Laborübung 7

(Avalon-MM, PWM)

In dieser Übung soll das bereits von Ihnen entwickelte programmierbare PWM-Modul an den NiosII-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.

Teil 1

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 des Quartus II Handbuchs beschrieben (Datei „**Avalon-MM.pdf**“ im Übungsverzeichnis). Die Spezifikation des gesamten Fabrics finden sich in der Datei „**mnl_avalon_spec.pdf**“.

Implementierung:

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 für Zettel 5 Teil 4 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 damals 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 24 bzw. 3-10 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 (sinnvollerweise in einem Unterverzeichnis wie z. B. „hw“), und wählen im SOPC-Builder den Menüpunkt File → 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 Conduit-Output, welches das Signal außerhalb des SOPC-Cores verfügbar macht. Beim Slave-Interface sollte das Slave-Adressierungs-Schema vorerst auf **„NATIVE“** eingestellt werden. Weiterführende Erläuterungen dazu finden Sie in der oben erwähnten Datei „Avalon-MM.pdf“ auf Seite 8 und folgenden.

Beim Timing sollten Sie alle Werte 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. Geben Sie der Instanz einen sinnvollen Namen (z. B. „pwm“) und sorgen Sie dafür, dass die Komponente an den Systemtakt „clk“ angebunden wird.

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

Teil 2

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

Ein „Mini-Treiber“ in Form von Makros für den Registerzugriff ist bereits mit der Datei „pwm_regs.h“ gegeben, die Sie im Übungsverzeichnis finden. Es können damit ähnlich wie bei den PIO-Ports die Register des PWM-Moduls geschrieben und gelesen werden. Achten Sie beim Aufruf auf die Übergabe der richtigen Basisadresse.

Erstellen Sie einen „richtigen“ kleinen Treiber (Funktionen), dem Sie die Werte für den Dutycycle sowie den Frequenzteiler übergeben können. Schreiben Sie auch eine Routine, welche eine direkte Übergabe der Frequenz in Hz 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 den Dutyclyle zyklisch von 0% bis 100% (also von 0x00 bis 0xff) und wieder zurück verändert, so dass sich ein **sichtbares Pulsieren** der gründen 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.

Teil 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).

Geben Sie diese beiden Werte 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.

Ansteuerung des LCDs:
Nachdem es in der letzten Übung ein Problem mit dem Namen des LCDs gegeben hat, wollen wir den Zugriff auf den Treiber nun selbst übernehmen.

Nach dem Einbinden der Header-Dateien „stdio.h“ und „altera_avalon_lcd_16207.h“ muss zunächst ein Filedescriptor erstellt werden:

```
FILE *lcd;
```

Binden des LCDs an diesen Filedescriptor:

```
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 kann mittels

```
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 aus dem Verzeichnis der letzen Übung entnommen werden.

Teil 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) oder statt dessen den Adressraum vergrößern. Achten Sie in jedem Fall auf eine durchdachte Aufteilung der Parameter auf Register.

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.