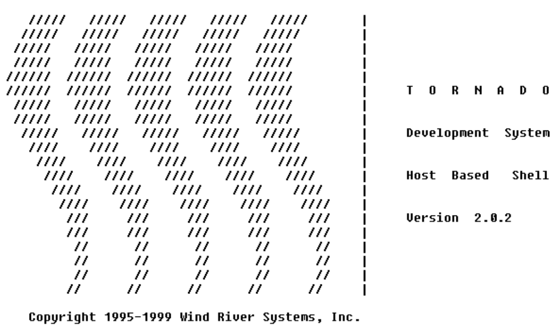
FYS 4220 / 9220 – 2012 / #7

Real Time and Embedded Data Systems and Computing

Clocks and time



T.B. Skaali, Department of Physics, University of Oslo

Real-Time clock and time facilities

- Interfacing to the passage of time of ”the real world” is through ”Clocks”
  - Absolute and relative time
  - Global time UTC
  - Delays
- Periodic activities
- Timers
- Watchdogs
- Connection to interrupt sources
- Timeouts
- Global timing and synchronization
- Sampling at high clock rates

C STANDARD LIBRARY

UNIVERSITY OF OSLO

asctime

Syntax:

```
#include <time.h>
char *asctime( const struct tm *ptr );
```

The function asctime() converts the time in the struct *ptr* to a character string of the following format:

```
day month date hours:minutes:seconds year\n\0
```

An example:

```
Mon Jun 26 12:03:53 2000
```

Related topics:
localtime(), gmtime(), time(), and ctime().

clock

Syntax:

```
#include <time.h>
clock_t clock( void );
```

The clock() function returns the processor time since the program started, or -1 if that information is unavailable. To convert the return value to seconds, divide it by CLOCKS_PER_SECOND. (Note: if your compiler is POSIX compliant, then CLOCKS_PER_SECOND is always defined as 1000000.)

Related topics:
time(), asctime(), and ctime().

Standard C Date & Time - 2 of 4

ctime

Syntax:

```
#include <time.h>
char *ctime( const time_t *time );
```

The ctime() function converts the calendar time *time* to local time of the format:

```
day month date hours:minutes:seconds year\n\0
```

using ctime() is equivalent to

```
asctime( localtime( tp ) );
```

Related topics:
localtime(), gmtime(), time(), and asctime().

difftime

Syntax:

```
#include <time.h>
double difftime( time_t time2, time_t time1 );
```

The function difftime() returns *time2-time1*, in seconds.

Related topics:
localtime(), gmtime(), time(), and asctime().

Standard C Date & Time - 3 of 4

gmtime

Syntax:

```
#include <time.h>
struct tm *gmtime( const time_t *time );
```

The gmtime() function returns the given *time* in Coordinated Universal Time (usually Greenwich mean time), unless it's not supported by the system, in which case NULL is returned. Warning!

Related topics:
localtime(), time(), and asctime().

localtime

Syntax:

```
#include <time.h>
struct tm *localtime( const time_t *time );
```

The function localtime() converts calendar time *time* into local time. Warning!

Watch out.

This function returns a variable that is statically located, and therefore overwritten each time this function is called. If you want to save the return value of this function, you should manually save it elsewhere.
Of course, when you save it elsewhere, you should make sure to actually copy the value(s) of this variable to another location. If the return value is a struct, you should make a new struct, then copy over the members of the struct.

Standard C Date & Time - 4 of 4

mktime

Syntax:

```
#include <time.h>
time_t mktime( struct tm *time );
```

The mktime() function converts the local time in *time* to calendar time, and returns it. If there is an error, -1 is returned.

Related topics:
time(), gmtime(), asctime(), and ctime().

---

strftime

Syntax:

```
#include <time.h>
size_t strftime( char *str, size_t maxsize, const char *fmt, struct tm *time );
```

The function strftime() formats date and time information from *time* to a format specified by *fmt*, then stores the result in *str* (up to *maxsize* characters). Certain codes may be used in *fmt* to specify different types of time:

---

time

Syntax:

```
#include <time.h>
time_t time( time_t *time );
```

The function time() returns the current time, or -1 if there is an error. If the argument *time* is given, then the current time is stored in *time*.

Related topics:
localtime(), gmtime(), strftime(), ctime(),

POSIX / VXWORKS - CLOCKS AND TIME

some lines of VxWorks ...\target\h\time.h – 1 of 2

```
/* time.h - POSIX time header */

/*
 * Copyright (c) 1992-2005 Wind River Systems, Inc.
 */

typedef int clockid_t;

#define CLOCKS_PER_SEC       sysClkRateGet()
#define CLOCK_REALTIME       0x0                    /* system wide realtime clock */
#define TIMER_ABSTIME        0x1                    /* absolute time */
#define TIMER_RELTIME   (~TIMER_ABSTIME)            /* relative time */

struct timespec
    {                        /* interval = tv_sec*10**9 + tv_nsec */
    time_t tv_sec;           /* seconds */
    long tv_nsec;            /* nanoseconds (0 - 1,000,000,000) */
    };
```

```
struct itimerspec
  {
   struct timespec it_interval;       /* timer period (reload value) */
   struct timespec it_value;          /* timer expiration */
   };

struct tm
   {
   int tm_sec;          /* seconds after the minute   - [0, 59] */
   int tm_min;          /* minutes after the hour     - [0, 59] */
   int tm_hour;         /* hours after midnight       - [0, 23] */
   int tm_mday;         /* day of the month           - [1, 31] */
   int tm_mon;          /* months since January       - [0, 11] */
   int tm_year;         /* years since 1900    */
   int tm_wday;         /* days since Sunday          - [0, 6] */
   int tm_yday;         /* days since January 1       - [0, 365] */
   int tm_isdst;        /* Daylight Saving Time flag */
   };

/* function declarations */
extern uint_t           _clocks_per_sec(void);
extern char *             asctime (const struct tm *_tptr);
extern clock_t            clock (void);
extern char *             ctime (const time_t *_cal);
extern double             difftime (time_t _t1, time_t _t0);
extern struct tm        * gmtime (const time_t *_tod);
extern struct tm        * localtime (const time_t *_tod);
```

Clocks, POSIX, time.h (1/3)

- The <*time.h*> header shall declare the structure **tm**, which shall include at least the following members:
  - int tm_sec Seconds [0,60].
  - int tm_min Minutes [0,59].
  - int tm_hour Hour [0,23].
  - int tm_mday Day of month [1,31].
  - int tm_mon Month of year [0,11].
  - int tm_year Years since 1900.
  - int tm_wday Day of week [0,6] (Sunday =0).
  - int tm_yday Day of year [0,365].
  - int tm_isdst Daylight Savings flag.

  The value of *tm_isdst* shall be positive if Daylight Savings Time is in effect, 0 if Daylight Savings Time is not in effect, and negative if the information is not available.

