Thoughts from Old Man

I originally wrote the Chip application for myself using different hardware than the Kflop. For various reasons I decided to switch to Kflop. The Dynomotion people were interested in my code so I gave it to them. They can do anything they want with it so note that any documentation supplied by me refers to the original code. This may have been changed by Dynomotion. Hopefully they will keep the original and make it available so my documentation matches the original application, so people evaluating or just starting out can have accurate documentation.

The following is a general philosophy of my software development and machine tool experience quirks. I think it helps to know where someone is coming from when trying to figure out why they did the things they did.

My experience is based on over 50 years of computer hardware, software and machine tool experience. Hence the name ‘Old Man’, a moniker given to me by daughter.

I worked for the first machine tool company that put a computer on a machine tool (it was a DEC PDP8 mini-computer). Before that, machine tool control was called NC (numeric control). Now it is called CNC. They hired me to design the next generation computer. So in a sense, I helped put the C in CNC. It was one of the fastest computers of the time. They made thousands of them.

OK, first things first, don’t refer to me as ‘The Old Man’. That was my father, have some respect for the dead. I am “Old Man’.

**Glossary of terms :**

**Chip :** This is the name of an application that controls a machine tool.

**Developer :** Someone who changes the Chip code.

**User or Operator :** Someone who operates a machine that is controlled by Chip.

**Visual Studio :** Microsoft compiler used to write Chip.

**Visual Basic :** Language used, one of many that Visual Studio supports.

**Class:** Visual Studio module of code.

**Controls:** Visual Studio controls like buttons and text boxes.

**Chip Controls :** Controls developed by me. Similar to buttons and text boxes.

**Control Properties:** Are like variable that can be set or read. (like Text, Background color, Font etc.). These affect how the control looks and acts.

**Chip Control Handlers:** These routines are written by the developer to handle events like button clicks, or text editing. Every Chip control has a property called ‘Handler’. The name of the handler (software routine) is put here. When a Chip Button is clicked, for example the handler for that button will be run.

**Kflop C Program:** Not to be confused with Chip software. These are C programs that are downloaded into the Kflop and run. These are typically routines that need to have quick real time response, or routines that control Kflop hardware like spindle speed. Some examples are homing and probing routines.

When I changed to Kflop hardware, I revamped my original Chip application to be easier to adapt by developers. For normal changes (ie. adding buttons or text boxes), only the main form and one class needs to be changed. The main form is changed using the Visual Studio forms designer. Controls can be added, deleted, sized, and moved around. Control properties can also be changed. It should be a rare thing to have to change anything other than the main form and the one class (control handler routines).

The control handler routines do the appropriate functions when a control event occurs, like the mouse being clicked on a control. These handlers were designed to handle several events including initialization, updating along with mouse and other events. This was done this way to keep all the code for a particular function in one place, where it is easily viewed and modified. This keeps code from being scattered around in several routines (like one for initialization, several for handling events, one for updating information etc.). In addition several controls can have the same handler, so like functions like jogging for all axes can be handled by the same routine.

The code is written in Microsoft Visual Studio. This can be downloaded for free if your company is small, or you are an individual developer.

The language used is Visual Basic. C programmers will not like this, but Visual Basic is a higher level language than C. C programmers should not have any trouble writing code in Basic.

There will be endless arguments about what is the best programming language. I know several (around 25), most long gone that you would not recognize (Like Fortran, Cobal, Snowball etc.). I have done a lot of both C and Basic programming. I prefer C for writing code for micro-processors and Basic for PC programming.

I taught languages at the Milwaukee School of Engineering, C, Pascal and Basic for a couple of years. Students picked up Basic easier than Pascal, and Pascal much easier than C. C has some nasty concepts like pointers that are hard to understand and use by novices, and they lead to some really rough debugging.

This software may well be used by someone that has machine tool experience, but not programming experience, so Basic would benefit this person. If a C programmer cannot quickly learn basic, then in my opinion is they should not be writing software.

