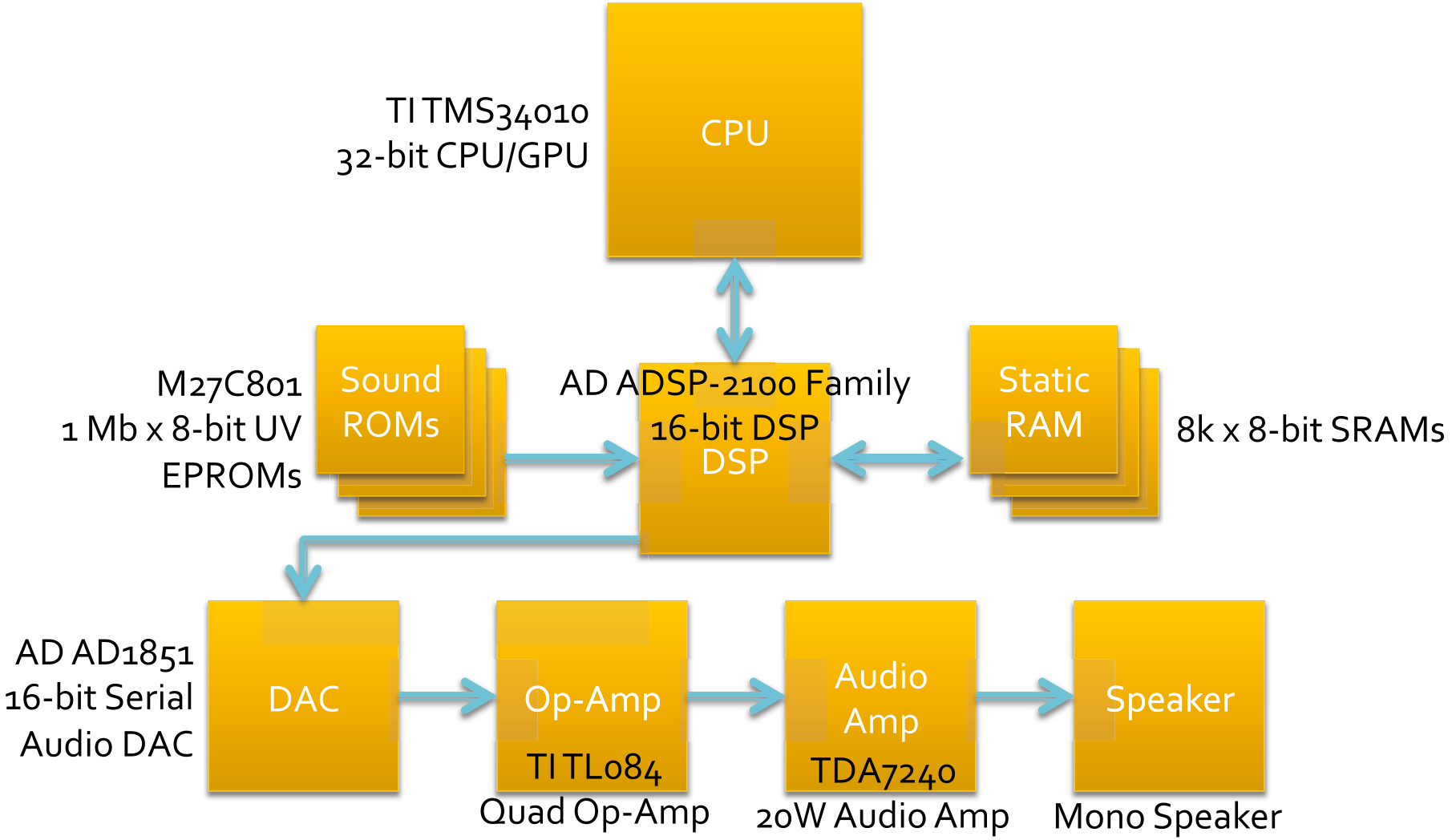
- Uncompressed audio is encoded offline on a PC (486-class typ.)
- Source audio files broken down into frames of 240 samples
- Each frame is 7.68 ms of audio
- Sounds can range from one to several thousand frames
- Each frame is transformed from time-domain to frequency-domain by a 256-point Fast Fourier Transform (FFT)
  - Cosine windowing & 8 samples of overlap on each end
- Resulting spectrum is further broken down into 16 sub-bands and quantized according to masking curves & user-controlled parameters
- The quantizing levels & resulting data for each frame are entropy-encoded into variable length packets
- The packets for each file are combined with header blocks and stored as files that are used later to generate ROM images

Source: http://www.planetarypinball.com/mm5/Williams/archives/willy3.htm

DCS System Architecture

- Architecturally similar to MPEG-1 Layer I & Layer 2 Audio (MP1, MP2)
  - Similar complexity to Sony SDDS & ATRAC (MiniDisc), DTS, Philips Digital Compact Cassette (DCC) tapes
  - MPEG-1 Layer 3 (MP3) more complex than DCS
- ~10:1 compression ratio
- Very difficult to make encoder, due to missing masking curve logic, entropy-coding logic, quantizer behavior
  - Might be possible, but audio quality likely poor

DCS System Architecture



Digital Signal Processor (DSP)

Analog Devices ADSP-2100 Family

SUMMARY

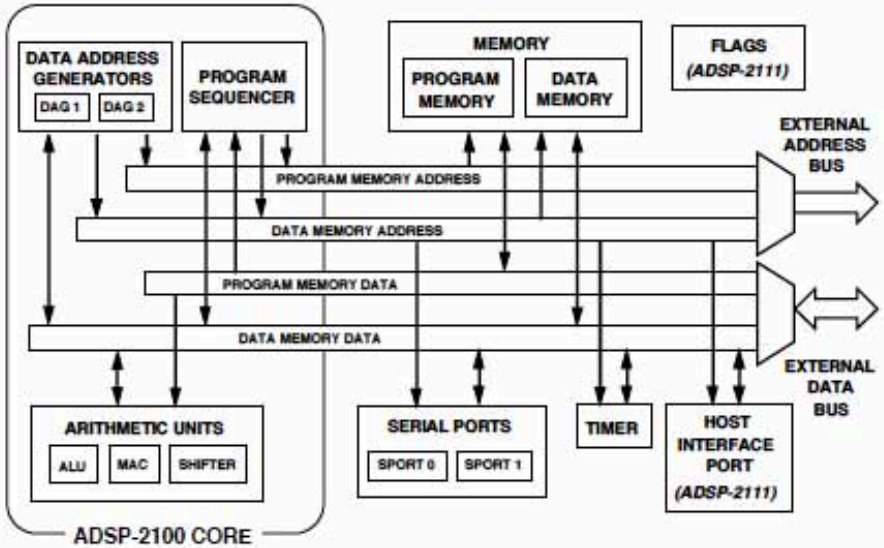
- 16-Bit Fixed-Point DSP Microprocessors with On-Chip Memory
- Enhanced Harvard Architecture for Three-Bus Performance: Instruction Bus & Dual Data Buses
- Independent Computation Units: ALU, Multiplier/ Accumulator, and Shifter
- Single-Cycle Instruction Execution & Multifunction Instructions
- On-Chip Program Memory RAM or ROM & Data Memory RAM
- Integrated I/O Peripherals: Serial Ports, Timer, Host Interface Port (ADSP-2111 Only)

10 MHz XTAL in use

FEATURES

- 25 MIPS, 40 ns Maximum Instruction Rate
- Separate On-Chip Buses for Program and Data Memory
- Program Memory Stores Both Instructions and Data (Three-Bus Performance)
- Dual Data Address Generators with Modulo and Bit-Reverse Addressing
- Efficient Program Sequencing with Zero-Overhead Looping: Single-Cycle Loop Setup
- Automatic Booting of On-Chip Program Memory from Byte-Wide External Memory (e.g., EPROM)
- Double-Buffered Serial Ports with Companding Hardware, Automatic Data Buffering, and Multichannel Operation

FUNCTIONAL BLOCK DIAGRAM

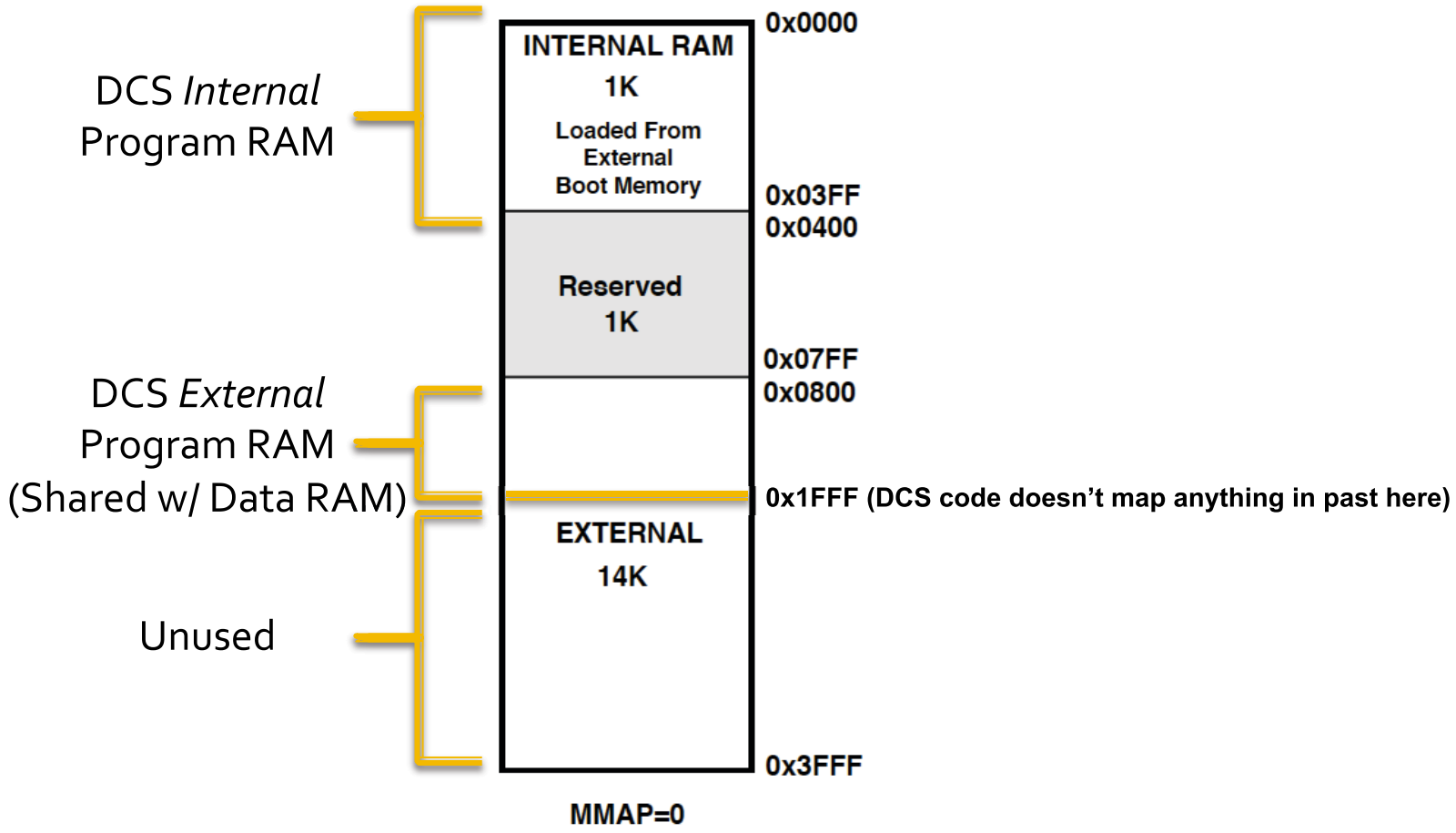


This data sheet describes the following ADSP-2100 Family processors:

ADSP-2101	
ADSP-2103	*3.3 V Version of ADSP-2101*
ADSP-2105	*Low Cost DSP*
ADSP-2111	*DSP with Host Interface Port*
ADSP-2115	
ADSP-2161/62/63/64	*Custom ROM-programmed DSPs*

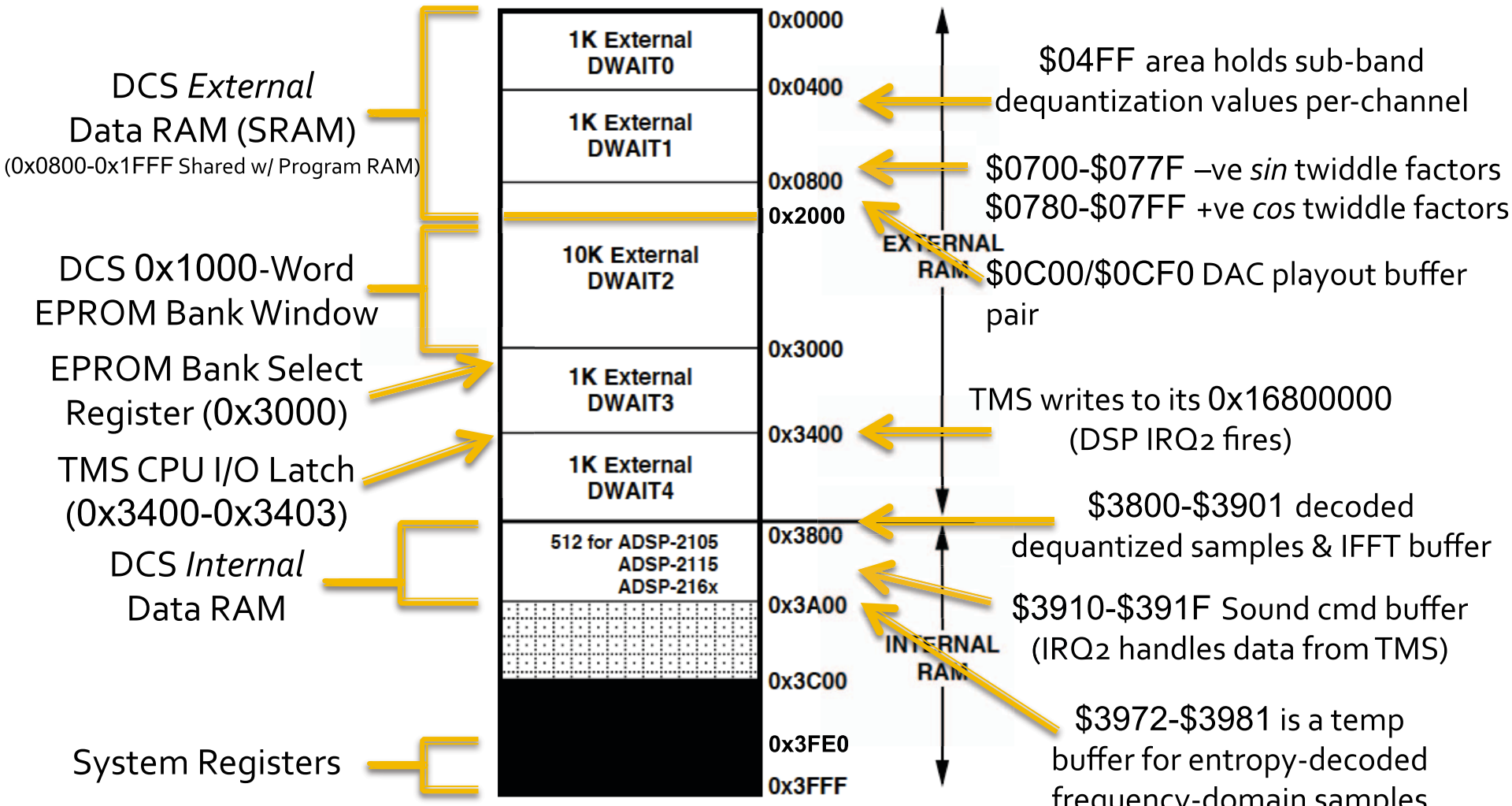
Source: Analog Devices ADSP-2100 Family Data Sheet

