Univerza v Ljubljani

Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Večna pot 113
1000 Ljubljana

Organizacija računalnikov

Professor: viš. pred. dr. Robert Rozman
Izdelal: Jan Drole (63200086)

Uvod

V sklopu predmeta sem dobil inspiracijo za izdelavo pretvornika morsejeve abecede v tekst. Morsejeva abeceda je bila uporabljena v preteklosti za prenos sporočil preko telegrafije. V današnjem času je uporaba te abecede zelo redka, vendar je še vedno zanimiva za uporabo v programiranju. Ker sem se odločila nalogo izdelati na plošči STM32H750B, ki ima na njej že integriran zaslon, se mi je zdelo edino smiselno, da tega tudi uporabim. To pa je prineslo veliko izzivov, katere bom predstavil ter pojasnil mojo rešitev.

Program za branje morsejeve abecede

V prvi fazi, sem želel ploščo stesirati in napisati program, ki bere znake iz gumba na plošči, ter jih pretvori v ascci znake in te poščlje preko COM porta s pomočjo usart priključka. Z računalnikom se na ta vmesnik nato povežemo in znake prikazujemo na zaslonu v terminalu.

1 Initializacija usart3:

```
{

  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart3.Init.ClockPrescaler = UART_PRESCALER_DIV1;
  huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetTxFifoThreshold(&huart3, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetRxFifoThreshold(&huart3, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_DisableFifoMode(&huart3) != HAL_OK)
  {
    Error_Handler();
  }

}
```

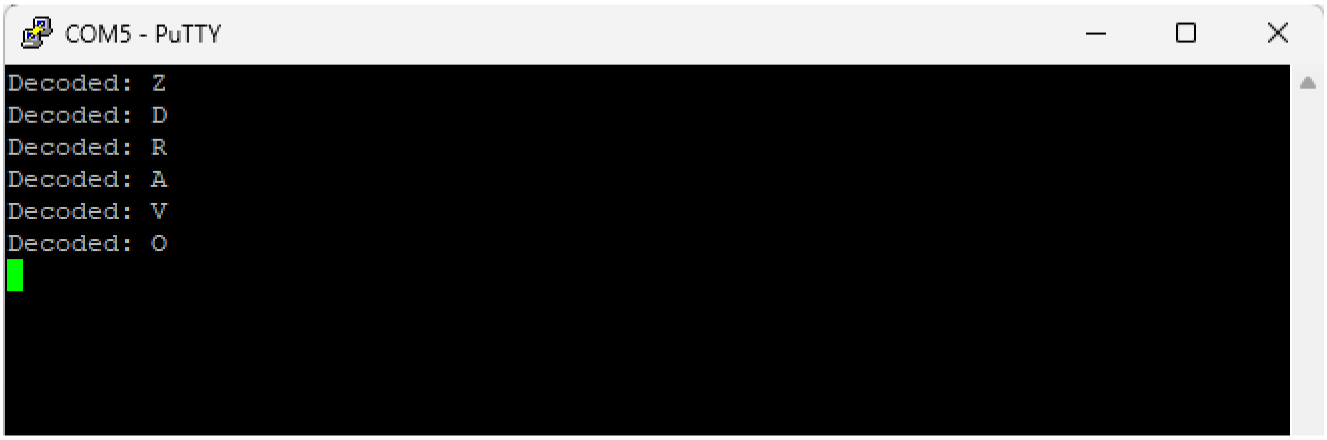
2 Inicializacia gpio:

```
{

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}
```

