EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
ORGANIZATION EUROPEENNE POUR LA RECHERCHE NUCLAIRE

CERN-PS DIVISION

PS/BD NOTE 2002-210

FAST WIRE BEAM SCANNER MOVEMENT CONTROL FOR THE PSB

M. Ludwig, S. Burger, U. Raich, J. Olsfors

Abstract

The fast wire scanner scanner uses a bunch of thin carbon fibers which are quickly run through the beam with speeds of 10, 15 or 20 m/s and peak accelerations of several 100 g. This 'wire' is fixed at the tip of a rotary arm as a part of a mechanical system which is driven by a motor outside the vacuum chamber. The motor is controlled by an embedded VME system which relies on speed tables in order to meet the stringent real-time constraints. It provides an optimal movement trajectory and assures independently a reliable and safe operation of the device. The speed tables are derived from the geometry of the mechanical system and from the movement requirements, the embedded software layout and the interfacing electronics are described and the most important technical details are mentioned.

Geneva, Switzerland
13 December 2002

Fast Wire Beam Scanner Movement Control for the PSB

M. Ludwig, S. Burger, U. Raich, J. Olsfors

Contents:

1 Introduction

The fast wire scanner scanner uses a bunch of thin carbon fibers which are quickly run through the beam. The fibers are 6 micron in diameter and approximately 10-15 fibers are twisted into a single "wire". When the wire intercepts a high energy primary beam (> 200 MeV) secondary particles are created that leave the vacuum chamber and are detected by a scintillator and photomultiplier system. The wire position and the photomultiplier signals are sampled with the revolution frequency of the beam. At low energies, below the threshold for pion creation, almost no signal can be seen on the photomultiplier, but the secondary electron emission from the wire can be used instead: the current created by the emission of electrons from the wire is measured.

The fast wire scanner uses speeds of 10, 15 and 20 m/s. With these high speeds heating of the wire can be limited due to the short time of interaction with the beam. The traversal of the wire has hardly any effect on the primary beam and the beam profile does not change when measured during acceleration, so that the effect of adiabatic damping is negligible during the short measurement time.

The PS wire scanner, which is the predecessor of the system described here, has been installed 8 years ago and some of the data acquisition modules are not available any more or much cheaper modules of same performance are available now. In addition a weakness of the system was the dependence of movement control on the operating system. It was largely preferable to liberate the DSC processor from the very stringent real time requirements imposed by the control of the fast wire scanner motion and to use a dedicated processor for this task.

2 System Overview

The fast wire scanner system is now split into two distinct blocks, the data acquisition (DAQ) and the motor control (MC). The MC is provided by a VME slave processor board (Janz VMOD-32)[2] with ADC [3], DAC [5] and parallel I/O [4] units and an interface box for cable adaptation and signal conditioning. The DAQ part contains ADC modules for position (see below) and analogue signal readout from the wires (photomultiplier or secondary emission) and parallel I/O for communication with the motor control block. The DAQ software is running in the VME processor.

Fig. 1 shows an overview of the complete system. Due to the very high acceleration needed, a 400 W brush less motor is employed, controlled by the VMOD-32 DAC using a servo amplifier [6]. The servo amplifier uses speed feedback where the feedback signal is provided by a tachometer. A resolver is mounted on the motor axis to acquire independently the crankshaft angle. It needs an AC voltage of about 7V amplitude and a frequency of 7812Hz applied to its excitation coil, which is delivered by the resolver excitation module inside the VME crate. The sine and cosine output voltages are sampled by the VMOD-32 ADC (for movement control purposes) as well as VD71 VME ADC modules (for position acquisition and data treatment). All units are connected to the interface box in a star-like structure in order to avoid ground loops and facilitate maintenance and diagnostic.

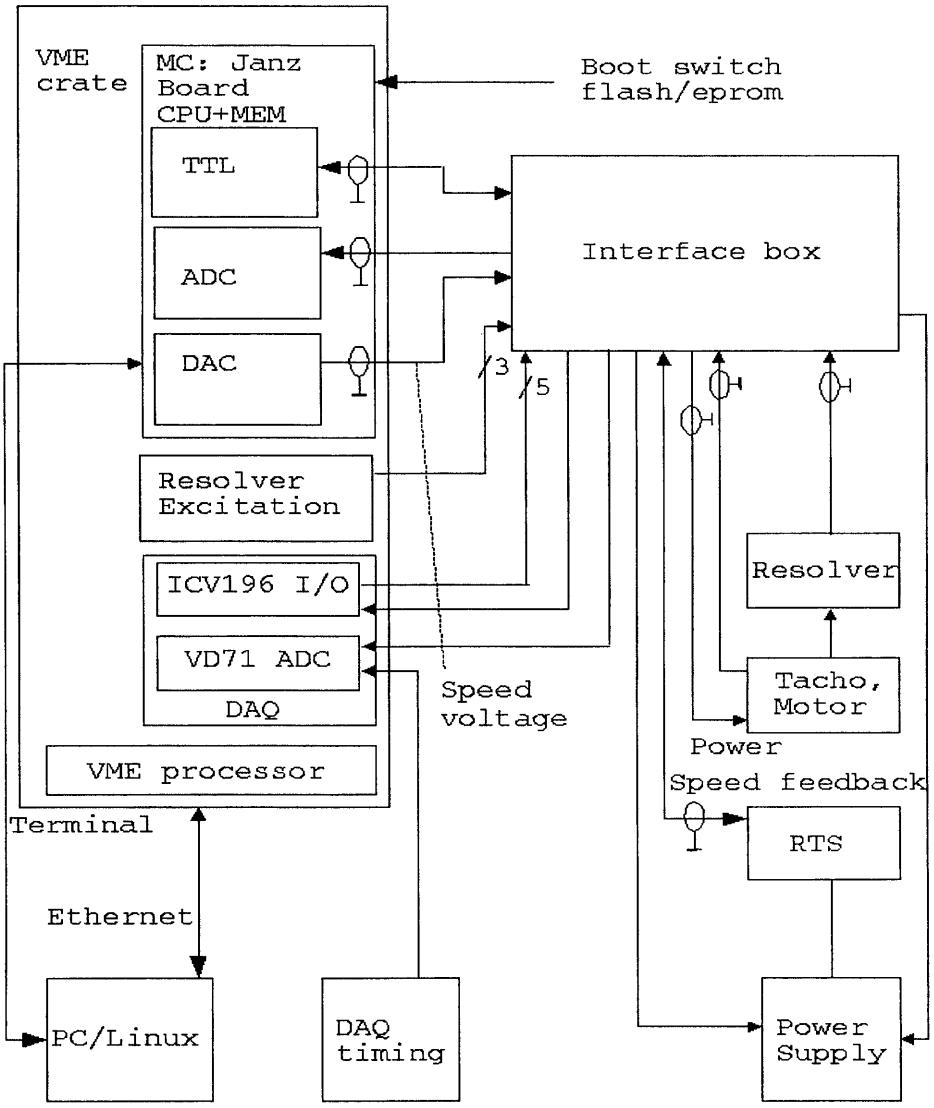


Fig.1: System Overview for one Movement Control Unit (MCU) including the power supply, DAQ timing interface and Development workstation which are shared by all MCUs.

The driving voltage function for the servo amplifier is generated by the embedded software running on a m68322 CPU in the VMOD-32 using a vector table whose values are output in timesteps of 64us to the DAC. The synchronisation for the embedded software is provided using the VMOD-ADC as interrupt source triggered by the resolver excitation module. The VMOD-TTL module serves as digital I/O and interrupt source to read in the limit switches and the triggers: *forewarning*, *start* and

return from the DAQ timing unit. It also controls the status and reset of the servo amplifier and power supply.

In order to download the embedded software, to serve as error terminal, debugger terminal and (networked) cross-development system a PC running Linux can be connected to the front panel serial port of the VMOD-32 module, but the system is operational independently of this PC.

