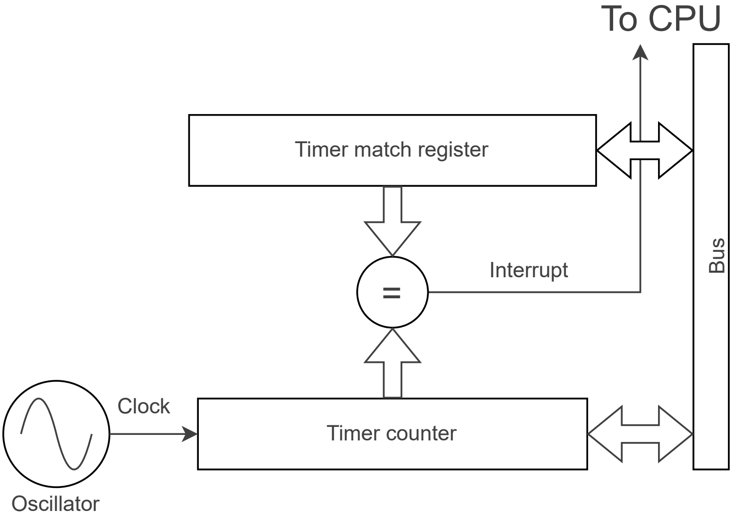
Efficient Abstraction of Clock Synchronization at the Operating System Level

Alessandro Sorrentino, Federico Terraneo, Alberto Leva

Politecnico di Milano, Italy

How does a real time (operating) system know time?

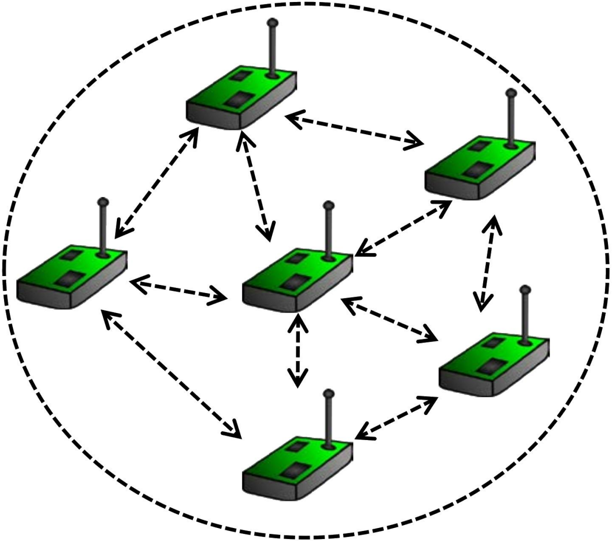
How does a real time (operating) system know time?



How does a real time (operating) system know time?

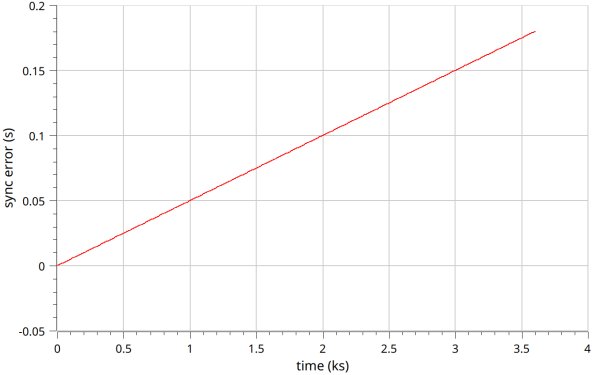
```
int getTime()
{
    return TIM_CNT;
}

void setNextInterrupt(int t)
{
    TIM_MATCH = t;
}
```

Enter distributed real-time systems



The effect of frequency error

Synchronization error between two oscillators that differ in frequency by 50ppm, or 0.005%.



$$e = \int_0^t \frac{\delta_s(\tau)}{f_0} d\tau$$

Fixing the problem

Many approaches to clock synchronization.

- Master-slave: all nodes synchronize to a time reference
- Consensus-based: nodes synchronize among each other
- ...

In this presentation we focus on the master-slave case.

