Tech Screening

Kenkou GmbH

January 31, 2018

1 Code Review

Following is a code snippet from the Nordic nRF5 SDK V 14.2 and a few challenges. Documentation in the Official Infocenter is missing for this function. The documentation found with the source code also has a few mistakes.

```
/**@brief     Function for checking if the CCCD of DFU Control point is configured for Notification.
 *
 * @details   This function checks if the CCCD of DFU Control Point characteristic is configured
 *            for Notification by the DFU Controller.
 *
 * @param[in] p_dfu DFU Service structure.
 *
 * @return    True if the CCCD of DFU Control Point characteristic is configured for Notification.
 *            False otherwise.
 */
uint32_t ble_dfu_init(ble_dfu_t * p_dfu)
{
    ble_uuid_t service_uuid;
    uint32_t   err_code;

    ASSERT(p_dfu != NULL);

    m_conn_handle = BLE_CONN_HANDLE_INVALID;

    BLE_UUID_BLE_ASSIGN(service_uuid, BLE_DFU_SERVICE_UUID);

    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
                                        &service_uuid,
                                        &(p_dfu->service_handle));
    VERIFY_SUCCESS(err_code);

    ble_uuid128_t const base_uuid128 =
    {
        {
            0x50, 0xEA, 0xDA, 0x30, 0x88, 0x83, 0xB8, 0x9F,
            0x60, 0x4F, 0x15, 0xF3,  0x00, 0x00, 0xC9, 0x8E
        }
    };

    err_code = sd_ble_uuid_vs_add(&base_uuid128, &p_dfu->uuid_type);
    VERIFY_SUCCESS(err_code);

    err_code = dfu_pkt_char_add(p_dfu);
    VERIFY_SUCCESS(err_code);

    err_code = dfu_ctrl_pt_add(p_dfu);
    VERIFY_SUCCESS(err_code);

    m_flags |= DFU_BLE_FLAG_SERVICE_INITIALIZED;

    return NRF_SUCCESS;
}
```

1. Identify issues with the block comment describing the *ble_dfu_init* function.
2. Suggest corrections to the comment provided the source code of the function.
3. Describe what you expect each function call within *ble_dfu_init* to do.
4. List any problems you see with the way this function was written.

2 Code Challenge

A skeleton program has been provided. It contains a shell program which monitors keypresses. Each press of the <ENTER> key is acknowledged as a tap, and the time of each tap is pulled from the system with nanosecond precision. The header file kenkou_triple_tap.h contains a function which requires implementation.

The code challenge is to provide a working implementation of this function.

A weak implementation is already in the code skeleton. This allows one to compile and test their development environment. So, before starting the function implementation, please try compiling using the provided makefile, and running the produced bin/.out file.

Note: You retain all rights to code submitted as part of the code challenge. We will not use any code submitted for company projects, and will only be using it for evaluational purposes.

2.1 Requirements

- The contract for *isTripleTap* must be fulfilled
- *isTripleTap* must be unit tested (use ThrowTheSwitch's Unity, or a test framework of your choice).
- Instructions to run the unit test and sample program must be provided.
- The unit test and the program must compile with GCC.
- Source code should be tracked in git, but not publicly.
- The repo should be submitted as a gzip file.

2.2 Recommendations

- Create smaller functions which will be called inside isTripleTap to keep the code readable.

2.3 Constraints

- Only standard libraries should be used, unit testing frameworks are excepted.

2.4 Contract for isTripleTap()

```
/**@brief     Function to detect triple tap for various channels.
 *
 * @details   For a given channel, this function checks a relative time in seconds and ms.
 *            if the current and two previous calls to the channel fall within TRIPLE_TAP_MAX_DUR,
 *            and are each a maximum of TRIPLE_TAP_MAX_DELAY apart from the last,
 *            the function will return True.
 *            (Optional) if true internally trigger a callback if set by
 *            setTripleTapCallback(uint8t channel, void* callback)
 *
 *Conditions
 * @pre
 *           TRIPLE_TAP_MAX_DELAY is set before compile time
 *           TRIPLE_TAP_MAX_DUR is set at least 2x TRIPLE_TAP_MAX_DELAY before compile time
 * (optional)tripleTapCallback is set with setTripleTapCallback
 *
 * @post
 *           True or false have been returned.
 *           All recorded taps are cleared when a triple tap state is encountered
 *               (to prevent a following 4th tap from triggering another triple tap).
 *
 * @inv
 *           TRIPLE_TAP_DUR must be at least 2x TRIPLE_TAP_DELAY
 *               otherwise a warning is provided at compile time
 *
 *Parameters
 * @param[in] uint8_t tripleTapChannel, time_t epochTime, uint16_t additionalMillis.
 *
 * @return    True if the last three calls to the specified channel qualify as a triple tap,
 *                and no triple tap has been executed for the oldest two.
 *            False otherwise.
 */
bool isTripleTap(uint8_t channel, time_t epochTime, uint16_t additionalMillis) . . .
```