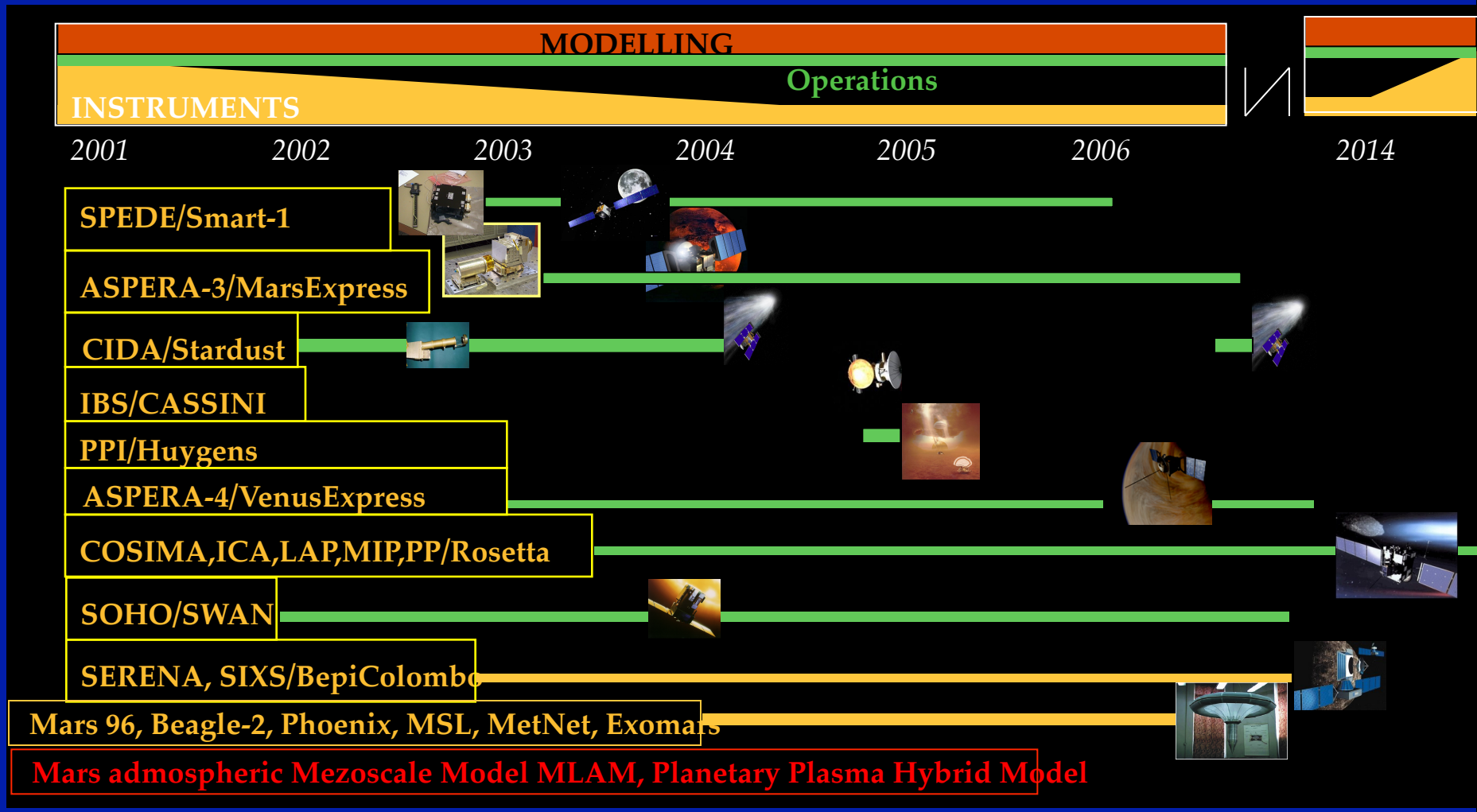
ILMATIETEEN LAITOS
METEOROLOGISKA INSTITUTET
FINNISH METEOROLOGICAL INSTITUTE

SINGLE-CHIP FPGA-BASED PROCESSOR-SENSOR-CONTROLLER

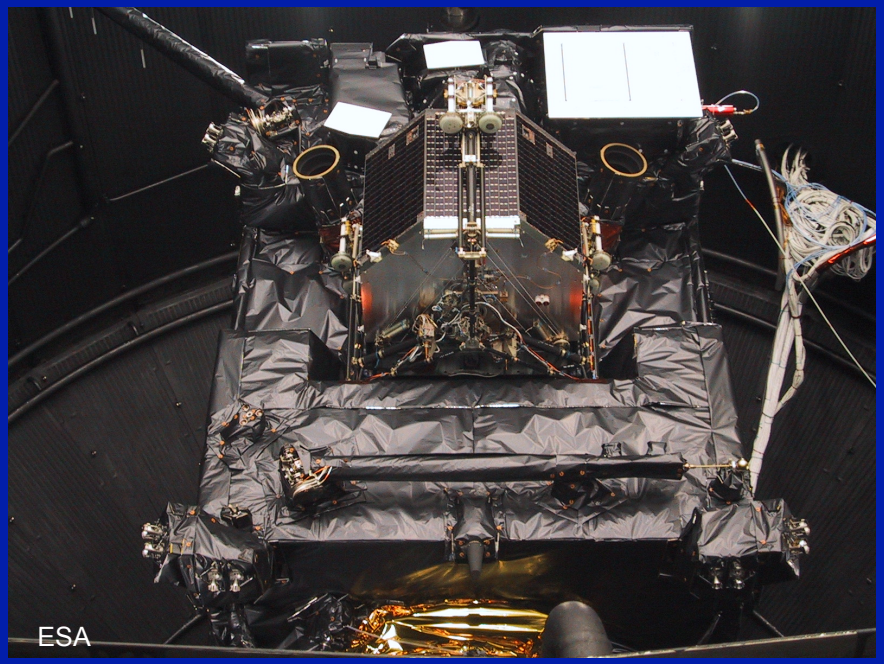
5th International Planetary Probe Workshop, Bordeaux

Walter Schmidt, Anssi Mälkki
Finnish Meteorological Institute, Helsinki
Space Research Division

