Electro-Hydraulic
Control Solutions

Quick Start Guide

DVC Family

The information in this publication is intended as a guide only, and HCT takes NO responsibility for usage and implementation in any user written application code structure.
HCT strongly suggests that the user attends one of the product training courses to ensure correct and full understanding of this product and its intended use.
Please contact HCT customer service to book one of the scheduled training dates or to discuss arranging a course specific to your company needs.

Thank you for using High Country Tek Inc. Products.

Table of Contents

Software Installation 3
System Requirements 3
Installation 3
Software Overview 3
DVC Expansion Modules 3
Graphical Displays 4
Creating an Application 4
Create machine function flowchart (i.e. Sequence of Operations) 4
Separate different functions into different logic sequences. 4
Use meaningful variable names. 5
Declare all variables in one location 5
Comment important information into the program 5
Programming the Application 6
Structuring the Application 6

Software Installation

System Requirements

Windows XP Professional or higher windows revision
230 Mb of hard disk space to support a complete system install
PC with Serial Port - RS232 or USB
DVC710 or DVC707 Master control module
DVC serial cable and a USB to RS232 converter (i.e. Dongle) if using a USB port[1]

Installation

To install the Intella™ development software, close all program applications. Insert the Intella™ Software Installation CD in the CD-ROM drive, and wait for the installer program to execute. The installers on screen prompts will guide you through the installation procedure. If the installation does not automatically begin, open a Windows explorer window and navigate to the CD-ROM Drive and run the file, "setup.exe" located in the root directory.

Software Overview

The DVC software package includes two applications, the Programming Tool and the Program Loader Monitor (PLM). These applications allow the user to configure, program, compile, load, and monitor applications in using DVC modules. The Programming Tool is used to configure the modules inputs and outputs and program the control logic for an application. The PLM is used to download user applications into the DVC710 or DVC707. It also performs real time monitoring of system inputs and outputs. The PLM also allows the user to modify some input and output configuration settings including EEMEM variables. For example, this is a convenient way to have the user interact with the system to change a maximum pressure setting or input a customer job number while the system is running.

DVC Expansion Modules

Expansion Modules are available or use in systems that require more inputs or outputs than the DVC710 or DVC707 provide alone. Also, additional DVC710s or DVC707s can be configured (each as a master controller) that have the ability to share information over the CAN Bus. It is important to note that if a system requires multiple modules of the same model, specific module names and Mac ID# must be unique. For instance, if the system uses (2) DVC750 modules, one could be named DVC750_1 with a Mac ID of 50 while the other could be named DVC750_2 with a Mac ID of 51.

DVC Expansion Modules available to date:

DVC722	40 Sinking Digital Inputs
DVC741	12 High Side Outputs for Bang-Bang Valves or LEDs
DVC750	3 Output Groups, 8 Digital, 3 Analog and 3 Universal I/Os similar to the DVC10
DVC61	4x20 character screen display with 12 display variables and 5 single pole double throw digital inputs
DVC Master to DVC Master	Provides the ability for multiple D710s or DVC707s to talk to each other

[1] To ensure Driver compatibility with the DVC Program Loader Monitor, HCT recommends the use of the 108-00119 USB to RS232 dongle available wherever HCT products are sold.

Refer to our expansion module section located in both the HCT Programmer's User's guide and on our website, http://hctcontrols.com/products/programmable/index.htm

Graphical Displays

HCT offers several Graphical displays that work well with DVC systems. The PV series graphical displays communicate over the J1939 bus and are easily used in a project. The programmer would treat the display as a node on the Bus by configuring and programming messages to interact with the display in the DVC application. Then program the display to receive these messages, act on them as desired and return information to the DVC modules as needed. The displays are programmed using the Power Vision development kit. HCT offers both sales and training of the Power Vision tools as well as assistance in project programming of the displays. More information is available by calling HCT Customer Service and on our websites display page, http://hctcontrols.com/products/hmi/hmi.htm.

