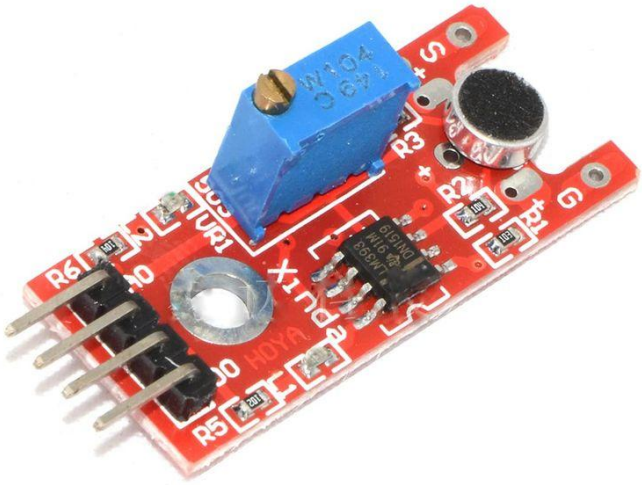
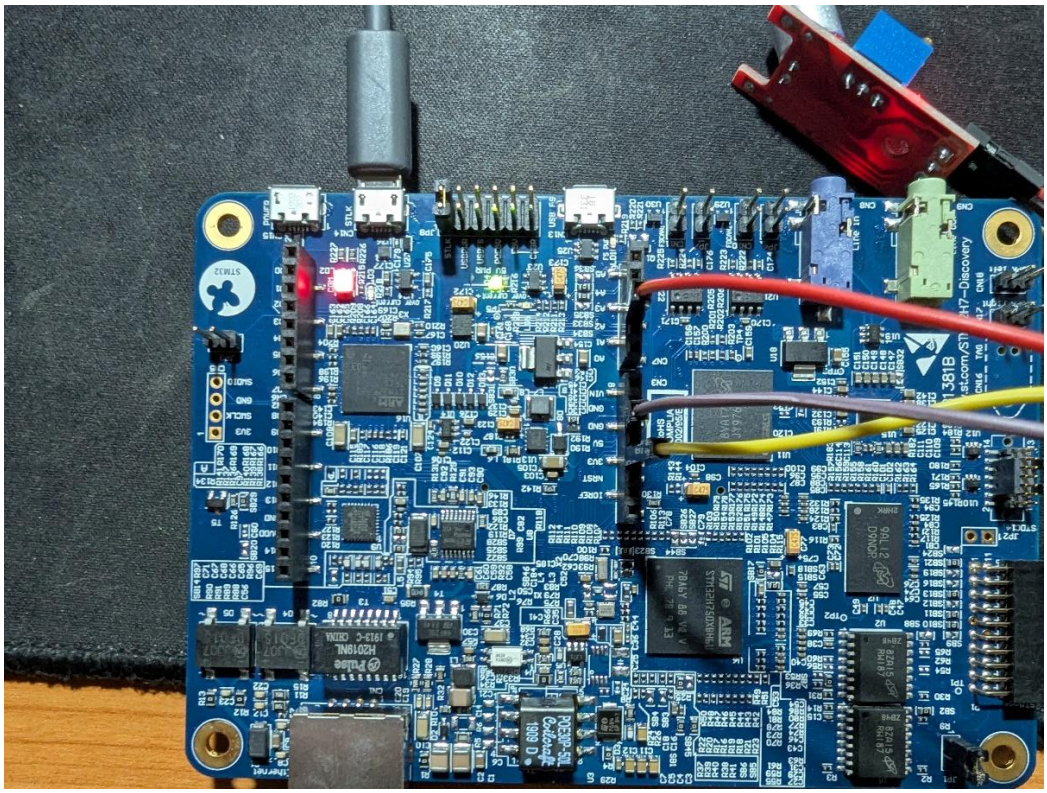
2. Domača Naloga – Flappy Bird

Na STM32H750B-DK board-u sem naredil igrico Flappy Bird. Naredil sem jo tako, da igralec lahko premika ptico na dva načina. Ali prek itegriranega zaslona ali pa z glasom prek LM393 avdio enote (Na spodnji sliki).



