FRI | UNIVERZA V LJUBLJANI
Fakulteta za računalništvo in informatiko

Simulator klavirja

2. domača naloga

Avtor: Nejc Šneberger
Mentor: viš. pred. dr. Robert Rozman
Šolsko leto 2024/25

Kazalo

Kazalo slik

1. Uvod

Za drugo domačo nalogo sem se odločil izdelati simulator klavirja. Na zaslonu plošče STM32H750B-DK je prikazanih 12 tipk (5 črnih in 7 belih), ki jih lahko pritiskamo posamezno ali v kombinaciji, s čimer je mogoče igrati tudi akorde.

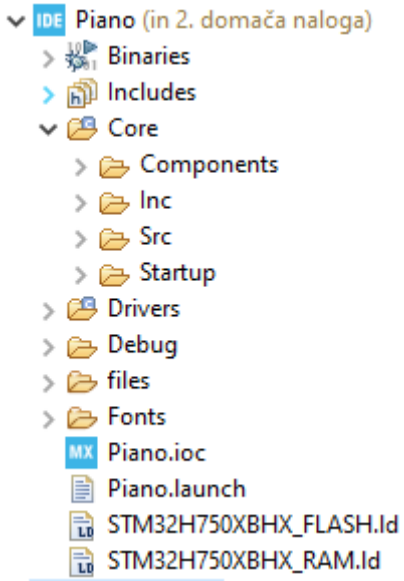
Tipke so grafično prikazane in občutljive na dotik, kar omogoča interaktivno igranje. Vsaka tipka je povezana s tonom, ki se predvaja preko zvočnika ali zvočnega modula.

Projekt vključuje tudi logiko za zaznavanje večkratnih pritiskov, tako da so akordi pravilno reproducirani. Za razvoj sem uporabil modula za izhodni zvok in modul za zaznavanje dotika na plošči.

2. Opis konfiguracije plošče STM32H7

Za razvoj kode sem uporabil razvojno okolje STM32CubeIDE (v nadaljevanju IDE), ki omogoča ustvarjanje pravilno konfiguriranih praznih projektov s pomočjo *.ioc datoteke. Pri ustvarjanju novega projekta IDE avtomatsko generira vrsto map in datotek, ki so ključne za razvoj aplikacij na sistemu STM32H750B-DK.

V projektu lahko najdemo zagonsko datoteko Piano.launch ter slike flash in RAM pomnilnika. Posebno zanimive pa so mape Core in Drivers. V obeh mapah se nahajajo datoteke z zaglavji (.h) in izvorno kodo (.c) različnih modulov. Razlika med njima je v tem, da mapa Drivers vsebuje datoteke, pridobljene iz STM-ovega uradnega repozitorija, medtem ko mapa Core poleg STM-ovih datotek vsebuje tudi datoteke, ki so nastale med razvojem simulatorja.



Slika 1: Struktura projekta

3. Opis razvoja aplikacije

Izris na zaslon

Sprva je bilo na zaslon potrebno izrisati vse potrebne tipke in tone, ki jih predstavljajo. Inicializirati je bilo potrebno LCD zaslon, da se nanj lahko riše. V datoteki display.c to storimo s klicom metode DisplayInit(void), ki uporablja API stm32h750b_discovery_lcd za vklop potrebnih modulov.

```
void DisplayInit(void)
{
    uint32_t x_size, y_size;

    BSP_LCD_Init(0, LCD_ORIENTATION_LANDSCAPE);

    BSP_LCD_GetXSize(0, &x_size);
    BSP_LCD_GetYSize(0, &y_size);

    _hTS.Width = x_size;
    _hTS.Height = y_size;
    _hTS.Orientation = TS_SWAP_XY;
    _hTS.Accuracy = 5;

    UTIL_LCD_SetFuncDriver(&LCD_Driver);
    UTIL_LCD_SetLayer(0);
    UTIL_LCD_SetFont(&Font16);

    if (BSP_TS_Init(0, &_hTS) != BSP_ERROR_NONE) {
        UTIL_LCD_DisplayStringAt(0, 220, (uint8_t*)"TS_Init Failed!", CENTER_MODE);
    }
}
```

