Proceedings of GMC2003:
Gas Machinery Conference
October 6, 2003 Salt Lake City, Utah

RECIPROCATING COMPRESSOR PERFORMANCE PREDICTIONS: CONTROL METHODOLOGIES FROM THE PLC TO THE PC

Dwayne A. Hickman
ACI Services, Inc.

Roy Milum
Siemens Energy & Automation

ABSTRACT

To gain optimal control of reciprocating compressors, it is necessary to predict loads and flows throughout the unit's defined operating map. Moreover, for automated stations these predictions need to happen at the control level – a PLC and/or control panel. Thus arises the need to model single and multistage compressors with a good degree of accuracy within the abilities of control devices. Furthermore, similar prediction abilities must lie with Gas Control and station operators via high-end performance prediction software.

Typical compressor performance software packages generate useful load and flow predictions; however, they almost always do so free from the effects of pulsation and other real world phenomena. Unfortunately, nearly 100% of reciprocating compressors have noteworthy pulsations. This means that to truly control a unit based on load and flow, adjustments to theoretical load predictions must be made that consider the effects of pulsation et al. Fortunately, some motors, engines and peripheral devices have sensors that can generate real-time, measured loads and/or flows.

In many control situations, control is based simply on predicted load from parameters such as fuel usage curves or electricity consumed. Using feedback loops, decisions are made to load or unload the unit. These types of methods lend themselves reasonably well to simple, single-stage models using only clearance pockets for load control. Nevertheless, even in simple cases, a careful review of the compressor and its safe operating map is still very much warranted.

In general, load control of compressors is typically done with clearance pockets, speed adjustments, end deactivation, timed suction valve closings, suction throttling and bypass. Some of these methods are more likely than others to contribute to excessive pulsations. Furthermore, concerns with curve crossing, net rod loads, non-reversing rod loads, high interstage temperatures and pressures, low volumetric efficiencies, blank-off, and stage throttling can quickly arise. As such, more demanding methods for load control must often be employed. Today's advanced PLCs can handle many of the requirements for today's newer, more robust and exacting control methodologies. However, even these newer devices can be taxed by the amount of calculations and storage required to effectively model many applications.

This paper will cover theoretical predictions for Load and Flow for single and multistage compressors, implementation of those methods into PLCs with minimal coding, and methodologies for using real-time measured Loads and Flows to fine-tune the theoretical predictions. Various software packages will be utilized to assist in the learning of these processes, as well as spreadsheets to indicate the simplicity of the algorithms going into the PLCs, including flow balancing and gas compressibility algorithms, valve loss and parasitic loss horsepowers, and flow slippage. Integration of PC and PLC to control compressors will be discussed. And finally, results from a recent implementation will be presented.

INTRODUCTION

With the increasing demand for natural gas in the USA for heating and generation of electricity, comes the need to compress more and more gas within the current transport network. Sometimes, new pipelines and compressor stations are required. At other times, reapplications of current facilities are favored.

Regardless of new or reapplication, maximizing the utilization of the compressor-driver usually leads to increased flows and to reduction of operating costs. To maximize utilization, complex and adaptable models of the unit must be created and implemented in a reliable control system. PCs can easily model complex systems and tune them with real-world measured data: PLCs can reliably control compressor units and safely engage hardware changes in real-time. Combining the power of both into a single integrated solution provides unparalleled control abilities.

By using a PC to calculate the intense numeric algorithms used in modeling compressors, and by using the PLC to actually control hardware changes and maintain safe operations, the unit being controlled can be more effectively utilized – more flow means more revenue, and lower BHP/MM means lower operating costs.

NOMENCLATURE

IEC 1131-3	International Standard for Programmable Controller Programming Languages
FBD	Function Block Diagram
BHP	Brake Horsepower
HMI	Human-Machine Interface
OPC	OLE for Process Control
PC	General computer workstation
PLC	Programmable Logic Controller
SCL	Structured Control Language
SCADA	Supervisory Control And Data Acquisition
STL	Statement Lists

Reciprocating compressors have been automated for many years. As with most automation tasks, the primary issues are safety related. To this end, many sensors are monitored and recorded. From this array of sensor data, warnings, alarms, and shut downs can be implemented and appropriate safety systems can be engaged. Secondary issues are related to efficient control of appropriate hardware.

A basic automation project would allow a station operator the ability to change unit speed, open and close volume pockets, deactivate ends, etc. from the convenience of the station control room. Typically, the operator would change the unit's load configuration based on information such as fuel usage of engine, expected changes in inlet or outlet pressures, data from tables or from reviewing printed performance curves.

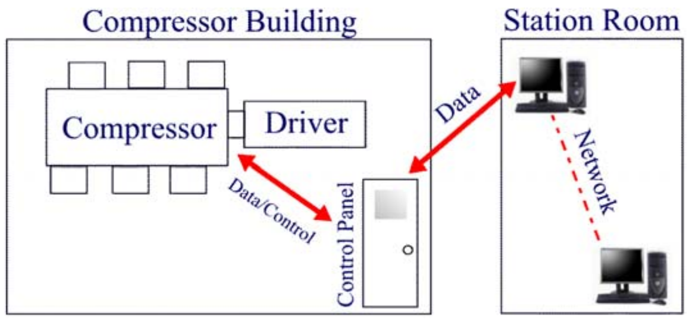


Figure-1: Typical Compressor Station

More advanced automation projects provide not only for safe operation, but also for increasing the performance capabilities of the unit. Utilizing the driver at its peak operation and operating the compressor at its ideal load configurations can usually result in compressing additional gas often at lower BHP/MM rates. Developing methodologies for achieving these ideal operating modes can be non-trivial.

In general, theoretical predictions based on just hardware geometry are useful, but cannot typically account for real-world phenomena. Pulsations, real valve losses, parasitic losses, low volumetric efficiencies, effective versus real clearance factors, lubrication and friction issues, real slippage, cylinder preheating of inlet gases, real pressure drops and others all contribute to less than perfect accuracy for predictions of load and flow. Hence, today there is an increasing trend toward real-time measurement of loads and flows.

Items such as fuel usage rates and inlet manifold pressures, along with associated speed can be used to predict the loading on an engine. Curves and/or functions based on these criteria can be developed when the engine is at the OEM's test facilities. In the case of two CAT engines for a Talisman project in Canada, relatively simple curvefit functions were derived that predicted the load within ±2.0% of the test facility's measured loads, with an average of ±0.5% over the engine's entire load map (75% to 100% speed, 50% to 100% applied load). While this is a great start, as the engine is used, deviations from the original predictions will increase as the equipment acquires more run-hours. Future developments from driver OEMs will likely allow for the driver's control panel to predict real-time load based on the OEM's proprietary calculations and years of research.

