Jet Propulsion Laboratory
California Institute of Technology

Mitigation of Errors In Computer Systems

Steven M. Guertin
steven.m.guertin@jpl.nasa.gov
818-321-5337
NASA/JPL

Acknowledgment:
This work was sponsored by the JPL Commercial Off The Shelf (COTS) Initiative

© 2016 California Institute of Technology. Government sponsorship acknowledged.

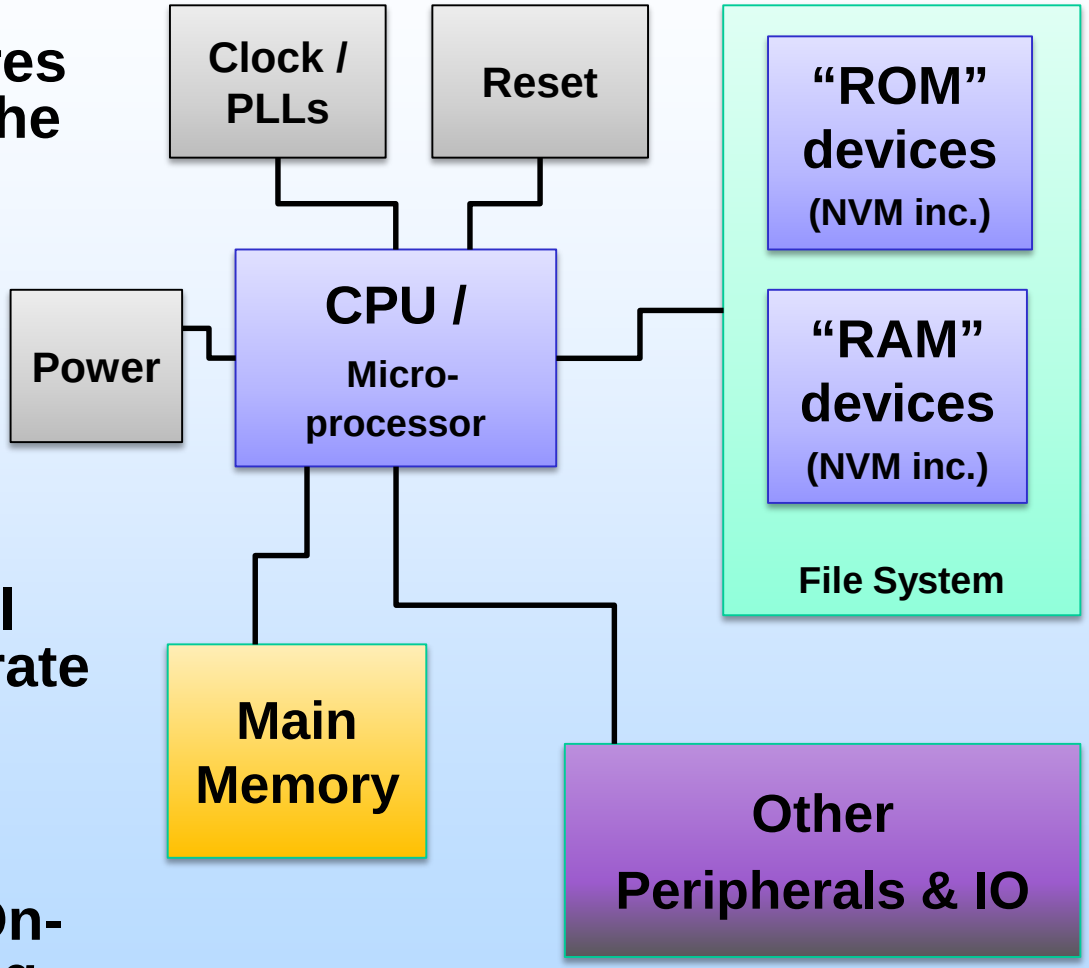
Outline

- Quick overview
- Error Protection in Microprocessors
- Error Correction in Memory Flash and DRAM
- Active Recovery – Power Cycling, Watchdogs
- Redundancy

This is a look at error/fault correction and mitigation in digital systems, with focus on computers.

