Xputer

Inhaltsverzeichnis

Geschichte der Prozessoren

Ende der 60er Jahre entwickelten die hiesigen Pioniere der Branche, Ilse und Otto Müller in Konstanz, noch Schaltungstechnik Konzepte, die von den neuen Chip- und Computer-Firmen aufgegriffen wurden. Sie verdrängten die erste Generation der ´Arbeitsplatzrechner´, die die Müllers für Heinz Nixdorf entwickelt hatten. Diese Konzepte waren die 16-Bit-Struktur, die freie Programmierbarkeit, das erste Multitasking-Betriebssystem (iTOS), der Einsatz von Bildschirm und Magnetbandkassette (Nachfolgemodell der Nixdorf 820, später unter dem Namen CTM 70 verkauft).
In den früheren 70er Jahren entstanden die ersten Mikroprozessoren von Intel (4004, 8008, 8080), Zilog (Z80), Motorola (6800) und MOS Technologies (6502), die legitimen Ahnen von Spare und Mips, Pentium, Alpha und PowerPC. Das Prinzip des Mikroprozessors, so flexibel wie möglich riesige Datenmengen in kürzester Zeit zu verarbeiten, trat seinen Siegeszug an. Ein Mikroprozessor allein vollbringt bis heute jedoch wenig. Stets ist er auf andere Bausteine angewiesen, die ihm Daten liefern und Ergebnisse weiterleiten – Chips, die den Zugriff auf Arbeits- und Festplattenspeicher regeln und die Grafikausgabe übernehmen. Andere Aspekte wie Zuverlässigkeit und Sparsamkeit in Herstellung und Betrieb traten im nun entbrannten Kampf um den schnellsten Prozessor in den Hintergrund. Klotzen, nicht kleckern hieß das Prinzip im gnadenlosen Überlebenskampf einer Branche, deren Produktzyklen immer kürzer wurden.
Im Rennen um Taktrate und Benchmark-Ergebnisse war der Kunde gleichzeitig Gewinner und Verlierer. Während in den 80er Jahren C64, ZX81 und Atari XL, demonstrierten, dass ein Computer für unter 500 Mark durchaus Nützliches leisten kann, liegt die Einstiegsschwelle heute mit 2000 Mark wesentlich höher. Allein die Kosten für den Prozessor übersteigen bisweilen den Preis der ´veralteten´, aber effektiven Systeme.
Im Gegensatz zum Mikroprozessor, der rein auf Leistung getrimmt ist, optimieren die Hersteller ihre Mikrocontroller, die auf das gleiche Herstellungsverfahren zurückgreifen, mit anderer Zielrichtung: Hohe Produktionsausbeute, geringer Energiebedarf und damit auch weniger Abwärme. Unempfindlichkeit gegen klimatische Bedingungen, Schmutz und elektromagnetische Störungen sind hier die ausschlaggebenden Faktoren. Zudem wird der Platzbedarf der Gesamtsysteme durch Integration der Peripherie wie Input/Output-Systeme (I/O), Speichercontroller (MMU), Analog/Digital- und Digital/Analog-Wandler (A/D, D/A) sowie DSP-Funktionen für Grafik, Sound und Kompressionsaufgaben minimiert.

Die Idee der Xputer

Professor Dr. Reiner Hartenstein begann als erster Deutscher im Jahre 1979 hierzulande Chip-Design als eigenes Fach zu lehren. Er hat mit das Konzept der Xputer entwickelt.
In einem Interview erläutert Professor Dr. Hartenstein die Idee des Xputers:

Wir stehen heute an dem Punkt, wo wir sehen der klassische Mikroprozessor ist am Ende. Selbst bei Konsum- Elektronik- Erzeugnissen verbrauchen die durch die Rückständigkeit der CPUs notwendig gewordenen Beschleuniger (beispielsweise 3D-Chips) wie mehr Silizium-Fläche als der Mikroprozessor selbst. Der Mikroprozessor wird dadurch zwar nicht überflüssig, aber er ist nichts weiter als der Schwanz, der mit den Hund wedelt, und teuer dazu. Auch diese Add- on- Beschleuniger sollten von ´fest verdrahtet´ mit hohen Design- Kosten auf programmierbare Plattformen umgestellt werden. Im Gegensatz zur festverdrahteten

Logik eines Prozessor-Chips lässt sich die Logik eines Xputers zur Laufzeit ändern. Es wird also nicht eine Folge einzelner Befehle aufgerufen und aus dem sequentiellen RAM ausgeführt, sondern mit einer strukturellen Programmierung hochparalleler Hardware gearbeitet. An die Stelle der prozeduralen Programmierung (Programmierung über die Zeit) tritt die strukturelle Programmierung (Programmierung im Raum).
Der Xputer bietet damit eine effizientere Art der Parallelität als die klassischen Parallel-rechner, ja nur eine Bündelung vieler Engpässe (´Von Neumann-Bottlenecks´) des klassischen Ansatzes sind, der auf sequentielle Bearbeitung von Programm-Codes setzt. Bei Xputer-Anwendungen tritt an die Stelle der horizontalen Parallelisierung die neuartige ´vertikale Parallelisierung´.
Für erste Anwendungen eignen sich vor allem solche Branchen, in denen Endgeräte als Marktöffner (fast) kostenlos abgegeben werden, wie etwa bei Kommunikations-Netzwerkdiensten, die von den den laufenden Gebühren ihren Profit machen. Hier ist – im Gegensatz zur PC – besonders langlebige Hardware gefragt. Nötige Upgrades für Bug-Fixes wegen häufiger Änderungen der Übertragungsprotokolle oder zur Verbesserung der Attraktivität der Dienste, möchte man hier billig und einfach per Broadcast über das Netz in die Endgeräte herunterladen. Eines von vielen Beispielen sind die künftigen Multimedia-Handys, die ja kleine Supercomputer sind: Sie nehmen beispielsweise mit der Camcorder-Funktion einen Video-Film auf und übertragen diesen sofort drahtlos ins Studio oder an Ihre Familie zu Hause. Anschließend surfen Sie mit dem gleichen Ding im Internet. Das alles funktioniert von jedem beliebigen Ort in der Welt aus.
Wenn dann bessere Algorithmen entwickelt werden, müssen Sie nicht jedes Mal das teure Gerät wegwerfen, weil nach dem alten Prinzip neue Support-Chips, also Beschleuniger, nötig würden, sondern Sie laden die Hardware-Upgrades einfach vom Netz, wie man das heute von Flash-Programmen für BIOS- oder Modem-Firmware kennt. Der Xputer ist so gesehen ein Hardware-programmierbarer Beschleuniger.

