Funktioniert's?

Ja, zur Hälfte schon.

D.h. es werden nur Nullen ausgegeben, aber keine Einsen?

Ja, so ungefähr.

Diese Übung schließt das Gesamtprojekt ab. Als Ergebnis soll ein System-on-Chip (SoC) zur Verfügung stehen, welches einen Applikationsprozessor und ein FSK-Modem vereint. Als Applikationsprozessor wird das im Cyclone V fest integrierte Hard Processing System (HPS) mit seinen beiden ARM Cortex-A9 verwendet. Darauf wird ein Linux-Betriebssystem zum Einsatz kommen. Abbildung 1 gibt einen Überblick über das Gesamtsystem.

Hinweis: Zur Vollständigkeit sind in der Abbildung sowohl Sende- als auch Empfangspfad dargestellt. Tx-Gruppen müssen selbstverständlich nur das FSK-Sendesystem, Rx-Gruppen nur das FSK-Empfängersystem in ihrem SoC implementieren.

Das Modem steht bereits aus den vergangenen Übungen vollständig zur Verfügung. In dieser Übung sind nun noch folgende Teile zu ergänzen:

- eine Konfiguration für das HPS,
- die Peripheriekomponenten zur Anbindung des Modems an das HPS,
- eine bootfähige Micro SD-Karte mit einem Linux-Betriebssystem, und
- ein einfaches Programm zur Ansteuerung des Modems.

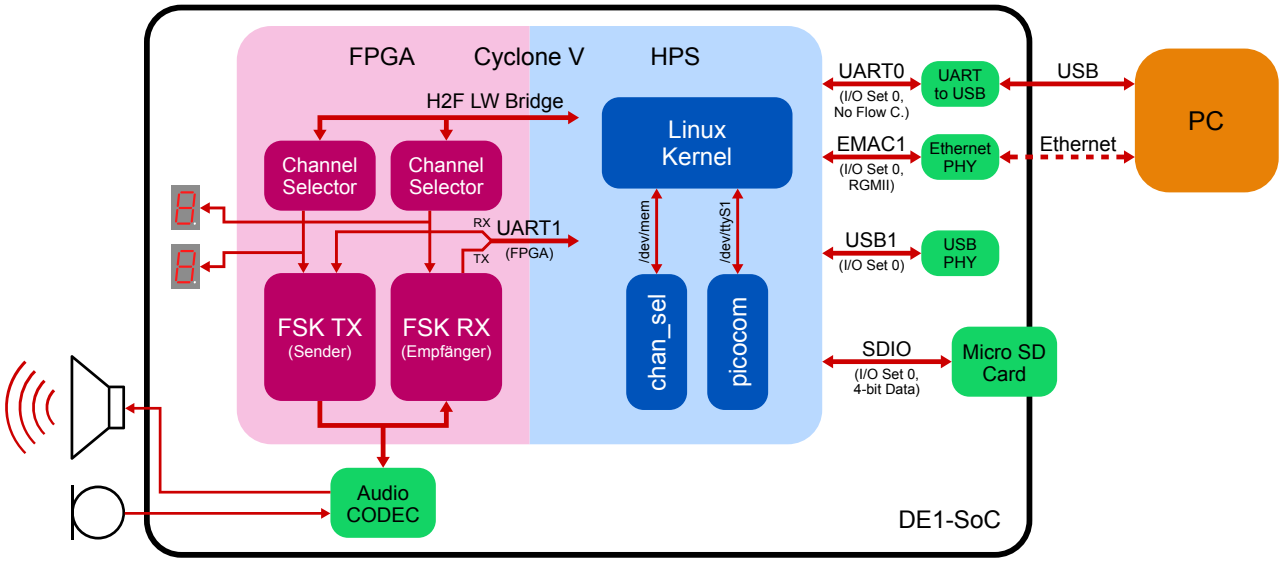

Abbildung 1: Überblick über das Gesamtsystem

1. Aufgabe *Hard Processing System (HPS)*

In der Unit `HPSComputer1` (in der Group `AlteraCycloneV`) finden Sie ein Quartus-Projekt (`DE1_SoC_Computer.qpf`) und ein Qsys-Projekt (`Computer_System.qsys`, Abb. 2) in dem bereits ein vorkonfiguriertes Hard Processing System (HPS) zusammen mit einigen grundlegenden Peripheriekomponenten enthalten ist.

