Button Box

Button Box, neboli čudlíková krabička je je projekt na který jsem si už delší dobu brousil drápky. Jde o příslušenství k PC, které umožní rozšířit herní ovladače jako joystick atd. o nepřeberné množství čudlíků. Zvláště se hodí u leteckých simulátorů, ale nejen tam Zvýší herní prožitek nejen gamblerům, ale i svátečním hráčům. První prototyp je určen k otcově simulátoru IL-2 Sturmovik, či Warthunder. Proto jsem vzhled navrhl trochu více do retro stylu. Vypínače jsou páčkové, tlačítka pak panelová, kovová. Knoflíky enkodérů mají také příjemný „přístrojový“ design. Celý button box se pak chová jako další klasický joystick USB HID zařízení a není třeba jej nějak instalovat

Návrh

Teorie byla velmi jednoduchá, praxe už trošku složitější. Zejména proto, že jsem si na sebe upletl bič v podobě páčkových spínačů. Pro začátek jsem vyšel z již dobře popsaného projektu podobné krabičky zde. Kod je použitelný, není tedy třeba vynalézat vynalezené. Jedná se jen o propojení knihoven Keypad.h a Joystick.h, pokud byste chtěli zůstat jen u tlačítek a enkodérů, tak zde je v podstatě vše hotovo. Na jedno Arduino Micro jde pověsit až 22 tlačítek a tři enkodéry. Na základě počtu pinů na Arduinu a požadavku, aby tam bylo co nejvíce „točítek“ jsem ve Fusion360 vymodeloval návrh celé sestavy.

Nakonec jsem se ustálil na 9 spínačích, každý s LED indikací sepnutí, 4 enkodéry. (Tedy nekonečné točítko které vysílá několikrát za otočku stejný puls – vlevo, či vpravo) hodí se na jemné doladění hodnot, trimování atd. – jeden enkodér tedy ovládá celkem 3 tlačítka. 1 - mačká otáčením vlevo, 2- mačká otáčením vpravo, 3- je bonus, ovládá se stiskem osy a funguje jako klasické tlačítko. Dále jsem pro testovací účely použil otočný 6 pozicový přepínač.

Nechal jsem si vypálit nerezové čelo. Krabici jsem použil standardizovanou z GM Electronic viz seznam komponentů. Objednal jsem součástky a při čekání na zásilku jsem se pustil do návrhu software.

Požadované funkce

Většina her je navržena tak, aby šla ovládat ze standardní PC klávesnice. A pokud připojíte nějaký joystick, nebo další ovládací zařízení, všechna jsou opatřena pouze tlačítky. Nikoli spínačem ON-OFF. Proto všechny vstupy ve hrách většinou očekávají pouze krátký impulz, který přijde z klávesnice. Zapnutí světel: stisk klávesy „L“ ; vypnutí světel: opětovné stisknutí klávesy „L“. Hra tedy neočekává, že budete stále držet sepnutou klávesu „L“ tak dlouho, dokud budete chtít svítit. Bylo tedy nutno nějak ošetřit jak se bude vypínač chovat.

Všechny páčkové spínače jsou tedy softwarově řešeny tak, že při ZMĚNĚ vyšlou 100ms impuls na HIGH a potom pošlou LOW. Vypínač se tedy chová tak, že když jej přepneme do horní pozice „klikne“ jako zvolené tlačítko a dále je nečinný, když jej vrátíme do spodní

pozice, opět „klikne“ na to stejné tlačítko a opět je nečinný. Tímto tedy simuluji „mačkání“ dané klávesy na klávesnici. Problém však je, že ve stavu LOW, zůstává jen softwarově, fyzicky je ale sepnutý a s tím si matrix knihovna neporadí.

Problém se jmenuje matrix

Bohužel, jak jsem se již zmínil, níže popsaný kód využívá k namapování tlačítek knihovnu keypad.h , která využívá tzv. matrix, tedy síť tlačítek, kdy stačí propojit pouze řady a sloupce. Důvod je jednoduchý a to je ten, že Arduino má omezený počet pinů a pomocí matrixu můžete na 5+5 pinů pověsit 5×5 tlačítek. Tedy 10 vs. 25. Bohužel tato knihovna nepracuje s možností, že by bylo stisknutých více tlačítek najednou a to je problém právě spínačů, které zůstávají sepnuté.

