The Button_Utils library

2012-11-22

Content

Introduction

This library provides three “button” related types of behaviour that often are wanted in a project:

- Every time a button is pushed, something must happen once, or
- A button must be “autorepeating”: something must happen when the button is pushed and after that something must happen every xxx milliseconds as long the button is pushed.
- A button must have 2 different functions: one function when the key is pressed for a short time, another function when the key is pressed for a long time.

Those parts of this library are called furtheron the “Non Repeat”, the “Repeat” and the “Timed” part. All function completely independent of each other.

The “non repeat” functionality

This part of the library provides the simple button functionality: “Every time a button is pushed, something must happen once”.
The “button event” is given by setting a boolean variable (the “Event Flag”) to “true”. It is the responsibility of the using program to set it to “false” again after processing it.

Initialisation

The initialisation of this part of the library is done as follows:

```
Button_NoRepeat_Init;
```

This will clear the “non repeated” button queue in the library.

Submitting a “button” to the library

Every button of which the behaviour is “non repeated” as described above can be submitted to the library for processing as follows:

```
Button_NoRepeat_Add(PortA, 4, 100, 0, Key1);
```

The first 4 parameters are the same as those of the “Button” procedure:

- Port
- Bit
- Debounce time in milliseconds
- Active State

The last parameter is the button “event” flag (boolean).

A maximum of 5 “non repeated buttons” can be submitted for processing simultaneously.

A “non repeated button” can also be removed for processing by the library, e.g.:

```
Button_NoRepeat_Delete(Key1);
```

Making the library do its work

- ***Call the “Button_NoRepeat_Step” procedure***

To make the library do its work the procedure “Button_NoRepeat_Step” has to be called regurarly in the main program loop.

This procedure does the scanning of the buttons submitted to the library and performs the necessary state activities. When appropriate it sets the “Event Flag” to true to signal “button pressed” to the using program. It is the responsibility of the latter to set it to “false” again after processing it.

The procedure is not blocking.

- ***Monitor the "Event Flag"***

Do not forget to monitor the "Event Flag" variable in your (main) program loop, and to clear it after processing.

Example:

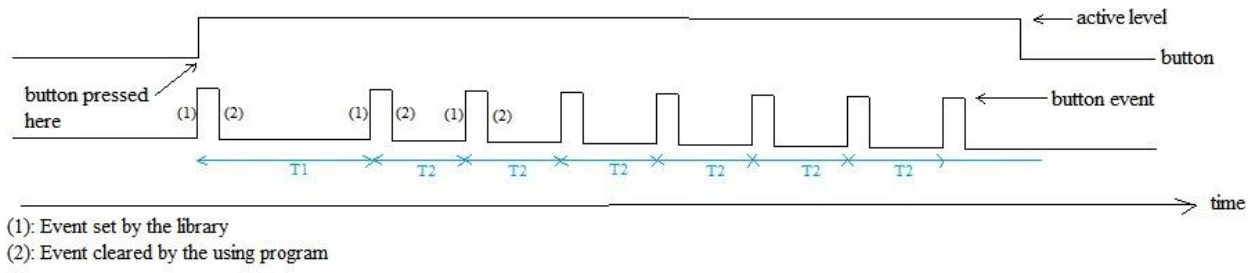
```
Button_NoRepeat_Init;
Button_NoRepeat_Add(PortA, 4, 50, 0, Key1);

while true do
  begin
    Button_NoRepeat_Step; // ←--- do the processing of the non repeated keys

    if Key1 then          // ←--- monitor the "Event Flag"
    begin
      Key1 := false;      // ←--- clear the "Event Flag"
      Do_Something;       // ←--- process according the "Event Flag"
    end;
  end;
```

The “Repeat” functionality

This part of the library provides the possibility to have some “repeating” buttons. This means that an event is given to the using program when:

- The button is pressed after being released previously
- When the button is kept pressed:
  - The first time after a time T1 (initial repeat time)
  - All other times with an interval of T2 (final repeat time)

Graphically:


active level
button
button pressed here
button event
(1)
(2)
T1
T2
time
(1): Event set by the library
(2): Event cleared by the using program

See the top line for the actual state of the button, the second line for the events that are given to the using program.