Ein Xputer Konzept

Das Xputer-Konzept der Uni Kaiserslautern löst sich vom konventionellen Ansatz, indem an der Stelle einer mit vielen kleinen Programmschritten gesteuerten, festverdrahteten ALU (Arithmetic Logic Unit, der Kern eines klassischen Prozessors) ein sogenanntes Kress-Array tritt. Diese von Dr. Rainer Kress – jetzt in der Forschung bei Siemens – entwickelte Matrix von r-ALUs erledigt die Verknüpfung der Daten.
Das Konzept fußt auf der Idee der FPGAs (Field Programmable Arrays): Derartige Bausteine enthalten eine Vielzahl von vielfältig verbindbaren Gattern, deren Zusammenspiel auch im Betrieb änderbar ist. So kann ein Teil eines solchen FPGAs im einem Moment als Multiplizierer, gleich darauf als Addierer und kurze Zeit später als Schieberegister fungieren. Andere Teile des Bausteins können gleichzeitig ergänzende Funktionen ausführen.
Diese Wandelbarkeit nutzt der Xputer, um sich in Abhängigkeit von der zu lösenden Aufgabe umzukonfigurieren. Die Konfigurationsinformationen stammen dabei nicht aus einem sequentiellen Programm, sondern aus vordefinierten Codes, die Blockweise geladen werden. Dabei muss man natürlich dafür sorgen, dass die zu verarbeitenden Daten so vorsortiert werden, dass nicht nach jedem kurzem Datenblock eine zeitraubende Umkonfiguration fällig wird: An Stelle der Befehlssequenzen treten Datensequenzen.
Auf höherer Ebene erscheint ein Xputer beispielsweise DSP-ähnlich für Audio-Daten-verarbeitung, als Rendering-Subsystem für 3D-Darstellungen oder als Decoder für DVD-Wiedergabe. Solange man nicht alles gleichzeitig will, könnten ´Coprozessoren´ für diese

Aufgaben entfallen. Bei speziellen Aufgaben der digitalen Signalverarbeitung und in der Bildverarbeitung soll eine experimentelle Xputer-Architektur Durchsatzsteigerungen bis zum Hundertfachen erreicht haben.
Vorteile der Xputer-Technik sind unter anderem, dass kein Programmspeicher im klassischen Sinn nötig ist und dass – abhängig von der Größe des rekonfigurierbaren Teils und der Aufgabenstellung – auch massiv parallele Lösungen denkbar sind.
Die Vielfalt hatte bisher auch einen massiven Nachteil: Die konfigurierbaren Gatter derzeit erhältlicher FPGAs benötigen etwa 100- bis 200fach mehr Transistorfunktionen – und damit Chipfläche – pro Gatter als gewöhnliche Prozessoren. Netto dient nur einer der Transistoren dabei der eigentlichen Anwendung, der Rest ist für die strukturelle Programmierbarkeit nötig.

Das Prinzip vom Xputer

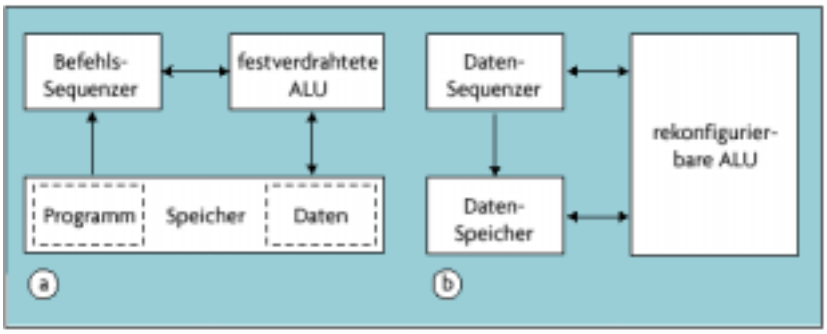


Bild 1. Unterschiede zwischen Computern (a) und Xputern (b): Sowohl die ALU als auch der Sequenzer spiegeln die jeweiligen Paradigmen wider.

Da ist, neben dem Daten-Sequenzer, auch noch eine „reconfigurable“ Arithmetic Logic Unit (rALU, Bild 1). Sie kann, im Unterschied zu den festverdrahteten ALUs herkömmlicher CPUs zur Laufzeit programmiert werden: eine softe Hardware, sozusagen. Ihre Bausteine sind reconfigurable Application Specific Intergrated Circuits, zu denen die Field Programmable Gate Arrays (FPGA) ebenso zählen wie andere Programmable Logic Devices (PLD). Für die Möglichkeit, die Hardware umprogrammieren zu können, ist leider ein heute noch Preis zu entrichten: Die softe Hardware verbraucht mehr Silizium als die festverdrahtete. Sowohl die Kosten als auch der Energieverbrauch der Chips steigen. Aber: Bei der nach der Jahrtausendwende zu erwartenden 0,1-Mikron-Technologie spielt das vielleicht gar keine Rolle mehr. Xputer verfügen über eine Reihe von Feldregistern, die Scan Fenster oder Scan Cache genannt werden (Bild 2).

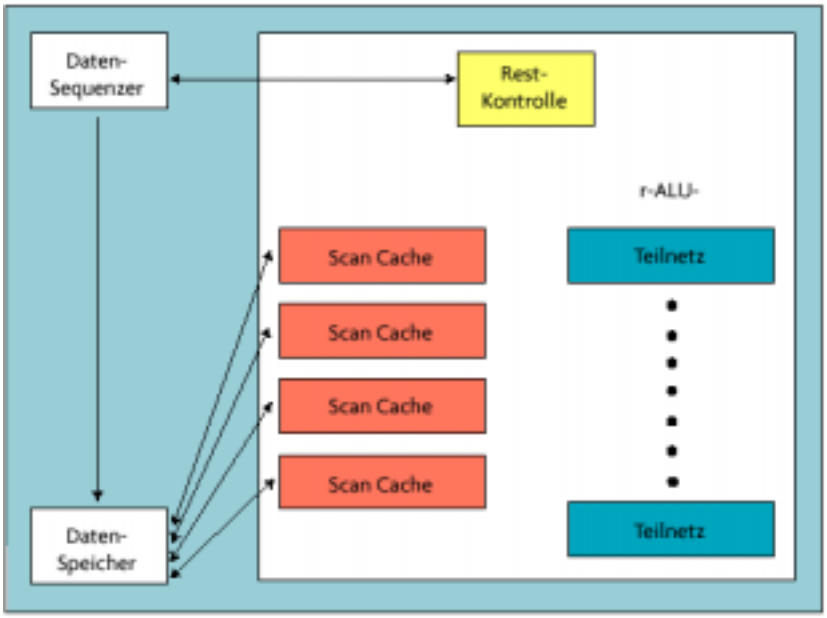
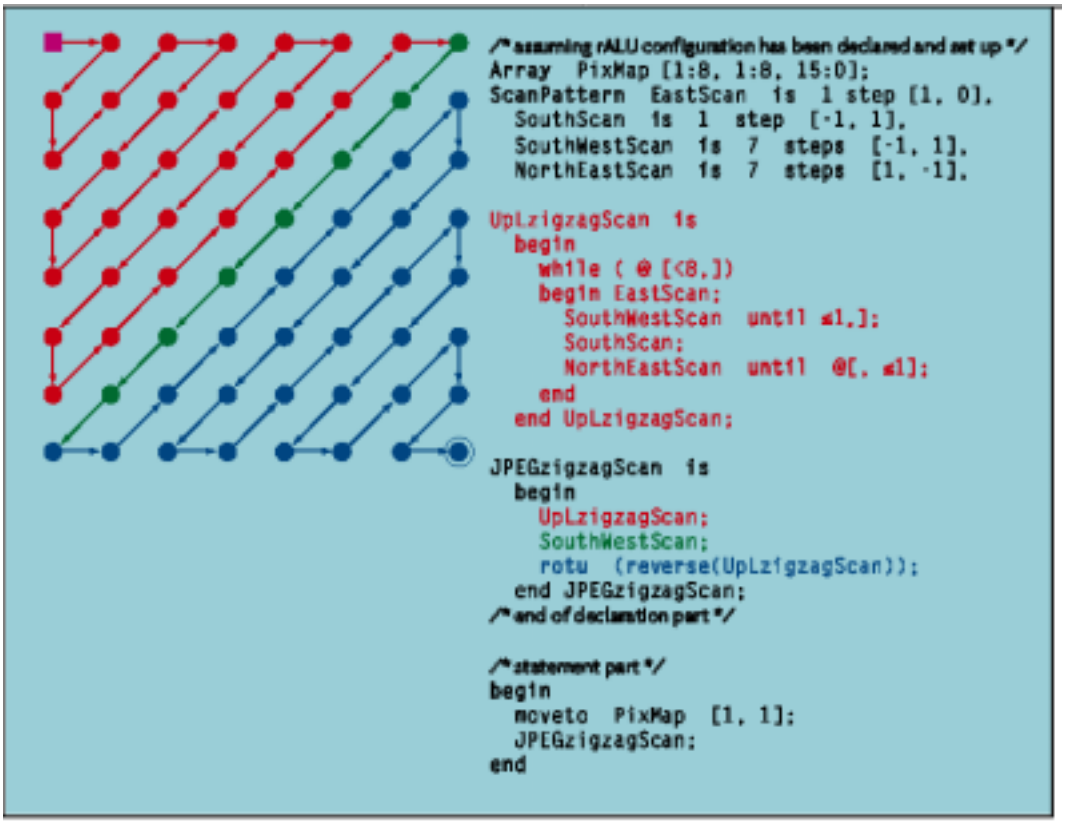