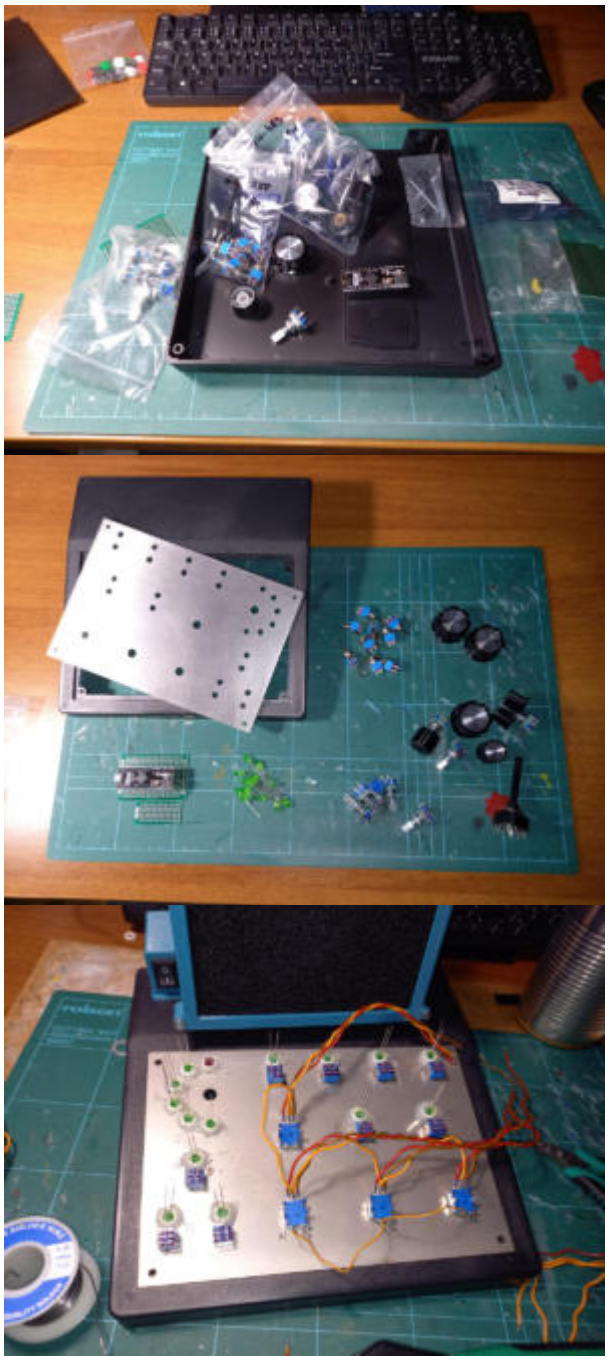
Dostal jsem se takto do velmi prekérní situace, kdy jsem měl již nakoupené součástky a hlavně vypálený nerezový plech na míru a díry na spínače byly menší než na panelová tlačítka. Mohl jsem to hodit za hlavu, použít původní plastový kryt a pověsit na to jen tlačítka a bylo by vymalováno. Nicméně mi to nedalo a chtěl jsem tam ty páčkové spínače prostě mít

Naštěstí jsem objevil velmi užitečnou věcičku jménem I/O expander, která se přes I2C sběrnici rozšíří Arduino o 8 vstupů/výstupů. Navíc se tyto expandéry dají spojit za sebe a to až 8x, takže 64 vstupně výstupních pinů, s tím už se dá pracovat (pomocí jumperů se změní adresa). Nakonec jsem použil dva, tedy 16 vstupů.

