Cycle 수준의 Control Flow Description에서 합성 가능한 VHDL 기술로의 변환 방법에 관한 연구

윤 창 열, 장 경 선
충남대학교 컴퓨터공학과
전화 : 042-821-7720 / 핸드폰 : 011-9816-5527

A Translation Method from Control Flow Descriptions in cycle-accurate level to Synthesizable RTL VHDL Descriptions

Chang-Ryul Yun, Kyoung-Son Jhang
Dept. of Computer Engineering, ChungNam University
E-mail : yun@ce.cnu.ac.kr

Abstract

This paper defines an algorithmic description language in cycle-accurate level which can be used to design hardware components. The proposed language is less complex and more flexible than VHDL language. In that the language includes C-like control flow descriptions and brief timing information(i.e. clock cycle boundaries) indicated by 'wait_edge' statements. We generate RTL VHDL codes from the descriptions. The proposed language requires only 10~30% of the # of lines to describe the same functionality compared with RTL VHDL.

I. 서론

집적도가 증가함에 따라, 단일 칩에 넣을 수 있는 회로의 양이 증대되고 복잡해졌다. 이런 ASIC을 개발하는 데에는 상당한 비용과 시간이 필요하기 때문에, 설계 비용을 줄이기 위해 HDL(VHDL, Verilog 등) 언어를 이용하여 하드웨어 회로를 검증, 합성하는 논리합성 도구의 사용이 일반화되었다. HDL을 이용하여 SoC의 설계나 큰 용량의 회로를 설계하는 데에는 몇 가지 어려움이 있다. 구현하고자 하는 알고리즘을 설계자가 C로 프로그래밍하여 검증한 후에, 각각의 부분을 설계자들이 분담하여 HDL로 변환하는 과정을 거친다. 수작업으로 이런 알고리즘을 HDL로 변환하는 과정은 오류를 내포하기 쉽고, HDL 모델로 기술하였으나, 사용 요구의 변경이 있어 알고리즘이 변경될 경우에는 전체 HDL 설계를 다시 해야 할 수도 있다. 이런 어려움 때문에 설계 수준을 높이려는 연구가 있었다. 하드웨어/소프트웨어의 Co-design[1]을 위해 알고리즘 수준의 C 언어를 VHDL로 변환하려는 연구[2]도 있었다. VHDL의 경우에는 behavior 수준의 기술이 가능하나, 논리합성 도구를 이용하여 behavior 수준의 기술을 합성한 결과는 performance가 좋지 못하고, 많은 부분을 설계자에 의존하고 있다[3].

본 논문에서 제안한 언어는 알고리즘 수준의 기술에 타이밍 정보를 합하여, 효율적인 합성이 되도록 정의하였다. Ⅱ장에는 예제를 통해 정의한 cycle 수준의 Control Flow Description을 설명하고, Ⅲ장에는 정의한 언어에서 합성 가능한 VHDL 기술로의 변환 방법을 설명한다. Ⅳ장은 자동 생성한 VHDL 모듈과 사람이 설계한 모듈의 합성 결과를 비교한다. Ⅴ장은 결론 및 향후 과제를 기술한다.

Ⅱ. Cycle-C

본 논문에서 제안하는 cycle 수준의 Control Flow Description을 Cycle-C로 부른다. 그림 1은 Cycle-C의 기술 예이다. C 언어가 하드웨어 설계자에 친숙하기 때문에 문법의 형태는 C언어와 거의 비슷하고, C 언어

에서 제공하는 표현식을 거의 허용한다. 그림 1은 제안한 Cycle-C로 UTOPIA[4] Transmit 프로토콜의 기술이다.

```
Core Utopia_Tx {
    out bit TxSOC; /* ① ports */
    out bit TxEnbn; out byte TxData;
    out bit Data_delete = '1';
    in bit TxClav; in byte Data;
    in bit TxFulln;

    reset rst_n low; /* ②reset */
    clock TxClk rising; /* ③clock */

process(TxClav : TxSOC, TxEnbn, TxData,
        Data_delete, Data )
{       /* ④ algorithm description */
  int i, j; /* 사용자 정의 변수 */
  while(TxClav != 1)
      wait_edge(); /*⑤cycle boundary*/
  TxEnbn = 0;          TxSOC = 1;
  Data_delete = '1'; /*⑥ FIFO operation */
  wait_edge();
  TxSoC = 0;
  for (i = 1; i <= 52; i++) {
    if(TxFulln == 0) {
      for (j = i; j < i + 4; j++)    {
        Data_delete = '1'; /* FIFO operation */
        assert(TxFulln == 0); /*⑦ assertion */
        wait_edge();
      }
      wait_edge();
    }
    TxEnbn = 0;
    Data_delete = '1'; /* FIFO operation */
    wait_edge();
 }
}
netlists {
    TxData = Data;
} }
```