Clocks, POSIX, time.h (2/3)

- The <*time.h*> header shall define the following symbolic names:
- NULL
  - Null pointer constant.
- CLOCKS_PER_SEC
  - A number used to convert the value returned by the *clock*() function into seconds.
- CLOCK_PROCESS_CPUTIME_ID
  - [TMR|CPT]
    The identifier of the CPU-time clock associated with the process making a *clock*() or *timer**() function call.
- CLOCK_THREAD_CPUTIME_ID
  - [TMR|TCT]
    The identifier of the CPU-time clock associated with the thread making a *clock*() or *timer**() function cal

Clocks, POSIX, time.h (3/3)

- The <*time.h*> header shall declare the structure ***timespec***, which has at least the following members:
  - time_t tv_sec — Seconds.
  - long tv_nsec — Nanoseconds.
- The <*time.h*> header shall also declare the ***itimerspec*** structure, which has at least the following members:
  - struct timespec it_interval — Timer period.
  - struct timespec it_value — Timer expiration.
- The following manifest constants shall be defined:
  - CLOCK_REALTIME
    - The identifier of the system-wide real-time clock.
  - TIMER_ABSTIME
    - Flag indicating time is absolute. For functions taking timer objects, this refers to the clock associated with the timer. If one wants to work in relative time specify TIMER_RELTIME.
  - CLOCK_MONOTONIC
  - The identifier for the system-wide monotonic clock, which is defined as a clock whose value cannot be set via ***clock_settime*()** and which cannot have backward clock jumps. The maximum possible clock jump shall be implementation-defined

VxWorks POSIX *clockLib*

clockLib - clock library (POSIX)

ROUTINES
***clock_getres*( )** - get the clock resolution (POSIX)
***clock_setres*( )** - set the clock resolution
***clock_gettime*( )** - get the current time of the clock (POSIX)
***clock_settime*( )** - set the clock to a specified time (POSIX)

DESCRIPTION
This library provides a clock interface, as defined in the IEEE standard, POSIX 1003.1b. A clock is a software construct that keeps time in seconds and nanoseconds. The clock has a simple interface with three routines: ***clock_settime*( )**, ***clock_gettime*( )**, and ***clock_getres*( )**. The non-POSIX routine ***clock_setres*( )** is provided (temporarily) so that **clockLib** is informed if there are changes in the system clock rate (e.g., after a call to ***sysClkRateSet*( )**). Times used in these routines are stored in the timespec structure:

```
struct timespec
{
time_t tv_sec;          /* seconds */
long tv_nsec;           /* nanoseconds (0 -1,000,000,000) */
};
```

IMPLEMENTATION
Only one *clock_id* is supported, the required **CLOCK_REALTIME**.

clockLib facilities - example

```
void     ClockResolution()
{
         int                 status;
         clockid_t           clock_id = CLOCK_REALTIME;
         struct timespec     res;
         char * array[2] = {"OK","ERROR"};

         res.tv_sec = 0;
         res.tv_nsec = 0;

         status = clock_getres (clock_id, &res);
         if (status == ERROR) status = 1;
         printf("clock_getres status = %s\n", array[status]);
         printf("clock resolution = %d nsec = %f msec\n",
                  (int)res.tv_nsec, (float)(float)res.tv_nsec/1000000);
         printCR;
}
```

Run the code:

```
-> ClockResolution
clock_getres status = OK
clock resolution = 16666666 nsec = 16.666666 msec
```

TIME DELAYS AND PERIODIC PROCESSES

Delays (VxWorks)

taskDelay( ) - delay a task from executing

SYNOPSIS

STATUS taskDelay (int ticks /* number of ticks to delay task */)

DESCRIPTION

This routine causes the calling task to relinquish the CPU for the duration specified (in ticks). This is commonly referred to as manual rescheduling, but it is also useful when waiting for some external condition that does not have an interrupt associated with it.

Note! If the calling task receives a signal that is not being blocked or ignored, ***taskDelay*( )** <u>immediately</u> returns ERROR and sets **errno** to EINTR after the signal handler is run.

RETURNS

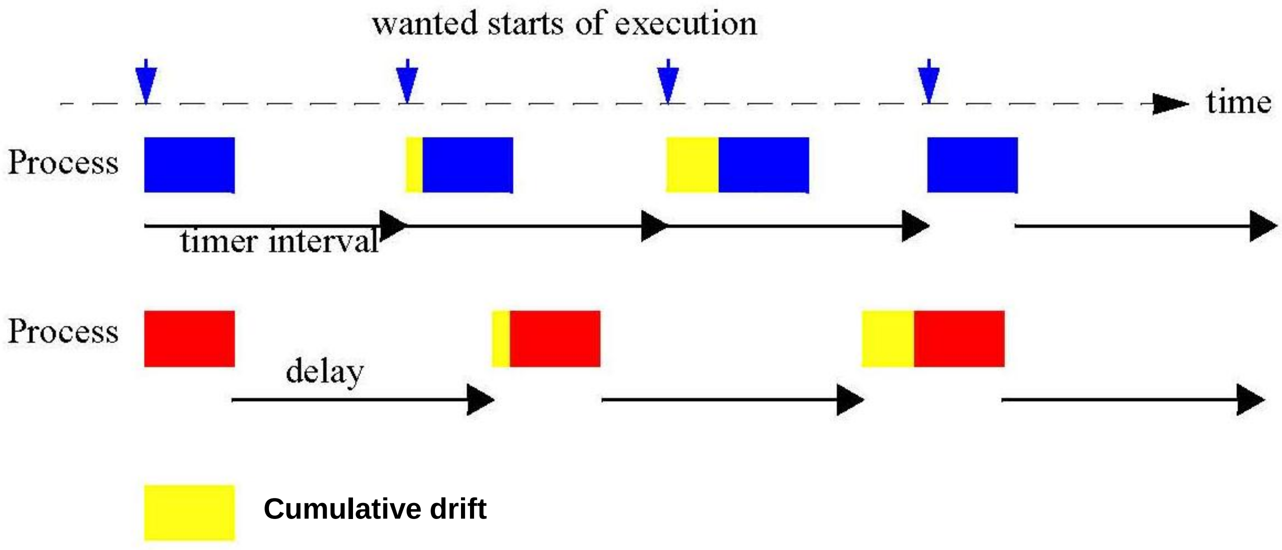
OK, or ERROR if called from interrupt level or if the calling task receives a signal that is not blocked or ignored.

Periodic activities

- Measurements of physical values are often periodic, for instance logging of temperature, pressure etc.
- A periodic process with periods from millisec and upwards can be implemented using the taskDelay() system call. However, this implementation suffers from three sideeffects:
  - If the processing time varies between each taskDelay() call, the period time will vary;
  - A signal will cause an immediate return;
  - Since the timing reference is relative, a cumulative time drift may occur, see next page
- Sampling of rapidly varying signals, say CD audio at 44.1 kHz, can not be implemented using standard periodic processes.

Time drift

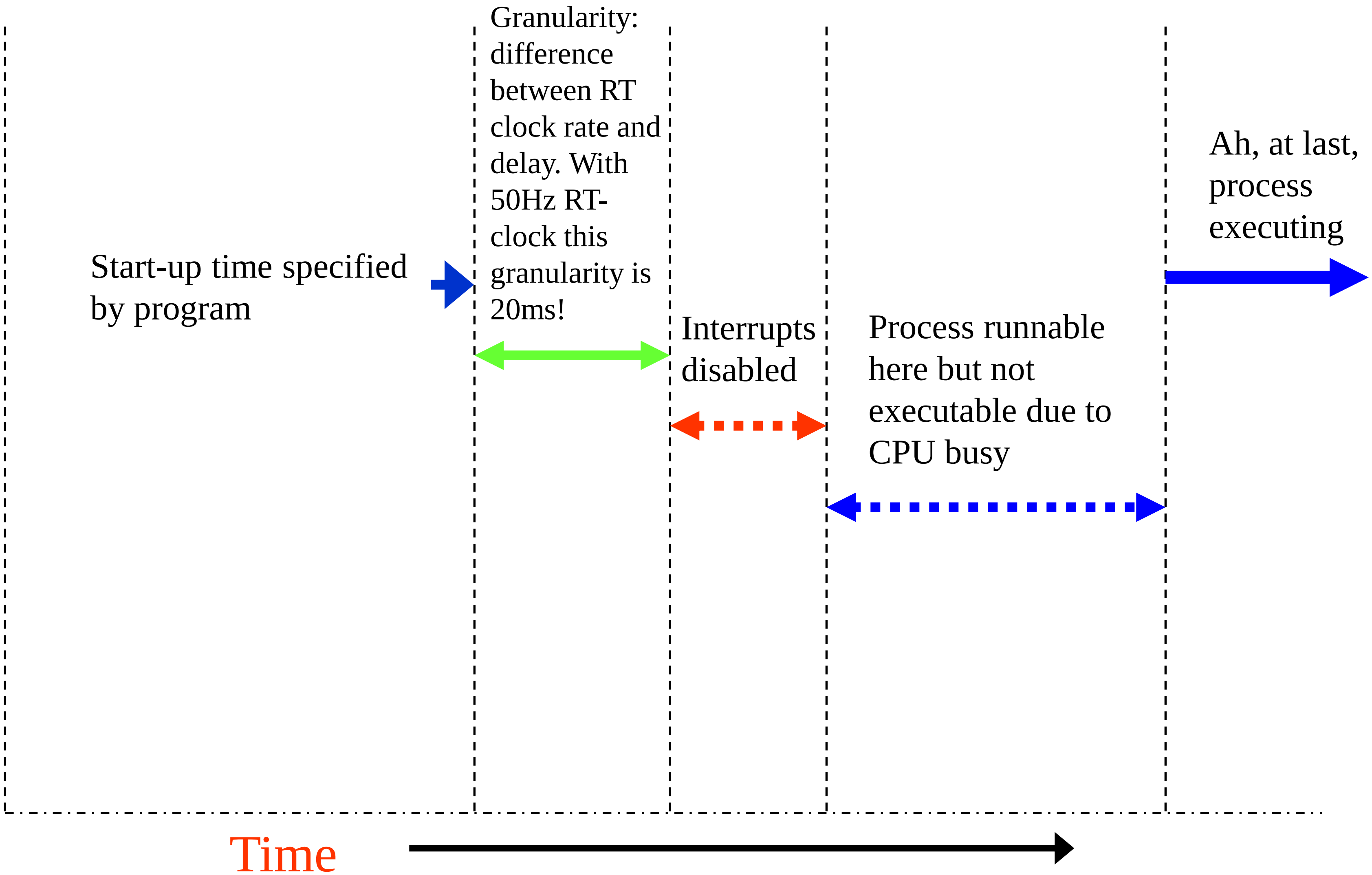
- Cumulative time drift must be avoided when a process is executed periodically
- Using VxWorks taskDelay() instead of a timer will result in a cumulative drift!



