[Book Title], Edited by [Editor's Name].
ISBN 0-471-XXXXX-X Copyright © 2000 Wiley[Imprint], Inc.

CHAPTER 4

NEXT GENERATION LABORATORY - A solution for remote characterization of analog integrated circuits

Carsten Wulff, Trond Ytterdal, Thomas Aas Sæthre, Arne Skjevlan

4.1 INTRODUCTION

In this chapter we describe the development and use of the remotely operated laboratory Next Generation Lab (NGL) that has been implemented at the Department of Physical Electronics, Norwegian University of Science and Technology, Trondheim, Norway. NGL is based on Microsoft's .NET technology and combines the latest in Web technology with standard industrial instruments to make a cost effective solution for education in the field of analog CMOS integrated circuits. NGL has evolved from the work done in [1-7]. The main objectives for developing NGL was:

- To provide a remote laboratory course for the education in design of analog integrated circuits at our department.
- To create a platform for circuit experiments where it is easy to add new experiments.

- To create a prototype experiment that measures frequency response of operational amplifiers that have been designed by students.

The outline of this chapter is as follows. In Section 4.2 we give an overview of the software technologies used. Then in Section 4.3 we describe the physical architecture and setup of the laboratory followed by a discussion of the software architecture employed. In Section 4.5, we describe the implemented prototype experiment and in Section 4.6 we present a lab assignment given as part of the course Analog CMOS 1 that is offered to fourth year students at our department. We end this chapter by discussing our experiences with NGL and future plans.

4.2 TECHNOLOGY

The development of NGL started in the summer 2001. At that time, several possible programming platforms for a remote laboratory were available. Including Active Server Pages (ASP), PHP: Hypertext Preprocessor (PHP), Practical Extraction and Report language (PERL), .NET platform (beta 1) and several others. To select the programming platform that would suit us we looked at the most critical portion of the application. The instruments we were going to use had, as most commercial instruments, an implementation of General Purpose Interface Bus (GPIB) through which one can remotely operate the instruments. This restricted our choices to a programming platform that could use the C++ library provided by the GPIB card. Although we could have used a number of programming platforms to interface with the GPIB C++ library, the .NET platform was emerging as a promising new technology and we wanted to explore its capabilities. The .NET platform has a large class library and seamless integration between standard windows components and Web applications. In addition, a new object oriented programming language C# was released together with the .NET platform. The C# programming language combines a low learning threshold with the power of C++ and was therefore a prime candidate for the Web application. The .NET platform has the advantage of being, in theory, both platform and language independent. When source code is compiled using a .NET compiler, the output is Microsoft Intermediate Language (MSIL). When a program runs, the MSIL is compiled using a Just In Time (JIT) compiler, the same principle as Java. Since all languages produce MSIL, when compiled with a .NET compiler, MSIL entities can be used as components without the need to register them as is the case of binary COM components. Hence new features will emerge. An exciting consequence of this technology is the ability to write a class in C++ and inherit the class in C# or PERL. All these languages can make use of the extensive class library in the .NET platform. At the time of writing, several languages support the .NET platform, among them are VB.NET (Visual Basic 7.0), C#, C++ and PERL.

In addition to the server-side program necessary to perform the experiments, we needed a way to display the results to the user. The result from the prototype experiment consists of plots of the magnitude and phase of the output signal. To display these plots we considered three solutions; creating a bitmap server-side that is sent to the client, creating a Java applet to display the plots or using a new technology from Adobe called SVG (Scalable Vector Graphics). SVG was at the time implemented as a graph viewer at the “Lab-On-Web” [8] remote laboratory at UniK/NTNU, and we were fortunate to use this component and extend it to suite our purposes.

An explanation of the different technologies used in the NGL application follows.

4.2.1 .NET PLATFORM

.NET Framework

The .NET Framework consists of the Common Language Runtime (CLR), a JIT compiler and a large class library. It is delivered in two versions, one with SDK (Software Development Kit) and one without. Both can be downloaded free of charge from http://www.asp.net/.

ASP.NET

ASP.NET enables rapid development of powerful Web applications and services and can be deployed and run on any browser or device.

ASP.NET come with a number of useful features for Web developers. Among them, we have employed Pagelets, Web Controls, and Web Services in this project.

Pagelets are reusable HTML code, which can be included into an ASP.NET page. They can be used as a container for common elements on HTML pages such as a menu or a header.

Web Control makes it possible to create customized tags. Each tag has a class that defines its properties and what it displays on a Web page. It is therefore an ideal solution for a system where several programmers are working on the same application. In the same way that a class can provide an easy to use interface for advanced logic, a Web Control can be used without knowing the intricate details of the inner logic.

Web Service is application logic that is programmatically available over the Internet. Web Services provide a programming abstraction that allows developers to easily make their applications available over the Internet, without knowledge about HTTP, COM, TCP/IP or data marshaling.

C#

C# is a new object-oriented language developed by Microsoft. The reason for developing this new language was the lack of a sufficiently productive, yet flexible language. C/C++ has the flexibility, but developing new applications with C/C++ is too time-consuming with the current demands on time to market. Microsoft's solution to this problem was C#, which possesses the productivity of Visual Basic without sacrificing the power and control of the C/C++ language.

