Message Dispatch Engine

OVERVIEW

As we all know, the 8-pin PICmicro™ has limited resources. A nice way of using interrupts is for "queuing" events, prioritizing them, or even “buffering” them. This piece of Quick Code handles queuing of events to perform, without using interrupts. With the rest of the code written correctly, this engine is quite easy to use for implementing a kind of “almost-multitasking.” Any function in the entire software may put events in the queue, and a dedicated piece of the main routine will handle the events in its own time.

APPLICATION OPERATION

Queuing is made possible by extensively using “messages.” These messages may be in any form the designer chooses. The supplied code handles messages of byte-size, with no additional information. To use the messaging system, three functions are implemented to form the “engine.” These functions are:

- `void PostMessage(char);`  
  Puts a message in the queue (15 instr)

- `char PeekMessage();`  
  Checks the next message in the queue (10 instr)

- `char GetMessage();`  
  Retrieves/Pulls the next message from the queue (35 instr)

These functions have a predefined RAM area where the messages reside. This RAM area is easiest to cope with, if it has a fixed size. Of course, variable size is possible, but difficult to handle safely. In addition to this RAM area, a global variable for message-count is recommended for fast execution of the `PostMessage(char)` function.

The functions are fairly small, and therefore suitable for the entire range of controllers. On a small controller (i.e., PIC12C508/9), a queue with the size of 8 bytes + 1 byte, for message count, should be quite sufficient.

This is of course extremely dependent on how often messages are posted versus how often messages are retrieved.

Another version of the message dispatch engine may include additional information along with the messages. If the functions were written to handle WORD messages, the high byte could be the message, and the low byte could be message-dependent information (i.e., which key was pressed, temperature, error-message, etc.).

This system may appear to be slow, causing error messages to be detected too late. The quick error-messaging can be implemented by using an exception handler. This can be a global variable that contains the exception information. `PeekMessage()` and `GetMessage()` should override the normal message queue reading, by returning a message that could be `MDE_EXCEPTION`, if an exceptions exists. This will ensure that errors are always handled first.

EXAMPLE USAGE

Let’s say you have implemented some sort of button debounce in your application. When the debounce routine decides that a valid keypress has occurred, it can post a message to inform the main routine about it with a C-source line like:

```c
PostMessage(MDE_KEYPRESS);
```

And/or, you may have a routine you call frequently to check for a temperature threshold. When the temperature is too high, a `PostMessage(MDE_HIGTEMP)` could be executed. A nice way to queue events would be to have several routines initiating some sort of action. If you write the rest of the code to make extensive use of the engine, it may reduce the amount of overhead context-switching and variable space.

MICROCHIP TOOLS USED

Assembler/Compiler version

CC5X C-compiler v2.1H

Microchip Technology Inc., has been granted a nonexclusive, worldwide license to reproduce, publish and distribute all submitted materials, in either original or edited form. The author has affirmed that this work is an original, unpublished work and that he/she owns all rights to such work. All
APPENDIX A: SOURCE CODE

A.1 CC5X v2.1H listing

#include <..\.cc5x\12c508.h> // Define controller

#pragma ramdef 0x10:0x17 remove // RAM for message-queue
#define MDEloc 0x10 // Set start-address
#define MDEsize 8 // Set size

/* Messages */
#define MDE_NONE 0x00 // No message
#define MDE_KEYPRESS 0x01 // Key pressed
#define MDE_TEMPHIGH 0x02 // Temperature high

char MDEcount; // Messages in queue
char RetVal; // Return value from functions.

/* Function for looking at the next message */
char PeekMessage()
{
    if (MDEcount) // Any messages ?
    {
        FSR=MDEloc; // Set first address
        RetVal=INDF0; // Get message
        return 0x00; // Done
    }
    RetVal=MDE_NONE; // Set to "No message"
    return 0x00; // return
}

/* Function for looking at next message, and remove it from the message queue */
char GetMessage()
{
    char Z,m;
    if (MDEcount) // Any messages ?
    {
        FSR=MDEloc; m=INDF0; // Save message
        if (MDEcount>1) // More than one message in queue ?
        {
            Z=MDEloc+MDEcount;
            #ifdef _12C508
            Z|=0xE0; // Compensate for 5-bit FSR
            #endif
            #ifdef _12C509
            Z|=0xC0; // Compensate for 6-bit FSR
            #endif
            for (FSR=MDEloc;FSR<Z;FSR++)
            { /* Run through all present messages */
                FSR++; // Get next message
                Z=INDF0; // Put it
            }
        }
        MDEcount--; // Fix message counter
        RetVal=m; // Set return value
        return 0x00; // Return
    }
    RetVal=MDE_NONE; // Set #No message
return 0x00;  // Return
}

/* This function will put a message in the end of the queue */
void PostMessage(char m)
{
    if (MDEcount>=MDEsize)  // Queue full ?
    { MDEcount=MDEsize;  // Make sure its correct
        return;  // Done
    }

    FSR=MDEloc+MDEcount;  // Point to end of queue
    INDF0=m;  // Insert message
    MDEcount++;  // Increase count
}

/* Main routine to show usage */
void main()
{
    MDEcount=0;  // Zero message-counter

    do {
        PeekMessage();  // Check message. Use this form
        // if you want to remove the
        // message after it has been
        // processed.

