Alternative Approaches to Diagnostic Intelligence in Process Control Systems

[1] Ifeyinwa Obiora-Dimson [2] Hyacinth C. Inyiama [3]Christiana C. Okezie
ifeyinwa29@yahoomail.com hcinyiama@gmail.com christianaobioma@yahoomail.com
[1,2,3] Department of Electronic and Computer Engineering,
Nnamdi Azikiwe University, Awka.
Anambra State, Nigeria.

Abstract

Process control systems usually obtain their primary inputs from sensors that monitor the process under control. Use of sensors and alternative approaches to monitoring processes with an aim of detecting faults is highlighted. A process example involving blending of fruit juices, coolers, heaters, valves etc were also used to showcase some of these approaches. A combination of more than one approach is possible thus taking advantage of the inherent strength in each approach. This paper should be of great interest to anyone interested in obtaining the diagnostic intelligence needed for intelligent process automation.

Keywords: sensors, diagnostic intelligence, automation, monitoring, process control

I. INTRODUCTION

Some of the primary inputs to a control system come from sensors that monitor the process under control [1]. This paper sheds light on the nature of such sensors and other means of providing intelligence to a process control system. The following were discussed: smart scale, electronic eyes, device timing, device time out, use of state code in both Algorithmic State Machine (ASM) chart and flow charts etc.

II. USING SMART SCALE

Consider a situation where fruit juices are being blended by weight. If the mixer is positioned on a smart scale, the smart scale will feed its digital output to a microprocessor/microcontroller each time fruit juice is added. Consider the blending of four beverages [2]. The smart scale would have a reading when the mixer is placed on it. When beverage 1 (BV1) is added, the smart scale will yield the weight for the mixer and BV1. By sensing the weight input by the smart sensor, the processor would know when the weight of the mixer plus BV1 has reached the desired weight and then cut OFF BV1. A similar process is used to add beverage 2 (BV2) until the weight of the mixer plus BV1 and BV2 is as desired. This process continues until beverages 3 and 4 (BV3 and BV4) respectively are added to the mixer. Thereafter the mixing will take place and the mixture is extruded. Suppose the processor turns ON one of the beverage valves for some time and the weight output of the smart scale does not change. It means that the particular valve turned ON is faulty or that its corresponding beverage tank is empty. This would then serve as diagnostic information.

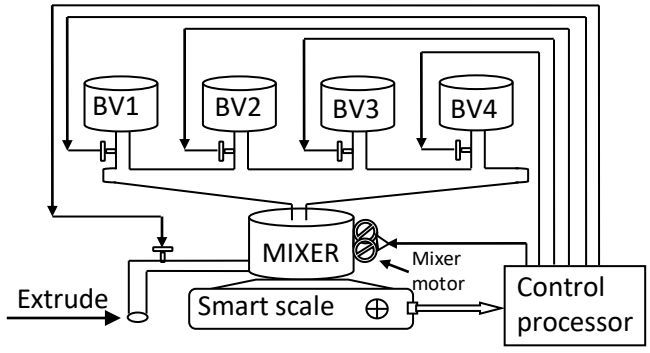


Fig 1: smart scale based diagnostic system for beverage mixing

III. USING ELECTRONIC EYES

Sometimes because of the limited number of I/O lines available to a microprocessor say, there would not be enough spare lines to feed in the digital pattern to the processor because of other I/O commitments. In this case, electronic eyes may be used to reduce the number of input lines required from the remote sensor [3]. This scheme works as follows.

a. The scale is spring loaded, such that when the mixer is empty, the piston blocks only the uppermost electronic eye (fig 2). When a measured weight of BV1 say is added, the piston descends below the first eye and another electronic eye is positioned where it is just blocked by the piston. Thereafter another quantity of a beverage say BV2 of known weight is added and another electronic eye is positioned to match the piston and this continues until electronic eye for the beverage 4 (BV4) is added. In this context, an electronic eye is an opto-coupler or light source/light sensor pair as shown in fig 2. We see therefore that five signal lines would reach the processor, one for the mixer weight alone. Another for mixer and BV1, another when BV2 is added and so on until BV4.

