CHEM+ TECH '80

INTERNATIONAL CONGRESS

Symposium 6

COMPUTERISED PROCESS CONTROL SCHEMES

Dr. S. SRIKANTAN

ABSTRACT

A scheme of computer process control that permits maximum human interaction with the process under control is discussed. A table-driven operating system performs all the routine real-time data acquisition and control functions, and all application programs for modelling and optimisation computations are in a high level language. The scheme permits extensive on-line experimentation with various control strategies, involving Feedforward and Feedback and Cascading of loops, and combinations of Proportional, Integral and Derivative algorithms. The scheme can be adopted for supervisory control as well as DDC, depending on the application. Provisions have been suggested for practical situations like computer shutdown, selective or total isolation in emergencies, checks against program and operator errors, detection and handling of human override actions, etc.

I. 'TRANSPARENT' PROCESS CONTROL SYSTEM

In this paper, the primary concern is the design of a 'See through' computer process control system, which for all practical purposes is invisible to the console operators and process engineers. The operator, in essence, should not have to know about the operating systems, queues and flags but yet be able to interact intimately with the process and the control logic in his native terminology.

Often, Interaction is not a major consideration at all in computer process control applications, and most of the standard process control systems allow interaction only at a very high level — as for management information/process evaluation, start up/shut down sequences, emergency actions, etc. Micro-level interaction with the process and its control is a must for on-line experimentation as development and tuning of control strategies, process identification and optimization etc., but most general purpose systems do not provide this facility as a natural feature. The efforts should be to develop a system which has the inherent capability to accommodate on-line monitoring of every meaningful parameter in the process and its control logic with ease. The computer (rather, the process operator's console — POC) will act as an effective centralized and total information and control device for the process.

The computer software in conventional process control systems basically belongs to two distinct types, — (i) High Level systems, wherein the application engineer writes his process control program in a high level language like Real Time Fortran (Fig. 1), the low level — 'Fill in the Tables' type of systems, wherein all process Data Acquisition and Control functions are built into the standard operating system, and the engineer specifies his requirements through a set of tables (Fig. 2). Both these systems have their specific advantages/limitations in specific applica-

ABOUT THE AUTHOR

Dr. Srikantan (47) is heading the Computer Group of the Electronics Corporation of India Limited, Hyderabad since 1969. He is responsible for the development, manufacture and marketing of Digital, Analogue and Micro computer systems. Dr. Srikantan had been with the Bhabha Atomic Research Centre for nearly 12 years since 1957. He was responsible for the design and development of the Trombay Analog Computer which was commissioned in 1960. After his return from U.S.A., he organised the digital computer development of the Trombay Digital Computer — TDC-12, the first real-time digital computer developed indigenously. He has received the VIKRAM SARABHAI RESEARCH AWARD endowed by Hari Om Ashram in the year 1974 in the field of Electronics and Telecommunications.

```
10    .SNAP
      .STORAGE FLOW, HLIMIT/3.457/,
      LLIMIT/2.957/                         Storage and value assignment.
200   FORMAT  XX, " : ", XX. "CHANNEL       "HIGH", XX.XXXX
      NO." XXXX.
300   FORMAT  XX1 " : ", XX1" ...........   "LOW"   XX.XXXX
                                            Format specifications for alarm messages.
      .PROCESS                              Beginning of executable statements.
      GET (CLOCK) IHOUR, IMIN, ISEC         Get time in specified variables.
      GET (ADC, # 3) FLOW                   Scan ADC, apply CONV3 Conversion.
      IF (FLOW — HLIMIT) 301, 302, 302
302   IF (FLOW — LLIMIT) 500, 401, 401
301   SEND (TTY1, 200) IHOUR. IMIN, I,      Print Alarm message.
      FLOW GO TO 500
401   SEND (TTY1, 300) IHOUR, IMIN, I,      Print Alarm message.
      FLOW
500   EXIT
      .SUBROUTINE CONV3 (OUTPUT,            Conversion law CONV3
      INPUT, DUMMY)
      LET OUTPUT = SQRT (INPUT)
      * C1 +C2
      RETURN
END
```

