Mini Rapport

Kungliga Tekniska Högskolan

Johan N. & Erik B.

Maj 2017

IE1206 Inbyggd Elektronik

Innehållsförteckning

1 Mini Rapport

1.1 Titel på Projektet

PIC Smartcard Klippkort

1.2 Beskrivning av tänkt ändamål

Produkt är ett klippkortssystem till ett tivoli där antalet åkturer för en person sparas med hjälp av ett smartcard. Varje gång en besökare tar en åktur så dras en åktur av med hjälp av en kortläsare som är monterad vid grinden.

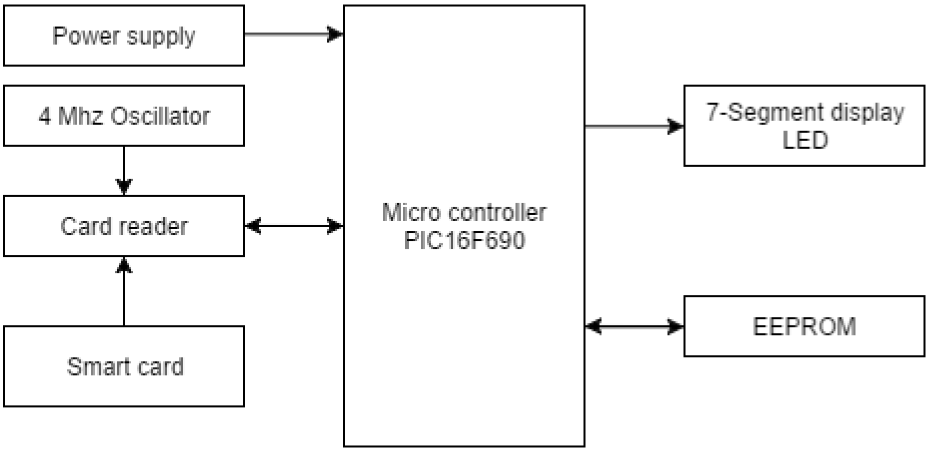
1.3 Beskrivning av prototyp

Programmet som är installerat på kortet skickar ut ett ID varje gång det sätts in i en kortläsare. Detta ID tas emot av en PIC-processor från kortläsaren och antalet åkturer för det ID minskas med ett. Alla värden och ID sparas i flashminnet på mikroprocessorn och inte på korten.

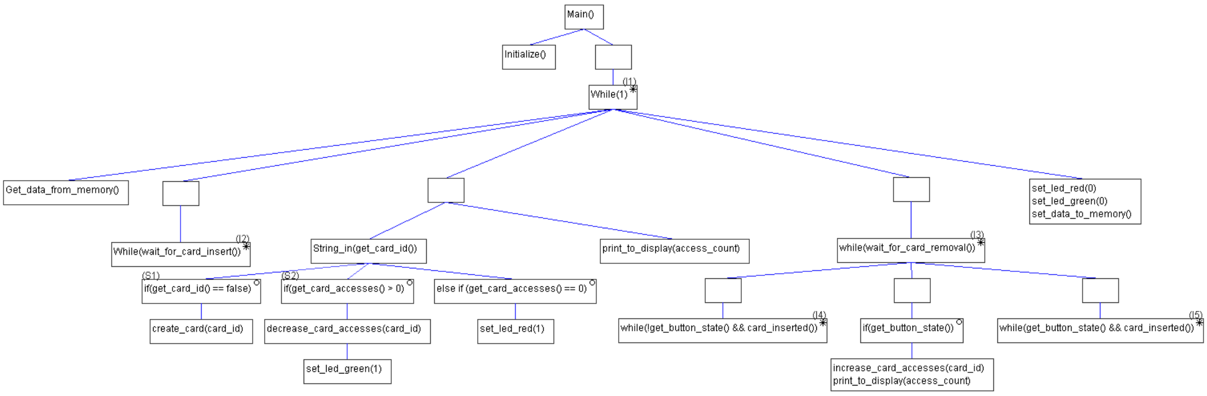
En 4 MHz kristall är kopplad till kortläsaren som kommunicerar med processorn. Processorn i sin tur kommunicerar med flashminnet och skriver/hämtar data under programmets gång. Antalet åkturer som är kvar visas sedan på en 7-segments display. Detta kan sedan kopplas till ytterligare komponenter som t.ex grind för ingång till en åkattraktion.

Med nuvarande processor finns det plats för två kort men det kan lätt ökas om processorn byts ut till en med mer RAM-minne.

1.4 Diagram



Figur 1: Blockdiagram


Main()
Initialize()
(I1)
While(1) *
Get_data_from_memory()
(I2)
While(wait_for_card_insert()) *
String_in(get_card_id())
print_to_display(access_count)
(S1)
if(get_card_id() == false) ○
(S2)
if(get_card_accesses() > 0) ○
else if (get_card_accesses() == 0) ○
create_card(card_id)
decrease_card_accesses(card_id)
set_led_green(1)
set_led_red(1)
(I3)
while(wait_for_card_removal()) *
(I4)
while(!get_button_state() && card_inserted()) *
if(get_button_state()) ○
increase_card_accesses(card_id)
print_to_display(access_count)
(I5)
while(get_button_state() && card_inserted()) *
set_led_red(0)
set_led_green(0)
set_data_to_memory()

Figur 2: Strukturdiagram

JSP diagrammet som visas är till processorns Main funktion. Först initialiseras alla värden som krävs och sedan börjar Main loopen som hela programmet körs i. All data (kort ID och antal åkturer) hämtas från EPROM sedan. Koden väntar den på att kortet sätts i och att ett ID ska skickas från kortet. När det är mottaget kollar processorn hur många åkturer det kortet har och drar bort en om det finns mer än noll. Efter det finns möjligheten att fylla på kortet med mer åkturer genom att klicka på en knapp. Detta är dock bara för att testa funktionalitet och är inte del av kretskortet som levereras eftersom användaren inte är tillåten att fylla på sitt kort själv. Sedan sparas all data till EPROM minnet och Main loopen startas om.

