Open Source in Industry:
All about mainline real-time Linux

Technical Heidelberg OSADL Talks, April 29, 2020, Online Session 1

History and functionality of real-time Linux
Functionalities of PREEMPT_RT
Linux Foundation RTL Collaborative Project
Live real-time demonstrator

Some information on today’s sessions

- Please provide feedback on Legal HOT using the online form
  - Use the quick link **osadl.org/FB** (FeedBack), same as osadl.org/?id=3325
- You may ask questions during the session to be answered online, if possible
  - The quick link URL is **osadl.org/AQ** (AskQuestion), same as osadl.org/?id=3321
- You may join an online discussion on all topics of today at 4 pm
  - The quick link URL is **osadl.org/OD** (OnlineDiscussion), same as jitsi.osadl.org
  - Meeting name **OSADLTechnicalHOT**
  - Username and password will be displayed here after the last presentation

(We will show this slide again at the end of this session)

Quote of the year 2004

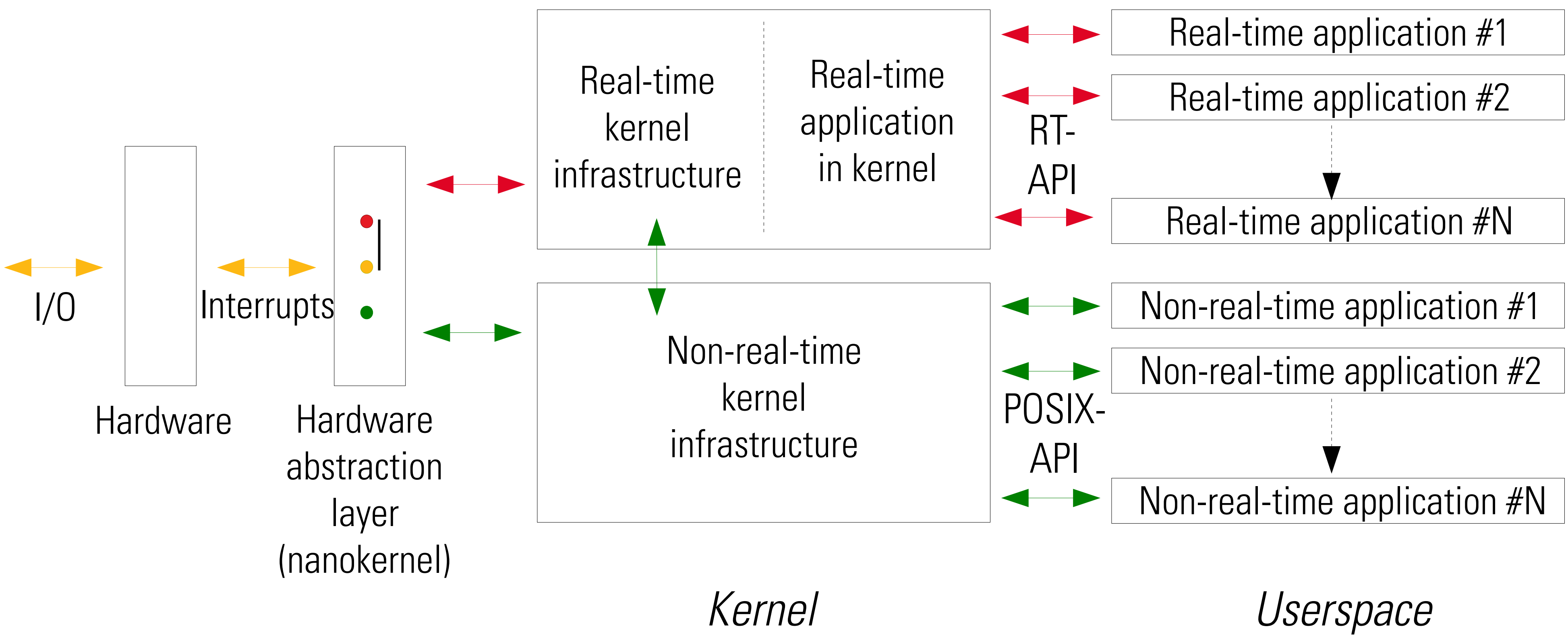
“It is *impossible* to turn a General Purpose

