Wie funktionieren FPGAs?

Veröffentlichungsdatum: Mai 27, 2013

Inhaltsverzeichnis

Bei FPGAs (Field-Programmable Gate Arrays) handelt es sich um wiederprogrammierbare Schaltkreise. Der erste FPGA wurde 1985 von Ross Freeman erfunden, dem Mitbegründer der Firma Xilinx. Da FPGAs die besten Eigenschaften von ASICs und prozessorgestützten Systemen vereinen, werden sie branchenübergreifend eingesetzt. Im Gegensatz zu ASICs bieten FPGAs demzufolge hardwaregetaktete Geschwindigkeit und Zuverlässigkeit bei gleichzeitiger Kosteneffizienz.

Laden Sie sich ein kostenloses Resource Kit zum Thema FPGA herunter

Die wiederprogrammierbaren Siliziumchips sind genauso flexibel wie Software, die auf einem Prozessor ausgeführt wird. Ihre Leistungsfähigkeit wird jedoch nicht von der Anzahl der verfügbaren Prozessorkerne eingeschränkt. Im Unterschied zu Prozessoren bieten FPGAs echte Parallelität, so dass verschiedene Verarbeitungsoperationen nicht auf die gleiche Ressource angewiesen sind. Jeder einzelne Verarbeitungs-Task wird einem dedizierten Bereich auf dem Chip zugewiesen und kann so autonom und ohne Beeinflussung anderer Logikblöcke ausgeführt werden. Daher wird die Leistungsfähigkeit der Anwendung nicht eingeschränkt, wenn weitere Verarbeitungs-Tasks hinzugefügt werden.

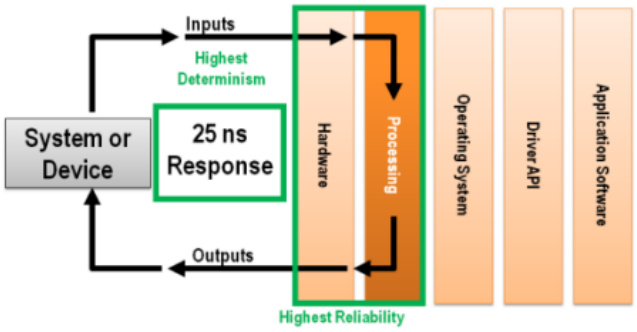

Abb. 1: Einer der Vorzüge von FPGAs gegenüber prozessorbasierten Systemen besteht darin, dass die Anwendungslogik in Hardwareschaltkreise implementiert und nicht auf einem Betriebssystem mit Treibern und Anwendungssoftware ausgeführt wird.

Das vorliegende Whitepaper wurde für Entwickler erstellt, die über wenig Erfahrung im Bereich der FPGA-Programmierung verfügen. Es dient als Einführung in die Bausteine eines FPGAs und in die Entwicklungswerkzeuge für rekonfigurierbare Schaltkreise.

Bestandteile eines FPGAs

Ein FPGA-Chip besteht aus einer bestimmten Anzahl vordefinierter Komponenten mit programmierbaren Verbindungsleitungen und dient der Entwicklung rekonfigurierbarer digitaler Schaltungen und I/O-Blöcke, so dass der Schaltkreis auch auf die Umgebung zugreifen kann.

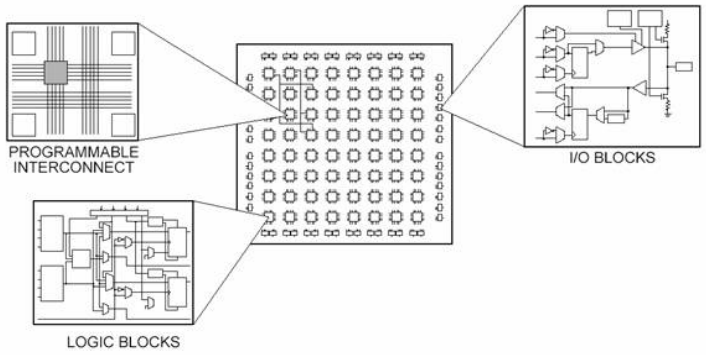

Abb. 2: Bestandteile eines FPGAs

Spezifikationen für FPGA-Chips bezeichnen in der Regel die Anzahl konfigurierbarer Logikblöcke und festgelegter Funktionslogikblöcke, wie z. B. Multiplizierer, sowie die Größe der Speicherelemente, wie z. B. Embedded-Block-RAM. Ein FPGA-Chip umfasst zahlreiche weitere Komponenten, jedoch sind diese die bedeutendsten, wenn FPGAs für eine bestimmte Anwendung ausgewählt und verglichen werden sollen.