Avdio enota je povezana na 3.3 V, GND in pin A2, na katerem deluje ADC1.



Za podalgo projekta sem uporabil Touchscreen demo, ki smo ga uporabljali pri predmetu Vhodno-izhodne naprave, tako da samega zaslona mi ni bilo treba na roke vzpostaviti. Sam pa sem vzpostavil analogni pin in posledičino tudi analogno-digitalno pretvorbo. To vse naredi spodnji izsek kode.

```
void adc_init()
{
    // clock ADCs from PER_CK
    RCC->D3CCIPR = (RCC->D3CCIPR & (~RCC_D3CCIPR_ADCSEL_Msk)) |
RCC_D3CCIPR_ADCSEL_per_ck;

    // enable peripheral clocks
    RCC->AHB4ENR |= RCC_AHB4ENR_GPIOAEN;
    RCC->AHB1ENR |= RCC_AHB1ENR_ADC12EN;

    // set pin to analog
    GPIOA->MODER  = (GPIOA->MODER & ~(0x3u << GPIO_MODER_MODE0_Pos)) | (0x3u
<< GPIO_MODER_MODE0_Pos);
    GPIOA->PUPDR &= ~(0x3u << GPIO_PUPDR_PUPD0_Pos);
    // enable ADC1
    ADC1->CR &= ~(ADC_CR_DEEPPWD);
    ADC1->CR |= ADC_CR_ADVREGEN;
    ADC1->CR |= ADC_CR_ADCAL;
    // wait for calibration
    while (ADC1->CR & ADC_CR_ADCAL)
    {
    }
    // continious mode, allow overwrite, 10bit res and disable injected queue
    ADC1->CFGR = ADC_CFGR_JQDIS |
                 ADC_CFGR_CONT |
                 ADC_CFGR_OVRMOD |
                 ADC_CFGR_RES_10bit;

    // setup sequence { (0, 16.5) }
    ADC1->SQR1 = /* sequence length */ 1 |
                 /* 1st channel */ (0ul << 6ul);
    ADC1->SMPR1 = /* sample length 1 */ (0x3ul << 0ul);

    // enable ADC
    ADC1->CR |= ADC_CR_ADEN;

    // wait for it to be ready
    while (!(ADC1->ISR & ADC_ISR_ADRDY))
    {
    }
    ADC1->ISR = ADC_ISR_ADRDY;
    // start
    ADC1->CR |= ADC_CR_ADSTART;
}
```

Po temu, lahko samo iz ADC1->DR berem trenutno vrednost na A2. Vrednost bo 10 bitna, torej med 0 in 1023.

Ko se inicializacija konča se požene logika igre s funkcijo ***FlappyBird()***, ki požene zanko, ki teče v neskončnost (Vsak krog ima še 40 milisekundni delay-a). Vsak krog v zanki preveri ali je prišlo do dotika na zaslonu. Če je, potem odvisno od stanja igre naredi neko akcijo. Vsak krog, ko je stanje igre enak 1 preveri tudi ali je povečana aktivnost na mikrofonu. Če je, potem se sporži **FLAP**, ki dvigne ptico v igri. Isto se zgodi ob pritisku na zaslon.

```
      ts_status = BSP_TS_GetState(0, &TS_State);
      if(TS_State.TouchDetected)
      {
          if (game_state == 0) {
              game_state = 1; // starts game
              UTIL_LCD_Clear(UTIL_LCD_COLOR_LIGHTBLUE);
              UTIL_LCD_FillCircle(bird_pos_x, (int)bird_y, CIRCLE_RADIUS,
UTIL_LCD_COLOR_YELLOW);
          }
          else if (game_state == 1) {
              bird_vy = V_FLAP_px_s;
          }
          else if (game_state == 2) {
              game_state = 0;
              GameInit();
              DrawStartScreen();
          }

      }
      if (game_state == 1 && ADC1->DR > 400) {
          bird_vy = V_FLAP_px_s;
      }
```