Creating an Application

The Intella™ programming environment allows the programmer to be creative in their programming style. Everyone has their own style of writing code. Different programmers using the same tools and writing the same application will write two completely different versions of code. Even though they are writing in the same language, for each to read the others application would be akin to a European Spaniard trying to read Mexican Spanish. The words are basically the same but the format is entirely different. The following is a guideline of how to program DVC systems using the Intella™ Programming Tool Suite based the experience of HCT that includes writing applications and modifying debugging and troubleshooting other programmers' applications.

Create machine function flowchart (i.e. Sequence of Operations)

This exercise helps the development team to better understand the detailed function of the machine. Creating a Functional Flowchart will also stimulate thought on how the more complex functions of the application need to be performed. Using flowchart notation, the development team documents each separate function of the machine. The first draft of the flowchart for a project should contain a very simple outline of the machines functions. Subsequent revisions would expand these simple function outlines into more detailed operations. Insert as much information in this flowchart as possible including safety and error behaviors. Ensure that the finial flowchart contains all the IO that will be used, including scaling and normalization calculations required for the IO. When the flowchart is finished, a design review with the entire development team should take place to accept and adopt the flowchart as the programming specification. While creating a flowchart may take a considerable amount of effort, the information derived is invaluable when writing the application program.

Separate different functions into different logic sequences.

Time critical logic should be contained in the 'always' bubble (e.g. monitoring for high pressures, fire or other safety related functions). Unless declared as a private variable, all application variables are global and may be accessed from anywhere in the application. Therefore, a variable set in one logic sequence may be accessed in any other logic sequence. Because of this, different operational functions may be contained in separate logic sequences while sharing information without the need to rerun a particular logic statement to set a variable that was already set in another logic sequence. For example, a pump control's logic can be contained in a one logic sequence while the logic to control the track drive of a bulldozer can be contained in its own logic sequence while both operations may be monitoring the same pressure sensor or enable switches etc. This is one way to organize the application to make it easier to navigate or troubleshoot. Try to design the application to use as little overhead as possible from the start because as the program grows, it is difficult to go back and "clean" code that is already working to make room for more features.

Use meaningful variable names.

Using meaningful names that describe a variables function is one way of writing code that is easy to follow for both the original author and also for subsequent authors assigned to make changes to an application. Some company's even have policies concerning variable naming conventions. The limit on variable name size is 32 characters, no spaces are allowed.

Declare all variables in one location

Define all variables such as Timers, UINTs EEMEMs etc in one place and group them appropriately for ease of managing them later. These variables as well as program notes, version notes etc can be written in a bubble that has no transition to or from it. HCT recommends that you define variables in the Entry Code space and write version notes etc in the Repeat Code space. The variables will still be mapped when the application is compiled while there is no system time spent trying to execute application code where it is not needed. The Intella Programming tool allows for unlimited (not to exceed the Memory file constraint of 15Kb) declaration of UINT and 512 EEMEM variables for the DVC710 and 120 EEMEM variables for the DVC707. This can make some programming easier by allowing the programmer to use multiple variables in a way that is easier to read and follow. However, care should be taken to ensure that variables do not end up linked in a chain whereas changing a variables definition for one function does not adversely affect other functions or processes in the application. I recommend that Global variables be defined and used only to scale or normalize basic input data such as Oil Pressure, Engine Temperature or RPM etc. Use these variables anywhere needed without rescaling them for special functions or you risk losing resolution. When special scaling is needed for whatever reason, declare a new variable if needed (or simply execute the calculation where needed) then start from the input itself and scale as needed. EEMEM variables are mapped in the order that they are defined in the application code. Be careful not to change the order then load the application onto a machine with an older EEMEM map where operation could become unstable without updating the EEMEM values before operation.

Comment important information into the program

Write through comments, but rather than trying to comment every line of code with an explanation, write a heading or short note before different sections of code within a Bubble and save the line by line comments for a simple explanation of a calculation, decision etc. When changing an existing program, comment the new code with the version number and possibly a very short explanation of the change. In the Bubble that was used for defining variables, a more detailed explanation of the change can be recorded without adding too much "fluff" to actual application code space. Commenting the version number into changes allows the programmer to search for the string containing the version number and obtain a list of each place the code was touched for each version. Always delete old code that has been commented out as soon as it is decided that it will not be used. The programmer can always copy it from a backup copy of an earlier version if needed. Code that has been commented out does not compile and so doesn't take up program memory but it is confusing to read through and leaves subsequent programmers wondering if it should have been commented out or not requiring more time for them to make sense of things.

