```c
#include <stdio.h>          // xil_printf and so forth.
#include <stdint.h>         // uint32_t and so forth.

#include "xgpio.h"          // Provides access to PB GPIO driver.
#include "platform.h"       // Enables caching and other system stuff.
#include "mb_interface.h"   // provides the microblaze interrupt enables, etc.
#include "xintc_l.h"        // Provides handy macros for the interrupt controller.

#include "clock/clock.h"
#include "debouncer/debouncer.h"

XGpio gpPB;             // This is a handle for the push-button GPIO block.
u8 counter = 0;         // tens of milliseconds
u8 refresh_counter = 0; // screen refresh counter
u8 bouncing = 0;        // whether or not the button is in debounce mode

// This is invoked in response to a timer interrupt.
// It does 2 things: 1) debounce switches, and 2) advances the time.
void timer_interrupt_handler() {
    uint32_t seconds = 0;
    uint32_t minutes = 0;
    uint32_t hours = 0;

    if (++counter == ONE_SECOND && !in_auto_increment_mode()){
        // clear the counter, it's been a second
        counter = 0;

        // increment the clock
        incrementClock();
    } else if (++refresh_counter == FAST_COUNT_MAX) {
        // clear the counter, it's time to refresh
        refresh_counter = 0;

        // grab the clock values so we can print them out
        getClock(&seconds, &minutes, &hours);

        // make sure to backspace so clock changes in place
        xil_printf("\b\b\b\b\b\b\b\b%02d:%02d:%02d", hours, minutes, seconds);
    }

    // let the bouncer know an interrupt occurred so it can debounce the button
    if (bouncing) tick_bouncer();

}

// This is invoked each time there is a change in the button state (result of a push or a
bounce).
void pb_interrupt_handler() {
    // Clear the GPIO interrupt.

    // Turn off all PB interrupts for now.
    XGpio_InterruptGlobalDisable(&gpPB);
    // Get the current state of the buttons.
    u32 currentButtonState = XGpio_DiscreteRead(&gpPB, 1);

    bouncing = bouncer(currentButtonState);

    // Ack the PB interrupt.
    XGpio_InterruptClear(&gpPB, 0xFFFFFFFF);
    // Re-enable PB interrupts.
    XGpio_InterruptGlobalEnable(&gpPB);
}

/*****************************************************************************
 * Main interrupt handler, queries interrupt controller to see what peripheral
 * fired the interrupt and then dispatches the corresponding interrupt handler.
 * This routine acks the interrupt at the controller level but the peripheral
 * interrupt must be ack'd by the dispatched interrupt handler.
 ****************************************************************************/
void interrupt_handler_dispatcher(void* ptr) {
    int intc_status = XIntc_GetIntrStatus(XPAR_INTC_0_BASEADDR);

    // Check the FIT interrupt first.
    if (intc_status & XPAR_FIT_TIMER_0_INTERRUPT_MASK){
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, XPAR_FIT_TIMER_0_INTERRUPT_MASK);
        timer_interrupt_handler();
    }

    // Check the push buttons.
    if (intc_status & XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK){
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK);
        pb_interrupt_handler();
    }
}

int main (void) {
    init_platform();
    // Initialize the GPIO peripherals.
    XGpio_Initialize(&gpPB, XPAR_PUSH_BUTTONS_5BITS_DEVICE_ID);
    // Set the push button peripheral to be inputs.
    XGpio_SetDataDirection(&gpPB, 1, 0x0000001F);
    // Enable the global GPIO interrupt for push buttons.
    XGpio_InterruptGlobalEnable(&gpPB);
    // Enable all interrupts in the push button peripheral.
    XGpio_InterruptEnable(&gpPB, 0xFFFFFFFF);

    microblaze_register_handler(interrupt_handler_dispatcher, NULL);
    XIntc_EnableIntr(XPAR_INTC_0_BASEADDR,
            (XPAR_FIT_TIMER_0_INTERRUPT_MASK |
XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK));
    XIntc_MasterEnable(XPAR_INTC_0_BASEADDR);
    microblaze_enable_interrupts();

    while(1);  // Program never ends.

    cleanup_platform();

    return 0;
}
```

```c
/*
 * debouncer.h
 *
 *  Created on: Sep 10, 2015
 *      Author: superman
 */

#ifndef DEBOUNCER_H_
#define DEBOUNCER_H_

#include <stdint.h>
#include "../clock/clock.h"

// button masks
#define BTN_MIN_MASK    0x01
#define BTN_S_MASK      0x02
#define BTN_DOWN_MASK   0x04
#define BTN_HR_MASK     0x08
#define BTN_UP_MASK     0x10

#define BTN_TIME_MASK   (BTN_HR_MASK | BTN_MIN_MASK | BTN_S_MASK)
#define BTN_INC_MASK    (BTN_UP_MASK | BTN_DOWN_MASK)

// timing constants
#define MAX_DEBOUNCE    5
#define ONE_SECOND      100
#define FAST_COUNT_MAX  20

// Start the button debouncer
uint8_t bouncer(uint32_t newButtonState);

// Returns whether or not we are auto incrementing
uint8_t in_auto_increment_mode();

// Tell the bouncer that an interrupt has occured
// so it can debounce the button
void tick_bouncer();

#endif /* DEBOUNCER_H_ */
```

```c
#include "debouncer.h"

static volatile uint32_t oldButtonStates = 0;
static volatile uint32_t debounced = 1;

static volatile uint8_t bouncer_counter = 0;
static volatile uint32_t auto_counter = 0;
static volatile uint8_t inc = 0;
static volatile uint8_t fast_count = 0;
static volatile uint8_t auto_mode = 0;

static volatile uint32_t time_pressed;
static volatile uint32_t updown_pressed;

static void inc_time();

// ------------------------------------------------------------------------------

void tick_bouncer(){
    // keep track of how many interrupts have happened
    // so we know when to enable auto-increment mode
    auto_counter++;

    // if I haven't already been debounced, then debounce me!
