Pinlock: Dokumentation

Mikrocontroller-Seminar

Leonard Kugis, Mattes Melius

20. März 2021

Inhaltsverzeichnis

1 Idee

Für unser Mikrocontroller-Projekt haben wir uns für ein vierstelliges Zahlenschloss entschieden. Dieses soll nach Eingabe eines validen Codes in einen "GeöffnetZustand wechselt und im "GeschlossenZustand verweilt, wenn der Code inkorrekt war. Zudem soll dem Benutzer die Möglichkeit gegeben werden einen persönlich gewählten Code einzuspeichern und den alten zu Überschreiben.

2 Hardware

2.1 Platinenaufbau

Unser Mikrocontroller liegt auf einer 100mmx80mm großen Platine welche über zwei Kupferlagen verfügt. Als Hauptbaustein wird der ATmega16A benutzt. Des weiteren befinden sich auf unserer Platine neben üblichen Grundbausteinen eine Ausgabe- sowie eine ëxterneËingabe-Einheit, als auch Status-LEDs.

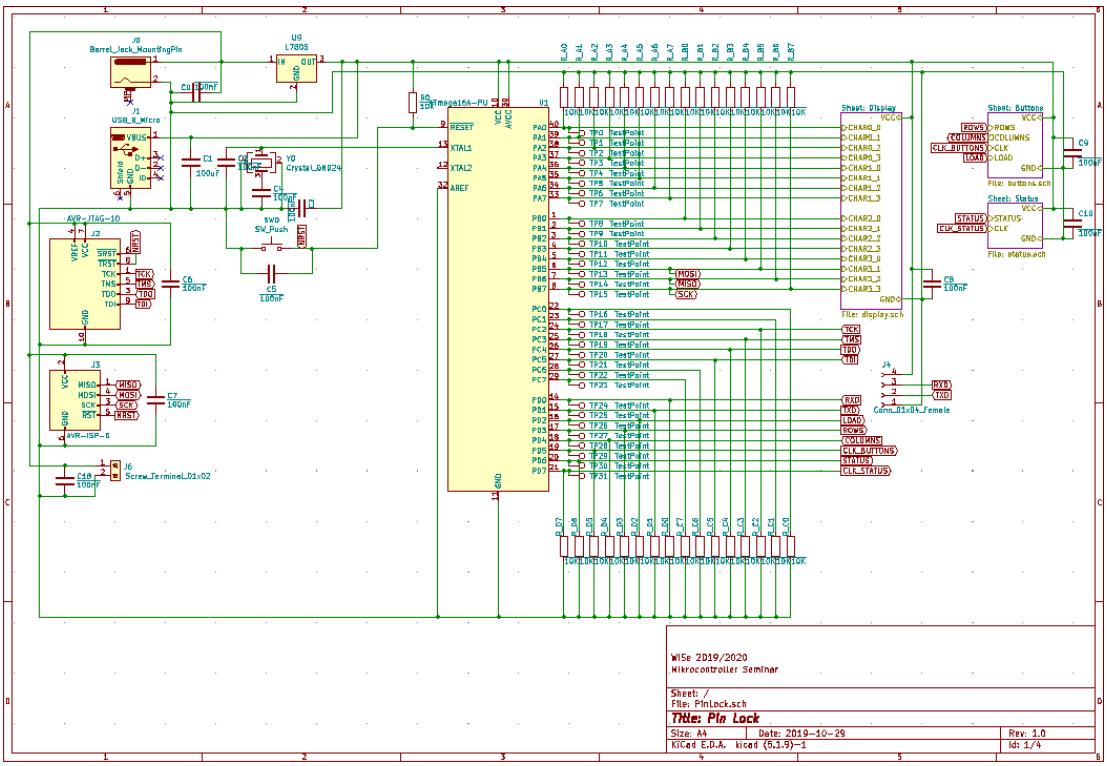

Abbildung 1: Schaltbild Hauptschaltung

2.2 Mikrocontroller ATmega16A

Der ATmega16A ist ein 8-Bit CMOS Controller, welcher bei einer Spannung von 2,7- 5V arbeitet. In unserem Projekt nutzen wir nicht alle Features des Bausteins. Um unsere Schaltung zu realisieren, brauchen wir die GPIO, die eingebaute AVR-CPU und das EEPROM.

