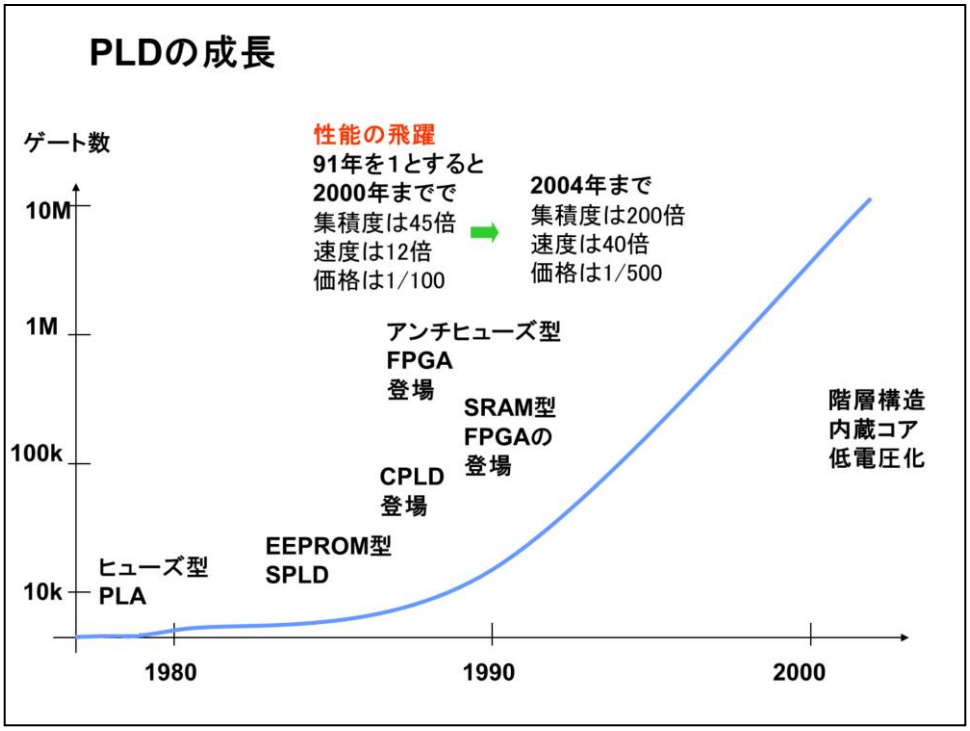
FPGA概論

マイクロプロセッサ特論　9回

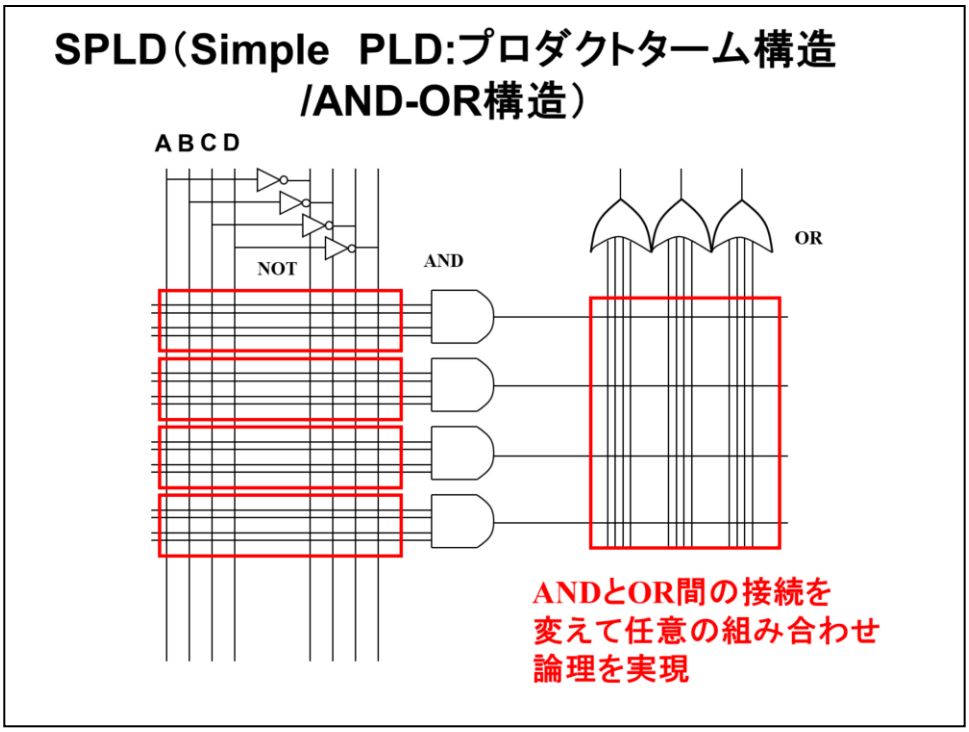
慶應義塾大学
天野英晴

PLD（Programmable Logic Device)とは、ユーザが論理機能を決めることのできるICのことです。メモリやCPU、ASIC、昔の74シリーズのような標準ディジタルICはその機能が決まっていて、これらはプログラマブルデバイスとは言いません。CPUはソフトウェアで動作を変えられるので究極のプログラマブルデバイスだ、という人も居ますが、一般的には専用目的ICに分類されます。PLDには小規模なAND-OR構造でできたSPLD(Simple PLD）とこれを拡張したCPLD（Complex PLD）、LUT(LookUpTable）を用いた大規模なFPGA（Field Programmable Gate Array)に分類されます。SPLDはPLAとかPAL（これは製品名）とも呼ばれますし、用語は統一されておらず混乱しています。ただし、最近は大規模なものはほとんどFPGAになっていて、これだけ覚えておけば、まず問題ないかもしれません。



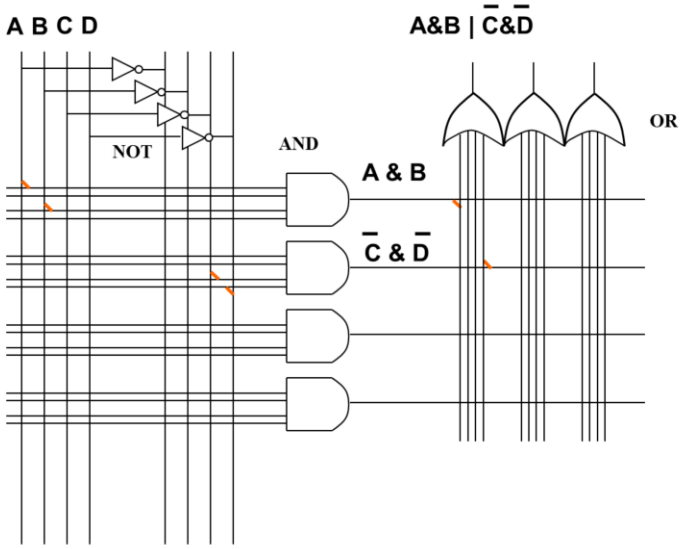
PLDの歴史は案外古く、70年代に用いられたバイナリジャンクショントランジスタを用いたヒューズ型のSPLDに遡ります。この型のSPLDは、単純なAND-OR構造の結線を外部からヒューズを切断することによってプログラムしました。小規模で再プログラムができなかったが、高速で、当時のディジタルICの主流であったTTL (Transistor Transistor Logic) の74シリーズで実現が困難な特殊な論理回路に用いられました。80年代になってCMOSを用いた再プログラム可能な素子が登場しました。Lattice社のGALシリーズがこの代表で、AND-OR構造にフリップフロップを含んだ出力ブロックを接続することにより、やや複雑な組み合わせ回路、順序回路が実現可能でした。このシリーズは再プログラミング可能であり、簡単なハードウェア記述言語からプログラミングを行う環境も整い、広く利用されるようになりました。

80年代の終わりに大規模なCPLD, FPGAが登場し、簡単なディジタルシステム全体がPLD上に実装可能となり、PLDは急成長時代に突入しました。アンチヒューズ型、EEPROM型、SRAM型等さまざまな特徴を持った方式が普及し、2004年現在に至るまで、集積度、速度は凄まじい勢いで進歩し、価格は急速に低下した。1991年を1とすると、2000年までの9年間で集積度は45倍、速度は12倍、価格は1/100となっている。さらに2004年には集積度は200倍、速度は40倍、価格は1/500となっています。最近は、内部構造階層化が進むと共に、メモリ、CPU, DSP, 演算器、高速インタフェースを内蔵し、基板に代わってPLD上にシステムを実装するSoPD (System on Programmable Device)の考え方が登場するに至りました。また、低電圧化、低消費電力化したデバイスも登場しています。この発展速度は集積度においてメモリ素子を上回っており、もっとも急速に発展し続けるデバイスと言えます。