Operating System Kernel

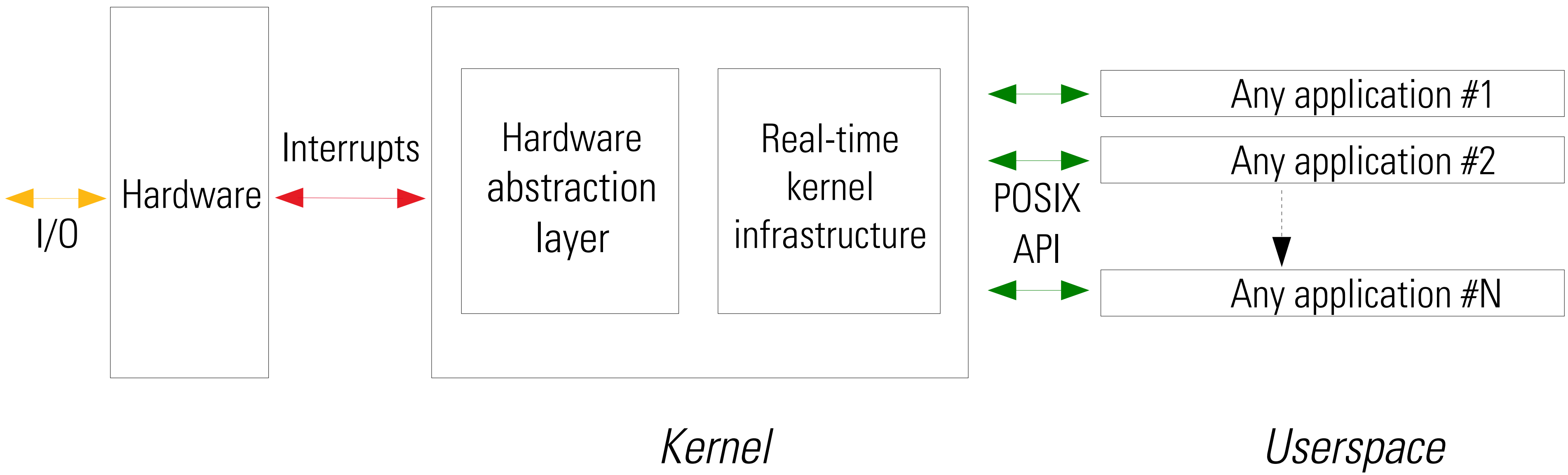
into a

RealTime Operating System Kernel.”

What is a real-time extension?



What is a real time operating system?



Terminology

- RTLinux (Dual-kernel approach) uses its own nanokernel

  RTAI, Xenomai, RTCore etc.

- CONFIG_PREEMPT_RT (Single-kernel approach)

  Linux mainline realtime, “Linux RTOS”

Goals

- Fully preemptive kernel
- Realtime guarantees suitable for the vast majority of applications
- POSIX compliance (single API):

  `sched_setscheduler()`

  `sched_setaffinity()`

  and friends

Driving forces

- Symmetric multi-processing
- (Bug fixing of race conditions)
- Audio recording and audio processing
- Video recording and video processing
- Reliable time-stamps in financial transactions
- Machine industry and embedded systems

People behind real-time Linux

- Doug Niehaus, University of Kansas
- Thomas Gleixner, Linutronix
- Ingo Molnàr, Red Hat
- Peter Zijlstra, Red Hat
- Paul E. McKenney, IBM
- Steven Rostedt, Red Hat
- many other

History

- Autumn 2004 MontaVista, Timesys, Lynuxworks post realtime related patch fragments
- Ingo Molnar re-implements parts from scratch and posts the real-time preemption patch
- A core team forms
- Kernel Summit 2006 in Ottawa accepts a plan to merge all components into mainline over time

Kernel-Summit, Ottawa, August 2006

Kernel-Summit, Ottawa, August 2006

"Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT."

Linus Torvalds

Main components

- Deterministic Scheduler
- Preemption Support
- PI Mutexes
- High-Resolution Timer
- Preemptive Read-Copy Update
- IRQ Threads (selected, forced)
- Raw Spinlock Annotation
- Preemptive Memory Management
- Full Realtime Preemption Support

