Mobiles Hardware-Praktikum 2017

Jan Burchard, Tobias Schubert, Bernd Becker

Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau

Abschlusswettbewerb (50 (+5) Punkte)

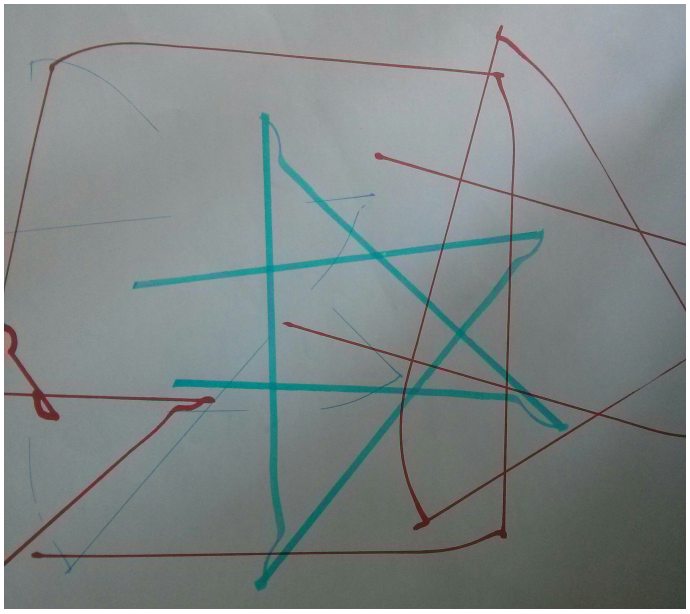
Abgabe: 27.07.2017 bis 17:00 Uhr

Ziel:

Zum Abschluss des Hardwarepraktikums möchten wir Ihnen die Möglichkeit geben das gelernte Wissen in einem größeren Projekt umzusetzen. Das Ziel ist es mit dem Roboter komplexe Figuren zu fahren. Dazu erhalten Sie von uns **verschlüsselte** Fahrbefehle für den Roboter. Diese müssen zunächst entschlüsselt und dann abgefahren werden. Die Bevor Sie die Daten entschlüsseln können müssen Sie diese zunächst auf Fehler überprüfen und diese Fehler gegebenenfalls korrigieren.

Das Szenario simuliert damit einen Roboter der per Funk Daten erhält (“möglicherweise fehlerhaft”) um eine geheime Mission (“verschlüsselt”) durchzuführen. Um den Umfang zu beschränken haben wir den Funk-Teil hier ausgelassen.

Bitte lesen Sie dieses Blatt aufmerksam bis zum Ende durch bevor Sie mit Ihrer Implementierung beginnen damit Sie alle benötigten Informationen kennen.



Organisatorisches:

Der Wettbewerb findet am 27.07. ab 14:15 Uhr im Vorlesungsraum statt. Bitte beachten Sie, dass beim Wettbewerb **Anwesenheitspflicht** für die gesamte Gruppe besteht. Im Anschluss an den Wettbewerb können Sie den Roboter in Gebäude 51, Raum 00 022 zurückgeben. Die Anwesenheit am Wettbewerb und Abgabe der Hardware sind notwendig zum Bestehen des Praktikums.

Die Punkte für dieses Blatt werden etwas anders vergeben als bisher: 35 Punkte werden wie bekannt für die Implementierung selbst vergeben. Die verbleibenden Punkte für ein erfolgreiches, korrektes und exaktes Fahren der Figur. Zusätzlich können Sie für besondere (sinnvolle) Funktionen des Roboters bis zu 5 Extrapunkte erhalten. Die Bewertung der Figur und Vergabe möglicher

Extrapunkte erfolgt beim Wettbewerb.

Jedes Gruppenmitglied sollte die Funktionsweise Ihrer Implementierung beim Wettbewerb während ihr Roboter fährt kurz beschreiben können.

Ablauf

Im laufe der nächsten Tage wird im Forum ein Link zu einer Website veröffentlicht auf der Sie speziell für Ihre Gruppe verschlüsselte Daten abrufen können. Diese Daten können Sie direkt auf dem FPGA als Konstante speichern.