Bild 2. Datenaustausch zwischen Scan Cache und Datenspeicher: Der Daten-Sequenzer stellt ein festverdrahtetes, aber hoch parametrisiertes Repertoire von Daten-Adreßfolgen bereit.

Sie sind größenverstellbar und können wie ein adressierbares Fenster über den zweidimensionalen Datenspeicher gelegt werden. Der Scan Cache wird entlang eines Pfades über den Datenspeicher bewegt, der Scan-Muster oder Scan Pattern heißt (Bild 3).



```
/* assuming rALU configuration has been declared and set up */
Array  PixMap [1:8, 1:8, 15:0];
ScanPattern  EastScan  is  1 step [1, 0],
  SouthScan  is  1  step  [-1, 1],
  SouthWestScan  is  7  steps  [-1, 1],
  NorthEastScan  is  7  steps  [1, -1],

UpLzigzagScan  is
  begin
    while ( @ [<8,])
    begin EastScan;
      SouthWestScan  until <1,];
      SouthScan;
      NorthEastScan  until  @[, <1];
    end
  end UpLzigzagScan;

JPEGzigzagScan  is
  begin
    UpLzigzagScan;
    SouthWestScan;
    rotu  (reverse(UpLzigzagScan));
  end JPEGzigzagScan;
/* end of declaration part */

/* statement part */
begin
  moveto  PixMap  [1, 1];
  JPEGzigzagScan;
end
```

Bild 3. Scan Pattern für den JPEG-Algorithmus: Selbst komplexe Datenzugriffspfade sind in MoPL einfach zu beschreiben.

Den Datenaustausch zwischen Scan Cache und Datenspeicher regelt ein Daten- Sequenzer, der ein festverdrahtetes Repertoire von Daten- Adressfolgen bereitstellt. Ein einfaches Beispiel demonstriert das Zusammenspiel der Komponenten. Bild 4a beschreibt einen Algorithmus, wie er in einer Hochsprache formuliert sein könnte. Bild 4b zeigt den dazugehörigen Datenabhängigkeitsgraphen, aus dem der Compiler (Xpiler) eine Speicherkartierung ableitet (Bild 4c). Er spezifiziert Scan-Cache-Format und rALU-Teilnetze (Bild 4d), die sich wie Verbundoperatoren über das aktive Scan-Fenster legen. Schließlich legt der Xpiler auch das Scan Pattern fest (Bild 4e).

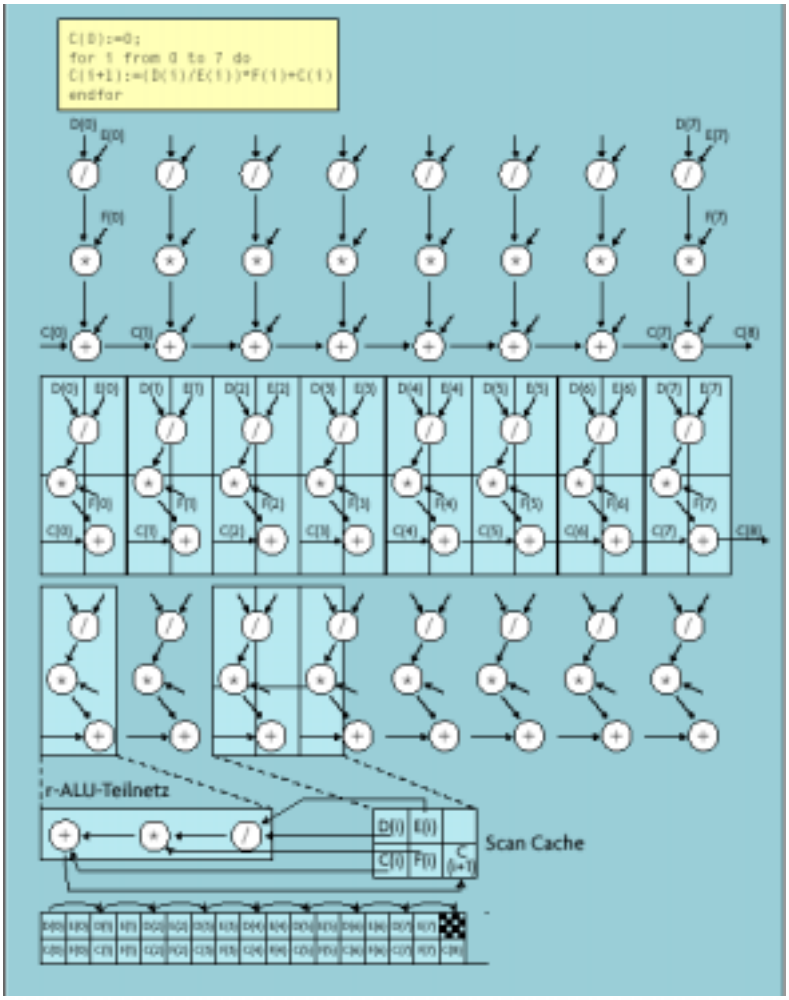


Bild 4. Zusammenspiel aller Xputer-Komponenten: Der Compiler analysiert einen vorgegebenen Algorithmus, in (a) als Textprozedur und in (b) als Datenabhängigkeitsgraph dargestellt. Er generiert einen Belegungsplan des Speichers (c) und spezifiziert Scan-Cache-Format und rALU-Teilnetze (d) sowie das Scan Pattern (e).

Am Ende des Scan Pattern steht eine Markette, ein sogenanntes Tagged Control Word (TCW). Es ist als einzige Anleihe an Neumann unverzichtbar. Als „klassische“ Kontrollmechanismen sind nur bedingte Verzweigungen nötig, denn der Xpiler kennt noch andere unkonventionelle Entscheidungsmechanismen – keine Befehle, sondern Verbundoperatoren, Scan Caches oder Scan Patterns. Da TCWs nur selten vorkommen, überprüft die Datenkomponente den Kontrollfluss bei weitem. Der große Vorteil gegenüber der von Neumann-Architektur. Der Compiler muss nur wenig Speicheradressen berechnen.

Die Software

Wie sind nun Xputer zu programmieren? Auch darüber hat sich die Gruppe rund um Hartenstein Gedanken gemacht und eine eigene Programmiersprache für den Xputer und seinen Host entwickelt. Sein Name: q-X-C – q wie Quelle, X wie Xputer und C wie ANSI-C. Das X heißt auch Map oriented Programming Language (MoPL) und ist ein Supplement von ANSI-C. Es spiegelt die Datenfluss- getriebene Hardware wider und setzt eine detaillierte Kenntnis des Xputer-Paradigmas voraus.

Vor dem eigentlichen Übersetzungsschritt (Bild 5) trennt ein X-C-Filter und Optimierer den Code für Host und Xputer.

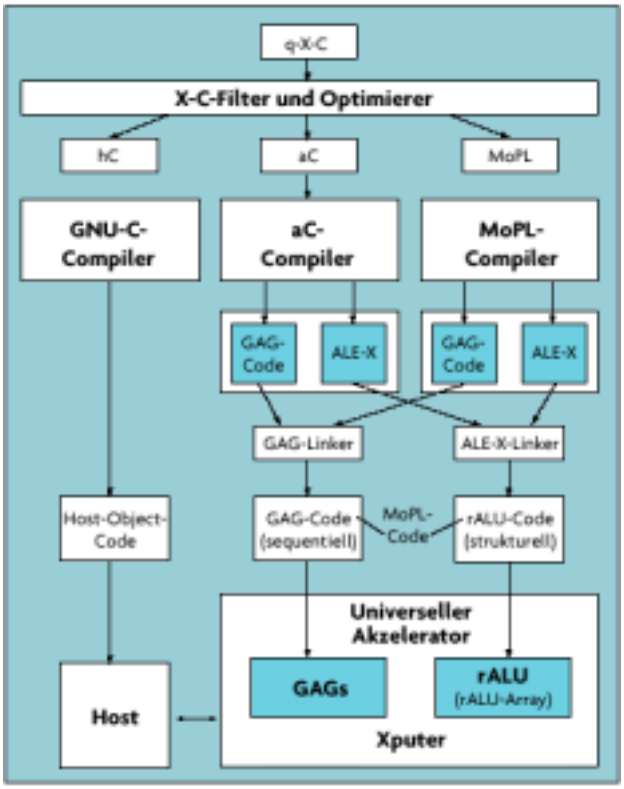


Bild 5. Co-Compiler von Host und Xputer: Die verglichen mit Computern stark verkürzte Laufzeit macht den etwas aufwendigeren Compiler-Gang bei weitem wett.

Diejenigen C-Konstrukte, welche der Xputer nicht beschleunigen kann, sind in einer Untermenge der Sprache C, Host C (hC), ausgewiesen, alle anderen in Accelerator C (aC). Reinen MoPL-Code kann nur der Xputer ausführen. Der X-C-Filter und Optimierer ergänzt den hC-Code um Synchronisations- und Kommunikations-Routinen, die dann ein GNU-C-Compiler in Host Code übersetzt. Den aC-Code strukturiert und partioniert ein sogenannter

aC-Compiler, bevor er ihn in Generic Address Generator (GAG) Code und Arithmetic and Logic Expressions for Xputer (ALE-X) übersetzt. Somit realisiert der aC-Compiler den Übergang vom Kontrollfluss- zum Datenfluss-Paradigma. Den MoPL-Code übersetzt ein MoPL-Compiler in GAG- und ALE-X-Code. Ein GAG- sowie ein ALE-X Linker führen sequentielle und strukturelle Codes in der richtigen Reihenfolge zusammen.
Listing 1 beschreibt einen Weichzeichner-Algorithmus in q-X-C. Der X-C-Filter und Optimierer zerlegt das Programm in einen aC- und einen hC-Code (Listing 2 und 3). Aus dem aC-Code erzeugt der aC-Compiler sowohl den GAG-Code als auch den ALE-X-Code.

Listing 1. Der Weichzeichner als q-X-C-Programm

```
/* Dateiname: Weichzeichner.c
Dieses Programm filtert Rauschen aus einem
Bild aus. Es liest das Bild ein und gibt es
geglättet wieder aus.                    */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include „gif-gray-load.h"
#include „gifsave.h"
static unsigned char global_color_map_out
➥ = 0;
static unsigned char print_image = 0;
int image[640][480];
/*---------------------------------*/
void FIR()
{
  int x,y;
  int i,j;
  int sum;
  sum = 0;
  for(x=0; x<640-2; x++)
  { for(y=0; y<480-2; y++)
    {
      for(  i=0; i<2; i++)
       for( j=0; j<2; j++)
        { sum += image[x+i][y+j]; }
          image[x][y] = sum / 3;
    }
  }
}
/*---------------------------------*/
int main(int argc, char * argv[])
{
  FILE * fp;
  int i, t;
  if((fp = fopen(argv[1],"r")) == NULL)
  {
   printf(„# error #  cannot open %s\n",
➥ argv[1]);
   exit(1);
  }
  GIFRead_gray(argv[1], image);
  FIR();
  GIF_Create(„Weichzeichner.gif",640,480,
➥ 2,1);
  GIF_Close();
  return 1;
}
```

Listing 2.aC-Code des Weichzeichners

```
int image[640][480];
int x,y;
main()
{ for(x=0; x<640-2; x = x + 1)
  { for(y=0; y<480-2; y = y + 1)
    {
      image[x][y] = ( image[x+0][y+0] +
                      image[x+0][y+1] +
                      image[x+1][y+0] +
                      image[x+1][y+1] ) / 3;
    }
  }
}
```

