ADL Synthesis using ArchC

Samuel Goto (goto at google.com)

February 22, 2010

Outline

What?

- Synthesize processors from architecture description languages.

What?

- Synthesize processors from architecture description languages.
- ADL $\Rightarrow$ SDL $\Rightarrow$ HDL $\Rightarrow$ hardware.

What?

- Synthesize processors from architecture description languages.
- ADL $\Rightarrow$ SDL $\Rightarrow$ HDL $\Rightarrow$ hardware.
- ArchC $\Rightarrow$ SystemC $\Rightarrow$ Verilog $\Rightarrow$ FPGA.

Why?

- Provide a concrete and real life processor

Why?

- Provide a concrete and real life processor
- Give feedback to designers about area, power and speed

Why?

- Provide a concrete and real life processor
- Give feedback to designers about area, power and speed
- Provide faster simulations

How?

- Providing the ArchC RTL Specification

How?

- Providing the ArchC RTL Specification
- Building a generator: acrtl
  - acsim generates a Behavioral skeleton binding all resources
  - acrtl does the same, but in RTL

How?

- Providing the ArchC RTL Specification
- Building a generator: acrtl
  - acsim generates a Behavioral skeleton binding all resources
  - acrtl does the same, but in RTL
- Creating a library of RTL Resources

Outline

What is ArchC

- ArchC is an architecture description language, based on SystemC.

What is ArchC

- ArchC is an architecture description language, based on SystemC.
- The simulator is 100% SystemC.

What is ArchC

- ArchC is an architecture description language, based on SystemC.
- The simulator is 100% SystemC.
- Automatic generation of tools: assemblers, memory hierarchies, simulators, etc.

What is ArchC

- ArchC is an architecture description language, based on SystemC.
- The simulator is 100% SystemC.
- Automatic generation of tools: assemblers, memory hierarchies, simulators, etc.
- Existing models include: PowerPC, MIPS-1, Sparc V8, Intel 8051, Java JVM, etc.

AC_ARCH

```
AC_ARCH(mips){
  ac_wordsize 32;
  ac_mem MEM:256K;

  ac_pipe pipe = {IF, ID, EX, MEM, WB};

  ac_format Fmt_IF_ID = "%npc:32";
  ac_format Fmt_MEM_WB = "%wbdata:32 %rdest:5 %regwrite:1";
  ac_reg<Fmt_IF_ID> IF_ID;
  ac_reg<Fmt_MEM_WB> MEM_WB;

  ARCH_CTOR(mips) {
    ac_isa("mips_isa.ac");
    set_endian("big");
  };
};
```

AC_ISA

```
AC_ISA(mips){
  ac_format Type_R = "%op:6 %rs:5 %rt:5 %rd:5 0x00:5 %func:6";
  ac_format Type_I = "%op:6 %rs:5 %rt:5 %imm:16:s";

  ac_instr<Type_R> add, sub, instr_and, instr_or, mult, div;
  ac_instr<Type_I> lw, sw, beq, bne;

  ISA_CTOR(mips){
    lw.set_asm("lw %reg, %imm(%reg)", rt, imm, rs);
    lw.set_decoder(op=0x23);

    sw.set_asm("sw %reg, %imm(%reg)", rt, imm, rs);
    sw.set_decoder(op=0x2B);
  };
};
```

ac_behavior

```
void ac_behavior( add, stage ){
  switch(stage) {
  case IF:
    IF_ID.npc = ac_pc + 4;
    break;
  case EX:
    EX_MEM.alu_result = ID_EX.rs + ID_EX.rt;
    break;
  case MEM:
    MEM_WB.alu_result = EX_MEM.alu_result;
    MEM_WB.rd = EX_MEM.rd;
    break;
  case WB:
    RB.write(MEM_WB.rd, MEM_WB.alu_result);
    break;
  }
};
```