        GetMessage();  // Check message and remove.
        // Use this form if you want to
        // Remove the message before
        // processing it.

        switch(RetVal)  // Handle message
        {
            case MDE_KEYPRESS:  // A key has been de-bounced and approved
                nop();  // Handle keypress
                break;
            case MDE_TEMPHIGH:  // Temperature too high
                nop();
                break;
        }

        /* if (!error) GetMessage();  // Insert this if "PeekMessage()"
        // is the form you want to use
        // when checking messages. */

    } while(1);  // End of loop

    /* Do some other tasks that may post */
    /* new messages like : */
    PostMessage(MDE_KEYPRESS);  // Insert a message
} while(1);  // End of loop
A.2  **ASM-code generated by CC5X v2.1H**:

; CC5X Version 2.1H, Copyright (c) B. Knudsen Data
; C compiler for the PIC16CXX microcontroller family

processor 12C508

INDF0 EQU 0x00
FSR EQU 0x04
Carry EQU 0
Zero_ EQU 2
MDEcount EQU 0x09
RetVal EQU 0x0A
z EQU 0x07
m EQU 0x08
m_2 EQU 0x07

GOTO main

; FILE C:\MICROCHI\PROGRAM\MPLAB\TEST\dsp.c
  INCLUDE "..\..\cc5x\12c508.h"
  ;
  ;#pragma ramdef 0x10:0x17 remove
  ;#define MDEloc 0x10
  ;#define MDEsize 8
  ;
  ;#define MDE_NONE 0x00
  ;#define MDE_KEYPRESS 0x01
  ;#define MDE_TEMPHIGH 0x02
  ;
  ;char MDEcount;
  ;charRetVal;
  ;
  ;char PeekMessage()
  ;{

PeekMessage
  ; if (MDEcount)
  MOVF MDEcount,1
  BTFSC 0x03,Zero_
  GOTO m001
  ;
  ; FSR=MDEloc;
  MOVLW .16
  MOVWF FSR
  ; RetVal=INDF0;
  MOVF INDF0,W
  MOVWF RetVal
  RETLW .0
  ;
  ;
  ; RetVal=MDE_NONE;

m001 CLRF RetVal
  ; return 0x00;  // no messages
  RETLW .0
  ;
  ;
  ;

GetMessage
  ; char z,m;
  ;
; if (MDEcount)
MOVF  MDEcount,1
BTFSC 0x03,Zero_
GOTO  m004

; 
;   FSR=MDEloc; m=INDF0;
MOVWF FSR
MOVF  INDF0,W
MOVWF m
; if (MDEcount>1)
MOVWF .2
SUBWF MDEcount,W
BTFSC 0x03,Carry
GOTO  m003

; 
;   Z=MDEloc+MDEcount;
MOVLW .16
ADDWF MDEcount,W
MOVLW .224
IORWF Z,1
#ifdef _12C508
;   Z|=0xE0;
MOVILW 2
#endif
#ifdef _12C509
;   Z|=0xC0;
#endif
; for (FSR=MDEloc;FSR<Z;FSR++)
MOVLW .16
MOVWF FSR
m002
MOVF  Z,W
MOVF Z,W
SUBWF FSR,W
BTFSC 0x03,Carry
GOTO  m003

; 
;   FSR++; Z=INDF0; // get next message
INCF FSR,1
MOVF  INDF0,W
MOVWF Z
; FSR--; INDF0=Z; // put current
DECF FSR,1
MOVW  INDF0
; }
INCF FSR,1
GOTO  m002
; }
; MDEcount--;
m003
DECF  MDEcount,1
; 
; RetVal=m;
MOVF m,W
MOVWF RetVal
RETLW .0
; return 0x00;
; }
; 
; RetVal=MDE_NONE;
m004
CLRWF RetVal
; return 0x00; // no messages
RETLW .0
; }
; void PostMessage(char m)
{PostMessage
MOVWF m_2
    ; if (MDEcount>=MDEsize)
MOVWF MDEcount
SUBWF MDEcount,W
BTFSS 0x03,Carry
GOTO m005
    ; { MDEcount=MDEsize;
MOVLW .8
MOVWF MDEcount
RETLW .0
    ; return;
    }
    ; FSR=MDEloc+MDEcount;
m005 MOVLW .16
ADDWF MDEcount,W
MOVWF FSR
    ; INDF0=m;
MOVF m_2,W
MOVWF INDF0
    ; MDEcount++;
INCF MDEcount,1
    ;
RETLW .0
    ; void main()
{
main
    ; MDEcount=0;
    ;
    ; do {
    ;    PeekMessage(); // Check message. Use this form
    CALL PeekMessage
        ; // if you want to remove the
        ; // message after it has been
        ; // processed.
        ;    GetMessage(); // Check message and remove.
    CALL GetMessage
        ; // Use this form if you want to
        ; // Remove the message before
        ; // processing it.
        ;    switch(RetVal) // Handle message
        ;    {
        ;      case MDE_KEYPRESS:// A key has been de-bounced and approved
        ;          nop(); // Handle keypress
        m007 NOP
            ;     ; break;
        GOTO m009
        ;     ; case MDE_TEMPHIGH:// Temperature too high
        ;          nop();
        m008 NOP
            ;     ; break;
        ; } /*
        ;    if (!error) GetMessage();
        ;/*
;    /* Do some other tasks that may post */
;    /* new messages like :               */
;    PostMessage(MDE_KEYPRESS);

m009    MOVLW .1
CALL    PostMessage

}    } while(1);
GOTO    m006
}

END