Implementation of IEC61499 Distributed Function Block Architecture for Industrial Measurement and Control Systems (IPMCS)

Yang Wei

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
2001/2002

Implementation of IEC61499 Distributed Function Block Architecture for Industrial Measurement and Control Systems (IPMCS)

Submitted by
Yang Wei

Department of Electrical & Computer Engineering

In partial fulfillment of the
requirements for the Degree of
Bachelor of Engineering
National University of Singapore

ABSTRACT

Distributed measurement and control systems have begun to be accepted in manufacturing automation systems. Software modules called function blocks are applied to lessen the complexity and high engineering cost of a distributed system. The IEC 61499 is a new international standard introduced for the application of function blocks in distributed control environment. It aims to standardize the design and application of IEC 61131 function blocks for PLC and IEC 61158 function blocks for Fieldbus Technology.

This project is an industrial collaboration project with Yamatake Corporation, Japan and is a continuation of work from the Industrial Attachment. A configuration tool was developed using Java during that period. This tool allows user to load and display function blocks, remotely configure system and start up the devices through network. As a result, a conference paper was published in TOTAL ENTERPRISE SOLUTIONS CONFERENCE, ICAM Asia 2001.

This project requires the use of XML for defining common execution of command transfer in control network and function block development for real-time control applications. C++ is used for the control of I/O board. Java is used for the prototyping development of embedded real time execution algorithm. Java Native Interface is also implemented to link C++ and Java programs. Besides, this project includes model description schema using UML and investigation on Internet protocol for application of system in open network.

A set of function blocks for process control based on the IEC61499 standard are developed, and the linkage of the function blocks with the computer I/O card is established. As such, the function blocks can be configured for real time process analog control and batch control. A coupled tank with analog inputs and outputs is successfully controlled and some classic control examples like open loop control, close loop control and PID control are realized. A distributed batch control through computer network is also successfully implemented. This is the first trial of distributed real-time process control and batch control using IEC61499 in Asia.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

“IEC” « International Electrotechnical Commission »

“NWP” « New Work Proposal »

“FB” « Function block»

“IPMCS” « Industrial-process measurement and control systems »

“PAS” « Publicly Available Specification »

“ITA” « Industry technology Agreement »

“OPC” « Object language embedded for Process Control »

“UML” « Unified Modeling Language »

“ECC” « Execution Control Chart »

“XML” « Extensible Markup Language »

“FBD” « Function Block Diagram »

“DFC” « Data Flow Chart »

“I/O” « Input and output »

“DLL” « Dynamic link libraries »

“FIFO” « first in first out »

“NRSE” « Non-referenced Single-Ended »

“A/D” « Analog to Digital »

“ADC” « Analog Digital Converter»

“JNI” « Java Native Interface »

“PID” « Proportional Derivative Integral »

“HMI” « Human machine interface »

“SISO” « single input single output »

“*PV*” *« process variable »*

“*MV*” *« manipulated variable »*

“*SP” « set point »*

“*IP” « Internet Protocol »*

“*MVC*” *« Model View Controller »*

CHAPTER 1

INTRODUCTION

1.1 Background

In early 1990, Technical Committee 65 of the International Electrotechnical Commission (IEC TC65) received a New Work Proposal (NWP) to standardize certain aspects of the application of software modules called Function blocks (FB) in distributed industrial-process measurement and control systems (IPMCS). IPMCSs utilizing the "fieldbus" (IEC 61158) standard then in development in Working Group 6 of Subcommittee 65C (SC65C/WG6) were especially emphasized in the NWP. However, function blocks were also an essential part of the programming language standard IEC 61131-3 for programmable controllers being developed in SC65B/WG7. Therefore, TC65 determined that a common model for the use of function blocks was required and assigned the new Project 61499 to a new Working Group 6 (TC65/WG6) of the parent committee.

Due to the relative immaturity of the IEC 61158 project at the time of the proposal, experts and a Project Leader were not available for the 61499 project until two years after its inception, at which time the first edition of IEC 61131-3 was also completed and available for reference.

TC65/WG6 identified a number of fundamental issues which had to be resolved in order to achieve a common model for the application of function blocks in distributed IPMCS.

Through a long process of systematic application of software engineering and open systems principles, with intensive international review and revision, a consensus has emerged on the basic concepts and detailed technical approach to the resolution of these issues.

The evolving IEC 61499 is an architectural standard for building distributed systems consisting of devices that are interconnected with communication networks, have interfaces to the controlled process or machine and contain distributed applications. Currently, IEC 61499 is of a Publicly Available Specification (PAS) and the implementation of this next generation IPMCS architecture will be under ITA (Industry technology Agreement) procedure with several major contributors in the world such as Rockwell Automation (USA), Softing Gmbh (Germany), Siemens AG (Germany) and Yamatake Corporation (Japan).

1.2 Motivation of project

Distributed measurement and control system has begun to be accepted in manufacturing and processing industry. It is estimated that an increasing proportion of industrial automation solutions will be implemented by decentralised systems. Central control systems will be replaced by smart field devices and distributed controllers. Heterogenous systems with components from different hardware and software manufacturers will often be required. Future industry solutions will demand system engineering for integration and seamless communication between different components in the system. However, to realize such a distributed system in today's industry is complicated and requires high engineering cost.

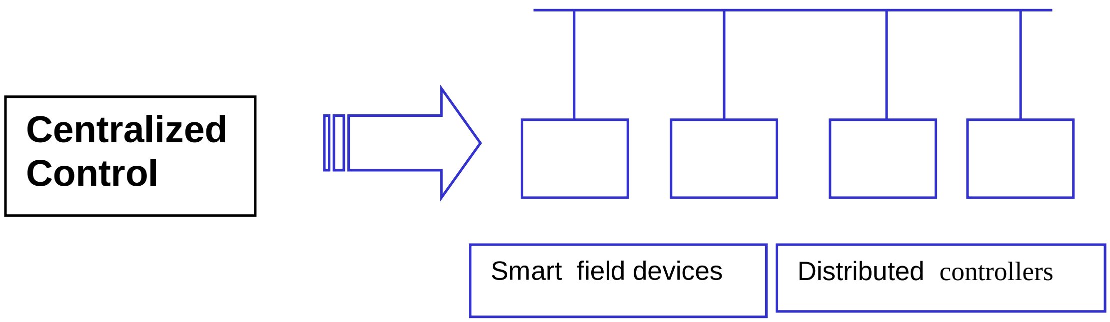


Fig.1.1 Centralized control to distributed control

By producing a common interface of execution platform, network protocol and system configuration. IEC 61499 provides schema for realizing distributed application, in which interface for network protocol and configuration are both considered.

In the IEC 61131-3 configuration, the execution of a program is triggered by a periodic or non-periodic task. Upon triggering, the function block instances in the program are executed in a predetermined order. Similarly, execution of function block instances in a distributed system can be performed according to a predetermined cyclic schedule. However, IEC 61499 supports a more generalized model in which a centralized mechanism for enforcing execution schedules may not exist. Function block execution is high-speed and event driven but can also be cyclic. This is essential to many modem machine control applications.

In the IEC 61131-3 model, communication and I/O functions are only loosely coupled to variables used in programs through the “access path" and "global and directly represented variable" mechanisms. In Field Bus, no communication function block is available for inter-communication between systems. OPC (Object language embedded for Process Control) is used to establish network links which is done separately from application design. IEC 61499 provides communication function blocks, which are easy to implement and support access of different networks so that function block can be easily distributed to field devices and controllers.

IEC 61499 function block model accommodates the existence of multiple alternative algorithms within a single function block body, selectable depending on some external event or condition, e.g., initialization algorithms, normal operation algorithms or fault processing/recovering algorithms. All this is possible because IEC 61499 function blocks can be designed using different programming language. For Fieldbus, limited flexibility is given to user where only basic function blocks can be used.

As manufacturing control becomes more distributed, encapsulation and reuse of control algorithms by end users, equipment suppliers and system integrators become more important and widespread. Following the extension of IEC 61131-3 programming languages, IEC 61499 supports not only encapsulation of algorithms but also sub-application or even system application so that system designer who may not have any programming knowledge is able to design applications and reuse them to simplify design procedures. Such technology is not available for Fieldbus.

Future industrial processes will exhibit much degrees of physical reconfigurability in order to accommodate frequent changes in product mixing and volume as well as frequent introduction of new technology. In addition, rapid configuration of industrial process will be used much more frequently to recover from machines and process faults with minimal loss of production. Using IEC 61499, the control systems can be configured quickly for the most part automatically thus greatly improving agility.

Therefore, there is no doubt that by using IEC 61499 concept, engineering cost is expected to reduce and system will be more flexible and maintainable.

1.3 Project overview

This project is an industrial collaboration project between National University of Singapore and Yamatake Corporation, Japan and is a continuation of work from the six months industrial attachment with Yamatake Corporation. In this project, the concept of IEC 61499 and its implementablity, reliability and performance on process control system are studied and investigated.

The objectives include:

1. Development of a system configuration tool.
2. Implementing a reliable function block set of IEC 61499 standards, and applying them to the field devices for real-time process control.

The configuration tool was developed using Java during industrial attachment in Japan. The tool allows users to load and display function blocks, remotely configure the system, download function blocks to field devices and remotely start up the devices through network. It will be discussed in Chapter 2 of this thesis.

The second objective forms the main part of the final year project. A set of function blocks based on the IEC 61499 standard were developed, and the linkage of the function blocks with the computer I/O card was established. As such, the function blocks can be configured for real time process control. Details are illustrated in Chapter 4.

A coupled tank with analog inputs and outputs is successfully controlled and some classic control examples like open loop control, close loop control and PID control are realized. A batch control through network is also implemented. Details are illustrated in Chapter5. This is the first trial of distributed real-time process control using IEC 61499 in ASIA.

CHAPTER 2

BASICS CONCEPTS AND CONFIGURATION TOOL DEVELOPMENT FOR IEC 61499

2.1 Basics concepts of IEC 61499

In order to design a working application, the concept of IEC 61499 is first studied. IEC 61499 defines four models, namely System, Application, Resource and Execution model.

2.1.1 System model

Unlike classical PLCs, where function blocks are elements in a centralized controller, function blocks in IPMCS are considered to be elements of distributed application in a decentralized control system.

Fig.2.1.1 illustrates the general concept of a distributed system in IEC 61499, in which devices may communicate with each other over one or more communication links and may interface to controlled machines and processes. This can be recognized as an abstractive of a type of distributed IPMCS (industrial-process measurement and control systems).

Applications may be distributed among one or more devices.

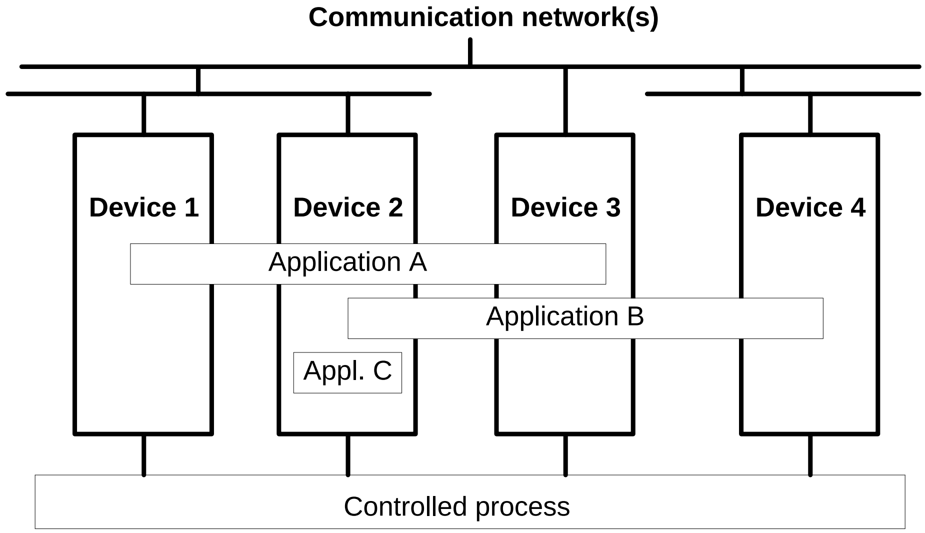


Fig.2.1.1 System model

2.1.2 Application model

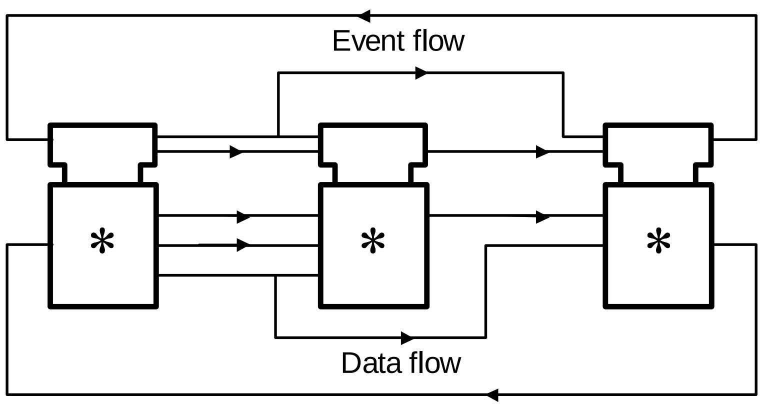


Fig.2.1.2 Application model

Fig.2.1.2 illustrates an abstractive model of a distributed application. The application consists of one or more function block instances connected by event and data connections. The function block instances may be distributed among devices.

Execution sequence is decided by event flow and data transfer among function blocks is represented by data flow. In an executable model, event and data connections among devices can be represented by Service Interface function blocks.

2.1.3 Resource model

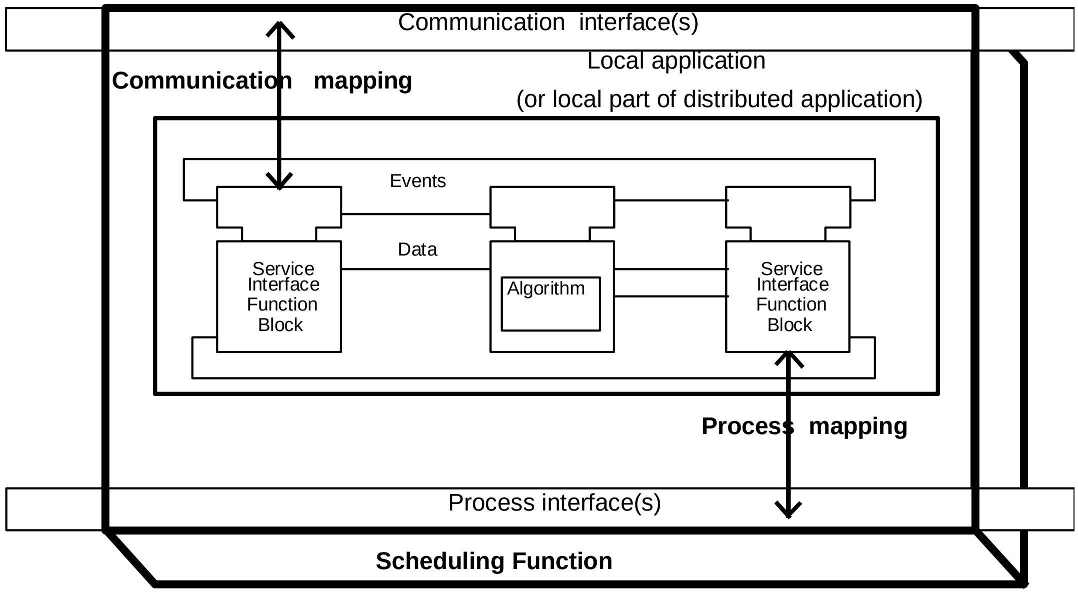


Fig.2.1.3 Resource model

An abstractive resource model is shown in Fig.2.1.3. This model is a local application in a device or a part of distributed model. Event and data connections do not exist among devices. Communication mapping exists between communication interface and service interface function block. Similarly process mapping exists between process interface and service interface function block.

2.1.4 Execution model

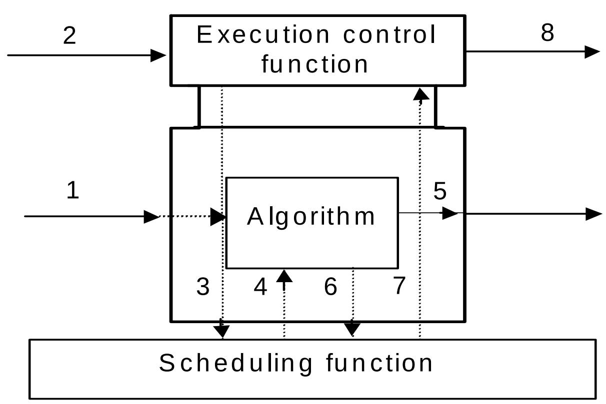


Fig.2.1.4 Execution model

As illustrated in Fig.2.1.4, in response to an input event, the execution control function portion of a function block instance invokes the execution of algorithm for function block. After completing execution of algorithm, the execution control will generate zero or more output events respectively.

2.1.5 Execution control chart (ECC)

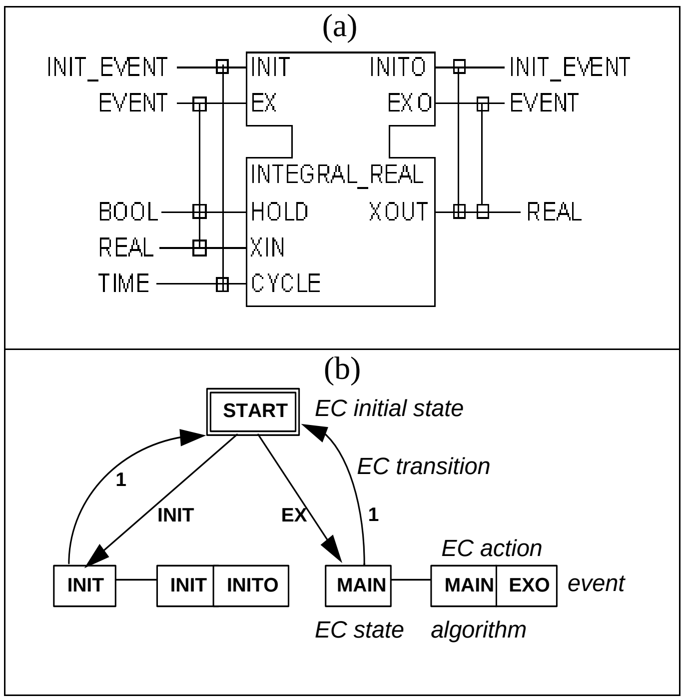


Fig.2.1.5 Execution control chart

In contrast to a centralized system, where execution of programs can be scheduled on a scanned or, possibly multitasking basis, a distributed system requires explicit mechanisms to specify the relationships between the transmission of data and the execution of control algorithms. To meet this need, IEC 61499 adds an explicit event-interface and execution control function to the control algorithms of IEC61131-3 function blocks.

Execution of algorithm in Basic Function Block is controlled by Execution Control Function, which is represented by Execution Control Chart as shown in Fig.2.1.5. In this example, EC initial state is START. In response to an INIT event input, EC state will be transferred to INIT state where EC action will be executed. First, the corresponding INIT algorithm is executed, and upon completion of executing algorithm, event output INITO will be generated. Condition 1 then reset the INIT state back to START state.

2.1.6 Service interface

A service interface function block provides services to application such as interaction between application and resources like network or HMI. As a distributed control application is to be realized in this project, extensive research is done on those function blocks. Two examples of service interface function blocks, PUBLISHER and SUBSCRIBER, are illustrated in Fig.2.1.6 and Fig.2.1.7.

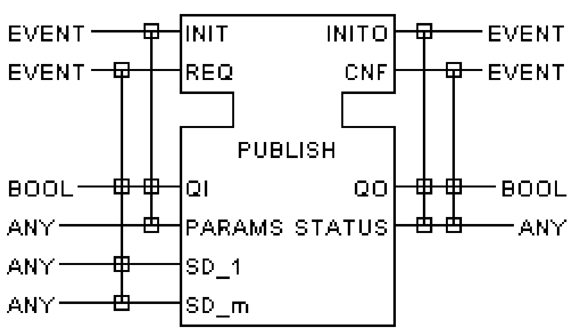


Fig.2.1.6 Service Interface/PUBLISHER

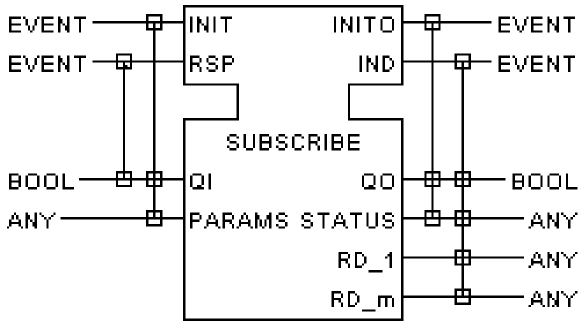


Fig.2.1.7 Service Interface/SUBSCRIBER

Function block PUBLISH and SUBSCRIBE are implemented in different devices and they communicate through network connection. Both function blocks are initialized by input event INIT+ (QI = True) as shown in Fig.2.1.8. To ensure proper communication between two function blocks, input PARAMS value (e.g. IP address) must match.

After establishment of communication link, an input event at REQ of PUBLISHER will trigger the accepting of input data at SD_1 to SD_m. Subsequently the same data is broadcast to SUBSCRIBER. At the same time, input event is also broadcast to the subscriber IND output as output event, which then triggers data output through RD_1 to RD_m. A graphic representation is presented in Fig.2.1.9. An input event INIT- (QI = False), will terminate communication between PUBLISHER and SUBSCRIBER as shown in Fig.2.1.10.

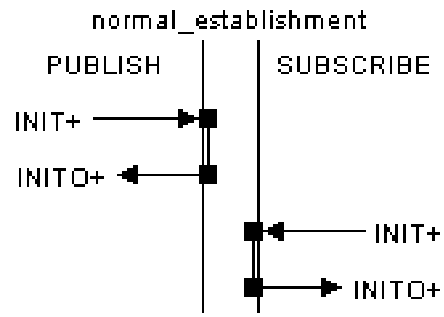


Fig.2.1.8 Interaction establishment time-sequence

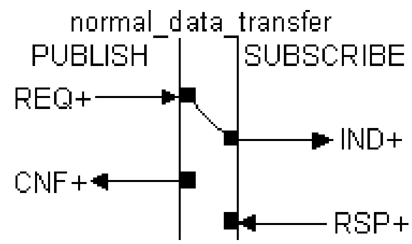


Fig.2.1.9 Data transfer time-sequence

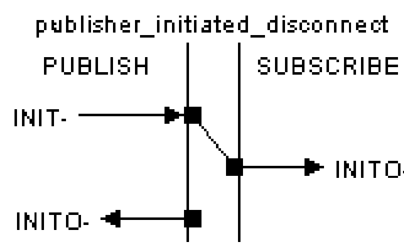


Fig.2.1.10 Interaction termination time-sequence

2.2 Configuration tool

The preliminary goal is to build a system configuration tool for the application portion which includes a graphical user interface, XML file loader, command list generator and file transfer via open network. The diagram of an open distributed system with the configuration tool under the IEC 61499 concepts is shown below.