FIG. 1. HIGH LEVEL SYSTEM REPRESENTATI ON

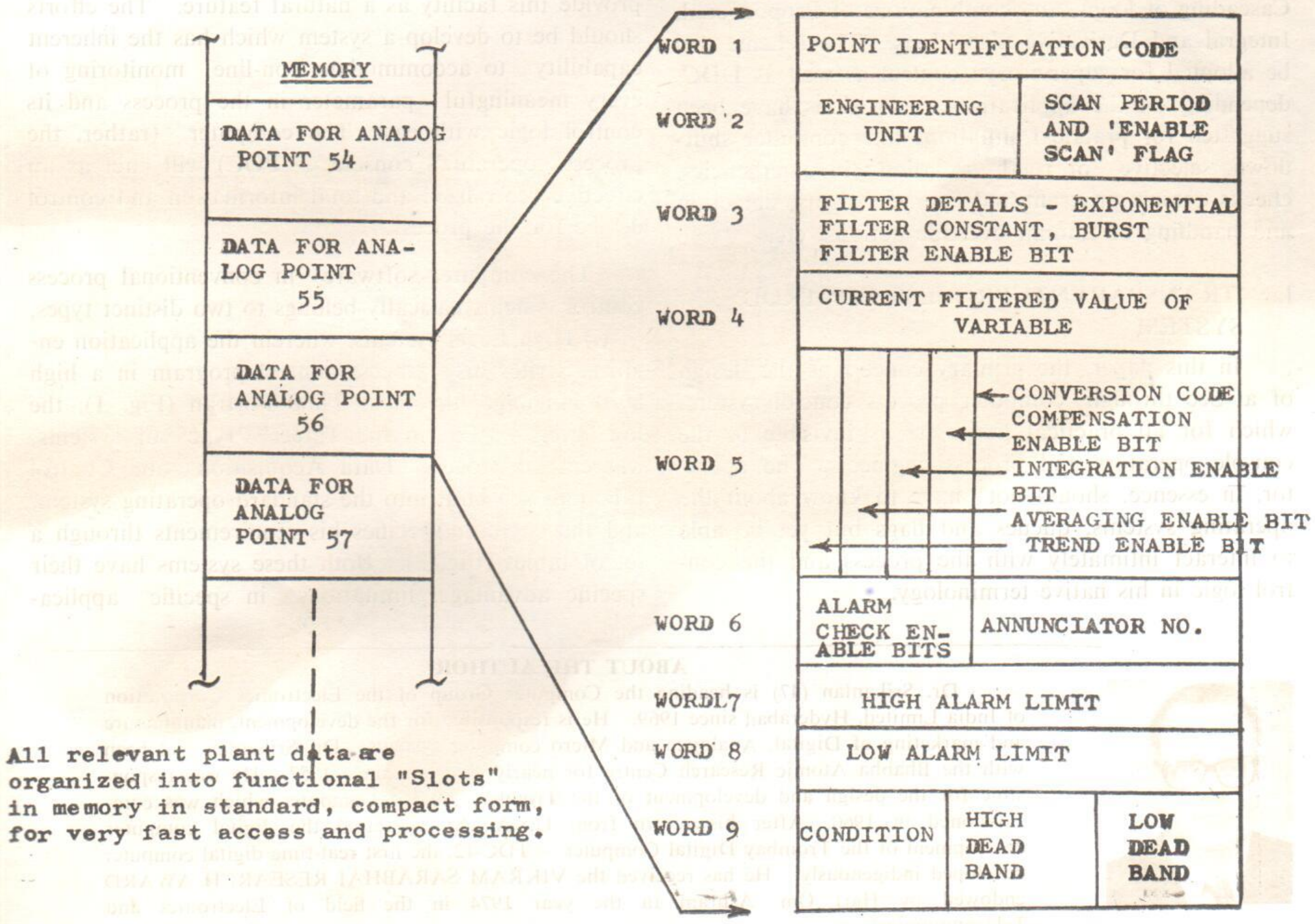


FIG.2. LOW LEVEL SYSTEM REPRESENTATION

tions; while the high level systems are self documenting and very flexible; the latter are extremely efficient, and have fast response time. However, we are currently interested in specific aspects of these systems—the ease with which complex control strategies can be implemented, and the degree of interaction the systems provide thereon.

The computational power and elegant organization of high level language based systems make them ideal for implementing complex real-time process models. But the infinite flexibility given to the programmer in the naming and interpretation, of variables, process parameters and process functions, makes such language based systems difficult for close dynamic monitoring and interaction. For instance, if the console operator has to know the Integral Time Constant (ITC) setting for control loop CO12 at any instant, it is necessary that,

i. The programmer has somewhere in his system, defined that the variable ITC 12 in program FLUEGC is the Integral time constant for loop CO12 and he has also established its access from POC, or,

ii. The operator has to know that he should actually request for the value of variable ITC 12 in program FLUEGC to know Integral Time Constant for CO12.

There could be hundreds of process variables, each with as many associated parameters (alarm limits, set points, PID constants, filter details, etc.) in a typical application, that approach (i) above will involve a huge programming effort to define the functional significance of program variables in real life and strictly adhere to these definitions in the program itself; and also, the dynamic mapping from real life variable to program variable will sap CPU time on-line. Approach (ii) by nature defeats the purpose of console interaction. Only the programmer will be able to interact effectively, and interaction speed will be very low. Hence a pure high level implementation cannot by nature support intimate console interaction (unless one is ready to put in a great effort for this specific end, and does not mind inefficiency).

The low level systems score will in this regard, since all the process variables and parameters have just one identification, based strictly on their function (as High Alarm Limit of F123). But these systems often lack a powerful computation media, with sufficient software interfacing to the process variables. Such systems are hence very successful for data loggers, without much computation. The real major effort in such systems is the identification and standardization of process parameters and functions. It is next to impossible to evolve a standard set of functions and terminologies for all types of processes and applications. Within a range of similar processes, one could be successful.

Neither of these two basic approaches would hence be completely suitable for implementing a highly interactive, versatile process control system. The solution appears to be a hybrid system, combining the power of interaction of low level systems with the computation power and programming simplicity of a high level system. System GDACS (Generalised Data Acquisition and Control System) developed for ECIL's computers is designed on these lines.

