Appendix A Framework Overview

BSVC provides a framework for writing microprocessor simulators with a set of C++ classes. This chapter gives an overview of the framework describing the class layout along with the user interface communication protocol[8].

Overview

The framework is as generic as possible to allow simulation of any processor or device. It handles all interaction with the user interface process, provides base classes for common objects, furnishes timed events to simulate interrupts, and manages breakpoints. Figure A-1 shows the framework's layout. All the classes whose names start with 'Basic' are abstract base classes. These base classes are "molds" for the actual classes used in the simulator. For example in the Motorola 68000 simulator the `BasicCPU` class is the base class for the `m68000` class.

[8] The BSVC C++ class declarations are in Appendix B.

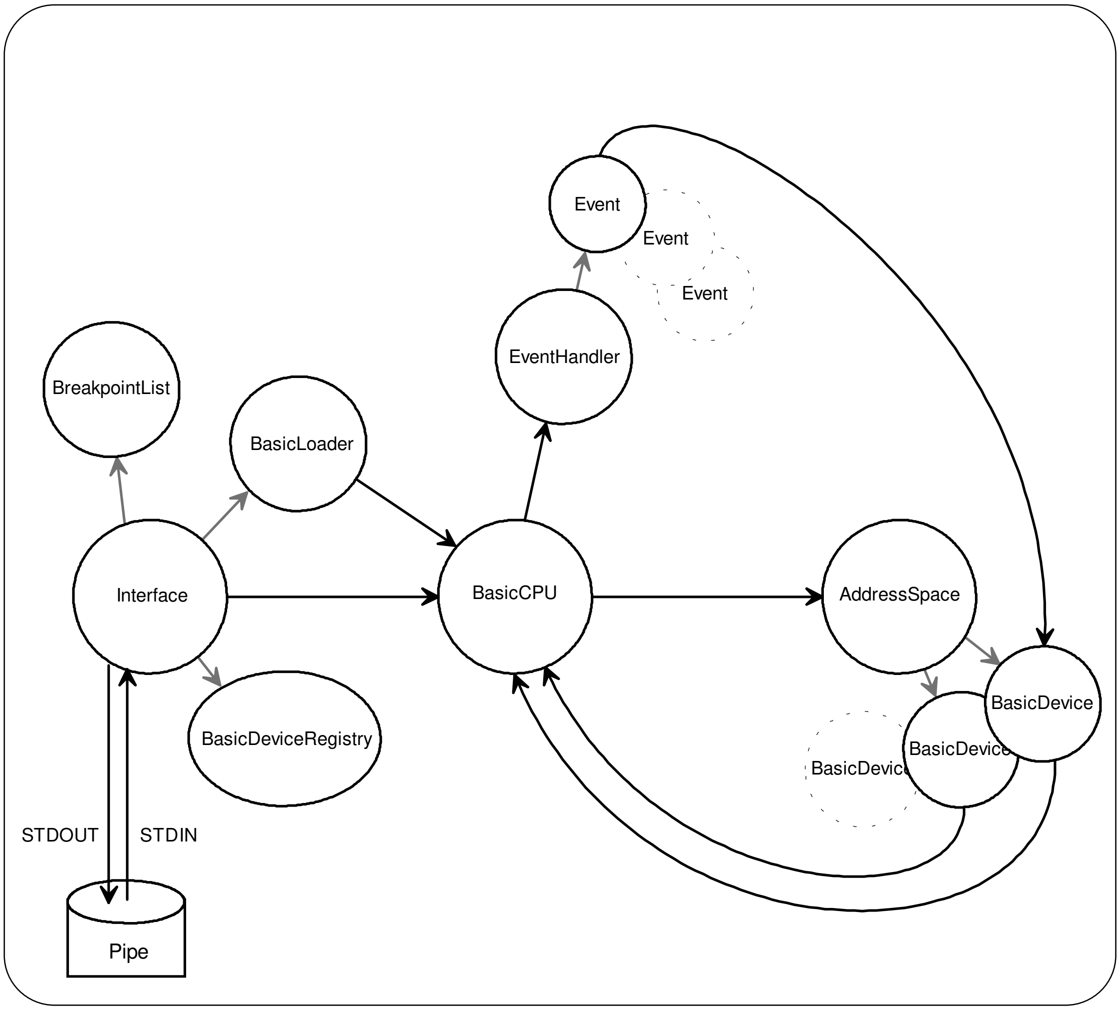

FIGURE A-1 Framework Layout

Class Descriptions

This section provides a brief summary describing the purpose of each of the C++ classes in the framework.