그림 1. UTOPIA Protocol

UTOPIA transmit 동작은 8bit의 데이터 53개를 전송하기 위한 프로토콜이다. Cycle-C의 기술은 Core를 기술함으로 시작된다. ①에 생성되는 모듈의 입출력 포트를 기술하고, ②의 리셋은 리셋 신호의 이름과 동작 타이밍(low/high)을 기술한다. ③은 클럭의 이름과 동작특성(rising/falling)을 기술한다. process구문 다음에 모듈의 동작이 기술된다. process 문의 전달인자는 입출력 포트를 기술하고 ":"를 기준으로 앞부분은 입력포트, 뒷부분은 출력포트를 기술한다. 이는 여러 프로세스의 기술에서 사용자의 주의가 필요하기 때문이다. ④부터는 생성되는 모듈의 동작에 대한 기술이다. 사용자가 임의의 변수를 정의해서 표현 할 수 있도록 허용했다. ⑤에서 알 수 있듯이, 한 사이클의 경계는 wait_edge() 문으로 표현한다. 임의의 wait_edge()문에서 다음 wait_edge() 문까지 한 상태로 생성되고, 두 wait_edge()문 사이에 기술된 구문은 전이 동작이 된다. ⑥은 외부의 FIFO에서 데이터를 출력함을 의미한다. 이때 FIFO에서 데이터가 삭제되기 때문에, "delete"의 이름으로 기술했다. ⑦의 표현은 프로토콜의 오류를 파악할 수 있는 표현식이다. 오류 발생 시에 FSM이 초기상태로 전이한다. ⑧은 생성된 모듈의 크기를 줄이기 위해 추가되었다. netlists 안에 기술된 할당문은 Concurrent Signal assignment문으로 합성된다.

Ⅲ. RTL VHDL 코드 생성

그림 2는 코드 생성 과정이다. RTL 수준의 VHDL 코드는 Cycle-C를 입력으로 생성된다. 생성 과정은 다음과 같다.

```
Parsing()                                  --- ⓐ
Make_Control_Flow_Graph()                  --- ⓑ
    Avoid_Infinite_Loop_Processing()       --- ⓒ
    Assert_Statement_Processing()          --- ⓓ
    Search_Argument_Variable()             --- ⓔ
    Make_FSM()                             --- ⓕ
    Detect_Latch_and_UnexpectedStorage()   ⓖ
Variable_Control_Processing()              --- ⓗ
Generating_IPC()                           --- ⓘ
```

그림 2. IPC 생성 과정

단계 ⓐ에서는 입력된 Cycle-C를 정의된 문법에 맞는지 파악한다. 단계 ⓑ는 파싱 과정에서 제어 흐름 그래프를 생성한다(그림 3). 단계 ⓒ는 생성된 그래프를 분석하여 반복문 내에 사이클의 경계를 나타내는 wait_edge()이 기술되어 있는지 조사한다. wait_edge() 문이 없는 경우는, 시간의 지연 없이 계속 반복됨을 의미하기 때문에 무한루프가 발생한다. 이런 경우를 방지하기 위해 그래프를 분석하고 Cycle-C가 잘못 기술되었음을 알린다.

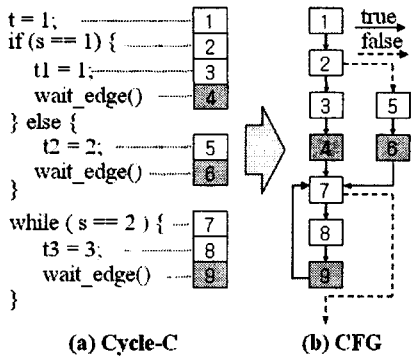

그림 3. CFG의 생성

단계 ⓓ에서는 사용자가 기술한 assert문을 VHDL로 변환하기 위해 제어 흐름 그래프를 재구성한다. 단계 ⓔ에서는 각 문장에 사용자 정의 변수가 정의 되어있는지 찾는다. 이는 변수가 래치로 합성되는 것을 방지하기 위해 한 상태의 각각의 변수 값에 대한 신호를 생성한다. 따라서 흐름 제어 그래프에 변수의 사용여부를 표시한다.