1.5 Kriterier för utfört projekt

Eftersom kommunikation sker mellan olika delar och inte bara inom processorn så är det extra viktigt att testa så att programmet och delarna fungerar som det är tänkt. När följande krav är uppfyllda är programmet i princip felfritt.

- Antal åkturer för ett ID minskas med ett varje gång kortet sätts i
- Ovanstående funktion fungerar felfritt vid alternering med olika kort
- Inget fel uppstår om man sätter in ett kort uppochned
- Om ett kort inte finns i flashminnet så läggs detta till och kan användas
- Om strömmen stängs av under användning ska flashminnet fortfarande visa de senast uppdaterade värdena som skrevs in innan det

Alla dessa punkter är uppfyllda och därför anses programmet som fullt fungerande.

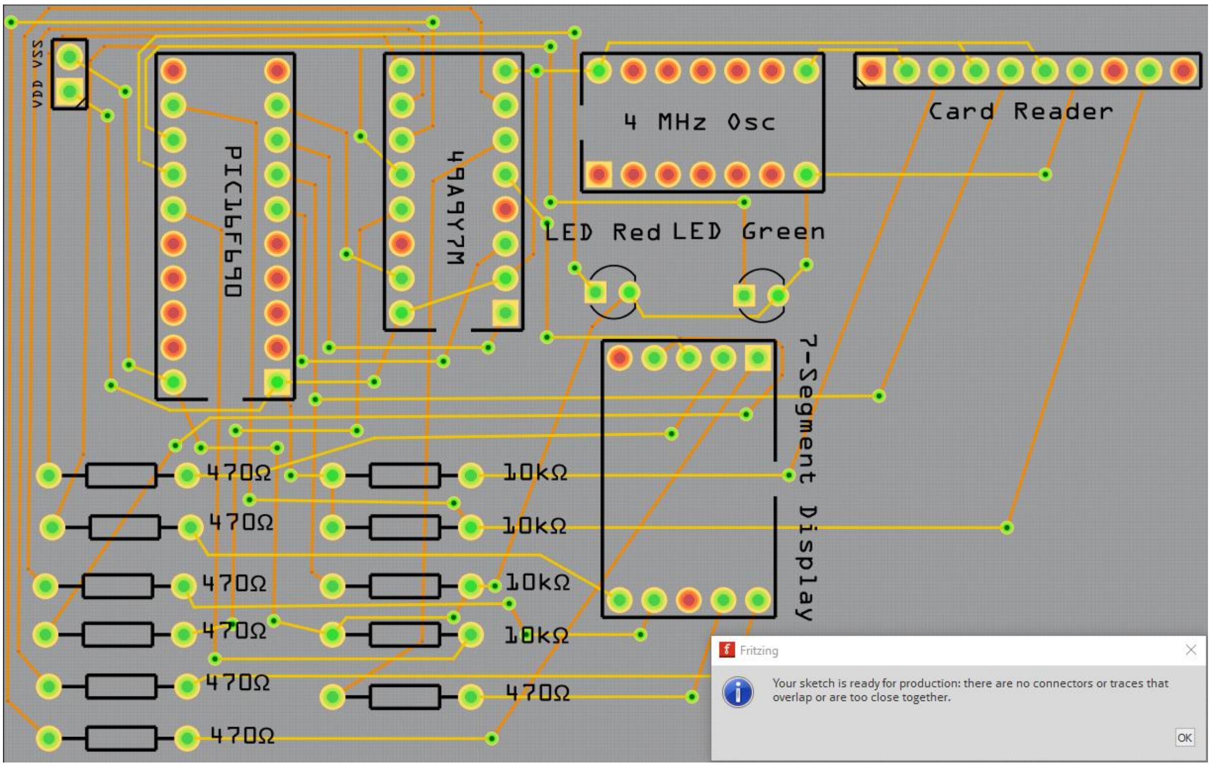
1.6 Publicerad kod

Koden till vårt projekt finner ni i följande GitHub repository.
https://github.com/vonNiklasson/KTH-smartcard

En kopia av koden och tillhörande ASCII grafik finner ni bland bilagorna.

Kompilatorn som används är CC5X Knudsen. Den finner ni här:
http://www.bknd.com/cc5x/

1.7 Fritzing



Your sketch is ready for production: there are no connectors or traces that overlap or are too close together.

Figur 2: PCB Diagram med Design Rules Check

1.8 Bilagor

Kod till processorn, smartcardet och ett testprogram finns här.
Om inget annat anges är koden skriven av oss.

