設 計 自 動 化 71－6
（1994．6．24）

UDL/I 第2期ソフトウェア開発 — 合成系ソフトウェアの開発

遠藤 真

NTT LSI 研究所

LSI 設計言語の標準化が進む中、UDL/I 言語は、その標準化作業を担う日本電子工業振興協会 LSI 設計用記述言語標準化委員会 (UDL/I 委員会) が、標準化推進活動の一環として、言語の検討とともに、主要アプリケーションであるシミュレータと論理合成システムの開発を進めて来た点でユニークである。

殊に論理合成システムの開発は、UDL/I がその厳密な意味定義により合成向きであることを自ら示す意味で重要な活動として位置付けられる。

UDL/I 委員会の第2期開発項目の一つである論理合成システムは、日本、米国、インドに拠点を置く3つの会社が開発を分担し、委員会の指導のもと、1年半という極めて短期間のうちに開発された。本稿では、論理合成システム開発プロジェクトの概要、システムを構成する各処理プログラムの概要について述べる。

UDL/I 2ND PHASE DEVELOPMENT
– LOGIC SYNTHESIS SYSTEM PROJECT –

Makoto Endo

NTT LSI Laboratories
3-1, Morinosato Wakamiya, Atsugi-Shi, Kanagawa 243-01, Japan

UDL/I is a hardware design language for LSI. UDL/I Standardization Steering Committee in Japan Electronic Industry Development Association has been standardizing UDL/I. The committee has developped a simulator and a logic synthesis system.

Development of a logic synthesis system is an important activity for UDL/I standardization to show its feasibility, because the language is designed for synthesis.

A logic synthesis system has been developed by a Japanese company, a U.S. company and an Indian company in the second phase development project of the UDL/I committee for one and half years.

We present the scheme of a UDL/I logic synthesis system project and programs in the system.

1 はじめに

LSI 設計言語の標準化が進む中、UDL/I 言語 [1] は、その標準化作業を担う日本電子工業振興協会 (電子協) LSI 設計用記述言語標準化委員会 (UDL/I 委員会) が、標準化推進活動の一環として、言語の検討とともに、主要アプリケーションであるシミュレータと論理合成システムの開発を進めて来た点でユニークである。

殊に論理合成システムの開発は、UDL/I がその厳密な意味定義により合成向きであることを自ら示す意味で重要な活動として位置付けられる。

本稿では、UDL/I 委員会の第 2 期開発項目のうち論理合成システム開発プロジェクトの概要、システムを構成する各処理プログラムの概要について述べる。

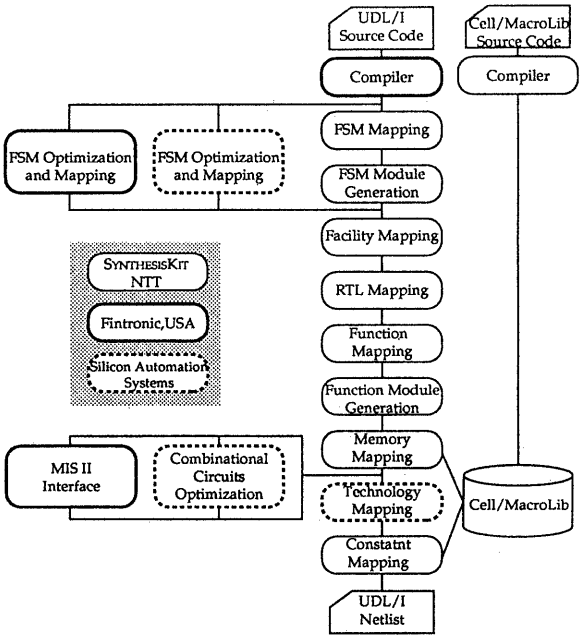

図 1: 論理合成システム概略構成

2 開発プロジェクト概要

2.1 開発の経緯

本論理合成システムは、UDL/I 委員会第 2 期ソフトウェア開発項目に挙げられ、開発完了を 1993 年度末の予定とする計画が 1992 年の DAC の時期に合わせて発表された。

