# Mobiflight Arduino - Mega

Modifications to MF v.7.4.0 by Giorgio CROCI CANDIANI

2018-01

*Note: most of the memory reduction measures listed in this document, while mentioned as simple optimizations, become fully necessary once the number of I/O lines is substantially increased due to the introduction of virtual I/Os (see below).*

| ***Modification #1*** | |
|---|---|
| ***Object*** | **Replace the standard digitalWrite()/digitalRead() with fast versions** |
| ***Purpose*** | Speed optimization (minor) |
| ***Description*** | Use of FastArduino library |
| ***Implementation notes*** | Existing `digitalWrite()`/`digitalRead()` calls are not altered but replaced by redefinition (through `#define`'s) |
| ***Limitations*** | none |
| ***Changes required in Mobiflight Connector or system*** | none |

| ***Modification #2*** | |
|---|---|
| ***Object*** | **Replace the standard LedControl library with optimized version** |
| ***Purpose*** | Speed / memory optimization |
| ***Description*** | The stock `LedControl` library has been replaced by the `MFLedControl` library, which has several optimization both for memory footprint (e.g. constant values kept in code memory) and speed. |
| ***Implementation notes*** | - |
| ***Limitations*** | none |
| ***Changes required in Mobiflight Connector or system*** | none |

| ***Modification #3*** | |
|---|---|
| ***Object*** | **Handler pointers made static** |
| ***Purpose*** | Memory optimization (major) |
| ***Description*** | For `MFButton`, `MFEncoder` and other input devices: pointers to event handler functions are made class static, since they're the same for every instance. |
| ***Implementation notes*** | - |
| ***Limitations*** | It is no longer possible to assign different handlers to specific individual objects (however this has little practical relevance, if at all) |
| ***Changes required in Mobiflight Connector or system*** | none |

| ***Modification #4*** | |
|---|---|
| ***Object*** | **Introduction of bitStores for I/O status flags** |
| ***Purpose*** | Memory optimization (major) |
| ***Description*** | The status values for I/O lines (existing Buttons / Outputs, I/O expansions) are no longer contained in the objects,  but kept in a packed bit buffer, implemented by a new class (`bitStore`), and accessible both bit-wise and bank- (byte-)wise. |
| ***Implementation notes*** | There are two `bitStore` s that contain bulk I/O data: one is for the validated status, while the other is for newly read values, in order to detect changes.<br>All objects that deal with I/O lines (Buttons, Outputs, I/O blocks...) share and reference the same two bitStores above.<br>Classes `MFButton` (now `MFButtonT`) and `MFOutput` are modified to carry a reference to the common bitStores through a class static pointer.<br>*Note: Encoders can only be based on onboard pins, so related classes are not modified to this regard.* |
| ***Limitations*** | none |
| ***Changes required in Mobiflight Connector or system*** | none |

| ***Modification #5*** | |
|---|---|
| ***Object*** | **Pin occupancy registry is simplified** |
| ***Purpose*** | Memory optimization (major) |
| ***Description*** | Previously, the data structure used to record the pin assignment carried the information about what kind of peripheral the pin has been assigned to (although not which of those peripherals). This information, however, is not used further, and takes 1 byte for each pin.<br>The assignment registry is now based on a bitStore structure (actually, three of them), which means reduced storage space and better manageability. |
| ***Implementation notes*** | *The only situation in which a difference is apparent is in `clearRegisteredPins(type)`. This function is actually almost useless in practice, because it is called just in one place and for ALL types (might as well use `clearRegisteredPins(void)`). In order to keed the differentiation, however, `clearRegisteredPins(type)` has been implemented differently by scanning the registry of objects of the desired type and clearing the pinsRegistered value of their pins.* |
| ***Limitations*** | currently none |
| ***Changes required in Mobiflight Connector or system*** | none |