For electric motors, load on the driver can often be readily monitored by measuring the electricity consumed by the motor. Caution needs to be applied to sensors that can be affected by the electromagnetic fields of other motors, and/or the spiking associated with bringing other motors online/offline. Care must be taken to properly calculate the load at the frame coupling based on current and voltage with consideration of the power factor, electrical losses (hysteresis and excitation) and mechanical losses (windage/fan and bearing friction/lube oil shear forces). PG&E used measured loads from electric motors to augment their control of two compressors at their McDonald Island facilities. This integration lead to improved load predictions when the active control permissives required the PLC to change the current load step. It also allowed for alarming when measured and predicted values deviated by certain percentages – an indication of pending problems.

In regards to the compressor itself, there are also some methods of measuring load. Real-time measurement of compressor load can be accomplished by commercial packages such as Windrock's HP-Guard system, or embedded systems from the OEM, such as Ariel's DMS system and Dresser-Rand's RECON ROM system. These systems measure internal cylinder pressures, flow rates, pressure drops, etc. Then, after some calculations arrive at the current load on the compressor at the coupling to the driver. An alternate method, and one that is more prevalent in this industry, is that of using analyzers to measure loads at key operating points.

Care must be extended to methodologies that use real-time measured loads. That is, measured values can only be taken when the compressor and driver are actually experiencing those conditions. Thus, if the current conditions are in fact safe, then all is great: if the current conditions are not safe, then you need to quickly change conditions or shut down. Hence, relying upon just measured values for safe operating would be analogous to blindly walking around in a landmine field with the mine detector directly beneath your feet.

In contrast to real-time measuring, performance prediction by thermodynamics theory readily dictates safe areas of operation without the unit having to experience those conditions – in fact, performance can be predicted prior to the unit running, or even being built. For applications that need the best of both real-time measurements and theory, the solution is obvious – combine the two together. That is, use measured values to adjust the theoretical models to generate predictions closer to observed values. And, if real-time load and/or flow measurements are available, allow for dynamic tuning of the models to use those values for even more accurate predictions. For a particular test application, Figure-2 represents areas (light) where the theory needed to be adjusted positively to match measured values and where the theory needed to be adjusted negatively (darker). This is plotted against speed and compression ratios for a single load step.

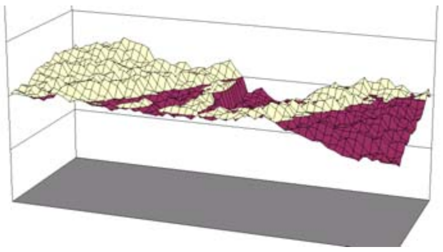


Figure-2: Theory versus Measured Discrepancies

Applying a hypothetical solution to a complex problem to a real-world application is where problems usually surface. The typical control PLC is often already taxed with monitoring sensors and equipment. Coding PLCs to collect and store large amounts of data, and then to process those volumes of data via statistical filtering and averaging algorithms in one, two and even three variables of freedom, can quickly become a challenge. In fact, many PLCs that currently control compressors by theory-based algorithms usually merge all cylinder ends per stage and perform very simplified load/flow calculations. Most do not calculate net rod loads but rather only simplified gas rod loads at flange pressures, and rarely do they calculate degrees of rod reversal. Many PLCs utilize simple algorithms to predict gas compressibility factors, while others jump between multiple algorithms simply based on non-convergence. Also, most PLCs that do correct loads for pulsations often do so by simple curvefit multipliers (typically constants, quadratics or cubics). Methods used to tackle multistage compressors are varied and are often very specific to a unit and to a defined operating map. Fortunately, the required computing power to do a thorough job of modeling the compressor is readily available via the PC.

PCs running Windows® NT/2000 have a respectable history of use in control applications as these operating systems are more robust and stable than most other Window® operating systems. But still, they do not come close to the stability of PLCs as control devices. PLCs tend to run much more basic and specific code, while a general operating system allows for complex and generic code. Managing simpler code is easier, and thus there are a lot fewer concerns with a PLC crashing than with a full-fledge operating system crashing. Furthermore, PLCs can usually restart within (milli-) seconds, while Windows® based operating systems can take upwards of two to six minutes to reboot and restart all software.

Because of the PC's power and ease of programming, it is desirable to have it do all of the complex calculations. Because of the PLC's stability, it is desirable to have it control the unit at all times. An ideal solution would be one in which the integration of the PLC with the PC is realized, as well as the integration of theoretical predictions with actual measured values.

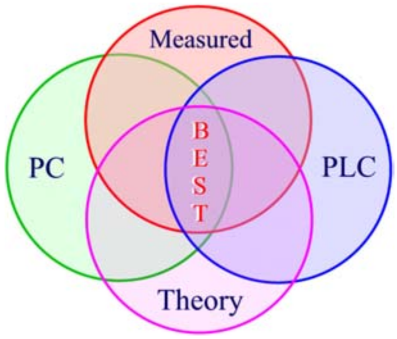


Figure-3

For safety, the PLC needs to be able to select which load steps are desirable and to only engage those load steps that are safe at the current operating conditions. This means that the PLC needs predictive algorithms for load and/or flow, as well as algorithms for predicting safe operating map. Since a complex model is typically beyond the abilities of most PLCs, simpler models must be employed in practice. These conservative models will keep units safe, but typically will sacrifice prediction accuracy as a consequence. This sacrifice often leads to less than ideal use of the compressor and driver, which in turn may lead to higher operating costs and/or reduced revenues.

To achieve desired accuracy, PC prediction software packages can model units in great detail. Intense algorithms will model gas compressibilities, rod reversals and net rod loads, pressure drops, and valve and parasitic losses, as well as being easily altered and adjusted to reflect known operating conditions. However, since PCs have a long history of crashing, they are not typically ideal for directly controlling certain processes.

To create an ideal control situation, one must use the stability of the PLC and the computing power of the PC. Namely, program the PLC with the control philosophy and with conservative prediction algorithms, but also allow it to retrieve more accurate predictions from an attached PC. Thus, as long as the PC-to-PLC communications are active, the PLC controls the unit using the PC's more accurate model and thus utilizing the compressor-driver unit as effective as possible. However, if the PC-to-PLC communications are interrupted, then the PLC resorts to its internal conservative methods, which may not fully utilize the unit, but nonetheless will keep it safely running.

Keeping the unit running safely regardless of the state of the PC is really the critical element. Appendix-A lists the pseudo code used for the test application. This code was easily converted to actual PLC code, in this case for a Siemens PLC, but could just as easily be converted to other commonly used PLCs.

Today's PLCs offer an automation engineer several choices of programming environments on the same platform. Therefore, when considering what environment(s) to use when writing PLC code the programmer must consider the strengths of each environment relative to the type of code to be produced. The programming options considered for the pseudo code conversion, were standard ladder logic, Function Block Diagram (FBD), Statement Lists (STL), and Structured Control Language (SCL).

Ladder logic and FBD are more suited to sequencing tasks and algorithms with relatively few arithmetic calculations. Because of the large number of arithmetic calculations and arrays documented in the pseudo code, SCL was selected as the programming environment. STL, an assembler-like language, is also appropriate for these types of calculations. However, it is not universally understood across a large section of programmers in the Oil and Gas industry. SCL is a Pascal-like high-level language that is suitable for programming complex algorithms or for tasks that require large amounts of data management. Because of its Pascal-like look, SCL permits easier and faster programming and provides an environment that allows for improved comprehension among typical programmers with BASIC and 'C' experience. A sample of the program is provided in Appendix-B.

Regardless of the programming environment selected, the programming languages provided in the S7 package comply with international programming standard IEC 1131-3. Programs written in IEC 1131-3 compliant environments can be more easily transported across other 1131 compliant PLC platforms.

The IEC 1131-3 standard, developed on well proven programming techniques in use today in many control products, has brought many common practices and techniques together to produce a well defined suite of languages. IEC 1131-3 provides a framework for developing structured control software and facilitates the development of reusable function blocks thus directly improving productivity of an application from development through commissioning and long-term maintenance. IEC 1131-3 is available in PLCs from Mitsubishi, Phillips, Rockwell, Siemens and others.

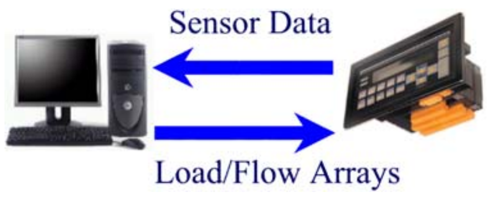


Figure-4

COPLEY STATION TEST APPLICATION:

Copley station is located in West Virginia and consists of three single stage compressors. The unit used in the test was Unit #3 [Cooper-Bessemer GMVH: 3-Throw, 14-inch stroke, 330 RPM, 16¼ inch cylinders) (Figure-5). SIMATIC WinCC (Figure-6) was used as the SCADA software with Siemens S7 PLC (Figure-7) and panel (Figure-8) and a remote PC running Windows® NT4. eRCM Controller™ software (Figure-9) was used on the PC to model the compressor unit. This software read sensor data from the unit (T_S, P_S, P_D, RPM) and generated an array of load predictions, one load prediction for each load step. If conditions were such that certain load steps should not be used, then their associated load prediction was so flagged. To verify PC-to-PLC communications remained active, a watchdog value was continually sent to the PLC. The PLC then used the PC's load array to best determine which load step to engage at the current operating conditions.

Figure-5: Copley Unit #3

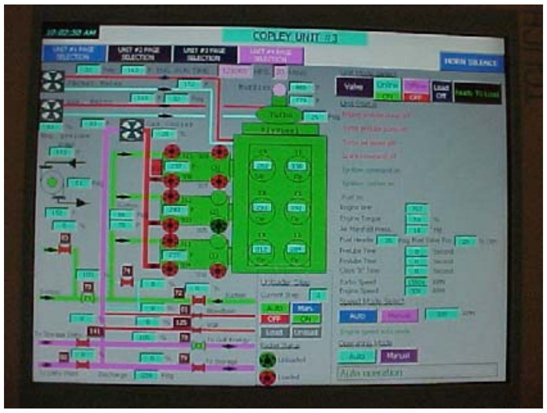


Figure-6: Local HMI Software

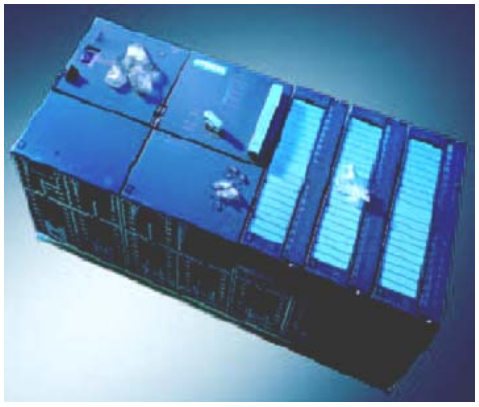
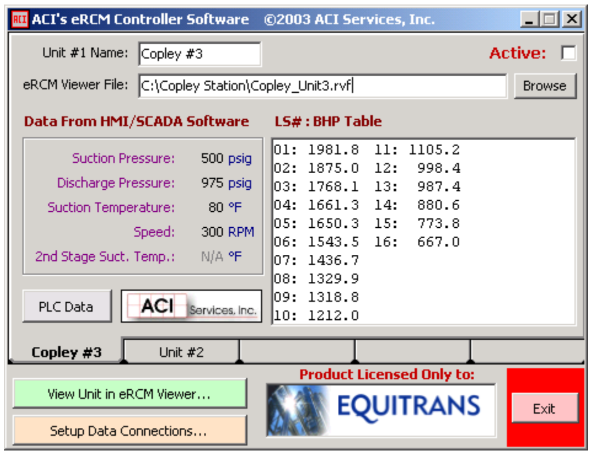


Figure-7: Siemens S7 Controller

Figure-8: Local Control Panel


ACI's eRCM Controller Software ©2003 ACI Services, Inc.

Unit #1 Name: Copley #3

Active:

eRCM Viewer File: C:\Copley Station\Copley_Unit3.rvf

Data From HMI/SCADA Software

Suction Pressure: 500 psig

Discharge Pressure: 975 psig

Suction Temperature: 80 °F

Speed: 300 RPM

2nd Stage Suct. Temp.: N/A °F

LS# : BHP Table

```
01: 1981.8  11: 1105.2
02: 1875.0  12:  998.4
03: 1768.1  13:  987.4
04: 1661.3  14:  880.6
05: 1650.3  15:  773.8
06: 1543.5  16:  667.0
07: 1436.7
08: 1329.9
09: 1318.8
10: 1212.0
```

Product Licensed Only to: EQUITRANS

Figure-9: eRCM Controller Software

During periods where the PC-to-PLC communications may be interrupted, the PLC is coded to switch over to its simplified and conservative, internal performance prediction algorithms. These routines were written by ACI and coded by Siemens to closely, albeit conservatively, match those generated by the eRCM Controller™ software.

FIELD RESULTS:

- **Initial Software Installation**: An important issue with installing additional software onto your HMI control station is that there is a possibility that the new software can make the system less reliable – by either directly conflicting with the HMI software while running, or indirectly by overwriting known and stable Windows support files during installation. To this end, one may elect to either install the additional software onto a different computer and allow for the data exchange over the network, or install the software after verifying with the HMI software provider that any updated Windows support files will not affect the HMI software. In the case of Copley, the software was installed directly onto the HMI PC. Initial standard installation failed to install the software as the HMI software had locked certain Window files it used for database support. A new installation program was generated that did not try to replace the existing Windows database files with newer versions. After installation of the ACI software, both the HMI and the ACI software were tested to make sure they ran properly.

