BEÁGYAZOTT RENDSZEREK (C)

3. forduló

BOSCH
Invented for life

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladatlaphoz

Kérjük, hogy a feladatlap indítása előtt mindenképp olvasd el az alábbi útmutatót:

> Amennyiben olyan kategóriában játszol, ahol van csatolmány, de hibába ütközöl a letöltésnél, ott valószínűleg a vírusirtó korlátoz, annak ideiglenes kikapcsolása megoldhatja a problémát. (Körülbelül minden 3000. letöltésnél fordul ez elő.)

> Helyezéseket a 4. forduló után mutatunk, százalékos formában: adott kategóriában a TOP 20-40-60%-hoz tartozol.
>
> A feltűnően rövid idő alatt megoldott feladatlapok kizárást vonnak maguk után, bármilyen más gyanús esetben fenntartjuk a jogot a forduló érvénytelenítésére!

Mekk Mester szeretné infra távirányítóval vezérelni az audio lejátszóját. Ehhez a NEC IR protokollt szeretné implementálni. Infra Vevőnek a TSOP1738-at választotta, aminek a jelét a TIM9-es timer segítségével Input capture módban szeretné feldolgozni.

TSOP1738 : https://www.batronix.com/pdf/tsop17xx.pdf

IR Protocol: https://sibotic.files.wordpress.com/2013/12/adoh-necinfraredtransmissionprotocol-281113-1713-47344.pdf

Jó versenyt kívánunk!

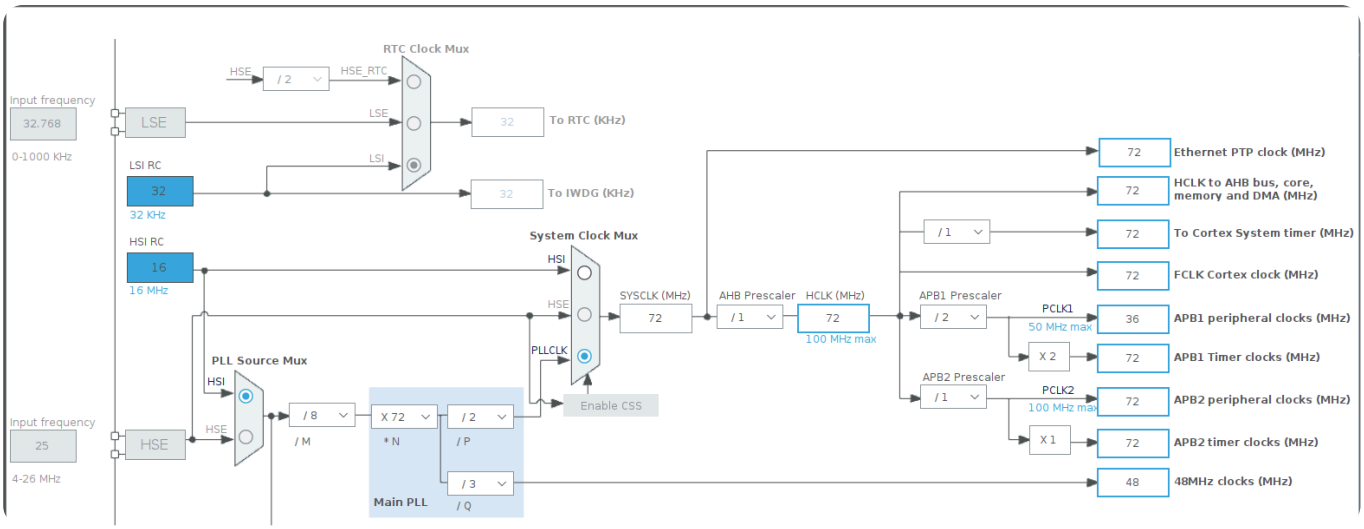
1. feladat 3 pont

Melyik csatornáját kell a TIM9 timernek ahhoz használni, hogy a PORTA PA2-es láb legyen a bemenet? (Csak a számot írd be, tehát Channel 0 esetén 0-át)

Válasz

2. feladat 3 pont

A mikrovezérlő órajel beállításait a következő ábrán láthatod. A TIM9 Internal clock division 4-re állítása esetén mekkora értékre kell beállítani a Prescaler(PSC) értékét, ha 0.01ms felbontással szeretnénk mérni a jelet?



Válasz

3. feladat 8 pont

A TIM9 Interruptjában szeretnénk kiszámítani a két megszakítás között eltelt időt, amit egy RTOS queue-ba küldünk a jelfeldolgozó egységnek. Válaszd ki a helyes megoldásokat (A TIM_CHANNEL_X az egyes feladatban megadott csatorna)!

