Mobiles Hardware-Praktikum 2023

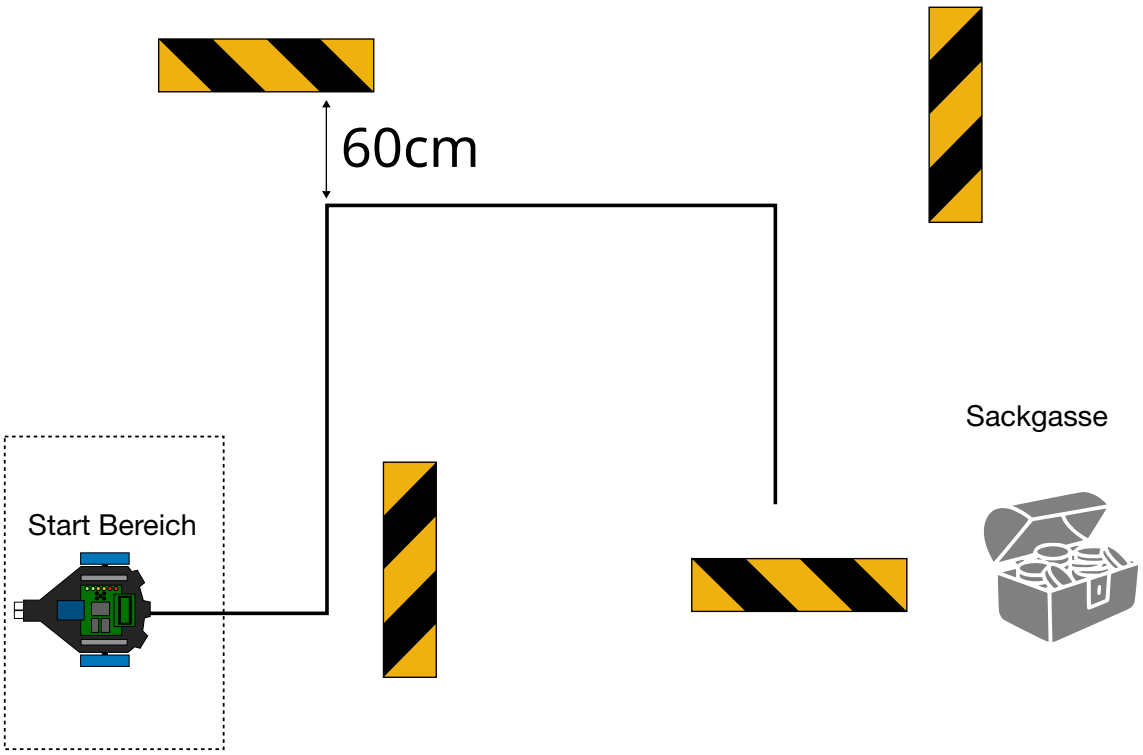
Marc Pfeifer, Philipp M. Scholl
Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau

Abschlusswettbewerb (50 (+5) Punkte)

Abgabe: 21.07.2019 bis 16:00 Uhr

Ziel:

Zum Abschluss des Hardwarepraktikums möchten wir Ihnen die Möglichkeit geben das gelernte Wissen in einem größeren Projekt umzusetzen. Das Ziel, dass der Roboter einen aufgebauten Parkour durchfährt. An dessen Ende befindet sich eine mit einem Schloss gesicherte Schatztruhe. Um diese Truhe zu öffnen, muss ein Code entschlüsselt werden. Danach kann die Truhe manuell geöffnet werden.
Um den Umfang zu beschränken sind einige Teilmodule bereits vorgegeben.



Organisatorisches:

Der Wettbewerb findet am 21.07. ab 16:00 Uhr im HS 00 006, Gebäude 82 statt. Im Anschluss an den Wettbewerb können Sie den Roboter in Gebäude 51, Raum 01 035 zurückgeben. Die Abgabe der Hardware ist notwendig um das Praktikums zu bestehen.