- **OPC Integration**: While integration to the HMI via OPC was very easy for single element tags, one feature that OPC lacks is the ability to nicely pass arrays of data to and from the server. Different methods of overcoming this limitation were reviewed – including using the OPC's DDE abilities, and passing concatenated strings and then parsing them with a macro in the HMI control software to specific PLC memory addresses. Eventually it was commonly desired to treat each load step element with its own register. Fortunately, the Siemens software allowed for easy programming of this on the HMI software side.

- **Data Transfer Speeds**: Initial tests were performed at 100 millisecond intervals to validate software communications at higher data transmission speeds. Eventually, data exchange was slowed down to one-second intervals. For each compressor, the PC software read four (4) tag values from the HMI software (P_S, T_S, P_D and Speed) and wrote data into seventeen (17) tags (one load for each of the sixteen load steps, and a WatchDog value).

- **Expected versus Observed Results**: *Original project startup date was changed. Consequently, observed data was not available by the date required for submission of this paper. If data is available by the paper presentation date, an addendum will be passed out to attendees.*

- **Changes to Controller Software**: One item added to the control software after initial design was the ability to toggle certain load steps on/off via the Windows software. This new feature allows an analyst to take analyzer data from the unit without worry that a load step change may occur during the data collection and thus, invalidate the data.

- **Results of Interrupting PC-to-PLC Link**: When the connection is lost, the WatchDog value is no longer written into the PLC. The PLC code considers the PC-to-PLC connection lost whenever the WatchDog value fails to change within five (5) seconds. With the connection lost, the PLC retuned to calculating the loads per load step via its conservative algorithms.

- **Results of Re-establishing PC-to-PLC Link**: As soon as the connection is made, the WatchDog value refreshes in the PLC. Future decisions to load or unload will once again be made based on the more accurate load predictions calculated by the PC.

- **Effort to Upgrade Compressor Model**: In the case of the PC compressor model, the compressor model was created and edited with ACI's eRCM software. Changes to the model were done using a friendly Windows-environment. The new file is then copied to the PC running the eRCM Controller software. Changes to the PLC modeling of the compressor are of two distinct flavors: simple and more involved. In the case of the simple changes, only a new data table was required to upload into the PLC. The was typical for changes in clearances, tuning items, safety related cutoff issues, changes in gas analysis, changes in pressure drops, etc. The more involved changes were those that affected load steps. Again, a new data table was simply uploaded to the PLC to handle the performance predictions of the new/altered load step. However, since actual engagement of hardware items is associated with each load step, changes to load step definitions typically involve re-writing of some code, especially if new hardware has been added.

- **<u>Tuning Model to Measured Loads and Flows</u>**: A major reason for using the PC to predict the compressor performance is that the PC can effortlessly review results from portable analyzers, or from real-time load measuring devices such as Windrock's HP-Guard System, to adjust theoretical predictions to more closely match measured values. This collection and processing of data has no impact on the PLC and requires no changes to the PLC's code. Furthermore, for those cases where pulsation effects are affected by the number of other compressors currently running, a separate tuning model can be developed for multiple scenarios and appropriately engaged when needed. Again, without ***any*** changes to the PLC's code.

- **<u>Benefits of PC/PLC Combined Model</u>**:
  - Identical performance prediction code in every PLC
  - Identical load control code in every unit PLC
  - Easy tuning of compressor model without code changes to the PLC
  - More efficient use of compressor and driver
  - Maximum utilization of compressor operating map
  - Failsafe, backup conservative PLC load predictions
  - Lets PLC concentrate on compressor safety, monitoring, and hardware changes without the complexity of sophisticated performance prediction algorithms – see Figure 10.
  - Allows for alternate compressor models of same unit (IE certain hardware removed for repairs, multi-compressor pulsations disparities, etc.)
  - Lower operating costs and more production.

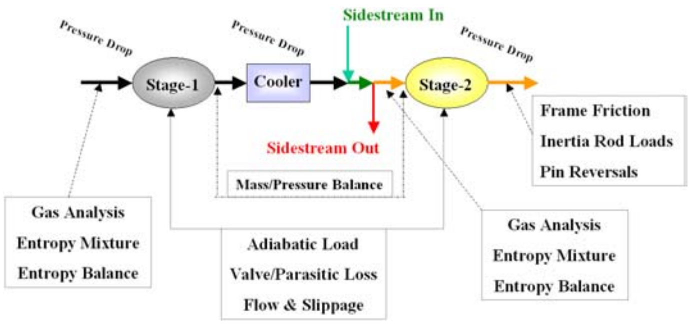


Figure-10: Generic Two-Stage Model

OVERVIEW OF PC VERSUS PLC ALGORITHMS

Item	PC Model	PLC Model
Max Allowed Discharge Pressures	●	●
Max Allowed Discharge Temperatures	●	●
Rod Loads: Gas Pressures at Flanges	●	●
Rod Loads: Net/Inertia Rod Loads	●	○
Low Suction Volumetric Efficiency	●	⭘
Low Discharge Volumetric Efficiency	●	○
Gas Compressibility Calculations	●	⊙
Rod Reversal Issues	●	○
Pressure Differential Limits	●	●
Interstage Pressure Balancing	●	⊙
Tuning towards Measured Values	●	⭘
Individual Head/Crank Calculations	●	○
Entropy Based Thermodynamics	●	○

●*=Excellent,* ⭘*=Okay,* ⊙*=Weak,* ○*=Seldom Calculated*

CONCLUSION

When there are needs to increase flows and/or to reduce operating costs, then Station Operations may often look towards maximizing the utilization of the compressor-driver. This maximization can be obtained by controlling units based on rigorous and tunable thermodynamic compressor performance models.

Algorithms for calculating entropy, balancing interstage masses/pressures, calculating gas compressibilities, checking inertia-based rod loads, and determination of minimal degrees of reversal for crosshead pin safety can be complex, iterative, and number-crunching intensive procedures. Ignoring these checks may lead to safety-related issues. Replacing these checks with simplified, alternate methods will tend to reduce the available operating map, reduce compressor-driver utilization, or both.

Complex modeling of compressor performance is not ideally handled by most PLCs. However, complex modeling is easily handled by PCs, while safety and reliability are ideally handled by PLCs. Integrating the power of both systems into a single, yet accommodating solution can often provide the desired increases in flows and/or reduction in operating costs.

FREQUENTLY ASKED QUESTIONS

Q: Why not just code the more accurate and thorough performance predictions into the PLC?

A: This in fact could be done, but it would involve a lot of PLC programming. On a 6-throw, 2-stage, 20-load step compressor, you would have to independently model 12 ends (load and flow), 6 throws for net rod loads, 6 throws for rod reversals (with at least 36 crank angle positions for each throw), flow balancing for interstage pressure predictions, etc. for each of the 20 load steps. For PLCs that are not ideal for these intensive and often iterative calculations, the PC is favored.

Q: Does the PC-based software have to run on the station's HMI/SCADA PC?

A: No. The software can run on any local PC that can communicate to the HMI/SCADA PC via a network connection.

