Discrete Logic Replacement

Debounce

Author: *Dag Bakken*
Component-74 Eidsvold AS
RAHOLT Norway
Email: dag.bakken@microchip.com

OVERVIEW

This piece of QuickCode is kind of a concept. The whole idea behind it, is to de-bounce without having to wait for the de-bouncing to finish without using interrupts or timers.

APPLICATION OPERATION

The concept is extremely simple and easy to use, and generates very few words of code. The amount of code will vary greatly, depending on how big your buffer must be. Usually, you can do with four bytes as in this piece of code. The two routines that handle the buffer/ debouncing are a total of 31 instructions with a 4-byte deep buffer. A few instructions will be added, if you require more buffer. The total amount of RAM is 4 for the buffer, 1 for 'last key', and 1 for return value (only 12-bit core of course). None of the functions needs any local variables.

The way this works, is by implementing some sort of multitasking. The basic idea behind it is that no tasks in your software should ever wait. By writing the entire software with this concept in mind, you can write software with virtually unlimited task-capacity. You can run fairly accurate PWMs together with other timers; All based on one timer. At the same time, you can implement the code supplied in this document to de-bounce some keys, and you can add software RS-232 communication - simultaneously. Of course, as you add functions to the software, the clock-speed may need some adjustment.

One of the things I've used it for, is interfacing to displays in fairly time-critical applications. Displays do tend to be slow, and a PICmicro™ spends most of its time waiting when updating an entire display.

To make full use of this kind of programming, a message-based program-loop really helps the multitasking work. Both the message-based program-loop and the "no-wait" programming method use very few instructions per loop, and this makes it easy to write large programs that use very little time per pass.

SUPPLIED FUNCTIONS

`char Debounce();`

This function checks the current key buffer and last valid keypress. If the test fails, a zero is returned, Meaning 'no key'.

`void PutKey(char k)`

This function pushes the currently pressed key (not debounced) into the key buffer.

`char ReadKeyboard()`

This is the function that handles the test for determining which key is currently pressed, and makes sure that it's pushed into the buffer. The return value should be the returned value from `char Debounce()`.

`void main()`

In this example, this function handles the calling of `char ReadKeyboard()`. This may, of course, be handled by any function your software requires. Either way, the calling function must call the `char ReadKeyboard()` function at appropriate intervals for your application.

MICROCHIP TOOLS USED

Assembler/Compiler version

CC5X v2.1H (C-Compiler)

The generated ASM-code assembles with MPASMWIN v1.50. A straight cut and paste from this document will work.

APPENDIX A: SOURCE CODE

CC5X v2.1H C-source

```
#include "c:\bruker\dag_s\progs\12c508.h"

#define BOOL bit

#pragma BOOL COL0 @ GPIO.0                      // Assigned keyboard-column 0
#pragma BOOL COL1 @ GPIO.1                      // Assigned keyboard-column 1
#pragma BOOL ROW0 @ GPIO.2                      // Assigned keyboard-row 0
#pragma BOOL ROW1 @ GPIO.3                      // Assigned keyboard-row 1


char retval;                                    // This is used to simulate
                                                // return values on a 12-bit core.
char LastValidKey;                              // This is used to test for
                                                // changes in valid key-presses.
char KBuf1,KBuf2,KBuf3,KBuf4;                   // Buffer for de-bouncing.  Set
                                                // this buffer to whatever your
                                                // application requires.

/* This function will check the current contents of the buffer,
   and the last valid key-press.  Returns the current valid
   key-press, or zero if it's not a valid key. */
char Debounce()
{
  retval=0;
  if (KBuf1!=KBuf2) return 0x00;                // Check buffer
  if (KBuf2!=KBuf3) return 0x00;                // Check buffer
  if (KBuf3!=KBuf4) return 0x00;                // Check buffer
  if (LastValidKey==KBuf1) return 0x00;         // Check last de-bounced
                                                // value against current
                                                // de-bounced value.
  LastValidKey=KBuf1;                           // Set this key-press
                                                // as valid.
  retval=KBuf1;                                 // Return de-bounced
  return 0;                                     // key-press
}

/* This function will put the current key-press in the de-bounce
   buffer */
void PutKey(char k)
{
  KBuf1=KBuf2;                                  // PUSH value
  KBuf2=KBuf3;                                  // PUSH value
  KBuf3=KBuf4;                                  // PUSH value
  KBuf4=k;                                      // PUSH value
}

/* This is the main function that checks the keyboard and handles
   all events.  This function is provided as a guide-line on how
   to use the other de-bouncing features. */
char ReadKeyboard()
{
  COL0=1; COL1=0;
  if (ROW0)
  { PutKey('1');                                // Key '1' detected
    goto _FOUND_ONE;
  }
  if (ROW1)
  { PutKey('2');                                // Key '2' detected
    goto _FOUND_ONE;
  }
  COL0=0; COL1=1;
```

```
  if (ROW0)
  { PutKey('3');                                // Key ‘3’ detected
    goto _FOUND_ONE;
  }
  if (ROW1)
  { PutKey('4');                                // Key ‘4’ detected
    goto _FOUND_ONE;
  }
  COL1=0;
  PutKey(0x00);                                 // If no key were pressed

_FOUND_ONE:
  COL0=0; COL1=0;

  Debounce();                                   // De-bounce, and return
  return 0;                                     // de-bounced value.
}

/* The main() function is provided so the program will compile if
   you do a cut n’ paste from this source into your editor. */
void main()
{
  do {
    ReadKeyboard();                             // By executing this line at
    switch(retval)                              // certain intervals, keyboard
                                                // will be de-bounced.
    {
      case '1': break;                          // Test
      case '2': break;                          // Test
      case '3': break;                          // Test
      case '4': break;                          // Test
    }
    /* Do something else while
       waiting for valid key-press */
  } while(1);
}
```

