# Design of a Cell Library for Formal High Level Synthesis

Sadiq M. Sait, Masud-ul-Hasan and Khalid Elleithy

## Abstract

In this paper we present a complete design and implementation of a CMOS cell library which supports a formal high level synthesis framework. The library contains the logic level models and VLSI layouts of all primitive functions of the Realization Specification Language (RSL) [1] as well as some commonly used functions which are also built using these basic functions. Modular design methodology is employed to support the expandibility of the basic cells. Example of a formal matrix-matrix multiplier is presented to illustrate the application of the cell library.

## 1 Introduction

Since the fabrication cost of VLSI chips is very high, the designer is required to have high degree of confidence in the correctness of a design before committing it to hardware. Conventional method of verifying correctness is computer simulation. Simulation may leave many design errors undetected, since exhaustive simulation of complex circuits and systems is not feasible. It is now widely accepted that current simulation techniques are not by themselves adequate to ensure the correctness of complex designs. Testing is another popular method that is used to prove the correctness of systems for specific sets of inputs. With the current advances in VLSI circuits, there is no testing procedure that is capable of accomplishing an exhaustive examination of complex circuits. To overcome these difficulties, a number of formal high level synthesis techniques are now being developed, and they are likely to become practical tools for detection of design errors.

A formal high level synthesis system is a system which transforms the formal specifications to implementable hardware. Here, synthesis is performed within the framework of a suitable formal system, such as first-order logic, higher-order logic, temporal logic or ASL (Algorithmic Specification Language), etc [1]. In formal high level synthesis system the design specifications (also called formal specifications) can be verified for correctness by applying mathematical rules.

Any high level synthesis system has two tightly coupled sub-systems, a **front-end** and a **back-end**. The front-end accepts a high level input description and produces an intermediate form. The back-end takes this intermediate form and produces corresponding VLSI layout. The front-end of the system under consideration accepts input in ASL and produces RSL (Realization specificastion Language) as an intermediate form. The back-end takes RSL as input and produces corresponding VLSI layout. The block diagram of a VLSI high level synthesis system is shown in Figure 1. It consists of two parts, the synthesis part and the physical design part. The role of the synthesis part

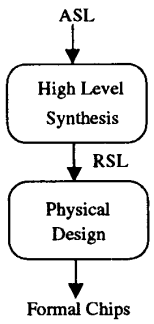


Figure 1: Block diagram of a VLSI synthesis system.

is to automatically translate an algorithmic specification into an architectural specification that is realizable in hardware. The physical design part translates the hardware specification to VLSI layout. In a general synthesis system, the front-end specifications to the synthesis can be behavioral descriptions of digital systems in hardware description languages, or even in programming languages. The synthesis system under consideration uses $\mu$-recursive algorithms to model the behavior to be synthesized. These algorithms can be mathematically verified for correctness before being subjected to the task of translation to architecture and then to corresponding VLSI layouts. Therefore this synthesis system is termed as **formal synthesis system**. The objective of this work is to make the building elements (cell library) required by the back-end to generate VLSI layouts. The cell library is usually the central part of the back-end of any cell based high level synthesis system.

In this paper we present a cell library which consists of both logic level and layout level cells used to support **formal high level synthesis** of digital systems modeled as $\mu$-recursive algorithms. In the next

0-7803-1772-6/94/$3.00 

section the algorithmic specification language (ASL) is introduced which is used in the formal synthesis system under consideration. In Section 3 the description of the *matrix-matrix multiplier* is presented as an example of $\mu$-recursion modeling. Section 4 introduces the design of the cell library. The design of a *matrix-matrix multiplier* in VLSI is used as an example to illustrate modeling in ASL, RSL (hardware) constructs, and application of the cell library.

# 2 Language ASL

A formal behavioral framework for synthesis is introduced in [2]. The given algorithm is represented using a newly developed language, termed Algorithm Specification Language (ASL). ASL consists of a limited number of constructs and is capable of representing any algorithm using these constructs. It has only three initial functions i.e., Zero, Projection and Successor functions, and three operations i.e., Composition, Recursion and $\mu$-Recursion [1].

These three initial functions and three operations of ASL can be applied in a certain sequence to obtain any computable function. Although this language of specification is complete it may be tedious to model a large digital system. A library of basic functions is defined starting from the initial functions to be used in the definition of larger functions [1]. This approach is useful in building a cell library to support VLSI synthesis. All basic functions that have been designed using the proposed approach can be used for specifying other functions. This technique supports a hierarchical design methodology in the sense that the specification can be stopped at any level as long as the lower levels are previously defined.