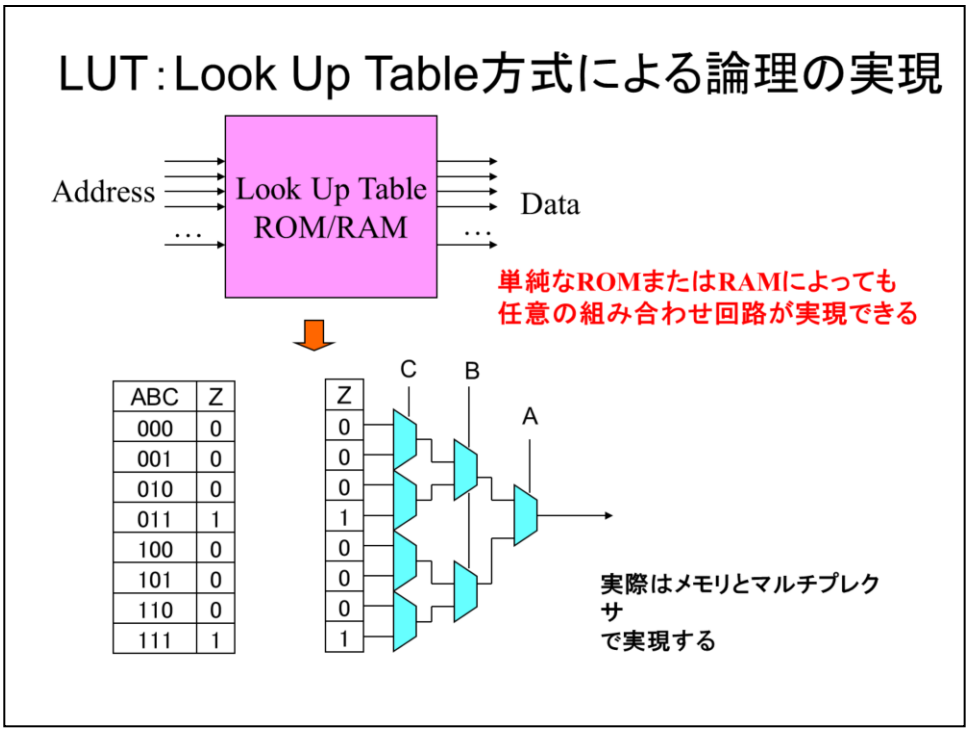


皆さんはブール代数を習ったときに、全ての論理式は加法標準形、すなわち、NOT-AND-ORの形で実現できることがわかったと思います。AND入力の選択（どの入力を繋ぐか、NOTかそのままか）と、AND－ORの結合を切ったり繋いだりすれば、任意の論理積項の組み合わせが作れ、任意の論理式を作ることができます。これがAND-ORを使ったプロダクトターム方式のSPLDの原理です。

プロダクトターム構造の作り方



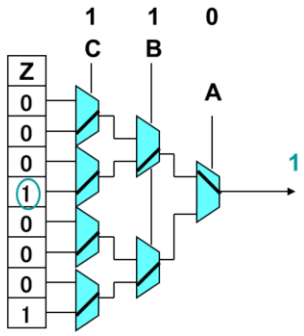
この例ではA＆BとCバー＆Dバーの論理積を実現する結線を示します。このように一つ一つの積項をANDゲートで作り、これらをORゲートに入力します。



単純なメモリは、アドレスを入力、データを出力として考えると、真理値表の代わりに使えるので、任意の組み合わせ回路を実現することができます。しかし、通常のメモリは小さい面積で大きな容量を実現するのに特化した構造を持っているので、小規模の入出力で高速性が要求される論理回路に使う場合不利が大きいです。このため、実際には記憶要素にマルチプレクサのツリーを組み合わせて表を実現します。これをLook Up Table (LUT)と呼びます。

LUT：Look Up Tableによる論理の実現例

ABC	Z
000	0
001	0
010	0
011	1
100	0
101	0
110	0
111	1



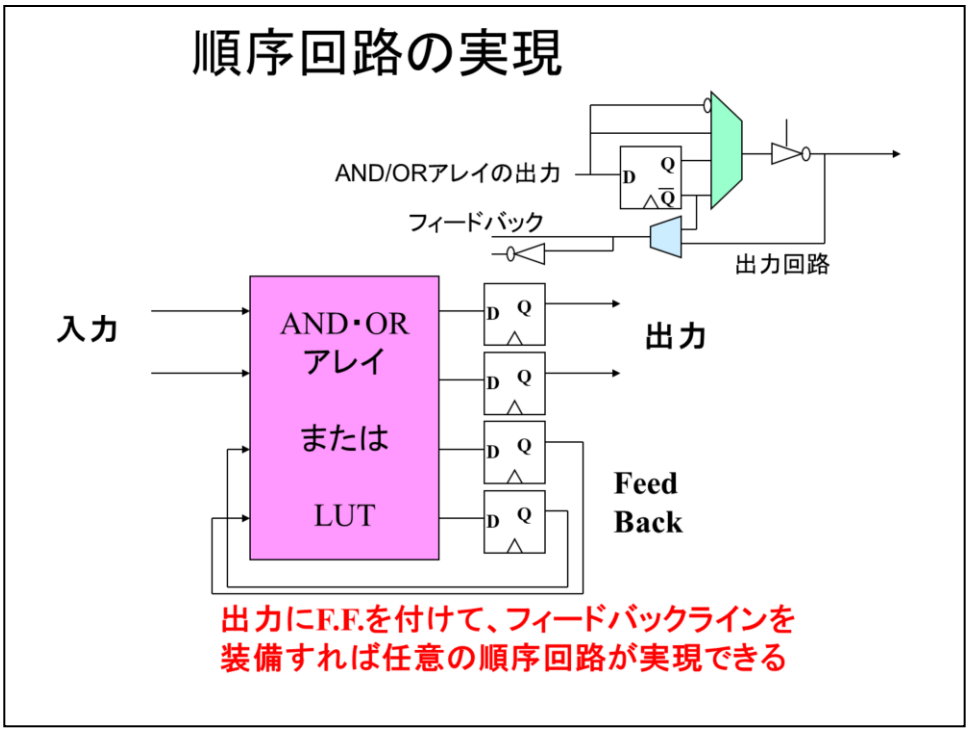
LUTによる論理の実現の例を示します。同じ列のマルチプレクサには同じ制御入力を繋ぎ、1ならば下から、0ならば上からの入力を出力に流します。ABCの順に011を入れると、上から3番目のデータが取り出せることが分かります。(表と右の図でABCが逆順な点にご注意ください。)すなわちこの回路は表として働いています。マルチプレクサは以前紹介した方法で簡単に作れるので、この方法は入力が6,7よりも小さい場合には効率的です。

プロダクトターム方式 vs. LUT

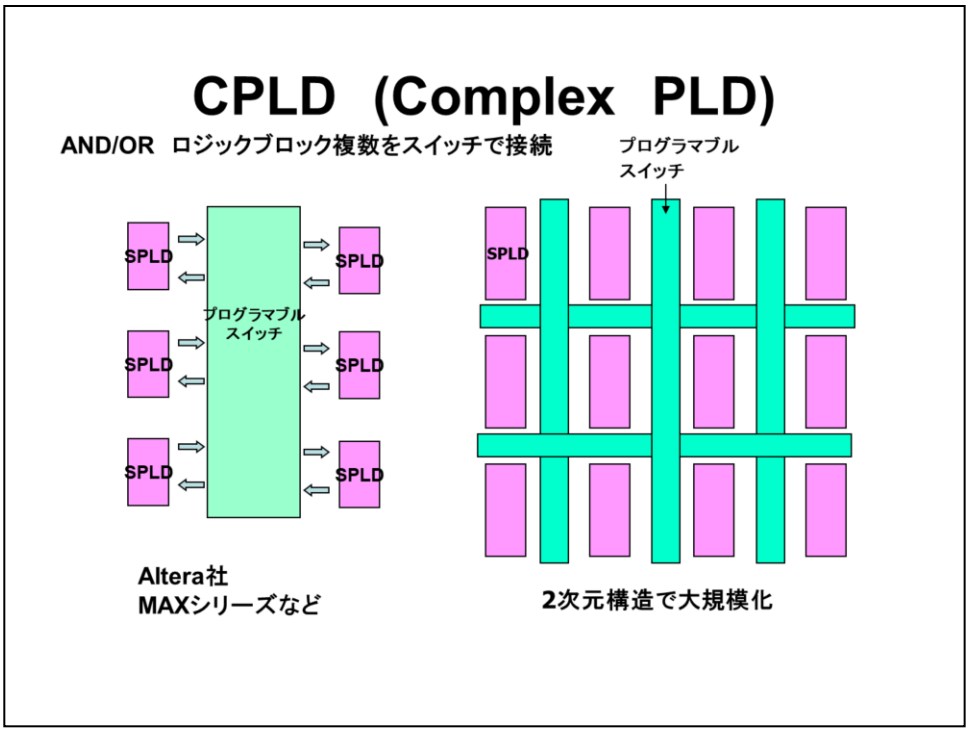
- プロダクトターム方式（AND-OR構造）
  - 多入力多出力回路が効率良く実現できる
  - 場合によっては入力項数が不足する
  - EEPROM、フラッシュROMでの実現に適している
- LUT
  - 任意の論理が実現できる
  - 出力が少なく小規模な論理に有利
  - フラッシュ、アンチヒューズ、SRAM型に適している