Mit der GPIO und der AVR-CPU ist es uns möglich die Inputs aus der Eingabematrix in Sinnvolle werte umzurechnen und in den Registern der CPU einzuspeichern. Diese werden nun auch an die Display-Units weitergegeben, welche diese anzeigen können um dem Benutzer visuell darzustellen, welche Zahlen eingegeben wurden, ob richtige Eingabe erfolgt ist oder ob ein Code invalide war. Auch der Zustand (geöffnet/geschlossen) wird in den Registern des ATmega's gespeichert und auch an die Status-LEDs gesendet.

Im EEPROM wird der valide Code abgespeichert, sodass dieser auch nach einen Neustart zur Verfügung steht und nicht gelöscht wird. So ist es dem Nutzer möglich den Code bis zu 100,000 zu ändern.

2.3 Sonstige Komponenten

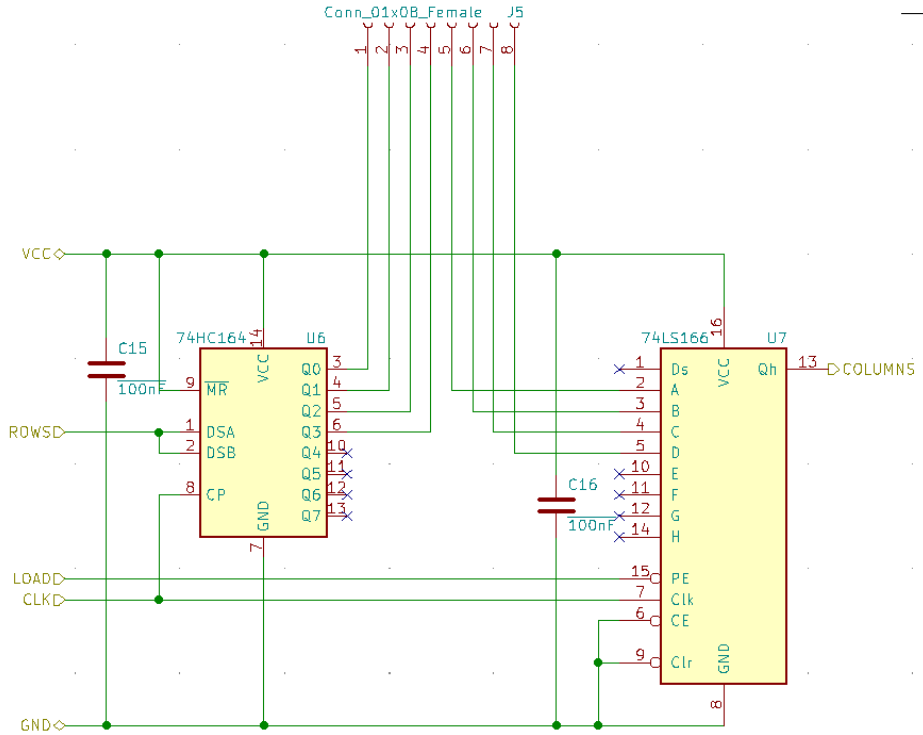

Abbildung 2: Schaltbild Eingabecontroller

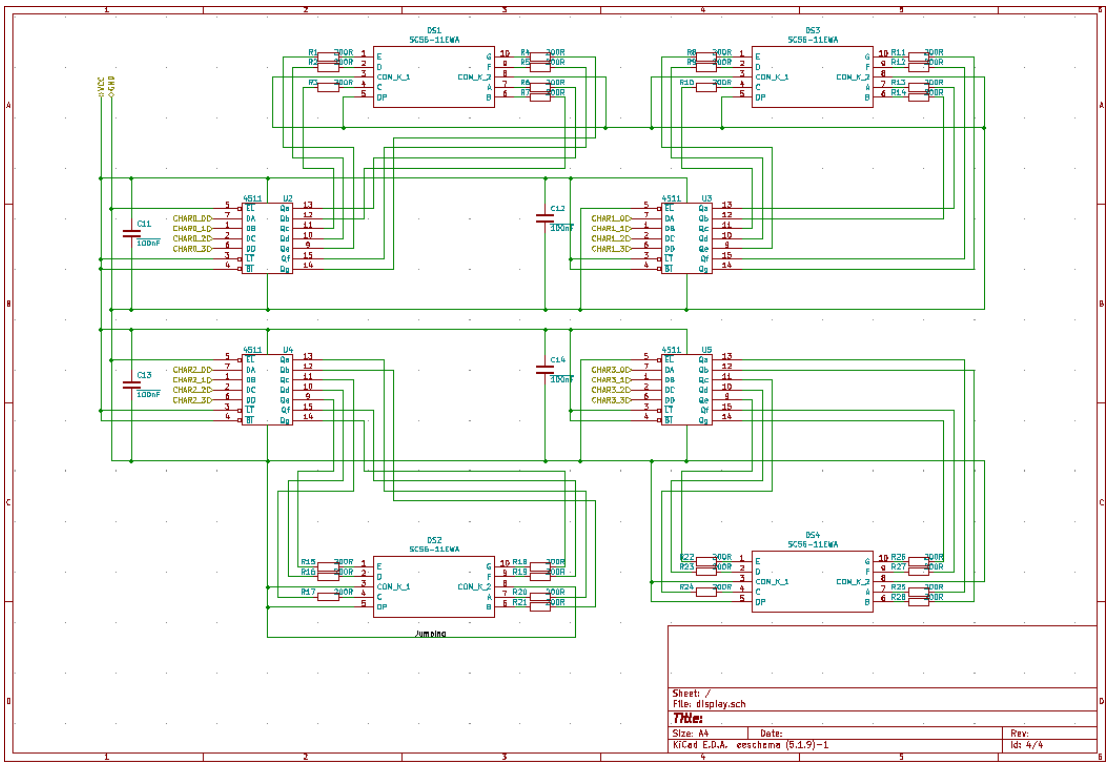

Abbildung 3: Schaltbild Display-Units

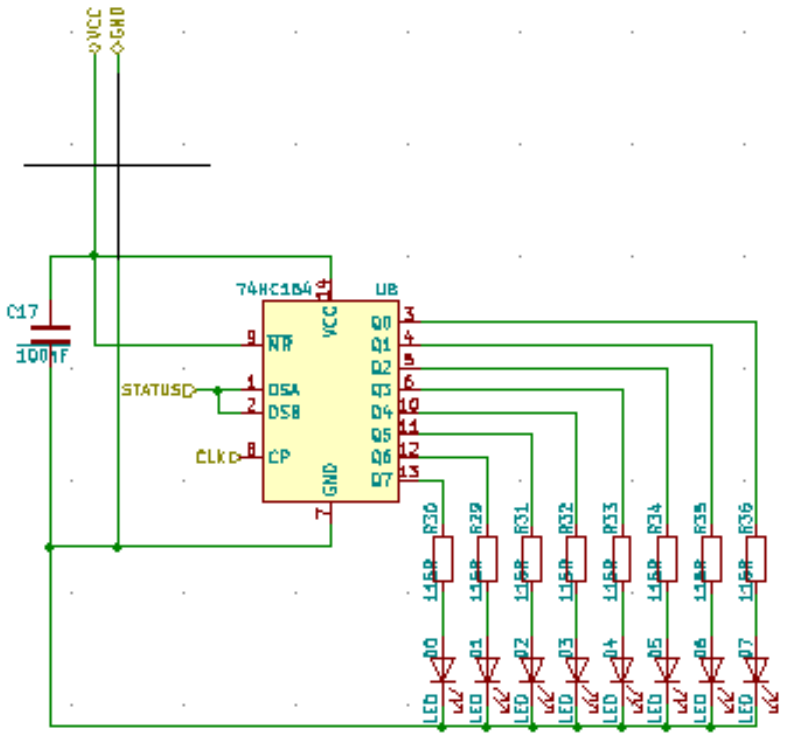

Abbildung 4: Schaltbild Status-LEDs

3 Software

Das Programm wurde in AVR-Assembler geschrieben, mit Assemblerdirektiven für den *Avra*-Assembler. Grundsätzlich wurden für dieses Programm die in Tabelle 1 Register-Neudefinitionen vorgenommen, um den Code lesbarer zu gestalten.