このような短期間のうちに開発を終えるために、UDL/I 委員会は NTT が開示する論理合成システム SERAPHIM[2][3] の基本合成部を SYNTHESISKIT と呼ぶフレームワークとして委員会の論理合成システムのベースして採用することにした。次に、当時まだ検討中であった UDL/I 言語第 2 版のコンパイラと SYNTHESISKIT に含まれていない最適化部、テクノロジマッパ部の開発を担当する企業を募り、米国の Fintronic,USA 社と Silicon Automation Systems(SAS) 社が受注した。このうち SAS 社の開発拠点はインドなので、実質的には本論理合成システムの開発に日本、米国、インドの 3 か国の企業が携わっていることになろう。共通の言語は英語と C++、そして UDL/I である。

2.2 開発項目

電子協が開示を受けた部分 (NTT SYNTHESISKIT) と新規開発部分 (Fintronic, SAS) を図に示す。

SYNTHESISKIT は FSM・RTL を入力とする合成部分とセルライブラリ入力を含んでいる。新規開発部分は UDL/I コンパイラ、最適化部、テクノロジマッピング部である。また、外部の最適化システムを利用できるように、MIS II[4] インタフェースを設けている。

なお、システムの外部仕様、詳細構成については、文献 [5] を参照されたい。

2.3 開発方針

およそ 1 年半という限られた期間内で本論理合成システム開発を終えるために、以下の点を考慮した。

1. 入出力インタフェースの統一
   本論理合成システムを開発する 3 社 (NTT、Fintronic、SAS) の処理プログラム間のインタフェースにミスマッチが起きないようにする。

2. コード共有化
   開発期間短縮のため、出来るだけコードを共有する。前項のインタフェースの統一による入出力処理の共通化だけでなく、できるだけデータ構造も共通化する。

3. デバグの容易化

(a) 問題の切り分けが容易になるように、各処理プログラムを分離できるようにする。開発・デバグ時には処理プログラム間をファイル結合にする。

(b) 内部データのチェックのため、テキスト形式の出力を設ける。

4. 開発環境の統一
開発マシンと OS 及び C++コンパイラを統一する。

電子協に開示された SYNTHESISKIT は以上の点を考慮して開発されており、

- 1と 2のための基本データ構造とアクセス関数
- 3のための中間ファイル形式とアクセス関数
- 内部データの pretty print 出力

を含んでいる。

本論理合成システムの各処理プログラム間で受け渡されるデータは、

1. FSM
2. RTL
3. ブーリアン・ネットワーク
4. ネットリスト
5. セルライブラリ

であり、全て上記の基本データ構造にカバーされているので、SYNTHESISKIT の基本データ構造を全面的に採用することに問題はない。対応する中間ファイル形式と pretty print 出力も同様である。

最適化処理など、アルゴリズムに依存したデータ構造を必要とするものは、入力の基本データ構造から必要な情報を抽出して用いればよい。処理後は、固有データ構造から基本データ構造に変換する。

これらの入出力処理は、SYNTHESISKIT の採用如何に拘らず処理プログラムが実現すべき部分であり、入出力処理が共通化されたことによる開発期間短縮の効果は大きい。

2.4 開発体制

UDL/I 委員会の開発専門委員会および合成仕様 WG の指導のもとに、開発を受託した京都高度技術研究所 (ASTEM) と開発を担当した Fintronic 社と SAS 社、SYNTHESISKIT を開示した NTT が開発・サポートにあたった。

開発進捗管理、バグ情報管理、共通コードの管理・リリースは ASTEM が一貫して行い、他の 3 社がそれに従った。委員会メンバに対するβリリース後も同様の体制を取った。

開発担当間の情報交換の手段は主に、電子メール、FAX、電話である。また詳細な打合せは Fintronic、SAS 社担当者が来日した時と、ASTEM と著者ら合成仕様 WG メンバが現地 (米国、インド) を訪問したおりに行った。