DSP *Program Memory* Map



Source: Analog Devices ADSP-2100 Family Data Sheet

DSP *Data Memory* Map



Source: Analog Devices ADSP-2100 Family Data Sheet

Digital-to-Analog Converter (DAC)

Analog Devices AD1851 16-bit Serial PCM Audio DAC

DSP (PLCC-68)

56 (SCLK)

53 (TFS)

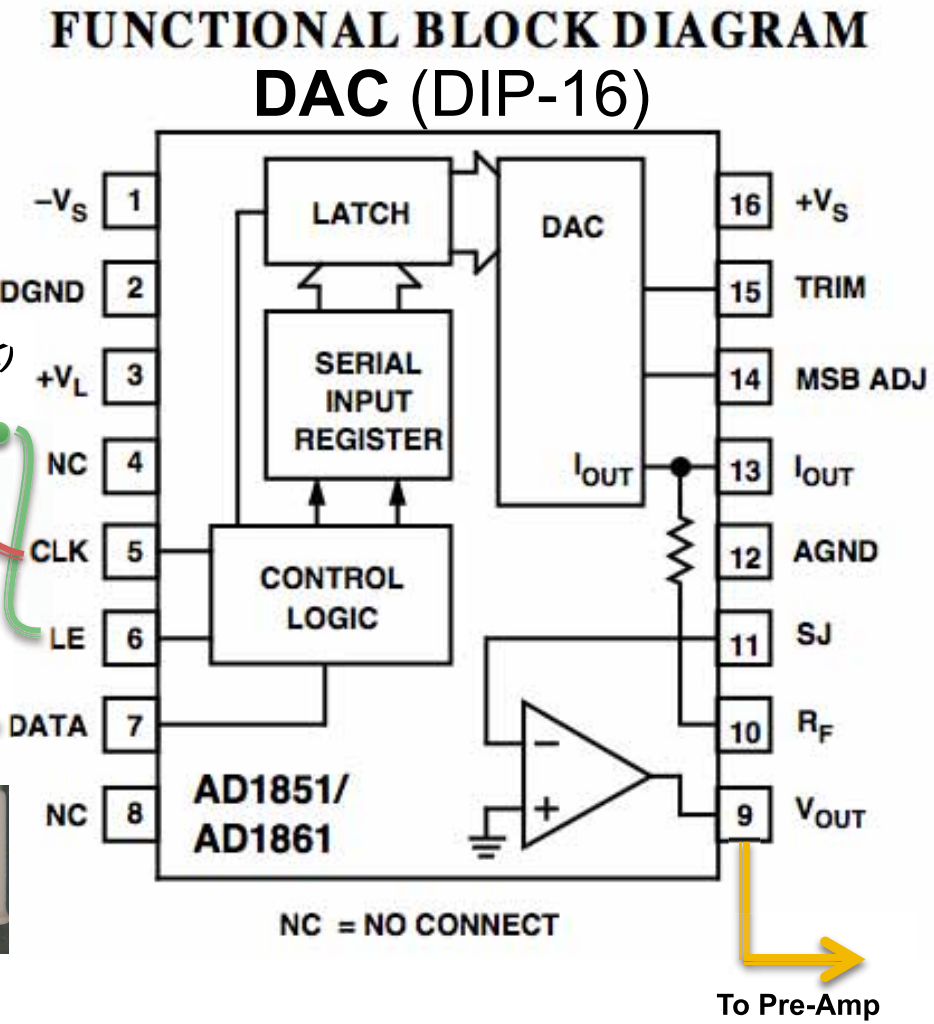
52 (TX)

41 (XTAL)

10 MHz XTAL

SCLK = 500 kHz
500 kHz / 16-bit = 31250 Hz
(ie, 32 kHz Sample Rate)

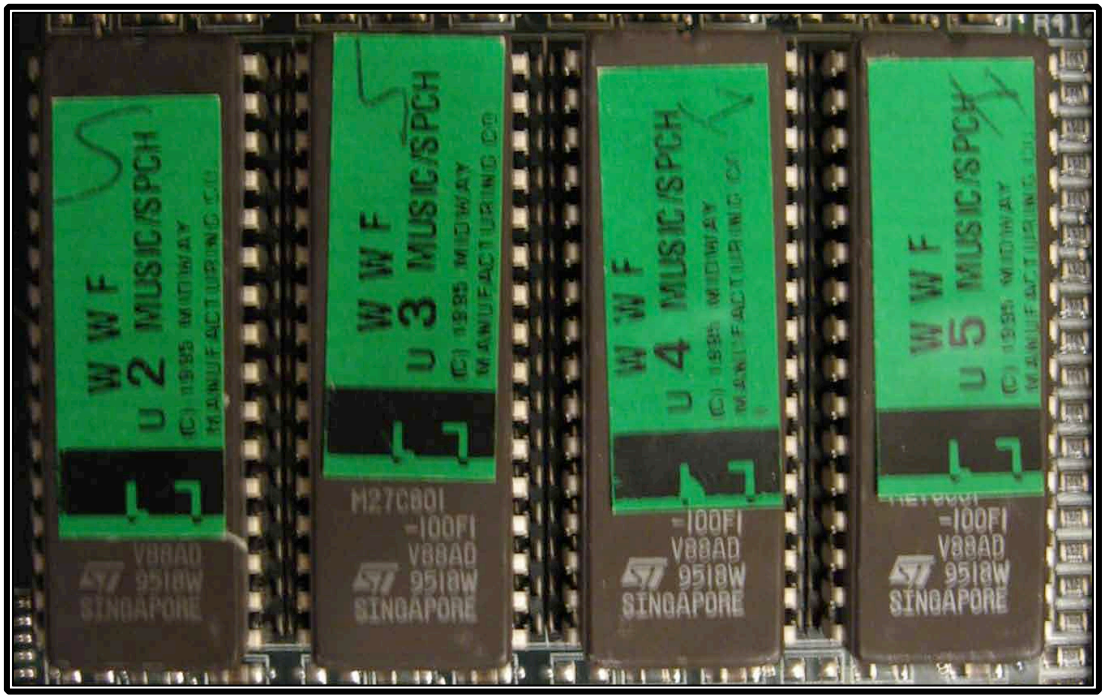
DAC (DIP-16)



Source: Analog Devices AD1851 Data Sheet

Sound ROMs

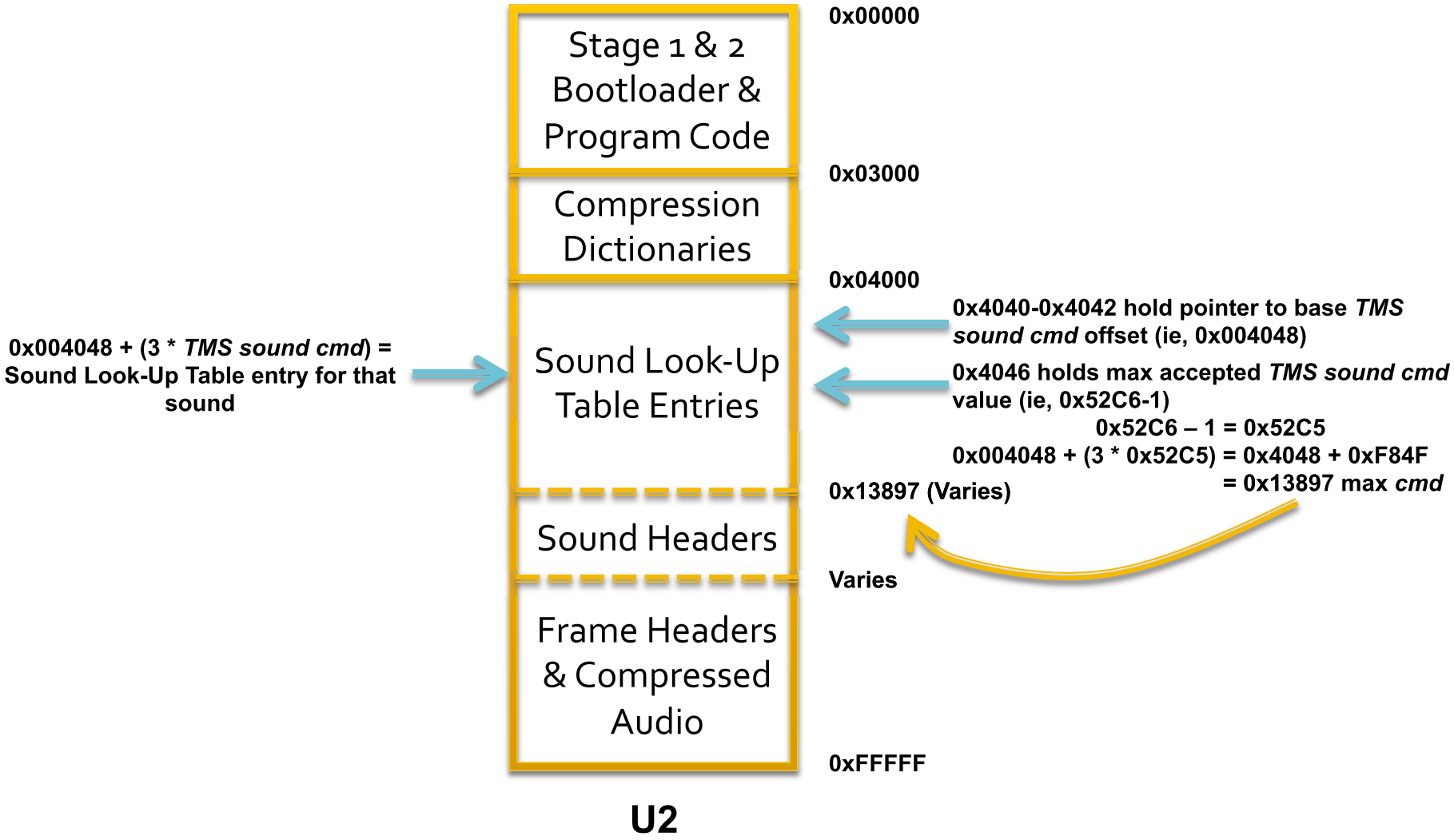
- No encryption or obfuscation
- Remove from PCB and dump w/ parallel E(E)PROM programmer
  - Willem
  - Xeltek
  - Your own design



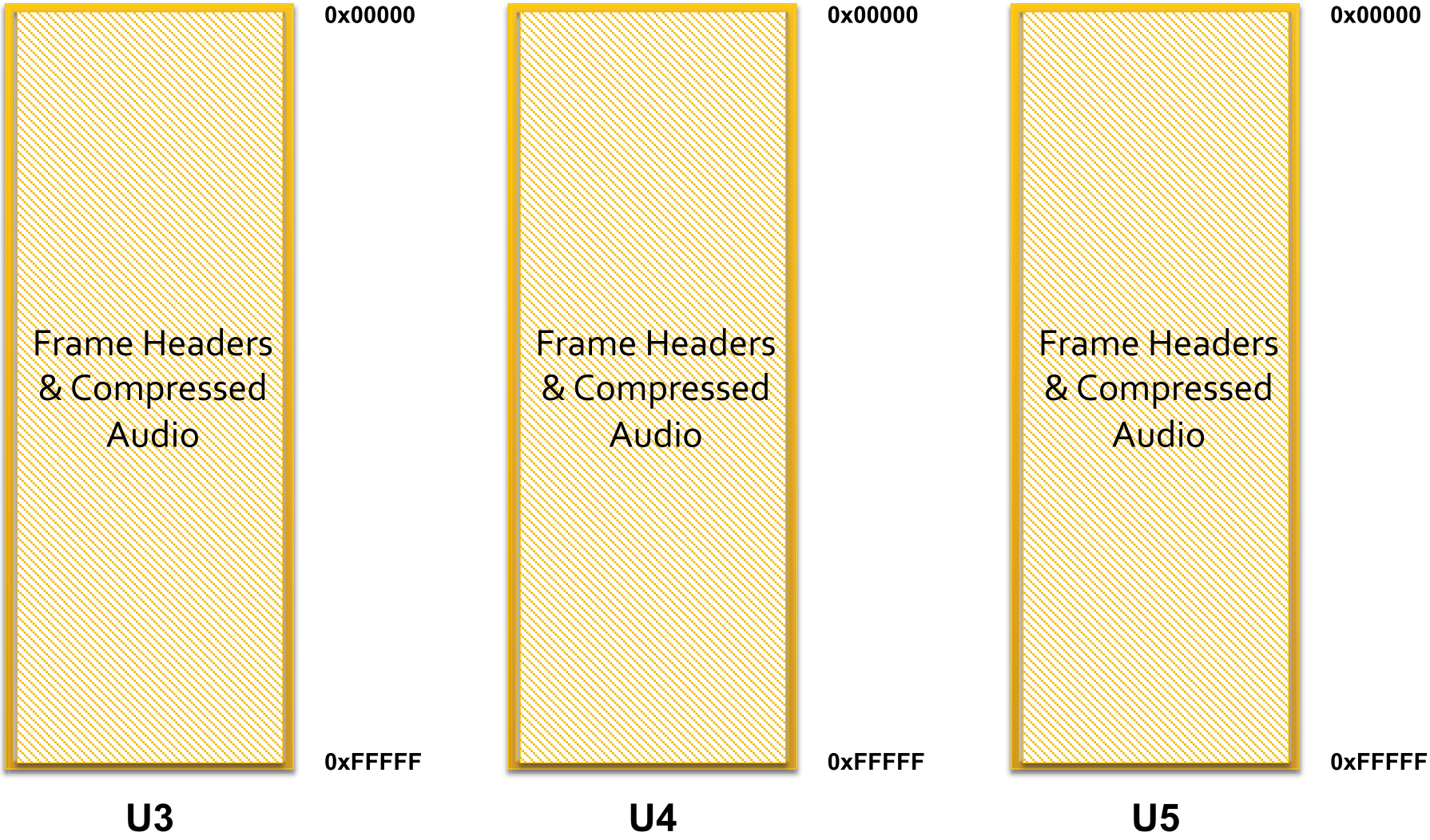
Sound ROMs

- *Wolf* MK3 hardware uses 4x M27C801 (1 Mb x 8-bit = 1 MB) UV EPROMs
  - ie, 4x of these would provide 4 MB total for sound code & data
  - ~10:1 audio compression yields ~40 MB uncompressed
- Organization
  - DSP bootloader (stage 1 & 2) code, program code, data (U2 - 0x0000, 0x1000)
  - Compression dictionaries (U2 - 0x3000)
  - Look-up table entries (U2 - 0x4000)
  - Sound headers (U2 – 0x13800 & up)
  - Frame headers (U2, U3, U4, U5 - varies)
  - Compressed audio (U2, U3, U4, U5 – varies)
- Later games use IDE hard drives & copy to DRAM(s)
  - What does partitioning or file system structure look like?