The modern design of C# eliminates the most common C++ programming errors by applying:

- Garbage collection that relieve the programmer from the burden of manual memory management.
- Variables in C# are automatically initialized by the environment.
- Type-safe variables.

This makes it easier for developers to write and maintain applications.

4.2.2 XML

Extensible Markup Language (XML) is a markup language for documents containing structured information. The purpose of a markup language is to identify structures in a document. XML defines a standard, which does just that. In our application the XML is used for structuring the results from an experiment. Below is an example of the result information. Applying “tags” to the information elements makes the structures in XML. The properties of the experiment are the elements between the `<PROPERTIES>` `</PROPERTIES>` tags, the properties provide information on the parameters of the experiment. For example the numbers between the `<STARTFREQUENCY>` and the `<STOPFREQUENCY>` are the start and stop frequencies of the frequency sweep. The second portion (`<GRAPH>`) contains all information used to draw the plots from the experiment (the X and Y values have been abbreviated to save space).

An XML example:

```
<?xml version='1.0'?>
<RESULT  ID="Frequency  Response  AnCMOS"  dateTime  =  "26.08.2002
13:03:21">
<PROPERTIES>
    <OPAMP>2</OPAMP>
    <RESISTOR>2</RESISTOR>
    <STARTFREQUENCY>10000</STARTFREQUENCY>
    <STOPFREQUENCY>40000000</STOPFREQUENCY>
    <BIASCURRENT>0.0001</BIASCURRENT>
    <VIPVOLTAGE>1.57</VIPVOLTAGE>
</PROPERTIES>
<GRAPH>
    <XVALUES>10000,10209.5159,… </XVALUES>
    <YVALUES>12.5311794782872,13.2404220730507,…</YVALUES>
    <XLABEL>Frequency</XLABEL>
    <YLABEL>Magnitude</YLABEL>
    <YVAL>dB</YVAL>
    <XVAL>Hz</XVAL>
    <VSIZE>250</VSIZE>
    <HSIZE>450</HSIZE>
    <TYPE_X>LOG</TYPE_X>
    <TYPE_Y>LIN</TYPE_Y>
```

```
        <SAMELINE>TRUE</SAMELINE>
    </GRAPH>
    <GRAPH>
        <SAMELINE>FALSE</SAMELINE>
        <XVALUES>10000,10209.5159,… </XVALUES>
        <YVALUES>-105.487563913513,-106.635818813394,… </YVALUES>
        <XLABEL>Frequency</XLABEL>
        <YLABEL>Phase</YLABEL>
        <YVAL>DEG</YVAL>
        <XVAL>Hz</XVAL>
        <VSIZE>250</VSIZE>
        <HSIZE>450</HSIZE>
        <TYPE_X>LOG</TYPE_X>
        <TYPE_Y>LIN</TYPE_Y>
    </GRAPH>
    </RESULT>
```

4.2.3 SVG

Scalable Vector Graphics (SVG) is a new graphics file format and Web development language based on XML. SVG enables Web developers to create dynamically generated graphics on a Web page. Main reasons for choosing SVG are:

- SVG is not a proprietary standard. This means that nobody owns it, and everybody is free to use it in their products.
- SVG has zooming capabilities.
- SVG supports scripting, which enables a wide range of development possibilities.
- SVG is based on XML, and it is therefore possible to manipulate SVG-files using standard class libraries

SVG-files tend to be very small. The main reasons for that is:

- Shapes and characters are saved as text, not as graphics.

- Complex figures are defined by a `<PATH>` command. There is also a Bezier-curve command.
- You can define a symbol once, and refer to it next time you need a similar symbol.

In SVG, the Painter's algorithm is implemented. This means that the current object is painted on top of the previous object. SVG can consist of three different types of graphic elements: shapes, text and bitmaps.

4.2.4 CSS

Cascading Style Sheets (CSS) is a simple way of defining in a centralized location, what a page should look like, thus avoiding the time-consuming job of setting i.e. font properties in every Web page.

4.2.5 JavaScript

JavaScript was originally a platform- and browser-independent client side scripting language. JavaScript was developed by Netscape, but has been extended in the later years by Microsoft. This has led to a loss of browser-independency, were some functions not being understood by some browsers. But as long as you limit yourself to the functions that are understood by all browsers it is a good client side scripting language.

4.3 PHYSICAL ARCHITECHTURE AND EXPERIMENTAL SETUP

This section will concentrate on the physical architecture of the laboratory and the experimental setups. The physical architecture of NGL is shown in Figure 4.1. The architecture was based on the idea that NGL should be a scalable solution. The NGL Web server runs the main Web application that contains the application logic. On the next level, we have Lab servers running the LabServer Web Service. The LabServer Web Service can be viewed as easy to use logic for accessing a GPIB and a Data Acquisition Board (DAQ) card installed on the Lab server. Several instruments can be connected to the GPIB board. The instruments can be connected to a device under test (DUT). The instruments hooked up to a Lab server can all be connected to the same DUT, or they can be connected to different DUTs. This results in a great flexibility for specifying experimental setups.

