Renesas Electronics Corporation

Linux PTP Using PHC Adjust Phase Reference Manual

December 16, 2020

CONTENTS:

CHAPTER

ONE

REVISION HISTORY

The following table shows the revision history for the “Linux PTP Using PHC Adjust Phase Reference Manual”.

Table 1: **Revision History**

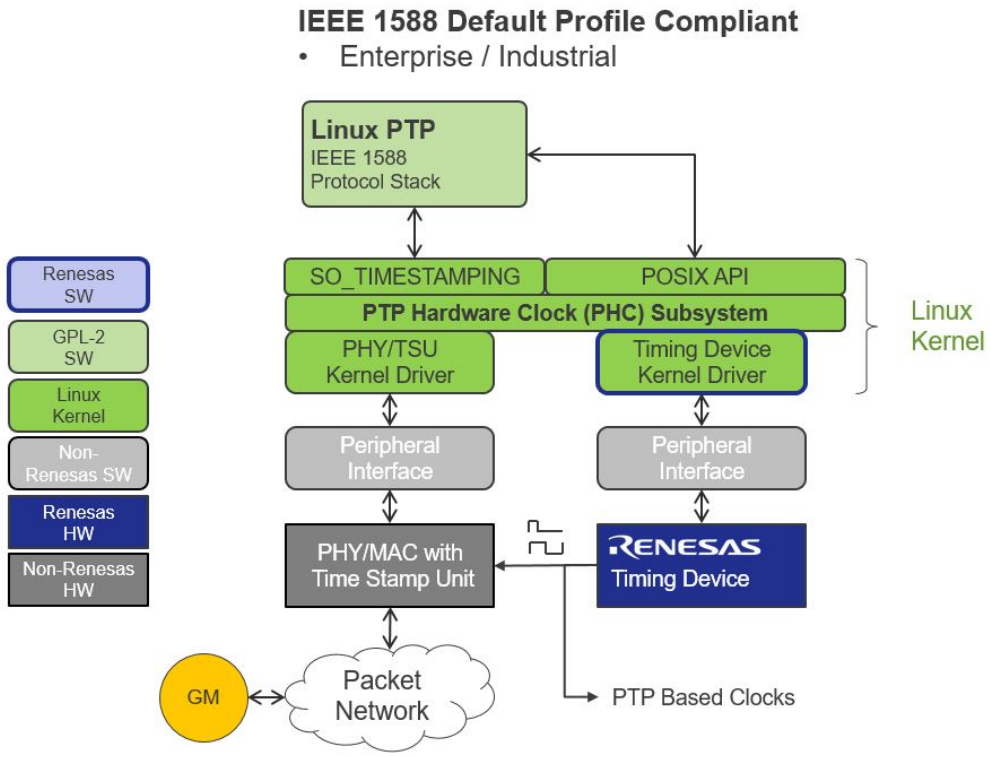
Revision	Date	Description
0.2	Dec 16, 2020	Supports software release 4.0.1.69477 Document changes include: • Add Revision History section • Add section numbers to bookmarks on PDF • Remove IDT from “The IDT ClockMatrix (TM) family …” • Update with ts2phc.cfg change • Revise cover page, add logo, page numbering
0.1	Aug 07, 2020	Supports software release 4.0.0.62805 • First release of this document

CHAPTER

TWO

INTRODUCTION

The Linux PTP project is an implementation of the Precision Time Protocol (PTP) according to IEEE standard 1588 for Linux licensed under the GNU General Public License.



Linux PTP supports both hardware and software timestamping via the Linux SO_TIMESTAMPING socket option. Hardware timestamping is used because it provides better accuracy by timestamping packets at the exact moment they are sent and received. Hardware time stamping requires hardware PTP support from the PHY/MAC.

Some PTP hardware clocks have a adjust phase mode that has a built-in hardware filtering capability. The adjust phase mode utilizes a phase offset control word instead of a frequency offset control word.

Adjust phase support was introduced into the Linux PHC subsystem in Linux kernel v5.8.

Linux PTP v3.0 added `ts2phc` that is used to align the PHC and timestamper when the PHC and timestamper are not the same device.

CHAPTER

THREE

SYSTEM REQUIREMENTS

- Linux PTP v3.0
- Linux kernel v5.8+; can be patched for v3.0+
- Network interface
  - Implements SIOCETHTOOL
  - Supports PTP hardware clock (PHC)
  - Pass PTP Ethernet packet time stamps using the SO_TIMESTAMPING API
- Linux PTP Hardware Clock driver
  - Renesas timing device that supports adjust phase
    - For ClockMatrix, already part of Linux kernel v5.8+; can be patched for v3.0+
  - For other timing devices, patch available for v3.0+

3.1 Adjust Phase Mode

Some PTP hardware clocks have a adjust phase mode that has a built-in hardware filtering capability. The adjust phase mode utilizes a phase offset control word instead of a frequency offset control word. Linux kernel v5.8's PTP Hardware Clock interface includes the capability to accept phase offset control word. Linux PTP v3.0 has also added support for adjust phase (ie. write_phase_mode configuration parameter).

3.2 Linux PTP Source

Instructions on cloning and compiling Linux PTP from source can be found at http://linuxptp.sourceforge.net/.

Linux PTP added support for adjust phase in `2020-05-04 commit 7df88a`.

Host site:

- https://sourceforge.net/p/linuxptp/code/ci/master/tree/
- https://sourceforge.net/p/linuxptp/code/ci/v3.0/tree/

Mirror site:

- https://github.com/richardcochran/linuxptp

3.3 Linux Kernel

Host site: https://github.com/torvalds/linux

Write phase mode was introduced into the Linux PHC subsytem in Linux kernel v5.8.

If using Linux kernel v3.x to v5.7, then patching the linux kernel is required to support adjust phase.

3.3.1 Adjust Phase Patch Files

The patch files below are the differences between Linux kernel v5.8-rc5 and v5.6.

Files affected

- drivers/ptp/ptp_chardev.c
- drivers/ptp/ptp_clock.c
- include/linux/ptp_clock_kernel.h
- include/uapi/linux/ptp_clock.h
- tools/testing/selftests/ptp/testptp.c

3.3.1.1 Patch file for Linux v5.6

Copy and paste below and save as a text file on the root directory of your linux kernel v5.6.

```
From 2ae6eb4fa4a7d7499322e68fc17fae14f35bfc1b Mon Sep 17 00:00:00 2001
From: Vincent Cheng <vincent.cheng.xh@renesas.com>
Date: Fri, 17 Jul 2020 16:50:20 -0400
Subject: [PATCH] adjust phase patch for linux v5.6

---
 drivers/ptp/ptp_chardev.c             |  1 +
 drivers/ptp/ptp_clock.c               |  9 +++++++++
 include/linux/ptp_clock_kernel.h      | 14 ++++++++++++++
 include/uapi/linux/ptp_clock.h        |  4 +++-
 tools/testing/selftests/ptp/testptp.c |  6 ++++++
 5 files changed, 26 insertions(+), 8 deletions(-)

diff --git a/drivers/ptp/ptp_chardev.c b/drivers/ptp/ptp_chardev.c
index 9d72ab5..4c9fc5c 100644
--- a/drivers/ptp/ptp_chardev.c
+++ b/drivers/ptp/ptp_chardev.c
@@ -133,12 +133,13 @@ long ptp_ioctl(struct posix_clock *pc, unsigned int cmd,␣
↪unsigned long arg)
                caps.n_alarm = ptp->info->n_alarm;
                caps.n_ext_ts = ptp->info->n_ext_ts;
                caps.n_per_out = ptp->info->n_per_out;
                caps.pps = ptp->info->pps;
                caps.n_pins = ptp->info->n_pins;
                caps.cross_timestamping = ptp->info->getcrosststamp != NULL;
+               caps.adjust_phase = ptp->info->adjphase != NULL;
                if (copy_to_user((void __user *)arg, &caps, sizeof(caps)))
                        err = -EFAULT;
                break;
```

(continues on next page)

