

nutzt und sind für transaktionsbasierte Verifikation optimiert.«

Die CHIPit-Lösungen sind 1 Mio. Mal schneller als Software-Simulatoren. Wichtiger aber ist, dass damit auch Tests durchgeführt werden können, die früher nicht möglich waren, weil z.B. ein Betriebssystem gebootet werden kann. Laut Jäger sind die Rapid-Prototyping-Lösungen von Synopsys darüber hinaus auch noch deutlich kostengünstiger. Liegen hier die Preise zwischen 20.000 und 100.000 Dollar, so muss für einen Emulator zwischen 2 und 10 Mio. Dollar bezahlt werden. Jäger: »Hinzu kommt noch, dass die Emulatoren auch noch deutlich langsamer sind.«

Dass Kunden bislang ihre Prototypen-Boards selbst aufgebaut haben, sieht Jäger ebenfalls nicht als Problem für Synopsys an. Denn es stellt sich immer die Frage: Selbstmachen oder kaufen? Und hier ist sich Jäger sicher, dass Kaufen durchaus Vorteile für die Entwickler bringt. Laut seiner Schätzung brauchen Firmen rund 6 Mannmonate, um ein Board zu bauen. Hinzu kommen noch die Komponentenkosten, die im Schnitt bei 20.000 Dollar pro Board liegen. Pro Design sind rund 10 Boards notwendig, so dass allein die Komponenten auf rund 200.000 Dollar kommen. Hinzu kommen noch versteckte Kosten wie Garantie, Dokumentation,

Support, Zeit etc. Jäger: »Unsere Boards liegen bei 30.000 Dollar, mal 10 ergibt 300.000 Dollar und die Boards sind innerhalb von ein paar Tagen zu haben.«

Ergänzt werden die Produktangebote durch ein breites Spektrum an Service-, Training- und Support-Optionen. Des Weiteren bietet Synopsys umfassenden weltweiten Support. Synopsys plant ab Februar 2009 eine Serie von halbtägigen Management-Seminaren sowie ganztägigen technischen Workshops zum Thema Rapid-Prototyping. Weitere Informationen und Anmeldungen zu den Seminaren finden Sie unter: <http://www.synopsys.com/Company/Pages/Events.aspx>. (st) ■



Jürgen Jäger, Synopsys

» Prototyping gibt es schon lange. Aber bislang waren das eher ‚Bastellösungen‘, bei denen der Anwender verantwortlich für die Funktionsfähigkeit war. «

Selbstversuch

Wird ein Softie hart?

Kann ein Ingenieur mit Erfahrung in der Hardware-nahen Programmierung mal eben auch ein FPGA programmieren? Ein Selbstversuch soll Aufschluss geben.

Ingenieure sind neugierige Menschen. Redakteure sind auch neugierige Menschen. Wenn ein Ingenieur zum Redakteur wird, dann ist die Neugier schon mehr als eine Berufskrankheit. Wenn dann noch ein sehr bemerkenswertes

technisches Konzept in unmittelbarer Reichweite kommt, ist die Verlockung gigantisch. Ich gestehe: Als ich vom Quickstart-Kit »phyCore-MPC5200B« von Phytex erfuhr – einer Kombination aus MPC5200 und einem FPGA in einem Komplettpaket, das sich innerhalb einer Stunde in Betrieb nehmen lassen soll, übermannte mich der berühmte Gedanke: »Haben will!«

Neben dieser impulsiven Begehrlichkeit nagte schon seit geraumer Zeit das Interesse an der

Programmierung eines FPGAs mittels VHDL. Mal abgesehen von einer recht kurzweiligen Erfahrung in der Programmierung eines »AGA«-Bausteins während des Studiums, lag der Entwicklungsschwerpunkt bei mir auf der Hardware-nahen Programmierung – heterogene Mehrprozessorsysteme machten dabei besonders Spaß. Bei den Programmiersprachen lag der Schwerpunkt auf C – Assembler und Objektorientierte Sprachen wurden aber auch nicht gescheut.

