TMS - Training

General Overview of the TMS system

TMS hardware

TMS Software

Cycle Parameters (State/Phase tables)

FPGA Firmware packaging

Testing

Troubleshooting

TMS - Introduction

Particle trajectory measurement system for the CERN Proton Synchrotron

40 Pick-up Channels - 120 analogue channels sampled at 125MHz, 14 bits

FPGA based system processes this data in real-time

Captures and processes about 15 billion samples per second

Network Data access at approx 65 MBytes per second

256 MBytes of memory per pick-up channel

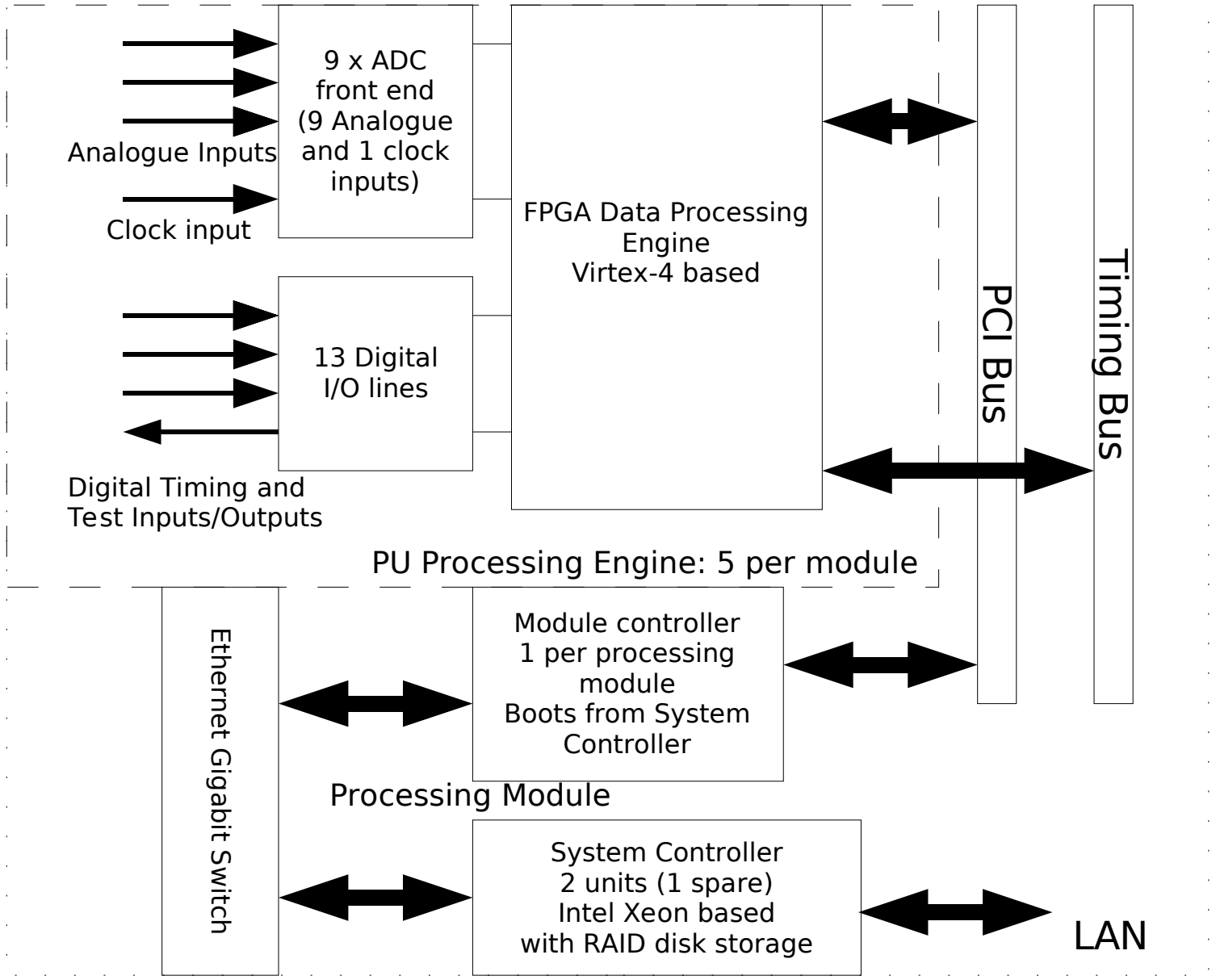
Gigabit Ethernet interface used internally and externally