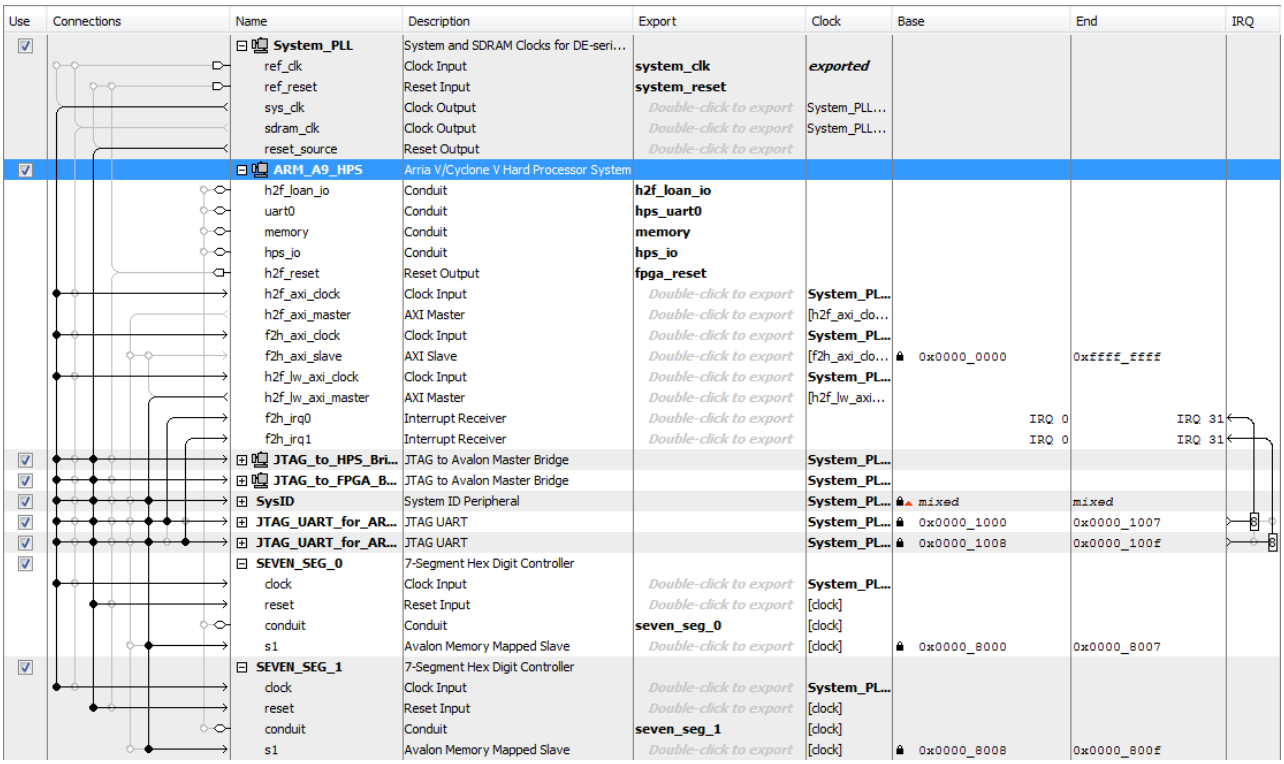

Abbildung 2: Qsys-Projekt

a☐ Machen Sie sich mit diesem System vertraut.

b☐2 Passen Sie die Konfiguration der HPS-Komponente an Ihre Bedürfnisse (siehe Abb. 1) an:

- Sie möchten UART0 als serielle Konsole für das Linux-System verwenden. Verdrahten Sie daher den UART-Controller UART0 mit dem UART-to-USB-Adapter (HPS I/O Set 0). Pins für die Flusskontrolle sind *nicht* vorhanden. Beachten Sie bitte, dass Sie dazu die ev. bestehende Belegung der Pins UART0.RX/TX (Set 0) als GPIO oder Loan-IO (49 bzw. 50) aufheben müssen.
- Sie möchten UART1 zur Kommunikation mit Ihrer Modem-Komponente im FPGA verwenden. Der UART-Controller UART1 muss daher mit dem FPGA verbunden werden. Exportieren Sie diese Signale unter dem Namen `hps_uart1`.