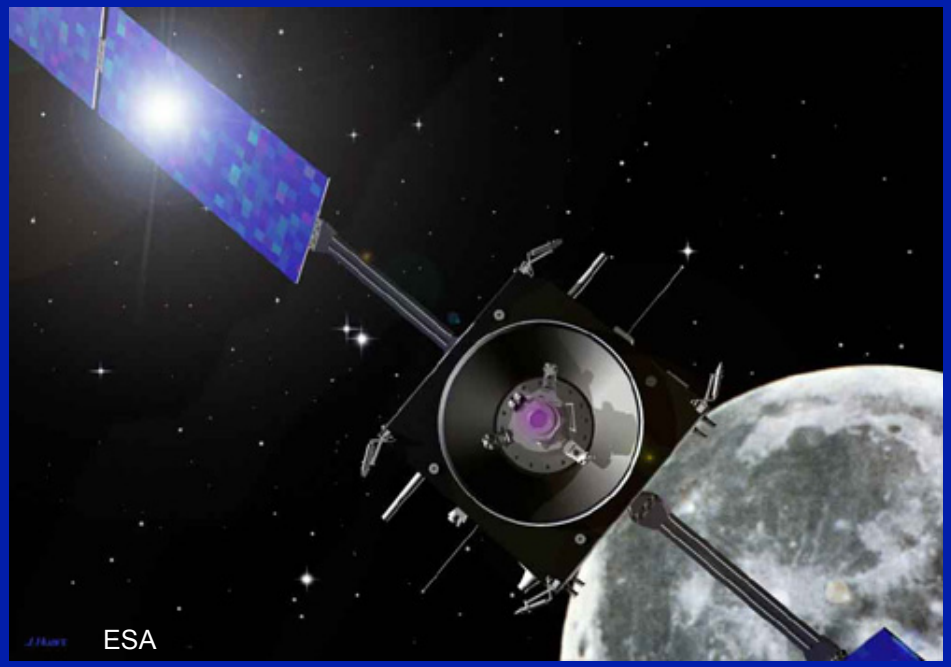
FMI's Activities in Planetary Research



Example Instruments



Permittivity Probe on ROSETTA's Comet Lander PHILAE



Plasma Monitor SPEDE on SMART-1 satellite

Contents

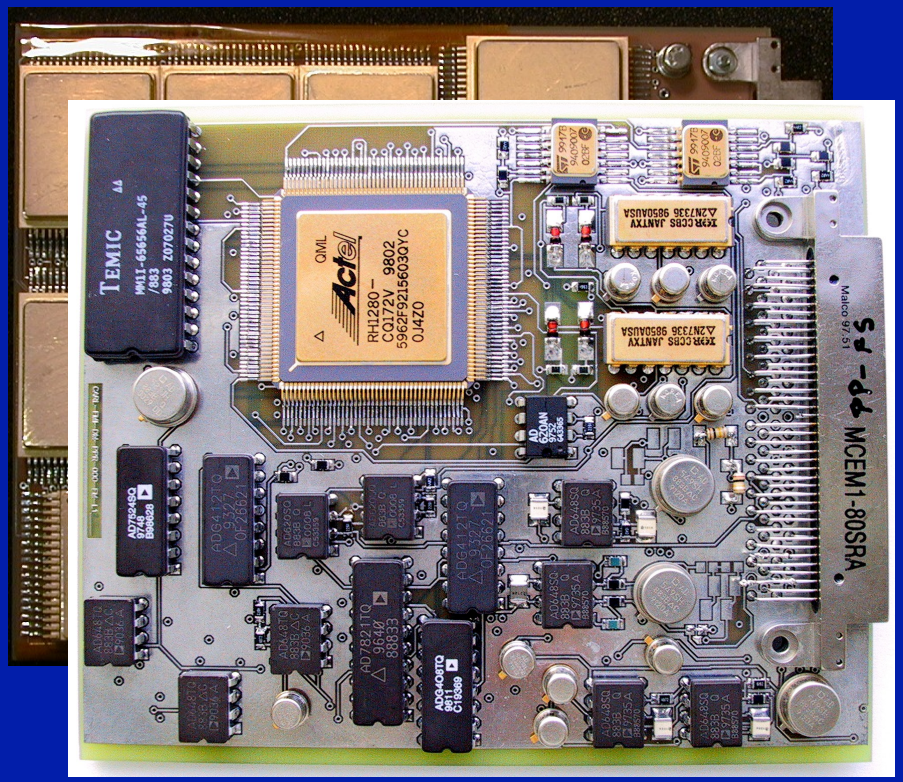
- **INTRODUCTION: Constraints for Instrument Electronics**
- **POWER SAVING STRATEGIES**
- **FPGA-BASED LOW-POWER PROCESSOR**
- **FPGA-BASED ADC/DAC**
- **INSTRUMENT CONTROL & SIMULATION**
- **TOWARDS A *SINGLE-CHIP FPGA-BASED PROCESSOR-SENSOR-CONTROLLER***

Constraints for Instrument Electronics

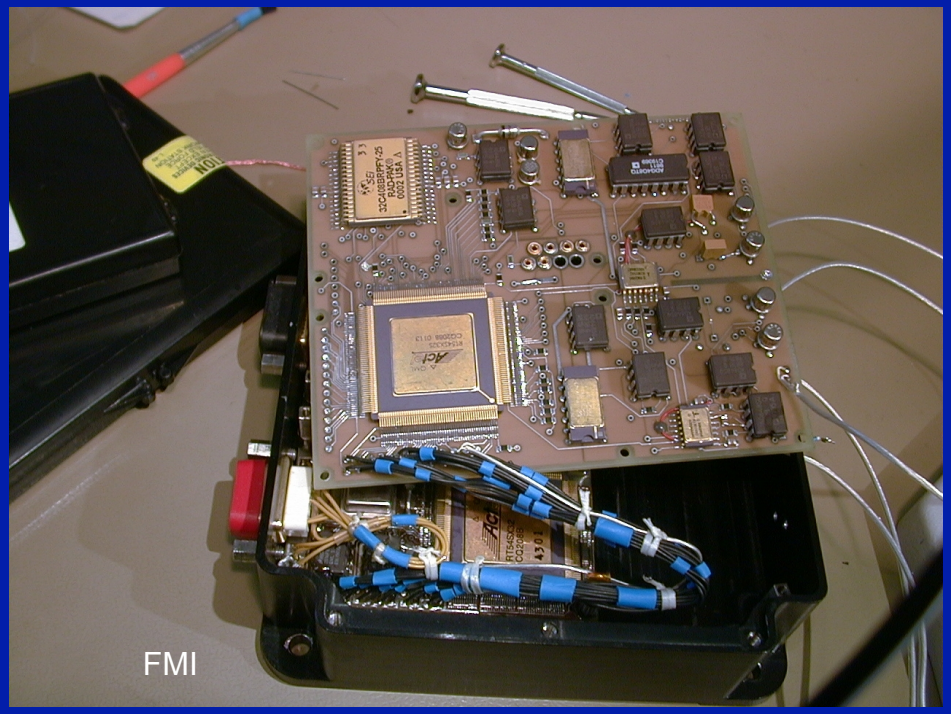
The requirements especially for planetary exploration are partly contradictory:

- **High demands on processing speed but low power consumption**
- **Low mass and small volume**
- **Operational at extreme low and/or high temperatures**
- **Hardened against large accumulated radiation doses but low mass allocation (shielding ?)**
- **Recovery possibility from many failure causes without requiring extra resources.**