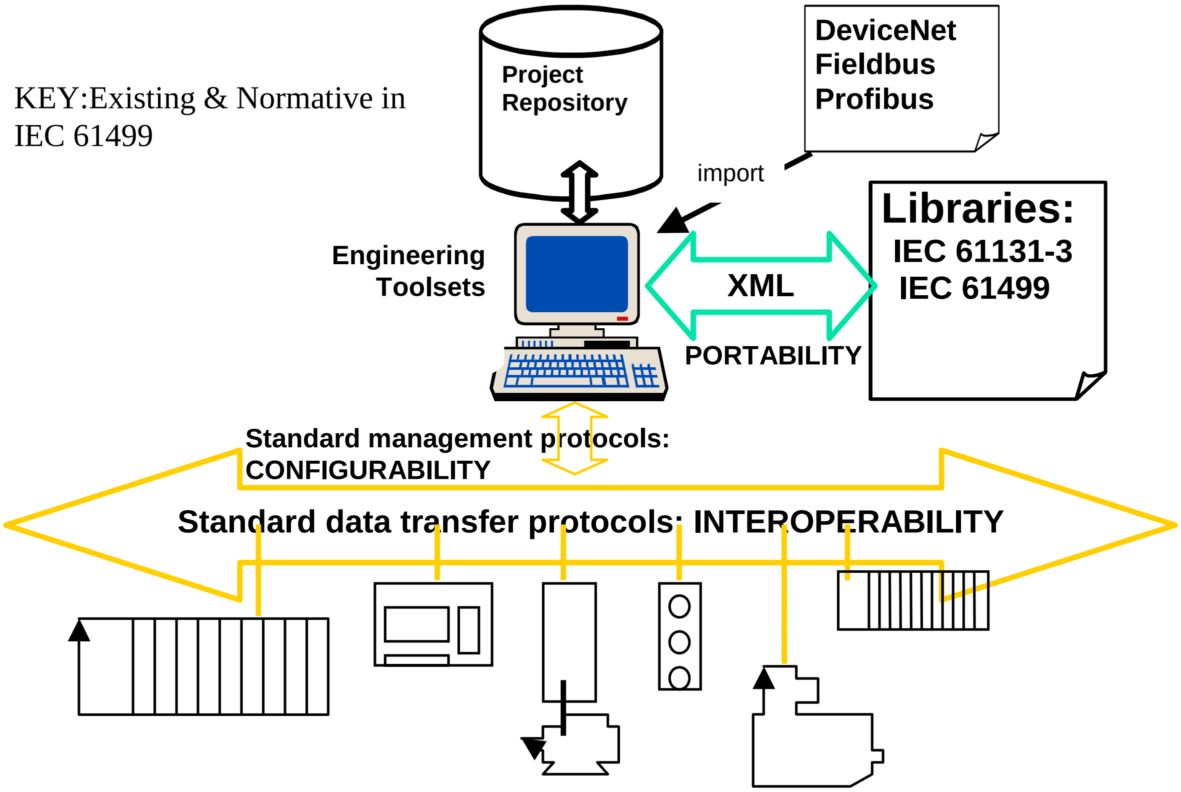


Fig.2.2.1 Distributed intelligent devices & controllers

Under the vision of IEC 61499, smart field devices with microprocessor will replace central controllers in distributed control systems and function blocks can be implemented in field devices and controllers and communicate with each other to form a complete system. This is shown in Fig.2.2.2 below.

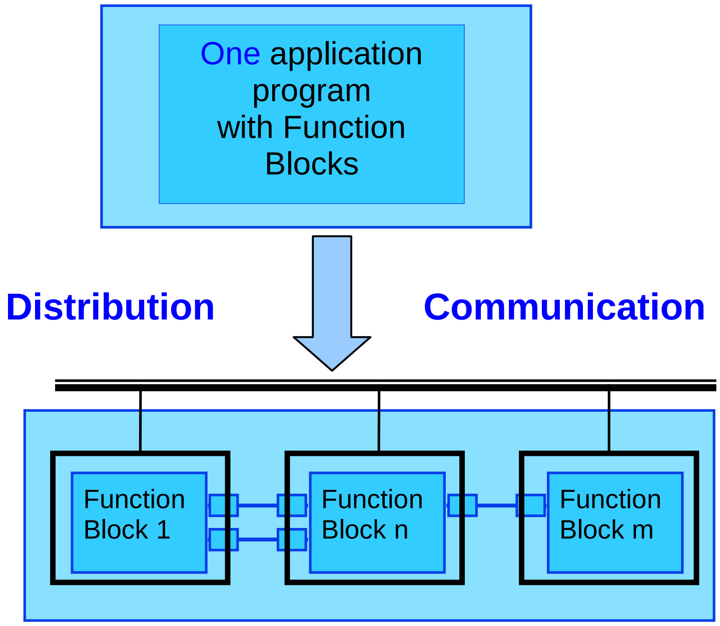


Fig.2.2.2 Smart field devices

A system configuration tool is essential in building a distributed system. Using configuration tool, a system designer is able to load function blocks or pre-designed applications from online library, project repository or even field devices and starts system design, configuration and execution.

The features of the configuration tool developed in this project include:

1. Load system file and extract necessary information for configuration.
2. Display function block diagram of Application, Device etc.
3. Generate a list of commands in XML for configuration.
4. Remote start up of function block applications in field devices.

IEC 61499 architecture is the first item to be studied. For better understanding of the design, implementation and to allow incremental improvement of an IPMCS, use case diagrams are drawn first. The software used in the project is Rational Rose which provides drag and drop option for easy design of the diagram. A simple use-case diagram for the configuration tool is displayed below.

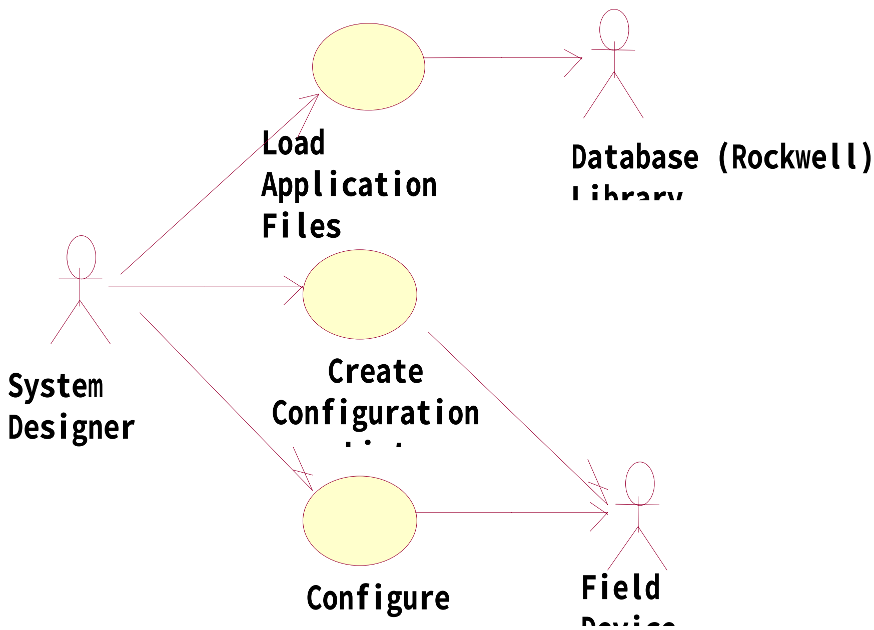


Fig.2.2.3 Use case diagram for the configuration tool

The Model View Controller (MVC) design pattern is used to develop a reusable object-oriented Graphic User Interface (GUI) rather then a conventional GUI, which tends to lump every object together. MVC decouples them to increase flexibility and reusability. Sequence diagrams are first being drawn to pick up different objects for easy design.

2.3 Data extraction from system file

As the GUI is developed in Java and IEC 61499 system files are represented in XML format, XML parser is needed to extract necessary data and present it to a Java program as a DOM object tree, or convert those representations back to XML syntax. After the extraction of data, command lists are generated to send to field devices.

The IBM's XML Parser is selected because it provides high performance and support XML1.0 and DOM level 1 and 2. A system file can be separated into 4 sections, namely:

- System name and version
- Application - Name
  - Function block instances
  - Data and event connections
- Device - Type
  - Instance name
  - Configuration of all resources
- Resources - Type
  - Instance name
  - Function block instance
  - Connections

Once all the necessary information has been extracted using XML parser, they will be stored for the next function: displaying diagram.

2.4 Display function block diagram

The configuration tool developed is able to show different diagrams of the application such as Function Block Diagram (FBD), Data Flow Chart (DFC) and the composing function block layouts. It is also able to switch between old and new application examples and make comparison between two models and generate command list for implementation. A tree is displayed on the left. Fig.2.4.1 below illustrates the concept in a simple form.

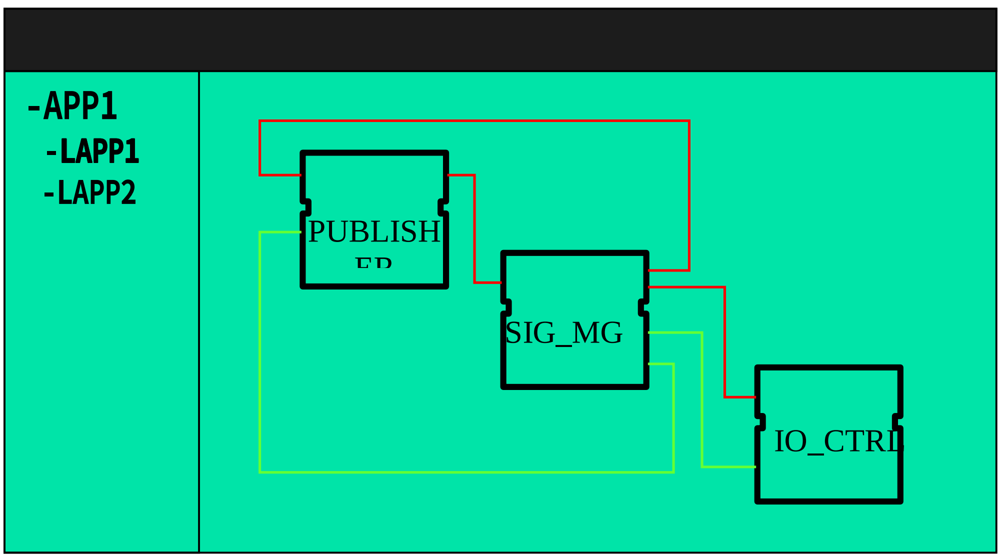


Fig.2.4.1 Configuration tool concept

The displaying part consists of mainly 3 parts: the Function Block itself, the application/device/resource view and the connection between input and output. Once the information about what are the function blocks needed is extracted, they will be compared with the individual function block file before all the function blocks are drawn out. The connections are the next to be drawn. It includes the identification of the coordinates of the respective points. In the case where data input is used instead of connections, a string input is considered. Fig.2.4.2 shows the finalized version of the configuration tool displaying an application.

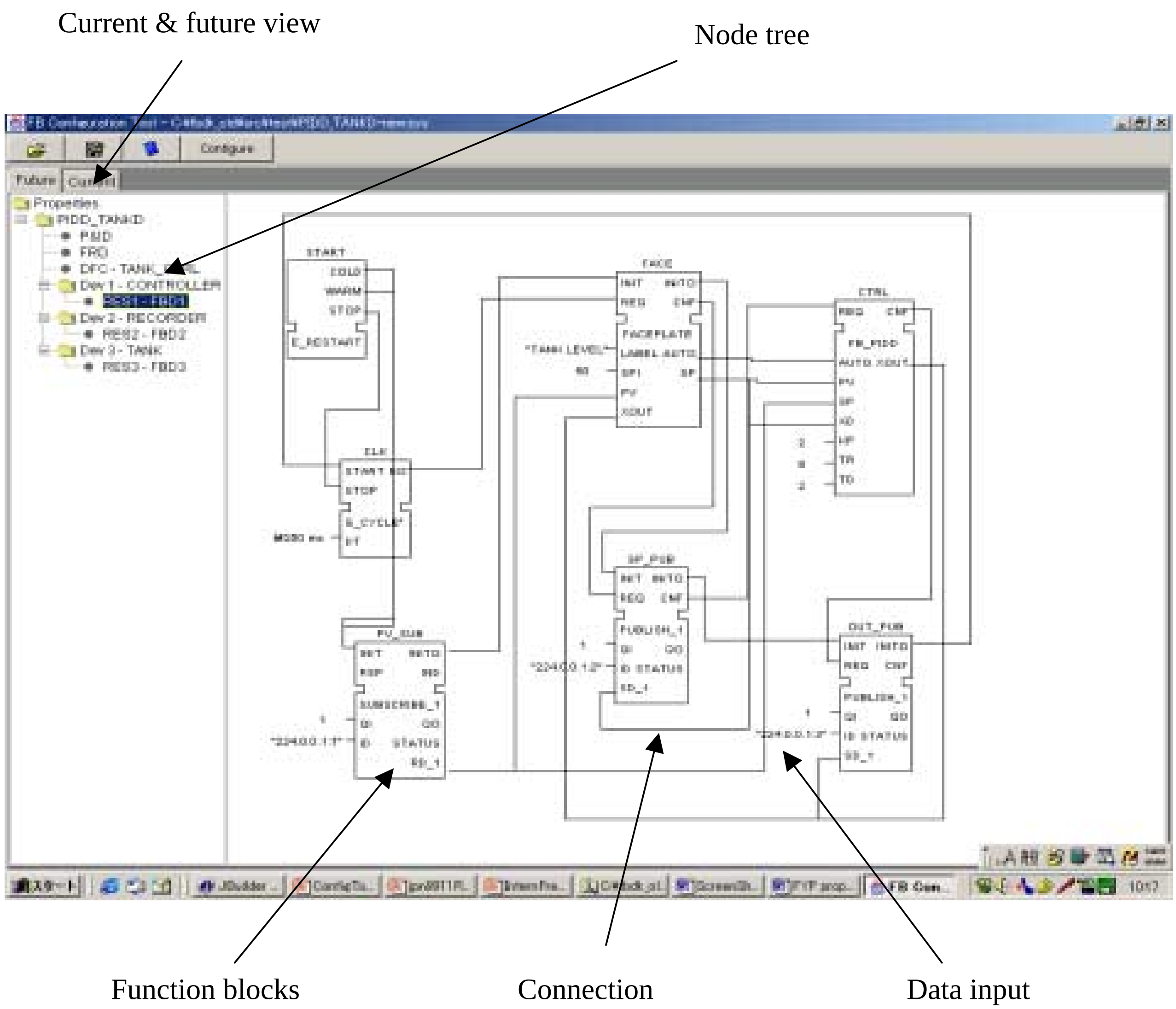


Fig.2.4.2 A typical application file displayed in the configuration tool

2.5 Configuration list generation

Configuration list is needed to be generated and send to connecting devices for the application to operate. A list of standard configuration commands is provided in the guideline and should be in XML format. Some standard commands are as follows:

Command syntax examples:

—CREATE functionBlockInstance
—CREATE functionBlockType
—QUERY dataTypeName
—CREATE connection
—START functionBlockInstance
—STOP functionBlockInstance
—DELETE connection

After pressing the configuration button in the configuration tool, user is required to enter a valid network ID before the generation of commands. After that, a list of configuration list generated automatically according to the specific application and will be displayed in the pop-up window. The user may either choose to reconfigure, send commands to devices, cancel, or start the application. Sending data only ensures that the data input will be inputted to function blocks but will not triggering any execution as the data and event commands are separated in IEC 61499. Only by pressing the start button will send the event input and triggers the execution of algorithm. The diagram is shown below.

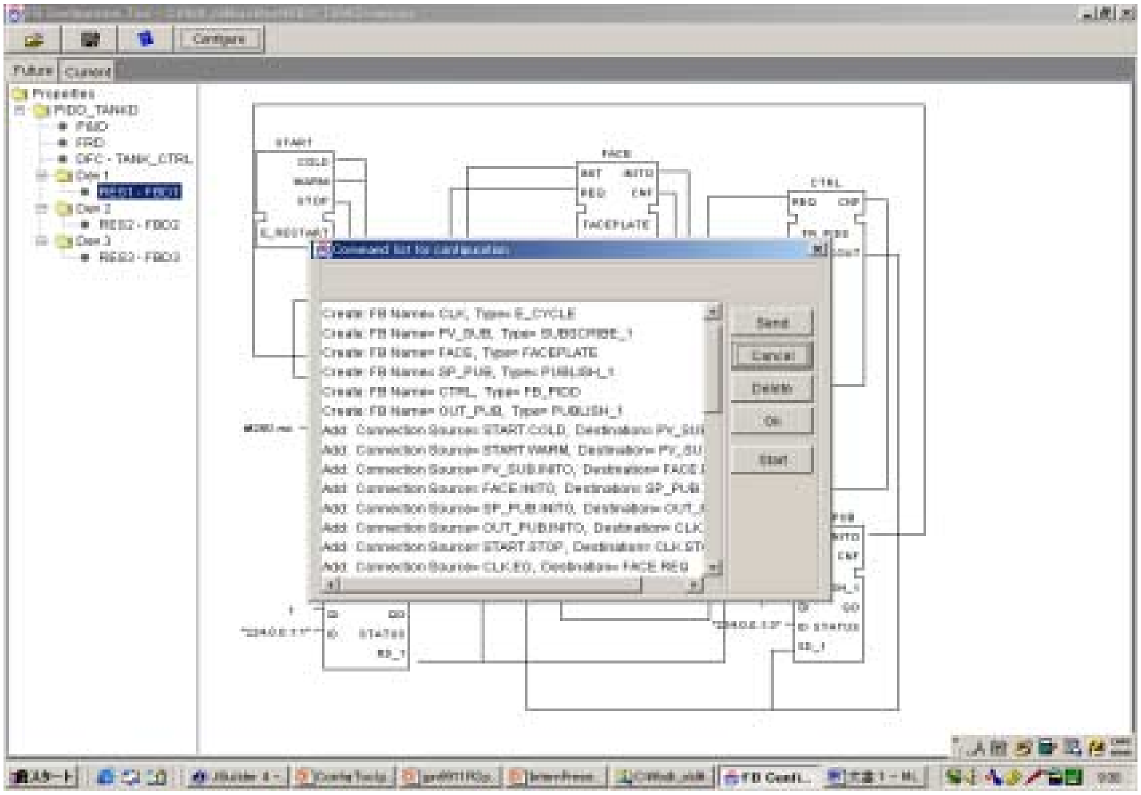


Fig.2.5.1 Configuration tool with a list of commands to apply

After the command is being send and start command being generated, a net status log will then be generated to show the sending status. From this, the user is able to monitor the results of the configuration. The generation of commands in XML by the configuration tool is therefore complete and intensive research on XML is required on structure requirements and speed of data transmission, etc. Original XML files may be required. Internet protocol is also being studied to investigate the communication of function block modules in open network.

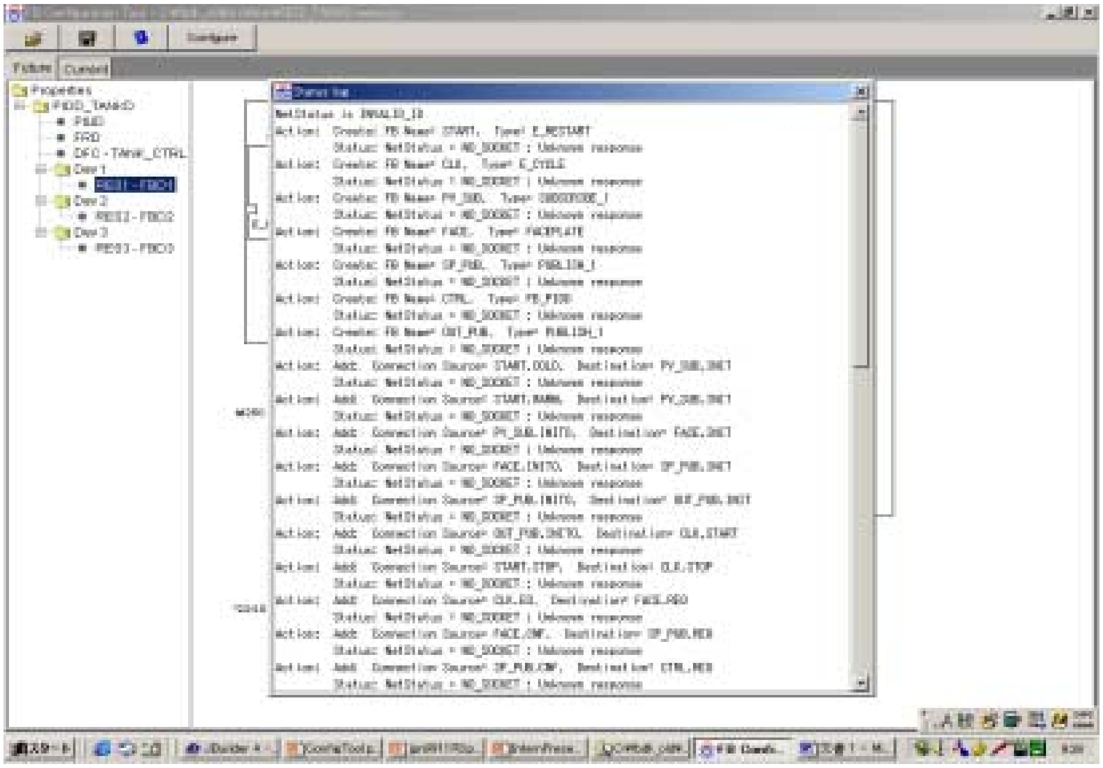


Fig.2.5.2 Configuration tool with send status

Fig.2.5.3 below illustrates an application, which is implemented using the configuration tool.

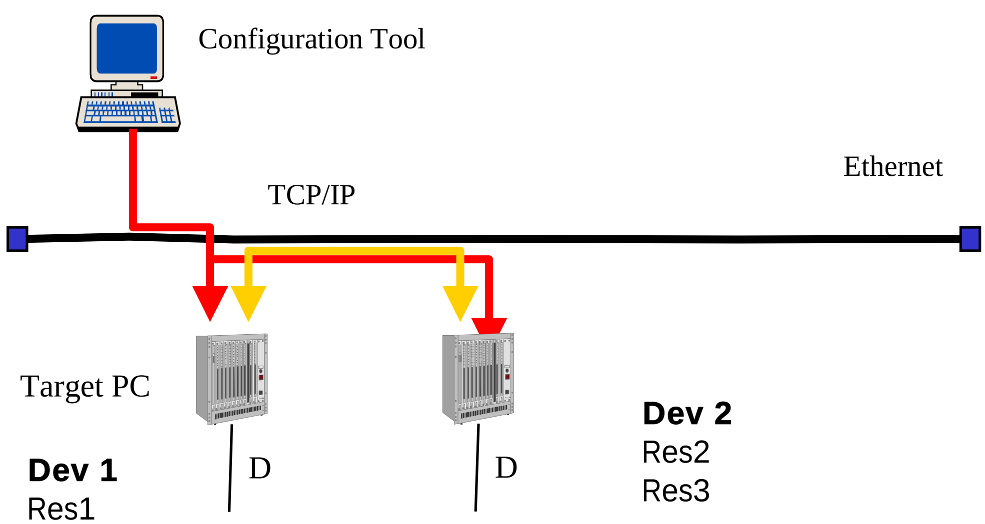


Fig.2.5.3 A configuration test application

A IEC 61499 standard application is loaded in the configuration tool. After configuration commands are being generated, they are downloaded to two target PCs which act as intelligent devices. Once the start command is sent, the two target PCs start to operate individually and communicate with each other to perform the re-designed task. No central controller is needed.

CHAPTER 3

TEST PLATFORM INVESTIGATION AND I/O CONTROL USING C++ AND JAVA

3.1 Test platform

Two process control applications are formulated namely water level control and temperature control of mixing liquid. Due to the availability of hardware in Control and Simulation Lab, water level control is chosen. The test platform used in this project is the Coupled-Tanks Control Apparatus model: PP-100 from KentRidge Instruments.