If you want to use this software as is. No changes need be made, but I you want to modify it, there is no way around it. You must learn to write software, but there is lots of documentation to help you learn. Don’t be afraid. Jump in. A good way to start is to look at the code in the Chip application. The class “Class\_Display\_Handlers” are the routines that are called by Chip controls and are the heart of the control functions.

One concept that must be understood early is that Chip relies on controls (like buttons and text boxes) that I developed. These custom controls are similar to the standard controls that come with Visual Basic, and have many of the same functions and characteristics, but are extended to provide more functionality.

The reason I chose to develop these controls is that they encapsulate functions that would be difficult for the novice programmer to handle. The two primary Chip controls are the Chip\_Button and the Chip\_Textbox. These are the two controls that a developer might want to add. There are other Chip controls, but these are much more complex, and are already in the original application. The following is a brief description of them:

**Chip\_G\_Code:** Displays the currently loaded G\_Code program. The program can be loaded, edited, saved with this control. Functions like single stepping and starting from a line are also handled by this control.

**Chip\_G\_Drawing:** 3D display of the cutter paths, and cutting progress. Drawing can be panned, zoomed and rotated.

**Chip\_Trace:** Debugging tool used to log messages from Kflop and control handlers. You can log messages in your Chip Control handlers, or Kflop C programs, so you can look back and see what happened.

**Chip\_C\_Interface:** This is primarily a development and debugging tool for Kflop C programs. Chip interfaces to Kflop C programs through Kflop User Data, which are locations in the Kflop that can be read and written to by the PC program. This custom control (Chip\_C\_Interface) shows various sets of data that can be downloaded, displays the actual contents of Kflop User Data locations, and can download, run or kill a Kflop C program. Combined with the Chip\_Trace control it is very useful for developing and debugging Kflop C programs.

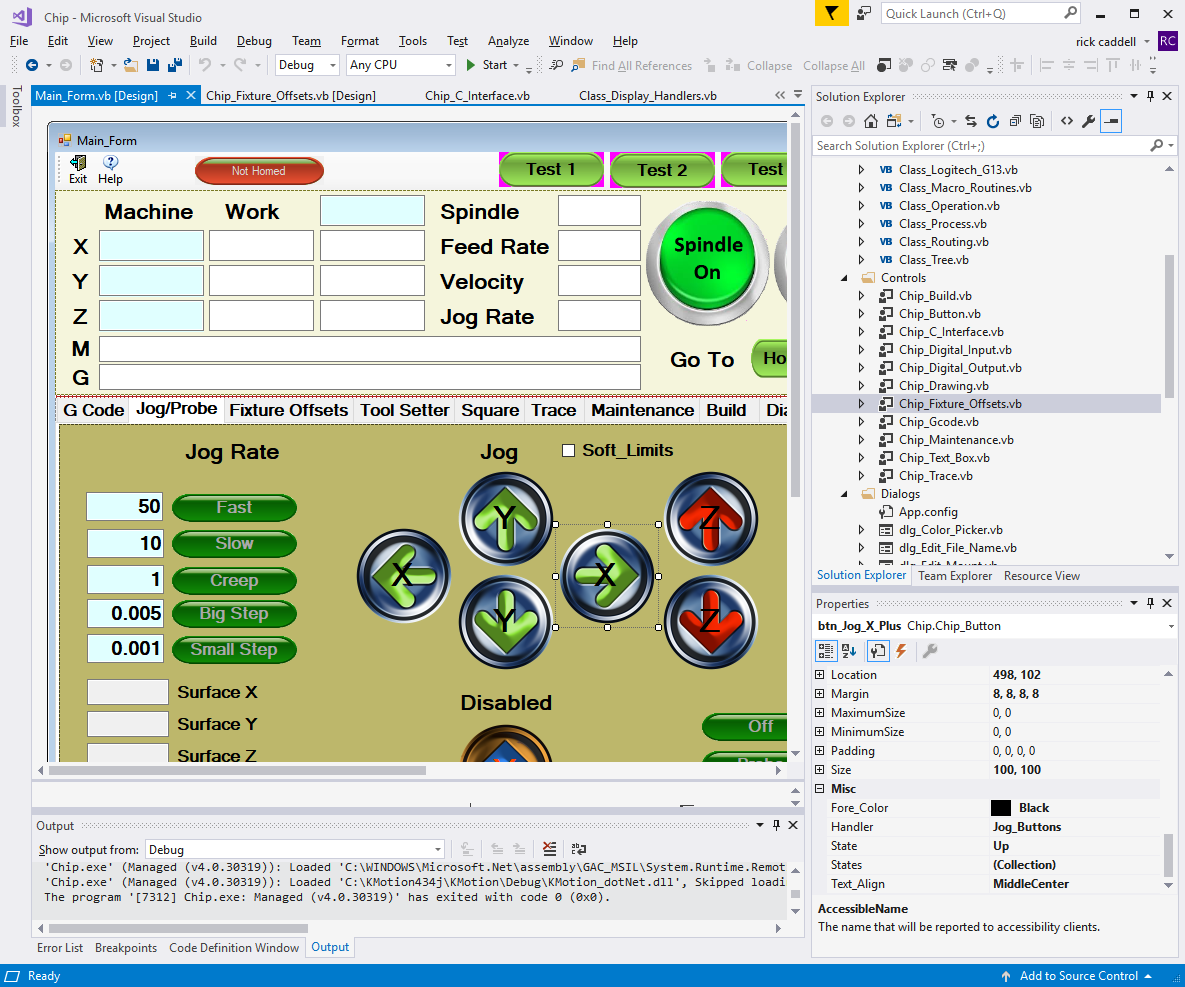
**Chip\_Digital\_Input & Chip Digital Output:** These are controls that are used to setup, control and monitor Kflop digital inputs and outputs.

