Journal of Korea Multimedia Society Vol. 13, No. 7, July 2010(pp. 1076-1087)

# 클래스-기반 아키텍처 기술 언어의 설계 및 검증

고 광 만†


## 요 약

특정 응용 분야를 위해 개발된 임베디드 프로세서의 진화 및 새로운 출현과 더불어 이를 지원할 수 있는 소프트웨어 개발 환경에 관한 연구와 상용화 시도가 활성화되고 있다. 재목적성(retargetability)은 프로세서나 메모리에 대한 아키텍처 정보를 아키텍처 기술 언어(ADL)로 기술하여 컴파일러, 시뮬레이터, 어셈블러, 프로파일러, 디버거 등과 같은 소프트웨어 개발 도구를 생성하는데 이용된다. EXPRESSION ADL은 아키텍처 모델링, 소프트웨어 개발 도구 생성, 빠른 프로토타입핑, 아키텍처에 대한 설계 탐색과 SoC에 대한 기능적인 검증을 위해 개발된 ADL로서 프로세서 코어, 코프로세서, 메모리 등으로 구성된 소프트웨어적인 아키텍처를 구조와 동작 정보를 혼합하여 자연스럽게 모델링하였다. 이 논문에서는 EXPRESSION ADL을 기반으로 ADL의 작성 편리성, 확장성을 높이기 위해 클래스 기반 ADL을 설계하고 문법의 타당성을 검증하였다. 이를 위해, 6개의 핵심 클래스를 정의하고 MIPS R4000에 대한 ADL을 표현으로부터 EXPRESSION과 동일한 컴파일러, 시뮬레이터를 생성하였다.


# Design and Verification of the Class-based Architecture Description Language

Kwang-Man Ko†


## ABSTRACT

Together with a new advent of embedded processor developed to support specific application area and it evolution, a new research of software development to support the embedded processor and its commercial challenge has been revitalized. Retargetability is typically achieved by providing target machine information, ADL, as input. The ADLs are used to specify processor and memory architectures and generate software toolkit including compiler, simulator, assembler, profiler, and debugger. The EXPRESSION ADL follows a mixed level approach-it can capture both the structure and behavior supporting a natural specification of the programmable architectures consisting of processor cores, coprocessors, and memories. And it was originally designed to capture processor/memory architectures and generate software toolkit to enable compiler-in-the-loop exploration of SoC architecture. In this paper, we designed the class-based ADL based on the EXPRESSION ADL to promote the write-ability, extensibility and verified the validation of grammar. For this works, we defined 6 core classes and generated the EXPRESSION's compiler and simulator through the MIPS R4000 description.


**Key words:** Architecture Description Language(아키텍처 기술 언어), Retargetability(재목적성), Software Development Toolkit(소프트웨어 개발 도구)

※ 교신저자(Corresponding Author) : 고광만, 주소 : 강원도 원주시 우산동 660번지 상지대학교 컴퓨터정보공학부 (220-702), 전화 : 033-730-0486, FAX : 033-730-0480, E-mail : kkman@sangji.ac.kr

접수일 : 2010년 5월 4일, 수정일 : 2010년 7월 7일

완료일 : 2010년 7월 20일

† 상지대학교 컴퓨터정보공학부

※ 이 논문은 2008년도 상지대학교 교내 연구비 지원에 의한 것임.

## 1. 서 론

SoC 기술의 빠른 변경은 임베디드 시스템 프로세서 설계에서 하드웨어 설계 전에 소프트웨어적으로 프로세서의 동작과 성능을 검증하기 위한 시뮬레이터 개발과 더불어 양질의 코드 생성을 위한 최적화된 컴파일러 개발이 신속하게 진행되어야 한다. 임베디드 프로세서 설계에서는 프로세서 및 메모리 구조를 신속하게 탐색하고 평가하는 문제와 시뮬레이터 및 컴파일러와 같은 소프트웨어 개발 도구를 지원하여 코드 질, 실행 속도, 메모리 사용량, 전력 소비 등을 예측하고 개선하는 문제가 중요한 부분이다[1]. 따라서 다양한 프로세서의 출현 및 변화에 대해 컴파일러, 시뮬레이터와 같은 핵심적인 소프트웨어 개발 도구를 효과적으로 생성하기 위해 아키텍처 기술 언어(Architecture Description Language; ADL)를 사용하여 Time-to-Market 및 짧은 라이프-시간 변화에 효율적으로 대처할 수 있는 방안으로 제시되고 있다. ADL은 아키텍처에 대해 프로세서 구조 및 동작 정보를 정형화된 방법으로 기술하여 컴파일러, 어셈블러, 최적화기, 시뮬레이터 등을 생성하는 재목적 기술의 핵심 요소 이다. 따라서 응용 분야에 적합한 프로세서를 위한 컴파일러, 어셈블러, 시뮬레이터 등을 개발할 경우 ADL의 변경만으로 개발 시간과 비용을 줄일 수 있는 중요한 의미를 가지고 있다[2,3].

ADL 표현으로부터 재목적 기술을 적용하여 컴파일러를 생성하는 연구는 LCC[4,5], LISA ADL[6]을 이용하는 LANCE[7]를 주목할 수 있으며 ASIP를 위해서는 MDes[8], EXPRESSION[9,10]이 연구용 및 상용화 제품으로 발표되었다. ADL 표현으로부터 재목적 기술을 적용한 시뮬레이터의 생성은 임베디드 SoC 하드웨어와 소프트웨어의 통합 설계에서 매우 중요한 부분이다. 시뮬레이션의 속도가 빠르고 재목적성을 갖는 시뮬레이터 [11]에서는 시뮬레이터가 탑재되는 시스템의 자원을 적극적으로 사용하여 전통적인 정적 컴파일 시뮬레이션 기법의 성능을 개선하였다. 시스템 자원의 사용은 코드 생성 인터페이스로서 C 언어를 사용하기 보다는 ISA를 위한 저수준의 코드 생성 인터페이스를 특별히 정의하여 사용하였다. ADL에 기반을 둔 빠른 재목적 시뮬레이터의 개발에 관한 연구는 FACILE[12], MIMOLA[13], EXPRESSION SIMPRESS[14]에서 많은 인용과 주목을 받았다.