(continued from previous page)

```
         case PTP_EXTTS_REQUEST:
         case PTP_EXTTS_REQUEST2:
diff --git a/drivers/ptp/ptp_clock.c b/drivers/ptp/ptp_clock.c
index ac1f2bf..4fc6e4b 100644
--- a/drivers/ptp/ptp_clock.c
+++ b/drivers/ptp/ptp_clock.c
@@ -143,12 +143,21 @@ static int ptp_clock_adjtime(struct posix_clock *pc, struct __
↪kernel_timex *tx)
                         return -ERANGE;
                 if (ops->adjfine)
                         err = ops->adjfine(ops, tx->freq);
                 else
                         err = ops->adjfreq(ops, ppb);
                 ptp->dialed_frequency = tx->freq;
+        } else if (tx->modes & ADJ_OFFSET) {
+                if (ops->adjphase) {
+                        s32 offset = tx->offset;
+
+                        if (!(tx->modes & ADJ_NANO))
+                                offset *= NSEC_PER_USEC;
+
+                        err = ops->adjphase(ops, offset);
+                }
         } else if (tx->modes == 0) {
                 tx->freq = ptp->dialed_frequency;
                 err = 0;
         }

         return err;
diff --git a/include/linux/ptp_clock_kernel.h b/include/linux/ptp_clock_kernel.h
index c64a1ef..aa805f6 100644
--- a/include/linux/ptp_clock_kernel.h
+++ b/include/linux/ptp_clock_kernel.h
@@ -33,13 +33,13 @@ struct system_device_crosststamp;
 struct ptp_system_timestamp {
         struct timespec64 pre_ts;
         struct timespec64 post_ts;
 };

 /**
- * struct ptp_clock_info - decribes a PTP hardware clock
+ * struct ptp_clock_info - describes a PTP hardware clock
   *
   * @owner:     The clock driver should set to THIS_MODULE.
   * @name:      A short "friendly name" to identify the clock and to
   *             help distinguish PHY based devices from MAC based ones.
   *             The string is not meant to be a unique id.
   * @max_adj:   The maximum possible frequency adjustment, in parts per billon.
@@ -62,12 +62,15 @@ struct ptp_system_timestamp {
   * @adjfreq:  Adjusts the frequency of the hardware clock.
   *            This method is deprecated.  New drivers should implement
   *            the @adjfine method instead.
   *            parameter delta: Desired frequency offset from nominal frequency
   *            in parts per billion
   *
+ * @adjphase:  Adjusts the phase offset of the hardware clock.
+ *             parameter delta: Desired change in nanoseconds.
```

(continues on next page)

(continued from previous page)

```
+ *
  * @adjtime:  Shifts the time of the hardware clock.
  *            parameter delta: Desired change in nanoseconds.
  *
  * @gettime64:  Reads the current time from the hardware clock.
  *              This method is deprecated.  New drivers should implement
  *              the @gettimex64 method instead.
@@ -102,16 +105,16 @@ struct ptp_system_timestamp {
  *            zero if the function can be assigned to this pin, and
  *            nonzero otherwise.
  *            parameter pin: index of the pin in question.
  *            parameter func: the desired function to use.
  *            parameter chan: the function channel index to use.
  *
- * @do_work:  Request driver to perform auxiliary (periodic) operations
- *              Driver should return delay of the next auxiliary work scheduling
- *              time (>=0) or negative value in case further scheduling
- *              is not required.
+ * @do_aux_work:  Request driver to perform auxiliary (periodic) operations
+ *                Driver should return delay of the next auxiliary work
+ *                scheduling time (>=0) or negative value in case further
+ *                scheduling is not required.
  *
  * Drivers should embed their ptp_clock_info within a private
  * structure, obtaining a reference to it using container_of().
  *
  * The callbacks must all return zero on success, non-zero otherwise.
  */
@@ -125,12 +128,13 @@ struct ptp_clock_info {
         int n_per_out;
         int n_pins;
         int pps;
         struct ptp_pin_desc *pin_config;
         int (*adjfine)(struct ptp_clock_info *ptp, long scaled_ppm);
         int (*adjfreq)(struct ptp_clock_info *ptp, s32 delta);
+        int (*adjphase)(struct ptp_clock_info *ptp, s32 phase);
         int (*adjtime)(struct ptp_clock_info *ptp, s64 delta);
         int (*gettime64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
         int (*gettimex64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                           struct ptp_system_timestamp *sts);
         int (*getcrosststamp)(struct ptp_clock_info *ptp,
                               struct system_device_crosststamp *cts);
diff --git a/include/uapi/linux/ptp_clock.h b/include/uapi/linux/ptp_clock.h
index 9dc9d00..ff070aa 100644
--- a/include/uapi/linux/ptp_clock.h
+++ b/include/uapi/linux/ptp_clock.h
@@ -86,13 +86,15 @@ struct ptp_clock_caps {
         int n_ext_ts;  /* Number of external time stamp channels. */
         int n_per_out; /* Number of programmable periodic signals. */
         int pps;       /* Whether the clock supports a PPS callback. */
         int n_pins;    /* Number of input/output pins. */
         /* Whether the clock supports precise system-device cross timestamps */
         int cross_timestamping;
-        int rsv[13];   /* Reserved for future use. */
+        /* Whether the clock supports adjust phase */
+        int adjust_phase;
+        int rsv[12];   /* Reserved for future use. */
```

(continues on next page)

(continued from previous page)

```
 };

 struct ptp_extts_request {
         unsigned int index;  /* Which channel to configure. */
         unsigned int flags;  /* Bit field for PTP_xxx flags. */
         unsigned int rsv[2]; /* Reserved for future use. */
diff --git a/tools/testing/selftests/ptp/testptp.c b/tools/testing/selftests/ptp/
↪testptp.c
index c0dd102..da7a9dd 100644
--- a/tools/testing/selftests/ptp/testptp.c
+++ b/tools/testing/selftests/ptp/testptp.c
@@ -266,20 +266,22 @@ int main(int argc, char *argv[])
                                 "  %d maximum frequency adjustment (ppb)\n"
                                 "  %d programmable alarms\n"
                                 "  %d external time stamp channels\n"
                                 "  %d programmable periodic signals\n"
                                 "  %d pulse per second\n"
                                 "  %d programmable pins\n"
-                                "  %d cross timestamping\n",
+                                "  %d cross timestamping\n"
+                                "  %d adjust_phase\n",
                                 caps.max_adj,
                                 caps.n_alarm,
                                 caps.n_ext_ts,
                                 caps.n_per_out,
                                 caps.pps,
                                 caps.n_pins,
-                                caps.cross_timestamping);
+                                caps.cross_timestamping,
+                                caps.adjust_phase);
                 }
         }

         if (0x7fffffff != adjfreq) {
                 memset(&tx, 0, sizeof(tx));
                 tx.modes = ADJ_FREQUENCY;
--
2.7.4
```

To apply the patch file,

`git am --signoff < a_file.patch`

Where `a_file.patch` is the name of your patch file.

```
$ git am --signoff < 0001-adjust-phase-patch-for-linux-v5.6.patch
Applying: Adjust phase patch for linux v5.6

$ git log
commit d6e62c17b920a4425395fdacc7c174be955e7568
Author: Vincent Cheng <vincent.cheng.xh@renesas.com>
Date:   Fri Jul 17 16:50:20 2020 -0400

    adjust phase patch for linux v5.6

    Signed-off-by: Vincent Cheng <vincent.cheng.xh@renesas.com>
```

(continues on next page)

(continued from previous page)

```
commit 7111951b8d4973bda27ff663f2cf18b663d15b48
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Sun Mar 29 15:25:41 2020 -0700

    Linux 5.6

...
```

or

`patch -p1 < a_file.patch` and manually commit the changes afterwards.

```
$ patch -p1 < 0001-adjust-phase-patch-for-linux-v5.6.patch
patching file drivers/ptp/ptp_chardev.c
patching file drivers/ptp/ptp_clock.c
patching file include/linux/ptp_clock_kernel.h
patching file include/uapi/linux/ptp_clock.h
patching file tools/testing/selftests/ptp/testptp.c

$ git status
On branch tmp
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   drivers/ptp/ptp_chardev.c
    modified:   drivers/ptp/ptp_clock.c
    modified:   include/linux/ptp_clock_kernel.h
    modified:   include/uapi/linux/ptp_clock.h
    modified:   tools/testing/selftests/ptp/testptp.c

no changes added to commit (use "git add" and/or "git commit -a")
```

3.3.2 Backporting to Linux v3.x+

For backporting adjust phase changes into other linux kernel versions, examine the individual patch files below and manually make the corresponding appropriate changes for your linux kernel version.