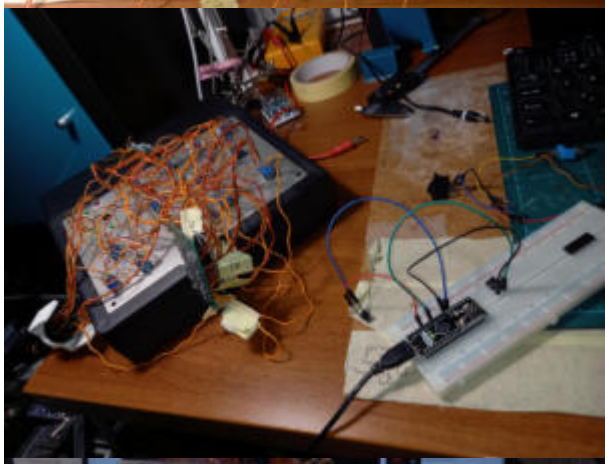
A jsme zase na začátku

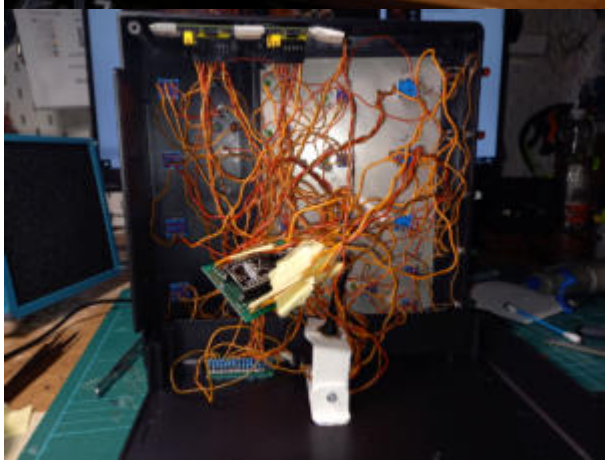
Vlivem výše uvedených okolností jsem se dostal zpět k návrhu, protože použitím expanderů mi vysvitla nová možnost rozšířit Button Box o další tlačítka. Všech 9 páčkových spínačů + otočný přepínač jsem tedy napojil na dva expandery, kde mi zbyly ještě dvě pozice pro další spínače. No a na uvolněné piny arduina jsem tak mohl pověsit matici 4×4, tedy dalších 16 tlačítek navíc! Respektive 12, protože čtyři pozice využiji pro středová tlačítka enkodérů. Takže nakonec slušná nálož 40 unikátních joystick výstupů. Už jsem se s tím moc nepáral a

vyvrtal do plastové skříně dalších 12 otvorů pro tlačítka a spínače.