The “button event” is given by setting a boolean variable (the “Event Flag”) to “true”. It is the responsibility of the using program to set it to “false” again after processing it.

Intialisation

This is simply:

```
Button_Repeat_Init;
```

This will clear the “repeated” button queue in the library.

Do not forget to initialise also the “Timers” library (and provide it its 1 ms timer tick), since this library uses “Timers”.

Submitting a “button” to the library

Submitting a button to the library for “repeat” processing is done as follows:

```
Button_Repeat_Add(PortA, 4, 100, 0, 1000, 300, Key1);  // key 1
```

The first 4 parameters are the same as those of the “Button” procedure:

- Port
- Bit
- Debounce time in milliseconds
- Active State

The three following parameters are:

- T1 (initial repeat time in ms)
- T2 (final repeat time in ms)

- The button “event” flag (boolean)

A maximum of 5 “repeated buttons” can be submitted for repeat processing simultaneously.

A “repeated button” can also be removed for processing by the library, e.g.:

```
Button_Repeat_Delete(Key1);
```

Making the library do its work

- ***Call the “Button_Repeat_step” procedure***

To make the library do its work the procedure “Button_Repeat_Step” has to be called regurarly in the main program loop.

This procedure does the scanning of the buttons submitted to the library and performs the necessary timing/state activities. When appropriate it sets the “Event Flag” to true to signal “button pressed” to the using program. It is the responsibility of the latter to set it to “false” again after processing it.

The procedure is not blocking.

- ***Monitor the “Event Flag”***

Do not forget to monitor the “Event Flag” variable in your (main) program loop, and to clear it after processing.

Example:

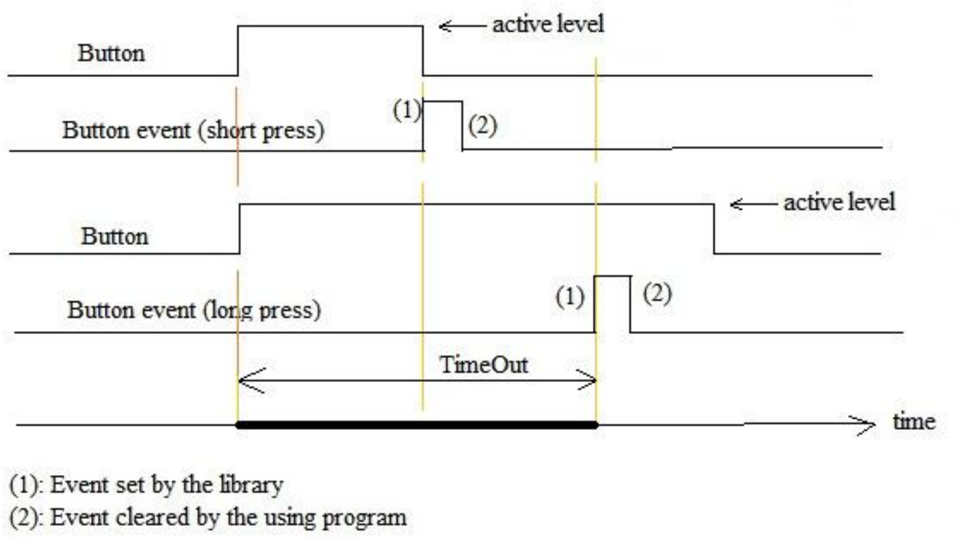
```
Button_Repeat_Init;
Button_Repeat_Add(PortA, 5, 50, 0, 1000, 300, Key1);

while true do
  begin
    Button_Repeat_Step; // ←--- do the processing of the repeated keys

    if Key1 then        // ←--- monitor the “Event Flag”
    begin
      Key1 := false;    // ←--- clear the “Event Flag”
      Do_Something;     // ←--- process according the “Event Flag”
    end;
  end;
```

The “Timed” functionality

This part of the library provides the possibility to have some “double function buttons:

- If the button is pressed for a short time (shorter than the “TimeOut” time) then the “short push” event is generated,
- If the button is pressed for a long time (longer than the “TimeOut” time) then the “long push” event is generated.

Graphically:


Button
active level
Button event (short press)
(1)
(2)
Button
active level
Button event (long press)
(1)
(2)
TimeOut
time
(1): Event set by the library
(2): Event cleared by the using program

The first two lines give the timing diagram when the button is pressed for a short time. The first line is the state of the button itself, the second line shows the event generated. As you can see the event is generated when the button is released. The “button event” is given by setting a boolean variable (the “Event Flag”) to “true”. It is the responsibility of the using program to set it to “false” again after processing it.

The next two lines show the timing diagram when the button is pushed for a long time (longer than “TimeOut”). The first of these lines is again the state of the button itself, the second of theses lines shows the event generated. As you can see the event is generated when the TimeOut expires. The “button event” is given by setting a boolean variable (the “Event Flag”) to “true”. It is the responsibility of the using program to set it to “false” again after processing it.

The bottom line shows the “TimeOut” period, which starts when the button becomes pressed.

Initialisation

This is simply:

```
Button_Timed_Init;
```

This will clear the “timed” button queue in the library.

Do not forget to initialise also the “Timers” library (and provide it its 1 ms timer tick), since this library uses “Timers”.

Submitting a “button” to the library

Submitting a button to the library for “timed” processing is done as follows:

```
Button_Timed_Add(PortA, 4, 100, 0, 1000, Key1, Key2);
```

The first 4 parameters are the same as those of the “Button” procedure:

- Port
- Bit
- Debounce time in milliseconds
- Active State

The three following parameters are:

- the TimeOut (in ms)
- the “Short” button press event flag (boolean)
- the “Long” button press event flag (boolean)

A maximum of 5 “timed buttons” can be submitted for timed processing simultaneously.

A “timed button” can also be removed for processing by the library, e.g.:

```
Button_Timed_Delete(Key1); // where Key1 is the "Short" button event flag
```

Making the library do its work

- ***Call the “Button_Timed_step” procedure***

To make the library do its work the procedure “Button_Timed_Step” has to be called regurarly in the main program loop.

This procedure does the scanning of the buttons submitted to the library and performs the necessary timing/state activities. When appropriate it sets one the “Event Flags” to true to signal “button pressed short” or “button pressed long” to the using program. It is the responsibility of the latter to set them to “false” again after processing it.

The procedure is not blocking.

- ***Monitor the “Event Flags”***

Do not forget to monitor the “Event Flag” variables in your (main) program loop, and to clear them after processing.

Example:

```
Button_Timed_Init;
Button_Timed_Add(PortA, 5, 50, 0, 1000, Key1, Key2); // 1000 ms timeout

while true do
  begin
    Button_Timed_Step;       // ←--- do the processing of the timed keys

    if Key1 then             // ←--- monitor the “short push Event Flag”
    begin
      Key1 := false;         // ←--- clear the “short push Event Flag”
      Do_Something;          // ←--- process according the “short push Event Flag”
    end;

    if Key2 then             // ←--- monitor the “long push Event Flag”
    begin
      Key2 := false;         // ←--- clear the “long push Event Flag”
      Do_Something_else;     // ←--- process according the “long push Event Flag”
    end;
  end;
```