Die Punkte für dieses Blatt werden etwas anders vergeben als bisher: 30 Punkte werden für die Implementierung vergeben. Die verbleibenden Punkte für ein erfolgreiches Durchfahren des Parkours (10) und Entschlüsseln des Codes (10). Zusätzlich können Sie für besondere (sinnvolle)

Funktionen des Roboters bis zu 5 Extrapunkte erhalten. Die Bewertung der 20 Zusatzpunkte erfolgt während des Wettbewerbs.

Jedes Gruppenmitglied sollte die Funktionsweise der Implementierung beim Wettbewerb kurz beschreiben können (Aufgabenverteilung: Wo wurde was implementiert? Besonderheiten, etc.).

Wichtig: Die Abgabe-Deadline für Ihre Implementierung ist am 21.07. um 16:00 Uhr, also unmittelbar vor dem Beginn der Abschlussveranstaltung.

Ablauf

Bei dem Wettbewerb muss der Roboter den Weg durch einen Parkour finden. Der Parkour besteht aus Wänden, auf die der Roboter geradeaus zufahren soll. 60cm vor der Wand soll der Roboter jeweils stehen bleiben und dann eine 90°-Drehung, hin zur nächsten Wand, machen. Anschließend soll der Roboter auf die nächste Wand zufahren, 60cm davor stehen bleiben, sich zur um 90°zur nächsten Wand drehen usw... . Zur Vereinfachung soll folgende feste Sequenz von Drehungen nacheinander eingehalten werden: links, rechts, rechts. Der abzufahrende Parkour inkl. gewünschtem Fahrweg ist in der Abbildung oben zu sehen. Sobald der Roboter das Ende des Parkours erreicht (Sackgasse), soll er automatisch stoppen. Dort wird per seriellem Interface ein 72 Bit (9 Byte) Ciphertext auf den Roboter übertragen. Dieser wurde mit einem 22 Bit Schlüssel verschlüsselt. Nach erfolgreicher Decodierung des Ciphertextes soll der Plaintext auf dem Display des Roboters angezeigt werden. Ist es möglich die Schatztruhe mit dem Code im Plaintext zu öffnen gilt auch diese Aufgabe als gelöst. Sollten Sie das Ende des Parkours nicht erreichen (Teilaufgabe 1), können Sie trotzdem versuchen das Schloss zu öffnen (Teilaufgabe 2).

Anforderungen:

Bitte beachten Sie, dass es viele verschiedene Möglichkeiten zur Lösung der gestellten Aufgabe gibt. Es steht Ihnen jedoch vollkommen frei wie Sie bei der Implementierung vorgehen!
Der Ciphertext muss über den HardwareSerial des Arduino entgegen genommen werden und der Plaintext über das Display ausgegeben 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(en) und die Testbench(es) bei.

Fahren

Um die Schatztruhe zu erreichen, muss Ihr Roboter bis zu einem definierten Abstand auf Wände zufahren können. Hierzu können Sie einen oder mehrere Ultraschall Sensoren verwenden. In Übungsblatt 2 haben Sie gelernt, wie die Ultraschall Sensoren angesteuert und ausgelesen werden. Im selben Übungsblatt haben Sie sich auch mit der Ansteuerung der Motoren vertraut gemacht. Nutzen Sie dieses Wissen, um den Parkour zu durchfahren. Für exakte 90°-Drehung bietet sich die Nutzung des Drehratensensors an. Wie Sie diesen Sensor verwenden haben Sie in Übungsblatt 8 gelernt.
Zum Starten der Fahrt reicht zum Beispiel ein einfacher Knopfdruck.