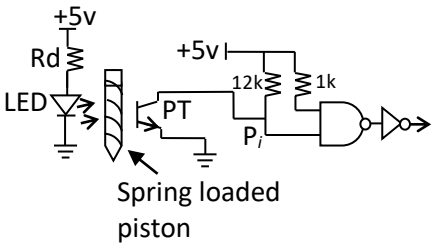


Figure 2a: Basic block of fig 2b

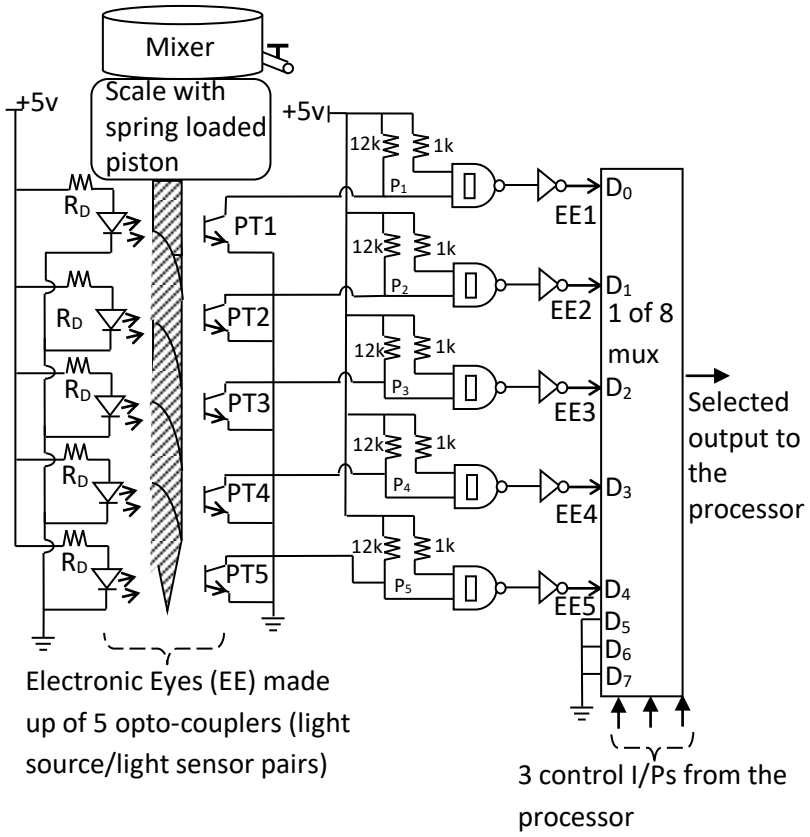


Figure 2b: Spring loading involving electronic eye measurement

A multiplexer can be used to select each of those lines at a time [4]. Even if there were to be up to 7 beverages to be blended, given 8 lines of input (the mixer alone + 7 beverages). These input lines can go to a 1-out-of-8 multiplexer that can be provided with control input from three output line of the processor. This limits the demand of the I/O line to 4 only for such a complex arrangement, three for the control and one for the selected bit.

This is only half of a byte compared to one byte that would have been used if a smart sensor were in use. Note that the experiment to pour measured weights of BV1 to BV4 into the mixer so as to adjust the electronic eyes /sensors BV1 through BV4 must be done before actual production starts. Thereafter, the processor during a production run would follow the following steps:

1. Turn ON the valve to dispense BV1 into the mixer. Apply the bit pattern to select the electronic eye output of BV1. Wait until that electronic eye sees the piston and its output goes high. Turn OFF the valve for BV1.
2. Repeat step (a) for BV2, BV3, and BV4 before going into the mixing operation.

After the mixing and extrusion of the product, only the electronic eye for the mixer weight should be ON. Others would be turned OFF progressively as the product is extruded. This can be used to indicate the end of extrusion. Note that the electronic eye of the mixer only is adjusted during the experiment to a position where it just turns OFF when only the mixer weight is on the scale.

If the Light Emitting Diode (LED) in fig. 2b is rated 20mA then $V/Rd = 20\times10^{-3}A$

Or $Rd = V/20x10^{-3} = 5$ volts$/20x10^{-3} = 5000/20 = 250\Omega$.

Schmitt type NAND type is used to make possible sharp transitions at the threshold point to logic 1 or 0.