Vsak krog se prav tako izvede izbirs in izris vseh premikajočih se elementov (ptica in ovire). Spodnji izsek prikazuje izbris ptice in izračun njene naslednje pozicija ter na koncu še izris.

```
        // clear previous bird
        UTIL_LCD_FillCircle(bird_pos_x, prev_bird_y_px, CIRCLE_RADIUS,
UTIL_LCD_COLOR_LIGHTBLUE);

        // Compute dt (seconds) since last update
        uint32_t now = HAL_GetTick();
        float dt = (now - last_tick_ms) * 0.001f;
        if (dt > 0.05f) dt = 0.05f; // safety clamp (50 ms)
        last_tick_ms = now;

        // Physics: gravity, clamp velocity, integrate position
        bird_vy += G_px_s2 * dt;
        if (bird_vy > V_DOWN_MAX) bird_vy = V_DOWN_MAX;
        if (bird_vy < V_UP_MAX)   bird_vy = V_UP_MAX;

        bird_y += bird_vy * dt;

        // convert to integer pixel for drawing
        int bird_y_px = (int)(bird_y + 0.5f);

        // bounds + game over
        if (bird_y_px - CIRCLE_RADIUS/2 < 0 || bird_y_px + CIRCLE_RADIUS/2 >
SCREEN_H) {
            game_state = 2;
            // Optionally pin to edge so draw doesn't under/overflow
            if (bird_y_px < CIRCLE_RADIUS/2) bird_y_px = CIRCLE_RADIUS/2;
            if (bird_y_px > SCREEN_H - CIRCLE_RADIUS/2) bird_y_px = SCREEN_H -
CIRCLE_RADIUS/2;
        }


        UpdatePipes(dt);
        // Draw at new position
        UTIL_LCD_FillCircle(bird_pos_x, bird_y_px, CIRCLE_RADIUS,
UTIL_LCD_COLOR_YELLOW);
```

Preden se ptice ponovno izriše se spremeni še stanje ovir (v funkciji ***UpdatePipes(dt)***). Ovire se rišejo in brišejo majhen košček na enkrat. To omogoči gladko trazicijo ovire na/z zaslona, prav tako pa prepreči utripanje zaslona, ki bi se nasprotno ob velikih izbrisih dogajal. Spodnji izsek kode prikazuje vsebino funkcije ***UpdatePipes(dt)***.

```
static void UpdatePipes(float dt) {
    float dx_f = game_speed * dt;
    dist_since_last_pipe += dx_f;

    int dx = (int)dx_f;
```

```
    if (dx > 0) {
        for (int i = 0; i < pipe_count; i++){
            if (pipes[i].alive == 0) continue;
            pipes[i].x -= dx;

            if (pipes[i].cleared == 0) CollisionDetection(pipes[i]);

            DrawPipe(pipes[i], dx);
            pipes[i].prev_x = pipes[i].x;
            if (pipes[i].x + PIPE_W < 0) pipes[i].alive = 0;
            if (pipes[i].cleared == 0 && pipes[i].x + PIPE_W < bird_pos_x) {
                score++;
                pipes[i].cleared = 1;
            };
        }
    }

    while(pipe_count < MAX_PIPES && dist_since_last_pipe >=
(float)PIPE_SPACING){
        SpawnPipe();
        dist_since_last_pipe -= (float)PIPE_SPACING;
    }
}
```

Najprej izračuna dolžino za katero se morajo premakniti vse ovire in jo prišteje vsem oviram, ki so še vidne na zaslonu. Sproti preverija tudi ali je prišlo do kolizije med ptico in oviro. Zadnji odsek funkcije je namenje postavljanu novih ovir.

Igra se konča, ko se igralec zaleti, ali pa ko preide 100 ovir.