3. The Embedded System

The motor control unit (Janz VMOD-32) is an intelligent VME slave board [2], featuring an 21 MHz MC68322 CPU, EPROM, flash EPROM and SDRAM and 3 ModulBus sockets carrying ADC, DAC and TTL-Parallel I/O piggyback boards. It uses a single VME slot and performs the following tasks:

- interface to an UNIX cross development environment.
- control the wire movement providing 3 user selectable speed curves
- ensure position check and position error recovery in order to protect the wire
- provide error reporting
- interface with the rest of the system

The VMOD-32 includes a simple debug monitor on EPROM providing basic, assembler level debug facilities including a line assembler and a disassembler. Downloading of Motorola S-records (binary program format) over an RS232 serial connection into flash-EPROM or double-buffered SDRAM is also implemented in the monitor. The board may be booted from EPROM (the debug monitor) or from flash EPROM (a user program) selectable by jumpers. Programs are developed on a PC/Linux based workstation using the C language GNU-cross compiler for the M68020 and compatible processors.

Following table 1 lists all used programs, their purpose, their run-time environment and the code formats, including all tools.

Table1: software and formats used in the developing process and during runtime.

program	purpose	run-time	source format
motor	real-time motor control	m68332 embedded system	C and m68020-assembly
termloader	s-record terminal downloader	PC/Linux	C
seyon	Linux terminal emulator on serial line	PC/Linux	C
tabgen	assembler code generator for motor speed vector table	PC/Linux	C
m68debug	built-in assembler	m68332 embedded system	C and m68020-assambly
motordebug	VME auxiliary interface	DSC/LynxOS	C

3.1 VMOD-32 Piggyback Boards

The TTL module (on ModulBus slot 0) provides 20 bits of parallel I/O implemented as two ports A and B of 8 bits and one port C of 4 bits which can either act as input or as output. Port A is used to start the movement which is realized as three triggers: forewarning, start and return. These trigger-bits generate local interrupts on the ModulBus on a high-low TTL signal transition. The other bits of port A read in the limit switches, the power supply and servo amplifier status and the double-sweep status, but they do not generate local interrupts. The port B acts as output to pilot the DAQ during the movement, control the power supply and servo amplifier and visualize the presence of any errors. The port C is used as input and it uses two bits to select one of three vector tables $V_{10}(t)$, $V_{15}(t)$ and $V_{20}(t)$ for peak speeds of 10m/s, 15m/s and 20m/s respectively.

The ADC (on ModulBus slot 1) has 4 single-ended inputs, 12bit resolution, external triggering capabilities and a conversion time of maximum 3us. This is fast enough to sample the minimum and the maximum of the resolver cosine signal at the resolver excitation frequency of 7815 Hz. At every conversion a resolver reference signal which indicates the phase of the excitation and the sine-signal are sampled as well, so that the crankshaft angle can be calculated. The ADC trigger is also used to step through a vector table which represents the "speed voltage" V(t) for a forward movement, or backwards for a return movement, controlling the motor current and thus the movement curve. The stepping of the vector tables is synchronized using external conversion triggers which are followed by "ready conversion" local interrupts on the ModulBus.

The DAC (on ModulBus slot 2) with two available output channels offers 12bit resolution over an output voltage range of 0..10V with 10us maximum conversion time. The first channel is used to control the motor, the second channel is programmed to put out a signal corresponding to the resolver amplitude (calculated from the resolver cosine signal), which is therefore a real-time measure for the crankshaft angle.

The priority of the local interrupts on the ModulBus is determined in descending order: slot 0 (high) .. slot 2 (low), so that i.e. a TTL start trigger interrupt has priority over an ADC ready conversion interrupt. The interrupt latency time is about 20us, which is still fast enough to keep synchronization with the 64us vector table stepping.

3.2 Board Initialization Sequence

Upon power-up the VMOD-32 board boots from flash-EEPROM in order to execute the specific motor control program, this section describes the booting and initialising sequence in a more detailed way, the most important sequences from the source code are quoted in Annex B in the code fragments.

After having found the system stack pointer and the initial program counter the system clock has to be locked to 21MHz (code fragment 1). Then the on-processor System Integration Module (SIM) module and chip select pin assignments are set up.

The 1Mbyte flash-EEPROM on CS4 is mapped to its natural location at 0x500000 (A22 and A20) with asynchronous RW access and 2 wait states (code fragment 2).

In order to use the terminal for message reporting the queued serial module (QSM) and serial line are initialised to 9600 baud bidirectional transmission, and a version message is sent to the terminal (code fragment 3). Initially the booting memory is mapped at 0x0, the EEPROM is already mapped correctly, but at this point all other memories of the board must be mapped to their natural locations, by toggling a built-in address switch (code fragment 4).

Still, the chip select base ***boot*** address register CSBARBT and base address registers CSBAR0..CSBAR3 and their option registers CSORBT, CSOR0 .. CSOR3 must be set up for the ROM containing the JBUG68 assembler/disassembler, the 2k Real-Time-Clock, the RAM, the dual ported battery backed up SDRAM and the local I/O for status and configuration (code fragment 5). With this the physical board set up is finished, and the interrupt vector table which points to the according interrupt service routines (ISR) is set up. Since the cross-development environment offers a comfortable environment under UNIX/Linux for building the binary (as a Motorola-format s-record) the amount of assembler has been kept to the necessary minimum. Only the entry- and return points of the ISR cannot be realised using the GNU-gcc compiler so that they remain in assembler, but the body of the ISR can be written in C (code fragment 6 is the code for the ISR ModulBus slot0).

At booting all the code resides in flash-EEPROM, but in order to achieve faster execution all the code including the ISRs is relocated to address 4000 in RAM for run-time. The function which relocates the code is called and returns in a similar way as an ISR from the initialisation sequence. After the code is relocated the execution of the specific mymain()-function can begin at an address which is calculated (code fragment 7).

Annex A summarises the memory layout during run-time of the embedded program, when the board initialisation and memory moves are finished.

3.3 Motor Control Initialisation Sequence

This section describes the initialisation to the physical motor control system. Firstly the three vector tables for the movements, which are linked and downloaded together with the program, are hooked to the according C-structures, then the functionality of the TTL, ADC and DAC modules are mapped for access, as documented in their respective hardware manuals.

Generally the crankshaft position and movement is not known at this stage, therefore any ongoing movement is stopped and the power supply is switched off. If the limit switches do not indicate HOME position the power supply is switched on again and the servo amplifier is reset. This is followed by a slow movement of the crankshaft until either the limit switches indicate HOME position or a time-out has expired (=ERR1). The power supply goes off again so that the crankshaft stays in HOME position, and the amplitude of the cosine resolver signal is averaged over 128 values and several seconds. This averaged amplitude must be higher than about 5V (if

not, ERR2) so that the resolver cosine signal can be calibrated to the crankshaft angle.

From then on the endless loop handles any occurring errors, the slot0 ISR (TTL) waits for any of the three triggers from the DAQ timing and the slot1 ISR (ADC) calculates the cosine resolver amplitude and steps through the vector table. Annex D gives a list of all errors and their possible causes.

4 Mechanical System: Geometry and Kinematics

The mechanical system [1] transmits the 180° rotational movement of the motor crankshaft outside the vacuum chamber into a longitudinal push-rod movement of small amplitude at the vacuum bellows, and then back into a 136° circular movement of the wire arm inside the vacuum chamber. The wire at the tip of the forked arm crosses the beam center transversally, but for geometrical reasons this movement is highly non-linear. Fig. 2 shows a kinematic representation of the arm mechanics, a technical drawing can be found in the Annex C, the parameters of the geometry are listed in Table2..