Timing accuracy and time jitter

- Several factors will influence the accuracy of timing instants based on the RT-clock, they are illustrated on the next page;
- In particular, disabling the interrupt system or task switching for a prolonged period (which is possible under VxWorks) can have a very detrimental effect on the performance of a Real-Time system;
- The correct method for running periodic activities is to implement them as **POSIX timers.**

Time reference and time jitter



POSIX timers (VxWorks)

- The POSIX standard provides for identifying multiple virtual clocks, but only one clock is required--the system-wide real-time clock, identified in the clock and timer routines as **CLOCK_REALTIME**. VxWorks provides routines to access the system-wide real-time clock; see the reference entry for **clockLib**. (No virtual clocks are supported in VxWorks.)
- The POSIX timer facility provides routines for tasks to signal themselves at some time in the future. Routines are provided to create, set, connect and delete a timer; see the reference entry for **timerLib**. When a timer goes off, the default signal (**SIGALRM**) is sent to the task. ***sigaction*( )** can be used to install a signal handler that executes when the timer expires. Alternatively, **timer_connect()** can be used.
- **Note! A timer should be programmed with the same restrictions as for signal handlers, no waiting!**

timerLib - timer library (POSIX)

ROUTINES

***timer_cancel* ( ) -**	cancel a timer
***timer_connect* ( ) -**	connect a user routine to the timer signal
***timer_create* ( ) -**	allocate a timer using the specified clock for a timing base (POSIX)
***timer_delete* ( ) -**	remove a previously created timer (POSIX)
***timer_gettime* ( ) -**	get the remaining time before expiration and the reload value (POSIX)
***timer_getoverrun* ( ) -**	return the timer expiration overrun (POSIX)
***timer_settime* ( ) -**	set the time until the next expiration and arm timer (POSIX)
***nanosleep* ( ) -**	suspend the current task until the time interval elapses (POSIX)

DESCRIPTION

This library provides a timer interface, as defined in the IEEE standard, POSIX 1003.1b.
Timers are mechanisms by which tasks signal themselves after a designated interval.
Timers are built on top of the clock and signal facilities. The clock facility provides an absolute time-base.
Standard timer functions simply consist of creation, deletion and setting of a timer.
When a timer expires, ***sigaction* ( )** (see **sigLib**) must be in place in order for the user to handle the event.
The "high resolution sleep" facility, ***nanosleep* ( )**, allows sub-second sleeping to the resolution of the clock.
The **clockLib** library should be installed and ***clock_settime* ( )** set before the use of any timer routines.

Timer code: see demo program next page. It is also referred to RTlab_no2 2012

Using a timer to run *timerhandle()* periodically

```
/* POSIX timers */

#include "vxWorks.h"
#include "time.h"
#include "timexLib.h"
#include "taskLib.h"
#include "sysLib.h"
#include "stdio.h"

#define         TIMER_START        10
#define         TIMER_INTERVAL  5

/* timer is connected to timerhandle() */
void timerhandle(timer_t timerID, int targ)
{
   int i;
   printf("timerhandle invoked with targ = %d\n", targ);
   /* some CPU eating stuff */
   for (i = 0; i < 200000; i++) {};
}

/* run the demo from here */
int execTimer (void)
{
   timer_t timerID;
   struct itimerspec value, ovalue, gvalue;
   int t_arg = 12321;
   int i;

   if (timer_create (CLOCK_REALTIME, NULL, &timerID) == ERROR)
      {
      printf ("create FAILED\n");
      return (ERROR);
      }
   if (timer_connect (timerID, (VOIDFUNCPTR)timerhandle, t_arg) == ERROR)
      {
      printf ("connect FAILED\n");
      return (ERROR);
      }
   value.it_value.tv_nsec = 0;
   value.it_value.tv_sec = TIMER_START;
   value.it_interval.tv_nsec = 0;
   value.it_interval.tv_sec = TIMER_INTERVAL;
                  printf("timer set up for start after %ld sec and interval %ld sec\n",
                            value.it_value.tv_sec, value.it_interval.tv_sec);
   if (timer_settime (timerID, TIMER_RELTIME, &value, &ovalue) == ERROR)
      {
      printf ("timer_settime FAILED\n");
      return (errno);
      }
   /* some diagnostics during 25 sec */
   for (i = 0; i < 25; i++) {
    if (timer_gettime (timerID, &gvalue) == ERROR)
      {
      printf ("gettime FAILED\n");
      return (errno);
      }
    printf("gvalue.it_value.tv_sec = %ld\n", gvalue.it_value.tv_sec);
    printf("gvalue.it_interval.tv_sec = %ld\n", gvalue.it_interval.tv_sec);
    taskDelay (CLOCKS_PER_SEC);
   }

   if (timer_cancel (timerID) == ERROR)
      {
      printf ("cancel FAILED\n");
      return (errno);
      }
   if (timer_delete (timerID) == ERROR)
      {
      printf ("delete FAILED\n");
      return (errno);
      }
   return (OK);
}
```

vxsim0@fyspc-elg045 - Host Shell

```
->
-> execTimer
timer set up for start after 10 sec and interval 5 sec
gvalue.it_value.tv_sec = 10
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 9
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 8
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 7
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 6
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 5
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 4
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 3
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 2
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 1
gvalue.it_interval.tv_sec = 5
timerhandle invoked with targ = 12321
gvalue.it_value.tv_sec = 5
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 4
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 3
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 2
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 1
gvalue.it_interval.tv_sec = 5
timerhandle invoked with targ = 12321
gvalue.it_value.tv_sec = 5
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 4
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 3
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 2
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 1
gvalue.it_interval.tv_sec = 5
timerhandle invoked with targ = 12321
gvalue.it_value.tv_sec = 5
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 4
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 3
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 2
gvalue.it_interval.tv_sec = 5
gvalue.it_value.tv_sec = 1
gvalue.it_interval.tv_sec = 5
timerhandle invoked with targ = 12321
value = 0 = 0x0
->
```

Watchdogs (voff-voff)