プロダクトターム方式は、AND出力から出てくる積項を複数のORゲートで共有することができます。すなわち、場合によっては多入力、多出力回路が効率的に実現できます。しかし、ANDゲートの数、ORゲートの入力数にかなり余裕がないと、本当に任意の論理式は実現できません。先にしめした例では、ANDゲートは4つしかないので積項が4つを越えると実現できなくなってしまいます。プロダクトターム方式は結線の交点上のスイッチで実現するので、EEPROM、フラッシュROMなどによる実現に適しています。

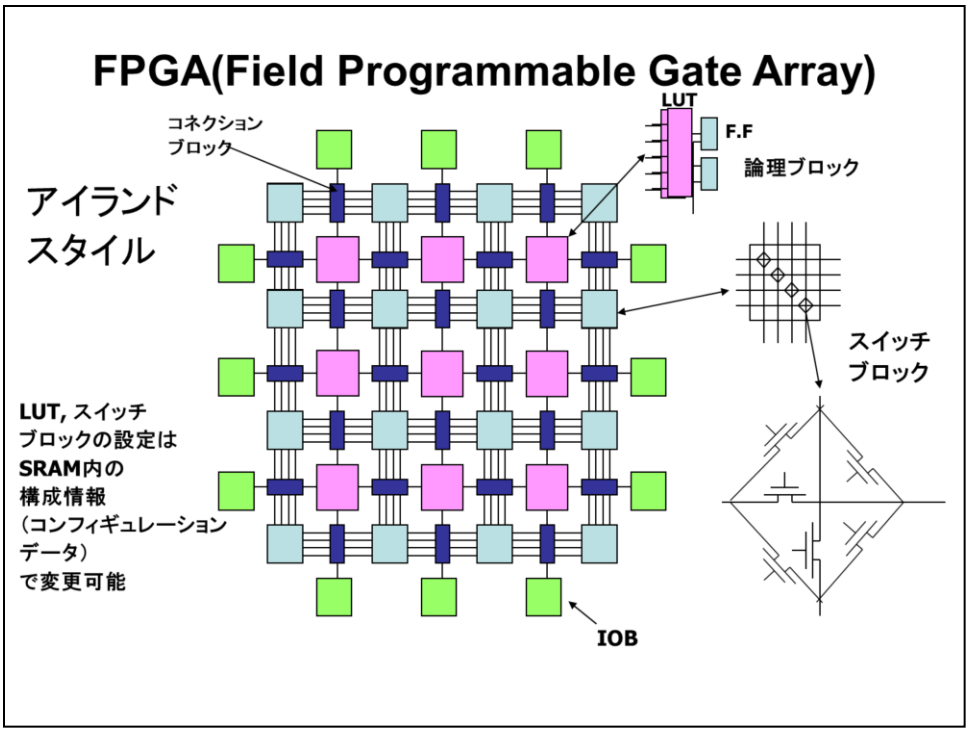
一方で、LUTは真理値表なので本当に任意の論理式が実現できます。しかし、2の入力数乗のオーダーで表のサイズが大きくなるので、入力数は通常4-6程度に限定されます。途中結果の共有はできません。LUT方式は様々な方式で実現できますが、最近はSRAM型が良く使われます。



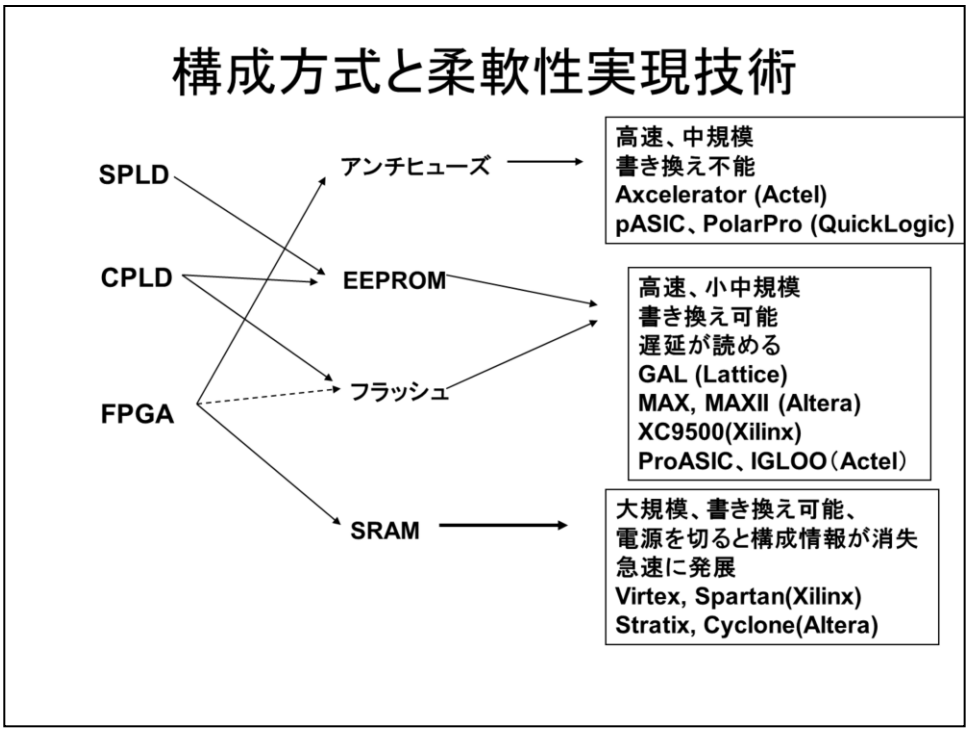
組み合わせ回路の出力にD-F.F.を付け、さらにフィードバックを付けることで順序回路が実現できます。このためには、F.F.の出力回路を選択可能にしてやる必要があります。図はLattice社のGALの出力回路の例です。出力はF.F.を介したものと介さないものが選択可能で、3ステート出力になっています。また、組み合わせ回路にフィードバックすることもできるようになっています。このような構成のPLDをSimple PLD（SPLD)と呼びます。



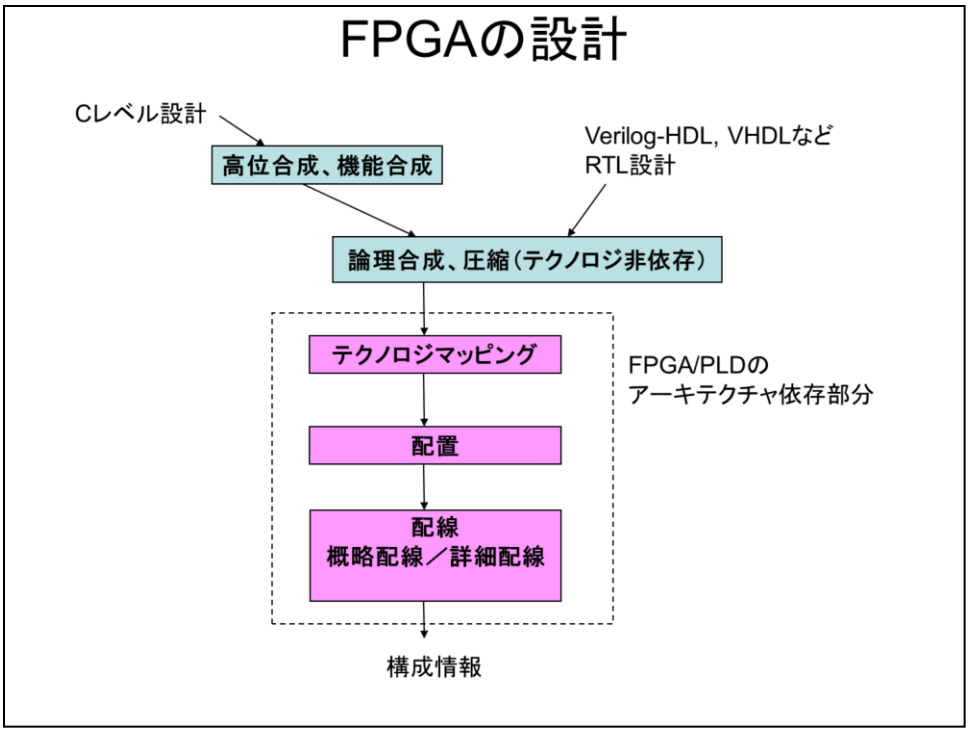
大規模なディジタル回路は、複数の組み合わせ回路、順序回路から出来ています。これに対応するため、先に示したSPLDを複数個、スイッチで接続します。さらに大規模なシステムを作るためには2次元構造のスイッチを利用します。このようなチップをComplex PLD（CPLD)と呼びます。