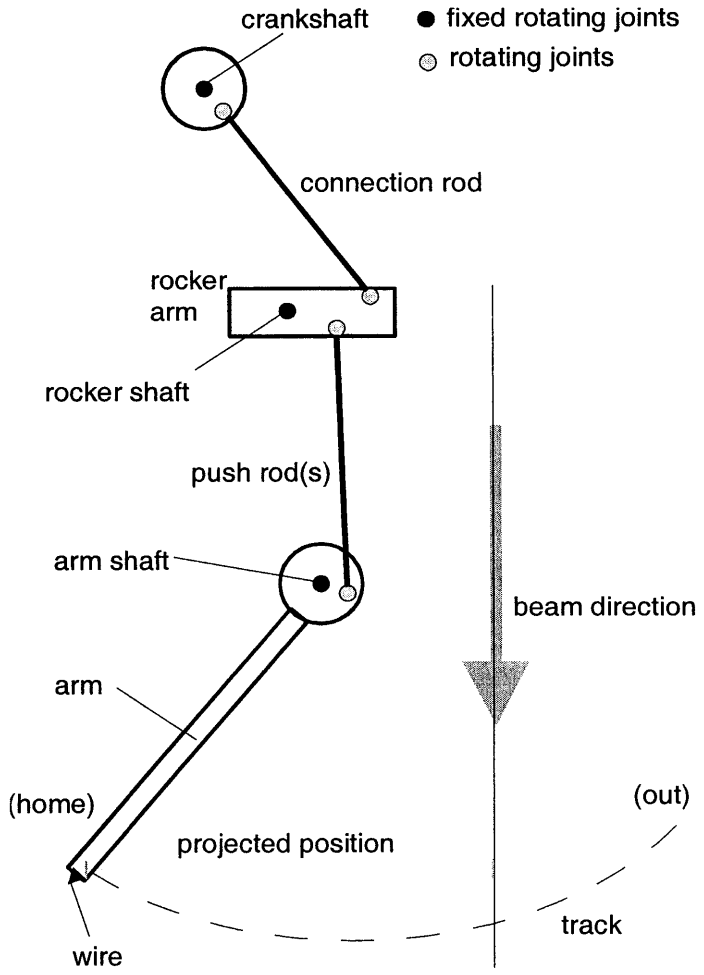


Fig.2: Kinematic representation of the arm system.

Table2

geometrical parameters	
d_h=80mm	horizontal distance crankshaft to rocker shaft
d_v=36mm	vertical distance crankshaft to rocker shaft
$d^2 = d_h^2 + d_v^2$	
l=84.5mm	connection rod length
r_1=11mm	crankshaft radius
r_2=26mm	rocker radius motor side
r_3=16mm	rocker radius rod side
r_4=5.7mm	radius arm shaft
r_5=164mm	fork length

Let $\varphi(t)$ be the crankshaft angular position (crankshaft angle) as a function of time for a movement from HOME to OUT, which is determined by the vector tables $V_{10}(t)$, $V_{15}(t)$ and $V_{20}(t)$ as shown in Fig. 3.

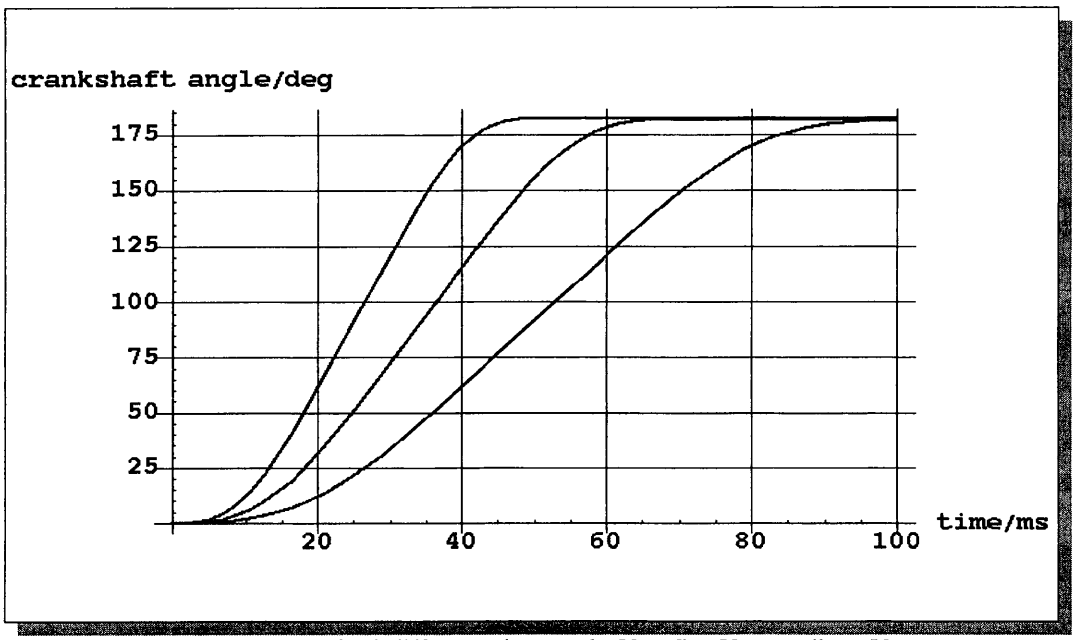


Fig. 3: crankshaft movement for 3 different wire speeds. V_{10}=flat, V_{15}=medium, V_{20}=steep trace .

The longitudinal displacement dl(t) of the push rods is then given by:

$dl(t) = dl_0 - r_3 \cdot \cos(\gamma(t))$ where $dl_0 = r_3 \cdot \cos(\gamma(0))$ is the initial position

and

$$\gamma(t) = pi - \arccos\left(\frac{r_2^2 + d^2 - x(t)^2}{2 \cdot r_2 \cdot d}\right)$$

gives the angle of the rocker arm.

The length x(t) is found from the triangles crankshaft center -rocker arm center - crankshaft arm:

$$x(t) = l \cdot \sin(p - \arcsin(\frac{r_1}{l} \cdot \sin(\varphi))) - \sin(\varphi)$$

The push rod displacement dl(t) varies over some 12mm during a movement , therefore the displacement at the vacuum bellows is kept small (Fig. 4).

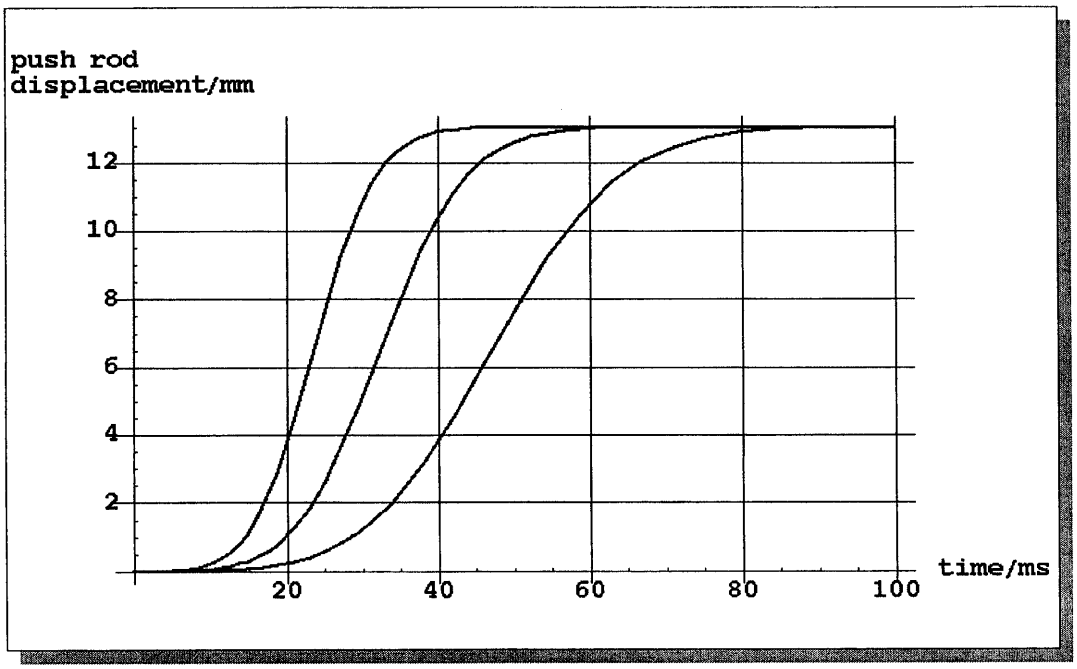


Fig. 4: longitudinal push rod displacement for V_{10}, V_{15}, V_{20} .