Modular and scalable system

Software open source Linux based

FPGA firmware written in VHDL

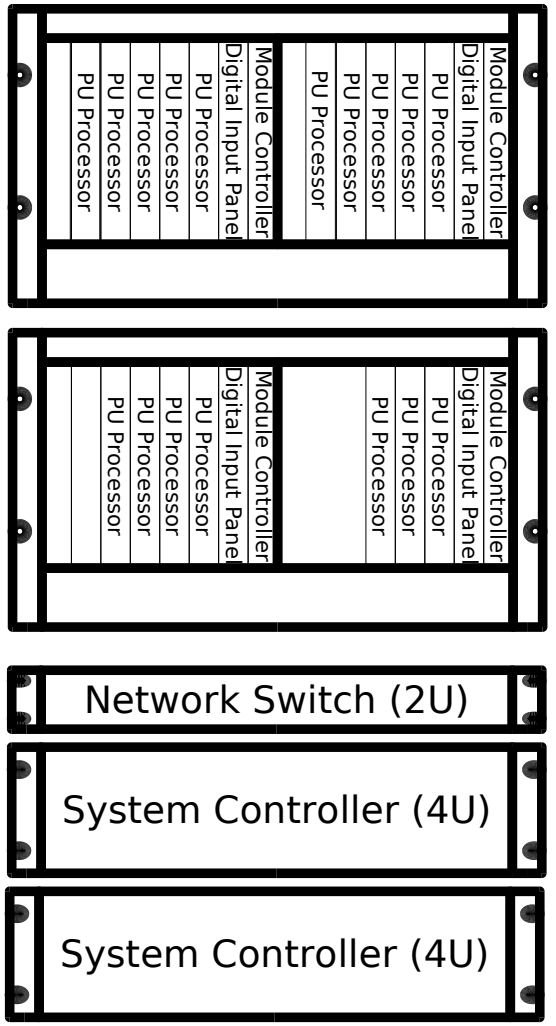
TMS Overview



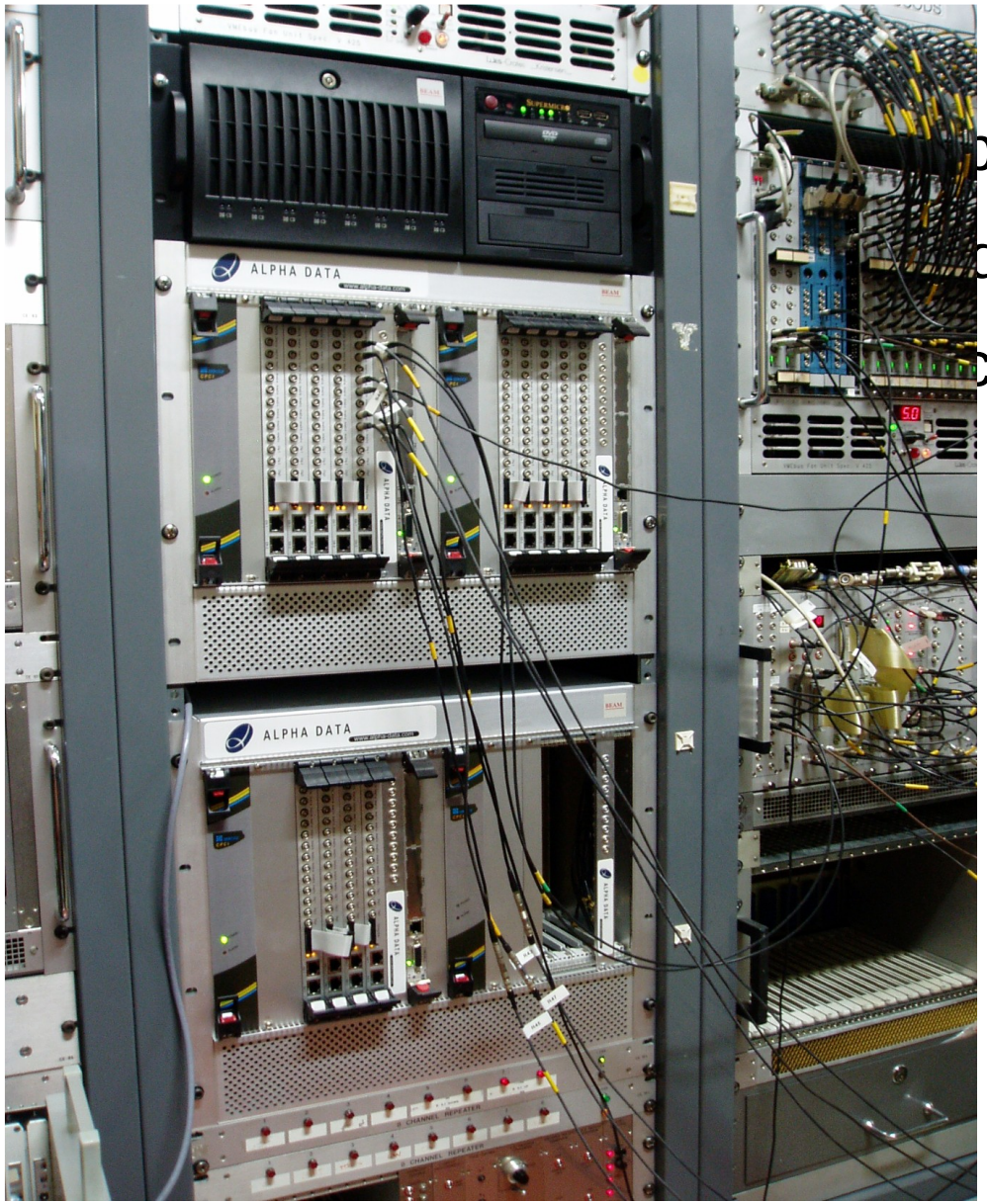
External signals

Name	Number	Description
ADC Input	120 + 33 spare	Analogue signal inputs. 2 Volts peak to peak into 50 ohms. Sampled at 125 MS/sec at 14 bits.
10 MHz system clock	17 + 3 for spares	Master system clock. The ADC's 125 Mhz sampling clock is optionally synchronised to this clock and all of the digital timing signals, except the Injection signal, will re-synchronised to this clock within each FPGA. Positive TTL into 50ohms.
FREF Input	4	Reference frequency. Positive TTL into 50ohms. (437KHz)
CYCLE_START Input	4	Start of a machine cycle. Positive TTL into 50ohms.
CYCLE_STOP Input	4	End of Last Flat Top, effectively end of cycle. Positive TTL into 50ohms.
CAL_START Input	4	Start of calibration period. Positive TTL into 50ohms.
CAL_STOP Input	4	End of calibration period. Positive TTL into 50ohms.
INJECTION Input	4	Injection. Positive TTL into 50ohms.
HCHANGE Input	4	Harmonic changes. Positive TTL into 50ohms.
Spare Inputs	4 * 3	Spare digital inputs. Positive TTL into 50ohms.
Outputs	17 * 3	Digital outputs
setNextCycle()	1	Information on the next cycle is sent in this call

TMS – Hardware



TMS – Hardware Installed in Rack



bove racks.

d rack.

connection have been made.

TMS - Hardware

System Controller – Dual Intel Xeon PC architecture system with 2 GBytes of RAM, dual Gigabit Ethernet ports and dual SATA disks in a RAID1 configuration. Boots from hard disk system. Has spare PCI slots for post processing modules or extra network ports.

Module Controller – Intel Duo Core based cPCI controller with 1GByte of RAM and 3 Gigabit Ethernet ports. Boots over Ethernet from System controller. Has PMC slots for extra post processing modules.

PUPE Boards – Xilinx Virtex-4 FX100 based, 1GByte of SDRAM, 2 Gigabit Ethernet ports, 9 x 14bit ADC's, 13 Digital I/O lines, 1 Digital clock input.

Rack systems – Power supply for cPCI systems. 4 separate 8 slot card frames, 1 card frame as a spare.