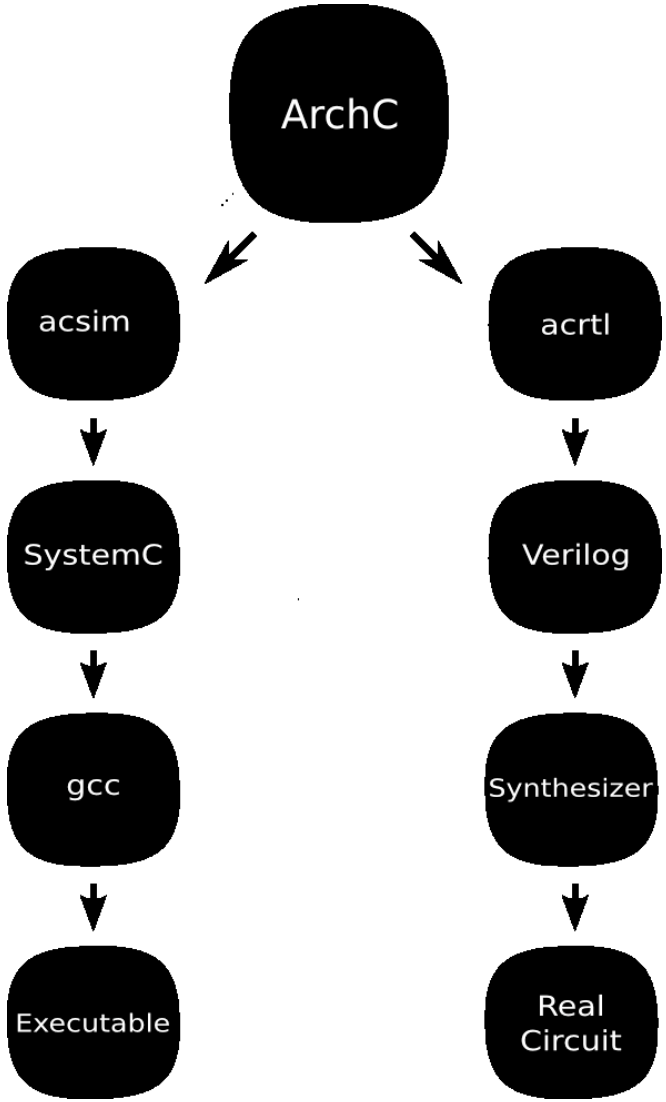

ArchC
acsim
acrtl
SystemC
Verilog
gcc
Synthesizer
Executable
Real
Circuit

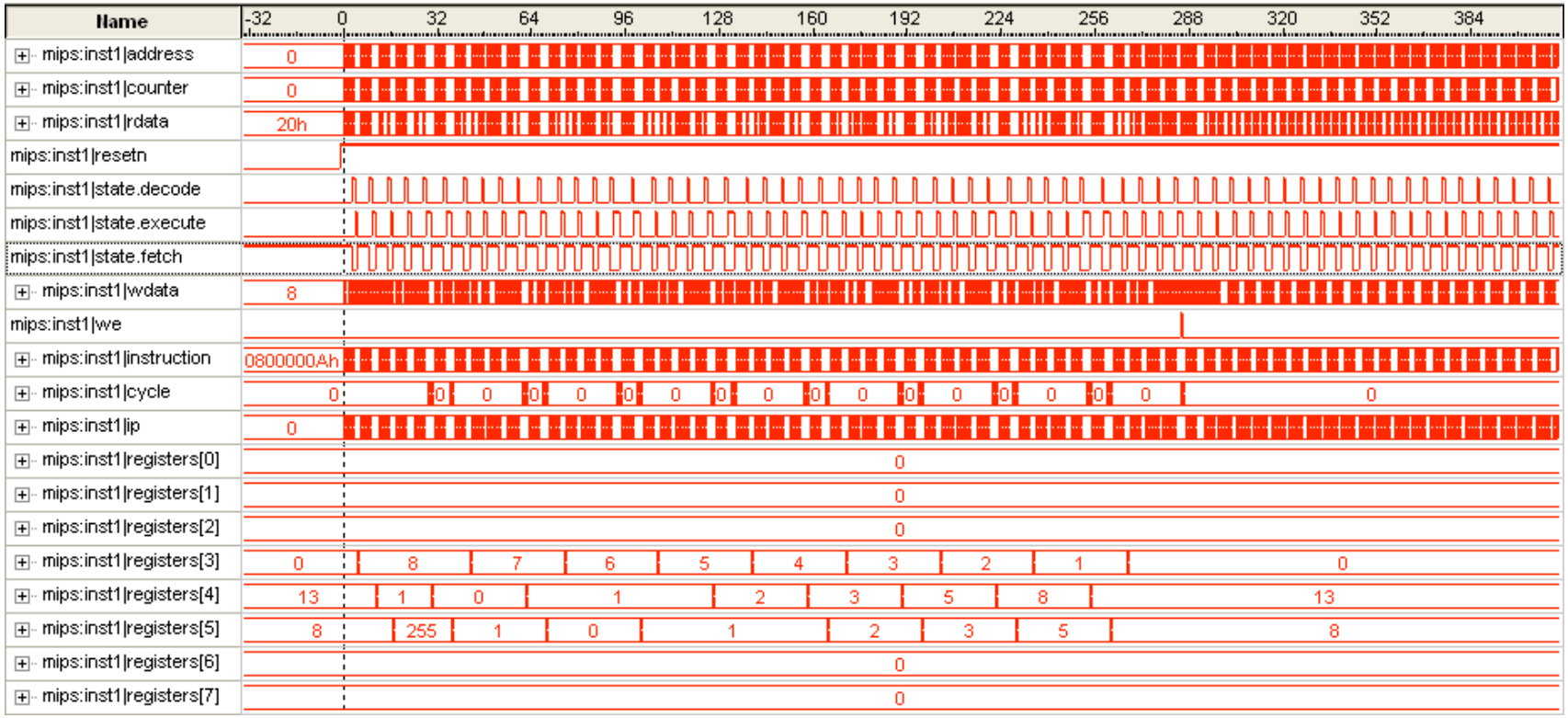
Table: DLX based experimental ISA

Type of operation	Instructions
load/store	sb, lb
arithmetic	add, sub, addi
bit logic	and8, or8
flow control	beq, jmp