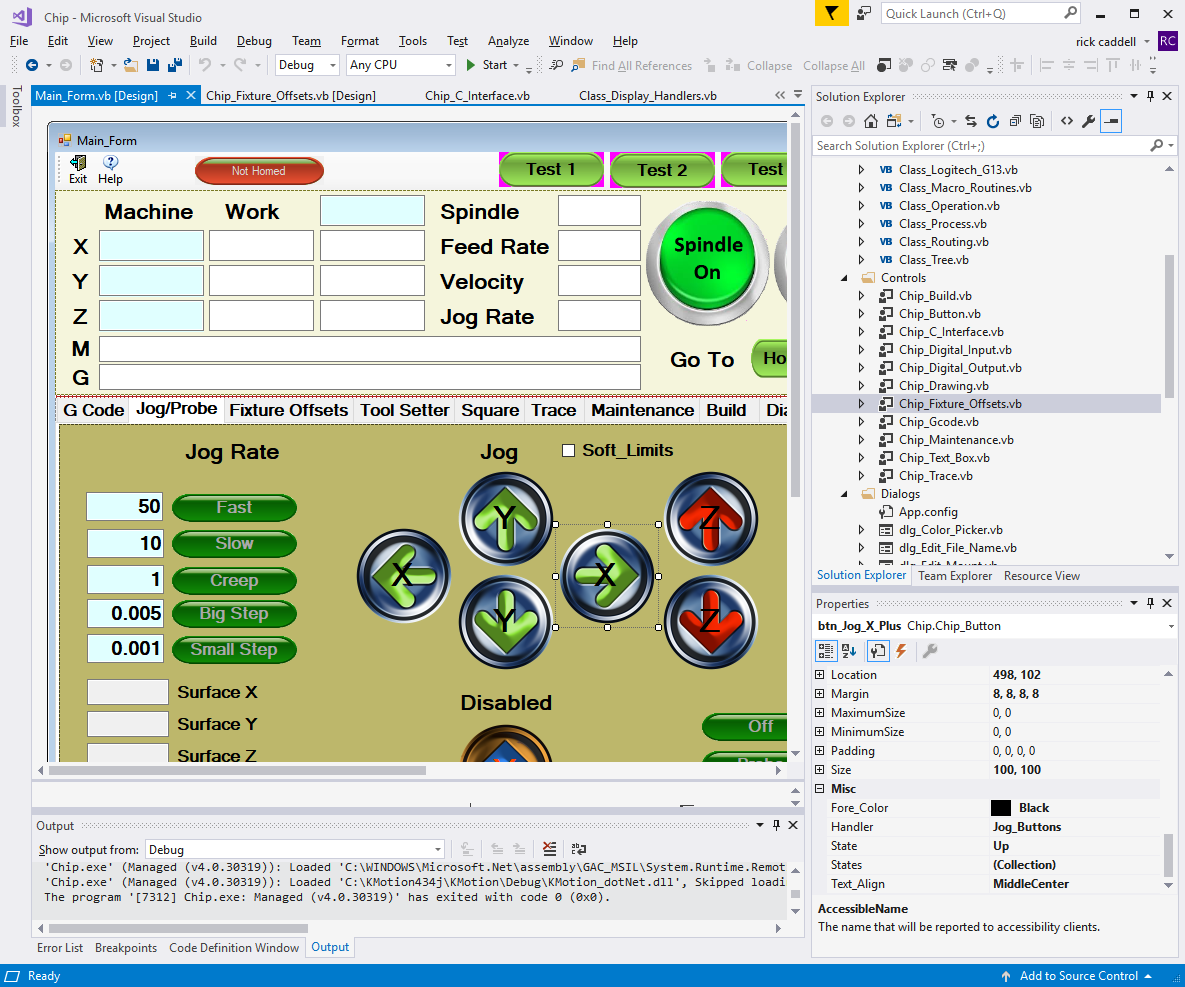
**Chip\_Fixture\_Offsets:** Offsets are often confusing and hard to implement. This control helps with managing and applying offsets.