Hardware Installation and Setup

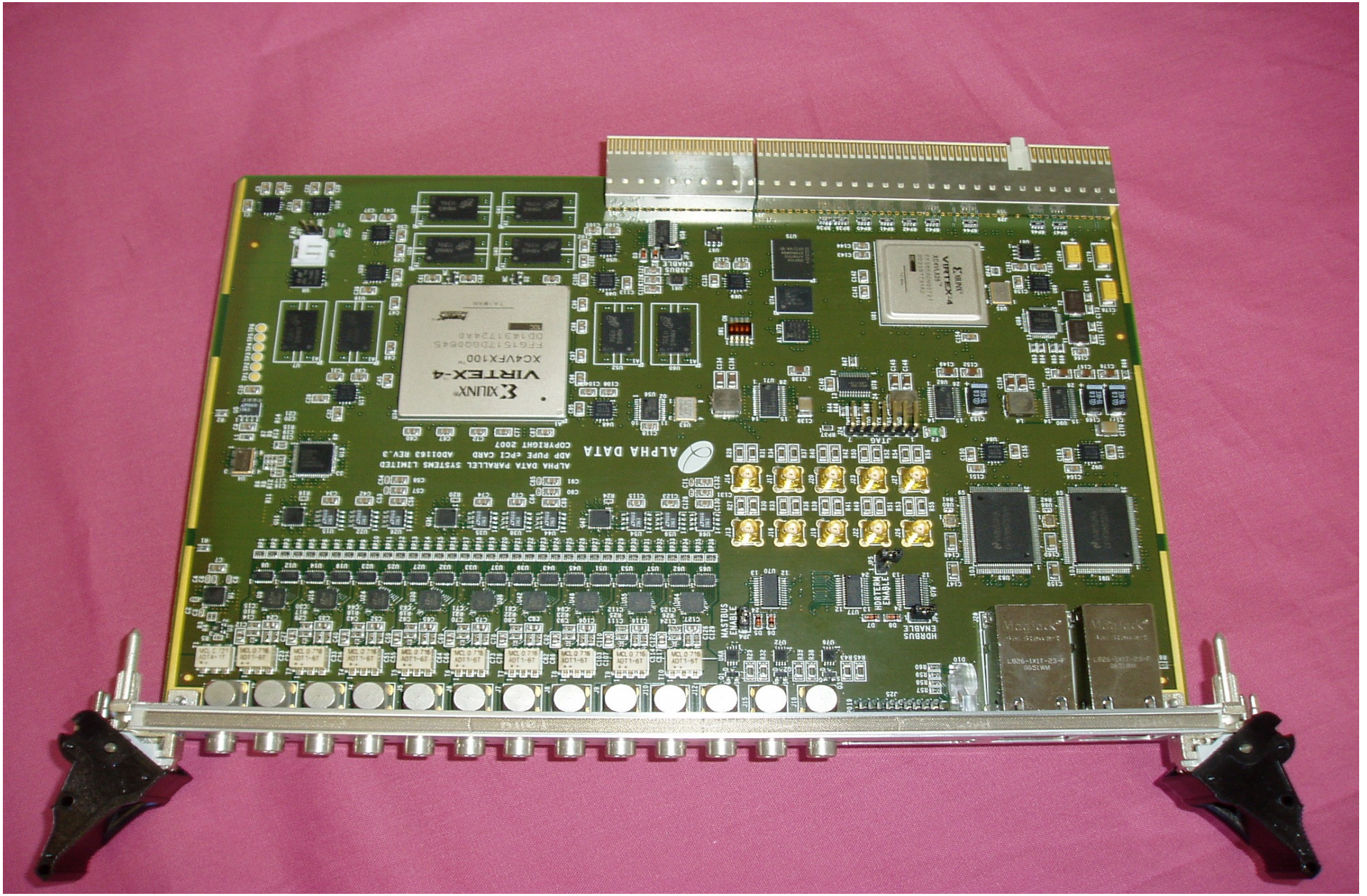
System Controller – No special set-up, BIOS at defaults. One Ethernet port to LAN and the other to the TMS private network switch.

Module Controller – Some BIOS changes, can be set-up using VGA monitor and keyboard plugged into the special breakout lead. Install in far right slots in cPCI rack with transition module at the rear. Network interface Eth2 to TMS switch.

PUPE Boards – Installed in the slots leaving a spare slot next to the power supply and a free slot next to the Module Controller to reduce ADC noise. The Master PUPE should in the far right slot. This has the extra digital timing panel installed. The timing bus cable should connect all of the PUPE's together and the PUPE farthest away from the Master PUPE should have the timing bus termination jumper installed.

TmsMaintenance manual gives full details.

PUPE Board


XILINX
VIRTEX-4
XC4VFX100
ALPHA DATA
ADP PUPE ePCI CARD
COPYRIGHT 2007
ALPHA DATA PARALLEL SYSTEMS LIMITED
JTAG
MASTBUS ENABLE
HOSTBUS ENABLE

Hardware Maintenance

Change hard disks – one every three years. The disk system runs as a RAID 1 system, so one disk can be swapped without having to re-install the software.

Change real-time clock batteries – once every 5 years. The System Controller and Module controllers have these.

Clean the airflow paths – once every 3 years.

Replace the cooling fans in the rack once every 5 years.

Replace the cooling fans in the System Controller every 5 years.

The System controllers and PUPE boards have on-board thermal monitoring. This can be used to check that the fans are working correctly.

Hardware Trouble shooting

System Controller – Spare controller can be used. The IPMI interface can be used to diagnose boot problems or a VGA Monitor and keyboard can be attached. Normal PC architecture

Module Controllers – Spare controller can be used. VGA Monitor and keyboard can be attached via breakout lead to diagnose boot problems.

PUPE Boards – Spare PUPE boards can be used. The TMS system performs basic PUPE board tests and gives status information on the power supply voltages and temperature

PUPE boards are identified by module and slot position. A spare PUPE in the spare rack can be used to replace a faulty PUPE without a power cycle.

Power Supply's – These will show a red light if there is a problem. There is a spare power supply in the system.

TMS - Software

Operating System is Linux, based on Fedora Core 6 distribution.

Module controller has a very small, network boot Linux system based on the Busybox utility.

TMS software is predominantly written in 'C++' in an object orientated style.

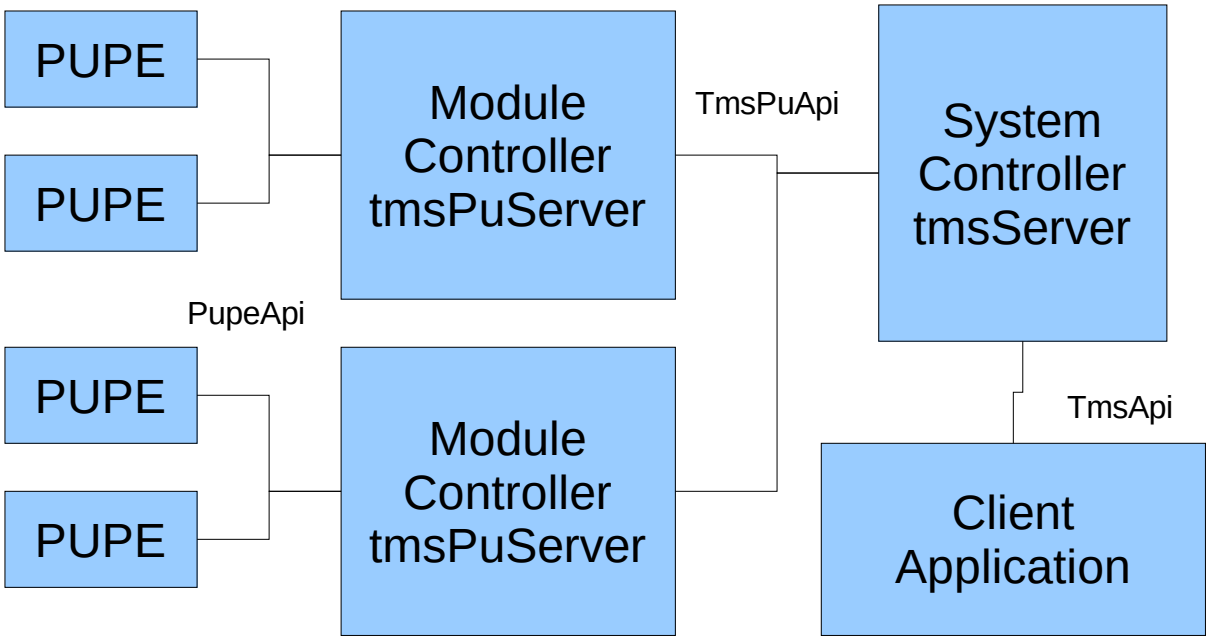
TMS software is multi-threaded.

The GNU software development tool-set is used for development.

The TMS Server has the complete development environment installed.

Two main programs: TmsServer and TmsPuServer.