A.1 MPASM-code generated by CC5X v2.1H

```
; CC5X  Version 2.1H, Copyright (c) B. Knudsen Data
; C compiler for the PIC16CXX microcontroller family
; *************   1. Aug 1997  14:38  **************

         processor  12C508

Zero_       EQU   2
COL0        EQU   0
COL1        EQU   1
ROW0        EQU   2
ROW1        EQU   3
retval      EQU   0x08
LastValidKey EQU   0x09
KBuf1       EQU   0x0A
KBuf2       EQU   0x0B
KBuf3       EQU   0x0C
KBuf4       EQU   0x0D
k           EQU   0x07

         GOTO main

  ; FILE C:\TEMP\temp.c
                         ;#include "c:\bruker\dag_s\progs\12c508.h"
                         ;
                         ;#define BOOL bit
                         ;
                         ;#pragma BOOL COL0 @ GPIO.0       // Assigned keyboard-column 0
                         ;#pragma BOOL COL1 @ GPIO.1       // Assigned keyboard-column 1
                         ;#pragma BOOL ROW0 @ GPIO.2       // Assigned keyboard-row 0
                         ;#pragma BOOL ROW1 @ GPIO.3       // Assigned keyboard-row 1
                         ;
                         ;
                         ;char retval;                     // This is used to simulate
                         ;                                 // return values on a 12-bit core.
                         ;char LastValidKey;               // This is used to test for
                         ;                                 // changes in valid key-presses.
                         ;char KBuf1,KBuf2,KBuf3,KBuf4;    // Buffer for de-bouncing.  Set
                         ;                                 // this buffer to whatever your
                         ;                                 // application requires.
                         ;
                         ;/* This function will check the current contents of the buffer,
                         ;   and the last valid key-press.  Returns the current valid
                         ;   key-press, or zero if it's not a valid key. */
                         ;char Debounce()
                         ;{
Debounce
                         ;  retval=0;
         CLRF  retval
                         ;  if (KBuf1!=KBuf2) return 0x00; // Check buffer
         MOVF  KBuf1,W
         XORWF KBuf2,W
         BTFSS 0x03,Zero_
         RETLW .0
                         ;  if (KBuf2!=KBuf3) return 0x00; // Check buffer
         MOVF  KBuf2,W
         XORWF KBuf3,W
         BTFSS 0x03,Zero_
         RETLW .0
                         ;  if (KBuf3!=KBuf4) return 0x00; // Check buffer
         MOVF  KBuf3,W
         XORWF KBuf4,W
         BTFSS 0x03,Zero_
         RETLW .0
```

```
                        ;  if (LastValidKey==KBuf1) return 0x00;// Check last de-bounced
        MOVF  LastValidKey,W
        XORWF KBuf1,W
        BTFSC 0x03,Zero_
        RETLW .0
                        ;                                  // value against current
                        ;                                  // de-bounced value.
                        ;  LastValidKey=KBuf1;             // Set this key-press
        MOVF  KBuf1,W
        MOVWF LastValidKey
                        ;                                  // as valid.
                        ;  retval=KBuf1;                   // Return de-bounced
        MOVWF retval
                        ;  return 0;                       // key-press
        RETLW .0
                        ;}
                        ;
                        ;/* This function will put the current key-press in the de-bounce
                        ;   buffer */
                        ;void PutKey(char k)
                        ;{
PutKey
        MOVWF k
                        ;  KBuf1=KBuf2;                    // PUSH value
        MOVF  KBuf2,W
        MOVWF KBuf1
                        ;  KBuf2=KBuf3;                    // PUSH value
        MOVF  KBuf3,W
        MOVWF KBuf2