II. GDACS FUNCTIONS

The system GDACS is organised to act as a versatile communication medium that establishes a means of correspondence between the three entities in a Process Control Application — the process, the application software (Fortran in Real Time) and the console operator. It is basically a table-driven system (Similar to the one in Fig. 2) with a powerful process interface provided for Fortran application programs and performance computations. The system dynamically maintains a Plant Data Base, which at any moment gives the exact picture of plant status (Fig. 3). The system software consists of the communication elements between the Process, Console and the Application Software, amongst themselves, and with the plant data base. Besides, timing and scheduling functions, safety procedurees and self diagnostics are built in.

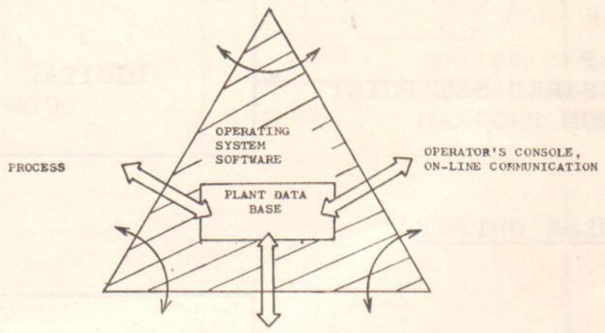


FIG.3. GDACS SYSTEM SOFTWARE FUNCTIONS

The essence of effective communication between the various entities is that all the parameters in the system that are meant for overall access are housed in fixed slots in the plant data base, with respect to their functional significance (for instance, High Alarm

Limit for a point will always be in the seventh word of the block of memory reserved for that point — Fig. 2). This obviates the need on the part of the Application Programmer to define and maintain a large volume of process information and also establish access to them from the process and the console. The task of data acquistion, which is often the major programming effort, is completely handled by the system, as well as all well defined activities like filtering, averaging, history, alarm generation, engineering conversions, compensation, logging etc. Similarly, the minute level operations in closing the loop with due considerations for safety are handled by the system. The programmer thus acts in a macro level, giving powerful commands to the operating system embedded in computations. This contrasts with normal high level systems, wherein, computations are in high level but process operations are done in the very basic level. Example — a command GETVAL in conventional systems might get the raw 12 bit ADC value, but in GDACS it gets the value in the final engineering unit, say °C, removing the need for further engineering conversion. Besides the macro level operation and built in safety, the engineer gets the higher throughput and faster response of an efficient low level Data Acquisition System.

Needless to say, the effort in designing such a system would be meaningful only if it is fairly general purpose with multiple applications, and the success of the effort **hangs a lot on the** success with which we can identify and freeze an exhaustive set of process parameters, communication elements and modes, macro operations, terminology etc., at least for a class of processes and build them into the operating system.

Let us now discuss at length this process of identification and standardization of functions and parameters, with respect to one subsystem—the GDACS control scheme.

III. EVOLUTION OF THE GDACS PROCESS CONTROL SCHEME

A. Supervisory Control :

In the most elementary control scheme supported by (and embedded in) all Language-based Process Control Systems, the user application program calculates the new set point for a control channel (for, say, fuel flow) and the operating system does the job of converting this into Analog or Pulse train output for the Analog Control element in the plant. Pulse train output consists of a specific number of pulses, proportional to the desired change in set point; The pulses drive a stepper motor, which interfaces the computer to the analog controller. A common variation is that the program calculates the 'change' in set point rather than the absolute set point. Fig. 4 shows the simple supervisory control scheme and Fig. 5 is a generalization of Fig. 4 for Pulse/Analog output.

In Fig. 5, it could as well be that the last value of set point (SP) was stored in computer memory, and difference △SP calculated and output. But this is not a reliable scheme, because (i) If computer power fails, and stand by (manual or analog controllers)