3.3.2.1 drivers/ptp/ptp_chardev.c

```
--- a/drivers/ptp/ptp_chardev.c
+++ b/drivers/ptp/ptp_chardev.c
@@ -133,12 +133,13 @@ long ptp_ioctl(struct posix_clock *pc, unsigned int cmd,␣
↪unsigned long arg)
                caps.n_alarm = ptp->info->n_alarm;
                caps.n_ext_ts = ptp->info->n_ext_ts;
                caps.n_per_out = ptp->info->n_per_out;
                caps.pps = ptp->info->pps;
                caps.n_pins = ptp->info->n_pins;
                caps.cross_timestamping = ptp->info->getcrosststamp != NULL;
+               caps.adjust_phase = ptp->info->adjphase != NULL;
                if (copy_to_user((void __user *)arg, &caps, sizeof(caps)))
                        err = -EFAULT;
```

(continues on next page)

(continued from previous page)

```
                break;

        case PTP_EXTTS_REQUEST:
        case PTP_EXTTS_REQUEST2:
```

3.3.2.2 drivers/ptp/ptp_clock.c

```
--- a/drivers/ptp/ptp_clock.c
+++ b/drivers/ptp/ptp_clock.c
@@ -143,12 +143,21 @@ static int ptp_clock_adjtime(struct posix_clock *pc, struct __
↪kernel_timex *tx)
                        return -ERANGE;
                if (ops->adjfine)
                        err = ops->adjfine(ops, tx->freq);
                else
                        err = ops->adjfreq(ops, ppb);
                ptp->dialed_frequency = tx->freq;
+       } else if (tx->modes & ADJ_OFFSET) {
+               if (ops->adjphase) {
+                       s32 offset = tx->offset;
+
+                       if (!(tx->modes & ADJ_NANO))
+                               offset *= NSEC_PER_USEC;
+
+                       err = ops->adjphase(ops, offset);
+               }
        } else if (tx->modes == 0) {
                tx->freq = ptp->dialed_frequency;
                err = 0;
        }

        return err;
```

3.3.2.3 include/linux/ptp_clock_kernel.h

Change for comments have been left out.

```
--- a/include/linux/ptp_clock_kernel.h
+++ b/include/linux/ptp_clock_kernel.h
@@ -125,12 +128,13 @@ struct ptp_clock_info {
        int n_per_out;
        int n_pins;
        int pps;
        struct ptp_pin_desc *pin_config;
        int (*adjfine)(struct ptp_clock_info *ptp, long scaled_ppm);
        int (*adjfreq)(struct ptp_clock_info *ptp, s32 delta);
+       int (*adjphase)(struct ptp_clock_info *ptp, s32 phase);
        int (*adjtime)(struct ptp_clock_info *ptp, s64 delta);
        int (*gettime64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
        int (*gettimex64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                          struct ptp_system_timestamp *sts);
        int (*getcrosststamp)(struct ptp_clock_info *ptp,
                              struct system_device_crosststamp *cts);
```

3.3.2.4 include/uapi/linux/ptp_clock.h

```
--- a/include/uapi/linux/ptp_clock.h
+++ b/include/uapi/linux/ptp_clock.h
@@ -86,13 +86,15 @@ struct ptp_clock_caps {
         int n_ext_ts;  /* Number of external time stamp channels. */
         int n_per_out; /* Number of programmable periodic signals. */
         int pps;       /* Whether the clock supports a PPS callback. */
         int n_pins;    /* Number of input/output pins. */
         /* Whether the clock supports precise system-device cross timestamps */
         int cross_timestamping;
-        int rsv[13];   /* Reserved for future use. */
+        /* Whether the clock supports adjust phase */
+        int adjust_phase;
+        int rsv[12];   /* Reserved for future use. */
 };

 struct ptp_extts_request {
         unsigned int index;  /* Which channel to configure. */
         unsigned int flags;  /* Bit field for PTP_xxx flags. */
         unsigned int rsv[2]; /* Reserved for future use. */
```

3.3.2.5 tools/testing/selftests/ptp/testptp.c

```
--- a/tools/testing/selftests/ptp/testptp.c
+++ b/tools/testing/selftests/ptp/testptp.c
@@ -266,20 +266,22 @@ int main(int argc, char *argv[])
                                "  %d maximum frequency adjustment (ppb)\n"
                                "  %d programmable alarms\n"
                                "  %d external time stamp channels\n"
                                "  %d programmable periodic signals\n"
                                "  %d pulse per second\n"
                                "  %d programmable pins\n"
-                               "  %d cross timestamping\n",
+                               "  %d cross timestamping\n"
+                               "  %d adjust_phase\n",
                                caps.max_adj,
                                caps.n_alarm,
                                caps.n_ext_ts,
                                caps.n_per_out,
                                caps.pps,
                                caps.n_pins,
-                               caps.cross_timestamping);
+                               caps.cross_timestamping,
+                               caps.adjust_phase);
                 }
         }

         if (0x7fffffff != adjfreq) {
                 memset(&tx, 0, sizeof(tx));
                 tx.modes = ADJ_FREQUENCY;
```

3.4 Network Interface Requirements

The network interface driver would need to:

- Support ioctl command SIOCETHTOOL
- Support PTP hardware clock (PHC)
  - Pass PTP Ethernet packet time stamps using the SO_TIMESTAMPING API
  - Support POSIX clock API clock_adjtime() with `struct timex modes`
    - ADJ_FREQUENCY
    - ADJ_OFFSET
    - ADJ_NANO
    - ADJ_SETOFFSET
  - Support PTP request type
    - PTP_CLK_REQ_PPS
    - PTP_CLK_REQ_EXTTS (PTP_ENABLE_FEATURE)

`ethtool` can show whether a MAC supports hardware or software time stamping and whether it supports PHC.

The following example output indicates support for hardware time stamping listed under `Capabilities` and is registered as PHC device 0, `PTP Hardware Clock: 0`.

```
$ ethtool -T eth0

Time stamping parameters for eth0:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
```

3.4.1 ethtool

If the `ethtool` utility is not on your system, you can compile the utility from the source.

`https://git.kernel.org/pub/scm/network/ethtool/ethtool.git`

Build ethtool from source example:

```
# Build ethtool from source
$ git clone git://git.kernel.org/pub/scm/network/ethtool/ethtool.git

$ cd ethtool

$ ./autogen.sh
$ ./configure
```

(continues on next page)

(continued from previous page)

```
# CROSS compile for target board, ie. sudo apt-get install gcc-aarch64-linux-gnu
$ make clean all CC=/usr/bin/aarch64-linux-gnu-gcc

# Copy ethtool binary to target board
```

3.4.2 SIOCETHTOOL

Used in `linuxptp/sk.c` to interrogate network interface timestamping capabilities:

```
int sk_get_ts_info(const char *name, struct sk_ts_info *sk_info)
{
#ifdef ETHTOOL_GET_TS_INFO
    struct ethtool_ts_info info;
    ...

    info.cmd = ETHTOOL_GET_TS_INFO;

    ...

    err = ioctl(fd, SIOCETHTOOL, &ifr);
    if (err < 0) {
        pr_err("ioctl SIOCETHTOOL failed: %m");
        close(fd);
        goto failed;
    ...
```

ETHTOOL_GET_TS_INFO Linux kernel: include/uapi/linux/ethtool.h:

```
#define ETHTOOL_GET_TS_INFO 0x00000041 /* Get time stamping and PHC info */
```

3.4.3 Linux PTP Hardware Clock

Linux PTP works with devices with drivers that support the PTP hardware clock (PHC) infrastructure for Linux.

Details on Linux PHC subsystem can be found at https://www.kernel.org/doc/Documentation/ptp/ptp.txt.

```
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.
```

For examples of drivers that support PHC, search for `ptp_clock_register` in the linux kernel source.

Sample of devices that support PHC API (Linux Kernel v5.7):

- drivers/net/ethernet/broadcom/bnx2x/bnx2x_main.c
- drivers/net/ethernet/broadcom/tg3.c
- drivers/net/ethernet/cadence/macb_ptp.c
- drivers/net/ethernet/cavium/common/cavium_ptp.c
- drivers/net/ethernet/freescale/fec_ptp.c
- drivers/net/ethernet/intel/e1000e/ptp.c
- drivers/net/ethernet/renesas/ravb_ptp.c
- drivers/net/phy/dp83640.c

Renesas Timing Devices with PHC driver support:

- drivers/ptp/ptp_clockmatrix.c
- drivers/ptp/ptp_idt82p33.c

3.4.3.1 SO_TIMESTAMPING

https://www.kernel.org/doc/Documentation/networking/timestamping.txt

```
1. Control Interfaces

The interfaces for receiving network packages timestamps are:
...
* SO_TIMESTAMPING
  Generates timestamps on reception, transmission or both. Supports
  multiple timestamp sources, including hardware. Supports generating
  timestamps for stream sockets.

...

1.3 SO_TIMESTAMPING (also SO_TIMESTAMPING_OLD and SO_TIMESTAMPING_NEW):

Supports multiple types of timestamp requests. As a result, this
socket option takes a bitmap of flags, not a boolean. In

  err = setsockopt(fd, SOL_SOCKET, SO_TIMESTAMPING, &val, sizeof(val));

val is an integer with any of the following bits set. Setting other
bit returns EINVAL and does not change the current state.

The socket option configures timestamp generation for individual
sk_buffs (1.3.1), timestamp reporting to the socket's error
queue (1.3.2) and options (1.3.3). Timestamp generation can also
be enabled for individual sendmsg calls using cmsg (1.3.4).


1.3.1 Timestamp Generation

Some bits are requests to the stack to try to generate timestamps. Any
combination of them is valid. Changes to these bits apply to newly
created packets, not to packets already in the stack. As a result, it
is possible to selectively request timestamps for a subset of packets
(e.g., for sampling) by embedding an send() call within two setsockopt
```

(continues on next page)

