21

KEK Preprint 95-134
October 1995
H

A GUI Approach for Control of the BELLE Silicon Detector Data-acquisition System

T. KORHONEN, C. FUKUNAGA and M. TANAKA

50360

Submitted to Nucl. Instrum. Meth., A.

National Laboratory for High Energy Physics, 1995

KEK Reports are available from:

Technical Information & Library
National Laboratory for High Energy Physics
1-1 Oho, Tsukuba-shi
Ibaraki-ken, 305
JAPAN

Phone: 0298-64-5136
Telex: 3652-534 (Domestic)
Fax: (0)3652-534 (International)
Cable: 0298-64-4604
E-mail: KEK OHO
Library@kekvox.kek.jp (Internet Address)

A GUI approach for control of the BELLE silicon detector data-acquisition system

Timo Korhonen

*Research Institute for High Energy Physics in Finland(SEFT)
Siltavuorenpenger 20C, SF-00170, Helsinki, Finland*

Chikara Fukunaga

Tokyo Metropolitan University

1-1 Minami-Ohsawa, Hachioji, Tokyo, 192-03 Japan

Manobu Tanaka *

National Laboratory for High Energy Physics (KEK)

1-1 Oho, Tsukuba, Ibaraki, 305 Japan

Abstract

A graphical user interface (GUI) based control system is being built for the silicon microvertex detector (SVD) readout for the BELLE experiment at KEK. The readout system is based on a processor(Transputer) network, and is controlled through a TCP/IP connection. The control system gives users an integrated system maintenance and run control environment. The system maintenance part gives operators a user-friendly method to modify the large system and troubleshooting tools. The software has been built using a platform independent development library, and is portable between a wide range of platforms, including Unix workstations with Motif or OpenLook and Microsoft Windows.

1 Introduction

The silicon microvertex detector (SVD) data acquisition (DAQ) system for the BELLE experiment at KEK[1] is based on a Transputer network [2]. The system comprises

*corresponding author:E-mail address is tanakam@kekvox.kek.jp

several tens of processors (Transputers) that have to be downloaded, monitored and supervised under centralized control via an SVD system controller (a VME cpu-board) using a TCP/IP connection[3]. Figure 1 shows a conceptual diagram of the SVD DAQ system with emphasis on the control workstation network.

The SVD control system is an integrated package of management tools for system maintenance and run control tools for system operation. The management part includes troubleshooting tools and those necessary to keep track of any hardware modifications, and to change the software configuration accordingly. The run control part provides a visual interface for system operation. The system and operator interface have to be designed to provide a user friendly system for a large collaboration of users having a variable level of expertise. In order to provide a comfortable and misoperation free system from a routine work operator to a detector expert uniformly, we intend to use a GUI operation for total control of the system for the build-up of the hardware system, software configuration on every processor element, system maintenance and operating the system(run control). We aim all operations to be based on point-and-click operation with a mouse in a multi window environment.

In this report we discuss the technical aspects and benefits of introducing GUI to a DAQ system. While there are already many high energy experiments which have adopted graphical human interfaces for their data taking operation all over the world, there will not yet so many examples which have applied graphical operation in their DAQ system for even system construction, diagnostics and maintenance parts. We report on such an example based on our experience involving the DAQ system construction, which is based on a Transputer network.

2 Building Blocks of the System

2.1 Transputer network

A Transputer is a RISC chip developed by INMOS. The performance of the Transputer and its applications to a wide range of engineering fields can be found in [2] and further references therein. The most characteristic feature of the Transputer is that it has four serial interfaces. The interface is called link. Through this link a Transputer can communicate with other Transputers. The data transfer rate of the link is about 1.8 MByte/sec. Through this link a Transputer can communicate with other Transputers. The Transputers can therefore form a network by connecting links among them. A user can choose any network structures to be linear, tree, mesh or any other topological forms which one can establish using the links.

In a particular network, one can assign different tasks for different Transputers. While most of Transputers will do the same work in a pipe line network, a mother

and daughters at various levels should have different jobs in a tree network.

As an example of the Transputer network, Figure 2 shows one used for the SVD DAQ system. The SVD is physically divided into 32 independent units. At one interaction of e^+e^- at $\Upsilon(4S)$ energy range, several outgoing particles deposit signals on the units of the SVD. Signal processing in one unit can be accomplished independently of another unit. We introduce a node called a Data Scanner(DS) for front-end data processing of each unit in which a Transputer initiates data taking operations. All of the Transputers at the DS level have a common program.