이 논문에서는 다양한 프로세서의 진화와 출현에 신속하게 대처하고 ADL의 작성 편리성과 확장성을 높이기 위해 LISP-like 형식[9], LISA ADL[6], XML-like 형식[15]을 바탕으로 클래스-기반 ADL(Class-based ADL)을 설계하여 MIPS R4000 프로세서에 대한 실제 표현을 작성하여 타당성을 검증하였다. 이 논문에서 핵심적으로 활용되는 EXPRESSION ADL은 LISA와 더불어 최근까지 연구, 상업용 ADL 설계와 재목적 소프트웨어 개발 도구 생성 분야에서 활발하게 인용되고 있으며 소스가 공개되어 있어 관련 연구에서 적극적으로 활용되고 있다. 이 연구에서 EXPRESSION ADL을 클래스-기반 ADL 구조로 설계하기 위해 전체 ADL의 최상위 클래스인 ArchiDes를 정의한 후 이를 상속받아 아키텍처 동작 정보를 표현하는 Behavior 클래스와 아키텍처 구조 정보를 표현하는 Structure 클래스를 정의하였다. 아키텍처 동작에 대한 구체적인 표현을 위해 Behavior 클래스를 상속받아 OpSpec 클래스, ParSlot 클래스, OpMapping 클래스를 정의하였으며, 아키텍처 구조 및 컴포넌트 표현을 위해 Structure 클래스를 상속받아 Component 클래스, Datapath 클래스, Memory 클래스를 정의하였다. 이 연구에서 설계한 제안한 클래스-기반 ADL을 구현하기 위해 GNU 환경의 Flex[16], Bison[17]을 이용하여 어휘 분석 및 구문 분석을 수행하였으며 문법의 타당성과 생성되는 컴파일러 및 시뮬레이터의 기능을 검증하기 위해 MIPS R4000[18] 프로세서에 ADL을 작성하여 컴파일러와 시뮬레이터를 생성한 후 EXPRESSION 시스템에서 제공한 벤치마킹 프로그램의 실행 결과와 동일한 결과를 생성하였다.

이 논문의 구성은 제2장에서 ADL의 기본적인 특성과 이 논문의 핵심 기반이 되는 EXPRESSION ADL에 대한 고찰을 기술하였다. 제3장에서는 이 논문에서 설계하고 검증한 클래스-기반 ADL에 대한 상세한 소개하였으며 제 4장에서는 구현 및 검증 결과를 설명하기 위해 실제 MIPS R4000 프로세서 대한 ADL의 표현을 소개하였다. 하였다. 제5장에서는 이 논문의 결론과 향후 연구 내용에 대해 기술하였다.

## 2. 아키텍처 기술 언어(Architecture Description Language; ADL)

### 2.1 ADL의 분류

ADL은 아키텍처 명세를 정형화된 언어 형식으로

기술하고 반복된 검증을 통해 아키텍처 설계를 탐색할 수 있으며 컴파일러, 시뮬레이터와 같은 프로세서에 적합한 소프트웨어 개발 도구를 자동 생성할 수 있는 장점을 가지고 있다. 따라서 최근에는 SoC 아키텍처 분야에서 하드웨어·소프트웨어 통합 설계에 적극적으로 활용하고 있다. 최근까지 사용되고 있는 ADL은 표현되는 내용과 목적에 따라 [그림 1]과 같이 구조-ADL, 행위-ADL, 혼합-ADL, 부분-ADL로 분류되고 있다[3,14].

구조-ADL은 특정 분야에 적합한 아키텍처에 대해 구성 요소인 컴포넌트 정의와 컴포넌트간의 관계를 세밀하게 표현할 수 있는 장점으로 인해 하드웨어 설계 분야에 효과적으로 활용되었지만 재목적 소프트웨어 개발 도구에 관한 생성 방법이 단점으로 지적되었으며 대표적으로 MIMOLA[19]가 구조-ADL로서 많은 인용이 진행되었다. 동작-ADL은 인스트럭션-셋 아키텍처(ISA)에 대해 상세한 하드웨어 표현보다는 인스트럭션의 의미와 동작을 명확하게 표현하여 효율적인 재목적 소프트웨어 개발 도구를 생성하는 장점이 부각되었으며 ISDL[20]과 nML[21]이 기반 연구 및 상용화 제품 개발에 활용되었다. 혼합-ADL은 아키텍처의 구조와 동작 정보를 동시에 표현하여 하드웨어 설계 자동화, 소프트웨어 개발 도구 생성, 하드웨어 탐색 및 평가 분야에서 활용되고 있으며 최근에는 임베디드 시스템, SoC 분야에서 중요성과 활용 가치가 매우 강조되고 있다. 혼합-ADL에는 대표적으로 아키텍처의 구조 및 행위 정보를 LISP-like 형태로 기술한 후 재목적 컴파일러와 시뮬레이터를 생성하는 EXPRESSION ADL과 LISA가 대표적으로 인용되고 있다. 특히, LISA는 ISA를 모델링하여 아키텍처의 정보 표현이 가능하도록 고안된 C-like한 문법 구조를 갖는 혼합-ADL로서 Cosy 시스템의 재목적 C 컴파일러[22,23], 고속의 인스트럭션-셋 시뮬레이터 등을 재목적 기술을 통해 생성하는 특징을 가지고 있으며 정확한 컨트롤 경로와 데이터 전송 경로를 기술할 수 있는 장점을 가지고 있다. 국내에서도 SoarDL[24] 설계로부터 실행 속도가 빠른 코드를 생성하는 재목적 컴파일러인 SoarGen[25]을 생성한 연구와 하드웨어 및 소프트웨어 통합 설계를 위한 시뮬레이터 생성에 관한 연구[26]가 선행되었다.

