Working with HardIRQs:
Life Beyond Static IRQ Assignments
2011-04-12

Renesas Electronics
Paul Mundt
paul.mundt@renesas.com
lethal@linux-sh.org

Outline

- The Olden Days
- The Problem
- What they all have in common
- Overview of sparseirq
- Overview of genirq changes
- Dynamic IRQs
- Transparent demux - A case study
- Future work
- Questions?

The Olden Days, abridged for sanity

- NR_IRQS
  - Statically sized.
  - Assumed linear mapping.
  - Assumed fairly constrained number of interrupt sources.
- Flat irq_desc descriptor array bounded by NR_IRQS.
  - Many other platform-specific structures end up being equally sized.
- Cascade and demux ranges using arbitrary IRQ assignments.
  - Drivers must depend on driver model for unallocated range.
  - IRQ range generally relative to a bit position.
    - In practice this implies a bias for 16-32 IRQs per range.

The Olden Days, abridged for sanity

- Architecture workarounds
  - Translation layer for exception vector to Linux IRQ mapping.
    - Further complicated by per-CPU vector mapping, migration, etc.
  - Whimsical upper bounding of NR_IRQS via Kconfig/machine descriptor/etc.
    - (alpha, arm, blackfin, <insert random embedded arch here>..)
  - Invention of entirely new subystems layered on top of genirq.
    - Architecture people like to be different -- Not Invented Here Syndrome.
    - Usually abstracted to the point where no arch-specific data exists.
    - Propagation of non-portable data structures and APIs.

- No centralized overview of IRQ allocation and mapping.
  - Code duplication all over the tree.
    - Some implementations less broken than others.
  - Unusable in shared drivers.
    - Again, falling back on driver model shim.

The Problem

- Linear IRQ mapping is a terrible fit for vectored CPUs.
  - The world is not an i8259 or flat IO-APIC (fortunately!)
- Device support, old and new
  - MFDs and SuperIOs
  - GPIO Expanders
  - PCI-X / PCI Express
    - MSI/MSI-X/multi-MSI
    - A new world of pain - up to 2048 potential IRQ mappings per controller!
- People keep using SMP for some reason.
  - Multiple levels of big IRQ locks.
  - CPU hotplug
  - IRQ migration (CPU/NUMA node/etc.)

What they all have in common

- Architectures
  - The need to track and manage a centralized IRQ bitmap.
  - The need to support an unbounded NR_IRQS.
    - Something that can scale not only with architecture configuration, but drivers too.
    - Growing and shrinking.
  - The ability to define IRQ state.
    - IRQ reservations.
  - Allocation/freeing/binding/unbinding of IRQs dynamically.
  - Provisioning of per-IRQ data.
  - Scalable lookups.
- Drivers
  - The ability to acquire dynamic IRQs in a portable way.
  - Acccess to per-IRQ data and state.
    - Ideally without requiring irq_desc awareness.

Overview of sparseirq

- Introduces an irq_desc as an array of pointers model.
  - NR_IRQS becomes run-time selectable.
    - Conservative platforms simply wrap their NR_IRQ probe to their machine descriptors.
  - Ability to dynamically expand.
    - To an extent.
  - Node awareness for backing irq_desc at instantiation time.
    - Originally GFP_ATOMIC/bootmem backed, now GFP_KERNEL (implied GFP_NOWAIT for early boot).

- irq_descs tracked in centrally-managed radix tree.
- NUMA friendly.
- Lightweight-ish.

Overview of sparseirq

- Originally very x86-centric, but completely rewritten.
  - Now unexpectedly sensible.
    - Underwent tglx post-processing.
  - Beginning to be used by embedded platforms.
    - Originally on SuperH, ARM SH/R-Mobile, now also PowerPC and other ARMs.
    - For some of these, it is the only supported IRQ model.

- Originally intended for systems with large NR_CPUS
  - Equally suitable for vectored CPUs with sparse IRQ instantiation patterns.