c☐6 Erstellen Sie eine Qsys-Komponente `ModemChannelSelector` (eine Grundstruktur finden Sie bereits im Verzeichnis `unitHPSComputer1/ip/ModemChannelSelector`) mit folgender Schnittstelle:

```
1  entity ModemChannelSelector is
2    generic (
3      CHANNEL_WIDTH   : natural := 4;
4      CHANNEL_DEFAULT : natural := 0);
5    port (
6      clk             : in  std_logic;
7      reset_n         : in  std_logic;
8      channel         : out std_logic_vector(CHANNEL_WIDTH-1 downto 0);
9      s1_address      : in  std_logic_vector(0 downto 0);
10     s1_read         : in  std_logic;
```

```
    s1_readdata    : out std_logic_vector(31 downto 0);
    s1_write       : in  std_logic;
    s1_writedata   : in  std_logic_vector(31 downto 0));
end entity ModemChannelSelector;
```

Die Ports `s1_*` entsprechen einer Avalon-MM-Slave-Schnittstelle. Der Port `channel` (Conduit-Interface) dient der Ausgabe der Kanalauswahl an das FSK-Modem. Implementieren Sie die Komponente so, dass

- beim Lesen des Registers an Adresse +0 (Avalon-MM) der aktuell eingestellte Kanal zurückgegeben wird,
- beim Schreiben des Registers an Adresse +0 (Avalon-MM) die Kanalauswahl an den neu geschriebenen Datenwert angepasst wird, und
- beim Lesen des Registers an Adresse +1 (Avalon-MM) immer der Wert `X"5CD01234"` zurückgegeben wird.

Bei der Implementierung der Komponente können Sie sich an der fertig zur Verfügung stehenden Komponente `HexDigitController` orientieren.

d☐3 Erstellen Sie eine Testbench und weisen Sie die korrekte Funktion Ihrer Komponente in der Simulation nach.

Tipp: Bedenken Sie, dass Sie bei jeder Änderung der Komponente auch das Qsys-Projekt, den Preloader, etc. neu generieren müssen.

e☐2 Fügen Sie zwei Instanzen Ihrer Komponente `ModemChannelSelector` („Modem Channel Selection Controller“) zum Qsys-Projekt hinzu. Benennen Sie die erste `MODEM_CHANNEL_TX` und die zweite `MODEM_CHANNEL_RX`. Verbinden Sie beide Komponenten mit dem Systemtakt und -reset, sowie mit dem AXI-Master der Lightweight HPS-to-FPGA-Bridge. Vergeben Sie folgende Basisadressen:

- `MODEM_CHANNEL_TX`: `0x0000_4000`
- `MODEM_CHANNEL_RX`: `0x0000_4008`

Exportieren Sie anschließend die Conduit-Signale unter den Namen `modem_tx_channel` bzw. `modem_rx_channel`.

Tipp: Auch hierbei können Sie sich an der fertig zur Verfügung stehenden Komponente `HexDigitController` („7-Segment Hex Digit Controller“) orientieren.

f☐1 Generieren („Generate HDL...“) Sie Ihr Design. Als Ausgabeformat („Create HDL design files for synthesis“) stellen Sie bitte „VHDL“ ein. Als Ausgabeverzeichnis sollten Sie den Unterordner `Computer_System` (innerhalb der Unit `HPSComputer1`) wählen.

g☐1 Passen Sie die Wrapper-Unit (`src/HPSComputer1-Inst-a.vhd`) an Ihr neu generiertes Design an. Fügen Sie insbesondere die neue Deklaration der Component `Computer_System` ein. Die neu generierte Component-Definition finden Sie in `Computer_System/Computer_System_inst.vhd`. Passen Sie ggf. auch Signalnamen an Ihr System an.

h☐ Abschließend kompilieren Sie das Quartus-Projekt (`DE1_SoC_Computer.qpf`).