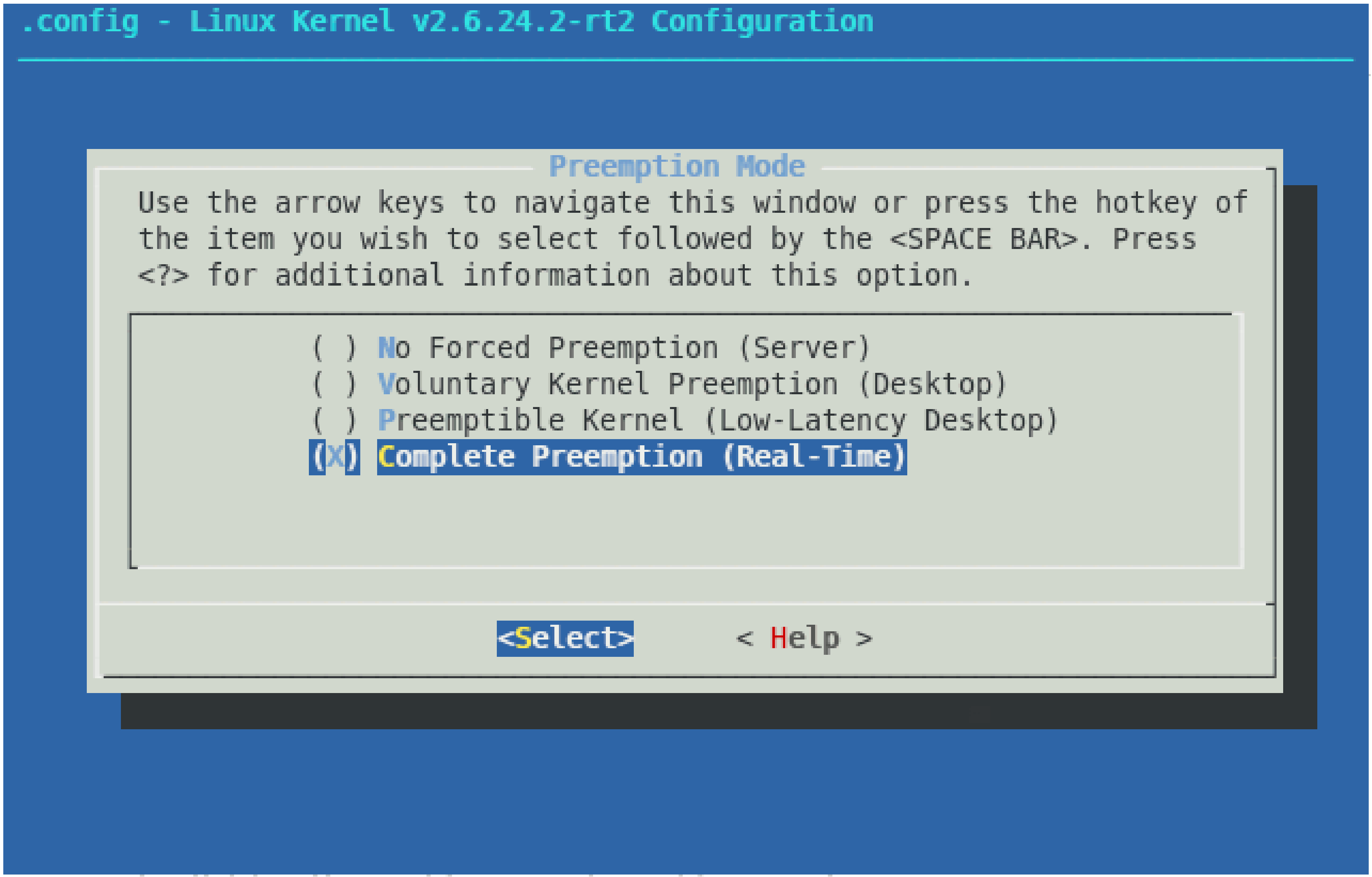
Merge details

Feature \ Architecture	x86	x86/64	powerpc	arm	mips	68knommu
Deterministic Scheduler	🟢	🟢	🟢	🟢	🟢	🟢
Preemption Support	🟢	🟢	🟢	🟢	🟢	🟢
PI Mutexes	🟢	🟢	🟢	🟢	🟢	🟡 [3]
High-Resolution Timer	🟢	🟢 [1]	🟢 [1]	🟢 [1]	🟢 [1]	🟢
Preemptive Read-Copy Update	🟢 [2]	🟢 [2]	🟢 [2]	🟢 [2]	🟢 [2]	🟢 [2]
IRQ Threads	🟢 [4]	🟢 [4]	🟢 [4]	🟢 [4]	🟢 [4]	🟡 [3,4,5]
Raw Spinlock Annotation	🟢 [6]	🟢 [6]	🟢 [6]	🟢 [6]	🟢 [6]	🟢 [6]
Full Realtime Preemption Support	🟡	🟡	🟡	🟡	🟡	🟡 [3]

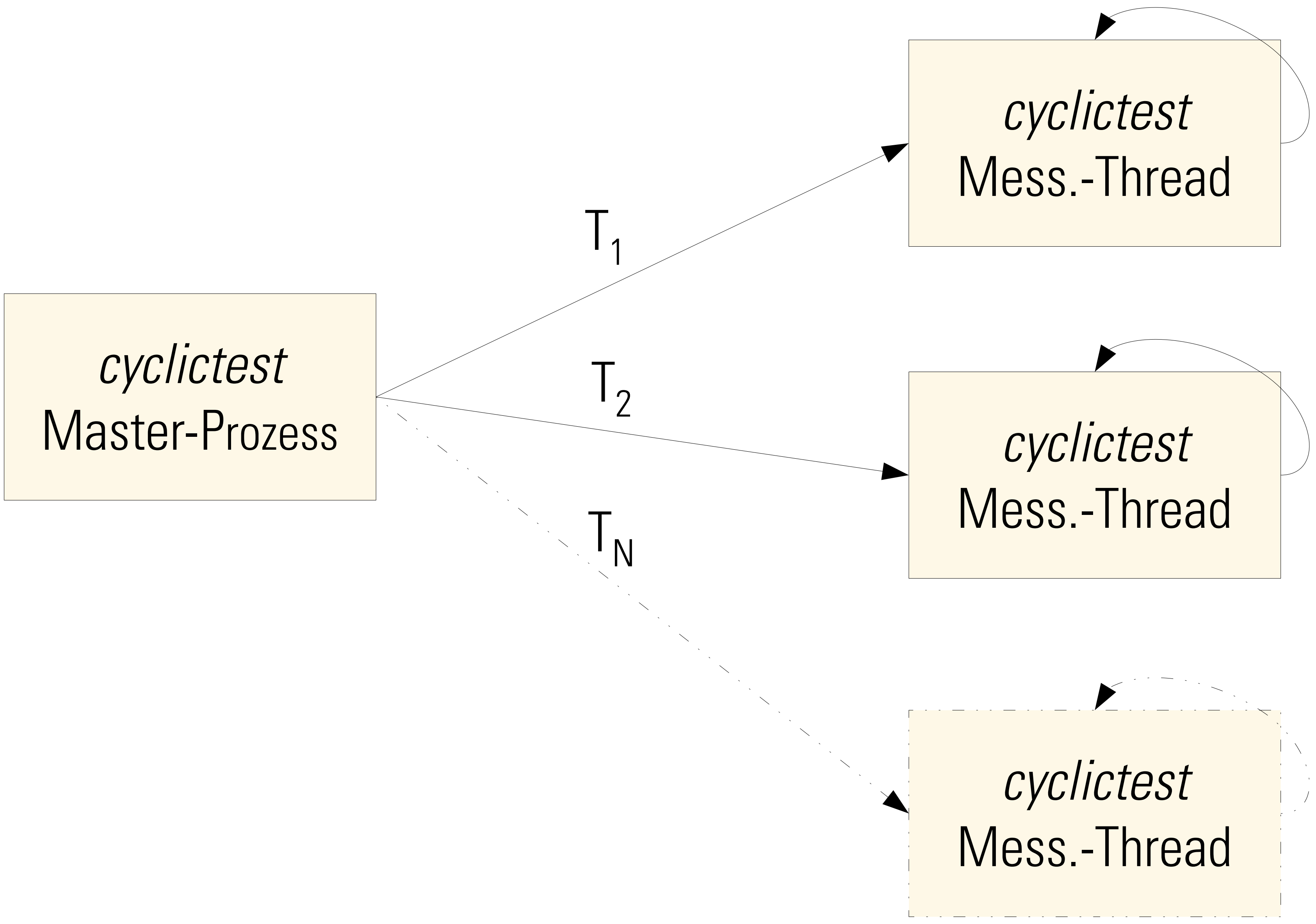
🟢 Available in mainline Linux

🟡 Available when Realtime-Preempt patches applied

Configuration



Latency measurement



Latency measurement

Run on CPU #0

Priority 99

High-resolution timer

Start one thread

Interval 100 µs

```
# cyclictest -n -a0 -t1  -p99 -i100

T: 0 (26623) P:99 I:100 C:2244063 Min:  2 Act:  5 Avg:  4 Max:   27
```