3.1.1 Coupled-tanks control apparatus

The equipment consists of two small perspex tower-type tanks mounted above a reservoir as water storage, see Fig.3.1.1. Water is pumped into the top of each tank by two independent pumps. The head of water in each can be visually read on the attached scales at the front of the tanks. Each tank is fitted with an outlet which is connected by a hose with a return through reservoir lid. The amount of water which returns to the reservoir is approximately proportional to the head of water in the tank since the water return tube at the base of the tank functions as a pseudo-linear hydraulic resistance. If needed, this resistance may be increased by the use of a screw type clamp.

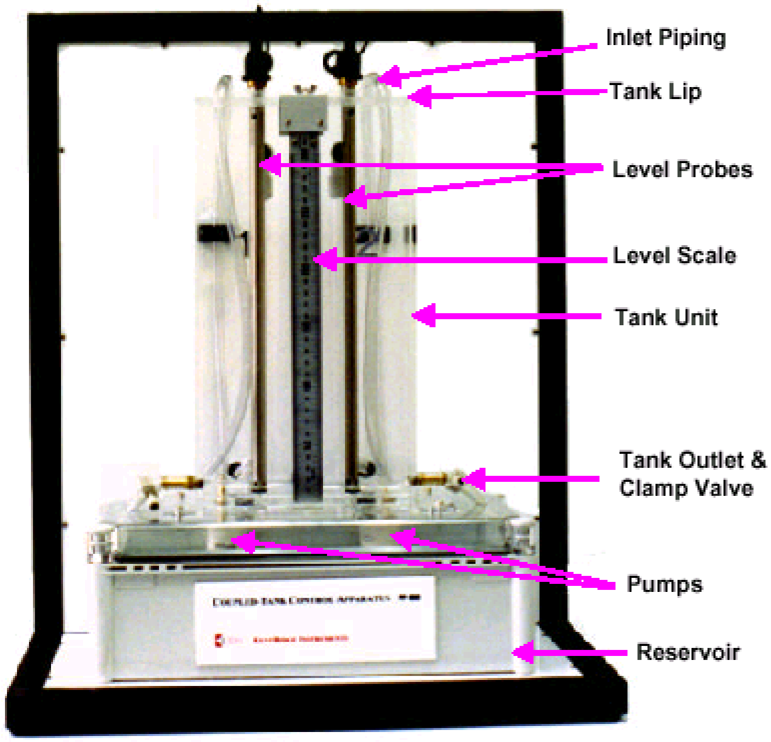


Fig.3.1.1.1 Coupled-Tanks

The level of water in each tank is monitored by a capacitive-type probe, which provides an output signal proportional to the water level. This DC signal voltage is in the range 0 to 5 volts, for which, zero level represents the rest point of water when tank is empty at 20mm, and the full state at approximately 300mm. An internal baffle which controls the leakage between two tanks is also present. The baffle can be adjusted to provide a range of inter-tank resistance.

Fig.3.1.1.2 shows the layout of the control panel. The input, output and ground terminals are arranged in two rows: the top row is for Tank 1 and the bottom is for Tank2. The ground terminals are connected together as one common reference point for all input and output signals. The coupled-tank can be operated in LOCAL and REMOTE modes. In this case, REMOTE mode is chosen where external voltage signals are used to control the pumps.

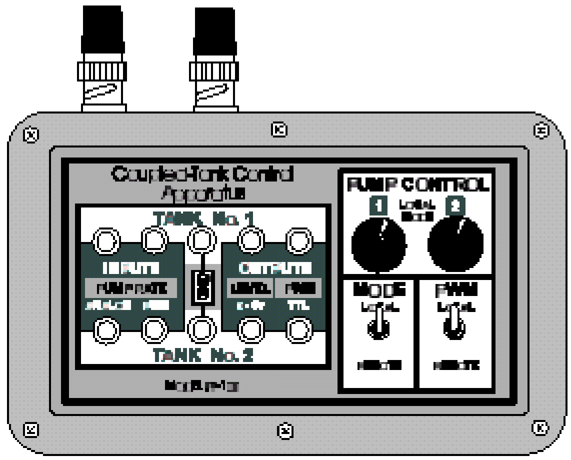


Fig.3.1.1.2 Control panel of coupled-tanks

3.1.2 Coupled-tank Experiment

In order to get familiar with the Coupled-Tanks PP-100 and its functions, an experiment taken from EE 4305 Introduction to Fuzzy/Neural Systems is conducted. The software used for the experiment is the CAIRO CTS001. It provides options for simulation and real-time control of the coupled-tanks as well as collecting system response for analysis.

Calibration of level sensor and pump is first done using open loop control. A real-time PID control application is subsequently tested. The parameters of Kp, Ki, Kd are tuned to get the better step response.

3.1.3 I/O board

The I/O board Lab PC-1200 from National Instruments is used as an interface between PC and the coupled-tank. This board is high-performance analog, digital, and timing board for PC AT and compatible computers. Additionally, the Lab-PC-1200 has analog output capabilities. It has eight analog input channels that can be configured as eight single-ended or four differential inputs, a 12-bit successive-approximation ADC, 24 lines of TTL-compatible digital I/O, and three 16-bit counter/timers for timing I/O.

The Lab PC-1200 is a completely switchless and jumperless data acquisition board. It allows DMA, interrupts, and base I/O addresses to be assigned by the system to avoid resource conflicts with other boards in the same system. It is designed for high-performance data acquisition and control for applications in production testing, and industrial process monitoring and control.

Connection from Lab PC-1200 to coupled-tank is established via connectors. Since the coupled-tank uses only two analog input and output respectively, only six pins are used. Table 3.1.3 below shows the detailed pin assignment to the coupled-tanks.

Table 3.1.3 Pin assignment to coupled-tanks

Pin	**Signal name**	**Direction**	**Description**
1	ACH0	AI	Connected to tank 1 level sensor
2	ACH1	AI	Connected to tank 2 level sensor
9	AIGND	I/O	Analog input ground
10	DAC0OUT	AO	Connected to tank 1 pump
11	AGND	N/A	Analog ground
12	DAC1OUT	AO	Connected to tank 2 pump

Calibration of the board is necessary before it can be put to use. NI-DAQ is the driver software for Lab PC-1200. It provides a configurator which can be used to configure the hardware settings of the board. The NI-DAQ Configuration Utility is a Windows-based application, which is used to configure and view National Instruments Lab PC-1200 settings. One thing to note is that both the AI and AO channels are set to “Referenced Single Channel” and “Unipolar” so that the input and output voltages will be from 0 to 10 V accommodating the specification of the coupled-tank.

3.2 Read and Write I/O in C++

Since Lab PC-1200 board supports only C++ other than NI LabView or LabWindow, the only way that a function block can control the I/O board to read and write data is through C++ programming. However no sample programs in C++ is available for reference as no such work was done using C programs. Extensive research was done to investigate how to program I/O using C++. It was found that the driver software, NI-DAQ, supplies header files and function prototype file for use with Borland C++.

The NI-DAQ for Windows function libraries are Dynamic link libraries (DLLs), which hold a collection of routines that can be called by applications and other DLLs. It means that NI-DAQ routines are not linked into the executable files of applications. Only the information about the NI-DAQ routines in the NI-DAQ import libraries is stored in the executable files. Both Nidaq32b.lib and Nidex32b.lib should be included in the program. Import libraries contain information about their DLL-exported functions. They indicate the presence and location of the DLL routines. Information can be sent to the DLL routines through import libraries or through function declarations. Using function prototypes is a good programming practice.

Several function prototypes are chosen to be used in the program. A list is given below.

1. **Format**

   status = **AI_Clear** (deviceNumber)

   Purpose

   Clears the analog input circuitry and empties the FIFO memory.

   Parameters

   Input

Table 3.2.1 Input parameters for function AI_CLear

Name	**Type**	**Description**
deviceNumber	i16	assigned by configuration utility

   AI_Clear clears the analog input circuitry and empties the analog input FIFO memory. AI_Clear also clears any analog input error conditions. AI_Clear can be called before AI_Setup to clear out the A/D FIFO memory before any series of externally triggered conversions begins.

2. **Format**

   status = **AI_Configure** (deviceNumber, chan, inputMode, inputRange, polarity, driveAIS)

   Purpose

   Informs NI-DAQ of the input mode (single-ended or differential), input range, and input polarity selected for the device. This function can be used if the jumper is changed affecting the analog input configuration from their factory settings.

Parameters

Input

Table 3.2.2 Input parameters for function AI_Configure

Name	Type	Description
deviceNumber	I16	assigned by configuration utility
chan	I16	channel to be configured
inputMode	I16	indicates whether channels are configured for
inputRange	I16	voltage range of the analog input channels
polarity	I16	indicates whether the ADC is configured for unipolar or bipolar operation
driveAIS	I16	indicates whether to drive AISENSE to onboard

chan is the analog input channel to be configured. **Chan** must be set to –1 because the same analog input configuration applies to all of the channels. If all of the channels are to be configured identically set **chan** to –1.

InputMode indicates whether the analog input channels are configured for single-ended or differential operation:

0: Differential (DIFF) configuration (default).

1: Referenced Single-Ended (RSE) configuration (used when the input signal does not have its own ground reference. The negative input of the instrumentation amplifier is tied to the instrumentation amplifier signal ground to provide one).

2: Non-referenced Single-Ended (NRSE) configuration (used when the input signal has its own ground reference. The ground reference for the input signal is connected to AISENSE, which is tied to the negative input of the instrumentation amplifier).

InputRange is the voltage range of the analog input channels. **Polarity** indicates whether the ADC is configured for unipolar or bipolar operation:

0: Bipolar operation (default value).

1: Unipolar operation.

3. Format

status = **AI_VRead** (deviceNumber, chan, gain, reading)

Purpose

Reads an analog input channel (initiates an A/D conversion on an analog input channel) and returns the result scaled to a voltage in units of volts.

Parameters
Input

Table.3.2.3 Input parameters for function AI_VRead

Name	Type	Description
deviceNumber	i16	assigned by configuration utility
chan	i16	analog input channel number
gain	i16	gain setting for the specified channel

Output

Table.3.2.4 Output parameters for function AI_VRead

Name	Type	Description
voltage	f64	the measured voltage returned, scaled to units of volts

chan is the analog input channel number.

Gain is the gain setting for the specified channel. This gain setting applies only to the DAQ device. If an invalid gain is used, NI-DAQ returns an error. In this project, **gain** is set to 1 so that the received voltage is in the range from 0-10V.

4. **Format**

status = **AO_Configure** (deviceNumber, chan, outputPolarity, intOrExtRef, refVoltage, updateMode)

Purpose

Informs NI-DAQ of the output range and polarity selected for each analog output channel on the device and indicates the update mode of the DAQs. If an analog output configuration is recorded which is not a default through the NI-DAQ Configuration Utility, it is not necessary to use AO_Configure because NI-DAQ uses the settings recorded by the NI-DAQ Configuration Utility. However, AO_Configure is used to change the analog output configuration on the fly.

Parameters
Input

Table.3.2.5 Input parameters for function AO_Configure

Name	**Type**	**Description**
deviceNumber	i16	assigned by configuration utility
chan	i16	analog output channel number
outputPolarity	i16	unipolar or bipolar
intOrExtRef	i16	reference source
refVoltage	f64	voltage reference value
updateMode	i16	when to update the DACs

chan is the analog output channel number.

Range: 0 or 1 for Lab PC-1200 analog output device.

outputPolarity indicates whether the analog output channel is configured for unipolar or bipolar operation.

For the Lab PC-1200 analog output device:

0: Bipolar operation (default setting, output range is from –5 to +5 V).

1: Unipolar operation (output range is from 0 to +10 V).

intOrExtRef indicates the source of voltage reference.

0: Internal reference.

1: External reference.

refVoltage is the analog output channel voltage reference value. Each channel is configured to use an internal reference of +10 V (the default) or an external reference.

Range: –10 to +10 V.

updateMode indicates whether an analog output channel is updated when written to:

0: Updated when written to (default setting).

1: Not updated when written to, but updated later after a call to AO_Update

2: Not updated when written to, but updated later upon application of an active low pulse.

5. Format

status = **AO_VWrite** (deviceNumber, chan, value)

Purpose

Accepts a floating-point voltage value, scales it to the proper binary number, and writes that number to an analog output or current channel to change the output voltage.

Input

Table.3.2.6 Input parameters for function AO_VWrite

Name	**Type**	**Description**
deviceNumber	i16	assigned by configuration utility
chan	i16	analog output channel number
voltage	f64	floating-point value to be scaled and written

chan is the analog output channel number.

Range: 0 or 1 for Lab PC-1200

voltage is the floating-point value to be scaled and written to the analog output channel. The range of voltages is set to Unipolar, 0 to 10V.

AO_VWrite scales voltage to a binary value and then writes that value to the DAC in the analog output channel. If the analog output channel is configured for immediate update, the output voltage or current changes immediately.

Before a program is written, it is always important to formulate a state diagram first. Fig.3.2.1 and 3.2.2 below illustrate the sequence for read sensor value and write pump voltage.

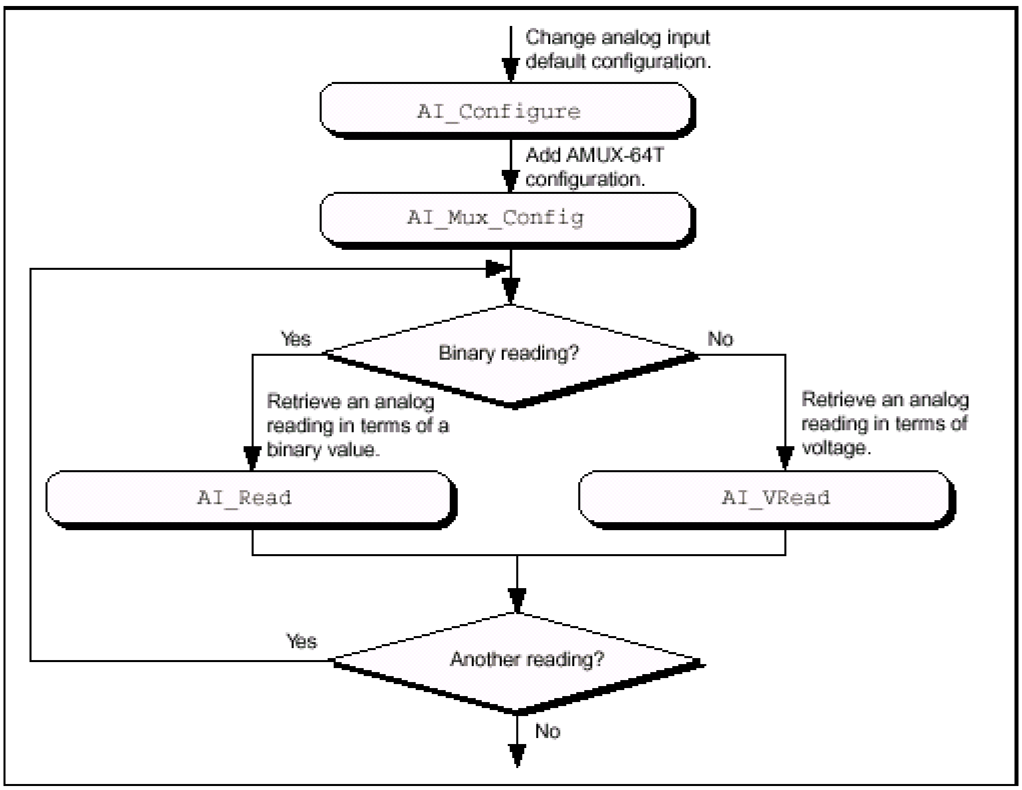


Fig.3.2.1 Read sequence

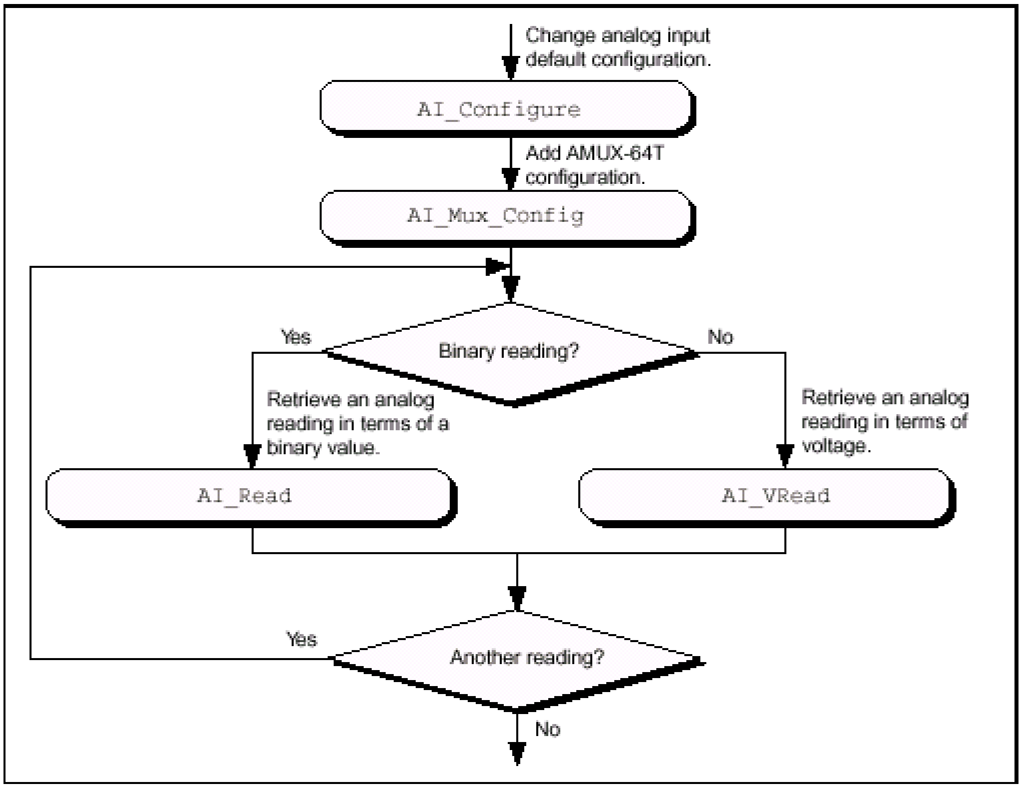


Fig.3.2.2 Write sequence

Following the above state diagram, two programs, TestAI.cpp and TestAo.cpp, are written using Borland C++ builder and successfully tested. TestAI.cpp attempts to read the level sensor and get its voltage as return value. TestAo.cpp attempts to output 4V and then 0V to analog output pump. Please refer to Appendix for the detailed code. A list of procedure was formulated and listed below.

1. Create a project module, TestAI.bpf to manage application code.
2. Create files of .cpp (C++ source code).
3. Set **Options\Application** to **ConsoleApp** creating a 32-bit application.

   To use the NI-DAQ functions, NI-DAQ DLL must be used.

4. Create source file, write the application code and call NI-DAQ functions as typical function calls.
5. Prototype any NI-DAQ routines used in the application. Include the NI-DAQ header file, which prototypes all NI-DAQ routines, as shown : #include "nidaqex.h"
6. Add the NI-DAQ import library nidaq32b.lib and nidex32b to the project module. Put this imported library under the same node as your source code files that use NI-DAQ functions. Also, in the Project option, under “Directories and Conditions”, all the files in folder NIDAQ\include should be added to the project included path and NIDAQ\Lib should also be included under project library path.
7. Compile and run the project.

3.3 Read and Write I/O in Java

Because IEC 61499 function block runtime is in Java, therefore, Java program is needed to control I/O. However, the standard Java class library doesn't support platform-dependent features needed by Lab PC-1200 I/O board. Since I/O can already be controlled using C++ in a Win32 platform, what is required is a link between Java and C++. The Java Native Interface (JNI) is a standard cross-platform programming interface included in the JDK. It enables programmer to write Java programs that can operate with applications and libraries written in other programming languages, such as C and C++.

Using JNI, Java native methods can be written to access an existing library in C++ and make it accessible to Java code.

Dynamic-link libraries (DLLs) in C++ are to be used with JNI. They provide a way to modularize applications so that functionality can be updated and reused more easily. In Windows, DLLs contain functions and data. A DLL is loaded at runtime by its calling modules. DLLs can define two kinds of functions: exported and internal. The exported functions can be called by other modules. Internal functions can only be called from within the DLL where they are defined. In this case, an exported function is needed.

Therefore, the aim is to implement a Java Native Class and make C++ program to read and write I/O into Dynamic-link libraries, which is accessible from native methods in the Java application.

The procedure of implementation is as follows.

1. Implement Java native class

    - Create new project "AIOJni" and assign package name "aiojni".
    - Create new Java native class "JNINativeClass" and the following method was added:

```
public class JNINativeClass {

  public JNINativeClass() {
  }
  native void pumpOutput(int x, double y);
  native double sensorInput(int z);
}
```

The first function calls for *pumpOutput* in the C++ application (to be implemented later) with channel number x and actual analog output voltage y as input parameters. The second function calls for *sensorInput* in the C++ application to read the level sensor voltage specified by the channel number z as input parameter.

- Build JNINativeClass to generate a C header file, “aiojni_JNINativeClass.h”.

2. Create the Dynamic-link libraries

- Create new DLL project “AIO.bpf” and write application in “aio.cpp” in C++ builder. In the application two methods are written which correspond to the method call in “JNINativeClass”. The method names are:

JNIEXPORT void JNICALL Java_aiojni_JNINativeClass_pumpOutput
*(JNIEnv *, jobject, jint AOchannel, jdouble AOvoltage)*

JNIEXPORT jdouble JNICALL Java_aiojni_JNINativeClass_sensorInput
*(JNIEnv *, jobject, jint AIchannel)*

The form is taken straight from the “aiojni_JNINativeClass.h” header file generated earlier. In those methods, various function calls and application codes are written to read and write I/O using NIDAQ similar to the programs, TestAI and TestAO, written in **Section 3.2.2**.

- Include "aiojni_JNINativeClass.h", “jni.h” and “jni_md.h” header files in C++ application.

- Make project and obtain “Aio.dll” dynamic-link library. Move this file under folder: Jbuilder\jdk1.3\bin.

3. Make Java main class

   - A Java main class has to be written for an application to run. In the main class file, the Java native class “JNINativeClass” has to be called and the dynamic-link library, “Aio.dll” , has to be explicitly loaded. Once done, methods in the Java native class can be called and in term methods in dynamic-link library can be called to read and write I/O.

As a test example, a main application with a graphic user interface, as shown below, is written. Through this application, user is able to read and write I/O to both tanks in the coupled-tank.

To read sensor values, the user only need to input channel number to indicate which tank’s sensor is to be read and the value in voltage is displayed at a push of a button. Similarly, to control pump, user need to input channel number to indicate which pump to be controlled and the actual voltage value to be outputted. Since the output voltage is only valid from 0-10V, precaution algorithms are implemented to prevent user from inputting any value outside this range. The complete code is attached in Appendix for reference. A status bar was also implemented to monitor the status of read and write I/O.

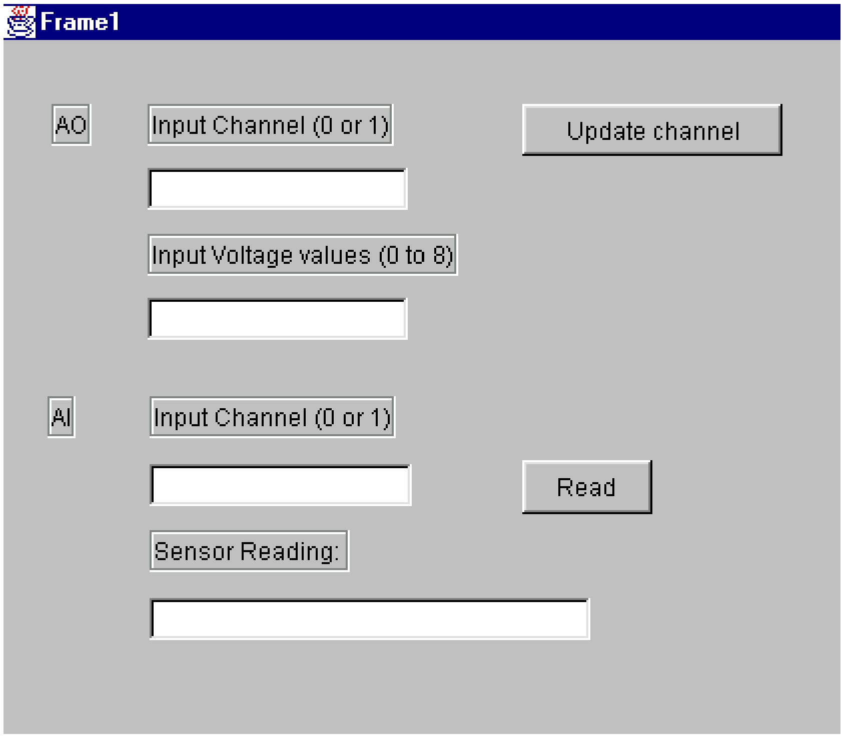


Fig.3.3 Java program to read and write I/O

CHAPTER 4

FUNCTION BLOCK DESIGN

In order to realize a process control example, the most important part is to develop the analog input and analog output function blocks. Analog input function block is to read the value of level sensor in the coupled-tank and monitor the water level. The analog output function block is to write voltage values to the coupled-tank pump to control the water level. Those function blocks have to be implemented under IEC 61499 standard and form the foundation of this project.

4.1 Analog input and output function blocks

Since I/O can be controlled using Java, the next step is to make analog input and analog output function blocks. Java methods to read and write I/O in the previous section have to be modified and implemented in function block Java files. The software tool used to design and generate function blocks and applications is the FBDK Editor by Rockwell Automation.

4.1.1 AO function block

The aim of the function block AO is to output the desired voltage to pump at the particular channel. Therefore, there should be at least two inputs to the function block AO. Shown below in Fig.4.1.1 is the AO function block with interaction establishment sequence.

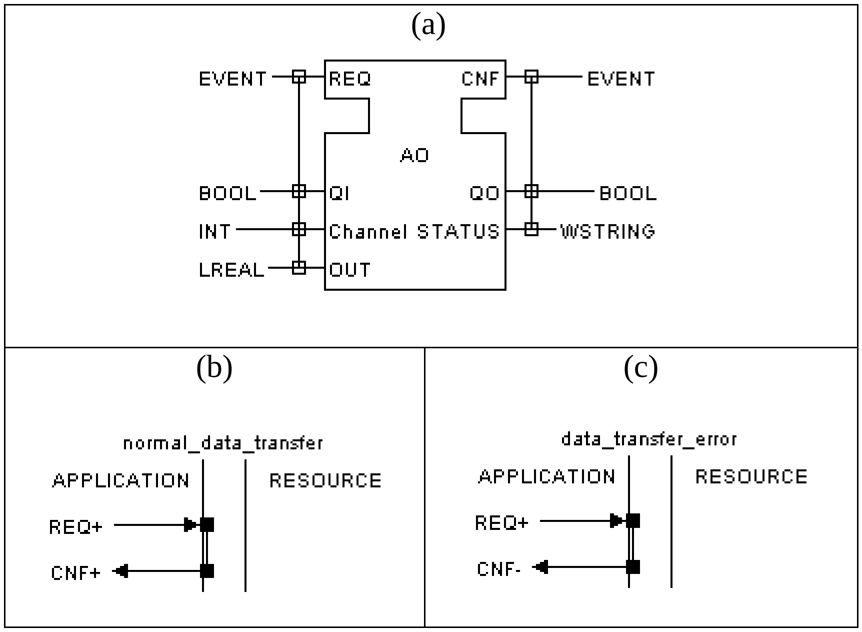


Fig.4.1.1 AO function block and data transfer sequence

The AO function block is a service interface function block. It has one event input, REQ, linked with three data input, QI, Channel and OUT. QI is the event input qualifier and data type boolean. Channel is the channel number of the pumps in coupled tank and of data type integer. OUT is the voltage value to be outputted to pump and is of data type double. It should be in the range of 0-10V. AO function block also has a CNF output event linked with output data QO, and STATUS. QO is the event output qualifier and of type boolean. STATUS is to monitor the status of write data to analog output and of data type string.

Under normal data transfer as illustrated in Fig.4.1.1 (b), the arrival of REQ event will trigger the AO function block to read in data at inputs QI, Channel and OUT. The internal algorithm associated with REQ event will be executed and after execution, confirmation event CNF will be outputted together with data output QO and STATUS. In times of data transfer error shown in Fig.4.1.1 (c), no confirmation event CNF will be outputted and therefore no data output as well.

Using the FBDK editor, a Java file, "AO.java", is generated. Event REQ algorithm has to be written for the function block to function. To execute voltage output, Java Native Class has to be declared and native methods called in REQ algorithm. The difference between code implementation in FBDK and Java applications in **Chapter 3** is that majority of the code are to be written under REQ request event, indicating that only the arrival of REQ event will then trigger the execution of the code. A new Java Native Class, "AOJNIClass.java", is written for AO function block. The new header generated is "fb_rt_Yw_AOJNIClass.h". Both files are placed under the same folder as "AO.java". Using C++ builder, again, a new program "Fbao.cpp" is written to get the new Dynamic library "Fbao.dll" which is placed under folder "Fbdk\bin". A checking algorithm is also implemented to prevent any input voltage data to exceed the range 0-10V. "AO.java" is then compiled and "AO.class" file is generated. Function block AO is ready to be run.

AO function block is then successfully tested and found to be functioning perfectly.

4.1.2 AI function block

The aim of the function block AI is to read sensor voltage at the particular channel. Therefore, there need to be only one channel input to the function block AI. Shown below in Fig.4.1.2 is the AI function block with interaction establishment sequence.

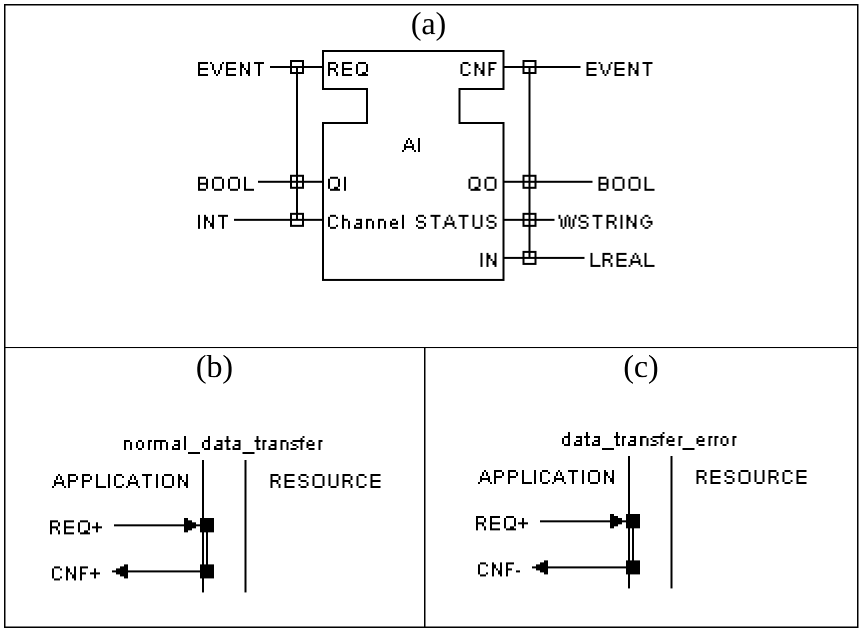


Fig.4.1.2 AI function block and data transfer sequence

The AI function block is also a service interface function block. It has one event input, REQ, linked with two data input, QI and Channel. QI is the event input qualifier and data type boolean. Channel is the channel number of the sensors in coupled tank and of data type integer. AI function block also has a CNF output event linked with three output data QO, STATUS and IN. QO is the event output qualifier and of type boolean. STATUS is to monitor the status of read data from analog input and of data type string. IN is the voltage value read from sensor and is of data type double. It should be in the range of 0-10V.

Under normal data transfer as illustrated in Fig.4.1.2 (b), the arrival of REQ event will trigger the AI function block to read in data at inputs QI and Channel. The internal algorithm associated with REQ event will be executed and after execution, confirmation event CNF will be outputted together with data output QO, STATUS and IN. During data transfer error shown in Fig.4.1.2(c), no confirmation event CNF will be outputted and therefore no data output as well.

Similarly, a Java file, “AI.java”, is generated using FBDK editor. To execute sensor reading, Java Native Class is declared and native methods called in REQ algorithm. A new Java Native Class, “AIJNIClass.java”, is written for AI function block. The new header generated is “fb_rt_Yw_AIJNIClass.h”. Both files are placed under the same folder as “AI.java”. Using C++ builder, a new program “Fbai.cpp” is written to get the new Dynamic library “Fbai.dll” which is placed under folder “Fbdk\bin”. “AI.java” is then compiled to obtain “AI.class” file. AI function block is also successfully tested.

4.2 Conversion function blocks

AI and AO function block can not be used in an application alone. Voltage value can not be an intermediate value in an application. Variables in an application should be free of units thus percentage values are preferred. In addition, the sensors need to be calibrated before they can be used as there might exist a nonlinear relationship between the voltage from sensor and the actual height. If ignored, it might lead to control errors such as offset. As a result, conversion function blocks are needed for both AI and AO function blocks.

4.2.1 AI conversion function block

In order to obtain the actual relationship between sensor voltage readings and actual height, experiments are conducted to collect a number of voltage and height points. The aim is to eventually, derive a function of actual height in terms of voltage. The Java test program “AIOApplication” shown in **Chapter 3 section 3.3** is used in this experiment. A set of data is presented below in Table 4.2.1.1 and 4.2.1.2.

Sensor in Tank 1

Table 4.2.1.1 Sensor calibration data for Tank 1

Height	15	50	100	120	150
Voltage	0.30	1.66	3.5	4.37	5.48
Height	180	200	230	250	270
Voltage	6.64	7.42	8.51	9.24	9.9

Sensor in Tank 2

Table 4.2.1.2 Sensor calibration data for Tank 2

Height	15	50	100	120	150
Voltage	0.7	1.9	3.86	4.6	5.76
Height	180	200	230	250	260
Voltage	6.84	7.68	8.8	9.628	9.9

Matlab is used to obtain a polynomial based on the points collected. Matlab has a function for interpolation and polynomials called “polyfun”. Under “polyfun”, the function “POLYFIT(X, Y, N)” can be used to fit a polynomial of degree N to data X and Y in a least square method. Two Matlab programs, “yangpoly.m” and “yangpoly2.m”, are written using “PLOYFIT” to obtain the relationship between voltage and height in the form of a third degree polynomial. Please refer to Appendix for both programs. Fig.4.2.1.1 and 4.2.1.2 below show the actual curves generated and the corresponding polynomials for two sensors.

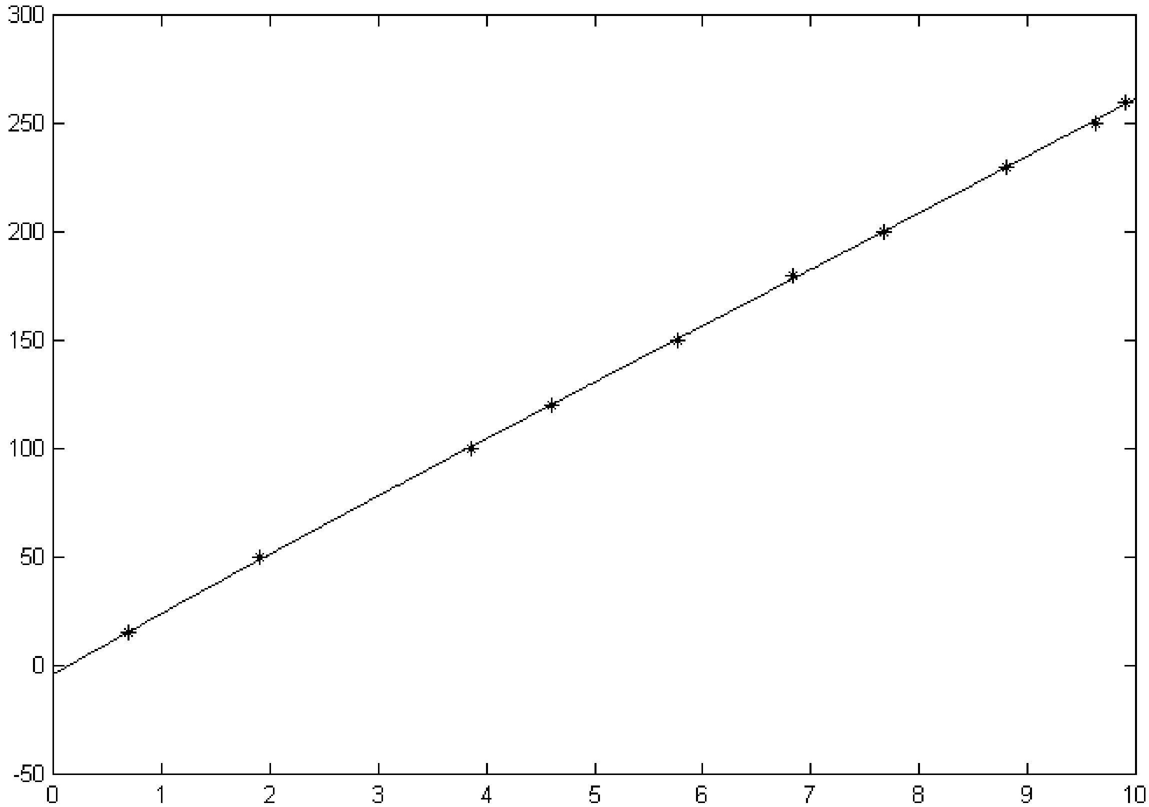


Fig.4.2.1.1 Relationship between height and voltage for sensor in Tank 1

$Y = 0.0374X^3 - 0.4704\,X^2 + 27.5768\,X + 6.3639$ ------------------------(1)

where X is voltage and Y is corresponding height.

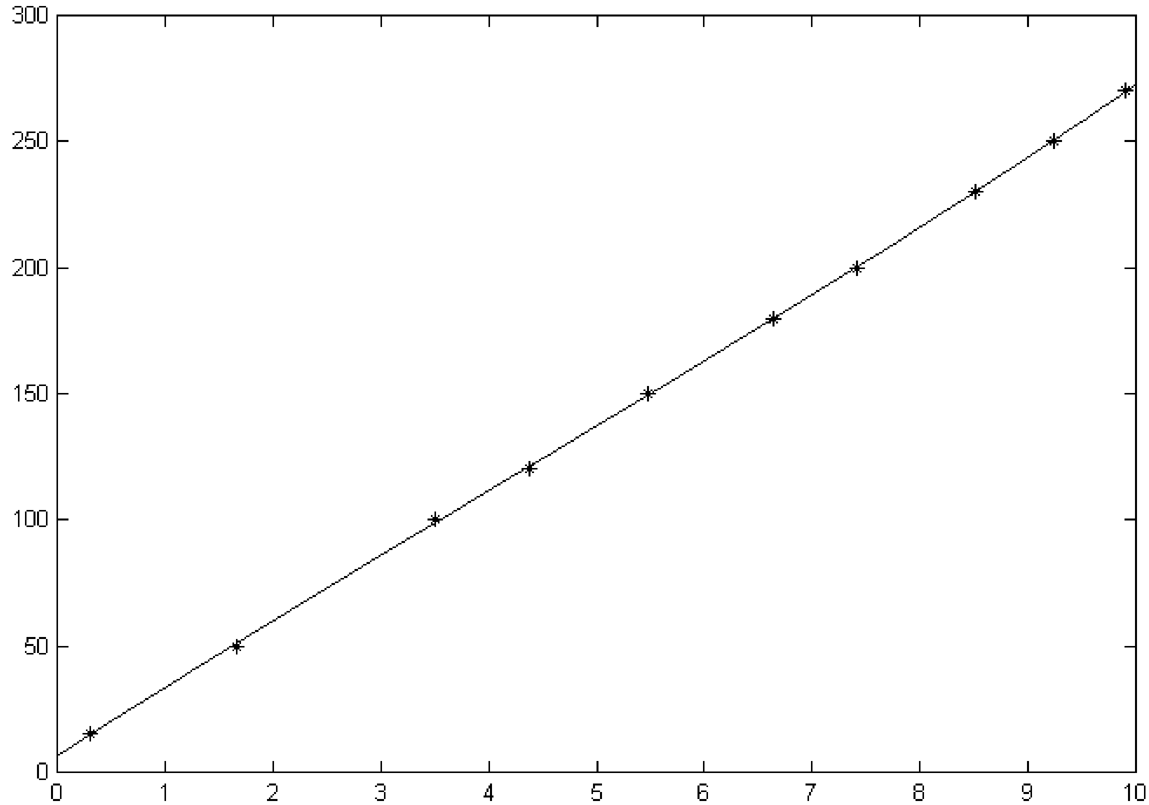


Fig.4.2.1.2 Relationship between height and voltage for sensor in Tank 2

$Y = 0.0204X^3 - 0.3864X^2 + 28.3849X - 4.0769$ ------------------------ (2)

where X is voltage and Y is corresponding height.

One thing to note is that, although the physical scale shown on the coupled-tank is from 0-300mm, voltage reading of Tank 1 sensor saturates at 270mm and voltage reading of Tank 2 sensor saturates at 260mm. Therefore, the maximum height is set at 270mm for Tank 1 and 260 for Tank 2 in this project. With those two equations, given any sensor value in voltage, a corresponding value in height can be found.

Using FBDK editor, a function block AIConv is designed and shown below.

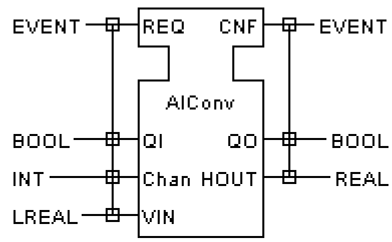


Fig.4.2.1.3 AIConv function block

The aim of this function block is to convert the sensor readings in voltage to actual height in percentage and it should be applied to both sensors. The AIConv function block is a basic function block. It has one event input, REQ, linked with three data input, QI, Chan and VIN. QI is the event input qualifier and data type boolean. Chan is the channel number of the sensors in coupled-tank and of data type integer. This data is necessary because the function block can now be used with both sensors. VIN is the input sensor reading in voltage and of data type double. It also has a CNF output event linked with two output data QO and HOUT. QO is the event output qualifier and of type boolean. HOUT is the height of water in percent after conversion and is of data type float.

REQ algorithm is implemented using equation (1) and (2) shown above. It servers two purposes: (a) to convert voltage to height, and (b) to convert height to percentage with respect to total height. Depending on the input Chan data, different equations will be applied. Please refer to “AIConv.java” in Appendix for full code.

4.2.2 AO conversion function block

It is also necessary to determine the voltage value outputted and the actual send to the pump. Another experiment is conducted with the help of a multimeter attached to the output pins of the Lab PC-1200 I/O board. It is found that the physical output is relatively accurate for both pumps, therefore a linearization algorithm is not necessary. Only a conversion algorithm, which converts percentage value to actual voltage for output, is implemented.

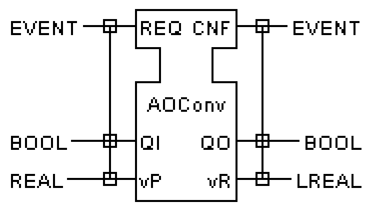


Fig.4.2.2 AOConv function block

As shown above in Fig.4.2.2, the AOConv function block is a basic function block. It has one event input, REQ, linked with two data input, QI and vP. QI is the event input qualifier and data type boolean. No channel number is needed in this case because both pump outputs are the same. This data is necessary because the function block can now be used with both sensors. Data vP is the input voltage reading in percent and of data type float. It also has a CNF output event linked with two output data QO and vR. QO is the event output qualifier and of type boolean. Output vR is the actual voltage in volts after conversion and is of data type double. Detailed code can be found in "AOConv.java" in the Appendix.

4.3 Sensor and pump function blocks

One feature in IEC 61499 is that it allows for encapsulation and reuse of control algorithms. Therefore, a number of function blocks interconnected can be encapsulated to form a composite function block, which will function according to how the consisting function blocks are connected. Such technique allows for function block reuse and reduces the degree of complexity in an application design.

4.3.1 Sensor function block

A composite function block, Sensor, is made of AI and AIConv function blocks.

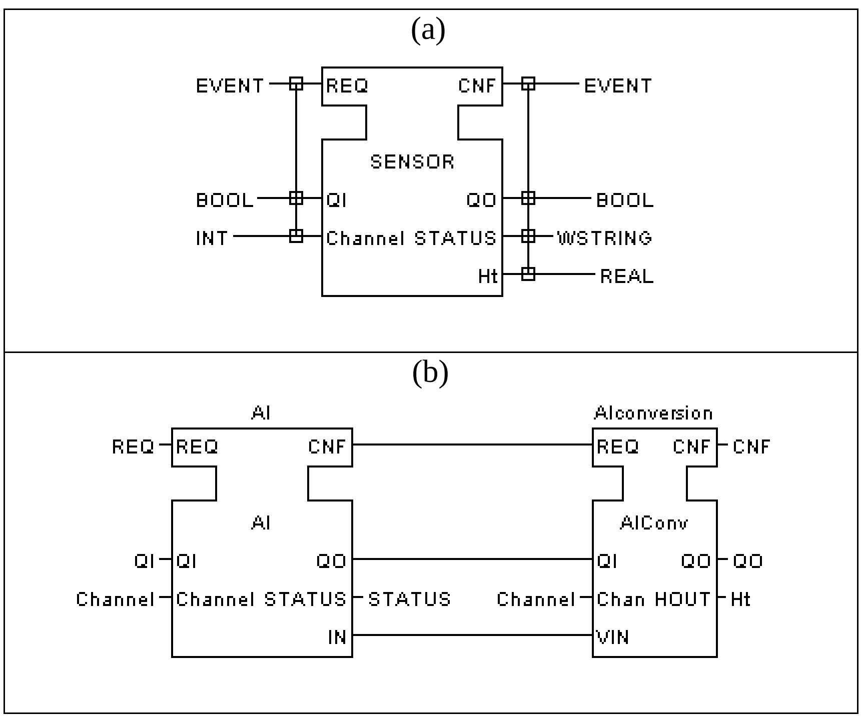


Fig.4.3.1 Sensor function block and components

Fig.4.3.1 above shows the interface of Sensor function block. Its interface is similar to a basic function block. It has one event input, REQ, linked with two data input, QI and Channel. Channel input decides which sensor to read. It also has a CNF output event linked with three output data QO, STATUS and Ht. Output Ht is the actual height in percent after conversion. Fig.4.3.1 (b) shows the components of function block Sensor. They are AI and AIConv function blocks connected with event and data connections. An event REQ of Sensor function block will trigger the reading of level sensor for the coupled-tank by AI function block. After sensor is successfully read, confirmation event AI.CNF together with data output AI.IN from AI function block will arrive at REQ event input and data input VIN of the AIConv function block and cause it to convert the voltage reading in volts to percentage. Upon completion, Sensor output event CNF together with output data Ht, in percent, will be outputted.