Sound ROM Structure



Sound ROM Structure

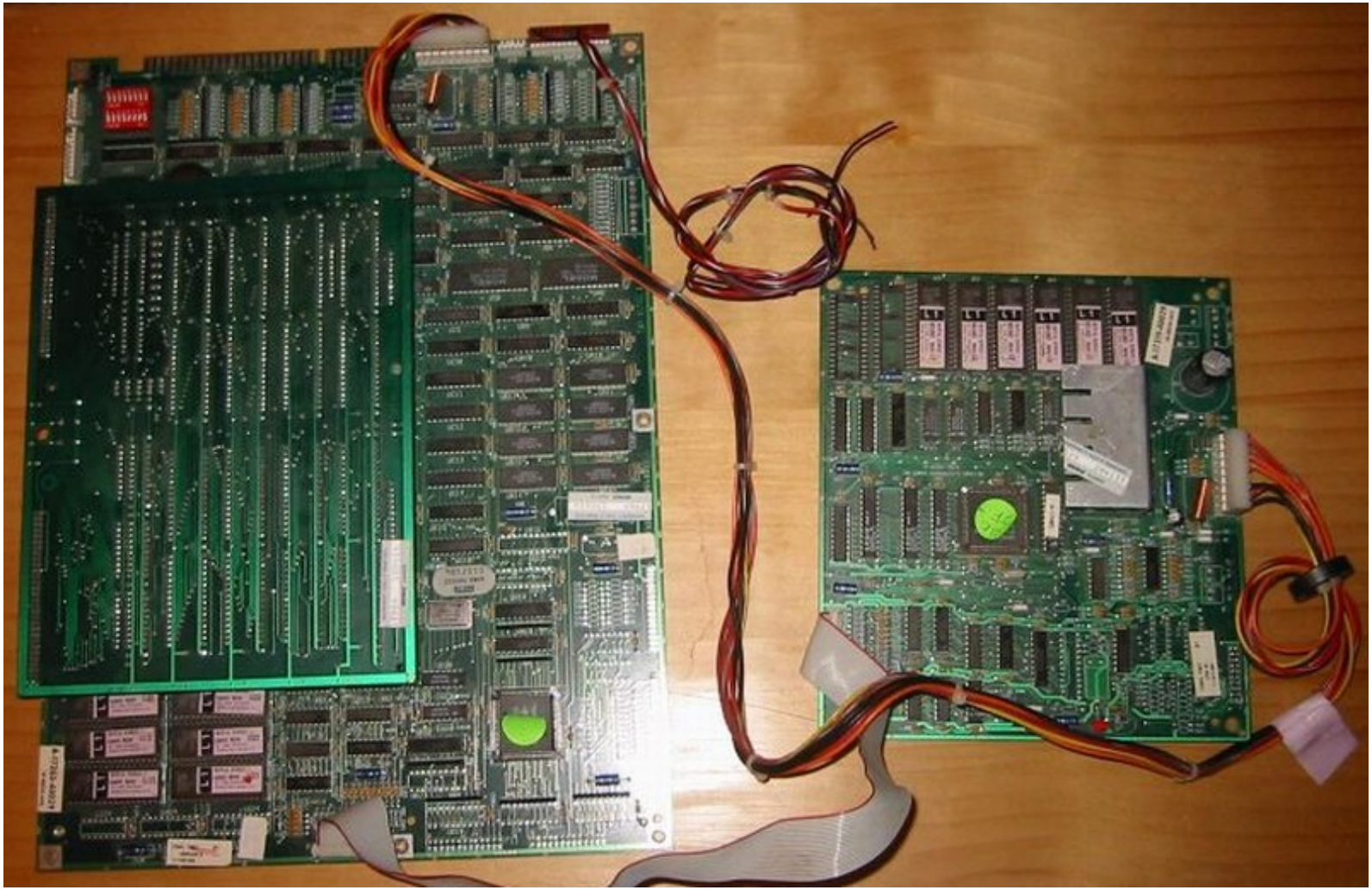


Sound ROM Bank-Switching

- DSP uses a bank-switched window to access 0x1000 16-bit words from ROM at a time
- Writing to DM($3000) will map in a given bank at a given offset into DM($2000-$2FFF)
  - Upper 8-bits of word are modulo-4 values ($00,$01,$02,$03) determining which ROM image to select (U2,U3,U4,U5)
  - Lower 8-bits of word determine the offset in the ROM image (ie, $17 is offset 0x17000)
- Some examples:

DM($3000)	ROM Image	Offset
$0000	U2	0x00000
$0001	U2	0x01000
$0017	U2	0x17000
$0104	U3	0x04000
$0220	U4	0x20000
$0314	U5	0x14000

T Unit (MKII Version)

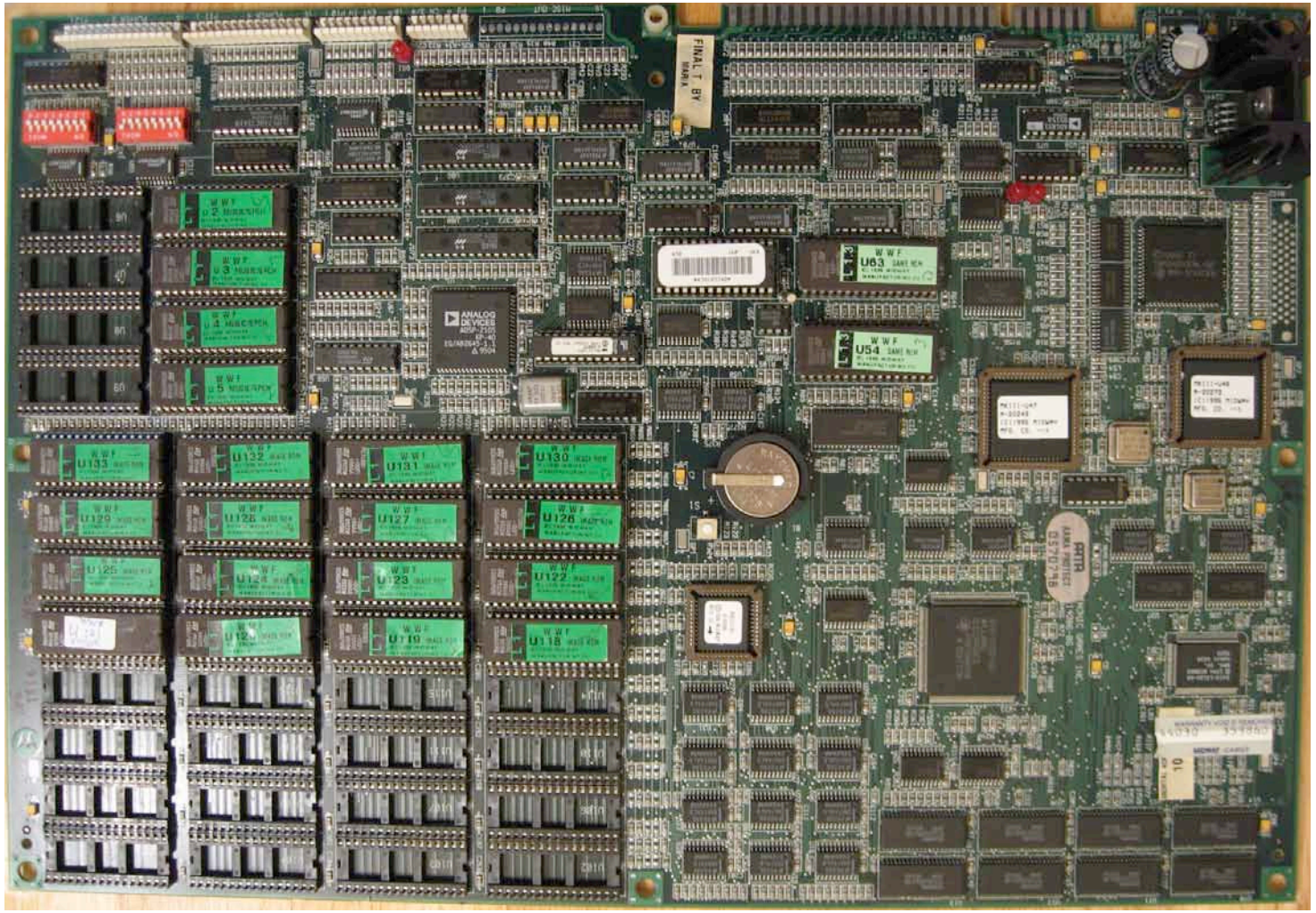


Source: http://www.crazykong.com/pcbs/G%20-%20O/MortalKombat2Upgrade.pcb.jpg

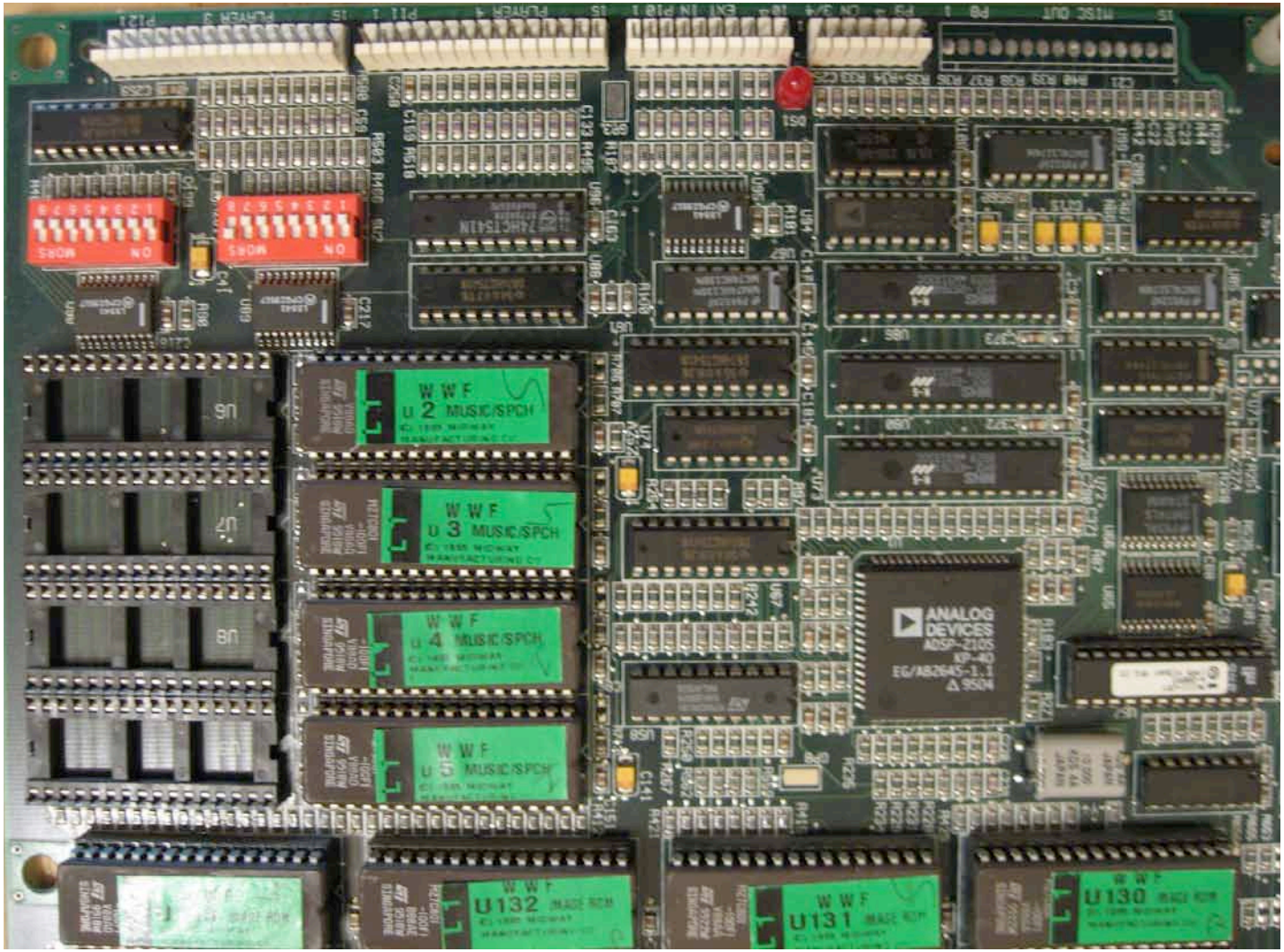
T Unit (MKII Version)

Source: http://www.tvspels-nostalgi.com/Bilder/PCB/romboard_mortal%20kombat%202.jpg