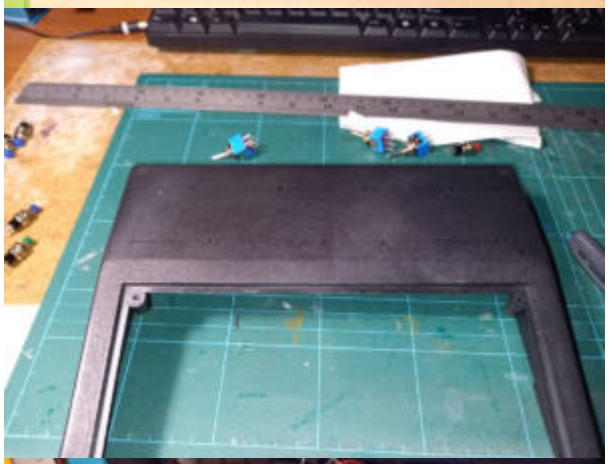


Button Box 1.0





Button Box 1.0



Program

Jelikož jsem věčný začátečník, tak je program poslepován z různých příkladů a jiných řešení. Nicméně funguje perfektně. Stejně jsem musel většinu odladit řádek po řádku. Hojně se pracuje s poli (array). Nechal jsem tam několik zakomentovaných pasáží, které mi sloužily pro ladění programu. Kód je celkem dobře okomentovaný, takže by neměl být problém se v něm vyznat. V principu jde o to, že program hlídá změny tlačítek a ukládá si informaci o tom, zda jsme tlačítko zmáčkli nebo uvolnili a podle toho reaguje. Nejvíce jsem se pral s expanderem. Existuje mnoho knihoven, ale s každou se pracuje trošku jinak a v INO se mi křížily, takže jsem musel všechny PCF8574.h odinstalovat a nahrát pouze jednu, ke které jsem našel patřičný návod.

```
//Simple buttonbox sketch
```

```
//Supports up to 25 buttons and up to 4 encoders
//version 0.2 by TOPMO3
//
//
//Arduino IDE 1.6.6 (or above) !
//
//Joystick library from  Matthew Heironimus,
https://github.com/MHeironimus/ArduinoJoystickLibrary
//
//Encoders code from Ben Buxton
//More info:
http://www.buxtronix.net/2011/10/rotary-encoders-done-properly.html
//
//Thank you guys! :)
//



//Uncomment this for HALFSTEP encoder operation
//#define HALF_STEP

#include <Keypad.h>
#include <Joystick.h>
//#include "Arduino.h"
#include <PCF8574.h>
#include <Wire.h>



PCF8574 expander1(0x20);
PCF8574 expander2(0x21);



#define INPUTMODE INPUT_PULLUP    // INPUT or INPUT_PULLUP
#define BOUNCETIME 120 // button debouncer on PCF8574, checked with scope 120
is minimum for maximum reliable response
```

```
byte E1[] = {0, 1, 2, 3, 4, 5, 6, 7}; // piny expanderu jako tlačítka 0-7 //
a nebo čísla pinů na Arduino desce
#define NUMBUTTONS_E1 sizeof(E1) // return size of array (see above)
byte E1_State[NUMBUTTONS_E1];  // array holds the actual HIGH/LOW states
byte E1_Change[NUMBUTTONS_E1]; // array holds the state changes when button
is pressed or released
//enum {E1_UNCHANGED, E1_BUTTONUP, E1_BUTTONDOWN};

byte E2[] = {0, 1, 2, 3, 4, 5, 6, 7}; // piny expanderu jako tlačítka 0-7 //
a nebo čísla pinů na Arduino desce
#define NUMBUTTONS_E2 sizeof(E2) // return size of array (see above)
byte E2_State[NUMBUTTONS_E2];  // array holds the actual HIGH/LOW states
byte E2_Change[NUMBUTTONS_E2]; // array holds the state changes when button
is pressed or released



//enum {E2_UNCHANGED, E2_BUTTONUP, E2_BUTTONDOWN};
/*
  byte A[] = {7, 8}; // čásla pinů na arduinu
  #define NUMBUTTONS_A sizeof(A) // return size of array (see above)
  byte A_State[NUMBUTTONS_A];  // array holds the actual HIGH/LOW states
  byte A_Change[NUMBUTTONS_A]; // array holds the state changes when button
is pressed or released
  /*
  byte B[] = {4, 5}; // čásla pinů na arduinu
  #define NUMBUTTONS_B sizeof(B) // return size of array (see above)
  byte B_State[NUMBUTTONS_B];  // array holds the actual HIGH/LOW states
  byte B_Change[NUMBUTTONS_B]; // array holds the state changes when button
is pressed or released
  //enum {A_UNCHANGED, A_BUTTONUP, A_BUTTONDOWN};
*/
enum {UNCHANGED, BUTTONUP, BUTTONDOWN};


unsigned long previousMillis = 0;

#define ENABLE_PULLUPS
#define NUMROTARIES 4
```

```
#define NUMBUTTONS 16
#define NUMROWS 4
#define NUMCOLS 4



//spínací tlačítka + středy trimů
byte push[4][4] = {
  {8, 9, 10, 11},
  {12, 13, 14, 15},
  {16, 17, 18, 19},
  {20, 21, 22, 23},


};
// už nepoužívám
int state[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
char tlac[5] = {'TL1', 'TL2', 'TL3', 'TL4', 'TL5'};


struct rotariesdef {
  byte pin1;
  byte pin2;
  int ccwchar;
  int cwchar;
  volatile unsigned char state;
};

rotariesdef rotaries[NUMROTARIES] {
  {4, 5, 0, 1, 0},
  {6, 7, 2, 3, 0},
  {8, 9, 4, 5, 0},
  {10, 11, 6, 7, 0},
};



#define DIR_CCW 0x10
#define DIR_CW 0x20
```

```
#define R_START 0x0

#ifdef HALF_STEP
// Use the half-step state table (emits a code at 00 and 11)
#define R_CCW_BEGIN 0x1
#define R_CW_BEGIN 0x2
#define R_START_M 0x3
#define R_CW_BEGIN_M 0x4
#define R_CCW_BEGIN_M 0x5
const unsigned char ttable[6][4] = {
  // R_START (00)
  {R_START_M,            R_CW_BEGIN,     R_CCW_BEGIN,  R_START},
  // R_CCW_BEGIN
  {R_START_M | DIR_CCW, R_START,        R_CCW_BEGIN,  R_START},
  // R_CW_BEGIN
  {R_START_M | DIR_CW,  R_CW_BEGIN,     R_START,      R_START},
  // R_START_M (11)