3 論理合成フレームワーク概要

本章では、論理合成フレームワーク SYNTHESISKIT の

1. 基本データ構造
2. セル / マクロ・ライブラリ
3. 中間ファイル形式と pretty print 出力

について概略を述べる。

3.1 基本データ構造

SYNTHESISKIT の基本データ構造 MONAD (Monolithic Data Representation for Logic Design Data Structures) は、C++のクラスとして実現され、以下のクラスから構成される。

1. ベース・クラス：ダブルリンクト・リスト。
2. 共通クラス：名称のハッシュテーブル等。
3. 構造クラス：モジュール、ピン、ネット等の構造部品。
4. 動作クラス：FSM、RTL 動作文、UDL/I のファシリティ等。
5. 組合せ論理クラス：ブーリアン・ネットワーク、BDD[1] 等。

ここで FSM は UDL/I のオートマトンにあたる。また、RTL 動作文は UDL/I の動作記述に対応する単純な文の集合である。

[1] 開示された SYNTHESISKIT には含まれない

UDL/I の「コアサブセット」は意味定義のために簡潔な形式で定められており、全て合成可能であるが、レジスタとターミナルに対する動作の情報に全て還元されており、FSM の状態・タスク等の固有の情報が失われている。また、FSM の状態割り当てが 1 状態 1 レジスタに固定されているなど、合成のためには簡略化され過ぎている。そこで、合成用のコアサブセットにあたるものとして、RTL 記述を単純化すると共に、合成の自由度を確保するように RTL 動作文を定めた。

RTL 動作文は UDL/I の動作記述のサブセットと見なすこともできる。相違点は単純代入文や関数参照文にも offstate 値があることである。全ての RTL 動作文が単純代入文とブーリアン関数参照文に還元されるマッピングの過程を通じて、「信号の衝突」を正しく扱えるようにするためである。

RTL 動作文の種類と対応する UDL/I 記述例を付録 A に示す。

なお、MONAD の詳細については、文献 [2][3] を参照されたい。

3.2 セル / マクロ・ライブラリ

UDL/I 第 2 版の仕様には、ライブラリ記述が含まれていない。第 1 期開発分のシミュレータのライブラリ記述も論理合成に必要な情報を記述するようにはなっていない。

そこで、SYNTHESISKIT に含まれる独自のセル / マクロ・ライブラリ記述言語とそのコンパイラ、MONAD と同様に C++クラスとメンバ関数として実現した Cell/MacroLib パッケージを採用した。

Cell/MacroLib は、レジスタ、ラッチや組合せ回路のスタンダードセル、UDL/I のシステム関数のマッピングで使用できるマクロセルをライブラリの要素にでき、SYNTHESISKIT 中の基本合成部と本開発のテクノロジーマッピングで共用することが出来た。

セルの記述例を付録 B に示す。

各セルには、総称名 (generic name) という機能に対して一意に決まる名称を与えて、これによってレジスタ・バインディングやテクノロジーマッピング時に、機能を容易に識別できるようにしている。レジスタ等のピンにも総称名が与えられるものがある。総称名はセル・ライブラリに依存しないので、合成 / マッピング処理の実現が容易である。以下に総称名の例を示す。

表 1: 総称名の例 (セル)

総称名	機能
BFI	Inverter
BFZB	Tristate/Buffer
FDRP	D-FF with $\overline{\text{Reset}}$ and $\overline{\text{Preset}}$
FJRP	JK-FF with $\overline{\text{Reset}}$ and $\overline{\text{Preset}}$
LD	D-Latch($\overline{\text{Hold}}$)

表 2: 総称名の例 (ピン)

総称名	機能	セル名
Q	Data Output	FDxx,FJxx,
QB	$\overline{\text{DataOutput}}$	LDxx
CLOCK	Clock	FDxx,FJxx
D	Data Input	FDxx,LDxx
J	J Input	FJxx
K	K Input	
RESET	$\overline{\text{Reset}}$	xxRx
PRESET	$\overline{\text{Preset}}$	xxPx
HOLD	$\overline{\text{Hold}}$	LDxx