Using the radius of the arm axis and the arm length the wire track position z(t) is

$$z(t) = dl(t) \cdot \frac{r_5}{r_4}$$

as shown in Fig. 5. The wire track speed and acceleration (in g) are then dz/dt and d^2z/dt^2 (Fig. 6, Fig. 7).

The peak accelerations for a wire speed of 20m/s are around +230g and -210g; wire speeds of 15m/s and 10m/s need accelerations around +-100g and +-50g respectively. The acceleration curves are slightly asymmetric with respect to the zero-crossing for this arm geometry.

The wire movement with respect to the beam center, projected on a line traversing the beam horizontally or vertically should not vary too much during the data-taking phase in order to achieve uniform position resolution. The projected speeds are shown in Fig. 8. Further improvements to achieve more constant projected speeds during the data-taking phase near to the beam might be possible by

further adjusting the vector tables $V_{10}(t)$, V_{15} (t) and $V_{20}(t)$, but one will have to pay the price of higher peak accelerations. Therefore we choose to minimize the mechanical stress rather instead of optimizing the projected speeds at this point.

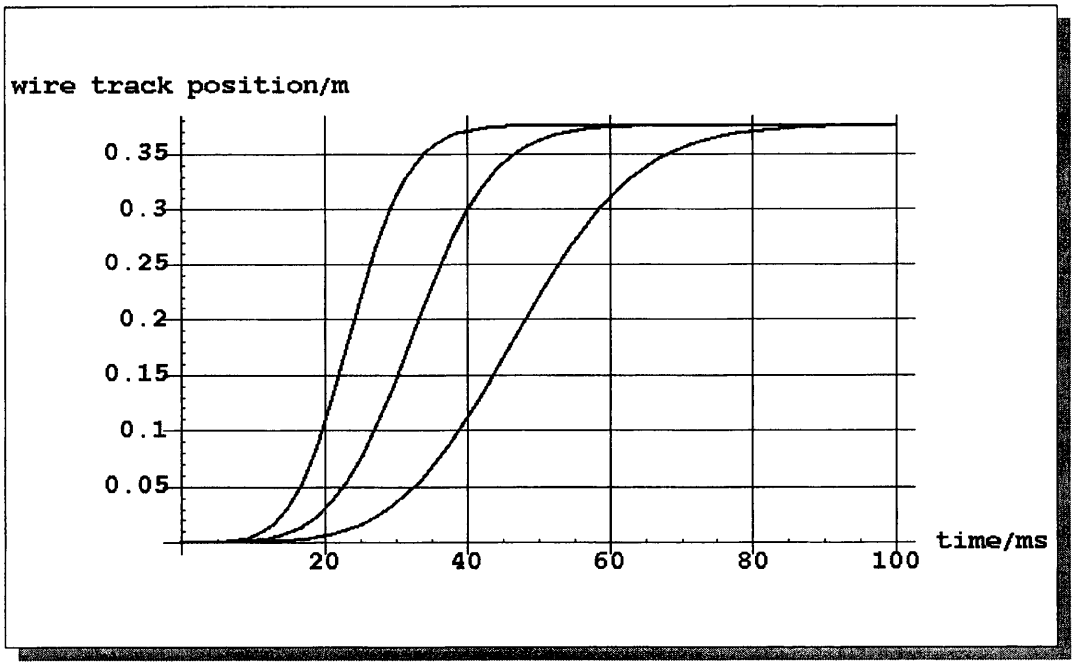


Fig 5: track positions for V_{10}, V_{15}, V_{20}.

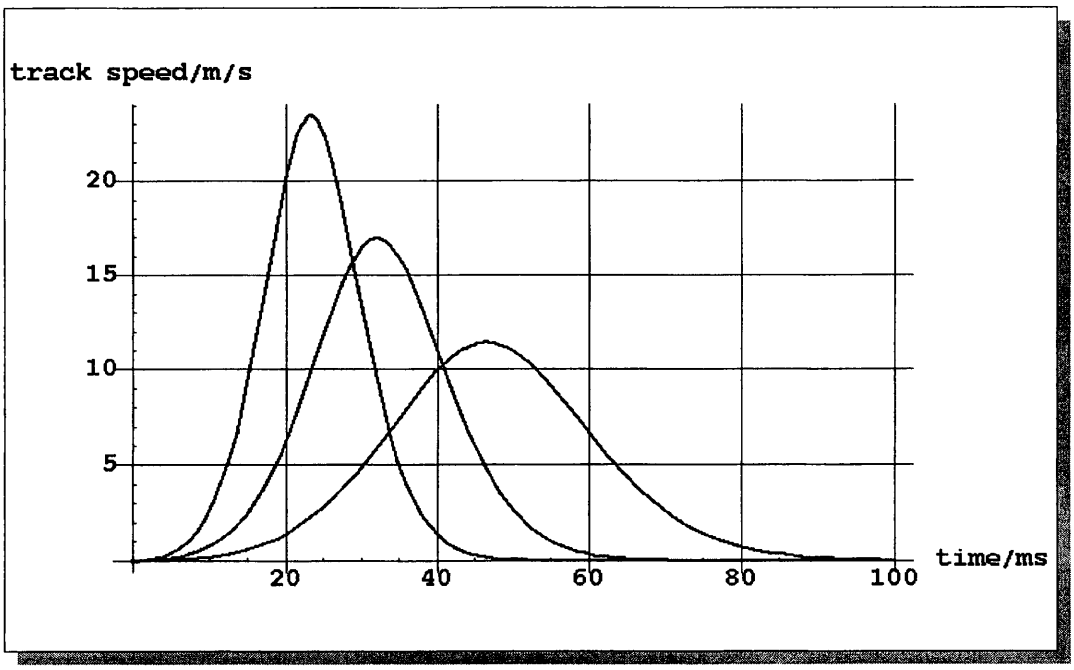


Fig 6: track speeds for V_{10}, V_{15}, V_{20}.

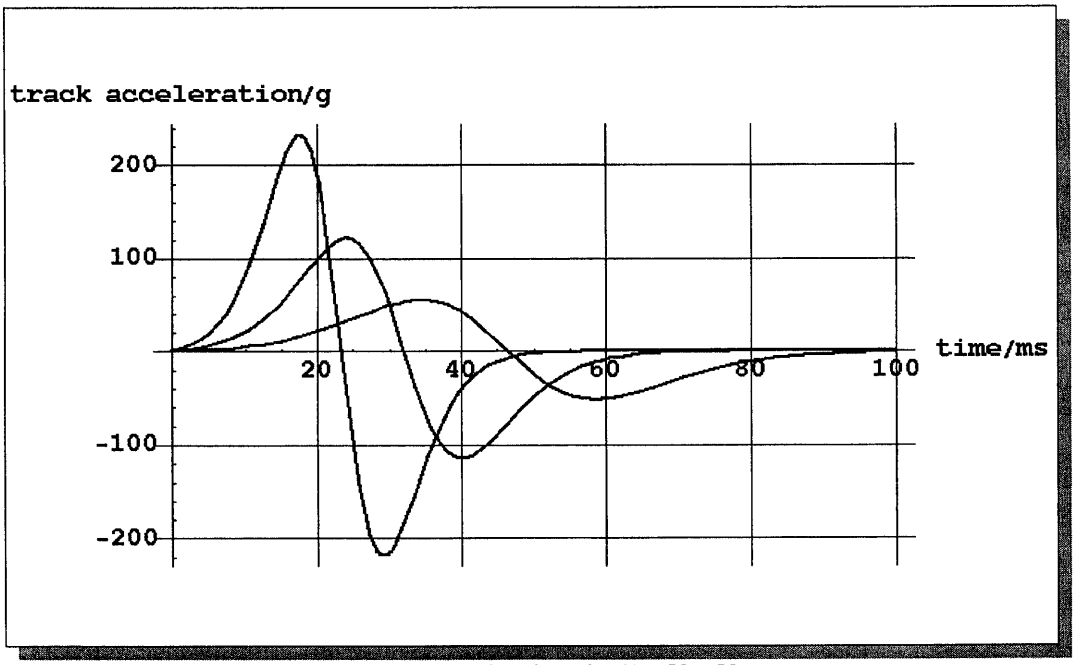


Fig 7: track accelerations for V_{10}, V_{15}, V_{20}.