- "Watchdog" stands for a periodic activation of a process which gives a signal, for instance by lighting up a lamp, to show that a system is alive and operates correctly
- VxWorks includes a watchdog-timer mechanism that allows any C function to be connected to a specified time delay.
  - Watchdog timers are maintained as part of the system clock ISR. Normally, functions invoked by watchdog timers execute as interrupt service code at the interrupt level of the system clock. However, if the kernel is unable to execute the function immediately for any reason (such as a previous interrupt or kernel state), the function is placed on the **tExcTask** work queue. Functions on the **tExcTask** work queue execute at the priority level of the **tExcTask** (usually 0). Restrictions on ISRs apply to routines connected to watchdog timers
  - **Note! Watchdogs can only be used in the kernel, i.e. the type of modules developed for the lab exercises;**
  - Demo: see code next page

Watchdog example


```c
/* This example creates a watchdog timer and sets it to go off in 3 seconds. */

#include "vxWorks.h"
#include "sysLib.h"
#include "logLib.h"
#include "wdLib.h"
#include "taskLib.h"
#include "tickLib.h"

#define  SECONDS (3)

WDOG_ID myWatchDogId;

int task (void)
{
    /* Create watchdog */

    if ((myWatchDogId = wdCreate( )) == NULL)
        return (ERROR);

    /* Set timer to go off in SECONDS - printing a message to stdout */

    if (wdStart (myWatchDogId,
                 sysClkRateGet( ) * SECONDS,
                 (FUNCPTR)logMsg,
                 "Watchdog timer just expired\n") == ERROR)
        return (ERROR);

    taskDelay (sysClkRateGet( ) * 10);
    return (OK);
}
```

Wind River Systems | NUM | Ln 2, Col 2

Output on console terminal: –> interrupt: Watchdog timer just expired

CONNECTING TO CLOCK INTERRUPTS

VxWorks Kernel Programming – *intConnect* and *sysAuxClkConnect*

- VxWorks provides the routine intConnect( ), which allows C functions to be connected to any interrupt.
  - The arguments to this routine are the byte offset of the interrupt vector to connect to, the address of the C function to be connected, and an argument to pass to the function. When an interrupt occurs with a vector established in this way, the connected C function is called at interrupt level with the specified argument. When the interrupt handling is finished, the connected function returns. A routine connected to an interrupt in this way is called an ***interrupt service routine*** (ISR).
- A similiar facility for connecting a routine to the auxiliary clock interrupt is sysAuxClkConnect( )
  - The routine specifies the ISR to be called at each Auxiliary Clock interrupt
  - Auxiliary clock routines are BSP-specific, i.e. availability and functionality depends on the target package
  - sysAuxClkConnect( ) is very useful for sampling at higher rate than a process/task can support
  - sysClkConnect( ) is a similiar facility for the system clock interrupt

sysLib – provides board specific routines

sysLib routines for sysClk and AuxClk:

sysClkConnect( ) - connect a routine to the system clock interrupt

sysClkDisable( ) - turn off system clock interrupts

sysClkEnable( ) - turn on system clock interrupts

sysClkRateGet( ) - get the system clock rate

sysClkRateSet( ) - set the system clock rate

sysAuxClkConnect( ) - connect a routine to the auxiliary clock interrupt

sysAuxClkDisable( ) - turn off auxiliary clock interrupts

sysAuxClkEnable( ) - turn on auxiliary clock interrupts

sysAuxClkRateGet( ) - get the auxiliary clock rate

sysAuxClkRateSet( ) - set the auxiliary clock rate

sysAuxClkConnect( ) code from RTlab_no2

```
sysAuxClkEnable();
if (sysAuxClkRateSet(FUELCONTROL_SR) == ERROR) {
    logMsg ("->Can't set sampling rate to requested Hz\n", FUELCONTROL_SR,0,0,0,0,0);
    logMsg ("->Actual sampling rate = %d Hz\n",sysAuxClkRateGet(),0,0,0,0,0);
}
if (sysAuxClkConnect((FUNCPTR)FuelController, 0) != OK)
        logMsg("FuelController sysAuxClkConnect error %d\n", errno,0,0,0,0,0);
```

```
/* =====  User defined Interrupt Service Routines ===== */
/* FuelController task connected to sysAuxClkConnect , max sampling freq = 60Hz */
#define FUELCONTROL_SR       60
float   rpm2fuel = 0.0000167;           /* Company Confidential!! */

void    FuelController ()
{
    float fuel_rate_period = 1.0/(float)FUELCONTROL_SR;
    Porsche.fuel_consumption = rpm2fuel*Porsche.engine_rpm;
    Porsche.fuel_level_liter = Porsche.fuel_level_liter -
            Porsche.fuel_consumption * (fuel_rate_period * Porsche.speed_km_per_hours/3600);
}
```

Note! It seems that the rate of vxsim AuxClk can be set to all integer values from 2 (but not 1!) and up to 1000 Hz.

UNIVERSITY OF OSLO

TIMING FAILURES AND TIMEOUTS

Timing failures

- Detection of timing failures?
  - Overrun of deadline
  - Overrun of worst-case execution time
  - Timeouts
- And what could be the consequences?
  - Hard Real-Time: potentially disastrous
  - Soft Real-Time: can be accepted from time to another, provided that the overrun is not too large and does not occur too often (whatever that means)
- **POSIX (not VxWorks)**
  - Two clocks are defined: `CLOCK_PROCESS_CPUTIME_ID` and `CLOCK_THREAD_CPUTIME_ID`
  - These can be used in the same way as `CLOCK_REALTIME`
  - Each process/thread has an associated execution-time clock; calls to:

```
clock_settime(CLOCK_PROCESS_CPUTIME_ID, &some_timespec_value);
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &some_timespec_value);
clock_getres(CLOCK_PROCESS_CPUTIME_ID, &some_timespec_value)
```

  - will set/get the execution-time or get the resolution of the execution time clock associated with the calling process (similarly for threads)
- **But, how to detect timing failures?**

Timeouts

- Some of the API routines for VxWorks take a timeout parameter. Timeouts are important for ensuring that a process/task will not waiting forever. Examples are:
  - Wind message queue routines msqQSend( ) and msgQReceive(), but not the corresponding POSIX message queue routines!
  - Wind semTake(), but not POSIX sem_wait()
  - Networks timeouts
  - Wind timer objects
- It is also possible, but tricky, to have a process checking that other processes are not stuck in a pending state due to an unexpected situation

A GLOBAL PICTURE

Global timing and synchronization

- In wide area or global data acquisiton system one needs access to a global clock if the registration of data must be time synchronized. The GPS system gives a very high accuracy in the nsec domain
  - To obtain this accuracy, the GPS signals are corrected for relativistic effects
  - However, not all regions on Earth are well covered by GPS
- IEEE 1588 Precision Time Protocol (PTP) [2002, 2008] is a protocol used to synchronize clocks throughout a computer network. On a local area network it achieves clock accuracy in the sub-microsecond range, making it suitable for measurement and control systems.
  - IEEE 1588 is designed to fill a niche not well served by either of the two dominant protocols, NTP and GPS. IEEE 1588 is designed for local systems requiring accuracies beyond those attainable using NTP. It is also designed for applications that cannot bear the cost of a GPS receiver at each node, or for which GPS signals are inaccessible.
  - The Network Time Protocol (NTP) [1985] is a protocol for synchronizing the clocks of computer systems over packet-switched, variable-latency data networks

SAMPLING OF DATA AT HIGH CLOCK RATES

Requires special hardware, an example is presented on following pages.