Übertragung des Ciphertextes

Der Ciphertext wird über das serielle Interface auf Ihren Roboter übertragen. Dazu wird eine serielle Schnittstelle über Kreuz mit dem Arduino des Roboter wie folgt verbunden:

Arduino	Token
rx	tx
tx	rx
GND	GND

Alternativ kann direkt der auf dem Roboter bereits installierte FTDI-Adapter verwendet werden (hierüber wurde der Arduino immer programmiert). Dieser Adapter stellt eine serielle Verbindung zum HardwareSerial des Arduino per USB zur Verfügung. Bei dem Wettbewerb kann der Ciphertext wahlweise über den FTDI-Adapter (Anschluss lediglich per USB) oder über eine extern angeschlossene serielle Schnittstelle (Anschluss über *TX*, *GND* und ggf. *TX*) auf Ihren Roboter übertragen werden.

Der Ciphertext wird mit einer Baud von 9600 übertragen. Nach dem Start der Übertragung des Ciphertextes, wird zunächst ein 'c' als Präfix gesendet (ohne \r\n). Danach folgen 9 Byte Ciphertext. Diese können zum Beispiel als *uint8_t* gespeichert werden. Terminiert wird abschließend mit einem "\r\n". Nun haben Sie 30 Sekunden Zeit die Schatztruhe zu öffnen.

Sie können die Übertragung des Ciphertextes z.B. simulieren, indem Sie Ihren Roboter über den FTDI-Adapter mit Ihrem Computer verbinden und auf der bereitgestellten seriellen Schnittstelle (*COMx*-Port unter Windows, */dev/ttyUSBx* unter Linux) einen Test-Ciphertext schicken. Hierzu gibt es verschiedene Programme. Für die Erstellung eines Test-Ciphertextes steht Ihnen ein Phyton-Skript zur Verfügung.

Der Verschlüsselungsalgorithmus

Der verwendete Verschlüsselungsalgorithmus ist eine abgewandelte Form der DES Verschlüsselung [1]. Die Ver- und Entschlüsselung erfolgt in mehreren Runden und in Blöcken von 12 Bits. D.h. ein Plaintext oder Ciphertext muss zur Ver- oder Entschlüssellung in Blöcke der Größe 12 Bits unterteilt und mit 0 aufgefüllt werden. Beispiel mit 16 Bit:

$$01001000\ 01010111 \rightarrow \text{Block}_1\text{: } 010010000101\ \text{Block}_2\text{: } 011100000000$$

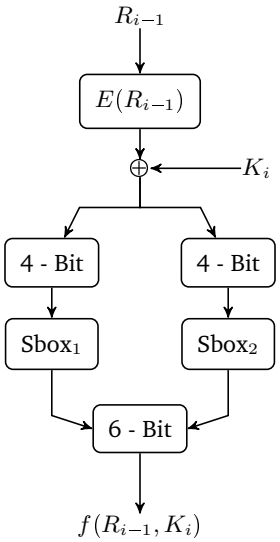
Verschlüsselung

Jeder Plaintext Block der Größe 12 Bit wird nacheinander verschlüsselt. Die verwendete DES Implementierung gehört zu den rundenbasierten Blockchiffren. Dabei wird zu einem 12 Bit Block jede Runde ein 12 Bit Ausgang berechnet.
Zunächst wird der 12 Bit Block in zwei 6 Bit Mini-Blöcke geteilt L_0R_0 (L_0 := *Block(11 downto 6)*, R_0 := *Block(5 downto 0)*).
Jede Runde i des Algorithmus werden nun für die Eingänge $L_{i-1}R_{i-1}$ die Ausgänge L_iR_i mit einem Rundenschlüssel K_i berechnet. Die Ausgänge der i-ten Runde sind dabei wie folgt definiert:

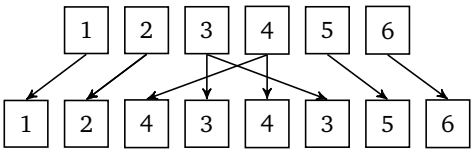
$$L_i = R_{i-1} \tag{1}$$
$$R_i = L_{i-1}\ \ xor\ \ f(R_{i-1}, K_i) \tag{2}$$

Die Funktion $f(R_{i-1}, K_i)$ ist im folgenden Schaubild näher beschrieben.

[1] aus: Introduction to Cryptography with Coding Theory, Wade Trappe, and Lawrence C. Washington, 2002.