| ***Modification #6*** | |
|---|---|
| ***Object*** | **Optimization of Encoder-related classes** |
| ***Purpose*** | Memory optimization (major) |
| ***Description*** | Each encoder currently uses a huge amount of memory, particularly through the inheritance chain `MFEncoder` <- `RotaryEncoder` <- `TicksPerSecond`; this latter class in particular has lots of *long* member variables. Currently, each encoder takes about 80 bytes; 10 Encoders already take 10% of MCU memory!<br>**Solution:**<br>Turn a few attributes in the involved classes to static.<br>The max number of encoders has also been limited to a smaller value (from original 20 to max 10), although this is no longer strictly necessary and it can be increased again if required. |
| ***Implementation notes*** | Use a single, shared, static `TicksPerSecond` object common to all `RotaryEncoders` (one `TicksPerSecond` takes 35 bytes alone).<br>Class `RotaryEncoder` is replaced by the slightly modified `RotaryEncoderShd`, which uses the shared `TicksPerSecond` and also has some other members shared. |
| ***Limitations*** | Encoders can be now operated only one at a time (which reasonably is what happens anyway); otherwise, they should still work correctly (*to be tested*), however the timing for fast increment at higher rotation speeds will not work. |
| ***Changes required in Mobiflight Connector or system*** | none |

| ***Modification #7*** | |
|---|---|
| ***Object*** | **Removal of user-assigned names for inputs (Buttons / Encoders)** |
| ***Purpose*** | Memory optimization (major) |
| ***Description*** | Currently, when a Button is created in a module in Connector, the user must assign a name to it. This name is:<br>- transmitted as configuration info to the Arduino firmware<br>- stored therein (by dynamic allocation)<br>- used to be sent back later, in the transmission of an event regarding that Button.<br>The same also applies for Encoder lines (for other pins, e.g. Outputs, names are transmitted but neither stored nor used in any other way).<br>Names in the end occupy a large quantity of memory (in the heap, which can cause not easily traceable runtime problems), even more so if I/O number is expanded.<br>**Solution:**<br>Since names are not actually used by the Arduino firmware, except for retransmission, it appears that a sensible solution would be to completely get rid of them.<br>To send back an event, a conventional name is used: **B*nnn*** for buttons, **E*nnn*** for encoders (where *nnn* is the button pin / encoder first pin). The connector application will easily re-translate these codes by associating their original name back.<br>This also has advantages, albeit not drastic, on the transmission rate. |
| ***Implementation notes*** | This solution involves modifications to classes `MFButtonT` and `MFEncoder`. |
| ***Limitations*** | none |
| ***Changes required in Mobiflight Connector or system*** | MFConnector must now include the capability to translate codes received for input events back into user-assigned names. |

| ***Modification #8*** | |
|---|---|
| ***Object*** | **Introduction of remote I/O banks** |
| ***Purpose*** | Use of peripherals for digital I/Os, like I/O expanders or shift registers |
| ***Description*** | The firmware has been extended in order to support digital I/O lines based on external “blocks” (eg I/O extenders, shift registers etc).<br>I/Os on peripherals are distinguished from on-board I/Os by calling them “Virtual” pins (or lines), as opposed to on-board or “Real” pins.<br>Virtual lines are allocated in the index range 64 and up (the upper limit chosen is 255); lines 0...63 (actually 0...56 maximum for a Mega2560) are identifiable as onboard, “actual” I/Os.<br>Blocks implemented so far:<br>• *MFOutput595* ........ 8-bit Output banks based on 74HC(T)595, TPIC6B595 etc.; chainable up to 8 blocks<br>• *MFInput165* .......... 8-bit Input banks based on 74HC(T)165; chainable up to 8 blocks<br>• *MFInputMtx* .......... Matrix keyboard/switchboard input, up to 8 rows x 8 columns<br>• *MFIO_MCP0* ............ 2 x 8-bit Input/Output banks (configurable) based on MCP23017 I/O expander. I2C interface (HW, on standard pins, or SW, on assignable pins)<br>• *MFIO_MCPS* ............ 2 x 8-bit Input/Output banks (configurable) based on MCP23017 I/O expander. SPI interface<br>• *MFOutLEDDM13* ...... 16-bit LED Output. Suits many equivalent (or compatible) ICs, e.g. DM13A, DM13x, TLC5926, MBI5026, , SM16126, STP16CP05, SCT202x, TB627xx, A6276... |

| | |
|---|---|
| ***Implementation notes*** | In order to use remote I/O banks, two features are necessary:<br>- find a way to make the handling of newly available lines seamless with respect to existing on-board I/Os<br>- introduce classes for peripheral IC "drivers"<br>**I/O value handling:**<br>When an I/O peripheral block is instantiated, a corresponding number of ordinary Button / Output objects are also created; these behave and are managed exactly like existing onboard Buttons / Outputs.<br>The usage of packed bitStores allows to handle the status of the Virtual lines in the very same way as for the currently available onboard lines, except only for the lowest level read operation (`digitalRead()`/`digitalWrite()` vs bitStore lookup).<br>**Peripheral drivers:**<br>Drivers are implemented through following set of classes.<br>Class `MFPeripheral`: represents a peripheral block that is driven through onboard pins.<br>Used as interface, this class has (mostly virtual) members to manage:<br>- a set of driver pins;<br>- a set of generic functions for operations (eg attach – detach – update – test)<br>*This class is also applied to existing "peripherals" (eg LED/LCD displays).*<br>Class `MFIOBlock`: represents a peripheral block that has banks of digital I/Os mapped to a bitStore (see further below).<br>Used as interface, this class has (mostly virtual) members to manage:<br>- the definition of how many banks the peripheral owns, and how they are arranged;<br>- a set of generic functions for operations (actual data R/W, I/O direction setting, info request)<br>- the link to the `bitStores` (and the positions therein) containing bulk I/O data, validated and newly read (the same ones used for Buttons and Outputs).<br>Classes for actual peripherals: represent a specific peripheral block type.<br>They inherit the above classes, implement their virtual members (plus others specific to the peripheral), and embed configuration and business logic specific to their components.<br>These classes are the ones already mentioned above in the listing (*`MFOutput595, MFInput165, MFInputMtx, MFIO_MCP0, MFIO_MCPS, MFOutLEDDM13`*).<br>New `kType[IOBlockXXX]` constants are also defined, as well as `Add[IOBlockXXX]()` functions, each with its own specific HW driver params and configuration values; the IO blocks are set during the configuration phase in the same way as existing peripherals. |
| ***Limitations*** | - Virtual I/Os can only be used as Buttons or Outputs, not for any other purpose (e.g. for Encoders or as driver pins, which of course have to be physical pins)<br>- For consistency and simplicity, a Virtual pin can only be either an input or an output (while, theoretically, I/O spaces could be made to overlap)<br>- Different peripheral blocks of the same type can be controlled through common pins (eg. MCP23S17 units with different addresses). *This is supported by the Arduino firmware*, however the Connector application should be modified to account for the fact that several peripherals may share some of their driver pins.<br>*Some of the above limitations are by choice and could be removed by extending the code.* |
| ***Changes required in Mobiflight Connector or system*** | Implementation of "plugins" for the configuration of added I/O blocks.<br>*(support for shared driver pins as possible improvement, see limitations)* |

| *Modification #9 (TO BE IMPLEMENTED)* | |
|---|---|
| ***Object*** | **Add LED Display block for boards using the TM1638 driver** |
| ***Purpose*** | Extension |
| ***Description*** | - |
| ***Implementation notes*** | - |
| ***Limitations*** | none |
| ***Changes required in Mobiflight Connector or system*** | Implementation of "plugins" for the configuration of LED display (or extension of current one). |

| *Modification #10 (TO BE DEFINED)* | |
|---|---|
| ***Object*** | **Add Ethernet communication capability** |
| ***Purpose*** | Allow alternative (and usually more reliable) way of communication |
| ***Description*** | Add the option of addressing the board (during operation) through a LAN (or WiFi). |
| ***Implementation notes*** | Largely to be defined.<br>LAN comms should be limited to the operation phase (Firmware programming still occurs through USB)<br>Support for which Arduino Ethernet/WiFi boards TBD |
| ***Limitations*** | TBD |
| ***Changes required in Mobiflight Connector or system*** | Implementation of comm layer for Ethernet<br>Support for LAN configuration (IP setting etc)<br>Support for dual Arduino board identification (consistent through both USB and LAN) |