On the one hand, one must place the DSs close to the SVD, because the main purpose of the DS is to digitize analog signals to digital ones; on the other hand, one must send the digitized data and merge the data of various units into one event data at the vicinity of the central facility, which is located about twenty meters from the SVD. In order to receive the data from each DS in the central facility, we introduce a slave memory module (SMM). The data sent by a DS through the links are received by an SMM, and are stored in a dual port memory on it. The data management on a SMM is also performed by an embedded Transputer. Finally, the fragmented data stored in various SMMs for an event are merged and sent to the BELLE central DAQ processing element. The merging is performed by a Transputer on the DMA master module(DMM). The DMM also initiates data transfer to the BELLE central event builder. We provide the same program for every SMM, and one specified program for the DMM.

We have other auxiliary networks within the DS and SMM levels. We will use these links for passing the histogramming data used to monitor the system performance.

2.2 Management Tools

When building software to be run on a Transputer network such as, that exemplified above, in addition to writing the software as in a normal development cycle, one has to first decide (or know):

- how the network is physically organized (link connections),
- how software is to be placed on the Transputers.

Writing and updating large multiprocessor configurations can be rather hard work unless the process is somehow visualized.

In our DAQ system, one has to take care of the bookkeeping of hardware and configuration the software that is run on each Transputer. Since the system has to be maintained over a long period and it is more than likely that the people in charge of the system will change, maintenance of the system has to be made as simple as possible. During an actual data-taking period, since the system is run 24 hours a day and a

convenient, easily accessible troubleshooting tool is necessary to reduce the workload of the (probably very few) people that have detailed knowledge of the system. Although the failure rate of the system is expected to be very low, there are bound to be times, like when bringing the system up after shutdowns and power failures etc., when immediate maintenance is necessary. Also, in the system installation phase, since the network topology and the software to be placed on each Transputer are modified frequently, to have good troubleshooting tools is indispensable. Generally speaking, there are no commercial or other widely available tools for this troubleshooting task.

Based partly on our previous experience[4], we decided to build a set of tools for this task. In our set of management tools, both hardware and software configurations can be done using graphical tools. The user can pick up a Transputer icon, place it on the display window and connect its links to other Transputers. Figure 3 shows one of the management tools, which configures the Transputer network visually instead having to write complicated documentation. In the figure a Transputer is expressed as a rectangular box and the embedded four links are indicated as a small box with a pair of arrows. The arrows for a link indicates the input and output of the link. By clicking an input of a link and dragging to an output of another link, one can establish a communication link between two Transputers. When the resulting configuration is ready, its topological description is translated into a documentation file, which can be saved for further use.

We build another tool for management called network probe tool. The configuration can be checked for the intended link connection pattern using this network probe tool. The main purpose of this tool is to ensure that the intended software configuration can be downloaded on the network. If there are some non-functioning Transputers or communication links, they are visually indicated on a display of the Transputer network, as shown in Figure 4. In the example shown in the figure, seven Transputers are connected to form a network. All of the Transputers, except for the most upper and left one, respond well as expected from the configuration file. The one which does not respond will not be in good shape. The mark on this Transputer indicates no response within a certain time duration (time over). In such a way one can identify graphically defective Transputers in the system network.

The network probe includes, basically, the same functionality as that provided by such tools as ispy, provided by Transputer manufacturer (SGS-Thomson)[5], with some enhanced features like the possibility to control the order in which processors are checked.

When the desired hardware configuration is ready, one has to go to the second stage for the system buildup, which is to configure the set of software so that it can be downloaded to multiple Transputers. Occonf for Transputer toolsets[6] does this

task, based on a configuration file, which states how many Transputers there are and how they are connected to each other. To assist in writing this configuration file, elements of the software set can be picked and placed on the Transputers using similar operations as those in the hardware configuration. By again connecting the communication channels and links appropriately, a full description of a downloadable program can be generated on the display. Also, software for converting an existing textual configuration into a graphical description is included in the tools.

2.3 Run Control

The run control part of the system deals with the control operation necessary for the actual data taking or a calibration run. The tasks include startup and preparation for data taking and starting and stopping the data acquisition. Also care has to be taken concerning run statistics, error handling and reporting.

The large number of Transputers and their related hardware can control a run hard. Since the DAQ software of the SVD has been designed and constructed based on a state transition model, the run control is also operated according to the same state transition model[7]. In this model the software system is divided into several levels(states). A transition from a state to another is done by external stimulation(control command), and is completed when all of the internal functions defined in the transition have successfully ended. This model is reflected in the GUI. Figure 5 shows an example of a GUI window for run control based on this state transition model. Controlling the system can be done by directly moving between the states, which are shown in rectangular boxes in the figure. The GUI simultaneously displays the current state plus additional information related to the state transitions. For example, when the system is brought from an uninitialized state to a state in which run start is possible(called DAQ-Ready in the current example) checks are performed in order to confirm that the intended configuration can be properly downloaded into the Transputer network. In the initial state, a configuration file has to be selected. This file is created beforehand of a run using the network probe tool in the system buildup phase. In this file (as described in the previous subsection) the active network topology, i.e. number of Transputers and the connection of communication links in the network, is defined. The software to be bootloaded into the system is also described in the configuration file. Before downloading the software the network probe mentioned in the previous section is run during the transition from Standby state to DAQ-Ready in order to confirm that the intended software network configuration is still valid. Then, on an auxiliary window the configuration is displayed and the Transputers, the operation of which has been confirmed, excite their state to DAQ-Ready, indicated by changing the color of an icon on the display. If some Transputers

are not working properly, for example have not been powered on, the place in the network where trouble occurred is indicated on display. We prepare functions for status display and alarm and control message handling between the Transputer network and the controlling workstation. Our system attempts to make it easier by providing some visual clues of the location and cause of the trouble.

The state transitions can be initiated either under operator (manual) control or remotely by the central DAQ system, through the network. The control is granted to only one controller at a time. To prevent any unintended interference, a change in control can only occur during the initial state, and can be password-protected.

The run control enables the system to be operated under remote control or manually from the GUI operator interface. The SVD system in the remote control mode is operated under supervision of the central data acquisition system. The control is based on passing messages, like requests for state transition and supervisory, and monitoring data to and from the central system.

2.4 GUI Development Tools

Although the SVD DAQ system uses particularly Transputer hardware, a large part of this work is targeted towards a general-purpose GUI tool to build distributed real-time systems for a hardware environment that uses the message-passing philosophy. The software has been built using C++ and a platform-independent GUI library and development tools. Although the system is primarily intended to run on Unix workstations and the X-windows system, it is portable between a wide range of platforms, including Motif, OpenLook and Microsoft Windows. A variety of methods to access the hardware, including TCP/IP sockets, can be used. There exist several platform-independent GUI building toolsets on the market. We evaluated mainly two toolsets, namely Galaxy from Visix, Co. (USA)[8]. and wxWindows (freely available software from University of Edinburgh, UK[9]). Although wxWindows has a price advantage and is a very powerful development tool, we decided to select Galaxy for the development work because of the completeness of the package. The software supports not only GUI development, but also operating system abstractions and the development of distributed applications in a very consistent manner. Although we plan to run the system only on Unix workstations and the multiplatform support may not seem to be an essential question, multiplatform libraries ease the programming of graphical user interfaces. Also, it gives us the possibility to easily port parts of the software for other architectures. For example, the Transputer configuration management tool is largely independent of our particular system and should be of value to anyone doing Transputer development work. The system could also be modified to support other message-passing architectures and also mixed architecture networks.

3 Underlying Architecture

To implement the described system on a Transputer/workstation platform, some additional features to the standard Transputer development environment are necessary. In a Transputer system, one Transputer should be connected to the host workstation with the link. This Transputer is called a root Transputer. No Transputer connects directly with the external system resources as an I/O terminal, printer or disks. Thus, if the system must access one of the system resources, the root Transputer asks the host to use it. The host server receives this demand via the link and sends back any information after some relevant operation. The Transputer system accesses the resources in such a way through the host server. However, the host server architecture was initially intended for a command-line, character-oriented interface and as such has no support for an asynchronous user interaction, for example replying to a mouse click on a control window (ignoring the possibility of active polling from the Transputer side, which is unpractical due to poor response and the large amount of link traffic it generates). For efficiency and minimization of effort, we decided to keep the existing server functions and to make minimal additions that are needed to achieve the desired functionality.

The host server works by exchanging packeted messages. In Figure 6 we show the packet protocol between the server and the root Transputer. The Transputer side initiates a transaction by sending a packet to the server. This packet consists of an operation tag (1 byte, denoted as TAG in the figure) followed by an operation-dependent amount of additional data. The server that runs on the host interprets this packet, takes the necessary action and sends a reply packet to the Transputer requesting the service.

This architecture can be made to work in a GUI environment if some modification to the server is added. At the outset we add another connection path between the server and the root Transputer. As shown in Figure 7, the new path is shown as a broken curve between the box indicating the server workstation and the bubble denoted as Multiplexer in the box of the Root Transputer. This can be achieved in a following way. In the server protocol the reply packets also have a tag byte, which is denoted Reply TAG in Figure 6. While the Reply TAG normally contains the result codes of the operations, all 256 numbers are not used by the server. We use a different one-byte number rather than the occupied ones for the reply from the new return path. As a consequence of this modification, on the root Transputer we should add a supplementary multiplexer process that distinguishes the packets from two paths. The interrelation among the processes and the flow of the packets on the root Transputer is schematically shown in Figure 7. In this way the system can be made responsive to asynchronous events and can avoid deadlock problems resulting

from mixing of the reply and command packets. Although the new path is installed, the original command channel is kept in situ. Any necessary handshakes can still be accomplished with normal server function calls.

On the Transputer side, a code to account for the control messages has to be written obviously. This can be illustrated in the following OCCAM sample code. A sample of the Multiplexer is shown first as:

```
--
-- Multiplexer distinguishes command packets from reply packets
-- Packets are sent from the host server
--
WHILE TRUE
  SEQ
    fromserver ? packet
    Reply.TAG := packet[1]
  IF
    Reply.TAG = server.reply -- If reply tag indicates server reply
    server ! packet          -- send it via channel server(new one)
  TRUE
    command ! packet         -- otherwise
                              -- send it via command chan.(original)
```

The modification required in the new user process is to add the following code in the beginning of the process:

```
-- the following code interprets user input on
-- the command channel and messages from
-- subnetwork on the server channel. Error
-- messages and responses to user actions are
-- reported to the host using fs, ts channels to
-- the server
SEQ
  ALT
    command ? input
    SEQ
      IF
        input = start.button -- if the command is to start
        SEQ
          code.startup()     -- then special function call for start
        input = param.load   -- Else if command is load parameters
        SEQ
          read.new.param(from.server,to.server,param.blk)
          ...etc, for more commands -- then get parameters from the server
          -- else another commands?
    server ? message
    SEQ
      IF
        message = start.success -- if the message is success of start
        SEQ
          system.started(fs,ts,OK) -- then send OK message
        message = emergency.stop -- otherwise report errors
        SEQ -- error occurred in subsystem
          ...do error handling
          ...more code
```

The above code fragments are written in OCCAM[10]. The relevant explanation necessary to understand the above codes is given briefly. Two consecutive hyphens mean the beginning of a comment line. Language elements, which are expressed with a few capital letters like ALT,SEQ,IF and WHILE, are called constructions. A structural block after a construction is executed according to the predefined meaning of the construction. One statement is terminated with a carriage return unless a continuous marker is at the top of the next line. Hence one can not put more than one statement in a line. The range of a block for a construction is specified as an indentation of two columns. In the examples above, we use SEQ for sequential execution of a specified block line by line. IF is used as a conditional. It evaluate the following boolean expressions. If an evaluation is true, the associated block below the boolean expression is processed. The ALT construction is used to select number of blocks which are guarded by channel inputs. The block whose channel input is firstly available will be processed. In OCCAM programing, instead of shared variables between processes, common data for two processes are transmitted between each other from one to the other (and vice versa) through channels. A channel is regarded primally as being a virtual port by which communication takes place. The communication and synchronization for two processes are performed with the access of channels. While channels are software objects if the processes are within the same Transputer, a channel is equivalent to a hardware link if two processes are installed in two different Transputers. We can place several software channels on one link. The important thing is that channels should be able to carry data of any type. There is a special link between the host server and the root Transputer as discussed in the beging of this section and two channels are placed in advance. In the examples we call them a `fromserver` for a channel directed to the root and `toserver` for one having direction to the host server.

In the first example the construction WHILE is used with its operand TRUE. This construction makes execution of the following block infinitely, since the meaning of WHILE is to execute its block while the operand is logically true. In the block after SEQ, waiting for a channel input is implemented as `fromserver ? packet`. The operation `?` means channel input. In this case data called `packet` are input through the channel called `fromserver`. The output operation of data through a channel is indicated by the `!` symbol. Hence this example receives a packet from the host server and analyses its second byte (Reply TAG). If this byte is matched with a number from the server input, this packet is output to the user process shown in the second example through the channel `server` with the statement `server ! packet`; otherwise, the packet is sent through `command`. The keyword TRUE in the IF construction means "else" or "otherwise". This example, therefore, multiplexes the input packet to either the channel `server` or `command`.

The second example is also communicated with this multiplexer process through the channels of `command` and `server`. The first ALT operation in the example tries to select the input channel from the multiplexer alternatively. If an input through `command` comes earlier than the other one, the block immediately after `command ?` input is executed. If a packet coming from the multiplexer is received by `command(server)`, the packet is stored in a variable called `input(message)`. The format of these variables(protocols of the channels) should be defined and declared ahead of this code fragment, and should be the same in the two processes. In both cases the structure of the blocks after the channel inputs are almost identical to analyze the input data using the IF judgment. In this way the root Transputer can select the route of the input message asynchronously.

In addition to this OCCAM program, we need to write some extra functions for GUI-dependent operations and for the control system functions, and to allocate some unused operation tags for them. Other than that, all of the server functions are kept intact. Our system mainly uses TCP/IP to access the Transputer link, and can thus be operated remotely. It is also simple to use other possible hardware for the link connection like access through a VME-bus or others, by using a different low-level code.

4 Discussions and outlook

A control and maintenance system for the BELLE experiment has been designed. For most parts, the system is ready, although work must still be done to improve the user interface and to provide the necessary libraries for exception handling, statistics and other necessary functions. An attempt has been made to make the operation of the distributed multiprocessor system as easy as possible by using several visual tools to reflect the state of the system. A set of troubleshooting tools is integrated into the system from the very beginning. In addition to the operating tools, the state of building the system has also been integrated into the system so that all phases of the operation from startup to normal operation can be done in a unified environment. We believe that this is beneficial for a smooth operation of the system throughout the long lifetime of the experiment. The building of this system was greatly simplified by using a cross-platform development library that supports not only GUI development, but also distributed applications as a whole. The commercially available development tools for Transputers can and need to be used in conjunction with this system, although we had to make some minimal changes to the server software, but kept it compatible with the standard version.

5 Acknowledgment

This work is part of a developmental effort to build a silicon vertex detector for the BELLE experiment. The authors are grateful to KEK directors, S.Iwata, F.Takasaki, M.Kobayashi and Y.Watase, for their encouragement. One of the authors (CF) would like to express his appreciation to Professors K.Kubo and T.Hirose for their kind support.

References

- [1] M.T.Cheng et al., the BELLE Collaboration, "Technical Design Report", KEK Preprint 95-02, 1995
- [2] SGS-Thompson Microelectronics, "The Transputer Data Book", second edition 1991.
- [3] M.Tanaka et. al., "Readout System of Silicon Microvertex Detector for KEK B-factory", Nuclear Instr. and Methods A342 169 (1994).
- [4] T. Korhonen et. al., "Software Development Environment for VENUS Parallel Processor Trigger". Proceedings of Real Time '91 pg. 186-189.
- [5] SGS-Thompson Microelectronics, INMOS Technical note "Exploring Multiple Transputer Arrays", The Transputer Applications Notebook - Systems and Performance, INMOS Ltd., First Edition, June 1989.
- [6] INMOS limited., Transputer Toolkit Reference Manual, 1994.
- [7] C. Fukunaga et al., "A Transputer-Based Distributed Front-End Data Acquisition", Proceedings of the 6-th Transputer/occam International Conference, 1994, IOS press, ISBN 90-5199-174-6.
- [8] Visix Software Inc., Reston, Va 22091, USA.
- [9] Available by anonymous ftp from skye.aii.ed.ac.uk, in /pub/packages/wxwin.
- [10] INMOS limited, Occam 2 Reference manual, Prentice hall, 1988, ISBM 0-13-629312-3.

Figure captions

Figure 1 Microvertex detector control system.

Figure 2 Transputer network of the Silicon Vertex Detector Data Acquisition System

Figure 3 Network configuration screen.

Figure 4 Display of the processor network status (tentative design).

Figure 5 Run control display (dimmed states are disabled).

Figure 6 Transputer server protocol.

Figure 7 Processes running on the root processor of a GUI-based Transputer system.

Figure 1

Figure 2

Figure 3

Figure 4

Figure 5

Figure 6

Figure 7