ASCII grafik

```
/* *********************************** */
/*            HARDWARE                 */
/* *********************************** */


/*
   Use "PICkit2 UART Tool" as a 9600 Baud terminal to save data.
   with BitBanging routines.
           _____________  ____________
          |             \/            |
    +5V---|Vdd        16F690       Vss|---GND
          |RA5               RA0/(PGD)|bbTx ->- PK2Rx/PGD
          |RA4/AN3   AN1/REF/RA1/(PGC)|------<- PGC
          |RA3/!MCLR/(Vpp) RA2/AN2/INT|-<- U
          |RC5/CCP                 RC0|->- LED
          |RC4                     RC1|
          |RC3                     RC2|
          |RC6                     RB4|
          |RC7                  RB5/Rx|
          |RB7/Tx                  RB6|-<- SW
          |___________________________|


SmartCard contact
             __ __ __
       +5V  |C1|   C5| Gnd
            |__|   __|
      MCLR  |C2|  |C6|
            |__|  |__|
       OSC  |C3|  |C7| RB7/PGD I/O -><- Txd/Rxd half duplex
            |__|  |__|
            |C4|  |C8|
            |__|__|__|
Card Reader
            ______
            |C4  |
            |CLK |<--- 4MHz Osc
            |RST |
            |VCC |
            |SW2 |
            |SW1 |
            |GND |<--- GND
            |VPP |<--- +5V
            |I/O |<--- I/O
            |C8  |
            |____|

4MHz Oscillator
            ___________
    GND --->|GND   VDD|<--- +5V
            |         |
            |         |
            |         |
            |         |
            |  CLK out|<--- Clk out
            |_________|

7-Segment display

            ____________
            |b         h|
            |a         c|
            |VDD     VDD|
            |f         d|
            |g         e|
            |___________|

L293B 7-segment driver

            ________  __________
Pulse   --->|Clock  \/      VCC|<--- +5V
        --->|Disable      Reset|<---
        --->|En. disp.  Not Out|
            |En. Out      Out c|<--- c
    GND --->|GDN          Out b|<--- b
    f   --->|Out f        Out e|<--- e
    g   --->|Out g        Out a|<--- a
    GND --->|VDD          Out d|<--- d
            |__________________|
*/
```