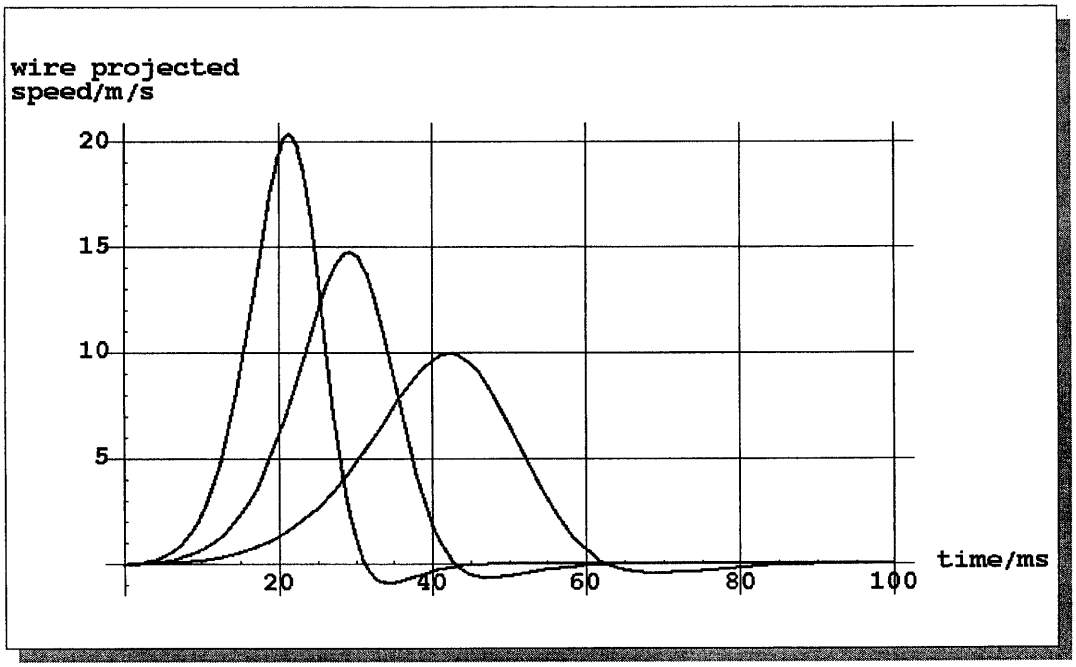


Fig 8: wire projected speeds for V_{10}, V_{15}, V_{20}.

5. Motor Control Hardware

For the motor control of the fast wire scanners an interface box (IMCU) is necessary to connect and condition the numerous signals involved for all eight units in the PS Booster. The acquisition system is equipped with an interface box as well (IDAQ), mainly for the amplification and the conditioning of the signals coming from the wire or the photo-multiplier.

Since the motor chosen to drive the scanner has to be powerful to reach speed and accuracy in positioning, we have designed and developed a special three-phase power supply, the circuit diagram is given in Annex E.

5.1 IMCU: Interface Box for the Motor Control Units

The interface hardware is shown in the system overview in Fig. 1 for one MCU, for the complete set of eight fast wire beam scanners eight of these units are needed plus an additional board that gives common signals like triggers, timing and the resolver excitation signal to all units. All these units are hosted in a 3U Europe rack referred to as IMCU (Annex G). The signals coming in and going out are listed in Table 3, together with some specific features.

The two limit switch signals (HOME and OUT) pass via a 1 bit memory latch to avoid switch rebounds. Both positions are visually indicated by green LEDs on the MCU front panel. When the crankshaft is neither HOME or OUT, generally during a scanner movement, both LEDs are off. These 2 signals are transmitted to the VMOD-TTL (MCU) and to the ICV196 (DAQ).

The signal for the resolver, which is produced by the resolver excitation board (Annex F), excites the rotating coil of the resolver. The two orthogonal (fixed) coils give the sine and cosine of the resolver's axis position, and thus the crankshaft angle can be derived. All three signals are amplified and conditioned to match the specifications of the VMOD-ADC on one side and the specifications of the VD71-ADC on the other side.

Table3: MCU signals

Signal name	Function	Electronics involved
Limit switches	Mechanically detect HOME and OUT position of the crankshaft	Noise filtering, low-pass 1 bit memory
Resolver excitation	Excite the rotor of resolver	Resolver excitation board
Resolver cosine	Detect the crankshaft angle cosine	Noise filtering, scaling for MCU and DAQ
Resolver sine	Detect the crankshaft angle sine	Noise filtering, scaling for MCU and DAQ
RTS Parvex reset	Fault reset	Separate galvanically
RTS Parvex status	Error detect	Separate galvanically
Three-phase PS control	Switch on/off 380V main power supply	Separate galvanically
Three-phase PS status	Status and Error detect of 380V main power supply	Separate galvanically
General error	Signal general error detected by embedded system	Separate galvanically

Signal name	Function	Electronics involved
ADC trigger	Shape ADC compatible trigger signal from resolver excitation output	Monostable + impedance adaptation

The status of the servo amplifier can be read back and it can be reset remotely when it is in error state. The three-phase power supply is controlled by the VMOD-TTL and an opto-coupler. The status of the three-phase power supply is signaled by the servo amplifier which shows +/- 15V only when the three-phase power supply is on.

A red LED permits to see when an error, detected by the embedded system, occurs in the system. For further information on error codes, possible causes and remedies see Annex D.

The trigger signal for the VMOD-ADC which is provided by the resolver excitation board is not supported by the ADC module: it generates a square signal of 16.5Khz with pulse duration of 32ms, which is too long in our case. This value has been reduced to 6.5µs using a monostable chip.

5.2 Resolver Excitation Board

The resolver excitation signal and its associated clocks are generated in the resolver excitation board shown in the system overview (Fig. 1). This card is hosted in a VME crate for convenience and uses only the VME power supply and is otherwise inactive with respect to the VME bus. Its outputs are a 7182Hz sine wave with 13V amplitude for resolver excitation, a TTL-compatible reference output which is in phase with the resolver excitation, and a TTL compatible trigger output with twice the frequency as the resolver excitation (also see Annex F).

6. DAQ Interface

The FWS uses two different signals which are created by the wire passing through the beam: the secondary electron emission (SEM) signal generated in the wire and the signal of the scattered particle shower picked up by a photo multiplier situated near the tank of the FWS per plane.

The SEM signal and the PM signal first pass pre-amplifiers that convert the currents into voltages in the range ±5V. These pre-amplifiers are situated as close to the monitors as possible to get the maximum signal to noise ratio. The SEM pre-amplifier also provides gain control and an option to check the wire, which is activated via the MCU. The SEM and PM signals are then filtered and amplified to match the VD71 ADC inputs. An overview of the DAQ system interfaces can be found in Annex H.

7. Conclusion

The new Fast Wire Beam Scanner System which is installed in the PS Booster now meets the stringent real-time constraints which are imposed by the movement control using a dedicated embedded system, which runs as a slave in a VME crate. This

embedded system relies on standard GNU-cross compilation for the well known m68k processor family and the use of assembler code is kept to the necessary minimum, so that the system integrates well into the PS control architecture and a UNIX workstation development environment can be used for maintenance.

The movement speed and acceleration functions can be optimised to produce 10, 15 and 20m/s peak speeds during the DAQ phase whilst minimising the mechanical stress on the arm. The presented kinematic analysis can be extended to other geometries, mechanics and speeds.

Due to the modular design and the usage of vector tables for the speed control the general flexibility of the system should significantly facilitate an adaptation for use in other accelerators in the future. The modular electronics permits simple scaling between 1 and 8 motor control units, but many more units are possible. Nevertheless the commercially available servo amplifier produces strong radio noise so that extra effort for effective shielding of the electronic components is needed.