FPGAは今までのAND-OR構成と違って4-6入力程度のLUT2セット程度を一つの論理要素として使います。それぞれの出力にはD-FFを備えます。この論理要素の周辺に配線を敷き詰め、交点にスイッチブロックを置きます。スイッチブロックはトランジスタのON/OFFで接続をON/OFFし、論理要素間の配線を自由に行えるようにします。また、論理要素と、配線の間にも同様のコネクションブロックを置き、論理要素との入出力配線を制御します。また、チップの入出力PINとの間の配線も行います。LUTの内容、スイッチブロックのトランジスタのON/OFFを設定することにより、様々な回路構成を実現することができます。この設定情報のことを構成情報（Configuration Data)と呼び、これをどのように蓄えるかによりFPGAの性質が決まります。このような構成を、配線の海の中に論理要素の島があるイメージからアイランドスタイルと呼ぶ場合があります。



PLDをどのように構成するか、と、柔軟性をどのように実現するか、は互いに関連しています。プロダクトターム構成は、EEPROM型に向いており、FPGAは、SRAMやアンチヒューズ型に向いていますが、それ以外の型でも利用可能です。



設計は、Verilog-HDLやVHDLなどのハードウェア記述言語で行うことが多いのですが、最近はCレベルでの設計も多く使われるようになりました。これらの記述は論理合成、圧縮の手順を経て、対象のFPGAの種類に応じてゲートなどの割り当て(テクノロジマッピング）が行われ、配置、配線の結果、構成情報が出力されます。これをFPGAに流し込めば、設計通りの動作を行わせることができます。

FPGAの設計ツール

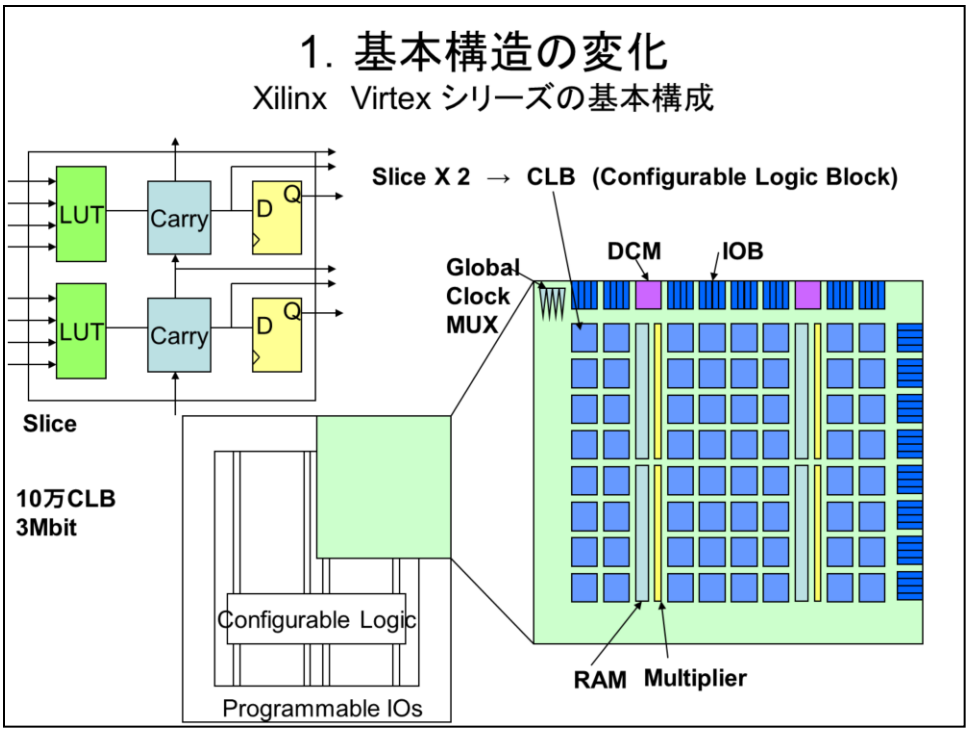
- ベンダが統合型のツールを提供
  - シミュレーション、配置、配線、構成情報の転送、デバッグツールが一式入っている
  - お試し版Web packでもかなりのことができる
- 設計入力
  - Verilog HDL、VHDLなどハードウェア記述言語が多い
  - 最近、C言語設計が発達
    - Impulse Cなどサードパーティの供給
    - Xilinx Vibado, SDAccel
    - Intel(Altera) OpenCL

Intel（Altera）社のQuartusなどがこの一例です。お試し版のWeb Packは無料でダウンロード可能で、かなり大きいチップまで設計ができます（実は実験ではこの無料版を使っているが、大学では大きいチップが設計できるアカデミックライセンスもある）。皆さんのPCにダウンロードして使うこともできます。

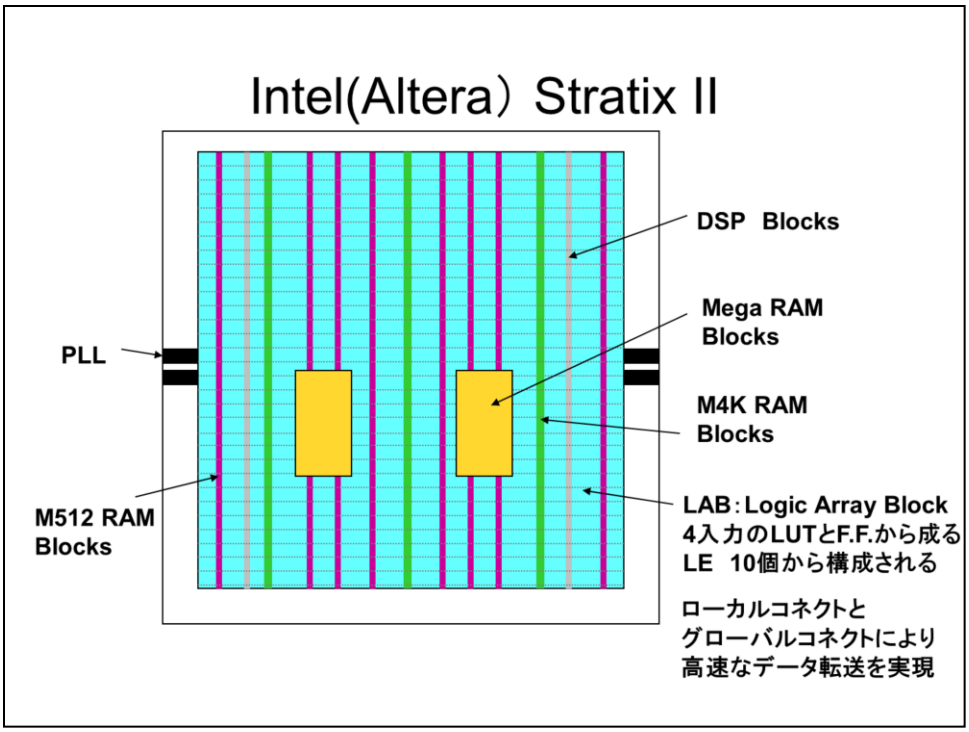
FPGAのトレンド

1. 階層的構造による大規模化、高速化：
   - Xilinx社Virtexシリーズ、Altera社Stratixシリーズ
2. ハイエンドとローコストの分化、ミドルレンジの登場
- System on Programmable Device
  - DLL、DSP、メモリ、乗算器、高速リンクをハードIPとして混載
  - SoC(System-on-a Chip)タイプのFPGA
    - CPU（ARM)を混載
- アクセラレータとしてのFPGA
- 部分再構成機能の充実

