Motivation
1. Implementing an errata today is verbose

Majority set a bit at reset

a) /* Errata Workaround for Cortex A77 Errata #1925769. */
   * This applies to revision <= r1p1 of Cortex A77.
   * Inputs:
   * x0: variant[4:7] and revision[0:3] of current cpu.
   * Shall clobber: x0-x17
   */
   
   func errata_a77_1925769_wa
       /* Compare x0 against revision <= r1p1 */
       mov x17, x30
       bl check_errata_1925769
       cbz x0, if

       / * Set bit 8 in ECTRL_EL1 */
      mrs x1, CORTEX_A77_CPUECTRL_EL1
     orr x1, x1, #CORTEX_A77_CPUECTRL_EL1_BIT_8
      msr CORTEX_A77_CPUECTRL_EL1, x1
      isb
    1:
      ret x17
   endfunc errata_a77_1925769_wa

   func check_errata_1925769
       /* Applies to everything <= r1p1 */
       mov x1, #0x11
       b cpu_rev_var_ls
   endfunc check_errata_1925769

b) func cortex_a77_reset_func
    mov x19, x30
    bl cpu_get_rev_var
    mov x18, x0

    // ...
    if ERRATA_A77_1925769
    mov x8, x18
    bl errata_a77_1925769_wa
    endif
    // ...
    isb
    ret x19
endfunc cortex_a77_reset_func

c) func cortex_a77_errata_report
    stp x8, x30, [sp, #-16]
    bl cpu_get_rev_var
    mov x0, x0

    // ...
        report_errata ERRATA_A77_1925769, cortex_a77, 1925769
    // ...
    ldp x8, x30, [sp], #16
    ret
endfunc cortex_a77_errata_report

+ make rule
+ docs mention
2. The errata ABI

```c
#if CORTEX_A77_H_INC
{
    .cpu_dn = CORTEX_A77_MIDR,
    .cpu_errata_list = {
        {1508412, 0x00, 0x10},
        {1791578, 0x00, 0x11},
        {1925769, 0x00, 0x11},
        {1946167, 0x00, 0x11},
        {2356587, 0x00, 0x11},
        {UINT_MAX}, {UINT_MAX},
        {UINT_MAX}, {UINT_MAX},
        {UINT_MAX}, {UINT_MAX},
        {UINT_MAX}, {UINT_MAX},
    }
},
#endif
```

- 1 more place to edit
- Information again redundant
- But not accessible
All the useful code

The rest is boilerplate

And very annoying to get past review
Of course, some are more involved

- Longer workaround sequence
- More involved rev check
- Not applied at reset
Practically all errata can be pigeonholed to this template

With small provisions to account for variations
Proposal – Aarch64 erratum implementation

- `workaround_reset_start cortex_a77, ERRATUM(1925769), ERRATA_A77_1925769`
- `sysreg_bit_set CORTEX_A77_CPUECTRL_EL1, CORTEX_A77_CPUECTRL_EL1_BIT_R`
- `workaround_reset_end cortex_a77, ERRATUM(1925769)`
- `check_erratum_ls cortex_a77, ERRATUM(1925769), CPU_REV(1, 1)`

+ `workaround_reset_{start, end}` - wrapper of erratum workaround function
+ `workaround_runtime_{start, end}` - same but workaround manually applied
+ `sysreg_bit_set` - reads back and asserts bit set when DEBUG=1
+ `check_erratum_{ls, hs, range}` - checker helper
+ A runtime

+ make rule
+ docs mention
The runtime

workaround_*_start registers an erratum entry
  • In per-cpu errata_entries section (like cpu_ops)

cpu_reset_func applies selected ones from the list
  • Special cpu behaviour can happen after

erata_report_shim does reporting when DEBUG=1
  • Common C function for all CPUs iterates the list
This covers the majority of cases

Each macro can be incrementally unraveled to the old method for particularly nasty errata
The Procedure Call Standard