## 2.2 EXPRESSION ADL 구조

EXPRESSION ADL은 LISP-like 형식의 혼합-ADL로서 컴파일러, 시뮬레이터와 같은 소프트웨어 개발 도구를 생성하며 메모리 구조를 계층적으로 정의하고 프로세서 코어, 코프로세서, 메모리를 GUI 방식으로 탐색할 수 있는 프레임워크를 지원하고 있다. EXPRESSION ADL의 구조는 명령어-셋을 기술하는 동작 섹션과 아키텍처 구조 및 구성 컴포넌트를 기술하는 구조 섹션으로 구성되어 있다. 동작 섹션은 (i)오퍼레이션, (ii)인스트럭션, (iii)오퍼레이션 매핑 서브섹션으로 구성되어 있다. 오퍼레이션 서브섹션에서는 프로세서의 인스트럭션-셋을 연산코드, 오퍼란드의 다양한 속성을 고려하여 표현한다. 인스트럭션 서브섹션은 아키텍처에서 가능한 병렬성을 기술하는 부분으로서 각 인스트럭션은 오퍼레이션으로 구성되는 슬롯 리스트를 포함하고 있다. 오퍼레이션 매핑 서브섹션은 컴파일러의 핵심 정보인 인스트럭션 선택과 최적화에 관련된 정보를 포함하고 있다.

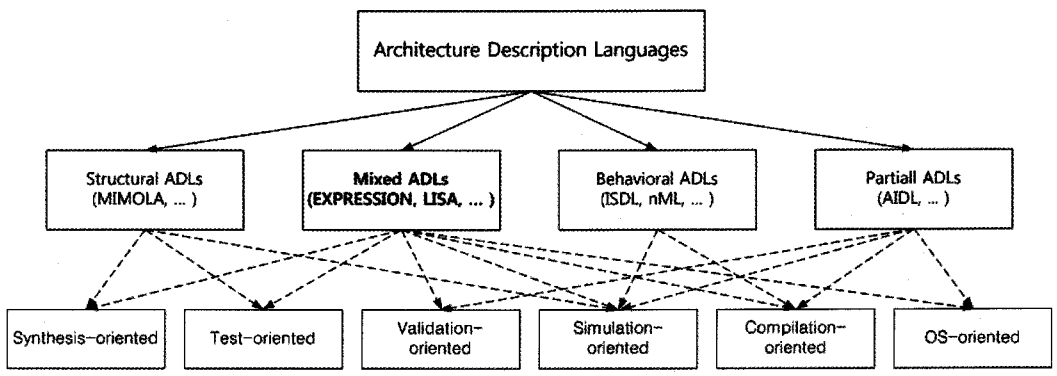


그림 1. ADL 분류

```
//OPERATION_SECTION
(OPERATION_SECTION
  (OPCODE and
    (OP_TYPE DATA_OP)
    (OPERANDS (_SOURCE_1_ int_any) (_SOURCE_2_ int_any) (_DEST_ int_any))
    (BEHAVIOR "_DEST_ = _SOURCE_1_ AND _SOURCE_2_")
    (ASMFORMAT ( ( COND "dst1=reg,src1=reg,src2=reg" )
                 ( PRINT "\t<opcode>\t$<dst1>,$<src1>,$<src2>\n" ) ) )
    (IRDUMPFORMAT ( ( COND "dst1=reg,src1=reg,src2=imm" )
                 ( PRINT "\t4\t<opcode>\t($<dst1>)\t($<src1>,<src2>)\n" ) ) )
  )
)
// ...
// INSTRUCTION_SECTION
(INSTRUCTION_SECTION
  (WORDLEN 32)
  (SLOTS
   ((TYPE DATA) (BITWIDTH 8) (UNIT ALU1_EX))
   ((TYPE DATA) (BITWIDTH 8) (UNIT ALU2_EX))
   ((TYPE DATA) (BITWIDTH 8) (UNIT FALU_EX))
   ((TYPE DATA) (BITWIDTH 8) (UNIT LDST_EX))
   ((TYPE CONTROL) (BITWIDTH 8) (UNIT BR_EX))
  )
)
// ...
// OPMAPPING_SECTION
(TREE_MAPPING
  ( ( GENERIC ( (IADD DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3) ) ) )
    ( TARGET ( (addu DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)) ) ) )
  // ...
)
```

그림 2. EXPRESSION ADL 동작 서브섹션

구조 섹션은 (i)아키텍처 컴포넌트, (ii)파이프라인 및 데이터-전송 통로, (iii)메모리 서브시스템 섹션으로 구성되어 있다. 아키텍처 컴포넌트 서브섹션에서는 파이프라인 유닛, 기능 유닛, 스토리지, 포트와 같은 아키텍처의 컴포넌트와 컴포넌트간의 연결성에 대한 정보를 기술하는 서브섹션이다. 파이프라인 및 데이터-전송 경로 서브섹션에서는 파이프라인 스테이지를 구성하는 각 유닛을 지정하고 올바른 데이터 전송 경로를 지정한다. 이러한 정보는 재목적 시뮬레이터와 스케줄러가 요구하는 예약 테이블의 생성에 활용된다. 메모리 서브시스템 섹션에서는 아키텍처의 다양한 스토리지 컴포넌트에 대한 타입과 속성을 기술한다.