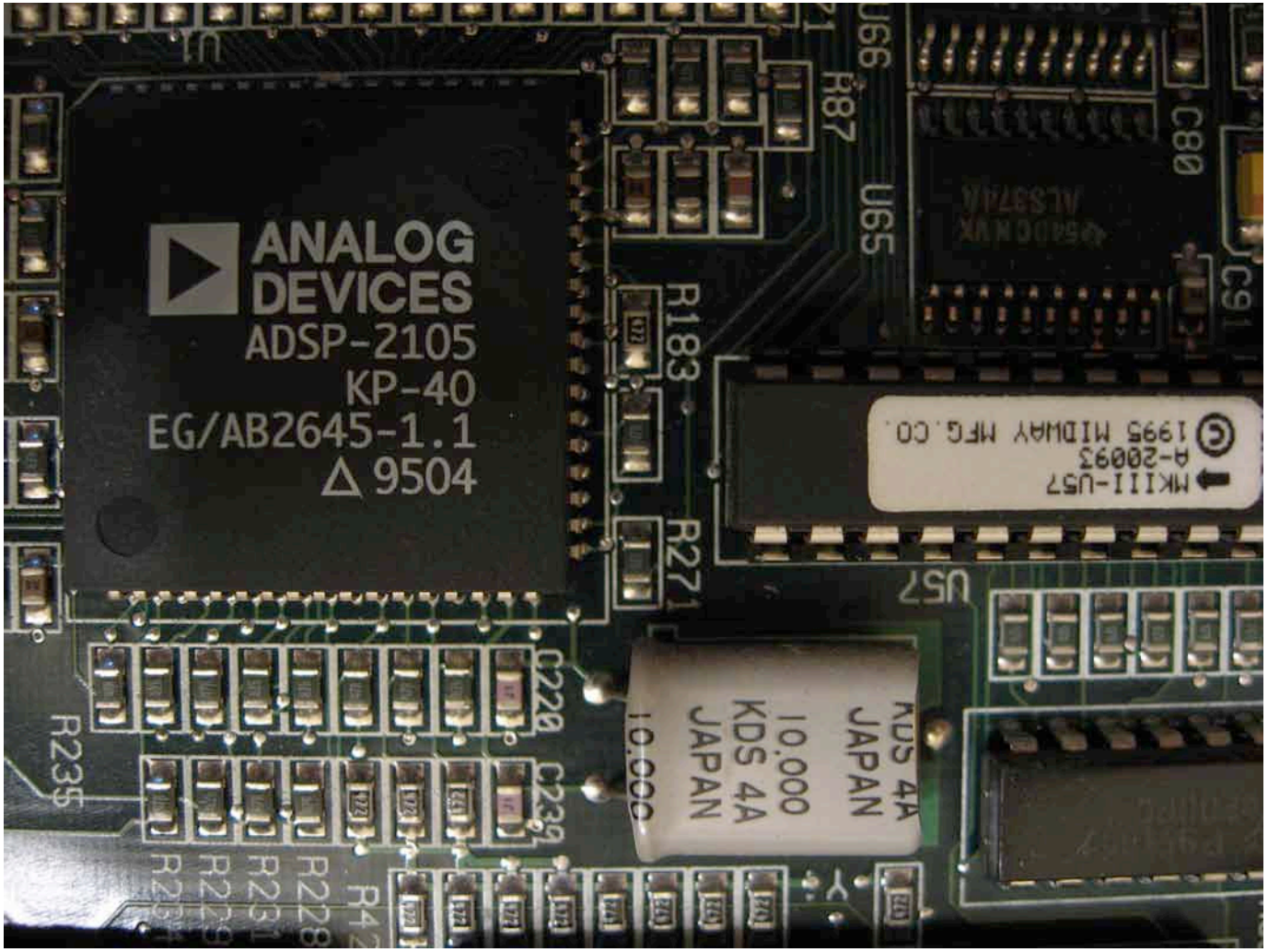
Wolf Unit (ie, MK3)



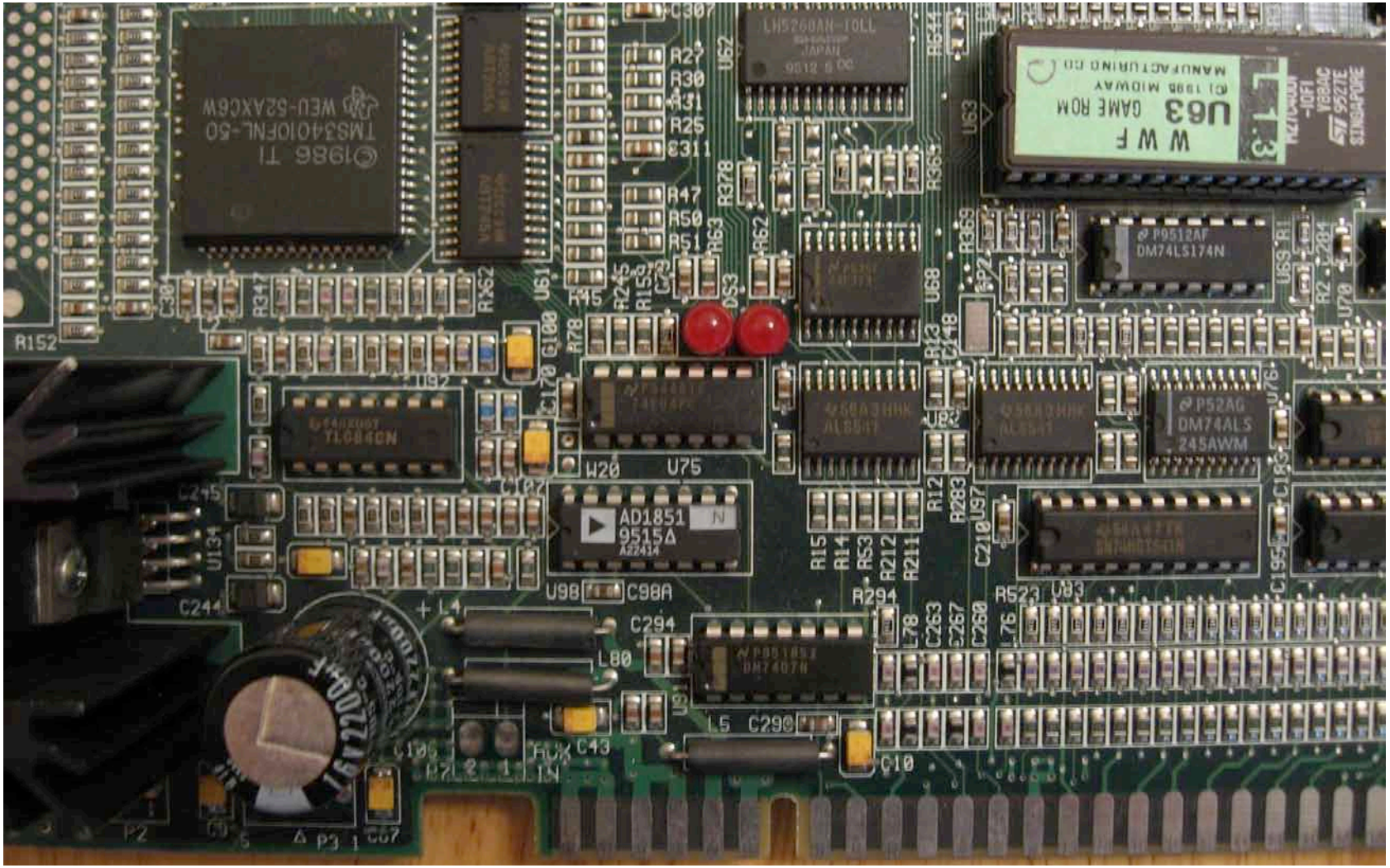
Wolf Unit (ie, MK3)



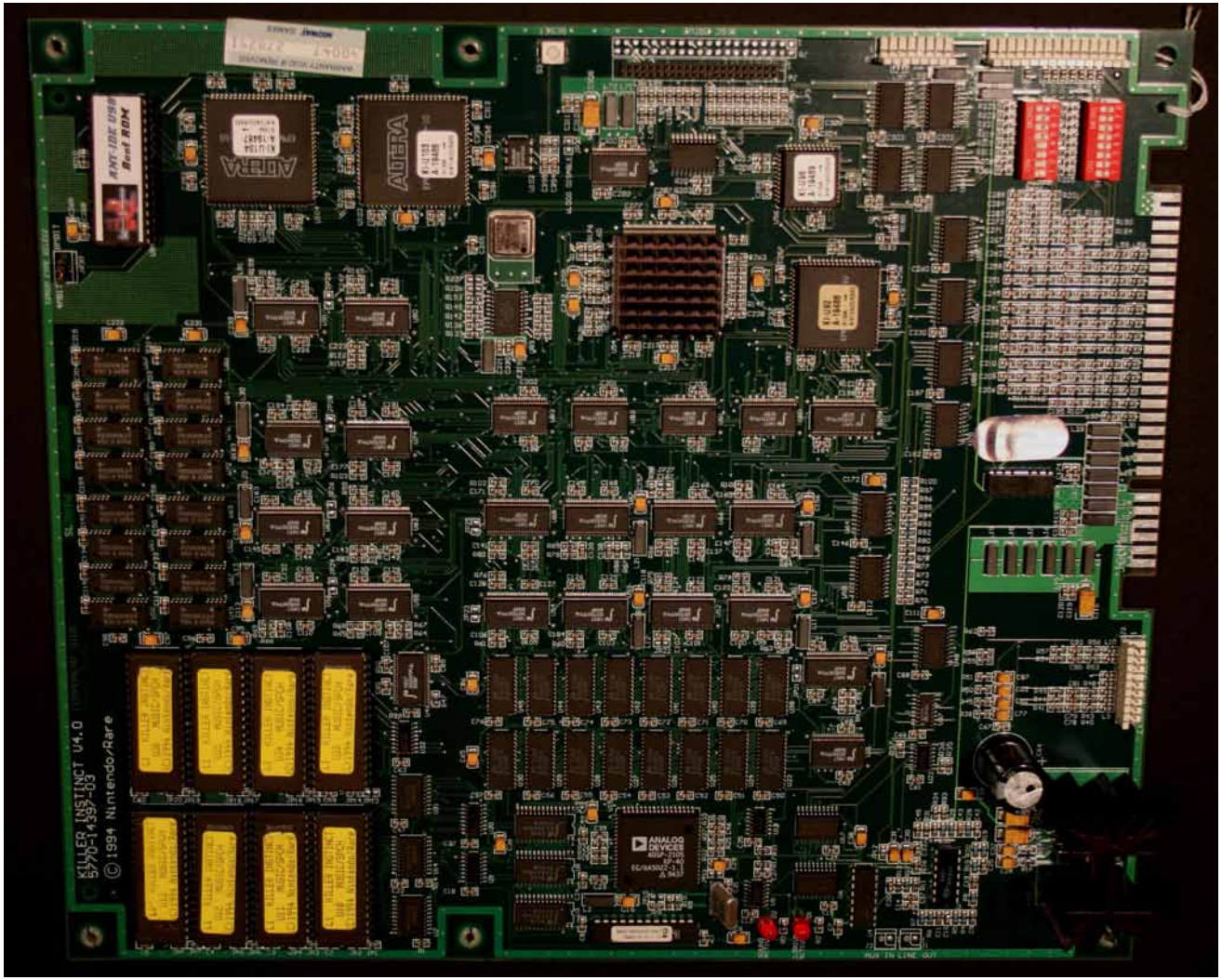
Wolf Unit (ie, MK3)



Wolf Unit (ie, MK3)