Rosetta And SMART-1 Electronics



Rosetta/Permittivity Probe Controller with analog electronics + separate DPU

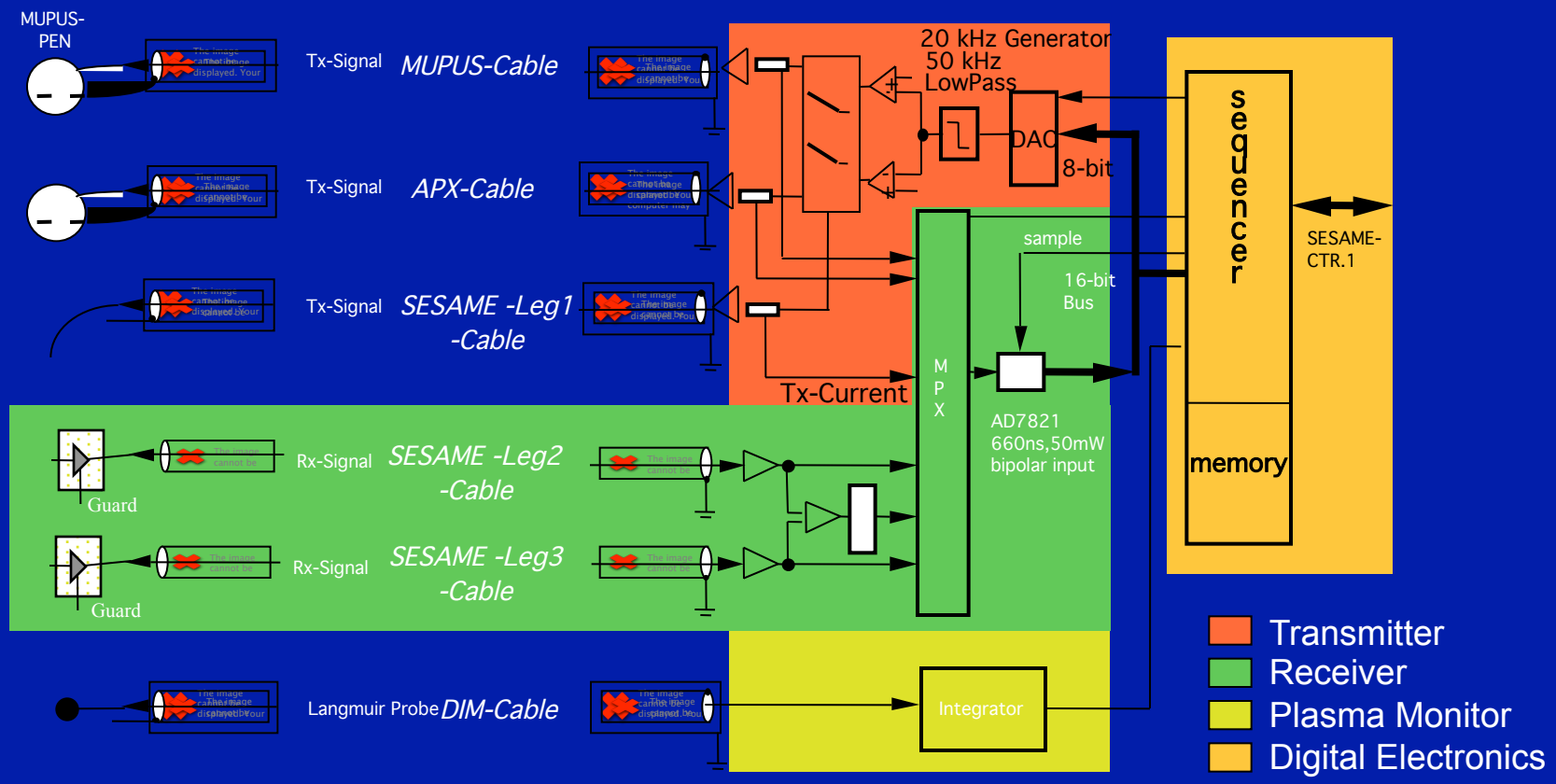


SMART-1/SPEDE combined Controller-DPU with analog amplifiers

Power Consumption Control

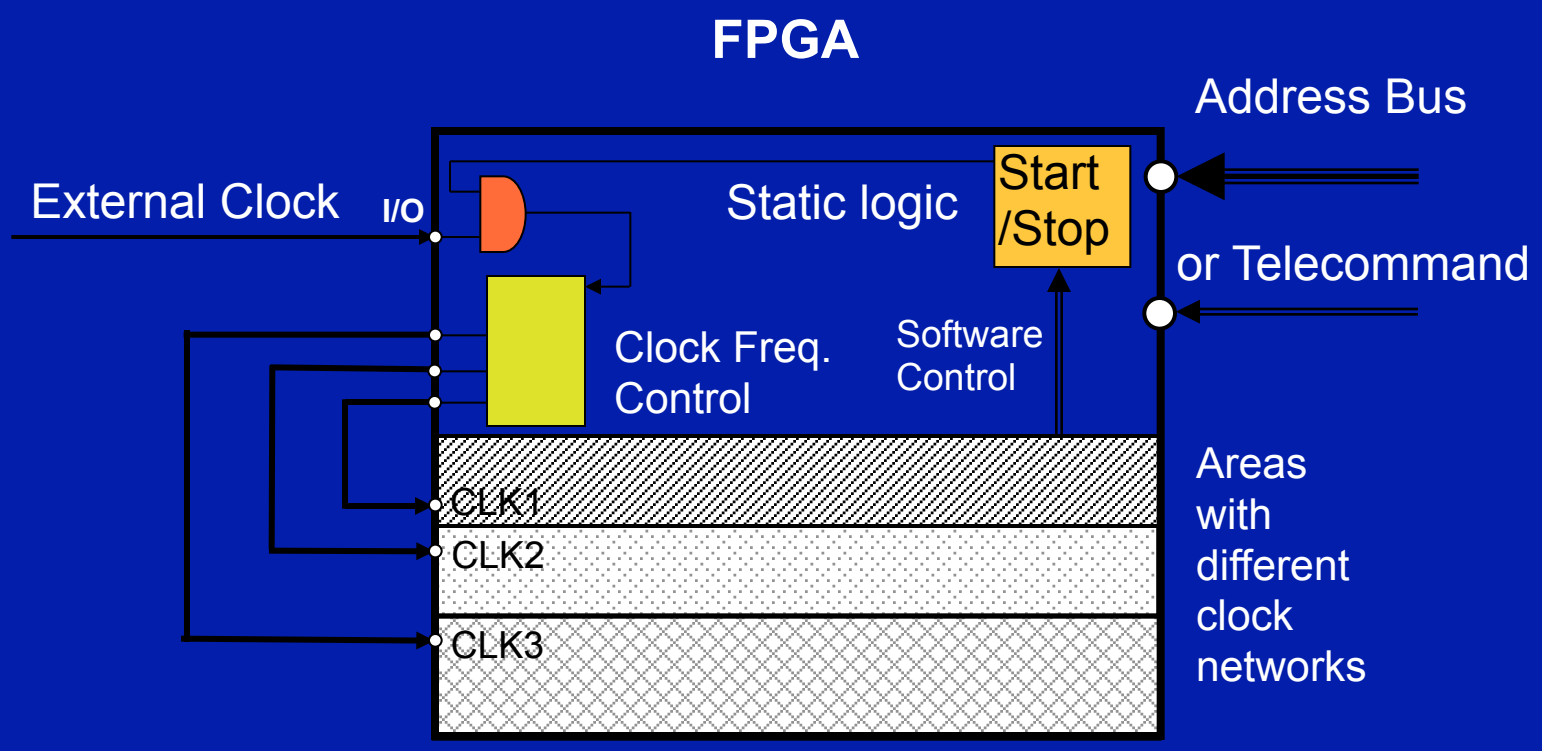
- Separately Powered Electronics Groups

Rosetta / SESAME / PP Electronics Design



Power Consumption Control

- Software / Hardware controlled clock
- Optimized clock speeds for different application areas



Power consumption with disabled clock: 50 µW

Low Mass / Small Volume

- **Integrating most components into 1 chip saves both.**
- **Many earlier available radiation hardened components are no longer available. Rad.tolerant devices need extra mass for shielding.**
- **Digital implementation offers raditation hazard avoidence techniques without excess shielding needs (majority-voting logic, error-correcting codes)**
- **Reducing amount of components reduces needed connections, solder, cable**

Operation at extreme low / high temperatures

- **Removing component types reduces temperature optimization to fewer problem areas**
- **Low temperature limit of electronic components often defined by ceramics carrier, not the die itself**

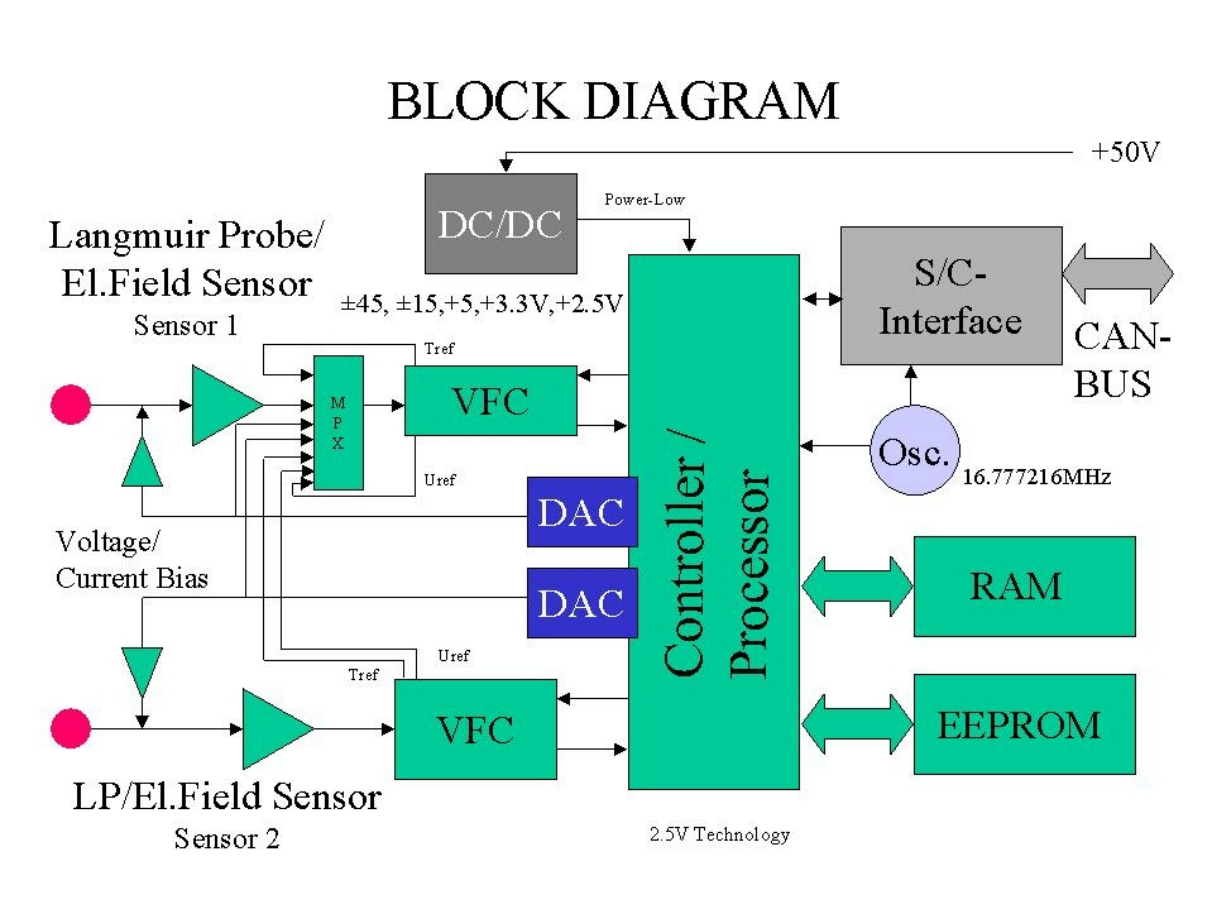
Example: APX on Pathfinder, Rosetta etc uses standard dies, but selfmade high quality ceramics and bonding.