Processor

```
/* This file contains the main
 * function and the program logic */

/* First include of chipkit-header */
#include "16F690.h"

#pragma config |= 0x00D4
#define MAX_STRING 16
#define NEW_ACCESS_COUNT 5
```

```
/* Allocate space for 7 cards */
char memory_cards[8 * 2];
char memory_card_count;

char create_card(char * card_id);

void get_data_from_memory(void);
void set_data_to_memory(void);

char get_card_offset(char * card_id);
char get_card_accesses(char card_offset);
void set_card_accesses(char card_offset, const char accesses);
char increase_card_accesses(char card_offset);
char decrease_card_accesses(char card_offset);

void reg_put_char(char data, char EEPROMadress);
char reg_get_char(char EEPROMadress);

void reg_put_word(const char * word, char reg_offset);
void reg_get_word(char * word, char reg_offset);

/*String related functions*/
void put_char(char d_out);
char get_char(void);
void string_in(char * string);
void string_out(const char * string);
bit compare_string(char * input_string, const char * candidate_string);

/*Hardware related functions*/
void wait_for_card_insert(void);
void wait_for_card_withdraw(void);
bit get_button_state(void);
void set_led_red(bit state);
void set_led_green(bit state);
void print_to_display(char val);
void delay(char millisec);

/*Initializing and registry clearing*/
void initialize(void);
void overrun_recover(void);

void main(void) {
    /* String to store text from card */
    char card_str[MAX_STRING];
    char card_offset;
    char card_access_count;
    bit has_access = 0;
    bit test1 = 0;
    bit test2 = 0;
    /* Initialize some code */
    initialize();

    /* Extended initialize */
    memory_cards[0] = 0;
    memory_card_count = 0;

    /* Loop forever, program logic below */
    while (1) {
        /* Reset the display */
        print_to_display(-1);
        get_data_from_memory();
```

```
/* Wait for card insertion */
while (PORTC.3 == 0);
delay(100); /* card debounce */
delay(50);  /* extra delay   */

/* ask the question */
string_out("Send the ID please\r\n");

delay(100); /* USART is buffered, so wait until all chars sent  */

/* empty the reciever FIFO, it's now full with garbage */
overrun_recover();

/* Get id from card, stored in card_str */
string_in(&card_str[0]);

/* Get the card offset id (if it exists) */
card_offset = get_card_offset(&card_str[0]);

if (card_offset == -1) {
    //Add new card
    card_offset = create_card(&card_str[0]);
}

card_access_count = get_card_accesses(card_offset);
nop();

//Check if any accesses are left on the card and makes you refill
if (card_access_count > 0) {
    set_led_green(1);
    nop();
    // Decrease the number of accesses
    card_access_count = decrease_card_accesses(card_offset);
}
else if (card_access_count == 0) {
    set_led_red(1);
    nop();
}

print_to_display(card_access_count);

//While card is inserted loop the
// ability to increase the number of access
while (PORTC.3 == 1) {
    //Wait for button presses and add 1 access
    while (!get_button_state() && PORTC.3 == 1);
    /* If the button is pressed, increase the access count */
    if (get_button_state()) {
        card_access_count = increase_card_accesses(card_offset);
        nop();
        print_to_display(card_access_count);
    }
    /* Wait for debounce of the button */
    while (get_button_state() && PORTC.3 == 1);
}

delay(10);

/* Reset the LED:s */
set_led_red(0);
set_led_green(0);
```

```
        delay(100);   /* card debounce */

        /* Write the new information to the memory */
        set_data_to_memory();
    }
}

/********************
     FUNCTIONS
     =========
*********************/

char get_card_offset(char * card_id) {
    int i, j, k;
    char tmpChar1, tmpChar2;
    /* Loops through the number of cards */
    for (i = 0; i < memory_card_count; i++) {
        /* Sets j to the card offset */
        j = i * 8;
        /* Iterate the chars in the card strings */
        for (k = 0; k < 7; k++) {
            /* Check if the string matches */
            tmpChar1 = memory_cards[j + k];
            tmpChar2 = card_id[k];
            if (tmpChar1 != tmpChar2) {
                break;
            }
        }
        /* If all chars matched, return the offset id */
        if (k == 7) {
            return i;
        }
    }

    /* Otherwise, return -1 */
    return -1;
}

//Get the number of accesses of a specific card
char get_card_accesses(char card_offset) {
    char current_accesses = memory_cards[(card_offset * 8) + 7];
    return current_accesses;
}

//Get the number of accesses of a specific card
void set_card_accesses(char card_offset, const char accesses) {
    memory_cards[(card_offset * 8) + 7] = accesses;
}

//Increase the number of accesses of a specific card by one
char increase_card_accesses(char card_offset) {
    char current_access = get_card_accesses(card_offset);

    if (current_access < 9) {
        current_access++;
    }

    set_card_accesses(card_offset, current_access);
    return current_access;
}
```

```
//Decrease the number of accesses of a specific card by one
char decrease_card_accesses(char card_offset) {
    char temp_offset = card_offset;
    memory_cards[(temp_offset * 8) + 7] = memory_cards[(temp_offset * 8) +
7] - 1;
    return memory_cards[(temp_offset * 8) + 7];
}

char create_card(char * card_id) {
    /* Get the new card offset */
    char card_offset = memory_card_count;
    /* Add 1 to the memory card count */
    memory_card_count = memory_card_count + 1;

    int i;
    char temp_char;
    for (i = 0; i < 7; i++) {
        temp_char = card_id[i];
        memory_cards[(8 * card_offset) + i] = temp_char;
    }
    /* Set the last byte to 0       (8 * card_offset) + i + 1   */
    memory_cards[(8 * card_offset) + i] = 0;

    /* Return the new card offset */
    return card_offset;
}

void get_data_from_memory(void) {
    /* Get how many cards that are saved in the memory */
    memory_card_count = reg_get_char(0);
    /* Temporay string for card data */
    char card[8];
    /* Initialize temp vars */
    int i, j, k;
    char temp_char;
    /* Count for how many cards that are stored */
    for (i = 0; i < memory_card_count; i++) {
        /* Get card i from memory */
        reg_get_word(&card[0], i);
        /* Get start offset for card in local string */
        k = i * 8;
        /* Loop through the next 8 bytes in the local string */
        for (j = 0; j < 8; j++) {
            temp_char = card[j];
            memory_cards[k + j] = temp_char;
        }
    }
}

void set_data_to_memory(void) {
    /* Store the number of saved cards */
    reg_put_char(memory_card_count, 0);
    /* Initialize temp vars */
    char i;
    /* Count for how many cards that are stored */
    for (i = 0; i < memory_card_count; i++) {
        reg_put_word(&memory_cards[0], i);
    }
}
```

```
void reg_put_word(const char * word, char reg_offset) {
    int offset = (reg_offset * 8) + 1;

    char c;
    int i;
    for (i = 0; i < 8; i++) {
        c = word[offset - 1 + i];
        reg_put_char(c, offset + i);
    }
}

void reg_get_word(char * word, char reg_offset) {
    int offset = (reg_offset * 8) + 1;

    char c;
    int i;
    for (i = 0; i < 8; i++) {
        c = reg_get_char(offset + i);
        word[i] = c;
    }
}

/* EXAMPLE CODE FROM IE1206 */
void reg_put_char(char data, char EEPROMadress) {
    /* Put char in specific EEPROM-adress */
    /* Write EEPROM-data sequence                          */
    EEADR = EEPROMadress; /* EEPROM-data adress 0x00 => 0x40 */
    EEDATA = data;        /* data to be written              */
    WREN = 1;             /* write enable                    */
    EECON2 = 0x55;        /* first Byte in comandsequence    */
    EECON2 = 0xAA;        /* second Byte in comandsequence   */
    WR = 1;               /* write                           */
    while( EEIF == 0) ;   /* wait for done (EEIF=1)          */
    WR = 0;
    WREN = 0;             /* write disable - safety first    */
    EEIF = 0;             /* Reset EEIF bit in software      */
    /* End of write EEPROM-data sequence                    */
}

/* EXAMPLE CODE FROM IE1206 */
char reg_get_char(char EEPROMadress) {
    /* Get char from specific EEPROM-adress */
    /* Start of read EEPROM-data sequence                */
    char temp;
    EEADR = EEPROMadress;  /* EEPROM-data adress 0x00 => 0x40  */
    EEPGD = 0;
    RD = 1;                /* Read                             */
    temp = EEDATA;
    RD = 0;
    return temp;          /* data to be read                   */
    /* End of read EEPROM-data sequence                       */
}

/* EXAMPLE CODE FROM IE1206 */
/* Sends one char */
void put_char(char d_out) {
    while (!TXIF);  /* wait until previus character transmitted   */
    TXREG = d_out;
    return; /* done */
}
```

```
/* EXAMPLE CODE FROM IE1206 */
/* Recieves one char */
char get_char(void) {
    char d_in = '\r';
    while (!RCIF && PORTC.3);  /* wait for character or card removal */
    if(!RCIF) return d_in;
    d_in = RCREG;
    return d_in;
}

/* EXAMPLE CODE FROM IE1206 */
void string_in(char * string) {
    char charCount, c;
    for(charCount = 0; ; charCount++) {
        c = get_char();         /* input 1 character         */
        string[charCount] = c;  /* store the character       */
        // put_char( c );       /* don't echo the character  */
        /* end of input */
        if((charCount == (MAX_STRING-1)) || (c=='\r')) {
            string[charCount] = '\0'; /* add "end of string" */
            return;
        }
    }
}

/* EXAMPLE CODE FROM IE1206 */
void string_out(const char * string) {
    char i, k;
    for(i = 0 ; ; i++) {
        k = string[i];
        if( k == '\0') return;   /* found end of string */
        put_char(k);
    }
    return;
}

/* EXAMPLE CODE FROM IE1206 */
bit compare_string(char * input_string, const char * candidate_string) {
    /* compares input with the candidate string */
    char i, c, d;
    for(i=0; ; i++) {
        c = input_string[i];
        d = candidate_string[i];
        if(d != c )     return 0;   /* no match    */
        if(d == '\0')   return 1;   /* exact match */
    }
}

/* Stall program til card is inserted */
void wait_for_card_insert(void) {
    while (PORTC.3 == 0);
}

/* Stall program til card is withdrawn */
void wait_for_card_withdraw(void) {
    while (PORTC.3 == 1);
}

bit get_button_state(void) {
    bit input;
    while(1) {
        input = PORTC.1;
        delay(10);
        if (input == PORTC.1) {
            return input;
        }
    }
}

void set_led_red(bit state) {
    PORTA.2 = state;
    nop();
}

void set_led_green(bit state) {
    PORTB.4 = state;
    nop();
}

void print_to_display(char val) {
    nop();
    /* Print hex-value to 7-segment display */
    char value = val;
    char i;
    if (value == -1) {
        PORTC.4 = 0;
        nop();
    }
    else if (value >= 0 && value < 10) {
        PORTC.4 = 1;
        nop();
        delay(1);
        PORTC.7 = 1;
        nop();
        delay(1);
        PORTC.7 = 0;
        nop();
        delay(1);
        for (i = 0; i < value; i++) {
            PORTC.6 = 1;
            nop();
            delay(1);
            PORTC.6 = 0;
            nop();
            delay(1);
        }
    }
}

/*  Delays a multiple of 1 milliseconds at 4 MHz
    using the TMR0 timer by B. Knudsen */
void delay(char millisec) {
    OPTION = 2;  /* prescaler divide by 8        */
    do {
        TMR0 = 0;
        while (TMR0 < 125);   /* 125 * 8 = 1000  */
    } while (--millisec > 0);
}

void initialize(void) {
    TRISA.0 = 1; /* RA0 not to disturb PK2 UART Tool */
    ANSEL.0 = 0; /* RA0 digital input */
    TRISA.1 = 1; /* RA1 not to disturb PK2 UART Tool */
    ANSEL.1 = 0; /* RA1 digital input */

    /* Initialize PIC16F690 serialcom port */
    /* One start bit, one stop bit, 8 data bit, no parity. 9600 Baud. */

    TXEN = 1;      /* transmit enable                  */
    SYNC = 0;      /* asynchronous operation           */
    TX9  = 0;      /* 8 bit transmission               */
    SPEN = 1;

    BRGH  = 0;     /* settings for 6800 Baud           */
    BRG16 = 1;     /* @ 4 MHz-clock frequency          */
    SPBRG = 25;

    CREN = 1;      /* Continuous receive               */
    RX9  = 0;      /* 8 bit reception                  */
    ANSELH.3 = 0;  /* RB5 digital input for serial_in  */

    /* More init */
    TRISC.3 = 1;  /* RC3 card contact is input       */
    ANSEL.7 = 0;  /* RC3 digital input               */
    TRISA.2 = 0;  /* RC2 Red LED for no access */
    PORTA.2 = 0;  /* RC2 initially off            */
    TRISB.4 = 0;  /* RB4 Green LED for access */
    PORTB.4 = 0;  /* RB4 initially off            */
    TRISC.1 = 1;  /* RC1 Button is set to input */
    ANSEL.5 = 0;  /* RC1 digital input               */

    //Display initialize
    TRISC.7 = 0;  /* RC7 Output to reset pin on display */
    PORTC.7 = 0;  /* RC7 initially off */
    TRISC.6 = 0;  /* RC6 Output to display clock */
    PORTC.6 = 0;  /* RC6 initially off */
    TRISC.4 = 0;  /* RC4 Enable display */
    PORTC.4 = 1;  /* RC4 on */
}

/* EXAMPLE CODE FROM IE1206 */
void overrun_recover(void) {
    char trash;
    trash = RCREG;
    trash = RCREG;
    CREN = 0;
    CREN = 1;
}
```