  {R_START_M,            R_CCW_BEGIN_M,  R_CW_BEGIN_M, R_START},
  // R_CW_BEGIN_M
  {R_START_M,            R_START_M,      R_CW_BEGIN_M, R_START | DIR_CW},
  // R_CCW_BEGIN_M
  {R_START_M,            R_CCW_BEGIN_M,  R_START_M,    R_START | DIR_CCW},
};
#else
// Use the full-step state table (emits a code at 00 only)
#define R_CW_FINAL 0x1
#define R_CW_BEGIN 0x2
#define R_CW_NEXT 0x3
#define R_CCW_BEGIN 0x4
#define R_CCW_FINAL 0x5
#define R_CCW_NEXT 0x6

const unsigned char ttable[7][4] = {
  // R_START
  {R_START,    R_CW_BEGIN,  R_CCW_BEGIN, R_START},
  // R_CW_FINAL
  {R_CW_NEXT,  R_START,     R_CW_FINAL,  R_START | DIR_CW},
  // R_CW_BEGIN
  {R_CW_NEXT,  R_CW_BEGIN,  R_START,     R_START},
```

```
  // R_CW_NEXT
  {R_CW_NEXT,  R_CW_BEGIN,  R_CW_FINAL,  R_START},
  // R_CCW_BEGIN
  {R_CCW_NEXT, R_START,     R_CCW_BEGIN, R_START},
  // R_CCW_FINAL
  {R_CCW_NEXT, R_CCW_FINAL, R_START,     R_START | DIR_CCW},
  // R_CCW_NEXT
  {R_CCW_NEXT, R_CCW_FINAL, R_CCW_BEGIN, R_START},
};
#endif


byte rowPins[NUMROWS] = {A0, A1, A2}; //connect to the row pinouts of the
keypad
byte colPins[NUMCOLS] = {A3, A4, A5}; //connect to the column pinouts of the
keypad

byte rowPins1[4] = {12, 13, A0, A1}; //řada 1-4 tlačítek
byte colPins2[4] = {A2, A3, A4, A5}; //sloupce 1,4 tlačítek


//initialize an instance of class NewKeypad
Keypad buttbx1 = Keypad( makeKeymap(push), rowPins1, colPins2, 4, 4);


//initialize an Joystick with 43 buttons; // windows dokáže zobrazit jen 32
tlačítek, hry pobírají i více
Joystick_ Joystick(JOYSTICK_DEFAULT_REPORT_ID,
                   JOYSTICK_TYPE_JOYSTICK, 47, 0,
                   true, false, false, false, false, false,
                   false, false, false, false, false);

// nastavení pole čísel tlačítek pro odesílání přes Joystick knihovnu
byte JoyE1[] = {27, 28, 29, 30, 31, 32, 33, 34};
byte JoyE2[] = {38, 40, 39, 41, 42, 35, 36, 37};
//byte JoyE1[] = {1, 2, 3, 4, 5, 6, 7, 8};
//byte JoyA[] = {25, 26};
//byte JoyB[] = {0, 1};
```

```
void rotary_init() {
  for (int i = 0; i < NUMROTARIES; i++) {
    pinMode(rotaries[i].pin1, INPUT);
    pinMode(rotaries[i].pin2, INPUT);
#ifdef ENABLE_PULLUPS
    digitalWrite(rotaries[i].pin1, HIGH);
    digitalWrite(rotaries[i].pin2, HIGH);
#endif
  }
}


void setup() {
  Joystick.begin();
  rotary_init();

  Serial.begin(9600);
  Wire.begin();
  expander1.begin();
  expander2.begin();
  // for (int i = 0; i < NUMBUTTONS_A; i++) pinMode(A[i], INPUTMODE);
  // for (int i = 0; i < NUMBUTTONS_B; i++) pinMode(B[i], INPUTMODE);



}



void loop() {

  CheckAllEncoders();

  tlacitka();

  // input_A();
  // input_B();
  input_E1();
  input_E2();
```

```
  // output_A();
  // output_B();
  output_E1();
  output_E2();
  // output_E2_sw();


}

//-------------------------------------kontrola spínačů Arduina-------------
-----------------------------------------
/*
  void input_A() {
  // read the input state and state changes of all buttons
  static unsigned long lastButtonTime; // time stamp for remembering the time
when the button states were last read
  memset(A_Change, 0, sizeof(A_Change)); // reset all old state changes
  if (millis() - lastButtonTime < BOUNCETIME) return; // within bounce time:
leave the function
  lastButtonTime = millis(); // remember the current time
  for (int i = 0; i < NUMBUTTONS_A; i++)
  {
    byte curState = digitalRead(A[i]);      // current button state
    if (INPUTMODE == INPUT_PULLUP) curState = !curState; // logic is inverted
with INPUT_PULLUP
    if (curState != A_State[i])                  // state change detected
    {
      if (curState == HIGH) A_Change[i] = BUTTONDOWN;
      else A_Change[i] = BUTTONUP;
    }
    A_State[i] = curState; // save the current button state
  }
  }
*/
/*
  void input_B() {
  // read the input state and state changes of all buttons
  static unsigned long lastButtonTime; // time stamp for remembering the time
when the button states were last read
```

```
  memset(B_Change, 0, sizeof(B_Change)); // reset all old state changes
  if (millis() - lastButtonTime < BOUNCETIME) return; // within bounce time:
leave the function
  lastButtonTime = millis(); // remember the current time
  for (int i = 0; i < NUMBUTTONS_B; i++)
  {
    byte curState = digitalRead(B[i]);      // current button state
    if (INPUTMODE == INPUT_PULLUP) curState = !curState; // logic is inverted