Temperatures of < -180°C were be reached

Bonding to carrier can be done with high-temperature welding, extending upper temperature limit

FPGA-Based Low-Power Processor

SMART-1/SPEDE:



Processor Characteristics

- **16-bit RISC processor code, 510 modules in FPGA**
- **8-bit external memory architecture**
- **Sequencer, Telemetry and low-power events directly mapped into flag register: no interrupts needed**
- **RISC instruction set includes conditional subroutine return definition for each instruction**
- **Memory management system separates code from data area by hardware**
- **Page control system allows simple cross-page subroutine calls and returns**

Software Characteristics

- **Strictly sequential code with endless main loop**
- **Real-time aspects are handled by separate sequencer block**
- **Software language: Specialized assembler**
- **Extremely compact code:**

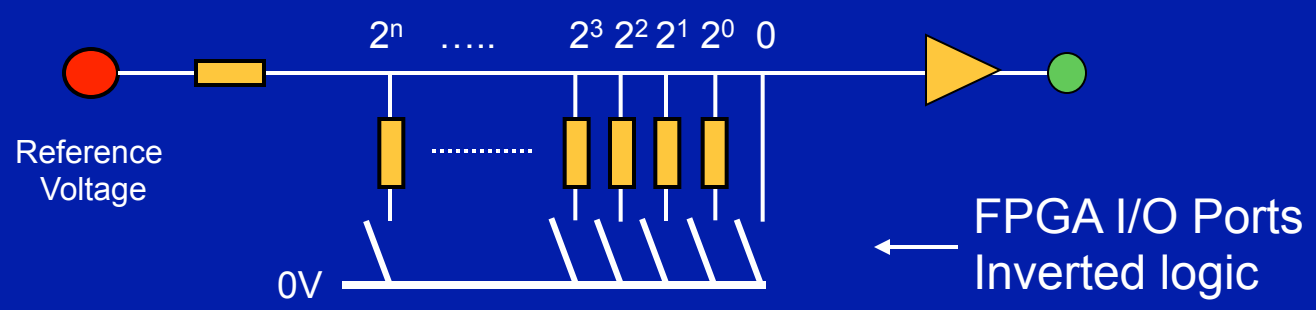
Total code size including wavelet data compression and telecommand/telemetry handling < 16kByte

Software Integrity Checks

- **Boot code implemented in hardware inside FPGA**
- **Software pages are checksum protected**
- **If uncorrectable error dected during boot phase or if software crashes (SEU,error,..) system boots automatically from other memory page.**
- **Per telecommand inter-page copy / refresh can be initiated.**
- **Allows co-residing of different software versions for different operational scenarios**

FPGA-BASED DAC

- **Software controllable analog voltages are generated by a combined serial-parallel resistor network connected to the FPGA I/O-ports.**
- **Any precision, dynamic range and characteristics can be implemented**
- **Conversion time is extremely short. Only 1 external buffer-amplifier for decoupling and passive resistors are needed.**



FPGA-BASED ADC

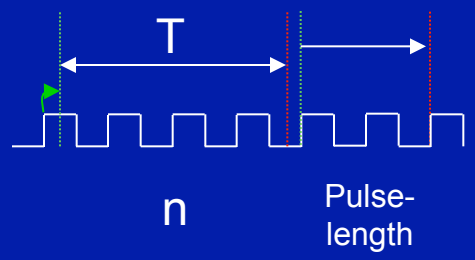
- **Analog voltages are converted to frequencies by small rad-hardended Voltage-To-Frequency Converters**
- **Frequencies are compared to system clock**

a) Number of pulses per integration window

b) Number of system clock pulses per pre-defined number of converter pulses

c) Combination out of a) and b)

Note: c) was used for Huygens/HASI pressure sensor data digitizing



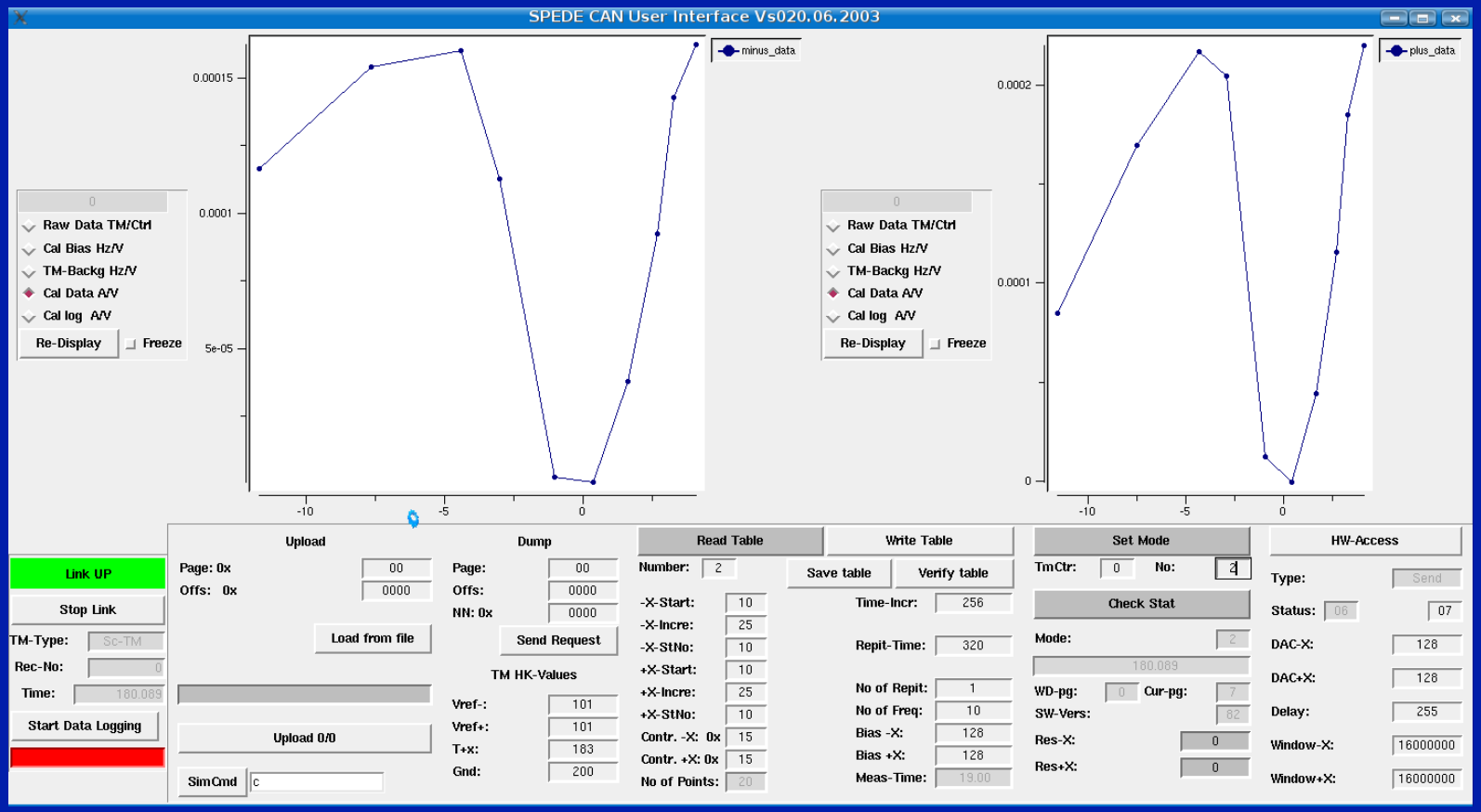
Advantage: Integrating, no loss of information