Process ID

Cycles

Worst-case latency in µs

Latency measurement

Run on CPU #0

Priority 99, 98, 97, ... 88

High-resolution timer

Start 12 threads

Interval 100 µs

No delay

```
# cyclictest -n -a0 -t12 -p99 -i100 -d0
T: 0 ( 2910) P:99 I:100 C:3217008 Min:  2 Act:  6 Avg:  4 Max:    32
T: 1 ( 2911) P:98 I:100 C:3217008 Min:  1 Act:  4 Avg:  3 Max:    59
T: 2 ( 2912) P:97 I:100 C:3217007 Min:  2 Act:  4 Avg:  3 Max:    47
T: 3 ( 2913) P:96 I:100 C:3217007 Min:  2 Act: 11 Avg:  3 Max:    53
T: 4 ( 2914) P:95 I:100 C:3217007 Min:  2 Act:  9 Avg:  4 Max:    53
T: 5 ( 2915) P:94 I:100 C:3217007 Min:  3 Act:  9 Avg:  7 Max:    89
T: 6 ( 2916) P:93 I:100 C:3217007 Min:  2 Act:  5 Avg:  4 Max:    85
T: 7 ( 2917) P:92 I:100 C:3217006 Min:  2 Act: 10 Avg:  5 Max:   119
T: 8 ( 2918) P:91 I:100 C:3217006 Min:  2 Act: 13 Avg:  9 Max:   148
T: 9 ( 2919) P:90 I:100 C:3217007 Min:  1 Act:  4 Avg:  4 Max:   178
T:10 ( 2920) P:89 I:100 C:3217006 Min:  1 Act:  4 Avg:  3 Max:  1413
T:11 ( 2921) P:88 I:100 C:3217006 Min:  3 Act:  7 Avg: 10 Max: 27331
```

Latency measurement

Run on all CPUs

Priority 99 of all threads

High-resolution timer

Start 12 threads

Interval 100 µs

No delay

```
# cyclictest       -S      -p99 -i100 -d0
T: 0 (15350) P:99 I:100 C:3839755 Min:  2 Act:  6 Avg:  3 Max:   24
T: 1 (15351) P:99 I:100 C:3839755 Min:  2 Act:  7 Avg:  4 Max:   19
T: 2 (15352) P:99 I:100 C:3839755 Min:  2 Act:  8 Avg:  4 Max:   27
T: 3 (15353) P:99 I:100 C:3839755 Min:  2 Act:  5 Avg:  4 Max:   24
T: 4 (15354) P:99 I:100 C:3839755 Min:  2 Act:  5 Avg:  3 Max:   20
T: 5 (15355) P:99 I:100 C:3839755 Min:  2 Act:  5 Avg:  5 Max:   52
T: 6 (15356) P:99 I:100 C:3839755 Min:  2 Act:  5 Avg:  4 Max:   20
T: 7 (15357) P:99 I:100 C:3839755 Min:  2 Act:  5 Avg:  3 Max:   17
T: 8 (15358) P:99 I:100 C:3839755 Min:  2 Act: 10 Avg:  4 Max:   28
T: 9 (15359) P:99 I:100 C:3839754 Min:  2 Act:  5 Avg:  4 Max:   22
T:10 (15360) P:99 I:100 C:3839755 Min:  2 Act:  5 Avg:  4 Max:   42
T:11 (15361) P:99 I:100 C:3839755 Min:  2 Act:  5 Avg:  5 Max:   34
```

Conclusion (1)

"It is *possible* to turn a General Purpose Operating

System Kernel

into a

Realtime Operating System Kernel."

Linux is a Real-Time Operating System (RTOS) now.

Conclusion (2)

- The real-time capabilities of the mainline Linux kernel are comparable to those that can be achieved using a traditional RTOS.
- The Linux real-time concept allows to switch to real-time without any major changes to existing drivers or user-space code.

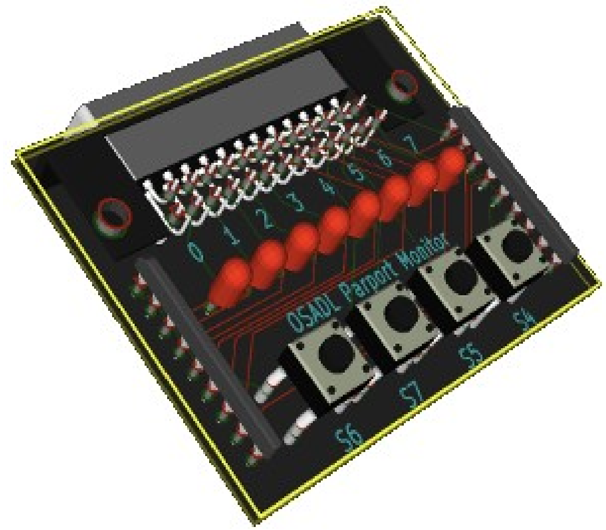
Conclusion (3)

- For the time being, however, the mainline kernel does not contain all needed components, but a patch (about 750 kByte) still needs to be applied.
- The maintenance and the final merging of the patches to mainline is funded by a Linux Foundation project. It is expected that an important progress towards complete mainline integration will happen in kernel 5.8.

Real-time demonstrator using square wave signal

OSADL Parport adapter (https://www.osadl.org/?id=1575)

PCIe Parallelport adapter



OSADL Parport adapter

OSADL Parport adapter
Shell interface

```
echo 0 .. 255 >/dev/setparport set output byte
echo 256 .. 511 >/dev/setparport boolean "or" action with output byte
echo 512 >/dev/setparport clear all output bits
echo 513 >/dev/setparport set all output bits
echo 514 >/dev/setparport invert output bits
echo 515 >/dev/setparport increment output bits
echo 516 >/dev/setparport decrement output bits
echo 517 >/dev/setparport copy status register to output bits
echo 518 >/dev/setparport copy jiffies LSBs >> 10 to output bits
```

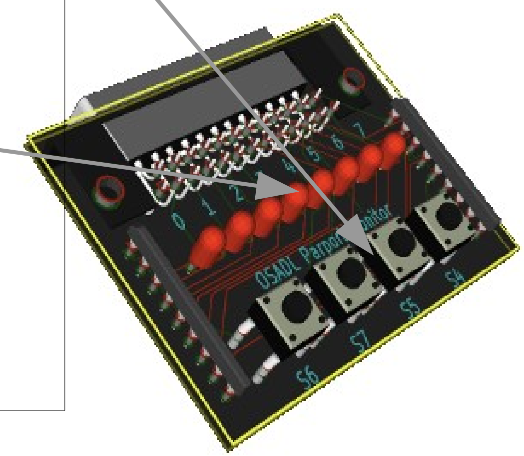
OSADL Parport adapter
NMI interface

The 4 input bits of the parallel port interface can be used for polling buttons when NMI triggers arrive. This makes it possible to execute particular actions such as SysReq commands in order to diagnose a crashed system. In addition NMIs can be used to generate LED display codes such as 515 to increment the displayed number:

```
modprobe setparport actions=yes nmicode=515
```

or

```
echo Y >/sys/module/setparport/parameters/actions
echo 515 >/sys/module/setparport/parameters/nmicode
```



OSADL Parport adapter
C interface

```
char *device = "/dev/setparport";
int port = open(device, O_RDWR);
int count = 1;

char *value = "0";
if (write(port, value, count) != count) {
  perror(errtext);
  return(errno);
}
```

Code *squarewave*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <locale.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define RECIPROCAL(a) (1.0/(a))
#define HALFCYCLE(a) ((a)/2.0)
#define NANOSECONDSPERSECOND 1000000000

int main(int argc, char *argv[])
{
  char *device = "/dev/setparport";
  int port = open(device, O_RDWR);
  char *value;
  int count = 1;
  struct timespec request, remain;
  char errtext[256];
  int i, verbose = 0;

  request.tv_sec = 0;
  request.tv_nsec = 500000; /* 500 microseconds (1 kHz) */

  for (i = 1; i < argc; i++) {
    if (!strcmp(argv[i], "-v"))
      verbose = 1;
    else if (!strcmp(argv[i], "-?")) {
      fprintf(stderr, "Syntax: squarewave [<option>] [<frequency in Hz>]\n");
      fprintf(stderr, "Function: Generate square wave at pin #0 of parport (default 1 kHz)\n");
      fprintf(stderr, "Options:   -v show calculated half wave cycle interval (verbose)\n");
      fprintf(stderr, "           -? you already figured this one out\n");
      exit(1);
    } else {
      char *endptr;
      double freq = strtod(argv[i], &endptr);
      double duration, wholeseconds;
      int endposition = endptr - argv[i];
```

```
      if (endposition < strlen(argv[i])) {
        fprintf(stderr, "Could not parse frequency input '%s' at position %d ('%c')\n",
          argv[i], endposition + 1, argv[i][endposition]);
        exit(1);
      }
      duration = HALFCYCLE(RECIPROCAL(freq));
      wholeseconds = (double) ((int) duration);
      request.tv_sec = (int) wholeseconds;
      request.tv_nsec = (int) ((duration - wholeseconds) * NANOSECONDSPERSECOND);
      if (verbose) {
        setlocale(LC_NUMERIC, "");
        printf("Half wave cycle interval is %d second%s and %'d nanosecond%s\n",
          request.tv_sec, request.tv_sec == 1 ? "" : "s",
          request.tv_nsec, request.tv_nsec == 1 ? "" : "s");
      }
    }
  }


  if (port < 0) {
    snprintf(errtext, sizeof(errtext),
      "Could not open device '%s'", device);
    perror(errtext);
    return(errno);
  }

  while (1) {
    value = "0";
    if (write(port, value, count) != count) {
      snprintf(errtext, sizeof(errtext),
        "Could not write %d value%s to device '%s' at port #%d",
        count, count == 1 ? "" : "s", device, port);
      perror(errtext);
      return(errno);
    }
    clock_nanosleep(CLOCK_MONOTONIC, 0, &request, &remain);
    value = "1";
    write(port, value, 1);
    clock_nanosleep(CLOCK_MONOTONIC, 0, &request, &remain);
  }