## 3. 클래스-기반 아키텍처 기술 언어(Class-based ADL)

### 3.1 클래스 구조

이 논문에서 설계하고 구현한 클래스-기반 ADL의 특징은 EXPRESSION ADL에서 제안한 6개의 서브섹션을 클래스로 정의하여 아키텍처의 동작 및 구조 정보를 표현하도록 하였다. 6개의 클래스를 중심으로 클래스간의 계층성을 지원하기 위해 아키텍처의 명령어-셋에 관한 정보를 중심으로 동작 정보를 지원하기 위한 Behavior 클래스와 아키텍처를 구성하는 컴포넌트와 컴포넌트간의 연결성을 표현하는 Structure 클래스를 정의하여 전체 클래스가 [그림

```
//ARCHITECTURE_SECTION
(ARCHITECTURE_SECTION
 (SUBTYPE UNIT FetchUnit DecodeUnit OpReadUnit ExecuteUnit BranchUnit LoadStoreUnit WriteBackUnit
)
 (SUBTYPE PORT UnitPort Port)
 (SUBTYPE CONNECTION MemoryConnection  RegisterConnection  RegisterConnection)
 (SUBTYPE STORAGE Storage InstStrLatch PCLatch InstructionLatch OperationLatch)
 (FetchUnit FETCH
  (CAPACITY 1) (INSTR_IN 4) (INSTR_OUT 4) (TIMING (all 1)) (OPCODES all)
  (LATCHES (OUT FetDecLatch) (OTHER pcLatch)) )
)
// ...
//PIPELINE_SECTION
(PIPELINE_SECTION
 (PIPELINE FETCH DECODE READ_EXECUTE WB)
   (READ_EXECUTE (ALTERNATE read_execute0 read_execute1 read_execute2 read_execute3 read_execute4))
     (read_execute0( PIPELINE ALU1_READ ALU1_EX ))
     // ...
 (DTPATHS
  (TYPE UNI
   ( FPRFile ALU1_READ FprReadPort6 FprReadPort6ALU1ReadPort1 Alu1ReadPort1)
   ( FPRFile ALU1_READ FprReadPort7 FprReadPort7ALU1ReadPort2Cxn Alu1ReadPort2)
   //...
  ) ) )
// ...
//STORAGE_SECTION
(STORAGE_SECTION
 (GPRFile (TYPE VirtualRegFile)   (WIDTH 32)   (SIZE 32)  (MNEMONIC "R") )
 (FPRFile (TYPE VirtualRegFile)   (WIDTH 64)   (SIZE 32)   (MNEMONIC "f") )
 //...
)
```

그림 3. EXPRESSION ADL 구조 서브섹션

4]와 같은 계층성을 갖도록 설계하였다.

ArchiDes 클래스는 Behavior 클래스와 Structure 클래스에 대한 최상위 클래스로서 아키텍처의 동작과 구조 정보를 표현하는데 공통적으로 제공되는 정보가 표현된다. Behavior 클래스는 상위 ArchiDes 클래스를 상속받아 아키텍처의 프로세서에 대한 오퍼레이션, 병렬성, 중간코드에 대한 코드 매핑 시 공통적으로 필요한 정보가 정의된다. Behavior 클래스에서는 오퍼레이션 정의 시에 필요한 연산코드, 연산코드 동작, 오퍼란드 정보가 정의되는데 오퍼란드가 갖는 레지스터 속성을 지정하기 위해 (자료형, 레지스터 종류)로 구성된 33가지 레지스터 자료형이 [그림 5]와 같이 정의된다.

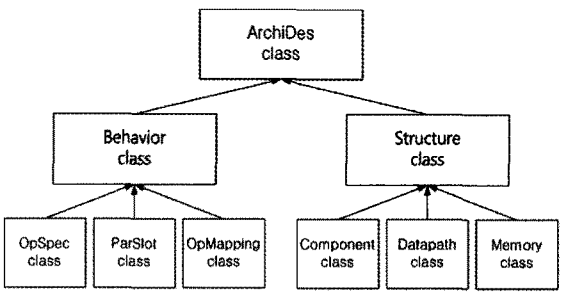


그림 4. 핵심 클래스 및 계층성

Structure 클래스는 아키텍처 구조에 대한 표현을 위해 ArchiDes 클래스를 상속받아 아키텍처의 구조를 컴포넌트 단위로 정의한 후 각 컴포넌트간의 연결성에 관한 정보를 표현하기 위해 [그림 6]과 같이 컴포넌트 형(component type)을 정의하며 컴포넌트의 속성을 정의하는 Component 클래스, 파이프라인과 자료 흐름을 표현하는 Datapath 클래스, 스트로지 속

```
ADLclass Behavior {
    vardef int_normal(DATATYPE INT, REGS
          GPRFile[1-28]) ;
    vardef float_immediate(DATATYPE FLOAT,
          REGS IMM) ;
    vardef double1_retval(DATATYPE DOUBLE,
          REGS FPRFile[0]) ;
    vardef any_sp(DATATYPE INT, REGS SP) ;
        // ...
};
```

그림 5. Behavior 클래스

```
ADLclass Structure {
    CompType UNIT;
    CompType STORAGE;
    CompType Connection;
    CompType PORT;
    // ...
};
```

그림 6. Structure 클래스

성과 스트로지간의 연결성 정보를 표현하는 Memory 클래스에 의해 상속된다.

### 3.2 Behavior 클래스

Behavior 클래스는 프로세서의 인스트럭션-셋, 병렬성 및 데이터 전송 경로, 중간코드에 대한 코드 선택 규칙 및 최적화 정보를 표현하기 위한 공통적인 속성을 정의하고 있으며 OpSpec, ParSlot, OpMapping 클래스에 의해 상속된다. OpSpec 클래스에서는 프로세서의 인스트럭션-셋에 대해 연산코드(OPCODE)와 오퍼란드의 다양한 속성을 정의하기 위해 오퍼레이션을 유사한 기능을 갖는 그룹으로 분류한 후 [그림 7]과 같은 계층성을 갖도록 설계하였다.

OpSpec 클래스는 각 연산 클래스에서 공통적으로 사용하는 필드값과 asmFormat(), irDumpFormat() 멤버 함수를 [그림 8]과 같이 정의하고 있다. 오퍼레이션의 유사한 기능에 따른 6개의 클래스와 ComputeOps 클래스를 상속하는 4개의 연산 클래스의 분류는 MIPS R4000 지침서[27]에 구분된 내용을 따르고 있다.

```
ADLclass OpSpec : Behavior {
    void asmFormat(DstRegType Dst, SrcRegType
         Src1);
    void asmFormat(DstRegType Dst, SrcRegType
         Src1, SrcRegType Src2);
    void irDumpFormat(DstRegType Dst,
         SrcRegType Src1);
    void irDumpFormat(DstRegType Dst,
         SrcRegType Src1, SrcRegType Src2);
         //...
};
```

그림 8. OpSpec 클래스

asmFormat() 멤버 함수는 Dst, Src1, Src2가 갖는 레지스터 자료형(DstRstType, SrcRegType, ... )에 따라 출력되는 어셈블리 코드의 형태를 지정한다. irDumpFormat() 멤버 함수는 Dst, Src1, Src2가 갖는 레지스터 자료형에 따라 실제적인 아키텍처 기술(*.xmd)에 대한 중간 표현을 지정된 형식에 따라 출력한다. 위 멤버 함수는 오버로딩을 적용하여 멤버 함수의 명칭이 일치하더라도 매개 변수의 자료형과

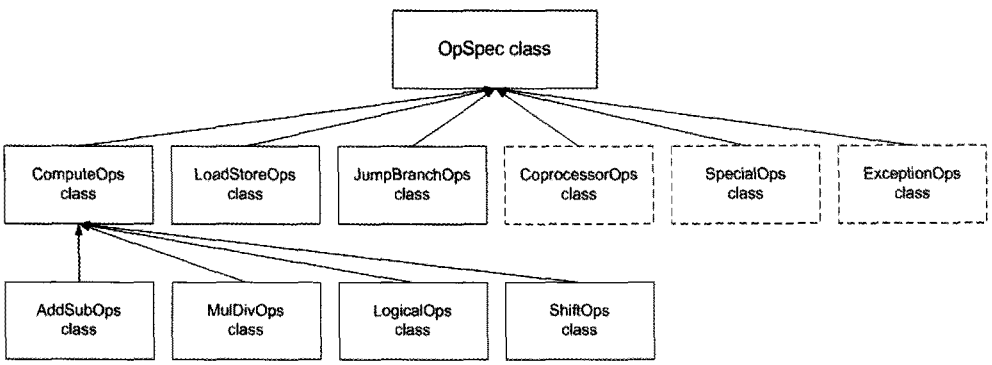


그림 7. OpSpec 클래스의 계층성

개수에 따라 서로 다른 멤버 함수로 구분되어 진다.

ComputeOps 클래스는 독립적인 기능을 갖는 4가지 계산을 위한 공통적인 속성과 멤버 함수를 [그림 9]와 같이 정의하고 있다. 기능에 따라 4가지 계산에 대한 연산코드의 오퍼란드 정보를 지정하는 operand() 멤버 함수는 모든 계산에서 공통적으로 적용되므로 ComputeOps에 정의되며 각 계산 클래스에서 실질적인 계산 동작을 정의한다.

```
ADLclass ComputeOps : OpSpec {
    void operand(SrcRegType1 src1, DstRegType dst);
    void operand(SrcRegType1 src1, SrcRegType2
        src2, DstRegType dst);
        //...
};
```

그림 9. ComputeOps 클래스

각 연산에 대한 클래스 정의는 ComputeOps 클래스를 상속한 후 오퍼레이션의 동작을 표현하는 멤버 함수를 정의한다. ComputeOps 클래스를 상속하여 덧셈과 뺄셈 연산을 정의하는 AddSubOps 클래스는 [그림 10]과 같이 정의한 후 멤버 함수를 구현하였다.

객체는 최하위 클래스인 AddSubOps, MulDivOps, LogicalOps, ShiftOps를 이용하여 생성하며 객체를 이용하여 클래스에 정의된 필드 값과 멤버 함수를 호출한다. 실제로 MIPS R4000의 오퍼레이션 클래스를 정의한 후 객체를 생성하는 과정과 기술 방법은 4.2에서 설명한다.

ParSlot 클래스는 아키텍처에서 가능한 병렬성을 기술하는 부분으로서 병렬로 실행되는 오퍼레이션인 슬롯의 집합으로 구성되어 있다. 각 슬롯은 자료형, 크기, 유닛 이름을 매개변수로 갖는 Slot 클래스에 의해생성된다.

OpMapping 클래스는 중간 코드에 대한 타깃 코드 선택과 타깃 의존적인 최적화를 위한 인스트럭션 매칭(instructionMapping() 멤버 함수) 및 피연산자 매칭(operandMapping() 멤버 함수)을 위해 [그림 12]에 정의된 두 개의 멤버 함수를 이용하여 트리 패턴 매칭을 수행한다.

operandMapping() 멤버 함수를 이용하여 Generic의 피연산자로 표현될 수 있는 데이터 자료형과 레지스터 자료형을 타깃 인스트럭션의 피연산자로 매칭할 수 있는 정보가 기술된다. 예를 들어, Generic의 "(DATATYPE INT) (CLASSTYPE IMM)"은 MIPS R4000 프로세서에서 "int_immediate" 피연산자와 매칭될 수 있는 정보가 operandMapping( INT, IMM, int_immediate) 멤버 함수로 표현한다. 실질적인 인스트럭션 매칭 정보를 표현하기 위해서 instructionMapping( "IADD DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)", "addu DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)")에 기술된 정보는 트리 패턴 매칭을 수행하는 모듈에 전달된다.

```
ADLclass AddSubOps : ComputeOps {
    int_any add(DstRegType dst, SrcRegType src1, SrcRegType src2);
    int_any sub(DstRegType dst, SrcRegType src1, SrcRegType src2);
    // ...
};
int_any AddSubOps::add(DstRegType dst, SrcRegType src1, SrcRegType src2) {
    dst = src1 + src2;
}
```

그림 10. ComputeOps 클래스

```
ADLclass ParSlot : Behavior {
    void Slot(DATATYPE DataType, int Bitwidth, MyString Unit);
};
```

그림 11. ParSlot 클래스

```
ADLclass OpMapping : Behavior  {
    void operandMapping( DataType dataType, RegType regType, TargetType TgtType);
    void instructionMapping( MyString genericInstruction, Mystring targetInstruction);
};
```

그림 12. OpMapping 클래스

```
ADLclass Component : Structure {
        SUBTYPE UNIT FetchUnit, DecodeUnit, OpReadUnit, ExecuteUnit ;
        SUBTYPE PORT UnitPort, Port ;
        SUBTYPE CONNECTION MemoryConnection, RegisterConnection, RegisterConnection ;
        SUBTYPE STORAGE Storage, InstStrLatch, PCLatch, InstructionLatch, OperationLatch ;
        void capacityAttribute( int );
            void instrInOutAttribute( int, int );
        void timingAttribute( MyString, int );
        void opcodeAttribute( MyString );
        void latchAttribute( MyString, MyString );
        // ...
};
```

그림 13. Component 클래스

### 3.3 Structure 클래스

Structure 클래스는 아키텍처 구조에 대한 표현을 위해 ArchiDes 클래스를 상속받아 아키텍처의 구조를 컴포넌트 단위로 정의한 후 각 컴포넌트간의 연결성에 관한 정보를 표현하기 위해 아키텍처 컴포넌트의 속성을 정의하는 Component 클래스, 파이프라인과 자료 흐름을 표현하는 Datapath 클래스, 메모리 속성과 각 메모리간의 연결성 정보를 표현하는 Memory 클래스에 의해 상속된다.