Fixing the problem

We need three new actors

- A sensor, to measure our synchronization error
- An actuator, to correct our clock
- A clock synchronization algorithm

Basic clock synchronization (no skew compensation)

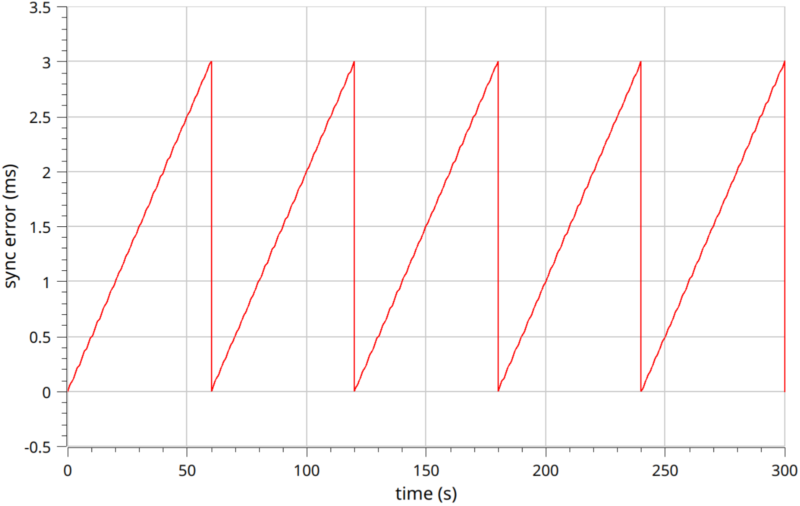
Basic clock synchronization

- The sensor: clock synchronization packets from the master
- The actuator: overwriting the timer counter
- The clock synchronization algorithm: TPSN, DMTS, ...

Basic clock synchronization (no skew compensation)

```
int getTime()
{
    return TIM_CNT;
}

void setNextInterrupt(int t)
{
    TIM_MATCH = t;
}

void onClockSyncPacket(int timestamp)
{
    TIM_CNT = syncAlgorithm(timestamp);
}
```

Basic clock synchronization (no skew compensation)



Advanced clock synchronization (with skew compensation)

Advanced clock synchronization

- The sensor: clock synchronization packets from the master
- The actuator: something that can change clock frequency ...
- The clock synchronization algorithm: FTSP, RBS, TATS, FLOPSYNC-2 ...

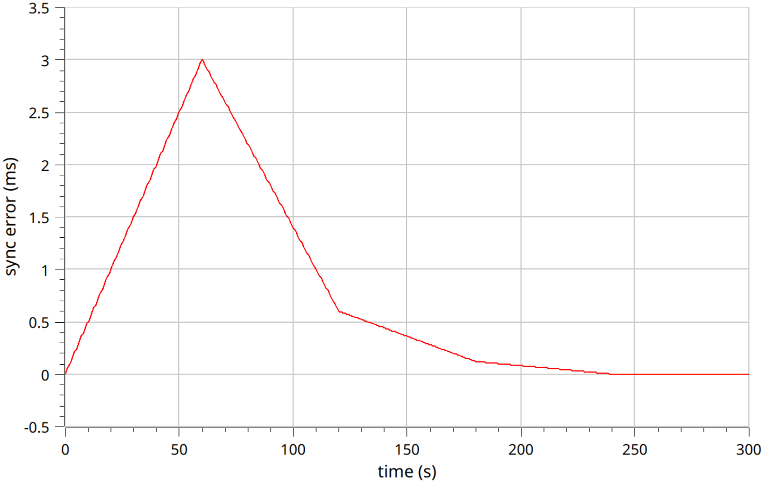
How to change the clock frequency

- hardware approach: possible, but expensive
- software approach: *virtual clock*

Advanced clock synchronization (with skew compensation)

```c
float a;
int b;

int getTime()
{
    return a * TIM_CNT + b; //Virtual clock
}

void setNextInterrupt(int t)
{
    TIM_MATCH = (t - b) / a;
}

void onClockSyncPacket(int timestamp)
{
    int error = computeError(tiemstamp);
    tie(a,b) = syncAlgorithm(error);
}
```

Advanced clock synchronization (with skew compensation)



A virtual clock makes it *possible* to have a continuous/monotonic clock, but a good sync algorithm is required.

Motivation of this work

When adding a virtual clock to your codebase, you have two times

- Corrected time (virtual clock time)
- Uncorrected time (HW clock time)

Motivation of this work

When adding a virtual clock to your codebase, you have two times

- Corrected time (virtual clock time)
- Uncorrected time (HW clock time)

From a software engineering perspective, uncorrected time is best encapsulated.

Motivation of this work

However, uncorrected timestamps are used by the *sensor* code that *measures* clock synchronization error.

Motivation of this work

However, uncorrected timestamps are used by the *sensor* code that *measures* clock synchronization error.

If the entire OS uses corrected time, the clock synchronization algorithms sees the error *corrected by the previous iteration of the algorithm itself*.