  close(port);
}
```

Code *squarewave* (selections)

Calculation

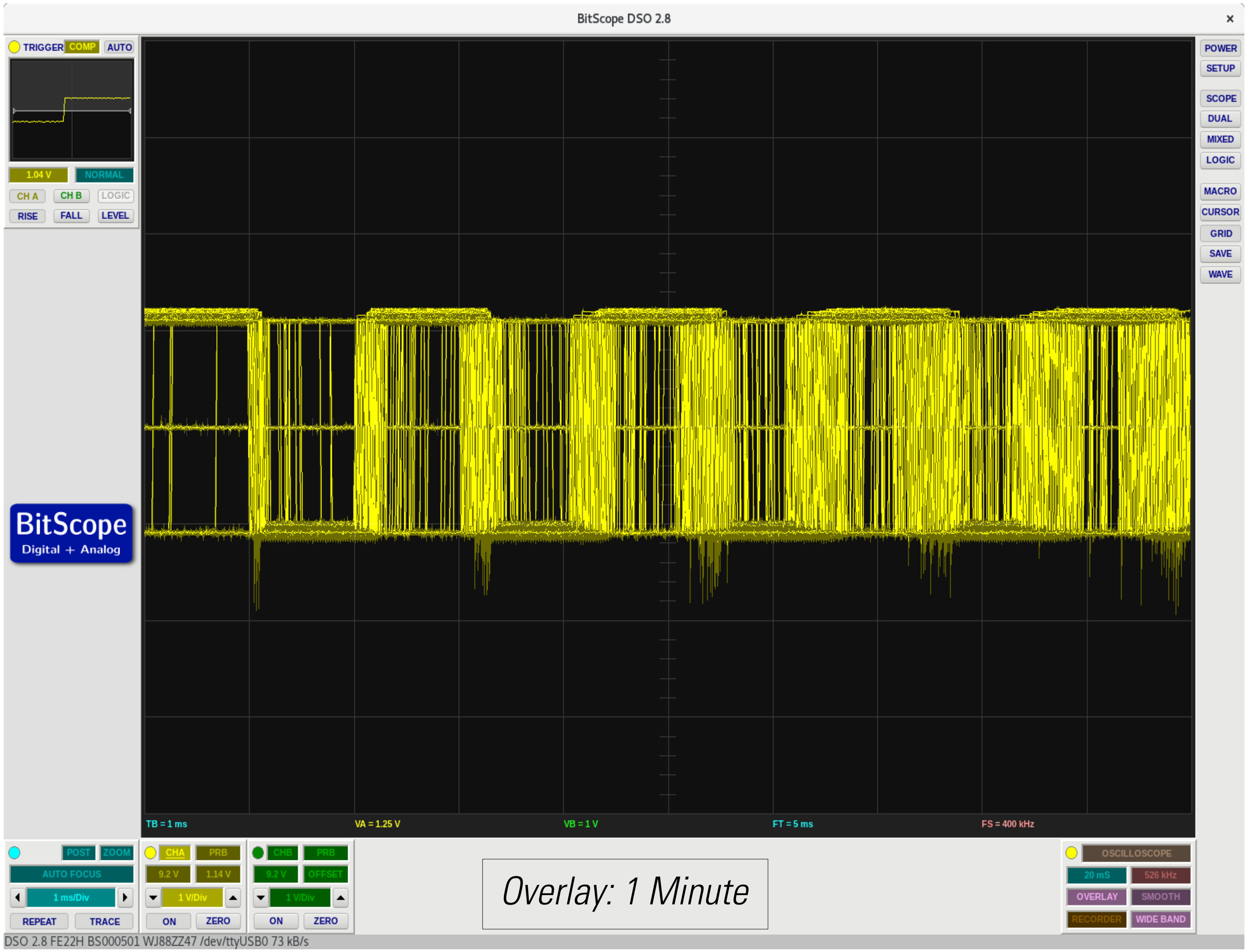
```
#define RECIPROCAL(a) (1.0/(a))
#define HALFCYCLE(a) ((a)/2.0)
#define NANOSECONDSPERSECOND 1000000000

duration = HALFCYCLE(RECIPROCAL(freq));
wholeseconds = (double) ((int) duration);
request.tv_sec = (int) wholeseconds;
request.tv_nsec = (int) ((duration - wholeseconds)
  * NANOSECONDSPERSECOND);
```

Generator