Computer Block Diagram

- General features are basically the same over the years.
- Our focus is:
  - Processor
  - Memory
- Note modern computers will likely incorporate FPGAs – See Quinn
- And System-On-Chip is blurring all the lines



Mitigation in Space

- **Microprocessor and memory is used a lot**
  - **Error correction in use on early solid-state memory systems**
  - **Solid state recorders from early 1990s were using single error correction, double error detection (SECDED) error detection and correction (EDAC)**
  - **Microprocessor mitigation is relatively new, though fault detection been around – parity**

- **Fielding questions in last 10 years related to minimal radiation performance testing**
  - **System designs for unknown radiation performance**
  - **Worst-case rates for destructive SEE**

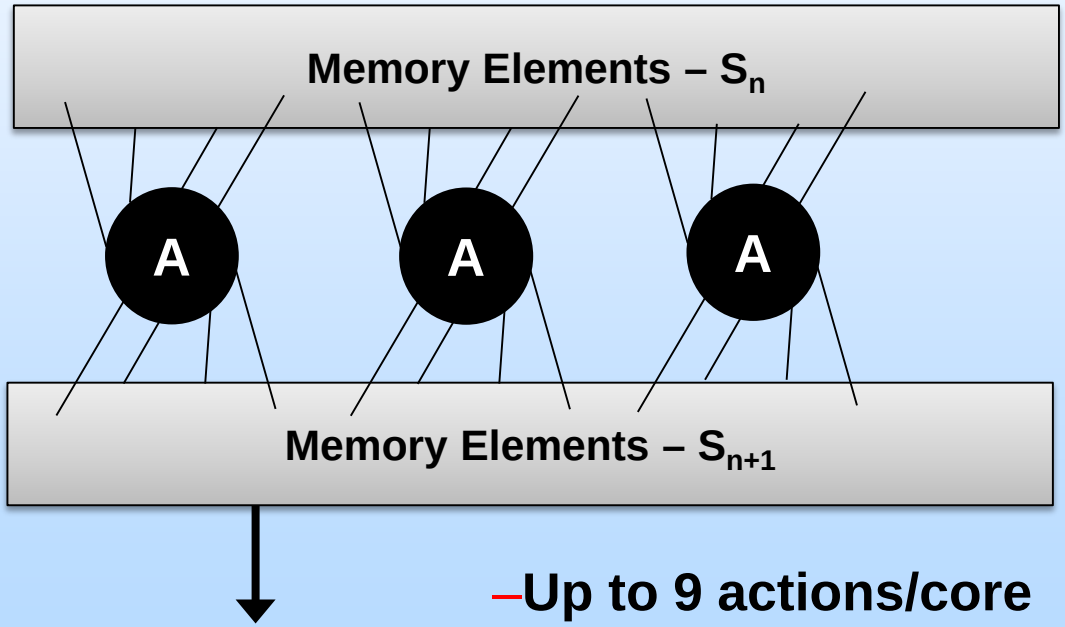
Microprocessor Errors

- Microprocessors are sets of memory elements, and logic units, active to produce a new state from an old state.
  - $S_{n+1} = F [S_n]$
- Side units act to keep the system fed and to store information