$E(R_{i-1})$ expandiert den 6 Bit Block zu einem 8 Bit Block:



Dieser Block wird nun mit dem Rundenschlüssel K_i durch ein bitweises xor verknüpft. Danach wird er in zwei 4 Bit Blöcke geteilt (l, r). Jeder dieser 4-Bit Blöcke wird nun mit einem Eintrag aus einer $Sbox$ substituiert. Eine $Sbox$ ist im Prinzip eine Lookup Table. Hier wird mit einem 4 Bit Input ($2^4 = 16$ Kombinationsmöglichkeiten) einer der 16 Einträge der Sbox gewählt.

Sbox1:

LSB		000	001	010	011	100	101	110	111
MSB	0	101	010	001	110	011	100	111	000
	1	001	100	110	010	000	111	101	011

Sbox2:

LSB		000	001	010	011	100	101	110	111
MSB	0	100	000	110	101	111	001	011	010
	1	101	011	000	111	110	010	001	100

Der Ausgang der beiden Sboxen wird zu einem 6Bit Vector zusammengefügt ($s_1(l)s_2(r)$) der dem Ausgang der Funktion $f(R_{i-1}, K_i)$ entspricht.
Nun können mit $f(R_{i-1}, K_i)$ sowie 1 und 2 die Ausgänge (L_i und R_i) der Runde i berechnet werden.
Wenn der Algorithmus n Runden ausgeführt wird, erhält man L_n und R_n. Der 12 Bit Ciphertext wird im finalen Schritt durch eine einfache Vertauschung realisiert $R_n L_n$. Dabei wurden in den einzelnen Runden die Keys $K_1, \ldots, K_n$ der Länge 8 Bit verwendet.

K_i einer Runde erhält man aus dem Gesamtschlüssel K, indem man diesen $(i-1)$ Stellen zirkulär nach links shifted und die ersten 8 Bit entnimmt. Beispiel $K = 111000111$:

K: 111000111
$K_1 = K_{10}$: 11100011
$K_2 = K_{11}$: 11000111
$K_3 = K_{12}$: 10001111
$K_4 = K_{13}$: 00011111
...

Entschlüsselung

Soll ein Ciphertext entschlüsselt werden müssen lediglich die Rundenschlüssel in umgekehrter Reihenfolge angelegt werden. Wurde mit $K_1, K_2 \ldots K_n$ verschlüsselt, muss mit $K_n, K_{n-1} \ldots K_1$ entschlüsselt werden.

Beispiel:

In diesem Beispiel wird der Ciphertext 100010110101 mit dem Key K =111000111 in 2 Runden entschlüsselt.
Runde 1 ($i = 1$):

- $L_0 = 100010 \quad R_0 = 110101$
- $K_2 = 11000111$
- $E(R_0) = 11101001$
- $E(R_0)\ \ xor\ \ K_2 = 00101110 \quad \rightarrow l_0 = 0010 \quad r_0 = 1110$
- $s_1(l_0) = 001 \quad s_2(r_0) = 001$
- $f(R_0, K_2) = 001001$
- $L_0\ \ xor\ \ f(R_0, K_2) = 101011$
- L_1 =110101 $\quad R_1 = 101011$

Runde 2 ($i = 2$):

- L_1 =110101 $\quad R_1 = 101011$
- $K_1 = 11100011$
- $E(R_1) = 10010111$
- $E(R_1)\ \ xor\ \ K_1 = 01110100 \quad \rightarrow l_1 = 0111 \quad r_1 = 0100$
- $s_1(l_1) = 000 \quad s_2(r_1) = 111$
- $f(R_1, K_1) = 000111$
- $L_0\ \ xor\ \ f(R_0, K_1) = 110010$
- L_2 =101011 $\quad R_2 = 110010$

Dies liefert mit der finalen Vertauschung (R_2L_2) den Plaintext 110010101011.

Bruteforce

Um einen Ciphertext zu entschlüsseln benötigt man einen implementierten DES Algorithmus, die Anzahl der DES Runden und den Schlüssel K.

