Personsökar-Projekt

Knappsats 4x4 - komponentbeskrivning

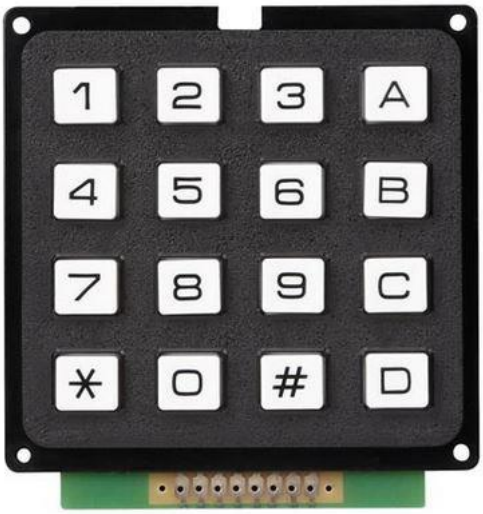

Bild 1: Källa electrokit.se

Abstract

This document describes the required behavior and interfaces of a 4x4 keypad component. The over all purpose is to let the keypad be replaceable by another keypad of another type.

Version History

Date	Version	Author	Description
21/05/2020	1.0	Mikael Andersson	

An Essential Unified Process Document

Innehållsförteckning

1 Inledning

1.1 Syfte

Som en del i kursen ii1302 vid KTH vårterminen 2020 genomför studenterna ett projekt. Det projekt (projektgrupp nr 8) som författaren deltog i syftade till att bygga en personsökare som använde sig av Internet för kommunikation.

Detta dokuments syfte är att beskriva en 4x4 knappsats[1] funktion och mjukvarugränssnitt tillsammans med en mikrokontroller (STM32F4 Discovery) för att kunna:

- underlätta för andra studenter att implementera samma knappsats eller ersätta knappsatsen mot annan knappsats, t.ex. en 3x3 knappsats.
- ge förståelse för hur andra utvecklare kan testa knappsatsens funktion.
- föreslå förbättringar till förmån från framtida studenter.

1.2 Avgränsning

Detta dokument begränsas till att behandla:

- knappsatsens funktionskrav i systemet.
- gränsnittet för att utläsa intryckt knapp.
- vilka konfigurationer som STM32F4 måste ställas in med för att få knappsatsen att fungera.

Detta dokument begränsas till att INTE behandla:

- de funktioner som kan tänkas vara önskvärda att skapa som en utveckling av att ha implementerat knappsatsen i systemet.
- andra komponenter i systemet som direkt är beroende av inmatad data via knappsatsen.

1.3 Överblick

Detta dokument innehåller följande sektioner:

- **Funktionskrav** – de funktionella krav som ställs på knappsatsen för att knappsatsen ska anses fungera korrekt.
- **Begränsningar i funktionen** – identifierade begränsningar i knappsatsens funktion.
- **Mjukvarugränssnitt UT**– det mjukvarumässiga gränsnittet som knappsatsen erbjuder och dess begränsningar.
- **Mjukvarugränssnitt IN** – vilka mjukvarugränssnitt som komponenten kräver av STM32F4.
- **Hårdvara beskrivning** – kort beskrivning om hur knappsatsen är uppbyggd.

- **Hårdvara/mjukvara interaktion - beskrivning** – beskrivning hur knappsatsen fungerar ihop med mjukvaran.
- **Testning –** kortare beskrivning av den testdrivna utvecklingsmetoden.
- **Förbättringsförslag –** identifierade brister och möjliga lösningar.
- **Bilaga A - Referenser**– lista med källhänvisning till de dokument som användes till detta dokument.
- **Bilaga B – Mjukvara keypad**– komplett C-kod som sköter knappsatsens funktion.