This function block Sensor can now be used in different application which make use of the coupled-tanks.

4.3.2 Pump function block

A composite function block, Pump, is also made using AO and AOConv function blocks. Fig.4.3.2 below shows the interface of function block Pump. It has one event input, REQ, linked with three data input, QI, Channel and vP. The data input vP is the output value in percentage. Channel input decides which pump to be controlled. A CNF output event is linked with two output data QO and STATUS.

The components of function block Pump are AO and AOConv function block connected with event and data connections. An event REQ of Pump function block will trigger the conversion of data output vP by AOConv function block to the actual voltage. After conversion is successfully, confirmation event AOConv.CNF together with data output AOConv.vR from AOConv function block will arrive at REQ event input and data input OUT of the AO function block and cause it to output the voltage in volts to pump. Upon completion, Pump output event CNF together with output data STATUS will be outputted.

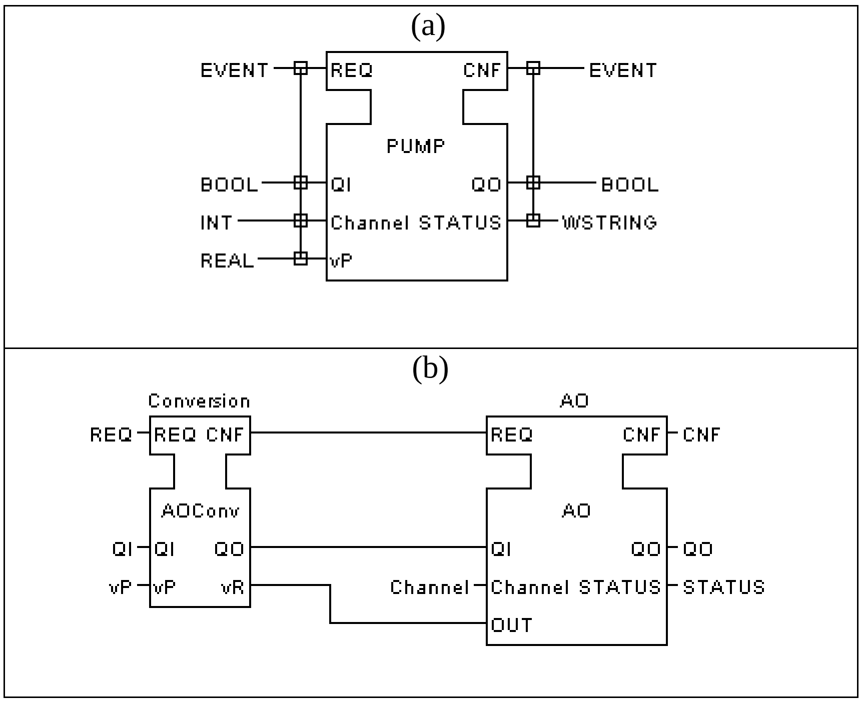


Fig.4.3.2 PUMP function block and components

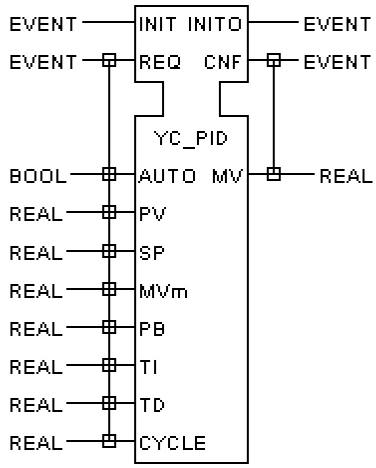
4.4 Other function blocks used

Some other function blocks designed or used in this project are illustrated in this section.

4.4.1 PID function block

A PID controller is a typical controller widely used in many process control applications. A PID function block is therefore an important building block in building a process application. However, how to design a robust PID algorithm is not the main objective of this project, therefore only basic algorithms are used.

Interface



ECC

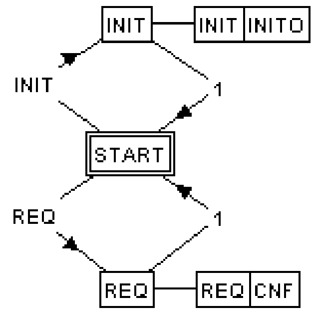


Fig.4.4.1 PID function block

Table 4.4.1 Input and output data for PID function block

Input Data		
AUTO	*BOOL*	0-manual,1-auto
PV	*REA*L	Process Variable
SP	*REAL*	Set Point
MVm	*REAL*	Manipulated Variable in MANUAL mode
PB	*REAL*	Proportional Band (%)
TI	*REAL*	Integral Time (Sec)
TD	*REAL*	Derivative Time (Sec)
CYCLE	*REAL*	Sampling Interval (Sec)
Output Data		
MV	*REAL*	Manipulated Variables (%)

Two mode AUTO and MANUAL are possible for the PID function block. If data input AUTO is true the PID algorithm will be functioning and manipulated variables will be calculated based on past process variable. If mode is set to MANUAL, then PID algorithm will not function and manipulated variables will just be the user input value.

4.4.2 Process variable checking function block

The purpose of this PV_Check function block is to determine whether the values of process variable have become stable with respect to set point. It constantly checks 10 consecutive values of process variable with set point and if each difference between them is less than a tolerance value, then the process is stable. The detailed code is attached in Appendix.

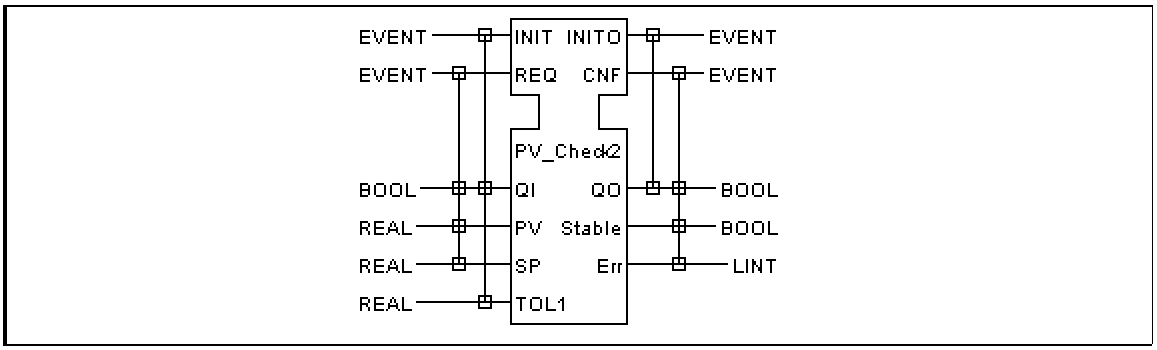


Fig.4.4.2 PV_Check function block

Table 4.4.2 Input and output data for PV_Check function block

Input Data		
QI	*BOOL*	Input event qualifier
PV	*REA*L	Process Variable
SP	*REAL*	Set Point
TOL1	*REAL*	Tolerance
Output Data		
QO	*BOOL*	Output event qualifier
Stable	*BOOL*	Stable or unstable

4.4.3 Human machine interface (HMI) function block

The HMI function block allows user to input certain values such as PID parameters which are needed in an application. It allows user to start and stop the execution of an application and displays process variables real time in the form of a graph for monitor purposes. It is very portable and can be used in various process control applications.

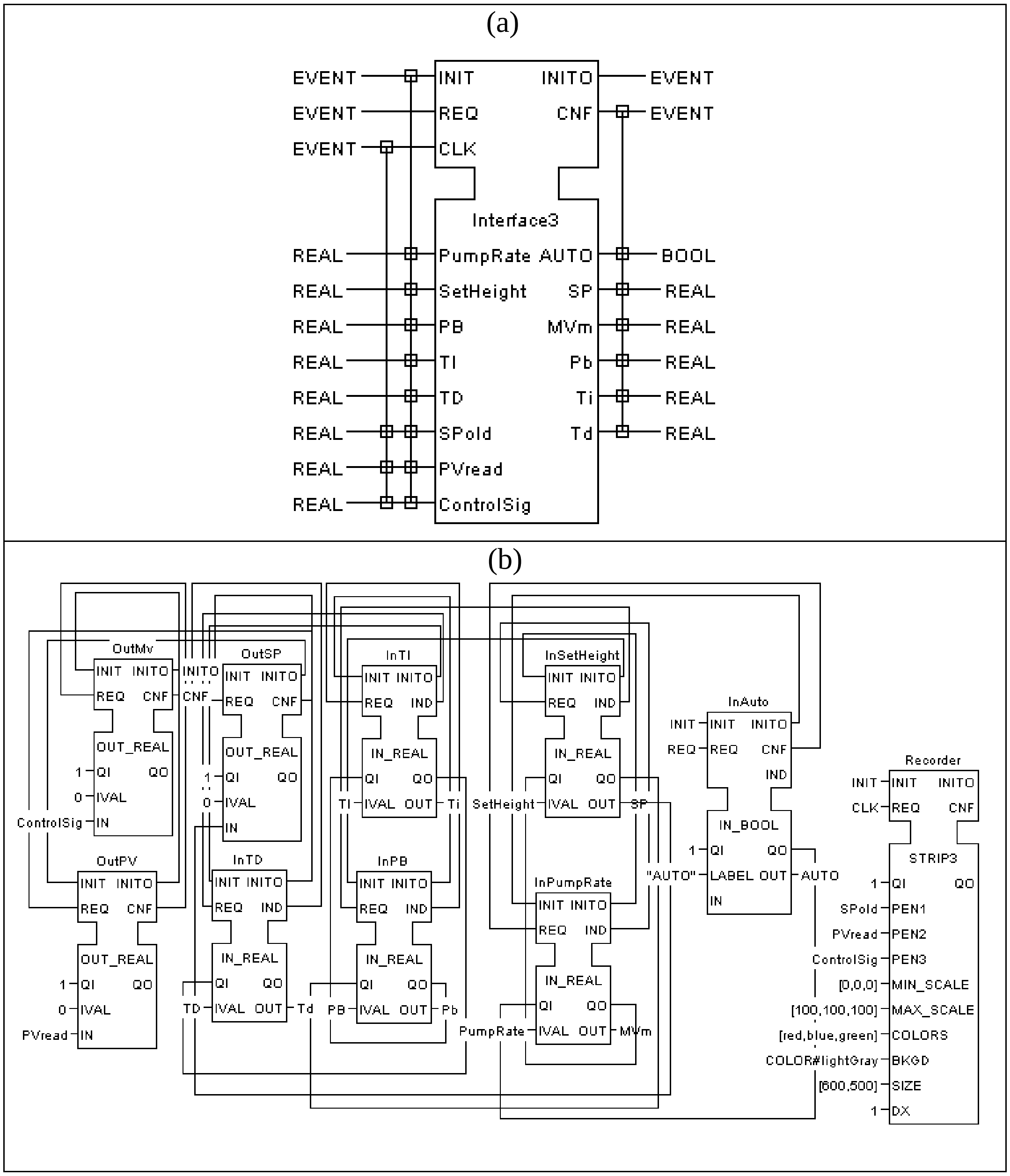


Fig.4.4.3 HMI function bock and components

This HMI function block is a composite function block and consists of IN_REAL, OUT_REAL and STRIP3 function blocks. IN_REAL and OUT_REAL function blocks are both service interface function block which allows for user input and displaying of process data. STRIP3 function block serves to display the received process data in the form of a graph.

4.4.4 Cyclic event generator function block

A process may need to perform certain procedure cyclically and since IEC 61499 function blocks need event input to begin execution of application, a cyclic event generator is necessary. The function of E_Cycle function block is to generate a event request cyclically defined by a time constant specified by user. It is made using a event delay function block with the event output EO connected to event input START. Data input DT is the time constant.

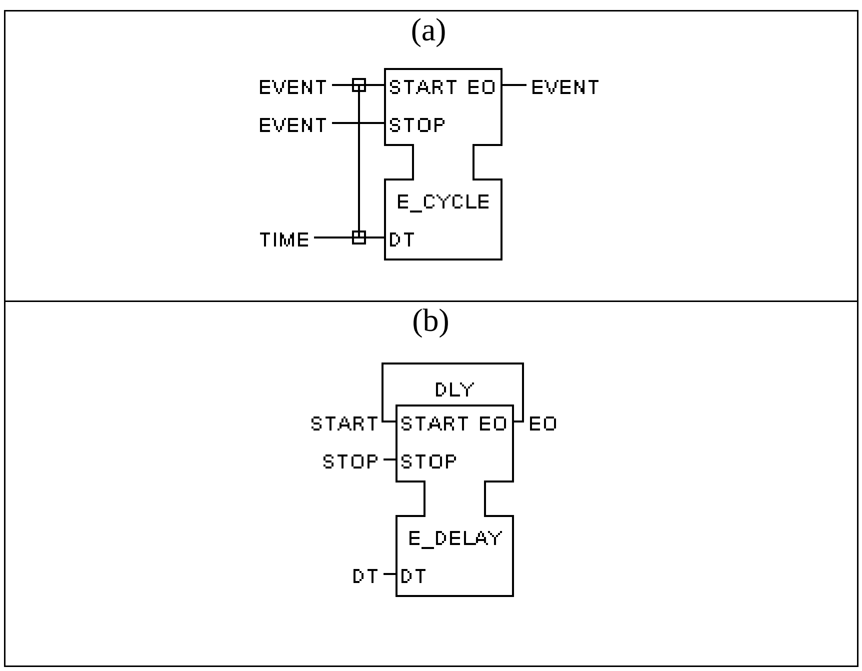


Fig.4.4.4 The E_CYCLE function block and components

CHAPTER 5

REAL TIME PROCESS CONTROL APPLICATION

In order to investigate the implementablity, reliability and performance on process control system, the concepts of IEC 61499 is applied to two process control examples using the coupled-tank. One application involves the water level control of a single input and single output plant with PID controller and is discussed in **Section 5.1**. The other example is a distributed batch control application over network and is illustrated in **Section 5.2**.

5.1 PID control

This section describes water level control of a single input and single output plant with PID controller.

5.1.1 System overview

The two tanks of the coupled-tank can be configured to operate independently by lowering the baffle completely. As each tank has an independent pump and sensor, the apparatus is then two independent single input single output (SISO) plants. For each plant, the manipulated variable is the voltage applied to the pump and the process variable is the water level in the tank. The disturbance is the pressure applied to the water outlet. The block diagram for the plant is given below:

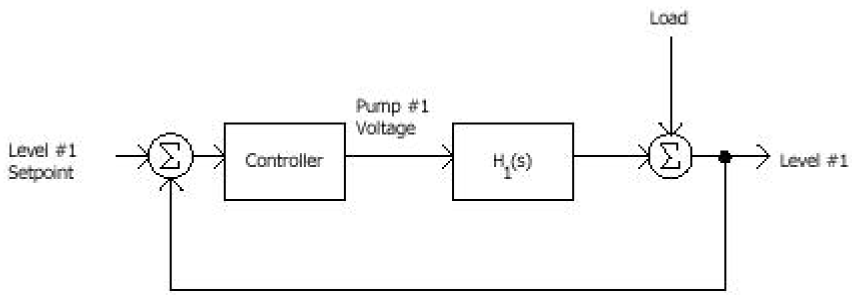


Fig.5.1.1.1 Block diagram for closed loop control

Fig.5.1.1.2 shows the physical connection when Tank 1 of the coupled-tanks is controlled.

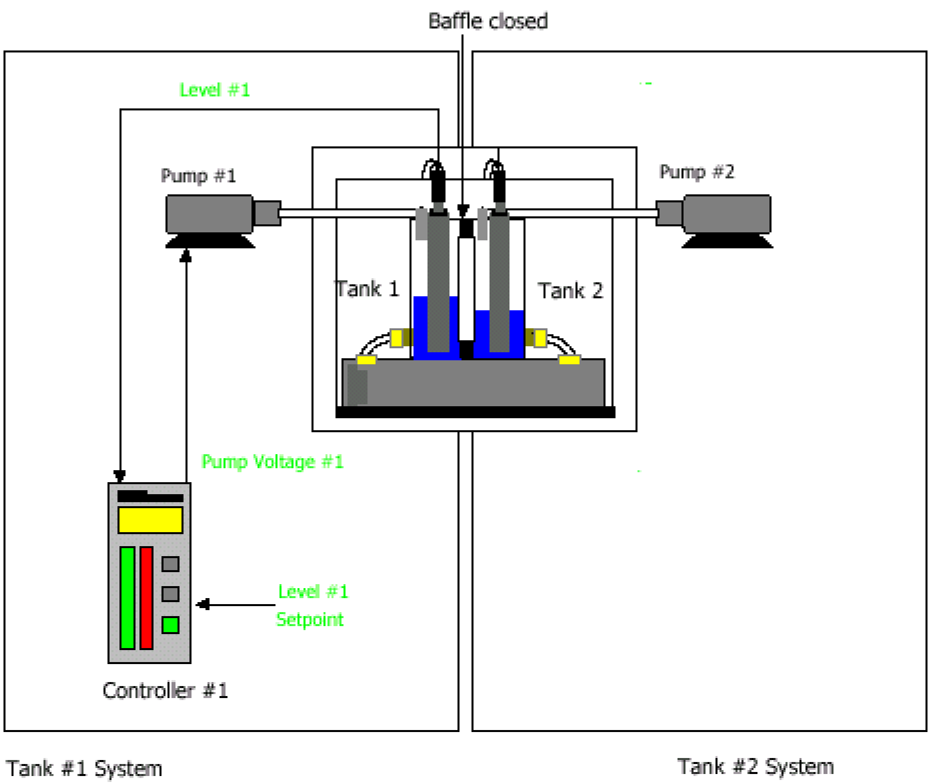


Fig.5.1.1.2 Tank and controller connection

The equipment used in this application are:

a. 1 PC pentium 350 Mhz, 64 MB RAM with Win 98 OS

b. coupled-tank Pp-100

c. connectors

d. I/O board Lab PC-1200 by NI

e. FBDK editor by Rockwell Automation

5.1.2 Application design

The Main part of PID control consists of input from sensor, PID controller and output to pump. An abstractive model is shown below in Fig.5.1.2.

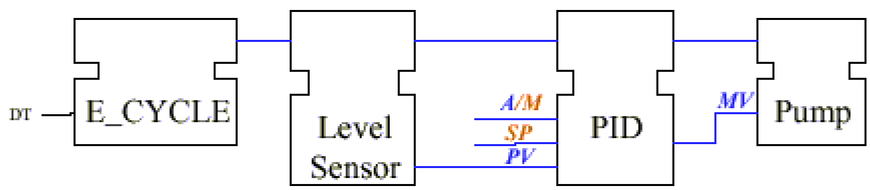


Fig.5.1.2.1 Abstractive model of closed loop control

DT represents the cyclic execution time of this sequence of control. Based on the model, a composite function block “PID_Tank.fbt” is designed using FBDK editor.

Table 5.1.2 below shows the input and output of this function block.

Table 5.1.2 Input and output data for PID function block

Input Variables		
QI	*BOOL*	Input event qualifier
TANK	*INT*	Tank number (1 or 2)
AUTO	*BOOL*	0-manual,1-auto
PumpRate	*REA*L	Manipulated Variable in MANUAL mode (%)
SetHt	*REAL*	Set Point (mm)
PB	*REAL*	Proportional Band (%)
TI	*REAL*	Integral Time (Sec)
TD	*REAL*	Derivative Time (Sec)
Output Variables		
QO	*BOOL*	Output event qualifier
STATUS	*STRING*	Write AO status
SP	*REAL*	Set point (%)
PV	*REAL*	Process variable (%)
MV	*REAL*	Manipulated Variables (%)

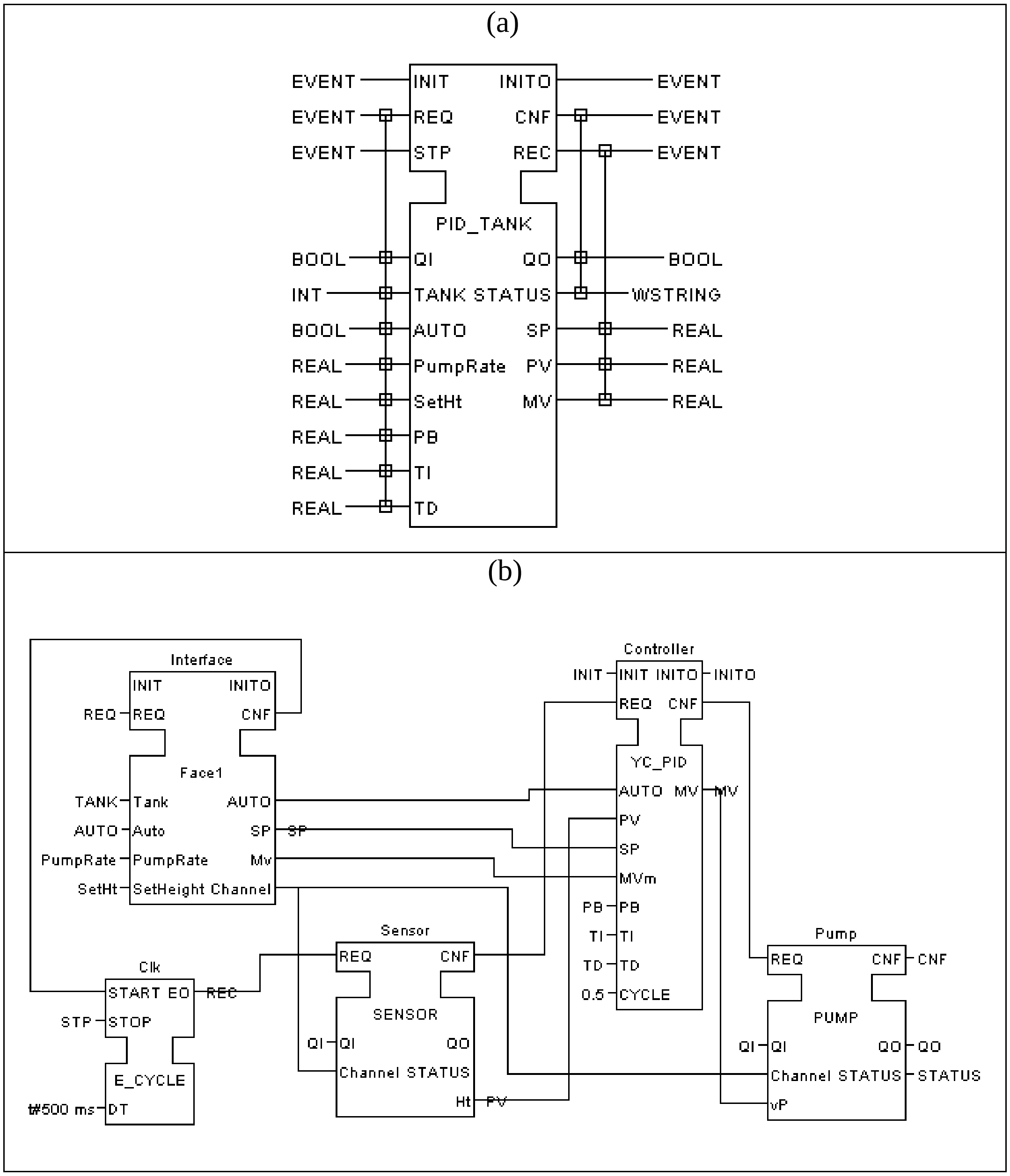


Fig 5.1.2.2 PID_TANK function block and components

The interface is shown above in Fig.5.1.2.2 (a) together with its components in part (b). Cyclic time, DT, is set to be 500ms and so is the sampling period for PID function block.