Beim Wettbewerb werden Sie den 32 Bit (4 Buchstaben) Schlüssel zum Entschlüsseln der Daten erhalten. Diesen müssen Sie über die Taster des Roboters eingeben. Sobald der richtige Schlüssel eingegeben wurde soll Ihr Roboter die entsprechenden Fahrbefehle entschlüsseln und ausführen.

Anforderungen:

Bitte beachten Sie, dass es viele verschiedene Möglichkeiten zur Lösung der gestellte Aufgabe gibt. Abgesehen von den folgenden Anforderungen steht es Ihnen frei wie Sie bei der Implementierung vorgehen.

- Die Fehlerkorrektur muss während dem Wettbewerb entweder auf dem Arduino oder auf dem FPGA durchgeführt werden.
- Die Daten müssen auf dem FPGA während dem Wettbewerb entschlüsselt werden.
- Für **jedes** von Ihnen erstellte FPGA-Modul (jede von Ihnen erstellte VHDL-Datei) muss eine sinnvolle Simulation durchgeführt werden. Sie benötigen keine Testbench für das Blockdiagramm und die zur Verfügung gestellten Module. Fügen Sie Ihrer Abgabe jeweils ein Bild der Simulation und die Testbench bei.
- Um exakte Drehungen zu fahren muss der Drehratensensor verwendet werden.
- Der Schlüssel muss über die Taster eingegeben und auf dem Display angezeigt werden.

Technische Details

Fahrbefehle

Die Fahrbefehle bestehen immer aus 16 Bit: $\langle f_{15} \dots f_0 \rangle$. Die einzelnen Bits haben dabei die folgenden Bedeutungen:

- $\langle f_{15} f_{14} \rangle$: “00” = gerade aus, “01” = im Uhrzeigersinn drehen, “10” = gegen den Uhrzeigersinn drehen, “11” = nicht benutzt. [1]
- $\langle f_{13} \dots f_0 \rangle$: Beim gerade aus Fahren: Distanz in mm, bei Drehung: Winkel in Grad.

Alle Drehungen sollen auf der Stelle erfolgen (die Räder drehen sich mit gleicher Geschwindigkeit und entgegengesetzter Richtung). Die Drehung sollte sehr exakt sein. Insbesondere sollte der Roboter sich unabhängig von der Boden-Beschaffenheit weder zu wenig noch zu weit drehen.

[1] Möglicherweise befinden sich in den Fahrbefehlen auch ungültige Befehle mit dem Befehlscode “11”. Diese müssen Sie in Ihrem Programm erkennen und ignorieren.

Verschlüsselung

Die Verschlüsselung ist eine (stark vereinfachte) Abwandlung des AES. Es werden immer Blöcke von 32 Bit verarbeitet. Demnach befindet sich in jedem Block 2 Fahrbefehle. Zusätzlich hat der erste Block in den für Sie verschlüsselten Nachrichten nach dem Entschlüsseln immer den Wert "11111111000000001111111100000000". Diese Tatsache können Sie verwenden um zu Prüfen ob der eingegebene Schlüssel korrekt ist. Dieser Block darf nicht als Fahrbefehl verwendet werden.

Die Entschlüsselung läuft in **4 Runden** in denen jeweils 3 Operationen wiederholt werden:

- Substitute: Die Daten werden in 8 × 4 Bit Blöcke aufgeteilt. Diese Blöcke werden jeweils mit einem Wert aus einer festen Tabelle ersetzt:

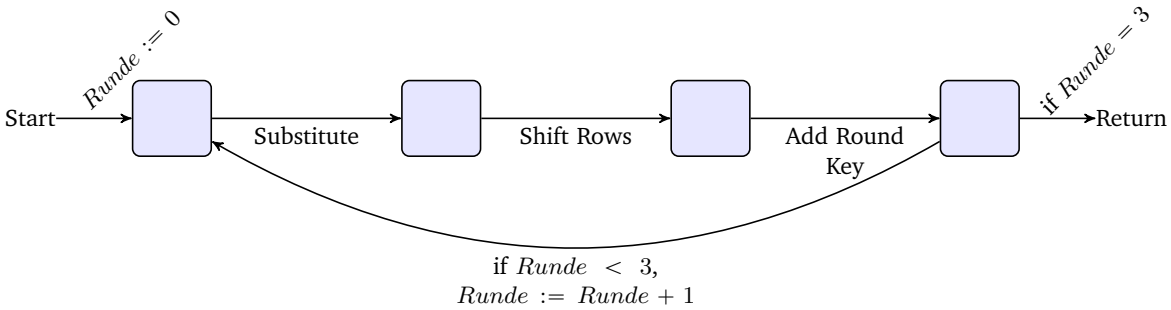
Original:	0000	0001	0010	0011	0100	0101	0110	0111
Substitution:	1010	1100	1000	1111	1110	0110	0011	1011
Original:	1000	1001	1010	1011	1100	1101	1110	1111
Substitution:	0111	0101	0000	0010	1001	0001	0100	1101

- Shift-Rows: Die Daten werden in 16 × 2 Bit Blöcke aufgeteilt. Diese Blöcke werden als Matrix betrachtet. Die i-te Zeile der Matrix wird um $i - 1$ Positionen nach Links verschoben:

$$\begin{bmatrix} d_{31}, d_{30} & d_{29}, d_{28} & d_{27}, d_{26} & d_{25}, d_{24} \\ d_{23}, d_{22} & d_{21}, d_{20} & d_{19}, d_{18} & d_{17}, d_{16} \\ d_{15}, d_{14} & d_{13}, d_{12} & d_{11}, d_{10} & d_9, d_8 \\ d_7, d_6 & d_5, d_4 & d_3, d_2 & d_1, d_0 \end{bmatrix} \Rightarrow \begin{bmatrix} d_{31}, d_{30} & d_{29}, d_{28} & d_{27}, d_{26} & d_{25}, d_{24} \\ d_{21}, d_{20} & d_{19}, d_{18} & d_{17}, d_{16} & d_{23}, d_{22} \\ d_{11}, d_{10} & d_9, d_8 & d_{15}, d_{14} & d_{13}, d_{12} \\ d_1, d_0 & d_7, d_6 & d_5, d_4 & d_3, d_2 \end{bmatrix}$$

- Add Round Key: Die Daten werden über XOR mit dem aktuellen Rundenschlüssel verbunden.

Der gesamte Ablauf der Entschlüsselung sieht also wie folgt aus:



Beispiel:

- Schlüssel: 01100001 01100010 01100001 01100010 (RK 1: Siehe nächster Abschnitt)
- Verschlüsselte Nachricht: 11100001 10011111 10110010 11000011
- Substitute: 01001100 01011101 00101000 10011111
- Shift Rows: 01001100 01110101 10000010 11100111
- Add RK 1: 11011001 10000101 01111110 11001110
- ...
- Klartext Nachricht: 00000001 00101100 01000000 01011010

Rundenschlüssel

Für jede Runde wird ein anderer Rundenschlüssel benötigt. Diesen werden jeweils aus dem vorangegangenen Rundenschlüssel (bzw. für den ersten Rundenschlüssel aus dem eingegebenen Schlüssel) berechnet. Dabei werden immer zwei Operationen ausgeführt:

- Rotate: Der Schlüssel wird um 8 Bit nach Links verschoben:
  $\langle k_{31} \dots k_0 \rangle \rightarrow \langle k_{23} \dots k_0 k_{31} \dots k_{24} \rangle$
- XOR mit einer Konstante. Für jede Runde wird eine andere andere Konstante verwendet.
  Runde 1: "11110111100100011001111001001000"
  Runde 2: "11000110010011001010001101010111"
  Runde 3: "10111001100111111011011111010100"
  Runde 4: "01100111001010110011000011101101"

Zum Schlüssel 01100001 01100010 01100001 01100010 ergeben sich damit zum Beispiel die folgenden Rundenschlüssel:
Runde 1: 10010101111100001111110000101001
Runde 2: 00110110101100001000101011000010
Runde 3: 00001001000101010111010111100010
Runde 4: 01110010010111101101001011100100

Schlüssel

Der Schlüssel besteht (hier) nur aus den Kleinbuchstaben “a-z” in ASCII Codierung (also den Werten 0x61 bis 0x7A). Insgesamt werden 4 Buchstaben verwendet. Diesen Schlüssel erhalten Sie beim Wettbewerb und müssen ihn über die Buttons auf dem Roboter eingeben.