–Scratchpads –& More…

–Registers –L1 Cache

–L2 Cache –Configuration



–Up to 9 actions/core every clock cycle

Errors in Memory Elements

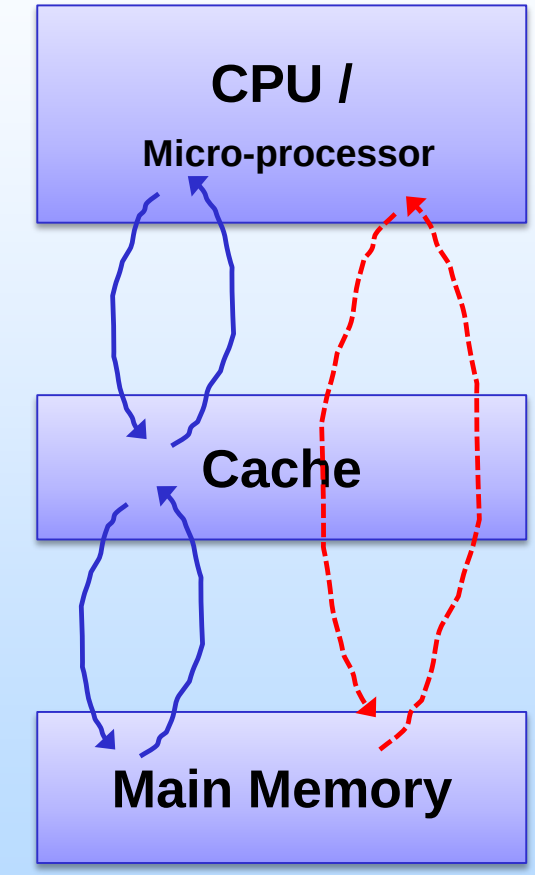
- Most common, by 100x
- L2 caches have highest sensitivity
  - Due to density
  - But also most likely to have error correction

- Protections/error correction
  - L2 caches usually have SECDED these days
    - Though not always on all portions
  - L1 caches have had parity for many years but this is changing
    - Some manufacturers unwilling to tell what type of error correction they use in L1 – L1s in general are mystical…
      - EDAC? Parity? Modified parity? Dynamic power? Clocked?
    - ARM L1 caches can be configured for SECDED
    - L1 caches must respond, with data, within 1 clock cycle (300 ps)… SECDED logic depth (3-4) makes this hard

Biggest Issue - Speed

- For microprocessors in particular:
- Biggest issue is that configuration of error correction could cause 4-60x performance hit

- Many caches cannot correct errors in-place and require re-fetching of data from memory.
- This means the off-chip memory must always be “right”.
- This is not how cache systems are designed.
- To implement this, you have to run in “write-through” which (seriously) impacts performance.



Write Back (fast)

Write Through (slow)

Microprocessor Overview

- Memory elements are the most vulnerable
- Use of error correction in L1 cache is best error mitigation that can be found today
- When memory elements get errors, many things can happen
  - Infinite loops
  - Incorrect branches
  - Mathematics errors
  - Bottom line: corruption, or SEFI (hang/reset/etc)
- Off-chip error mitigation recommended (hangs): watchdog timer
- Data corruption is rare unless other error modes are mitigated

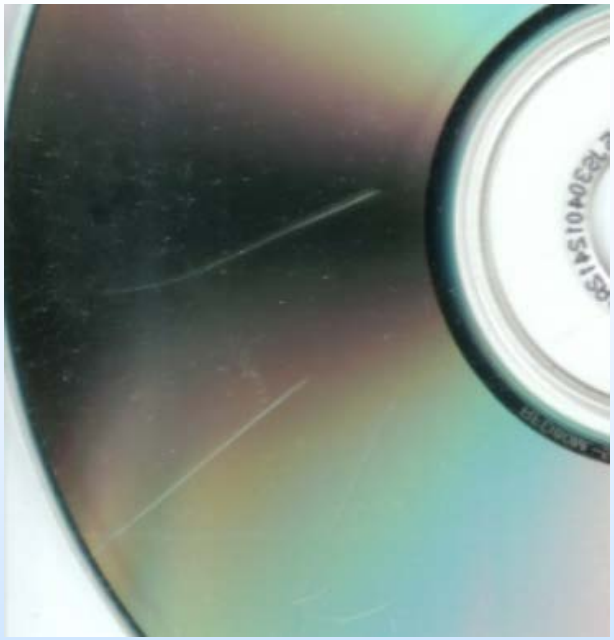
Memory Devices

- Many memory devices are used in computers
- Includes some NVM/ROM devices for bootup and certain types of file storage
- There are several types of error correction that are employed, with varying success
- Some error correction can be employed at the memory interface
  - But anything else must be done with custom hardware in-situ.

- Main memory has data delivery requirements, while file system memory can stall.
  - Although the error modes are the same, the available error correction types are slightly different.

Did you know?

- **CD ROMs are encoded with two levels of error correction.**
  - **Local encoding enables fixing of bit errors**
  - **Global encoding enables fixing of physical damage on the surface, taking out chunks of data**
- **NAND Flash memory requires error correction**
  - **The highest reliability parts require algorithms to correct 10s of bits out of every ~10000 bits.**
  - **MLC Flash has error rates of 1×10^{-6} errors/bit – which can be ~1 million bit errors on a whole device write.**



Error Correcting Codes

- **Hamming Codes**
  - Use parity of subsets of bits to generate ECC bits
  - Can correct single bit errors – and with simple parity addition, they can detect double bit errors
  - Very fast encode and decode (3-5 levels of logic)

- **Reed-Solomon Codes**
  - Symbol-based, with arbitrary length symbols
  - Addition of #t symbols to a set of symbols allows detection and correction of errors in (t/2) of symbols
  - Encoding is not too bad, but decoding and correcting errors can take very long

ECC - 2

- **BCH Codes**
  - **Can correct arbitrary number of bits (depending on the code)**
  - **Decoding is simple, but algorithmic – following an algorithm with an unknown number of iterations (not good for deadline-driven data delivery).**

- **Other codes**
  - **This appears to be a relatively active research field. There are some R&TD efforts (proposals) and Caltech looking at alternatives.**
  - **RAD750 uses something like IBM's chipkill which can correct entire 4-bit symbols (uses many x4 devices)**
  - **And some other organizations have indicated they can do arbitrary ECC.**

Active Recovery

- The main issues we have to deal with are CPU hangs and memory devices going “out to lunch”
- Watchdogs (off-chip) can help with hangs
  - Off-chip
  - Require intelligent system state determination
  - In most cases CPUs can simply be reset, but newer devices may have more problematic modes
- Memory devices – particularly DRAMs may require power cycle to recover
  - Due to SEFIs
  - In newer DRAM technologies everyone is going with 3rd party IP to operate the device, and can only recover at reset of the memory interface

