Laborübung 6

(Nios II IDE, PIO, LCD)

Nachdem in der letzen Woche eine Einführung in den SOPC-Builder und den Nios II Softcore-Prozessor sowie seine Programmierung gegeben wurde, sollen in dieser Übung weitere Komponenten kennengelernt und programmiert werden. Als Entwicklungsumgebung dafür dient die Nios II IDE.

Teil 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 diesmal 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.
Abbildung 1 verdeutlicht diese Zusammenhänge.

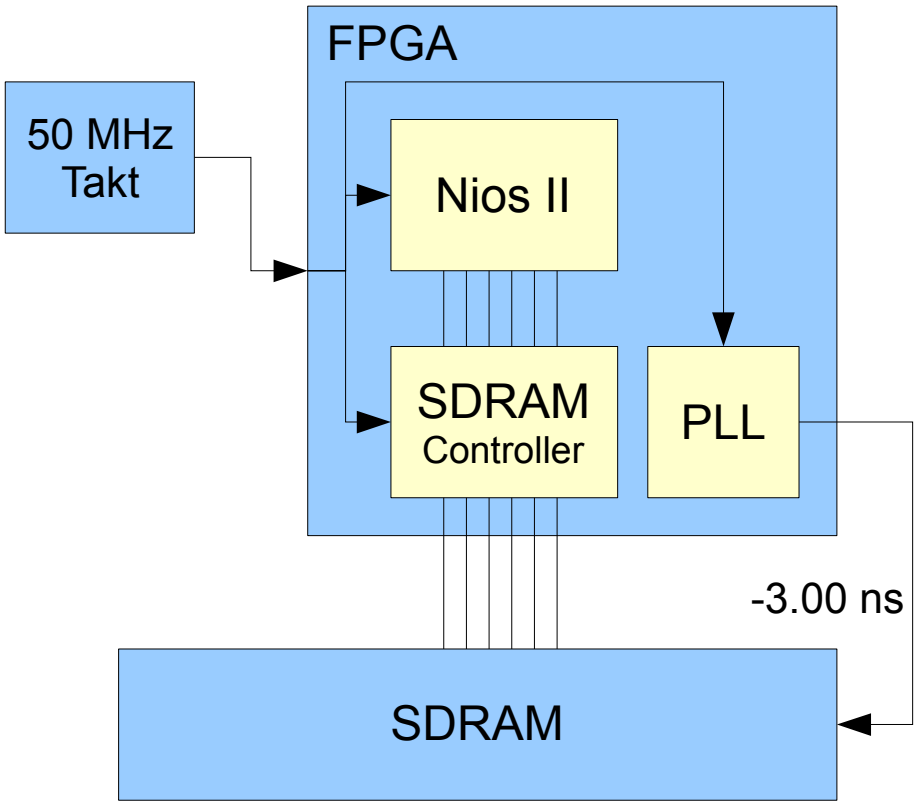

Abbildung 1: PLL für die Phasenverschiebung des SDRAM-Taktes

Um Ihnen etwas Arbeit abzunehmen, liegt ein fertiges SOPC-Design als „lab6_core.sopc“ im Verzeichnis dieser Laborübung. Ebenso ist die VHDL-Datei „Lab6.vhd“ vorhanden, welche bereits den SDRAM sowie einige weitere Komponenten einbindet.

Erzeugen Sie aus den gegebenen Dateien ein lauffähiges System und machen Sie sich mit der Architektur dieses Systems vertraut.

Teil 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, probieren Sie „Hello LED“ und „Count Binary“ aus.
Beachten Sie jedoch für weitere Versuche, dass „Hello LED“ eine stark reduzierte HAL verwendet, 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.

Teil 3

Erweitern Sie das Lauflicht so, dass sich die Geschwindigkeit über die Taster auf dem Board ändern lässt. Der entsprechende PIO-Port ist bereits unter dem Namen „button_pio“ im SOPC-Design vorhanden, ist aber in der VHDL-Beschreibung auf Dummy-Signale gemappt. Sorgen Sie für die richtige Verbindung zu den Tastern. Die SOPC-Komponente muss dafür nicht neu generiert werden, jedoch muss eine Synthese etc. des Gesamtsystems erfolgen.

Die Datei „altera_avalon_pio_regs.h“ (eingebunden über „altera.components“) enthält Makros für den Zugriff auf PIO-Ports. Die wichtigsten sind

```
IORD_ALTERA_AVALON_PIO_DATA(base)
```

für den lesenden Zugriff auf einen Port, wobei für „base“ die Konstante der Basisadresse sein sollte, die sich aus dem Namen des Ports im SOPC-Builder ableitet (z. B. `BUTTON_PIO_BASE`).
Mit

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

kann entsprechend auf einen PIO-Port geschrieben werden.

Die Systemkonstanten lassen sich auch in der Datei „system.h“ im „Debug“-Verzeichnis der System-Library des jeweiligen Projekts finden, die bei der Kompilation automatisch generiert wird.

Eine allgemeine Einführung in den PIO-Core von Altera ist in der Datei „PIO Core.pdf“ im Verzeichnis dieser Übung zu finden.

Teil 4

Es ist auch bereits ein PIO-Port für die 7-Segment-Anzeigen HEX3 bis HEX0 vorgesehen. Unter dem Namen „seven_seg_pio“ wird ein 32 Bit breiter Ausgabe-Port bereitgestellt, der ebenfalls zwar bereits im SOPC-Design vorhanden, aber im VHDL-Code noch nicht passend verdrahtet ist.
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 einen **hexadezimalen** sowie einen **dezimalen** Zahler.

Teil 5

Auf dem Board ist auch ein 16x2 Zeichen LCD vorhanden. Dieses soll nun angesteuert werden. Dazu müssen Sie einerseits im SOPC-Builder die passende IP-Komponente hinzufügen und andererseits für deren Anbindung in der VHDL-Beschreibung des Systems sorgen.
Achten Sie beim Hinzufügen von Komponenten im SOPC-Builder darauf, dass diese am Takt „clk“ und nicht am verschobenen Takt „sdram_clk“ hängen. Stellen Sie sicher, dass in Ihrem Design alle Komponenten an „clk“ angebunden sind. **Der Name des LCD-Cores soll „lcd_display“ sein.**

Da sich nun die Schnittstelle zum SOPC-Design ändert, müssen Sie die neue Port-Definition aus der Datei „lab6_core.vhd“ entnehmen (Suche nach „entity lab6_core“). Passen Sie das Port-Mapping der instanziierten SOPC-Komponente an und stellen Sie die Verbindung zu den richtigen Pins des LCDs her (Schauen Sie in die Pin-Assignments).
Das Display wird über den Pin „LCD_ON“ ein/ausgeschaltet. Wenn Sie mögen, können Sie einen PIO-Port dafür hinzufügen, um das Display per Software schalten zu können. Ansonsten sorgen Sie dafür, dass der Pin dauerhaft mit '1' getrieben wird.

Die Ansteuerung des Displays wird im Dokument „LCD Controller Core.pdf“ beschrieben, das im Verzeichnis dieser Übung zu finden ist. Einige interessante Routinen sind auch in der Datei „count_binary.h“ des Projektes „Count Binary“ zu finden.

Experimentieren Sie mit dem Display, indem Sie eigene Texte darauf ausgeben. Wie funktioniert das Scrolling?
Falls Die die Zeit haben, implementieren Sie eine Funktion, die einen gegebenen String im Kreis über das Display laufen lässt (Buchstabe für Buchstabe und an den Rändern von oben nach unten bzw. von unten nach oben).

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“).