Datenblock

Sie erhalten im laufe der nächsten Tage über das Forum eine Liste an Daten die Sie auf dem FPGA speichern sollen. Diese Daten besteht aus Dem Initialblock (Entschlüsselt “11111111 00000000 11111111 00000000”), einer Menge an Datenblöcken mit jeweils zwei Fahrbefehlen und einem Endblock (Entschlüsselt “11111111 11111111 00000000 00000000”).
Die Datenblöcke enthalten potentiell auch ungültige Fahrbefehle (siehe oben).

Testdaten

Um Ihre Implementierung der Entschlüsselung zu testen können Sie die folgenden Testdaten verwenden.

- Ein Quadrat mit Kantenlänge 300 mm.

  Schlüssel: “abab” (01100001 01100010 01100001 01100010)

Verschlüsselt	Entschlüsselt
01110110110110011111111111100100	11111111 00000000 11111111 00000000
11100001100111111011001011000011	00000001 00101100 01000000 01011010
11100001100111111011001011000011	00000001 00101100 01000000 01011010
11100001100111111011001011000011	00000001 00101100 01000000 01011010
11100001100111111011001011000011	00000001 00101100 01000000 01011010
01110110110011000100001011100100	11111111 11111111 00000000 00000000

Weitere Beispiele werden bei Bedarf im Forum veröffentlicht.

Fehlerkorrektur

Da die verschlüsselten 32 Bit Datenblöcke möglicherweise fehlerhaft sind wurden sie mit einer Fehlerkorrektur versehen. Dazu wurden 6 parity Bits für einen Hamming Code eingefügt (insgesamt werden damit dann 38 Bit pro Block übertragen). Es ist immer höchstens 1 Fehler in einem 38 Bit Block.

Die Parity Bits p_i befinden sich an den Stellen 2^0, 2^1, 2^2, 2^3, 2^4 und 2^5 (Zählung beginnt hier bei 1), ein übertragener Block hat also die folgende Struktur:

$block = \langle d_{31}, d_{30}, d_{29}, d_{28}, d_{27}, d_{26}, \quad p_5, d_{25}, d_{24}, d_{23}, d_{22}, d_{21}, d_{20}, d_{19},$
$d_{18}, d_{17}, d_{16}, d_{15}, d_{14}, d_{13}, d_{12}, d_{11}, \quad p_4, d_{10}, d_9, d_8, d_7, d_6, d_5, d_4, \quad p_3, d_3, d_2, d_1, p_2, d_0, p_1, p_0 \rangle$

Das Datenbit an Position n im übertragenen Block (Achtung: auch hier beginnt die Zählung bei 1, und die parity Bits werden mitgezählt) wird von den parity Bits geprüft die in der Binärdarstellung von n eine 1 haben:

Data bit	n	n in bianry	parity bits
d_0	3	000011	p_1, p_0
d_1	5	000101	p_2, p_0
d_2	6	000110	p_2, p_1
...			
d_{10}	15	001111	p_3, p_2, p_1, p_0
d_{11}	17	010001	p_4, p_0
...			
d_{31}	38	100110	p_5, p_2, p_1

Die parity Bits p_i werden berechnet indem alle Bit die von p_i geprüft werden über XOR kombiniert werden. Zum Beispiel:
$p_3 = d_4 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_8 \oplus d_9 \oplus d_{10} \oplus d_{18} \oplus d_{19} \oplus d_{20} \oplus d_{21} \oplus d_{22} \oplus d_{23} \oplus d_{24} \oplus d_{25}$

Ablauf der Fehlerkorrektur

Wenn Sie einen Datenblock erhalten teilen Sie diesen am Besten zunächst in die Daten- und parity Bits auf. Berechnen Sie dann auf Basis der Daten welche Werte die parity bits haben sollten. Stimmen diese Werte mit den empfangenen parity Bits überein sind die Daten fehlerfrei. Ansonsten ist ein Fehler aufgetreten, den Sie korrigieren müssen.