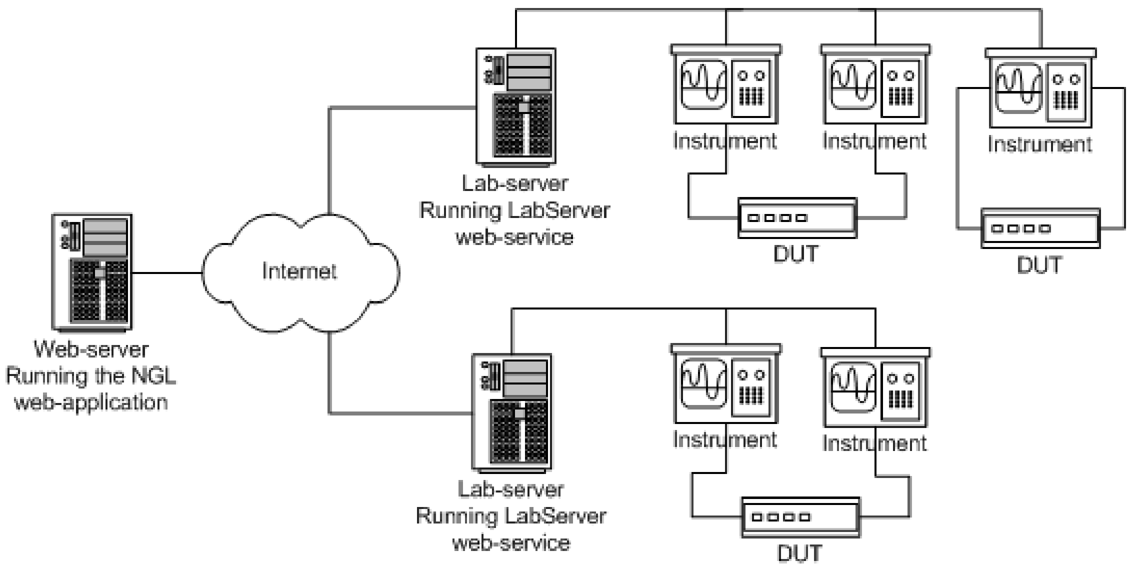


Figure 4.1. The physical architecture of the NGL Web application.

The physical architecture of the NGL prototype, shown in Figure 4.2, is a subset of the architecture in Figure 4.1. It has one server that runs the Web application and another server that runs the LabServer Web Service. Connected to the Lab server is a vector network analyzer from Rode & Schwarz, an Agilent power supply, and a DAQ board. These are in turn connected to the DUT, the Analog CMOS chip, which contains 9 operational amplifiers (OPAMPS).

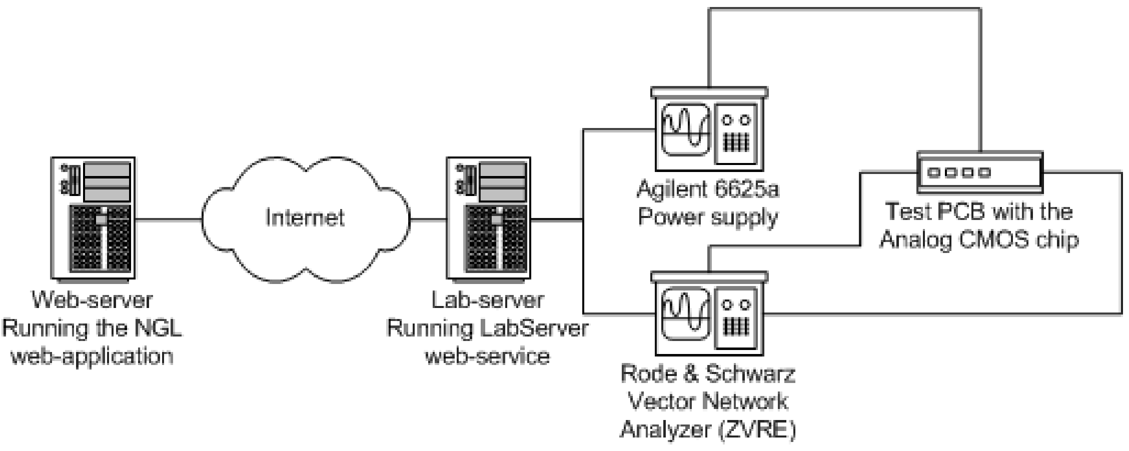


Figure 4.2. The physical architecture of the NGL prototype.

Fourth year M.Sc students at NTNU designed the OPAMPS in the fall of 2000 as a term project in the course Analog CMOS 1. The OPAMPS were integrated on the same chip together with some digital logic and an output buffer. The wiring diagram for the experimental setup is pictured in Figure 4.3. The prototype experiment allows the user to specify closed loop gain, bias current to the OPAMP, offset from common mode on positive input of the OPAMP and which OPAMP to use in the measurement. The resistors can be switched to give a closed loop gain of 0dB, 20dB, 40dB or 60dB. The bias current can be varied from 0.5uA to 25uA and the offset from common mode can be varied from –100mV to + 100mV.