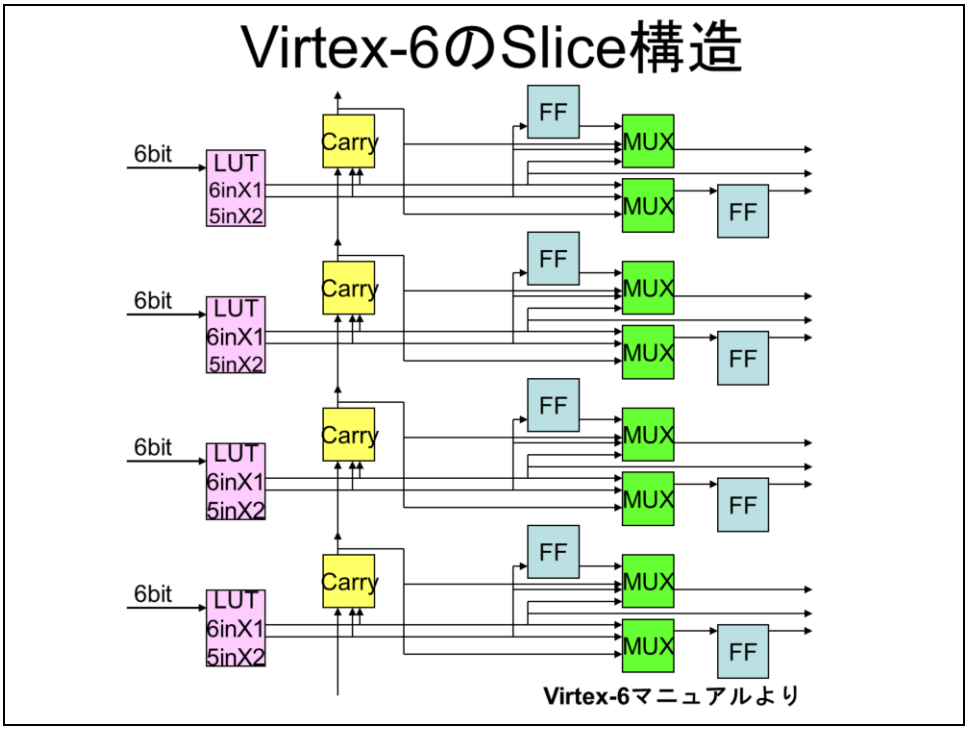
最近のFPGAは、ネットワークルータ、ハブなどに用いられる大規模、高速なハイエンドの製品と組み込み用の安価なローエンド製品に分化が進んでいます。両方共に、メモリ、クロックコントローラ、乗算器、高速リンクをハードIPとして搭載する製品が増えていて、システムをまるごとプログラマブルデバイスで実現するSoPD(System on Programmable Device)が利用されています。さらに低電圧、低電力に特化した製品も登場し、一部のみを書き換える部分再構成機能が充実してきています。



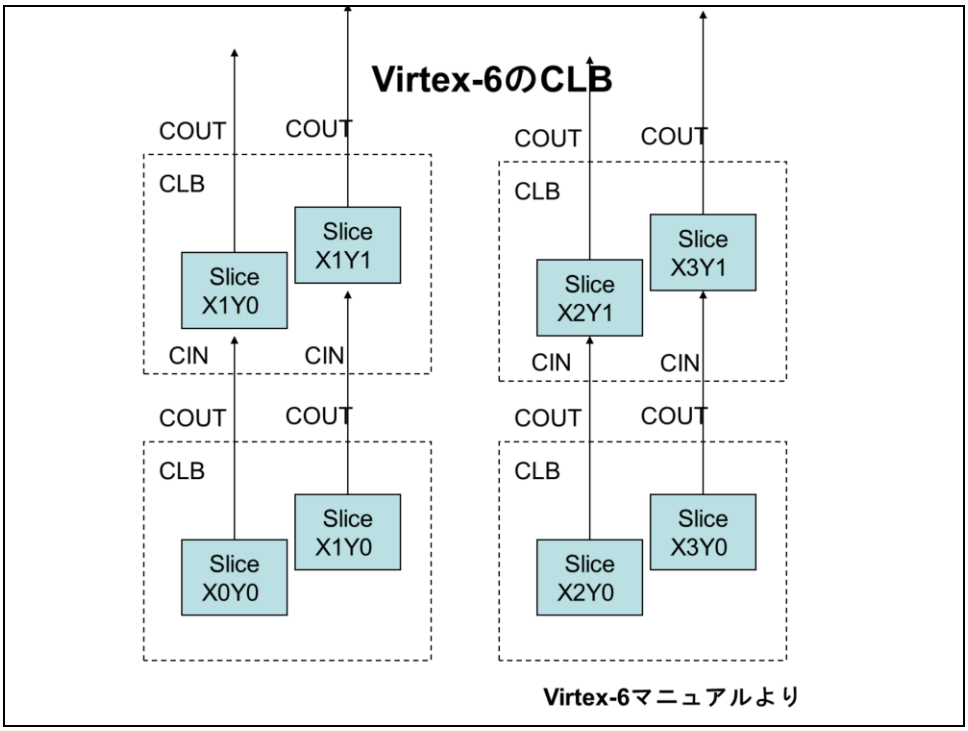
この図はXilinx社のVirtexシリーズの基本構成です。論理要素の配列の間にRAM, 乗算器などのIPが配置されており、DCM（クロックのコントローラ）、クロックバッファが周辺に配置されています。



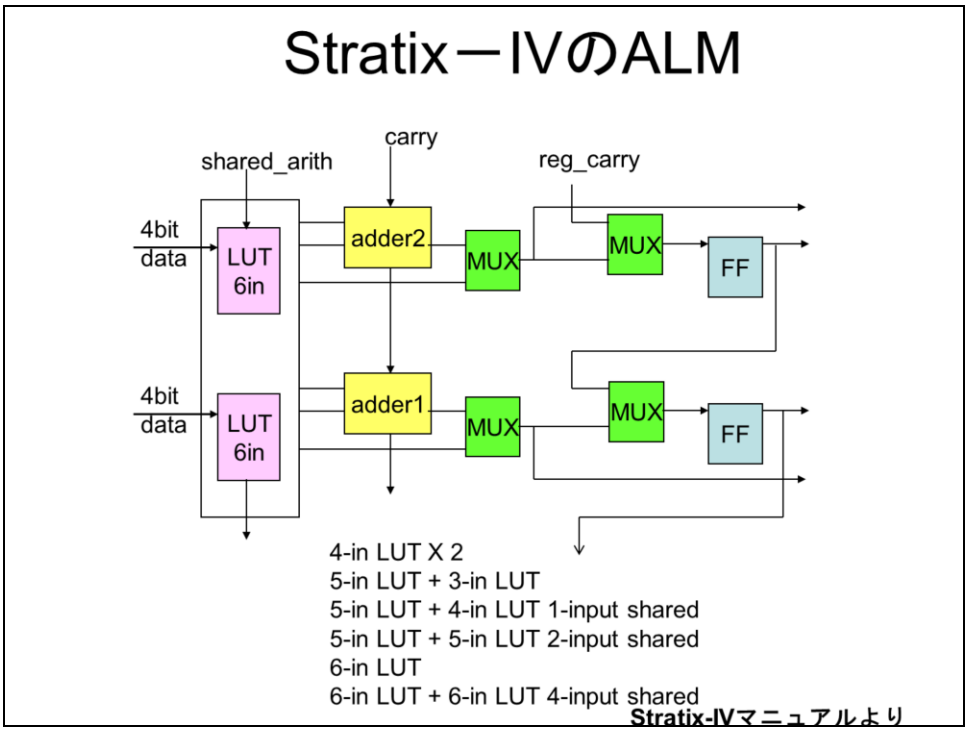
これはIntel(昔のAltera)のStratixシリーズです。これも一定の間隔でRAMやDSPブロック(積和演算を行うハードウェアのことをこのように呼んでいます)が装備されています。演算器は論理要素を使って作ることもできるのですが、専用のハードウェアを使った方がはるかに速いです。



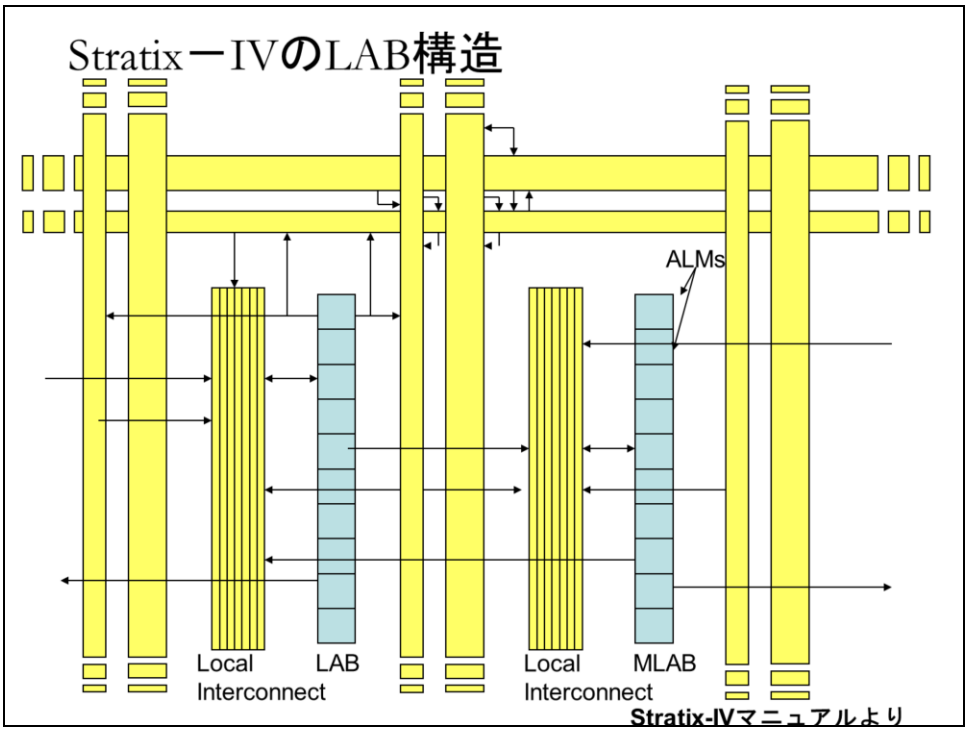
最近のFPGAの基本論理要素は、古典的なものに比べてLUTの入力が増えています。このLUTは6入力のものを1つとしても5入力のものを2つとしても使えるようになっています。論理要素間を直結するCarry（桁上げ）も用意されています。出力周辺の回路はマルチプレクサが入って柔軟性を増しています。



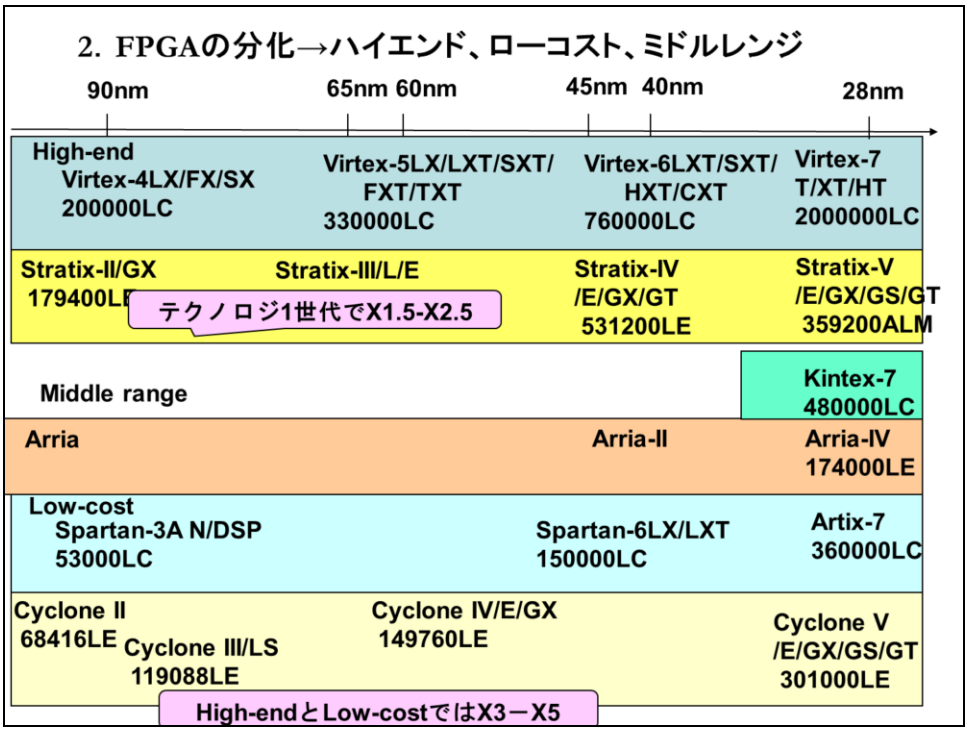
このスライス構造を二つでCLBと呼ぶ論理素子を構成します。スライス同士は直結線がなく、それぞれが隣のCLBのスライスと直結線を持っているのが特徴です。



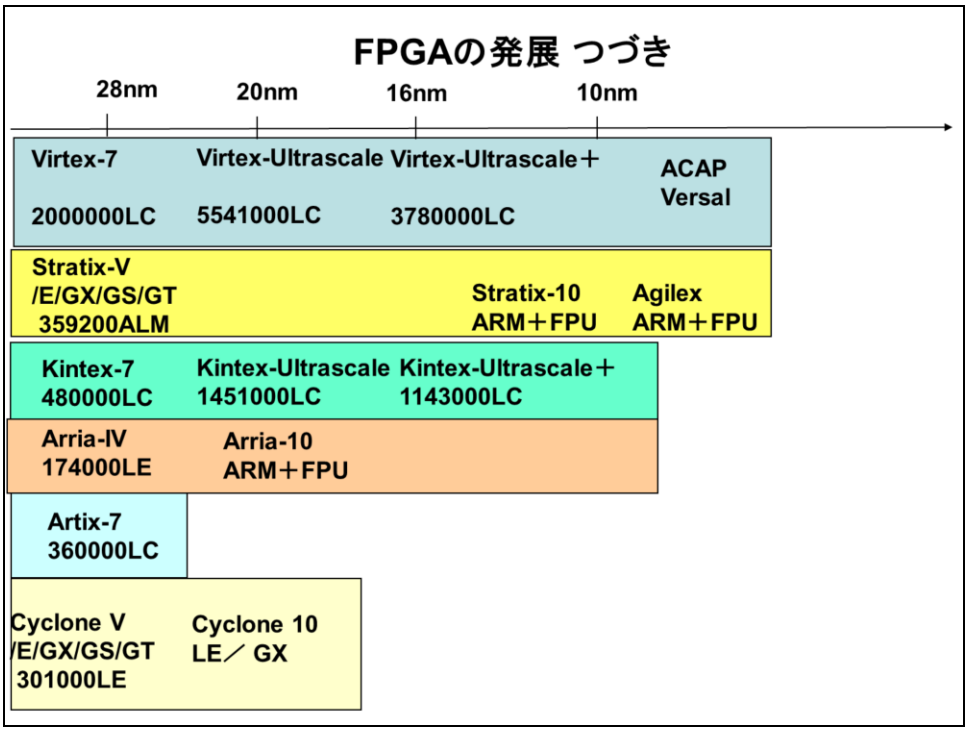
Xilinxのライバルの Altera社の基本論理構造です。Altera社は最近Intelに買収され、やや高性能製品に特化している傾向が見られます。Altera社のハイエンド製品Stratixの基本構成要素は、Xilinx同様6入力のLUTを使いますが、このLUTはさらに柔軟に様々な組み合わせを取ることができます。直結線、出力のマルチプレクサ構成など、両社の基本構成要素は似てきています。



Intel(Altera)のFPGAの特徴は、階層型の強力なネットワークを持つ点です。これにより、Intel社のStratixは非常に複雑な論理回路を搭載しても250MHzを越える周波数で動作します。



FPGAはその構造の単純さを利用して新しい半導体プロセスをいち早く取り入れて来ました。テクノロジが一世代違うと集積度は1.5-2.5倍になっています。低コスト用のチップも、容量はハイエンドの1/3から1/5ですが、積極的に新しいテクノロジを取り入れていることがわかります。

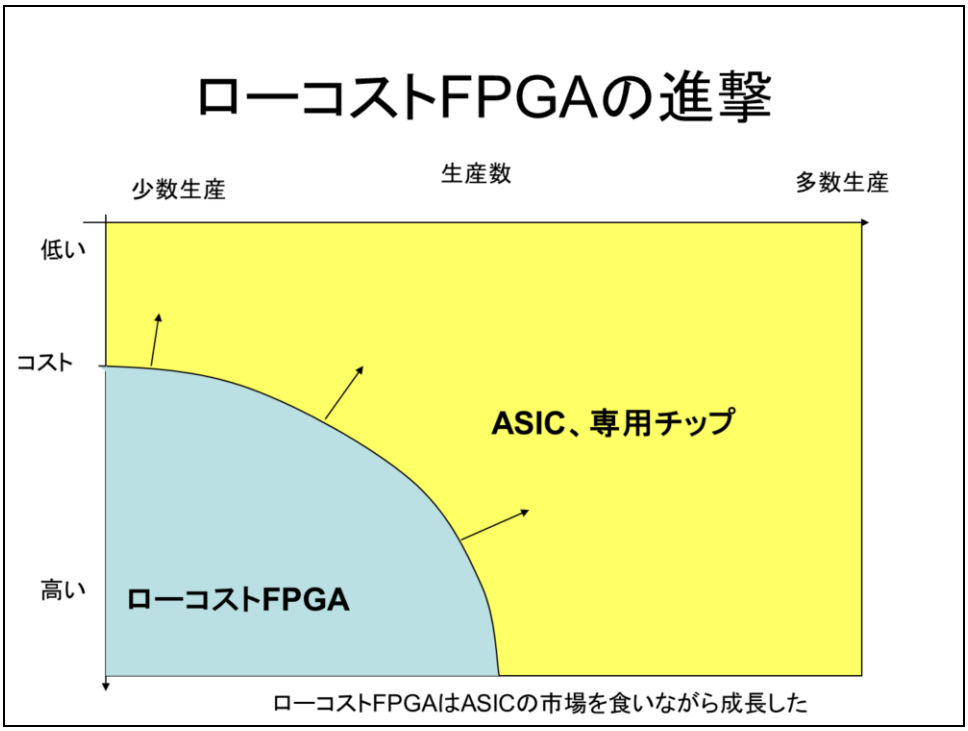


FPGAの発展はこれ以降も続いていますが、Low costの製品には最新のプロセスは使わないようになっています。

FPGA vs. ASIC

- ASIC(Application Specific IC) 特殊目的用IC
  - 製品に特化したICチップを特別に開発するもの
  - スマートフォン用、カメラ用、地デジ用、etc.
  - SoC(System on a Chip)、システムLSIと呼ばれ日本の半導体の主力だった
  - 最近のプロセスは開発コストが高騰し、非常に多くの個数を開発しない限り元が取れないことが多い
  - ビジネスとして難しい
  - 本来、FPGAよりも高速、低消費電力、低コスト
- FPGAの低価格品は、ASICの市場を食いつぶして成長
- FPGAは最新プロセスを使うため、小規模なチップでは性能的にはASICとあまり変わらない
- 消費電力、コスト（大量生産できれば）の点でASICが有利

