Komplettierung, Simulation und Verbesserung von VHDL-Code zur Kommunikation zwischen FPGA und CPLD

Ranjeet Kuruvilla

19. March 2011

Inhaltsverzeichnis

1 Einleitung

Der Gegenstand dieses Praktikums besteht aus der Komplettierung und dem Testen des VHDL-Codes, welcher für den FPGA und den CPLD des *DajaSittah* zum Einsatz kommt. Der Code wurde an einem früheren Zeitpunkt geschrieben, war allerdings unvollständig und konnte zu jenem Zeitpunkt allerdings nicht ausführlich getestet werden.
Die Aufgabe des Codes besteht vorallem darin, Daten vom FPGA über den CPLD an die Endgeräte bzw. in die andere Richtung zu bringen. Die korrekte Funktionalität wird also dadurch getestet, indem der Wert welcher in jedwede Richtung ausgegeben mit dem eingebenenen Wert verglichen wird.

Systembeschreibung Das System besteht aus einem FPGA, welches als MASTER fungiert, und einem CPLD, welcher den SLAVE stellt. Im FPGA ist der *spi_ master.vhd* implementiert, im CPLD der *spi_ slave.vhd*. Verbunden sind diese beiden Geräte über eine

serielle Verbindung [1], die besteht aus dem MASTER-SELECT-Signal SMS, dem Takt SCK, dem Dateneingan SDI und dem Datenausgang SDO, wobei das Signal SDO, SMS und SCK vom MASTER getrieben wird. Dabei läuft die SDO-Leitung des Vorgängergerätes in den SDI-Eingang des aktuellen Gerätes. Die Daten werden mittels SHIFT-Register durch SDO des einen in den SDI des anderen Partners geschoben, sobald MASTER das SMS-Signal auf LOW setzt. Die Frequenz der Kommunikation wird durch SCK bestimmt, welche in unserem Fall der Betriebs-Frequenz des CPLD entspricht. Wird SMS auf LOW gesetzt, so tauschen MASTER und SLAVE im Handshake-Verfahren die Daten aus. Der CPLD ist verbunden mit allen anzusprechenden Entitys. Dieses System wurde im Praktikum simuliert und getestet.

MASTER wurde um zwei Eigenschaften erweitert.