Q: Is using the active status of the PLC-to-HMI communications okay to validate the status of the PC-to-PLC communications?

A: No. The PLC and HMI/SCADA software may be communicating properly even if the PC running the PC-based software is down (power loss, operating system crash, software crash, network error, etc.).

Q: During test interruptions of the PC-to-PLC communications, the compressor sometimes changes load step. Why?

A: The PLC's load predictions and safety cutoffs are on the conservative side to keep its software coding minimal. This difference in load/cutoffs predicted for the active load step may tend to result in a load step (or more) change. This change is also an indication of how having the PC-based software handle the performance predictions is keeping your unit running at its ideal operation.

Q: Where can a person get a copy of the sample PLC control algorithms?

A: Sample pseudo code for single stage control is included in this paper (Appendix-A). Contact your vendor for your PC-based control software for 2-stage (and higher) code.

ACKNOWLEDGMENTS

Basic Systems, Inc., Derwent, OH
Caterpillar, Lafayette Testing Center, Lafayette, IN
Chungo Project, Talisman Energy, Alberta, Canada
Equitrans, Copley Compressor Station, WV
PG&E, McDonald Island Station, CA

REFERENCES

DMS System, Ariel Corporation, Mt. Vernon, OH
eRCM™ Software, ACI Services, Inc., Derwent, OH
HP-Guard, Windrock, Inc., Knoxville, TN
RECON ROM Sys., Dresser-Rand Co., Painted Post, NY
S7 PLCs, Siemens Energy & Automation, Chicago, IL
SIMATIC WinCC, Siemens Canada Limited, Calgary
Windows® 2000, Microsoft Corporation, Redmond, WA