Component 클래스는 RT-수준의 아키텍처 컴포넌트 정보를 정의하는 클래스로서 파이프라인 유닛, 스트로지, 버스 등의 구성 요소와 각 요소가 갖는 속성을 [그림 13]과 같이 정의한다. 아키텍처의 컴포넌트 선언시 서브 컴포넌트를 지정하기 위해서는 SUBTYPE 키워드와 컴포넌트 이름(UNIT, PORT, CONNECTION, STORAGE, ...)을 기술한 후 사용자 정의 명칭에 따라 서브컴포넌트 이름을 기술하여 각 컴포넌트의 속성을 지정하기 위해 정의된 멤버 함수를 이용한다.

Component 클래스를 정의하고 FetchUnit형의 FETCH 컴포넌트 객체를 생성한 후 [그림 14]와 같이 속성을 지정한다.

Datapath 클래스는 아키텍처를 구성하는 컴포넌트간의 데이터 전송 방향과 경로에 대한 정보를 [그림 15]와 같은 클래스를 통해 정의한다.

컴포넌트간의 데이터 흐름을 지정하기 위해서는 DataPath 클래스를 이용하여 dataPathObj와 같은 객체를 생성한 후 dataPathObj.UniDataParh(RFA, X_bus, "C1") 멤버 함수를 이용하며 하나 이상의 경로를 지정할 수 있도록 설계되어 있다.

Memory 클래스는 아키텍처의 구성 요소인 스트로지 정보를 정의하기 위해 [그림 16]과 같은 멤버 함수를 이용한다. 스토로지형은 REGFILE, SRAM, DRAM CACHE 등이 지정되며 각각의 크기가 2, 64, 512로 지정된다. 특히, 주소 분할을 통해 복잡한 구조를 갖는 스트로지의 속성을 표현할 수 있도록 addrr-RangeAttribute() 멤버 함수를 이용한다. Memory RFA와 같이 레지스터형의 스트로지를 생성한 후 속성을 지정하기 위해 RFA.typeAttribute( "REGFILE" ), RFA.sizeAttribute(2), RFA.widthAttribute(24) 등과 같이 표현한다.

## 4. 구현 및 검증

### 4.1 개발 환경 및 구현

이 논문에서는 ADL의 작성 편리성, 구조화된 설계, 확장성을 위해 EXPRESSION ADL 정보를 바탕으로 클래스-기반의 계층적 구조를 갖도록 설계하였다. 새로 설계된 문법은 Flex, Bison 도구를 이용하여

```
FetchUnit FETCH ;
    FETCH.capacityAttribute(1), FETCH.INSTR(4, 4),
    FETCH.timingAttribute(all, .1), FECTH.opcodesAttribute(all);
    // ...
```

그림 14. 컴포넌트 객체 생성 및 속성 지정

```
ADLclass Datapath : Structure {
    void UniDataPath( MyString Direction, MyString sre, MyString sink, MyString actPath ) ;
    void BiDataPath( MyString Direction, MyString src, MyString sink, MyString actPath ) ;
};
```

그림 15. Datapath 클래스

```
ADLclass Memory : Structure {
  void typeAttribute( MyString memType );
  void sizeAttribute( int memSize );
  void widthAttribute( int memWidth ) ;
  void addrrRangeAttribute( int addrLow, int addHigh );
  void accessTimeAttribute( int accTime );
  // ...
};
```

그림 16. Memory 클래스

개발하고 검증하였으며 최초에 EXPRESSION이 개발된 환경(Visual C++ 6.1/Win XP)을 개선하여 Visual C++ 2008/Win Vista에서 개발하고 검증하였다. 클래스-기반 ADL로 작성된 입력(mips4000.cadl)에 대해 어휘 분석, 구문 분석 과정을 거쳐 EXPRESSION의 컴파일러와 시뮬레이터 생성에 필요한 정보를 생성하는 구현 과정은 [그림 17]과 같다. 새로 설계된 문법에 대해 어휘 및 구문 분석을 수행하고 그 결과를 LCC DAG[4] 구조의 중간 표현으로 생성한 후 툴킷 디스크립션 생성기를 통해 EXPRESSION의 컴파일러와 시뮬레이터 생성에 필요한 정보를 저장한다.

### 4.2 검증 및 평가

클래스-기반으로 설계된 ADL의 검증을 위해 EXPRESSION에서 최초에 공개한 MIPS R4000에 대한 기술을 새롭게 설계된 문법 형식으로 설계하고 동일한 컴파일러와 시뮬레이터를 생성한 후 벤치마킹을 위해 활용되었던 어플리케이션의 중간 표현(LL1~LL24.defs, procs)을 이용하여 검증하였다. 실

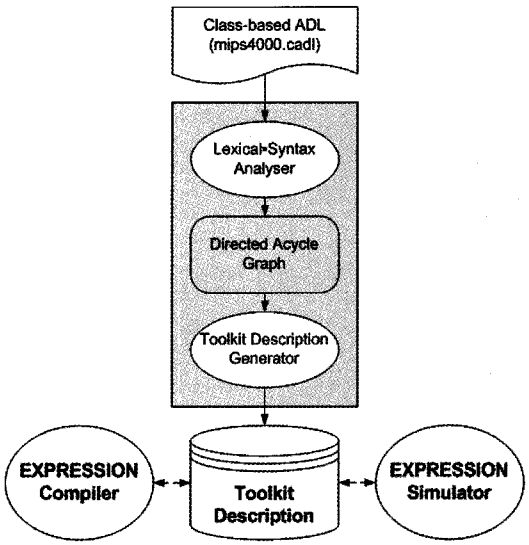


그림 17. 클래스-기반 ADL로부터 Toolkit 생성

제로 [그림 2, 3]에서 간략하게 표현하였던 EXPRESSION ADL에 대해 [그림 18]과 같이 클래스-기반 ADL로 표현하였다.