(continued from previous page)

```
calls, one to enable timestamp generation and one to disable it.
Timestamps may also be generated for reasons other than being
requested by a particular socket, such as when receive timestamping is
enabled system wide, as explained earlier.

SOF_TIMESTAMPING_RX_HARDWARE:
  Request rx timestamps generated by the network adapter.

SOF_TIMESTAMPING_RX_SOFTWARE:
  Request rx timestamps when data enters the kernel. These timestamps
  are generated just after a device driver hands a packet to the
  kernel receive stack.

SOF_TIMESTAMPING_TX_HARDWARE:
  Request tx timestamps generated by the network adapter. This flag
  can be enabled via both socket options and control messages.

SOF_TIMESTAMPING_TX_SOFTWARE:
  Request tx timestamps when data leaves the kernel. These timestamps
  are generated in the device driver as close as possible, but always
  prior to, passing the packet to the network interface. Hence, they
  require driver support and may not be available for all devices.
  This flag can be enabled via both socket options and control messages.

...

1.3.4. Enabling timestamps via control messages

In addition to socket options, timestamp generation can be requested
per write via cmsg, only for SOF_TIMESTAMPING_TX_* (see Section 1.3.1).
Using this feature, applications can sample timestamps per sendmsg()
without paying the overhead of enabling and disabling timestamps via
setsockopt:

  struct msghdr *msg;
  ...
  cmsg                    = CMSG_FIRSTHDR(msg);
  cmsg->cmsg_level        = SOL_SOCKET;
  cmsg->cmsg_type         = SO_TIMESTAMPING;
  cmsg->cmsg_len          = CMSG_LEN(sizeof(__u32));
  *((__u32 *) CMSG_DATA(cmsg)) = SOF_TIMESTAMPING_TX_SCHED |
                SOF_TIMESTAMPING_TX_SOFTWARE |
                SOF_TIMESTAMPING_TX_ACK;
  err = sendmsg(fd, msg, 0);

The SOF_TIMESTAMPING_TX_* flags set via cmsg will override
the SOF_TIMESTAMPING_TX_* flags set via setsockopt.

Moreover, applications must still enable timestamp reporting via
setsockopt to receive timestamps:

  __u32 val = SOF_TIMESTAMPING_SOFTWARE |
          SOF_TIMESTAMPING_OPT_ID /* or any other flag */;
  err = setsockopt(fd, SOL_SOCKET, SO_TIMESTAMPING, &val, sizeof(val));

...
```

(continues on next page)

(continued from previous page)

```
3.1 Hardware Timestamping Implementation: Device Drivers

A driver which supports hardware time stamping must support the
SIOCSHWTSTAMP ioctl and update the supplied struct hwtstamp_config with
the actual values as described in the section on SIOCSHWTSTAMP.  It
should also support SIOCGHWTSTAMP.

Time stamps for received packets must be stored in the skb. To get a pointer
to the shared time stamp structure of the skb call skb_hwtstamps(). Then
set the time stamps in the structure:

struct skb_shared_hwtstamps {
    /* hardware time stamp transformed into duration
     * since arbitrary point in time
     */
    ktime_t hwtstamp;
};

Time stamps for outgoing packets are to be generated as follows:
- In hard_start_xmit(), check if (skb_shinfo(skb)->tx_flags & SKBTX_HW_TSTAMP)
  is set no-zero. If yes, then the driver is expected to do hardware time
  stamping.
- If this is possible for the skb and requested, then declare
  that the driver is doing the time stamping by setting the flag
  SKBTX_IN_PROGRESS in skb_shinfo(skb)->tx_flags , e.g. with

       skb_shinfo(skb)->tx_flags |= SKBTX_IN_PROGRESS;

  You might want to keep a pointer to the associated skb for the next step
  and not free the skb. A driver not supporting hardware time stamping doesn't
  do that. A driver must never touch sk_buff::tstamp! It is used to store
  software generated time stamps by the network subsystem.
- Driver should call skb_tx_timestamp() as close to passing sk_buff to hardware
  as possible. skb_tx_timestamp() provides a software time stamp if requested
  and hardware timestamping is not possible (SKBTX_IN_PROGRESS not set).
- As soon as the driver has sent the packet and/or obtained a
  hardware time stamp for it, it passes the time stamp back by
  calling skb_hwtstamp_tx() with the original skb, the raw
  hardware time stamp. skb_hwtstamp_tx() clones the original skb and
  adds the timestamps, therefore the original skb has to be freed now.
  If obtaining the hardware time stamp somehow fails, then the driver
  should not fall back to software time stamping. The rationale is that
  this would occur at a later time in the processing pipeline than other
  software time stamping and therefore could lead to unexpected deltas
  between time stamps.
```

3.4.4 PTP_CLK_REQ_EXTTS

Can check if device supports `PTP_CLK_REQ_EXTTS` by using the `testptp` utility.

For details on `testptp`, see *testptp*.

For example, the device `/dev/ptp0` supports `PTP_CLK_REQ_EXTTS` because it says `1 external time stamp channels`.

```
# testptp -d /dev/ptp0 -c
capabilities:
  999999999 maximum frequency adjustment (ppb)
  0 programmable alarms
  1 external time stamp channels
  0 programmable periodic signals
  0 pulse per second
  0 programmable pins
  0 cross timestamping
```

CHAPTER

FOUR

CLOCKMATRIX PHC DRIVER

The ClockMatrix (TM) family includes integrated devices that provide eight PLL channels. Each PLL channel can be independently configured as a frequency synthesizer, jitter attenuator, digitally controlled oscillator (DCO), or a digital phase lock loop (DPLL). Typically these devices are used as timing references and clock sources for PTP applications.

4.1 Source Code

The latest ClockMatrix PHC driver with adjust phase support is part of the Linux v5.10 kernel.

It can be found in the Linux kernel `drivers/ptp` directory.

https://github.com/torvalds/linux/tree/v5.10/drivers/ptp

Files of interest:

- Kconfig
- Makefile
- idt8a340_reg.h
- ptp_clockmatrix.h
- ptp_clockmatrix.c

Kconfig:

```
...
config PTP_1588_CLOCK_IDTCM
   tristate "IDT CLOCKMATRIX as PTP clock"
   depends on PTP_1588_CLOCK && I2C
   default n
   help
     This driver adds support for using IDT CLOCKMATRIX(TM) as a PTP
     clock. This clock is only useful if your time stamping MAC
     is connected to the IDT chip.

     To compile this driver as a module, choose M here: the module
     will be called ptp_clockmatrix.
 ...
```

Makefile:

```
...
obj-$(CONFIG_PTP_1588_CLOCK_IDTCM)  += ptp_clockmatrix.o
...
```

4.2 Device Connection

The ClockMatrix device is expected to output 2 signals to the PTP time stamping device.

- x MHz as 1588 PTP clock
- 1 PPS to align the rising clock edge of the `PTP Timer` with `ClockMatrix`

```
      PTP 0                                          PTP 1
+-----------------+                               +-------------------+
|               IN |<------------------------------|OUT                |
|                  | 1588 PTP clock          x MHz |                   |
|    PTP Timer     |                               |    ClockMatrix    |
|                  |                               |                   |
|               IN |<------------------------------|OUT                |
+-----------------+ 1588 Timestamp event    1 PPS +-------------------+

 PTP Timestamper driver supports: PTP_CLK_REQ_EXTTS
```

4.3 Kernel Device Tree

Insert ClockMatrix in the appropriate section of the device tree.

Sample entry on the Xilinx ZCU102 board:

```
...
i2c-mux@75 {
   i2c@1 {
      phc@5b { /* Clock Matrix */
         compatible = "idt,8a34000";
         reg = <0x5b>;
      };
   };
...
```

4.4 Kernel .config

```
#
# PTP clock support
#
CONFIG_PTP_1588_CLOCK=y

#
# Enable PHYLIB and NETWORK_PHY_TIMESTAMPING to see the additional clocks.
#
CONFIG_PTP_1588_CLOCK_IDTCM=m
```

4.5 Device Configuration

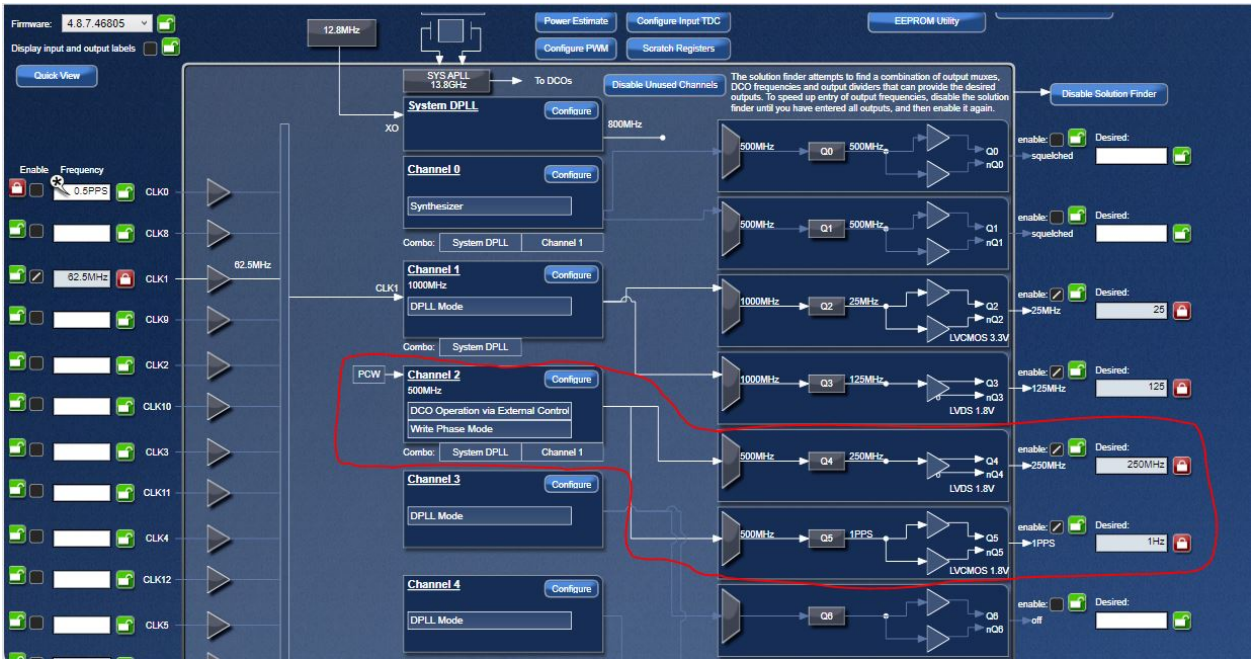
When the ClockMatrix driver is loaded, it looks in `/lib/firmware` a configuration file to load into the ClockMatrix device. The filename is determined during compile time by `FW_FILENAME`:

```
static int idtcm_load_firmware(struct idtcm *idtcm,
                               struct device *dev)
{
   const struct firmware *fw;

   ...

   err = request_firmware(&fw, FW_FILENAME, dev);
```

`FW_FILENAME` macro is defined in `drivers/ptp/ptp_clockmatrix.h`:

```
#define FW_FILENAME  "idtcm.bin"
```

Renesas Timing Commander Software using Clockmatrix Timing Commander Personality v8.4. can be used to generate ClockMatrix device configuration `.bin` file.



Please email IDT-support-1588@lm.renesas.com for device configuration support.

CHAPTER

FIVE

GETTING STARTED

`ts2phc` is used to align the PTP time stamping device with source of the PTP clocking device.

`ptp4l` is used to align the source of the PTP clocking device with a PTP master clock.

5.1 ts2phc

ts2phc, one of the utilties under Linux PTP, is used to align the time stamping device with the Renesas timing device.

```
# ./ts2phc -h

usage: ts2phc [options]

 -c [dev|name]  phc slave clock (like /dev/ptp0 or eth0)
                (may be specified multiple times)
 -f [file]      read configuration from 'file'
 -h             prints this message and exits
 -l [num]       set the logging level to 'num'
 -m             print messages to stdout
 -q             do not print messages to the syslog
 -s [dev|name]  source of the PPS signal
                may take any of the following forms:
                    generic   - an external 1-PPS without ToD information
                    /dev/ptp0 - a local PTP Hardware Clock (PHC)
                    eth0      - a local PTP Hardware Clock (PHC)
                    nmea      - a gps device connected by serial port or network
 -v             prints the software version and exits
```

5.2 ptp4l

ptp4l is an implementation of the Precision Time Protocol (PTP) according to IEEE standard 1588 for Linux. It implements Boundary Clock (BC), Ordinary Clock (OC), and Transparent Clock (TC).

```
usage: ptp4l [options]

 Delay Mechanism

 -A        Auto, starting with E2E
 -E        E2E, delay request-response (default)
 -P        P2P, peer delay mechanism
```

(continues on next page)

(continued from previous page)

```
 Network Transport

 -2        IEEE 802.3
 -4        UDP IPV4 (default)
 -6        UDP IPV6

 Time Stamping

 -H        HARDWARE (default)
 -S        SOFTWARE
 -L        LEGACY HW

 Other Options

 -f [file] read configuration from 'file'
 -i [dev]  interface device to use, for example 'eth0'
           (may be specified multiple times)
 -p [dev]  Clock device to use, default auto
           (ignored for SOFTWARE/LEGACY HW time stamping)
 -s        slave only mode (overrides configuration file)
 -l [num]  set the logging level to 'num'
 -m        print messages to stdout
 -q        do not print messages to the syslog
 -v        prints the software version and exits
 -h        prints this message and exits
```

Various ptp4l sample configuration files are in the <linuxptp>/configs directory.

For more info about ptp4l configuration parameters see man page for ptp4l.

```
man -l ptp4l.8
```

To use the adjust phase feature, the following configuration parameters are important:

```
servo_num_offset_values
   The number of offset values considered in order to transition from the
   SERVO_LOCKED to the SERVO_LOCKED_STABLE state.
   The transition occurs once the last 'servo_num_offset_values' offsets
   are all below the 'servo_offset_threshold' value.

servo_offset_threshold
   The offset threshold used in order to transition from the SERVO_LOCKED
   to the SERVO_LOCKED_STABLE state.  The transition occurs once the last
   'servo_num_offset_values' offsets are all below the threshold value.
   The default value of offset_threshold is 0 (disabled).

tsproc_mode
   Select the time stamp processing mode used to calculate offset and delay.
   Possible values are filter, raw, filter_weight, raw_weight. Raw modes perform
   well when the rate of sync messages (logSyncInterval) is similar to the rate of
   delay messages (logMinDelayReqInterval or logMinPdelayReqInterval). Weighting
   is useful with larger network jitters (e.g. software time stamping).
   The default is filter.

write_phase_mode
   This option enables using the "write phase" feature of a PTP Hardware
   Clock.  If supported by the device, this mode uses the hardware's
```

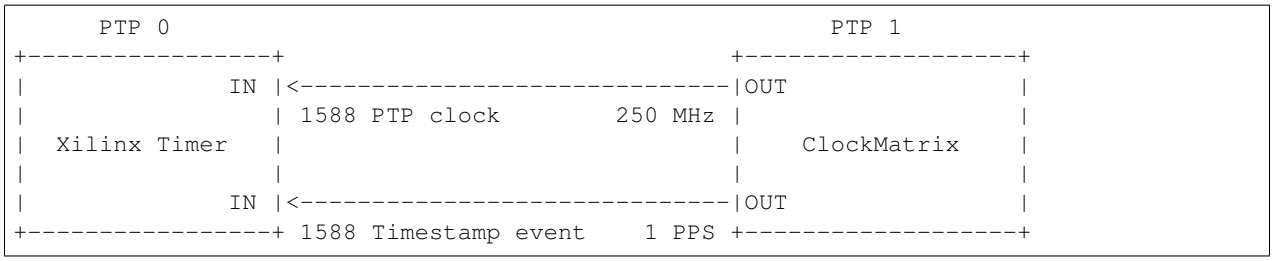
(continues on next page)

(continued from previous page)

```
   built in phase offset control instead of frequency offset control.
   The default value is 0 (disabled).
```

5.3 Sample Session

The sample session below is on a Xilinx ZCU102 board with ClockMatrix device. CM DPLL2 is the 1588 PTP clock to the Xilinx Timer. The Xilinx Timer is registered as ptp0 and ClockMatrix is registerd as ptp1.



5.3.1 Start ts2phc

We want time adjustments and no frequency adjustments, so we configure `ts2phc` to use the `nullf` servo.

Sample configuration `ts2phc.cfg`:

```
$ cat ts2phc.cfg

#
# ts2phc config file to get it to behave like syncd to align
# timestamper to PHC device's 1 PPS signal.
#
# Example:
# ./ts2phc -m -q -f ts2phc.cfg
#

[global]
clock_servo             nullf
first_step_threshold    0.000000001
step_threshold          0.000000001

# timestamper, slave device
[/dev/ptp0]
ts2phc.channel          0

# PHC device (ex. CM), master device
[/dev/ptp1]
ts2phc.master           1
ts2phc.channel          5
```

Continuing with the example on the Xilinx ZCU102 board:

- Xilinx time stamping device is `eth0` or `ptp0`
- ClockMatrix is `ptp1`

ptp0 is the slave clock, ie. it needs to align itself with the incoming 1 PPS from the source clock (ClockMatrix).

When first starting ts2phc, it may be beneficial to see the log messages.

```
$ ./ts2phc -m -q -f ts2phc.cfg -l 7

ts2phc[169484.381]: config item /dev/ptp0.ts2phc.master is 0
ts2phc[169484.382]: config item (null).clock_servo is 3
ts2phc[169484.382]: config item /dev/ptp0.ts2phc.pin_index is 0
ts2phc[169484.382]: config item /dev/ptp0.ts2phc.channel is 0
ts2phc[169484.382]: config item /dev/ptp0.ts2phc.extts_polarity is 2
ts2phc[169484.382]: config item /dev/ptp0.ts2phc.extts_correction is 0
ts2phc[169484.382]: config item /dev/ptp0.ts2phc.pulsewidth is 500000000
ts2phc[169484.382]: config item (null).free_running is 0
ts2phc[169484.382]: PHC slave /dev/ptp0 has ptp index 0
ts2phc[169484.382]: config item (null).step_threshold is 0.000000
ts2phc[169484.382]: config item (null).first_step_threshold is 0.000000
ts2phc[169484.382]: config item (null).max_frequency is 900000000
ts2phc[169484.382]: config item (null).servo_offset_threshold is 0
ts2phc[169484.382]: config item (null).servo_num_offset_values is 10
ts2phc[169484.382]: config item /dev/ptp1.ts2phc.master is 1
ts2phc[169484.383]: PHC master /dev/ptp1 has ptp index -1
ts2phc[169484.383]: config item /dev/ptp1.ts2phc.channel is 5
ts2phc[169484.383]: config item /dev/ptp1.ts2phc.pin_index is 0
PTP_PIN_SETFUNC failed: Invalid argument
ts2phc[169484.383]: Failed to set the pin. Continuing bravely on...
ts2phc[169485.113]: /dev/ptp0 extts index 0 at 169492.956561892 corr 0 src 10.465738␣
↪diff 169482956561892
ts2phc[169485.113]: /dev/ptp0 master offset 169482956561892 s1 freq      +0
ts2phc[169486.112]: /dev/ptp0 extts index 0 at 11.000000000 corr 0 src 11.265748 diff␣
↪0
ts2phc[169486.112]: /dev/ptp0 master offset          0 s2 freq      +0
ts2phc[169487.112]: /dev/ptp0 extts index 0 at 12.000000000 corr 0 src 12.265798 diff␣
↪0
ts2phc[169487.112]: /dev/ptp0 master offset          0 s2 freq      +0
ts2phc[169488.112]: /dev/ptp0 extts index 0 at 13.000000000 corr 0 src 13.265798 diff␣
↪0
ts2phc[169488.112]: /dev/ptp0 master offset          0 s2 freq      +0
```

`Failed to set the pin. Continuing bravely on...` is expected, this will be addressed in future version of the ClockMatrix PHC driver.

`ptp0 extts index 0 at 169492.956561892` tells us the counter value on the rising edge of the incoming 1 PPS signal on ptp0. Alignment is achieved when the sub-second portion of the counter is 0.

ts2phc shows that `ptp0` had a 956561892 ns offset at the rising edge of the incoming 1 PPS signal.

```
ts2phc[169485.113]: /dev/ptp0 extts index 0 at 169492.956561892 corr 0 src 10.465738␣
↪diff 169482956561892
```

ts2phc adjusted Xilinx's timer to reduce the sub-second offset to 0.

```
ts2phc[169485.113]: /dev/ptp0 master offset 169482956561892 s1 freq      +0
```

ts2phc subsequently reports that ptp0 is aligned with ptp1's PPS, ie. `diff 0` The Xilinx timer time stamp jumped from `xxxx.956561892` to `xxxx.000000000`.

```
ts2phc[169486.112]: /dev/ptp0 extts index 0 at 11.000000000 corr 0 src 11.265748 diff␣
↪0
```

The subsequent 1 PPS events show 0 sub-second values `12.000000000` and `13.000000000` which indicates

ptp0 is aligned with the incoming 1 PPS signal from ClockMatrix.

```
ts2phc[169487.112]: /dev/ptp0 extts index 0 at 12.000000000 corr 0 src 12.265798 diff␣
↪0
ts2phc[169488.112]: /dev/ptp0 extts index 0 at 13.000000000 corr 0 src 13.265798 diff␣
↪0
```

Once the system is behaving as expected, ts2phc can be started as a background process without the verbose messaging.

```
$ ./ts2phc -m -q -f ts2phc.cfg &
PTP_PIN_SETFUNC failed: Invalid argument
ts2phc[8331.038]: Failed to set the pin. Continuing bravely on...
```

5.3.2 Start ptp4l

5.3.2.1 Unicast

Below is a sample ptp4l configuration file for using adjust phase and a unicast master at 1 packet per second. Please ensure the domainNumber matches the unicast master's domainNumber otherwise the unicast requests will be ignored by the unicast master.

```
$ cat uslave_1pps_wp.cfg
   #
   # Enters adjust phase mode when display shows "s3 freq     +0".
   #
   #   ex. master offset        -48 s3 freq      +0
   #
   # For 16 pps:
   #   logSyncInterval     -4
   #   logMinDelayReqInterval    -4
   #   ...
   #   servo_num_offset_values   160
   #

   [global]

   domainNumber      0
   clockClass        255
   clockAccuracy     0x21
   masterOnly        0

   inhibit_multicast_service  1
   unicast_listen             1

   announceReceiptTimeout  3
   logAnnounceInterval     1
   logSyncInterval         0
   logMinDelayReqInterval  0

   first_step_threshold    0.001000000
   step_threshold          0

   clock_servo             pi

   write_phase_mode        1
```

(continues on next page)

(continued from previous page)

```
  servo_offset_threshold  50
  servo_num_offset_values 10
  tsproc_mode             raw

  sanity_freq_limit       0

  #
  # Request service for sixty seconds.
  #
  unicast_req_duration    60

  #
  # This table has four possible UDPv4 master clocks.
  #
  [unicast_master_table]
  table_id                1
  logQueryInterval        2
  UDPv4                   10.64.10.1

  #
  # eth0 uses the master table with ID 1 over UDPv4.
  #
  [eth0]
  unicast_master_table    1
```

With `ts2phc` running in the background, `ptp4l` is used to synchronize with a PTP master.

```
./ptp4l -m -q -p /dev/ptp1 -s -f uslave_1pps_wp.cfg
```

Sample unicast log:

```
$ ./ptp4l -m -q -p /dev/ptp1 -s -f uslave_1pps_wp.cfg

ptp4l[148.656]: selected /dev/ptp1 as PTP clock.cfg
ptp4l[148.657]: port 1: taking /dev/ptp1 from the command line, not the attached ptp0
ptp4l[148.668]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[148.668]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[153.886]: port 1: new foreign master 00b0ae.fffe.02e810-1
ptp4l[156.563]: selected local clock 000a35.fffe.00bf01 as best master
ptp4l[157.885]: selected best master clock 00b0ae.fffe.02e810
ptp4l[157.885]: updating UTC offset to 37
ptp4l[157.885]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[162.572]: master offset -1595444236006389861 s0 freq      +0 path delay      ␣
↪5422
ptp4l[163.577]: master offset -1595444236006389881 s1 freq     -20 path delay      ␣
↪5422
ptp4l[164.572]: master offset -797722117003244291 s2 freq -244000 path delay      5422
ptp4l[164.573]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[165.573]: master offset -797722118003056843 s2 freq -244000 path delay      5422
ptp4l[166.572]: master offset     510865 s2 freq +244000 path delay   -107180
ptp4l[167.572]: master offset     389063 s2 freq +244000 path delay   -107180
ptp4l[168.572]: master offset     116651 s2 freq +116631 path delay     43400
ptp4l[169.572]: master offset     -80505 s2 freq  -45530 path delay     60172
ptp4l[170.572]: master offset     -57799 s2 freq  -46975 path delay     60172
ptp4l[171.572]: master offset      -2721 s2 freq   -9237 path delay     -7118
ptp4l[172.572]: master offset      13365 s2 freq   +6033 path delay      4848
ptp4l[173.572]: master offset      11833 s2 freq   +8510 path delay      7948
```

(continues on next page)

(continued from previous page)

```
ptp4l[174.572]: master offset       3421 s2 freq   +3648 path delay      9048
ptp4l[175.571]: master offset       1577 s2 freq   +2830 path delay      9048
ptp4l[176.572]: master offset      -3265 s2 freq   -1538 path delay      6294
ptp4l[177.571]: master offset      -2721 s2 freq   -1974 path delay      5030
ptp4l[178.571]: master offset       -759 s2 freq    -828 path delay      4776
ptp4l[179.571]: master offset        111 s2 freq    -186 path delay      5252
ptp4l[180.571]: master offset        399 s2 freq    +135 path delay      5408
ptp4l[181.571]: master offset        331 s2 freq    +187 path delay      5446
ptp4l[182.571]: master offset        111 s2 freq     +66 path delay      5450
ptp4l[183.571]: master offset        -93 s2 freq    -104 path delay      5462
ptp4l[184.571]: master offset        -83 s2 freq    -122 path delay      5404
ptp4l[185.571]: master offset        -21 s2 freq     -85 path delay      5394
ptp4l[186.571]: master offset         -9 s2 freq     -79 path delay      5394
ptp4l[187.570]: master offset         27 s2 freq     -46 path delay      5418
ptp4l[188.570]: master offset         27 s2 freq     -38 path delay      5418
ptp4l[189.570]: master offset         15 s2 freq     -42 path delay      5418
ptp4l[190.570]: master offset         -1 s2 freq     -53 path delay      5418
ptp4l[191.570]: master offset         -7 s2 freq     -60 path delay      5420
ptp4l[192.570]: master offset        -13 s2 freq     -68 path delay      5434
ptp4l[193.570]: master offset        -11 s2 freq     -70 path delay      5428
ptp4l[194.570]: master offset         19 s3 freq      +0 path delay      5410
ptp4l[195.570]: master offset         13 s3 freq      +0 path delay      5428
ptp4l[196.570]: master offset         25 s3 freq      +0 path delay      5416
ptp4l[197.570]: master offset         59 s3 freq      +0 path delay      5410
ptp4l[198.569]: master offset         47 s3 freq      +0 path delay      5410
ptp4l[199.569]: master offset         69 s3 freq      +0 path delay      5416
ptp4l[200.569]: master offset         81 s3 freq      +0 path delay      5424
ptp4l[201.569]: master offset         83 s3 freq      +0 path delay      5422
ptp4l[202.569]: master offset         37 s3 freq      +0 path delay      5440
ptp4l[203.569]: master offset         13 s3 freq      +0 path delay      5440
ptp4l[204.569]: master offset         17 s3 freq      +0 path delay      5432
ptp4l[205.569]: master offset         11 s3 freq      +0 path delay      5434
ptp4l[206.569]: master offset         11 s3 freq      +0 path delay      5434
...
ptp4l[311.559]: master offset          3 s3 freq      +0 path delay      5430
ptp4l[312.559]: master offset         19 s3 freq      +0 path delay      5418
ptp4l[313.558]: master offset          1 s3 freq      +0 path delay      5436
```

5.3.2.2 Multicast

Below is a sample `ptp4l` configuration file for write phase mode and a multicast master (16 packet per seccond).

```
$ cat multicast_L2_wp.cfg

#
# Enters adjust phase mode when display shows "s3 freq     +0".
#
#   ex. master offset        -48 s3 freq      +0
#

[global]
domainNumber            24
network_transport       L2

first_step_threshold    0.001000000
```

(continues on next page)

(continued from previous page)

```
step_threshold          0

clock_servo             pi

write_phase_mode        1
servo_offset_threshold  100
servo_num_offset_values 64
tsproc_mode             raw

sanity_freq_limit       0

#
# eth0
#
[eth0]
```

With `ts2phc` running in the background, `ptp4l` is used to synchronize with a PTP multicast master.

`./ptp4l -m -q -p /dev/ptp1 -s -f multicast_L2_wp.cfg`

```
$ ./ptp4l -m -q -p /dev/ptp1 -s -f multicast_L2_wp.cfg

ptp4l[182.806]: selected /dev/ptp1 as PTP clock
ptp4l[182.808]: port 1: taking /dev/ptp1 from the command line, not the attached ptp0
ptp4l[182.853]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[182.853]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[182.894]: port 1: new foreign master 0080ea.fffe.4de260-1
ptp4l[183.144]: selected best master clock 0080ea.fffe.4de260
ptp4l[183.144]: updating UTC offset to 37
ptp4l[183.144]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[183.852]: port 1: minimum delay request interval 2^-4
ptp4l[184.152]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[184.839]: rms 1595444825147073024 max 1595444825147131904 freq -183009 +/-␣
↪105639 delay  2322 +/- 2594
ptp4l[185.839]: rms 445940385422364160 max 1595444825146950912 freq -164754 +/- 51704␣
↪delay  2294 +/- 1908
ptp4l[186.839]: rms 296386 max 402218 freq +224194 +/- 76709 delay -14613 +/- 64966
ptp4l[187.839]: rms 102476 max 179252 freq +161578 +/- 42173 delay  7786 +/- 1484
ptp4l[188.839]: rms 17214 max 25514 freq +37892 +/- 26203 delay  5806 +/- 619
ptp4l[189.839]: rms 21913 max 25586 freq -11870 +/- 4835 delay  4991 +/-  94
ptp4l[190.839]: rms 8473 max 14104 freq -12029 +/- 2523 delay  4906 +/- 121
ptp4l[191.838]: rms 1149 max 1966 freq  -3580 +/- 1988 delay  5087 +/-  53
ptp4l[192.838]: rms 1545 max 1710 freq   +513 +/- 470 delay  5153 +/-  11
ptp4l[193.838]: rms  711 max 1110 freq   +843 +/- 153 delay  5157 +/-   9
ptp4l[194.838]: rms  100 max  234 freq   +249 +/- 154 delay  5147 +/-   8
ptp4l[195.838]: rms  108 max  122 freq    -78 +/-  46 delay  5143 +/-   6
ptp4l[196.838]: rms   57 max   96 freq   -122 +/-  16 delay  5141 +/-   9
ptp4l[197.838]: rms   12 max   28 freq    -82 +/-  16 delay  5146 +/-   7
ptp4l[198.838]: rms   11 max   20 freq    -54 +/-  12 delay  5148 +/-   5
ptp4l[199.838]: rms    8 max   14 freq    -54 +/-  12 delay  5150 +/-   6
ptp4l[200.838]: rms    7 max   18 freq    -50 +/-  17 delay  5150 +/-   7
ptp4l[201.838]: rms   10 max   20 freq     +0 +/-   0 delay  5146 +/-   8
ptp4l[202.837]: rms   18 max   34 freq     +0 +/-   0 delay  5148 +/-   6
ptp4l[203.837]: rms   28 max   46 freq     +0 +/-   0 delay  5148 +/-   7
ptp4l[204.837]: rms   37 max   48 freq     +0 +/-   0 delay  5148 +/-   7
ptp4l[205.837]: rms   46 max   56 freq     +0 +/-   0 delay  5145 +/-   6
ptp4l[206.837]: rms   48 max   62 freq     +0 +/-   0 delay  5148 +/-   6
```

(continues on next page)

(continued from previous page)

```
ptp4l[207.837]: rms   38 max   56 freq     +0 +/-   0 delay  5144 +/-   5
ptp4l[208.837]: rms   34 max   50 freq     +0 +/-   0 delay  5150 +/-   7
ptp4l[209.837]: rms   28 max   42 freq     +0 +/-   0 delay  5144 +/-   6
ptp4l[210.837]: rms   21 max   32 freq     +0 +/-   0 delay  5146 +/-   6
ptp4l[211.836]: rms   23 max   34 freq     +0 +/-   0 delay  5147 +/-   5
ptp4l[212.836]: rms   22 max   32 freq     +0 +/-   0 delay  5146 +/-   7
ptp4l[213.836]: rms   20 max   34 freq     +0 +/-   0 delay  5147 +/-   9
ptp4l[214.836]: rms   21 max   38 freq     +0 +/-   0 delay  5146 +/-   6
ptp4l[215.836]: rms   21 max   30 freq     +0 +/-   0 delay  5144 +/-   7
ptp4l[216.836]: rms   21 max   30 freq     +0 +/-   0 delay  5147 +/-   6
ptp4l[217.836]: rms   24 max   36 freq     +0 +/-   0 delay  5146 +/-   5
ptp4l[218.836]: rms   18 max   34 freq     +0 +/-   0 delay  5147 +/-   7
ptp4l[219.836]: rms   19 max   34 freq     +0 +/-   0 delay  5150 +/-   4
ptp4l[220.836]: rms   20 max   38 freq     +0 +/-   0 delay  5146 +/-   7
ptp4l[221.835]: rms   20 max   32 freq     +0 +/-   0 delay  5144 +/-   8
ptp4l[222.835]: rms   19 max   32 freq     +0 +/-   0 delay  5147 +/-   7
ptp4l[223.835]: rms   20 max   34 freq     +0 +/-   0 delay  5149 +/-   9
ptp4l[224.835]: rms   22 max   34 freq     +0 +/-   0 delay  5148 +/-   6
ptp4l[225.835]: rms   21 max   40 freq     +0 +/-   0 delay  5147 +/-   9
ptp4l[226.835]: rms   21 max   30 freq     +0 +/-   0 delay  5149 +/-   7
```

CHAPTER

SIX

APPENDIX

6.1 Identifying PHC Device Number

6.1.1 ClockMatrix

On a successful CM PHC driver load, the registered ptp device number can be see in the system log.

View the kernel start-up log with 'dmesg' command.

```
dmesg | grep idt
```

```
[    3.419848] idtcm 15-005b: 4.8.0, Id: 0x4001  HW Rev: 5  OTP Config Select: 15
[    7.128855] idtcm 15-005b: PLL2 registered as ptp1
```

6.1.2 Network Interface

The ethernet interface in this example is `eth0`, it uses the Xilinx Timer as it's PTP time stamping device.

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:BF:01
          inet addr:10.64.10.191  Bcast:10.64.11.255  Mask:255.255.254.0
          inet6 addr: fe80::20a:35ff:fe00:bf01%4886680/64 Scope:Link
```

`PTP Hardware Clock: 0` indicates that eth0 is registered as `ptp0`.

If the ethernet interface is not supported as a PTP hardware clock, it will say `PTP Hardware Clock: none`. For support in getting ethernet interface to register as a PTP Hardware Clock, contact PHY/MAC vendor for support.

```
$ ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
```

6.2 View ptp Device Name

View registered ptp devices:

```
$ ls /sys/class/ptp
ptp0  ptp1
```

View name of a ptp device:

```
$ cat /sys/class/ptp/ptp1/clock_name
IDT CM PLL2

$ cat /sys/class/ptp/ptp0/clock_name
Xilinx Timer
```

6.3 testptp

testptp is a Linux PHC test tool that is part of the kernel.

It is located in `tools/testing/selftests/ptp/testptp.c`.

```
usage: testptp [options]
 -a val      request a one-shot alarm after 'val' seconds
 -A val      request a periodic alarm every 'val' seconds
 -c          query the ptp clock's capabilities
 -d name     device to open
 -e val      read 'val' external time stamp events
 -f val      adjust the ptp clock frequency by 'val' ppb
 -g          get the ptp clock time
 -h          prints this message
 -i val      index for event/trigger
 -k val      measure the time offset between system and phc clock
             for 'val' times (Maximum 25)
 -l          list the current pin configuration
 -L pin,val configure pin index 'pin' with function 'val'
             the channel index is taken from the '-i' option
             'val' specifies the auxiliary function:
             0 - none
             1 - external time stamp
             2 - periodic output
 -p val      enable output with a period of 'val' nanoseconds
 -P val      enable or disable (val=1|0) the system clock PPS
 -s          set the ptp clock time from the system time
 -S          set the system time from the ptp clock time
 -t val      shift the ptp clock time by 'val' seconds
 -T val      set the ptp clock time to 'val' seconds
```

The example below is cross compiling the `testptp` tool in a linux v5.8-rc5 repository on an Ubuntu 16.04 build machine.

6.3.1 Sample Build for ZCU102 board

```
# Compile for ZCU102
$ sudo apt-get install gcc-aarch64-linux-gnu
$ export CROSS_COMPILE=/usr/bin/aarch64-linux-gnu-

# Linux kernel root (v5.8-rc5)
$ cd tools/testing/selftests/ptp
$ make
```

6.3.1.1 Compilation Error

When cross-compiling, the build machine's Linux kernel's `ptp_clock.h` may be outdated.

For example, the Ubuntu 16.04 build machine is using Linux v4.15.0.

```
$ uname -srm
Linux 4.15.0-107-generic x86_64
```

And produces the following compilation error:

```
# Linux kernel root (v5.8-rc5)
$ make
Makefile:10: warning: overriding recipe for target 'clean'
../lib.mk:126: warning: ignoring old recipe for target 'clean'
/usr/bin/aarch64-linux-gnu-gcc -I../../../../usr/include/    testptp.c  -lrt -o␣
↪testptp
testptp.c: In function ‘do_flag_test’:
testptp.c:61:3: error: ‘PTP_EXTTS_REQUEST2’ undeclared (first use in this function)
   PTP_EXTTS_REQUEST2,
   ^
testptp.c:61:3: note: each undeclared identifier is reported only once for each␣
↪function it appears in
testptp.c:68:25: error: ‘PTP_EXTTS_VALID_FLAGS’ undeclared (first use in this␣
↪function)
   PTP_ENABLE_FEATURE | (PTP_EXTTS_VALID_FLAGS + 1),
                         ^
testptp.c: In function ‘main’:
testptp.c:281:15: error: ‘struct ptp_clock_caps’ has no member named ‘adjust_phase’
           caps.adjust_phase);
               ^
<builtin>: recipe for target 'testptp' failed
make: *** [testptp] Error 1
```

6.3.1.2 Workaround

In `testptp.c`, replace `#include <linux/ptp_clock.h>` with a relative include to the local repo's `ptp_clock.h`.

```
testptp.c:
      #include <sys/time.h>
      #include <sys/timex.h>
      #include <sys/types.h>
      #include <time.h>
      #include <unistd.h>
```

(continues on next page)

(continued from previous page)

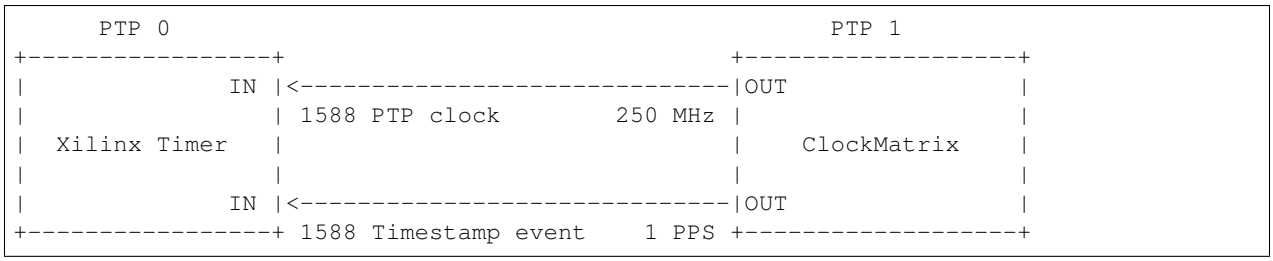
```
-      #include <linux/ptp_clock.h>
+      #include "../../../../include/uapi/linux/ptp_clock.h"
```

With above workaround, `testptp` compiles.

```
$ make
Makefile:10: warning: overriding recipe for target 'clean'
../lib.mk:126: warning: ignoring old recipe for target 'clean'
/usr/bin/aarch64-linux-gnu-gcc -I../../../../usr/include/    testptp.c  -lrt -o␣
↪testptp
```

6.3.2 Sanity Testing with tesptp

Examples below are run on a Xilinx ZCU102 board with ClockMatrix device. CM DPLL2 is the 1588 PTP clock to the Xilinx Timer. The Xilinx Timer is registered as ptp0 and ClockMatrix is registerd as ptp1.



6.3.2.1 Query Capabilities

```
$ ./testptp -d /dev/ptp0 -c
capabilities:
  999999999 maximum frequency adjustment (ppb)
  0 programmable alarms
  1 external time stamp channels
  0 programmable periodic signals
  0 pulse per second
  0 programmable pins
  0 cross timestamping

$ ./testptp -d /dev/ptp1 -c
capabilities:
  244000 maximum frequency adjustment (ppb)
  0 programmable alarms
  0 external time stamp channels
  12 programmable periodic signals
  0 pulse per second
  0 programmable pins
  0 cross timestamping
```

6.3.2.2 Get clock time (should increment)

If the ptp device is working (ie. has the input clock it needs, etc), repeated ./testptp -g on same device should show incrementing timestamp.

```
$./testptp -d /dev/ptp0 -g
clock time: 1595449857.649425272 or Wed Jul 22 20:30:57 2020

$ ./testptp -d /dev/ptp0 -g
clock time: 1595449859.400688308 or Wed Jul 22 20:30:59 2020

$ ./testptp -d /dev/ptp0 -g
clock time: 1595449860.264714740 or Wed Jul 22 20:31:00 2020


$ ./testptp -d /dev/ptp1 -g
clock time: 1595449899.113274703 or Wed Jul 22 20:31:39 2020

$ ./testptp -d /dev/ptp1 -g
clock time: 1595449900.072974835 or Wed Jul 22 20:31:40 2020

$ ./testptp -d /dev/ptp1 -g
clock time: 1595449901.088974943 or Wed Jul 22 20:31:41 2020
```

6.3.2.3 Read external timestamp event - should see 1 second increments

This is only relevant to Xilinx timer because it is suppose to have CM's 1 PPS as input to the timestamper.

Should show periodic 1 second events.

```
$ ./testptp -e 50 -d /dev/ptp0
external time stamp request okay
event index 0 at 1595450035.000000000
event index 0 at 1595450036.000000000
event index 0 at 1595450037.000000000
event index 0 at 1595450038.000000000
event index 0 at 1595450039.000000000
```

If there is not a 1 PPS clock going into the Xilinx timer, there will be no `event index` output. To quit, press Ctrl+C.

```
$ ./testptp -e 50 -d /dev/ptp0
external time stamp request okay
```

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.