3.1 Grundprinzip

Nach Zurücksetzen des Mikrocontrollers durchläuft das Programm eine Initialisierungsroutine. Diese beinhaltet im Wesentlichen folgende Schritte:

1. Initialisierung des Stackpointers mit den Grenzen des verfügbaren RAMs
2. Konfiguration der GPIO-Ports
3. Initialisierung der Displays
4. Lesen des Zugangscodes aus dem EEPROM

Anschließend wird eine endlose Hauptschleife durchlaufen, die folgende Aktionen periodisch ausführt:

1. Warte 12240 Zyklen.

ATmega16A Register	Neu definierte Registerbezeichnung	Funktion
r16	tmp0	Temporäres Pufferregister 0
r17	par0	Parameterregister 0
r18	par1	Parameterregister 1
r19	tmp1	Temporäres Pufferregister 1
r20	tmp2	Temporäres Pufferregister 2
r21	ret0	Rückgaberegister 0
r22	ret1	Rückgaberegister 1
r23	par2	Parameterregister 2
r24	par3	Parameterregister 3
r25	par4	Parameterregister 4
r26	ret2	Rückgaberegister 2
r27	heart	Debug
r28	dig0	Pufferregister für Ziffern 0 und 1
r29	dig1	Pufferregister für Ziffern 2 und 3

Tabelle 1: Neu definierte Registerbezeichnungen für Register des ATmega16A

2. Lese alle Tasten der Eingabematrix aus.
3. Ermittle Änderungen des neuen Tastenstatus zu dem alten Tastenstatus.
4. Bei positiver Flanke (Taste war vorher nicht gedrückt und ist jetzt gedrückt): Behandle den Tastendruck bezüglich des aktuellen Zustands (s.u.).
5. Puffere den neuen Tastenstatus.

Die einzelnen Routinen, die dabei aufgerufen werden, werden genauer in Sektion 3.2 erläutert.

Die Behandlung der (neu) gedrückten Taste bezüglich des aktuellen Zustandes erfolgt nach der in Abbildung 5 dargestellten *Finite-State-Machine* (FSM). Die Zustände sind dabei nach folgendem Schema benannt:

`A`: *Authenticating*, Authentifizierung findet gerade durch Codeeingabe statt
`AS`: *Authentication Submitted*, Code wurde bestätigt und wird überprüft
`AD`: *Authenticated*, Benutzer ist authentifiziert und hat Zugriff
`C`: *Change*, Benutzer ändert den Zugriffscode

Dabei folgt die Notation der Transitionsbedingungen dem Schema `<Taste> / <Wesentliche Aktionen>`. Die Tastendarstellungen innerhalb des Programmes unterliegen dabei der Zuordnung in Tabelle 2.