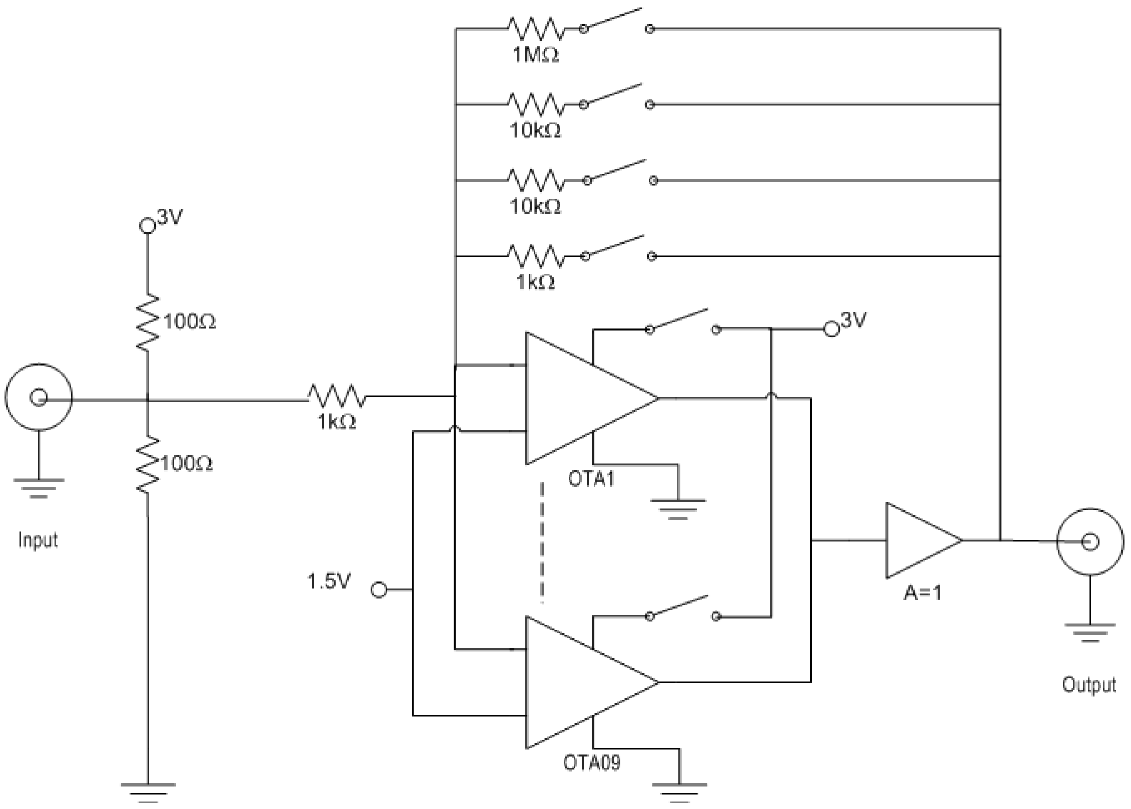


Figure 4.3. Analog CMOS chip test-setup.

4.4 SOFTWARE ARCHITECTURE

The software architecture of NGL is divided into three parts (see Figure 4.4): Web application, client side graphics and the LabServer Web Service. The Web application is divided into two components: Presentation and Logic. Presentation consists of the layout and the user interface of the NGL Web site. Logic contains the application logic that receives the parameters from the user, controls the execution of the experiments, performs the experiment, and prepares the data for presentation. Client side graphics contains the SVG control that is used to present the experimental results graphically on the client side. LabServer Web Service is, as mentioned, easy to use logic for accessing the GPIB and DAQ card for controlling instruments and the Analog CMOS test board. Each part will be explained separately starting with Web application, followed by Client side graphics module and LabServer Web Service.