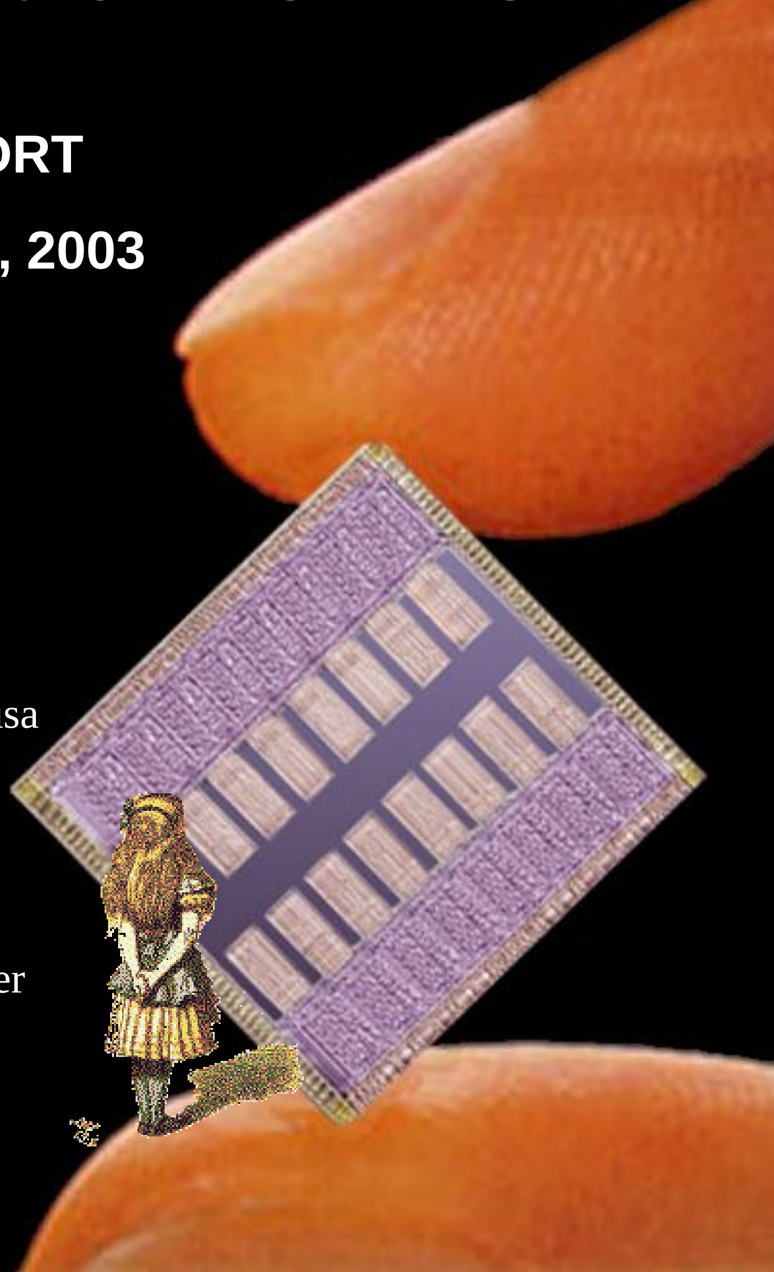
Front End Electronics for the ALICE TPC

PROGRESS REPORT

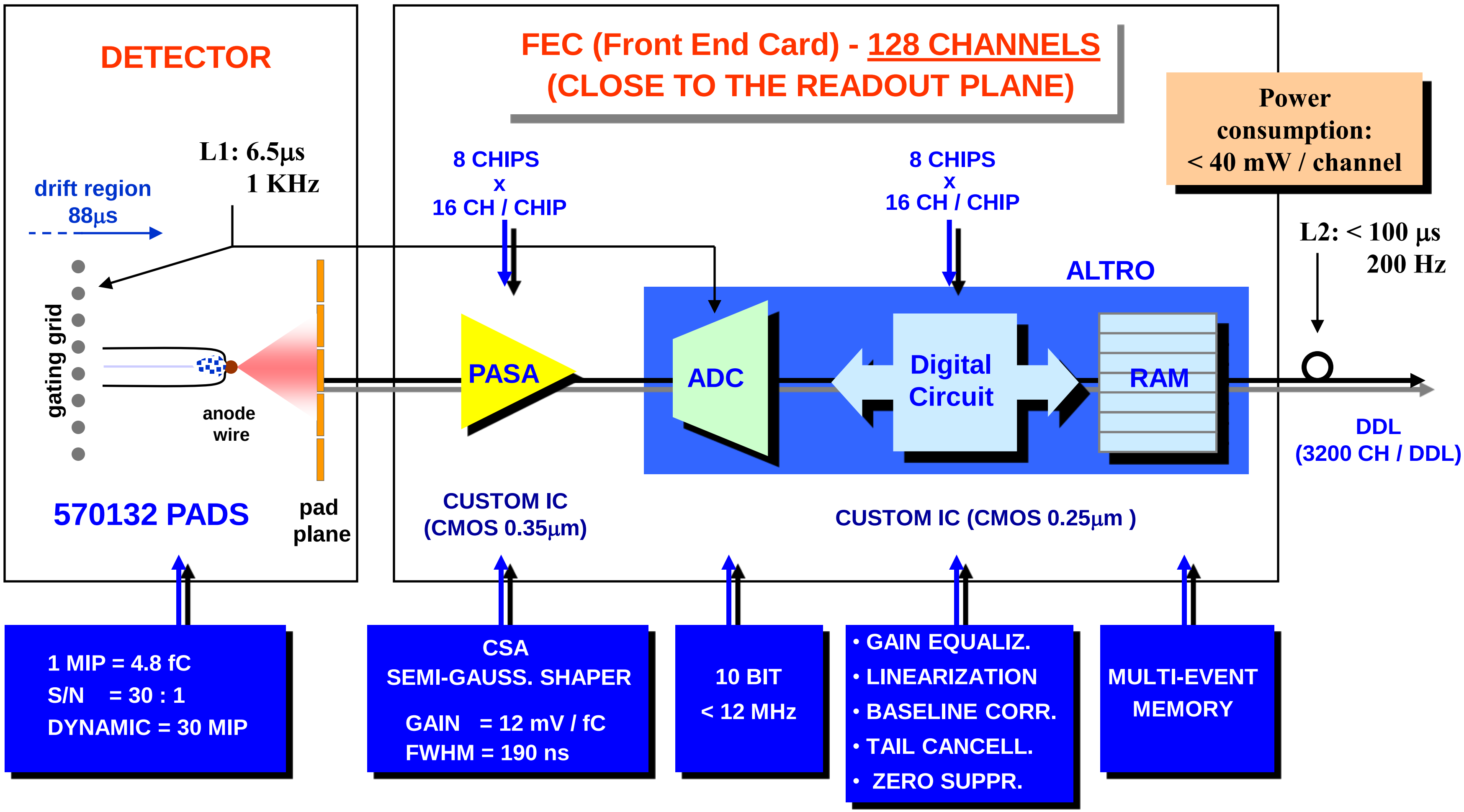
CERN, November 10, 2003

TPC FEE Collaboration

Bergen:	H. Helstrup, J. Lien, A.S. Martinsen, D. Roherich, K. Roed, K. Ullaland,
CERN:	R.Bramm, R.Campagnolo, C.Engster, C.Gonzalez, A.Junique, B. Mota, L. Musa
Darmstadt TU:	U. Bonnes, S. Lange, H. Oeschler
Frankfurt:	R. Renfordt, G. Ruschman, N. Bialas
Heidelberg:	V. Lindenstruth, H.K. Soltveit, H. Tilsner
Lund:	H.-A. Gustafsson, L. Ostermann
Oslo:	B. Skaali, J. Wikne, D. Wormald



