Input and output devices engineering essay

Engineering

OverviewBefore PLC’s were created many industries such as the automobile industry used hundreds of thousands of relays to control their processes. This was very time consuming and expensive, sometimes resulting in a two year change around between different products because electricians would have had to individually rewire every relay to change a production system for a different line of cars etc. Programmable Logic Controllers or PLC’s were first designed in 1968 when General Motors decided that a replacement for this type of process was needed. PLC’s are designed for multiple inputs and outputs. A PLC is essentially a small computer which is used for mainly industrial purposes but also has other uses. Industrial uses could be; Packaging linesRobotsHydraulic machinesPneumatic machinesOther uses could be; Traffic lightsCar parking barriersSignalling systemsThere are three types of programmable logic controllers (PLC’s). They are Unitary, Modular and Rack mounted. PLC UnitsThere are three types of programmable logic controllers (PLC’s). They are Unitary, Modular and Rack mounted. All three types have different features both good and bad which make each type suitable for many different applications.

Unitary PLC

Design and characteristics

A Unitary PLC contains all the features of a basic system in one compact unit, the features include; A power supply. The main module which contains a central processing unit (CPU). The input module. The output module. Unitary PLC's are fitted directly to the component or machine that they are controlling.

The advantages of a unitary system are;

They are small and compact. They hold all the basic components in one unit. They allow portable and easy access. They are usually the cheapest type of PLC.

The disadvantages of a unitary system are;

They cannot be expanded. If any feature fails then the whole unit has to be replaced. They are very simple and basic.

Applications

Unitary PLC's can be used for any application that does not require a lot of inputs or outputs. For example a car park barrier, this would not use many inputs/ outputs and would only require a simple program.

Modular PLC

Design and characteristics

Modular PLC's are a system of modules that can be slotted together to build up a system. The basic modules contain; A power supply. The main module which contains a central processing unit (CPU). The input module. The output

module. Other types of module can be attached as well as extra input and output modules to increase the capacity or to cope with changes in hardware system.

The advantages of a modular system are;

The amount of input and output terminals can be expanded to cope with any changes to the hardware system. If any feature fails then only that part has to be changed saving on cost.

The disadvantages of a modular system are;

They are expensive compared to unitary PLC’s.

Applications

Modular PLC’s are useful for applications where there would be a lot of inputs and outputs because more input/output modules can be added if needed. This type of PLC is widely used in the manufacturing industry to control process lines.

Rack-mounted PLC

Design and characteristics

The design and characteristics of a rack mounted system are very similar to a modular system however these modules are on standard cards which then slot together into a rack inside a control cabinet. These modules communicate via the rack.

The advantages of a rack-mounted system are;

They are easily modified or expanded. They have more I/O points than any other type of PLC. If any feature fails then only that part has to be changed saving on cost.

The disadvantages of a rack-mounted system are;

They are usually the most expensive of the three PLC’s

Applications

Like the modular PLC this type of system is widely used in the manufacturing industry. This is mainly because of the amount of input/output points that the system can contain but also because it has the ability to communicate with other networks. Internal Workings

CPU

The CPU- Central Processing Unit acts as the brain of the PLC. It contains a memory unit into which the PLC program is written into. It is basically used to process the information it receives from inputs and carries out instructions to the outputs according to the logic programmed into it. This process is called the scan cycle and it occurs every 5ms. The scan cycle is shown below. When a program is written on a programming device such as a laptop it is uploaded to the CPU, once it has been uploaded it is then written into the CPU’s memory. The programming device can then be disconnected without losing any data from the PLC because the program is now saved into the memory of the CPU. Only the unitary PLC has a CPU built into it, on the modular and rack-mounted PLC’s the CPU comes as a separate module. On modular PLC’s the CPU would be the first module whereas on rack-mounted

PLC’s the CPU is governed by the amount of inputs and outputs that are being used.

Input and Output Devices

In PLC’s data is collected from inputs which are connected to the input modules and sent to the CPU, this data is then executed within the program logic and then sent to the output devices via the output modules to perform a specific task. The inputs and outputs are assigned a reference point in the PLC so that they are easily recognisable when programming.

Here is an example of an I/O listing table which I have created.

INPUT = I

OUTPUT = Q

I 0. 0

Reset_PB

Q 0. 0

Relay 1

I 0. 1

Start_PB

Q 0. 1

Relay 2

I 0. 2

Up_Switch

Q 0. 2

Indicator_light

I 0. 3

Down_Switch

Q 0. 3

I 0. 4

Q 0. 4

I 0. 5

Q 0. 5

I 0. 6

I 0. 7

Inputs

An input device is any peripheral piece of hardware that is used to send data information and control signals to any information processing system, in this case the PLC. In terms of PLC's there are two different types of inputs, analogue and digital. An example of an analogue input would be a sensor or transmitter and an example of a digital input would be a switch, push button etc.

Inductive proximity sensor

An inductive proximity sensor will detect metal surfaces or devices without coming into contact with them. The sensor face is made up of windings from the oscillator. These form an alternating magnetic field around the sensor face called an inductance loop. The inductance of a loop will change if a

metal is placed inside it because metals are much more effective inductors than other types of material. Once the inductance loop is altered the change is detected by sensing circuitry which then signals to another device such as a PLC.

Capacitive sensor

A capacitive sensor will detect any surface or device without coming into contact with it. A capacitive sensor uses an alternating voltage which in turn causes the positions of the charges to continually reverse. This then creates an alternating electric current which is detected by the sensor face. The sensor face is made up of capacitors from the oscillator. The amount of current flow is determined by the capacitance, and the capacitance is determined by the area and proximity of the conductive objects. Therefore the larger and closer the object then the greater the current will be and vice versa.

Photo Electric Sensor

A photoelectric sensor uses light to operate. When a preset level of light is picked up by the sensor the sensor switches. This type of sensor is used to detect moving objects, for example bottles on a conveyor are detected due to the bottles breaking the line of light. Once the line has been broken the switch signal is fed into the electrical control circuit which provides a corrective action. There are three types of photo electric sensor. These are Reflective, retro-reflective and separate type.

Optical Reflective Sensor

The light is reflected off the surface of the bottle back to the receiver.

Optical Retro-reflective Sensor

The light is transmitted and reflects off a reflective surface back to the receiver.

Optical Separate Sensor

Outputs

An output device is basically any device that is controlled via the output module of a PLC. In terms of PLC's there are two different types of outputs, analogue and digital. Analogue devices are devices that the output can be varied. Digital outputs are used to control two state devices, this means that they are either on or off. An example of an analogue output would be motor speed; valve position etc and an example of a digital output would be a relay, solenoid valve etc.

Indicators

Indicator light

The simplest of the PLC outputs, the indicator light simply shows that a certain function has worked correctly, for example if a machine was in run mode then a green indicator light could also be activated to notify operators etc that the machine is running.

Indicator buzzer

Acts in the same principle as the light but uses sound; it is more effective at alerting someone to a problem especially if that person does not have a line of sight of the machine. For example if there was an error that has caused a machine to stop then the buzzer would sound.

Solenoid valve

A solenoid valve can be operated electrically and pneumatically. In the case of a normally open 3/2 valve it uses one of these to switch it from off to on. Solenoids are named after the amount of states or ports they have and also what function they do. For example a 3/2 spring return solenoid has 3 ports, 2 states and once it has been deactivated it returns back to its natural position via a spring. The 3/2 valve below is shown in two stages. Stage 1 is where the valve is in its natural state where air is flowing from ‘ P’ (the air supply) to ‘ b’ (the exhaust). Whereas in stage 2 the valve is activated allowing air to flow from ‘ P’ to ‘ a’ where the air is sent to another device for example a cylinder.

Stage 1

Stage 2

SignallingSignalling refers to the use of signals for controlling communications. Signals can be either digital or analogue. With digital signals the signal is logic therefore it is either a ‘ 1’ (True) or a ‘ 0’ (False). With analogue signals the entity is continuous where the signal is constantly varying based on real time. For example the amount of light a sensor receives is an analogue signal because it can be any value within the range that the sensor can pick up. In some systems both digital and analogue signalling is needed. For example a house phone shares the same network connection as a pc. In this case a device called a modem is used to convert the signal between the two. NetworksA PLC must communicate with other devices in order to operate. These devices are connected to the PLC via a network. There are many different types of network such as:

Bus Network

A bus network is a long cable acting as a backbone which links devices together, the backbone has nodes connected to it via a single line. The signal travels in both directions until they are terminated at either end. The amount of nodes that can be attached however is limited to the strength of the signal.

LAN- Local Area Network

A LAN network is used to connect different hosts in a small area such as an office or a small building. Data is sent in the form of a data packet which includes the address of both the sender and the destination. ProgrammingThere are three main types of programming that are used in PLC’s. These are; Ladder logicStructured textFunctional block

Ladder logic

Ladder logic is one of the most common styles of PLC programming and it is used in most manufacturing companies. It is called ladder logic simply because the style resembles a ladder as shown below where the two vertical rails are connected by a rung. It is popular because of its similarity to relay logic which in turn allows most program designers to easily grasp the concept. Ladder logic is written using logic symbols which are shown below and is read from left to right.

Ladder logic symbols

Normally openNormally closedNormally open immediateNormally closed immediateNOTPositive transitionNegative transitionOutputOutput immediateSetSet immediateResetReset immediateSet-dominate

bistableReset-dominate bistableNo operationExample of Ladder logicBelow is a simple circuit designed to mimic a drilling process where 3 cylinders are used to drill a part. Cylinder ‘ a’ when extended places a part below cylinder ‘ b’, whilst cylinder ‘ c’ is already extended holding the part securely in place and also preventing the part from falling through the gap below cylinder ‘ b’. Cylinder ‘ b’ then comes down, drills the part, goes back up and cylinder ‘ c’ retracts allowing the part to fall through the gap. The ladder diagram for this circuit is shown below. As can be seen from above the system uses sensors to show whether each cylinder is extended or retracted. This information is then used to activate each cylinder. For example in the first part of the programme which has been highlighted; Relay R1 is activated when the start button is pressed given that the stop button is in the normally closed position. Sensor 5 (S5) is off which is showing that cylinder 3 is extended. When this is off it allows Relay 1 to stay permanently on because one of the contacts from R1 can then complete the circuit and create a latch. R1 is then used to activate solenoid 1(Sol1) to extend cylinder 1 provided that S2 and R4 are off.

Structured text

Structured text is a text based language that uses statements to define what needs to be executed. Structured text uses statements such as: IF…THEN… CASE…OF…FOR…DO…WHILE…DO…REPEAT…UNTIL…People who are trained in computer language find structured text easier to understand than ladder logic because they are of a similar format. Example of Structured textBelow is an example of how structured text is used in the workplace, it is taken from the program [OCD_PA2] [u71_dat. src [Structured text]] which is used

for a coating line at Ortho Clinical Diagnostics. This programme is used for data transfer to the database for the well fill process.(* cyclic program *)

(*
****
****
****
*** Tasku71_dat. src*** DescriptionArrange data transfer to database for well fill data

***
*** Author: RTH*** Date: 30. 09. 2008*** Version: V1. 00

****
****
****
*** History:

***
****
****
****
*)
(* check if actual data from well fill verification available ?*)IF stLaserValues. bStartDataTransfer AND NOT bInit THENstLaserValues. bStartDataTransfer:= FALSE; bStartSendData := TRUE; END_IF(* trigger send data *)IF bStartSendData THENbStartSendData := FALSE;(* fill in CTQ data *)memcpy(ADR(InternalPlate. diSeqNo), ADR(Plate_MEM. diSeqNo),

SIZEOF(InternalPlate)); IF (usiStepData_A = CTQ_WAIT_NEW_DATA) AND NOT bWriteData THENIF (usiDataP_Buffer_A > 1) AND (usiDataP_Buffer_A < 20) THEN(* load actual data Buffer position *)memcpy(ADR(InfoPlate[0]. diSeqNo), ADR(InfoPlate[usiDataP_Buffer_A]. diSeqNo), SIZEOF(InternalPlate)); memcpy(ADR(Plate), ADR(InfoPlate[0]. diSeqNo), SIZEOF(Plate));(*transfer data into stored mem -> 1 position above ! *)memmove (ADR(InfoPlate[2]. diSeqNo), ADR(InfoPlate[1]. diSeqNo), (19*SIZEOF(InfoPlate[0])));(* transfer information to fifo buffer *)memcpy(ADR(InfoPlate[1]. diSeqNo), ADR(InternalPlate), SIZEOF(InternalPlate)); usiDataP_Buffer_A := usiDataP_Buffer_A + 1; (*increment buffer position *)bWriteData := TRUE; ELSE(*transfer data into stored mem *)memmove (ADR(InfoPlate[1]. diSeqNo), ADR(InfoPlate[0]. diSeqNo),(20*SIZEOF(InfoPlate[0])));(* transfer information to fifo buffer *)memcpy(ADR(InfoPlate[0]. diSeqNo), ADR(InternalPlate), SIZEOF(InternalPlate)); memcpy(ADR(Plate), ADR(InfoPlate[0]. diSeqNo), SIZEOF(Plate)); bWriteData := TRUE; END_IFELSE(*transfer data into stored mem -> 1 position above !*)memmove (ADR(InfoPlate[2]. diSeqNo), ADR(InfoPlate[1]. diSeqNo),(19*SIZEOF(InfoPlate[0])));(* transfer information to fifo buffer *)memcpy(ADR(InfoPlate[1]. diSeqNo), ADR(InternalPlate), SIZEOF(InternalPlate)); usiDataP_Buffer_A := usiDataP_Buffer_A + 1; (*increment buffer position *)END_IFEND_IF(* error check on data buffer *)IF usiDataP_Buffer_A >= usiDataP_Buffer_Alarm THEN(* force alarm if buffer is overfilled *)bDataP_Buffer_Alarm_MEM := TRUE; END_IFIF bDataP_Buffer_Alarm_MEM THENError. Units. U71[9] := TRUE; (* Error 71010 *)Error. bStopCode_NS:= TRUE; END_IFIF usiDataP_Buffer_A <= 1 THEN(* clear alarm if buffer is empty again *)bDataP_Buffer_Alarm_MEM := FALSE;

END_IF(* Message Buffer not empty - still data transfer active*)IF usiDataP_Buffer_A >= usiDataP_Buffer_Msg THENError. bMessage[100] := TRUE; (* Message 20101 *)ELSEError. bMessage[100] := FALSE; END_IFCASE usiStepData_A OFCTQ_WAIT_NEW_DATA:(* Step 0 *)IF bWriteData THENiCountWait_Data_A := iCountWait_Data_A + 1; IF iCountWait_Data_A >= iMinWait_Send_A THEN(*Wait XXXms !!! *)bWriteData:= 0; iCountWait_Data_A:= 0; usiStepData_A:= CTQ_DATA_START_CMD; END_IFEND_IFCTQ_DATA_START_CMD:(* Step 5 *)(* start command new CTQ data *)iCountWait_Data_A := iCountWait_Data_A + 1; IF NOT bDataSQL_Cmd_A THENbDataSQL_ok_A:= FALSE;(* clear state *)bDataSQL_error_A:= FALSE; bDataSQL_timeout_A:= FALSE; bDataSQL_Cmd_A:= TRUE;(* start command *)END_IF(* check acknowledge data handle from database *)IF bDataSQL_Cmd_A AND (iDataSQL_State_A <> 0) THEN(* iDataSQL_State => see below(* Bit 0= DONE(* Bit 1= ERROR(* Bit 2-15= Errorcode *)(* save last feedback state *)iDataSQL_State_A_MEM := iDataSQL_State_A; IF (iDataSQL_State_A = 1) THENbDataSQL_ok_A:= TRUE;(* all data successfully done *)usiStepData_A:= CTQ_DATA_OK; ELSEbDataSQL_error_A:= TRUE;(* data transfer with error *)usiStepData_A:= CTQ_DATA_ERROR; END_IFbDataSQL_Cmd_A:= FALSE;(* quit data command *)iDataSQL_State_A:= 0; END_IFIF iCountWait_Data_A >= iTimeoutDataRequest_A THEN (*Wait XXXms !!! check timeout -> feedback result *)iCountWait_Data_A:= 0; bDataSQL_timeout_A:= TRUE; usiStepData_A:= CTQ_DATA_TIMEOUT; END_IFCTQ_DATA_OK:(* Step 11 *)IF ((usiDataP_Buffer_A > 1) AND (usiDataP_Buffer_A < 20)) THENusiDataP_Buffer_A := usiDataP_Buffer_A - 1;(* transfer information from fifo to transfer mem *)memcpy(ADR(InfoPlate[0]. diSeqNo),

ADR(InfoPlate[usiDataP_Buffer_A]. diSeqNo), SIZEOF(InfoPlate[0]));

memcpy(ADR(Plate), ADR(InfoPlate[0]. diSeqNo), SIZEOF(Plate));

bWriteData:= TRUE; ELSEusiDataP_Buffer_A := 1; END_IFusiStepData_A:=

CTQ_WAIT_NEW_DATA; iCountWait_Data_A:= 0; (* clear internal counter

)CTQ_DATA_ERROR:(Step 12*)bWriteData:= TRUE; usiStepData_A :=

CTQ_WAIT_NEW_DATA; iCountWait_Data_A:= 0; (* clear internal counter

)CTQ_DATA_TIMEOUT:(Step 13*)bWriteData:= TRUE; bDataSQL_Cmd_A:=

FALSE;(* quit data command *)iDataSQL_State_A:= 0; usiStepData_A:=

CTQ_WAIT_NEW_DATA; iCountWait_Data_A:= 0; (* clear internal counter

)END_CASE(clear init flag *)IF bInit THENbInit := 0; END_IF