PTP hardware clock infrastructure for Linux

This patch set introduces support for IEEE 1588 PTP clocks in Linux. Together with the SO_TIMESTAMPING socket options, this presents a standardized method for developing PTP user space programs, synchronizing Linux with external clocks, and using the ancillary features of PTP hardware clocks.

A new class driver exports a kernel interface for specific clock drivers and a user space interface. The infrastructure supports a complete set of PTP hardware clock functionality.

- Basic clock operations - Set time - Get time - Shift the clock by a given offset atomically - Adjust clock frequency
- Ancillary clock features - Time stamp external events - Period output signals configurable from user space - Low Pass Filter (LPF) access from user space - Synchronization of the Linux system time via the PPS subsystem

PTP hardware clock kernel API

A PTP clock driver registers itself with the class driver. The class driver handles all of the dealings with user space. The author of a clock driver need only implement the details of programming the clock hardware. The clock driver notifies the class driver of asynchronous events (alarms and external time stamps) via a simple message passing interface.

The class driver supports multiple PTP clock drivers. In normal use cases, only one PTP clock is needed. However, for testing and development, it can be useful to have more than one clock in a single system, in order to allow performance comparisons.

PTP hardware clock user space API

The class driver also creates a character device for each registered clock. User space can use an open file descriptor from the character device as a POSIX clock id and may call clock_gettime, clock_settime, and clock_adjtime. These calls implement the basic clock operations.

User space programs may control the clock using standardized ioctls. A program may query, enable, configure, and disable the ancillary clock features. User space can receive time stamped events via blocking read() and poll().

Writing clock drivers

Clock drivers include include/linux/ptp_clock_kernel.h and register themselves by presenting a 'struct ptp_clock_info' to the registration method. Clock drivers must implement all of the functions in the interface. If a clock does not offer a particular ancillary feature, then the driver should just return -EOPNOTSUPP from those functions.

Drivers must ensure that all of the methods in interface are reentrant. Since most hardware implementations treat the time value as a 64 bit integer accessed as two 32 bit registers, drivers should use spin_lock_irqsave/spin_unlock_irqrestore to protect against concurrent access. This locking cannot be accomplished in class driver, since the lock may also be needed by the clock driver's interrupt service routine.

Supported hardware

- Freescale eTSEC gianfar
  - 2 Time stamp external triggers, programmable polarity (opt. interrupt)
  - 2 Alarm registers (optional interrupt)
  - 3 Periodic signals (optional interrupt)
- National DP83640
  - 6 GPIOs programmable as inputs or outputs
  - 6 GPIOs with dedicated functions (LED/JTAG/clock) can also be used as general inputs or outputs
  - GPIO inputs can time stamp external triggers
  - GPIO outputs can produce periodic signals
  - 1 interrupt pin
- Intel IXP465
  - Auxiliary Slave/Master Mode Snapshot (optional interrupt)
  - Target Time (optional interrupt)
- Renesas (IDT) ClockMatrixâ„¢
  - Up to 4 independent PHC channels
  - Integrated low pass filter (LPF), access via .adjPhase (compliant to ITU-T G.8273.2)
  - Programmable output periodic signals
  - Programmable inputs can time stamp external triggers
  - Driver and/or hardware configuration through firmware (idtcm.bin)

- LPF settings (bandwidth, phase limiting, automatic holdover, physical layer assist (per ITU-T G.8273.2))
- Programmable output PTP clocks, any frequency up to 1GHz (to other PHY/MAC time stampers, refclk to ASSPs/SoCs/FPGAs)
- Lock to GNSS input, automatic switching between GNSS and user-space PHC control (optional)