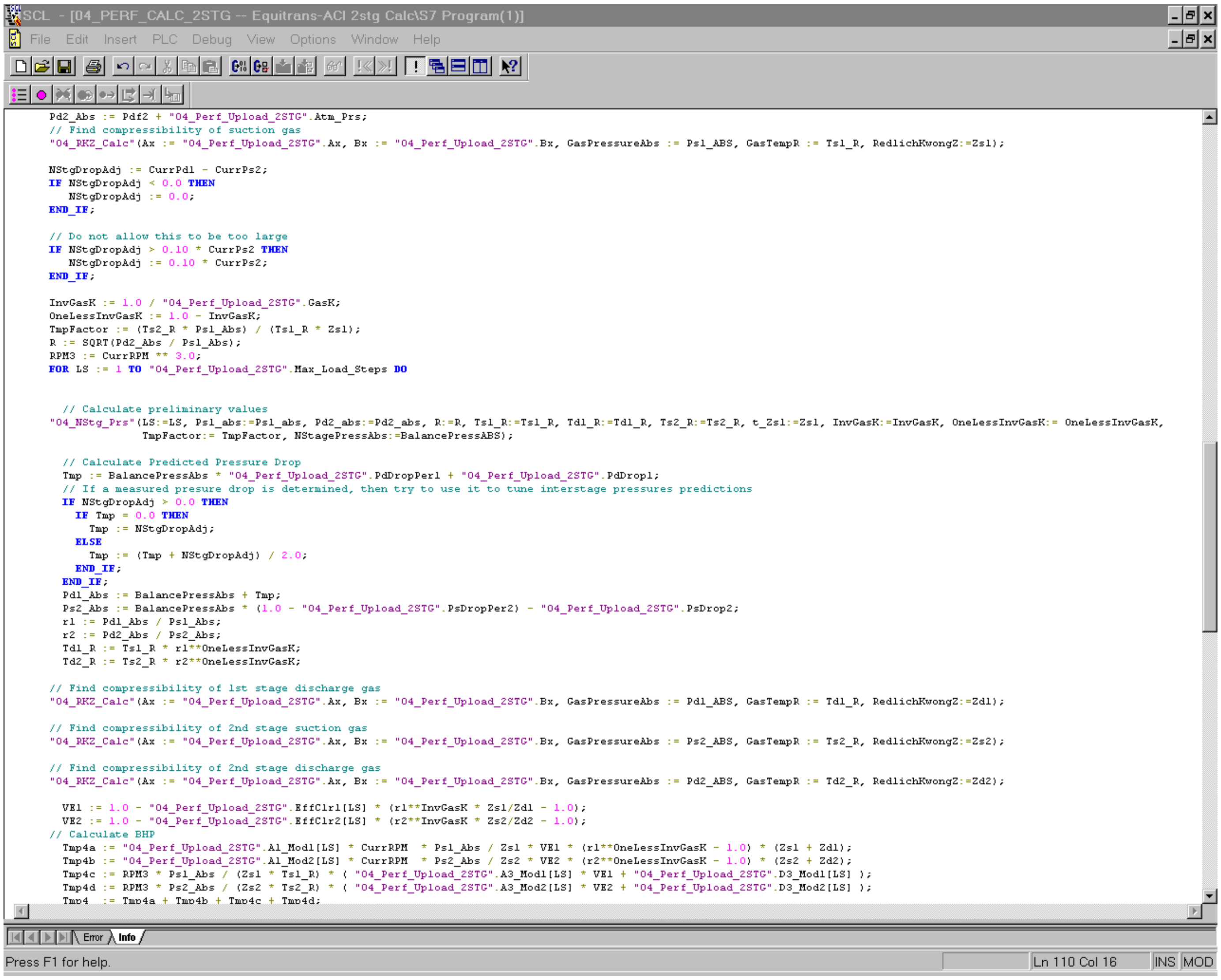
Appendix A

```
Single Stage PLC Control Logic (Pseudo Code):
========================================================
Sub CalculatePerformance()
' Determines BHPs and Flows for all load steps. Stores data in lookup table.
' If load step invalid (except for overload), then BHP and Flow = -1.
  IF IsPCConnectionAlive THEN
    ' PC Software has uploaded latest BHPs and Flows to PLC tables BHP_For_LS() and Flow_For_LS()
    ' PLC does not have to do any performance calculations.
  ELSE
    ' PLC does have to do simplified performance calculations that will likely over-predict consumed BHP.
    ' Convert to absolute pressures.
    PsABS = Psf1 + AtmPress
    PdABS = Pdf1 + AtmPress
    ' Convert to degrees Rankin.
    TsR = CurrTs + 459.67
    ' Get ratio of compression.
    Ratio = PdABS / PsABS
    ' Find compressibility of suction gas.
    Zs = RedlichKwongZ(PsABS, TsR)
    Tmp1 = Ratio^(1/GasK1)
    ' Determine adiabatic discharge temperature.
    TdR = TsR * Ratio/Tmp1
    IF TdR > MaxTdR1 THEN
      ALARM: Cylinder Temperatures Exceeding Allows Temperature Limits!
      ShutDownUnit
    END IF
    ' Find compressibility of discharge gas.
    Zd = RedlichKwongZ(PdABS, TdR)
    Tmp2 = CurrRPM * PsABS * (Ratio/Tmp1 - 1) * (1 + Zd/Zs)
    Tmp3 = CurrRPM^3 * PsABS / (TsR * Zs)
    ' Correct Pressure Ratio for Effects of Gas Compressibility for VE Calculations.
    Tmp1 = Tmp1 * Zs/Zd
    ' Cycle through all defined load steps and determine each step’s load.
    FOR LS = 1 TO NumberOfLoadSteps
      ' Calculate Volumetric Efficiency (no slippage in load VE’s)
      VE = 1 - EffClr1(LS) * (Tmp1 - 1)
      ' BHP = Adiabatic + ValveLoss + ParasiticLoss + AuxHP
      Tmp4 = Tmp2 * VE * A1Mod1(LS) + Tmp3 * (VE * A3Mod1(LS) + D3Mod1(LS)) + MaxAuxHP
      ' Check if this load step is safe at these conditions, based on low volumetric efficiencies.
      VE = 1 - ACIe1(LS) * (Tmp1 - 1)
      IF VE < MinVEe1 THEN Tmp4 = -1
      ' Check if this load step is safe at these conditions, based on crosshead pin failing to reverse.
      VE = 1 - ACIx1(LS) * (Tmp - 1)
      IF VE < MinVEx1 THEN Tmp4 = -1
      ' Check if Gas Flange Rod Loads are safe. This code only handles deactivation at Suction Pressure.
      IF RL13(LS) < 2 THEN
        TmpRLC = RL11*Pdf1 ' Cases 0 and 1
      ELSE
        TmpRLC = RL11*Psf1 ' Case 2
      END IF
      TmpRLC = TmpRLC - RL12*Psf1
      IF RL13(LS) = 1 THEN
        TmpRLT = RL12*Psf1 ' Case 1
      ELSE
        TmpRLT = RL12*Pdf1 ' Cases 0 and 2
      END IF
      TmpRLT = TmpRLT - RL11*Psf1
      IF TmpRLC > RLC1 THEN Tmp4 = -1
      IF TmpRLT > RLT1 THEN Tmp4 = -1
      BHP_For_LS(LS) = Tmp4
    NEXT LS
  END IF
End Sub
========================================================

========================================================
Function RedlichKwongZ(GasPressureAbs, GasTempR)
' This method is reasonable for most natural gases -- some restrictions exist.
' Use AGA-8 for process gases.
  ab = Ax * Ax / Bx
  b1 = Bx * GasPressureAbs / GasTempR
  a1 = b1 * ab / (GasTempR * SQRT(GasTempR))
  Q = (a1 - b1 - b1 * b1) * OneThird
  R = One27th + (a1 * b1 - Q) * 0.5
  a = Q - OneNinth
  d = a * a * a + R * R
  IF d > 0 THEN
      d1 = SQRT(d)
      a1 = ABS(R - d1) ^ OneThird
      b1 = ABS(R + d1) ^ OneThird
      IF R - d1 >= 0 THEN Zfact = a1 ELSE Zfact = -a1
      IF R + d1 >= 0 THEN Zfact = Zfact + b1 ELSE Zfact = Zfact - b1
      IF Zfact < 0  AND  a < 0 THEN Zfact = Zfact + SQRT(ABS(1 - 9 * Q))
      Zfact = Zfact + OneThird
      IF Zfact > 0.254 THEN RedlichKwongZ = Zfact ELSE RedlichKwongZ = 0.254
  ELSEIF d < 0 THEN
      IF r - d1 = 0 THEN
        a1 = Pi / 2
      ELSE
        IF D < 0 THEN
          a1 = ABS(ATN(SQRT(-D) / (r - d1)))
        ELSE
          IF r - d1 > 0 THEN a1 = 0 Else a1 = Pi
        END IF
      END IF
      RedlichKwongZ = 2 * SQRT(-a) * COS(a1 * OneThird) + OneThird
  ELSE ' d=0
      IF a >= 0 THEN
        RedlichKwongZ = 2 * ABS(R) ^ OneThird + OneThird
      ELSE
        RedlichKwongZ = 2 * SQRT(-R) + OneThird
      END IF
  END IF
End Function
========================================================


========================================================
' Main Code:
' This is the main control section. It decides when/if a load step change is required.

IF In_ESD_Mode THEN ShutdownUnit   ' May be triggered by code or by user.

' Initialize actions to NOT call for changes to load step.
Load = FALSE
Unload = FALSE

' Set items required if Stop Button pressed. This will lead to quickly unloading the unit.
' Only set timer if not already set, or if time before next load step check is too long.
IF InStopMode AND ( LS_Time <> FastUnloadTime OR NextLSCheckTimer > FastUnloadTime ) THEN
  LS_Time = 0
  SET NextLSCheckTimer TO LS_Time
  Unload = TRUE
END IF

' Get sensor data.
' If no real-time BHP or Flow available then CurrBHP = 0 and/or CurrFlow = 0.
READ CurrTs, CurrPs, CurrPd, CurrRPM, CurrBHP, CurrFlow, IsPCConnectionAlive

IF CurrTs < 40  OR  CurrTs > 120 THEN
  WARNING: Suction Temperature Sensor not working. Assuming 60 Deg F.
  CurrTs = 60
END IF
IF CurrPs < -15 OR  CurrPd < 0  OR  (CurrRPM < 0 OR  CurrRPM > 1800) THEN
  ALARM: Critical Sensor not working. Shutting Down.
  In_ESD_Mode = TRUE
END IF

' Take into account pressure drops to arrive at expected pressures at cylinder flanges.
Psf1 = CurrPs * (1 - PsDropPer1) - PsDrop1