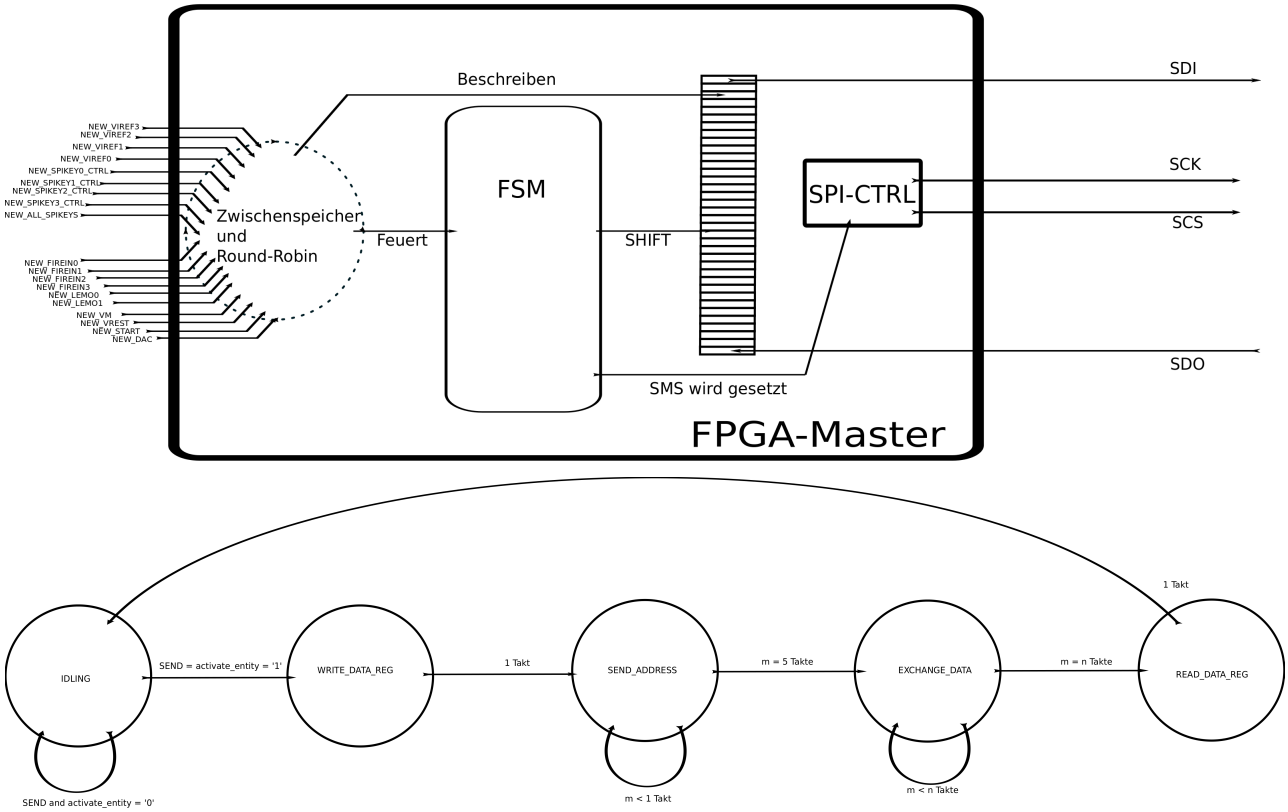
1. Daten können zwischengespeichert werden.
2. Ein Scheduler sorgt dafür, dass die Daten in einem gewissen Zeitfenster gesendet werden.
3. Beim ADC, Temperatursensor und den LOCKED-Zuständen des *Spikeys* sendet der CPLD Daten zum FPGA, bei allen anderen Entitys werden Daten vom FPGA zum CPLD gesendet. Aussnahme bildet der CPLD-Test, dessen Funktionsweise später beschrieben wird, bei dem Daten in beide Seiten gesendet wird.
4. Es wurden Mechanismen eingebaut um permanent die Verbindung zwischen FPGA und CPLD zu testen. Dadurch ist es möglich Fehler in der Verbindung zu überwachen.

Funktionsweise des SPI-Kontrollers im FPGA Die Kontrolle über die SPI-Verbindung obliegt der Entity *spi_ master.vhd* auf FPGA-Seite und auf CPLD-Seite der Entity *spi_ slave.vhd*.

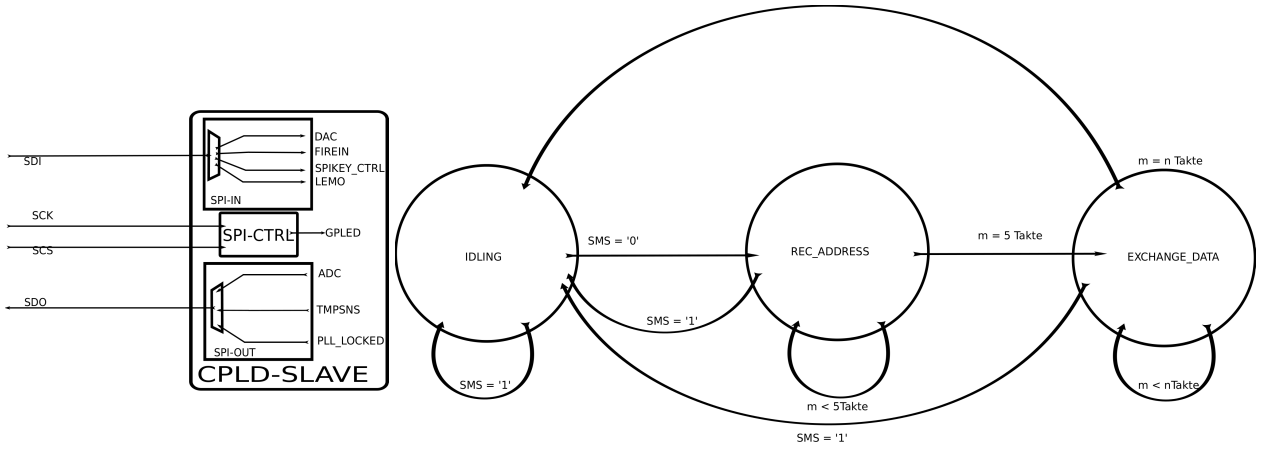
1. Die Aussenwelt signalisiert den Wunsch Daten an einen der mit dem CPLD verbundenen Entitys zu senden via einem NEW_<ENTITY>-Signal; dabei ist <Entity> die jeweilige Entity, die mit neuen Daten versorgt werden soll.
2. Der SPI-Controller nimmt die Daten von aussen an und speichert sie in einem Datenregister. Sobald dieses Signal auf HIGH gesetzt wird, werden die anliegenden Daten der entsprechenden Entity im FPGA zwischengespeichert.
3. Ein Scheduler ordnet jeder Entity ein Zeitfenster zu; in diesem Zeitfenster werden die Daten einer Entity verschickt bzw. die Daten von einer Entity abgerufen.
4. Jede Entity zeigt an, ob Daten versendet werden. Ist dies der Fall wird die Zustandsmaschine veranlasst seinen Zustand zu wechseln und erst die Adresse der angesprochenen Entity als auch die Daten selbst zu übertragen.