Overview of genirq changes

- Generalization of sparseirq/arch features
  - Big NR_IRQS IRQ bitmap with accessor APIs
    - alloc/free/reserve
    - Private bitmaps largely killed off from all non-ia64 architectures.

- The complete decoupling of irq_desc/irq_data.
  - Systematic overhaul of all in-tree code for irq_data utilization.
  - irq_desc size reduction
    - possible to bloat NR_IRQS to ridiculous proportions.
  - irq_data and status accessor driver APIs.

- IRQ threading

Dynamic IRQs

- Originally an awkward IO-APIC "inspired" API.
  - create_irq()/create_irq_nr()/destroy_irq()
  - API inconsistency with regards to signedness.
  - Now deprecated.
- Now generically facilitated through the genirq bitmap.
  - irq_alloc_desc()/irq_alloc_descs()
  - irq_alloc_desc_at()/irq_alloc_desc_from()
  - irq_free_desc()/irq_free_descs()
- Reservations of bitmap positions also possible
  - irq_reserve_irq()/irq_reserve_irqs()
- Transparent expansion of nr_irqs

Dynamic IRQs

- Architectural flow
  - nr_irqs initially sized.
  - CPU registers vector to IRQ mapping for initial bitmap population.
    - Representing the list of "possible" hardware IRQs in the global bitmap.
  - Any additional CPU reservations.
  - Insertion of demux ranges for various irq_chips by SoC code.
    - Can be at a fixed location to facilitate compatibility with arbitrary assignments.
- Subsequent IRQ allocations scan for bitmap holes.
  - Allows for NR_IRQS compaction
    - Only need to encapsulate the highest possible vector.
  - Bitmap density increases, allowing for more efficient radix tree utilization.
    - And space savings!

Dynamic IRQs

- Drivers can allocate/free dynamically with a portable API
  - Any reverse mapping from the IRQ cookie is pushed down to the architecture code.
    - Archictectures are still free to implement per-CPU vector maps as they see fit.
  - irq_chip registration requires no awareness of the backing IRQ
    - No need for platform data designation, as one will be dynamically assigned.
    - -ENOMEM is a possibility here.
- Potential for creative and perverse utilization patterns!

Transparent demux - A case study

- An extreme dynamic IRQ utilization example.

- Traditional demux flow
  - Chained demux handler checks a cause register
    - Iterative looping and kicking of handlers for triggered bits.
  - Obviously not very fun
  - IRQF_SHARED is an abomination
    - often forcing completely logically disconnected drivers to share handler glue due to register layout.

Transparent demux - A case study

- Transparent demux - giving each bit its own IRQ, because we can.
  - Platform submits bit positions per cause register to split out.
    - Tagged and inserted in to the controller’s radix tree for lazy IRQ allocation.
    - Subsequent grouped IRQ mapping by way of tagged radix tree gang lookups.
  - Once the bitmap is populated, resolve pending allocations.
    - Each dynamic IRQ is added to a linked list under the hardware IRQs private data.
  - Interrupt core inserts its own chained handler for the hwirq.
    - Cause register data encoded under hardware IRQ handler data.
    - Original handler data for hardware IRQ inherited by the child.
    - Normal generic_handle_irq() dispatch for asserted bits.
  - Radix tree tag is dropped and the slot replaced with an IRQ<->handle translation for subsequent lookup.
    - Once resolved, drivers can fetch the dynamically resolved IRQ number and use it as normal.
  - No need for IRQF_SHARED.

Future work

- Killing off NR_IRQS completely?
  - Some platforms have special HARDIRQ_BITS constraints
    - Effectively neutering the upper bounds.
  - Or an asm-generic/ version with a ridiculously high number.
    - No need for future ports to concern themselves with these things.
- Generalization of per-controller radix trees?
  - Different use cases between the sh and ppc IRQ host radix trees.
- Device tree bindings?

Questions?