Pdf1 = CurrPd * (1 + PdDropPer1) + PdDrop1

' We do not handle throttling here.
IF Psf1 > Pdf1 THEN
  ALARM: Throttling conditions!
  CurrPs = CurrPd
  ' Psf1 is not changed so rod loads will be correct.
END IF

' Derate driver power available if RPM changes. Also apply engine limits, if any, here.
MaxAllowedBHP = RatedBHP / RatedRPM * CurrRPM * TorqueAdj

' Calculate BHP/Flow for all load steps.
CalculatePerformance()

' If real-time horsepower is available, use it to fine tune predictions.
IF CurrBHP <= 0  OR  BHP_For_LS(CurrLS) = -1 THEN
  BHPRealTimeAdj = 1
ELSE
  BHPRealTimeAdj = BHP_For_LS(CurrLS) / CurrBHP
END IF

' Try to prevent cycling of hardware when the current unloading step is lost.
IF BHP_For_LS(CurrLS) = -1 THEN
  ' Let us unload for now and set PLC to immediately pick the next safe load step.
  LS_Time = 0
  SET NextLSCheckTimer TO LS_Time
  Unload = TRUE
  ' Return to most conservative setting.
  BHPRealTimeAdj = 1
  ' Set to slow mode to prevent hardware cycling around these current conditions.
  InPreventCyclingMode = TRUE
ELSE
  InPreventCyclingMode = FALSE
END IF

' Limit correction amounts. If adjustments are too large, then a warning/alarm may be desirable.
IF BHPRealTimeAdj < 0.92 THEN BHPRealTimeAdj = 0.92
IF BHPRealTimeAdj > 1.08 THEN BHPRealTimeAdj = 1.08
IF TorqueAdj < TorqueMinAdj THEN BHPRealTimeAdj = TorqueMinAdj
IF TorqueAdj > TorqueMaxAdj THEN BHPRealTimeAdj = TorqueMaxAdj

' TRUE if permissive is not an active permissive, or if it is active and it is calling for load.
LoadPs   = NOT IsActivePsPermissive   OR [ CurrPs   >= MaxPs   ]
LoadPd   = NOT IsActivePdPermissive   OR [ CurrPd   <= MinPd   ]
LoadBHP  = NOT IsActiveBHPPermissive  OR [ CurrBHP  <= MinBHP  ]
LoadFlow = NOT IsActiveFlowPermissive OR [ CurrFlow <= MinFlow ]

' TRUE only if permissive is active permissive and it is calling for less load.
UnloadPs   = IsActivePsPermissive   AND [ CurrPs   <= MinPs   ]
UnloadPd   = IsActivePdPermissive   AND [ CurrPd   >= MaxPd   ]
UnloadBHP  = IsActiveBHPPermissive  AND [ CurrBHP  >= MaxBHP OR CurrBHP > MaxAllowedBHP ]
UnloadFlow = IsActiveFlowPermissive AND [ CurrFlow >= MaxFlow ]

' Set load and unload statuses
Unload = [ UnloadPs OR  UnloadPd OR  UnloadBHP OR  UnloadFlow OR InStopMode ] OR      Unload
Load   = [ LoadPs   AND LoadPd   AND LoadBHP   AND LoadFlow   OR Load ]       AND NOT Unload

' The main decision block
IF (NOT In_ESD_Mode AND NOT Maintenance_Mode) AND NextLSCheckTimer <= 0 AND (Load OR Unload) THEN
  ' Determine where we are relative to unit's current maximum allowed horsepower.
  Adj = BHP_For_LS(CurrLS) / MaxAllowedBHP
  IF Adj > 1 THEN Adj = 1
  AdjLast = Adj   ' Remember last adjustment factor.
  Cnt = 0
  DO
    ' To prevent possibility of an infinite loop, we will limit our time in this loop to 16 passes.
    Cnt = Cnt + 1
    IF Cnt > 16 THEN EXIT LOOP
    ' If we need to jump real far, then let's start to normalize our correction factor for safety.
    IF Cnt =  6  OR  Cnt = 10 THEN BHPRealTimeAdj = (1 + BHPRealTimeAdj) / 2
    ' Adjust load goal up/down by 2% of allowed load (or of rated for some applications).
    IF Load   AND Adj < 0.985 THEN Adj = Adj + 0.02
    IF Unload AND Adj > 0.025 THEN Adj = Adj - 0.02
    IF Adj > 1.00 THEN Adj = 0.99
    IF Adj < 0.00 THEN Adj = 0.02

    ' Set DesiredBHP based on loading/unloading needs.
    DesiredBHP = Adj * MaxAllowedBHP * BHPRealTimeAdj
    ' Next loop finds the best load step, based on allowed load, to engage for the current operating point.
    ' Could be altered to consider Flow or BHP/MM instead of just plain load.
```

```
      LS_Found = -1
      BestLoad = -1
      FOR LS = 1 TO NumberOfLoadSteps
        IF BHP_For_LS(LS) <= DesiredBHP  AND  BHP_For_LS(LS) > BestLoad THEN
          BestLoad = BHP_For_LS(LS)
          LS_Found = LS
        END IF
      NEXT LS
      IF ABS(AdjLast - Adj) > 16 AND NOT (StartUp OR InStopMode) THEN
        WARNING! Excessively large change in load step required.
      END IF
      ' Exit main loop if we in fact found a new and valid load step.
    LOOP UNTIL (LS_Found <> CurrLS  AND LS_Found <> -1)

    ' If valid load step found, engage it.
    IF LS_Found <> CurrLS  AND LS_Found <> -1  AND  Cnt <= 16 THEN
      ' Force a hardware change.
      EngageHardwareFor LS_Found
      CurrLS = LS_Found
    ELSEIF (LS_Found<>-1 AND Cnt<=16)  OR  (Unload AND LS_Found = -1  AND _
            BHP_For_LS(CurrLS) <= MaxAllowedBHP * BHPRealTimeAdj) THEN
      ' Do nothing. Not meeting permissives, but then we can't satisfy them right now so stay where we are.
    ELSE
      ' Unit is in trouble. Set alarm and shut down then unit.
      ALARM: No valid load step found!
      In_ESD_Mode = TRUE
    END IF

    ' Determine amount of time to at least stay in the current load step.
    IF IsPCConnectionAlive THEN
      IF Load   AND NOT StartUp THEN
        IF InPreventCyclingMode THEN LS_Time = SlowLoadTime  ELSE  LS_Time = NormalLoadTime
      END IF
      IF Load   AND     StartUp    THEN LS_Time = FastLoadTime
      IF Unload AND NOT InStopMode THEN LS_Time = NormalUnloadTime
      IF Unload AND     InStopMode THEN LS_Time = FastUnloadTime
    ELSE
      ' React quicker if in blind mode.
      IF Load   THEN LS_Time = FastLoadTime
      IF Unload THEN LS_Time = FastUnloadTime
    END IF

    ' Set timer so that we do not change hardware again for a certain amount of time.
    SET NextLSCheckTimer TO LS_Time
  END IF
```

