Kapitel 3

Wiederholung SOPC und Nios II

Aufgabe 1

Erstellen Sie ein neues System auf der Grundlage eines Nios II Prozessors. Um mehr Platz für den kompilierten Programmcode sowie die Laufzeitdaten zu haben, soll als Speicher der auf dem DE2-Board vorhandene SDRAM verwendet werden. Im SOPC-Builder gibt es bereits eine fertige IP-Komponente für den SDRAM-Controller. Die physikalische Verdrahtung der Taktleitung zum RAM auf dem Board führt jedoch zu einer Taktverzögerung (clock skew), so dass die Taktflanke in Relation zu den anderen Signalen zu spät am RAM-Baustein ankommt. Es ist in diesem Fall notwendig, die Flanke für den SDRAM um 3 ns nach vorn (früher) zu verschieben. Dies wird über einen Phase-Locked-Loop (PLL) realisiert, der ein Taktsignal mit der selben Frequenz aber zusätzlicher negativer Phasenverschiebung liefern kann. Der Takt für die restlichen Komponenten wird dabei unverändert weiterpropagiert. Abbildung 3.1 verdeutlicht diese Zusammenhänge.

Im Dokument *„Using the SDRAM Memory on Altera's DE2 Board with VHDL Design“* (`tut_DE2_sdram_vhld.pdf`) finden sie eine entsprechende Anleitung.

Neben einem Nios II Prozessor und dem SDRAM soll Ihr System über einen simplen Taktgeber (Timer) mit 1000 Hz sowie über parallele Ein- bzw. Ausgänge für LEDs und Taster verfügen. Weiterhin ist eine Schnittstelle zur Konsole auf dem PC gewünscht, wofür in diesem Fall das JTAG-UART verwendet werden soll.

Die Wahl des Prozessortyps steht Ihnen frei, wobei vorerst die kleinste Variante ausreicht. Die Priorität des IRQs des Timers sollte höher als die des UARTs sein.

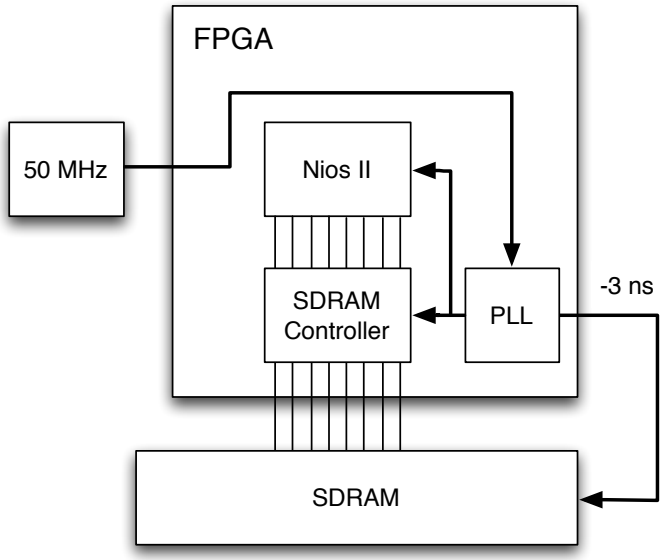

Abb. 3.1 PLL für die Phasenverschiebung des SDRAM-Taktes

Für die Ausgabe sehen Sie bitte 16 rote LEDs und für die Eingabe die vier Taster vor. Verwenden Sie für die PIO-Ports die Komponentennamen `led_pio` bzw. `button_pio`.

Aufgabe 2

Die Software soll nun mittels der auf Eclipse basierenden Entwicklungsumgebung Nios II IDE erstellt und getestet werden. Grundlage dafür ist u. A. die Altera HAL (Hardware Abstraction Library), welche die Initialisierung des Systems, die Anbindung an die Standard-C-Library sowie die Einbindung von Gerätetreibern vornimmt. Mit der Altera HAL als Zwischenschicht wird die Ansteuerung von Hardwarekomponenten ermöglicht, ohne dass ein direkter Zugriff auf die Adressen der Komponenten notwendig ist. Die Nios II IDE generiert dazu aus den Informationen des SOPC-Designs eine Systembeschreibung, die einen Zugriff auf Komponenten

per Namen statt Adressen ermöglicht. Weiterhin werden auf diese Weise nur für die im konkreten System vorhandenen Komponenten die Treiber kompiliert und hinzugelinkt. Auch die Befehle zur Systeminitialisierung werden automatisch generiert.

All dies ermöglicht es, Software wie auf einem normalen Hostsystem ausgehend von einer `main()`-Methode zu entwickeln, ohne sich um Dinge wie Sicherung und Wiederherstellung des Systemzustands sowie das Routing von Ein- und Ausgabe kümmern zu müssen. Dank der Altera HAL kann beispielsweise die Funktion `printf()` zur Ausgabe von Text verwendet werden, wenn im System eine passende Schnittstelle vorhanden ist (in unserem Fall das JTAG-UART).

Ein Projekt in der Nios II IDE besteht üblicherweise aus zwei Komponenten: einer System-Library, welche die HAL und weitere Systemkomponenten enthält und konfiguriert, sowie der eigentlichen Anwendung. Letztere sollte nur den eigenen Code enthalten und ist von der System-Library abhängig. Normalerweise wird bei der Erstellung eines neuen Projekts eine zum System passende Library automatisch generiert.

Erstellen Sie in der Nios II IDE ein neues Projekt. Es sind bereits einige Templates vorhanden. Beachten Sie jedoch für weitere Versuche, dass einige von ihnen eine stark reduzierte HAL verwenden, die auf viele Funktionen verzichtet. So ist dort z. B. `printf()` nicht vorhanden. Machen Sie sich mit den grundlegenden Funktionen der Nios II IDE vertraut. Lassen Sie sich ein „Hello World!“ auf der Konsole anzeigen.

Aufgabe 3

Erstellen Sie ein Lauflicht, das einen Leuchtpunkt auf den roten LEDs anzeigt, welcher sich hin und her bewegt. Erweitern Sie das Lauflicht so, dass sich die Geschwindigkeit über die Taster auf dem Board ändern lässt. Die per `#include` einzubindende Datei `altera_avalon_pio_regs.h` (aus `altera.components`) enthält Makros für den Zugriff auf PIO-Ports. Die wichtigsten sind

```
1 IORD_ALTERA_AVALON_PIO_DATA(base)
```

für den lesenden Zugriff auf einen Port, wobei als `base` die Konstante der Basisadresse verwendet wird (s. u.). Mit

```
IOWR_ALTERA_AVALON_PIO_DATA(base, data)
```

kann entsprechend auf einen PIO-Port geschrieben werden.

Die benötigten Systemkonstanten lassen sich in der Datei `system.h` im `Debug`-Verzeichnis der System-Library des jeweiligen Projekts finden, die bei der Kompilation automatisch generiert wird. Um die in dieser Datei hinterlegten Basisadressen nutzen zu können, muss sie ebenfalls zuvor per `#include` in Ihr Programm eingebunden werden. Die Konstanten der von Ihnen im SOPC-Builder verwendeten Komponenten basieren auf dem Schema

```
<Komponentenname in Großbuchstaben>_BASE
```

also beispielsweise `LED_PIO_BASE`.

Eine allgemeine Einführung in den PIO-Core von Altera ist im Quartus II Handbuch, Volume 5, Sektion I, Kapitel 9 zu finden (Datei `n2cpu_nii51007.pdf` im Verzeichnis dieser Übung).

Aufgabe 4

Erweitern Sie ihr System um einen PIO-Port für die 7-Segment-Anzeigen HEX3 bis HEX0. Stellen Sie unter dem Namen `seven_seg_pio` einen 32 Bit breiten Ausgabe-Port bereit. Sorgen Sie für eine **sinnvolle** Verbindung der 32 Ausgabe-Bits mit den benötigten 4*7 Pins für die Anzeigen (Stichwort: „Byte“).

Entwickeln Sie eine Ansteuerung für die Segmentanzeigen und realisieren Sie darauf als Softwarelösunge einen hexadezimalen sowie einen dezimalen Zähler.

Freiwillige Zusatzaufgabe

Verschönern Sie das Lauflicht, indem Sie eine Spur bzw. einen Schatten aus dunkler werdenden LEDs hinter dem aktiven Lichtpunkt herziehen und somit für weichere Übergänge sorgen („Knight Rider“).