# 3 A Design Example: Matrix-Matrix Multiplier

An example of a formal *matrix-matrix multiplier* cell is introduced in this section, as an application of ASL language [1]. It is implemented by applying recursion construct on *inner-product* units. The architecture accepts two matrices as input, and produces a third matrix as an output. The multiplication is done in a recursive way, and can be described by the following high level subroutine:
Suppose A and B are the two input matrices and C is the output matrix.

```
matrix-multiplication (A, B, C)
begin
   for i = 1 to n
      for j = 1 to n
      begin
         C_{i,j,0} = 0
         for k = 1 to n
            C_{i,j,k} = C_{i,j,k-1} + A_{i,k} * B_{k,j}
         next k
      end
      next j
   next i
end
```

## 3.1 ASL and RSL Representation

The ASL description of *matrix-matrix multiplier* using recursion is as follows:

$$C_{1,1}(A_{1,k}, B_{i,j}, 0) = \xi()$$

.....

$$C_{n,n}(A_{n,k}, B_{n,j}, 0) = \xi()$$

$$C_{1,1}(A_{1,k}, B_{i,j}, K) = inner - product(A_{1,k}, B_{i,j},$$

$$C_{1,1}(A_{1,k}, B_{k,j}, K-1))$$

.....

$$C_{n,n}(A_{n,k}, B_{k,n}, K) = inner - product(A_{n,k}, B_{k,n},$$

$$C_{n,n}(A_{n,k}, B_{k,n}, K-1))$$

The RSL representation of *matrix-matrix multiplier* is as follows:

$$Initp(0, n; 1, A_{1,1}; \cdots; n^2, A_{n,n}; n^2{+}1, B_{1,1}; \cdots; 2n^2, B_{n,n})$$

$$suc_{control_1} = \xi()^{ready}$$

.....

$$suc_{control_{n^2}} = \xi()^{ready}$$

$$I = \rho_1^{n+1} suc(I)$$

$$Ready = eq?(I, m)$$

$$Result_1 = comp(\overline{arg}, I, \rho_{\xi()}^{n^2+1} Result\#inner{-}product)$$

.....

$$Result_n = comp(\overline{arg}, I, \rho_{\xi()}^{n^2+1} Result\#inner{-}product)$$

Figure 2 shows the RSL architecture obtained for matrix multiplication. The architecture consists of $n^2$ inner-product cells.

| Unit | No. of Devices | Area ($\lambda^2$) | Other Units Used |
|---|---|---|---|
| counter | 336 | 288320 | successor |
| add | 974 | 739480 | incrementer |
| pro | 1674 | 1284376 | add, incrementer |
| inner-product | 2720 | 2244528 | add,pro,incrementer |
| multiplier | 12060 | 12325000 | inner-product |

Table 1: Number of devices and layout area of 8-bit units.

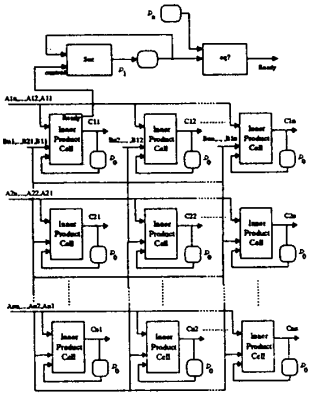

Figure 2: Implementation of *matrix-matrix multiplier.*

```
; The matrix-matrix multiplier circuit using
; simultaneous recursion

(include "oasis'lib.def")
(include "formal'lib.def")

(macro mul1(o11 o12 o21 o22 i11 i12 i21 i22 a11 a12
a21 a22 b11 b12 b21 b22 phi1 phi2 load b ready )

(local out count con a1 a2 a3 a4 b1 b2 b3 b4 ready1
r1b r11 rb11 r12 rb12 r21 rb21 r22 rb22)
;generate "dff"
      (dff con phi2 out.1 )
;generate "adl8"
      (adl out in1 phi1 phi2 load count b ready1)
;generate "inner-product cells"
(inpro o11 i11 in1 in5 phi1 phi2 load con m1 m2 r11 rb11)
(inpro o12 i12 in1 in5 phi1 phi2 load con m1 m3 r12 rb12)
(inpro o21 i21 in1 in5 phi1 phi2 load con m4 m2 r21 rb21)
(inpro o22 i22 in1 in5 phi1 phi2 load con m4 m3 r22 rb22)

      (o4 rb11 rb12 rb21 rb22 counta )
      (oi2 ready1 counta count )
      (o4 rb11 rb12 rb21 rb22 readya )
      (oi2 ready1b readya ready )

)
;generate globle node names
(node o11 o12 o21 o22 i11 i12 i21 i22 a11 a12
a21 a22 b11 b12 b21 b22 phi1 phi2 load b ready)
(mul o11 o12 o21 o22 i11 i12 i21 i22 a11 a12
a21 a22 b11 b12 b21 b22 phi1 phi2 load b ready)
```

Figure 5: Logic level model of *matrix-matrix multiplier.*

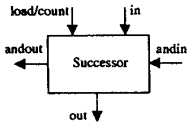


Figure 3: Successor function.