The mechanical structure of the present design supports at least 300g, and the destruction threshold is estimated to be above 400g, so that operation at 20m/s around 200g leaves a reasonable safety margin. The real operation of the 8 units installed now in the PS Booster will permit to judge the quality of the overall system.

8 Acknowledgment

We would like to thank R. Sautier and S. Dechamps (PS industrial support, AMSE, St. Genis-Pouilly) for the very active part they took during the discussions in the design phase of the interface electronics, and for their very efficient co-operation.

9 References

[1] http://www.triumf.ca/lhc/txt/instrum/
Triumpf, Canada: mechanical system for the fast wire beam scanner.

[2] Janz Computer AG: VMOD 32. VMEbus 68332 embedded microcomputer. (c) 1995 Janz Computer AG , D-33049 Paderborn.

[3] Janz Computer AG: VMOD 12E4. ModulBus Analog Input. (c) 1999 Janz Computer AG , D-33049 Paderborn.

[4] Janz Computer AG: VMOD TTL. ModulBus Digital 20 channel TTL-IO. (c) 1997 Janz Computer AG , D-33049 Paderborn.

[5] Janz Computer AG: VMOD 12A2. ModulBus Analog Output. (c) 1992 Janz Computer AG , D-33049 Paderborn.

[6] RTS Parvex manual: "Notice d'utilisation RtS 2eme Generation SERVOAMPLIFICATEUR PVD 3487 F - 08/98 - b". (c) PARVEX SA, 8 avenue du lac, B.P. 249, F-21007 Dijon Cedex.

Annex A

Complete memory map of VMOD-32 module at run-time (all numbers hex)

Address	Name	Function	Comment
0	*DRAM lo*	Start DRAM, Init vectors	volatile
3000	DRAM IVT lo	Start Interrupt Vector Table	only MB1, slot0 and slot1 are set up
3210	DRAM: IV MB1	Mailbox 1 interrupt vector	
321c	DRAM: IV slot0	ModulBus slot0 interrupt vector	TTL
3220	DRAM: IV slot1	ModulBus slot1 interrupt vector	ADC
323c	DRAM IVT hi	End Interrupt Vector Table	not all are used
4000	DRAM exec	start user program code, including speed tables	relocated from flash-EPROM to DRAM at initialisation
1FFFFF	DRAM hi 1M	DRAM max.	volatile
200000	n.c.		
3fffff	n.c.		
400000	EPROM lo	EPROM Init and JBUG68 code	set J15 to Boot
4FFFFF	EPROM hi 1M	End EPROM	
500000	flash EPROM lo	bootstrap and user program code, including speed tables	unset J15 to Boot
538000	flash EPROM	approx. end user code	
5FFFFF	flash EPROM hi 1M	n.u.	
600000	DPRAM:MB1 lo	Start Dual Ported RAM MB1 Register	MB1
600008	DPRAM:MB2	Start Dual Ported RAM MB2 Register	MB2 n.u.
600010	DPRAM:MB3	Start Dual Ported RAM MB3 Register	MB3 n.u.
600018	DPRAM:MB4	Start Dual Ported RAM MB4 Register	MB4 n.u.
600020	DPRAM	start VME mapping motor control, W16	logical unit lu
600022	DPRAM	VME mapping motor control	error
600024	DPRAM	VME mapping motor control	Amplitude
600026	DPRAM	VME mapping motor control	resolver cosine
600028	DPRAM	VME mapping motor control	resolver sine
600030	DPRAM	VME mapping motor control	movement table lo, length 1000
601030	DPRAM	VME mapping motor control	movement table maxhi
601032	DPRAM	VME mapping motor control	movement table high
6FFFFF	DPSRAM hi 1M	End DPRAM	n.u.
700000	LIO:RTC	Start local I/O	Real Time Clock
720000	LIO:CAN	CAN controller data	CAN data n.u.
728000	LIO:CAN	CAN controller address	CAN address n.u.
730000	LIO:CFG	Start configuration	
730001	LIO:CFG	Read Hex switch	set lu of module, starting with 0, see DPRAM:600020
73A001	LIO:CFG	VME base address and bus request level 800000+lu*40000	set VME base adress for lu
780000	LIO:slot0	ModulBus slot0: VMOD-TTL Digital I/O	TTL
780200	LIO:slot1	ModulBus slot1: VMOD-12E4 Analog Input	ADC
780400	LIO:slot2	ModulBus slot2: VMOD-12A2(4) Analog output	DAC
780600...	LIO: slot3	n.c.	
...780E00	LIO: slot7	n.c.	

Annex B

code fragment 1

```
        /* set clock synthesizer control reg. VCO=20.97MHz, Y=0x13  */
        move.w  #0xd303,(0xfffa04).L

        /* wait until PLL of synthesizer has locked: test SLOCK bit */
clock:
        move.w  (0xfffa04).L,d0
        move.w  #0xd30b,d1
        cmp.w           d0,d1
        bne             clock
```

code fragment 2

```
        /* chip select pin assignements: CS0 .. CS5                  */
        move.w  (0x7FFA44).L,D0          /* CSPAR0                   */
        andi.w  #0xFFCF,D0               /* set CS1 to 00            */
        ori.w           #0x002c,D0              /* set CS0 11 and CS1 10    */
        move.w  D0,(0x7FFA44).L
```

```
        /* map flash at CSBAR4 to 0x500000: natural location         */
        move.w  #0x5007,(0x7FFA5C).L
        /* CSOR4=async, word, rw, assync, 2WS, unrestricted          */
        move.w  #0x78b0,(0x7FFA5E).L
```

code fragment 3

```
qsm:
        /* clear periodic interrupts                                 */
        move.w    #0x0,(0x7FFA22).L
        movea.l         #0x7FFC00,A0     /* A0=QSM master reg.       */

        /* set up QSM  */
        clr.b           (0x4,a0)         /* clear interrupt level reg */

        /* normal QSM, halt on boundary, interrupt arbitration
       * id 0xE (quite high) */
        move.w    #0x400e,(a0)
        move.b          #0x40,(0x5,a0)   /* QSM IVC reg.             */
wstat:
        /* wait for status & transmit: test TDRE bit in status
         * register. */
        move.w          (0x0FFC0C),d0 /* QSM is still mapped twice    */
        move.w          0x100,d1      /* TDRE: transmit data reg empty*/
        cmp.w             d0,d1
        beq.l             wstat


        /* serial line setup: set TXD (=transmit data)               */
        move.b          #0x80,(0x15,a0)  /* QPDR: port data reg.     */
        move.b          #0x80,(0x17,a0)  /* QDDR: data direction reg. */

        /* SCCR1: SCI Ctrl Reg 1: disable receiver & transmitter     */
        move.w          (0xa,a0),d0
        andi.w          #0xfff3,d0
        move.w          d0,(0xa,a0)

        /* now receiver & transmitter are off, we can safely
       * set the baud rate to 9600                                */
        move.w    #0x44,d0
        move.w          D0,(0x8,A0)    /* SCCR0: SCI control reg.0    */

        /* SCCR1: switch on again: enable receiver & transmitter     */
        move.w          #0xa,d0
        move.w          d0,(0xa,a0)

        /* set up greeting message                                   */
ostart:
        lea             (otext,pc),a1  /* point to msg. below        */
        move.w          #180,d3        /* which has exacly 180 chars  */
tdre:
        move.w          (0xC,A0),D0      /* wait loop until ready     */
        and.w           #0x100,D0
        beq.l           tdre
```

```
ostep:
        /* write the message of 180 chars */
        clr.w           d0
        move.b          a1@+,d0
        move.w          D0,(0xE,A0)
        subq.w          #1,d3
        bne             tdre

        /* jump to flash at 2nd location natural address
       * EEPROM_NATURAL_ADDRESS=0x500000                          */
        jmp             (reloc + EEPROM_NATURAL_ADDRESS)

otext:  /* line below has 60 chars, \n\r counts as 2 chars */
 .ascii         "\n\rWire Beam Scanner Motor Control v.1.0 (c) FelixSoft     \n\r"
 .ascii         "CERN PS/BD Michael.Ludwig@cern.ch  Uli.Raich@cern.ch      \n\r"
 .ascii         "Stephane.Burger@cern.ch                                   \n\r"
 .byte          0x4
```