이 논문의 최초 의도는 ADL 문법 설계의 개선으로서 새롭게 설계된 문법에 대한 검증을 위해 기술된 ADL 입력으로부터 컴파일러, 시뮬레이터를 생성한 후 어플리케이션에 대해 정상적인 동작과 올바른 실행 결과의 생성을 확인하였으며 이 연구를 통해, EXPRESSION ADL보다 문법의 크기를 축소하였고 실제로 MIPS R4000 표현에 대한 입력의 크기가 aces-MIPS.xmd(107.kb)에서 42% 감소된 mips4000. cadl (45kb)을 확인하였다. 이를 통해, ADL 입력을 보다 쉽게 작성하는데 기여하였으며 컴포넌트의 추가, 삭제가 기존 방식보다 편리하게 처리되는 장점을 가지고 있다.

```
//OpSpec 클래스
ShiftOps AndOps; // "and" 연산자를 위한 객체 생성
AndOps.operand(int_any, int_any, int_any);
AndOps.and(int_any dst, int_any src1, int_any src2) ;
AndOps.asmFormat(reg dst, reg src1, imm src2);
AndOps.irDumpFormat(reg dst, reg src1, imm src2);

//ParSlot 클래스
ParSlot pslot; //객체 생성
pslot.Slot(DATA, 8, "ALU1_EX");
pslot.Slot(DATA, 8, "ALU2_EX");
// ...

//OpMapping 클래스
OpMapping mapping;  //객체 생성
mapping.instructionMapping("IADD DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)",
                           "addu DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)");

//Component 클래스
FetchUnit FETCH;  //객체 생성
FETCH.capacityAttribute(1), FETCH.INSTR(4, 4), FETCH.timingAttribute(all, .1) ;
// ...

// Datapath 클래스.
Datapath dataPathObj;  // 객체 생성
dataPathObj.UniDataPath(FPRFile, ALU1_READ
                        "FprReadPort6 FprReadPort6ALU1ReadPort1 Alu1ReadPort1") ;
dataPathObj.UniDataPath(FPRFile, ALU1_READ,
                        "FprReadPort7 FprReadPort7ALU1ReadPort2Cxn Alu1ReadPort2") ;
//...

//Memory 클래스
Memory GPRFile, FPRFile;
GPRFile.typeAttribute("VirtualRegFile"), GPRFile.widthAttribute(32), GPRFile.sizeAttribute(32);
FPRFile.typeAttribute("VirtualRegFile"), FPRFile.widthAttribute(64), FPRFile.sizeAttribute(32);
 //...
)
```

그림 18. MIPS R4000에 대한 클래스-기반 ADL 표현

## 5. 결론 및 향후 연구방향

최근 SoC 환경에서 다양한 프로세서의 출현 및 변화에 대해 컴파일러, 시뮬레이터와 같은 핵심적인 소프트웨어 개발 도구를 효과적으로 생성하기 위해 아키텍처 기술 언어(Architecture Description Language; ADL)를 사용함으로써 Time-to-Market 및 짧은 라이프-시간 변화에 효율적으로 대처할 수 있는 방안으로 제시되고 있다. ADL로부터 재목적 기술을 적용하여 컴파일러 및 시뮬레이터를 생성하는 LISA, EXPRESSION ADL이 연구 또는 상업용 소프트웨어 개발 툴킷 생성기의 기반으로 폭넓게 활용되고 있다. 이 논문에서는 다양한 프로세서의 진화와 출현에 신속하게 대처하고 ADL의 작성 편리성과 확장성을 높이기 위해 EXPRESSION ADL을 핵심 기반으로, LISA 등의 연구 내용을 참고하여 클래스-기반 ADL을 설계하고 MIPS R4000 프로세서에 대한 실제 표현을 작성하여 타당성을 검증하였다. 이 논문

에서 설계하고 구현한 클래스-기반 ADL의 특징은 EXPRESSION ADL에서 제안한 6개의 서브섹션을 클래스로 정의하여 아키텍처의 동작 및 구조 정보를 표현하도록 하였다. 6개의 클래스를 중심으로 클래스간의 계층성을 지원하기 위해 아키텍처의 명령어-셋에 관한 정보를 중심으로 동작 정보를 지원하기 위한 Behavior 클래스와 아키텍처를 구성하는 컴포넌트와 컴포넌트간의 연결성을 표현하는 Structure 클래스를 정의하여 전체 클래스가 계층성을 갖도록 설계하였다. Behavior 클래스는 프로세서의 인스트럭션-셋, 병렬성 및 데이터 전송 경로, 중간코드에 대한 코드 선택 규칙 및 최적화 정보를 표현하기 위한 공통적인 속성을 정의하고 있으며 OpSpec, ParSlot, OpMapping 클래스에 의해 상속된다. Structure 클래스는 아키텍처 구조에 대한 표현을 위해 ArchiDes 클래스를 상속받아 아키텍처의 구조를 컴포넌트 단위로 정의한 후 각 컴포넌트간의 연결성에 관한 정보를 표현하기 위해 아키텍처의 컴포넌트 속성을 정의하는 Component 클래스, 파이프라인과 자료 흐름을 표현하는 Datapath 클래스, 메모리 속성과 각 메모리간의 연결성 정보를 표현하는 Memory 클래스에 의해 상속된다. 클래스-기반으로 설계된 ADL의 검증을 위해 EXPRESSION에서 최초에 공개한 MIPS R4000에 대한 기술을 새롭게 설계된 문법 형식으로 설계하고 동일한 컴파일러와 시뮬레이터를 생성한 후 벤치마킹을 위해 활용되었던 어플리케이션의 중간 표현을 이용하여 검증하였다.