Disadvantage: Slow conversion times.

INSTRUMENT CONTROL & SIMULATION

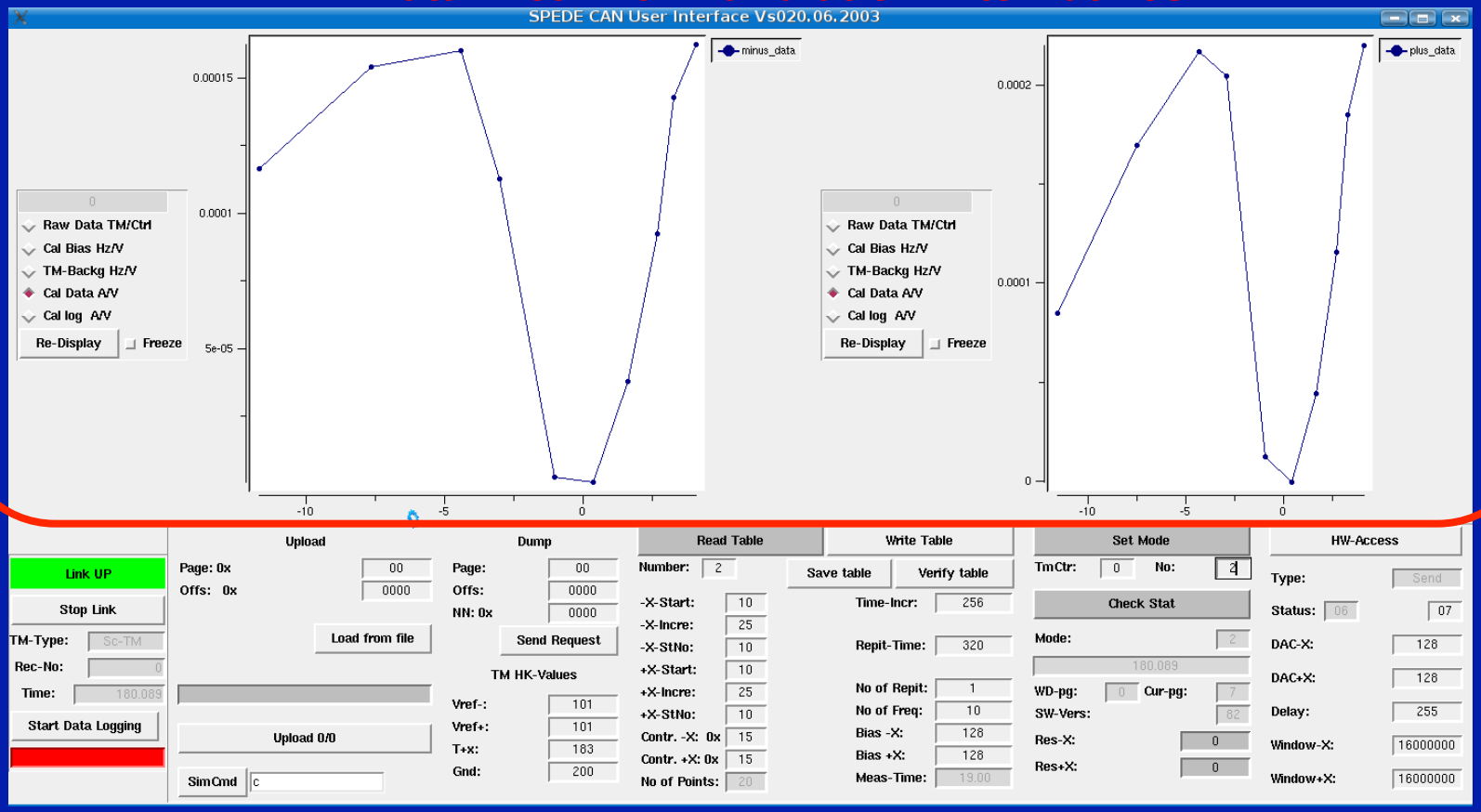
- A highly integrated system needs good verification tools
- FPGA constructs were directly transferred into C-language
- Connection between simulator and GUI via network port
- Simulator includes hardware access option for direct instrument control
- Concept allows identical environment for FPGA development, software development, hardware testing and flight operations. GUI and hardware interface can be at different network nodes