    if(!debounced){
        bouncer_counter++;
        if (bouncer_counter == MAX_DEBOUNCE) {
            // now that I'm debounced, let the world know.
            debounced = 1;

            // also, I can now go into the auto-increment mode
            auto_mode = 1;

            // also, if a time button and up/down button is pressed,
            // change time.
            if (inc) inc_time();
        }
    } else if (auto_counter > ONE_SECOND && inc) {
        // because I'm already debounced, now check if the button
        // has been held for longer than a second. If it has, and
        // the appropriate buttons are pressed, enable the fast count.
        fast_count++;
        if(fast_count == FAST_COUNT_MAX){
            fast_count = 0;
            inc_time();
        }
    }
}

// ------------------------------------------------------------------------------

uint8_t bouncer(uint32_t currentButtonStates){
    // capture the button states to use after buttons have been debounced
    oldButtonStates = currentButtonStates;

    // Decide if a time button is being pressed...
    time_pressed = currentButtonStates & BTN_TIME_MASK;
    // ...or the up/down button.
    updown_pressed = currentButtonStates & BTN_INC_MASK;

    // if a time button and an up/down is pressed,
    // then we are okay to change the time.
    inc = time_pressed && updown_pressed;

    // reset our counters and modes
    bouncer_counter = 0;
    auto_counter = 0;
    debounced = 0;
    auto_mode = 0;

    // Let the caller know to start debouncing,
    // but only if it was an up/down button.
    // (nothing happens unless an up/down button is pressed)
    return (updown_pressed) ? 1 : 0;
}

// ---------------------------------------------------------------------------

uint8_t in_auto_increment_mode() {
    return auto_mode;
}

// ---------------------------------------------------------------------------
// Private Methods
// ---------------------------------------------------------------------------

void inc_time(){
    // increment/decrement the appropriate time unit
    // according to the pressed buttons

    if (oldButtonStates & BTN_UP_MASK) {
        if (oldButtonStates & BTN_HR_MASK)  incrementHours();
        if (oldButtonStates & BTN_MIN_MASK) incrementMinutes();
        if (oldButtonStates & BTN_S_MASK)   incrementSeconds();
    }

    if (oldButtonStates & BTN_DOWN_MASK){
        if (oldButtonStates & BTN_HR_MASK)  decrementHours();
        if (oldButtonStates & BTN_MIN_MASK) decrementMinutes();
        if (oldButtonStates & BTN_S_MASK)   decrementSeconds();
    }
}
```

```c
#ifndef CLOCK_H
#define CLOCK_H

#include <stdint.h>
#include <stdio.h>

#define MAX_SECONDS 60
#define MAX_MINUTES 60
#define MAX_HOURS   24

// increments the clock, handling hours, minute, second rollovers
void incrementClock();

// increment/decrement individual time units
uint32_t incrementSeconds();
uint32_t incrementMinutes();
uint32_t incrementHours();
void decrementSeconds();
void decrementMinutes();
void decrementHours();

// Get the current time from the clock
void getClock(uint32_t *s, uint32_t *m, uint32_t *h);

#endif
```

```c
#include "clock.h"
#include <stdio.h>

static uint32_t seconds = 0;
static uint32_t minutes = 0;
static uint32_t hours = 0;

void incrementClock() {
    if (!incrementSeconds()) {
        if (!incrementMinutes()) {
            incrementHours();
        }
    }
}

// ------------------------------------------------------------------------------

void getClock(uint32_t *s, uint32_t *m, uint32_t *h) {
    *s = seconds;
    *m = minutes;
    *h = hours;
}

// ------------------------------------------------------------------------------

uint32_t incrementSeconds() {
    if (++seconds == MAX_SECONDS) {
        seconds = 0;
    }
    return seconds;
}

// ------------------------------------------------------------------------------

void decrementSeconds() {
    if (--seconds > MAX_SECONDS) {
        seconds = MAX_SECONDS - 1;
    }
}

// ------------------------------------------------------------------------------

uint32_t incrementMinutes() {
    if (++minutes == MAX_MINUTES) {
        minutes = 0;
    }
    return minutes;
}

// ------------------------------------------------------------------------------

void decrementMinutes() {
    if (--minutes > MAX_MINUTES) {
        minutes = MAX_MINUTES - 1;
    }
}

// ------------------------------------------------------------------------------
```

```c
uint32_t incrementHours() {
    if (++hours == MAX_HOURS) {
        hours = 0;
    }
    return hours;
}

// ---------------------------------------------------------------------------

void decrementHours() {
    if (--hours > MAX_HOURS) {
        hours = MAX_HOURS - 1;
    }
}
```