Smartcard

```
/* main_key_template.c  question and answer, compare strings     */
/* This program is for 16F84 Gold Card                         */

/*
SmartCard contact
```

```
             __ __ __
       +5V  |C1|   C5| Gnd
            |__|   __|
      MCLR  |C2|  |C6|
            |__|  |__|
       OSC  |C3|  |C7| RB7/PGD I/O -><- Txd/Rxd half duplex
            |__|  |__|
            |C4|  |C8|
            |__|__|__|

*/

/*
  SERIAL COMMUNICATION
  ============================
  One start bit, one stop bit, 8 data bit, no parity = 10 bit.
  Baudrate: 9600 baud => 104.167 usec. per bit.
  serial output PORTB.7  half duplex!
  serial input  PORTB.7  half duplex!
*/

#include "16F84.h"
#define MAX_STRING 16  /* string input max 15 characthers */
#pragma config |= 0x3ff1

/* Function prototypes                                    */
void initserial( void );
void putchar( char );
char getchar( void );
void string_out( const char * string );
void string_in( char * );
bit check_candidate( char * input_string, const char * candidate_string );
void delay( char );

void main( void)
{
   char i, c, d, charCount;
   char input_string[MAX_STRING]; /* 15 char buffer for input string */
   bit compare;
   delay(50);  /* delay to stabilize power */

   initserial();

   string_in( &input_string[0]);

   delay(150);     /* give the lock time to get ready */

   string_out("joherik\r\n");  /* Change this to the ID of the card */

   while(1) nop(); /* end of communication */
}

/********************
     FUNCTIONS
     =========
********************/

/* EXAMPLE CODE FROM IE1206 */
void initserial( void ) /* initialise serialcom port */
{
   PORTB.7 = 1;
   TRISB.7 = 1;  /* input mode */
}

/* EXAMPLE CODE FROM IE1206 */
void putchar( char d_out )  /* sends one char */
{
   char bitCount, ti;
   TRISB.7 = 0; /* output mode */
   PORTB.7 = 0; /* set startbit */
   for ( bitCount = 10; bitCount > 0 ; bitCount-- )
        {
         /* 104 usec at 3,58 MHz (5+27*3-1+9=104) */
         // ti = 27; do ; while( --ti > 0);
         /* 104 usec at 4 MHz (5+30*3-1+1+9=104)  */
          ti = 30; do ; while( --ti > 0); nop();
          Carry = 1;           /* stopbit                        */
          d_out = rr( d_out ); /* Rotate Right through Carry     */
          PORTB.7 = Carry;
        }
        nop2(); nop2();
   return; /* all done */
}

/* EXAMPLE CODE FROM IE1206 */
char getchar( void )  /* recieves one char */
{
   /* One start bit, one stop bit, 8 data bit, no parity = 10 bit. */
   /* Baudrate: 9600 baud => 104.167 usec. per bit.               */
   TRISB.7 = 1; /* set input mode */
   char d_in, bitCount, ti;
   while( PORTB.7 == 1 )  /* wait for startbit */ ;
   /* delay 1,5 bit is 156 usec                         */
   /* 156 usec is 156 op @ 4 MHz ( 5+47*3-1+2+9=156)    */
   ti = 47; do ; while( --ti > 0); nop2();
   for( bitCount = 8; bitCount > 0 ; bitCount--)
       {
        Carry = PORTB.7;
        d_in = rr( d_in);  /* rotate carry */
        /* delay 1 bit is 104 usec                       */
        /* 104 usec is 104 op @ 4 MHz (5+30*3-1+1+9=104) */
        ti = 30; do ; while( --ti > 0); nop();
        }
   return d_in;
}

/* EXAMPLE CODE FROM IE1206 */
void string_in( char * input_string )
{
   char charCount, c;
   for( charCount = 0; ; charCount++ )
       {
         c = getchar( );     /* input 1 character             */
         input_string[charCount] = c;  /* store the character */
         //putchar( c );     /* don't echo the character      */
         if( (charCount == (MAX_STRING-1))||(c=='\r' )) /* end of input
*/
           {
             input_string[charCount] = '\0';  /* add "end of string"
*/
             return;
           }
       }
}

/* EXAMPLE CODE FROM IE1206 */
void string_out(const char * string)
{
  char i, k;
  for(i = 0 ; ; i++)
   {
     k = string[i];
     if( k == '\0') return;   /* found end of string */
     putchar(k);
   }
  return;
}

/* EXAMPLE CODE FROM IE1206 */
bit check_candidate( char * input_string, const char * candidate_string )
{
   /* compares input buffer with the candidate string */
   char i, c, d;
   for(i=0; ; i++)
     {
       c = input_string[i];
       d = candidate_string[i];
       if(d != c ) return 0;       /* no match    */
         if( d == '\0' ) return 1; /* exact match */
     }
}

/* EXAMPLE CODE FROM IE1206 */
void delay( char millisec)
/*
  Delays a multiple of 1 milliseconds at 4 MHz
  using the TMR0 timer
*/
{
    OPTION = 2;  /* prescaler divide by 8        */
    do  {
        TMR0 = 0;
        while ( TMR0 < 125)   /* 125 * 8 = 1000  */
            ;
    } while ( -- millisec > 0);
}
```