Als Softworker tendiert man als erstes dazu, jedes Problem in einen Algorithmus und dann in den Prozessor zu packen; ist die Hard-

ware dabei überfordert, denkt man zuerst an mehr Prozessortakt und Speicher – eine Unsitte aus der allgemeinen IT. Aber nicht jedes Embedded-Projekt kann sich diesen Luxus leisten: Entweder erlaubt das Power-/Wärme-Budget keine Aufrüstung, oder der Einsatztemperaturbereich schränkt die Auswahl der Bausteine drastisch ein. An die Problematik der Abkündigung von Bauteilen und damit dem »Sterben« von Schnittstellen bzw. der Neuprogrammierung der Ersatzbausteine denkt man eigentlich nicht. Das Thema Sicherheit/Verschlüsselung/Plagiatenschutz ist im Embedded-Bereich auch erst in den »Kinderschuhen«.



Das FPGA-Kit

Die Kombination aus dem leistungsfähigen 32-Bit-Controller MPC5200B (400 MHz) und dem FPGA-Baustein Altera Cyclone II (EP2C8) bietet Phytex auf dem Modul »phyCORE-MPC5200B-IO«. Der Controller hat eine MMU sowie ein FPU. Auf dem Modul werden unter anderem bis zu 128 MByte DDR-RAM, bis zu 64 MByte Flash, 10/100-MBit/s-Ethernet, RTC, 2 x UART, USB-Host-Interface und 2 x CAN zur Verfügung gestellt.

Für den schnellen Serieneinstieg bietet Phytex ein Rapid Development Kit mit Inbetriebnahmegarantie mit Basisplatine, Modul, Netzteil, Kabelsatz und Software. Im Kit ist das OSADL-konforme Echtzeit-Linux (Phytex-Dis-

tribution) mit entsprechenden Tool-Chains und insbesondere auch mit der »Altera-Quartus«-Tool-Chain (Web-Edition) zur FPGA-Code-Entwicklung enthalten. Die ausführlichen Quickstart-Instructions führen Schritt für Schritt durch die Inbetriebnahme und zeigen praxisgerechte Wege zur schnellen Softwarerealisierung auf. Die Anbindung des FPGA an den MPC5200 und an die Steckerleisten zur Basisplatine ist optimiert für I/O-Anwendungen. Der »Local Plus«-Bus steht dem FPGA ebenfalls zur Verfügung. PCI- und IDE-Interfaces des Controllers sind auf die Steckerleisten geführt. Das FPGA kann vom Anwender frei programmiert werden und ermöglicht einen flexiblen Einsatz des phyCore-

Moduls in einer großen Bandbreite unterschiedlicher Serienanwendungen, bei denen es auf spezifische Aufbereitung und schnelle Verarbeitung von I/O-Signalen ankommt.

Als Besonderheit der Phytex-Entwicklung ist die direkte Umprogrammierung des FPGAs über den MPC5200-Controller zu sehen. Im Gegensatz zum Laden eines statischen Codes – wie sonst üblich – wird der FPGA-Code vom MPC5200 aus dem Flash gelesen und in das FPGA bei jedem Start programmiert. Für Serienprodukte ergibt sich hieraus die flexible Möglichkeit von Updates per Software im installierten Gerät (selbst wenn sich dieses schon beim Endanwender befindet). (mk)

Für die meisten der genannten Probleme bieten sich FPGAs als Alternative an: Algorithmen können Strom sparend beschleunigt und Schnittstellen »ewig« geliefert werden, zudem lässt sich das eine oder andere an Intellectual Property gut verstecken. Nach nur kurzem Überlegen fallen einem vielfältige Möglichkeiten ein, diese Bausteine zu nutzen.

Auch wenn man von der Programmierung eines FPGAs spricht, so meint man doch eigentlich Schaltungsdesign – und das ist ja nicht die Baustelle eines Softworkers. Stolpert ein Softie aber mal über VHDL-Code, so fühlt man sich angespornt, den Code lesen zu wollen – und über weite Strecken »versteht« man ihn auch bzw. erschließt, wofür er gut ist. In Summe: Es ist Zeit, Erfahrungen am eigenen Leib zu machen. Eine Redaktion ist aber nicht für Schaltungsdesign eingerichtet – zumindest solange nicht, bis eine gerade mal 30 cm x 50 cm x 11 cm große Schachtel von Phytec eintrifft. Den Mainzern ist es gelungen, alles hineinzupacken, was man für eine ordentliche Evaluierung braucht: ein serientaugliches CPU-Modul mit integriertem FPGA, ein passendes Eval-Board, ein Expansion-Board (Lochrasterplatine), Software sowie die notwendige Dokumentation und Quickstart-Instructions (Handbuch). Aber auch an einen JTAG-Programmieradapter und die notwendigen Kabel und Netzteile hat Phytec gedacht. Man braucht also nur einen PC mit Windows, alles andere liefern die Mainzer muntergültig – Kompliment, das ist wirklich ein Komplettpaket!