other	slt

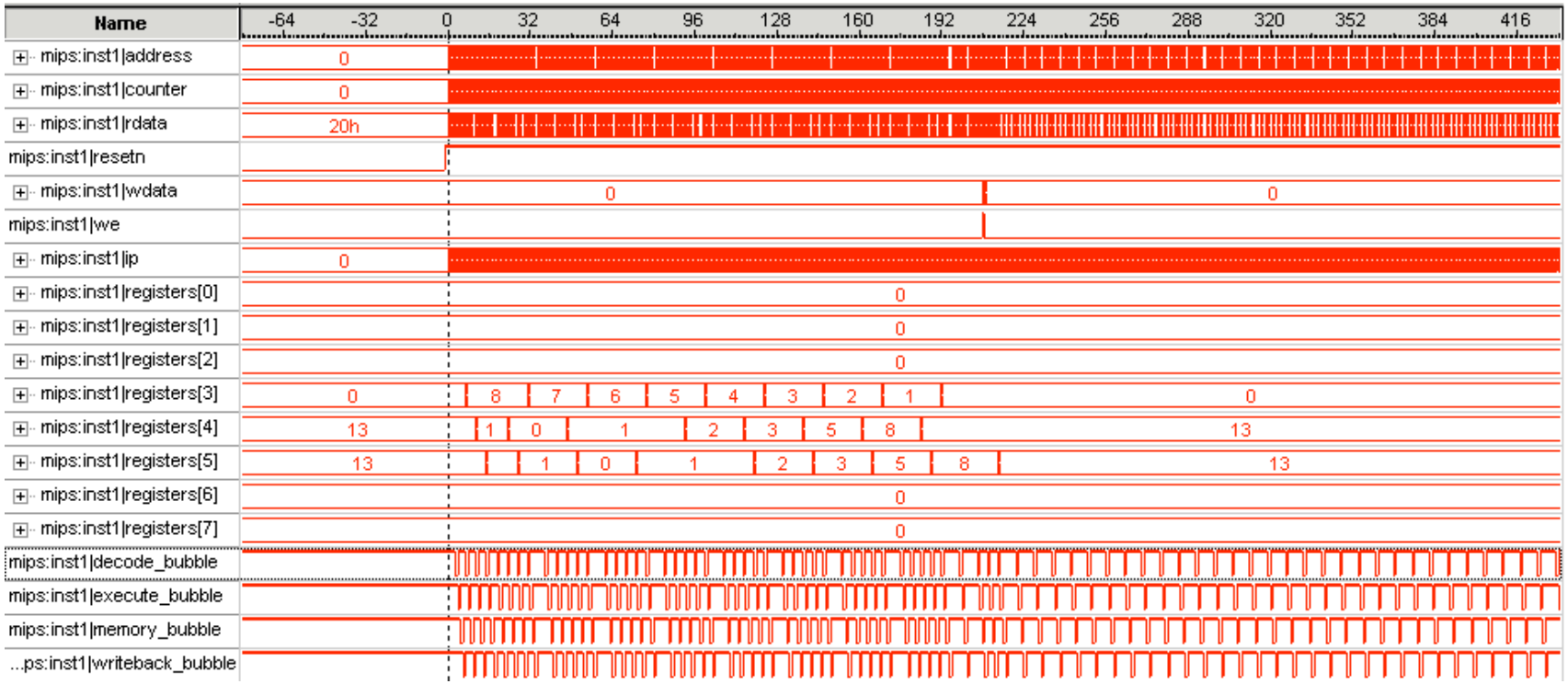
Table: Clock frequency and used area prototyped in an Altera device

Architecture	Clock frequency (Mhz)	Logic Elements
Monocycle	60.30	1,780
Multicycle	78.78	1,749
Pipeline	107.52	1,738

Multicycle signal tap



Pipeline signal tap



How does it work?