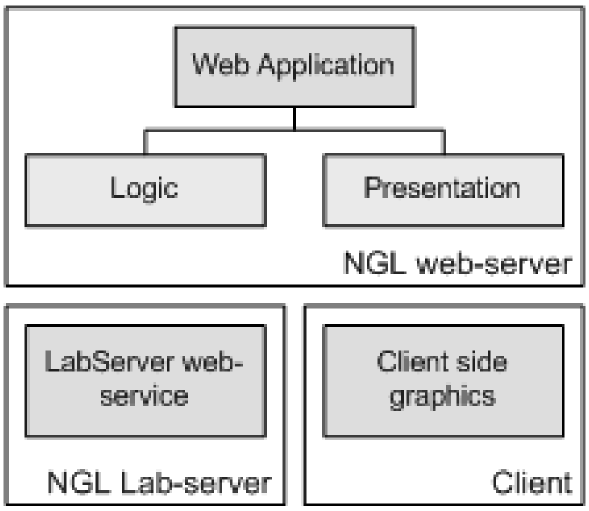
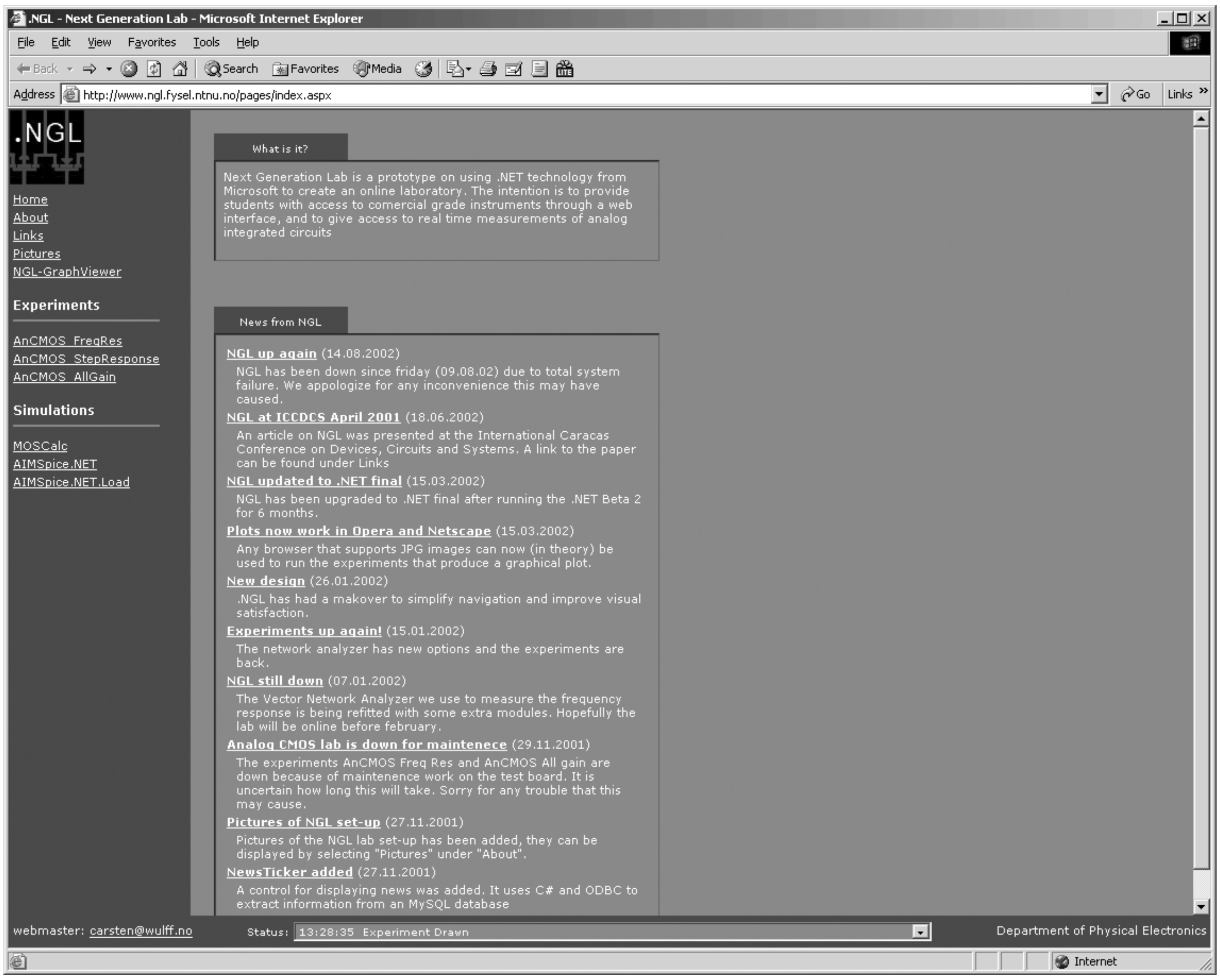


Figure 4.4. Software architecture overview.

4.4.1 Web Application Presentation

The layout and the user interface of NGL are designed with simplicity and ease of use in mind. The front page is shown in Figure 4.5. It was very important that the layout should be easy to use and without "eye-candy" that would distract the user. The page is divided into three sections, the menu area on the left, the main area on the right and a status field at the bottom of the page. The menu, cascading style sheets and JavaScript's are included into the page using a Pagelet. The experiments portion of the menu is automatically retrieved from the experiment classes in the application. We will show later how this is done. The status field provides the user with real time feedback on the progress of the experiment. This feature only works in Internet Explorer 5.0 and later versions because it takes advantage of Internet Explorers forgiveness concerning formatting of the HTML document. Providing real-time feedback with the HTTP protocol is not an easy task, since it follows a client request/server response model. To circumvent this problem the server-side application flushes the output buffer when a request to update the status field is made. Unfortunately, the start tags (for example `<HTML>` and `<HEAD>`) are not created before the experiment is finished, so the output buffer only contains JavaScript code to update the status field. Most browsers do not run a script that is received before the start tags, but because of Internet Explorers forgiving nature, this script is interpreted and executed even though the JavaScript is in the wrong place.