Slika 2: Inicializacija zaslona

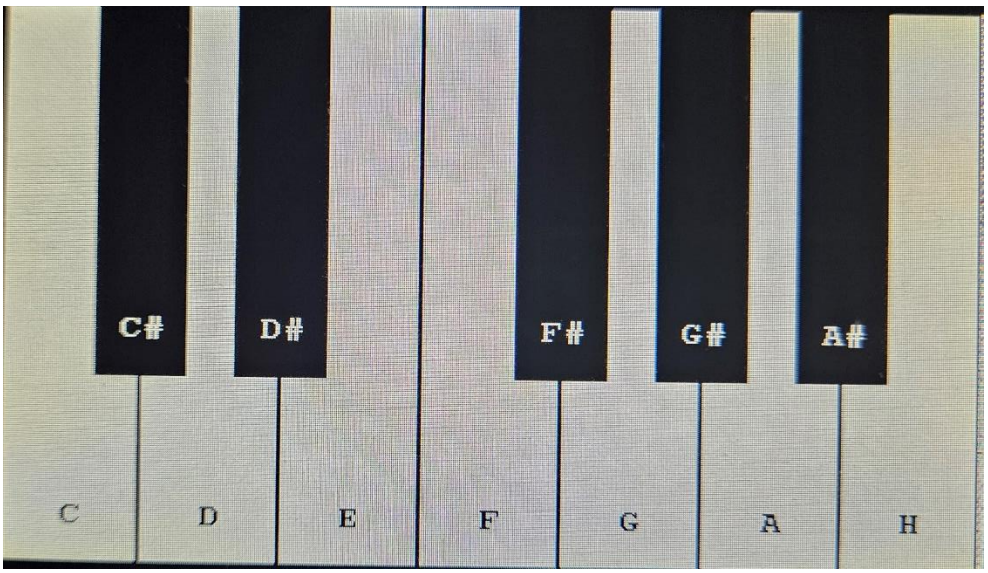
Ko je zaslon inicializiran, lahko nanj narišemo tipke. Za to uporabimo metodo DrawPiano(void). Tipke se potrebuje tudi v drugih modulih, ko na primer želimo določen ton predvajati na zvočnem izhodu plošče, zato je tipka shranjena kot podatkovna struktura, polje vseh tipk pa je globalno.

```
void DrawPiano(void)
{
    for (int i = 0; i < 7; i++)
    {
        int idx = whiteKeysIndices[i];
        DrawWhiteKey(&keys[idx]);
    }

    for (int i = 0; i < 5; i++)
    {
        int idx = blackKeysIndices[i];
        DrawBlackKey(&keys[idx]);
    }

}
```

Slika 3: Izris tipk na zaslon

```c
typedef struct
{
    uint32_t x;
    uint32_t y;
    uint32_t width;
    uint32_t height;
    char* symbol;
    float frequency;
    int white;
    int pressed;
    int currentlyTouched;
    uint32_t index;
} Key;

extern Key keys[12];
```

Slika 4: Podatkovna struktura tipke



Slika 5: Izgled izrisa tipk

Zaznavanje dotikov

Inicializaciji in risanju je sledilo zaznavanje dotikov na plošči. Za zaznavanje dotikov se uporablja API stm32h750b_discovery_ts. V datoteki display.c se nahaja metoda HandleTouchInput(void). Ta se uporabi v mainu v neprekinjeni while zanki.

```
void HandleTouchInput()
{
    TS_MultiTouch_State_t ts;
    BSP_TS_Get_MultiTouchState(0, &ts);

    for (int i = 0; i < sizeof(keys) / sizeof(keys[0]); i++)
    {
        keys[i].currentlyTouched = 0;
    }

    for (int t = 0; t < ts.TouchDetected; t++)
    {
        uint16_t x = ts.TouchX[t];
        uint16_t y = ts.TouchY[t];

        for (int j = 0; j < sizeof(blackKeysIndices)/sizeof(blackKeysIndices[0]); j++)
        {
            int idx = blackKeysIndices[j];
            if (DetectKeyPress(x, y, idx))
            {
                keys[idx].currentlyTouched = 1;
            }
        }

        for (int j = 0; j < sizeof(whiteKeysIndices)/sizeof(whiteKeysIndices[0]); j++)
        {
            int idx = whiteKeysIndices[j];
            if (DetectKeyPress(x, y, idx))
            {
                keys[idx].currentlyTouched = 1;
            }
        }
    }

    for (int i = 0; i < sizeof(keys) / sizeof(keys[0]); i++)
    {
        if (keys[i].currentlyTouched && !keys[i].pressed)
        {
            PressKey(i);
            keys[i].pressed = 1;
        } else if (!keys[i].currentlyTouched && keys[i].pressed)
        {
            ReleaseKey(i);
            keys[i].pressed = 0;
        }
    }
}
```