FPGAとASICを比較すると、ASICの方が高速、低消費電力です。それなのになぜFPGAがASICの市場を取り込んで成長しているのでしょうか？それは最近のプロセスが最初の1個を作るまでのコスト（Non-Recurrent Costと呼びます）が高騰しているためです。これは、複雑なマスクパターンをいくつも使って作るため、マスク代自体と設計費用が膨大になってしまうためです。一度作ってしまうと機能の変えられないASICは、NRCに見合うだけの製造個数があらかじめ見込めないと作ることができなくなっています。一方で、FPGAは構造自体が簡単なため、いち早く最新プロセスで多くの同一製品を作り、NRCを回収することができます。しかし、本当に多数作る場合は今でもASICの方が有利です。ある製品を開発する場合、それに向けたASICを作るかどうか、FPGAでなんとかするのか、経営上の決断が重要に成ってきます。



ローコストFPGAは、ASIC、専用チップの市場を食いつぶしながら成長しました。しかし、最近は一世代前のプロセスを用いたASICが、大量生産分野で踏みとどまっています。しかし、最先端プロセスはローコストFPGAには高価になりすぎたため、もうこれ以上新しいプロセスを使わなくなる可能性もあり、両者の棲み分け体制ができていくでしょう。

最新プロセス取り入れモデルの限界

- 以前に比べて新シリーズ投入が遅れている
  - FPGAといえども内部構造が複雑化しすぎている
    - Stratix10の遅れの一つの原因
  - ASICはもっと遅れているのでアドバンテージはキープしているが、追いつかれつつある
    - Artix7は28nmプロセスが3万円程度で利用可能
    - 小規模なCPLDの需要は存在
- ASICは低コストFPGAの挑戦を退けた？
  - Artix, Cycloneは最新プロセスを使わない（使えない）
- 高性能FPGAは高性能CPU、GPU、DSAと同じ土俵で勝負せざるを得なくなった。
- DSA(Domain Specific Architecture)の発展

3．IP（Intellectual Property)と SoC型 FPGA

- 標準装備IP
  - 内蔵RAM
  - Multiplier, DSPモジュール（乗算器＋ALU)
  - Clock Manager
  - 高速シリアルI/O（ハイエンド、ローエンドの一部）
- 最近のIP
  - PCI Express（Virtex-6, Stratix－IV)
  - EthernetのMACコントロール（Virtex-6)
  - DRAM用メモリコントロールブロック（Spartan-6)
  - ハードコアCPU → ARMの普及(Arria10、Zynq)
  - 浮動小数演算器 Area 10、Stratix 10

FPGAは、標準ロジック要素だけでなく、様々なハードウェアをIP（Intellectual Property：知的資産)として持っています。IPの中でレイアウトまで決まっているものをハードコアIP、ハードIPと呼びます。ハードIPの中にはほぼ標準装備となっているものと、チップ特有のものがあります。PCIeのインタフェースやDRAMコントローラなど大変便利です。最近はFPGAを科学技術計算に利用するために浮動小数演算器を搭載するものも現れています。

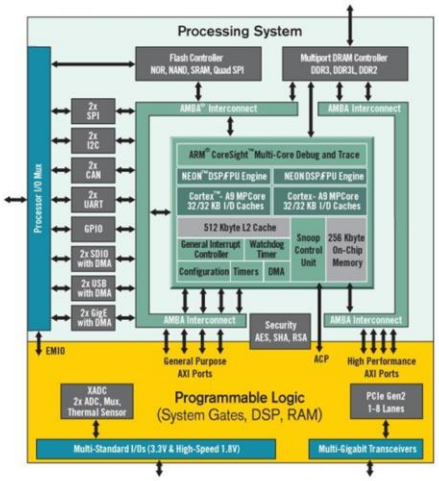
ハードコアとソフトコア

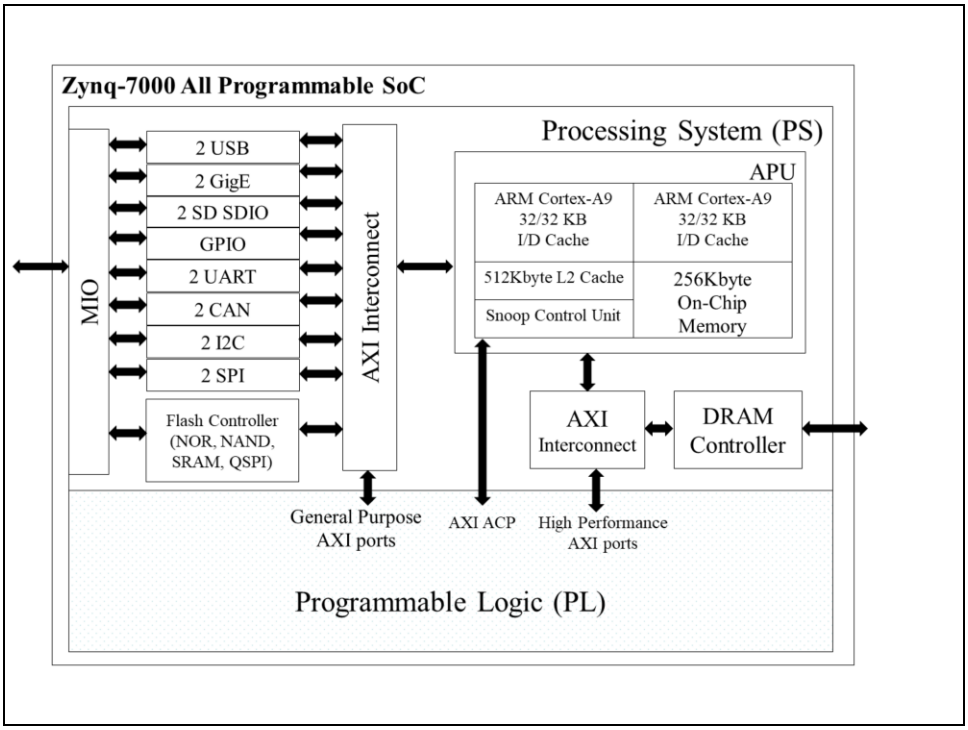
- ハードコア
  - レイアウトレベルで組み込んである
  - 標準的な命令セットを持つ
  - 利用可能なFPGAはかなり高価となる
  - ARM（Xilinx Xynq, Intel(Altera) Aria-10、Stratix10)
- ソフトコア
  - FPGA上のロジックブロックで構成する
  - 一定のサイズがあればどの製品でも利用可能
  - MicroBlaze (Xilinx)
    - 32ビット長、32レジスタを持つ
    - 約500スライス、85MHz動作(Spartan-3)
  - PicoBlaze (Xilinx)
    - 18ビット長、16レジスタを持つ
    - 96スライス、44MHz動作(Spartan-3)
  - Nios (Altera)
    - 16ビットのNios16と32ビットのNios32
    - 状況に応じて構成をチューンすることが可能
  - Cortex-M1(Actel)
    - ARM互換32bitプロセッサ

FPGAには、先に紹介にしたハードコアのCPUを持たないものは、内部の構成要素を組み合わせてソフトコアCPUを作ります。各社共に、FPGAに適した構成のCPUを用意しており、プログラム環境毎提供しています。かなり本格的なCPUもありますが、上位構成のものの多くは有料です。もちろん、オープンソースのCPUや各自設計したCPUを使う場合も多く、この辺がFPGAの特徴です。

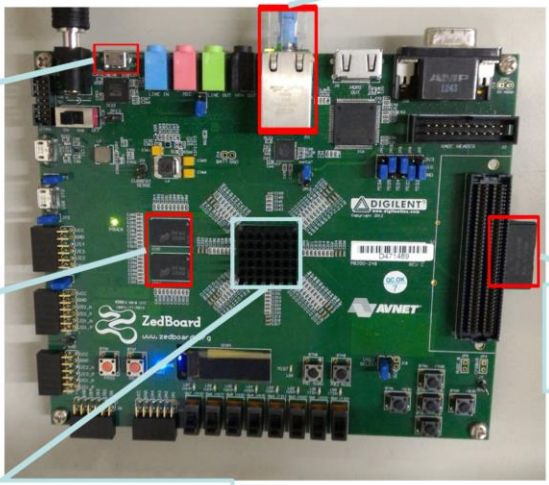
Zynq All programmable SoC

- ARM Cotex-A9プロセッサ（PS part)と、28nm Artix-7/Kintex-7 ベースのFPGA（PL part)の組み合わせ
- Vivado開発環境によりソフトウェア、ハードウェアの統合開発が可能
  - ソフトウェアで動かしてから HLSで簡単にオフロードが可能
- 安価はテストボードが普及
  - ZYBO Zynqボード
  - Zed Board
  - MicroZed Board