Presentation contains classes that the experiment uses to create input forms for the user. These classes take care of the layout of the input forms, the values returned from the user and browser capabilities. The three classes are: `Layout`, `TextLayout` and `GraphLayout`. The `Layout` class deals with event handling and is the base class for `TextLayout`. The `TextLayout` class contains methods that can be used to create an input form for the experiment as shown in Figure 4.6. The `GraphLayout` class includes methods to allow creation of a more intuitive design of the form, using background images and free positioning of form elements as shown in Figure 4.7. Since it inherits `TextLayout` standard form elements without free positioning can be requested. Figure 4.6 and Figure 4.7 show the user-interface of the prototype experiment, and we can easily see how Figure 4.7 provides a more intuitive interface for the user compared to Figure 4.6.


.NGL - Next Generation Lab - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.ngl.fysel.ntnu.no/pages/index.aspx
Go Links
.NGL
Home
About
Links
Pictures
NGL-GraphViewer
Experiments
AnCMOS FreqRes
AnCMOS StepResponse
AnCMOS AllGain
Simulations
MOSCalc
AIMSpice.NET
AIMSpice.NET.Load
Internet

What is it?

Next Generation Lab is a prototype on using .NET technology from Microsoft to create an online laboratory. The intention is to provide students with access to comercial grade instruments through a web interface, and to give access to real time measurements of analog integrated circuits

News from NGL

NGL up again (14.08.2002)
NGL has been down since friday (09.08.02) due to total system failure. We appologize for any inconvenience this may have caused.

NGL at ICCDCS April 2001 (18.06.2002)
An article on NGL was presented at the International Caracas Conference on Devices, Circuits and Systems. A link to the paper can be found under Links

NGL updated to .NET final (15.03.2002)
NGL has been upgraded to .NET final after running the .NET Beta 2 for 6 months.

Plots now work in Opera and Netscape (15.03.2002)
Any browser that supports JPG images can now (in theory) be used to run the experiments that produce a graphical plot.

New design (26.01.2002)
.NGL has had a makover to simplify navigation and improve visual satisfaction.

Experiments up again! (15.01.2002)
The network analyzer has new options and the experiments are back.

NGL still down (07.01.2002)
The Vector Network Analyzer we use to measure the frequency response is being refitted with some extra modules. Hopefully the lab will be online before february.

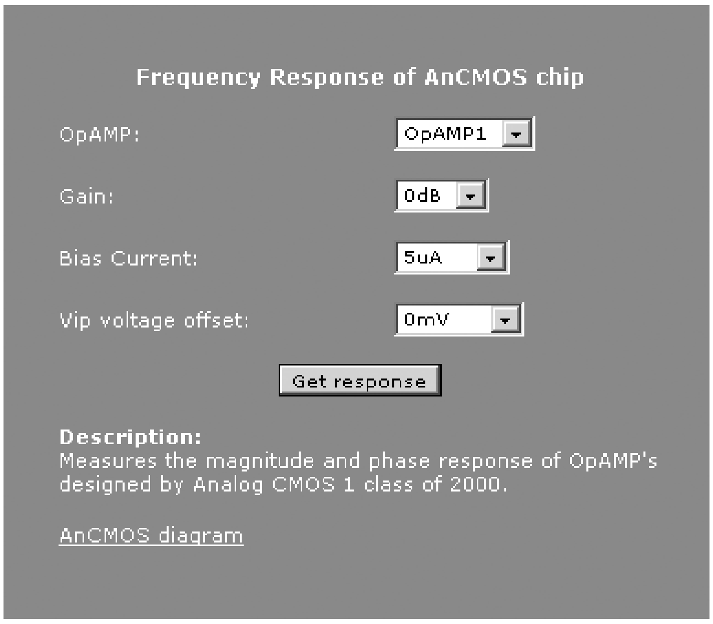
Analog CMOS lab is down for maintenece (29.11.2001)
The experiments AnCMOS Freq Res and AnCMOS All gain are down because of maintenence work on the test board. It is uncertain how long this will take. Sorry for any trouble that this may cause.

Pictures of NGL set-up (27.11.2001)
Pictures of the NGL lab set-up has been added, they can be displayed by selecting "Pictures" under "About".

NewsTicker added (27.11.2001)
A control for displaying news was added. It uses C# and ODBC to extract information from an MySQL database

webmaster: carsten@wulff.no   Status: 13:28:35 Experiment Drawn   Department of Physical Electronics

Figure 4.5. The NGL front page.