Listing 3. hC-Code des Weichzeichners

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include „gif-gray-load.h"
#include „gifsave.h"
static unsigned char global_color_map_out
➥ = 0;
static unsigned char print_image = 0;
int image[640][480];
int main(int argc, char * argv[])
{
  FILE * fp;
  int i, t;
  if((fp = fopen(argv[1],"r")) == NULL)
  {
   printf(„# error #  cannot open %s\n",
argv[1]);
   exit(1);
  }
  GIFRead_gray(argv[1], image);
  FIR();
  GIF_Create(„Weichzeichner.gif",640,480,
➥ 2,1);
  GIF_Close();
  return 1;
}
```

Die Realisierung

Am Department of Electrical and Electronic Engineering des Imperial College in London hat eine Gruppe um Peter Cheung zusammen mit den Hewlett Packard Labs und der Firma Xilinx ein PCI-Bus-Board entwickelt, das dynamisch rekonfigurierbare FPGAs enthält.
Den ersten Xputer, die MoM-1, schuf Hartensteins Gruppe in den 80er Jahren. Ihr Name steht für Map oriented Machine, first generation. „Map“ deutet die zweidimensionale Speicher-organisation an. Neben dem ersten Data Sequencer entwickelten die Forscher ein Dynamically Programmable Logic Array (DPLA), dessen Leistungsdaten damals die komerzieller FPGAs übertroffen haben.
Die Mom-1 war überragend schnell. Bei einem Design Rule Check mit einem der Bildverarbeitung entlehnten Pattern-Matching-Verfahren erzielte sie bei 800 Referenzmustern ein Speed Up von 2300 gegenüber einer VAX 11/750. Mit anderen Worten: Sie war mehr als 2000mal schneller als ein für damalige Verhältnisse rasanter Super-Minicomputer.
Um die Ergebnisse zu erklären, verglichen die Forscher die Operations-Mechanismen von Computern und Xputern. Dabei stellten sie fest, dass Computer besonders viel Zeit zum Adressieren des Speichers benötigen: bis zu 90 Prozent der Gesamtrechenzeit nämlich. Xputer hingegen sind darin recht sparsam und müssen nur wenige Adressen explizit berechnen.
Zur weiteren Beschleunigung trägt die Mikro-Parallelität der rALUs bei: die Software-zu-Hardware- oder besser die Software-zu-„Configware“-Migration. Außerdem findet eine Laufzeit-zu-Compile-Zeit-Migration statt, sprich: Compilieren dauert bei Xputern länger als bei Computern; dafür sind Xputer in der Ausführung schneller.
Im Gegensatz zu MoM-1 war der zweite in Kaiserslautern realisierte Xputer skalierbar, die MoM-2: Bis zu 15 rALUs konnten parallel betrieben werden. Allerdings erschien den Forschern die Architektur nicht einfach und nicht elegant genug. Eine rALU aus kommerziellen Bausteinen beanspruchte einerseits viel Platz und war anderseits sehr teuer. Deshalb entwickelte Reiner Kress, ein ehemaliger Doktorand von Hartenstein, die MoM-3 mit dem nach ihm benannten, skalierbaren rALU-Array. Der Pfiff dabei: Das dazugehörige Compiler-Backend, das sogenannte Data Path Synthesis System (DPSS), macht alle Routing- und Placement-Algorithmen obsolet, an denen sich schon Heerscharen von Forschern die Zähne ausgebissen haben.

Das Kress Array

Bei rekonfigurierbaren Rechnern wird häufig das Kress Array als „ALU“ eingesetzt.
Es ist ein allgemeines Konzept zur paralellen Berechnung und Manipulation von Daten, ist zur laufzeit rekonfigurbar und besteht im Allgemeinen aus 9 einzelnen rALU´s (rekonfigurierbare ALU´s) , die auch als rDPU´s (rekonfigurierbare Data Path Units) oder PE´s (processing elements) bzeichnet werden. Diese unterschiedlichen Begriffe spiegeln die verschiedenen Funktionen wieder, die sie zeitgleich durchführen können.

Bild 1 zeigt einen Überblick über das Kress Array

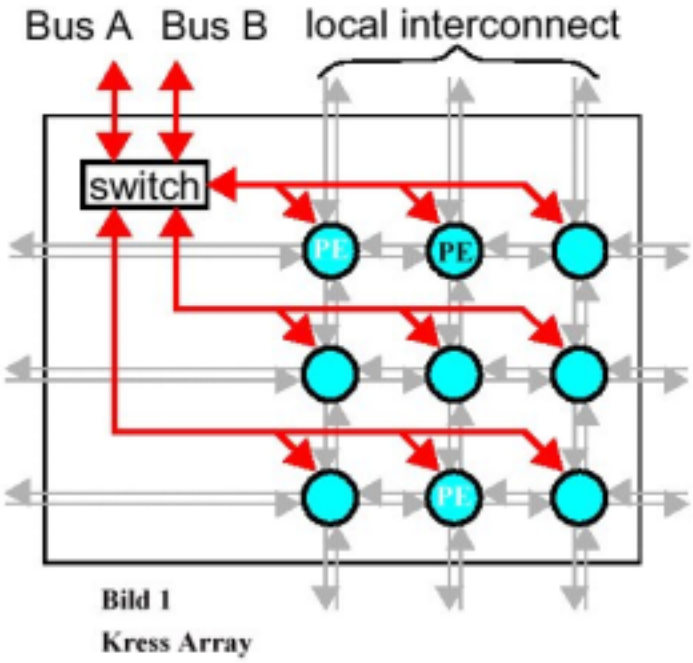


Bild 1
Kress Array

Das Kress Array hat zwei unabhängige 32-Bit Datenbusse (Bus A & Bus B) und „local interconnects“ im NEWS-Netzwerk (North East West South) , d.h. jede PE wird mit den 4 benachbarten PE´s über zwei ebenfalls unabhängige 32-Bit Datenbusse verbunden. Da theoretisch beliebig viele dieser Array´s in einem Chip seien können, werden die aussen liegenden PE´s mit den zugehörenden PE´s des benachbarten Array´s verbunden. Dieses Verfahren nennt man nearest neighbour connections.

Diese Verbindungen können unabhängig voneinander Daten in ein PE zur Verarbeitung leiten, oder sie in ein benachbartes PE weiterleiten.

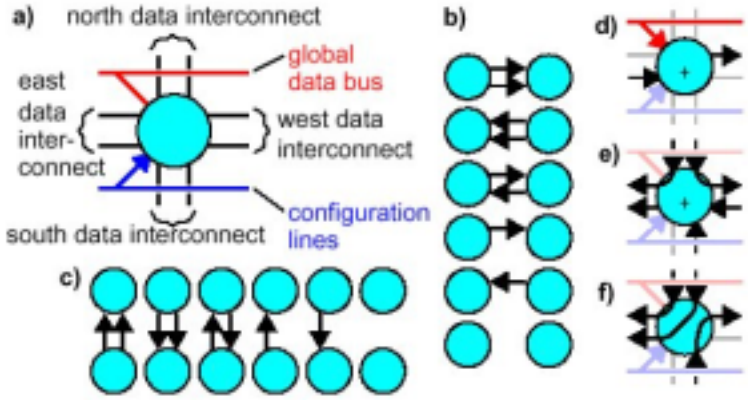


Bild 2

Bild 2a zeigt ein PE mit allen Bussen, inclusive des Konfiguartionsbusses (configuration line), über den die arithmetisch / logischen Befehle und die Konfiguration der local interconnections in die PE übertragen werden. Dieser Bus ist write only und arbeitet unabhängig von allen anderen.
Bild 2b zeigt alle east-west Verbindungen.
Bild 2c zeigt alle north-south Verbindugen.
Bild 2d zeigt das PE (rDPU) mit arithmetischer Funktion.
Bild 2e zeigt das PE (rDPU) mit arithmetischer Funktion mit Datendurchleitung.
Bild 2f zeigt die rDPU als reinen Datenrouter.

Bild 3

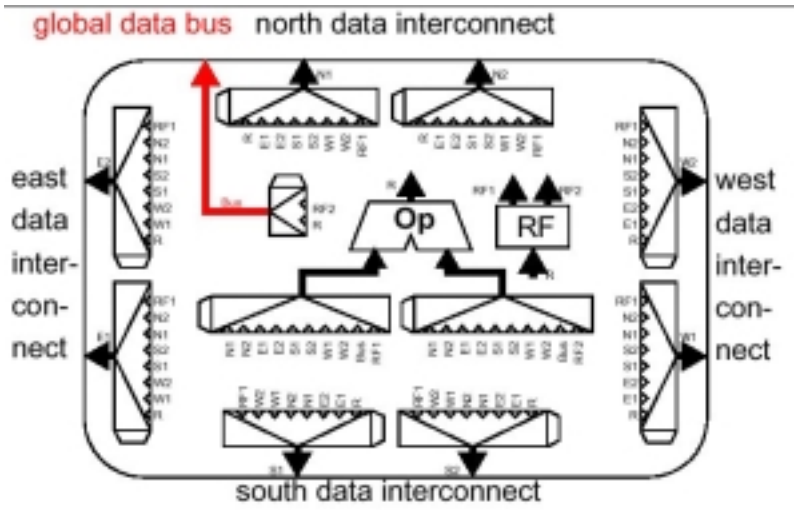


Bild 3 zeigt den internen Aufbau einer rDPU.

Sie besteht aus der ALU (Op), dem register file (rf), dem globalen Datenbusanschluß und den local interconnects.Desweiteren gehört der configuration memory mit angeschlossener configuration line dazu, welche in diesem Bild fehlen (siehe Bild 2a) .

Jeder Eingang der rDPU kann zu jedem Ausgang, der ALU oder dem register file durchgeschaltet werden.
In dem register file (rf) werden die zu verarbeitenden Daten, eventuelle Konstanten und Zwischenergebnisse gepeichert. Letzteres entspricht einem Cache mit einer Hit–Rate von 100 %. Das register file wird auch als smart interface bezeichnet.

Die ALU kennt alle logischen und arithmetischen Operationen der Sprache C.

Durch die Aufteilung einer komplexen Berechnung auf mehrere rDPU´s wird ein tiefes pipelining bzw. eine paralelle Verarbeitung erreicht.

Configuration Memory

Im configuration memory werden komplette Konfigurationssets für Operationen der ALU und den Datendurchleitungsinformationen gespeichert. Er ist in vier layern organisiert, von denen jeder layer ein vollständiges Set enthält. Die layer werden über die configuration line von einem host-computer programmiert. Verschiedene Funktionalität erreicht die rDPU durch den Wechsel der Layer, die auch zur Laufzeit umprogrammiert werden können. Dies geschieht ohne Zeitverlust, da immer nur layer umprogrammiert werden, die nicht aktiv sind.
Da immer nur ein layer aktiv ist, können quasi immer drei layer umprogrammiert werden.
Die configuration line ermöglicht kein auslesen der layer (des configuration memories), da er write only ist. Die Programmierung erfolgt in zwei Schritten : erst wird die Adresse der rDPU auf die configuration line gelegt, danach die Daten für einen layer übertragen.

Der Datensequenzer

Der Datensequenzer ist notwendig, da die Steuerung des „Programmablaufs“ nicht durch den Befehlsstrom sondern durch den Datenstrom geschieht.
Er übernimmt die Speicherverwaltung und die Berechnung der physikalischen Speicheradressen.
Die einzelnen Datenspeichermodule sind parallel ansprechbar, es kann so auf alle Module gleichzeitig Zugriff erfolgen. Die Daten werden vom Datenspeicher in die smart interfaces der rDPU´s kopiert, und nach der Berechnung wieder zurückgeschrieben.
Der Datenspeicher ist theoretisch 2-dimensional. Ein Bild wird zum Beispiel mit seinen x-und y Koordinaten direkt im Speicher abgelegt. Pysikalisch wird jedoch zumeist herkömmlicher 1-dimensionaler Speicher eingesetzt, obwohl es von Siemens für diesen Zweck mittlerweile auch 2-dimensionalen Speicher gibt. Die Umsetzung zwischen 2-dim. theoretischem und 1-dim. physikalischem Speicher erfolgt im Datensequenzer.

Das Scan Window

Der Zufgriff auf die Daten im 2-dim. Speicher erfolgt über Scan Windows. Die Grösse wird vom Compiler bestimmt und bleibt demnach konstant. Ein Scan Window enthält die Daten, die eine rDPU zur Verarbeitung bekommt. Die Daten werden in das Smart Interface kopiert und nach der Verarbeitung wieder zurückgeschrieben.
Das Scan Window markiert die zugehörenden Speicherplätze mit read, write oder read/write, da durch die prinzipiell mögliche Paralellverarbeitung ein Scan Window eines anderen Prozesses die gleichen Daten zur gleichen Zeit verändern könnte, was dann zu Daten-Hazards führt. Es darf immer nur ein Prozess auf eine Speicherstelle zugreifen.
Die linke untere Ecke des Scan Windows definiert die Position und wird als handle bezeichnet.

Die Algorithmen, die auf die Daten in einem Scan Window angewendet wurden sollen üblicherweise auf die gesamten Daten (ganzes Bild) angewendet werden. Hierzu verschiebt man nach der Bearbeitung der Daten im Scan Window selbiges nach einem vom Compiler bestimmten Muster über den Datensatz, den Scan Pattern.

Die Scan Pattern

Die Scan Pattern beschreiben durch wenige Paramater die Bewegung der Scan Windows über die Daten.

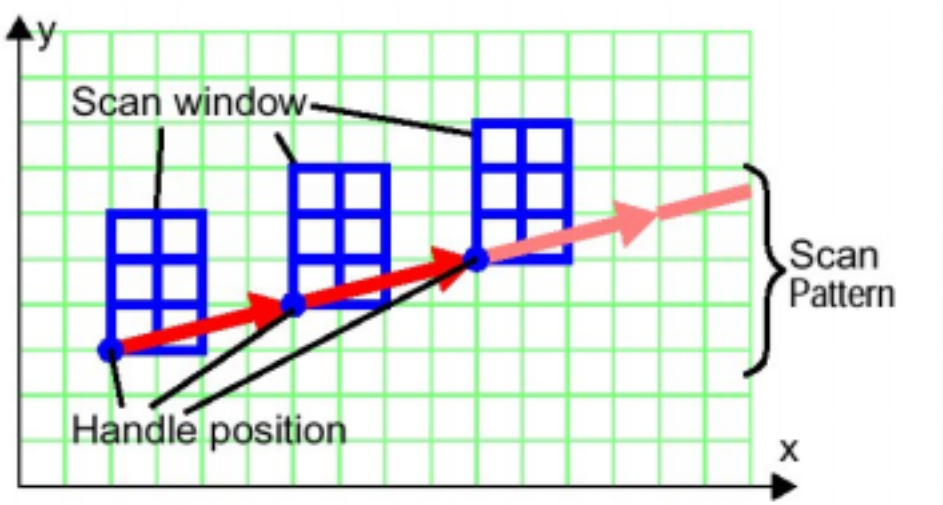


Figure 6. 2-dimensional memory organization

Folgende Speicherzugriffsoptimierungen sollen für Zeitersparnis sorgen :

1. Scan Windows : es werden grössere Datenmengen in einer Aktion kopiert
2. zeitgleicher Zugriff auf mehrere Speichermodule
3. burst modes beim Transport der Scan Windows in die Smart Interfaces

Die Entwicklung des XC6200 FPGA von Xilinx

Mit dem XC6200 FPGA Xilinx wird der erste kommerzielle verfügbare FPGA entwickelt für rekonfigurierbare Computer. Es hat einen komplett neue interne Architektur, neue entwickelte Algorithmen und die neu benötigte Software. Entsprechend der Tatsache das die meisten Anwendungen im Forschungsbereich liegen, sind die Anzahl verkaufter Einheiten gering. Aber der Fortschritt der Tool Entwicklung ist auch für diese Architektur ziemlich gering.

1 Einleitung

Der XC6200 ist ein FPGA der für zwei Klassen von Anwendungen entwickelt wurde. Die erste Klasse ist die konventionelle des ASIC Bauteil für Logik Integration. Die andere Rolle ist die einer Intelliegenten Peripherie die in einem Mikroprozessor wie ein Memory-mapped-coprozessor operiert. Dieser FPGA ist entsprechend groß fortgeschritten zur CPU Schnitt-stelle.