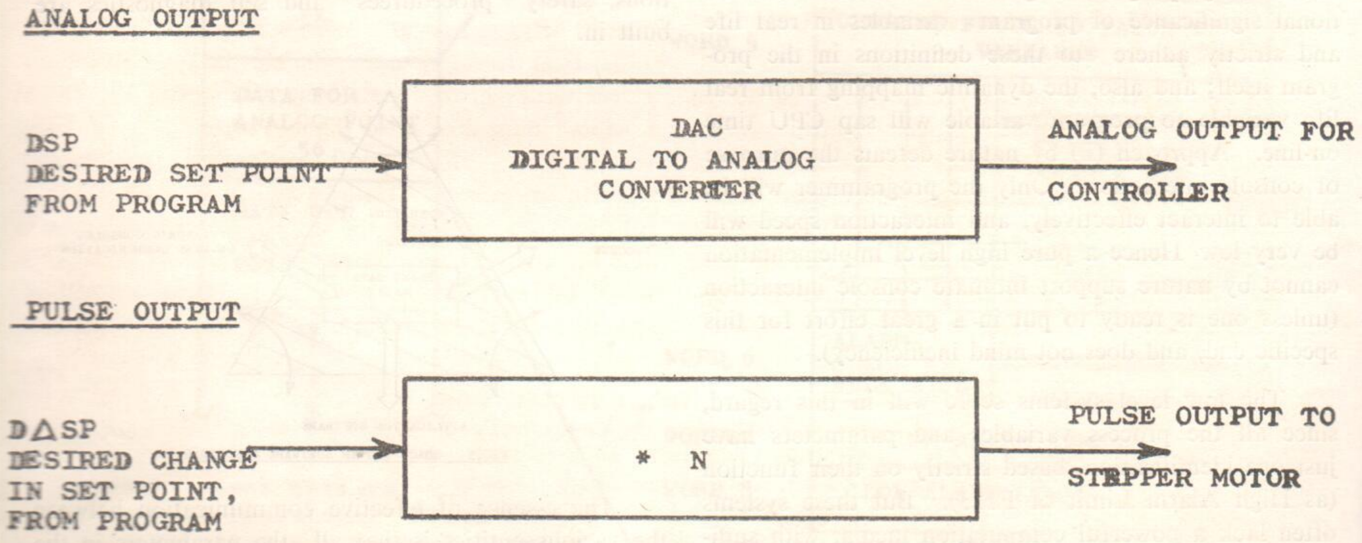


FIG. 4. SUPERVISORY CONTROL ELEMENTS

have taken over, the value of SP as available in memory would not be the current *updated value, so* $\triangle$SP would be wrong and (ii) most controllers provide for manual over-ride facility, wherein an operator can manually after the controller setting, even when computer is active. Such over-ride actions have to be sensed and provided for, before the computer outputs a change of set point command, and an analog input to sense the current controller position becomes essential. Besides, there are two other reasons because of which the actual set point setting in the controller could differ from what it should be. They are :

i. The control output channel is open or bad, and control output does not register on the controller.

ii. The rate at which set point output commands are generated by the program is too high for the hardware (Computer's as well as controller's) resulting in loss of output.

Considering these, an analog input to sense current controller position becomes a necessity; once this is provided. the operating system can as well generate alarm messages whenever it detects a discrepency in the controller position, as in Fig. 6.

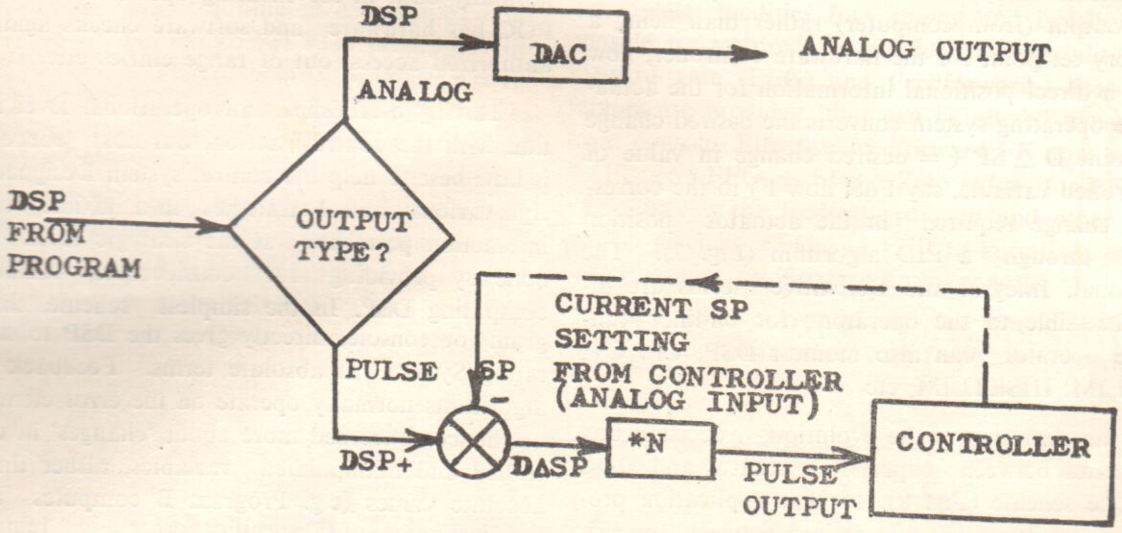
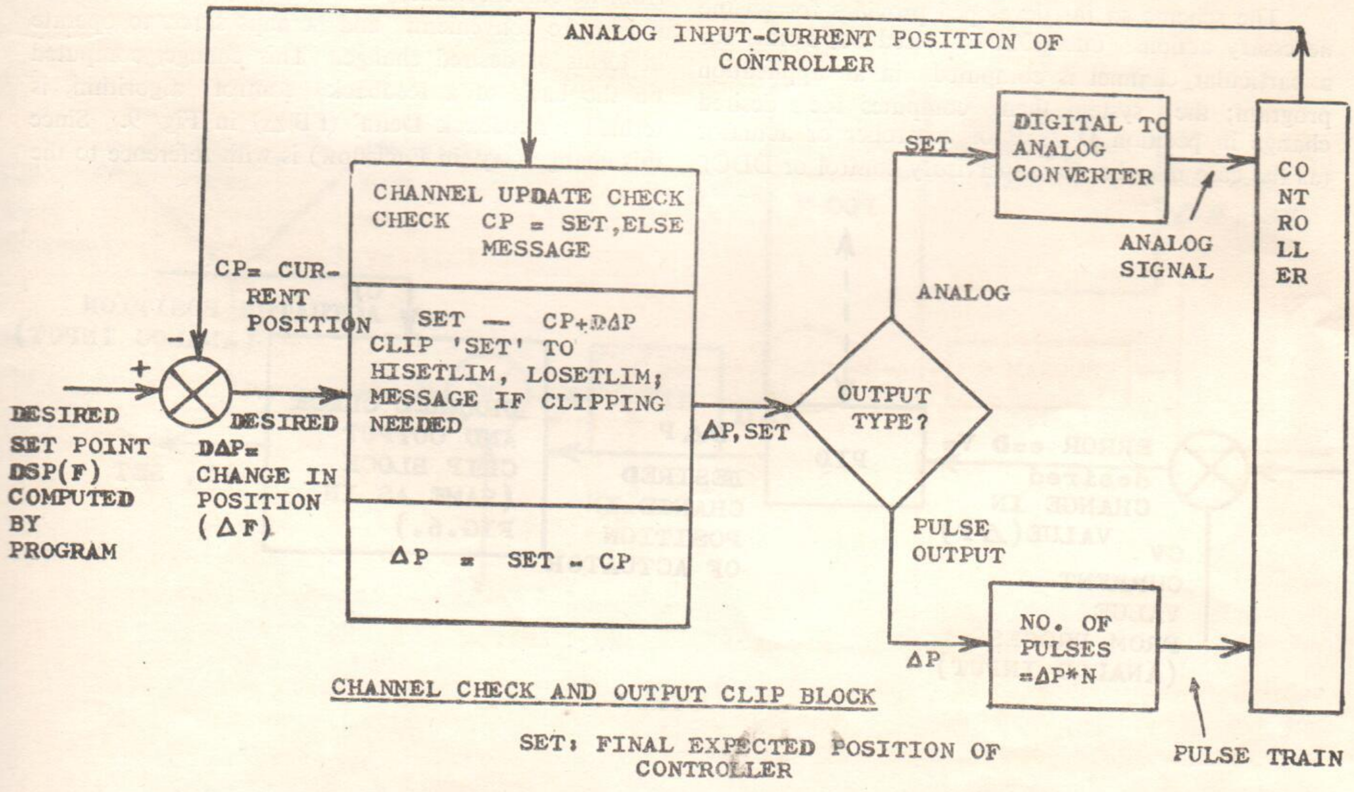