Frequency Response of AnCMOS chip

OpAMP: OpAMP1

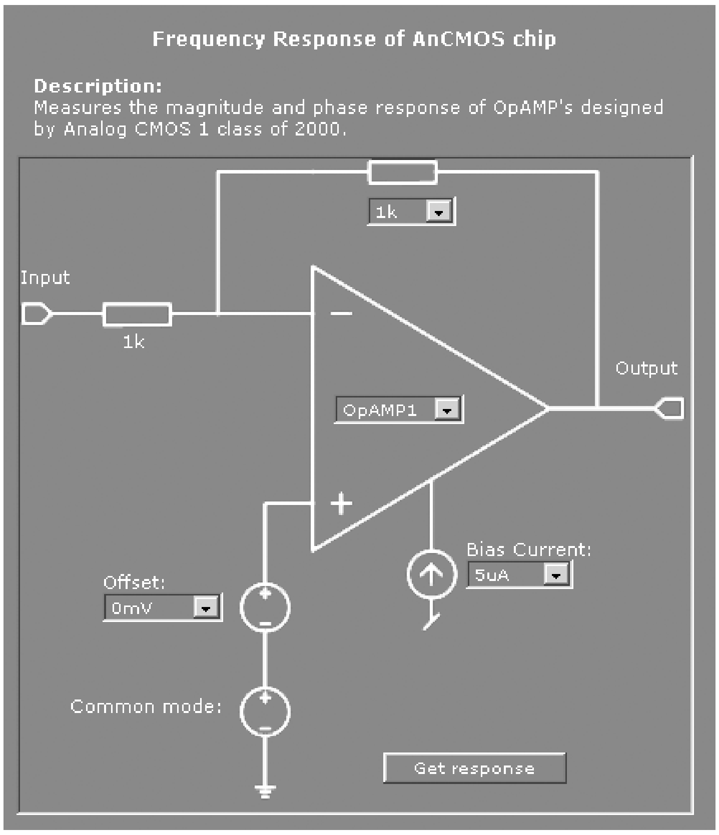
Gain: 0dB

Bias Current: 5uA

Vip voltage offset: 0mV

Description:
Measures the magnitude and phase response of OpAMP's designed by Analog CMOS 1 class of 2000.

Figure 4.6. Experiment using TextLayout



Frequency Response of AnCMOS chip

Description:
Measures the magnitude and phase response of OpAMP's designed by Analog CMOS 1 class of 2000.

Figure 4.7. Experiment using GraphLayout

4.4.2 Web Application Logic

This sub-section (and the following two) contains details on how the application logic was written. If you are not interested in programming, we would suggest skipping ahead to Section 4.5 The Frequency Response Experiment. However, if you intend to work on web laboratories or if you are a programmer we believe that this sub-section (and the following two) will provide you with some pointers.

Before we discuss the details of the Web application logic, we will give an explanation on how the Web application works.

Figure 4.8 shows the flow of an experiment and how the different objects interact. The flow is somewhat simplified from the actual flow of the experiment, but is sufficient for the discussion below. All the shaded boxes in the figure denote classes with the exception of `experiment.aspx`, which is a Web page.

When the user selects an experiment the page `experiment.aspx` is requested. Which experiment to draw is determined from the QueryString sent to the application. In this example we will use the `FreqResAnCMOS`, which measures the frequency response of an OPAMP on the Analog CMOS chip. The experiment utilizes the `GraphLayout` class to draw a form that contains the parameters the user can specify. `GraphLayout` makes use of a number of the standard Web Controls in the .NET framework to ensure as much browser interoperability as possible. When the user is finished selecting the parameters, he/she clicks the submit button. The click event is handled in the `GraphLayout` control since the submit button is one of its sub-controls. It parses the values returned and makes them available through a hash table. It then fires the `ValuesReady` event to notify the experiment that it has values ready for use. The

`FreqResAnCMOS` object catches the event and the event handler calls `FreqResAnCMOS.setupExperiment()` to parse the returned parameters. When the `ExperimentPage` has access to run an experiment it calls the `FreqResAnCMOS.run()` method. The `FreqResAnCMOS` object utilizes the instrument objects shown in the figure, but has no knowledge of where the actual instruments are located. After retrieving the raw data from the vector network analyzer, the `RsZVRE` parses the data into a form that is easy for the `FreqResAnCMOS` object to use. The flow at this point is not event driven but follow the normal stack flow of the Common Language Runtime (CLR). `ExperimentPage` retrieves an XML string that `FreqResAnCMOS` has built and gives it to `SVGControl`. Then, it returns control to the `experiment.aspx` page and the user sees the frequency response measured.

