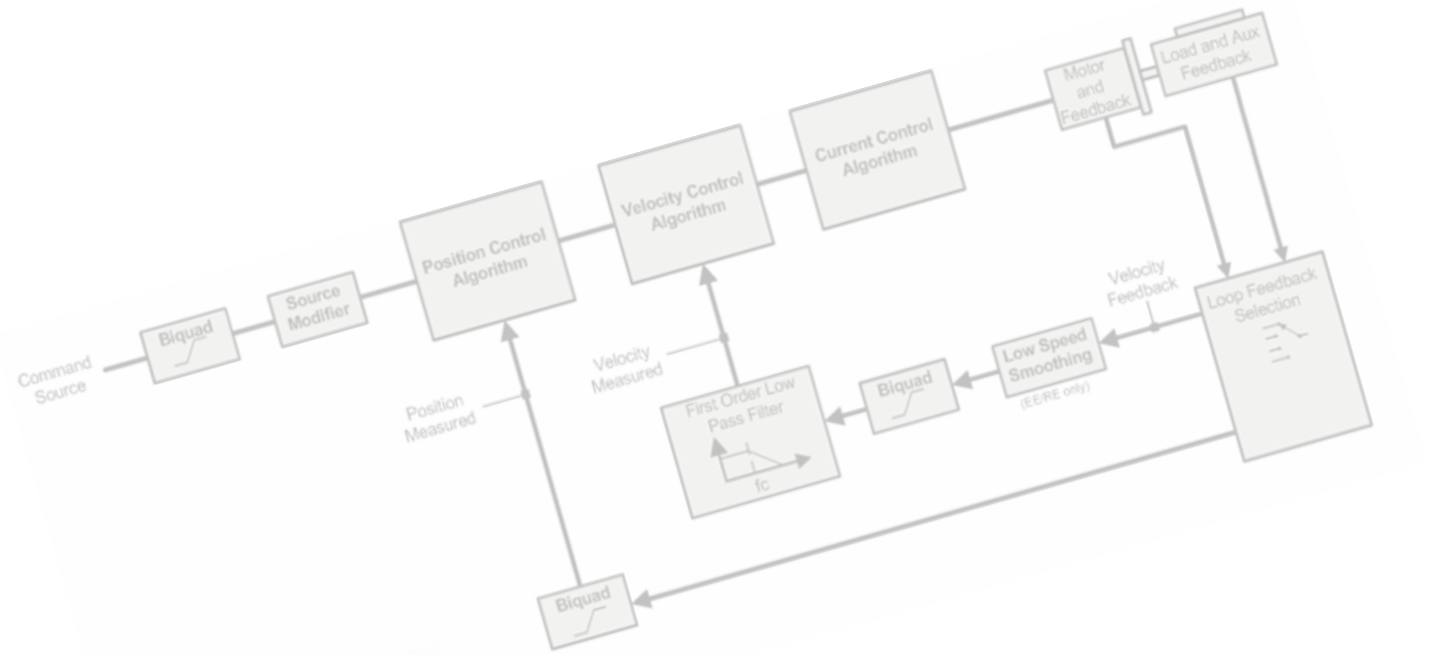
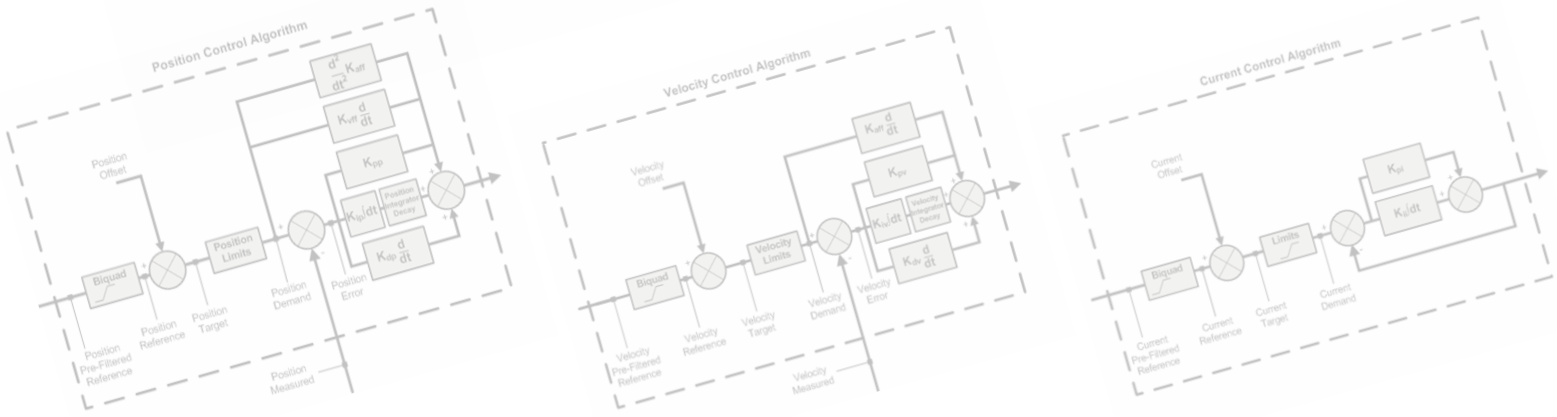
Visual Programming of Logic, Motion, and Robotics