Motivation of this work

However, uncorrected timestamps are used by the *sensor* code that *measures* clock synchronization error.

If the entire OS uses corrected time, the clock synchronization algorithms sees the error *corrected by the previous iteration of the algorithm itself*.

Formally, the model of the clock synchronization sensor becomes *nonlinear*.

Proposed solution

Make a control algorithm that can deal with the introduced nonlinearity.

Proposed solution

Make a control algorithm that can deal with the introduced nonlinearity.

How? *feedback linearization*.

Feedback linearization in a nutshell

I have a nonlinear process $x(k+1) = f(x(k), u(k))$.

Feedback linearization in a nutshell

I have a nonlinear process $x(k+1) = f(x(k), u(k))$.

I would like to make it a linear process with a "virtual input" $v(k)$.
$x(k+1) = ax(k) + bv(k)$.

Feedback linearization in a nutshell

I have a nonlinear process $x(k+1) = f(x(k), u(k))$.

I would like to make it a linear process with a “virtual input” $v(k)$.
$x(k+1) = ax(k) + bv(k)$.

$f(x(k), u(k)) = ax(k) + bv(k)$

Feedback linearization in a nutshell

I have a nonlinear process $x(k+1) = f(x(k), u(k))$.

I would like to make it a linear process with a "virtual input" $v(k)$.
$x(k+1) = ax(k) + bv(k)$.

$f(x(k), u(k)) = ax(k) + bv(k)$

If I can solve for $u(k)$ I can make a controller so that the compound system with input $v(k)$ and output $x(k+1)$ is linear.

Feedback linearization in a nutshell

I have a nonlinear process $x(k+1) = f(x(k), u(k))$.

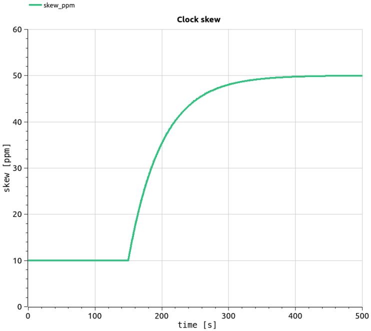
I would like to make it a linear process with a "virtual input" $v(k)$.
$x(k+1) = ax(k) + bv(k)$.

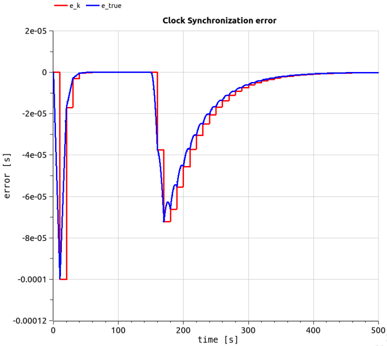
$f(x(k), u(k)) = ax(k) + bv(k)$

If I can solve for $u(k)$ I can make a controller so that the compound system with input $v(k)$ and output $x(k+1)$ is linear.

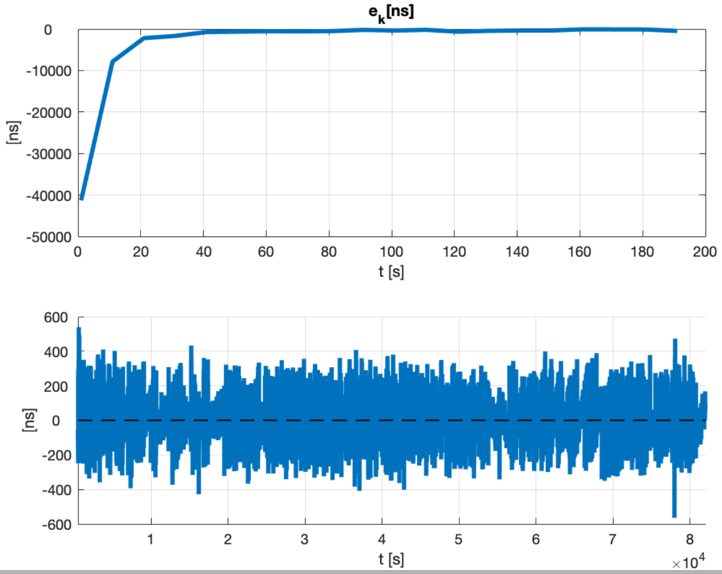
This approach has been applied producing the FLOPSYNC-3 controller (details in the paper).

Simulation results





Implementation results



That's all, but don't forget to...

- Read the paper!

Lots of interesting details, including how to *efficiently* implement a virtual clock.
https://re.public.polimi.it/handle/11311/1227829

Thanks for the attention, questions?