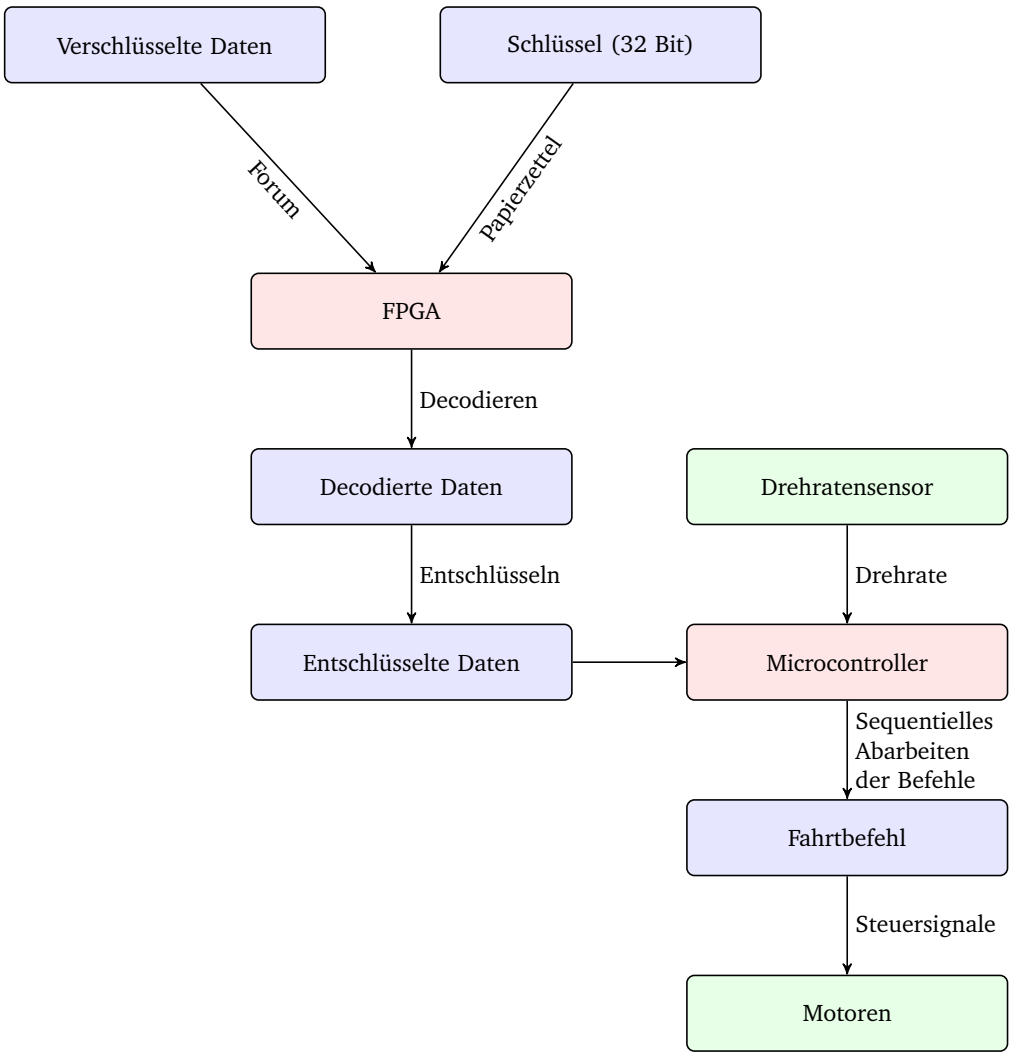
Die Position des fehlerhaften Bits erhalten Sie indem Sie die Positionen der fehlerhaften parity Bits addieren (wenn zum Beispiel p_1 und p_3 fehlerhaft sind befindet sich der Fehler an Position $2^1 + 2^3 = 2 + 8 = 10$, also in d_5. Ist nur ein einziges parity Bit fehlerhaft ist dieses parity Bit selbst fehlerhaft.