TMS Software Structure



File structure

Most of the TMS software is installed in /usr/tms this has the following directories:

bin	Executable programs
include	'C++' include files for development
lib	Libraries for development
config	Configuration utilities and template configuration files
fpga	FPGA firmware
stateTables	The Cycle Parameter tables
rootfs	The master root file system for the module controllers.
rootfs-[1234]	Copies of the master root file system for the individual module controllers. These are mounted as the root file system for the module controllers.
tmsExamples	Development example code
data	Data files such as test signals
html	HTML root for the TmsWeb program
tftpboot	The module controllers boot files master. These are copied into /tftpboot/tms-mcsys
doc	Documentation on the system.

Software Configuration

The System Controllers Linux configuration is standard. Limited package installation. tmsSetup sets-up default Network config.

The supplied install DVD has a Kickstart file to install and configure a system from bare metal. Only the network information and Cycle Parameter information needs to be configured.

Users can be added to the system if required.

The TMS software is configured with 5 main files.

/etc/tmsServer.conf – This configures the master TmsServer program.

/usr/tms/rootfs-[1234]/etc/tmsPuServer.conf – These configure the individual TmsPuServer programs running on the module controllers.

The Cycle parameters data is in /usr/tms/stateTables

TmsServer.conf

Parameter	***Default***	***Description***
TmsServer:	tmssc.tmsnet	The BOAP Servers host name. Normally the System Controllers host name on the TmsNet.
SptDir:	/usr/tms/stateTables	The directory where there State/Phase table library is stored.
SimulateData:	0	If set to 1, the TmsServer will simulate data capture internally. This is useful for debug whith using any TmsPuServer servers and thus any PUPE engine boards.
SimulateNextCycle:	0	If set to 1 the TmsServer will call the setNextCycle() call on each CYCLE_STOP event.
DefaultCycleType:	Beam3	This is the type of cycle that will be set on start-up. It defines which State/Phase tables will be loaded initially.
AdcSysclkSync:	0	Sets the ADC clock to be synchronised with the SYSCLK timing clock
PuServer1:	1	The is the number of the first TmsPuServer
PuServer2:	2	The is the number of the second TmsPuServer
PuServer3:	3	The is the number of the third TmsPuServer
PuServer4:	4	The is the number of the forth TmsPuServer
PickUp*:	1,1,1	This is the logical to physical pick-up table configuration. It is overwritten on configure() API calls. The values are: ModuleNum, PupeNum and PupeChan.

TmsPuServer.conf

Parameter	***Default***	***Description***
TmsServer:	tmssc.tmsnet	The BOAP Servers host name. Normally the System Controllers host name on the TmsNet.
ModuleControllerNumber:	1	The number of the Module Controller
SimulateFpga:	0	If set to 1, the TmsPuServer will simulate the PUPE FPGA boards internally. This is useful for debug without using any PUPE engine boards.
SimulateTiming:	0x00	Simulate Timing signals in software. Bit mask (0xFF all timing signals)
FpgaFirmwareFile:	/usr/tms/fpga/tms-fpga.bit	This is the path name for the FPGA bit file to use for the PUPE boards.
FpgaLclk	50	The is the PUPE LCLK frequency to use
FpgaMclk:	125	The is the PUPE MCLK frequency to use
PupeNumber:	5	Defines the number of PUPE boards
PupeMaster:	5	Defines the PUPE board that has the master timing inputs
PupePhysicalOn:	1	Use physical slot locations
PupePhysicalDevices:	10,11,12,13,14	The list of PCI device numbers

Module Controller Setup

The Module controller boots from the System Controller.

The kernel is loaded from /tftpboot/tms-mcsys using TFTP

The root file system is in: /usr/tms/rootfs-[1234]

There is a utility named “tmsSetupModuleController”. This will copy the template /usr/tms/rootfs into the appropriate root file system directory and optionally add the Ethernet MAC address to the System Controllers /etc/dhcpd.conf file.

It takes the module controller number as the first argument and optionally the Ethernet MAC address as the second argument.

Note that a default tmsPuServer.conf file will be installed from /usr/tms/rootfs. This will probably need editing, especially for Module Controller 3 and 4.

Module Controllers System

Fetches network information using DHCP from system controller.

Fetches the Linux kernel and initial RAM disk root file system using TFTP from system controller's /tftpboot/tms-mcsys.

Mounts the /usr/tms/rootfs[1234] file system read only using NFS from system controller.

Mounts local RAM file systems on /tmp and other appropriate places. Startup file /etc/init.d/rcS

Mounts the /usr/tms directory read only using NFS from system controller.

Mounts the /data directory read/write using NFS from system controller.

Busybox is used to implement most of the system programs and files as well as some shared libraries from a Fedora Core 6 system.

Software maintenance

The Linux system needs no normal system maintenance.

The TMS configuration files and Cycle Parameter files need to be backed up. The TMS script program “tmsBackup” will create a simple tar archive of the TMS configuration and Cycle Parameter files.

Any additional files added to the system, such as user's home directories need to be backed up.

All software is packaged as RPM packages, including the TMS software. This allows the “yum” and “rpm” tools to manage updating individual software packages.

System log messages in /var/log/messages.

TMS Software Interfacing

The TMS system implements a TCP/IP socket based RPC interface for control and data access.

The RPC system is based on the BEAM BOAP object access system. This implements an efficient binary communications system for performance.

A 'C++' API library, libTms, is provided that can be ported to different systems.

Two separate object interfaces: TmsControl and TmsService.

Multiple clients can access the system simultaneously.

Also implements an asynchronous event system.

Data Client applications using the TMS API

The TmsService object provides a few simple RPC functions.

Provides information functions for a given cycle type or an individual cycle number.

Returns data from the system.

Supports raw pick-up data (integrated per bunch).

Supports averaged pick-up data (integrated per ms)

Performance is around 65 MBytes/second across a Gigabit network interface.

Data comes from the PUPE memory. This is sufficient for 2 to 3 PS cycles worth of data.

Data bandwidth restricts the amount of data that can be returned to the user.

The main function call is: getData(DataInfo info, Data& data)

TMS GetData call

Field	*Description*
cycleNumber	The PS Cycle number to fetch data from.
channel	The pick-up channel number.
cyclePeriod	The cycle period the data to fetch data from.
startTime	The start time in milli-seconds from the start of the required Cycle Period.
orbitNumber	The starting orbit number (starting from 0).
bunchNumber	The bunch number (starting from 1 (0 is all bunches)).
function	The data processing function to perform or performed.
argument	The Argument to the data processing function.
numValues	The total number of data points to return.

Function	*Description*
DataFunctionRaw	The raw Sigma,DeltaX,DeltaY integrated data
DataFunctionMean	The mean Sigma, DeltaX, DeltaY integrated data over 1Ms sample periods. The mean values are available for all bunches on all channels.
DataFunctionMeanAll	The overal mean Sigma, DeltaX, DeltaY integrated data over 1Ms sample periods for all bunchens. The mean values are available for all channels.
DataFunctionMean0	The mean Sigma,DeltaX,DeltaY integrated data. 1Ms sample period for all bunches although this is programmable. This function is depreciated in favour of the new DataFunctionMeanAll function.