The light sensors are photo transistors that are driven to saturation when light from the LED reaches the transistor base. When each photo transistor receives light, point Pi (i.e P1 or P2 or P3 or P4 or P5) is grounded via the transistor emitter and is therefore at logic 0. The 1k pull-up resistor connects the upper end of the Schmitt-type NAND to logic 1. Therefore the i/ps to the NAND is (1,0) which yields an o/p of 1 from the NAND which is inverted to 0 at EE*i* (i.e. EE1 or EE2 or EE3 or EE4 or EE5) before being fed to the multiplexer (mux).

When the piston blocks the light rays from an LED and prevents it from reaching the corresponding photo transistor PT*i* (i.e. PT1 or PT2 or PT3 or PT4 or PT5), the photo transistor is cut-off, point Pi jumps to logic 1 via the 12k resistor (i.e. P1 or P2 or P3 or P4 or P5) as the case may be, becomes logic 1. The Schmitt type NAND therefore has its 2 inputs at logic 1 and therefore produces a logic 0 output which is then inverted to logic 1 before being fed to the multiplexer. In other words, when the piston blocks any of the 5 light source/light sensor pairs, the corresponding output is a logic 1 and logic 0 otherwise. Thus piston present =logic 1 and piston absent=logic 0

IV. USING DEVICE TIMING AND TIME OUT TECHNIQUES

A. Device timing

The time it takes to dispense into the mixer a known volume of a beverage can be measured by the processor during experimentation. For example, a known volume of the beverage that is required in the mixing operation can be poured into a graduated cylinder to indicate its volume. Then the graduated cylinder is marked at the level of the volume and subsequently emptied of the beverage. The processor can then be used to convert the dispensing of that volume of the beverage into a time measurement as shown in fig 3.

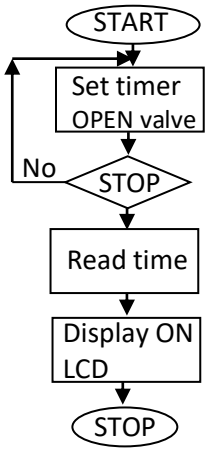


Fig 3: Dispensing time flow chart

First, the pipe sending the beverage into the mixer is brought out and placed inside the graduated cylinder such that when the beverage valve is turned ON, the liquid dispenses into the graduated cylinder. When the liquid reaches the mark in the graduated cylinder, a stop button is pressed whereupon the beverage valve is turned OFF. The processor then reads the timer and displays on the LCD for the designer to see. This is done for each of the beverages that would be blended before the blending operation starts. This enables the processor to handle all the timing involved in the blending in software during actual production. Note that this scheme does not demand any extra I/O port for determining the beverage weights. The processor turns each beverage valve ON one after the other. For each beverage type, it leaves the valve ON for the time needed for the predetermined volume to be added to the mixer.

B. Device time out

Suppose that one of the beverage valves is faulty such that when the signal to turn it ON is applied by the processor, it remains OFF and no beverage is supplied to the mixer. It is pertinent that such a fault condition is determined otherwise the wrong proportion of beverages would be mixed. One way of achieving this is to combine the spring loaded scale with electronic eyes together with software timing during beverage dispensing. Thus when the beverage timing is up, but the signal fed back through the multiplexer (fig 2b) does not change from LOW to HIGH, then that beverage valve is deemed to be timed out. The processor then displays the information on the LCD and suspends blending until it is rectified.

V. FAULTY ELECTRICAL DEVICE DETECTOR FOR VALVES, HEATERS AND COOLERS

A. Valve fault detection

1. Measure liquid level (fig. 4).
2. If LOW, turn valve ON for time T1 long enough to allow marginal rise in level.
3. Measure liquid level again.
4. If still low, there was no marginal rise in level, then flag valve fault else
5. If it has risen a bit then the valve is okay, turn valve on until the liquid gets to desired level .

End.

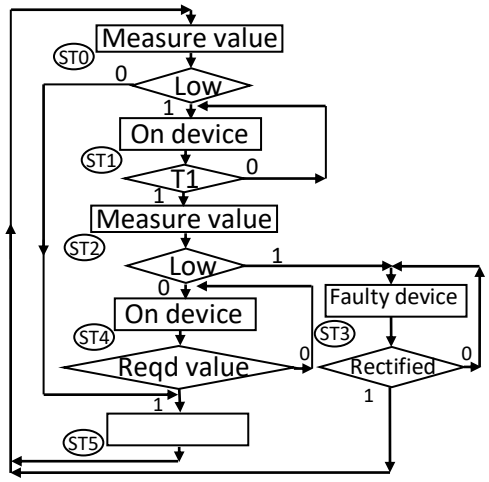
B. Heater fault detection

1. Measure temperature (fig. 4)

2. If LOW, turn ON heater for time T1 long enough to allow marginal rise in temperature else go to end.
3. Measure temperature again.
4. If there is no marginal rise in temperature level, then flag heater fault else turn ON heater until the desired temperature is achieved.
End.