Beispieldaten

Daten (Codiert)	Daten (Decodiert)
011101 10110110 01111111 11111001 00011110	01110110 11011001 11111111 11100100
011101 10110110 01111111 11111101 00011110	01110110 11011001 11111111 11100100
011101 10111110 01111111 11111001 00011110	01110110 11011001 11111111 11100100
111101 10110110 01111111 11111001 00011110	01110110 11011001 11111111 11100100

Implementierungsvorschlag

Die Details Ihrer Implementierung stehen Ihnen frei. In diesem Abschnitt wird eine mögliche Implementierung diskutiert.



Der FPGA übernimmt die Decodierung des Hamming-Codes und (wie vorgegeben) das Entschlüsseln der Daten. Diese sind in einem eigenen Modul gespeichert. Bei einer steigenden Flanke an einem Steuersignal gibt das Modul den jeweils nächsten 38 Bit-Block an einem Ausgang aus. Dieser Block wird dann zunächst vom Decodiermodul in einen fehlerfreien 32 Bit block umgewandelt. Dieser wird dann vom Entschlüsselungsmodul eingelesen und entschlüsselt. Die Interaktion zwischen den Modulen steuert ein Steuermodul. Dieses erzeugt die Steuersignale und liest die entschlüsselten Daten ein.

Der Schlüssel zur Entschlüsselung wird ebenfalls vom FPGA eingelesen. Das “analogToButtons” Modul von den letzten Übungsblättern wird dazu so angepasst, dass es **zuverlässig** die gedrückte Taste bestimmt. Dazu wird eine Taste nur dann als gedrückt erkannt wenn der analog Wert für mindestens 50 ms im richtigen Bereich liegt. Mit den Tasten wird eine einfaches Modul angesteuert, dass den Anfangsschlüssel “aaaa” modifiziert: Mit zwei Tasten wird der aktuelle Buchstabe erhöht / verringert und mit einer weiteren zum nächsten Buchstaben gesprungen. Das LCD gibt den aktuellen Schlüssel aus.

Ist der richtige Schlüssel eingegeben erkennt dies das Steuermodul (weil der Initialblock korrekt entschlüsselt wird) und speichert diesen.

Über eine Steuerleitung wird der Microcontroller informiert, dass der FPGA bereit ist Fahrbefehle zu übertragen. Der Microcontroller sendet auf einer weiteren Steuerleitung einen Puls an den FPGA. Dieser Puls startet die Entschlüsselung des nächsten Datenblocks und dessen Übertragung über eine Serielle Verbindung vom FPGA zum Microcontroller.

Auf dem Microcontroller wird der Fahrbefehl decodiert und umgesetzt. Dabei wird der Drehratensensor ausgelesen und bei jeder Drehung verwendet. Um ein zu weites Drehen zu vermeiden beendet der Roboter die Drehung nur wenn er eine bestimmte Anzahl von Schleifendurchläufen lang innerhalb eines ϵ-Bereichs um die Zielrichtung liegt. Anstatt sich immer nur in eine Richtung zu drehen vergleicht der Microcontroller dabei den “ist” und “soll” Wert der Fahrtrichtung und wählt die kürzere Drehrichtung aus.

Beim Erreichen des Endblocks startet der Microcontroller eine Endlosschleife und der Roboter bleibt stehen.

Tipps und Tricks

VHDL

Mit diesem Übungsblatt werden einige VHDL-Module mitgegeben. Diese können Sie verwenden wenn Sie sie benötigen. Es steht Ihnen außerdem frei die Module an Ihre Bedürfnisse anzupassen.