Testprogram i Fritzing

```
/*TEST FILE FOR PROCESSOR*/

/* This file contains the main
 * function and the program logic */
/* First include of chipkit-header */
#include "16F690.h"

#pragma config |= 0x00D4
#define MAX_STRING 16

/*String related functions*/
```

```
char get_char(void);
void string_in(char * string);
void string_out(const char * string);

/*Hardware related funcitons */
void set_led_red(bit state);
void set_led_green(bit state);
void print_to_display(char val);
void delay(char millisec);

/*Initializing and registry clearing*/
void initialize(void);
void overrun_recover(void);

void main(void) {
    /* String to store text from card */
    char card_str[MAX_STRING];

    /* Initialize some code */
    initialize();

    /* Loop forever, program logic below */
    while (1) {
        /* Wait for card insertion */
        while (PORTC.3 == 0);
        delay(100); /* card debounce */
        delay(50);  /* extra delay   */

        /* ask the question */
        string_out("Send the ID please\r\n");

        delay(100); /* USART is buffered, so wait until all chars sent  */

        /* empty the reciever FIFO, it's now full with garbage */
        overrun_recover();

        /* Get id from card, stored in card_str */
        string_in(&card_str[0]);

        set_led_red(1);
        set_led_green(1);

        //While card is inserted, loop
        while (PORTC.3 == 1) {
            if(i < 10)
                print_to_display(i);
            else
                i = 0;
            delay(1000);
        }

        delay(10);

        /* Reset the LED:s */
        set_led_red(0);
        set_led_green(0);
        delay(100);   /* card debounce */
    }
}

/********************
```

```
     FUNCTIONS
     =========
*********************/

/* EXAMPLE CODE FROM IE1206 */
void string_in(char * string) {
    char charCount, c;
    for(charCount = 0; ; charCount++) {
        c = get_char();         /* input 1 character          */
        string[charCount] = c;  /* store the character        */
        // put_char( c );       /* don't echo the character  */
        /* end of input */
        if((charCount == (MAX_STRING-1)) || (c=='\r')) {
            string[charCount] = '\0'; /* add "end of string" */
            return;
        }
    }
}


/* EXAMPLE CODE FROM IE1206 */
void string_out(const char * string) {
    char i, k;
    for(i = 0 ; ; i++) {
        k = string[i];
        if( k == '\0') return;   /* found end of string */
        put_char(k);
    }
    return;
}

void set_led_red(bit state) {
    PORTA.2 = state;
    nop();
}

void set_led_green(bit state) {
    PORTB.4 = state;
    nop();
}

void print_to_display(char val) {
    nop();
    /* Print hex-value to 7-segment display */
    char value = val;
    char i;
    if (value == -1) {
        PORTC.4 = 0;
        nop();
    }
    else if (value >= 0 && value < 10) {
        PORTC.4 = 1;
        nop();
        delay(1);
        PORTC.7 = 1;
        nop();
        delay(1);
        PORTC.7 = 0;
        nop();
        delay(1);
        for (i = 0; i < value; i++) {
            PORTC.6 = 1;
            nop();
            delay(1);
            PORTC.6 = 0;
            nop();
            delay(1);
        }
    }
}

/*  Delays a multiple of 1 milliseconds at 4 MHz
    using the TMR0 timer by B. Knudsen */
void delay(char millisec) {
    OPTION = 2;  /* prescaler divide by 8         */
    do {
        TMR0 = 0;
        while (TMR0 < 125);   /* 125 * 8 = 1000  */
    } while (--millisec > 0);
}

void initialize(void) {
    TRISA.0 = 1; /* RA0 not to disturb PK2 UART Tool */
    ANSEL.0 = 0; /* RA0 digital input */
    TRISA.1 = 1; /* RA1 not to disturb PK2 UART Tool */
    ANSEL.1 = 0; /* RA1 digital input */

    /* Initialize PIC16F690 serialcom port */
    /* One start bit, one stop bit, 8 data bit, no parity. 9600 Baud. */

    TXEN = 1;      /* transmit enable                   */
    SYNC = 0;      /* asynchronous operation            */
    TX9  = 0;      /* 8 bit transmission                */
    SPEN = 1;

    BRGH  = 0;     /* settings for 6800 Baud            */
    BRG16 = 1;     /* @ 4 MHz-clock frequency           */
    SPBRG = 25;

    CREN = 1;      /* Continuous receive                */
    RX9  = 0;      /* 8 bit reception                   */
    ANSELH.3 = 0;  /* RB5 digital input for serial_in   */

    /* More init */
    TRISC.3 = 1;  /* RC3 card contact is input       */
    ANSEL.7 = 0;  /* RC3 digital input               */
    TRISA.2 = 0;  /* RC2 Red LED for no access */
    PORTA.2 = 0;  /* RC2 initially off            */
    TRISB.4 = 0;  /* RB4 Green LED for access */
    PORTB.4 = 0;  /* RB4 initially off            */
    TRISC.1 = 1;  /* RC1 Button is set to input */
    ANSEL.5 = 0;  /* RC1 digital input               */

    //Display initialize
    TRISC.7 = 0;  /* RC7 Output to reset pin on display */
    PORTC.7 = 0;  /* RC7 initially off */
    TRISC.6 = 0;  /* RC6 Output to display clock */
    PORTC.6 = 0;  /* RC6 initially off */
    TRISC.4 = 0;  /* RC4 Enable display */
    PORTC.4 = 1;  /* RC4 on */
}
```

```
/* EXAMPLE CODE FROM IE1206 */
void overrun_recover(void) {
    char trash;
    trash = RCREG;
    trash = RCREG;
    CREN = 0;
    CREN = 1;
}
```