Sándor Barta

ADVANCED MOTION CONTROLS

Overview

The art of programming consists of mentally translating a workflow into a sequential programming language. Most people have an intuitive understanding of workflows because they mirror their real world experience but relatively few find it easy to translate a workflow into something that could be programmed on a computer. One solution to this problem is visual programming, a programming environment where users draw information flows between entities (i.e. function blocks) that translate, source or sink information. This diagram becomes the program, with the internals of the system taking care of the details of translating this program into a form that can be executed on a suitable target platform.

This paper discusses how complex, platform independent control systems can be implemented by simply drawing diagrams that represent both the specification and the implementation logic.

The Function Block Diagram as a Visual Programming Language

In order to describe actions in a complex system, it becomes necessary to formalize block diagrams into a visual programming language. A Function Block Diagram (**FBD**) is a visual programming language defined in the IEC 61131-3 PLC (Programmable Logic Controller) standard. A FBD-based solution is made up of function blocks (**FB**) interconnected by directed lines. A FBD is a real computer programming language with strict rules for how diagrams are to be built.

At the top level the design is a block diagram, becoming a more detailed set of nested block diagrams at the lower levels, ending in block diagrams that capture all the details required for generating the solution. The resulting set of diagrams resembles the schematics used in the electrical engineering world with the individual FBs connected by named paths.

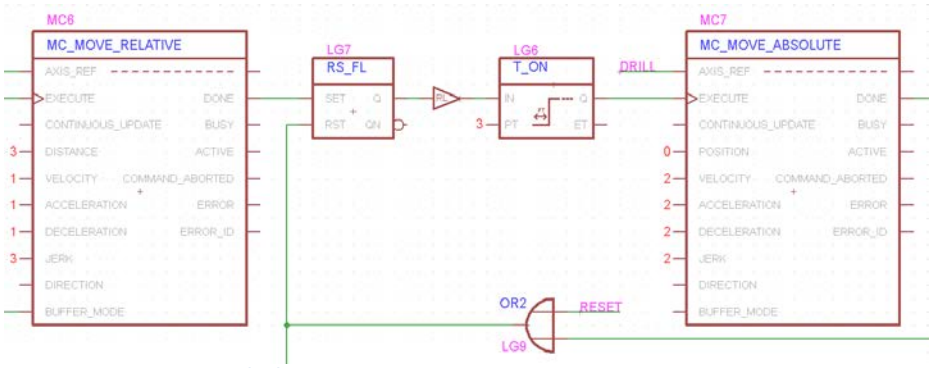


Figure 1 – Function Block Diagram

Function blocks can be either one of the basic function blocks ('primitives') provided by the system or a user-defined function block encapsulating a lower level function block diagram that acts like a primitive ("User-Defined Function Blocks"). User-Defined Function Blocks (**UDFB**) are an essential structuring tool that allows diagram complexity to be managed by breaking what would be a large, complex, diagram into a nested set of smaller diagrams that can be coded, built and tested independently. A UDFB may itself contain UDFBs, the only obvious limitation being on nesting depth and a prohibition on recursion where a block may not reference itself either directly or indirectly.

