ELG5191: Design of Distributed Software Systems: Project Report

Professor Ionescu
Fall Term 2000

Rami Abielmona†
Irenee Dupre la Tour†

†(rabiel,idupre)@site.uottawa.ca

December 25, 2000

1 Introduction

The main objective of this project is the remote programming of a Programmable Logic Device (PLD). Each PLD can be programmed by receiving a pre-derived bitstream, with which the internal logic blocks of the PLD are programmed. This bitstream is the output of the hardware logic design process and/or design automation tools. Traditionally, the bitstream is downloaded to the PLD through the use of a serial/parallel port, or more recently through a PCI bus. The crucial point is that the local host usually performs this download action to the PLD. The purpose of this project is to allow a remote host to program the logic blocks of a PLD across a certain type of network. The bitstream can then be sent from the *design host machine (DHM)* to the *field host machine (FHM)* for remote configurations and/or updates.

The design specifications are as follows:

- A bitstream will remotely program a PLD
- All PLDs in this project will actually be FPGAs
- At least one FHM will be present, with the option of selecting different bitstreams for each FHM (if more than one)

- A bitstream represents a hardware circuit
- One DHM will be present, with the option of selecting different bitstreams for different FHMs (again, if more than one)

The implementation specifications are as follows:

- CORBA will be utilized as the implementation standard
- Valid bitstreams will be utilized, hence must be created
- The FPGA vendors are Xilinx and Altera The FPGAs are to be integrated as CORBA objects, with direct hardware interface capabilities

2 System Architecture

The preliminary system architecture is pictured in figure 1. As we can see, there are multiple FHMs and one DHM, to which the client can connect to and remotely configure a PLD. The circuit configuration selected by the user is cross-referenced to a database, where all the possible configurations are stored. The faults with this design include the numerous paths between each of the FHMs and the DHM. Also, the client functionality has to be present in every FHM, reducing efficiency. The design evolved over time, and eventually was finalized to the diagram shown in figure 2. In this design, there is one main system client, which accesses the DHM, and selects the PLD as well as the circuit for configuration.

3 System Components

The main objective of this project is to be able to download a bitstream, configuring a PLD, across a network, from a machine to another. This will enable us to program various PLDs with predetermined logic, without having to be at the physical location of the FPGA. That said, the main components of the project have to correctly delineated and implemented. The following is a brief description of each of the major system components:

Main Server (DHM) This component will handle all database requests from the client. It will thus handle the database accesses. It will return, to the client, an *acknowledge* signal which will aid in error detection. The main server will also forward, to the appropriate host machine, the bitstream needed to configure the PLD.

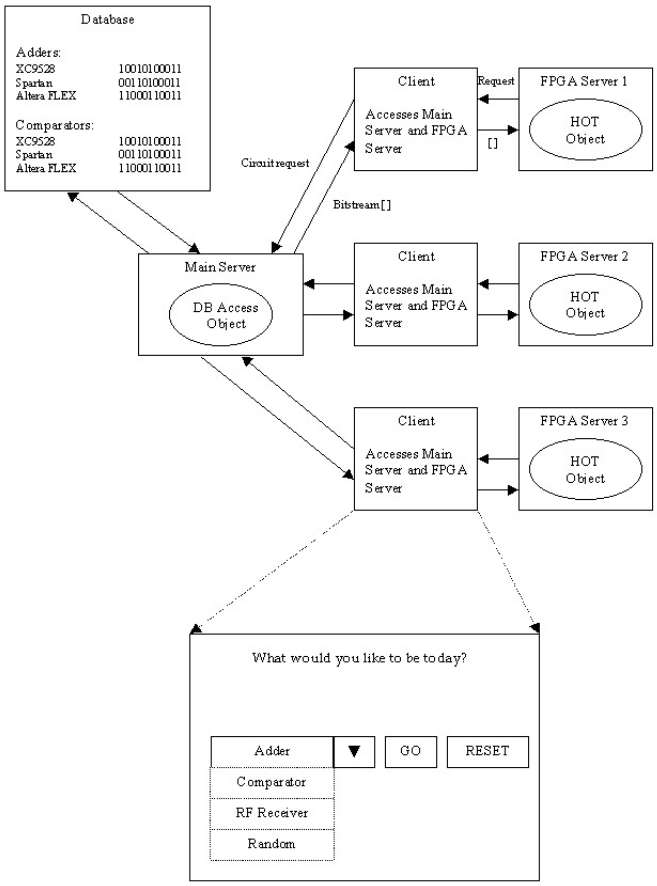

Figure 1: Preliminary System Architecture

Database This component will be used to store the myriad of circuit configurations, depending on vendor and technology utilized. It will contain the actual bitstreams for all the supported hardware devices.