TMS GetCycleInformation call

getCycleInformation (UInt32 cycleNumber, CycleInformation &cycleInformation)

CycleInformation	
cycleNumber	The PS Cycle number
cycleType	The Cycle Type Name
BList< CycleInformationPeriod >	The list of cycle periods

CycleInformationPeriod	
cyclePeriod	The Cycle Period
startTime	The start time in ms
endTime	The end time in ms
harmonic	Machines harmonic number
numBunches	The number of bunches
bunchMask	Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc
numValues	The total number of raw data values available

TMS GetCycleTypeInformation call

getCycleTypeInformation (BString cycleType, CycleTypeInformation &cycleTypeInformation)

CycleTypeInformation	
cycleType	The Cycle Type Name
info	Information string on this cycle type
BList<CycleTypeInformationPeriod>	The list of cycle periods

CycleTypeInformationPeriod	
cyclePeriod	The Cycle Period
harmonic	Machines harmonic number
numBunches	The number of bunches
bunchMask	Bitmask defining which buckets the bunches are captured from. Bit 0 is bucket 1, bit 1 is bucket 2 etc

Control client applications using the TMS API

SetNextCycle call needed to provide the TMS system with information on the next PS cycle. Needs to be called at least 30ms before the START_CYCLE event. Best time at CYCLE_STOP

Cycle Parameter table management

Diagnostics

System Testing

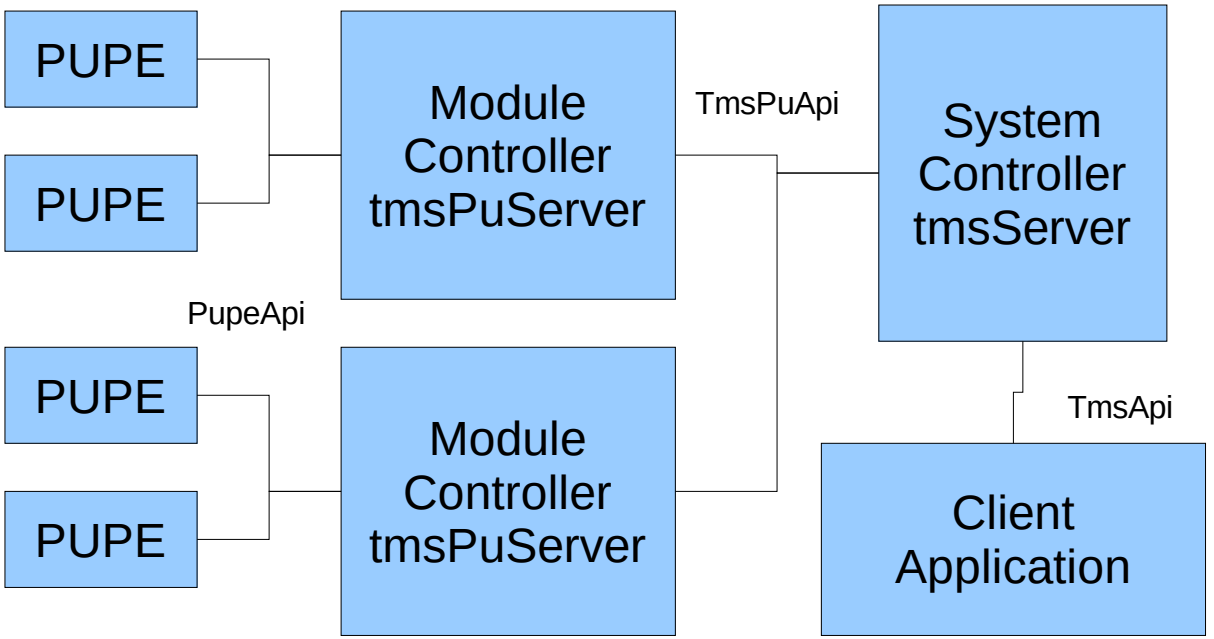
System Status

System Statistics

TMS Software Time Line

Event	Description
CYCLE_STOP	Master PUPE generates an interrupt.
	TmsPuServer loads the next set of CycleParameters if they are available
setNextCycle()	The CERN software will send the next cycle number and type information to the TmsServer program. This could occur any-time from CYCLE_Start to within 30ms of the CYCLE_START it refers to.
	The TmsServer programs sends the setNextCycle information to each of the TmsPuServers. The TmsPuServer programs load the CycleParameters into each PUPE.
CYCLE_START	All PUPE's start processing the cycle.
ErrorEvent	If the PUPE detects an error it will issue an error interrupt and will abort processing the cycle. The tmsPuServer program responds to the error by sending an Error event to the TmsServer program. The TmsServer program will send the error event to all clients that have registers an Error interface object and will store the error message with the cycles state information. This will be returned in any getData() requests.
CAL_START, INJECTION, H_CHANGE	The PUPE's respond to these events as required.
CYCLE_STOP	Master PUPE generates an interrupt.
	TmsPuServer loads the next set of CycleParameters if they are available
	The master TmsPuServer sends the cycleStop event to the TmsServer which wakes up and client threads awaiting data for the cycle.

TMS Software Structure



TMS Errors

Error	*Description*
ErrorOk	No Error. This is the status returned when the command completed with no errors.
ErrorMisc	A miscellaneous unclassified error occurred.
ErrorWarning	A warning message. No actual error occurred.
ErrorInit	An error occurred during initialisation of the system.
ErrorConfig	There is an error in the system configuration files.
ErrorParam	There was an error in one of the parameters passing in an API call.
ErrorNotImplemented	This function has not been implemented.
ErrorComms	A communication error occurred.
ErrorCommsTimeout	A communications time out occurred.
ErrorMC	A Module Controller has an error
ErrorFpga	There is an error with a PUPE FPGA board.
ErrorStateTable	An error event occurred due to an incorrect FPGA State table transition.
ErrorCycleNumber	The Cycle Number and Type was not updated in-time for this cycle.
ErrorDataNotAvailable	The required data is not available. This means that there is no data for the given cycle number and/or period requested.
ErrorDataGone	The required data has already been overwritten by new data. This means the client was too slow in fetching the data of the TMS system was heavily loaded and could not supply the data before it had gone from the PUPE data memory.
ErrorDataFuture	The required data is to far into the future. This means that the cycle number requested is too far into the future.

TMS Web Access

The TMS Server presents a simple web access system

Able to view system status

Able to view statistics

Able to get data from the system

Able to display simple graphs.

Could be easily extended.

TMS User Programs

tmsControl: Command line control application.

tmsControlGui: GUI Command line application

tmsTestData: Test application

“tmsTestData -simdata”

“tmsTestData -test all -check -cont”

tmsStateGen: Cycle parameter generator

tmsSigGen: Test signal generator

tmsRestart: Restarts tmsServer and tmsPuServer programs

TMS internal software development

All of the TMS software is available in source code form.

The TMS software can be built on the TMS system or another Fedora Core 6 Linux system.

Split into the following Modules:

- Tms – The main system software
- Tms-sys – System Controller configuration
- Tms-mcsys – Module Controller system
- Tms-fpga – FPGA firmware
- Tms-doc – system documentation

The tms-full-src-<VERSION>.tar.gz archive contains the full source code.