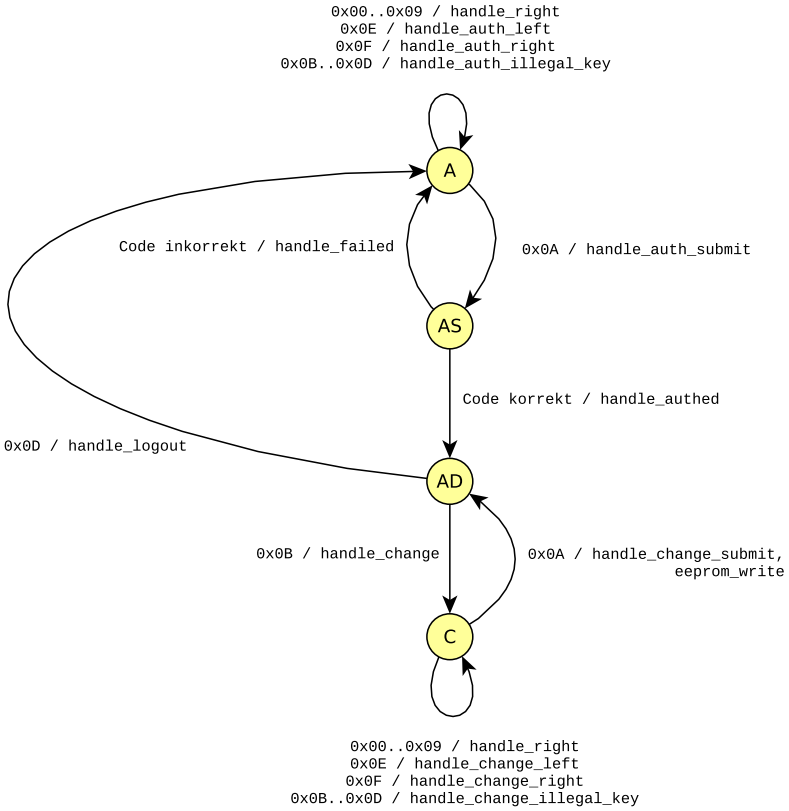

Abbildung 5: FSM der Behandlung von Tastendrücken (schematisch)

Taste extern	Taste intern
`0..9`	`0x00..0x09`
`A..D`	`0x0A..0x0D`
`*`	`0x0E`
`#`	`0x0F`

Tabelle 2: Zuordnung von externen zu internen Tastendarstellungen

3.2 Routinen

3.2.1 `wait`

Parameter: `par0`: Wartezyklen Stufe 1,
`par1`: Wartezyklen Stufe 2,
`par2`: Wartezyklen Stufe 3

Rückgabe: -

Warte über drei verschachtelte Schleifen mit verschiedenen Durchlaufzahlen. Die gesamte, zu wartende Zyklendauer beträgt somit $\Delta t_{\text{ges}} = \texttt{par0} \cdot \texttt{par1} \cdot \texttt{par2}$.

3.2.2 `write_code`

Parameter: -
Rückgabe: -
Schreibe statisch die Werte 0x01 und 0x23 (entspricht der Zahlenkombination `0123`) in das EEPROM an die Adressen 0x00 und 0x01.

3.2.3 `load_code`

Parameter: -
Rückgabe: `ret0`: Ziffern 0 und 1,
`ret1`: Ziffern 2 und 3
Lese die gespeicherte Zahlenkombination aus dem EEPROM (Adressen 0x00 und 0x01) aus.

3.2.4 `shift_out`

Parameter: `par0`: Zu schreibendes Byte,
`par1..par2`: Adresse zur Funktion, um 0 zu schreiben,
`par3..par4`: Adresse zur Funktion, um 1 zu schreiben
Rückgabe: -
Schreibe ein Byte Bit für Bit durch Funktionsaufrufe, welche das Schreiben des 0- und 1-Bits implementieren. Das können beispielsweise Funktionen sein, die 0 bzw. 1 an ein Schieberegister anlegen.

3.2.5 `shift_in`

Parameter: `par0..par1`: Adresse zur Funktion, um einen Clockzyklus zu durchlaufen,
`par2..par3`: Adresse zur Funktion, das zugehörige Bit zu lesen
Rückgabe: `ret0`: Gelesenes Byte
Lese ein Byte Bit für Bit aus einem Schieberegister. Das Durchlaufen eines Clockzyklus und Lesen des Bits ist in übergebenen Funktionen implementiert.

3.2.6 `eeprom_write`

Parameter: `par0..par1`: Zieladresse,
`par2`: Zu schreibender Wert
Rückgabe: -
Schreibt ein Byte an die gegebene Adresse in das interne EEPROM.

3.2.7 `eeprom_read`

Parameter: `par0..par1`: Adresse
Rückgabe: `ret0`: Gelesener Wert
Liest ein Byte von der gegebenen Adresse im internen EEPROM.