2. Aufgabe *FPGA*

a☐3 Erstellen Sie ein Testbed `TbdTxFskFull` (Tx-Gruppe) bzw. `TbdRxFskFull` (Rx-Gruppe), welches das in der 1. Aufgabe erstellte Qsys-Projekt und die Wrapper-Unit `HPSComputer1` einbindet. Gehen Sie dabei wie bereits in der 8. Übung vor. Die Wrapper-Unit hat einige neue Ports:

```
    -- MODEM CHANNEL SELECTION
    MODEM_TX_CHANNEL : out   unsigned(3 downto 0);
    MODEM_TX_7SEG    : out   std_ulogic_vector(6 downto 0);
    MODEM_RX_CHANNEL : out   unsigned(3 downto 0);
    MODEM_RX_7SEG    : out   std_ulogic_vector(6 downto 0);

```

```
7     -- 7-SEGMENT DISPLAY (7-Segment Hex Digit Controller)
8     SEVEN_SEGMENT_0  : out   std_ulogic_vector(6 downto 0);
9     SEVEN_SEGMENT_1  : out   std_ulogic_vector(6 downto 0);
```

Verbinden Sie die Ports `MODEM_*_7SEG` und `SEVEN_SEGMENT_*` jeweils mit einer 7-Segment-Anzeige.

Als Tx-Gruppe verwenden Sie den Port `MODEM_TX_CHANNEL` zur Auswahl des Kanals Ihrer Unit `TxFsk`. Als Rx-Gruppe verwenden Sie den Port `MODEM_RX_CHANNEL` zur Auswahl des Kanals Ihrer Unit `RxFsk`. Bitte beachten Sie dabei wieder, dass dieses Signal vom HPS kommt und daher asynchron zu Ihrem Design ist. Es muss daher innerhalb des Testbeds auf Ihren **48-MHz**-Takt einsynchronisiert werden bevor Sie es am Port `iChannelSelect` Ihrer FSK-Unit anlegen.

Tipp: Nachdem Sie nur zwischen *zwei* Kanälen (Kanal 0 und Kanal 1) auswählen werden, können Sie zur Synchronisation problemlos auf die Unit `Sync` zurückgreifen, ohne dabei den Zeitbezug zwischen meheren parallel übertragenen Bits eines Datenworts berücksichtigen zu müssen.

b☐ Als Ergebnis erhalten Sie eine RBF-Datei (`TbdTxFskFull.rbf` bzw. `TbdRxFskFull.rbf`).

3. Aufgabe *Bootvorgang: Preloader, Bootloader, Betriebssystem*

Der typische Bootvorgang für das DE1-SoC-Board ist in Abbildung 3 dargestellt. Nach dem Reset wird der Startup-Code im Boot-ROM ausgeführt. Dieser Code wird von Altera zur Verfügung gestellt und kann nicht modifiziert werden. Das Boot-ROM lädt den Preloader. Die Quelle des Preloaders wird anhand der Pegel an den BOOTSEL-Pins des Cyclone V ausgewählt. Am DE1-SoC-Board sind diese so verdrahtet, dass der Preloader von einer MicroSD-Karte geladen wird. Der Preloader (z.B. u-boot Secondary Program Loader) initialisiert das HPS (CPU, Takt, Speicher, Peripherie, Loan-IO, ...) und lädt anschließend den Bootloader (oder direkt ein Baremetal-Programm). Der Bootloader (z.B. u-boot) lädt und startet anschließend den Kernel (Kern des Betriebssystems). Dieser wiederum startet die erste User Space-Applikation (unter Linux ist das typischerweise `/sbin/init`).

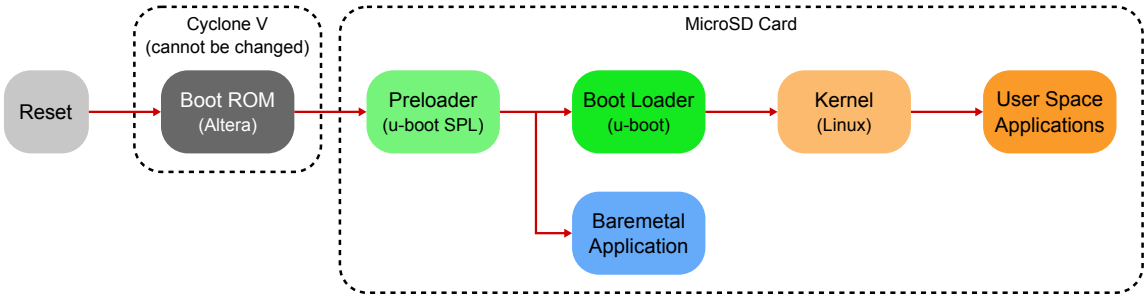

Abbildung 3: Typischer Bootprozess des HPS am DE1-SoC-Board.