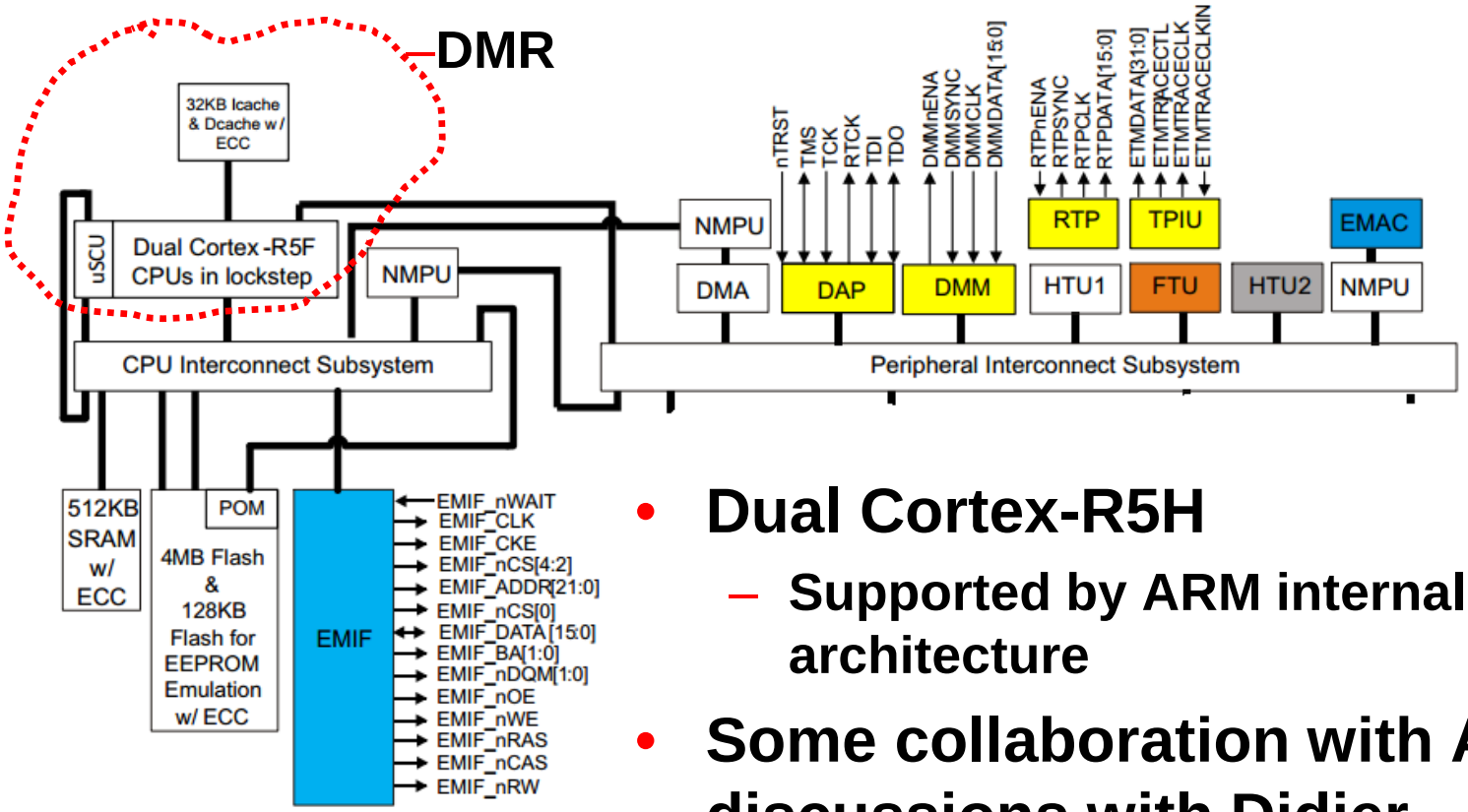
Limited Data

- Some users want to use parts with no data
- Some modern device types (i.e. Snapdragon) are difficult to get good datasets.

IC/System
Rad = ?

- Building systems with limited data, will need to harness dissimilar systems.
  - Redundant systems built with dissimilar components.
- Uncertain how this works with high integration.

Built-In Redundancy



– TI Hercules (partial) block diagram

- **Dual Cortex-R5H**
  - Supported by ARM internal architecture
- **Some collaboration with ARM (via discussions with Didier Keymeyulen**
  - Has student working at ARM in UK
  - Working on TMR ARM architecture

Conclusions - 1

- **Microprocessor error mitigation depends on device architecture**
  - **Prefer EDAC on the L1 cache**
  - **Use watchdogs for hang conditions**
  - **Beware of possible (though rare) data corruption**
  - **Permanent failures are rare due to technology (but not impossible)**
- **NVM/Boot memory, and solid state file storage**
  - **Several error correction schemes are used, though limited applications in space**
  - **For some types (notably Flash), datasheet error protection is sufficient**
  - **Permanent failures are rare as long as TID is limited**

Conclusions - 2

- **Main memory**
  - **Must meet data transfer requirements – which limits error correction**
  - **DRAM devices, in particular are prone to SEFI, which may require power-cycle of the problem device**
  - **Symbol-based correction desired, but slow.**
- **File system memory**
  - **Can use slower error correction schemes**
  - **Robust error correction can be used because the system can stall while correcting – can be implemented in controllers**
- **Redundancy**
  - **Dissimilar builds recommended for limited error information on components**
  - **ARM definitely interested in providing high reliability architecture**