The attached code, MAC_sla (Manual Axis Control - Superloop(a)) has been extracted for post on my Github page (github.com/DKWatson) as an open source excerpt from my commercial product SAC which is interrupt driven. The essential purpose of MAC_sla is to provide the clock and direction signals to a stepper motor driver (A4988, TB6600, DRV8825, etc.) based on input received from a quadrature encoder. In turn, mode-specific (CRUISE, STEP) action will either be a change in rotational velocity or a single step. Direction or increase/decrease is a function of encoder rotation:

- clockwise, one step clockwise in STEP mode or velocity increase by one 1 RPM in CRUISE mode
- counter-clockwise, one step ccw or velocity decrease

Serial input allows for:

display current release	-v
mode change (toggle)	-M/m
timer on	T
timer off	t
zero at current position	Z/z
set current position to n	Z/zn
execute command/function	/ (current bezel position used as function pointer)
adjust rollover to n	R/rn

As is often the case when developing libraries, the need/desire/requirement for a one-size-fits-all solution results in unnecessarily fat code. In resource-restricted environments, I like to trim as much of this fat as possible and as such have written many drivers for use is specific applications. My general replacement is with the serial libraries. I do a lot of networking in noisy environments and have developed my own peer-to-peer RS485 protocol. This is extremely light-weight with CRC checking and node addressing. For speed and practicality I make use of the multi-processor communication mode available with most embedded USARTs. This is a 9-bit serial protocol that allows for hardware checking of address or data bytes thereby relieving the processor of the task and resource consumption. Until now however, I have found no manufacturer that releases anything more than an application note pertaining to MPC communications so out of necessity I've developed both my own hardware drivers and a soft, non-blocking, interrupt-driven, half-duplex library to support 9-bit protocol. This is not seen in the sample code but merely explains why I am not using off-the-shelf serial/print libraries.

With most development I prefer using a simple text editor (the current flavor-of-the-month is Notepad++) coupled with command line compiling/linking. This particular snippet has been written for an Arduino environment and compiles to 2654 bytes on the 328p and 2702 bytes for the 2560 using avr-gcc 7.3.0. The hex file is then simply loaded with avrdude.

Each manufacturer has their own IDE of course and if that is the in-house preference, so be it, makes no difference to me except for the possibility of unnecessary overhead. I always like to check IDE compiled code against the command line compiled version. If you have not yet establish an in-house ecosystem/toolchain, I have my opinions on that as well, and no, it's not following what I do.