Válaszok

```
#include "Log.h"
#include "cmsis_os.h"
#include "main.h"

#define LOG_Tag "IR"

extern TIM_HandleTypeDef htim9;
static osThreadId infraReceiverTaskHandle;
static xQueueHandle infraMessageQueue = 0;

void InitInfraReceiverdDetector() {
    infraMessageQueue = xQueueGenericCreate(100, sizeof(uint16_t), 0);

    osThreadDef(infraReceiverTask, InfraReceiverTask, osPriorityLow, 0, 128);
    infraReceiverTaskHandle = osThreadCreate(osThread(infraReceiverTask), NULL

    HAL_TIM_IC_Start_IT(&htim9, TIM_CHANNEL_X);
}

static void InfraReceiverTask(void const* argument) {
    uint16_t elapsedTime;

    for (;;) {
        if (xQueueReceive(infraMessageQueue, &elapsedTime, osWaitForever) == p
            LOG_I(LOG_Tag, "IR Measurement: %ul", elapsedTime);
            // decodeBits(elapsedTime); // Itt fogjuk feldolgozni a mérést
        }
    }
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    uint16_t currentCapturedValue = 0;

    if (htim == &htim9) {
        currentCapturedValue = (uint16_t)HAL_TIM_ReadCapturedValue(htim, TIM_C
        htim->Instance->CNT = 0;

        if (infraMessageQueue) {
            xQueueSendFromISR(infraMessageQueue, &currentCapturedValue, &xHigh
            portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
```

```
        }
    }
}
```

```
#include "Log.h"
#include "cmsis_os.h"
#include "main.h"

#define LOG_Tag "IR"

extern TIM_HandleTypeDef htim9;
static osThreadId infraReceiverTaskHandle;
static xQueueHandle infraMessageQueue = 0;

void InitInfraReceiverdDetector() {
    infraMessageQueue = xQueueGenericCreate(100, sizeof(uint16_t), 0);

    osThreadDef(infraReceiverTask, InfraReceiverTask, osPriorityLow, 0, 128);
    infraReceiverTaskHandle = osThreadCreate(osThread(infraReceiverTask), NULL

    HAL_TIM_IC_Start_IT(&htim9, TIM_CHANNEL_X);
}

static void InfraReceiverTask(void const* argument) {
    uint16_t elapsedTime;

    for (;;) {
        if (xQueueReceive(infraMessageQueue, &elapsedTime, osWaitForever) == p
            LOG_I(LOG_Tag, "IR Measurement: %ul", elapsedTime);
            // decodeBits(elapsedTime); // Itt fogjuk feldolgozni a mérést
        }
    }
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    static uint16_t prevCapturedValue = 0;
    uint16_t currentCapturedValue = 0;
    uint16_t diff;

    if (htim == &htim9) {
        currentCapturedValue = (uint16_t)HAL_TIM_ReadCapturedValue(htim, TIM_C
        diff = currentCapturedValue - prevCapturedValue;
        prevCapturedValue = currentCapturedValue;

        if (infraMessageQueue) {
```

```
            xQueueSendFromISR(infraMessageQueue, &diff, &xHigherPriorityTaskWc
            portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
        }
    }
}
```

```
#include "Log.h"
#include "cmsis_os.h"
#include "main.h"

#define LOG_Tag "IR"

extern TIM_HandleTypeDef htim9;
static osThreadId infraReceiverTaskHandle;
static xQueueHandle infraMessageQueue = 0;

void InitInfraReceiverdDetector() {
    infraMessageQueue = xQueueGenericCreate(100, sizeof(uint16_t), 0);

    osThreadDef(infraReceiverTask, InfraReceiverTask, osPriorityLow, 0, 128);
    infraReceiverTaskHandle = osThreadCreate(osThread(infraReceiverTask), NULL

    HAL_TIM_IC_Start_IT(&htim9, TIM_CHANNEL_X);
}

static void InfraReceiverTask(void const* argument) {
    uint16_t elapsedTime;

    for (;;) {
        if (xQueueReceive(infraMessageQueue, &elapsedTime, osWaitForever) == p
            LOG_I(LOG_Tag, "IR Measurement: %ul", elapsedTime);
            // decodeBits(elapsedTime); // Itt fogjuk feldolgozni a mérést
        }
    }
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    static uint16_t prevCapturedValue = 0;
    uint16_t currentCapturedValue = 0;
    uint16_t diff;

    if (htim == &htim9) {
        currentCapturedValue = (uint16_t)HAL_TIM_ReadCapturedValue(htim, TIM_C
        diff = currentCapturedValue - prevCapturedValue;
```

```
            if (infraMessageQueue) {
                xQueueSendFromISR(infraMessageQueue, &diff, &xHigherPriorityTaskWc
                portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
            }
        }
    }
```

4. feladat 1 pont

Megérkezett a mérési eredmény az RTOS queue-ba. Azonban a feldolgozó egységbe hiba csúszott be. Válaszd ki a jó megoldást!

Válasz

```
#include <stdbool.h>
#include <stdint.h>

#include "Log.h"

#define LOG_Tag "IR"

enum DecoderState { DECODER_STATE_WAIT_FOR_START_BIT, DECODER_STATE_COLLECT_BI

static enum DecoderState decoderState;
static uint8_t decoderBitCount;
static uint32_t decodedData;

static void decodeBits(uint16_t measuredTime) {
    switch (decoderState) {
        case DECODER_STATE_WAIT_FOR_START_BIT:
            if (measuredTime >= 1300 && measuredTime <= 1450) {
                decoderBitCount = 0;
                decodedData = 0;
                decoderState = DECODER_STATE_COLLECT_BITS;
            }
            break;
        case DECODER_STATE_COLLECT_BITS:
            if (measuredTime >= 110 && measuredTime <= 120) {
                decoderBitCount++;
            } else if (measuredTime >= 225 && measuredTime <= 235) {
                decodedData |= 0x01UL << (31 - decoderBitCount);
                decoderBitCount++;
```

```
            } else {
                decoderState = DECODER_STATE_WAIT_FOR_START_BIT;
            }
            if (decoderBitCount == 32) {
                bool addressOk = (((decodedData >> 24) ^ (decodedData >> 16))
                bool commandOk = (((decodedData >> 8) ^ decodedData) & 0xFF) =

                if (addressOk && commandOk) {
                    LOG_I(LOG_Tag, "IR Decoded successfully, IR= 0x%04X", decc
                }
                decoderState = DECODER_STATE_WAIT_FOR_START_BIT;
            }
            break;
        default:
            break;
    }
}
```

```
#include <stdbool.h>
#include <stdint.h>

#include "Log.h"

#define LOG_Tag "IR"

enum DecoderState { DECODER_STATE_WAIT_FOR_START_BIT, DECODER_STATE_COLLECT_BI

static enum DecoderState decoderState;
static uint8_t decoderBitCount;
static uint32_t decodedData;

static void decodeBits(uint16_t measuredTime) {
    switch (decoderState) {
        case DECODER_STATE_WAIT_FOR_START_BIT:
            if (measuredTime >= 1300 && measuredTime <= 1450) {
                decoderBitCount = 0;
                decodedData = 0;
                decoderState = DECODER_STATE_COLLECT_BITS;
            }
            break;
        case DECODER_STATE_COLLECT_BITS:
            if (measuredTime >= 110 && measuredTime <= 120) {
                decodedData <<= 1;
                decoderBitCount++;
            } else if (measuredTime >= 225 && measuredTime <= 235) {
                decodedData |= 1;
```

```
                decodedData <<= 1;
                decoderBitCount++;
            } else {
                decoderState = DECODER_STATE_WAIT_FOR_START_BIT;
            }
            if (decoderBitCount == 32) {
                bool addressOk = (((decodedData >> 24) ^ (decodedData >> 16))
                bool commandOk = (((decodedData >> 8) ^ decodedData) & 0xFF) =

                if (addressOk && commandOk) {
                    LOG_I(LOG_Tag, "IR Decoded successfully, IR= 0x%04X", decc
                }
                decoderState = DECODER_STATE_WAIT_FOR_START_BIT;
            }
            break;
        default:
            break;
    }
}
```

```
#include <stdbool.h>
#include <stdint.h>

#include "Log.h"

#define LOG_Tag "IR"

enum DecoderState { DECODER_STATE_WAIT_FOR_START_BIT, DECODER_STATE_COLLECT_BI

static enum DecoderState decoderState;
static uint8_t decoderBitCount;
static uint32_t decodedData;

static void decodeBits(uint16_t measuredTime) {
    switch (decoderState) {
        case DECODER_STATE_WAIT_FOR_START_BIT:
            if (measuredTime >= 1300 && measuredTime <= 1450) {
                decoderBitCount = 0;
                decodedData = 0;
                decoderState = DECODER_STATE_COLLECT_BITS;
            }
            break;
        case DECODER_STATE_COLLECT_BITS:
            if (measuredTime >= 110 && measuredTime <= 120) {
                decoderBitCount++;
            } else if (measuredTime >= 225 && measuredTime <= 235) {
```

```
                decodedData |= 0x01UL << (32 - decoderBitCount);
                decoderBitCount++;
            } else {
                decoderState = DECODER_STATE_WAIT_FOR_START_BIT;
            }
            if (decoderBitCount == 32) {
                bool addressOk = (((decodedData >> 24) ^ (decodedData >> 16))
                bool commandOk = (((decodedData >> 8) ^ decodedData) & 0xFF) =

                if (addressOk && commandOk) {
                    LOG_I(LOG_Tag, "IR Decoded successfully, IR= 0x%04X", decc
                }
                decoderState = DECODER_STATE_WAIT_FOR_START_BIT;
            }
            break;
        default:
            break;
    }
}
```

Megoldások beküldése