Programming the Application

When programming a DVC application, the following steps preformed in order will help ensure faster development with fewer rewrites;

1. Save and name the project file.
2. Name and configure all Inputs, Outputs and Function Curves for the system, add expansion modules as needed. (Reference the Program Specification / Flowchart)
3. Define all program and EEMEM variables that are required. (Reference the Program Specification / Flowchart)
4. Use the Program Specification / Flowchart as a guide in programming the applications functions.
5. Compile the program. Refer '[DVC Family System and Programming User Guide]'.
6. Load the compiled program files into the DVC710 / DVC707.
7. Run your system and debug your application.

Structuring the Application

When writing application code, try to write in a straightforward, simple, efficient manner. Conceder the two example statements below;

```
'Example 1
if (dig_1 = true) then
   if (dig_2 = true) then
      PWM_1.enable = true
   elseif (dig_3 = true) then
      PWM_1.enable = true
   else
      PWM_1.enable = false
   end if
else
   PWM_1.enable = false
end if

'Example 2
PWM_1.enable = (dig_1 AND (dig_2 OR dig_3))
```

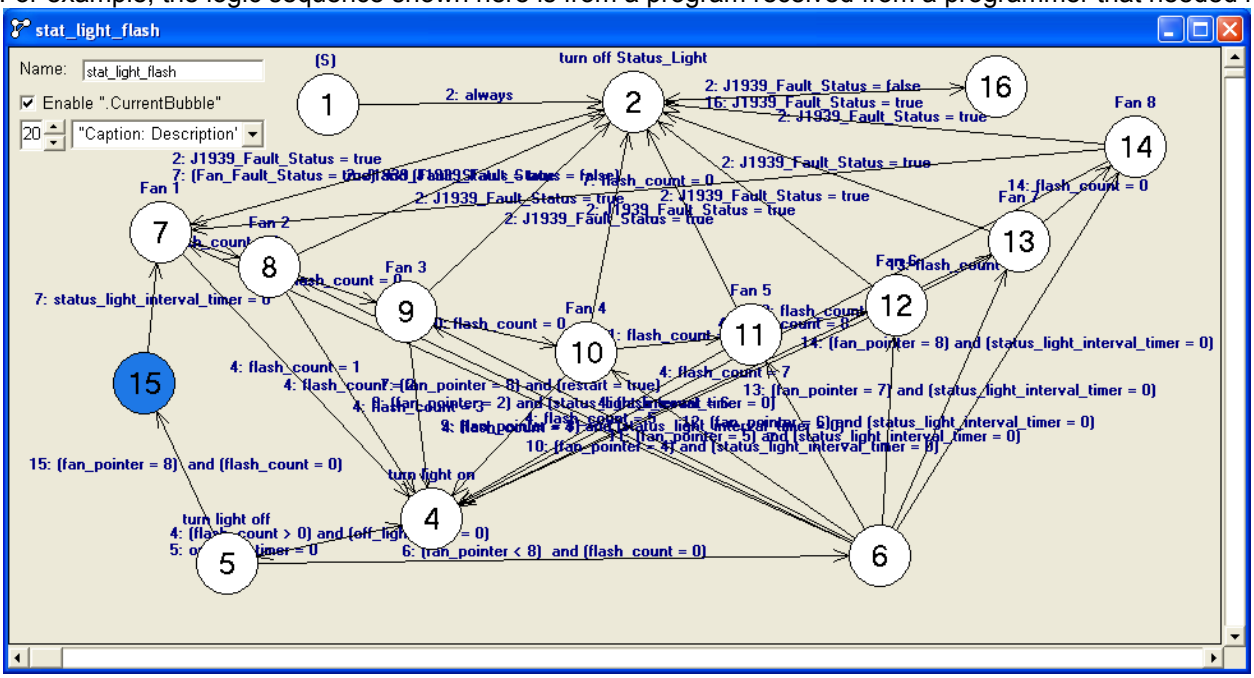
Both statements perform the exact same function; however example 2 is more efficient in that it is not only easier to read at a glance, but requires about 90% fewer lines subsequently using less program memory. An added bonus of writing code this way is that more information can be displayed within the programming window at one time allowing the programmer to scan more code without having to scroll back and forth inside the current window.