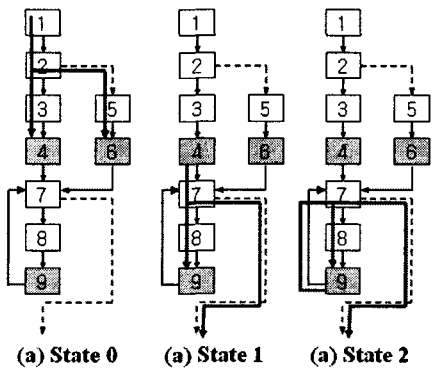

그림 4. State의 생성

단계 ⓕ는 FSM을 생성하는 단계이다. 흐름 제어 그래프의 루트에서 출발하여, wait_edge() 문이 나올 때까지를 하나의 상태로 생성한다. 각 wait_edg() 노드마다 흐름제어 그래프에 따라 상태를 생성한다(그림 4). 단계 ⓖ에서는 생성된 FSM의 각 상태에서 구동하는 신호의 목록을 찾고, 모든 경우에서 신호의 값을 구동하는지 찾아낸다. 만약 모든 조건에서 값을 구동하지 않는 경우가 있다면, 해당 신호는 래치가 생성되므로 이를 방지하기 위해, 해당 신호가 Flip-Flop이 되도록 설정한다. 단계 ⓗ는 변수를 처리하기 위해, 새로운 프로세스를 생성한다. 사용자가 기술한 Cycle-C에 근거하여 VHDL 모듈을 생성하면, 사용자가 정의한 변수는 사용자의 의도와 다르게 동작할 수 있다.

```
entity Utopia_Tx_test is
    port (  TxClk       : in std_logic; /* 포트 */
            rst_n       : in std_logic;
            Data        : in std_logic_vector(7 downto 0);
            TxFulln     : in std_logic              );
end Utopia_Tx_test;
architecture RTL of Utopia_Tx_test is   /* FSM 상태 */
            type StateType is ( S0, S1, S2, S3, S4, S5);
            signal CS, NS           : StateType;
            signal i31 : integer range 0 to 63;
            signal i51 : integer range 0 to 63;
            signal j51 : integer range 0 to 63;
begin                       /* 변수 처리 생성 신호 */
process(TxClk, rst_n)   /* 상태 전이 프로세스 */
begin
            if(rst_n = '1') then
                        CS <= S0;
            elsif  rising_edge(TxClk) then
                        CS <= NS;
            end if;
end process;
process (TxClav, Txfulln, Data, CS) /* 상태결정 프로세스 */
begin
    case CS is
            when S0 =>
                if (TxClav /= '1'  ) then
                        NS <= S1;
                else
                        TxEnbn <= '0' ;
                        TxSOC <= '1' ;
                        Data_delete <= '1' ;
                        NS <= S2;
                end if;
                        :
                        :
end process;
process( CS , i, j, TxClav, Data, TxFulln)  /* 변수처리*/
    variable tmp_i : integer range 0 to 63;
    variable tmp_j : integer range 0 to 63;
begin
    tmp_i := i;     tmp_j := j;
    case CS is
            when S0 =>
                        tmp_i       := 0;
                        tmp_j       := 0;
                        if (TxClav /= '1'  ) then
                        else
                        end if;
                        :
                        :
    n_i <= tmp_i;
    n_j <= tmp_j;
end process;
    TxData <= Data;  /* Concurrent Signal assignment */
end RTL;
```

그림 5. 생성된 VHDL 코드(일부)
- UTOPIA Transmit module

즉 임의의 상태에서 시그널에 값을 할당하고 바로 이어 그 할당된 변수를 사용하는 경우, 사용자는 변화된

변수의 값을 가정하고 있는 것이다. 그러나 VHDL 모듈에서 해당 신호를 변수로 처리하게 되면, 래치의 생성을 막을 수 없기에 시그널로 선언하고, 대신 변수의 값 변화에 따른 여러 시그널을 생성하여 사용자의 의도와 같게 동작하도록 구성하였다. 이때 필요한 시그널을 생성하기 위해 또 다른 프로세스가 필요하다. 단계 ⓘ에서는 흐름 제어 그래프를 근거로 VHDL 모듈을 생성한다.