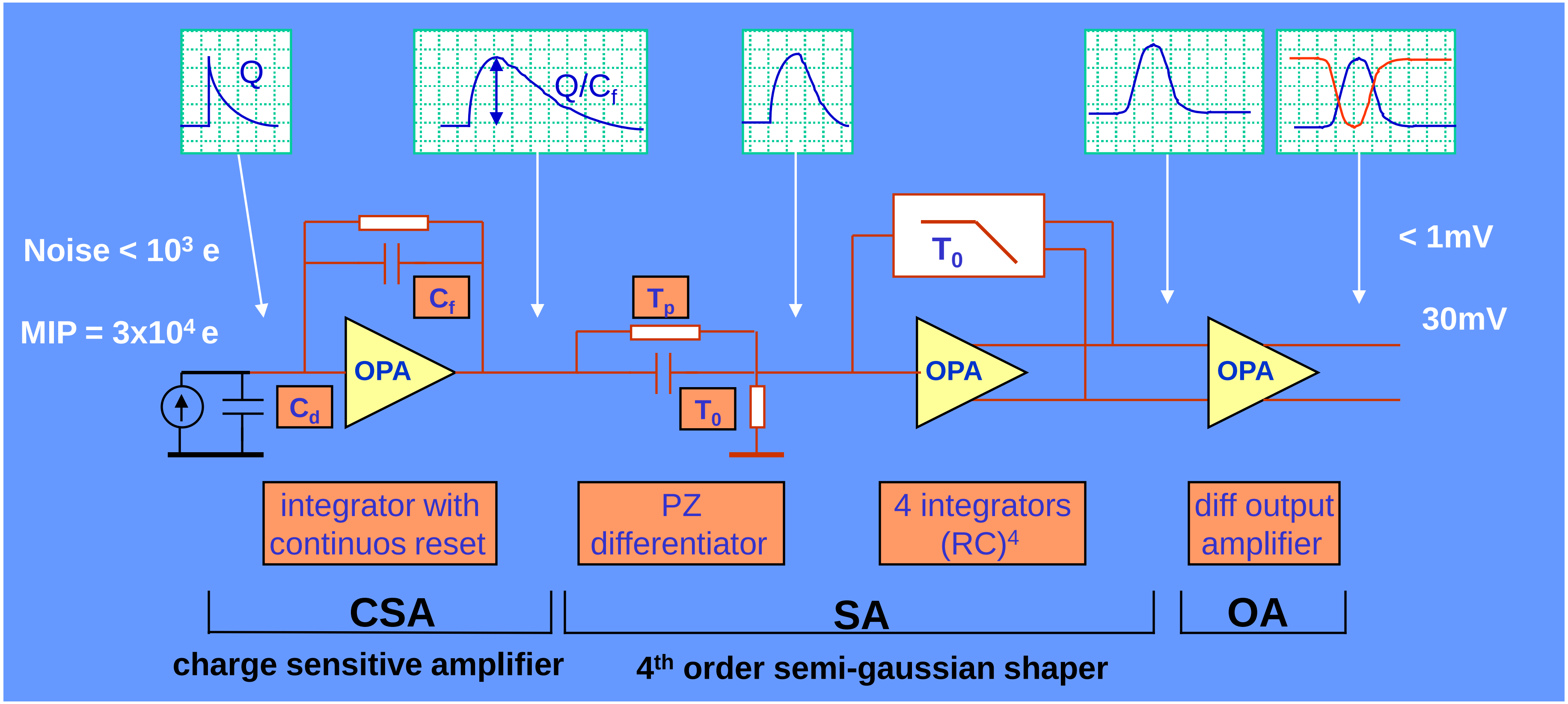
Architecture and Main Components



FEE FOR THE NA49 AND STAR TPCs

- analog memory in front of the ADC ⇒ readout time independent of the occupancy
- no zero suppression in the FEE ⇒ high data throughput on the detector data links

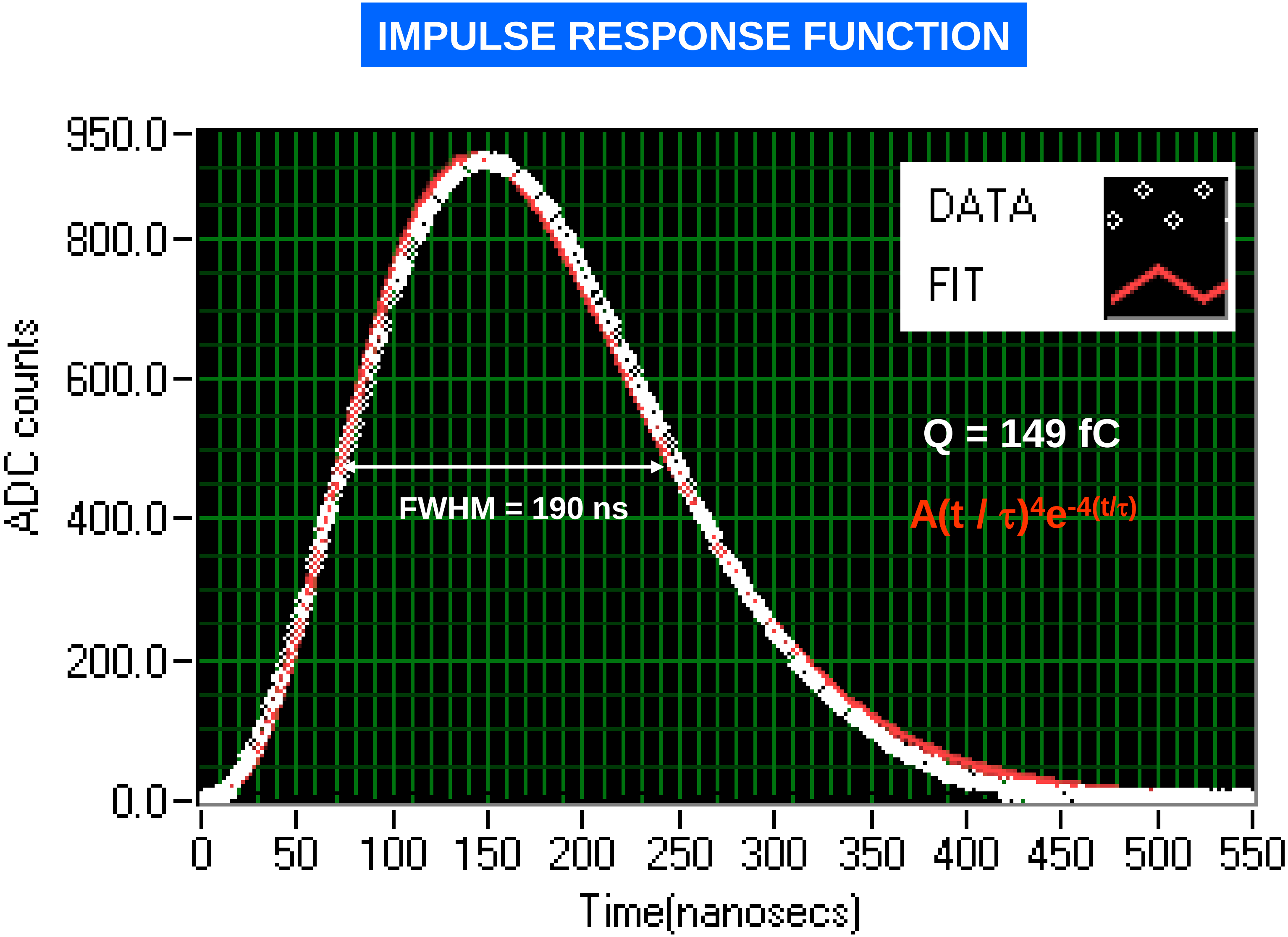
Pre-Amplifier Shaping Amplifier (PASA)