Appendix B

Example of SCL (Structured Control Language) Code for Two-Stage Model


SCL - [04_PERF_CALC_2STG -- Equitrans-ACI 2stg Calc\S7 Program(1)]
File Edit Insert PLC Debug View Options Window Help
Error
Info
Press F1 for help.
Ln 110 Col 16
INS
MOD

```
    Pd2_Abs := Pdf2 + "04_Perf_Upload_2STG".Atm_Prs;
    // Find compressibility of suction gas
    "04_RKZ_Calc"(Ax := "04_Perf_Upload_2STG".Ax, Bx := "04_Perf_Upload_2STG".Bx, GasPressureAbs := Ps1_ABS, GasTempR := Ts1_R, RedlichKwongZ:=Zs1);

    NStgDropAdj := CurrPd1 - CurrPs2;
    IF NStgDropAdj < 0.0 THEN
       NStgDropAdj := 0.0;
    END_IF;

    // Do not allow this to be too large
    IF NStgDropAdj > 0.10 * CurrPs2 THEN
       NStgDropAdj := 0.10 * CurrPs2;
    END_IF;

    InvGasK := 1.0 / "04_Perf_Upload_2STG".GasK;
    OneLessInvGasK := 1.0 - InvGasK;
    TmpFactor := (Ts2_R * Ps1_Abs) / (Ts1_R * Zs1);
    R := SQRT(Pd2_Abs / Ps1_Abs);
    RPM3 := CurrRPM ** 3.0;
    FOR LS := 1 TO "04_Perf_Upload_2STG".Max_Load_Steps DO


      // Calculate preliminary values
    "04_NStg_Prs"(LS:=LS, Ps1_abs:=Ps1_abs, Pd2_abs:=Pd2_abs, R:=R, Ts1_R:=Ts1_R, Td1_R:=Td1_R, Ts2_R:=Ts2_R, t_Zs1:=Zs1, InvGasK:=InvGasK, OneLessInvGasK:= OneLessInvGasK,
                  TmpFactor:= TmpFactor, NStagePressAbs:=BalancePressABS);

      // Calculate Predicted Pressure Drop
      Tmp := BalancePressAbs * "04_Perf_Upload_2STG".PdDropPer1 + "04_Perf_Upload_2STG".PdDrop1;
      // If a measured presure drop is determined, then try to use it to tune interstage pressures predictions
      IF NStgDropAdj > 0.0 THEN
        IF Tmp = 0.0 THEN
          Tmp := NStgDropAdj;
        ELSE
          Tmp := (Tmp + NStgDropAdj) / 2.0;
        END_IF;
      END_IF;
      Pd1_Abs := BalancePressAbs + Tmp;
      Ps2_Abs := BalancePressAbs * (1.0 - "04_Perf_Upload_2STG".PsDropPer2) - "04_Perf_Upload_2STG".PsDrop2;
      r1 := Pd1_Abs / Ps1_Abs;
      r2 := Pd2_Abs / Ps2_Abs;
      Td1_R := Ts1_R * r1**OneLessInvGasK;
      Td2_R := Ts2_R * r2**OneLessInvGasK;

    // Find compressibility of 1st stage discharge gas
    "04_RKZ_Calc"(Ax := "04_Perf_Upload_2STG".Ax, Bx := "04_Perf_Upload_2STG".Bx, GasPressureAbs := Pd1_ABS, GasTempR := Td1_R, RedlichKwongZ:=Zd1);

    // Find compressibility of 2nd stage suction gas
    "04_RKZ_Calc"(Ax := "04_Perf_Upload_2STG".Ax, Bx := "04_Perf_Upload_2STG".Bx, GasPressureAbs := Ps2_ABS, GasTempR := Ts2_R, RedlichKwongZ:=Zs2);

    // Find compressibility of 2nd stage discharge gas
    "04_RKZ_Calc"(Ax := "04_Perf_Upload_2STG".Ax, Bx := "04_Perf_Upload_2STG".Bx, GasPressureAbs := Pd2_ABS, GasTempR := Td2_R, RedlichKwongZ:=Zd2);

      VE1 := 1.0 - "04_Perf_Upload_2STG".EffClr1[LS] * (r1**InvGasK * Zs1/Zd1 - 1.0);
      VE2 := 1.0 - "04_Perf_Upload_2STG".EffClr2[LS] * (r2**InvGasK * Zs2/Zd2 - 1.0);
    // Calculate BHP
      Tmp4a := "04_Perf_Upload_2STG".A1_Mod1[LS] * CurrRPM  * Ps1_Abs / Zs1 * VE1 * (r1**OneLessInvGasK - 1.0) * (Zs1 + Zd1);
      Tmp4b := "04_Perf_Upload_2STG".A1_Mod2[LS] * CurrRPM  * Ps2_Abs / Zs2 * VE2 * (r2**OneLessInvGasK - 1.0) * (Zs2 + Zd2);
      Tmp4c := RPM3 * Ps1_Abs / (Zs1 * Ts1_R) * ( "04_Perf_Upload_2STG".A3_Mod1[LS] * VE1 + "04_Perf_Upload_2STG".D3_Mod1[LS] );
      Tmp4d := RPM3 * Ps2_Abs / (Zs2 * Ts2_R) * ( "04_Perf_Upload_2STG".A3_Mod2[LS] * VE2 + "04_Perf_Upload_2STG".D3_Mod2[LS] );
      Tmp4  := Tmp4a + Tmp4b + Tmp4c + Tmp4d;
```

Appendix C

```
Station Flow Control Pseudo Code
================================
    Input Current_Station_Flow from Flow Meter

    IF Within Flow_Setpoint Deadband THEN

      ConsideringAddingUnit   = FALSE
      ConsideringRemovingUnit = FALSE

    ELSEIF Wait_Timer = 0  AND  NOT CurrentlyChangingNumberOfUnits  THEN  ‘ Our waiting period is over!

      IF Current_Station_Flow < Flow_Setpoint THEN

        ConsideringRemovingUnit = FALSE
        FlowFlag = FlowFlag + 0.01
        IF FlowFlag > 1 THEN
          FlowFlag =1
          IF NOT ConsideringAddingUnit THEN
            ConsideringAddingUnit = TRUE
            Start ConsideringAddingUnitTimer        ‘ Typically 5 to 30 minutes.
          ENDIF
        ENDIF

      ELSE

        ConsideringAddingUnit = FALSE
        FlowFlag = FlowFlag - 0.01
        IF FlowFlag < 0 THEN
          FlowFlag = 0
            IF NOT ConsideringRemovingUnit THEN
              ConsideringRemovingUnit = TRUE
              Start ConsideringRemovingUnitTimer    ‘ Typically 5 to 30 minutes.
            ENDIF
        ENDIF

      ENDIF

      Send FlowFlag to all online Unit PLCs  ‘ Typically rescale to 70%..100% to become Unit_Desired_BHP%
      Start Wait_Timer to X seconds.   ‘ Typically 15 – 60 seconds based on user-desired reaction time.
    ENDIF


    IF ConsideringAddingUnit  AND  ConsideringAddingUnitTimer = 0  THEN    ‘ Need to add a unit persists!
      Bring next unit online, if one available (AND it does not add TOO much flow if used as a Swing Unit)
      CurrentlyChangingNumberOfUnits = TRUE
    ENDIF


    IF ConsideringRemovingUnit  AND  ConsideringRemovingUnitTimer = 0  THEN ‘ Need to stop a unit persists!
      Stop one of the units, except if only one left, then leave it on or stop it -- your call!
      CurrentlyChangingNumberOfUnits = TRUE
      If using Swing Unit philosophy, select which of the remaining units to become the new Swing Unit.
    ENDIF


    IF CurrentlyChangingNumberOfUnits THEN
      IF (Current unit is ready to be brought online (IE Oil Temp. Okay) )   OR   (it has stopped)  THEN
        CurrentlyChangingNumberOfUnits = FALSE
      ENDIF
    ENDIF
```