Nachdem Sie das Quartus-Projekt aus der 1. Aufgabe erfolgreich kompiliert haben, wurde automatisch das Verzeichnis `hps_isw_handoff/Computer_System_ARM_A9_HPS/` generiert. Dieses ist die Ausgangsbasis für den Preloader. Die darin enthaltenen Dateien ermöglichen es dem Preloader das HPS entsprechend der mit Qsys erstellten Konfiguration zu initialisieren.

a☐[3] Erstellen Sie einen Preloader (u-boot SPL) für Ihr Projekt. Öffnen Sie dazu zunächst die *SoC EDS Command Shell* und wechseln Sie in das Verzeichnis `grpAlteraCycloneV/unitHPSComputer1/`:

```
1 $ cd "Y:\grpAlteraCycloneV\unitHPSComputer1"
```

Starten Sie den BSP-Editor:

```
1 $ bsp-editor
```

Erstellen Sie ein neues HPS BSP Projekt und wählen Sie als „Preloader settings directory“ das Verzeichnis `./hps_isw_handoff/Computer_System_ARM_A9_HPS/` aus.

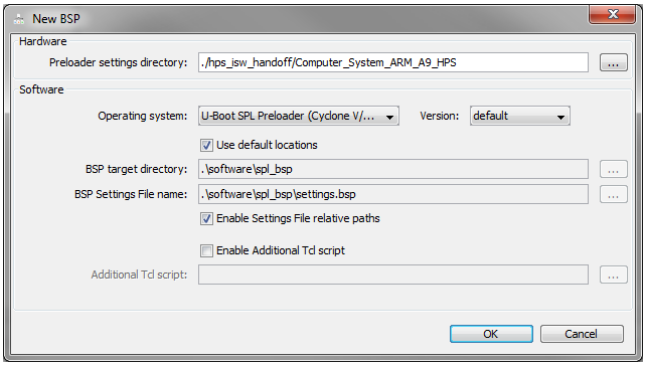

Abbildung 4: Anlegen eines neuen BSP-Projekts

Sie können die Standardeinstellungen für Ihr BSP-Projekt übernehmen. Die wichtigsten Einstellungen sind:

- `EXE_ON_FPGA = 0`
- `BOOT_FROM_SDMMC = 1` (alle anderen `BOOT_FROM_* = 0`)
- `SD_NEXT_BOOT_IMAGE = 0x40000`
- `FAT_SUPPORT = 0`
- `SEMIHOSTING = 0`
- `SERIAL_SUPPORT = 1`

Nachdem Sie die Einstellungen überprüft haben, können Sie das Preloader-Projekt generieren und den BSP-Editor schließen.

Wechseln Sie nun in das neu erstellte Preloader-Verzeichnis und kompilieren Sie den Preloader.

```
$ cd software/spl_bsp/
$ make
```

b☐ Als Ergebnis erhalten Sie das Preloader-Image `preloader-mkpimage.bin`.

c☐[1] Erstellen Sie nun auch den Bootloader (u-boot).

```
$ make uboot
```

d☐ Als Ergebnis erhalten Sie das Bootloader-Image `uboot-socfpga/u-boot.img`[1].

Damit haben Sie nun alle notwendigen Dateien um eine bootfähige Micro SD-Karte für Ihr Gesamtsystem zu erstellen:

- `scd5_linux_sdcard_base.img` (aus dem E-Learning-Kurs),
- `preloader-mkpimage.bin` (aus Aufgabe 3.b),
- `u-boot.img` (aus Aufgabe 3.d), und
- `TbdTxFskFull.rbf` bzw. `TbdRxFskFull.rbf` (aus Aufgabe 2.b).

Abbildung 5 zeigt den Aufbau der Daten auf der Micro SD-Karte. Auf der SD-Karte befinden sich drei Partitionen:

- Partition 3 besteht aus dem Preloader-Image (der Preloader ist darin in vier identischen Versionen enthalten) und dem Bootloader-Image. Das Boot-ROM erkennt diese Partition anhand des Partitions-Typs 0xA2, sucht nach einem gültigen Preloader und führt diesen aus. Der Preloader führt wiederum den eigentlichen Bootloader (von der Adresse `SD_NEXT_BOOT_IMAGE` innerhalb derselben Partition) aus.