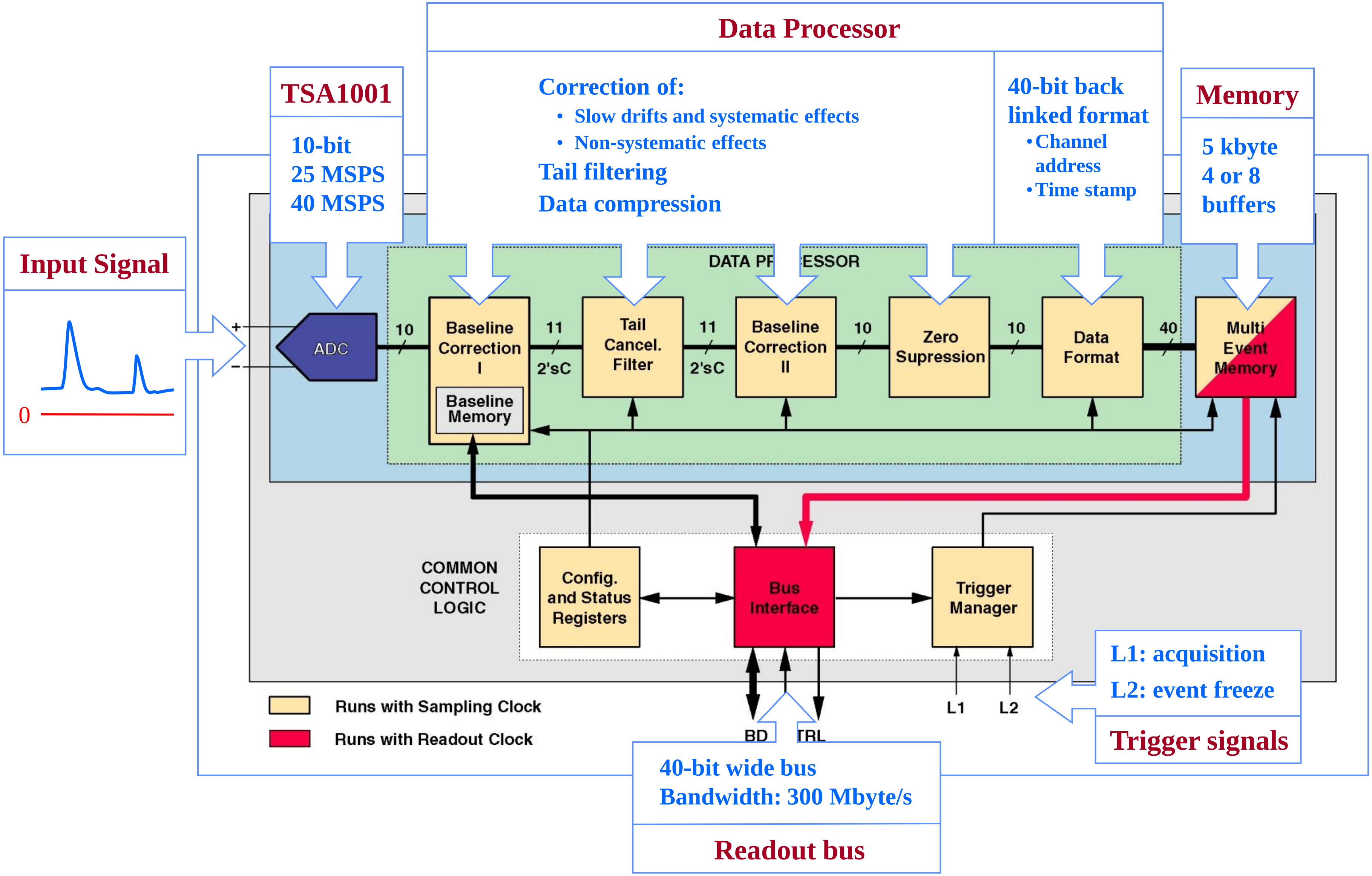
REQUIREMENTS

Gain:	12mV / fC	INL:	< 0.3%
FWHM:	190ns	Crosstalk:	< 0.1%
Noise:	< 1000	Power:	< 20mW / ch

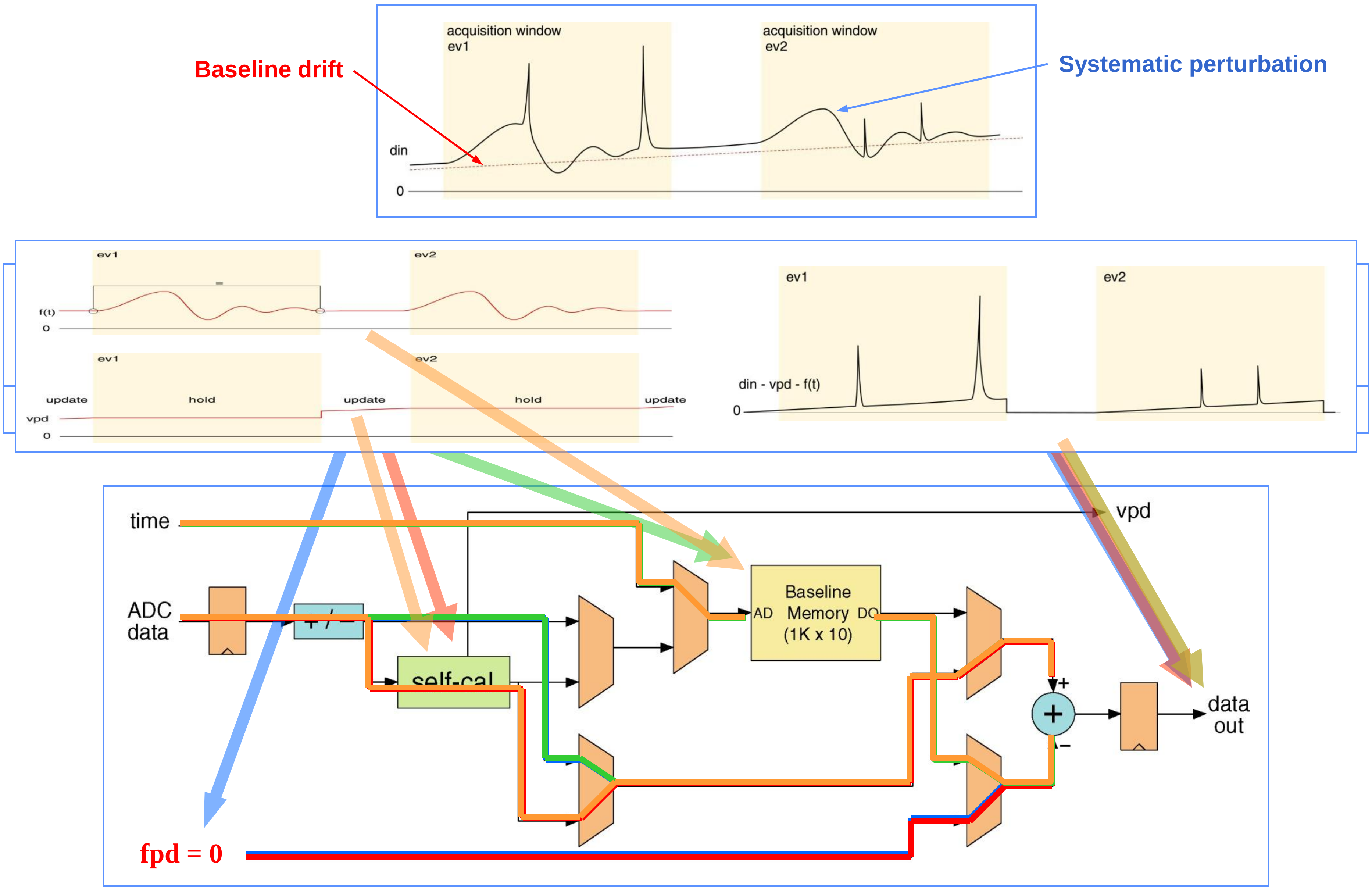
Pre-Amplifier Shaping Amplifier (PASA)



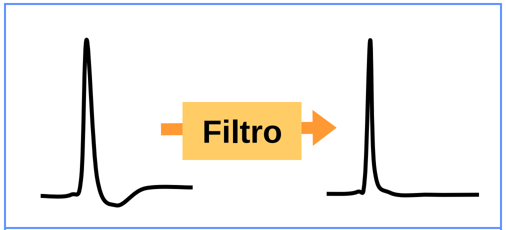
ALTRO Block Diagram



Baseline Correction 1

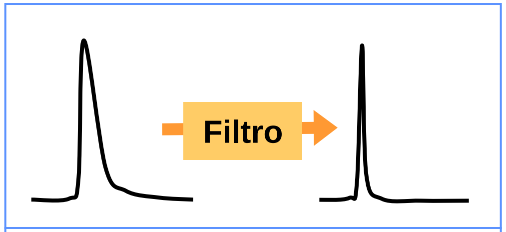


- **Functions**
  - **signal (ion) tail suppression**
  - **pulse narrowing ⇒ improves cluster separation**
  - **gain equalization**