What is in a Function Block?

FBs are the basic building blocks of the application. FBs represent mathematical, logical or physical operations that act on data being presented to their inputs and in turn generate output data. Some function blocks interact with the physical world performing functions such as moving a drive or reading information from an interface. These Function Blocks free the control engineer to focus on engineering the application rather than having to focus on the actual mechanics of programming each block. The process of generating a system is near to the mind-set of an industrial system designer who is familiar with connecting physical devices together in different ways to provide a particular system solution. Applications are created from reusable components that are already understood and proven.

Function Blocks may include static properties which are specified with a particular block to configure it for its intended use. An example of these properties are communications parameters; the function block itself would handle opening and closing a connection, message encapsulation and flow with the properties specifying the communications target, protocol specifics and the like.

User Derived Function Blocks as Libraries

UDFBs are not only a useful structuring tool but also a way of generating application specific libraries that encourage code reuse within an organization. Since it is also possible to embed actual program code in UDFBs these libraries may be used to manage Intellectual Property such as proprietary algorithms that cannot easily or safely be described using standard function blocks.

How Should FBDs be Structured?

For simple applications the designer intuitively wires FBs to get the desired functionality just like someone using a breadboard to prototype a simple circuit. A serious application tends to be more complex so requires a more structured approach to breaking the problem down before actually trying to implement it. These approaches are analogous to a traditional programming technique called *"Top Down With Prototyping"* where the outline of the problem is described with top level functional components (Function Blocks in this case) while each component is decomposed into smaller components that are generated and tested separately. Here UDFBs are essential because during this decomposition phase it is likely that many of the top level components will be functional dummies, components that simulate the expected behavior of the subsystem without actually being those components. This allows the designer to test overall system logic and the user interfaces that will be used to control and monitor the application. As the solution becomes clearer and so the designer gains more confidence in it then they gradually substitute 'live' blocks for the dummies and the solution emerges from the prototype.

Two important structuring tools for generating these solutions are *Flowcharts* and *State Machines*.

The Flowchart

A flowchart is a visual means to show the desired sequence of some activities. It shows the sequence depending on conditions. In control applications however activities may also depend on past conditions (e.g. a timer expired). Unfortunately there is no concept of a state in flowchart. Sure, flags or variables can be introduced but they in effect amounts to coded sates. If you try to put states into a flowchart then you are likely to end up with an asynchronous machine which tend to be unpredictable, hence the glitches and lockup situations.

The State Machine

A more precise and transparent way to describe a control application is the state machine, represented by a state diagram.

A state completely defines the current condition of a machine and represents information about the past too. A state machine serves to divide an application into a series of states, each of which has a set of acceptable input. The state specific logic is programmed in state UDFBs, but the central logic of the machine is handled by the state machine UDFB. Once in a given state, the user need only concern themselves with valid inputs. If the application has to consider all possible inputs at all times, the complexity of a real life application would be very difficult to deal with. Fortunately applications naturally deal with certain events at certain times.

Decomposing a problem into one or more state machines is an important part of designing a solution to a problem. The process usually starts by enumerating all

possible states into a table, identifying transitions between states, the outputs from those states and the inputs that cause transitions to subsequent states. Once again, if the user resists the temptation to go into too much detail at first, confining functions to easily identifiable, more general, operations then the overall complexity of the solution can be managed until the user is comfortable that the solution does indeed match the problem being solved. With a state machine there's a one-to-one correspondence between what the machine will do and the way the program is structured. There is almost no interpretation involved to get from concept to programming.

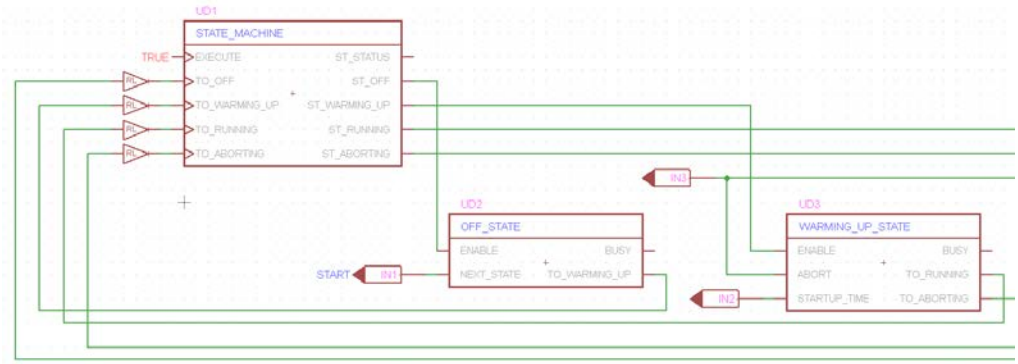


Figure 2 – FB Implementation of a State Machine

Implementing Function Block Based Solutions

Once the problem has been expressed as a set of interconnected function blocks then it is the task of the underlying system to translate this solution into a form that can be implemented on a target platform.

Click&Move®, a forward-looking and unique programming system

The *Click&Move* (C&M) system translates the solution into a standard programming language, C++. This code can then be compiled by an appropriate target platform compiler and combined with a combination of standard and C&M specific libraries to generate an executable that will run on one of a variety of platforms. This separation of program and platform allows the user to focus on solving the problems of the application, followed by selecting the best, most cost effective run time platform for a specific application.

Since FBs represent pure functionality and not the physical hardware they can be reusable and platform independent. Specific FBs are linked to physical hardware using Reference Structures in a process

One side effect of translating Function Blocks to C++ is that it is also possible to build Function Blocks that encapsulate user source code. The C&M system enables the user to create templates for this type of block.

Unlike other programming system C&M is C++ based and purely visual, no need to declare variables, no MAIN program to write in a textual language. It's only drawing FBDs.

The Function Block Libraries

The C&M system comes with a large set of prebuilt Function Block libraries that implement logic, math, motion, communications and storage functions.

Logic Library

This is based on the IEC61131-3 PLC standard for FBs.

Motion and Robotics Libraries

These are based on the www.PLCOpen.org *"Function Blocks for Motion Control"* standard which is emerging as the de-facto international standard for motion control.

The PLCopen Standard for Motion Control

The standardization is done by defining libraries of reusable components. PLCopen FBs are defined for point to point and coordinated motion, gearing, camming, and event triggered motion.

Traditionally, path oriented movements are typically programmed with manufacturer specific motion control or robot oriented programming languages. (For example, manufacturer specific flavors of the industry standard "G-code" are used in the CNC world.)

The PLCopen standard for coordinated motion transforms the functionality of dedicated motion controllers, robot controllers (and G-code) to platform independent reusable FB libraries.

The IDE (Integrated Development Environment)

The process of developing, building and testing a solution involves a number of steps, few of which are of interest to an end user. An integrated development environment not only automates the build steps but adds applications that edit, organize, archive and debug the application.

The right IDE streamlines the task of programming and program organization. A flexible FBD editor allows you to easily arrange your FBs in a clean program flow (including the state machine). FBD nesting capabilities and schematic editor type features such as, buses to bundle wires, named labels etc. are also very helpful.

A powerful feature of the C&M system is the visual debugging.

Debugging can be performed by using single and state steps, active net and FB highlighting, live readout labels and break points.