Test för smartcard

```
/* main_key_template.c  question and answer, compare strings     */
/* This program is for 16F84 Gold Card                  */
/*
SmartCard contact

             __ __ __
       +5V  |C1|   C5| Gnd
            |__|   __|
      MCLR  |C2|  |C6|
            |__|  |__|
       OSC  |C3|  |C7| RB7/PGD I/O -><- Txd/Rxd half duplex
            |__|  |__|
            |C4|  |C8|
            |__|__|__|

*/

/*
  SERIAL COMMUNICATION
  ============================
  One start bit, one stop bit, 8 data bit, no parity = 10 bit.
  Baudrate: 9600 baud => 104.167 usec. per bit.
  serial output PORTB.7  half duplex!
  serial input  PORTB.7  half duplex!
*/

#include "16F84.h"
#define MAX_STRING 16  /* string input max 15 charachters */
#pragma config |= 0x3ff1

/* Function prototypes                                   */
void initserial( void );
void putchar( char );
char getchar( void );
void string_out( const char * string );
void string_in( char * );
bit check_candidate( char * input_string, const char * candidate_string );
void delay( char );

void main( void)
{
   char i, c, d, charCount;
   char input_string[MAX_STRING]; /* 15 char buffer for input string */
   bit compare;
   delay(50);  /* delay to stabilize power */

   initserial();

   string_in( &input_string[0]);
   delay(150);     /* give the lock time to get ready */

   string_out("joherik\r\n");  /* Change this to the ID of the card */

   while(1) nop(); /* end of communication */
}

/********************
     FUNCTIONS
     =========
*********************/

/* EXAMPLE CODE FROM IE1206 */
void initserial( void ) /* initialise serialcom port */
{
   PORTB.7 = 1;
   TRISB.7 = 1;  /* input mode */
}

/* EXAMPLE CODE FROM IE1206 */
void putchar( char d_out )  /* sends one char */
{
   char bitCount, ti;
   TRISB.7 = 0; /* output mode */
   PORTB.7 = 0; /* set startbit */
   for ( bitCount = 10; bitCount > 0 ; bitCount-- )
        {
         /* 104 usec at 3,58 MHz (5+27*3-1+9=104) */
         // ti = 27; do ; while( --ti > 0);
         /* 104 usec at 4 MHz (5+30*3-1+1+9=104)  */
          ti = 30; do ; while( --ti > 0); nop();
          Carry = 1;           /* stopbit                        */
          d_out = rr( d_out ); /* Rotate Right through Carry     */
          PORTB.7 = Carry;
        }
        nop2(); nop2();
   return; /* all done */
}

/* EXAMPLE CODE FROM IE1206 */
char getchar( void )  /* recieves one char */
{
   /* One start bit, one stop bit, 8 data bit, no parity = 10 bit. */
   /* Baudrate: 9600 baud => 104.167 usec. per bit.                */
   TRISB.7 = 1; /* set input mode */
   char d_in, bitCount, ti;
   while( PORTB.7 == 1 )  /* wait for startbit */ ;
   /* delay 1,5 bit is 156 usec                         */
   /* 156 usec is 156 op @ 4 MHz ( 5+47*3-1+2+9=156)    */
   ti = 47; do ; while( --ti > 0); nop2();
   for( bitCount = 8; bitCount > 0 ; bitCount--)
       {
        Carry = PORTB.7;
        d_in = rr( d_in);  /* rotate carry */
        /* delay 1 bit is 104 usec                       */
        /* 104 usec is 104 op @ 4 MHz (5+30*3-1+1+9=104) */
        ti = 30; do ; while( --ti > 0); nop();
        }
   return d_in;
}
```

```
/* EXAMPLE CODE FROM IE1206 */
void string_in( char * input_string )
{
   char charCount, c;
   for( charCount = 0; ; charCount++ )
       {
         c = getchar( );     /* input 1 character             */
         input_string[charCount] = c;   /* store the character */
         //putchar( c );     /* don't echo the character      */
         if( (charCount == (MAX_STRING-1))||(c=='\r' )) /* end of input
*/
           {
             input_string[charCount] = '\0';  /* add "end of string"
*/
             return;
           }
       }
}

/* EXAMPLE CODE FROM IE1206 */
void string_out(const char * string)
{
  char i, k;
  for(i = 0 ; ; i++)
   {
     k = string[i];
     if( k == '\0') return;   /* found end of string */
     putchar(k);
   }
  return;
}



/* EXAMPLE CODE FROM IE1206 */
bit check_candidate( char * input_string, const char * candidate_string )
{
   /* compares input buffer with the candidate string */
   char i, c, d;
   for(i=0; ; i++)
     {
       c = input_string[i];
       d = candidate_string[i];
       if(d != c ) return 0;       /* no match    */
         if( d == '\0' ) return 1; /* exact match */
     }
}

/* EXAMPLE CODE FROM IE1206 */
void delay( char millisec)
/*
  Delays a multiple of 1 milliseconds at 4 MHz
  using the TMR0 timer
*/
{
    OPTION = 2;  /* prescaler divide by 8        */
    do  {
        TMR0 = 0;
        while ( TMR0 < 125)   /* 125 * 8 = 1000  */
            ;
    } while ( -- millisec > 0);
}
```