- adcReader: Liest den analog-digital Wandler aus. Verbindung wie auf den letzten Blättern.
- binaryToDecimal: Wird von “dataToLcd” benötigt um Zahlen dezimal darzustellen.
- dataToLcd: Zeigt Daten auf dem LCD an. Als Beispiel kann hier ein ADC-Wert, der gedrückte Button und der aktuelle Schlüssel angezeigt werden. Ein weiterer Eingang steuert die Anzeige ob der Schlüssel korrekt ist.
- lcd_driver: Erzeugt die Steuersignale für das LCD. Verbindung wie auf den letzten Blättern.

Arrays

Um die mitgegebenen Daten bequem zu speichern können Sie einen Array mit einem std_logic_vector verbinden:

```
-- size of array: 6 (stored in 4 bit vector)
constant dataSize: unsigned (3 downto 0) := to_unsigned(6, 4);

-- type definition
type dataElement is array (0 to to_integer(dataSize-1)) of std_logic_vector(31 downto 0);

-- array definition
constant dataStorage: dataElement := (
   "01010111011010010100011101010111",
   "10101001101000101111100011110100",
   "10101001101000101111100011110100",
   "10101001101000101111100011110100",
   "10101001101000101111100011110100",
   "01010111111100001011100001010111");
```

Über “dataStorage(1)” können Sie dann zum Beispiel den std_logic_vector “10101001 10100010 11111000 11110100” abrufen.

Den FPGA dauerhaft programmieren

Auf dem DE0-Nano befindet sich ein extra Chip in dem Sie eine dauerhafte Konfiguration für den FPGA speichern können. So behält dieser die Programmierung auch ohne Stromversorgung bei.

1. Das Quartus-Projekt muss fertig kompiliert sein.

2. In Quartus “File” → “Convert Programming Files” auswählen. In dem Fenster die folgenden Einstellungen vornehmen:

   Programming file Type: JTAG indirect configuration
   Configuraton Device: EPCS16
   File Name: Name der zu erstellenden Datei (merken!)

3. “Flash Loader” markieren und dann rechts auf “Add Device” klicken. Hier bei “Cyclone IV E” das Device “EP4CE22” markieren, und auf “ok” klicken.

4. “SOF Data” markieren und dann rechts auf “Add File” klicken. Hier die original Programmier-Datei auswählen (meistens “<Projektname>.sof”).

5. Auf “Generate” klicken. Nun sollte die Datei erfolgreich erstellt werden.

6. Das Fenster schließen und den Programmer öffnen. Hier die *.sof Datei markieren und entfernen, und über “Add File” die gerade erstellte *.jic auswählen

7. In der Spalte "Program / Configure" einen Haken bei der Datei setzen.

8. FPGA programmieren.

Microcontroller Tipps

- Wenn Sie eine weitere Serielle Schnittstelle benötigen können Sie über die Library “SoftwareSerial” beliebige Pins als Serielle Schnittstelle definieren. https://www.arduino.cc/en/Reference/softwareSerial

  Dies erlaubt es Ihnen die “normale” Serielle Schnittstelle zum Senden von Debug-Nachrichten an den PC zu verwenden. Diese können Sie dort mit dem Serial-Monitor anzeigen.

- Um von einer Seriellen Schnittstelle zu lesen können Sie den “read()”-Befehl verwenden. Dieser gibt das älteste empfangene Byte (8 Bit) zurück. Prüfen sie zuvor mit “available()” ob Daten verfügbar sind.

- Verwenden Sie Ihre Expertise von Blatt 3 um einzelne Bits aus einer Variable zu lesen oder zu modifizieren.

Abgabe:

Archivieren Sie das Quartus II Projekt und packen Sie es zusammen mit dem Arduino-Programm (wenn Sie eines verwendet haben) in einer ZIP-Datei. Schreiben Sie eine kurze Beschreibung Ihres Projekts (inklusive benötigter Pin-Verbindungen) in eine Textdatei und fügen Sie diese zur ZIP-Datei hinzu. Laden Sie die ZIP-Datei im Übungsportal hoch. Überprüfen Sie die Archiv-Datei auf Vollständigkeit. Überprüfen Sie insbesondere auch die im Übungsportal hochgeladene Datei. Vergessen Sie außerdem die Simulationsbilder und Testbenches nicht.