Die grundlegende Logikeinheit eines FPGAs sind die konfigurierbaren Logikblöcke (CLBs). CLBs werden mitunter auch als Slices oder Logikzellen bezeichnet und bestehen aus zwei grundlegenden Komponenten: Flipflops und Lookup-Tabellen (LUTs). Die verschiedenen FPGA-Familien verfügen über unterschiedliche Architekturen, je nachdem, wie Flipflops und LUTs miteinander kombiniert sind.

Mehr über die einzelnen Komponenten erfahren Sie durch Öffnen nachfolgender Abschnitte.

+ **Mehr über Flipflops**

+ **Mehr über LUTs**

+ **Mehr über Mulitplizierer und DSP-Slices**

+ **Mehr über Block-RAM**

Entwicklung von FPGAs für ein System

Die vorgestellen Eigenschaften der grundlegenden FPGA-Komponenten lassen erkennen, welche Vorteile die Implementierung von Logik in Hardwareschaltkreise bietet: Es werden Verbesserungen bei der Ausführungsgeschwindigkeit, Zuverlässigkeit und Flexibilität erzielt. Wird jedoch nur ein FPGA für die Verarbeitung und die I/O-Anschlüsse im System eingesetzt, ist auch mit Nachteilen zu rechnen: FPGAs verfügen nämlich nicht über das Treiber-Ökosystem und die Programmcode-/IP-Basis von Mikroprozessor-Architekturen und -Betriebssystemen. Zusätzlich stellen Mikroprozessoren in Kombination mit Betriebssystemen die Grundlage für Dateistrukturen und die Kommunikation mit Peripheriegeräten dar, welche für viele, oftmals grundlegende Tasks wie das Loggen von Daten auf die Festplatte eingesetzt werden.

Daher hat sich in den letzten zehn Jahren eine Hybridarchitektur (heterogene Architektur) entwickelt, bei der ein Mikroprozessor mit einem FPGA gekoppelt wird, der dann an I/O angebunden wird. Dieser Ansatz vereint die Vorzüge beider Hardwaresysteme. Unternehmen wie z. B. Xilinx mit seiner Zynq-Architektur verfolgen diesen Ansatz seit einigen Jahren und haben Lösungen hervorgebracht, die Prozessor und FPGA auf einem einzigen Chip kombinieren, so dass die genannte Hybridarchitektur entsteht.

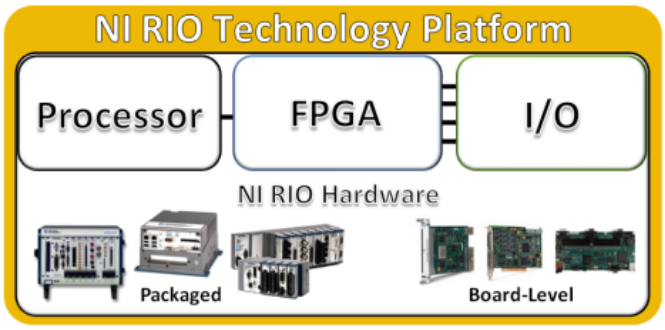

Abb. 8: NI verfügt über eine ganze Reihe an RIO-Geräten, sowohl mit Gehäuse als auch reine Platinen, die mithilfe von LabVIEW basierend auf der Hybridarchitektur aus Mikroprozessor und FPGA programmiert werden können.

National Instruments implementiert diese leistungsfähige Architektur aus Mikroprozessor und FPGA seit neun Jahren in seine Geräte mit rekonfigurierbarer I/O (RIO). Diese sind in vielen verschiedenen Formfaktoren erhältlich und reichen von kompakten bis zu leistungsstarken Systemen, die allesamt auf der gleichen RIO-Architektur basieren.

» Mehr zu FPGA-basierter RIO-Hardware von NI

FPGA-Designwerkzeuge

Nun stellt sich noch die Frage, wie diese unzähligen Komponenten konfiguriert werden müssen, um die gewünschten logischen Funktionen aufzubauen.

Dazu werden die digitalen Algorithmen unter Einsatz von Entwicklungswerkzeugen in Software definiert und anschließend in eine Konfigurationsdatei kompiliert. Dieser sogenannte Bitstream enthält Informationen zur Verbindung der einzelnen Komponenten im FPGA. In der Vergangenheit stand FPGA-Technologie nur den Anwendern zur Verfügung, die über fundiertes Wissen im Bereich digitales Hardwaredesign verfügten. Der verstärkte Einsatz von High-Level-Synthesewerkzeugen wie die Systemdesignsoftware NI LabVIEW vereinfacht jedoch auch die FPGA-Programmierung, da neue Technologien grafische Blockdiagramme und sogar C-Programmcode in digitale Hardwareschaltungen konvertieren können.

Traditionelle FPGA-Entwicklungswerkzeuge

Im Laufe der ersten 20 Jahre der FPGA-Entwicklung wandelten sich Hardwarebeschreibungssprachen (HDLs, Hardware Development Languages) wie VHDL und Verilog in die Primärsprachen für den Entwurf von Algorithmen, die auf dem FPGA ausgeführt werden. Diese systemnahen Sprachen umfassen einige der Vorzüge, die auch von anderen textbasierten Programmiersprachen geboten werden mit dem Unterschied, dass mit ihnen auf einem FPGA ein Schaltkreis entworfen wird. Die resultierende hybride Syntax erfordert, dass Signale abgebildet oder von externen I/O-Anschlüssen mit internen Signalen verbunden werden, die schließlich mit den Logikblöcken verdrahtet werden, welche die Algorithmen beinhalten. Diese Logikblocke werden sequenziell ausgeführt und können Verweise zu anderen Logikblöcken auf dem FPGA herstellen. Die Parallelität der Taskausführung auf einem FPGA lässt sich allerdings nur schwierig sequenziell und Zeile für Zeile darstellen. HDLs weisen auch einige der Merkmale anderer textbasierter Sprachen auf, unterscheiden sich jedoch wesentlich von ihnen, weil sie auf einem Datenflussmodell basieren, bei dem I/O über Signale an eine Reihe von Funktionsblöcken angebunden ist.

Zur Prüfung der von FPGA-Programmierern erstellten Logikfunktionen ist es üblich, Testumgebungen in HDL zu verfassen, um den FPGA-Entwurf abschließend mittels Simulieren von Eingaben und Überprüfen von Ausgaben zu erproben. Die Testumgebung und der FPGA-Code werden in einer Simulationsumgebung ausgeführt, die das Hardware-Timing-Verhalten des FPGA-Chips modelliert und dem Entwickler alle Ein- und Ausgangssignale für die Testvalidierung zur Verfügung stellt. Das Erstellen der HDL-Testumgebung sowie das Ausführen der Simulation erfordern häufig größeren Zeitaufwand als das eigentliche Erstellen des ursprünglichen FPGA-Entwurfs mit HDL.

Nach Erstellen des FPGA-Entwurfs mit HDL sowie dessen Prüfung muss dieser in ein Komplilierwerkzeug eingespeist werden, das die textbasierte Logik aufnimmt und über mehrere komplexe Schritte die HDL in eine Konfigurationsdatei oder in einen Bitstream synthetisiert, der Informationen darüber enthält, wie die Komponenten miteinander zu verdrahten sind. Als Teil dieses manuellen und mehrere Schritte umfassenden Prozesses ist häufig die Abbildung von Signalnamen auf den Pins des verwendeten FPGA-Chips erforderlich.

```
-- First we sychronize the asynchronous digital input to our clock
-- by inserting two flip flops.
SynchronizationFFs:
process( aReset, Clk )
begin
  if aReset then
    cDigitalInput_ms <= false;
    cDigitalInput <= false;
  elsif rising_edge(Clk) then
    cDigitalInput_ms <= aDigitalInput;
    cDigitalInput <= cDigitalInput_ms;
  end if;
end process SynchronizationFFs;

-- Then we keep track of what the digital input was on the previous
-- clock cycle by inserting another flip flop
PreviousDigitalInputFF:
process( aReset, Clk )
begin
  if aReset then
    cPrevDigitalInput <= false;
  elsif rising_edge(Clk) then
    cPrevDigitalInput <= cDigitalInput;
  end if;
end process PreviousDigitalInputFF;

-- Then we have a little combinatorial logic to detect a rising edge
cRisingEdgeDetected <= cDigitalInput and not cPrevDigitalInput;

-- And finally we have a register that increments when that rising
-- edge is detected.
CounterRegister:
process( aReset, Clk )
begin
  if aReset then
    cCountReg <= (others=>'0');
  elsif rising_edge(Clk) then
    if cRisingEdgeDetected then
      cCountReg <= cCountReg + 1;
    end if;
  end if;
end process CounterRegister;
cCount <= cCountReg;

end rtl;
```

Abb. 9: FPGA-Entwurf eines einfachen Counters in VHDL

Die Herausforderung in diesem Entwicklungsprozess besteht letztlich darin, dass das für die Programmierung in traditionellen HDLs erforderliche Know-how nicht weitverbreitet ist und die FPGA-Technologie daher lange Zeit für den Großteil der Ingenieure und Wissenschaftler unzugänglich war.

Werkzeuge für die High-Level-Synthese

Mit der Einführung grafischer Entwicklungswerkzeuge zur High-Level-Synthese, wie beispielsweise NI LabVIEW, sind einige der größten Hürden im traditionellen HDL-Entwurfsprozess beseitigt worden. Die Entwicklungsumgebung LabVIEW ist besonders für die FPGA-Programmierung geeignet, weil sie Parallelität und Datenfluss eindeutig abbildet, so dass im herkömmlichen FPGA-Design sowohl erfahrene als auch unerfahrene Anwender die FPGA-Technologie voll ausnutzen können. Zudem kann mit LabVIEW bestehender VHDL-Code in LabVIEW-FPGA-Entwürfe integriert werden, damit IP nicht verloren geht.

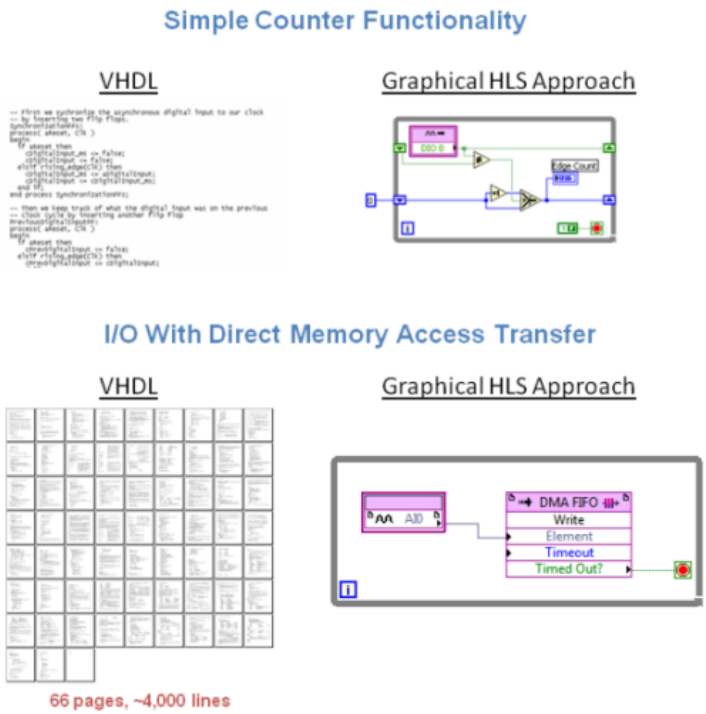

Abb. 10: Rechts dargestellt ist die Systemdesignsoftware LabVIEW, die als High-Level-Entwicklungswerkzeug für FPGAs in NI-RIO-Hardware dient. LabVIEW ermöglicht die Abstraktion der systemnahen Komplexität, wie sie häufig bei der Erstellung und Skalierung von VHDL-Entwürfen vorkommt.

LabVIEW bietet weiterhin direkt in der Entwicklungsumgebung Funktionen zur Simulation und Überprüfung des Verhaltens der FPGA-Logik. Auch ohne Kenntnisse der systemnahen HDL-Programmiersprache können Anwender Testumgebungen erstellen, um die Logik von Entwürfen zu erproben. Zusätzlich unterstützt die Flexibilität der LabVIEW-Umgebung fortgeschrittenere Anwender bei der Modellierung von Timing und Logik ihrer Entwürfe mit Exporten in zyklusgenaue Simulatoren wie Xilinx ISim.

Die Kompilierwerkzeuge von LabVIEW FPGA automatisieren den Kompilierprozess, so dass sich der Vorgang mit einem Klick starten lässt und Berichte sowie gegebenenfalls Fehler gemeldet werden, sobald die Kompilierungsphasen abgeschlossen sind. Falls Taktfehler aufgrund des FPGA-Entwurfs auftreten, hebt LabVIEW diese kritischen Pfade grafisch hervor, um die Fehlerbehebung zu beschleunigen.

» Erfahren Sie mehr in dem Webcast „Einführung in LabVIEW FPGA“

Fazit

Die Verbreitung der FPGA-Technologie nimmt weiterhin zu, da komplexe Werkzeuge wie LabVIEW, Standard-Mikroprozessoren und die FPGA-RIO-Architektur FPGAs zugänglicher machen. Dabei ist es jedoch wichtig, sich mit dem Innenleben von FPGAs vertraut zu machen, um zu verstehen, was bei der Kompilierung eines Blockdiagramms zur Ausführung in Hardware passiert. Der beste Weg, den passenden FPGA für eine Anwendung auszuwählen, ist der Vergleich von Flipflops, LUTs, Multiplizierern und Block-RAM. Für die Entwicklung ist es enorm hilfreich zu wissen, wie Ressourcen genutzt werden, insbesondere wenn es um die Optimierung von Größe und Geschwindigkeit geht. Dieses Dokument soll keine vollständige Liste aller grundlegenden FPGA-Bausteine darstellen. Weitere Informationen zu FPGAs und dem Design digitaler Hardware finden Sie in den untenstehenden Ressourcen.

Nächste Schritte

Laden Sie sich ein kostenloses Resource Kit zum Thema FPGA herunter
Hard- und Software für NI FPGA evaluieren
Mehr über FPGA-Design mithilfe der Systemdesignsoftware NI LabVIEW
Mehr über die FPGA-Hardwareprodukte von NI