3.2.8 `status`

Parameter: `par0`: Statusbits
Rückgabe: -
Gibt die gegebenen Statusbits über die LEDs aus. Dabei wird das Schieberegister dafür benutzt.

3.2.9 `display`

Parameter: `par0`: Displaynummer,
`par1`: Darzustellende Ziffer
Rückgabe: -
Zeige die gegebene Ziffer auf dem gegebenen Display.

3.2.10 `set_digit`

Parameter: `par0`: Ziffernstelle,
`par1`: Ziffer
Rückgabe: -
Schreibt die Ziffer an die Ziffernstelle des internen Puffers.

3.2.11 `get_digit`

Parameter: `par0`: Ziffernstelle
Rückgabe: `ret0`: Ziffer an der Stelle
Liest die Ziffer der gegebenen Stelle von dem internen Puffer.

3.2.12 `input`

Parameter: -
Rückgabe: `ret0`: Tastenstatus mit Belegung: `0852*741`,
`ret1`: Tastenstatus mit Belegung: `DCBA#963`
Liest den Tastenstatus aller Tasten aus. Ein 1-Bit entspricht einer gedrückten, ein 0-Bit einer nicht gedrückten Taste. Das Auslesen der Tasten der Eingabematrix erfolgt über die Schieberegister: eines (Serial-In-Parallel-Out) zur Selektion der auszulesenden Zeile und eines (Parallel-In-Serial-Out) zum eigentlichen Auslesen selbiger. Dies wird innerhalb der Ausleseroutine mit jeder Zeile gemacht, um gedrückte Tasten zu identifizieren.

3.2.13 `translate`

Parameter: `par0`: Tastenstatus Bits `7..0`,
`par1`: Tastenstatus Bits `15..8`
Rückgabe: `ret0`: Übersetzter Tastencode
Übersetzt den Tastenstatus in die entsprechende, interne Tastendarstellung. Dabei wird nach der Zuordnung aus Tabelle 2 vorgegangen.

3.2.14 `handle`

Parameter: `par0`: Aktueller Zustand,
`par1`: Tastencode
Rückgabe: `ret0`: Neuer Zustand

Behandle den Tastencode unter dem aktuellen Zustand. Das Zustandsfeld hat folgende Belegung (ein Zeichen entspricht einem Bit):

```
CSSSSSDD
```

`0`: Reserviert
`C`: Pufferbit für Blinken der ausgewählten Stelle
`S`: Zustandsinformationen
`D`: Aktuell ausgewähltes Display

Die Behandlung erfolgt nach dem in Abbildung 5 dargestellten Schema.

4 Bedienung

Das Zurücksetzen des Systems ist man in der Authentifizierungsphase. In dieser ist die richtige Ziffernkombination einzugeben, welcher vorher gespeichert wurde. Der Standardcode ist `0123`. Es stehen dabei folgende Optionen zur Auswahl:

`0..9`: Ziffer eingeben
`A`: Eingabe bestätigen
`*`: Zeigerposition nach links schieben
`#`: Zeigerposition nach rechts schieben

Wurde die falsche Ziffernkombination eingegeben und bestätigt, erlischen alle Displays für wenige Sekunden und es ist eine erneute Eingabe möglich.
Wurde die richtige Ziffernkombination eingegeben und bestätigt, wird dies mit setzen aller Displays auf 0 und erlischen des Zeigers angezeigt. Man ist nun authentifiziert, was einem Zugriff auf das System darstellt. In diesem Zustand stehen folgende Funktionen zur Auswahl:

`B`: Ändern des Zugangscodes
`D`: Ausloggen

Der Zugangscode lässt sich nach Drücken von `B` mit der selben Steuerung wie bei der Zifferneingabe in der Authentifizierungsphase ändern, wobei `A` dem Bestätigen der Änderung entspricht. Dabei wird der Zugangscode in den persistenten Speicher geschrieben und in zukünftigen Authentifizierungsphasen abgefragt.
Durch Ausloggen wird der authentifizierte Zustand verlassen und man ist wieder in der Authentifizierungsphase.