FIG.5 SUPERVISORY CONTROL SCHEME



ANALOG INPUT-CURRENT POSITION OF CONTROLLER
CHANNEL UPDATE CHECK CHECK CP = SET,ELSE MESSAGE
SET — CP+DΔP
CLIP 'SET' TO HISETLIM, LOSETLIM; MESSAGE IF CLIPPING NEEDED
ΔP = SET - CP
CP= CURRENT POSITION
-
+
DESIRED SET POINT DSP(F) COMPUTED BY PROGRAM
DESIRED DΔP= CHANGE IN POSITION (ΔF)
ΔP, SET
OUTPUT TYPE?
ANALOG
SET
DIGITAL TO ANALOG CONVERTER
ANALOG SIGNAL
CONTROLLER
PULSE OUTPUT
ΔP
NO. OF PULSES =ΔP*N
PULSE TRAIN
CHANNEL CHECK AND OUTPUT CLIP BLOCK
SET: FINAL EXPECTED POSITION OF CONTROLLER

FIG.6. SUPERVISORY CONTROL WITH LIMIT AND CHANNEL CHECKS

A safety feature is also built in at this stage, wherein the control channel output is not allowed to cross the high and low safety limits; The value of DSP as computed by program is 'Clipped' to these limits, before being output. The HISETLIM and LOSETLIM of Fig. 6 can be entered from console.

B. DDC Capability :

A step further is to provide for DDC also, wherein part of the functions of the analog controller are taken over by the computer. The input to the operating system still remains the desired set point DSP, but the output (from computer) rather than being a supervisory set point for the hardware controller, now becomes a direct positional information for the actuator. The operating system converts the desired change in set point D $\triangle$ SP (= desired change in value of the controlled variable, say Fuel flow F) to the corresponding change required in the actuator position (D $\triangle$ P) through a PID algorithm (Fig. 7). The Proportional, Integral and Derivative terms are directly accessible to the operator for on-line tuning. The operator can also monitor DSP, CP, CV, LOSETLIM, HISETLIM, etc.

As the next step in the evolution, a decision box discriminates between supervisory control and DDC in a single scheme (Fig. 8) so that application programs do not have to differentiate between supervisory control channels and DDC channels.

The scheme so far developed provides for all the necessary actions once DSP (Desired Set Point) for a particular channel is computed in an application program; the system then computes the desired change in position D $\triangle$ P of controller or actuator (as the case may be, for supervisory control or DDC), checks if the channel has registered the last output, clips the desired change in position to safe limits, and sends out an analog output or pulse train, as required. All these are routine functions, that would otherwise have to be done by Application Programs, at the language level. which would be basically inefficient and unsafe for such low level functions, besides not allowing for extensive interaction.

Though, as mentioned, all the parameters are directly accessible to POC (as well as programs), a certain degree of access inhibition and regulation are normally built in for ensuring operational safety. The POC has hardware and software checks against unauthorized access, out of range entries etc.

Having established an operational level interaction with the control action, our next concern now is how best to help the control system Designer to design various control strategies in a flexible way, with interaction permitted at the strategic level. This is done by providing for considerable variations in computing DSP. In the simplest scheme, the program (or console) directly gives the DSP to the Operating System in absolute terms. Feedback control algorithms normally operate on the error element and are hence concerned more about 'changes' in the controlled and manipulated variables, rather than their absolute values (e.g. Program B computes that for the correction of a specific error in temperature value, fuel flow has to change by +11% of range from its current value, whatever it is). In such cases, it is more convenient, and perhaps safer, to operate in terms of desired change. This change, computed on the basis of a feedback control algorithm, is termed 'Feedback Delta' (FB $\triangle$) in Fig. 9. Since this change (say, in Fuel flow) is with reference to the

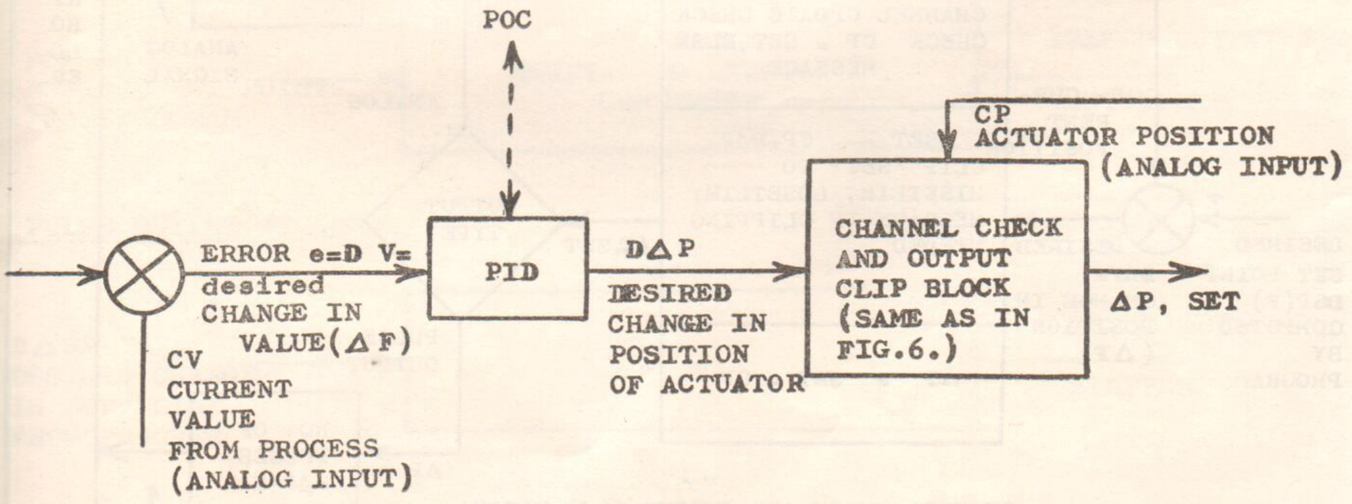

POC
ERROR e=D V= desired CHANGE IN VALUE(ΔF)
CV CURRENT VALUE FROM PROCESS (ANALOG INPUT)
PID
DΔP DESIRED CHANGE IN POSITION OF ACTUATOR
CP ACTUATOR POSITION (ANALOG INPUT)
CHANNEL CHECK AND OUTPUT CLIP BLOCK (SAME AS IN FIG.6.)
ΔP, SET

FIG. 7

current value (of fuel flow) the current measured value MV from the plant is added on to FB △ to get the desired set point DSP (for fuel flow) in Fig. 9.

Simple Feedback control strategies are often ineffective and lead to inefficient process operation, because, Feedback algorithms are of post-error, corrective nature—they act only after the controlled variable has already deviated from its desired value. Further, any corrective action taken by the control system is not felt until after the changing conditions have been propagated round the entire control loop, with considerable time lag, resulting in a sustained non-optimal operation, particularly in such processes with large time lag. Feedforward control is used to overcome these drawbacks, (if the process disturbances can be measured), by taking control action before an error occurs in the controlled variable. The success of Feedforward strategy depends on how accurately the process disturbances can be monitored and compensated for, and since it is difficult even to identify all disturbances, let alone monitor them, exacting feedforward models can never be developed. The most useful control strategy is hence a combination of Feedforward and Feedback, the latter operating on the small deviations that arise due to the incompleteness of the process model used in the feedforward path.

In Fig. 10, which provides for a mixed Feedback-Feedforward control, Feedforward is handled in terms of FF △, the change required in (Fuel Flow) set-point in order to maintain (temperature) conditions, in spite of disturbances just monitored (charging of furnace, etc.). Since the program models used to calculate FF △ are normally very complex and incomplete, facilities for tuning and isolation from console are required. To provide this capability, Feedforward gain (FFG) and Feedforward Bias (FFB) factors are provided, that can be altered dynamically from console. Effective feedforward FF now becomes = FF △ * FFG + FFB. Zero values for FFG and FFB disables the feedforward path and other values can be given to FFG and FFB to curtail or enhance the computed feedforward from console.

The scheme evolved so far is complete for a single level, feedforward cum-feedback control, with