SPEDE EGSE Interface

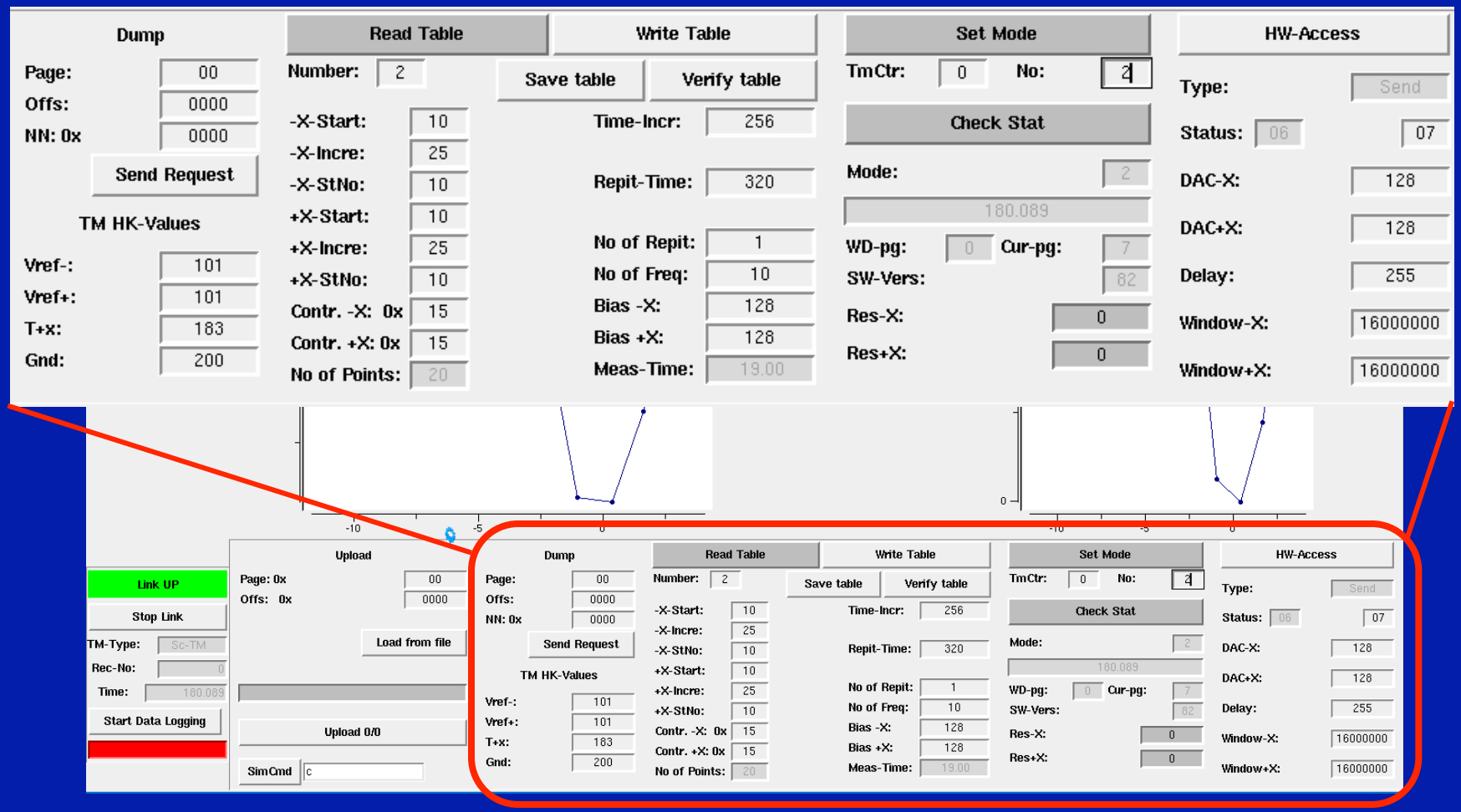


SPEDE EGSE Interface

Data Area with Calibration Alternatives



SPEDE EGSE Interface



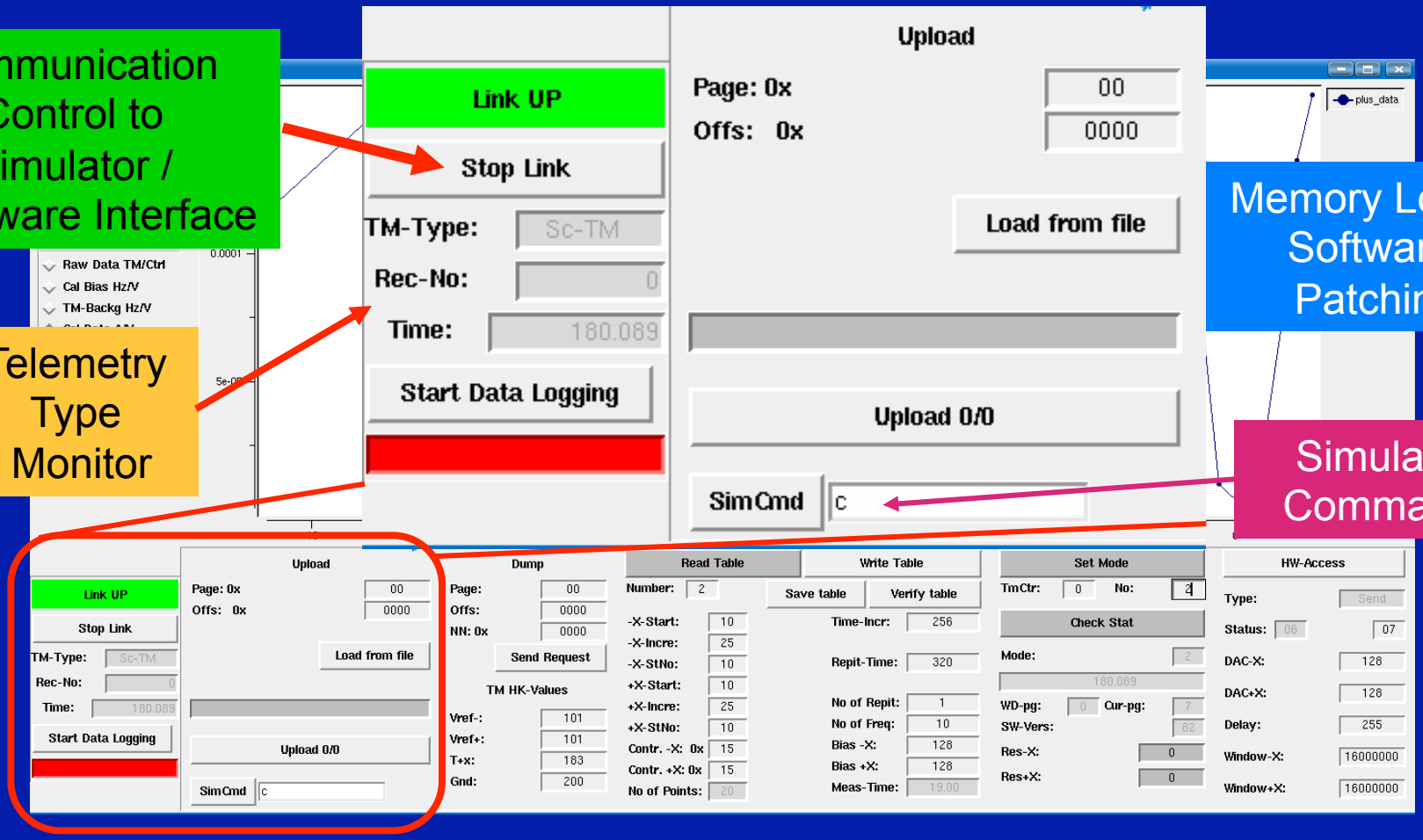
SPEDE EGSE Interface

Communication Control to Simulator / Hardware Interface

Telemetry Type Monitor

Memory Load / Software Patching

Simulator Commands



Simulator Output at Breakpoint

schmidt@ken: /proj/spede/simulator

File Edit View Terminal Tabs Help

```
SPEDE Simulator Version 26.2.2007
SPEDE Simulator: Waiting for interface connection...  Connection established
> Initializing from program file spede082.lst ... Initialization completed
 No of command lines: 2255, Address range: 0x00000-0x01648
> EEPROM Data-area initilized from file sim_eeprom.int
EEPROM Program-area initilized from file spede076.asc
00249:00000  F3018200          LXIW    R1, prog_vers
> Running...CAN-in pck: type=0x5F, len=8 (DataStart) APID=0x1400
CAN buffer accepted
CAN-in pck: type=0x61, len=4 (DataEnd)
CAN buffer accepted
CAN-in pck: type=0x61, len=8 (DataEnd)
CAN buffer accepted
CAN-in pck: type=0x61, len=8 (DataEnd)
CAN buffer accepted
02314:010F8  1110              CPI     R1, 0
> 02315:010FA  7D00            BRI_NZ  get_rt_lsword_0
>
```

… And Corresponding Code Part

ConTEXT - [E:\akt\Planetary-Probe-Workshop_2007\Material\spede082.lst]

File Edit View Format Project Tools Options Window Help

SPEDE Assembler

spede082.lst

```
                ;========== Level 3: Small subroutines ==================================

                ; These routines are small, general purpose ones that probably are called
                ; from several locations.

                ;---- get_rt_lsword: Fetch SPEDE RTclock seconds & sub_secs to R1
                ; Overwrites registers R2, R3
                ; Return: R1 = r1_save, @R3
                get_rt_lsword:
010F4   837A            IN      R2, io_time+1    ; R2 = sec
010F6   8F71            IN      R1, io_time      ; R1 = sub_secs
010F8   1110            CPI     R1, 0
010FA   7D00            BRI_NZ  get_rt_lsword_0
010FC   837A            IN      R2, io_time+1    ; Re-read seconds if it may have altered
                get_rt_lsword_0:
010FE   F3033E88        LXIW    R3, r1_save
01102   3361            STB_F   R3, R1
01104   3372            STB_B   R3, R2
01106   B701            LDW_RET R1, R3
                ;---- eof get_rt_lsword
```