[1] ***Achtung!*** Verwechseln Sie dieses nicht mit der Datei `uboot-socfpga/u-boot.bin`.

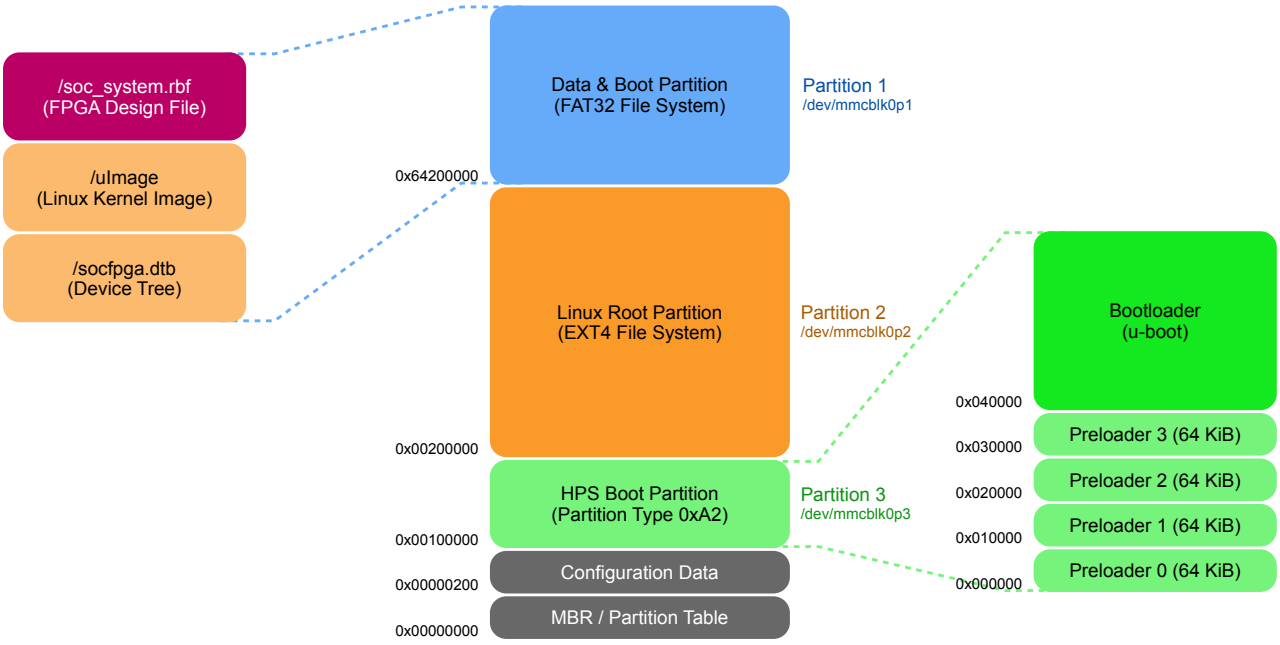

Abbildung 5: Aufbau der Daten im Speicher der Micro SD-Karte

- Partition 1 enthält ein FAT-Dateisystem in dem das FPGA-Design (`soc_system.rbf`), der Linux-Kernel (`uImage`) und der Device-Tree (`socfpga.dtb`) gespeichert sind. Der Bootloader löscht zunächst das FPGA und konfiguriert es mit dem neuen Design aus der Datei `soc_system.rbf`. Anschließend startet der Bootloader den mit dem Device-Tree und dem Root-File-System parametrierten Kernel.
- Partition 2 enthält das Root-File-System mit den (Linux) User-Space-Applikationen.

e☐1 Erzeugen Sie die das SD-Karten-Image für Ihr System, indem Sie Preloader und Bootloader an die entsprechenden Stellen in `scd5_linux_sdcard_base.img` schreiben. Sie können dazu das Programm `dd` aus der *SoC EDS Command Shell* heraus verwenden:

```
$ dd if=preloader-mkpimage.bin of=scd5_...base.img bs=64K seek=16 conv=notrunc
4+0 records in
4+0 records out
262144 bytes (262 kB) copied, 0.0201033 s, 13.0 MB/s
$ dd if=uboot-socfpga/u-boot.img of=scd5_...base.img bs=64K seek=20 conv=notrunc
3+1 records in
3+1 records out
238316 bytes (238 kB) copied, 0.0250593 s, 9.5 MB/s
```

Hinweis: Das SD-Karten-Image hat 2 GB. Um Ihre Subversion-Repository nicht unnötig aufzublasen, speichern Sie das Image bitte *nicht* in Ihrem Repository ab.

f☐ Schreiben Sie das SD-Karten-Image auf Ihre Micro SD-Karte[2].