Die Entwicklung für den XC6200 ist ähnlich wie für andere FPGA Familien, nur eben mit ein paar Einschränkungen Anwendbar. Die Entwicklung ist trotzdem in einem Beta Stadium und schreitet langsam voran, die Kommerzielle Benutzung ist gering. Nicht alle Eigenschaften sind bisher Implementiert und manche besitzen noch Bugs. Weiter ist es schwer irrelevante Strukturen ausfindig zu machen, nur wenige ´´routing resources´´ führen zur Ziel Architektur. Der Entwicklungsfluss (design flow) startet meist ab der Gatter Level, in der direkt die Logik Blocks von den FPGA programmiert werden (z.B.: Lola). Ein paar Schritte erscheinen ähnlich zu anderen FPGA Technologien, aber entsprechend zum XC6200 spezieller Probleme zeigt jeder Schritt Differenzen zu anderen Technologien.
Die Entwicklung besteht aus fünf Schritten:

- Entwicklungs- Werk (Design Creation)
- Logische Synthese (Logic Synthesis)
- Netzlisten Erstellung (Netlist Generation)
- Platz und Verknüpfung (Place and Route)
- Simulierte Entwicklung mit Zeitinformationen (Simulate Design with Timing Information)

Nach dem Entwurf der Eingeschränkt verfügbaren Entwicklungssoftware wird der ergebene Entwicklungsfluss dargestellt, der hauptsächlich auf definierte Makro Zellen basiert. Der Vorteil wird demonstriert an einem 3x3 linearen Filter mit konfigurierbaren Gewichten für Abbildungs- Verarbeitungsgeschwindigkeiten. Dieser Anwendungsgewinn der Prozessor Schnittstelle ermöglicht den Filter Koeffizienten zu wechseln ohne das komplette Bauteil neu zu Konfigurieren.

2 Übersicht über den Xilinx XC6k und den VCC HOT Works Board

Architektur Übersicht über die XC6200 Serien

Der XC6200 basiert auf eine dünne granulierten, „See- von Gattern“ - Architektur. Der XC6200 besteht aus einen Array von 64x64 Kern Zellen mit 256 Eingabe/Ausgabe Ports. Jede logische Zelle kann aus zwei Eingängen viele logische Kombinationen für logische Funktionen Implementieren. Jede Zelle kann also ein D-Type Flip-Flop implementieren, welches benutzt werden kann um die Kombinationsfunktionen zu registrieren. Zellen sind

mit ihren Nähesten Nachbarn im Norden, Westen, Osten und Süden verbunden. Das Bauteil hat eine Hierarchie des Bus Schemas. Zellen sind in Blöcken mit 4x4, 16x16 usw. organisiert, zunehmend um den Faktor vier. Ein Satz vom schnellen Bus ist mit jeder Größe von Blöcken verbunden. Zugang zu diesem schnellem Bus ist mittels ´´routing´´ Schaltbar.
Die Prozessor Schnittstelle ist ein 32 Bit breiter Datenbus der für 16 oder 8 Bit Operationen Konfiguriert werden könnte. Der XC6200 wurde Entwickelt um im System wie ein Random Access Memory zu erscheinen. Alle Daten Register auf dem Array sind zugänglich. Die Herstellung von Schnittstellen ist mit Benutzer Logik allein mittels der Prozessor Schnittstelle möglich. Register werden in Spalten mittels einem map Register adressiert. Bis zu 32 von den 64 Registern in Spalten könnte gelesen werden von oder geschrieben zu beim ernennen ihrer passenden Reihenfolge in dem Map Register. Diese kann sich soweit ausbreiten, dass alle der 64 Register in Spalten von einer einfach 8 Bit Operation geschrieben werden könnte. Dies ist im Detail nützlich für Rekonfigurierbare Berechnungsanwendungen.

Das HOT Works PCI-XC6200 Entwicklungs- System

Der vorgeschlagene design- flow und die Implementierten Beispiele sind auf dem HOT Works PCI Board von Virtual Computer Corporation getestet wurden. Die Bord Architektur ermöglicht dem XC6200 Zugang zu einem Host CPU mittels dem PCI- Bus.
Das Bord besteht aus :

- einem XC6216 für die benutzter Entwicklung
- einem XC4013 Implementiert die PCI- Bus Schnittstelle
- bis zu 2 Mbytes von schnellem SRAM für benutzter Daten
- einem programmierbaren Zeit- Generator.

Der XC6216, der SRAM und der Satz von Konfigurierbaren Registern sind gemappt vom dem Speicherplatz der Host CPU. Dies ermöglicht der Host CPU den SRAM Speicher auf dem Bord zu lesen oder schreiben und konfiguriert, liest, oder schreibt den benutzter FPGA und die benutzter Entwicklung.

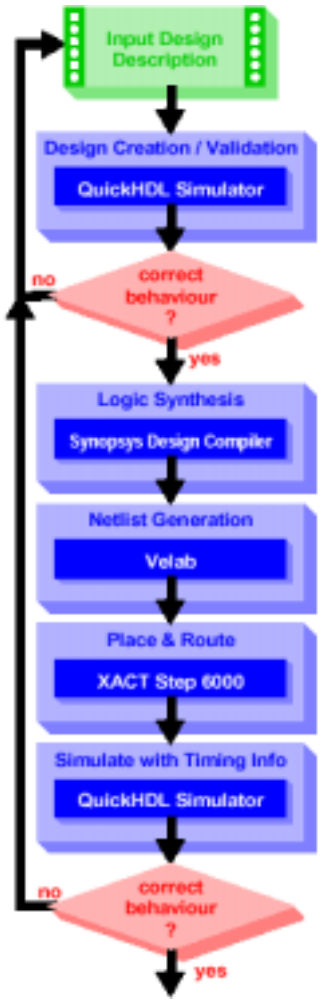
3 Entwicklungsfluss für den Xilinx XC6200

Die Entwicklungsumgebung besteht aus fünf teilen. Erstens, einem VHDL- Simulator der die VHDL Entwicklungsbeschreibung für gültig erklärt. Dann muss VHDL in primitive Gatter der Ziel Technologie umgesetzt werden, dafür wird ein Umsetzungs- Tool mit entsprechender Technologischen Bibliothek gebraucht. Das Ausgabe Format des Umsetzungs- Tool wird VHDL sein. Zum Umformen der VHDL Netzliste in das EDIF- Eingabe-Format wird ein kleines Programm benötigt, ein place-and-route Tool. Die place-and-route Software ist das letzte Tool das in der Entwicklungsumgebung erwähnt wird.

Technologische Bibliotheken

