Chapter 4 Building a Simulator

BSVC provides a framework of C++ classes for writing microprocessor simulators. Even though BSVC provides the user interface and basic layout of the simulator, the simulator designer still has a lot of work to do and many decisions to make. This section provides an in-depth view of building a simulator using the BSVC framework.

Creating a Development Directory

The first step in writing a simulator is to setup a directory structure to do development work in. The basic directory layout that's been used thus far and should be suitable for most simulators is:

Sim*Processor*/	Top level simulator directory
cpu/	Subdirectory to hold code to simulate the microprocessor
devices/	Subdirectory to hold code to simulate any devices
loaders/	Subdirectory to hold code to load object files

Sim*Processor* Directory

The Sim*Processor* directory usually contains one C++ source file named main.cxx and a make file named Makefile. The main.cxx file contains the main function that usually creates all the necessary objects and turns control over to an Interface object. The make file usually generates an executable named Sim*Processor*.

cpu Subdirectory

This directory usually contains several C++ source files, and a make file. One of the C++ source files is the microprocessor class derived from the `BasicCPU` class. The make file usually creates an archive named `libcpu.a` which is linked with the simulator.

device Subdirectory

This directory usually contains several C++ source files, a make file, and several Tcl/Tk device scripts. The make file usually creates an archive named `libdevice.a` which is linked with the simulator.

loader Subdirectory

This directory usually contains a C++ source file, and a make file. The make file usually creates an archive named `libloader.a` which is linked with the simulator.

Microprocessor Classes

Designing the "microprocessor" classes is the most important part of the simulator. These classes do the actual work of simulating the microprocessor. BSVC requires the main microprocessor class be derived from `BasicCPU`, but it does not place any restrictions on how the functionality needed by `BasicCPU` is implemented.

The first step in designing the microprocessor classes is to determine what level of simulation the simulator will be performing. The Motorola 68000 simulator simulates the 68000 at the software level, while the HECTOR 1600 simulator works at the data path level. After the level of simulation has been determined the next step is to determine the microprocessor classes. Basically the microprocessor needs to be broken down into objects that simulate the corresponding components of the microprocessor. For example the HECTOR 1600 has two main features the data path and the control unit. In the HECTOR 1600 simulator there are two classes that represent these two features of the Hector 1600. The microprocessor class layout of the HECTOR 1600 simulator is shown in Figure 12. In general you should try to break the processor into as many "reasonable" objects as possible. There is no correct breakdown, but some are better than others[8].

[8] If you're really interested in getting it "right" then take time to read a good Object-Oriented Design and Analysis book. I suggest Booch's Object-Oriented Analysis and Design with Applications or Jacobson's Object-Oriented Software Engineering.

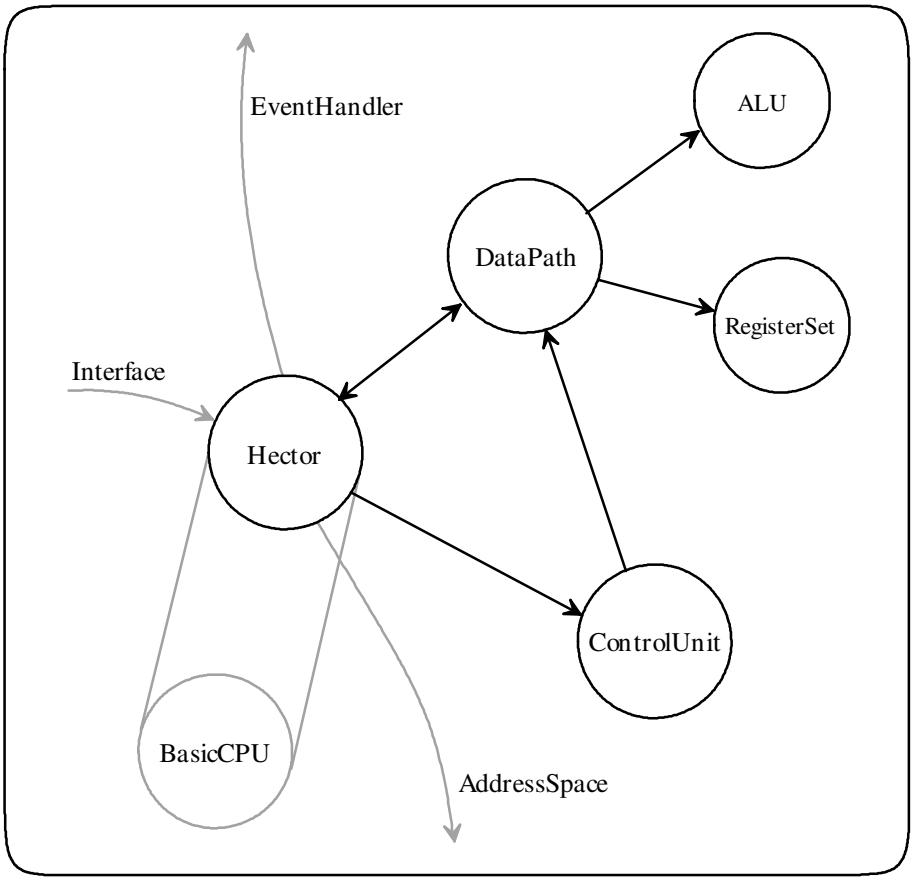

FIGURE 12 Hector 1600 Microprocessor Class Layout

Devices

Every simulator needs devices to simulate things that get attached to the microprocessor's address bus. In a BSVC simulator `BasicDevices` get attached to the `BasicCPU`'s `AddressSpace` object. When the CPU references memory, the `AddressSpace` object accesses the correct device. Simple simulators will only have a RAM device, while complex simulators may provide UARTs, timers, and other devices. Every device in the simulator has to be derived from the `BasicDevice` abstract base class. This base class defines the protocol the `AddressSpace` class uses to access the device.