with INPUT_PULLUP
    if (curState != B_State[i])                  // state change detected
    {
      if (curState == HIGH) B_Change[i] = BUTTONDOWN;
      else B_Change[i] = BUTTONUP;
    }
    B_State[i] = curState; // save the current button state
  }
  }
*/

//-------------------------------------kontrola spínačů Expanderu 1 --------
-----------------------------------------

void input_E1() {
  // read the input state and state changes of all buttons
  static unsigned long lastButtonTime; // time stamp for remembering the time
when the button states were last read
  memset(E1_Change, 0, sizeof(E1_Change)); // reset all old state changes
  if (millis() - lastButtonTime < BOUNCETIME) return; // within bounce time:
leave the function
  lastButtonTime = millis(); // remember the current time
  for (int i = 0; i < NUMBUTTONS_E1; i++)
  {
    byte curState = expander1.readButton(E1[i]);      // current button state
    // if (INPUTMODE == INPUT_PULLUP) curState = !curState; // logic is
inverted with INPUT_PULLUP
    if (curState != E1_State[i])                  // state change detected
    {
      if (curState == LOW) E1_Change[i] = BUTTONDOWN;
      else E1_Change[i] = BUTTONUP;
```

```
    }
    E1_State[i] = curState; // save the current button state
  }
}

//--------------------------------------kontrola spínačů Expanderu 2 --------
-----------------------------------------

void input_E2() {
  // read the input state and state changes of all buttons
  static unsigned long lastButtonTime; // time stamp for remembering the time
when the button states were last read
  memset(E2_Change, 0, sizeof(E2_Change)); // reset all old state changes
  if (millis() - lastButtonTime < BOUNCETIME) return; // within bounce time:
leave the function
  lastButtonTime = millis(); // remember the current time
  for (int i = 0; i < NUMBUTTONS_E2; i++)
  {
    byte curState = expander2.readButton(E2[i]);      // current button state
    // if (INPUTMODE == INPUT_PULLUP) curState = !curState; // logic is
inverted with INPUT_PULLUP
    if (curState != E2_State[i])                  // state change detected
    {
      if (curState == HIGH) E2_Change[i] = BUTTONDOWN;
      else E2_Change[i] = BUTTONUP;
    }
    E2_State[i] = curState; // save the current button state
  }
}



//--------------------------------------kontrola tlačítek--------------------
-----------------------------------------
void tlacitka(void) {
  if (buttbx1.getKeys())
  {
    for (int i = 0; i < LIST_MAX; i++) // Scan the whole key list.
    {
      if ( buttbx1.key[i].stateChanged )   // Only find keys that have
```

```
changed state.
      {
        switch (buttbx1.key[i].kstate) {  // Report active key state : IDLE,
PRESSED, HOLD, or RELEASED
          case PRESSED:

          case HOLD:
            Joystick.setButton(buttbx1.key[i].kchar, 1);
Serial.println(buttbx1.key[i].kchar);

            break;

          case RELEASED:

          case IDLE:
            Joystick.setButton(buttbx1.key[i].kchar, 0);

            break;


        }

      }
    }
  }
}




//--------------------------------Nastavení výstupů- A- Arduino------------
-------------------------------------------------------------------------
/*
  void output_A() {
  // send a message to Serial if a button state (pressed/released) has
changed
  // button pressed: Send button pin number with minus sign
  // button released: Send button pin number
  byte action;
```

```
  for (int i = 0; i < NUMBUTTONS_A; i++)
  {
    switch (A_Change[i])
    {
      case BUTTONUP:
        Joystick.setButton(JoyA[i], 1);
        delay(200);
        Joystick.setButton(JoyA[i], 0);
        Serial.println(JoyA[i]); break;
      case BUTTONDOWN:
        Joystick.setButton(JoyA[i], 1);
        delay(200);
        Joystick.setButton(JoyA[i], 0);
        Serial.println(-JoyA[i]); break;
    }
  }
  }


  /*
  void output_B() {
  // send a message to Serial if a button state (pressed/released) has
changed
  // button pressed: Send button pin number with minus sign
  // button released: Send button pin number
  byte action;
  for (int i = 0; i < NUMBUTTONS_B; i++)
  {
    switch (B_Change[i])
    {
      case BUTTONUP:
        Joystick.setButton(JoyB[i], 1);
        delay(200);
        Joystick.setButton(JoyB[i], 0);
        Serial.println(JoyB[i]); break;