3.3 中間ファイル形式

MONAD と Cell/MacroLib 共通の中間ファイル S.IF (SERAPHIM Interchanging File Format) は、それぞれのデータ構造と 1 対 1 に対応しており、メモリ上の MONAD、Cell/MacroLib データをそのままファイルに出力できる。即ち MONAD データを入出力する処理プログラムは、ほとんど手を加えずにファイルを介して結合する形態に移行できるということである。

S.IF はテキスト形式なので、データ保存のためにはかさばるという欠点があるが、pretty print の結果とあわせて、デバッグには都合がよい。

3.4 pretty print 出力

MONAD の動作クラスと構造クラスは UDL/I の基本動作と構造記述を表現している。また、残る組合せ論理クラスのブーリアン・ネットワークは、論理式と等価である。したがって、MONAD の内容の pretty print を UDL/I 形式とするのは自然である。その際ブーリアン・ネットワークは、階層を分けたサブモジュールとし、そのモジュール・タイプにあたるモジュールの

記述は、AND(&),OR(!),NOT(^) 等の演算子を用いたBEHAVIOR_SECTION の記述としている。

この UDL/I 形式の pretty print を用いると、UDL/I コンパイラや目視によって処理プログラムの異常を容易に発見することができる。また、処理プログラムの結果を UDL/I 記述として再利用することもできる。

4 処理プログラム概要

4.1 UDL/I コンパイラ

言語のパーサはシミュレータと共通である。コード生成部が UDL/I 記述を単純化し、MONAD への変換をしている。典型的な処理は、case 文と論理式である。

4.1.1 case 文

MONAD には case 文がないので、全て if 文 (MONAD では条件代入文とクロック付き条件代入文 [付録 A]) に変換している。else 節も同様に if 文の条件を反転させたものを条件とする if 文に変換している。これは一見冗長のようであるが、UDL/I では、あるファシリティに対する代入を 1 か所以上で記述できるため、MONAD が case 文のデータ構造を保存しても、合成時に case 外に同一ファシリティに対する代入がないか調べて offstate による信号の衝突を調べなければならないので、合成処理の手間は等しい。

4.1.2 論理式

代入文中の単項演算子、2 項演算子は MONAD では _NOT(a)、_AND(a,b,c) 等、対応するそれぞれの関数参照文に変換される。すなわち論理式は、以下のように生成された中間ターミナルと複数の関数参照文に分解される。

```
x := a & b & c ! d;   ⟶   x := _OR( t,d );
                          t := _AND( a,b,c );
```

これも処理上非効率のようであるが、最適化などの組合せ回路の処理には、各処理プログラムがアルゴリズムに応じたデータ構造に変換するので、基本となる MONAD の RTL 動作文では単純な関数参照文のみをサポートしている。MONAD でも組合せ回路用に、ブーリアン・ネットワークが定義されている (3.1節参照)。

4.2 論理合成

論理合成は、ある記述レベルからより「低い」記述レベルへのマッピングとみなすことができ、いわゆるテクノロジーマッピングと、テクノロジー独立なマッピングに分けられる。SYNTHESISKIT では、各マッピングを以下のようにモデル化している。

各マッピングにおけるモデルは、可読性の高いテンプレート記述で定義されており、変更、カスタマイズが容易になっている。

詳細は文献 [2][3] にゆずり、ここでは概略を述べる。

4.2.1 テクノロジー独立マッピング

1. ファシリティマッピング
2. FSM マッピング
3. 関数マッピング
4. RTL 動作文マッピング
5. FSM モジュール生成
6. 関数モジュール生成

ここで、1、2、3はファシリティ、FSM、関数をそれぞれ仮想のモジュールの外郭にマッピングする処理であり、4は単純代入文かブーリアン関数参照文にマッピングする処理、5、6は 2、3の仮想モジュール内の回路を生成する処理である。