Ln 1, Col 1 | Insert | Sel: Normal | UNIX | File size: 139414

SPEDE Simulator Command Set

sim.c

```
while (io == 1) {
  /* Command interpreter */

  if (iocmd == 'h' || iocmd == '?') {
printf(" h|?      help\n");
printf(" r        display registers\n");
printf(" a        display analog H/W-registers\n");
printf(" l [start [-end]]      list source lines\n");
printf(" l        default: current\n");
printf(" m [start [-end]]      list memory contents\n");
printf(" f value [addr] force value into memory at addr <addr>\n");
printf(" f        default: current; 0x10000r withr=reg.no\n");
printf(" L        (Re-)Define Logfile\n");
printf(" s [number] step through program, default: 1\n");
printf(" c        Continue from current line\n");
printf(" b [number] breakpoint at line .. [current]\n");
printf(" d [number] delete breakpoint at line\n");
printf(" D [number] delete all breakpoints\n");
printf(" g [number] Gauge activation/de-activation\n");
printf(" R        Reset/Restart processor\n");
printf(" W        Watchdog Reset\n");
printf(" C        CAN-register display\n");
printf(" i interr set interrupt number (0,1,2,3); reset: interr-0\n");
printf(" i        (CAN, analog, eeprom, power)\n");
printf(" -        interrupt of run-time software\n");
printf(" .        provision for low-level debugger breakpoint\n");
printf(" .        at simulator source line 997\n");
printf(" q        quit program\n");
printf("> "); fflush(stdout);
```

SPEDE Simulator Command Set

sim.c

```
while (io == 1) {
  /* Command interpreter */

  if (iocmd == 'h' || iocmd == '?') {
printf(" h|?      help\n");
printf(" r        display registers\n");
printf(" a        display analog H/W-registers\n");
printf(" l [start [-end]]      list source lines\n");
printf(" l        default: current\n");
printf(" m [start [-end]]      list memory contents\n");
printf(" f value [addr] force value into memory at addr <addr>\n");
printf(" f        default: current; 0x10000r withr=reg.no\n");
printf(" L        (Re-)Define Logfile\n");
printf(" s [number] step through program, default: 1\n");
printf(" c        Continue from current line\n");
printf(" b [number] breakpoint at line .. [current]\n");
printf(" d [number] delete breakpoint at line\n");
printf(" D [number] delete all breakpoints\n");
printf(" g [number] Gauge activation/de-activation\n");
printf(" R        Reset/Restart processor\n");
printf(" W        Watchdog Reset\n");
printf(" C        CAN-register display\n");
printf(" i interr set interrupt number (0,1,2,3); reset: interr-0\n");
printf(" i        (CAN, analog, eeprom, power)\n");
printf(" -        interrupt of run-time software\n");
printf(" .        provision for low-level debugger breakpoint\n");
printf(" .        at simulator source line 997\n");
printf(" q        quit program\n");
printf("> "); fflush(stdout);
```

SPEDE Simulator Command Set

sim.c

```
while (io == 1) {
  /* Command interpreter */

  if (iocmd == 'h' || iocmd == '?') {
printf(" h|?      help\n");
printf(" r        display registers\n");
printf(" a        display analog H/W-registers\n");
printf(" l [start [-end]]      list source lines\n");
printf(" l        default: current\n");
printf(" m [start [-end]]      list memory contents\n");
printf(" f value [addr] force value into memory at addr <addr>\n");
printf(" f        default: current; 0x10000r withr=reg.no\n");
printf(" L        (Re-)Define Logfile\n");
printf(" s [number] step through program, default: 1\n");
printf(" c        Continue from current line\n");
printf(" b [number] breakpoint at line .. [current]\n");
printf(" d [number] delete breakpoint at line\n");
printf(" D [number] delete all breakpoints\n");
printf(" g [number] Gauge activation/de-activation\n");
printf(" R        Reset/Restart processor\n");
printf(" W        Watchdog Reset\n");
printf(" C        CAN-register display\n");
printf(" i interr set interrupt number (0,1,2,3); reset: interr-0\n");
printf(" i        (CAN, analog, eeprom, power)\n");
printf(" -        interrupt of run-time software\n");
printf(" .        provision for low-level debugger breakpoint\n");
printf(" .        at simulator source line 997\n");
printf(" q        quit program\n");
printf("> "); fflush(stdout);
```

SPEDE Simulator Command Set

sim.c

```
while (io == 1) {
  /* Command interpreter */

  if (iocmd == 'h' || iocmd == '?') {
printf(" h|?      help\n");
printf(" r        display registers\n");
printf(" a        display analog H/W-registers\n");
printf(" l [start [-end]]      list source lines\n");
printf(" l        default: current\n");
printf(" m [start [-end]]      list memory contents\n");
printf(" f value [addr] force value into memory at addr <addr>\n");
printf(" m        default: current; 0x10000r withr=reg.no\n");
printf(" L        (Re-)Define Logfile\n");
printf(" s [number] step through program, default: 1\n");
printf(" c        Continue from current line\n");
printf(" b [number] breakpoint at line .. [current]\n");
printf(" d [number] delete breakpoint at line\n");
printf(" D [number] delete all breakpoints\n");
printf(" g [number] Gauge activation/de-activation\n");
printf(" R        Reset/Restart processor\n");
printf(" W        Watchdog Reset\n");
printf(" C        CAN-register display\n");
printf(" i interr set interrupt number (0,1,2,3); reset: interr-0\n");
printf(" i        (CAN, analog, eeprom, power)\n");
printf(" -        interrupt of run-time software\n");
printf(" .        provision for low-level debugger breakpoint\n");
printf(" .        at simulator source line 997\n");
printf(" q        quit program\n");
printf("> "); fflush(stdout);
```

SPEDE Simulator Command Set

sim.c

```
while (io == 1) {
  /* Command interpreter */

  if (iocmd == 'h' || iocmd == '?') {
printf(" h|?      help\n");
printf(" r        display registers\n");
printf(" a        display analog H/W-registers\n");
printf(" l [start [-end]]      list source lines\n");
printf(" l        default: current\n");
printf(" m [start [-end]]      list memory contents\n");
printf(" f value [addr] force value into memory at addr <addr>\n");
printf(" f        default: current; 0x10000r withr=reg.no\n");
printf(" L        (Re-)Define Logfile\n");
printf(" s [number] step through program, default: 1\n");
printf(" c        Continue from current line\n");
printf(" b [number] breakpoint at line .. [current]\n");
printf(" d [number] delete breakpoint at line\n");
printf(" D [number] delete all breakpoints\n");
printf(" g [number] Gauge activation/de-activation\n");
printf(" R        Reset/Restart processor\n");
printf(" W        Watchdog Reset\n");
printf(" C        CAN-register display\n");
printf(" i interr set interrupt number (0,1,2,3); reset: interr-0\n");
printf(" i        (CAN, analog, eeprom, power)\n");
printf(" -        interrupt of run-time software\n");
printf(" .        provision for low-level debugger breakpoint\n");
printf(" .        at simulator source line 997\n");
printf(" q        quit program\n");
printf("> "); fflush(stdout);
```