Für das Abschlussprojekt bekommen sie einen 72 Bit langen Ciphertext übermittelt und haben zusätzlich drei Informationen

1. Der Schlüssel ist 22 Bit lang.
2. Die Anzahl der DES Runden entspricht der Länge des Schlüssels und damit 22.
3. Die ersten 24 Bit des Plaintext lauten “KEY” in ASCII Codierung.

Den Schlüssel bzw. den Rest des Plaintext müssen Sie jedoch selbst herausfinden.
Für diese Aufgabe eignet sich eine Kombination aus einer *Bruteforce* Attacke und einer *Partially-Known-Plaintext* Attacke.
Bei der Bruteforce Attacke werden nacheinander alle möglichen Schlüssel angelegt und das Ergebnis geprüft. Bei unserer Schatztruhe würde das bedeuten, dass KEY<XYZABC> mit allen möglichen Kombinationen der letzten 48 Bits ausprobiert werden müsste ($2^{48} = 281474976710656$ Kombinationsmöglichkeiten). Außerdem können solche Attacken sehr leicht außer Kraft gesetzt werden, wenn man z.B. die Anzahl der Falscheingaben beschränkt (siehe Bankautomat).
Sie können jedoch die dritte Information nutzen, um Bruteforce nicht auf das Schloss direkt (bzw. den Plaintext) anzuwenden, sondern auf den Schlüssel.
Dazu müssen Sie alle möglichen Kombinationen des Schlüssels ausprobieren und den Ciphertext mithilfe dieses Schlüssels entschlüsseln. Stimmen nun die ersten 24 Bit des Plaintexts mit “KEY” überein, haben Sie den richtigen Schlüssel gefunden und können nun den entschlüsselten Code auf dem Display darstellen.
Bei einer Schlüssellänge von 22 Bit gibt es allerdings trotzdem noch $2^{22} = 4194304$ Kombinationsmöglichkeiten. Wenn Sie es also schaffen würden eine Bruteforce Runde innerhalb von $1\mu s$ auf zB dem Arduino zu implementieren (utopisch. . .), benötigt der Arduino im schlimmsten Fall dennoch

ca. 42 Sekunden.
Hier glänzt nun das FPGA, denn dieses ist nicht nur schneller als der Arduino (22 DES Runden benötigen bei unserer Implementierung 49 Clock Zyklen), sondern bietet die Möglichkeit der Parallelisierung. Wir können hierbei mehrere Instanzen eines DES erzeugen und Teile des Schlüssels fix setzen. Wenn man zum Beispiel 8 DES Instanzen bereitstellt, kann man die obersten 3 Bits des Schlüssels auf die 8 Instanzen verteilen.

$$i0 : key(21\ downto\ 19) = 000; \quad i1 : key(21\ downto\ 19) = 001; \quad \ldots \quad i7 : key(21\ downto\ 19) = 111;$$

Das hat denn Vorteil, dass der zu suchende Schlüssel nun nur noch 19 Bit lang ist ($2^{19} = 524288$ Kombinationsmöglichkeiten). Die Worst Case Bruteforce Zeit berechnet sich nun wie folgt:

$$t \approx \frac{\text{Anzahl Blöcke} \cdot \text{Anzahl Clock Zyklen pro Block}}{\text{Taktrate}} \cdot \text{Kombinationsmöglichkeiten} \tag{3}$$

$$\frac{4 \cdot 49}{50000000\frac{1}{s}} \cdot 524288 \approx 2s \tag{4}$$

Für die gestellte Aufgabe steht Ihnen eine DES Implementierung auf dem FPGA zur Verfügung. Zum Testen ihrer Implementierung steht ihnen zusätzlich ein Python Skript mit einer Implementierung des DES Algorithmus zur Verfügung mit der Sie sich verschiedene Cipher-/Plaintext Kombinationen generieren können. Ein beispielhafter Ciphertext ist:

$$0101101000110001001110011111011001001011001010110001100100011101100011000$$

Mit dem Key $11111010101010101111 10$ lässt sich dieser Ciphertext zu dem Plaintext

$$0100101101000101010110010011000100110010001100110011010000110101001101 10$$

entschlüsseln, was ASCII codiert “KEY123456” entspricht
Ein Beispiel für eine mögliche Implementierung ist im folgendem Schaubild gezeigt.