[1] im weiteren Verlauf mit SPI benannt

5. Dementsprechend muss der angelegte Wert 5 + n Takte nach dem Senden am Empfänger sein.



Der CPLD Die FSM im CPLD ähnelt der FSM im FPGA, allerdings besitzt sie nur nur drei Zustände. Diese FSM wird diesmal getriggert durch das SMS-Signal.



Der CPLD "weiss", dass die ersten fünf Bits der Adresse entspricht, dementsprechend zählt der CPLD einen Zähler fünf Takte, in dem er die Adresse aufnimmt. Diese Adresse wird dann an die entsprechenden Multiplexer geleitet, die erst schalten, wenn die FSM von REC_ADD in den Zustand EXCHANGE_DATA wechselt. Die Daten werden dann in die entsprechende Entity geleitet.
Die Addresse selbst wird während sie aufgenommen wird zurück an den FPGA geleitet: Der FPGA weiss bei korrektem Rücklauf, dass wenigstens die Adresse korrekt übertragen wurde und kann annehmen, dass der Datentransfer korrekt abgelaufen ist [2].

Die Anzahl der Takte, welche übertragen werden, ist abhängig von der Anzahl an Daten-Bits, maximal also 32 Takte für maximal übertragbare 32 Bits. Dazu kommen am Anfang einer Übertragung 5 Takte, in denen 5 Adress-Bits übertragen werden.

Der CPLD bedient 6 elektronische Geräte auf *DajaSittah*.

1. Das System aus Multiplexern, welche die neuronalen Spannungen an eine Hochfrequenz-Buchse anlegen.
2. Die Kontrolle der LEDs, welche die Funktionsweise des CPLD anzeigt.
3. Die Kontrollsignale des *Spikey*.
4. Vom Temperatursensor werden regelmässig Temperaturdaten ausgelesen.
5. Vom ADC werden regelmäsig von zwei Kanälen analoge Spannungen gemessen. An diesen Kanälen liegen je die gleichen Signale wie bei den LEMO-Buchsen an.
6. Der DAC wird von aussen mit Daten versorgt, welche die neuronalen Spannungen einstellen.

Fragenstellungen bei der Simulation

1. Wird die Zustandsmaschine korrekt abgewickelt und springt die Maschine zum richtigen Zeitpunkt in den korrekten Zustand?
2. Werden die Daten korrekt vom FPGA zum CPLD bzw. vom CPLD zum FPGA korrekt in 5 + n Takten übertragen?
3. Wird immer für jede Addresse die richtige Entity angesprochen?
4. Mit welcher Frequenz lässt sich der CPLD reprogrammieren?
5. Können mehrere Endgeräte angesprochen werden?
6. Unter welchen Umständen entstehen Fehler?