- ArchC RTL Synthesis is almost free: just run acrtl instead of acsim.

How does it work?

- ArchC RTL Synthesis is almost free: just run acrtl instead of acsim.
- Ok, ok, small changes are needed. But they are really small. And, in fact, often times you do want to change them.

How does it work?

- ArchC RTL Synthesis is almost free: just run acrtl instead of acsim.
- Ok, ok, small changes are needed. But they are really small. And, in fact, often times you do want to change them.
- How is it done?

How does it work?

- ArchC RTL Synthesis is almost free: just run acrtl instead of acsim.
- Ok, ok, small changes are needed. But they are really small. And, in fact, often times you do want to change them.
- How is it done?
- Two compiler optimization techiniques:

How does it work?

- ArchC RTL Synthesis is almost free: just run acrtl instead of acsim.
- Ok, ok, small changes are needed. But they are really small. And, in fact, often times you do want to change them.
- How is it done?
- Two compiler optimization techiniques:
  - Functions inlining

How does it work?

- ArchC RTL Synthesis is almost free: just run acrtl instead of acsim.
- Ok, ok, small changes are needed. But they are really small. And, in fact, often times you do want to change them.
- How is it done?
- Two compiler optimization techiniques:
  - Functions inlining
  - Constants folding and propagation: Dead code removal guarantees single driver.

before: ac_behavior

```
void ac_behavior( add, stage ){
  switch(stage) {
  case IF:
    IF_ID.npc = ac_pc + 4;
    break;
  case EX:
    EX_MEM.alu_result = ID_EX.rs + ID_EX.rt;
    break;
  case MEM:
    MEM_WB.alu_result = EX_MEM.alu_result;
    MEM_WB.rd = EX_MEM.rd;
    break;
  case WB:
    RB.write(MEM_WB.rd, MEM_WB.alu_result);
    break;
  }
};
```

after: inlined and code removed

```
SC_MODULE(processor) {
  void EXE_fsm() {
    if(resetn){
    } else {
      switch(EX.opcode){
        case ADD: {
          EX_MEM.alu_result = ID_EX.rs + ID_EX.rt;
        }
      }
    }
  }
  SC_CTOR(processor) {
    SC_METHOD(EXE_fsm);
    sensitive << clk << resetn;
  }
}
```

function inlining: before

```
void ac_behavior(add, cycle){
  switch(cycle):{
    case 0:
      result = regs[rs] + regs[rd]; break;
    case 1:
      regs[rd] = result; break;
  }
}
void fsm(){
  if(resetn){
  } else {
    switch(opcode){
      case ADD: {ac_behavior(add, cycle); cycle++;}
    }
  }
}
```

function inlining: after

```
void fsm(){
  if(resetn){
  } else {
    switch(opcode){
      case ADD: {
        switch(cycle):{
          case 0:
            result = regs[rs] + regs[rd]; break;
          case 1:
            regs[rd] = result; break;
        }
        cycle++;
      }
    }
  }
}
```

constant folding: before

```
void add(int stage){
  switch(stage):{
    case EX:
      result = regs[rs] + regs[rd]; break;
    case WB:
      regs[rd] = result; break;
  }
}
void WB_fsm(){
  if(resetn){
  } else {
    switch(opcode){
      case ADD: {add(WB); break;}
    }
  }
}
```

constant folding: after

```
void WB_fsm(){
  if(resetn){
  } else {
    switch(opcode){
      case ADD: {
            regs[rd] = result; break;
      }
    }
  }
}
```

constant folding: after

```
void EX_fsm(){
  if(resetn){
  } else {
    switch(opcode){
      case ADD: {
            result = regs[rs] + regs[rd]; break;
      }
    }
  }
}
```

Outline

Existing processors and architecture spectrum

- 8051
- PIC
- r3000
- JVM
- out of order execution
- behavioral synthesis

ADL Synthesis using ArchC

Samuel Goto (goto at google.com)

February 22, 2010