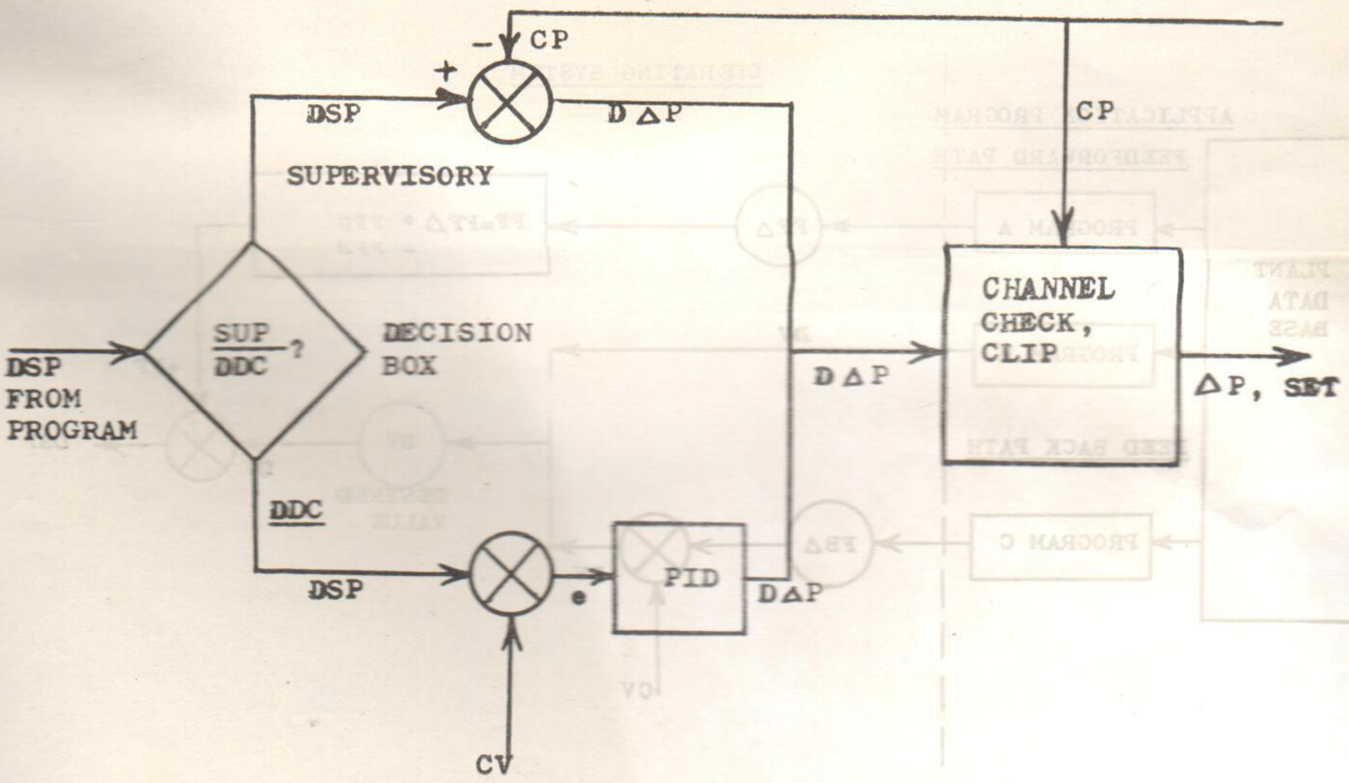


FIG.8

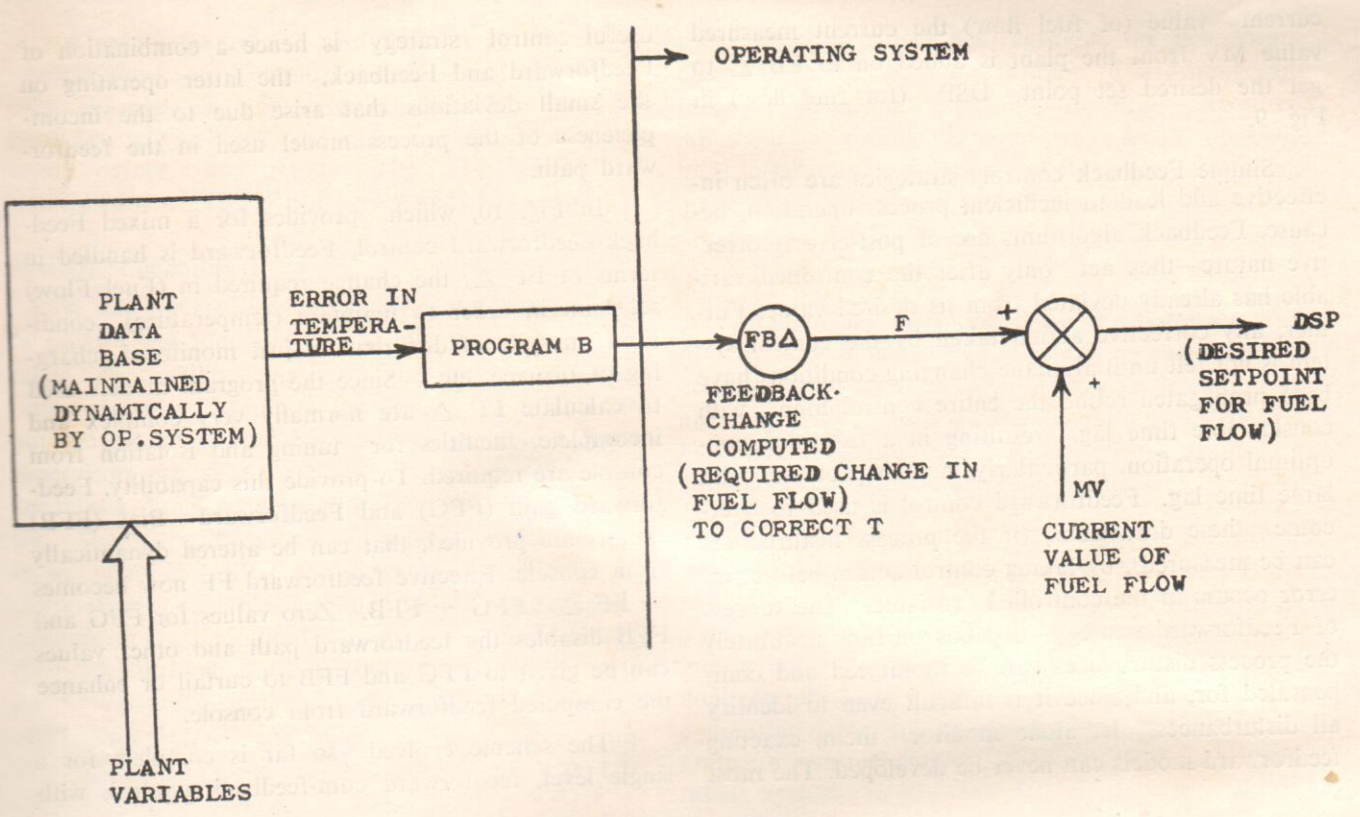

OPERATING SYSTEM
PLANT DATA BASE (MAINTAINED DYNAMICALLY BY OP.SYSTEM)
ERROR IN TEMPERATURE
PROGRAM B
FBΔ
F
+
DSP
(DESIRED SETPOINT FOR FUEL FLOW)
FEEDBACK CHANGE COMPUTED (REQUIRED CHANGE IN FUEL FLOW) TO CORRECT T
+
MV
CURRENT VALUE OF FUEL FLOW
PLANT VARIABLES

FIG. 9 FEEDBACK CONTROL

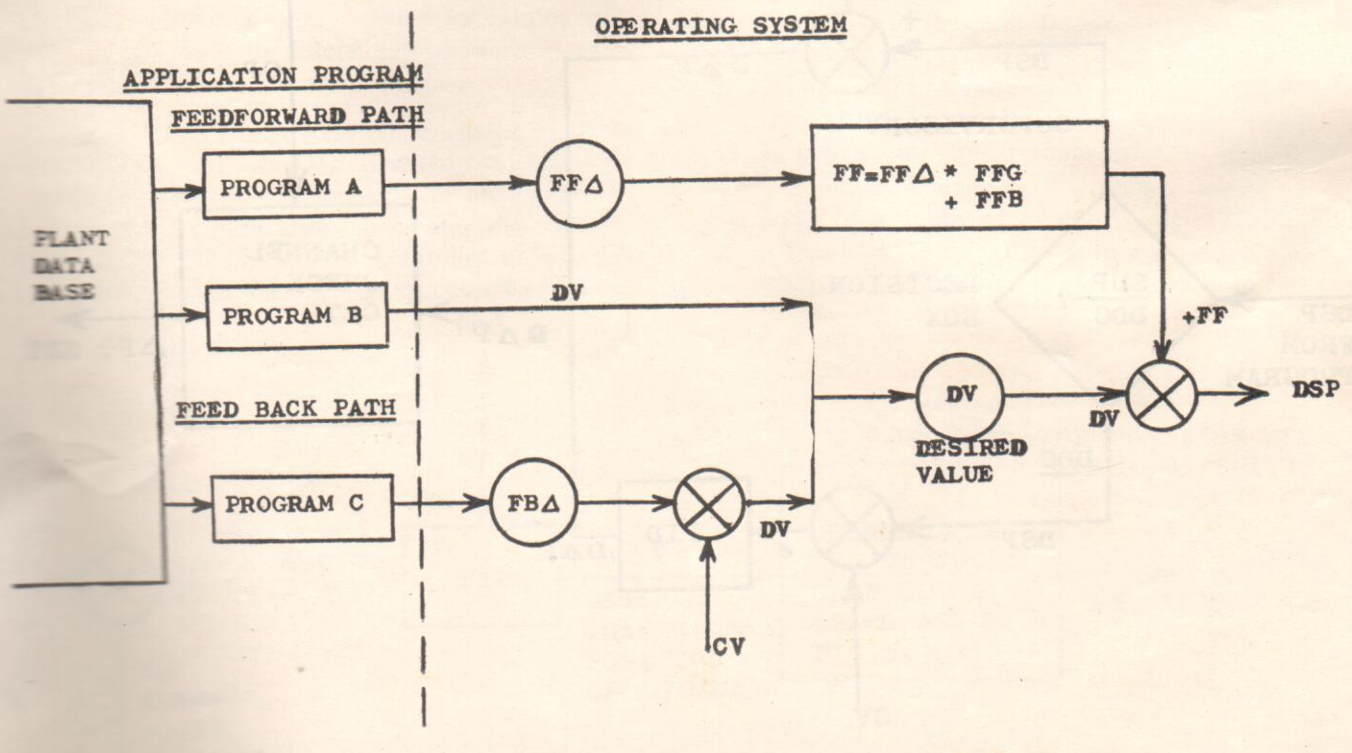


FIG.10: FEEDFORWARD CUM FEEDBACK CONTROL

supervisory control/DDC, Analog Output/Pulse output, fail safe with respect to computer shut-down/restart and operator over-ride actions, and completely transparent to the console operator.

The suggested scheme can also be extended to the more advanced control technique — Cascade control. Here, the output of one level becomes the FB △/FF △ of the lower level in the cascade.