The Library interface

```
unit Button_Utils;

// interface

{$IFDEF P16}
uses Timers_p16;
{$ELSE}
uses Timers;
{$ENDIF}

// ------------------- Non Repeated buttons interface --------------------------

procedure Button_NoRepeat_Init;
// Empties the "non repeated Buttons" Queue

function Button_NoRepeat_Add(var port : byte; pin, time, active_state : byte; var Flag: boolean):
boolean;
// Adds a "non repeated button" to the Queue; "Flag" will become true when the button is "pressed"
after it was released

procedure Button_NoRepeat_Delete(var Flag: boolean);
// Deletes a non repeated button from the Queue

procedure Button_NoRepeat_Step;
// To be called in the main program loop


// ------------------- Repeated buttons interface ------------------------------

procedure Button_Repeat_Init;
// Empties the "repeated Buttons" Queue

function Button_Repeat_Add(var port : byte; pin, time, active_state : byte; T1, T2: word; var Flag:
boolean): boolean;
// Adds a "repeated button" to the Queue; "Flag" will become true when the button is "pressed"

procedure Button_Repeat_Delete(var Flag: boolean);
// Deletes a repeated button from the Queue

procedure Button_Repeat_Step;
// To be called in the main program loop


// ------------------- Timed (Short/Long press) buttons interface --------------

procedure Button_Timed_Init;
// Empties the "timed Buttons" Queue

function Button_Timed_Add(var port : byte; pin, time, active_state : byte; T1: word; var FlagShort,
FlagLong: boolean): boolean;
// Adds a "timed button" to the Queue; "Flag1" will become true when the button is "pressed" and
"released" within time T1 millisecs
// after pressing, "Flag2" will become true when the button is pressed longer than T1 millisecs.

procedure Button_Timed_Delete(var Flag: boolean);
// Deletes a timed button from the Queue

procedure Button_Timed_Step;
// To be called in the main program loop
```

Usage example

```
program ButtonUtilsLib;

{ Declarations section }

uses Button_Utils, Timers;

var Event1, Event2, Event3, Event4 : boolean;           // events
    TimersTimerInterruptBit : sbit at TMR2IE_bit; sfr;  // 1 ms Timer interrupt enable bit, here
timer2

procedure interrupt;
begin
  if TMR2IF_bit = 1 then     // 1 ms interrupt from TMR2
  begin
    TMR2IF_bit := 0;         // reset the timer2 interrupt bit
    Timers_TimerInterrupt;
  end;
end;

procedure ToggleLed;
begin
  LatA.0 := not LatA.0;
end;

procedure LedOn;
begin
  LatA.0 := 1;
end;

procedure LedOff;
begin
  LatA.0 := 0;
end;

begin
  { Main program }

  CMCON  := 7;              // Disable Comparator module's
  ADCON1 := $0F;            // all digital ports

  TrisA := %11111110;      // Port A.0 is output
  LatA.0 := 0;

  // Initialise the 1 millisecs timer (here timer2)
  T2CON   := %01011001; // prescaler 4, postcaler 12
  PR2     := 250;       // preload value, interrupt every millisec  <-- value for 48 Mhz MPU clock
  TMR2    := 0;         // timer 2 register reset
  T2CON.TMR2ON := 1;    // start TMR2
  INTCON  := %11000000; // enable interrupts (PEIE and GIE)
  TMR2IE_bit := 1;      // enable Timer2 interrupt

  Timers_Init;

  Event1 := false;
  Event2 := false;
  Event3 := false;
  Event4 := false;

  Button_NoRepeat_Init;
  Button_NoRepeat_Add(PortA, 4, 50, 0, Event1);
  // Every push on the button will toggle the led

  Button_Repeat_Init;
  Button_Repeat_Add(PortA, 5, 50, 0, 1000, 300, Event2);
  // When the button is pushed first the led will be toggled, when the button is held pushed, the
led will toggle again after
  // 1000 msecs, and from that moment on every 300 msecs as long as the button is pushed
  Button_Timed_Init;
  Button_Timed_Add(PortA, 3, 50, 0, 1000, Event3, Event4); // timeout of 1000 ms
  // When the button is released after it was pushed less than 1000 ms the led will be switched on,
  // as soon as the button is pushed more than 1000 msecs, the led will be switched off.

  while true do
  begin
    Button_NoRepeat_Step;
    Button_Repeat_Step;
    Button_Timed_Step;

    if Event1 then
    begin
      Event1 := false;
      ToggleLed;
    end;

    if Event2 then
    begin
      Event2 := false;
      ToggleLed;
    end;

    if Event3 then
    begin
      Event3 := false;
      LedOn;
    end;

    if Event4 then
    begin
      Event4 := false;
      LedOff;
    end;

  end;

end.
```

[end of document]