Bevor die Partitionen der SD-Karte von Windows erkannt werden müssen Sie ggf. die SD-Karte auswerfen („Safely Remove Hardware and Eject Media“) und erneut in Ihr Lesegerät einlegen. Unter Windows sollten Sie anschließend die FAT32-Partition als Laufwerk „FAT_VOLUME“ sehen.

g☐1 Speichern Sie nun die RBF-Datei Ihres FPGA-Designs unter dem Dateinamen `soc_system.rbf` auf der FAT32-Partition.

[2]Sie können dazu beispielsweise den *Win32 Disk Imager* (siehe Abschnitt *Tools für SD-Karten* im E-Learning-Kurs) verwenden.

Die Micro SD-Karte ist nun für die Verwendung in Ihrem DE1-SoC-Board bereit. Sie können den Bootvorgang auf der seriellen Konsole verfolgen.

Tipp: Nutzen Sie z.B. das Terminal-Programm *PuTTY* um sich über die serielle Schnittstelle (UART-to-USB-Adapter) zu verbinden. Die dazu notwendigen Einstellungen sind 115200 Baud, 8 Daten-, 1 Stoppbit, keine Parität.

Zunächst startet der Preloader:

```
U-Boot SPL 2013.01.01 (Dec 07 2017 - 14:29:04)
BOARD : Altera SOCFPGA Cyclone V Board
CLOCK: EOSC1 clock 25000 KHz
CLOCK: EOSC2 clock 25000 KHz
CLOCK: F2S_SDR_REF clock 0 KHz
CLOCK: F2S_PER_REF clock 0 KHz
CLOCK: MPU clock 800 MHz
CLOCK: DDR clock 400 MHz
CLOCK: UART clock 100000 KHz
CLOCK: MMC clock 50000 KHz
CLOCK: QSPI clock 400000 KHz
RESET: COLD
INFO : Watchdog enabled
SDRAM: Initializing MMR registers
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
SDRAM: 1024 MiB
ALTERA DWMMC: 0
```

Danach startet der Bootloader:

```
U-Boot 2013.01.01 (Dec 07 2017 - 14:37:02)

CPU   : Altera SOCFPGA Platform
BOARD : Altera SOCFPGA Cyclone V Board
```

Tipp: Achten Sie auf die Zeitangaben in den ersten Zeilen von Preloader und Bootloader. Diese entsprechen dem Zeitpunkt zu dem diese Komponenten (***von Ihnen!***) erstellt wurden.

Anschließend startet der Kernel:

```
Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
```

Der Bootvorgang kann mehrere Minuten in Anspruch nehmen. Sobald der Bootvorgang beendet ist erhalten Sie eine Kommandozeile:

```
Welcome to Linaro 12.11 (GNU/Linux 3.18.0 armv7l)

 * MAC address: XX:XX:XX:XX:XX:XX
 * IPv4 address: 192.168.1.100 (subnet mask: 255.255.255.0)

 * Connect to this machine via SSH on port 22
     - Username: root
     - Default password: password

root@de1soclinux:~#
```

4. Aufgabe *Linux-Applikation*

Ihr Modem ist nun einerseits über die serielle Schnittstelle und andererseits über Ihren `ModemChannelSelector` ansteuerbar.

Die serielle Schnittstelle ist über das UART-Peripheral UART1 an der Adresse `0xFFC03000` erreichbar. Das Linux-Betriebssystem enthält bereits einen fertigen Treiber, der die Kommunikation mit dieser Peripheral-Komponente abstrahiert und eine einfache Schnittstelle unter dem Dateinamen `/dev/ttyS1` zur Verfügung stellt.

Die beiden `ModemChannelSelector`-Komponenten sind an den Adressen `0xFF204000` (TX) bzw. `0xFF204008` (RX) ansteuerbar. Diese Adressen ergeben sich aus der von Ihnen vergebenen Busadresse (`0x00004000` bzw. `0x00004008`) und der Basisadresse der Lightweight HPS-to-FPGA-Bridge (`0xFF200000`).

Zudem sind in Ihrem Design auch noch zwei Instanzen der `HexDigitController`-Komponente an den Adressen `0xFF208000` und `0xFF208008` vorhanden.

a☐3 Machen Sie sich mit der Funktionsweise des Programms `demo_seven_seg` vertraut. Sie finden es im Home-Verzeichnis des Benutzers „root“ unter `/root/demo_seven_seg`. Es sind sowohl der Sourcecode als auch das kompilierte Programm vorhanden. Erklären Sie, die Funktionsweise dieses Programms.

Tipp: Sie können die Datei direkt in der Konsole editieren. Als Texteditor können Sie beispielsweise `nano` verwenden:

```
root@de1soclinux:~# nano demo_seven_seg.c
```

b☐ Das fertige Programm `demo_chan_sel` ermöglicht es Ihnen bereits den Kanal Ihres Modems einzustellen:

```
root@de1soclinux:~# ./demo_chan_sel 0 1
```

c☐3 Erstellen Sie ebenfalls ein C-Programm `chan_sel`, mit dem Sie den Sende- bzw. Empfangskanal Ihres Modems einstellen können. Sie können es mit `gcc` kompilieren:

```
root@de1soclinux:~# gcc -o chan_sel chan_sel.c
```

Nachdem Sie gemeinsam mit Ihrer Partner-Gruppe einen Kanal eingestellt haben können Sie Daten über das Modem austauschen. Verwenden Sie dazu das Terminalprogramm `picocom`:

```
root@de1soclinux:~# picocom --baud 110 /dev/ttyS1
```

Tipp: Das Programm lässt sich mit der Tastenkombination CTRL+a, CTRL+x beenden.

d☐2 Testen Sie Ihr Modem mit unterschiedlichen Einstellungen für den Kanal und für die Baudrate. Beachten Sie dabei, dass Linux nicht jede beliebige Baudrate unterstützt. Es werden beispielsweise 50, 75, 110, 134, 150, 200, 300 Baud unterstützt. Was passiert, wenn Sie eine Baudrate auswählen, die nicht unterstützt wird? Was passiert, wenn Sie und Ihre Partner-Gruppe unterschiedliche Kanäle einstellen?

Abgabe des Sourcecodes

Neben den regulären ***Commits*** im Laufe der Arbeit an jedem Übungszettel ist der Endstand jedes Übungszettels als Tag im Repository abzulegen. Jeder Tag hat dabei einen Namen der Form „ue-<Nr>“ zu tragen, wobei <Nr> die Nummer des jeweiligen Übungszettels ist.

a☐ Erstellen Sie bis spätestens ***23:59 Uhr am Vortag des Abgabetags*** (siehe *Semesterübersicht*) einen Tag `/tags/ue-09` mit dem Endstand Ihrer Übungsausarbeitung.

Name: _____ Punkte: _____ / 12 P

Matrikelnummer: _____

Name: _____

Matrikelnummer: _____

Bitte arbeiten Sie diesen Übungszettel schriftlich aus und geben die Ausarbeitung bei Ihrer Präsentation des Gesamtsystems in Papierform ab.

Stimmen Sie sich bei Bedarf mit Ihrer Partner-Gruppe ab, um Informationen über den Aufbau der jeweils nicht von Ihnen selbst erarbeiteten Teile des Gesamtsystems zu erhalten.

5. Aufgabe *Gesamtüberblick*

a ☐ 4 Zeichnen Sie ein ***Blockschaltbild*** des Datenpfads von FSK-Sender und FSK-Empfänger. ***Jede Gruppe*** (also sowohl Rx als auch Tx) soll jeweils ein Blockschaltbild des ***gesamten Systems*** (Sender und Empfänger) zeichnen. Teile dieses Blockschaltbilds sind ev. bereits aus früheren Übungsaufgaben vorhanden.

b☐$_8$ Zeichnen Sie weiters ein Diagramm im dem Sie die Signalformen der einzelnen Verarbeitungsschritte (digitaler Bitstrom → FSK-moduliertes Signal → Empfangsfilter → Betragsbildung → Summierung → Glättung → Entscheidung/digitalisierter Bitstrom) für die Bitfolge `'10110'` darstellen.