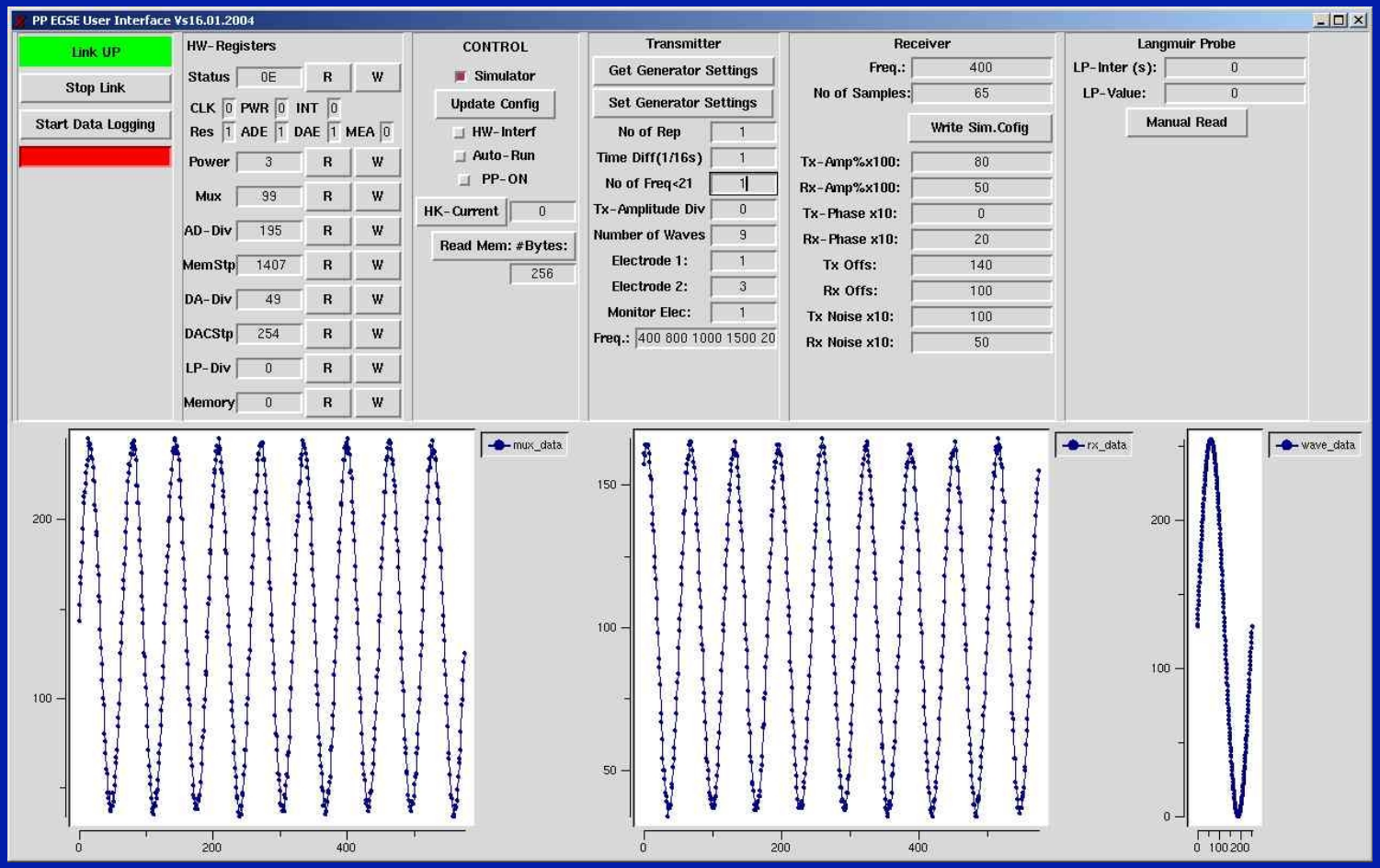
SPEDE Simulator Command Set

sim.c

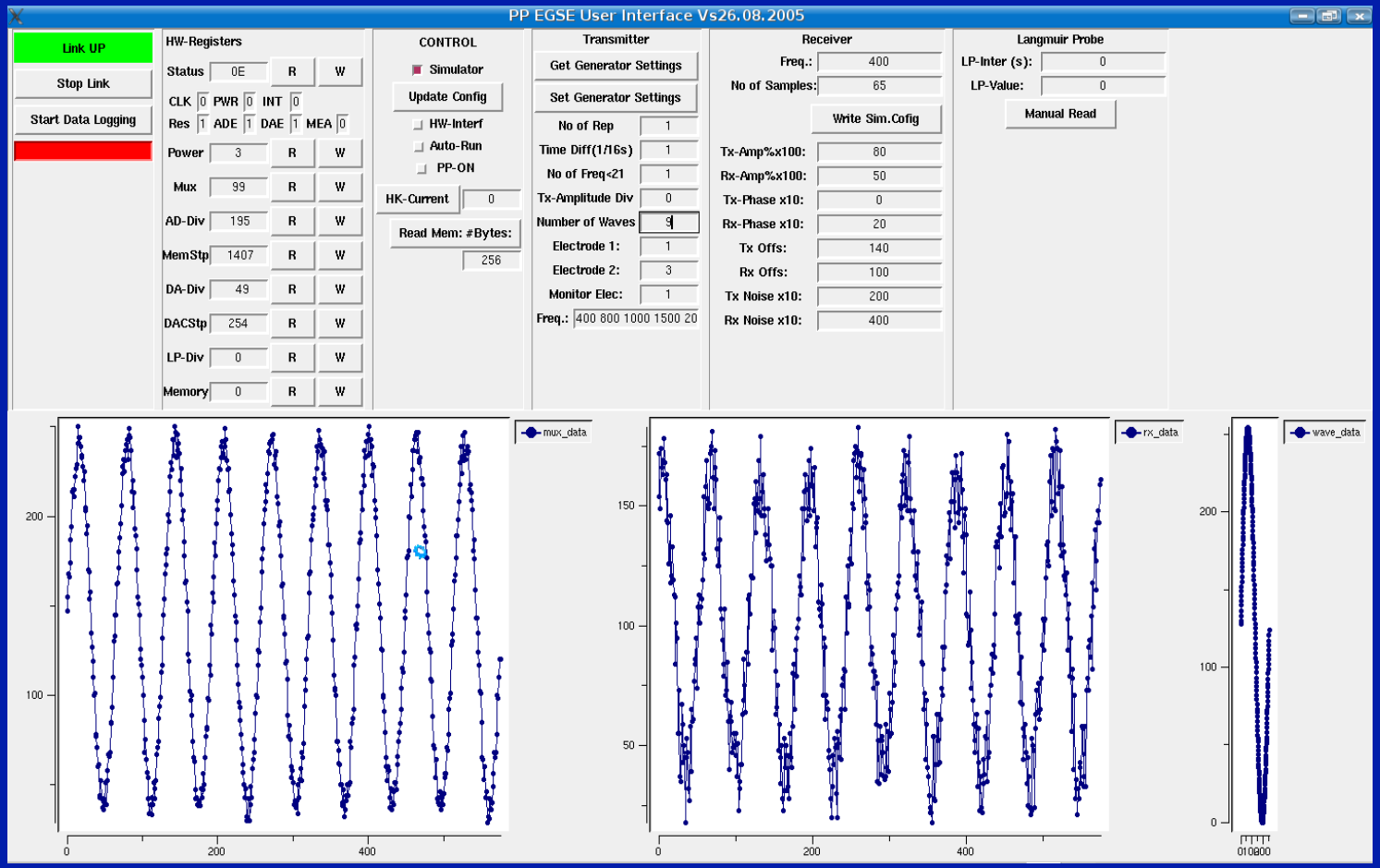
```
while (io == 1) {
  /* Command interpreter */

  if (iocmd == 'h' || iocmd == '?') {
printf(" h|?      help\n");
printf(" r        display registers\n");
printf(" a        display analog H/W-registers\n");
printf(" l [start [-end]]      list source lines\n");
printf(" l        default: current\n");
printf(" m [start [-end]]      list memory contents\n");
printf(" f value [addr] force value into memory at addr <addr>\n");
printf(" f        default: current; 0x10000r withr=reg.no\n");
printf(" L        (Re-)Define Logfile\n");
printf(" s [number] step through program, default: 1\n");
printf(" c        Continue from current line\n");
printf(" b [number] breakpoint at line .. [current]\n");
printf(" d [number] delete breakpoint at line\n");
printf(" D [number] delete all breakpoints\n");
printf(" g [number] Gauge activation/de-activation\n");
printf(" R        Reset/Restart processor\n");
printf(" W        Watchdog Reset\n");
printf(" C        CAN-register display\n");
printf(" i interr set interrupt number (0,1,2,3); reset: interr-0\n");
printf(" i        (CAN, analog, eeprom, power)\n");
printf(" -        interrupt of run-time software\n");
printf(" .        provision for low-level debugger breakpoint\n");
printf(" .        at simulator source line 997\n");
printf(" q        quit program\n");
printf("> "); fflush(stdout);
```

Detector Simulation for Analysis Development



Detector Simulation: Added Noise



Summary

- **Using a system approach which covers all aspects of an instrument up to operations and data analysis, highly integrated solutions with minimized resource demands might be possible.**
- **Current FPGA components allow integration of many instrument control electronics functions into a single chip, which can be optimized for harsh planetary landing environments.**
- **High-performance processors are not always the optimal answer for a set of requirements.**
- **Remaining analog functions can be integrated into a hybrid solution, partly available in rad.tolarant technology already today.**