Für die Inbetriebnahme der Software und Hardware veranschlagt das Unternehmen weniger als eine Stunde. Wenn es bis dahin nicht klappt, steht der Support mit Rat und Tat bereit. Innerhalb der Stunde gelang es mir allerdings nicht, die im Handbuch beschriebenen Schritte durchzuführen. Das lag aber nur daran, dass ein Lizenzschlüssel für die Altera-Entwicklungssoftware nicht rechtzeitig zu-

gemailt wurde – welcher Mail-Server bummelte, war nicht klar. Rechnet man die Wartezeit raus, kann man in deutlich weniger als einer Stunde zum »Hello world«-Äquivalent eines FPGAs kommen: eine blinkende LED.



Das FPGA-Kit von Phytec ist ein Komplettpaket mit CPU-Modul, Eval-Board und Software . . .



. . . auch Kabel und JTAG-Programmieradapter sind mit dabei.

Angespornt von diesem ersten Erfolgserlebnis hält das gelungene Handbuch weiterführende Kapitel für die Einführung in die Software-Tools, das FPGA und die Linux-Anbindung zum Modul-Prozessor bereit. Das Handbuch zeigt die ersten Schritte und das Potenzial des Paketes auf. Es ist aber kein Lehrbuch für VHDL oder eine Benutzerreferenz für Alteras »Quartus-II«-Software – darum soll sich der Neugierige schon selber kümmern. Was Phytec liefert, ist eine sehr solide Ausgangsbasis zur Evaluation des serientauglichen Moduls und dem darin schlummernden gehörigen Potenzial; als Ausbildungsobjekt ist es nicht gedacht – lehrreich ist es aber trotzdem.

Das Rüstzeug ist also vorhanden und der Eindruck gewonnen, dass für einen Softie eine FPGA-Programmierung durchaus möglich

sein könnte – weitere Experimente sind allerdings erforderlich.

Denn will man mehr, als an den vorhandenen Beispielen herumzubasteln, muss man ganz klar VHDL lernen. Der Aufwand für die Syntax entspricht dem einer üblichen

Programmiersprache: Kennt man C, kommt einem vieles bekannt vor. Länger dauert es allerdings, sich von den gewohnten Denkprozessen zu lösen: Man hat halt keinen Prozessor vor sich, aber auch keinen simplen I/O-Baustein. Um das volle Potenzial erkennen und nutzen zu können, braucht man vermutlich mehrere Projekte. Allerdings kommt man recht schnell auf das Niveau, um Open-Source-VHDL-Code einzubinden und kleinere Änderungen, z.B. I/O-Zuweisungen, durchzuführen. Um lahmende Software-Algorithmen in schnelle FPGA-Schaltungen überzuführen, braucht es Erfahrung und damit viel Zeit. Die Grundlagen für den Entwurf von Digitalschaltungen muss man aber schon haben – Elektro-/Elektronikingenieure sollten zurecht kommen, reine Informatiker werden aber kein einfaches Los haben.

Kann also ein Softworker hart werden? Ja – aber wie bei der Echtzeit ist die Frage, was »hart« ist, eine Sache der Definition. Nutzt man vorgefertigten VHDL-Code, um ihn für sein Projekt anzupassen, ist der Einarbeitungsaufwand recht überschaubar, und man kann sein neues Wissen durchaus produktiv nutzen: Der Arbeitsaufwand, schlecht dokumentierte Interfacekarten/-module mit mäßigen Treibern in ein System zu integrieren und die Software darauf anzupassen – und zu debuggen –, kann schnell größer sein, als VHDL-Code anzupassen, in den man schließlich reinschauen kann. Aber so ist man noch lange kein Schaltungsentwickler – man hat sich »nur« eine neue Art von Funktionsbibliothek erschlossen. Mein persönliches Fazit deshalb: Ausprobieren schadet nicht – ein Ingenieur kann nie zu viele Tricks kennen, sollte aber auch immer wissen, wo seine Grenzen liegen. (mk) ■