현재, 클래스-기반 ADL로부터 컴파일러와 시뮬레이터에 필요한 정보를 생성하는 현재의 연구 방법을 보완하여 직접 컴파일러, 어셈블러, 디버거, 시뮬레이터를 생성할 수 있도록 클래스-기반 ADL을 보완중이며 다양한 아키텍처에 대한 ADL 표현이 가능하도록 추가적인 연구를 지속적으로 진행할 예정이다.

## 참 고 문 헌


[1] Rainer Leupers, "Code Generation for Embedded Processor," ISSS;00: 13th International System Synthesis Symposium, pp.173~178, 2000.

[2] A. Hoffman, H. Meyr, and R. Leupers, *Architecture Exploration for Embedded Processors with LISA*, Kluwer Academic Publishers(ISBN: 1-4020-7338-0), 2002.

[3] Prabhat Mishra, and Nikil Dutt, *Processor Description Languages*, Morgan Kaufmann, 2008.

[4] C. W. Fraser, and D. R. Hanson, *A Retargetable C Compiler: Design and Implementation*, Addison-Wesley, 1995.

[5] LCC, A Retargetable Compiler for ANSI C: http://www.cs.princeton.edu/software/lcc/

[6] Anupam Chattopadhyay, Heinrich Meyr, and Rainer Leupers, "LISA: A Uniform for Embedded Processor Modeling, Implementation, and Software Toolsuite Generation," Processor Description Languages: Chap. 5, Morgan Kauffman, 2008.

[7] M. Hohenauer, et. al., "A Methodology and Tool Suite for C Compiler Generation from ADL Processor Models," *Design and Automation & Test in Eupore(DATE)*, 2004.

[8] J. C. Gyllenhaal et. al., The Mdes User Manual, Technical Report, Trimaran Release: http://www.trimaran.org, 1999.

[9] Peter Grun, Ashok Halambi, Vijay Ganesh, Nikil Dutt, and Alex Nicolau, "EXPRESSION: An ADL for System Level Design Exploration," Technical Report TR98-29, University of California Irvine, 1998.

[10] Nikill Dutt et. al., "EXPRESSION: A Language for Architectural Exploration through Compiler/Simulator Retargetability," DATE'99: Proceedings of the 1999 Design, Automation and Test in Europe Conference, 1999.

[11] Jianwen Zhu, and Daniel D. Gajski, "A Retargetable, Ultra-fast Instruction Set Simulator," DATE'99: Proceedings of the Design Automation and Test conference in Europe, 1999.

[12] Eric C. Schnarr, Mark D. Hill, and James R. Larus, "Facile: A Language and Compiler for High-Performance Processor Simulators,"

PLDI'99: Proceedings of the ACM SIGPLAN 1999 conference on Programming Language Design and Implementation, pp.1~11, 1999.

[13] R. Leupers, J. Elste, and B. Landwehr, "Generation of Interpretive and Compiled Instruction Set Simulators," ASP-DAC'99: Proceeding of the Asia South Pacific Design Automation Conference, 1999.

[14] Prabhat Mishra, Aviral Shrivastava, and NiKill Dutt, "Architecture Description Language-driven Software Toolkit Generation for Architectural Exploration of Programmable SOCs," *ACM Transactions on Design Automation of Electronics Systems*, Vol.11, No.2, pp. 626~658, 2006.

[15] Florian Brander, Dietmar Ebner, and Andreas Krall, "Compiler Generation from Structural Architecture Descriptions," CASES'07: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pp. 13~22, 2007.

[16] Flex, http://dinosaur.compilertools.net/flex/index.html

[17] Bison, http://www.gnu.org/software/bison/manual/

[18] Joe Heinrich, MIPS R4000 Microprocessor User's Manual 2nd Edition, MIPS Technologies, Inc., 1994.

[19] Peter Marwedel, MIMOLA-A Fully synthesizable Language in Processor Description Languages (Editor: Prabhat Mishra, Nikil Dutt), Morgan Kaufmann, pp. 35~63, 2008.

[20] G. Hadjiyiannis, S. Hanano, and S. Devadas, "ISDL: An Instruction Set Description Language for Retargetability," DAC'99: In Proceedings of Design Automation Conference, pages 299~302, 1997.

[21] nML Description of the TCT Core Processor: http://www.retarget.com/nml, March, 2008.

[22] O. Wahlen, M. Hohenauer, R. Leupers, and H. Meyr, "Instruction Scheduler Generation for Retargetable Compilation," *IEEE Design & Test of Computers*, pp. 34~41, 2003.

[23] J. Ceng, M. Hohenauer, R. Leupers, G. Ascheid, H. Meyer, and G. Braun, "C Compiler Retargeting Based on Instruction Semantics Models," DATE'05: In Proceedings of the Conference on Design, Automation and Test in Europe, pages 1150~1155, 2005.

[24] Minwook Ahn, Yunheung Paek, "A New ADL-based compiler for Embedded Processor Design," International SoC Design Conference, 2005.

[25] Minwook Ahn, SoarGen: A User Retargetable Compiler in the Design of Embedded Systems, Ph. D. Thesis, Seoul National University, 2009.

[26] S. Y. Hwang, and S. R. Lee, "Construction of a Retargetable Compiler Generation System from Machine Behavioral Description," *Journal of Korean Institute of Communication Sciences*, Vol.32, No.5, 2007.

고 광 만

1991년 2월 원광대학교 컴퓨터공학과(공학사)
1993년 2월 동국대학교 컴퓨터공학과(공학석사)
1998년 2월 동국대학교 컴퓨터공학과(공학박사)
1998년 3월~2001년 8월 광주여자대학교 컴퓨터과학과 전임강사
방문연구 : QUT(2003, 호주), UQAM(2008, 캐나다), UC Irvine(2010, 미국)
2001년 9월~현재 상지대학교 컴퓨터정보공학부 부교수
관심분야 : 프로그래밍언어론 및 컴파일러