C. Cooler (or air conditioner) fault detection

1. Measure temperature of environment (fig. 4)
2. If high, turn ON cooler for time T1 long enough to allow a marginal cooling else go to end.
3. Measure temperature of environment again.
4. If there is no marginal cooling, then flag cooler fault ELSE turn ON cooler until the desired coolness is achieved.
End.



Pseudo code

```
Start: measure value //                Sound device alarm
associated with valve or cooler        Until error corrected
If value is low turn on device         End repeat: go to start
  Else turn off device End if          Else repeat
Load timer T1                          Turn on device
Repeat                                 Until required reading
On device                              End repeat
Until time-out                         End if
Measure value                          Off device
If no change then                      Off device
Repeat                                 Go to start.
```

Fig 4: flow diagram for detecting electrical faults in valves, heaters and coolers and the corresponding pseudo code

VI. USE OF STATE CODES IN ASM CHARTS

In this paper, the present state of the system is monitored by the processor as the process progresses from one state to another in its ASM chart. The pattern that should constitute the next state is output and fed into the D-inputs of a set of flip-flops such that when a clock pulse occurs, they constitute the next present state. This process adds intelligence to the process control system as the processor uses it to determine how far the process under control has progressed in its ASM chart. If a fault should occur at any time**,** the processor is in a position to state where the fault occurred and the likely cause of fault at that stage. There is an STT corresponding to every ASM chart or flow chart to facilitate the interpretation of the machine's state.

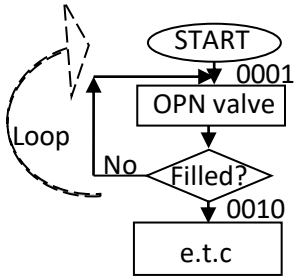


Fig 5a: Endless loop

VII. ADDING STATE CODES TO PROCESS CONTROL SOFTWARE FLOWCHARTS

Every microprocessor/microcontroller based software has a corresponding software flowchart [5]. A software flowchart is similar to an ASM chart but implemented using software rather than hardware. The researcher has found it very advantageous to append state codes to software flowcharts such that as the software progresses to any of the states in the flowchart, the state code is output and fed back into the processor to enable it tell were the software has reached. Should the software end up in an endless loop maybe as a result of device fault, this can be detected by monitoring the state code that is fed back. Fig 5a.

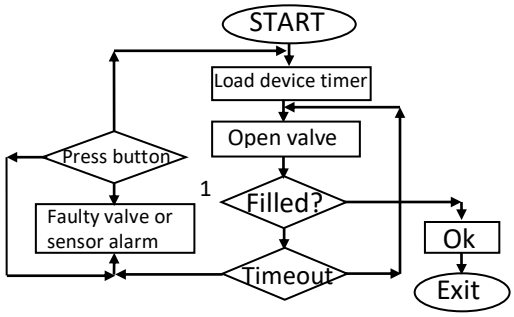


Fig 5b: Endless loop

Suppose that in figure 5a the beverage valve is faulty, when the signal to turn it ON is applied, it does not come ON. Suppose also the tank that one is trying to fill with beverage has tank full sensor that provides feed back to the processor, because the valve is not ON due to fault, the tank full signal will never go high because no liquid is going into the tank. Therefore the process would be stuck in that loop (fig 5a). Because the state code is output and fed back into the processor, it is easy to detect that the process is stuck at that particular state code box. A combination of sensors is required here to detect sensor fault e.g. device timing.

Device timeout technique can be employed to detect a situation where the tank full sensor is bad. Consider figure 5b, device timer is loaded before opening the valve. If the tank is full before time out, this indicates that both sensor and valve are in good condition. If the tank is not filled, and the system is not timed out, it indicates that the system is still filling. When the tank is not filled and the system is timed out, this indicates that both valve and sensor are bad or at least one of them is bad. The system then generates faulty valve/sensor alarm. If rectified, a button is pressed and the system cycle continues else it remains in the faulty position.

Consider a process control example of an upper tank refill system (shown in fig 6) used to store raw/new beverage. The system comprises of a lower tank used to fill the upper tank via a pump. When new beverage is poured in the lower tank, a spring loaded piston on being pressed down by the weight blocks the light from the Light Emitting Diode (LED). This causes the raw beverage to be sucked up automatically into the upper tank via the pump. If the tank indicates full, the pump does not pump up more beverage.

UBVTF is an indicator for full tank, UBVTL is an indicator for low tank while NBV is used to indicate new beverage in the lower tank. BVPMP is used to indicate beverage pump.

The ASM chart representation of the upper tank control system is shown in fig 7. It features three states ST0, ST1, ST2 with state code 000, 001, 010 respectively. The state transition table representation of this diagram is also shown in table 1 and its fully expanded version is shown in table 2. Suppose that at state ST2 in the ASM chart, the system remains in the loop from ST2, UBVTF=0, NBV=1 for longer than is necessary, then this is an indication of a pump fault. Thus if state code is used to monitor this process, the information contained in the STT (table 2) will indicate the present state of the system at any point in time. This would be used to populate the events table. Thus for such a situation where pump fault occurred, the present state code will continuously indicate 010 over a long period and this will be indicated in the events table. This enables the detection of the pump fault because upper beverage tank is not full (UBVT=0) and there is new beverage in the lower tank (NBV=1) the pump is turned ON and yet the new beverage remains in the lower tank. That can only happen when the pump has refused to function, though energized.

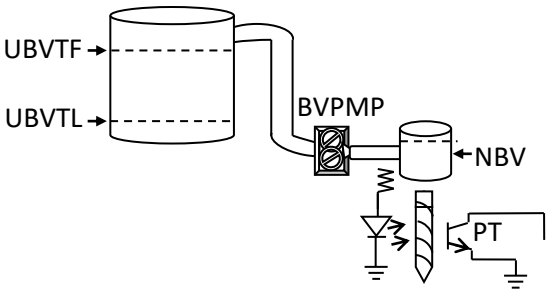


Fig 6: Automatic upper tank refill system

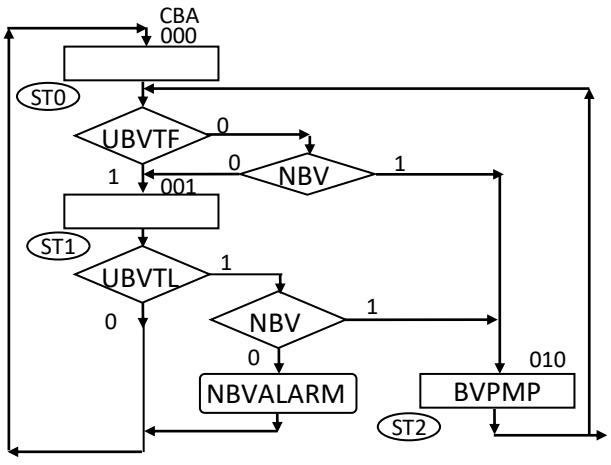


Fig 7: ASM chart of the upper tank refill system of fig 6

Table 1: STT of fig 7

Link path	Present state name	Present state code	Qualifier (UBVF UBVL NBV)	Next state name	Next state code	State output	Conditional output
L1	ST0	000	0 – 0	ST1	001	0	0
L2	ST0	000	0 – 1	ST2	010	0	0
L3	ST0	000	1 – –	ST1	001	0	0
L4	ST1	001	0 – 0	ST0	001	0	0
L5	ST1	001	0 – 1	ST0	010	0	1
L6	ST1	001	1 – –	ST2	001	0	0
L7	ST2	010	0 – 0	ST1	001	0	0
L8	ST2	010	0 – 1	ST2	010	0	0
L9	ST2	010	1 – –	ST1	001	0	0