Slika 6: Zaznavanje dotikov

Predvajanje tonov

Zaradi logične razdelitve modulov je za predvajanje tonov ustvarjena nova datoteka bsp_audio.c. Ta del razvoja je bil precej kompleksnejši kot prejšnji, zaradi narave delovanja zvočnega kodeka. V tej datoteki so uporabljeni klici metod iz modula stm32h750b_discovery_audio.

Sprva je bilo potrebno inicializirati audio strojno opremo. To naredi metoda AudioInit(void), ki nastavi urin signal za SAI1 in pokliče BSP_AUDIO_Init(void), ki konfigurira zvočni kodek za stereo predvajanje pri 16-bitni ločljivosti in 96 kHz frekvenci vzorčenja.

Ko je modul inicializiran program pokliče metodo StartAudio(void), ki periodično pridobi pritisnjene tipke in DMA napolni z ustreznimi vrednostmi. Če je to ena tipka, je to frekvenca pritisnjenega tona, če pa jih je več pa s kombinacijo frekvenc pritisnjenih tipk.

```
void AudioInit(void)
{
    RCC_PeriphCLKInitTypeDef periphClkInitStruct = {0};
    periphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_SAI1;
    periphClkInitStruct.Sai1ClockSelection = RCC_SAI1CLKSOURCE_PLL3;
    periphClkInitStruct.PLL3.PLL3M = 16;
    periphClkInitStruct.PLL3.PLL3N = 344;
    periphClkInitStruct.PLL3.PLL3P = 7;
    periphClkInitStruct.PLL3.PLL3Q = 7;
    periphClkInitStruct.PLL3.PLL3R = 7;
    HAL_RCCEx_PeriphCLKConfig(&periphClkInitStruct);

    BSP_AUDIO_Init();
}
```

Slika 7: Inicializacija urinega signala za SAI1

```
void BSP_AUDIO_Init(void)
{
    BSP_AUDIO_Init_t audioInit;
    audioInit.Device = AUDIO_OUT_DEVICE_HEADPHONE;
    audioInit.SampleRate = AUDIO_SAMPLE_RATE;
    audioInit.BitsPerSample = AUDIO_BITS;
    audioInit.ChannelsNbr = AUDIO_CHANNELS;
    audioInit.Volume = 100;

    if (BSP_AUDIO_OUT_Init(0, &audioInit) != BSP_ERROR_NONE)
    {
        Error_Handler();
    }

    BSP_AUDIO_OUT_Play(0, (uint8_t*) audioBuffer, AUDIO_BUFFER_SIZE * AUDIO_CHANNELS * sizeof(int16_t));
}
```

Slika 8: Konfiguracija zvočnega kodeka

```
void StartAudio()
{
    uint32_t pressedCount;
    Key* pressedKeys = GetPressedKeys(&pressedCount);

    if (pressedKeys == NULL || pressedCount == 0)
    {
        return;
    }

    if (pressedCount > 12)
    {
        pressedCount = 12;
    }

    if (bufferState == 1)
    {
        ProcessBuffer(&audioBuffer[0], AUDIO_BUFFER_SIZE / 2, pressedKeys, pressedCount);
        bufferState = 0;
    }
    else if (bufferState == 2)
    {
        ProcessBuffer(&audioBuffer[AUDIO_BUFFER_SIZE / 2 * AUDIO_CHANNELS], AUDIO_BUFFER_SIZE / 2, pressedKeys, pressedCount);
        bufferState = 0;
    }

    free(pressedKeys);
}
```

Slika 9: Igranje zvoka na podlagi pritisnjenih tipk