Daniel Miškovski in Dime Sazdev

Kontrola temperature in svetlobe v bazenu

Mentor: viš. pred. dr. Robert Rozman

Ljubljana, 2024

Vsebinsko kazalo

1. Ideja projekta

Projekt predstavlja enostaven sistem za nadzor razsvetljave in temperature bazena, ki je izveden z uporabo mikrokrmilnika STM32H750B. Za komunikacijo in nadzor je uporabljen LCD zaslon, na katerem je uporabniški vmesnik realiziran z uporabo sistema TouchGFX. Uporabnik lahko prek uporabniškega vmesnika na zaslonu nadzoruje osvetlitev in temperaturo bazena.

Ker nimamo dejanskega bazena, rezultat prikažemo z LED diodo na strani razvojne plošče. LED dioda se prižge ali ugasne, ko uporabnik pritisne gumb za vklop/izklop na zaslonu. Hitrost utripanja LED diode se poveča ali zmanjša glede na izbrano temperaturo za bazen.

Uporabila sva sledeče komponente:

- Mikrokrmilnik STM32H750B.

- LCD zaslon z uporabniškim vmesnikom TouchGFX.

- LED dioda (na razvojni plošči).

S tem sistemom lahko uporabnik simulira nadzor osvetlitve in temperature bazena, kar je prikazano z enostavnim vmesnikom in fizičnim odzivom LED diode.

2. Načrtovanje in izdelava uporabniškega vmesnika

Naš projekt uporablja STM32H750B mikrokrmilnik in TouchGFX za realizacijo nadzornega sistema za osvetlitev in temperaturo bazena. Grafični uporabniški vmesnik na LCD zaslonu omogoča uporabniku, da z zaslonskimi gumbi vklaplja ali izklaplja luči v bazenu in nastavlja temperaturo. Rezultate simuliramo z LED diodo na razvojni plošči, ki se prižge ali ugasne glede na ukaze iz vmesnika.

2.1 Gradnja Uporabniškega Vmesnika

Uporabniški vmesnik je zasnovan z uporabo TouchGFX in je sestavljen na eni strani zaslona. Ta vključuje gumbe za nadzor luči ("Pool Lights") in za nastavitev temperature bazena ("Pool Temperature"). Po pritisku gumba se prikaže nov zaslon za prilagoditev temperature, kjer lahko uporabnik izbere želeno temperaturo.

Ko uporabnik pritisne gumb za vklop luči, se LED dioda na razvojni plošči prižge ali ugasne. Če uporabnik prilagodi temperaturo bazena, LED dioda začne utripati hitreje ali počasneje, odvisno od izbrane temperature.

2.2 Implementacija Logike

TouchGFX uporablja arhitekturni vzorec MVP (Model-View-Presenter), ki omogoča ločitev logike uporabniškega vmesnika in zalednega dela. `Model` skrbi za zaledno logiko, kot je nadzor LED diode, `View` predstavlja uporabniški vmesnik, `Presenter` pa deluje kot povezava med njima.

Model.cpp

```
#include <gui/model/Model.hpp>
#include <gui/model/ModelListener.hpp>

Model::Model()
    : modelListener(0), poolLightState(false), poolTemperature(18)
{
}

void Model::tick()
{
}

void Model::userSetPoolLight(bool state)
{
    poolLightState = state;
}

void Model::userSetTemperature(uint32_t temp)
{
    poolTemperature = temp;
}
```

Ta datoteka vsebuje implementacijo razreda **Model**, ki je del MVC (Model-View-Controller) arhitekture za aplikacijo.

- Konstruktor **Model::Model()**: Inicializira član razreda modelListener na 0, stanje luči bazena (poolLightState) na false, in temperaturo bazena (poolTemperature) na 18 stopinj.
- Metoda **void Model::tick()**: Trenutno ne izvaja nobene operacije. Lahko se uporabi za osveževanje podatkov ali izvajanje periodičnih nalog v prihodnje.

- Metoda **void Model::userSetPoolLight(bool state)**: Nastavi stanje luči bazena na vrednost, ki jo poda uporabnik. Sprejme parameter state, ki je tipa bool.
- Metoda **void Model::userSetTemperature(uint32_t temp)**: Nastavi temperaturo bazena na vrednost, ki jo poda uporabnik. Sprejme parameter temp, ki je tipa uint32_t.