- Some of the cpu operations must obey the PCS

- => obey the PCS throughout

- Based on the following (simplified) interpretation

<table>
<thead>
<tr>
<th>Reg</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0 - r15</td>
<td>Scratch registers. Anyone can use at any time</td>
</tr>
<tr>
<td>r16, r17</td>
<td>Avoid using. Used by the linker. Any branch (with a relocation) may corrupt it</td>
</tr>
<tr>
<td>r18</td>
<td>Avoid using. Scratch, but may be used by the platform for inter procedure call state. Is this us?</td>
</tr>
<tr>
<td>r19 - r28</td>
<td>Caller saved</td>
</tr>
<tr>
<td>r29, r30</td>
<td>FP, LR</td>
</tr>
</tbody>
</table>
Mandated register assignments

+ to avoid having to do register management
  • Also will simplify implementation

+ Subset of the full PCS
  • To eliminate the problem

<table>
<thead>
<tr>
<th>function</th>
<th>register</th>
<th>treatment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any BL</td>
<td>r0-r4</td>
<td>May clobber</td>
</tr>
<tr>
<td>Workaround implementation</td>
<td>r0-r7</td>
<td>May clobber</td>
</tr>
<tr>
<td></td>
<td>r0, r5</td>
<td>Parameter to implementation - cpu_rev_var</td>
</tr>
<tr>
<td>Erratum checker function</td>
<td>r0-r4</td>
<td>May clobber</td>
</tr>
<tr>
<td>All other</td>
<td>r8 - r30</td>
<td>Treat as callee saved</td>
</tr>
</tbody>
</table>

+ Runtime has similar assignments, documented in code
Aarch32

- Implementation stays the same
- Only registered to the framework for debug and ABI reporting
- Removes some redundancy but little benefit to do fully

```c
add_erratum_entry cortex_a57, ERRATUM(813420), ERRATA_A57_813420
```
The implementation

Runtime part
Cost – workaround/check functions

- Check function identical

- Workaround function – practically identical
  - isb moved to reset_func
  - extra mov for compatibility
  - ASSERT when DEBUG=1 (gone on release builds)
Cost – errata_entries list

Per-cpu list

- 24 bytes per entry, 1 entry per erratum
  - some overlap with errata ABI. Designed to be reused
  - Minimal information to enable runtime and ABI reporting
Cost – reset_func

- Fixed size of **19 instructions** (76 bytes)
  - Previously 5 fixed + 2 per erratum

- Loop with **8 instructions** per erratum
  - Runs even if disabled (previously compiled out)
  - Previously only 2

- Space saving when > 7 errata per cpu

- disabled errata are not left out of the list due to the errata ABI
Cost – errata reporting

- **A debug feature.** Compiled out on release builds
  - Optimality superseded by ease of use

- **Common print function in C**
  - Around 250 instructions

- **Per-cpu shim**
  - 5 instructions
Going forward
The migration

+ Old and new style interoperable
  - old is inaccessible to framework

+ Migrate every aarch64 cpu - 3 patches

+ Fill-in every aarch32 cpu - 1 patch

+ Converge with errata ABI

+ Gradually submit to LTS
Aarch64 correctness

+ Patch 1 – reorder only
  • To enforce reporting and binary search requirements

+ Patch 2 – remove boilerplate and register to framework
  • Retain git blame of actual workaround

+ Patch 3 – move to bit setting helpers
  • Readability and consistency benefit
  • Strictly speaking optional

+ Script to verify identical binary result
  • Within established tolerances eg. missing isb

+ Manual debugger run
  • a few will need genuine refactors
  • i.e. the usual errata testing process
Correctness contd.

- **Open question** - build workarounds for CI runs
  - no elegant solution was apparent
  - Only done for Juno (hard-coded)

- **Add asserts** – location **open question**
  - Currently sysreg_bit_set reads the bit back
  - Assert workaround ran? How?
  - Assert get_cpu_var ran? How?

- Aarch32 avoids correctness since no refactoring there
Downstream errata

+ Please submit upstream
  • We will do migration work for each CPU

+ Changes easy to implement
  • But will assist

+ Platform errata unaffected, but inaccessible to framework and ABI
LTS

- Patches can be submitted to LTS

- LTS identical to master, no changes required
  - For errata, at least
Errata ABI convergence

Merge framework

Convert cpus

Merge ABI

cleanup
Code

https://review.trustedfirmware.org/q/topic:%22bk%252Ferrata_refactor%22+
Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה