Linux Plumbers Conference September 2022

Timed I/O: I/O Linked To System Time

Christopher Hall (christopher.s.hall@intel.com)

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

intel.

Introduction

- The Timed I/O device timestamps or generates external signal events based on the platform clock
- Timed I/O has been supported in Intel silicon since EHL/TGL (11th generation platforms)

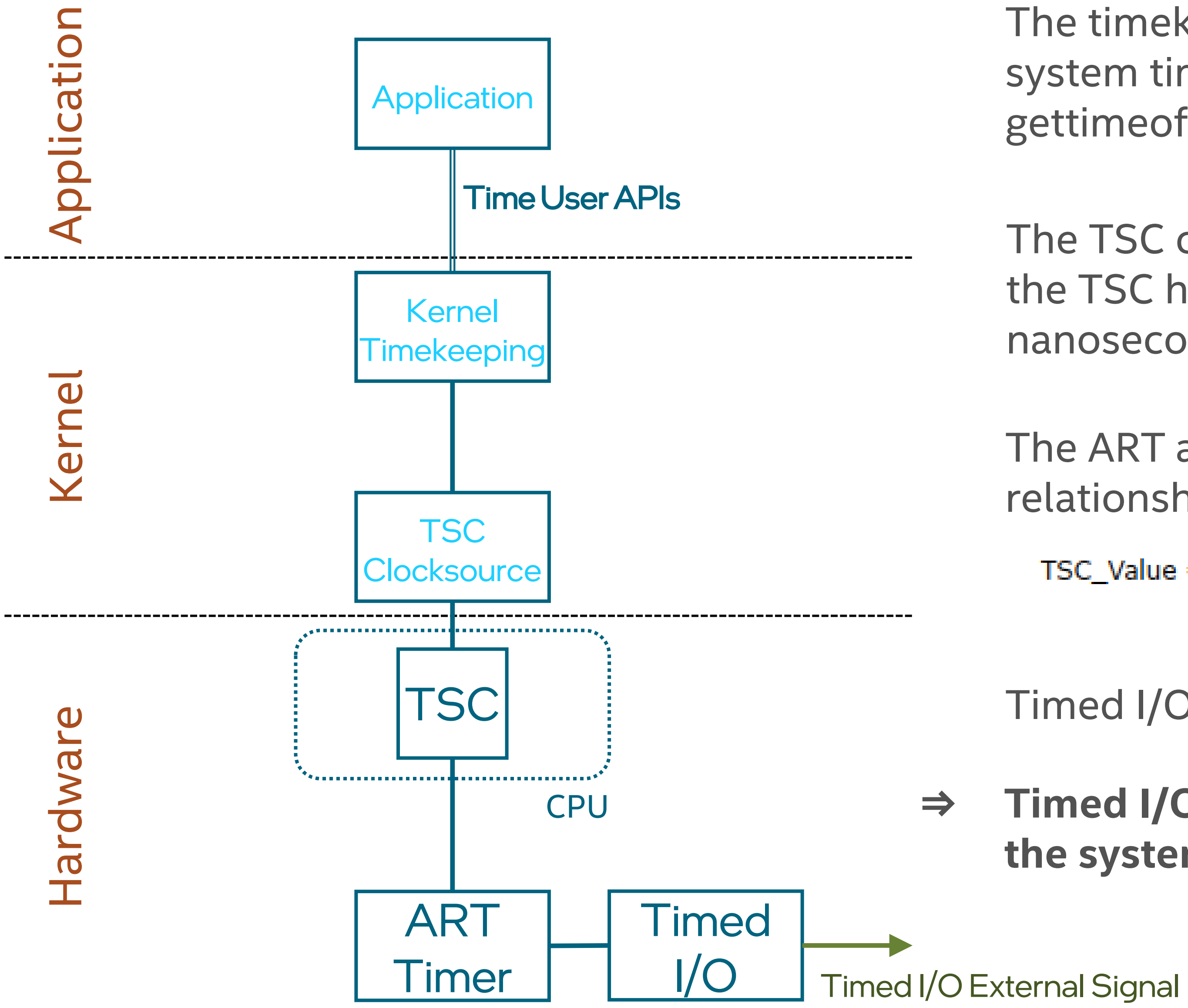
AGENDA

- Timed I/O Use Cases
- High-Level Hardware Architecture
- Hardware Function
- Alternatives – Why a new device type is needed
- User API
- Timekeeping Support

Use Cases

- Timed I/O is primarily used to import time from and export time to external devices
- Examples:
  - Import time from GPS module with PPS output
  - Export system time to compare clocks to measure accuracy of PTP time synchronization

HIGH-LEVEL HARDWARE ARCHITECTURE



The timekeeping kernel component implements the system time user APIs (e.g clock_gettime(), gettimeofday())

The TSC clocksource is the software representation of the TSC hardware converting TSC count to nanoseconds

The ART and TSC timers are phase locked and the relationship is defined by:

TSC_Value = (ART_Value * CPUID.15H:EBX[31:0])/ CPUID.15H:EAX[31:0] + K

Source: Intel Software Developer's Manual (SDM)

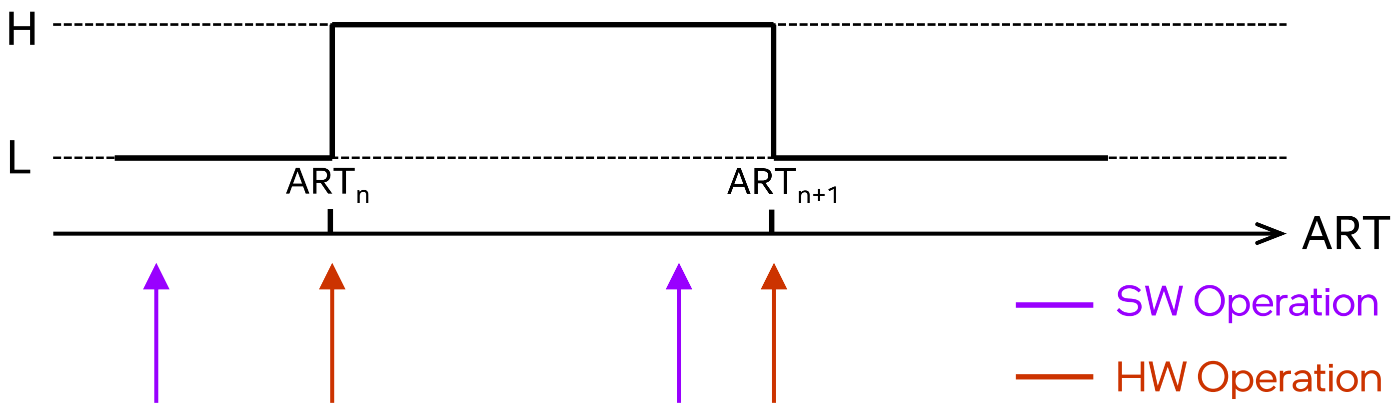
Timed I/O hardware is "driven by" ART

⇒ **Timed I/O hardware events are directly correlated with the system time**

Hardware Function – Output

Definition: an output event is a transition – low-to-high or high-to-low – of the output level driven by the platform on the Timed I/O signal

Single Programmed Events (Platform drives):



Write future ART value (ART_n) to comparator register

comparator == ART_n: hardware generates rising edge

Write future ART value (ART_{n+1}) to comparator register

comparator == ART_{n+1}: hardware generates falling edge

Hardware Function – Input

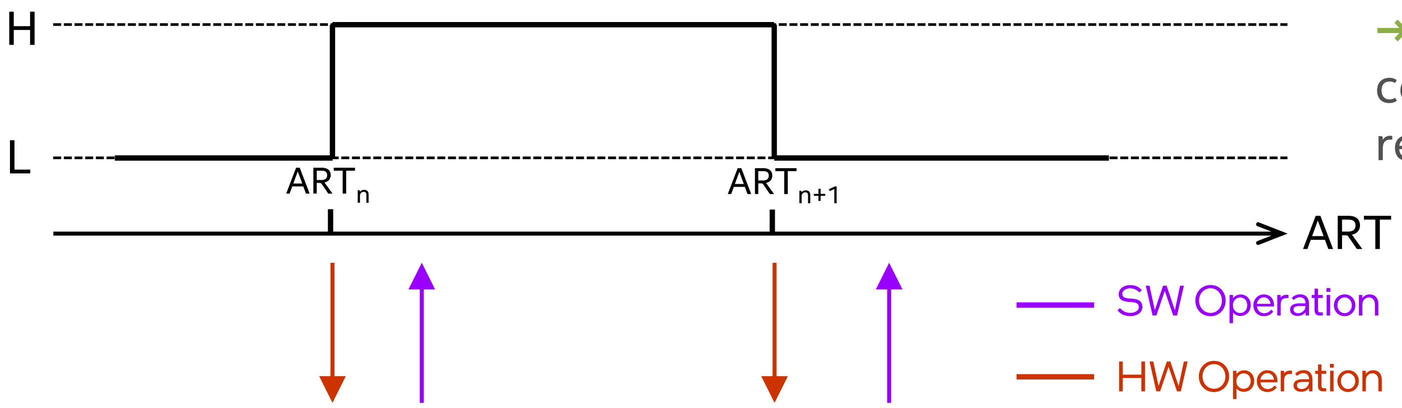
Definition: an input event is a transition – low-to-high or high-to-low – of the input level driven externally on the Timed I/O signal

Captured Events (Driven externally):

→The timestamp and count values are captured atomically

→The timestamp is over-written for each event

→The count is used to detect lost events or compute the average event rate with respect to ART



Hardware captures rising edge timestamp (time capture register = ART_n) and increments count value

Atomically read captured (ART_n, $count_n$) values

Hardware captures falling edge timestamp (time capture register = ART_{n+1}) and increments count value

Atomically read captured (ART_{n+1}, $count_{n+1}$) values

Hardware Function – Periodic Output

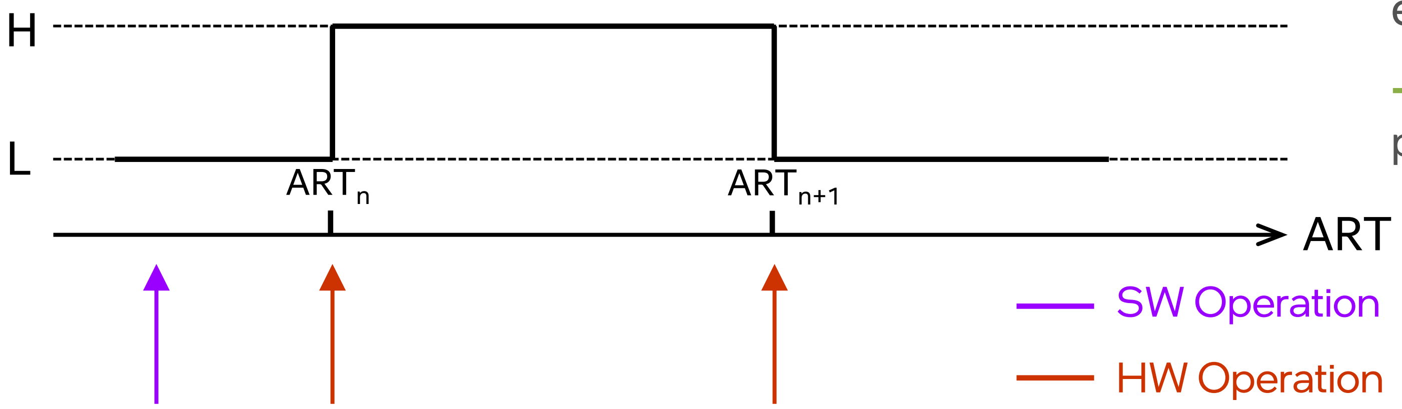
Periodic output extends the single programmed event model to re-trigger in hardware

→The timestamp and count values are captured atomically

→The count is used to compute the average event rate with respect to ART

→The computed event rate is used to adjust the periodic interval value

Periodic Programmed Events (Platform drives):



Write future ART value (ART_n) to comparator register and period (ART_{n+1}-ART_n) to periodic interval register

comparator == ART_n: hardware generates rising edge, increments count value, and adds periodic interval to the comparator

comparator == ART_{n+1}: hardware generates falling edge, increments the count value, and adds periodic interval to the comparator

ALTERNATIVES

- GPIO
  - Do not have output periodic or otherwise
  - Support for a polling interface is not present
- Comedi
  - No concept of system clock timestamping

API Overview

- Support input
- Support periodic and single shot output
- Support PPS input through existing PPS interface
- Support PPS output

PROPOSED API – CONFIGURATION

One character device per signal (e.g. /dev/timedioX)

Configuration:

```
#define TIMEDIO_INTERRUPT_CAPABLE 0x1

enum timedio_function { TIMEDIO_IN, TIMEDIO_OUT, TIMEDIO_PPS_IN, TIMEDIO_PPS_OUT };

struct timedio_config {
        enum timedio_function func;          /* Select signal function */
        clockid_t clockid;                   /* Select clock used for timestamping */
        unsigned int event_queue_size;       /* 1 = polled input interface, output = 1 */
        unsigned int capabilities;           /* e.g. check interrupt capable */
        char name[32];                       /* Name used to locate signal, for example, pad location, read only */
} timedio_config0;

ioctl( ..., TIMEDIO_SET_CONFIG, timedio_config0 );
ioctl( ..., TIMEDIO_GET_CONFIG, timedio_config0 );
```

PROPOSED API – INPUT

```
#define TIMEDIO_RISING_EDGE 0x1
#define TIMEDIO_FALLING_EDGE 0x2

ioctl( ..., TIMEDIO_INPUT_SET_EDGE_TYPE, unsigned edge_type );

#define TIMEDIO_TIME_INVALID 0x1;
struct timedio_time {
        __s64 sec;                    /* seconds */
        __u32 nsec;                   /* nanoseconds */
        unsigned int flags;
};

struct timedio_event {
        struct timedio_time event_time;
        unsigned int edge_type;
        unsigned int count;
} timedio_event0;

read( ..., timedio_event0, sizeof(timedio_event0));      /* Read event, return invalid time for empty queue */
```

PROPOSED API – OUTPUT

```
#define TIMEDIO_TIME_INVALID 0x1;
struct timedio_time {
        __s64 sec;                              /* seconds */
        __u32 nsec;                             /* nanoseconds */
        unsigned int flags;
} timedio_time0;

ioctl( …, TIMEDIO_OUTPUT_SET_PERIOD, timedio_time0 ); /* set invalid time for one shot */

#define TIMEDIO_RISING_EDGE 0x1
#define TIMEDIO_FALLING_EDGE 0x2
struct timedio_event {
        struct timedio_time event_time;
        unsigned int edge_type;                 /* ignored for output write */
} timedio_event0;

write( …, timedio_time0, sizeof(timedio_time0));        /* Generate event */

read( …, timedio_event0, sizeof(timedio_event0));       /* Read event */
```

PROPOSED API – PPS OUTPUT

Offset the PPS output

```
struct timedio_time {
        __s64 sec;                              /* seconds */
        __u32 nsec;                             /* nanoseconds */
        unsigned int flags;
} timedio_time0;

/* Offset the output PPS time by argument */
ioctl( ..., TIMEDIO_PPS_SET_OFFSET, timedio_time0 );
```

TIMEKEEPING SUPPORT

Translate between ART ↔ system clock

get_device_system_crosststamp() – exists, converts clocksource counter (TSC) → System Time
convert_art_to_tsc() – companion function in tsc.c

Propose:

ktime_real_get_cycles() – convert realtime clock to clocksource cycles
convert_tsc_to_art()