AddressSpace

This class maintains a collection of attached devices and provides methods to peek and poke into them. It's not an abstract base class and in most cases used as is. However using it as a super class you could derive a "VirtualAddressSpace" or "CachedAddressSpace" class supporting virtual or cached memory.

BasicCPU

This class is the heart of the simulator its main purpose is to execute instructions and handle interrupts. Most of the work writing a simulator goes into the "CPU" class derived from this abstract base class.

BasicDevice

This abstract base class is the foundation for all devices in the simulator. Its main job is to simulate the device at the register level and provided methods for the `AddressSpace` to read from and write to the device.

BasicDeviceRegistry

This abstract base class is used to derive a class that maintains a list of all the devices in the simulator and allows them to be created.

BasicLoader

This abstract base class provides methods to load object files into the simulator. A derived class should load all the object file formats supported by the simulator. The `Interface` object uses this derived object to load files into the simulator.

BreakpointList

The `Interface` class uses this class to manage a list of breakpoints. This should not have to be modified.

Event

This is a nested class in the `EventHandler` class. Whenever an `EventBase` derived object (i.e., a device) requests an event an `Event` object gets added to the `EventHandler`'s delta queue.

EventHandler

This class maintains a queue of events requested by `EventBase` derived objects. Whenever an event times-out the `EventBase` object's Callback method is executed. Events are useful for simulating regular time interval interrupts. You should not have to modify this class.

Interface

This class communicates with the user interface. It provides a standard set of commands allowing the user interface to "wrap" around any simulator written with the BSVC framework. This class should not have to be modified.

Communication with the User Interface

The simulator and user interface are two separate processes. The simulator process is a slave process that waits for commands from the user interface process. Whenever the user interacts with the user interface, commands are sent to the simulator to perform the necessary tasks. For example when the "Single Step" button is pressed the user interface sends "Step 1" to the simulator. In response to this command the simulator executes the next instruction. Listed below in Table A-1 are all the simulator commands.

TABLE A-1 Simulator Commands

Command	Arguments	Description
AddBreakpoint	<addr>	Sets a breakpoint at the address
AttachDevice	<addrspace> <name> <args>	Attaches the named device
ClearStatistics	*n/a*	Clears any statistics the simulator keeps
DetachDevice	<addrspace> <device index>	Detaches the indexed device
DeleteBreakpoint	<addr>	Clear the breakpoint at address
FillMemoryBlock	<addrspace> <addr> <len> <val>	Fills the block of memory with the value
ListAttachedDevices	<addrspace>	Lists devices attached to the address space
ListBreakpoints	*n/a*	Lists all the breakpoints that are set
ListDevices	*n/a*	Lists available devices
ListDeviceScript	<name>	Lists the Tcl script for the named device
ListExecutionTraceRecord	*n/a*	Lists the symbolic names in trace records
ListDefaultExecutionTraceEntries	*n/a*	Lists the default trace records to display
ListGranularity	*n/a*	Granularity in bytes
ListMaximumAddress	<addrspace>	Maximum address of the address space
ListMemory	<addrspace> <addr> <len>	Lists memory contents
ListNumberOfAddressSpaces	*n/a*	Number of address spaces
ListRegisters	*n/a*	List registers and values
ListRegisterValue	<register name>	Lists value of the named register
ListRegisterDescription	<register name>	Lists a description of the named register
ListStatistics	n/a	Lists any statistics kept by the simulator
LoadProgram	<addrspace> <filename>	Load the named program
ProgramCounterValue	*n/a*	Lists the value of the program counter
Reset	*n/a*	System reset
Run	*n/a*	Full speed execution
SetMemory	<addrspace> <addr> <value>	Sets the given memory location
SetRegister	<register name> <value>	Sets the named register to the value
Step	<number>	Execute instructions