3 Glavna zanka:

```
{
      while (1){
         if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_SET) { // Button pressed
             HAL_GPIO_WritePin(GPIOI, GPIO_PIN_13, GPIO_PIN_SET); // Turn on LED
             buttonPressTime = HAL_GetTick();
             while (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_SET) {} // Wait until button release
             lastButtonReleaseTime = HAL_GetTick();
             buttonPressTime = lastButtonReleaseTime - buttonPressTime; // Measure how long the button was pressed
             HAL_GPIO_WritePin(GPIOI, GPIO_PIN_13, GPIO_PIN_RESET); // Turn off LED when button is released

             // Determine if the signal is a dot or a dash
             if (buttonPressTime < (DOT_TIME + DASH_TIME) / 2) {
                 morseSequence[seqIndex++] = '.';
             } else {
                 morseSequence[seqIndex++] = '-';
             }
         } else {
             // Check if a letter has been completed
             if (seqIndex != 0 && (HAL_GetTick() - lastButtonReleaseTime >= LETTER_PAUSE_TIME)) {
                 morseSequence[seqIndex] = '\0'; // Null-terminate the sequence
                 char decodedChar = decodeMorse(morseSequence);  // Assume `decodeMorse` is implemented elsewhere
                 snprintf(transmitBuf, sizeof(transmitBuf), "Decoded: %c\r\n", decodedChar);
                 HAL_UART_Transmit(&huart3, (uint8_t *)transmitBuf, strlen(transmitBuf), HAL_MAX_DELAY);
                 seqIndex = 0; // Reset for the next letter
             }
         }
         HAL_Delay(50); // Delay for stability
     }
}
```

4 Rezultat:



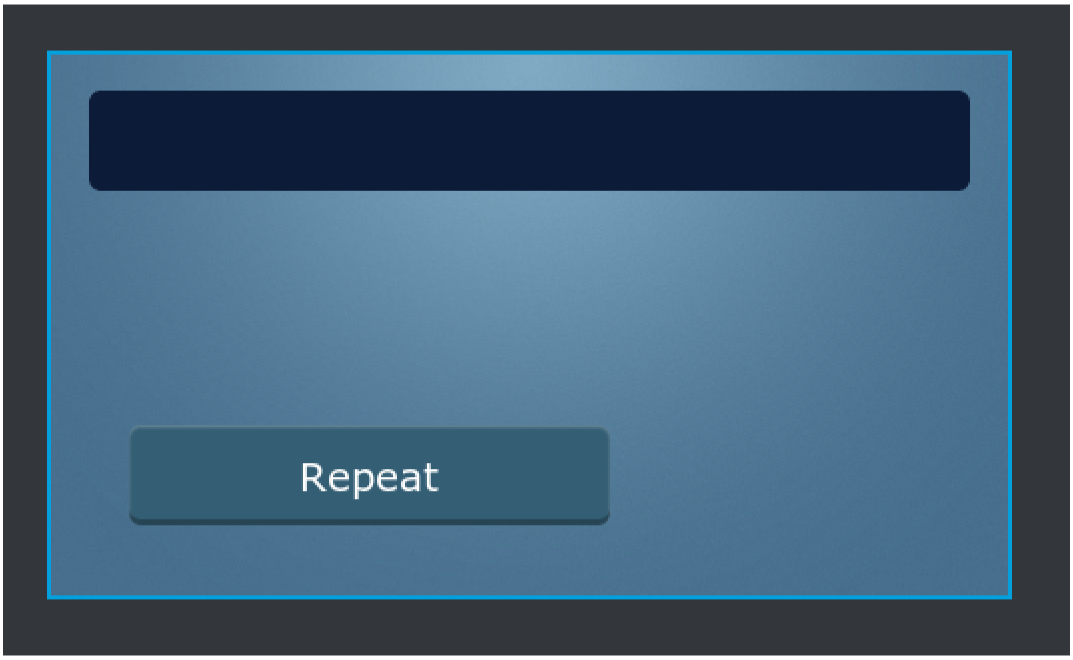
```
Decoded: Z
Decoded: D
Decoded: R
Decoded: A
Decoded: V
Decoded: O
```

Grafični vmesnik

Za izdelavo grafičnega vmesnika sem uporabil orodje TouchGFX Designer, kjer lahko s pomočjo "drag and drop" metode izdelamo poljuben vmesnik. Vse elemente označimo s svojim unikatnim identifikatorjem, nato pa funkcije definiramo znotraj generirane programske kode.

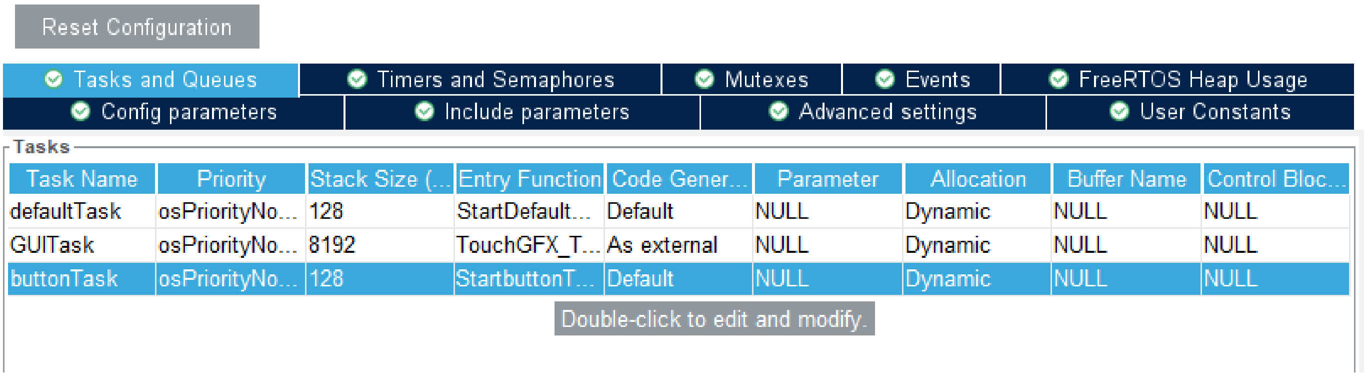
1. Izgled vmesnika:

Za izdelavo vmesnika sem najprej dal na zaslon gumb "Repeat", ki ga bomo kasneje uporabili za prikaz znakov v morsejevi abecedi z integrirano LED diodo. Nato sem dodal še polje za besedilo, kjer se bodo znaki prikazali, za konec pa sem dodal še svetlo modro ozadje, ter temno modro ozadje za besedilo. Ko sem zaključil, z ustvarjanjem vmesnika, sem avtomatsko zgeneriral vso potrebno kodo.



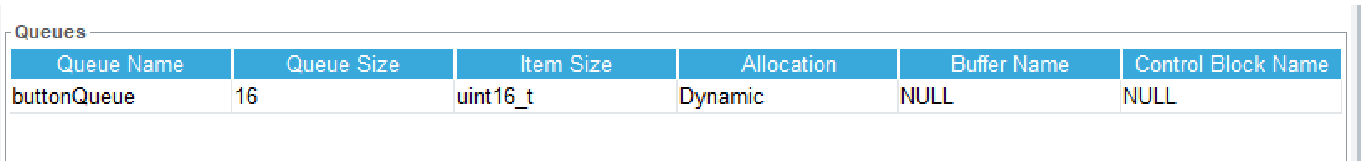
2. Potrebne nastavitve:

Ker TouchGFX za svoje delovanje uporablja FreeRTOS, sem vanjga dodal svoj task imenovan "buttonTask", ki se izvaja tekom delovanja knjižnice, tam pa bo vsa potrebna logika za delovanje. Poleg tega sem moral dodati še Queue, za namen prenosa znakov med glavno logiko in grafičnim vmesnikom.



Tasks

Task Name	Priority	Stack Size (...	Entry Function	Code Gener...	Parameter	Allocation	Buffer Name	Control Bloc...
defaultTask	osPriorityNo...	128	StartDefault...	Default	NULL	Dynamic	NULL	NULL
GUITask	osPriorityNo...	8192	TouchGFX_T...	As external	NULL	Dynamic	NULL	NULL
buttonTask	osPriorityNo...	128	StartbuttonT...	Default	NULL	Dynamic	NULL	NULL



Queues

Queue Name	Queue Size	Item Size	Allocation	Buffer Name	Control Block Name
buttonQueue	16	uint16_t	Dynamic	NULL	NULL

3. Povezovanje logike z grafičnim vmesnikom:

`main.c`:

Vsa logika, ki sem jo uporabil v prvem delu, se sedaj prenese v avtomatsko generirano funkcijo `void StartbuttonTask(void *argument){}`, dekodirane znake pa se namesto prek usart komukacijskega protoka, pošilja v prej definiran Queue s pomočjo `osMessageQueuePut(buttonQueueHandle, &decodedChar, 0, 0);`.

- `Model.cpp`:

V tej datoteki, smo najprej definirali zunanjo spremenljivko `buttonQueueHandle;`, nato pa v funkciji tick, dodali klic na funkcijo `addChar` v kolikor Queue ni prazen.

```
#include <gui/model/Model.hpp>
#include <gui/model/ModelListener.hpp>
#include <cmsis_os2.h>
#include <main.h>

extern "C"
{
    extern osMessageQueueId_t buttonQueueHandle;
}
Model::Model() : modelListener(0)
{

}

void Model::tick()
{
    if (osMessageQueueGetCount(buttonQueueHandle)> 0)
    {
        if (osMessageQueueGet(buttonQueueHandle, &button, 0, 0) == osOK){
            modelListener->addChar(button);
        }
    }
}
```

- `ModelListener.hpp`:

V tej datoteki smo definirali funkcijo `addChar`, s pomočjo katere bomo kasneje dodali znake v polje znakov, ki jih bomo prikazali na zaslonu.

```
#ifndef MODELLISTENER_HPP
#define MODELLISTENER_HPP

#include <gui/model/Model.hpp>

class ModelListener
{
public:
    ModelListener() : model(0) {}
    virtual ~ModelListener() {}

    void bind(Model* m)
    {
        model = m;
    }
    virtual void addChar(char value);
protected:
    Model* model;
};

#endif // MODELLISTENER_HPP
```

- `Screen1Presenter.cpp`:

Tu smo fukcijo realizirali, tako, da izvede klic na funkcijo `addChar` v `View` datoteki, ki bo dodala znak v polje znakov.

```

void Screen1Presenter::addChar(char value)
{
    view.addChar(value);
}
```

- `Screen1View.hpp`:

V tej datoteki je bilo ponovno potrebno definirati vse funkcije ter potrebne spremenljivke

```

#ifndef SCREEN1VIEW_HPP
#define SCREEN1VIEW_HPP

#include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
#include <gui/screen1_screen/Screen1Presenter.hpp>

#define DOT_TIME 100  // Duration of a dot in milliseconds
#define DASH_TIME (3 * DOT_TIME)  // Duration of a dash
#define LETTER_PAUSE_TIME (3 * DOT_TIME)  // Pause between letters
#define WORD_PAUSE_TIME (10 * DOT_TIME)  // Pause between words

class Screen1View : public Screen1ViewBase
{
public:
    Screen1View();
    virtual ~Screen1View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    virtual void button_input();
    virtual void repeat_button();
    virtual void addChar(char );
    virtual const char* charToMorse(char c);
protected:
};

#endif // SCREEN1VIEW_HPP
#ifndef SCREEN1VIEW_HPP
#define SCREEN1VIEW_HPP

#include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
#include <gui/screen1_screen/Screen1Presenter.hpp>

#define DOT_TIME 100  // Duration of a dot in milliseconds
#define DASH_TIME (3 * DOT_TIME)  // Duration of a dash
#define LETTER_PAUSE_TIME (3 * DOT_TIME)  // Pause between letters
#define WORD_PAUSE_TIME (10 * DOT_TIME)  // Pause between words

class Screen1View : public Screen1ViewBase
{
public:
    Screen1View();
    virtual ~Screen1View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    virtual void button_input();
    virtual void repeat_button();
    virtual void addChar(char );
    virtual const char* charToMorse(char c);
protected:
};

#endif // SCREEN1VIEW_HPP
```

- `Screen1View.cpp`:

Tu smo implementirali glavno logiko grafičnega vmesnika, kjer smo definirali funkcijo `addChar`, ki bo dodala znak v polje znakov, ter funkcijo `charToMorse`, ki bo znak pretvorila v morsejevo abecedo na LED diodi.

```


#include <gui/screen1_screen/Screen1View.hpp>
#include "stm32h7xx_hal.h"
#include "main.h"
#include "string.h"

extern "C"
{
```

```cpp
    extern UART_HandleTypeDef huart3;
}
void Screen1View::button_input(){
    HAL_GPIO_TogglePin(GPIOI, GPIO_PIN_13);

}


Screen1View::Screen1View()
{

}

void Screen1View::setupScreen()
{
    Screen1ViewBase::setupScreen();
    memset(textArea1Buffer, 0, TEXTAREA1_SIZE * sizeof(Unicode::UnicodeChar));
}

void Screen1View::tearDownScreen()
{
    Screen1ViewBase::tearDownScreen();
}

const char* Screen1View::charToMorse(char c) {
    switch (c) {
        case 'A': return ".-";
        case 'B': return "-...";
        case 'C': return "-.-.";
        case 'D': return "-..";
        case 'E': return ".";
        case 'F': return "..-.";
        case 'G': return "--.";
        case 'H': return "....";
        case 'I': return "..";
        case 'J': return ".---";
        case 'K': return "-.-";
        case 'L': return ".-..";
        case 'M': return "--";
        case 'N': return "-.";
        case 'O': return "---";
        case 'P': return ".--.";
        case 'Q': return "--.-";
        case 'R': return ".-.";
        case 'S': return "...";
        case 'T': return "-";
        case 'U': return "..-";
        case 'V': return "...-";
        case 'W': return ".--";
        case 'X': return "-..-";
        case 'Y': return "-.--";
        case 'Z': return "--..";
        default: return "";  // Return empty string for unsupported characters
    }
```

```
}


void Screen1View::repeat_button() {
    Unicode::UnicodeChar tempBuffer[TEXTAREA1_SIZE];  // Temporary buffer to store the original message

    // Copy the original content to a temporary buffer and clear the display buffer
    Unicode::strncpy(tempBuffer, textArea1Buffer, TEXTAREA1_SIZE);
    textArea1Buffer[0] = '\0';  // Clear the display buffer
    textArea1.invalidate();  // Update the display immediately

    int currentIndex = 0;  // Start from the first character of the temporary buffer
    while (tempBuffer[currentIndex] != '\0') {  // Loop over each character in the temporary buffer
        const char* morseCode = charToMorse(tempBuffer[currentIndex]);  // Get Morse code for the current character

        // Blink LED for each symbol in the Morse code
        int codeIndex = 0;
        while (morseCode[codeIndex] != '\0') {
            HAL_GPIO_WritePin(GPIOI, GPIO_PIN_13, GPIO_PIN_SET);  // LED ON
            // Determine the duration of the blink
            HAL_Delay(morseCode[codeIndex] == '.' ? DOT_TIME : DASH_TIME);
            HAL_GPIO_WritePin(GPIOI, GPIO_PIN_13, GPIO_PIN_RESET);  // LED OFF
            HAL_Delay(DOT_TIME);  // Inter-element gap within a character
            codeIndex++;
        }

        HAL_Delay(LETTER_PAUSE_TIME);  // Space between letters
        currentIndex++;  // Move to the next character
    }

    // Optionally flash LED or perform any other sign-off action here
    HAL_GPIO_WritePin(GPIOI, GPIO_PIN_13, GPIO_PIN_SET);  // LED ON

}

void Screen1View::addChar(char value) {
    // Find the current length of the Unicode string in textArea1Buffer
    int currentLength = 0;
    while (currentLength < TEXTAREA1_SIZE && textArea1Buffer[currentLength] != 0) {
        currentLength++;
    }

    // Append the character if there's room
    if (currentLength < TEXTAREA1_SIZE - 1) {
        textArea1Buffer[currentLength] = value;  // Cast char to UnicodeChar
        textArea1Buffer[currentLength + 1] = 0;  // Null-terminate the Unicode string
        textArea1.invalidate();  // Request a redraw of the text area
```

```
      }
  }
```

Končni rezultat