TMS Software Libraries

libBeam.a: BEAM Object API. Includes basic String, List and Array classes as well as the BOAP RPC system.

libBDebug.a: BEAM Debug utilities include crash backtrace system.

libTms.a: Main TMS API library. Includes client and server side BOAP interface objects as well as CycleParameter table generation and management classes.

libadmxrc2.so: Alpha Data ADMXRC interface library. Uses the admxrc2 Linux kernel driver to communicate with the PUPE boards.

Building and packaging main software

Uses the make system

Overall make configuration in “Makefile.config”. Includes version number.

- “make clean” - Cleans the software tree
- “make” - Builds the software tree
- “make rpm” - Builds the RPM package
- “make rpmInstall” - Installs the RPM's in the packages directory

You can also use make install, as root, to install the software directly without packaging it first.

Note tms-mcsys uses the kernel and drivers (including Admxrc2) from the build system.

TMS Cycle Parameter Tables

Contains FPGA state and phase table parameters describing a complete PS machine cycle.

Information on what to do on events and internal PLL tables.

Cycle is split into Cycle Periods: Calibration, Event0, Event1, ...

The FPGA hardware can acquire a complete set of data for a cycle with no software intervention.

The TMS Server keeps a library of Cycle Parameter tables in ASCII files indexed by a cycle type string.

The Cycle Parameter tables are passed to all Module controllers which store there contents in internal data structures. The same information is sent to all PUPE channels.

The Cycle Parameters are loaded into the FPGA's on the CYCLE_STOP event or on the setNextCycle() call whichever is later.

FPGA State tables

Set of 16 possible states

Change of state on timing event or a delay of 16 FREF periods.

Set of control bits for each state

State 14 is error state – interrupt generated

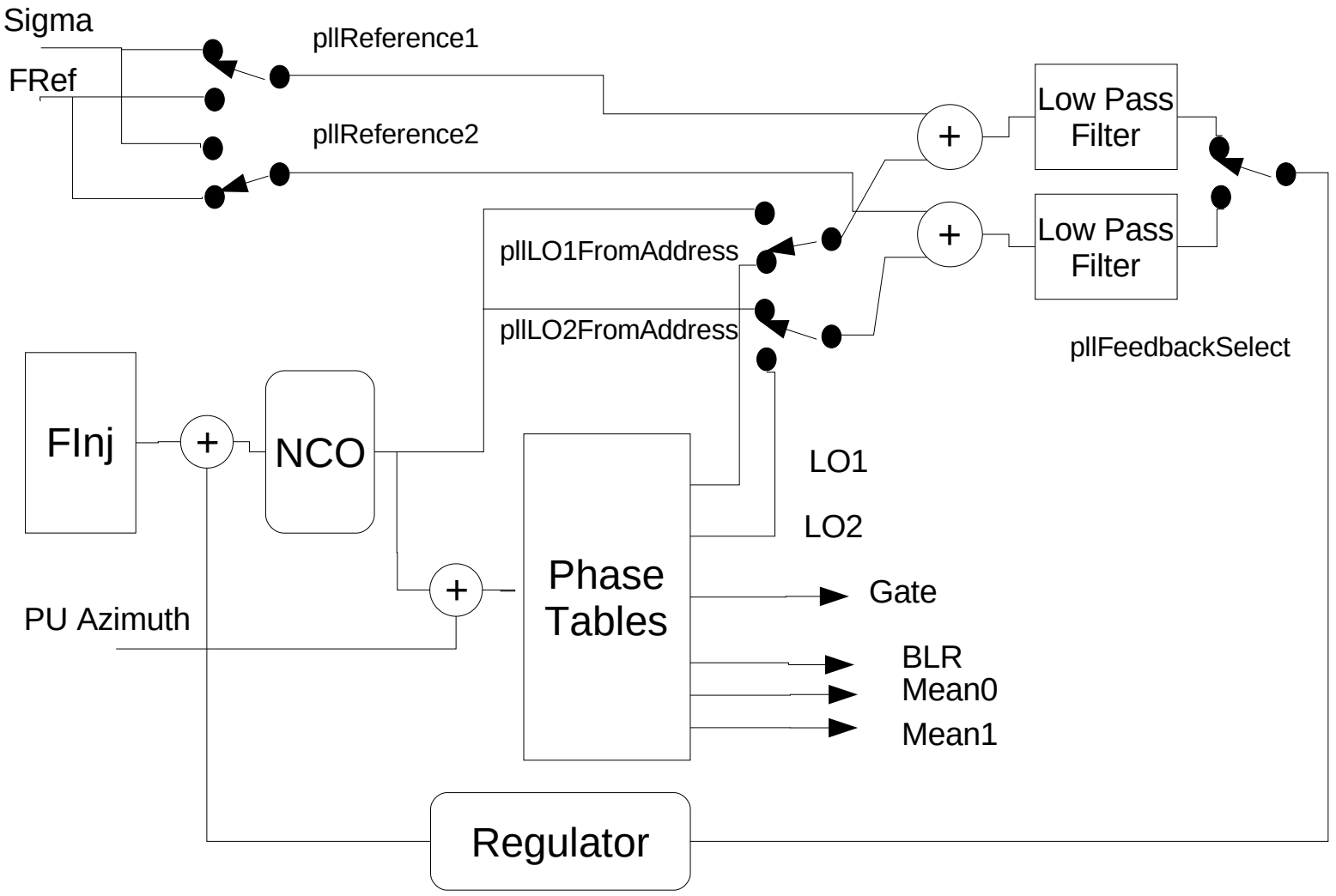
State 15 is stopped state

Each state has a separate phase table (512 bytes)

TMS State Tables – Control bits

Name	***Bits***	***Description***
acquireData	0	The system will perform data acquisition if this bit is set. This will allow entries to be made in the CycleTimingTable and in the CycleDataTable if appropriate GATE strobes are present in the PhaseTable for this state.
pllReference1	1	Selects the PLL source for the Filter 1 path. 0 – selects FREF, 1 – selects Sigma.
pllReference2	2	Selects the PLL source for the Filter 2 path. 0 – selects FREF, 1 – selects Sigma.
pllFeedbackSelect	3	Selects which of the Filter outputs to be used for the PLL error value. 0 – selects filter 1, 1 – selects filter 2.
pllLO1FromAddress	4	This selects which PLL signal is to be used as the PLL internally generated FREF for the Filter 1 path. 0 selects the LO1 phaseTable bit, 1 – selects the MSB of the PLL's phase counter.
pllLO2FromAddress	5	This selects which PLL signal is to be used as the PLL internally generated FREF for the Filter 2 path. 0 selects the LO2 phaseTable bit, 1 – selects the MSB of the PLL's phase counter.

State/Phase Table Switching



FPGA State Tables - Events

Name	***Bits***	***Description***
cycleStop	11:8	This defines which state to move to when a CYCLE_STOP event occurs.
calStop	15:12	This defines which state to move to when a CAL_STOP event occurs.
calStart	19:16	This defines which state to move to when a CAL_START event occurs.
injection	23:20	This defines which state to move to when a INJECTION event occurs.
hchange	27:24	This defines which state to move to when a HCHANGE event occurs.
delay	31:28	This defines which state to move to 16 FREF periods later. It can be used to add a delay to the state/phase table switch or add a section of different state/phase table settings for a 16 FREF period after an event.