Grundlagenteil des Entwicklungsprozesses ist eine technologische Bibliothek. Die einfachen der Bibliotheken haben irgendeine zwei Eingabe Gatter Funktion, irgendein 2:1 Multiplexer, konstant 0 oder 1, Puffer, Inverter und D-Type Register. Technologische Bibliotheken für die XC6200Familie existieren für den Viewlogic schematic entry Tool und dem Synopsys Design Compiler. In dieser Entwicklungsumgebung werden solche Umsetzungen angewendet.
Der Synopsys Libary Compiler erzeugt eine struktruelle Beschreibung der Ziel Technologie (VITAL). Die VITAL Bibliothek bietet alle einfachen Gatter mit Standard Zeiten an. Die Standard Zeiten können von genaueren Zeit-Werten überschrieben werden, die von der Place-

and-Route Software berechnet wurden (SDF = Standard Delay Format). Die Synopsys Bibliothek ist nur in der SunOS Version vom XACT Step 6000 beinhaltet.



Simulierung der Entwicklung

Wie die einfache Synthesis Bibliothek nur mit Synopsys Design Compiler benutzt werden kann, ist die VITAL Bibliothek Anbieter unabhängig. Die VITAL Bibliothek kann mit irgendeinem VHDL Simulator simulieren, solche wie Synopsys'VSS, Model Technology's V-System oder Mentor Graphics's QuickHDL. V-System und QuickHDL haben zwei Vorteile im Vergleich zu VSS. Erstens beide sehen eine Schnittstelle für ihre Simulatoren vor, diese Eigentschaft kann benutzt werden für die zusammengefasste Simulation von Hardware und Software. Zweitens Unterstützen beide den VHDL Standard'93, der wichtig für die Verschlüsselungs-technik ist. Es wurde QuickHDL für die Simulation ausgesucht.

Logische Synthese

Der Synopsys Design Compiler kann nicht pad- Zellen synthesieren, die für Anwender IOBs (Synopsys creates only input/output-buffer) gebraucht werden. Dafür besorgt sich der Entwickler eine top-level Struktur Beschreibung für die Eingabe/Ausgabe selber. Manche Entwicklerinformationen, solche wie der Pinout oder der Anordnung kann nur von Anwender definierten Attributen abhängen, die nicht von Synopsys Design Compiler unterstützt werden. Ein anderer großer Nachteil der Logik Synthese ist das Problem der Xilinx Configure FPGA Place-and-Route Software XACT Step 6000. Anordnung und Verknüpfung von längeren nicht Hierarchien Entwicklungen ohne Anordnungsinformationen ergeben ein schlechtes Design. Hierarchisch Entwicklungen mit Makro Zellen für Addierer, Subtraierer und anderen regelmäßigen Strukturen sind wichtig für manuelle floorplanning. Die Konsequenz daraus: Nur reine state-machines ohne Datenpfade (<100 einfache Gatter) sollten mit Synopsys synthesiert werden. Synthese von RTL-VHDL von größeren Entwicklungen (> 1000 einfacher Gatter) wie für andere FPGA Technologien ist momentan nicht machbar.

Netzlisten Erzeugung

Für den Datenpfad ist eine gute Hierarchie der Entwicklungsstruktur wichtig. Übliche Zellen-strukturen, solche wie Addierer und Multiplizierer, sollten Benutzerdefinierte Eigenschaften ersetzen und in größeren Zellen wie Pipelines instanzirren. Das Ergebnis des Entwicklungs-stils sind hierarchische VHDL- Netzlisten von instanzirren Komponenten und Technologisch einfach. Das eingebende Netzlistenformat vom XACTStep 6000 ist EDIF. Ein kleines Tool namens VELAB wird zum umformen von VHDL- Netzlisten in EDIF benutzt. VELAB ist ein freier VHDL Analysator und EDIF Netzlisten Erzeuger für die XC6200 Familie herausgebracht von Xilinx.

Placement and routing

Die XACTStep Series 6000 ist ein grafisches Tool für die Entwicklung der XC6200 Familie. Dieses System ist ein back-end Tool mit primärer Eingabe wie EDIF. Der XACTStep Series 6000 Editor bewahrt die Hierarchie des eingegebenen Designs. Die Hierarchie Informationen werden zur top down Entwicklung durch floorplanning und der bottom up Entwicklung benutzt, die entweder manuell oder automatisch Technologien Unterstützt. Volle Automatisches place-and-route wird Unterstützt. Der Grafische Editor hat vollen Zugriff auf alle Ressourcen der XC6200 Familien Architektur.
In der Praxis benötigen die automatischen Techniken ein bisschen Unterstützung. Gerade wenn ersetzende floorplanning Strukturen benutzt werden, sind manuelle Anordnungen unvermeidlich. Der automatische Router übergeht oft in der top-level Routing Entwicklung. Deswegen sind Placement und Routing kein automatischer Entwicklungsschritt wie für die meisten kommerziellen FPGA Technologien. Ein bisschen manuelle arbeit und Entwicklungsexperimente sind wesentlich für annehmbare Ergebnisse. Zeiteinstellungen können nicht gesetzt werden, Zeit Analysen für post-layout Entwicklung hat Interaktiv zu sein zum wählen der Quellen und der Ziel Zellen im grafischem Editor.

Entwicklungsschritt	Entwicklungs Tool	Version
Entwicklungsgültigkeitsüber-prüfung(1), Simulation (5)	Mentor Graphics QuickHDL	V8.5-4.6c
Logische Synthese (2)	Synopsys Design Compiler	1997.08
Netzlisten Erzeugung (3)	Xilinx Velab (freeware)	0.52
Place & Route (4)	Xilinx XACT Step 6000	1.1 beta build 4

Quellenverzeichnis :

Zeitung ct Ausgabe 12 Jahr 1998

Xputer Lab Universität Kaiserslauern
PDF – Dateien :
Paper82.pdf bis paper102.pdf

http://xputers.informatik.uni-kl.de/