**Chip\_Build:** This control has high level functions for keeping track of CAD drawings, CAM sessions, G\_Code programs, Routings and Operations. This functionality may not be for everyone, but for me I would not be without it. Even a simple product or project can have many CAD drawings, CAM sessions, G\_Code programs and other documentation to keep track of. This control helps with that. In addition Routings, and Operations can be developed to help run the correct G\_Code program, with the right fixture, the right offsets, the correct tooling and spindle speed. A product may have several routings, maybe one for each part in the product. Operations within the route are G\_Code programs that are to be run, including fixture, offsets, tooling and instructions to the operator. This control also handles things like multiple parts on a fixture, what G\_Code programs to run, and be able to setup for partially loaded fixtures. This control does not have to be used if you want to manually load and run the G\_Code using the Chip\_G\_Code control, but if you have several parts you mill, you may want to look into it.

**Chip\_Maintenance**: This control can be set up to provide warnings for preventive maintenance. Calendar elapsed time, or machine run time can be used to schedule maintenance.

Shown below is an example of some Chip\_Buttons and Chip\_Textboxes Shown in the Visual Studio forms designer. The X Plus jog button is currently selected (green button with arrow pointing to the right). The properties are shown in the lower left pane. The properties that are not standard are shown under ‘Misc’. These are the custom properties I added.