FPGA Phase tables

- Set of 16 Phase tables, one for each state.
- Each phase table has 512 byte wide entries
- Two separate PLL local oscillator generation tables, LO1 and LO2
- Note each pulse should be at least two clock cycles long due to PLL operation.

Name	***Bits***	***Description***
lo1	0	The LO1 PLL signal used in the filter 1 feedback path.
blr	1	The base line restoration signal. When high BLR is being calculated
gate	2	The gate signal used to acquire data.
lo2	3	The LO2 PLL signal used in the filter 2 feedback path.
meanFilter1	6	A pulse which sends the current integral values into the bunch mean filter 1. Typically used to return integral values every ms for all particle bunches.
meanFilter2	7	A pulse which sends the current integral values into the bunch mean filter 2. Typically used to return integral values every ms for the first particle bunch.

How to generate Cycle Parameter's

Cycle Parameters are stored in ASCII files on the TMS Server.

New or updated sets of parameters can be uploaded using the TMS API. The TmsServer program will send these out to all module controllers.

The tmsControl and tmsControlGui programs can read an ASCII Cycle Parameter file and send it to the TMS System using the TMS API. So the user can generate a set of Cycle Parameters in an ASCII file and upload this to the TMS server.

The tmsControlGui program has a simple high level Cycle Parameter editor built in.

The tmsStateGen program creates a set of simple test Cycle Parameter's. It can be extended to support other Cycle types.

The TMS API library has support for reading and writing the Cycle Parameter files and creating them based on a high level definition.

TMS FPGA Firmware

The TMS FPGA Firmware is packaged in RPM format

Written in VHDL

Build environment is Xilinx Foundation Express

To package:

- Build bit file. Make sure internal version number is updated.
- Copy bit file to tms-fpga directory with appropriate version number
- Symbolically link this bit file with tms-fpga.bit
- Run “make rpm” to build package
- Run “make rpmInstall” to install the package in packages

Make sure bit file does not cause overheating of the FPGA

The cern-tms-rel_<VERSION>.zip archive contains the source code.

FPGA VHDL Synthesis

Synthesis is achieved using ISE tools (9.2)

Build uses make file (makefile_tms_pupe or makefile_tms_pupe_inc)

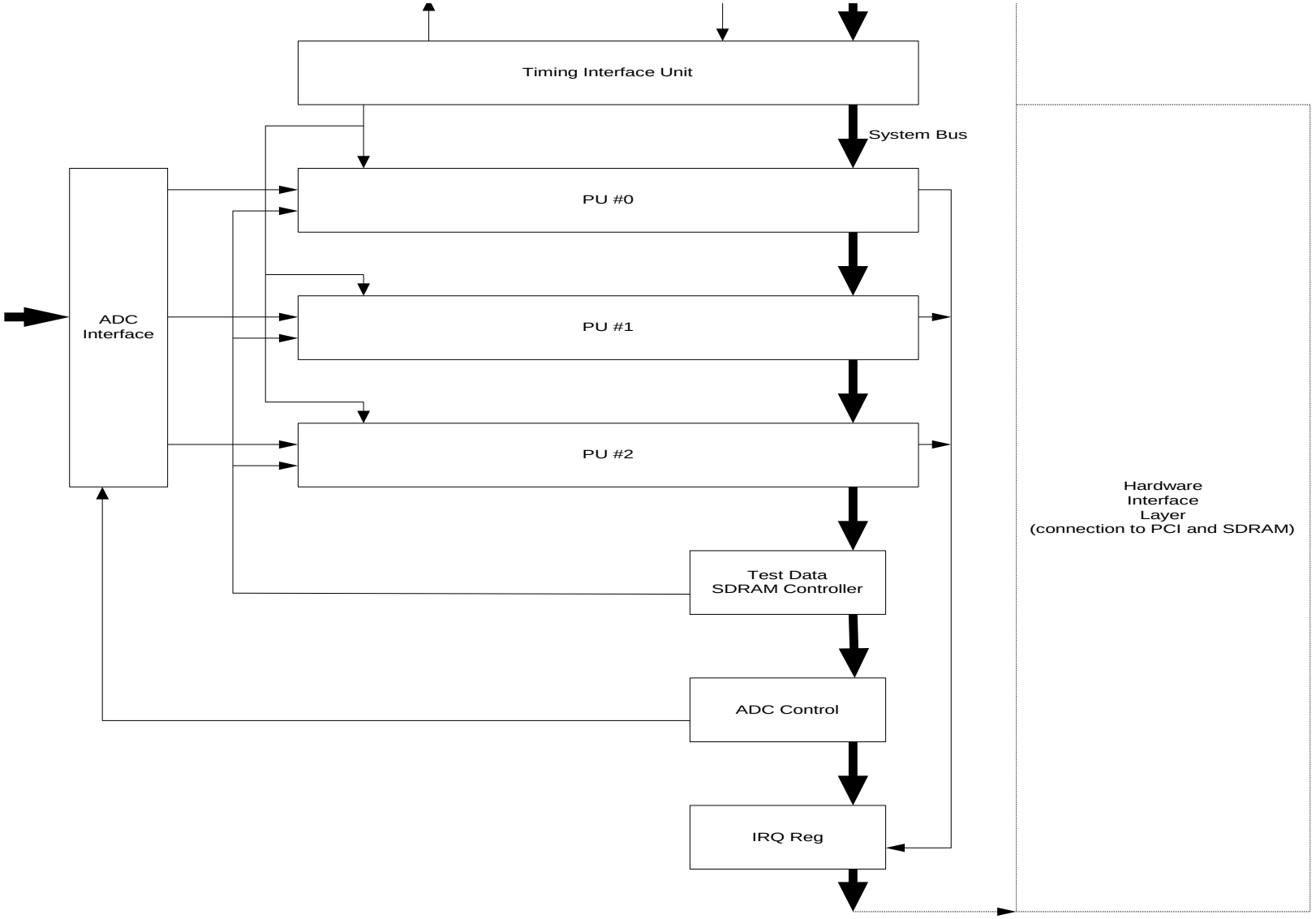
The XST project file and build scripts are: tms_pupe_xst.scr and tms_pupe_xst.prj

The project can be built from the command line using nmake -f makefile_tms_pupe

Synthesis scripts and files are located in the synthesis directory. The UCF file contains placement constraints for the SDRAM controller block memories. Without these constraints memory read errors can occur.

3 main directories: tms-processing contains the signal processing code core, pupe-wrapper contains the interfacing code, ddr2_memory_interface contains the DDR2 memory interfaces.