2 Funktionskrav

För att knappsatsen ska anses vara tillförlitlig så måste den:

- reagera på alla knapptryckningar som användaren uppfattar som en intryckt knapp.
- unikt identifiera varje av de sexton knapparna inviduellt.
- inte riskera att förstöra mikrokontrollern (STM32F4 Dicovery)

3 Begränsningar i funktionen

Följande begränsningar är identifierande vid tillfället för rapportens skrivande:

- Knappsatsen har en begränsad mekanisk hållbarhet med 1.000.000 antal knapptryckningar per tangent. [1]
- Knappsatsen garanterar funktionalitet mellan -20℃ till +60℃. [1]
- Mjukvaran måste vara skriven i C.
- En knapp i taget kan läsas av.
- Teoretisk kan en knapptryckning vara så snabb att skanningfunktionen inte hinner läsa av den intryckta knappen.

4 Mjukvarugränssnitt Ut

Funktionen keypad_scan() returnerar en variabel av datatypen *enum*. Datatypen enum definieras i h-filen som följer filen där keypad-scan() ligger lagrad. Varje knapp på knappsatsen har ett eget variabelnamn av typen enum tilldelat sig. Till exempel har knapp ”4” variabelnamnet ”key4” och knapp ”#” har variabelnamnet ”keyHash”.

I sin enklaste form kan man kalla på funktionen keypad_scan() och jämföra om detta motsvarar en viss knapp t.ex.

```
keys key = keypad_scan();
if(key==keyStar) {useful_action();}
```

Här kommer programmet att vänta tills det att knappen ”*” har tryckts och sen utföra det som definieras i funktionen useful_action().

5 Mjukvarugränssnitt In

I STM32 finns bibliotek som underlättar kommunikationen mot de register som ingår i STM32. Dessa bibliotek kallas Hardware Abstraction Layers (HAL)[2] och abstraherar bort mycket detaljer från programmeraren. Knappsatsen använder HAL_GPIO, som sköter detaljerna för General Purpose Input Output pins (GPIO-pins). Dvs, styr och läser av de fysiska gränssnitten (kopplingspinnar på STM32F4). I programmeringsgränssnittet STM32CubeIDE (IDE) så tilldelades dessa fysiska pinnar variabelnamn som förenklade programmeringen, t.ex ”ROW1_input” eller ”COL3_output”.

6 Hårdvara - beskrivning

På försättsbladet finns en bild på hur knappsatsen[1] ser ut i verkligheten. Nedan visas en schematisk bild hur knappsatsen implementerades tillsammans med STM32F4 Discovery.

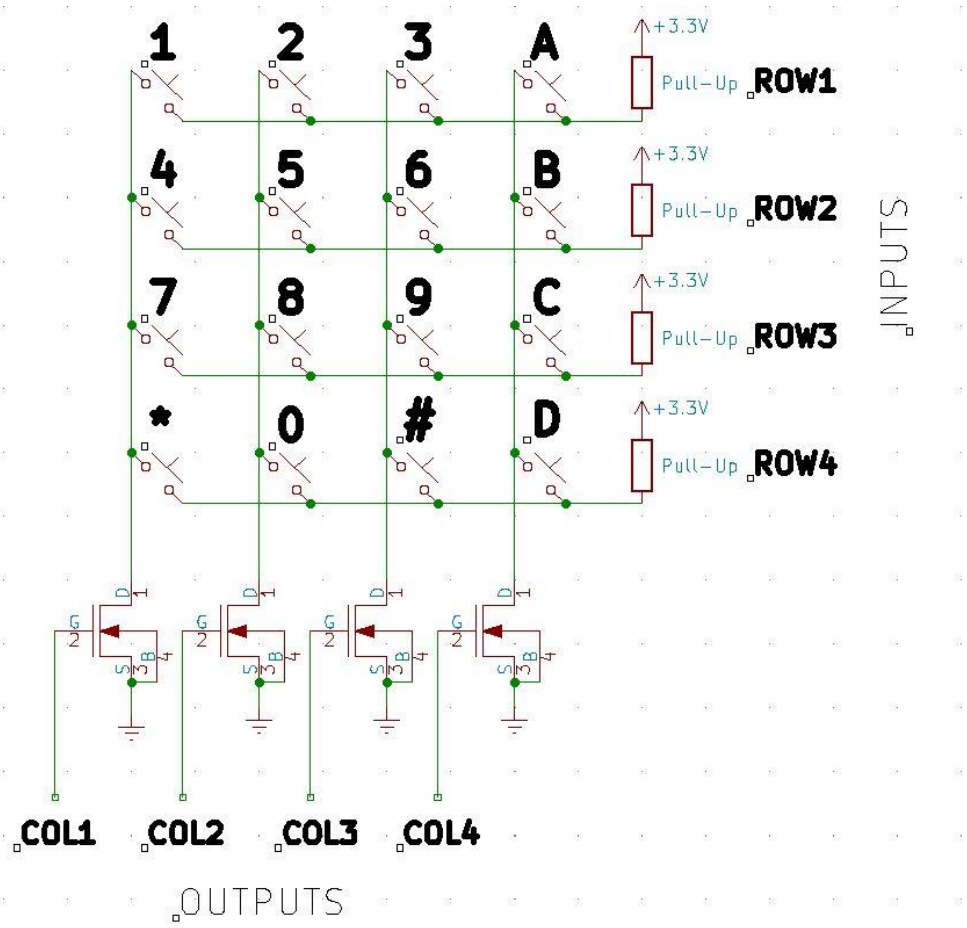

Bild 2: Av författaren skapat schema i KiCad

Utgångarna/outputs:

Dessa konfigurerades i IDE:n till *open-drain, no pull-up, no pull-down*.

Varför *open-drain*? För att inte riskera att skada STM32F4 om två knappar i olika kolumner, men samma rad, hålls in samtidigt som knappmatrisen skannas så användes open-drain. Hade *push-pull* använts så blir effekten att om två knappar i olika kolumner, men samma rad, trycks in samtidigt så kortsluts matningsspänningen (+3,3V) mot jord, med möjliga skador på STM32F4.

Varför *no pull-up, no pull-down*? När knapparna inte är intryckta så kräver inputs/ingångarna att *pull-up* eller *pull-down* stabiliserar avläsningen och detta görs inte på utgångssidan eftersom ingångarna då skulle vara flytande. Flytande ingångar ger i praktiken ett slumpmässigt avläst värde. *Pull-up* måste alltså sitta på inputs/ingångarna.

Inputs/ingångar:

Enligt resonemanget ovan så används *pull-up* på ingångarna för att ge en hög ingång vid opåverkad knappsats. Själva avläsningen in till STM23 sker på ingångens *pull-up* resistor, den delen av resistorn som inte inte är kopplad mot +3,3V. Denna avläsning kommer att vara hög såvida inte en tryckknapp sluter en krets mot jord i samverkan med en transistor (på nån utgång som skannas av mjukvaran).

7 Hårdvara/mjukvara interaktion - beskrivning

Funktionen för keypad_scan() tillsammans med knappsatsen interagerar på följande sätt. Nedan visas ett exempel för kolumn 1:

Funktionen är icke-blockerande och hanteringen att eventuellt stanna och vänta tills en knapp blir släppt, efter retur från keypad_scan(), lämnas till att skötas i logiken därifrån funktionen kallas.

T.ex. enligt:

```
keys key = keypad_scan();
while(key!=keyNone) {}
// continue the execution
```

Flera fördröjningar om 1 ms krävs då de snabba stig- och falltiderna som uppstår vid omslag mellan låg/hög orsakar EMC-störningar via tangentmatrisens ledningar. Fördröjningarna gör att dessa störningar hinner klinga av, men användaren hinner inte uppfatta fördröjningarna.

```
HAL_Delay(1); // eliminating EMC interference due to sharp rise/fall edges
```

I while loopen aktiveras utgången/kolumn till aktivt låg med *RESET:*

```
HAL_GPIO_WritePin(GPIOD, COL1_output_Pin, RESET);            // scan col 1
```

Ytterligare en fördröjning för att eliminerare EMC:

```
HAL_Delay(1); // eliminating EMC interference due to sharp rise/fall edges
```