Subsequently, this function block is executed and tested. However, this is not a application yet and there is also a need for a Human Machine Interface (HMI) to get user input of TANK, AUTO, PumpRate, SetHt, PB, TI and TD. HMI should also display the three output, SP, PV and MV in graph.

An abstractive model is provided in Fig.5.1.2.3.

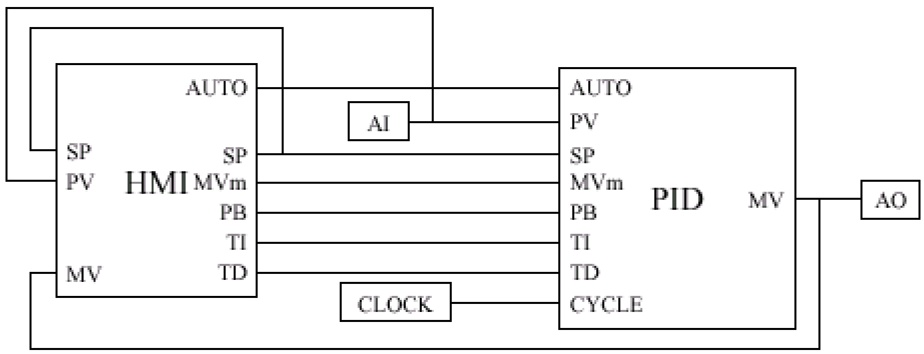


Fig.5.1.2.3 Abstractive model of a system application

Based on the model, an application, “SISOPID_TANK.sys” is implemented and shown below.

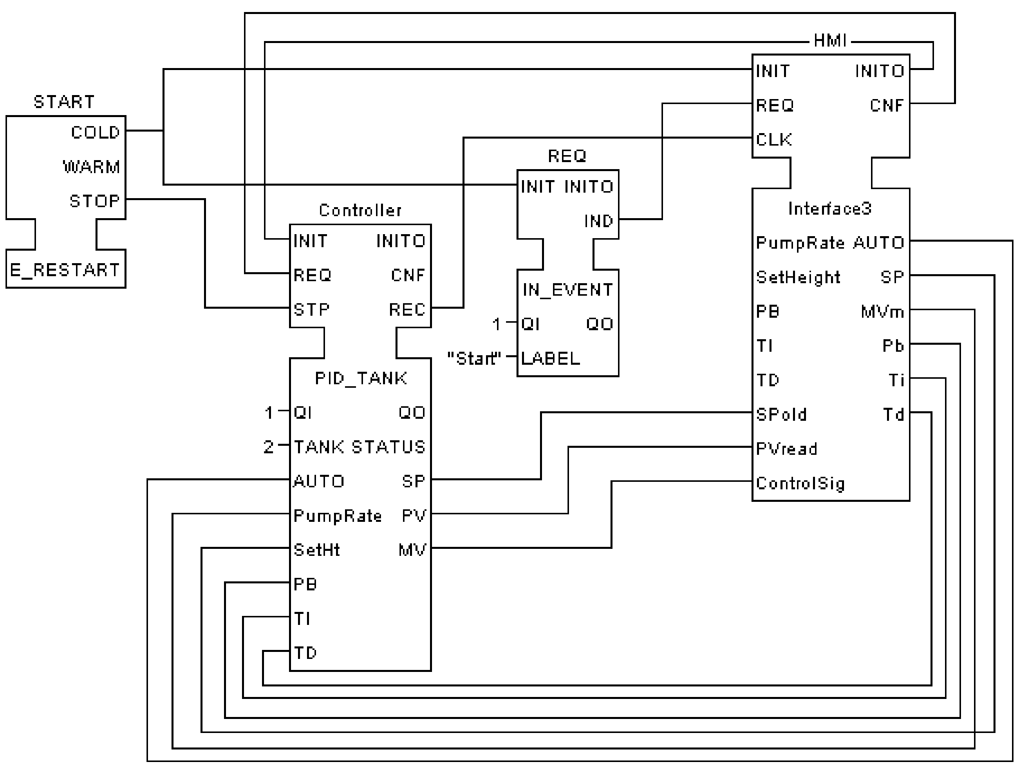


Fig.5.1.2.4 SISOPID_TANK application

In this application, PID_TANK function block is connected with HMI interface3 function block. It allows user to control water level in different tanks, choose AUTO or MANUAL control and enter different PID parameters to get different control results. A real time graph showing SP, PV and MV is also available for monitoring of control results.

5.1.3 PID tuning

To achieve a good control performance, appropriate values of PID parameters have to be chosen. A procedure for PID tuning is adopted and shown in Fig.5.1.3.1.

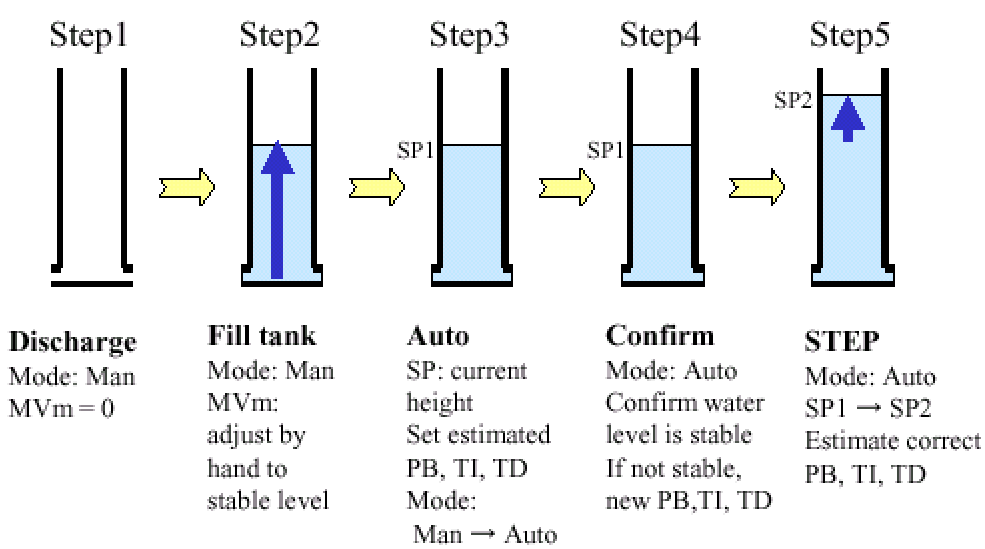


Fig.5.1.3.1 PID tuning procedure

A number of PID parameters are tried and their respective step responses are shown below. The set point is set at 60% of maximum height of the tank. The black line represents set point (SP), blue line represents process variable (PV) and yellow line represents manipulated variable (MV).

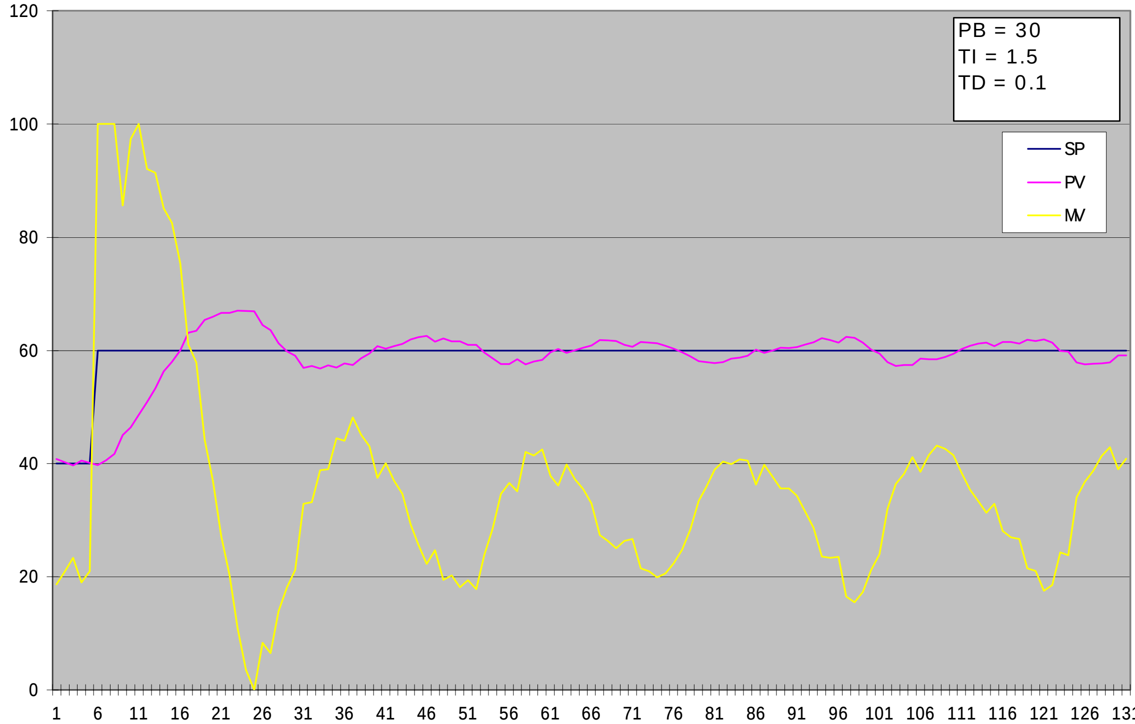


Fig.5.1.3.2 System performance with PB=30, TI=1.5 & TD=0.1

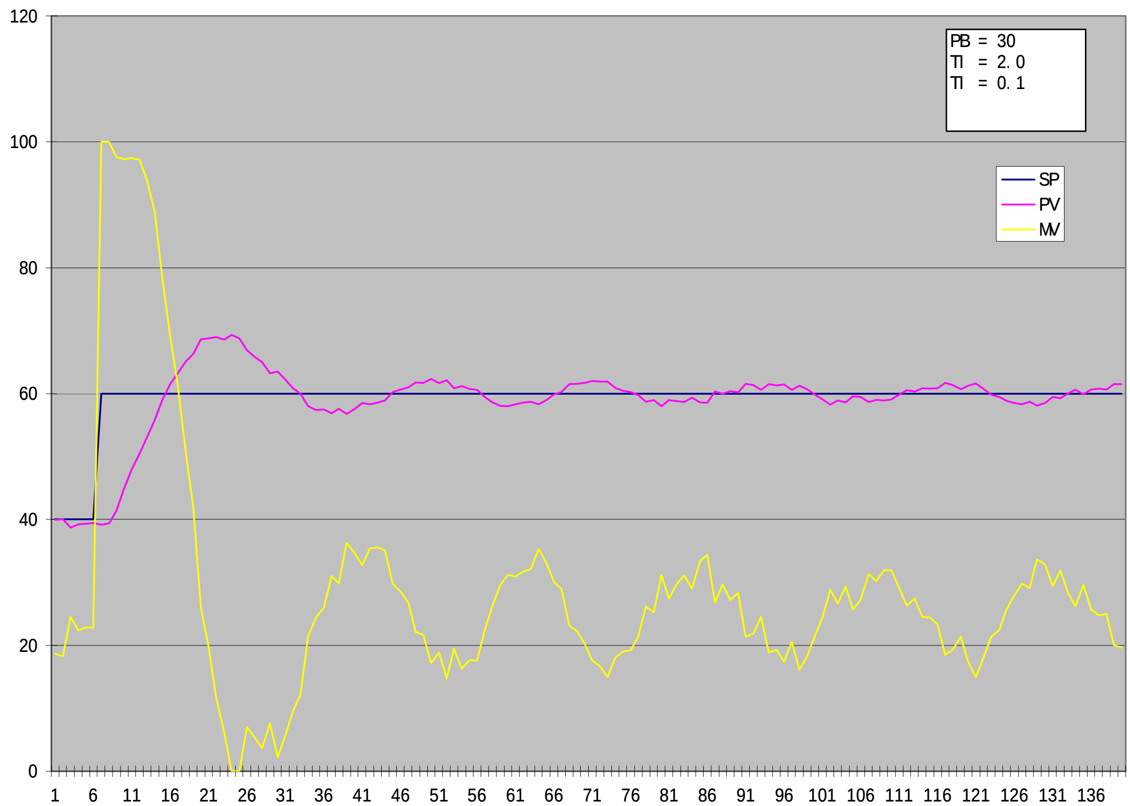


Fig.5.1.3.3 System performance with PB=20, TI=2.0 & TD=0.1

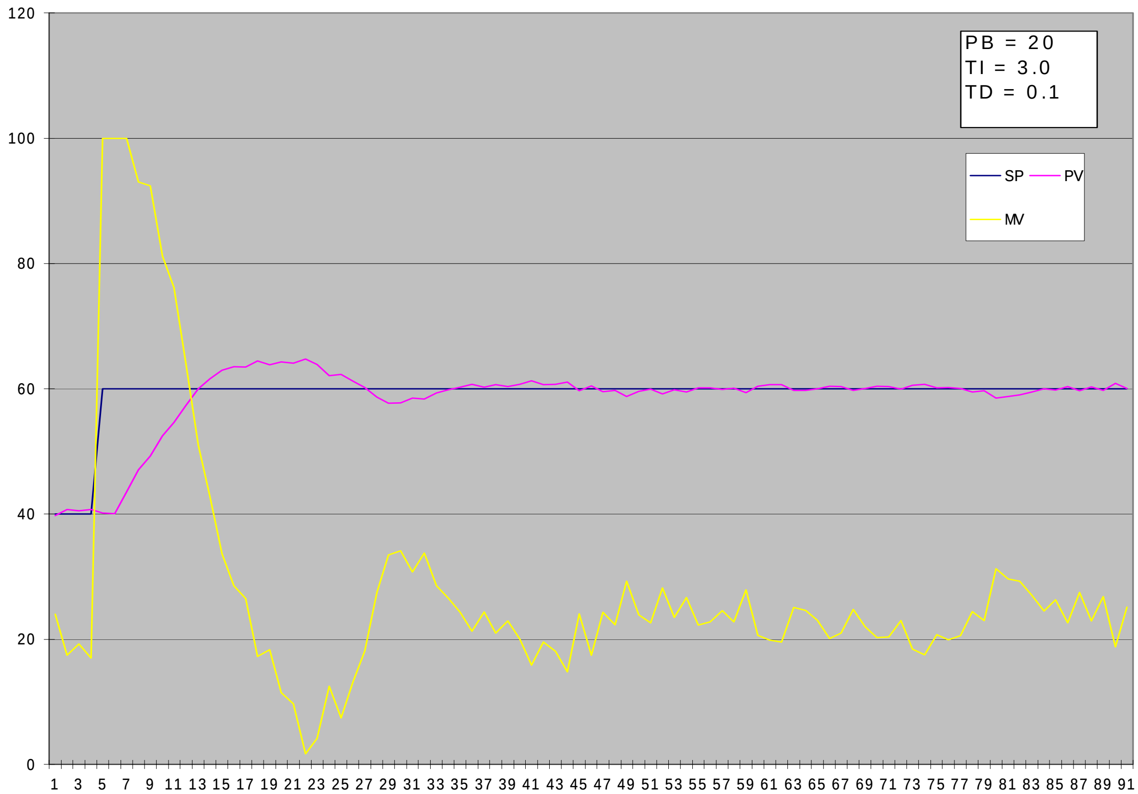


Fig.5.1.3.4 System performance with PB=20, TI=3.0 & TD=0.1

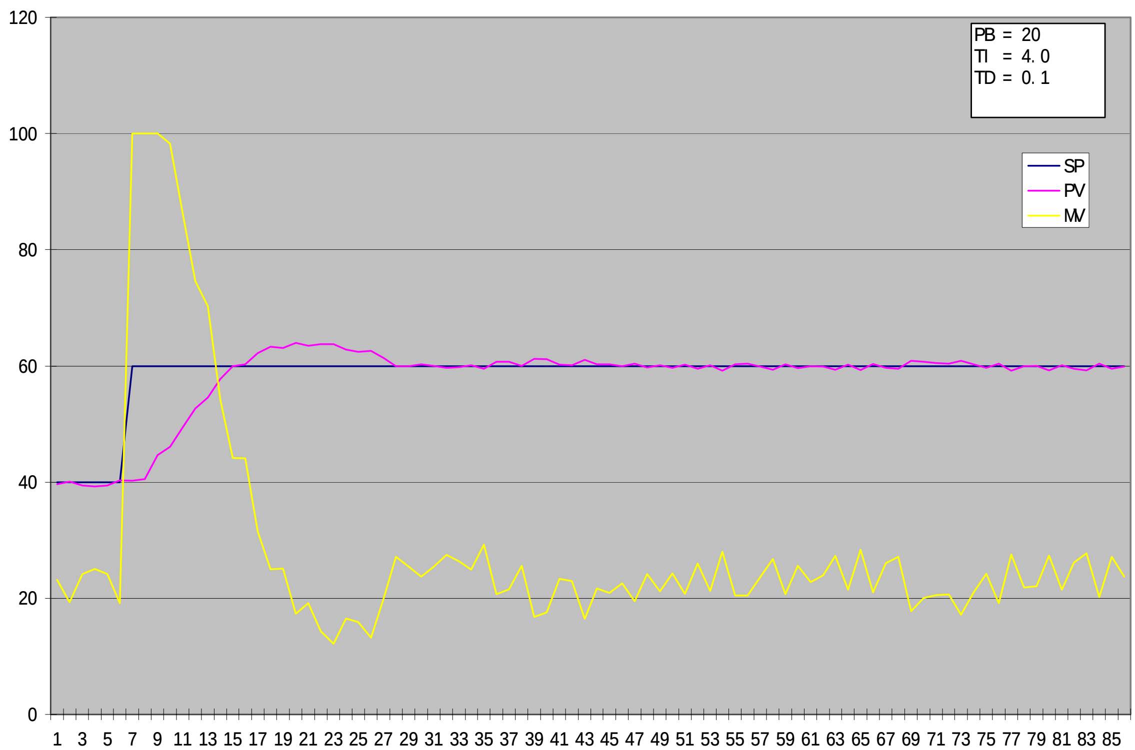


Fig.5.1.3.5 System performance with PB=20, TI=4.0 & TD=0.1

The PID parameters of PB=20, TI=4 and TD=0.1 are chosen as the final values. Based on the step response, the system produces an overshoot of 3% and has no undershoot. It has a rise time of about 7 seconds and a settling time of around 25 seconds. A good control performance is achieved.

5.2 Distributed batch control

A batch control application over a computer network is also implemented to demonstrate the distribution and reconfigure capability of IEC 61499. IEC 61499 provides communication function blocks, which are easy to implement and support access of different networks so that applications can be easily distributed to field devices and controllers.

5.2.1 System overview

Fig.5.2.1 below shows the physical model of this application. Three PCs will be connected together via a network and will act as central controller, controller 1 and controller 2. Controller 1 will be connected physically to Tank 1 of the coupled-tank and controller 2 connected to Tank 2 of the coupled-tank. A batch control sequence is to be implemented in central controller, which controls the sequence of batch control execution. Localized PID control application is to be implemented in controller 1 and controller 2 respectively. Their purpose is to receive instruction from central controller and to fill up water tank as instructed. Two local controllers will communicate with central controller via network.

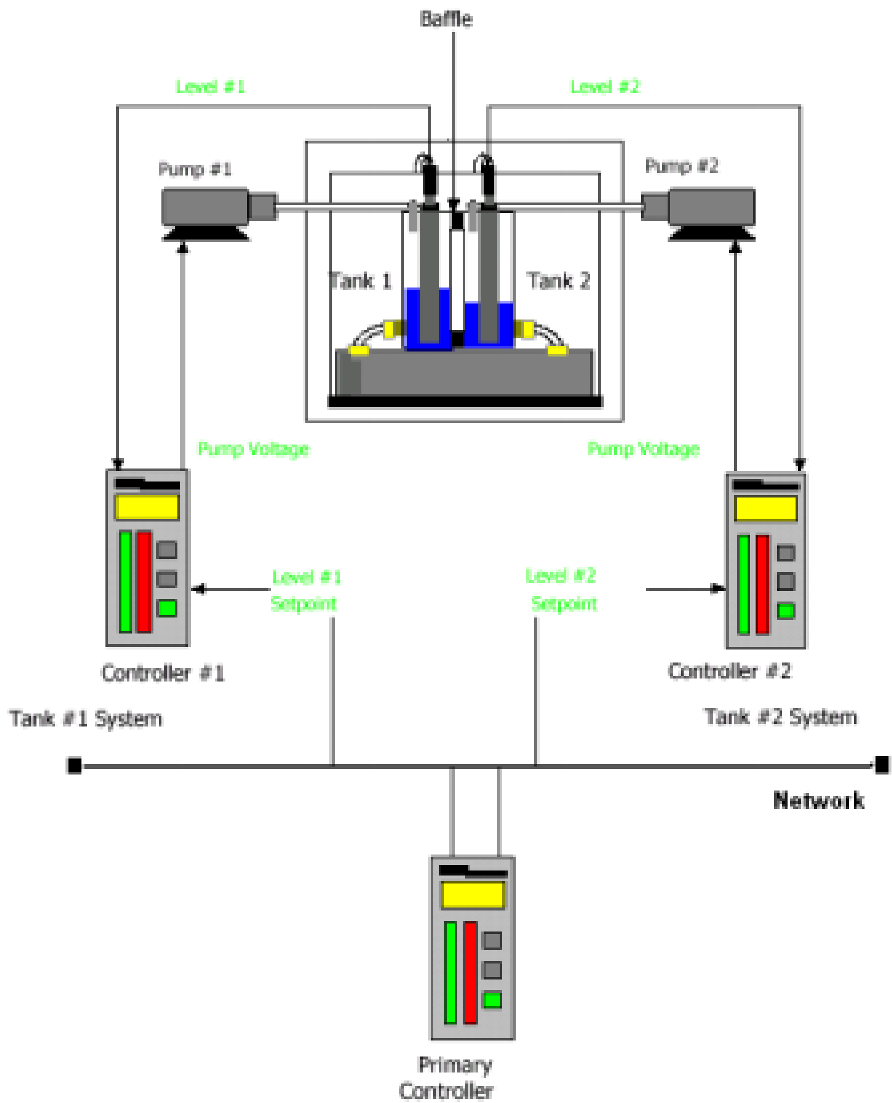


Fig.5.2.1 Physical model for batch control

The equipment used in this application are:

1. 3 x PC pentium 350 Mhz, 64 MB RAM with Win 98 OS
2. coupled-tank Pp-100
3. connectors and Ethernet cables
4. 2 x I/O board Lab PC-1200 by NI
5. FBDK editor by Rockwell Automation
6. OfficeConnect Ethernet Hub 4, 10 BASE T, by 3 COM

The three PCs are connected together by hub to form a network and IP addresses have to be assigned to each PC.

Table 5.2.1 IP address assignment for PC

PC	IP address	Subnet mask
PC1 (central controller)	172.20.20.172	225.225.0.0
PC2 (controller 1)	172.20.20.173	225.225.0.0
PC3 (controller 2)	172.20.20.174	225.225.0.0

5.2.2 Application design

In the central controller, a batch sequence event generator function block is designed. It acts as the brain of this application and schedules the execution of events.

The batch sequence for this application is set to be as follows:

a. Fill Tank 1 to 200mm and maintain

b. Fill Tank 2 to 150mm and maintain

c. Discharge both tank

The function block “BATCH_CTRL_SEQ.fbt” is shown in Fig.5.2.2.1 with its execution control chart. Different sates will be executed depending on event condition. START event will trigger the execution of the batch sequence and goes to next state FILL_TK1 where event SET1 will be outputted together with data output AUTO1, SP1 and MVm1 to controller 1 via network. Controller 1 will then start the filling of tank 1 to level 200mm and at the same time batch control will go to CHECK_TK1 state where water level at Tank 1 will be monitored. Once the level is stable, state WAIT_TK1 will be executed where a delay of 3 sec will be performed. Once finished, a TM1 event will set the state to FILL_TK2 when where event SET2 will be outputted together with data output AUTO2, SP2 and MVm2 to controller 2 via network.