Table 2: Fully expanded STT of table 1

Link path	Present state name	Present state code	Qualifier (UBVF UBVL NBV)	Next state name	Next state Code	State output	Conditional output
L1	ST0	000	0 0 0	ST1	001	0	0
L1	ST0	000	0 1 0	ST1	001	0	0
L2	ST0	000	0 0 1	ST2	010	0	0
L2	ST0	000	0 1 1	ST2	010	0	0
L3	ST0	000	1 0 0	ST1	001	0	0
L3	ST0	000	1 0 1	ST1	001	0	0
L3	ST0	000	1 1 0	ST1	001	0	0
L3	ST0	000	1 1 1	ST1	001	0	0
L4	ST1	001	0 0 0	ST0	000	0	0
L4	ST1	001	0 0 1	ST0	000	0	0
L4	ST1	001	1 0 0	ST0	000	0	0
L4	ST1	001	1 0 1	ST0	000	0	0
L5	ST1	001	0 1 0	ST0	000	0	1
L5	ST1	001	1 1 0	ST0	000	0	1
L6	ST1	001	0 1 1	ST2	010	0	0
L6	ST1	001	1 1 1	ST2	010	0	0
L7	ST2	010	0 0 0	ST1	001	1	0
L7	ST2	010	0 1 0	ST1	001	1	0
L8	ST2	010	0 0 1	ST2	010	1	0
L8	ST2	010	0 1 1	ST2	010	1	0
L9	ST2	010	1 0 0	ST1	001	1	0

L9	ST2	010	1 0 1	ST1	001	1	0
L9	ST2	010	1 1 0	ST1	001	1	0
L9	ST2	010	1 1 1	ST1	001	1	0

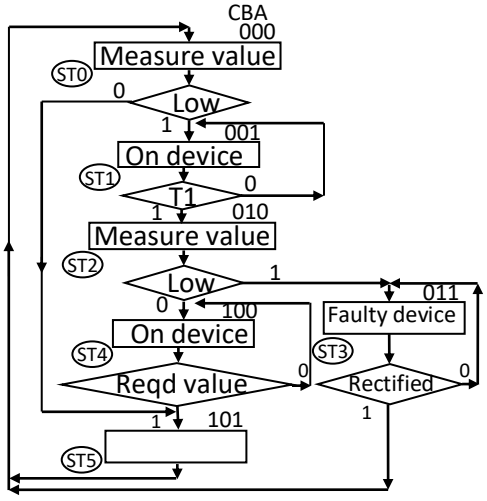


Fig 8: Software flowchart with state codes

By adding state codes to the flowcharts of figure 4, a flow chart of figure 8 is realized. A state transition table corresponding to figure 8 is as shown in table 3. Figure 9 is a microprocessor block diagram implementation of the system. Here the feedback C′B′A′ is not used to drive the system even though they are fed forward as present state CBA. However, they tell the user where the flowchart is in case of any eventuality. This is as a result of the fact that at any point in time, the present state code of the system is known. Thus when a device fault shown in figure 6 occurs, this would be indicated by the present state code CBA=011 at that point.

Table 3: STT of fig. 8

Link path	**Present state name**	**Present state code**	**qualifier**	**Next state code**	**Measure value**	**On device**	**Faulty device**
			LOW T1 RECTIFY REQD				
L1	ST0	000	0 - - -	1 0 1	1	0	0
L2	ST0	000	1 - - -	0 0 1	1	0	0
L3	ST1	001	- 0 - -	0 0 1	0	1	0
L4	ST1	001	- 1 - -	0 1 0	0	1	0
L5	ST2	010	0 - - -	1 0 0	1	0	0
L6	ST2	010	1 - - -	0 1 1	1	0	0
L7	ST3	011	- - 0 -	0 1 1	0	0	1
L8	ST3	011	- - 1 -	0 0 0	0	0	1
L9	ST4	100	- - - 0	1 0 0	0	1	0
L10	ST4	100	- - - 1	1 0 1	0	0	0
L11	ST5	101	- - - -	0 0 0	0	0	0

