A Brief Introduction to State Machine Methodology...

... and why you should use it.

A State Machine is a mathematical concept whereby a system is driven by stimuli and its state alters by reacting to them. The condition of the stimuli indicates the state of the system at any point in time.

In computing, state machine methodology is employed to get away from having a traditionally tightly interlocked system in favour of one that is only active when a stimulus is received; Changing its “state” depending on those stimuli. Stimuli could be I/O pins, a specific string or more often, flags. Such a system tends to be loose-coupled to its tasks and thus easier to program for multiple actions, each being self contained often as the body code inside an IF test in the main thread.

As an example, consider the following:

```
Do

If ThisFlag Then
  DoThisProcess
EndIf

If ThatFlag Then
  DoThatProcess
EndIf

...etc

Loop
```

The Main Thread spends all it's time zipping around simply and rapidly checking flags and only taking a self-contained action when it has to. If the code is simple enough and only used in one place, then it should exist inside the If/EndIf block, otherwise write the code in a Sub program and call that from the If block.

With micro-controllers you don't often have an Operating System to handle the CPU when it is idle - you control the CPU at a very low level. In modern computer languages, the “main loop” would be a dormant state and an event handler would wake your program when the state of the stimuli changed (you don't have to give the CPU something to do while it is waiting for a state change). This makes all of the resources (while the CPU isn't busy) available to the system for other tasks.

State machine techniques are a major step towards event-driven programming on systems that don't normally have it. It can provide an almost multi-tasking appearance in micro-controllers as sections of code execute asynchronously to others, signaling and interlocking with other processes by setting or clearing flags - a process called “semaphore”. The process of semaphore by tweaking flags goes back to the earliest CPUs, when an action would be taken as a result of one process signaling a specific state which resulted in another process doing something about it. All CPUs support semaphores, if only with a zero or carry flag in the status register. This is a good way to signal from a one section of code to another.

State-machine code makes for easily readable and modifiable programs as each action becomes an

atomic section of code with a clearly identifiable trigger. You don't have to worry about disrupting other sections of code so long as any process handles its input values and provides the expected output values. Flags can be set by any section of code, triggering activity from anywhere in your code.