```c
// Keyboard Driver
// Jason Losh

//-----------------------------------------------------------------------------
// Hardware Target
//-----------------------------------------------------------------------------

// 4x4 Keyboard
//   Column 0-3 outputs on PA6, PA7, PD2, PD3 are connected to cathode of diodes whose anode connects to column of keyboard
//   Rows 0-3 inputs connected to PE1, PE2, PE3, PF1 which are pulled high
//   To locate a key (r, c), the column c is driven low so the row r reads as low

//-----------------------------------------------------------------------------
// Device includes, defines, and assembler directives
//-----------------------------------------------------------------------------

#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
#include "kb.h"

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------

#define COL0 (*((volatile uint32_t *)(0x42000000 + (0x400043FC-0x40000000)*32 + 6*4)))
#define COL1 (*((volatile uint32_t *)(0x42000000 + (0x400043FC-0x40000000)*32 + 7*4)))
#define COL2 (*((volatile uint32_t *)(0x42000000 + (0x400073FC-0x40000000)*32 + 2*4)))
#define COL3 (*((volatile uint32_t *)(0x42000000 + (0x400073FC-0x40000000)*32 + 3*4)))
#define ROW0 (*((volatile uint32_t *)(0x42000000 + (0x400243FC-0x40000000)*32 + 1*4)))
#define ROW1 (*((volatile uint32_t *)(0x42000000 + (0x400243FC-0x40000000)*32 + 2*4)))
#define ROW2 (*((volatile uint32_t *)(0x42000000 + (0x400243FC-0x40000000)*32 + 3*4)))
#define ROW3 (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)*32 + 1*4)))

#define KB_BUFFER_LENGTH 16
#define KB_NO_KEY -1
char keyboardBuffer[KB_BUFFER_LENGTH];
bool debounceRequest = false;
uint8_t debounceCount = 0;
uint8_t keyboardReadIndex = 0;
uint8_t keyboardWriteIndex = 0;

//-----------------------------------------------------------------------------
// Subroutines
//-----------------------------------------------------------------------------

// Non-blocking function called to drive a selected column low for readout
void setKeyboardColumn(int8_t col)
{
    COL0 = col != 0;
    COL1 = col != 1;
    COL2 = col != 2;
    COL3 = col != 3;
}

```

```c
// Non-blocking function called to drive all selected column low for readout
void setKeyboardAllColumns()
{
    //COL0 = COL1 = COL2 = COL3 = 0;
    COL0 = 0;
    COL1 = 0;
    COL2 = 0;
    COL3 = 0;
}

// Non-blocking function called to determine is a key is pressed in the selected column
int8_t getKeyboardRow()
{
   int8_t row = KB_NO_KEY;
   if (!ROW0) row = 0;
   if (!ROW1) row = 1;
   if (!ROW2) row = 2;
   if (!ROW3) row = 3;
   return row;
}

// Non-blocking function called by the keyboard ISR to determine if a key is pressed
int8_t getKeyboardScanCode()
{
    uint8_t col = 0;
    int8_t row;
    int8_t code = KB_NO_KEY;
    bool found = false;
    while (!found && (col < 4))
    {
        setKeyboardColumn(col);
        waitMicrosecond(1);
        row = getKeyboardRow();
        found = row != KB_NO_KEY;
        if (found)
            code = row << 2 | col;
        else
            col++;
    }
    return code;
}

// 5ms keyboard timer interrupt used for key detection and debouncing
void keyboardIsr()
{
    bool full;
    int8_t code;
    // Handle key press
    if (!debounceRequest)
    {
        code = getKeyboardScanCode();
        if (code != KB_NO_KEY)
        {
            full = ((keyboardWriteIndex+1) % KB_BUFFER_LENGTH) == keyboardReadIndex;
            if (!full)
            {
                keyboardBuffer[keyboardWriteIndex] = code;
                keyboardWriteIndex = (keyboardWriteIndex + 1) % KB_BUFFER_LENGTH;
            }
            debounceRequest = true;
        }
    }
    // Handle debounce
    else
    {
        setKeyboardAllColumns();
        waitMicrosecond(1);
        if (getKeyboardRow() != KB_NO_KEY)
            debounceCount = 0;
        else
        {
            debounceCount ++;
            if (debounceCount == 10)
            {
                debounceCount = 0;
                debounceRequest = false;
            }
        }
    }
    TIMER1_ICR_R = TIMER_ICR_TATOCINT;
}

// Non-blocking function called by the user to determine if a key is present in the buffer
bool kbhit()
{
    return (keyboardReadIndex != keyboardWriteIndex);
}

// Blocking function called by the user to get a keyboard character
char getKey()
{
    const char keyCap[17] = {"123A456B789C*0#D"};
    while (!kbhit());
    uint8_t code = keyboardBuffer[keyboardReadIndex];
    keyboardReadIndex = (keyboardReadIndex + 1) % KB_BUFFER_LENGTH;
    return (char)keyCap[code];
}


```