4.2.2 テクノロジー依存マッピング

1. メモリマッピング
2. 定数マッピング
3. 組合せ回路マッピング

1は、いわゆるレジスタ・バインディングで、仮想レジスタをセルライブラリ中の実レジスタにマッピングする。2は定数の処理、3はいわゆるテクノロジーマッピングで、SYNTHESISKIT には含まれないので、新規開発の対象になった。

4.3 FSM 最適化

前節 4.2のように、SYNTHESISKIT は FSM の合成を状態とタスクに相当する端子を持つ仮想モジュールの外郭の生成 (FSM マッピング) と、内部回路の生成の 2 段階に分けている。UDL/I のオートマトンの各ステー

ト内に記述された動作は、この仮想モジュールの状態端子を参照することになる。

状態数の最少化や状態の符号最適割当は仮想モジュール内で行い、状態遷移回路やデコーダ等のレジスタ周辺回路の最適化、仮想モジュールの階層展開(フラッタリング)後に行うことにしている。SAS 社の場合も状態割当と回路生成・最適化の2段階に分けている。基本アルゴリズムは、文献[6]に基づいている。

一方 Fintronic 社は、仮想モジュールに閉じず、状態遷移文の実行条件や状態を参照する RTL 動作文からハード量を総合的に予測して状態割当を行っている。基本アルゴリズムは、文献[7]に基づいている。

4.4 組合せ回路最適化

SYNTHESISKIT に含まれているのは基本的なもので、

1. 組合せ回路部分を切り出しとブーリアン・ネットワークへの変換
2. ブーリアン・ネットワーク上の局所最適化
3. ブーリアン・ネットワークから RTL 動作文への変換

である。

SAS 社の最適化は、ESPRESSO[8]の改良アルゴリズムによる2段論理最適化と独自の多段論理最適化である。

Fintronic 社は、MIS II へのインタフェースを開発し、基本的なスクリプトをサポートしている。

ユーザが BDD 等の技術を用るなど独自の最適化を施す場合には、この MIS II のファイル・インタフェースを用いることができる。

4.5 テクノロジマッパ

SAS 社の開発したテクノロジマッパは定数処理も同時に行っている。従って実際には図1中の SYNTHESISKIT の定数マッピング・プログラムは使用されない。

SAS 社のアルゴリズムは、ツリーマッチングを基本とした DAGON[9]ライクの独自のものである。面積と遅延をコスト関数にして最適化を行っている。既存回路を含む場合、そのモジュールの遅延を Cell/MacroLib で記述し、コンパイルしてライブラリ形式にすることにより、その遅延を参照して最適化を行う。

組合せ回路最適化と合わせた性能比較については、文献[5]を参照されたい。

5 UDL/I 言語仕様へのフィードバック

第1期ソフトウェア開発では、UDL/I の厳密な意味定義に従ってシミュレータが開発され、その開発過程で明らかになった問題を言語仕様へフィードバックした。第2期開発においても、実用的な合成ができるという観点からいくつかの問題点をフィードバックして仕様に反映させることができた。

開発専門委員会合成仕様 WG から標準化専門委員会へ提案した項目は、

1. システム関数 SMULT(2 の補数の乗算)の追加
2. オートマトン同期割り込みの追加とシンタックスの修正

であり[10]、言語仕様第 2.1.0 版[1]に反映された。

オートマトンへの同期割り込みと等しい動作は従来の仕様でも記述できるが、たとえば以下のように繁雑になるばかりか、冗長な回路が合成される可能性がある。ここで、"cond" は同期割り込み条件、"sti" は割り込み先ステートである。

```
AUTOMATON : auto1 : .rst : FALL_HIGH(.clk) ;
  st1 :
    ...
    IF cond THEN -> sti END_IF ;
  st2 :
    ...
    IF cond THEN -> sti END_IF ;
    ...
```

同期割り込み文を用いると、以下のように記述も簡略になる。

```
IF cond THEN -> auto1*sti END_IF ;
AUTOMATON : auto1 : .rst : FALL_HIGH(.clk) ;
  st1 :
    ...
```