Programming, Sequencing and Execution

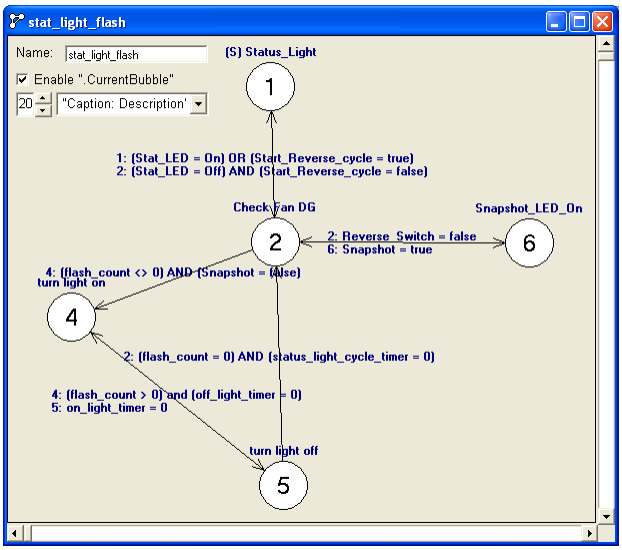
When laying out the application components, Logic Sequences and Groups etc., consider how the DVC executes code with respect to Logic Bubbles, Bubble Transitions, Logic Sequences, and Always Code as compared to how often variables and IO need to actually be updated to be efficient. Grouping Logic Sequences that do not contain speed sensitive code not only provides more execution cycles for the speed critical code in a given amount of time but also slows the update of less sensitive variables. This method can be used to dampen the way a display is updated or allow for some ramping of IO without actually having to actually write a ramp.

A good rule of thumb is to write a separate logic sequence for each major function. However, be careful of generating too many logic sequences with only one or two bubbles containing very few lines of code each. These can add up and each requires a certain amount of system and program memory. It is better to combine a few smaller functions' code in a single Bubble / Logic Sequence than waste resources trying to make a Logic Sequence for each and every different function. With the same respect, do not overcrowd a single Logic Sequence with too many bubbles either. Following and maintaining too many bubbles and transitions in one Logic

Sequence can be quite difficult especially when someone other than the original programmer is trying to troubleshoot or modify the application. Instead, limit a logic sequence to as few bubbles as needed to perform an entire function efficiently.
For example, the logic sequence shown here is from a program received from a programmer that needed help;



HCT engineering subsequently modified it as follows;



This modification is not only easier to understand at a glance but it also reduced the program memory used by a significant amount. In this new version, the average lines of code per bubble are only 24 (still easy to follow) while in the original version the average is only 6 lines of code per bubble. Using 30% more lines of code and 87% fewer Bubbles and Transitions, this version runs using less overhead and has allowed The customer to add several more features to the overall application while using less program memory than the original application.

The Always Bubble is executed before each Logic Sequence or typically once every 10mS. This code space should be exclusively reserved only for the most critical and time sensitive application code such as safety functions or functions that require the highest degree of resolution. Avoid inserting statements in the Always Bubble in order to “fix a bug” by overwriting something that was set in a Logic Sequence. Instead, spend the time to locate the actual issue and repair the root cause. Conflicting statements not only waste resources, they are very confusing for people other than the original programmer to understand or even the original programmer a few months down the road.

Virtual Displays

Virtual displays are excellent tools for development, however unless the end user will have a PC attached to the system on a regular basis, they should not be included in the production or released version of the application code. This is because the Virtual Display uses system resources whether or not it is being used. HCT recommends that the programmer maintain two versions of an application that has been published. One copy is the development version that contains the virtual display, test code and rough timing of J1939 messages etc. while the production version is exactly the same except compiled without the virtual display and test code.