MainView.cpp

```
#include <gui/main_screen/MainView.hpp>

uint8_t state = 0;

MainView::MainView()
{
}

void MainView::setupScreen()
{
}

void MainView::tearDownScreen()
{
}

void MainView::updatePoolLight()
{
    //report change to presenter
    presenter->setPoolLight(onOffButton.getState());

    state = onOffButton.getState();
}

void MainView::setPoolLight(bool state)
{
    onOffButton.forceState(state);
    bulbYellow.setVisible(state);

    onOffButton.invalidate();
    bulbYellow.invalidate();
}
```

Ta datoteka vsebuje implementacijo razreda **MainView**, ki predstavlja pogled glavnega zaslona v aplikaciji.

- Konstruktor **MainView::MainView()**: Inicializira pogled. Trenutno ne izvaja posebnih operacij.
- Metoda **void MainView::setupScreen()**: Nastavi zaslon ob inicializaciji pogleda. Trenutno je prazna, vendar se lahko uporabi za inicializacijo elementov zaslona.

- Metoda **void MainView::tearDownScreen()**: Metoda za čiščenje zaslona ob zapiranju pogleda. Trenutno ne izvaja nobene operacije.
- Metoda **void MainView::updatePoolLight()**: Posodobi stanje luči bazena glede na stanje gumba onOffButton. Spremembo poroča prezenterju in posodobi globalno spremenljivko state z vrednostjo gumba.
- Metoda **void MainView::setPoolLight(bool state)**: Nastavi stanje gumba onOffButton in slike žarnice (bulbYellow) glede na vrednost parametra state. Poskrbi, da se prikazane vrednosti posodobijo z uporabo invalidate().

PoolTempView.cpp

```
#include <gui/pooltemp_screen/PoolTempView.hpp>

volatile int brightness = 0;

PoolTempView::PoolTempView()
{
}

void PoolTempView::setupScreen()
{
}

void PoolTempView::tearDownScreen()
{
}

void PoolTempView::setPoolTemp(uint32_t temp)
{
    //update slider and temperature text
    slider.setValue(temp);
    Unicode::snprintf(tempTextBuffer, 3, "%d", temp);
    tempText.invalidate();
}

void PoolTempView::sliderMoved(int value)
{
    //report to presenter and update text
    presenter->userSetTemperature(value);
    Unicode::snprintf(tempTextBuffer, 3, "%d", value);
    tempText.invalidate();

    brightness = value;
}
```

Ta datoteka vsebuje implementacijo razreda PoolTempView, ki je pogled za prikaz in nastavitev temperature bazena.

- Konstruktor **PoolTempView::PoolTempView()**: Inicializira pogled temperature bazena. Trenutno ne izvaja posebnih operacij.
- Metoda **void PoolTempView::setupScreen()**: Nastavi zaslon ob inicializaciji pogleda. Trenutno je prazna, vendar se lahko uporabi za inicializacijo elementov zaslona.
- Metoda **void PoolTempView::tearDownScreen()**: Metoda za čiščenje zaslona ob zapiranju pogleda. Trenutno ne izvaja nobene operacije.
- Metoda **void PoolTempView::setPoolTemp(uint32_t temp)**: Posodobi drsnik (slider) in besedilo temperature (tempText) glede na dano temperaturo. Parametr temp predstavlja želeno temperaturo in je tipa uint32_t.
- Metoda **void PoolTempView::sliderMoved(int value)**: Ob premiku drsnika ta metoda poroča prezenterju novo nastavljeno vrednost temperature in posodobi besedilo temperature. Posodobi tudi globalno spremenljivko brightness z novo vrednostjo.

main.c

```
extern uint8_t state;
extern volatile int brightness;
void generatePWM() {
    unsigned int highTime, lowTime, totalTime;

    // Calculate high and low times based on ADC value
    totalTime = 255; // Assuming 8-bit PWM
    highTime = totalTime - (brightness); // Map Value inversely to a range of
0 to 255
    lowTime = totalTime - highTime;

    if(state) {
      // PWM signal generation using pin high-low method
            HAL_GPIO_WritePin(GPIOJ, GPIO_PIN_2, GPIO_PIN_RESET);
            osDelay(highTime);
            HAL_GPIO_WritePin(GPIOJ, GPIO_PIN_2, GPIO_PIN_SET);
            osDelay(lowTime);
    } else {
      HAL_GPIO_WritePin(GPIOJ, GPIO_PIN_2, GPIO_PIN_SET);
    }
}
```

Ta funkcija generira PWM (pulzno širinsko modulacijo) signal za nadzor nad svetlostjo luči na podlagi vrednosti brightness in stanja state.

Funkcija **void generatePWM()**:

- Najprej izračuna vrednosti za visok (highTime) in nizek čas (lowTime) glede na vrednost spremenljivke brightness, ki določa svetlost.

- Če je state (stanje) 1 (vklopljeno), funkcija ustvarja PWM signal na GPIO pinih z uporabo funkcij HAL_GPIO_WritePin() in osDelay() za nastavitev in zakasnitev signalov.
- Če je state 0 (izklopljeno), nastavi GPIO pin v stanje SET (kar pomeni, da je luč izklopljena).

main.h

```
#define FRAME_RATE_Pin GPIO_PIN_6
#define FRAME_RATE_GPIO_Port GPIOB
#define RENDER_TIME_Pin GPIO_PIN_7
#define RENDER_TIME_GPIO_Port GPIOB
#define LCD_DE_Pin GPIO_PIN_7
#define LCD_DE_GPIO_Port GPIOD
#define VSYNC_FREQ_Pin GPIO_PIN_3
#define VSYNC_FREQ_GPIO_Port GPIOG
#define LCD_BL_CTRL_Pin GPIO_PIN_0
#define LCD_BL_CTRL_GPIO_Port GPIOK
#define MCU_ACTIVE_Pin GPIO_PIN_6
#define MCU_ACTIVE_GPIO_Port GPIOA
```

Ta koda omogoča dinamično spreminjanje stanja LED diode in hitrosti utripanja na podlagi uporabniških interakcij. V `View::handleTickEvent()` je implementirana tudi logika za odziv na dogodke vmesnika.

FreeRTOSConfig.h

```
#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
  #include <stdint.h>
  extern uint32_t SystemCoreClock;
#endif
#ifndef CMSIS_device_header
#define CMSIS_device_header "stm32h7xx.h"
#endif /* CMSIS_device_header */

#define configENABLE_FPU                         0
#define configENABLE_MPU                         0

#define configUSE_PREEMPTION                     1
#define configSUPPORT_STATIC_ALLOCATION          1
#define configSUPPORT_DYNAMIC_ALLOCATION         1
#define configUSE_IDLE_HOOK                      1
#define configUSE_TICK_HOOK                      0
#define configCPU_CLOCK_HZ                       ( SystemCoreClock )
#define configTICK_RATE_HZ                       ((TickType_t)1000)
```

```
#define configMAX_PRIORITIES                    ( 56 )
#define configMINIMAL_STACK_SIZE                ((uint16_t)128)
#define configTOTAL_HEAP_SIZE                   ((size_t)100000)
#define configMAX_TASK_NAME_LEN                 ( 16 )
#define configUSE_TRACE_FACILITY                1
#define configUSE_16_BIT_TICKS                  0
#define configUSE_MUTEXES                       1
#define configQUEUE_REGISTRY_SIZE               8
#define configUSE_RECURSIVE_MUTEXES             1
#define configUSE_APPLICATION_TASK_TAG          1
#define configUSE_COUNTING_SEMAPHORES           1
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 0
```

Ta datoteka je konfiguracijska datoteka za FreeRTOS, imenovana `FreeRTOSConfig.h`, ki določa različne nastavitve za delovanje FreeRTOS jedra v določeni aplikaciji. FreeRTOS je realnočasovni operacijski sistem (RTOS), ki omogoča večopravilnost na mikrokrmilnikih, kot je STM32H7, za katerega je ta konfiguracija prilagojena. V tej datoteki so definirane ključne nastavitve, ki določajo, kako FreeRTOS upravlja z nalogami (taski), prekinitvami in pomnilnikom.

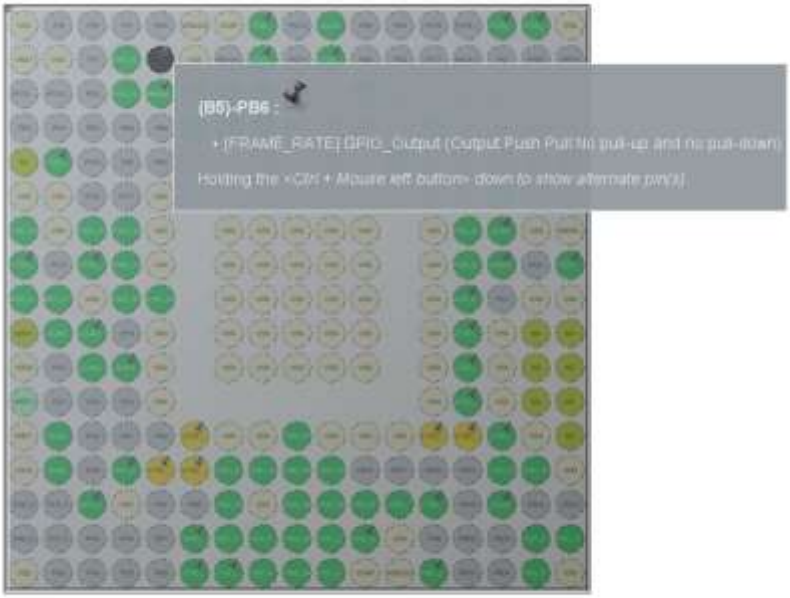
Datoteka najprej vključuje nekaj pomembnih parametrov za delovanje RTOS jedra. Na primer, `configUSE_PREEMPTION` je nastavljen na `1`, kar omogoča prekinjajoče načrtovanje, kar pomeni, da bo jedro vedno izbralo nalogo z najvišjo prioriteto za izvajanje. `configCPU_CLOCK_HZ` uporablja frekvenco sistemske ure mikrokrmilnika, definirano v `SystemCoreClock`. Hitrost takta je določena na 1000 Hz z `configTICK_RATE_HZ`, kar pomeni, da sistem izvede 1000 taktov na sekundo.

Konfiguracija podpira tako statično kot dinamično dodeljevanje pomnilnika, kar je nadzorovano z nastavitvami `configSUPPORT_STATIC_ALLOCATION` in `configSUPPORT_DYNAMIC_ALLOCATION`.

Pomemben del konfiguracije se nanaša na obravnavo prekinitev. Nastavljene so prioritete prekinitev, kot sta `configLIBRARY_LOWEST_INTERRUPT_PRIORITY` (nastavljena na `15`) in `configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY` (nastavljena na `5`), kar zagotavlja, da so prekinitve pravilno prednostno razporejene in da lahko klici FreeRTOS API varno tečejo znotraj dovoljenih meja.

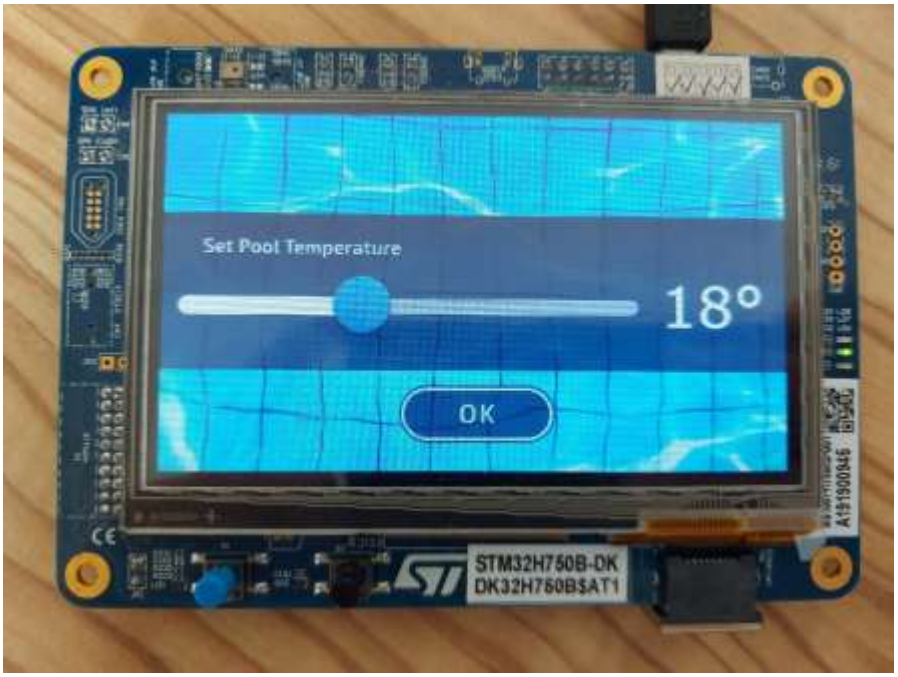
Za beleženje in odpravljanje napak je uporabljena makro funkcija `configASSERT()`, ki preveri, ali pogoj drži, in v primeru napake ustavi sistem. Dodatno so definirani "hook" funkcije (`traceTASK_SWITCHED_OUT()` in `traceTASK_SWITCHED_IN()`), ki omogočajo merjenje obremenitve MCU z merjenjem časa, porabljenega v funkciji `idle`.

Na koncu je konfiguracija prilagojena specifičnemu mikrokontrolerju STM32H7, pri čemer so uporabljeni specifični nastavitve za obdelavo prekinitev in nastavitev pomnilnika.



TFBGA240 +25 (Top view)

3. Delovanje in fotografije



[Predstavitev delovanja]

[Povezava do datoteke projekta]

[Povezava do datoteke projekta 2]