6 まとめ

UDL/I 委員会の第2期ソフトウェア開発として、合成系ソフトウェアを開発した。プログラムの規模は、新規開発分約 200k ステップ、SYNTHESISKIT 約 100k ステップであり、大部分 C++でコーディングされている。

およそ1年半という極めて短期間のうちに開発できた理由を以下のように考える。

1. 合成フレームワーク SYNTHESISKIT の提供により、新規開発部分が少かった。
2. SYNTHESISKIT のデータ構造、ファイル・インタフェースの共用により、開発者の異なる処理プログラム間インタフェースも容易に実現できた。

3. 開発を受注した2社は、それぞれの分野で開発経験があり、そのノウハウを本開発に活かせた。

4. 第1期ソフトウェア開発時に作成したテストデータの大部分を本開発に再利用でき、テストが容易になった。

5. SYNTHESISKIT のマッピング・モデルがテンプレート記述されていたため、変更・デバグが容易であった。

6. テキスト形式の中間ファイルと UDL/I 言語形式の pretty print 出力により、デバグが容易であった。

本合成システム開発を通じて、同期割り込み等の言語仕様の追加・修正を提案でき、UDL/I がより実用的なものになったと考える。

しかしながら本開発の最も重要な成果は、論理合成向き設計言語としての UDL/I の特長を、合成系ソフトウェアの開発によって実証できたことであろう。

謝辞

本開発を進めるにあたり、UDL/I 開発専門委員会においてご指導ご討論いただいた唐津修委員長をはじめとする関係各位に感謝します。同委員会合成仕様 WG において詳細にわたる検討を重ねていただいた竹田信弘 WG 主査、藤原誠氏、神原弘之氏、横田吏司氏、星野民夫氏および関係各位に感謝します。本システムの開発に携わり、熱心にご討論いただいた Fintronic,USA 社ならびに Silicon Automation Systems 社の関係各位に感謝します。SYNTHESISKIT として本システムの一部となった SERAPHIM の開発に著者らとともに参画した NTT 高原厚氏、稲森稔氏、湊真一氏および関係各位に感謝します。

参考文献

[1] UDL/I 標準化専門委員会. UDL/I 言語仕様. 日本電子工業振興協会, 2.1.0 edition, March 1994.

[2] 遠藤真, 高原厚. 論理合成システム SERAPHIM テンプレート化合成手法. 情報処理学会第 45 回 (平成4年後期) 全国大会予稿集 No. 2K-8, pp. 6_31–6_32, 1992.

[3] 遠藤真, 高原厚. テンプレート化論理合成手法. 情処研報 設計自動化 66-5, 1993.

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS : A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, Vol. CAD-6, No. 6, pp. 1062–1081, June 1987.

[5] 神原弘之, 横田吏司, 遠藤真. UDL/I 第2期ソフトウェア開発 — 合成系ソフトウェアのテスト. 情処研報 設計自動化 71-7, 1994.

[6] Ashish Sirasao. FAST: A state assignment heuristic for efficient multilevel logic implementation of finite state machines. Master's thesis, Indian Institue of Technology, Bombay, January 1993.

[7] James R. Story, Harold J. Harrison, and Erwin A. Reinhard. Optimum state assignment for synchronous sequential circuits. *IEEE Transactions on Computers*, Vol. C-21, No. 12, pp. 1365–1373, December 1972.

[8] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[9] Keutzer K. Technology binding and logical optimization by DAG matching. In *24th ACM/IEEE Design Automation Conference*, pp. 341, 1987.

[10] UDL/I 標準化専門委員会. LSI 設計用記述言語の標準化に関する調査報告書. Technical report, 日本電子工業振興協会, March 1994.