Controller 2 will then start the filling of tank 2 to level 150mm and at the same time batch control will go to CHECK_TK2 state where water level at Tank 2 will be monitored. Water level at Tank 1 is constantly being monitored and an event UNSTABLE1 will set the state back to FILL_TANK1 state and also MAINTAIN state the current water level in Tank 2 will be maintained while water level in Tank 1 is being adjusted. After both water levels are stable in Tank1 and Tank2, WAIT_TK2 state will be executed again to wait for 3 sec. Finally, both tank will be discharged as AUTO1, MVm1, AUTO2 and MVm2 are outputted to set pump rate for both tank to 0V.

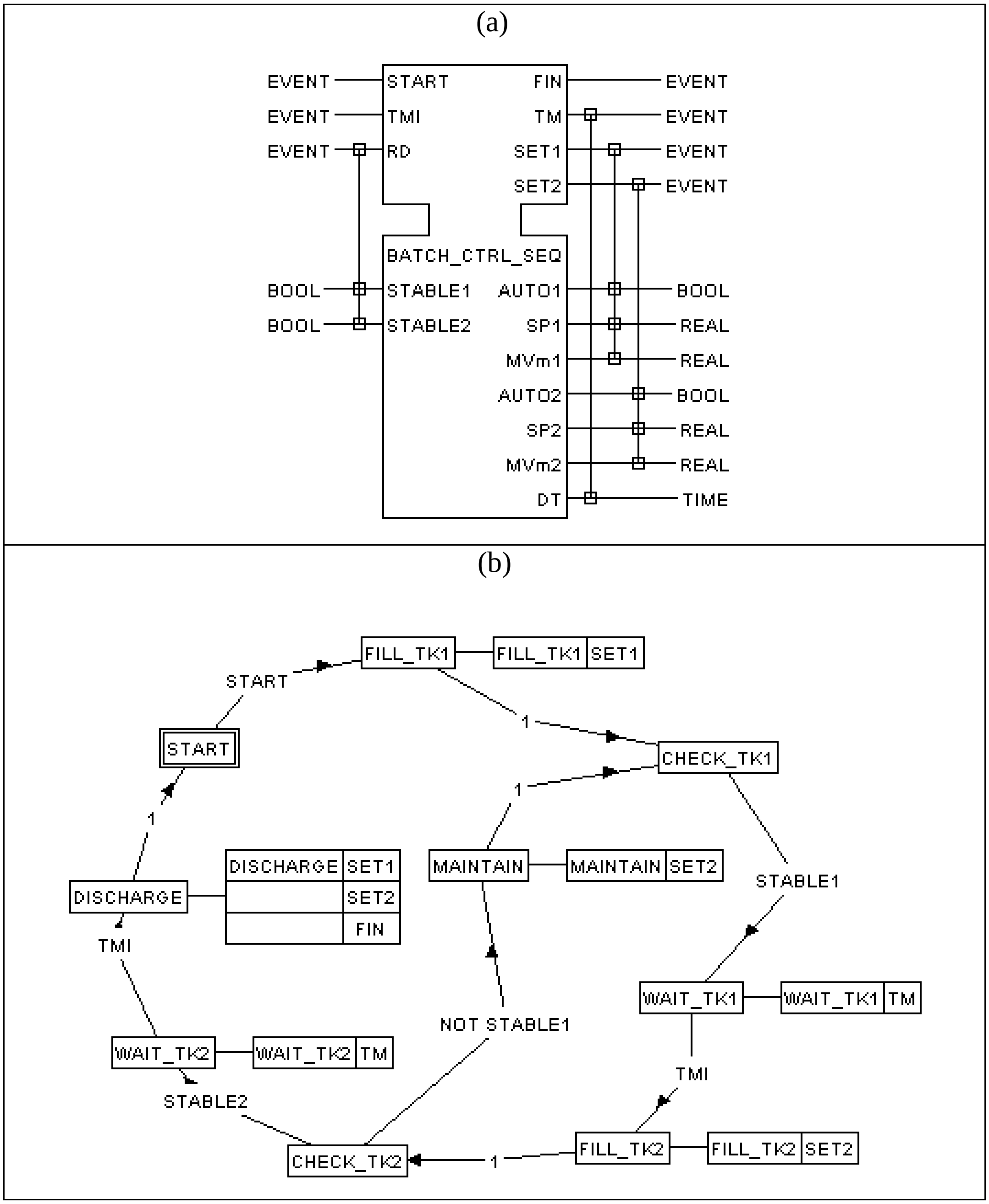


Fig.5.2.2.1 BATCH_CTRL_SEQ function block & batch control sequences

A composite function block “BATCH_CTRL.fbt” is also designed to introduce the delay required in the WAIT state.

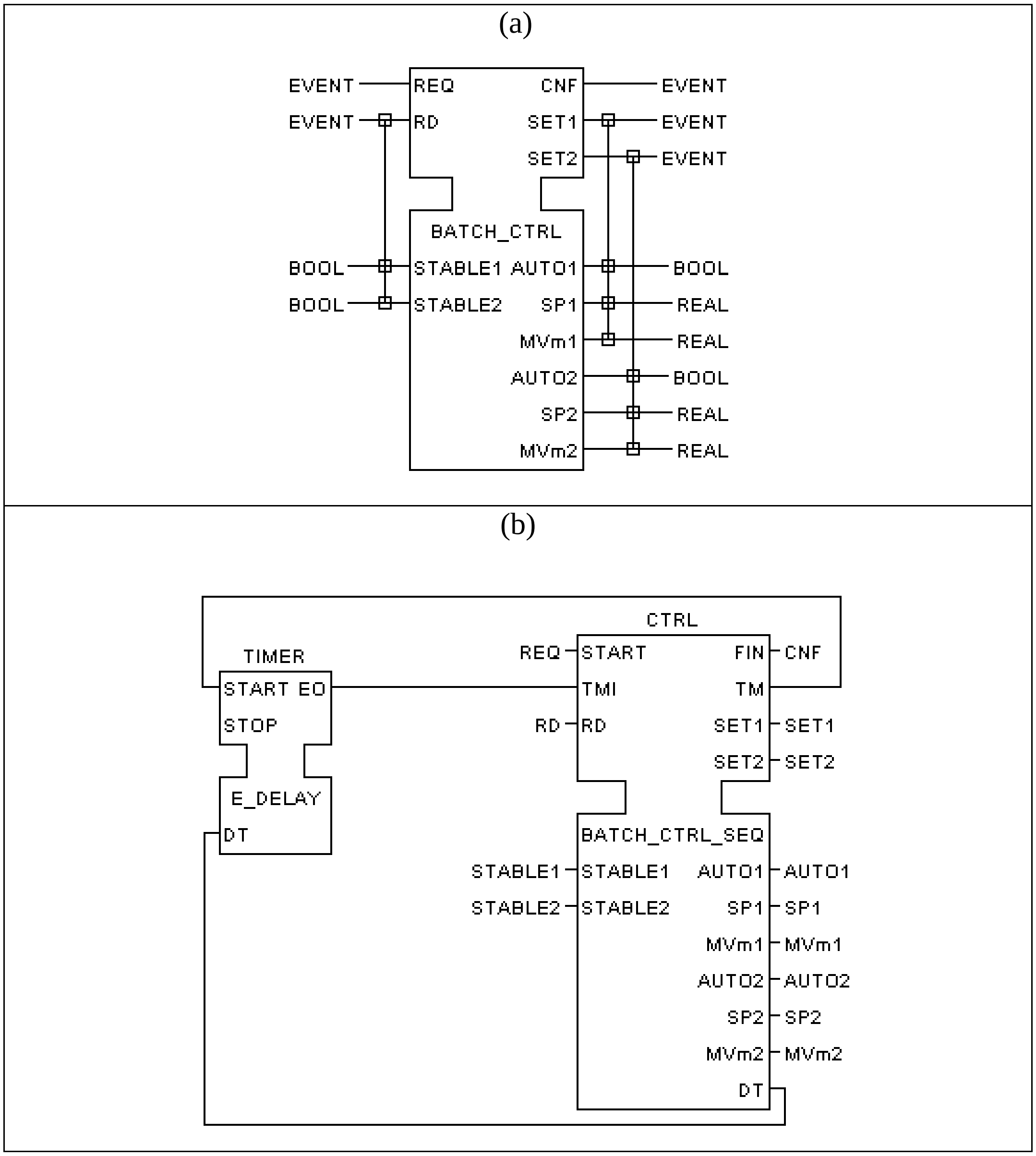


Fig.5.2.2.2 BATCH_CTRL function block and components

After the batch control sequence function block is implemented, the abstractive model for the application is considered.

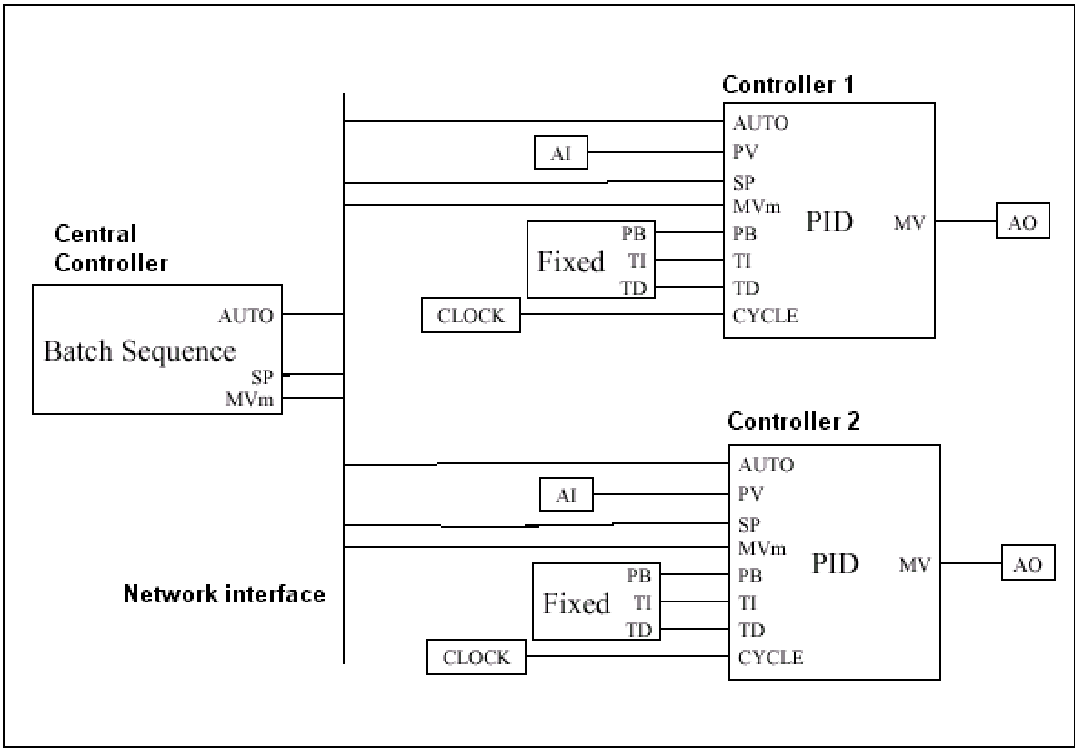
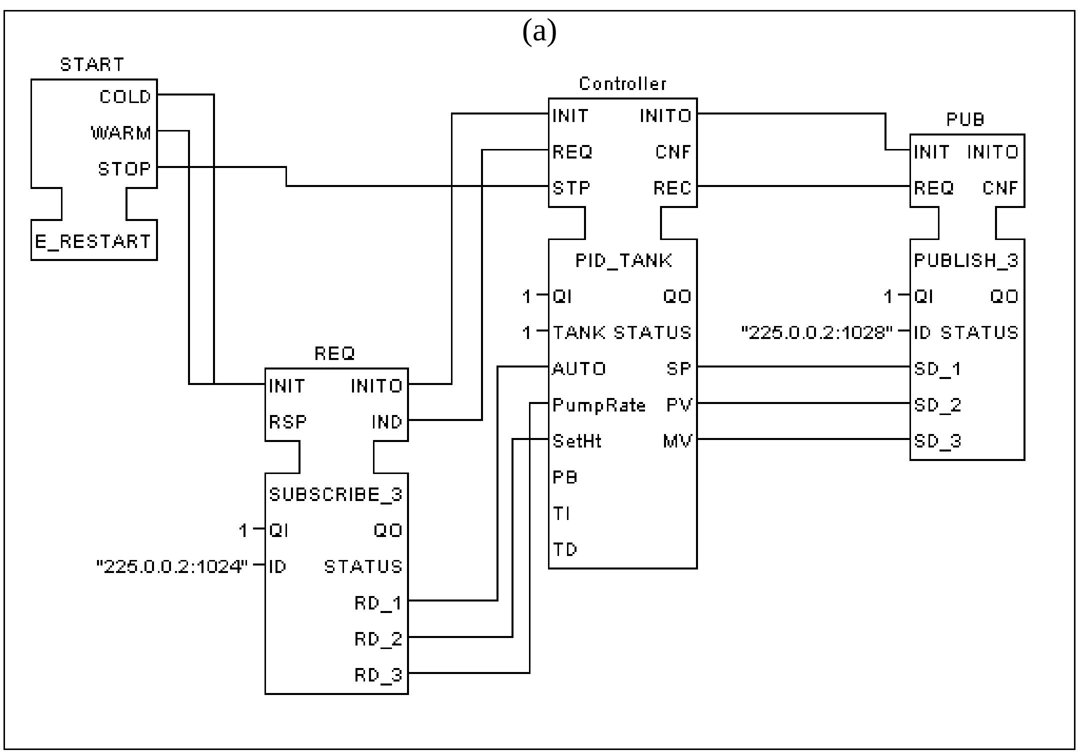


Fig.5.2.2.3 Abstractive model for batch control

Based on the abstractive model shown in Fig.5.2.2.3, the application to be implemented on Controller 1, “Batch_contoller1.sys”, is designed and shown below in Fig.5.2.2.4.



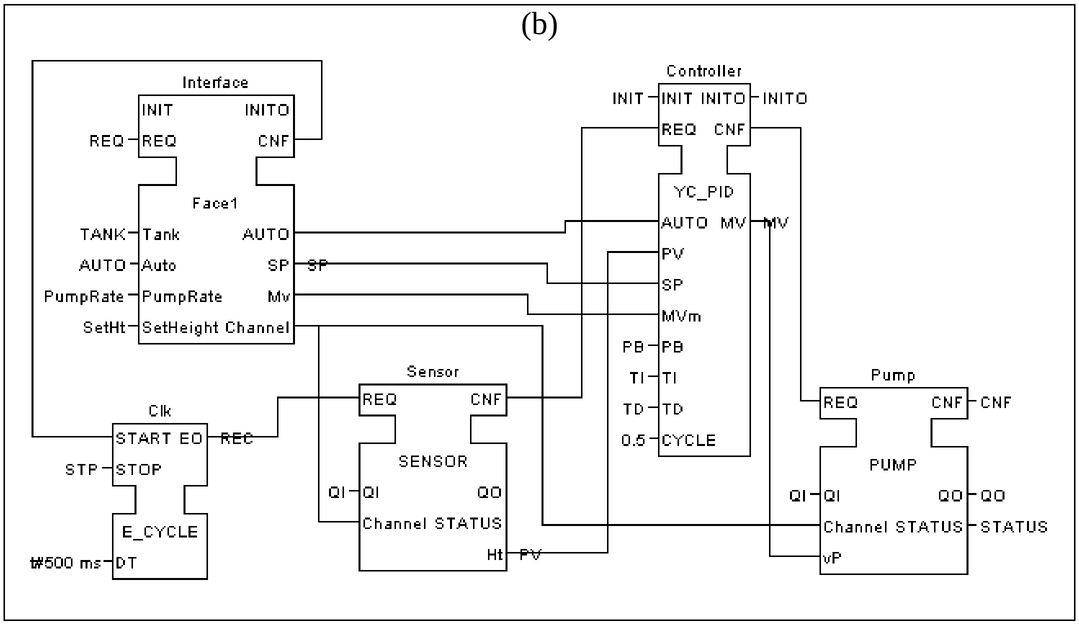


Fig.5.2.2.4 Batch_contoller1 application with components

A set of PUBLISH and SUBSCRIBE function block are the only network interface necessary for this application. During execution, SUBSCRIBLE_3 function block will constantly look for data published at address "255.0.0.2:1024" by Central controller to get SET 1 event request and AUTO1, SP1 and MVm1 data values. PUBLISH_3 function block will publish data values SP1, PV1 and MV1 to the address "255.0.0.2:1028" for Central controller to monitor and display. This application has to be run in Controller 1 PC.

It can be seen that the composite function block PID_TANK, which is used for PID level control and tuning in **Section 5.1.2**, is used again in the batch control application. Its components are displayed again in Fig.5.2.2.4 (b). It reflects the portability and reconfigurabilty of IEC 61499.

The application to be implemented on Controller 2, "Batch_contoller2.sys", is also shown in Fig.5.2.2.5. Its only difference from "Batch_contoller1.sys" is the new IP address used by PUBLISH_3 and SUBSCRIBE_3 function blocks.

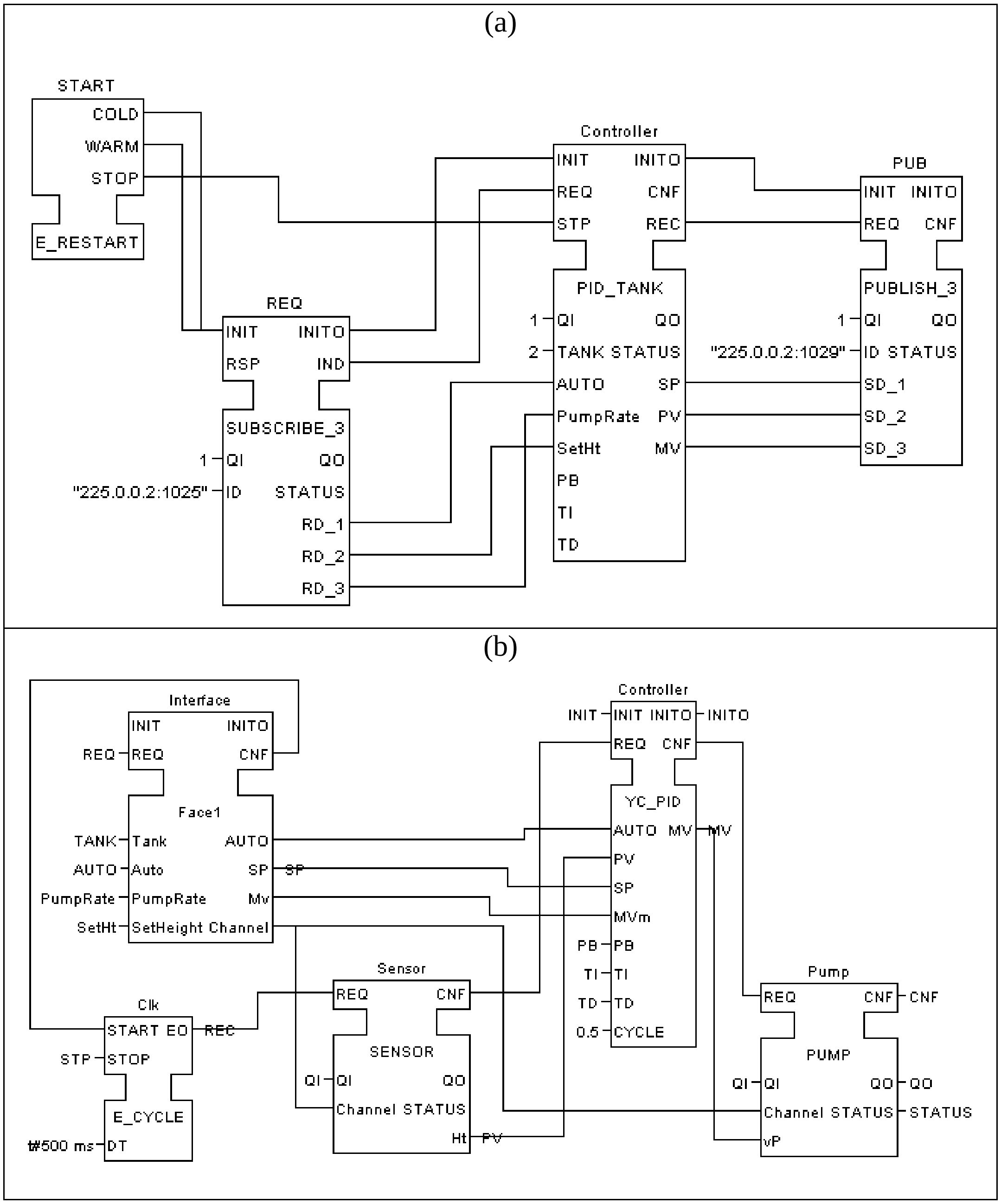


Fig.5.2.2.5 Batch_contoller2 application and components

The central batch application in Central controller, "Batch_Central.sys" is also designed.

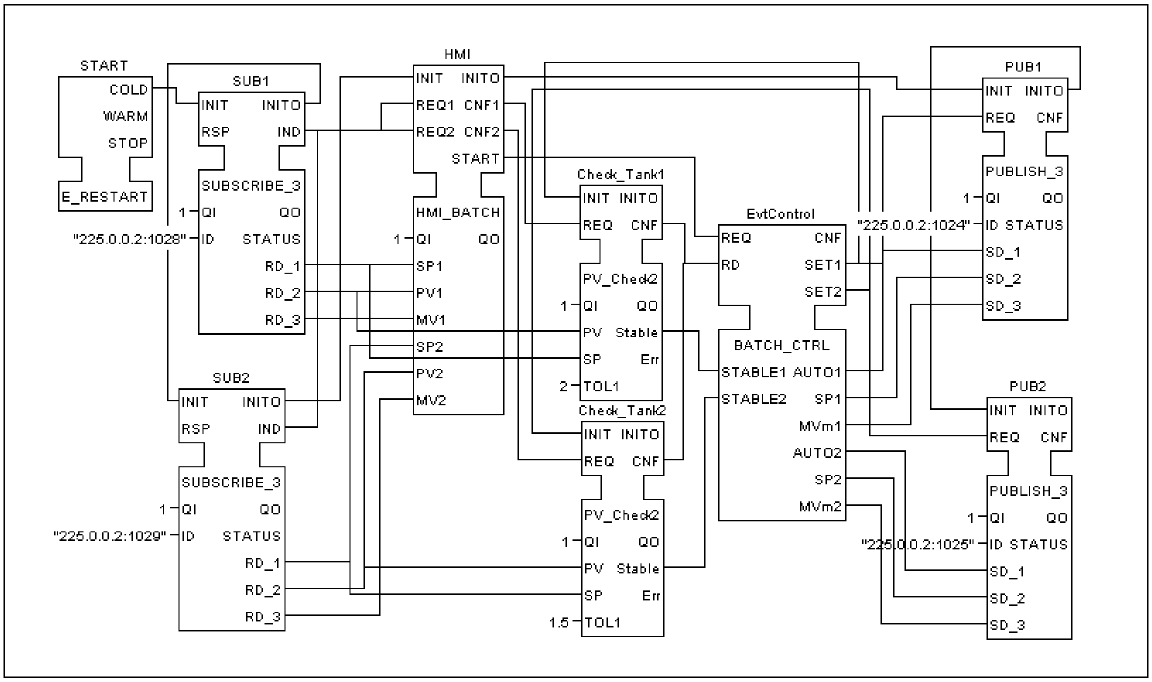


Fig.5.2.2.6 Batch_Central application

In this central application, PUBLISH function block, PUB1, broadcasts the data AUTO1, SP1 and MVm1 to address "255.0.0.2:1024" for Controller 1 to pick up. Note that this number corresponding to the address of the SUBSCIBE function block in Controller 1. On the other hand SUBSCIBE function block, SUB1, gets data SP1, PV1 and MV1 at address "255.0.0.2:1028" for monitoring and checking purpose. This address also correspond to the address of PUBLISH function block in Controller 1. Similarly, PUB2 and SUB2 function block is used with Controller2. PV_CHECK function block is implemented here to determine whether the water level is stable relative to the set points.