[2] Eine weitere Methode konnte bis dahin nicht gefunden werden.

Testumgebung Zum Testen wurde eine Testbench geschrieben, welche die vorhandenen Entitys zusammenfasst.

1. Sowohl die Entity *cpld_ interface.vhd* als auch *fpga_ interface.vhd* befinden sich in der Testbench. Beide sind verbunden durch ihre SPI-Verbindung.

2. Das *cpld_ interface.vhd* ist mit den entsprechenden Testentitys *adc_ test.vhdl*, *dac_ text.vhdl* und *tmpsns_ test.vhdl* verbunden, welche die Endgeräte emulieren.

3. Die Testentitys nehmen die Werte entgegenen und verarbeiten sie. ADC, Temperatursensor und der *Spikey* erzeugen Bit-Vektoren, die an den FPGA gesendet werden. Sie sind an den CPLD via serieller Verbindung verbunden. Bei den Daten für die LEMO-Buchse, die Kontrollsignale des *Spikey* und den FIREIN-Leitungen sind Testentitys nicht notwendig, da sie direkt ausgegeben werden.

4. Das Signal SEND wird via einem Counter geschaltet. Das bedeutet, dass eine Zählvariable kontinürlich hochzählt und bei einem bestimmten Punkt das SEND-Signal auf '1' gesetzt wird. Die Frequenz mit der SEND gefeführt wird, ist einstellbar. Das SEND-Signal wiederrum aktiviert eines der NEW_<Entity>-Signale, welche der SPI-Kontrolle mitteilt, dass neue Daten für die jeweilige Entity gesendet werden.

5. Die entsprechenden Variablen, ob sie nun vom CPLD zum FPGA oder andersherum gesendet werden, werden kontinürlich inkrementiert.

6. Sobald das DATA_VALID auf '1' gesetzt wird, wird die Eingabe mit der Ausgabe verglichen. Ein <ENTITY>_TEST_BIT, wobei <ENTITY> jeweils jede Entity darstellt, die an den CPLD angeschlossen ist, wird gesetzt, wenn die jeweilige Eingabe und Ausgabe übereinstimmt.

2 Tests

Im Laufe der Tests wurden Daten nacheinander für jede Entity übertragen.

NEW_CPLD_TEST Dieser Modus dient dazu die Kommunikation zwischen FPGA und CPLD festzustellen. Ein festgelegtes 32 Bit langes Pattern wird vom FPGA zum CPLD und von diesem wieder zurueckgesendet. Um die Funktionalität der Adressübertragung zu kontrollieren, wurde die Adresse des CPLD-Tests auf "01000"festgelegt. Der CPLD-Test wird in der achten Zeitscheibe ohne Einfluss von FPGA eigenständig ausgeführt.

Ist der CPLD-Test positiv, wird das Bit-Pattern korrekt hin- und wieder zurückgesendet, kann festgestellt werden, dass das SHIFT-Register des FPGA funktioniert und der Adressregister korrekt übertragen wird.

Zustandsmaschine Die Zustandsmaschine kennt die Zustände WRITE_DATA_REG, SEND_ADDRESS, SEND_DATA, READ_DATA_REG und IDLING. Im IDLING-Zustand läuft eine Zählervariable: Beim Wert activate_entity = '1', ein Wert der alle 256 Takte gesetzt wird, prüft die Zustandsmaschine, ob neue Daten gesendet werden sollen. Das erfährt die FSM dadurch, dass die entsprechende Entity, welche in jener Zeitscheibe seine Daten senden darf, ein SEND-Signal setzt, falls neue Daten anliegen; sollte activate_entity = SEND = '1' sein, so schaltet die Zustandsmaschine von IDLING in WRITE_DATA_REG. Wichtig hier ist neben der korrekten Ausführung der FSM ist das Timing der Zustandswechsel.