생성된 RTL VHDL 모듈은 다음의 네 부분으로 구성된다(그림 5). 상태 전이 프로세스, 상태 결정 프로세스, 변수처리 프로세스, Concurrent Signal assignment문이다. 상태 전이 프로세스는 순차회로로 구성되며, 각 시그널의 초기값을 설정하고 내부 FSM의 상태를 전이하는 프로세스이다. 상태 결정 프로세스는, 조합회로로 구성되고 입력 신호와 현재 상태에 근거하여 다음 상태를 결정하고, 해당 신호를 구동하는 프로세스이다. 변수처리 프로세스는 사용자 정의 변수를 처리한다. concurrent signal assignment문은 전달인자를 통과(bypass)하는 역할을 한다.

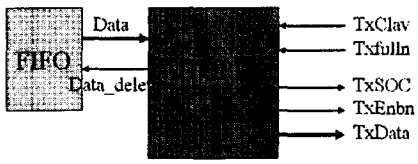

그림 6. Block Diagram(UTOPIA Transmit module)

그림 6은 생성된 UTOPIA Transmit 모듈의 블록다이어그램이다. 전달 데이터는 FIFO 모듈에서 입력받도록 구현되었다.

Ⅳ. 실험

사용자 기술에서 RTL VHDL 코드를 생성하는 생성기는 JavaCC[4]를 이용하여 약 6,800라인의 java 코드로 구현되었다. 표 1은 Cycle-C의 기술과 사용자가 직접 설계한 VHDL 코드와의 길이를 비교한 것이다. 각각의 예는 각 IP의 인터페이스를 설계한 것이다. M(Master)은 IP의 동작을 구동시키는 역할을 하는 인터페이스 회로이고, S(Slave)는 IP의 동작을 인식하는 인터페이스 회로이다. 평균적으로 VHDL에 비해 약 20%의 기술만으로 동일한 설계를 할 수 있음을 보인다.

Ⅴ. 결론 및 향후과제

본 연구에서는 RTL 수준의 VHDL기술보다는 덜 복잡하고, 변경하기 쉬운 알고리즘 기술 언어에 가까운 언어를 제안하였다. Cycle-C를 이용하면 VHDL 기술의 약 10~30% 정도의 기술만으로 동일한 설계가 가능하다. 생성된 결과는 설계자가 직접 RTL 수준으로 설계한 결과와 비슷한 performance(면적, 속도)를 보였다. 알고리즘 수준의 기술이기 때문에, 이해하기 쉽고, 설계를 변경할 때에는 알고리즘 수준에서 쉽게 변경가능하다.

향후 과제로는 Cycle-C를 확장하여, 다중 프로세스와 procedure 호출 등의 기법을 추가할 것이다.

표 1. Cycle-C의 기술과 사용자 설계의 비교

이 름		Cycle-C의 line 수	사용자 설계 VHDL의 line 수	state 수	비교 (%)
DES	M	40	212	6	19
	S	51	303	7	17
PVCI	M	48	151	4	32
	S	53	234	5	23
UTOPIA -TX	M	42	409	7	10
	S	34	401	5	9
UTOPIA -RX	M	48	283	8	17
	S	44	287	11	16
Wishbone	M	47	156	5	30
	S	42	258	6	17
평균 결과		41	270	6~7	20

참고문헌

[1] De Michell, G., Gupta, R.K., "Hardware/software co-design", Proceedings of the IEEE , Volume: 85 Issue: 3 , Mar 1997, Page(s): 349 -365

[2] Sankaran, S., Haggard, R.L., "A convenient methodology for efficient translation of C to VHDL", Southeastern Symposium on System Theory, 2001. Proceedings of the 33rd , Mar 2001, Page(s): 203 -207

[3] Mark Genoe, Paul Vanoostende, Geert van Waewe, "On the use of VHDL-based behavioral synthesis for telecom ASIC design", System Synthesis, 1995., Proceedings of the Eighth International Symposium on , 13-15 Sep 1995, Page(s): 96 -101

[4] TranSwitch Corporation, UTOPIA Interface for the SARA Chipset, Application Note, Document Number TXC-05501-0002-AN, 1.0, 4/11/95.

[5] http://www.suntest.com, "JavaCC Document"