5.2.3 Implementation and results

All three applications are implemented in respective PCs and tested. Different tolerance levels for PV_CHECK function block are tested to achieve a smooth control sequence. The application is first run with no disturbance and later run with disturbance in the form of a pressure applied to the water outlet in Tank1 when Tank 2 is filling up. The application is able to detect the change in the water level in Tank 1 due to the disturbance and halt the sequence. It goes back to re-adjust Tank 1 water level and at the same time maintaining the previous water level before disturbance at Tank 2. Attached below are the performance data. Black line represents set point (SP), blue line represents process variable (PV) and yellow line represents manipulated variable (MV).

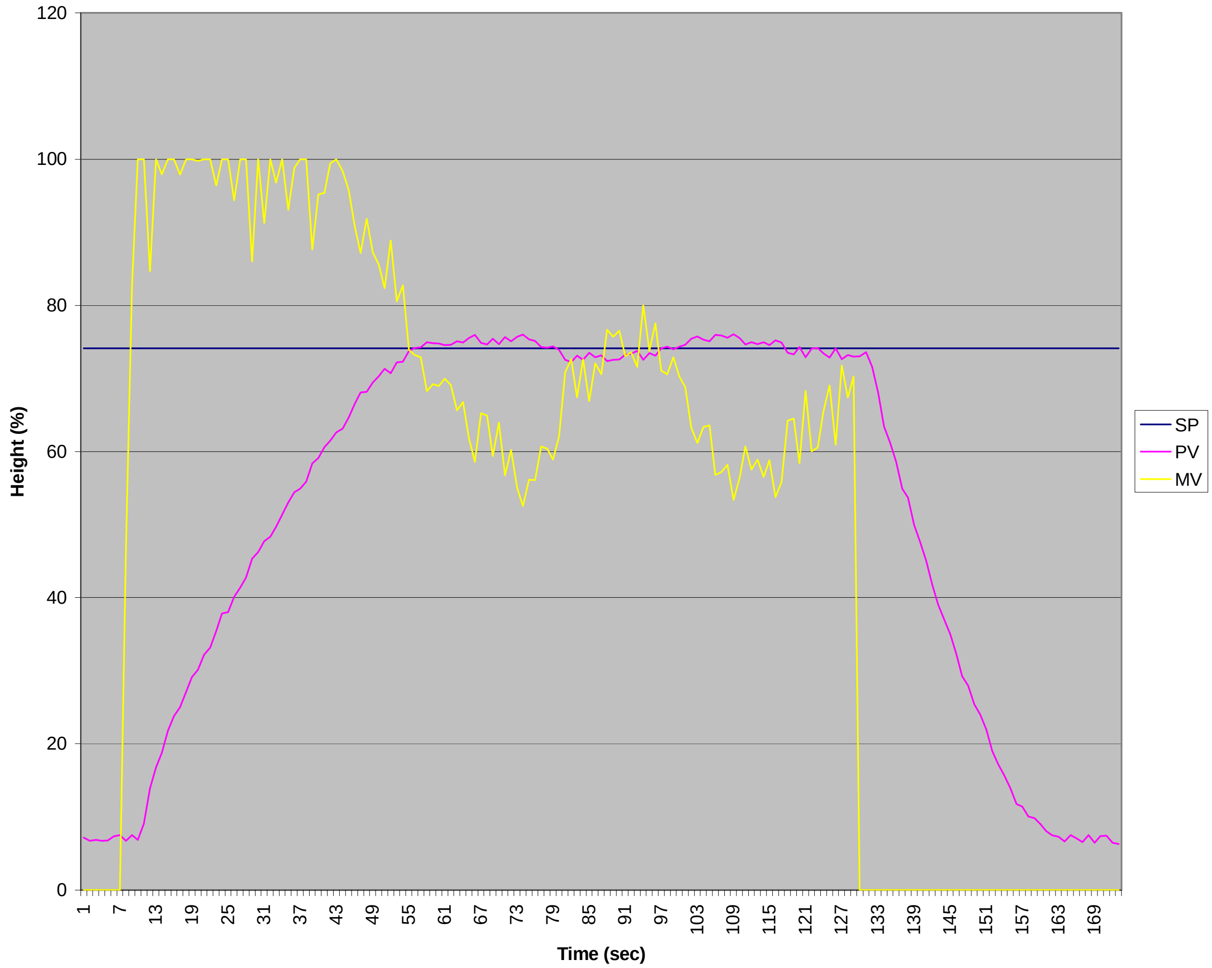


Fig.5.2.3.1 Batch control performance for Tank 1without disturbance

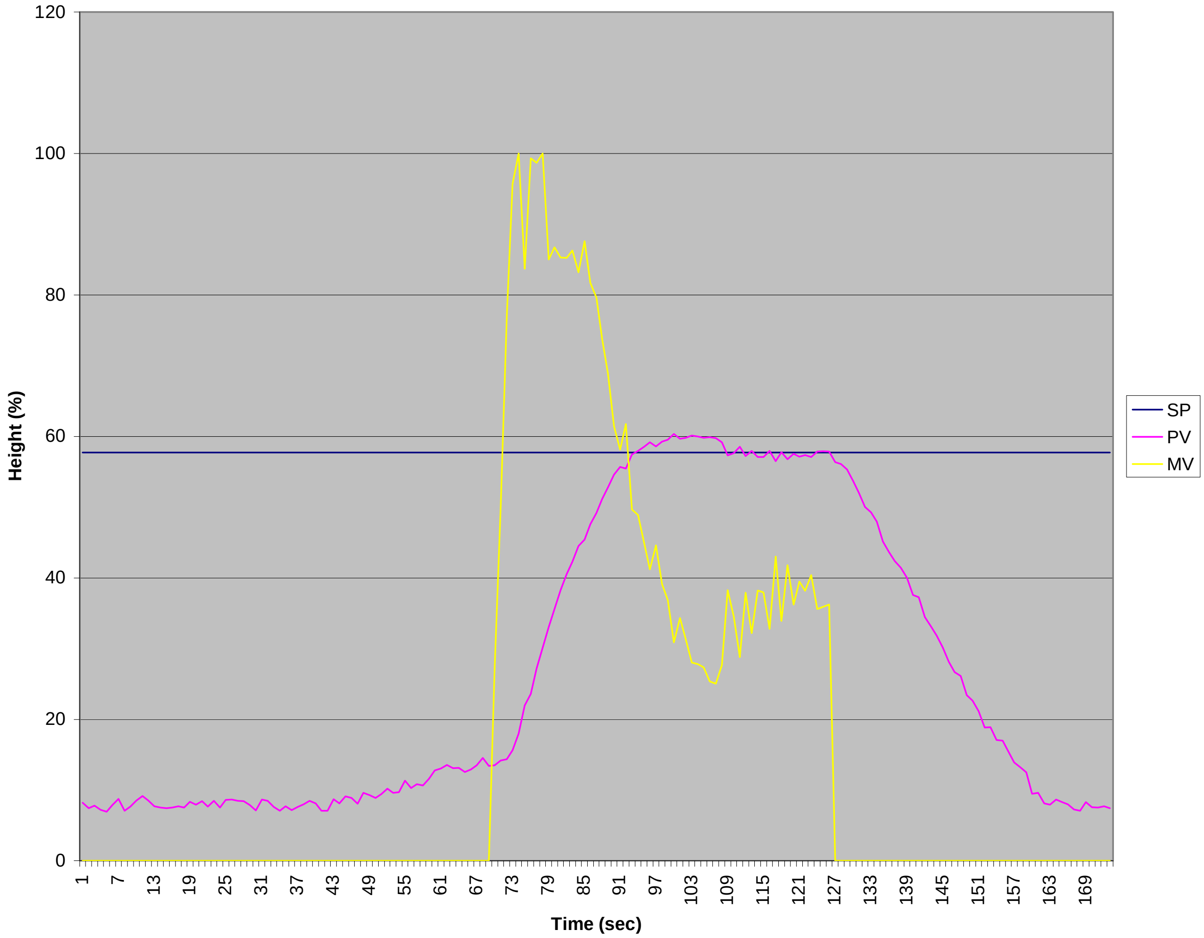


Fig.5.2.3.2 Batch control performance for Tank 2 without disturbance

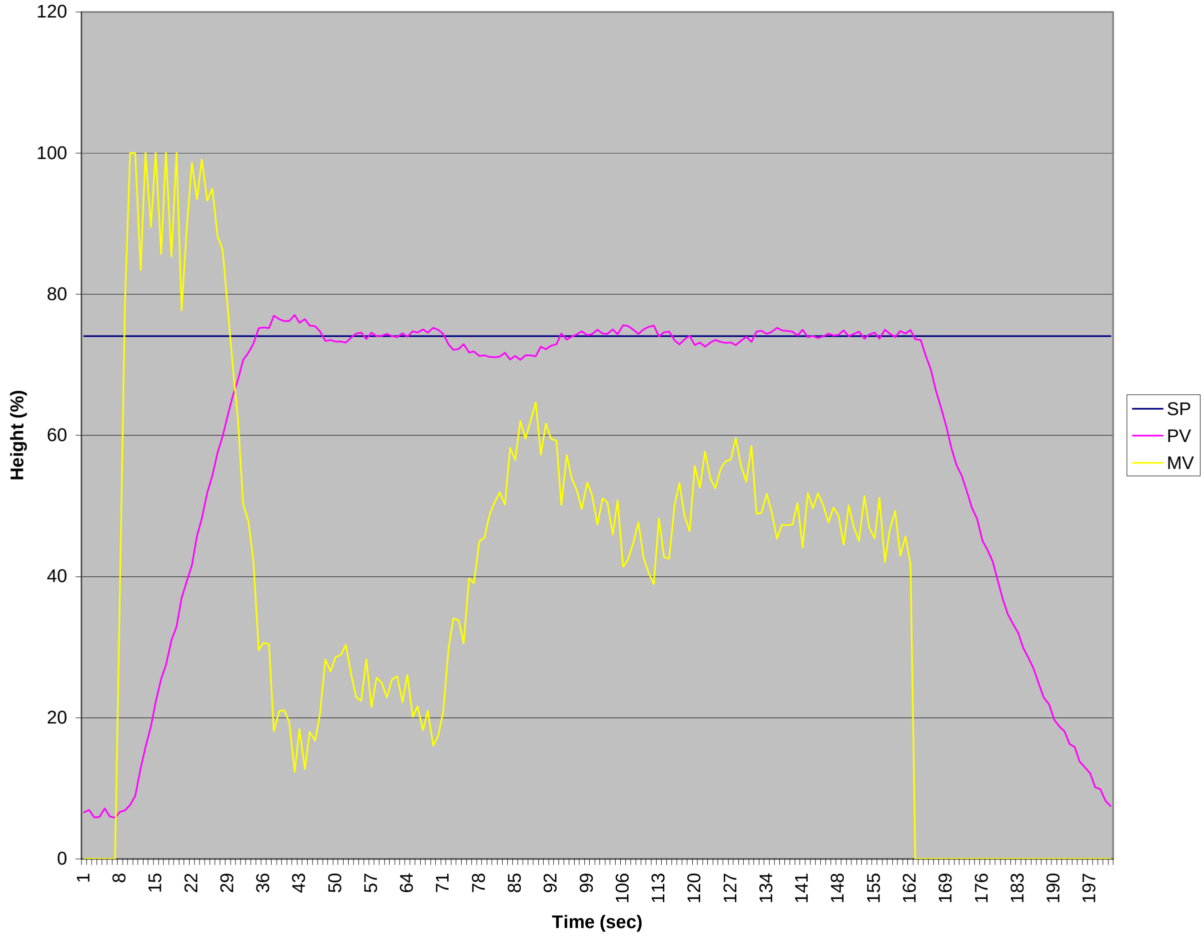


Fig.5.2.3.3 Batch Control performance for Tank 1with disturbance

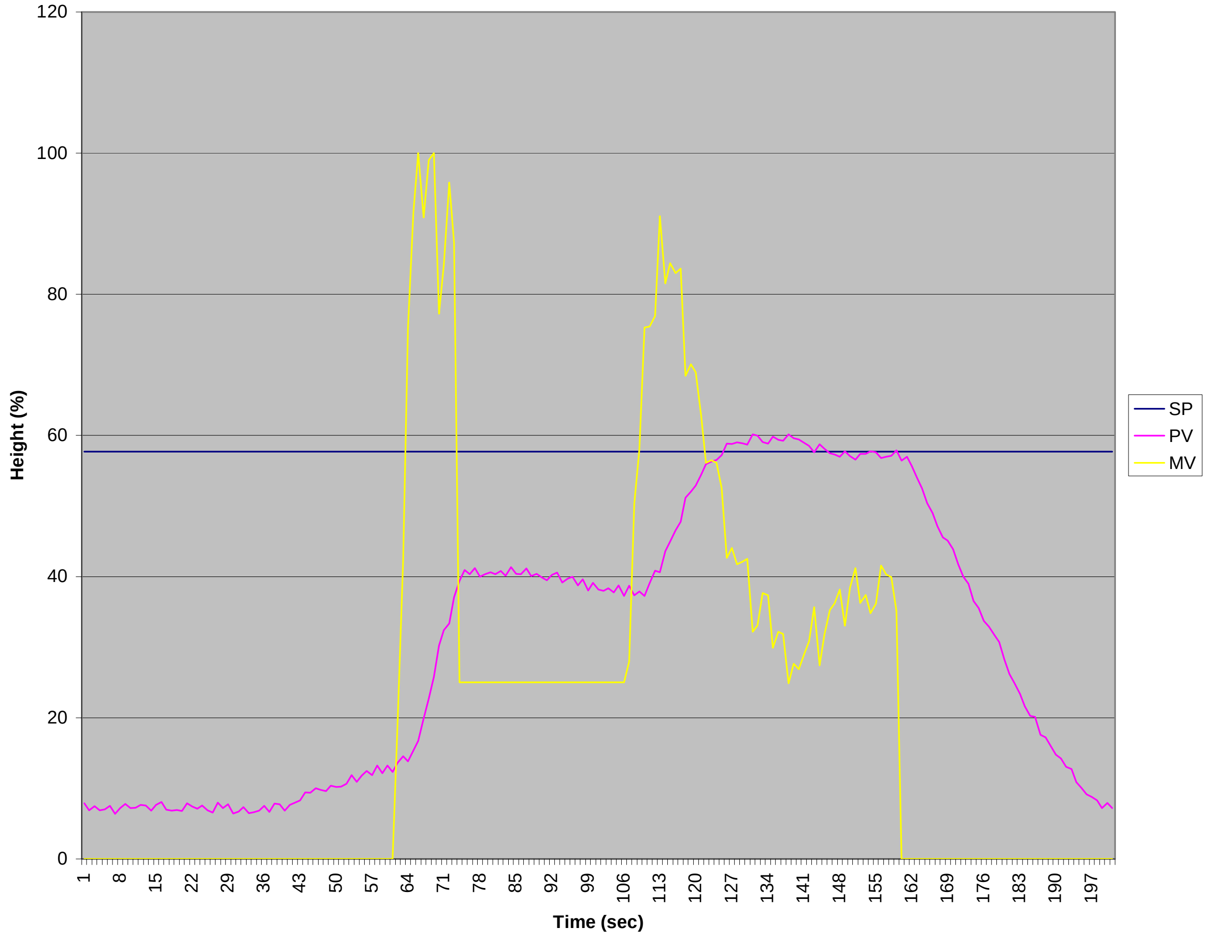


Fig.5.2.3.4 Batch Control performance for Tank 2 with disturbance

CHAPTER 6

DISCUSSIONS AND FUTURE WORK

6.1 Problems encountered

Many problems are encountered during the making of AI and AO function block. For example, the C++ software used to program I/O is Borland C++ builder. Functions in I/O board driver, NIDAQ, are used as part of the C++ program to control analog I/O. However, NIDAQ driver software does not officially support C++ builder, and therefore no sample code is available for reference. Some test programs are written but not executable. To search for the correct way to call NIDAQ functions in C++ builder, public discussion forums in National Instruments and Borland are visited and questions are posted. This problem is finally solved with the help of technical support staff in NI and feedback by other users in the forums. General programming skills are recapped and new programming techniques in C++ and Java are learnt.

Through this project, it is also learnt that no matter how good the simulated control is, when it comes to real time control, it is a totally different scenario. Some hardware problems are also encountered. The non-linear relationship between the height and voltage reading of the level sensor has to be considered in a real process. Therefore, calibration of a system is always necessary. Process variable will tend to oscillate due to noise and therefore will not become a constant value.

Most important of all, understanding of the IEC 61499 is deepened through this project and many fundamental issues are understood. In addition benefits of using IEC 61499, which are all previously learnt in reading, are fully appreciated after this project.

6.2 Benefits of using IEC 61499

After this project, many advantages of IEC 61499 are learnt and fully appreciated.

1. Under IEC 61499, algorithm design can be separated with system design. A PID function block can be designed by control experts. System design can be done by system engineer or designer who may not know well about how to design a robust algorithm for a good PID controller. Since function block is represented by GUI, it is easy to connect and design an application without knowing its algorithm. Therefore system designer no longer requires to be a programmer.

2. As shown in chapter 5 batch control application, the PID composite function block which controls the read and write cycle in PID control application can be reused again the batch control application. If system designers make changes to application or reconfigure application to perform other functions there is no need to redesign every thing again from scratch. Eg. If there is a need to combine more tank to batch application, user only need to add in PID local design and change batch control event sequence. This reusable model greatly reduces design time.

3. Batch sequence function block is relatively easy to design. If the sequence of events is clear and designed, then core of batch control is decided. The rest of the design will be simple since the sequence of execution is through event conditions.

4. IEC 61499 also provides easy communication via network. Ready PUBLISH and SUBSCRIBE, SERVER and CLIENT function blocks take care of Internet protocol and network communication. System designer does not have to be a network expert to establish network link between devices.

5. Encapsulation is possible to reduce the number of FB and links displayed and designed at once. As a result, it reduces chances of error. All application and sub-application can be separated so easy for debugging. The system designer is able to find out where the mistake lies by testing from small application to bigger application and then whole application. E.g. AO.fb follow by pump.fb then PID control

6.3 Possible areas for future work

1. Distributed batch control sequence function blocks can be implemented to Controller 1 and Controller 2 so that they can communicate and perform this application without the Central controller. Such application will then be 100% distributed. Intelligent field device with microprocessor and ready made I/O function block can be made so that device can operate independently in network.

2. The editor by FBDK is not mature enough. It is not easy to change algorithm in Java or XML. There is a need for a graphical and algorithm switching platform, e.g. Jbuilder from Borland. Much anticipation is set for the beta version of Microsoft's editor coming this April.

3. Currently, it is not easy to design application using IEC 61499 because it still requires user to have deep understanding about IEC 61499. Several aspects of the IEC 61499 standard are unfamiliar to most practitioners of control systems engineering, especially the concepts of distributed applications, event-driven execution control and service interface function blocks for communications, I/O and other operating system services. The use of design patterns can simplify the job of becoming familiar with the application of these new concepts. Any system engineer who may not understand about IEC 61499 should be able to design application. We need design pattern so that design process can become significantly easier. E.g. Given a batch control application, user is given a step by step guide which let him choose what control, temp, tank function block to use.

4. FBDK runtime should not be only in Java. It should be platform independent. It should allow user to program in different compiler language.

CHAPTER 7

CONCLUSION

The main objective of this project is to investigate the implementablity, reliability and performance of IEC 61499 on process control. It has been successfully achieved.

A system configuration tool is developed using Java during industrial attachment in Japan. The tool allows users to load and display function blocks, remotely configure the system, download function blocks to field devices and remotely start up the devices through network. It results in a conference paper being published jointly with staff in Yamatake Corporation in TOTAL ENTERPRISE SOLUTIONS CONFERENCE, ICAM Asia 2001.

A set of function blocks based on the IEC 61499 standard are developed, and the linkage of the function blocks with the computer I/O card is established. As such, the function blocks are configured for real time process control. The project requires the use of XML for defining common DTD, execution of command transfer in control network and function block development for real time control applications. C++ is used for the control of I/O board. Java is used for the prototyping development of embedded real time execution algorithm. Java Native Interface is also implemented to link C++ and Java programs. Besides, this project includes model description schema using UML and investigation on Internet protocol for application of system in open network. General programming skills are recapped and new programming techniques in C++ and Java are learnt.

A coupled tank with analog inputs and outputs is successfully controlled and some classic control examples like open loop control, close loop control and PID control are realized. A batch control through network is also successfully implemented. This is the first trial of distributed real-time process control and batch control using IEC 61499 in Aisa.

The knowledge of IEC 61499 is deepened through this project and many new issues are understood. In addition, benefits of using IEC 61499 are fully appreciated after this project. IEC 61499 provides easy communication via network. Function block PUBLISH and SUBSCRIBE, SERVER and CLIENT take care of internet protocol and network communications. Software reuse and reconfigurbility ability of IEC 61499 is also appreciated. If system designers make changes to application or reconfigure application to perform other functions, there is no need to redesign every thing again from scratch. It is well illustrated when the PID composite function block is reused again in the batch control application. Software encapsulation also allows the number of function block used in an application to reduce thus reducing number of connections thus reduces the chances of making an error and make debugging of application very easy. Therefore, by using IEC 61499 concept, design time is greatly reduced, engineering cost is expected to reduce and system will be more flexible and maintainable. Some areas for future work are also proposed in this project.

REFERENCES

[1] J.H. Christensen, " Basic Concepts of IEC 61499", Fachtagung Verteilte Automatisierung, Magdeburg DE, 22-23 March 2000.

[2] J.H. Christensen, "Design patterns for systems engineering with IEC 61499", Fachtagung Verteilte Automatisierung, Magdeburg DE, 22-23 March 2000.

[3] H. Tsunematsu, Yang Wei, "IEC 61499 Distributed Function Block Concept", Total Enterprise Solution 2001, Singapore, 10 June 2001.

[4] IEC 65/248/PAS, "Function blocks for industrial-process measurement and control systems - Part 1: Architecture", 19 April 2000.

[5] IEC 65/260/PAS, "Function blocks for industrial-process measurement and control systems - Part 2:Software tools requirements", 15 December 2000.

[6] IEC 61499-3 (working draft), "Function blocks for industrial-process measurement and control systems - Part 3: Application guidelines", 18 January 2001.

[7] IEC 61499-3 (working draft), "Function blocks for industrial-process measurement and control systems - Part 3: Rules for compliance profiles", 19 January 2001.

[8] Pradeep David, "Putting the E in Manufacturing", Providing Total Solutions for Process Plants of the 21st Century, Singapore, 1 November 2001.

[9] IEC 61499 home page: http://www.holobloc.com