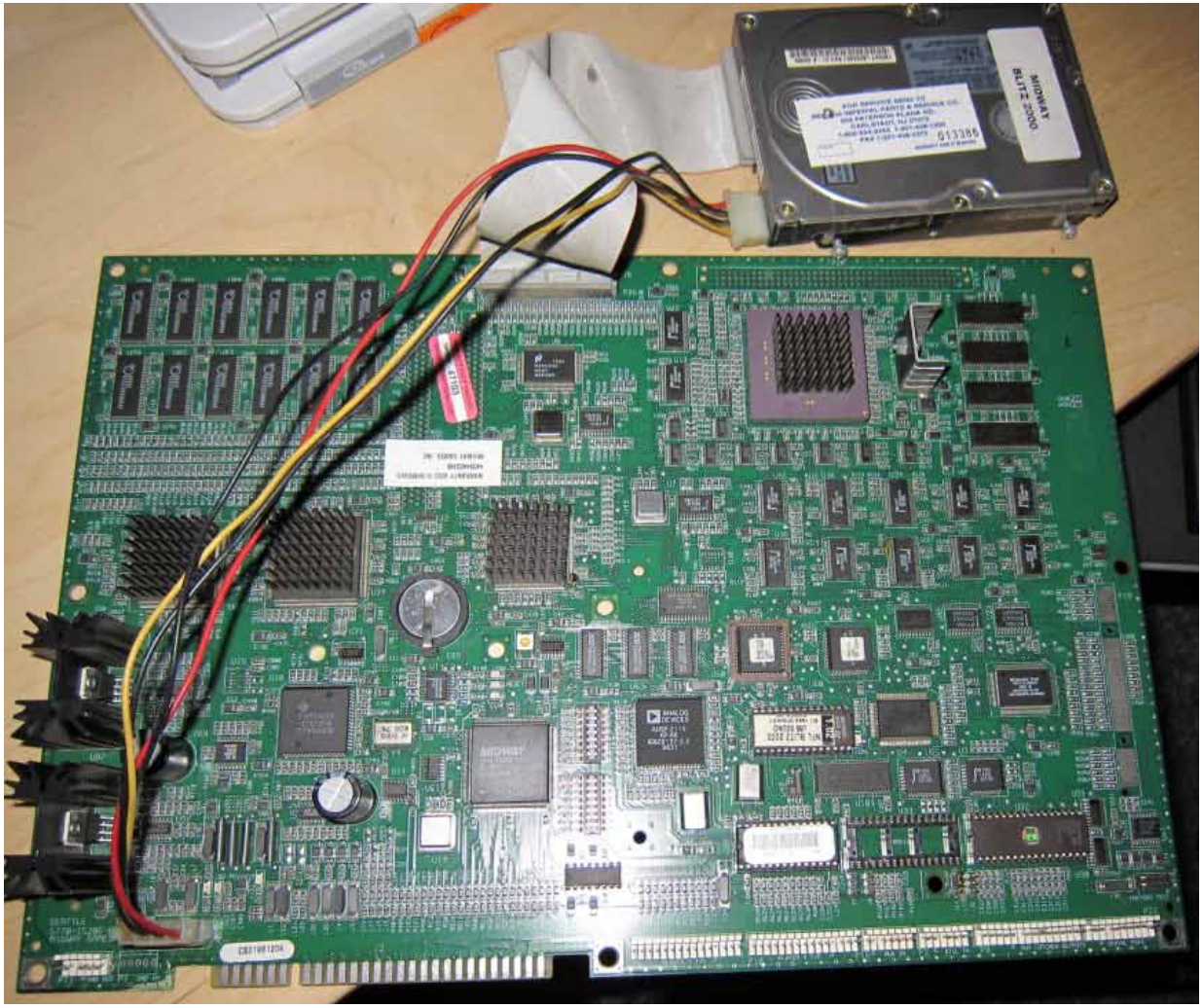


Killer Instinct



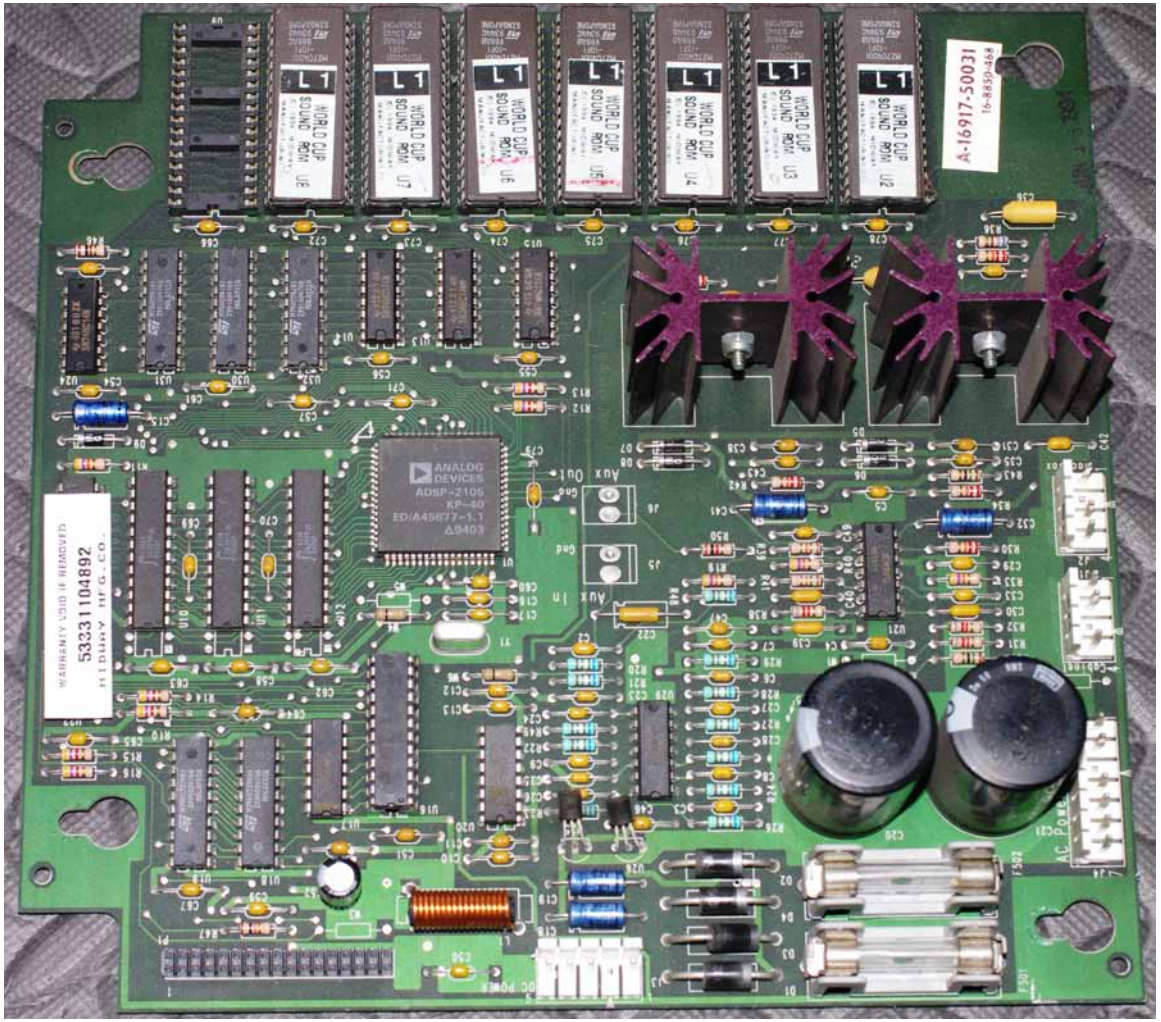
Source: http://www.thekillerinstinctproject.com/kiproject/images/kipcb.jpg

Seattle (ie, NFL Blitz 2000)



Source: http://s89.photobucket.com/user/yodafrommn/media/IMG_0435.jpg.html

World Cup Soccer Pinball



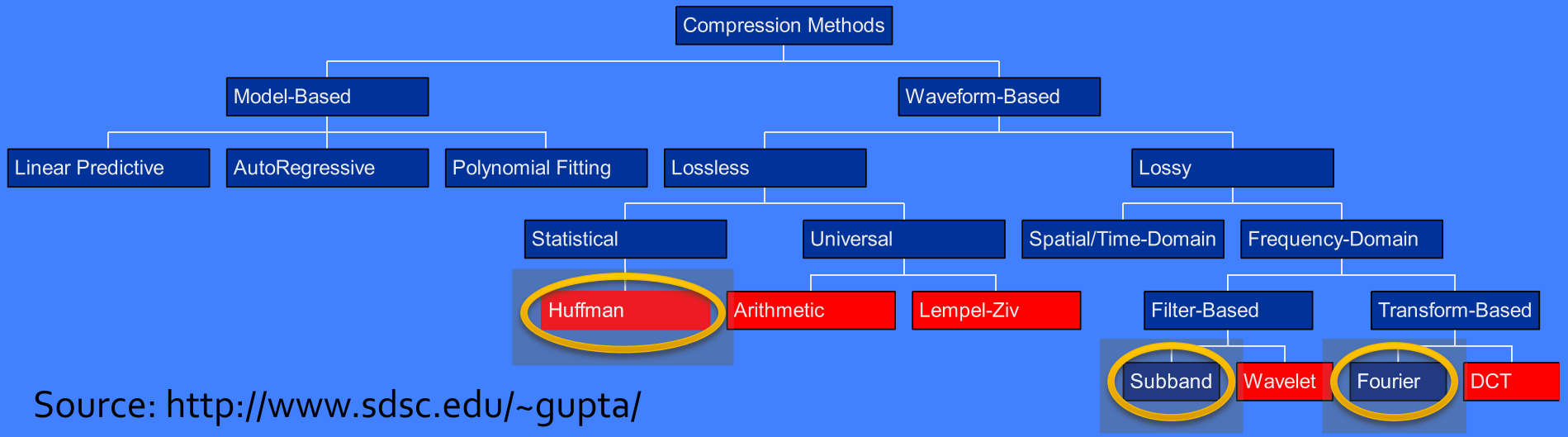
Source: http://pinwiki.net/images/1/11/WPC_DCS_Sound_Board.jpg

DCS System Design

- # of output channels – mono, stereo, multi-channel
- Polyphony / "voices" – 6
- Sample rate – 31250 Hz
- Bit depth – 16-bit
- Time-domain to frequency-domain (FFT) – in hardware (DSP)
- Sub-bands - 16
- Quantization – variable # of bits allocated per sub-band
- Entropy encoding – Lossless Prefix coding (Huffman coding)
- Bitstream generation
  - Look-up table entries pointing to sound headers
  - Sound header points to frame header
  - Frame header includes length & sub-band quantization values
  - Variable-length compressed audio data
- Image (ROM) creation
  - DSP bootloader/initialization code, program code, data
  - Compression codeword-symbol dictionaries & compressed audio
  - Data split into images & burned to individual EPROMs

Compression Background

- Fast Fourier Transform (FFT) used to convert input time-domain samples to frequency-domain
- Sub-band coding used to distribute bits based on the frequency bins
  - High frequencies less perceivable, so need less bits
- Lossless entropy coding (Huffman) used to pack samples into variable-length data stream



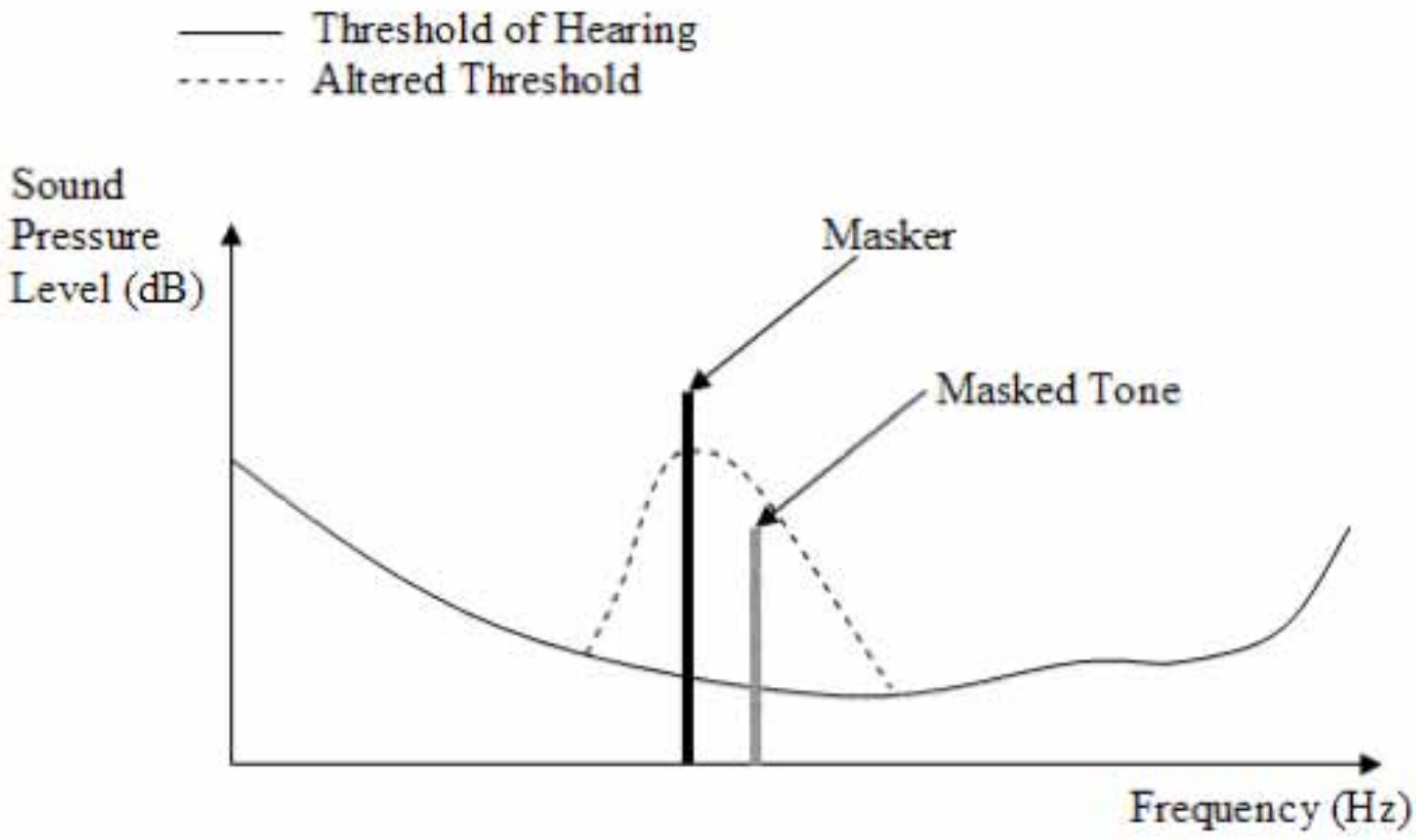
Source: http://www.sdsc.edu/~gupta/

Compression Background

- By identifying what can and can't be heard, audio compression discards information that can't be perceived
- Knowing that signals below a particular amplitude at a particular frequency are not audible, you can hide quantization noise from your brain
- A tone at a certain frequency will raise the *threshold of audibility* in a *critical band* around that frequency
- Temporal *masking* is the masking that occurs when a sound raises the *threshold of audibility* for a brief interval preceding and following the sound
- Vs. traditional lossless methods like ZIP or RAR, which don't discard data, and won't compress as small

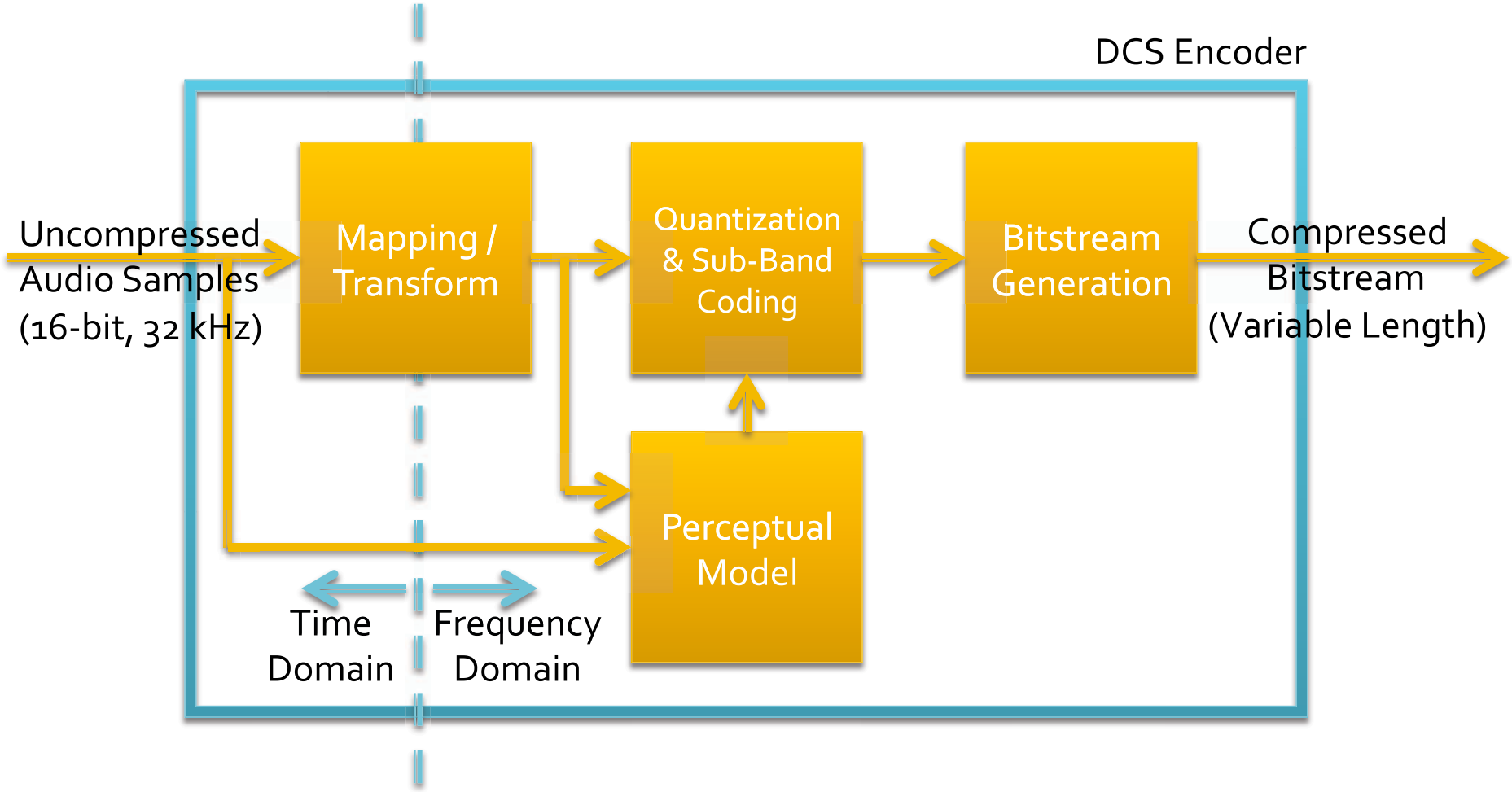
Source: http://www.ece.mcmaster.ca/~shirani/multi12/audiocompression.pdf

Compression Background

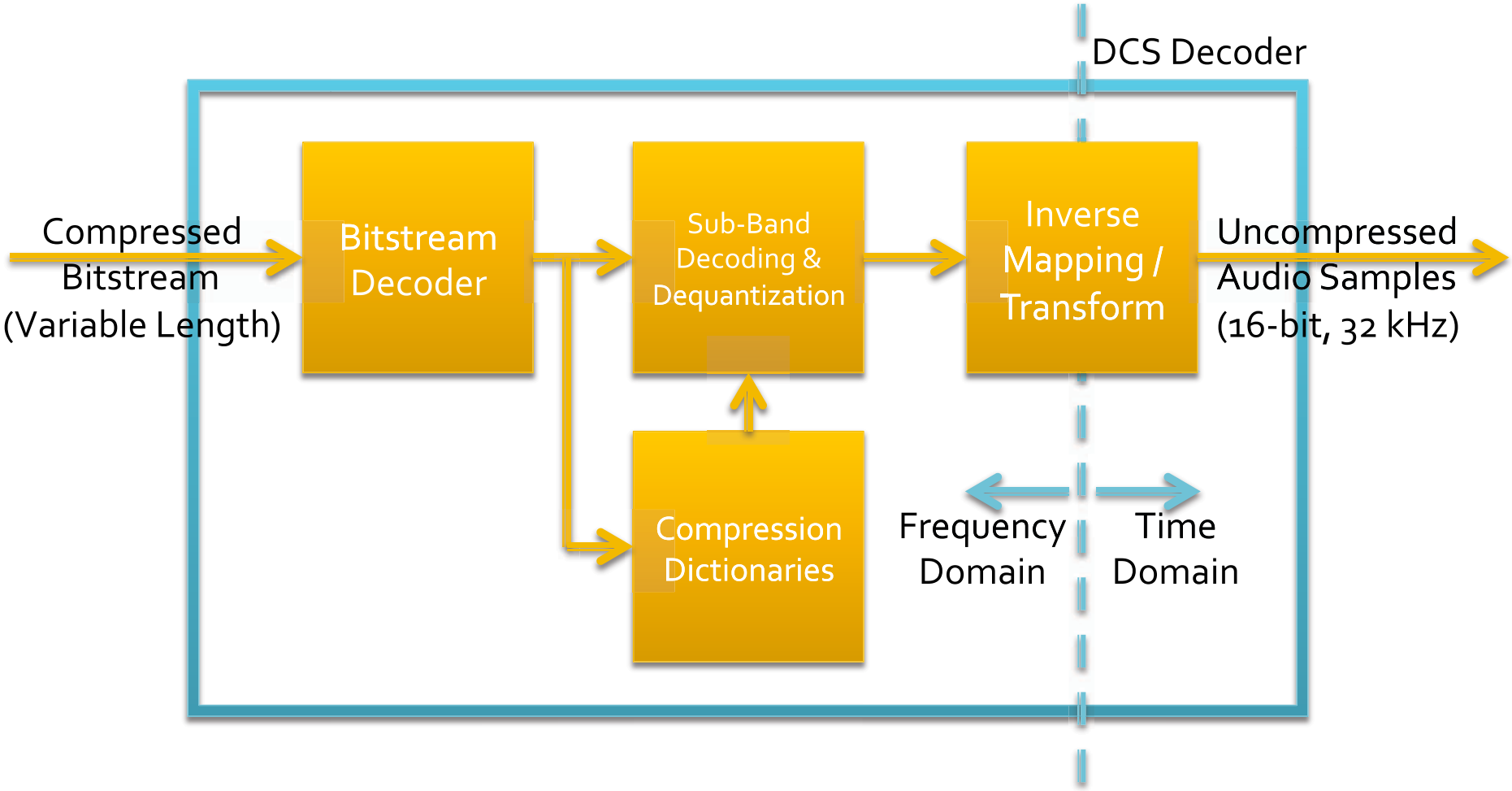


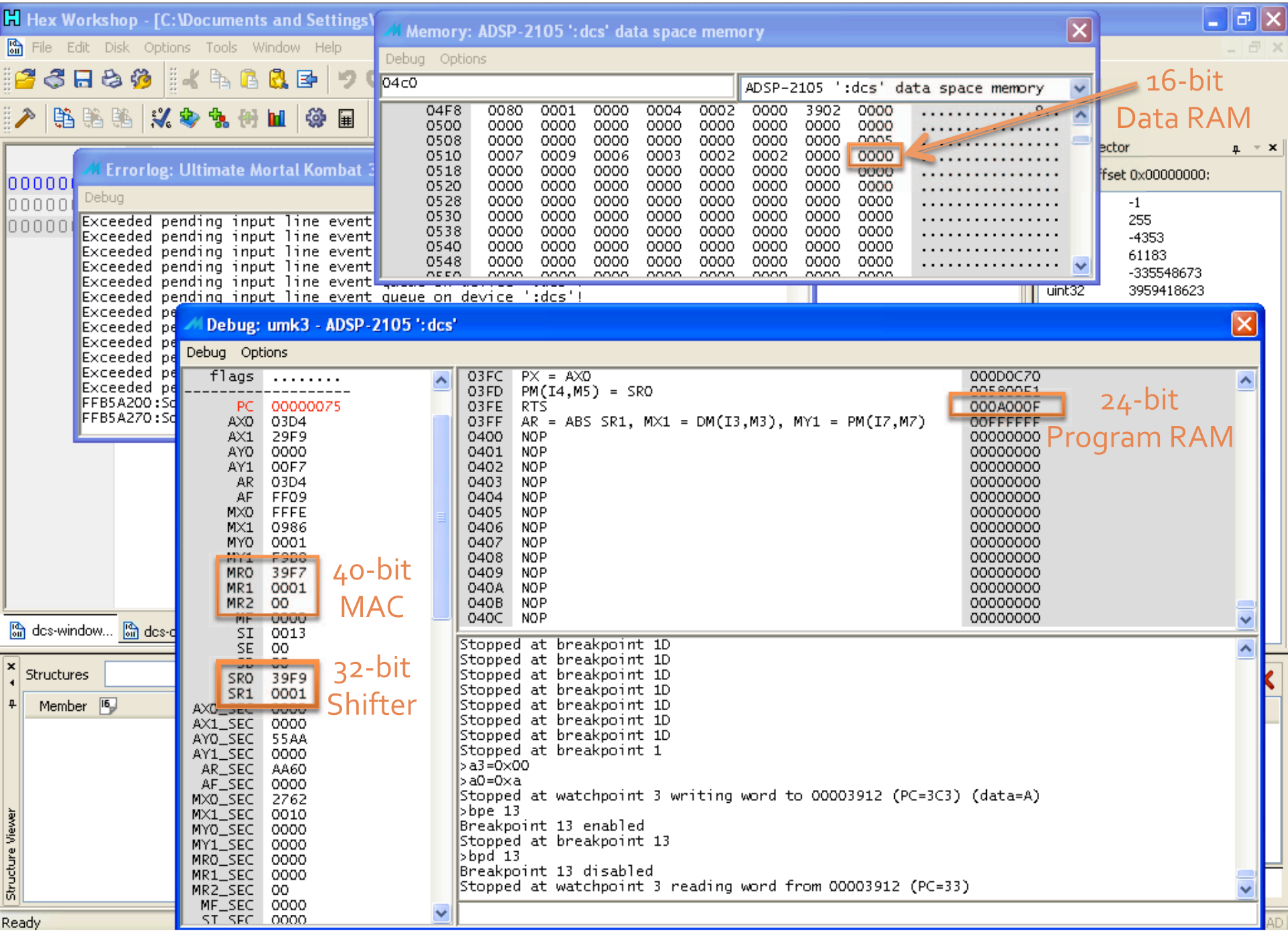
Source: http://www.ece.mcmaster.ca/~shirani/multi12/audiocompression.pdf

DCS Encoder


DCS Encoder
Uncompressed
Audio Samples
(16-bit, 32 kHz)
Mapping /
Transform
Quantization
& Sub-Band
Coding
Bitstream
Generation
Compressed
Bitstream
(Variable Length)
Perceptual
Model
Time
Domain
Frequency
Domain

DCS Decoder


DCS Decoder
Compressed Bitstream (Variable Length)
Bitstream Decoder
Sub-Band Decoding & Dequantization
Inverse Mapping / Transform
Uncompressed Audio Samples (16-bit, 32 kHz)
Compression Dictionaries
Frequency Domain
Time Domain


Hex Workshop - [C:\Documents and Settings\
File Edit Disk Options Tools Window Help
Memory: ADSP-2105 ':dcs' data space memory
Debug Options
04c0
ADSP-2105 ':dcs' data space memory
16-bit
Data RAM
04F8 0080 0001 0000 0004 0002 0000 3902 0000
0500 0000 0000 0000 0000 0000 0000 0000 0000
0508 0000 0000 0000 0000 0000 0000 0000 0000
0510 0007 0009 0006 0003 0002 0002 0000 0000
0518 0000 0000 0000 0000 0000 0000 0000 0000
0520 0000 0000 0000 0000 0000 0000 0000 0000
0528 0000 0000 0000 0000 0000 0000 0000 0000
0530 0000 0000 0000 0000 0000 0000 0000 0000
0538 0000 0000 0000 0000 0000 0000 0000 0000
0540 0000 0000 0000 0000 0000 0000 0000 0000
0548 0000 0000 0000 0000 0000 0000 0000 0000
Errorlog: Ultimate Mortal Kombat
Debug
Exceeded pending input line event queue on device ':dcs'!
Debug: umk3 - ADSP-2105 ':dcs'
Debug Options
flags ....
PC 00000075
AX0 03D4
AX1 29F9
AY0 0000
AY1 00F7
AR 03D4
AF FF09
MX0 FFFE
MX1 0986
MY0 0001
MR0 39F7
MR1 0001
MR2 00
SI 0013
SE 00
SR0 39F9
SR1 0001
AX1_SEC 0000
AY0_SEC 55AA
AY1_SEC 0000
AR_SEC AA60
AF_SEC 0000
MX0_SEC 2762
MX1_SEC 0010
MY0_SEC 0000
MY1_SEC 0000
MR0_SEC 0000
MR1_SEC 0000
MR2_SEC 00
MF_SEC 0000
40-bit
MAC
32-bit
Shifter
03FC PX = AX0 000D0C70
03FD PM(I4,M5) = SR0
03FE RTS 000A000F
03FF AR = ABS SR1, MX1 = DM(I3,M3), MY1 = PM(I7,M7) 00FFFFFF
0400 NOP 00000000
0401 NOP 00000000
0402 NOP 00000000
0403 NOP 00000000
0404 NOP 00000000
0405 NOP 00000000
0406 NOP 00000000
0407 NOP 00000000
0408 NOP 00000000
0409 NOP 00000000
040A NOP 00000000
040B NOP 00000000
040C NOP 00000000
24-bit
Program RAM
Stopped at breakpoint 1D
Stopped at breakpoint 1D
Stopped at breakpoint 1D
Stopped at breakpoint 1D
Stopped at breakpoint 1D
Stopped at breakpoint 1D
Stopped at breakpoint 1D
Stopped at breakpoint 1
>a3=0x00
>a0=0xa
Stopped at watchpoint 3 writing word to 00003912 (PC=3C3) (data=A)
>bpe 13
Breakpoint 13 enabled
Stopped at breakpoint 13
>bpd 13
Breakpoint 13 disabled
Stopped at watchpoint 3 reading word from 00003912 (PC=33)
Structures
Member
Structure Viewer
Ready
uint32
3959418623
-335548673
61183
-4353
255
-1

DSP Boot & Initialization

Stage 1

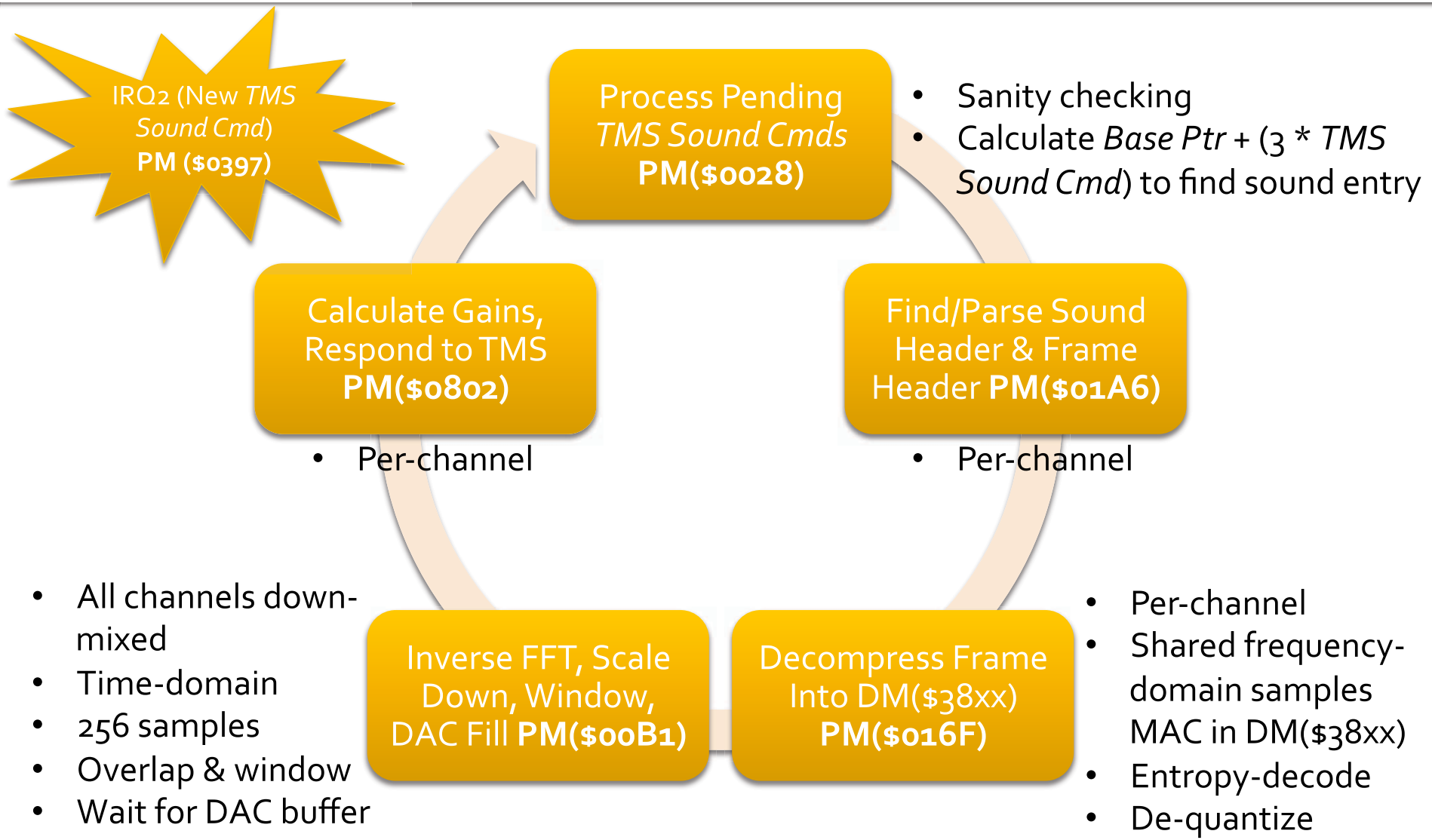
- Boot from ROM U2, bank 0
- Set *data memory* wait states
- Install IRQ handlers (level-triggered)
- Enable IRQ2 (TMS CPU) & wait
- Disable all IRQs
- Wipe internal *data* RAM
- Replace reset vector
- Soft reboot

Stage 2

- Wipe DAC sample buffers $0C00/$0CF0
- Copy 0x200 3 out of 4 bytes from U2, 0x800 to PM($0800) as 24-bit words
- Jump to new code at PM($0800)
- Wipe ext. *data* RAM $0300-$07FF
- Wipe int. *data* RAM $3800-$39FF
- Configure serial port for 16-bit, 500 kbaud, *Frame Sync* pin for DAC *LE*, auto-buffer from $0C00 DAC buffer
- Enable IRQ2 (TMS CPU)
- Copy *twiddle factors* from PM($0900) to DM($0700)
- Copy various compression dictionaries & other look-up tables from PM to DM
- Copy max allowed *TMS sound cmd* from U2, 0x4046 into DM($04CA)
- Copy 0x300 3 out of 4 bytes from U2, 0x2000 to PM($0800) as 24-bit words (decompression code)
- Enter main loop

Run

DSP Main Code Loop



- Sanity checking
- Calculate *Base Ptr* + (3 * *TMS Sound Cmd*) to find sound entry

- Per-channel

- Per-channel
- Shared frequency-domain samples MAC in DM($38xx)
- Entropy-decode
- De-quantize

- All channels down-mixed
- Time-domain
- 256 samples
- Overlap & window
- Wait for DAC buffer

1. Process *TMS Sound Commands*

PM($0028)

- Wipe previous contents of internal *data* RAM $38xx used for sample decoding & IFFT
- Check if any new *sound commands* have arrived from TMS CPU in DM($391x) buffer
  - IRQ2 fires when TMS sends data to DSP
  - Handler validates command & stores in buffer
  - TMS 0x16800000 mapped to DSP DM(0x3400)
- Verify *sound cmd* is in valid range for game
- Find & store pointer to valid *sound header*
- Check for more pending *sound cmds*

2. Find/Parse Sound & Frame Headers

PM($01A6)

- *TMS sound cmd* (ie, $0429) expands to 0x4048 + (3 * $0429) = $4CC3 (U2, bank 4)
- Sound LUT entry at $4CC3 is: 0x18183

- U2, offset 0x18183 contains *sound header*
  - 01 01 00 00 07 01 69 00 00 01 01 **10 47 54** 01 **01 47** 00
    - Sound contains 0x0147 frames
    - Pointer to frame header (bank $0104 = U3 offset 0x04754)

- *Sound header* points to *frame header* (U3, 0x04754)
  - **01 47** 95 95 91 11 11 11 11 11 11 51 51 FF FF FF FF FF
    - 11 non-$7F/$FF values indicate 11 sub-bands in frame (of 16 possible)

2. Find/Parse Sound & Frame Headers

PM($01A6)

- Compressed *sub-band de-quantization values* follow after *frame header*
- *Compressed audio data* follows these values

- Remaining frames consist of de-quantization values and compressed audio (no headers)

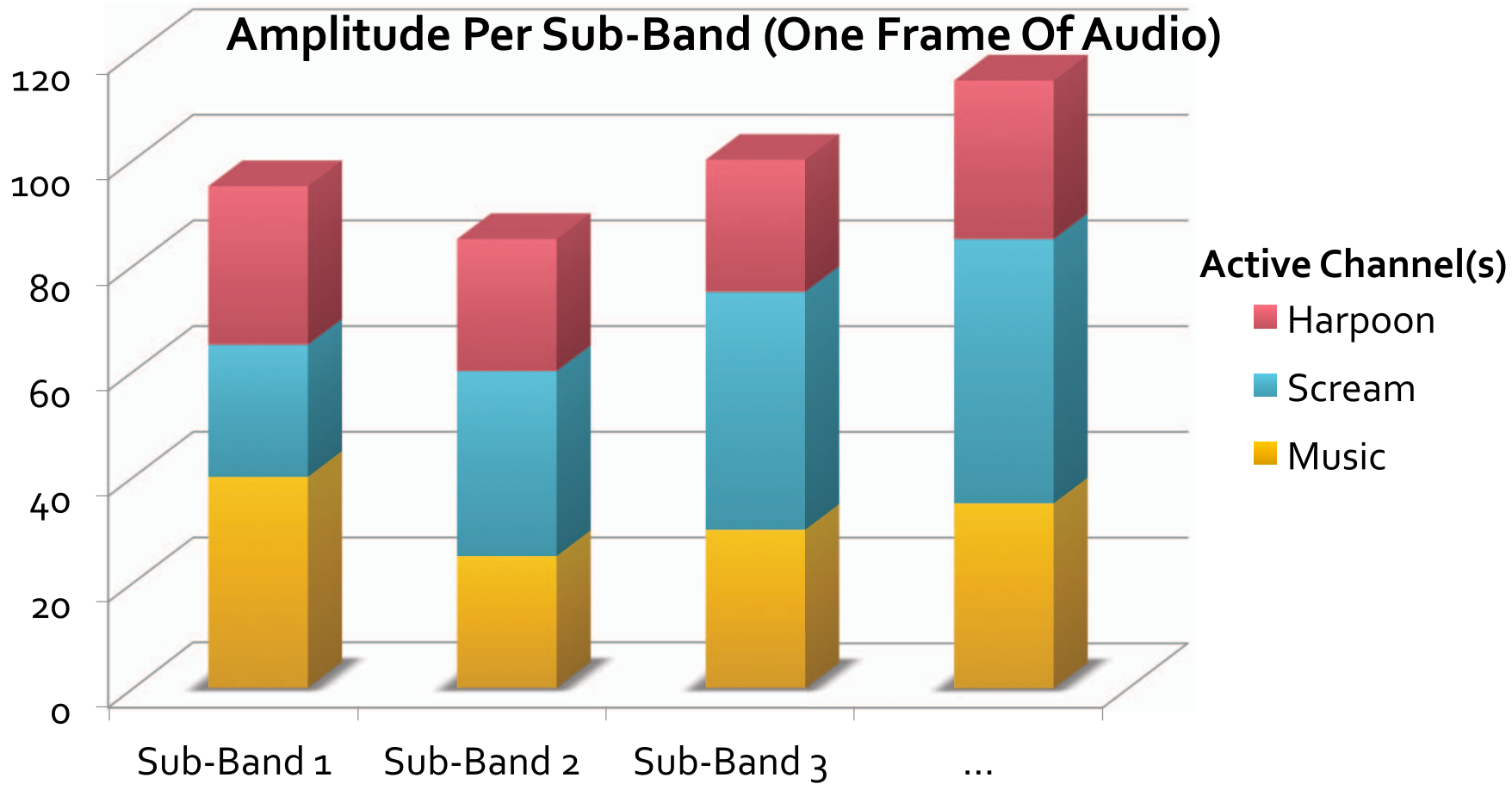
- For each channel
  - Find & parse headers as described
  - Check if all frames of sound processed yet
    - Wipe header pointers when frame done

3. Decompress Frame Into DM($38xx)

PM($016F)

- Call PM($0800) if 1st frame
  - Find the # of sub-band de-quantization values from *frame header*
  - Wipe *data* RAM area that will hold these values
- Call PM($0801) for all other frames
  - Entropy-decode all sub-band values (ie, 11) for frame to DM($04FF) region (delta-encoded after 1st frame)
  - Entropy-decode (ie, 11) audio codewords into symbols & store to DM($3972) region
  - De-quantize these symbols w/ scaling factor & accumulate w/ other channels' symbols & store to DM($3800) region
- Reach PM($0189) when all frames processed
  - Wipe current *frame* pointer & *# of decoded frames* counter

Accumulation in Frequency-Domain

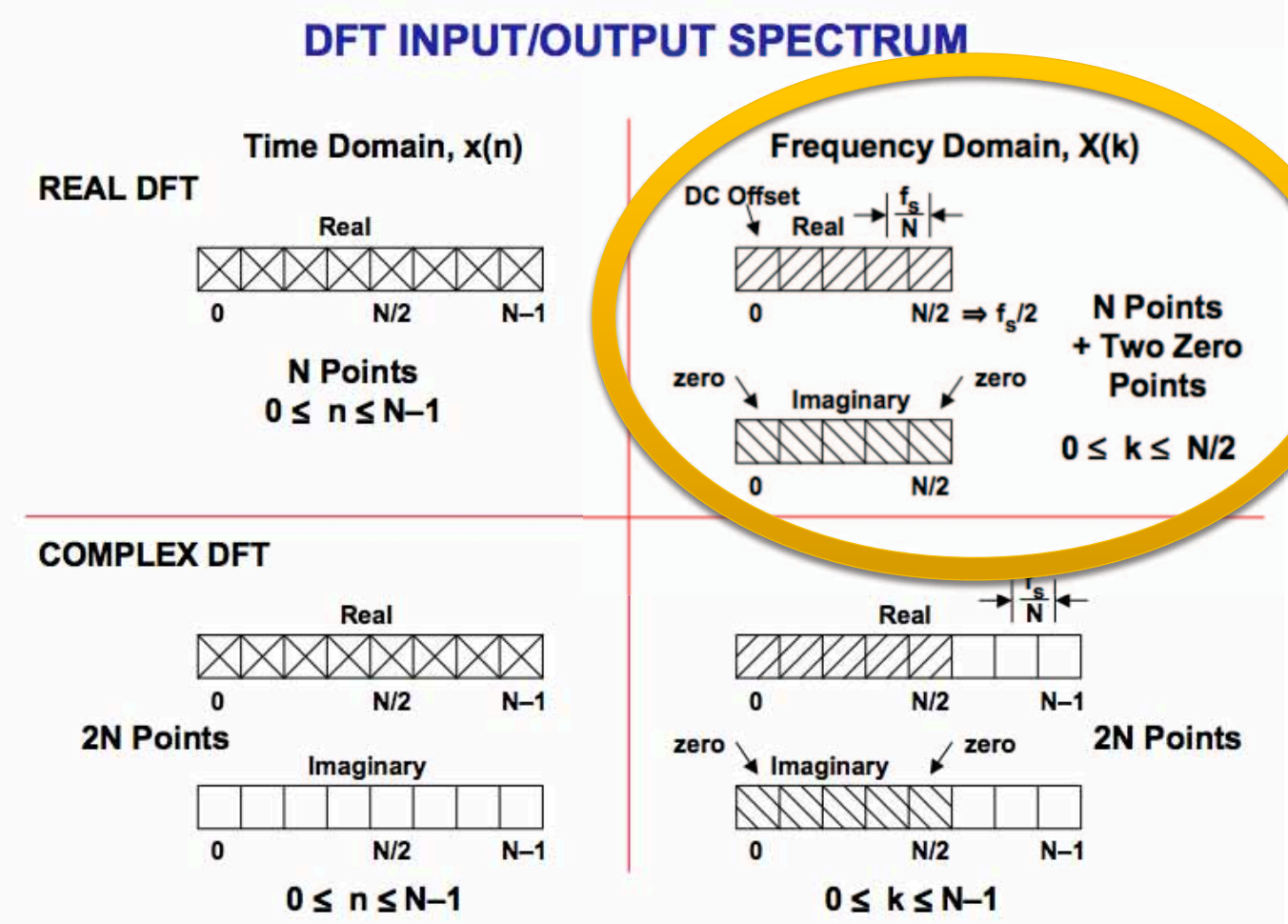


4. IFFT, Scale Down, Window, DAC Fill

PM($00B1)

- Perform 256-point inverse-FFT on frequency-domain samples in DM($3800) region
  - 1st & 2nd iterations are unrolled
    - Bit-reversed addressing mode used for accessing *twiddle factors* in scrambled order
  - Remaining 6 iterations are looped
  - 8 total iterations ($2^{8\ \text{iterations}} = 256$ points)
- Scale time-domain samples down by 1/256 or 1/sqrt(256), based on # active channel(s)
- Apply smoothing cosine window to samples
- MAC in 8 samples of overlap on each side (16 total) from prior frame (avoids audible artifacts)

Real-Valued FFT Array Layout



Source: Analog Devices Mixed-Signal Design Handbook

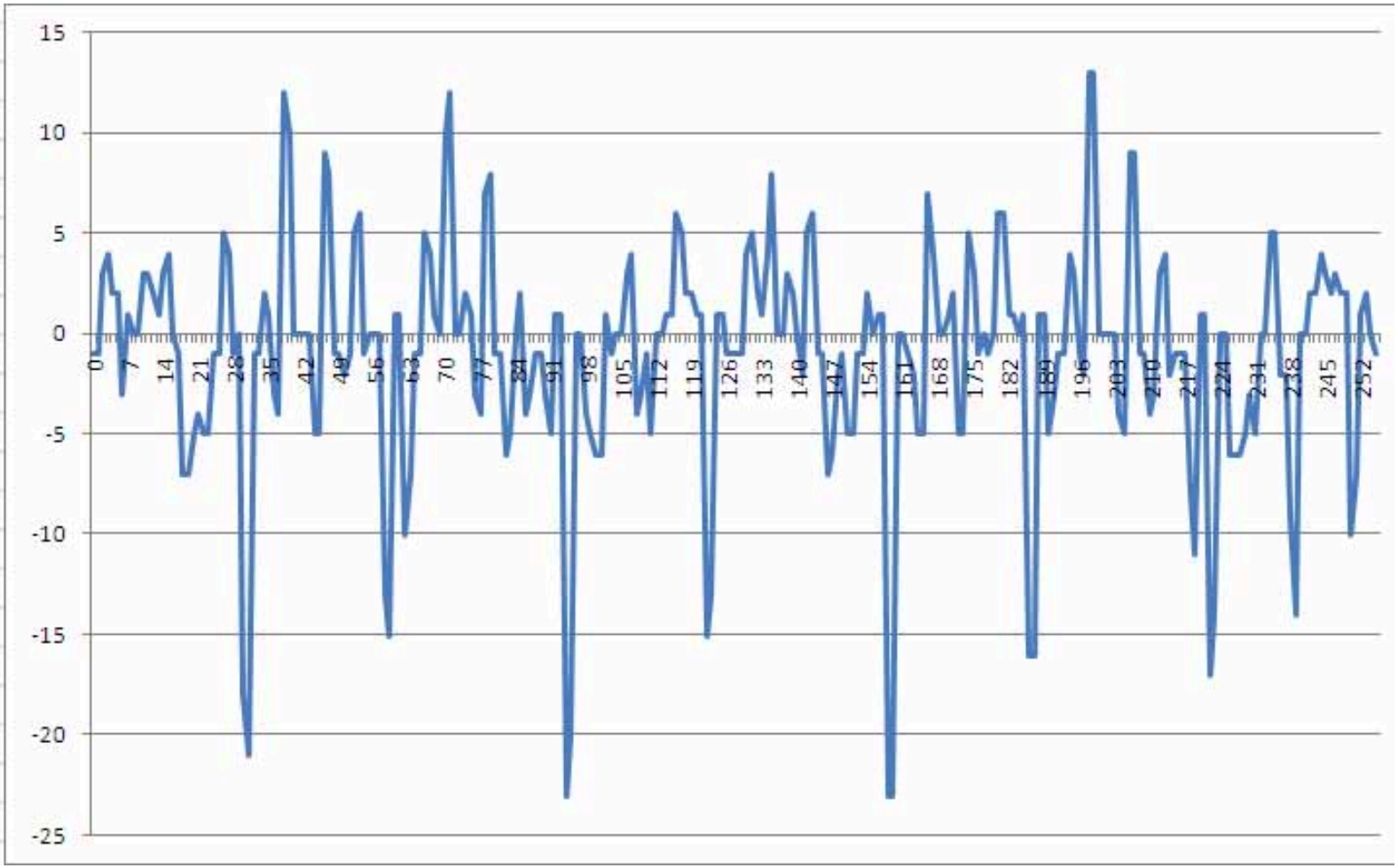
4. IFFT, Scale Down, Window, DAC Fill

PM($00B1)