Varje rad läses av och om nedtryckt knapp (1, 4, 7 eller *) så returneras en variabel av datatypen enum. T.ex för rad 4, *keyStar* returneras om knapp * tryckts in:

```
if(!HAL_GPIO_ReadPin(GPIOE, ROW1_input_Pin)) {return key1;}    //1
if(!HAL_GPIO_ReadPin(GPIOE, ROW2_input_Pin)) {return key4;}    //4
if(!HAL_GPIO_ReadPin(GPIOE, ROW3_input_Pin)) {return key7;}    //7
if(!HAL_GPIO_ReadPin(GPIOE, ROW4_input_Pin)) {return keyStar;} //*
```

Utgången/kolumn avaktiveras åter till hög med *SET*.

```
HAL_GPIO_WritePin(GPIOD, COL1_output_Pin, SET);       // stop scanning col 1
```

Om ingen knapp var intryckt under funktionsanropet så returneras *keyNone*, en variabel av datatypen enum. Detta kan användas i logiken som kallar på funktionen efter behov.

```
return keyNone;
```

I logiken som kallar på funktionen så måste återupprepade anrop göras tills önskad knapp returneras eller man av någon annan anledning vill avbryta skanningen. Ett exempel:

```
while(1) // condition to enter and stay in the while-loop
{
    key = keypad_scan_noblock();
    if(key==keyStar | key==keyA)
    {
         // do something here if * or A is pressed, and then exit the while-loop
         break;
    }
}
// when this code is reached, either * or A has been pressed
```

8 Testning

Centralt för tangentmatrisen är att fysiskt trycka in knapparna (indata) och avläsa förväntat resultat på displayen (utdata) eller läsa via seriegränssnittet kopplat mot dator. För att automatisera en testning av denna produkt så hade det krävts en robot som trycker in knapparna. Av den anledningen har testningen avgränsats till att en person som trycker in knappar och tolkar indata kontra utdata. Automatiserade tester var inte aktuellt.

En TDD (testdriven utveckling/development) metod användes vid utvecklingen av denna komponent. Det innebar att knapptryckningar avlästes via ett seriegränssnitt (en Arduino Uno som omvandlade UART till USB) till en PuTTy-terminal på datorn. Detta testförfarande gav helt avgörande underlag för analys av varför felaktiga värden erhölls.

TDD innebär att först skrivs testet (ett funktionsanrop) enligt det önskemål som finns, sedan skrivs och förfinas funktionen tills det att rätt värde returneras eller visas på skärmen/displayen. I vissa fall kan ett automatiserat testanropgöras om alla övriga komponenter är helt mjukvarumässiga, vilket inte var fallet med denna knappsats.

Det var viktigt att ha en loop som kontinuerlig gav uppdaterad avläsning av returvärdet från keypad_scan(). Detta för att första värdet kan vara korrekt, men efterföljande värden gav felaktiga avläsningar. Vissa knappar returnerade omväxlande

korrekta och okorrekta värden, om vart annat. Detta berodde på ”överhörning” till närliggande kablar. Nästa avsnitt behandlar förbättringar som kanske kan komma till rätta med dessa fel, utan att använda fördröjningar i koden.

9 Förbättringsförslag

Nedan föreslås förbättringsförslag som identifierats, men som ej kunnat åtgärdats i avsaknad av KTH:s laborationssal med mätutrustning.

Under utvecklingen upptäcktes att skanningen av tangentmatrisen skapade störande spänningar på närliggande kablar, vilket gav felaktiga avläsningar. Efter analys blev lösningen att lägga in fördröjningar i mjukvaran för att låta de störande spänningarna klinga av innan avläsningen skedde.

Vidare studier på denna lösning föreslås att inbegripa mätningar med oscilloskop för att ta reda på vilka kablar som påverkas av störningen, samt hur länge eventuella störningar finns på kablarna. Studenter på framtida kurser kan experimentera med att sätta ferritpärlor på kablarna för att minska dessa störningar eller möjligen sänka skanningshastigheten.