Multiplexer und Demultiplexer Multiplexer und Demultiplexer erfüllen multiplexe Funktionen: Einstellung Anzahl Bits, welche beim Senden der Daten gesendet wird, Sie alle stellen ein, welche Entity die SPI-Kette bedient bzw. die Daten aus der Kette verwendet, sprich wie SDO und SDI angeschlossen sind. Hier existeren Probleme, wenn die Eingänge falsch angeschlossen sind oder der Ausgang nicht zum richtigen Zeitpunkt aktiviert wird. MUX und DEMUX selbst sind einfach aufgebaut: Wird eine SET-Variable eingestellt, wird der gewünschte Eingang eingestellt. SET wird gesetzt wenn oder kurz bevor die Zustandsmaschine von IDLING in WRITE_DATA_REG wechselt.

NEW_*SPIKEY*_CTRL Dieses Signal führt die Programmierung der Kontrollsignale von jeweils eines der *Spikey* aus. Diese bestehen aus CI_MODE, C_DELAY, PLL_RESET, PLL_BYPASS. Die Daten für die Kontrolle jeweils eines *Spikey* werden in Zeitscheibe 5, 6, 7 und 8 übertragen, sofern welche vorliegen. Das Übertragen der Daten läuft einwandfrei.

NEW_*SPIKEYS*_CTRL Dieses Signal programmiert sowohl die Kontrollsignale als auch die FIREIN-Signale für alle *Spikey*. Eventuell vorliegende Daten werden in Zeitscheibe 4 übertragen. Das Übertragen der Daten läuft einwandfrei.

***SPIKEYS*_LCKD** In regelmässigen Abständen ruft der FPGA diese Daten ab jeweils in Zeitscheibe 3. Diese Daten bestehen aus den vier PLL_LOCKED-Signalen, welche von den *Spikey* erzeugt werden.

NEW_LEMO Hier werden die entsprechenden Signale für die Multiplexer gesetzt. Das wären 5 Adress-Bits, welche den OUT_AMP-Kanal einstellt, und 5 ENABLE-Signale, welche den Multiplexer aktiviert.

Die Entity *lemo_test.vhd* nimmt die Daten an und setzt entsprechend der Adresse den Multiplexer. Der Nutzer kann also feststellen, ob der richtige Kanal auf dem MUX ausgegeben wird.

Die Daten für LEMO werden in Zeitscheibe 9 und 10 übertragen.

TMPSNS Temperaturdaten werden in regelmässigen Abständen beim CPLD abgerufen und vom FPGA vom CPLD. Interner Zähler aktivieren das SMS-Signal für den Temperatursensor. Der Temperatursensor wird nicht programmiert, die Standartprogrammierung ist für den Normalbetrieb ausreichend. Der Temperatursensor beansprucht Zeitscheibe 1.

ADC Die Implementierung des Vier-Kanal-ADC wurde stark verändert und anderen Anforderungen angepasst. Das entspricht je 12 Bits für analoge Spannungen für beide Kanäle, die verwendet werden. Zwischen CPLD und ADC werden 16 Bits verwendet, wobei die Adresse von der Entity *adc.vhd* festgelegt wird. Die Programmierung ist der Art, dass in bestimmter Frequenz die analogen Daten abgerufen und im CPLD gespeichert wird. Dabei wird jeweils Kanal 1 und 2 abgerufen, Kanal 3 und 4 werden nicht abgerufen, da diese Kanäle auf der Platine nicht von äusseren Spannungen belegt werden. FPGA und CPLD kommunizieren via 24 Bits; bei jedem Aufruf, werden beide Kanäle übertragen. Die Testbench erzeugt die analogen Daten mittels zwei 12-Bit-Vektoren die kontinuierlich inkrementiert werden. CPLD und ADC kommunizieren mittels 16 Bits. Es besteht nicht die Möglichkeit den ADC in irgendeiner Form zu programmieren, das Gerät läuft auf Standartprogrammierung; der ADC erlaubt nicht viele Programmierungen, die Standartprogrammierung ist ausreichend. Diese analogen Daten werden von der entity *adc_ test.vhd* seriallisiert und an die Entity *adc.vhd* gesendet, falls diese die Daten anfordert.

Der ADC beansprucht die Zeitscheibe 2.