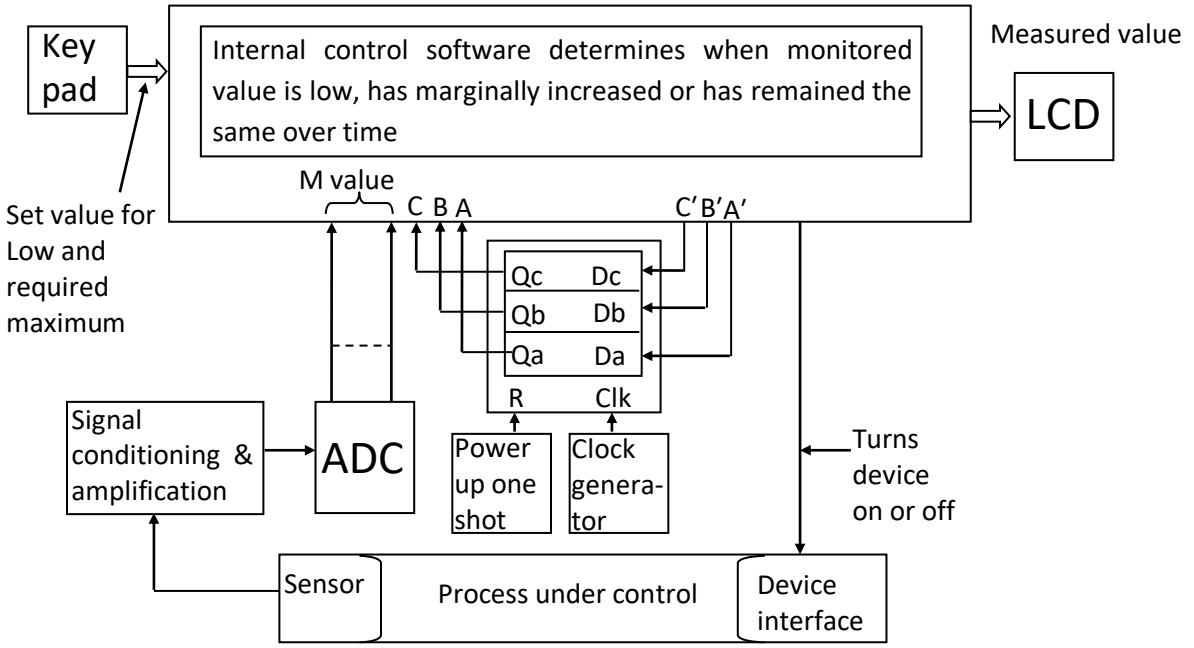


Fig 9: microprocessor based block diagram implementation of table 3

VIII. INDEXING PROCESS STATUS MESSAGE TABLE WITH STATE CODES

When a process under control is being monitored, the state code provides information as to where the process is at a particular time. Therefore by using the state code to index a pre-stored table in memory, the state of the process can be explicitly stated on an LCD or a video display unit (VDU). This is very helpful since every state code is unique and points to only one entry in the pre-stored table.

IX. USING A DIGITAL PH METER

When the process under control involves an action that would be taken only when a mixture reaches a particular pH composition, a digital pH meter (very much like the smart scale can be used to obtain a digital read out of the mixtures' pH, which is then fed back to the processor. This will then compare the feedback information with a set value in order to determine when it is right to carry out the next stage of the process control.

The foregoing highlights some of the steps that should be carefully taken in order to add self-diagnostic intelligence to any process control system.

X. CONCLUSION

Alternative approaches for obtaining diagnostic information used to realize intelligent process control systems has been explored in this paper. This is by no means an exhaustive treatment of the subject matter, but enough has been said to give the reader a firm grasp of the techniques used. It is hoped that this paper will encourage indigenous digital process control engineers to add more intelligence in their products. This is especially useful now that Nigeria hopes to become one of the top 20 industrialized nations by the year 20:20. Intelligent process control systems would no doubt help, as the nation tries to take firm control of industrialization processes.

REFERENCES

[1] K. C. Jain and Sanjay Jain, Principles of Automation and Advanced Manufacturing Systems, Khanna publishers. 2009. Pp 44.

[2] Tocci, Widmer and Moss. Digital Systems, Tenth edition, Pearson 2009. Pp464-469

[3] H. C. Inyiama, C. C. Okezie and I. C. Okafo, "Agent Based Process Control System Design" Proceedings of the peer revieved 2012 national conference on infrastructural development and maintenance in the Nigerian environment. pp 234-252, August 2012.

[4] H. C. Inyiama, C. C. Okezie and I. C. Okafo, "Complexity Reduction in ROM-Based Process Control Systems via Input Multiplexing and Output Decoding", International Journal of Engineering Innovations. Paper No: AJUS-2011-115, (in press)

[5] Curtis D. Johnson, Process Control Instrumentation Technology, 8th edition. Prentice Hall of India, New Delhi, 2006. Pp 1-9.