Bilaga A - Referenser

[1] ”Datasheet keypad”, https://www.electrokit.com/uploads/productfile/40220/40220004.pdf/ [Hämtad: 2020-04-03]

[2] ” UM1725: Description of STM32F4 HAL and LL drivers” *https://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf* [Hämtad: 2020-04-12]

Bilaga B – Mjukvara keypad

```
keys keypad_scan()
{
        keys key;

        //initiate all columns, in practice this makes the outputs floating,
        //but row-input are pulled high by input pull-up resistors,
        //unless output (columns) is RESET + button pressed.
        HAL_GPIO_WritePin(GPIOD, COL1_output_Pin, SET);
        HAL_GPIO_WritePin(GPIOD, COL2_output_Pin, SET);
        HAL_GPIO_WritePin(GPIOB, COL3_output_Pin, SET);
        HAL_GPIO_WritePin(GPIOB, COL4_output_Pin, SET);

        HAL_Delay(1);     // eliminating EMC interference due to sharp rise/fall edges
        HAL_GPIO_WritePin(GPIOD, COL1_output_Pin, RESET);              // scan column 1
        HAL_Delay(1);     // eliminating EMC interference due to sharp rise/fall edges
        if(!HAL_GPIO_ReadPin(GPIOE, ROW1_input_Pin)) {return key1;}    //1
        if(!HAL_GPIO_ReadPin(GPIOE, ROW2_input_Pin)) {return key4;}    //4
        if(!HAL_GPIO_ReadPin(GPIOE, ROW3_input_Pin)) {return key7;}    //7
        if(!HAL_GPIO_ReadPin(GPIOE, ROW4_input_Pin)) {return keyStar;} //*
        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOD, COL1_output_Pin, SET);                // stop scan column 1

        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOD, COL2_output_Pin, RESET);              // scan column 2
        HAL_Delay(1);
        if(!HAL_GPIO_ReadPin(GPIOE, ROW1_input_Pin)) {return key2;}    //2
        if(!HAL_GPIO_ReadPin(GPIOE, ROW2_input_Pin)) {return key5;}    //5
        if(!HAL_GPIO_ReadPin(GPIOE, ROW3_input_Pin)) {return key8;}    //8
        if(!HAL_GPIO_ReadPin(GPIOE, ROW4_input_Pin)) {return key0;}    //0
        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOD, COL2_output_Pin, SET);                // stop scan column 2

        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOB, COL3_output_Pin, RESET);              // scan column 3
        HAL_Delay(1);
        if(!HAL_GPIO_ReadPin(GPIOE, ROW1_input_Pin)) {return key3;}    //3
        if(!HAL_GPIO_ReadPin(GPIOE, ROW2_input_Pin)) {return key6;}    //6
        if(!HAL_GPIO_ReadPin(GPIOE, ROW3_input_Pin)) {return key9;}    //9
        if(!HAL_GPIO_ReadPin(GPIOE, ROW4_input_Pin)) {return keyHash;} //#
        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOB, COL3_output_Pin, SET);                // stop scan column 3

        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOB, COL4_output_Pin, RESET);              // scan column 4
        HAL_Delay(1);
        if(!HAL_GPIO_ReadPin(GPIOE, ROW1_input_Pin)) {return keyA;}    //A
        if(!HAL_GPIO_ReadPin(GPIOE, ROW2_input_Pin)) {return keyB;}    //B
        if(!HAL_GPIO_ReadPin(GPIOE, ROW3_input_Pin)) {return keyC;}    //C
        if(!HAL_GPIO_ReadPin(GPIOE, ROW4_input_Pin)) {return keyD;}    //D
        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOB, COL4_output_Pin, SET);                // stop scan column 4
        HAL_Delay(1);

        return keyNone;
}
```

I h-filen:

```
/* ********* Variable Declaration ********* */
typedef enum
{
        key0,           key1,           key2,           key3,
        key4,           key5,           key6,           key7,
        key8,           key9,           keyA,           keyB,
        keyC,           keyD,           keyHash,        keyStar,        keyNone
} keys;
```