Figure 4: VLSI layout of 4-bit *Successor.*

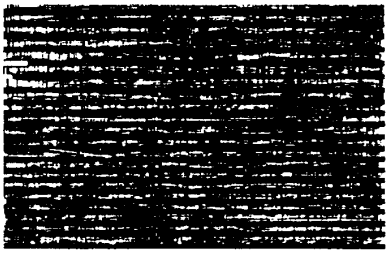

Figure 6: VLSI layout of an 8-bit *matrix-matrix multiplier.*

# 4 The Cell Library Design

In this section, we present the approach used for designing and implementing the formal cell library which consists of logic level models of basic units and their corresponding CMOS VLSI layouts. Corresponding to each construct in ASL is a construct in RSL that maps the constructs to hardware equivalent modules.

The RSL specification of basic functions is modeled as a logic netlist and simulated using a simulator [5]. This simulation verify the correctness of translation. Layouts of these functions are then synthesized [3]. The circuit is extracted from the layout and simulated to verify the functional correctness of layouts.

The layouts of larger functions can be synthesized by instantiating layouts of primitives and pre-defined macros. The synthesized logic and its corresponding layout are stored in the library for later use. In the design of both logic level and layouts of cells, care is taken to achieve modularity so that basic cells can be easily expanded to large word lengths and can be connected to build cells of larger functions. The VLSI layouts of cells are implemented using 2-phase dynamic CMOS logic.

## 4.1 The Successor Unit

In this section, we show the design of the successor unit as a case study of the methodology. One of the registers is for the argument n and the other for the value 1. The operation is done in one clock cycle. An input signal **control** is used to determine the starting of the operation. This unit can also be used as an up-counter by feeding back the output of the successor function to one of its two inputs. ***Load/Count*** control inputs are available to accomplish the necessary function. Two-phase clocking scheme is used in the design of all the cells, input is loaded during $\phi_1$ and the output is obtained during $\phi_2$. The cascading of successor units can be done by connecting the *andout* output of a unit to the *andin* input of the adjacent unit. The VLSI layout of 4-bit successor is shown in Figure 4.

## 4.2 The Matrix-Matrix Multiplier

In this section, we illustrate using the cell library to design the ***matrix-matrix multiplier***. It consists of one main building component i.e., *inner-product* unit which itself contains the components, *add* and *pro*. The *add* and the *pro* units are used with the recursion construct to build the ***inner-product*** unit. The logic model of ***matrix-matrix multiplier*** unit is illustrated in the Figure 5 [4]. The Figure 7 shows the VLSI layout of a matrix-matrix multiplier.

# 5 Conclusions

In this paper, we have presented a complete design of a cell library that supports the formal high level synthesis framework based on (RSL) realization specification language. The cell library is the heart of the back-end unit of a formal methodology for VLSI systems design. The zero, projection, successor, composition, recursion and $\mu$-recursion functions in RSL have been implemented using a SCMOS technology. Components in the RSL formal cell library are described at the logical level and layout level. The correctness of the components have been exhaustively verified through the RNL simulator. All the cells are made modular so that the design is capable of extending to any desired word length. These modules are also used to made the larger functions. A formal matrix-matrix multiplier circuit of three different types has been designed using the support of the cell library.

Table 1 shows the number of devices and layout area of these units for an 8-bit data bus. It can be observed that the area is high as compared to the that made by non-formal methods. It is the price paid for the functionally correct hardware made by formal techniques.

# Acknowledgements

The authors would like to thank the King Fahd University of Petroleum and Minerals for support.

# References

[1] Khalid M. Elleithy. *A Formal Framework For High Level Synthesis of Digital Designs*. PhD thesis, The Center for Advanced Computer Studies, University of South-Western Lousiana, 1990.

[2] Khalid M. Elleithy and Magdy A. Bayoumi. Synthesizing DSP architectures from behavioral specifications: A formal approach. *Proceedings-IEEE International Symposium on Circuits and Systems*, 2:1131–1134, May 1990.

[3] MCNC's Center for Microelectronics. *Open Architecture Silicon Implementation Software*, release 2.0 edition, December 1992.

[4] Masud ul Hasan. *Back-End Design of A Formal High Level Synthesis System*. Master's thesis, King Fahd University of Petroleum and Minerals, Dhahran, 1993.

[5] *VLSI Design Tools Reference Manual*. NW Laboratory for Integrated Systems, release 3.1 edition, February 1987