      case BUTTONDOWN:
        Joystick.setButton(JoyB[i], 1);
        delay(200);
        Joystick.setButton(JoyB[i], 0);
```

```
        Serial.println(-JoyB[i]); break;
    }
  }
  }
*/
//--------------------------------Nastavení výstupů- E1--------------------
-------------------------------------------------------------------

void output_E1() {
  // send a message to Serial if a button state (pressed/released) has
changed
  // button pressed: Send button pin number with minus sign
  // button released: Send button pin number
  byte action;
  for (int i = 0; i < NUMBUTTONS_E1; i++)
  {
    switch (E1_Change[i])
    {
      case BUTTONUP:
        Joystick.setButton(JoyE1[i], 1);
        delay(100);
        Joystick.setButton(JoyE1[i], 0);
        Serial.println(JoyE1[i]);
        break;
      case BUTTONDOWN:
        Joystick.setButton(JoyE1[i], 1);
        delay(100);
        Joystick.setButton(JoyE1[i], 0); Serial.println(-JoyE1[i]); break;
    }
  }
}

//--------------------------------Nastavení výstupů- E2--------------------
-------------------------------------------------------------------

void output_E2() {
  // send a message to Serial if a button state (pressed/released) has
changed
  // button pressed: Send button pin number with minus sign
```

```
// button released: Send button pin number
byte action;
int temp_i;
bool last;

for (int i = 0; i < 5; i++)
{
  switch (E2_Change[i])
  {
    case BUTTONUP:
      Joystick.setButton(JoyE2[i], 1);
      Serial.println(JoyE2[i]);
      delay(100);
      Joystick.setButton(JoyE2[i], 0);
      //Serial.println(-JoyE2[i]);
      if (JoyE2[i] == 38) {
        last = 1;

      }
      else {
        last = 0;
      }



      //temp_i = JoyE2[i];
      Serial.print("Last: ");
      Serial.println(last);
      break;
    case BUTTONDOWN:
      //Joystick.setButton(JoyE2[i], 1);
      //delay(200);
      if (i == 0 && last != 1) {
        Joystick.setButton(JoyE2[i], 1); Serial.println(JoyE2[i]);
        delay(100);
        Joystick.setButton(JoyE2[i], 0);
        Serial.println(JoyE2[i]);
        temp_i = 2;
        Serial.print("Last: ");
```

```
          Serial.println(last);
          break;



        }
    }



  }


  for (int i = 5;   i < 8; i++)
  {
    switch (E2_Change[i])
    {
      case BUTTONUP:
        Joystick.setButton(JoyE2[i], 1);
        delay(100);
        Joystick.setButton(JoyE2[i], 0);
        Serial.println(JoyE2[i]);
        break;
      case BUTTONDOWN:
        Joystick.setButton(JoyE2[i], 1);
        delay(100);
        Joystick.setButton(JoyE2[i], 0); Serial.println(-JoyE2[i]); break;
    }
  }



}

void output_E2_sw() {
  // send a message to Serial if a button state (pressed/released) has
changed
  // button pressed: Send button pin number with minus sign
  // button released: Send button pin number
```

```
  byte action;
  for (int i = 0; 5 < i < 8; i++)
  {
    switch (E2_Change[i])
    {
      case BUTTONUP:
        Joystick.setButton(JoyE2[i], 1);
        delay(100);
        Joystick.setButton(JoyE2[i], 0);
        Serial.println(JoyE2[i]); break;
      case BUTTONDOWN:
        // Joystick.setButton(JoyE2[i], 1);
        // delay(200);
        //  Joystick.setButton(JoyE2[i], 0); Serial.println(-JoyE2[i]);
        break;
    }
  }
}

//----------------------------------------enkodéry-------------------------
-----------------------------------------------------
/* Read input pins and process for events. Call this either from a
   loop or an interrupt (eg pin change or timer).

   Returns 0 on no event, otherwise 0x80 or 0x40 depending on the direction.
*/
unsigned char rotary_process(int _i) {
  unsigned char pinstate = (digitalRead(rotaries[_i].pin2) << 1) |
digitalRead(rotaries[_i].pin1);
  rotaries[_i].state = ttable[rotaries[_i].state & 0xf][pinstate];
  return (rotaries[_i].state & 0x30);
}

void CheckAllEncoders(void) {
  for (int i = 0; i < NUMROTARIES; i++) {
    unsigned char result = rotary_process(i);
    if (result == DIR_CCW) {
      Joystick.setButton(rotaries[i].ccwchar, 1); delay(50);
Joystick.setButton(rotaries[i].ccwchar, 0);
```

```
Serial.println(rotaries[i].ccwchar);
    };
    if (result == DIR_CW) {
      Joystick.setButton(rotaries[i].cwchar, 1); delay(50);
Joystick.setButton(rotaries[i].cwchar, 0);
Serial.println(rotaries[i].cwchar);
    };
  }
}
```