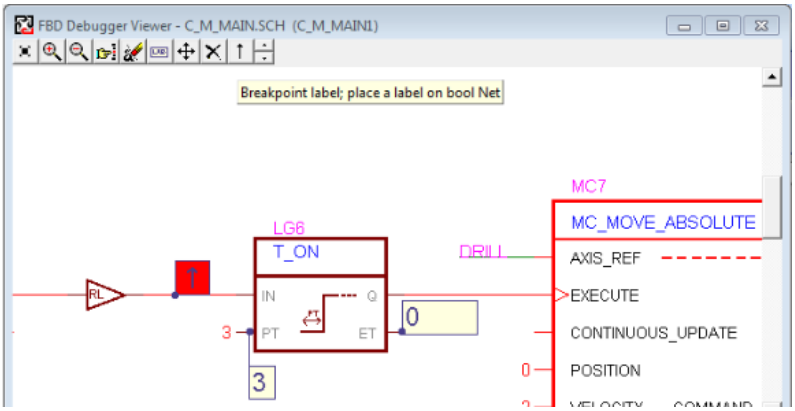


Figure 3 – Visual Debugging

Distributed Applications and Network Flexibility

The design of a solution may involve more than one platform. For example the HMI (Human-Machine Interface) and motion coordination software may run on a PC under Windows® OS while high-speed capturing and signal processing runs on a dedicated controller running Real-time Linux with the two communicating using a network connection. The Function Block programming approach does not need the entire applications solution to be on a single platform. Different parts of the solution can exchange data over a network or similar interface using appropriate communications FBs that are provided with the C&M system.

The C&M FB portfolio includes support for common Fieldbuses: RS232, USB, CAN, Ethernet (TCP/IP, UDP/IP), EtherCAT® and Powerlink.

The Advantages of Applying Open Standards

By employing state of the art technology, open architecture and adherence to international industry standards for communications, motion control, operating systems and programming languages, automation solutions merge the best of both the IT and the industrial automation worlds. Compliance with open standards facilitates full integration in a heterogeneous, multi-vendor control environment. Relying on open platforms will also provide quick access to evolving high performance technologies while protecting software investment.

Built-in Integrated C&M-HMI

The complete integration of the HMI (Human Machine Interface) permits features such as real time animation, data array access and powerful data acquisition/visualization features which are not available when working with a third party visualization tool.

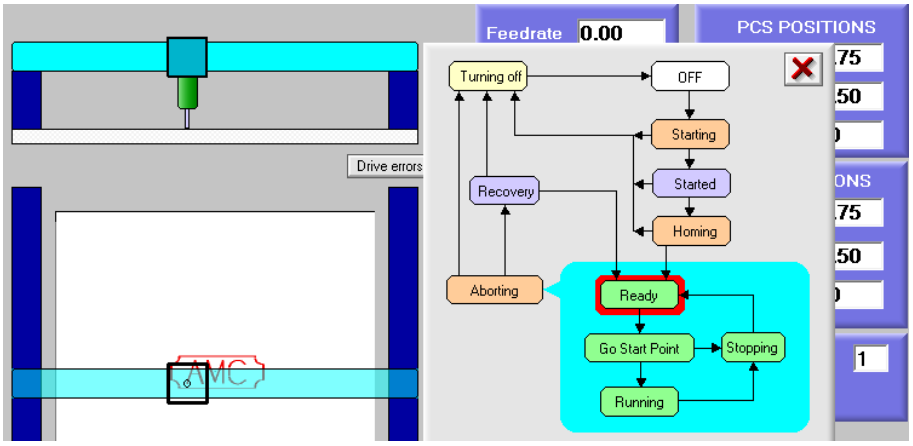


Figure 4 – HMI Screen (Showing a State Diagram)

Conclusions

In summary, a set of properly structured FBDs, managed with the right IDE, can be used to create complex logic, motion and robotics application by simply drawing block diagrams and compiling them into executable code for the selected target platform.

About the author – Sándor Barta is a founder and president of *ADVANCED* Motion Controls. He has an MS in Electrical Engineering / Control Systems. Contributions for the article also come from Martin Usher, firmware engineer at *ADVANCED* Motion Controls. Visit www.a-m-c.com for more information.