A RTL 動作文と UDL/I 記述例

- 単純代入文 *A* := *B*;
- 連結代入文 *A* := *B*!!*C*!!*D*!!...;
- 関数参照文 *A* := function(*B*,*C*,*D*,...);
- 条件代入文

  IF *cond* THEN

  A := *B* OFFSTATE *offstate* END_IF;
- クロック付き代入文

  AT RISE(*clock*) DO *A* := *B* END_DO;

- クロック付き条件代入文

  AT RISE(*clock*) DO IF *cond* THEN

  A := *B* END_IF END_DO;
- メモリ参照文 *A* := *MEM* < *address* >;
- レジスタ・リセット文

  IF *cond* THEN RESET(*REG*1,*REG*2,...);
- レジスタ・プリセット文

  IF *cond* THEN PRESET(*REG*1,*REG*2,...);
- オートマトン状態遷移文

  AT RISE(*clock*) DO IF *cond* THEN

  *state*1−>*state*2 END_IF END_DO;
- オートマトン同期割り込み文

  IF *cond* THEN −>*state* END_IF;
- オートマトン非同期割り込み文

  IF *cond* THEN =>*state* END_IF;
- タスク消滅文

  AT RISE(*clock*) DO IF *cond* THEN

  FINISH(*task*1,*task*2,...) END_IF END_DO;
- タスク転移文

  AT RISE(*clock*) DO IF *cond* THEN

  task−−>*task* END_IF END_DO;
- タスク発生文

  AT RISE_HIGH(*clock*) DO IF *cond* THEN

  state−>>*task* END_IF END_DO;
- タスク・リセット文

  IF *cond* THEN

  RESET(*task*1,*task*2,...) END_IF;

B Cell/MacroLib 記述例

UDL/I 開発委員会配布セルライブラリの抜粋

```
//      D-type FF With Reset
cell entity of DFFR : FDR, SEQPRIM;
  input D, RN;
  clock CK;
  output Q, QN;
  function
    TABLE: D, CK, RN : Q : Q, QN;
           0  P   1  : ? : 0  1 ;
           1  P   1  : ? : 1  0 ;
           ?  N   1  : ? : -  - ;
           ?  B   1  : ? : -  - ;
           ?  ?   0  : ? : 0  1 ;
    END_TABLE;
  end function
end entity
cell body of DFFR : FDR, SEQPRIM;
  fanin
    D : 1;
    CK : 1;
    RN : 2;
  end fanin
  max fanout
    Q : 19;
    QN : 19;
  end max fanout
  area 17;
  path delay
    Q<-CK :  Q(HIGH<-LOW): 0.74 + FO*0.065;
    Q<-CK :  Q(LOW<-HIGH): 0.57 + FO*0.053;
    QN<-CK: QN(HIGH<-LOW): 0.74 + FO*0.062;
    QN<-CK: QN(LOW<-HIGH): 0.92 + FO*0.043;
    Q<-RN :  Q(LOW<-HIGH): 0.61 + FO*0.053;
    QN<-RN: QN(HIGH<-LOW): 0.77 + FO*0.062;
  end path delay
  constraints
    setup : D : CK : 0.94 ;
    hold  : D : CK : 0.62 ;
    synchronous : D : CK ;
    default value RN : HIGH ;
    pin properties
      D : D;
      Q : Q;
      QB : QN;
      CLOCK : CK;
      RESET : RN;
      opposite Q , QN ;
    end pin properties
  end constraints
end body
//      2,2-Input AND-OR-Invert
cell entity of AOI22 : AO2I22 ;
  input A, B, C, D;
  output Y;
  function
    Y = ^((A & B) ! (C & D));
  end function
end entity
cell body of AOI22 ;
  fanin
    A : 1;
    B : 1;
    C : 1;
    D : 1;
  end fanin
  max fanout
    Y : 11;
  end max fanout
  area 5;
  path delay
    Y<-A,Y<-B: Y(HIGH<-LOW): 0.31 + FO*0.112
    Y<-A,Y<-B: Y(LOW<-HIGH): 0.10 + FO*0.063
    Y<-C,Y<-D: Y(HIGH<-LOW): 0.32 + FO*0.112
    Y<-C,Y<-D: Y(LOW<-HIGH): 0.19 + FO*0.063
  end path delay
  constraints
    pin properties
      IN : A, B, C, D;
      OUT : Y;
      NEGUNATE : A, B, C, D;
    end pin properties
  end constraints
end body
```