```
while (1) {
  value = "0";
  clock_nanosleep(CLOCK_MONOTONIC, 0, &request,
    &remain);
  value = "1";
  write(port, value, 1);
  clock_nanosleep(CLOCK_MONOTONIC, 0, &request,
    &remain);
}
```

taskset -c 1 squarewave

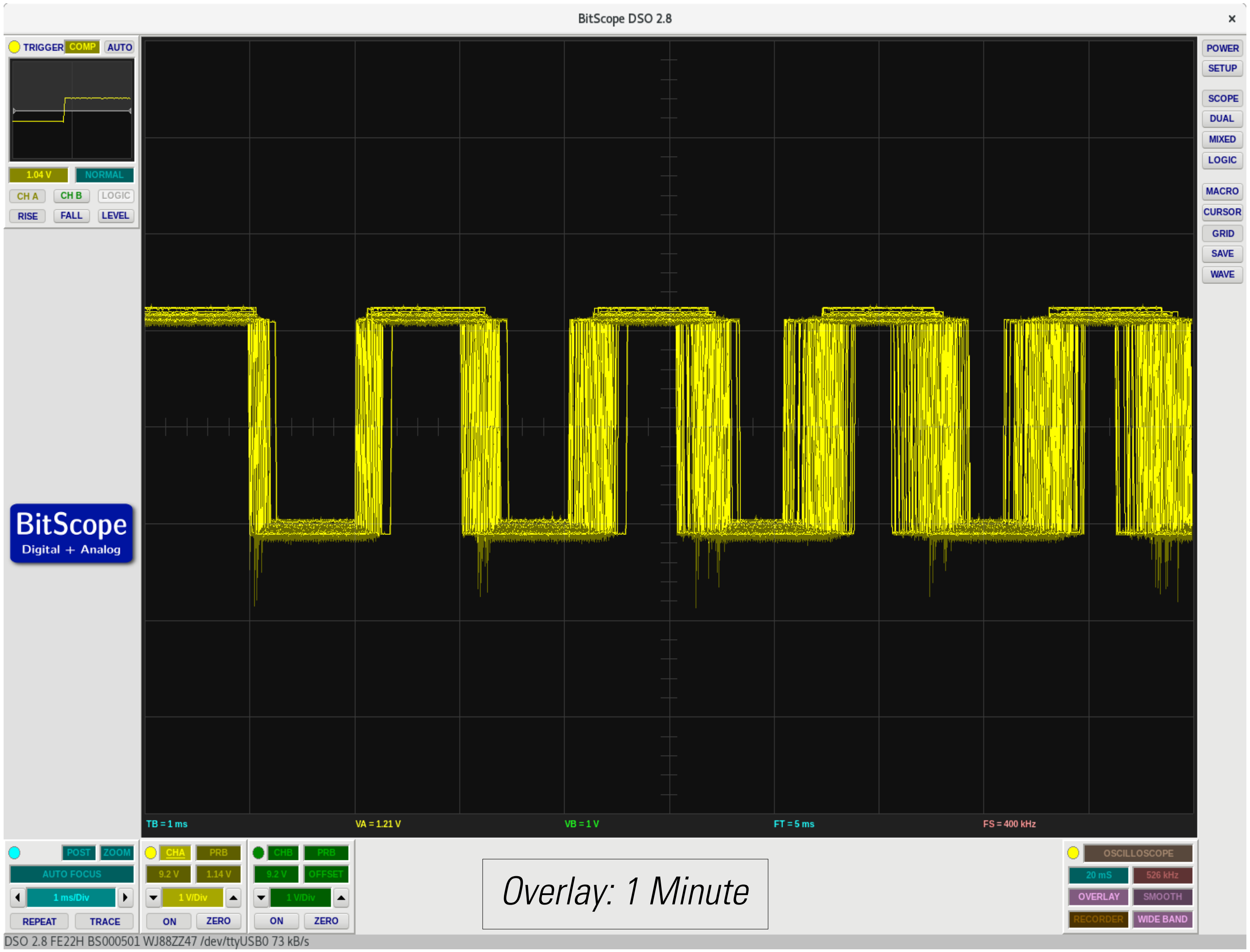
~~RT~~
C states



Concurrent process on same CPU core

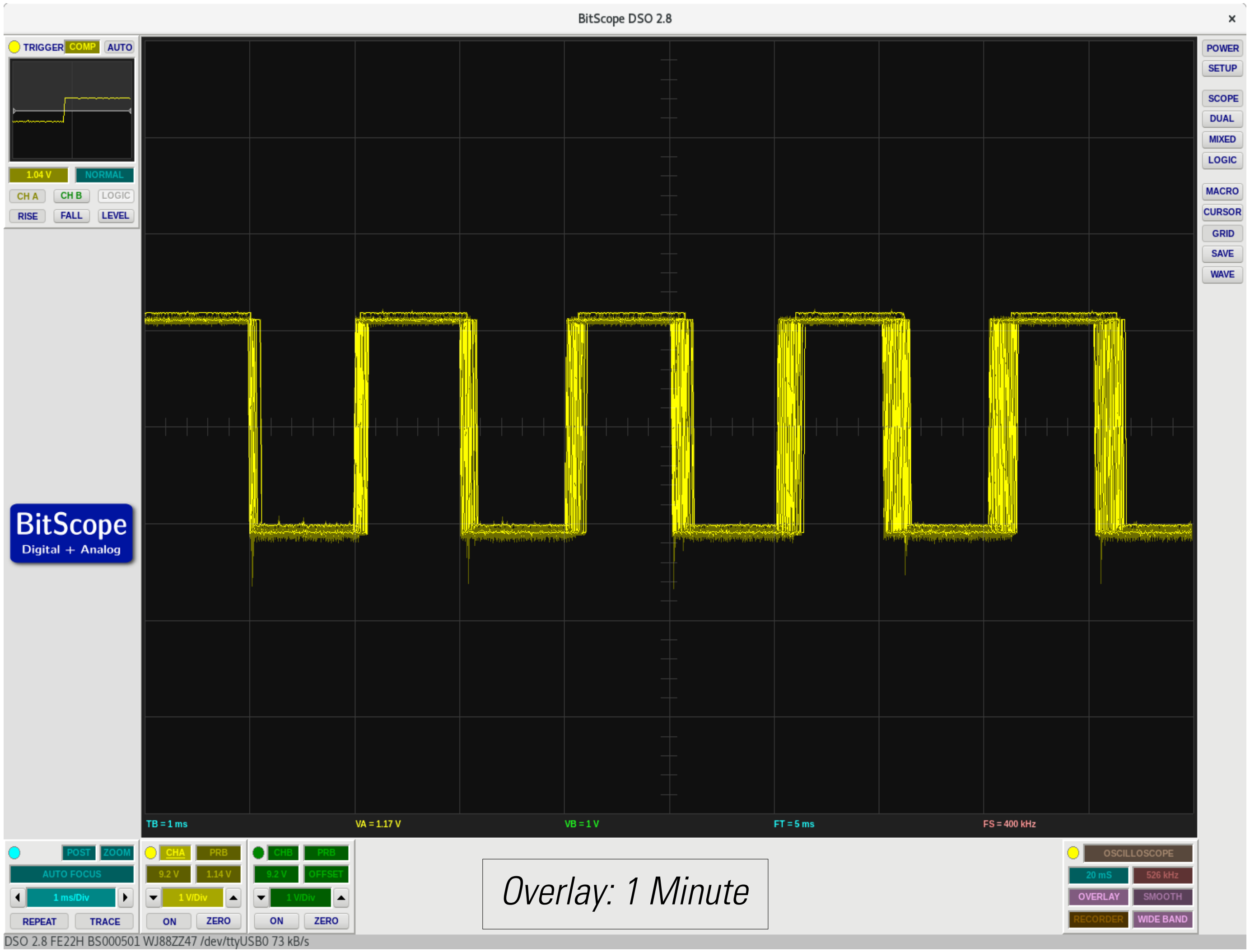
taskset -c 1 squarewave

~~RT~~
C states



taskset -c 1 chrt -f 90 squarewave

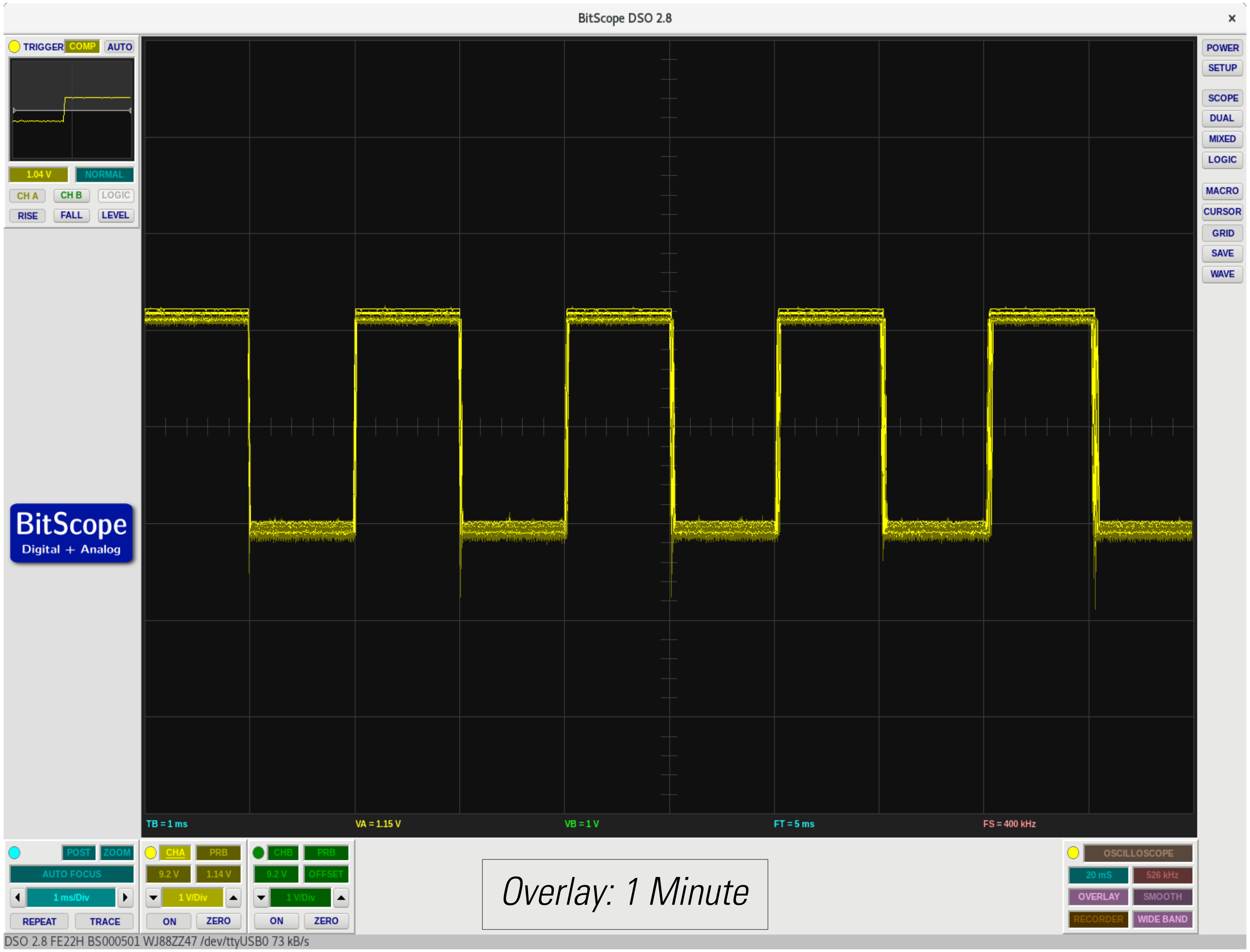
RT
C states



Overlay: 1 Minute

taskset -c 1 chrt -f 90 squarewave

RT
~~C states~~



Overlay: 1 Minute

Some information on today's sessions

- Please provide feedback on Legal HOT using the online form
  - Use the quick link **osadl.org/FB** (FeedBack), same as osadl.org/?id=3325
- You may ask questions at any time to be answered online, if possible
  - The quick link URL is **osadl.org/AQ** (AskQuestion), same as osadl.org/?id=3321
- You may join an online discussion on all topics of today at 4 pm
  - The quick link URL is **osadl.org/OD** (OnlineDiscussion), same as jitsi.osadl.org
  - Meeting name **OSADLTechnicalHOT**
  - Username and password will be displayed here after the last presentation