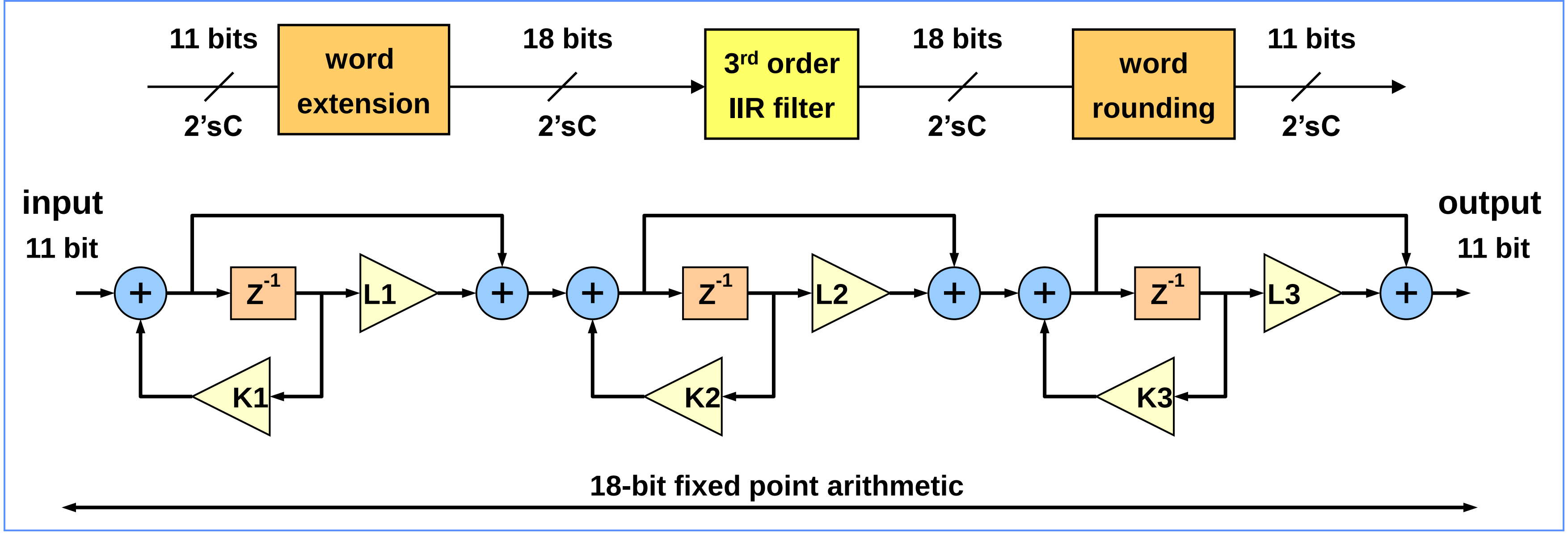


compensates undershoot

- **Architecture**
  - **3rd order IIR filter**
  - **18-bit fixed point 2’sC arithmetic**
  - **single channel configuration ⇒ 6 coefficients / channel**



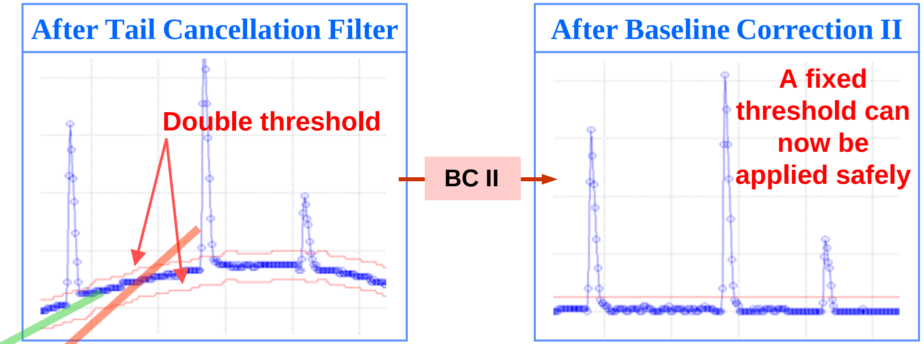
Narrows the pulse



Baseline Correction 2

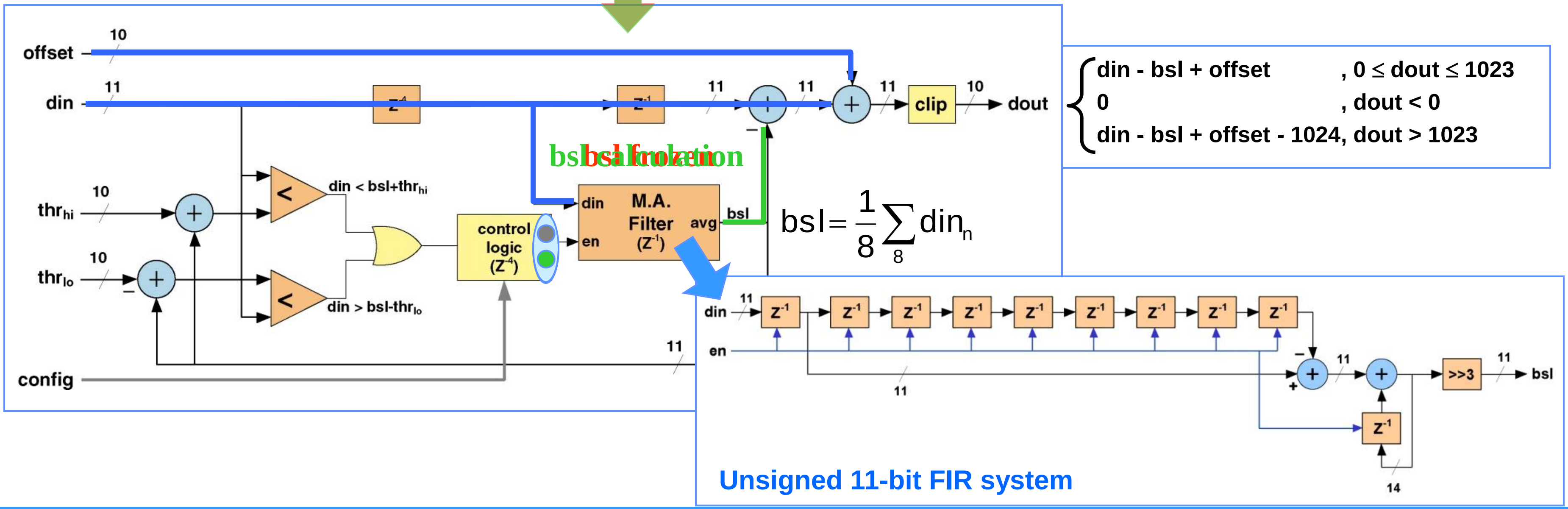
Characteristics:

- Corrects non-systematic perturbations during the processing time
- Moving Average Filter (MAF)
- Double threshold scheme (acceptance window)



Operation

1. Slow variations of the signal ⇒ Baseline updated
2. Fast variations of the signal ⇒ Baseline value frozen



$$\text{dout} = \begin{cases} \text{din} - \text{bsl} + \text{offset} & , 0 \le \text{dout} \le 1023 \\ 0 & , \text{dout} < 0 \\ \text{din} - \text{bsl} + \text{offset} - 1024, & \text{dout} > 1023 \end{cases}$$

$$\text{bsl} = \frac{1}{8}\sum_{8} \text{din}_n$$

Zero Suppression Operation