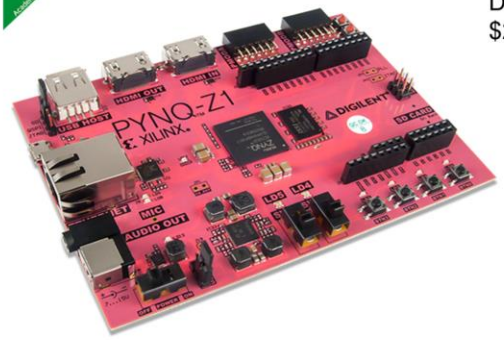

Zynq-7000 All Programmable SoC
Processing System (PS)
APU
ARM Cortex-A9
32/32 KB
I/D Cache
ARM Cortex-A9
32/32 KB
I/D Cache
512Kbyte L2 Cache
256Kbyte
On-Chip
Memory
Snoop Control Unit
MIO
2 USB
2 GigE
2 SD SDIO
GPIO
2 UART
2 CAN
2 I2C
2 SPI
Flash Controller
(NOR, NAND,
SRAM, QSPI)
AXI Interconnect
AXI
Interconnect
DRAM
Controller
General Purpose
AXI ports
AXI ACP
High Performance
AXI ports
Programmable Logic (PL)

ZedBoard


Ethernet
JTAG
DDR3 512MB
SD Card installed Linaro Ubuntu
Zynq-7000 All Programmable SoC
XC7Z020-CLG484-1
ZedBoard
DIGILENT
AVNET

31

PYNQ：Python+Zynq



DegilentのWebより
$229

PYNQとは？

- ハードウェアはZynq
- ARM上でLinuxが動作し、Pythonが使える
- ハードウェア部はPythonから使えるライブラリの形で供給される
  - Pythonで高位合成ができるのではない
- Pythonを使って、PL部でBNNによるDeep Learningのアクセラレーションができる
- FPGAによるDeep Learningの導入に優れた環境

4.アクセラレータとしてのFPGA

- Stratix 10
  - 14nm Intelプロセス利用
  - HyperFlexの採用によりGHz台の動作周波数
    - 配線構造上にレジスタを置く
  - 最大10TFLOPSの浮動小数DSPモジュール
  - ARM Cortex A53 Quad Core
- Arria 10
  - 20nm TSMCプロセス
  - 最大1.5TFLOPS
  - SoCタイプはDual Core ARM Cortex

Open-CLの利用

- GPU同様のアクセラレータとして扱う
  - ホストから入力データを転送→処理を起動→結果を収集
  - ホストの選択
    - PCIe経由でIntelのCPUを使う
    - SoCタイプでは内蔵ARMを使う
- BSP(Board Support Package)が必要
  - ボード依存性を吸収
  - 自作ボードにチップを使ってOpenCLを使うのが極めて難しい→ボード単位で利用するしかない
- HDLモジュールとの接続が難しい

Arria10 SoCボード

SSDより簡単にLinuxがブートする
Ethernetでネットワークに接続
内蔵ARMをホストにOpenCLでの設計ができる

ではGPUより速いのか？

	Stratix 10	Tesla P100
最大TFLOPS	9.2？	9.3
最大電力（W）	33－45	250
価格	177万（開発キット）	81万（Amazon）

- まだStratix 10とGPUとの比較は国際会議などでは出てきていない
- Arria10の場合は、アプリケーションとチューニングのテクニックによるがGPUには絶対性能では勝てない場合が多い。しかし電力性能では勝つ
- コストは現時点では不明（出だしなので高価すぎる）だが、Stratix Vを考える（シリーズによるが1チップ当たり130万円くらいする）と、GPUに比べて倍以上するのでは？

アクセラレータとしてのFPGAは成功するか？

- OpenCLの導入によりソフトウェア開発者、スパコン屋が利用可能になった
  - しかしFPGAの良い所が半減したかもしれない
- 科学技術計算では、GPUに性能面では勝てそうもない
  - Stratix 10ならば良い勝負ができるかもしれない
  - しかし値段が相当違う、、、
  - そのうちスパコン屋に飽きられるのでは？
- AI分野は？
  - OpenCLを使っているとGPU
- Intelには勝算があるのだろうか？？？

OpemCLの無料化
intel HLSの登場

FPGAアクセラレータには致命的な問題点がある

- 設計環境が悪いのではなく、設計時間が長すぎる。
  - 下手にOpenCLとか走るため、GPUと直接比較される。
- Kintex Ultrascale XCKU115に密行列のCG法を実装した場合
  - Vivado HLSの合成、ネットリスト生成：1時間
  - Vivadoの配置配線：1日~3日　時々配線に失敗する
- 特に利用率を上げると時間が増えていく。。。

→良い設計をするほど時間が掛かる

- GPUのコンパイルには、15分以上は掛からない
- 凄いメリットがあれば使うだろうが、良くてトントンの性能、エネルギー効率では勝負にならない。

→一般ユーザーはFPGAをアクセラレータとして認めないだろう

- FPGAを「ハードウェアを設計している」と思っている人は「これぐらい当然」だが、アクセラレータをプログラムしていると思っている人は、とても耐えられない。

ソリューションの提供

- Stratix10クラスの性能を持ち、ターゲットアプリケーションが決まっている（例えばCNNの学習）場合
  - HDLを使って専門家が根性でチューニングする
  - 性能、電力でGPUに勝てる可能性がある
- カスタマがはっきりしていれば勝機がある

→Maxelerのビジネスモデル

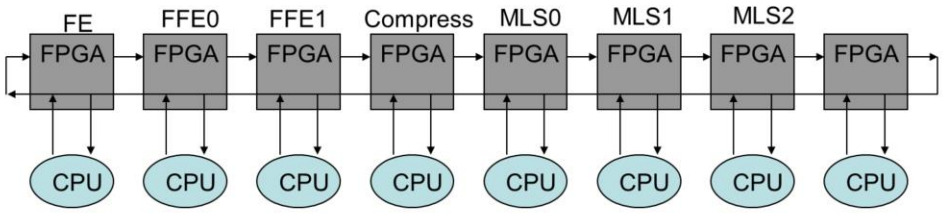
- Cloudで提供すれば機会が広がる

5.FPGA in Cloud

- Catapult project　by Microsoft [ISCA14他]
  - FPGAを用いた検索エンジンの導入
- FPGAの仮想化 [FPGA17]
- FPGA Supervessel Cloud by IBM
  - FPGAによるサービスの提供[ICFPT2016]
  - Xilinx SDACcelによるアプリケーション開発
    - OpenCLに似た開発環境
- Amazon EC2 F1インスタンス
  - Cloud上でのFPGAアプリケーション開発環境
  - Xilinx Ultrascale+を利用

Microsoft's Catapult

Rank computation for Web search on Bing.
Task Level Macro-Pipelining (MISD)
FE: Feature Extraction
FFE: Free Form Expression: Synthesis of feature values
MLS: Machine Learning Scoring



FPGA: Altera's Stratix V

2-Dimensional Mesh is formed (8x6) for 1 cluster.

元々CloudにはFPGAが使われている

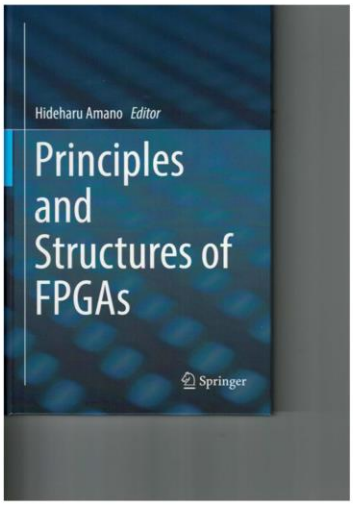
- ネットワークインタフェース、ネットワーク用スイッチはFPGAの主戦場
- ネットワークの機能を拡張するのはFPGAの利用法としては正統的
- 開発はベンダに限定、難しい
- NetFPGAは研究機関にも開放している

FPGA in Cloudの将来

- Cloud Computingのサービスの一部をFPGAが行うのは自然の流れ
  - Catapultなどの検索エンジン
  - 画像、文章などの認識処理
- ネットワーク制御の拡張は将来性がある
- CloudでFPGA開発を行う試みはどうか？
  - 環境を囲い込めるのでメリットはある
  - EC2　F1インスタンスは革命的
    - うまく行けばFPGA設計者にとってすごいことになる
    - しかし現実が付いてこないようだが、、、

ここまではFPGAの一般論でした

宣伝



ではPOCOをFPGA上に
実装してみよう