Jdeme do finále

Ač se zdálo, že spájet to vše dohromady bude nejsnadnější, nebylo tomu tak. S přibývajícím počtem spojů se z toho stávalo čím dál větší klubko drátů a někdy se s horkám hrotem manévrovalo hodně těsně kolem bužírek. Jako třešničku na dortu jsem ještě zjistil, že jsem nechal vypálit špatný průměr (samozřejmě menší) pro otočný přepínač, takže jsem ještě musel vyrobit konstrukci, která jej drží na dně krabice, místo aby byl zašroubovaný v panelu.

Diody nejsou ošetřeny programově ale natvrdo připájené k druhé větvi dvoupólového vypínače. Předřadné rezistory jsem naskládal vedle sebe na univerzální „dírkovanou“ desku DPS, stejně tak patici pro Arduino. Sofistikovanější by bylo navrhnout nějaký tištěný spoj, ale na to teď nebyl čas, hlavně se vše ještě ladí.

Ještě jsem doplnil jednu blbinku a to je „ARM“ tedy zajištění dvou dvojic spínačů s indikací červená/zelená. Kterými můžete zajistit tlačítka proti nechtěnému stisknutí (třeba u nouzového opuštění letadla, odhození bomba atd)

Pro finální vzhled jsem ještě na plotru vyřezal z vinylové folie čísla tlačítek pro lepší orientaci. Při ostrém provozu zřejmě pak doplníme přímo o nápisy funkcí, které se ustálí u jednotlivých ovládacích prvků.

Button Box 1.0