FPGA Block Diagram



FPGA PupeAPI

Register and shared memory Interface

4 MByte PCI window

System Control Registers: controlling the memory paging and interrupts

Application Registers: for use by the PUPE modules

2 MB Memory window: for accessing SDRAM or Block RAM banks

FPGA Memory Map

Address	Name	R/W	Description
0x000000	IMEM_REG	RW	Block RAM Index
0x000008	LOCKED	RO	DCM Locked Status
0x000010	ADDR_REG	RW	PAGE Register for SDRAM access
0x000018	MEM_REG	RW	SDRAM Bank Select Register
0x000020	IER	RW	Interrupt Enable Register
0x000028	ISR	RW	Interrupt Status Register
0x000030	MEM_RAS	RW	SDRAM Read Address Scaling
0x000038	MEM_RAO	RW	SDRAM Read Address Offset
0x000040 -0x000058	MEM_GNTx	WO	Memory Grant Registers
0x000800 -0x0009FF	Application Registers	RW	See Application Memory map
0x200000 -0x3FFFF	Memory Window	RW	2MB window for accessing SDRAM or Block RAM

FPGA Application Registers

Address	Name	R/W	Description
0x000800	FIRMWARE	RO	Firmware ID code “CN” + version numbers
0x000808	ADC	RW	ADC Control Register
0x000810	TIMING_IO	RW	Timing I/O Register
0x000818	TESTCTRL	RW	Test Data Control Register
0x000820	TESTLEN	RW	Test Data Pattern Length

Address	Name	R/W	Description
0x000880	CONTROL	RW	PU General Control and Status register
0x000888	CYCLE	RW	Cycle number
0x000890	TIME	RO	Time in ms from start of cycle
0x000898	TIME_TBLADDR	RO	Last write address in timing table
0x0008A0	PLL_FREQUENCY	RW	PLL Reference orbit frequency
0x0008A8	PLL_FREQDELAY	RW	PLL frequency load delay
0x0008B0	PLL_PHASEDELAY	RW	PLL phase delay
0x0008B8	PLL_GAIN	RW	PLL gain
0x0008C0	DDS_FREQ_MIN	RW	PLL DDS minimum frequency
0x0008C8	DDS_FREQ_MAX	RW	PLL DDS maximum frequency
0x0008D0	DIAG_CTRL	RW	Diagnostics Control/Status
0x0008D8	DIAG_TRIGGER	RW	Diagnostics Trigger
0x0008E0	DIAG_DELAY	RW	Diagnostics Capture Delay
0x0008E8	TEST	RW	Timing Test

FPGA Shared Memory

SDRAM BANKS

Bank	Addresses	Function
0	0x0000000-0xFDFFFFF	PU #0 Cycle Data Table
0	0xFE00000-0xFFFFFFF	PU #0 Bunch Mean Tables
1	0x0000000-0xFDFFFFF	PU #1 Cycle Data Table
1	0xFE00000-0xFFFFFFF	PU #1 Bunch Mean Tables
2	0x0000000-0xFDFFFFF	PU #2 Cycle Data Table
2	0xFE00000-0xFFFFFFF	PU #3 Bunch Mean Tables
3	0x0000000-0xFFFFFFF	Test Pattern Buffer

BLOCK RAM BANKS

Bank	Size (min 2kB)	PU	Bit Width	Function
0	32kB	0	64	Cycle Timing Table
1	2kB	0	64	Cycle Information Table
2	2kB	0	8	Timing Phase Table
3	2kB	0	32	Timing Switch Table
4	8kB	0	64	Diagnostics table
5	8kB	0	64	Bunch Mean Table #0
6	8kB	0	64	Bunch Mean Table #1

FPGA Interrupts

Bit	Function
0	PU #0 CYCLE_START
1	PU #0 CYCLE_STOP
2	PU #0 ERROR
3	PU #0 DIAGNOSTIC INFO CAPTURED
4	PU #0 SDRAM Write FIFO Half Full
5-7	PU #0 User Interrupts (Switch Table bits 5-7)
8	PU #1 CYCLE_START
9	PU #1 CYCLE_STOP
10	PU #1 ERROR
11	PU #1 DIAGNOSTIC INFO CAPTURED
12	PU #1 SDRAM Write FIFO Half Full
13-15	PU #1 User Interrupts (Switch Table bits 5-7)
16	PU #2 CYCLE_START
17	PU #2 CYCLE_STOP
18	PU #2 ERROR
19	PU #2 DIAGNOSTIC INFO CAPTURED
20	PU #2 SDRAM Write FIFO Half Full
21-23	PU #2 User Interrupts (Switch Table bits 5-7)

TMS Testing

The system has a number of features for testing:

- Internal API's test function can to perform a basic system test.
- Diagnostics API functions to capture important internal FPGA signals.
- Software simulation of timing signals.
- Generation of Sigma, DeltaX, DeltaY and FRef input signals from SDRAM based signal generator.
- Messages in /var/log/messages.
- Low level program debug arguments.
- TMS test signal generator.

Testing using simulated timing and data

Useful test method when no live data is available.

Timing signals generated in software and applied to Master PUPE in each sub-rack.

Data source comes from PUPE SDRAM instead of ADC input.

Can be setup manually using the tmsControlGui's “Pupe Simulation” tab.

The tmsTestData's “-simdata” option allows the complete system to be set to simulation mode.

Note that the “SimulateNextCycle:” field in the /etc/tmsServer.conf file needs to be set to 1 to simulate the setNextCycle call. External systems should be disabled from calling this call.

The tmsTestData program can be used to run soak tests on the data. “tmsTestData -test all -check -cont localhost”

Testing on a live system

The Test API function can use used to perform an overall test.

The Status API function can be used to find the status of the system, including the voltages and temperatures of the PUPE FPGA boards.

The Statistics API function keeps a count of errors that have occurred.

The Diagnostics capture function can be used to look at the internal FPGA signals.

The /var/log/messages log can be viewed for any warning or error messages.

The API's errorEvent can be monitored for errors.

TMS Trouble shooting

Hardware trouble shooting has been given earlier

Use the systems Test API function via the tmsControlGui program or the web interface.

If the systems Test API function is not working check that the “tmsServer” program is running on the system controller and that the “tmsPuServer” programs are running on the module controllers.

Check the /var/log/messages file for any errors listed.

The system can be restarted using the “tmsRestart” command.

If the “tmsServer” or any “tmsPuServer” programs crash, a backtrace will be listed in the /var/log/messages file.

The tmsServer or tmsPuServer programs can be started manually using the “-f” and “-d 0x03” flags to run them in the foreground and display debug messages. See the manuals on the programs for the “-d” options available.

Multiple System Controller Support

Two TmsServers: 192.168.100.1 and 192.168.100.2

Both systems can be setup identically apart from:

- DHCP disabled on second server
- TmsServer disabled on second server
- tmsServer.conf and tmsPuServer.conf files set to use second server.

Second Server could be configured to manage spare module and 3 PUPE boards.

- Server would supply DHCP information for spare module controller.
- tmsServer.conf and tmsPuServer.conf files set to use second server.

TMS Future Development

Improving the FPGA algorithms including: BLR, PLL.

Improving Software functionality.

Adding data post processing algorithms.

Adding support for linked PLL's taking Sigma from three channels.

Adding extra software API functions.

Increasing data bandwidth by using PUPE Ethernet interfaces.

Adding FPGA based post processing.

Could allow both System Controllers to function with automatic handover.

Could reduce ADC noise. Internal panels. Move power supplies.

TMS – Further Information

Further information is available on the support website at: http://portal.beam.ltd/support/cern