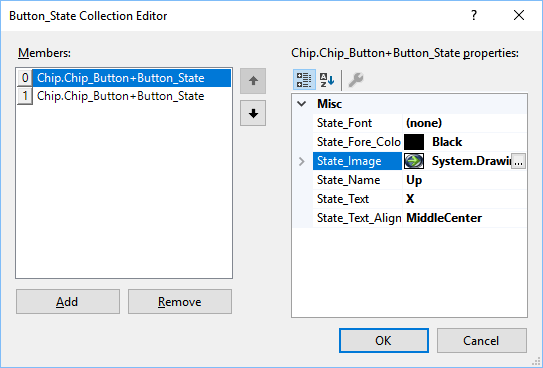


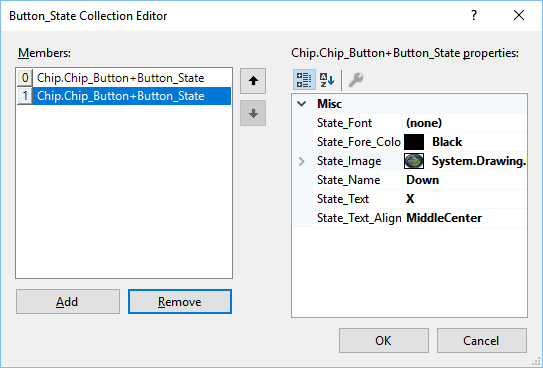
In addition to all the other standard properties, the Chip\_Button has Fore\_Color, Handler (routine to be called when button is pressed or let up), the State, a collection of States, and the Text\_Alignment. When the mouse button goes down or up when over this control the handler ‘Jog\_Buttons’ gets called and is passed the control, and the event type (either mouse button down, or mouse button up).

Based on the control and event type, the routine does the jog function. All of the jog buttons use the same handler routine. Grouping buttons with similar functions keeps all the code for those functions in one place, so it is easier to write and debug.

Clicking on the States collection will reveal the States and their properties.

In this case there are two states, ‘Up’ and ‘Down’.





The State is set by the Handler. So when the state is set to ‘Up’ the State\_Font, State\_Forecolor, State\_Image, State\_Text and State\_Text\_Align are set to the values defined in the ‘Up’ state. Same for the ‘Down’ state. In this manner what the button looks like can change depending on the state.

For example, when X Plus Jog Button is pressed with the mouse, the ‘Jog\_Buttons’ handler is called. The handler ‘Jog\_Buttons’, handles all jog buttons, but can determine which button was pressed, and by what event (mouse down or mouse up) by the Handler calling parameters.

The handler then sets the state to ‘Down’ causing the appearance of the button to change. (see second picture below). When the mouse button is let up, the handler sets the state to ‘Up’ and the button appears as in the first picture.