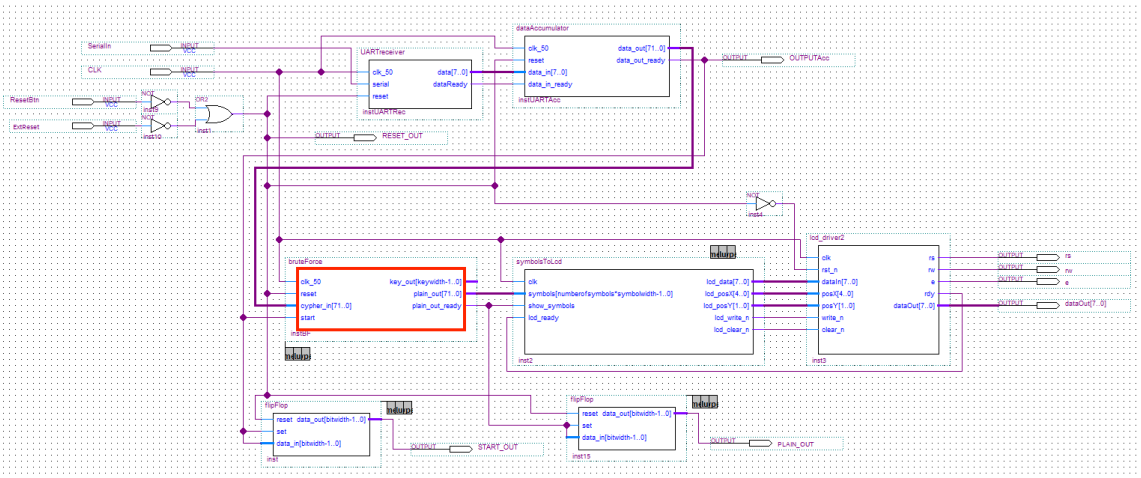
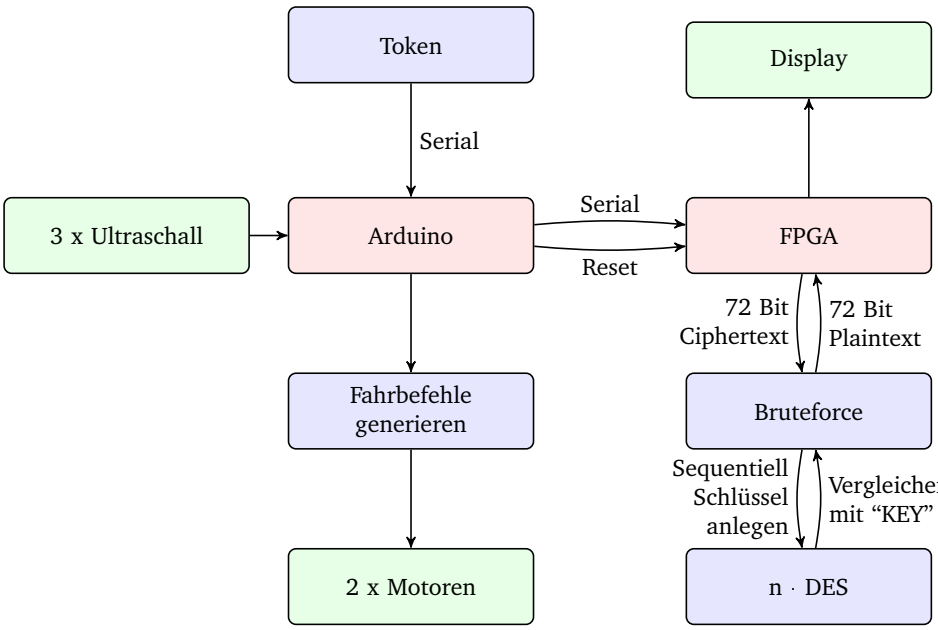