code fragment 4

```
reloc:
        move.b  #0,(0x300000).l         /* address switch JANZ          */
```

code fragment 5

```
        /* CSBARBT: chip select base boot: 1M ROM at 0x400000,
         * but the JBUG68 is not used                                   */
        move.w  #0x4007,(0x7FFA48).L

        /* CSORBT: chip select boot option reg.                         */
        move.w  #0x68b0,(0x7FFA4a).L /* word, RO, asyncroneous,
                                      * Supervisor/User, 2 wait */

        /* CSBAR1: chip select base address 1: 2k Real Time Clock   */
        move.w  #0x7000,(0x7ffa50).L
        move.w  #0x78b0,(0x7ffa52).L    /* CSOR1                */
        /* CSBAR0: chip select base address 0: map to hi and strobe */
        move.w  #0xfff8,(0x7ffa4c).L    /* CSBAR0               */
        move.w  #0x2bc0,(0x7ffa4e).L    /* CSOR0                */

        /* CSBAR2: chip select base address 2: map to hi and strobe */
        move.w  #0xfff8,(0x7ffa54).L    /* CSBAR2               */
        move.w  #0x28cc,(0x7ffa56).L    /* CSOR2                */

        /* DCBAR3: chip select base adress 3:
         * 128k local I/O 0x700000 .. 0x737fff: can controller and
         * status & config registers                                */
        move.w  #0x7204,(0x7ffa58).L    /* CSBAR3               */
        move.w  #0x7930,(0x7ffa5a).L    /* CSOR3                */
```

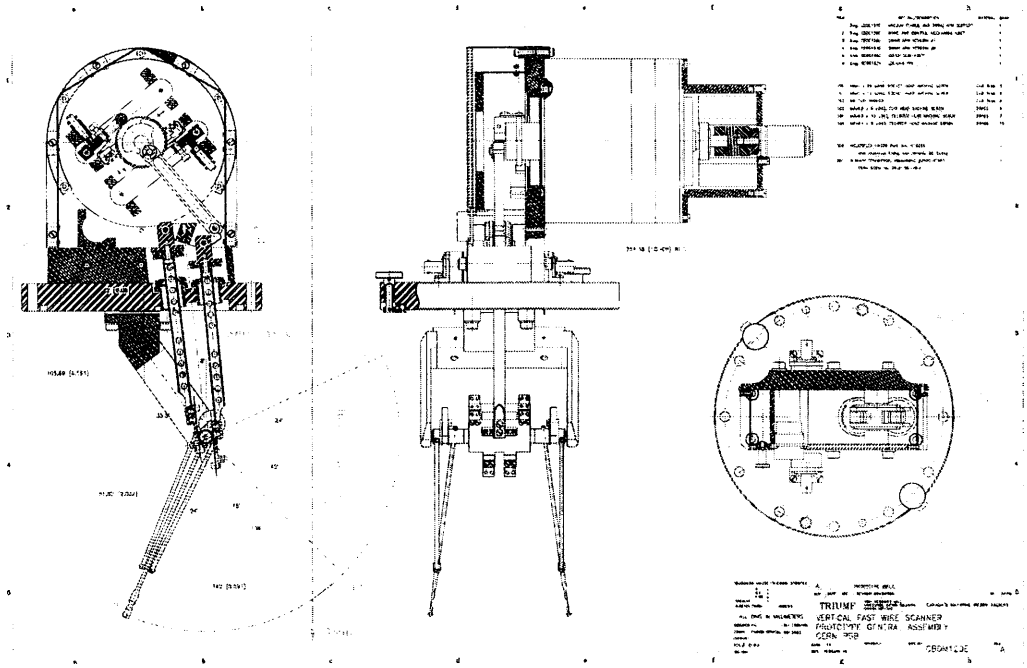
code fragment 6

```
_isr_slot0:
        movem.l D0-D7/A0-A6,-(a7)   /* save all context */

        /* call a C function (void) cisr_slot0(void) which acknowledges
         the interrupt, does something and then returns */
        jsr             (_cisr_slot0 + RELOCATE_START_ADDRESS)
        movem.l (a7)+,D0-D7/A0-A6   /* restore context */
        rte
```

code fragment 7

```
        move.l  #___main,d0
        move.l  #_mymain,d1
        sub.l           d0,d1
        addi.w  #0x2,d1

        /* code is relocated to RAM at 0x4000, that means 0x500000 -
         * 0x4000 = 0x4fc000 to __subtract__ from d1 */
        subi.l      #0x4fc000,d1
        jmp             (d1,pc)
___main:
```

Annex C



Technical drawing of the fast wire beam scanner

Annex D

ERROR INTERPRETATION FOR FAST WIRE BEAM SCANNER MOTOR CONTROL

ERR1: INIT couldn't move HOME

- noise on the TTL messed up the PS switch, try again.
- Motor does not move at all: If the RTS is blinking, check cables to motor
  check all cables on Janz' module and resolver excitation board for the ADC trigger.
- Motor moves: HOME position is not recognized because a switch doesn't work
- PS is broken, check fuses on the back side
  A very basic error, either very simple to fix or something really broken, can't continue

ERR2: INIT calib cos Amplitude too low

- the signal from the cosine resolver winding should have max (positive) amplitude at HOME, and max (negative) amplitude at OUT. At init we drive HOME, and if the cosine amplitude is lower than 50% (approx. 5V) that's an error. Probably the resolver clutch has slipped
- the ADC ext. trigger doesn't work any more, check resolver excitation card and interface box electronics.
- if nothing is connected the amplitude will be pulled-up to 0xfff (not reported as error)
- if it doesn't show an amplitude very low, the movement can in principal be done, BUT the offset position compensation (which reads the sine to decide in which direction it must go) will not work corretly, therefore slow erratic movements are very likely.
- Reset the system, try several times. If the amplitude is changing, the resolver slips.
  It is better not to continue and to switch off this unit after making sure that the wire is HOME, otherwise risk of wire burn.

ERR4: both limit switches indicate 1

- Probably the interface box is not on, therefore the electronics reports "high" twice
- Small blocking screw on one/both switches went loose with mechanical vibrations therefore the switching point is deadjusted.
- interface box electronics is broken
- A switch is broken
- resistors inside burndy connector on motor chassis are defect, cable disconnected
  A very basic error, either very simple to fix or something really broken, can't continue

ERR8: RTS is blinking/not OK
- The RTS is unhappy, this should normally not happen at all, but maybe there was noise somewhere (difficult to investigate). Normally the RTS gets resetted automatically, and after a full sequence of triggers all should be fine.
- If it is persistent, check the DAC output, the self inductance, the motor, the interface box electronics.
- There is some risk that the motor turns uncontrolled with max speed (=system destruction), but most of the time this should be noise-related and rather harmless. If it doesn't go away it is better not to continue and to switch off this unit and deconnect the RTS after making sure that the wire is HOME, otherwise risk of mechanical destruction.

ERR10: Power Supply is not ON for move, switch it off and on again.
- the triphase should be ON but isn't, this is checked every time before a movement is actually done. The PS is switched ON at forewarning trigger, stays always on in OUT, and goes OFF if in position HOME (switch) for longer than a timeout. The timeout does not expire if the position offset compensation is on too often, this means that the arm is making a slow oscillating movement around HOME or OUT. See Position offset compensation warning.
- check trigger sequence and PS fuses. If the error persists and the arm is HOME there is no danger. there is a low risk that the arm is somewhere stuck in middle position.
- the triphase is simple and solid, it is most probably the fuses or the interface box electronics.

W/ERR20: arm is HOME but shouldn't be
- that is a warning, the arm is expected to be somewhere else than HOME (i.e. moving or OUT), therefore it will be brought HOME by the pos. offset compensation. Should go away normally, can pop up once immediately after each start trigger.
- if really persistent then something else more serious is broken but doesn't get reported so it is better to be cautious and check the system.
- check trigger sequence.

