Following code snippets should give you an inspiration, how an existing assembly based applications like MIDIbox64 or MIDIbox64E can be easily enhanced by additional button functions, which send statically assigned MIDI events.

The idea is to overlay the USER_DIN_NotifyToggle hook. This hook is called by MIOS when a digital input has changed its state (0V→5V or 5V→0V). You can check if a DIN number is within the button range, which should be overlayed. If it is outside the range, then just continue with the application specific code, which handles the remaining buttons - thats all.

Technical details:

The hook is normaly located in the main.asm file of the application.

MIOS forwards following parameters to the hook: MIOS_PARAMETER1: contains the number of the digital input, counted from zero (0..127) MIOS_PARAMETER2: bit #0 is 0, when the digital input is at 0V, and 1, if the digital input is at 5V If buttons are connected, this means: 0V when button closed (pressed), 5V when button open (depressed)

The button range can be checked with the IFLEQ (if-less-equal) and IFGEQ (if-greater-equal) macros, which are defined in macros.h:

```
    ;; overlay DIN input #64-#127 (counted from zero) by a simple
    ;; function which sends dedicated Note Events
    movlw    64-1
    IFLEQ    MIOS_PARAMETER1, ACCESS, rgoto USER_DIN_NotifyToggle_NoOverlay
    movlw   127+1
    IFGEQ    MIOS_PARAMETER1, ACCESS, rgoto USER_DIN_NotifyToggle_NoOverlay
USER_DIN_NotifyToggle_Overlay
    movlw    0x90            ; Note Event, MIDI Channel #1
    call    MIOS_MIDI_TxBufferPut
    movf    TMP1, W            ; DIN number -> Note Number
    call    MIOS_MIDI_TxBufferPut
    movlw    0x7f            ; Velocity: 0x7f if button pressed
    IFSET    MIOS_PARAMETER2, , movlw 0x00    ;   0x00 if button depressed
    call    MIOS_MIDI_TxBufferPut
    return                ; exit
USER_DIN_NotifyToggle_NoOverlay

    ;; ...rest of the application specific code
```

An interesting variation is following example, which sends different MIDI events depending on a certain button - we could call it “SHIFT” button:

```
    ;; overlay DIN input #64-#127 (counted from zero) by a simple
    ;; function which sends dedicated Note Events
    movlw    64-1
    IFLEQ    MIOS_PARAMETER1, ACCESS, rgoto USER_DIN_NotifyToggle_NoOverlay
    movlw   127+1
    IFGEQ    MIOS_PARAMETER1, ACCESS, rgoto USER_DIN_NotifyToggle_NoOverlay
USER_DIN_NotifyToggle_Overlay
;; since MIOS_DIN_PinGet overwrites MIOS_PARAMETER1, store current button
number in TMP1
```

```
    movff    MIOS_PARAMETER1, TMP1

    ;; we are using button #8 as shift button (it's outside the range which
is overlayed)
    movlw    8        ; get value of this button (0=pressed, 1=depressed)
    call    MIOS_DIN_PinGet
    bz    USER_DIN_NotifyToggle_Overlay_1    ; branch depending on selection
state --- bz == "branch if zero"
USER_DIN_NotifyToggle_Overlay_0
    movlw    0x90            ; Note Event, MIDI Channel #1
    call    MIOS_MIDI_TxBufferPut
    movf    TMP1, W            ; DIN number -> Note Number
    call    MIOS_MIDI_TxBufferPut
    movlw    0x7f            ; Velocity: 0x7f if button pressed
    IFSET    MIOS_PARAMETER2, , movlw 0x00    ;   0x00 if button depressed
    call    MIOS_MIDI_TxBufferPut
    return                ; exit

USER_DIN_NotifyToggle_Overlay_1
    movlw    0x91            ; Note Event, MIDI Channel #2
    call    MIOS_MIDI_TxBufferPut
    movf    TMP1, W            ; DIN number -> Note Number
    call    MIOS_MIDI_TxBufferPut
    movlw    0x7f            ; Velocity: 0x7f if button pressed
    IFSET    MIOS_PARAMETER2, , movlw 0x00    ;   0x00 if button depressed
    call    MIOS_MIDI_TxBufferPut
    return                ; exit
USER_DIN_NotifyToggle_NoOverlay

    ;; ...rest of the application specific code
```

Please note: if you have ideas for much more complex button functions, but no motivation to learn assembly language, just consider the use of the SDCC wrapper - see also the examples at this Page

Unfortunately it is not possible to combine the assembly based applications like MIDIbox64 or MIDIbox64E with C programs, but maybe your requirements to the application don't match with these historic MIDIboxes anyhow (e.g. you don't need on-screen editing capabilities, you don't need several banks of buttons/encoders/pots/fader setups, you only want to send unique MIDI events to a host), that it makes sense to program a small C based application instead, and especially to share it with other MIDIbox users!

Additional Hints:

- please note, that digital inputs could already be overlayed by the encoder handler. The encoder pins are pre-defined in mios_tables.inc (if you are compiling a main.asm file, in some applications the table could also be located in a setup_*.asm file)
  So, if some of the DINs don't send a MIDI event, it makes sense to check the encoder table!
- you could also overlay *all* button functions this way if you want, just remove the button range checks - pins which are assigned to rotary encoders are not affected (they don't trigger the MIOS_DIN_NotifyToggle hook)