                        ;  KBuf3=KBuf4;                    // PUSH value
        MOVF  KBuf4,W
        MOVWF KBuf3
                        ;  KBuf4=k;                        // PUSH value
        MOVF  k,W
        MOVWF KBuf4
                        ;}
        RETLW .0
                        ;
                        ;/* This is the main function that checks the keyboard and handles
                        ;   all events.  This function is provided as a guide-line on how
                        ;   to use the other de-bouncing features. */
                        ;char ReadKeyboard()
                        ;{
ReadKeyboard
                        ;  COL0=1; COL1=0;
        BSF   0x06,COL0
        BCF   0x06,COL1
                        ;  if (ROW0)
        BTFSS 0x06,ROW0
        GOTO  m001
                        ;  { PutKey('1');                  // Key ‘1’ detected
        MOVLW .49
        CALL  PutKey
                        ;    goto _FOUND_ONE;
        GOTO  m005
                        ;  }
                        ;  if (ROW1)
m001    BTFSS 0x06,ROW1
        GOTO  m002
                        ;  { PutKey('2');                  // Key ‘2’ detected
        MOVLW .50
        CALL  PutKey
                        ;    goto _FOUND_ONE;
        GOTO  m005
                        ;  }
                        ;  COL0=0; COL1=1;
```

```
m002    BCF   0x06,COL0
        BSF   0x06,COL1
                        ;  if (ROW0)
        BTFSS 0x06,ROW0
        GOTO  m003
                        ;  { PutKey('3');                // Key ‘3’ detected
        MOVLW .51
        CALL  PutKey
                        ;    goto _FOUND_ONE;
        GOTO  m005
                        ;  }
                        ;  if (ROW1)
m003    BTFSS 0x06,ROW1
        GOTO  m004
                        ;  { PutKey('4');                // Key ‘4’ detected
        MOVLW .52
        CALL  PutKey
                        ;    goto _FOUND_ONE;
        GOTO  m005
                        ;  }
                        ;  COL1=0;
m004    BCF   0x06,COL1
                        ;  PutKey(0x00);                 // If no key were pressed
        MOVLW .0
        CALL  PutKey
                        ;
                        ;_FOUND_ONE:
                        ;  COL0=0; COL1=0;
m005    BCF   0x06,COL0
        BCF   0x06,COL1
                        ;
                        ;  Debounce();                   // De-bounce, and return
        CALL  Debounce
                        ;  return 0;                     // de-bounced value.
        RETLW .0
                        ;}
                        ;
                        ;/* The main() function is provided so the program will compile if
                        ;   you do a cut n’ paste from this source into your editor. */
                        ;void main()
                        ;{
main
                        ;  do {
                        ;    ReadKeyboard();             // By executing this line at
m006    CALL  ReadKeyboard
                        ;    switch(retval)              // By executing this line at
        MOVF  retval,W
        XORLW .49
        BTFSC 0x03,Zero_
        GOTO  m006
        XORLW .3
        BTFSC 0x03,Zero_
        GOTO  m006
        XORLW .1
        BTFSC 0x03,Zero_
        GOTO  m006
        XORLW .7
        BTFSC 0x03,Zero_
        GOTO  m006
        GOTO  m006
                        ;                               // certain intervals, keyboard
                        ;                               // will be de-bounced.
                        ;    {
                        ;      case '1': break;         // Test
                        ;      case '2': break;         // Test
                        ;      case '3': break;         // Test
```

```
;      case '4': break;           // Test
;    }
;    /* Do something else while
;       waiting for valid key-press */
;  } while(1);
;}

END
```

NOTES: