Kapitel 4

Memory-Mapped-IO

In dieser Übung soll das bereits von Ihnen entwickelte programmierbare PWM-Modul an den Nios II Prozessor angebunden und dann per Software gesteuert werden. Altera stellt dafür ein Konzept bereit, um per Memory-Mapped-IO auf die Register von Komponenten eines SOPC zugreifen zu können.

Das Avalon-MM System-Interconnect-Fabric ist ein Bus-ähnliches Konstrukt, das die Verbindung von Komponenten innerhalb eines mit dem SOPC-Builder erstellten Systems ermöglicht. Für die angeschlossenen Komponenten erscheint die Anbindung und Adressierung wie ein gewöhnlicher gemeinsamer System-Bus, intern arbeitet das Avalon-MM allerdings mit Eins-zu-Eins-Verbindungen, genauer gesagt mit Slave-seitiger Arbitrierung. Bei dieser Verbindungsart wird auf der Seite des Slaves entschieden, mit welchem Master er kommunizieren soll, so dass zur selben Zeit verschiedene Master mit verschiedenen Slaves Daten austauschen können.

Im Folgenden wird das Avalon-MM System-Interconnect-Fabric der Einfachheit halber als „Avalon-Bus“ oder schlicht „Avalon“ bezeichnet werden, auch wenn dies (wie oben dargestellt) technisch gesehen nicht ganz korrekt ist.

Um mit einer Komponente am Avalon-Bus teilnehmen zu können, ist es notwendig, das entsprechende Avalon-Interface zu implementierten. Dabei gibt es einige elementare Signale, die jedes Interface beherrschen muss, sowie darüber hinaus gehende Erweiterungen, die optional sind.

Die Idee des Interconnect-Fabrics wird in Kapitel 2 von Volume 4, Sektion I des Quartus II Handbuchs beschrieben (Datei `qts_qii54003.pdf` im Übungsverzeichnis). Die Spezifikation des gesamten Avalon-Systems findet sich in der Datei `mnl_avalon_spec.pdf`.

Aufgabe 1

Erstellen Sie ein neues System mit einem Nios II Prozessor, der Zugriff auf eine Instanz Ihres PWM-Moduls hat.

Dank der von Ihnen bereits implementierten Register kann eine Anbindung an den Avalon-Bus relativ leicht erfolgen. Die vorhandene Trennung von `readdata` und `writedata` passt direkt zum Avalon-Interface.

Benutzen Sie Ihre selbst entwickelte PWM-Komponente als Basis. Achten Sie darauf, dass nur die angegebenen Signale (und evtl. zusätzlich `read`) oder ihre Komplemente in der Schnittstelle nach außen geführt werden. Falls nötig, passen Sie Ihre Komponente noch einmal an.

Wenn Sie zuvor die Komponente wie angegeben erstellt haben, sollte sie bereits den Anforderungen eines Slave-Interfaces des Avalon-Busses genügen. Überprüfen Sie anhand des Timing-Diagramms für Lese- und Schreibzugriffe mit festen Wartezyklen (Seite 21 bzw. 3-9 in der Interface-Spezifikation), ob das auch für Ihre Implementierung gilt. Vorgesehen sind jeweils **null Wartezyklen**.

Anschließend können Sie beginnen, das PWM-Modul in den SOPC-Builder einzubinden. Dazu legen Sie die VHDL-Datei in Ihrem Projektverzeichnis ab (Alteras Entwurfsfluss sieht dafür das Unterverzeichnis `hw` vor, um den selbst erstellten Code von den automatisch erzeugten VHDL-Dateien unterscheiden zu können) und wählen im SOPC-Builder den Menüpunkt „File“ und dann „New Component“. Dort können Sie nun Ihren VHDL-Code auswählen und die Signale des Avalon-Busses mit Ihren eigenen verbinden.

Sie benötigen neben dem Slave-Interface einen Clock-Input; das Hinausführen des PWM-Signals geschieht über ein „export“ in einem Conduit-Output, welches das Signal außerhalb des SOPC-Cores verfügbar macht.

Beim Timing sollten Sie alle Werte vorerst auf **null** stellen. Geben Sie im „Component Wizard“ (rechte Registerkarte) als „Component Class Name“ `PWM` ein.

Sie können nun eine Instanz ihres PWM-Moduls dem System hinzufügen. Der Name dieser Instanz muss sich allerdings aus technischen Gründen vom Namen des Moduls unterscheiden.

Nun können Sie das System generieren. Binden Sie das exportierte PWM-Signal an die grüne LED `LEDG0` und synthetisieren Sie das Gesamtsystem.

Aufgabe 2

Erstellen Sie in der Nios II IDE ein neues Projekt, dass das PWM-Modul ansteuert.

Ein „Mini-Treiber“ für den Registerzugriff ist bereits mit der Datei `pwm_regs.h` gegeben, die Sie im Übungsverzeichnis finden. In dieser Header-Datei werden einige Makros definiert, die etwas von dem direkten Zugriff auf die IO-Ports der Komponente abstrahieren. So existieren für jedes Register eigene Lese- und Schreibbefehle und es muss nur noch die Basisadresse übergeben werden, der Offset zum gewünschten Register jedoch nicht. Die Form dieser Makros ist bereits von Altera für die Treiberentwicklung vorgesehen.

Erstellen Sie einen „richtigen“ kleinen Treiber in Form einer Auswahl an Funktionen, denen Sie die Werte für den Dutycycle sowie den Frequenzteiler übergeben können. Schreiben Sie auch eine Routine, welche eine direkte Übergabe der Frequenz in Hertz ermöglicht. Wenn das PWM-Modul am selben Takt wie die CPU angebunden ist (so wie es eigentlich sein sollte) dann können Sie die Konstante `ALT_CPU_FREQ` in Ihrer Rechnung für die Taktfrequenz benutzen. Achten Sie bei diesen Funktionen auch auf die korrekte bzw. sinnvolle Benutzung des „Enable“.

Schreiben Sie dann ein Hauptprogramm, dass unter Verwendung Ihrer Treiberroutinen den Dutycycle zyklisch von 0 % bis 100 % (also von `0x00` bis `0xff`) und wieder zurück verändert, so dass sich ein sichtbares Pulsieren der grünen LED ergibt. Für die Wartezeiten können sie die Funktion `usleep(unsigned int useconds)` verwenden, wenn Sie die Header-Datei `unistd.h` einbinden. Denken Sie auch daran, die Header-Dateien `system.h` und bei Bedarf `alt_types.h` (wenn Sie die Altera-Datentypen verwenden wollen) mit einzubinden.

Aufgabe 3

Verändern Sie ihr Hauptprogramm so, dass Sie mit je zwei Tastern den Dutycycle sowie die Frequenz des PWM-Signals frei einstellen können (also kein automatisches Pulsieren mehr).

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. Stellen Sie sicher, dass in Ihrem Design alle Komponenten an den zentralen Systemtakt angebunden sind.

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. Der Pin `LCD_BLON` (Backlight On) ist ohne Bedeutung, da das LCD auf dem Board keine Hintergrundbeleuchtung besitzt.

Die Ansteuerung des Displays wird in Volume 5, Sektion I, Kapitel 8 des Quartus II Handbuchs beschrieben (Datei `n2cpu_nii51019.pdf`). Den Zugriff auf den Treiber sollen Sie nun selbst übernehmen. Nach dem Einbinden der Header-Dateien `stdio.h` und `altera_avalon_lcd_16207.h` muss zunächst ein Filedescriptor erstellt werden, über den das Ausgabegerät später referenziert werden kann:

```
1 FILE *lcd;
```

Binden des LCDs an diesen Filedescriptor:

```
1 lcd = fopen("/dev/lcd", "w");
```

wobei für `lcd` der Name der entsprechenden Instanz aus dem SOPC-Builder verwendet werden muss. Ist der Rückgabewert ungleich `NULL`, so war die Initialisierung des Displays erfolgreich und es kann mittels

```
1 fprintf(lcd, ...);
```

analog zu einem normalen `printf()` auf das LCD geschrieben werden. Die Steuerkommandos für das Anspringen bestimmter Positionen etc. können aus dem entsprechenden PDF-Dokument entnommen werden.

Geben Sie den Dutycycle sowie die Frequenz des PWM-Signals auf dem LCD aus. Geben Sie zusätzlich den 16 Bit Wert, der wirklich als Taktteiler in die Register des PWM-Moduls geschrieben wird, auf den 7-Segment-Anzeigen aus.

Aufgabe 4

Erweitern Sie Ihre PWM-Komponente so, dass der Dutycycle mit einer Auflösung von 16 Bit eingegeben werden kann. Der Taktteiler soll auf mindestens 24 Bit erweitert werden.

Mit Ihrer bisherigen Implementierung, in der Sie die Skalierung des Dutycycle auf den jeweils aktuellen Zählerbereich in Hardware durchführen, würde dabei der Multiplizierer enorm größer werden. Deutlich sinnvoller ist es deshalb, die Berechnung der Skalierung von der Hardware in die Software zu verlagern, zumal die Rechnung nicht kontinuierlich sondern nur bei Änderungen der Parameter durchgeführt werden muss.

Entfernen Sie daher die Berechnung der Skalierung aus Ihrer Komponente und passen Sie den Rest an die neuen Anforderungen an. Es steht Ihnen dabei frei, ob Sie die Register verbreitern (sinnvollerweise nicht breiter als max. 32 Bit, der Datenbreite des Nios II Prozessors) oder statt dessen den Adressraum vergrößern (also mehr Register vorsehen). Achten Sie in jedem Fall auf eine durchdachte Aufteilung der Parameter auf Register.

Es bietet sich an, dafür eine Kopie Ihrer vorhandenen Komponente (sowohl VHDL-Code als auch im SOPC-Builder) zu erstellen. Denken Sie daran, auch die Header-Datei für den Register-Zugriff anzupassen. Hier müssen Sie falls notwendig mindestens die Lese- und Schreibbefehle sowie die Offsets aktualisieren.

Generieren Sie anschließend das neue System, und passen Sie Ihren Treiber an, in welchem Sie nun auch die Skalierung vornehmen müssen. Beachten Sie dabei die nun geänderten Wertebereiche.

Testen Sie Ihr neues System.