W/ERR40: arm is OUT but shouldn't be
- same as above but for OUT position. Normally pops up once immediately after each return trigger.

W/ERR80: position off. compensation
- a warning: the position offset compensation has decided to move the arm to the position where it should presently be, and to keep it in this position until other things happen: either in HOME or OUT position. Needs the sine signal to decide which direction to move.
- pops up after movement is finished but hasn't exactly arrived at the required position.
- Can pop up occasionally after a movement is finished, if too often maybe readjust the movement tables and RTS_TACHY constant (see table generation) to hit the end positions more precisely. Check end positions of movements with diagnostic contact.
- Can pop up regularly after movement HOME -> OUT, because it keeps the arm in OUT position. This is neccessary only if the speed offset of the RTS is not precise enough adjusted (to give 0 speed for 0 consigne), readjust RTS.
- if there is a slow oscillating movement at 90deg position then cosine and sine signals are swapped (maybe fatal for the wire).

Interpretation of the front-panel LEDS

the LEDS U0, U1, U2, U3 on the Janz' front panel are used to report certain status and action of the embedded SW.

all 4 LEDS blinking for a short while after power on:

- The arm is not HOME, the arm is moved HOME after power-on, as part of the initialisation sequence

U1 stays on for ca. 5 secs after power on:

- The cosine amplitude at HOME is read for 128 times and averaged, in order to normalize the resolver readout. The PS and RTS are off, the arm is HOME.

U0 is blinking:

- This means that the embedded SW is alive. Every 5th time the main() passes it switches U0.
- If U0 is blinking rather fast (disconnect ADC external trigger from i.e. resolver excitation board) this means that there are no ADC-interrupts (=fault). The blinking should slow down considerably when the ADC is triggered correctly.

U1, U2, U3 are sometimes flashing shortly:

these correspond to the triggers received on the TTL interrupt-service:

- U1: forewarning
- U2: start
- U3: return (normally in this order).

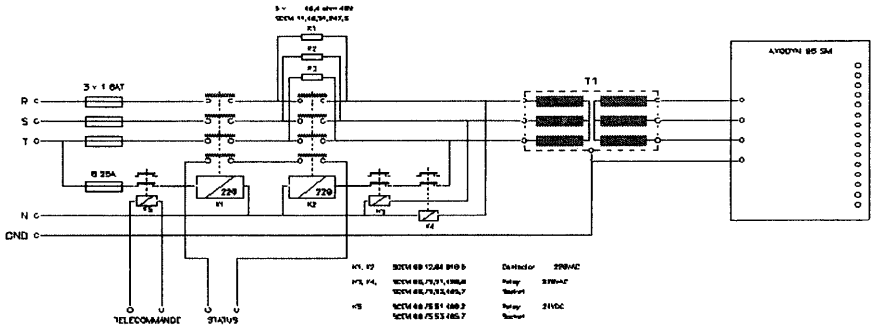
Interface box error LED

- it is ON if at least one error is reported, it is switched off again if all errors have gone away.
- Short flashing sometimes is OK, this can be the warning from the offset position compensation.

Interface box diagnostic LEMO contact

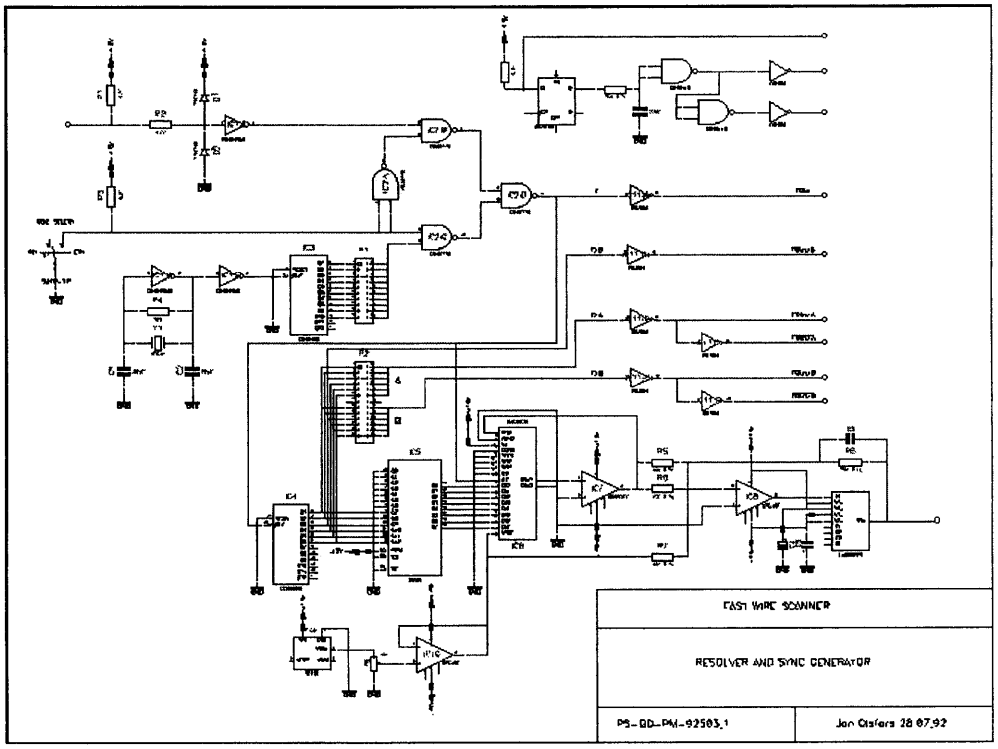
- Programmed signal on DAC channel 1, depends on version of SW.
- Gives an analog signal -5 .. +5V approx. which corresponds to the cosine amplitude (=crankshaft position) and refresh rate 128us .

Annex E

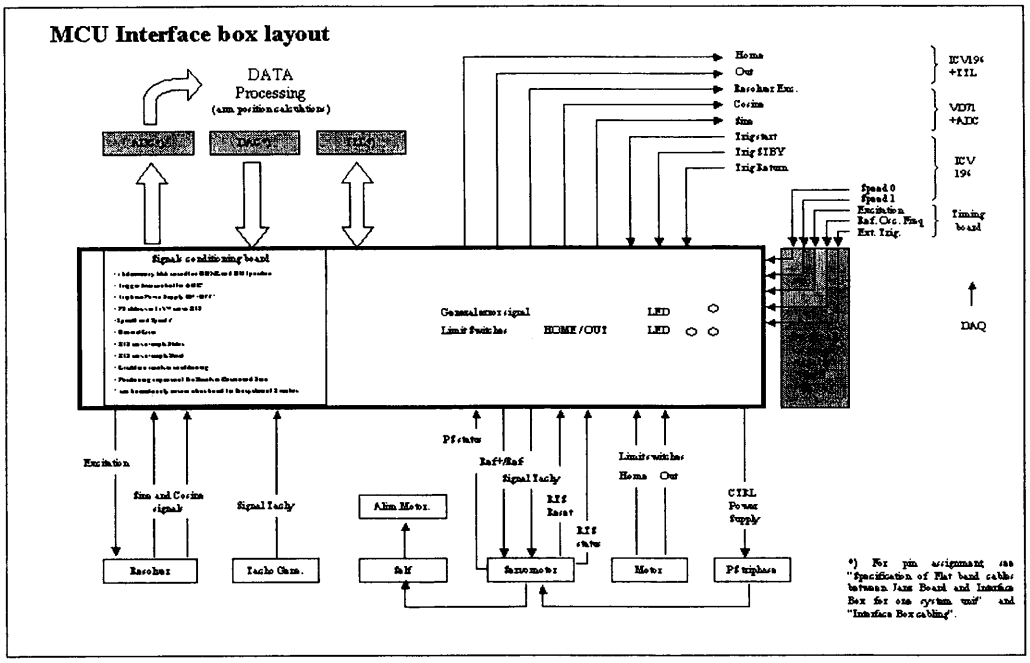


Three phases 380V power supply

Annex F



Annex G



Annex H