Figure 1: Beispielimplementierung der Bruteforce Attacke

Implementierungsvorschlag

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



Per Knopfdruck beginnt der Roboter durch den Parkour zu fahren. Dabei werden einer oder mehrere Ultraschallsensoren sowie der Drehratensensor ausgelesen und die Steuerbefehle der Motoren entsprechend gesetzt (siehe auch Abschnitt *Fahren*). Gerät der Roboter an das Ende des Parkours soll er automatisch anhalten.
Per UART wird dann der Ciphertext (siehe Abschnitt *Übertragung des Ciphertextes*) an den Roboter übertragen. Alle Module des FPGAs verfügen über einen Reset und werden über einen externen Pin des Arduino in ihren Initialzustand gebracht. Der Ciphertext wird über eine serielle Verbindung vom Arduino (SoftwareSerial) an das FPGA übermittelt (siehe Modul *UARTreceiver* und *dataAccumulator* in den angehängten Dateien).
Auf dem FPGA wird ein Modul *Bruteforce* angestoßen, welches den Plaintext ermittelt (siehe auch Bild 1). Der Plaintext wird auf dem Display angezeigt (siehe Modul *SymbolsToLcd*).

Tipps und Tricks

Hard- und SoftwareSerial

Öffnen Sie die Datei *SerialDemo.ino*. Diese enthält einige Beispiele, wie Daten seriell gelesen oder geschrieben werden können. Neben der HardwareSerial-Schnittstelle gibt es auch die Möglichkeit weitere serielle Schnittstellen über die Library “SoftwareSerial” hinzuzufügen. Mit dieser können beliebige Pins als serielle Schnittstelle definiert werden. `https://www.arduino.cc/en/Reference/softwareSerial`. Beispiele dazu gibt es ebenfalls in der Datei.

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.

- **des:** Implementierung des gezeigten DES Algorithmus. Verwenden Sie diesen für Ihre Bruteforce Attacke. Verbindungen wie in der Entity beschrieben.
- **desDummy:** Kann sowohl einen Dummy Plaintext oder Ciphertext (72 Bit) liefern.
- **adcReader:** Liest den analog-digital Wandler aus. Verbindung wie auf den letzten Blättern.
- **UARTreceiver:** Liest ein Byte eines seriellen Eingang. **Baudrate = 9600**.

- **dataAccumulator:** Verbindet 9 Byte zu einem 72 Bit großen Ciphertext.
- **symbolsToLcd:** Zeigt 9 ASCII-Zeichen an, bekommt 72 Input-Bits.
- **lcd_driver2:** Wird benötigt um das Display anzusteuern.

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(7 downto 0);

-- array definition
constant dataStorage: dataElement := (
   "01010111",
   "10101001",
   "00101001",
   "00001001",
   "00000001",
   "01010101");
```

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

Den FPGA dauerhaft programmieren

Auf dem DE0-Nano befindet sich ein extra Chip in dem Sie eine dauerhafte Konfiguration für das 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

- Für eine weitere serielle Schnittstelle, 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.
  Wichtig: Der Ciphertext muss über den HardwareSerial (Pin RX und TX) auf den Roboter übertragbar sein.

- 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. Wenn Sie mehrere SoftwareSerials verwenden lässt sich jeweils nur einer lesen. Über den Befehl “listen()” können Sie den jeweiligen Serialport auswählen.

Abgabe:

Archivieren Sie das Quartus Projekt und packen Sie es zusammen mit dem Arduino-Programm 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 ILIAS hoch. Überprüfen Sie die Archiv-Datei auf Vollständigkeit. Überprüfen Sie insbesondere auch die im ILIAS hochgeladene Datei. Vergessen Sie außerdem die Simulationsbilder und Testbenches nicht.