DAC Zum Senden von Daten an den Acht-Kanal-DAC steht die Entity *dac.vhd* zur Verfügung. Die Entity wird acht mal implementiert für jeweils einen DAC-Kanal. Der DAC insgesamt benötigt 32 Bit, die ersten und letzten 4 Bits sind unbedeutend, das zweite 4-Bit-Packet besteht aus der Addresse für den entsprechenden Kanal, das dritte Vier-Bit-Packet besteht aus Kommandobits. *dac.vhd* nimmt die Daten vom FPGA an und sendet sie weiter zum DAC, ohne sie in irgendeiner Art zu bearbeiten. Der DAC sendet keine Daten zum FPGA, da er keine zu senden hat. *dac.vhd* wurde überarbeitet: Es ist nicht mehr möglich den DAC mittels CMD zu programmieren, da der Nutzer den DAC nicht programmieren muss. Stattdessen wird immer der CMD "0011"übertragen, der den DAC dazu veranlasst den DAC zu programmieren und den Kanal der entsprechenden analogen Spannung zu aktualisieren.

Des weiteren hat jeder DAC-Kanal seine eigene Entity-Addresse; Die Addressen starten von "10000"für den Kanal von VM bis "10111"für VIREF3. Die entsprechenden Zeitscheiben sind 16 bis 24.

Der GPLED wird vollständig vom CPLD kontrolliert. Er zeigt an, ob ein Temperaturalarm vorliegt oder ob die LOCKED-Signale des *Spikey* gesetzt wurden.

2.1 Versenden von Daten, welche an mehrere Entitys versendet werden

Der Scheduler und das Zwischenspeichern der Eingabe macht es unwahrscheinlich, dass Daten verworfen oder nicht gesendet werden. Selbst wenn neue Daten an einer Entity

Sachverhalt	Taktzahl
SEND_ADDRESS	5
SEND_DATA_REG	max. 32
READ_DATA_REG, WRITE_DATA_REG	1
Minimaler Wartezyklus	256
Minimaler Abstand von 2 Zeitscheiben	256
Maximale Wartezeit zum Versenden von Daten	32 * (37 + 256)

anliegen, während die vorher angelegten Daten gesendet werden, werden die neuen Daten korrekt zwischengespeichert und dann in der nächsten Runde gesendet. Werden allerdings Daten an die Entity angelegt, während die vorher angelegten Daten nicht gesendet wurden, verwirft das System die alten nicht gesendeten Daten verworfen. Um zu erkennen, ob Daten überhaupt gesendet wurden, beherbergt jede Entity einen DATA_ACK-Bit; dieses Bit wird auf '1' gezogen, wenn SEND auf '0' gesetzt wurde und es wird auf '0' gezogen, wenn NEW_<ENTITY> auf '1' gezogen wird. Die Reihenfolge der Zeitfenster ist festgelegt, da der Adressvektor alle 256 Takte inkrementiert wird. Das Versenden der Daten verläuft problemlos ohne Konflikte.

2.2 Übertragungszeit

Daten werden korrekt in 5 + n Takten vom Zeitpunkt des Sendebeginns versendet. Vom Programmieren eines Datums bis zum Senden, vergehen maximal

$$Takte = AnzahlZeitscheiben * (Zustandsmaschine + IDLING - Zeit) = 32 * (37 + 256) Takte. \quad (1)$$

3 Ergebnisse

Folgende Ergebnisse wurden festgestellt.

3.1 Fehlerquellen

Es konnten keine konzeptionellen Fehler in der Simulation gefunden werden.

Fazit Die Daten werden in jedem Fall korrekt übertragen.
Es konnte sichergestellt werden, dass die Kommunikation zwischen CPLD und FPGA funktioniert.

4 Eckdaten

Adresse	Entity	Datenregister	Implementierungen
00000	Temperatursensor	13	1
00001	ADC	24	1
00010	*Spikey*_LCKD	6	1
00011	ALL_*Spikey*	14	1
00100 bis 00111	*Spikey*_CTRL	4	4
01000	CPLD_TEST	31	1
01001 bis 01010	LEMO	10	2
01110	FIREIN	10	4
10000 bis 10111	DAC	32	8
11000 bis 11111	empty		