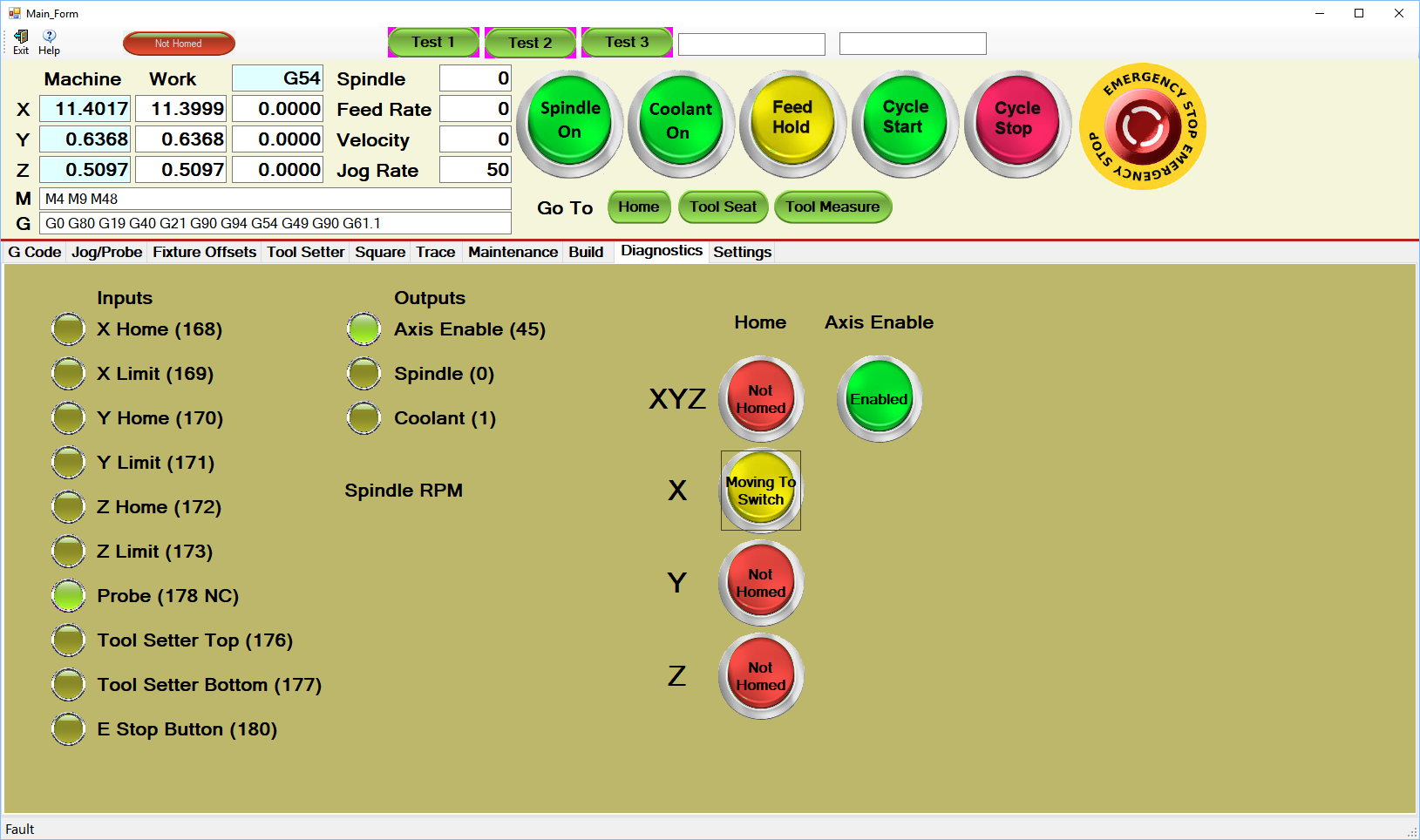
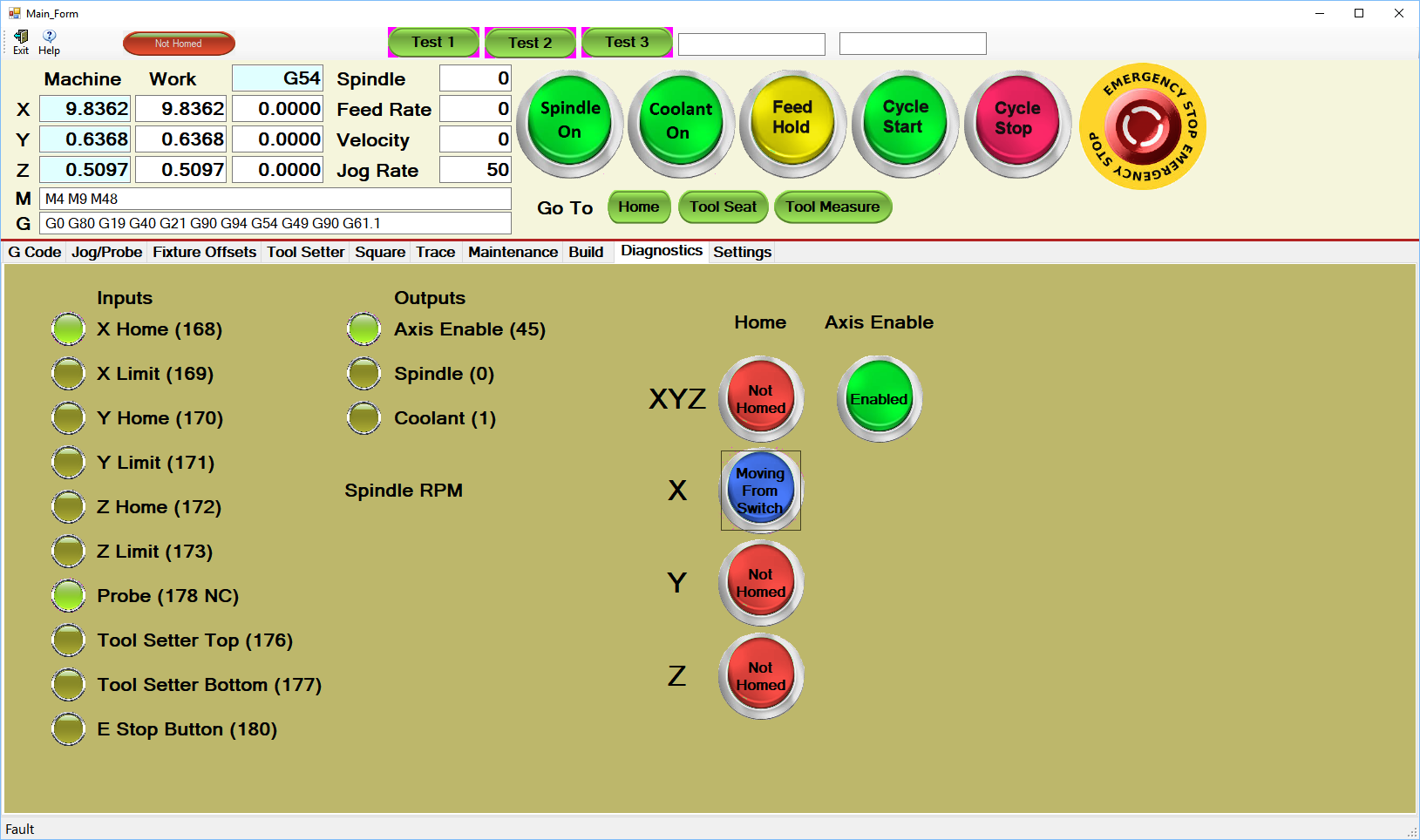