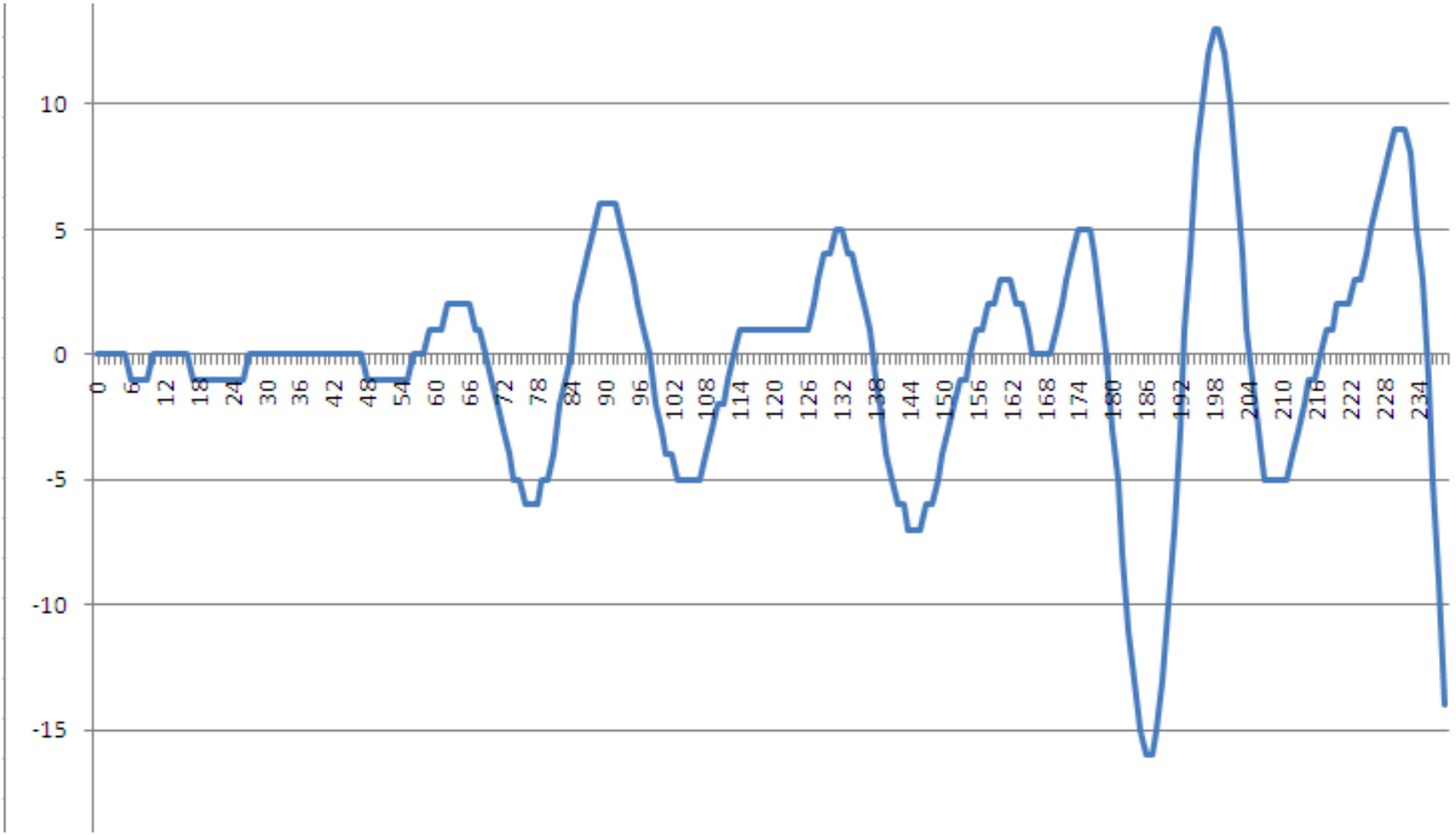
- Wait for serial port to finish clocking out existing sample buffer to DAC
  - DM($0C00) or DM($0CF0), whichever is currently in use
- Copy 240 time-domain samples to standby DAC buffer
- Keep remaining 16 samples for next frame's overlap
- When it's time, serial port treats standby buffer as active & clocks out to DAC

- Trivia
  - If no sound is requested, DSP calculates IFFT of zeroes, copies zeroes to DAC buffer, & clocks zeroes to DAC
  - 500 kHz DAC clock / 16-bit samples = 31250 Hz sample rate
  - DAC is termed *DMA-driven*, since DSP buffers are in external SRAM (not on-chip) & DSP ALU/MAC not involved in transfer

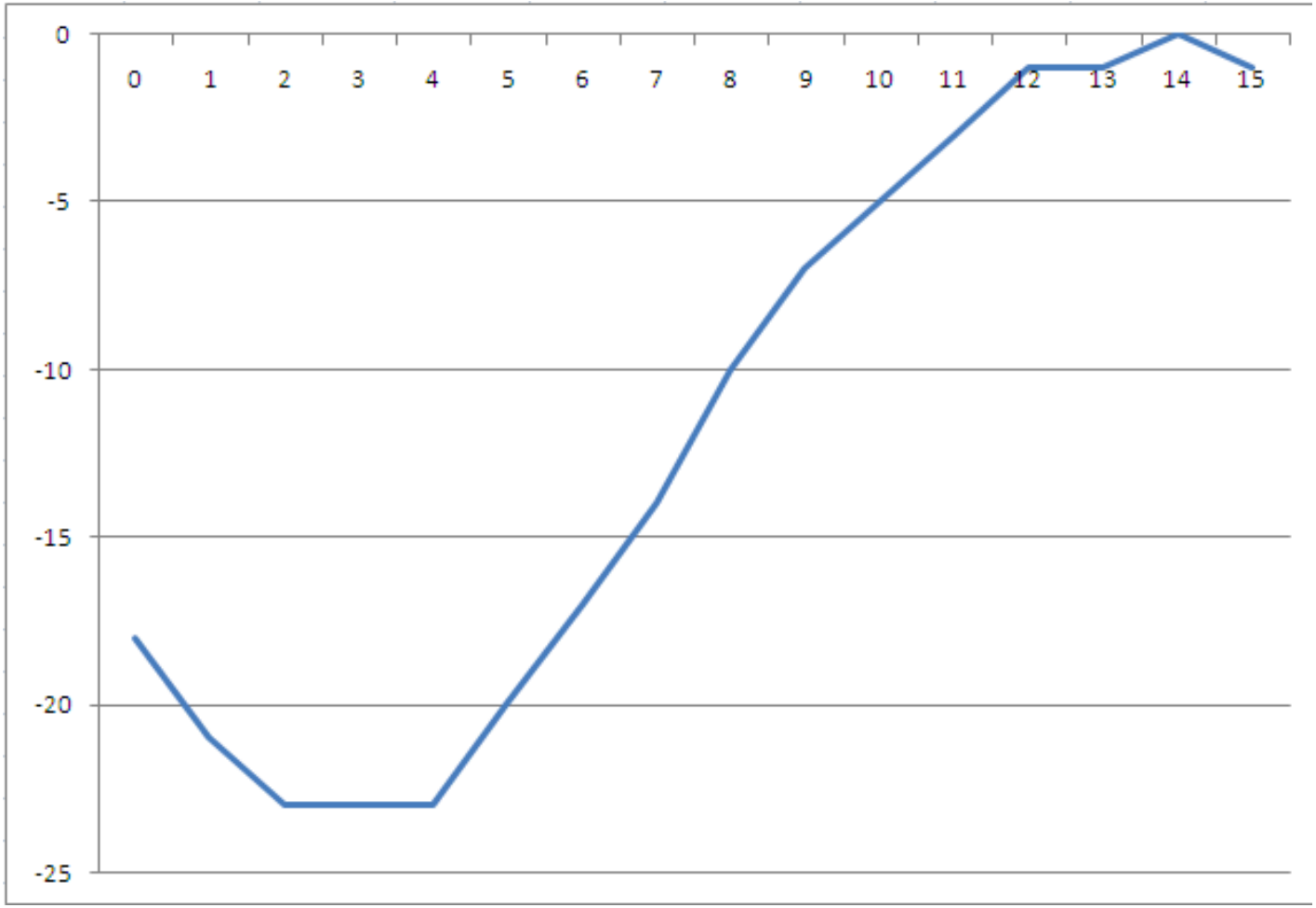
Frequency-Domain Samples



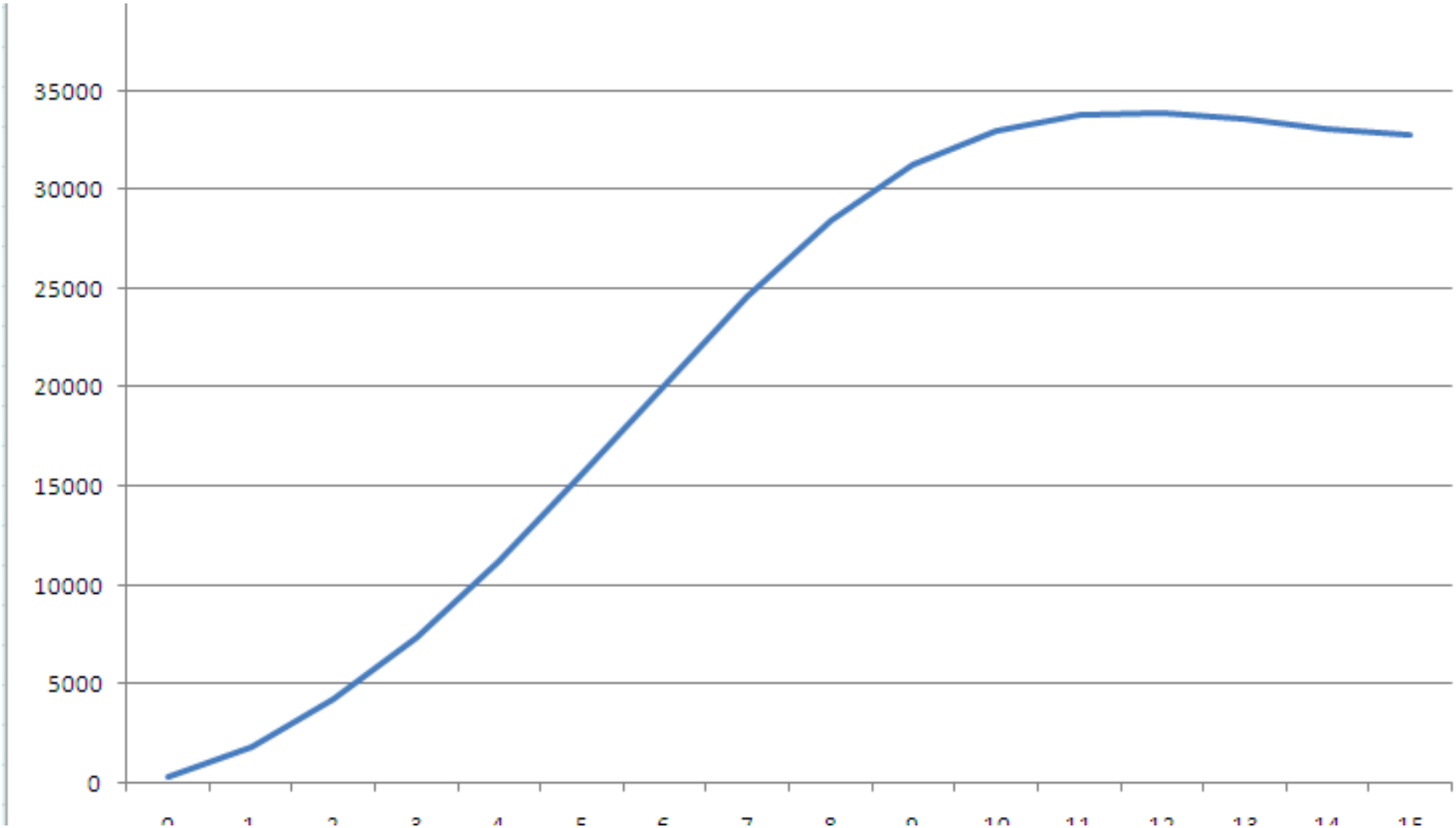
Time-Domain Samples



Leftover 16 Samples



Smoothing Cosine Window



5. Calculate Gains, Respond to TMS

PM($0802)

- Calculate gains used for channel volumes, cross-fades, sample scaling
- Respond to TMS CPU, if it requested sound timing/looping data from DSP
  - Useful for synchronizing pinball lighting, triggering mechanical flippers, etc.

Conclusion

- Using a low-cost DSP allowed high-efficiency, good quality, audio compression algorithm to be accelerated in hardware
- Provided an ~10-to-1 compression ratio
  - Game audio could still be squeezed into EPROMs (vs. non-solid-state CD-ROMs or hard drives)
- Competing systems were using FM synth and low-quality, short, sample-based playback
- DCS allowed musician the freedom to compose like they were in a traditional studio

Conclusion

- The eternal debate:

Street Fighter vs. MK

Super SF II Turbo (1994)

Mortal Kombat 3 (1995)

Q & A

Thank you!