Main Client At the moment, this component requests a circuit and a PLD configuration-pair, and receives an acknowledge signal of completion.

Field Host machine (FHM) This component is present in the structure as each FPGA connects to a host machine. The latter receives, from the DHM, the bitstream with which to configure the FPGA. It then forwards that result to its attached FPGA server.

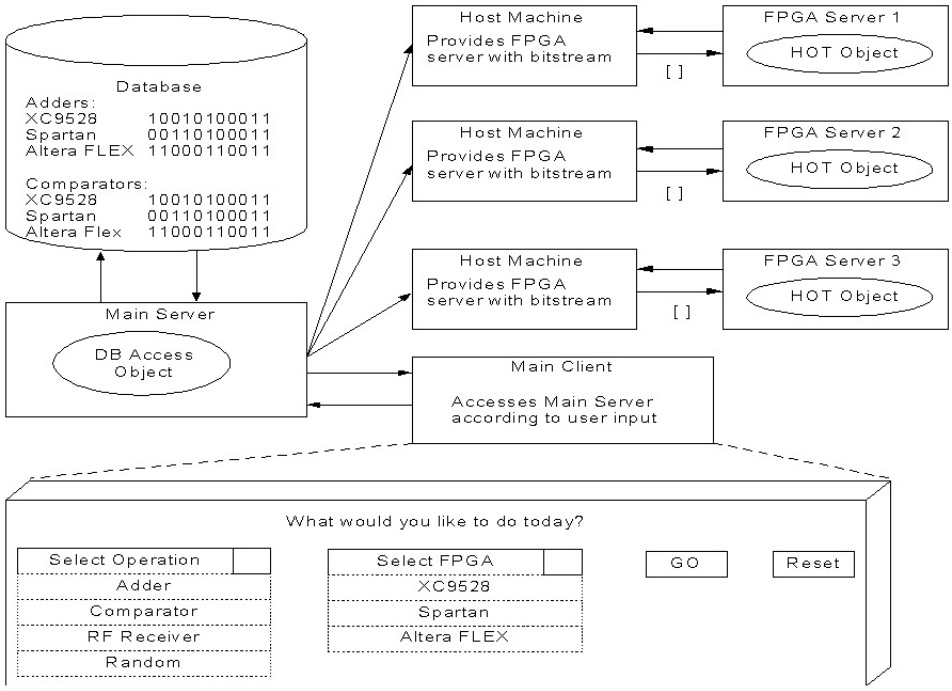

Figure 2: Final System Architecture

FPGA Server This component's main functionality is to handle its client's request in order to instantiate the FPGA with the bitstream that the client (FHM) provides, along with its request.

An object-oriented analysis of the system was performed, and the output of this phase is shown in figure3.

As the implementation phase progressed, the following was implemented for each class:

Main Server

- makeCircuit() is called by the go() method of the Main Client

Main Client

- On input of the user, the go() method is called

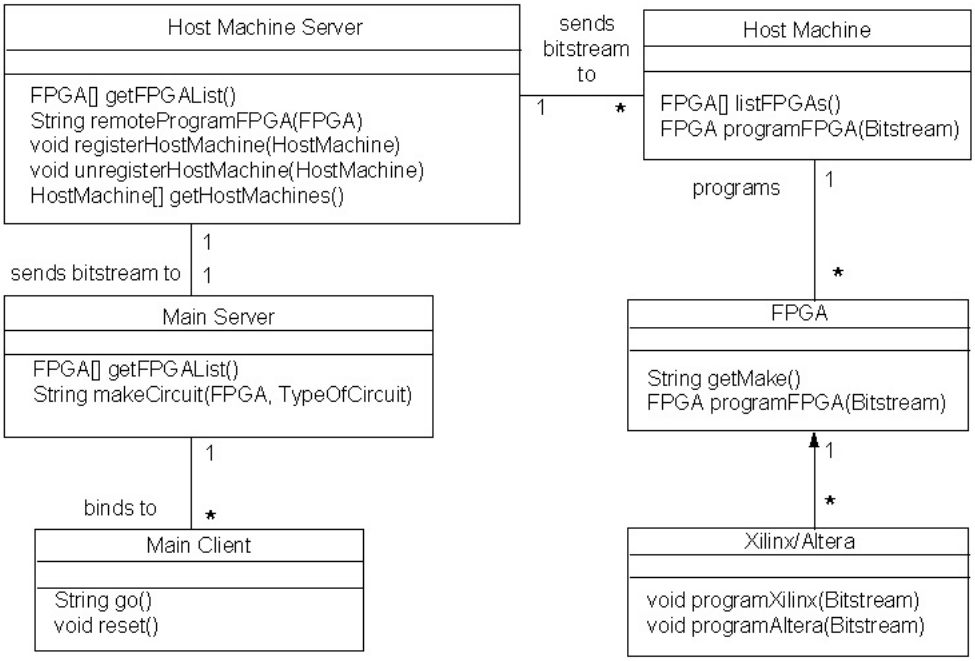

Figure 3: Class Diagram

Host Machine Server

- register/unregister is called by Host Machine(s)
- getHostMachine() is called by the Main Server
- rpf() is called to remotely program the FPGA

Host machine

- programFPGA(Bitstream) is called by the Host Machine Server to program the FPGA
- getFPGA() is called to get a list of the attached FPGAs

FPGA

- downloadBitstream(Bitstream) is called to program a particular FPGA

HOT

- downloadBitstream(Bitstream) is inherited from the FPGA parent class
- downloadBitstreamHOT(Bitstream) is called by to program the HOT card

Another operation schematic is the sequence diagram (figure 4), which shows the sequence in which all the operations are performed in order to better grasp the overall system architecture, and the interconnection of the componenets.

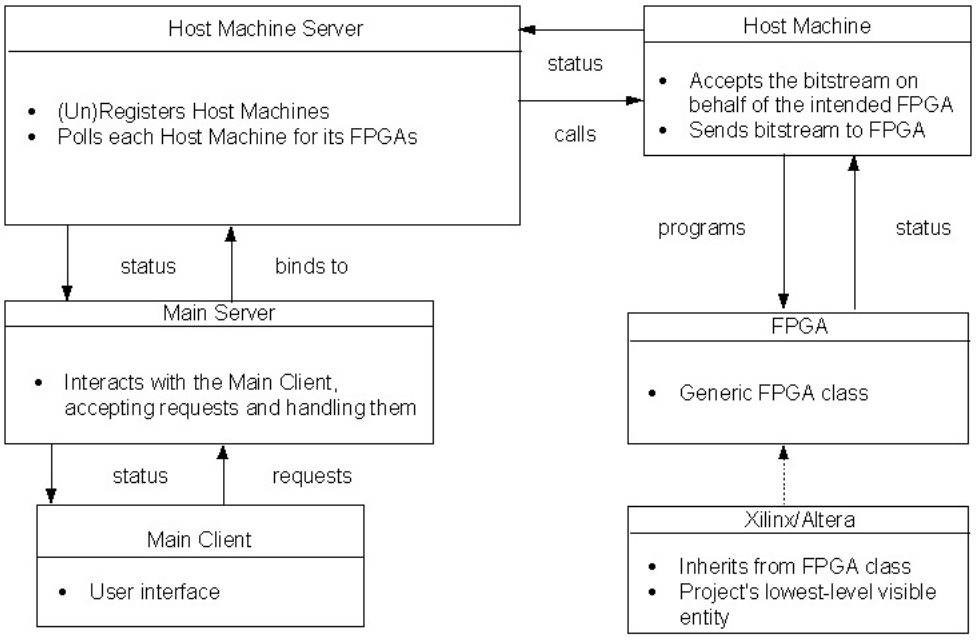

Figure 4: Sequence Diagram

As the implementation phase progressed, the following system operation can be currently observed:

- The user uses the main client to select a PLD-circuit configuration pair (currently limited to HOT card-cbmin);
- The file *cbmin.hot* is opened and its corresponding bitstream is read;
- The client binds to the host machine where the HOT card resides and sends the latter the bitstream;
- The Host Machine is created if not already running. It registers itself with the Host Machine Server, and proceeds to bind to the HOT card;
- Upon a successful bind, the Host Machine sends the HOT card object the bitstream and a request for configuration;
- The HOT card object proceeds in downloading the bitstream to the HOT card, and returns a status signal back to the Host Machine, which in turn forwards that back to the main client.

4 Implementation Details

The following lists this disk's directory structure:

src

- Makefile: The make file for this project;
- Client.cpp: The main client viewed by the user;
- FPGA.idl: The IDL interface of the parent FPGA object;
- FPGAImpl.cpp/.h: The implementation of the FPGA interface;
- FPGAMain.cpp: The main routine of the FPGA interface;
- HostMachine.idl: The IDL interface of the Host Machine object;
- HostMachineImpl.cpp/.h: The implementation of the Host Machine interface;
- HostMachineMain.cpp: The main routine of the Host Machine interface;
- HostServer.idl: The IDL interface of the Host Server object;

- HostServerImpl.cpp/.h: The implementation of the Host Server interface;
- HostServerMain.cpp: The main routine of the Host Server interface;
- HOT.idl: The IDL interface of the HOT object;
- HOTImpl.cpp/.h: The implementation of the HOT interface;
- HOTMain.cpp: The main routine of the HOT interface;
- MAX.idl: The IDL interface of the MAX object;
- MAXImpl.cpp/.h: The implementation of the MAX interface;
- MAXMain.cpp: The main routine of the MAX interface.

javaGui

- ClientGUI.java: The GUI java source file;
- MainInterfaceClient.java: The client that attaches to the the GUI;
- MainInterfaceImpl.java: The server that interfaces with the DHM.

5 Discussion

The project was implemented, and tested on two different machines, with only the HOT object registered in one machine (the one ideally containing the HOT card), and with the three other servers (hostServer, hostMachine and FPGA) on the other machine. The bitstream was read from a previously made design (cbmin.hot). The bitstream was sent from one machine to the other through the interaction of the HOT and Host Machine interfaces. The bitstream was then used to call the function which effectively downloads the bitstream to the HOT card. The only drawback is that the Orbix environment is not installed on the workstation that currently contains the HOT card, and thus the latter could not be completely configured. The other option is to install the HOT card in a workstation that already contains the Orbix environment.

Another major obstacle is the Java side of the project. The main client is ideally supposed to interface with the DHM in order to select the PLD-circuit configuration pair, but the Java client could not be correctly configured with the current OrbixWeb implementation, and thus could not properly

function. A C++ GUI is currently being worked on, with the hopes of replacement of the Java GUI in the near future. Also being worked on is the attempt to completely test the project, by sending a bitstream from the DHM to the FHM that contains the HOT card, as previously explained.

The project can be currently tested by running 'client ¡hostname¿'. The host where the client is run must contain the following server implementations: hostServer, hostMachine and FPGA. Also, in HostMachineImpl.cpp, when the bind is called to the HOT object, a hostname can be given in order to tell the HostMachine where the HOT card is located on the network. Thus, on that latter machine, the HOT server must be registered in the implementation repository. Upon calling the client, the hostMachine is contacted, which in turn contacts the hostServer, and after registration, contacts the HOT object in order to download the bistream.

6 Conclusions

The implementation phase of this project has fulfilled all the functional requirements. This project aids in the remote download of a configuration of a PLD from a design site, to the intended logic device, the latter ideally residing a good distance away from the design site. This project also aids in the development of some metrics for certain interesting hardware remote programming characteristics. Finally, this project provides us with a great, and detailed exposure to the CORBA interworkings