The Jog Rate buttons are like radio buttons. When one is clicked, it goes on and the others go off. Their states are ‘On’ and ‘Off’.

The Jog buttons act as momentary contact buttons and the Jog Rate buttons are modal, or change state when the button is clicked, but keep the same state until they (or other radio buttons in the same group) are clicked.

State names are arbitrary, you may call a state anything. The pictures below shows the Home X button which has four states : ‘Not Homed’, ‘Moving To Switch’, ‘Moving From Switch’ and ‘Homed’.

The handler changes the states to show the progress of the homing process. Showing what state a process is in is very useful for detecting and fixing problems.

If you have a several step process and you cannot easily tell that the states are changing properly, it will be tough to realize that something is wrong, and tough to find the problem once you do.

Old Man’s philosophy on indicators. Indicators (states of buttons, or contents of a text box) should be updated from information about the state of the actual hardware, not what is commanded.

For example. Suppose you have a button to control the master enable or disable for the axes motors. When the button is red and clicked, you want to enable the axes and change the color to green, and vise-versa.

The tendency of the novice is to change the button green when the command to the Kflop is sent to enable the axis, and change the button to red when the command is sent to disable the axis.

This is not the best way, and in some cases can be dangerous, because you really are not indicating the actual state of the hardware. This can be very confusing and make problems, especially intermittent ones hard to detect.

A much better option is to monitor the status of the Kflop axes enable output bit periodically, and change the color of the button based on the actual hardware state. If the state changes for some other reason than the button being clicked, for example a safety tripping in the Kflop that disables the axes, you will know it. If you based the indicator on the button being clicked, the indicator would be out of sync with the hardware.

In any case, machines tools are dangerous, to you, others and themselves. Always choose the safest way possible, and provide checks that keep the operator from doing something ‘stupid’.

The operator that does something ‘stupid’ and wrecks the machine or hurts someone is not ignorant; it’s the programmer who let him do it that needs to have his head examined.

Putting safety checks in the code tends to make it more verbose and sometimes more complicated, but it is well worth the effort. For example starting the spindle with a spindle probe mounted will ruin an expensive device. For that reason if you look at my original code, you will see a check to see if the probe input is on, indicating the probe is in the spindle (the spindle probe switch is normally closed, so when it is in the spindle and hooked up, the input will read 1).