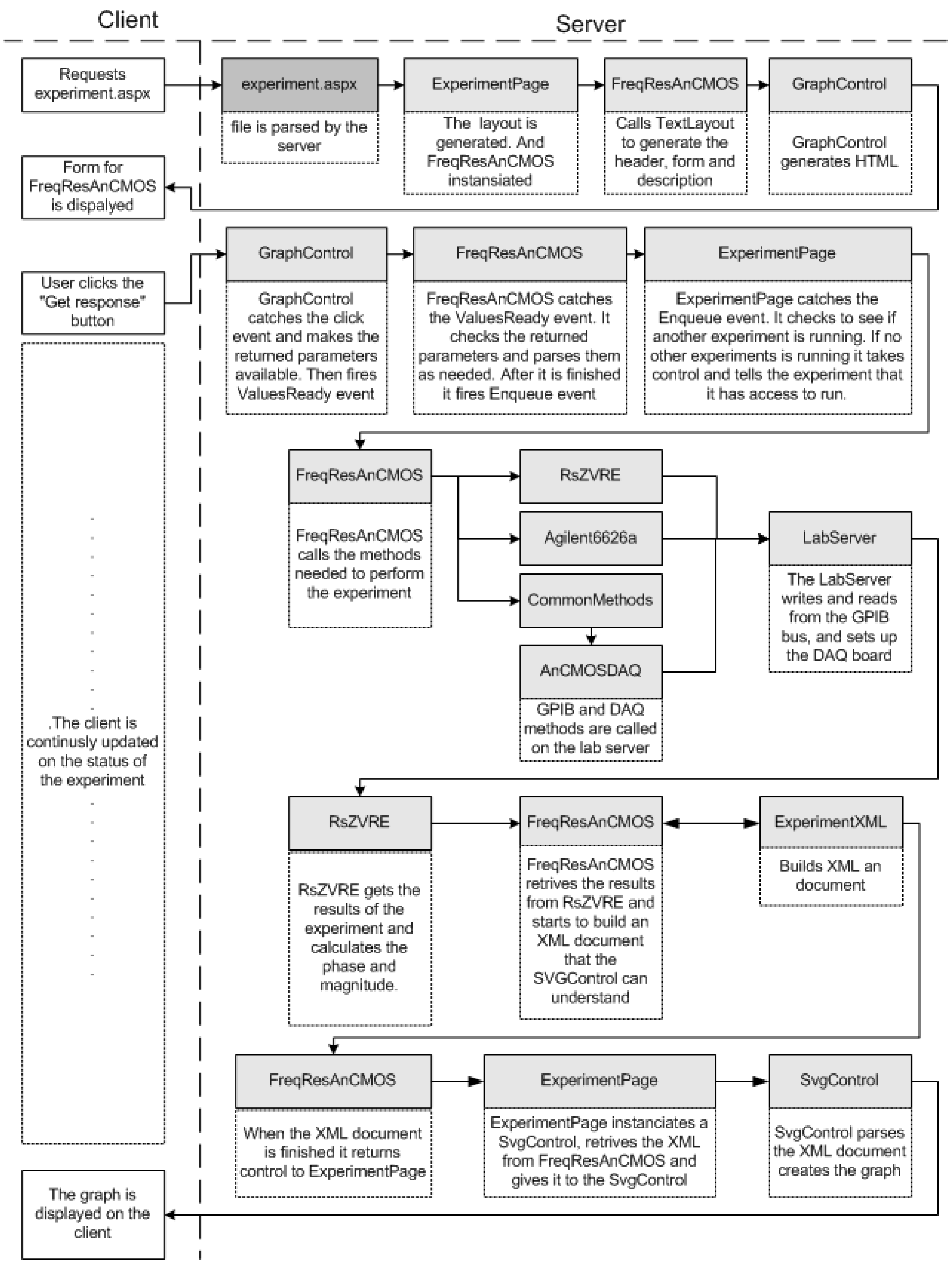

Client
Server
Requests experiment.aspx
experiment.aspx
file is parsed by the server
ExperimentPage
The layout is generated. And FreqResAnCMOS instansiated
FreqResAnCMOS
Calls TextLayout to generate the header, form and description
GraphControl
GraphControl generates HTML
Form for FreqResAnCMOS is dispalyed
User clicks the "Get response" button
GraphControl
GraphControl catches the click event and makes the returned parameters available. Then fires ValuesReady event
FreqResAnCMOS
FreqResAnCMOS catches the ValuesReady event. It checks the returned parameters and parses them as needed. After it is finished it fires Enqueue event
ExperimentPage
ExperimentPage catches the Enqueue event. It checks to see if another experiment is running. If no other experiments is running it takes control and tells the experiment that it has access to run.
.The client is continusly updated on the status of the experiment
FreqResAnCMOS
FreqResAnCMOS calls the methods needed to perform the experiment
RsZVRE
Agilent6626a
CommonMethods
AnCMOSDAQ
GPIB and DAQ methods are called on the lab server
LabServer
The LabServer writes and reads from the GPIB bus, and sets up the DAQ board
RsZVRE
RsZVRE gets the results of the experiment and calculates the phase and magnitude.
FreqResAnCMOS
FreqResAnCMOS retrives the results from RsZVRE and starts to build an XML document that the SVGControl can understand
ExperimentXML
Builds XML an document
FreqResAnCMOS
When the XML document is finished it returns control to ExperimentPage
ExperimentPage
ExperimentPage instanciates a SvgControl, retrives the XML from FreqResAnCMOS and gives it to the SvgControl
SvgControl
SvgControl parses the XML document creates the graph
The graph is displayed on the client

Figure 4.8. Flow of an experiment. The shaded boxes represent the different classes

The back-end of the Web application is collected in a .NET assembly, called `NextGenLab`, which is a collection of classes much like a COM (Component Object Model) object. However, unlike a COM object there is no need to register a .NET assembly. This makes it more manageable than COM and it does not require a Web server restart to be updated.

The class hierarchy of the `NextGenLab` assembly is shown in Figure 4.9. As we can see from this diagram, several of the classes inherit `WebControl`, all these classes can be instantiated directly in a Web page by using a tag. The classes in the `NextGenLab` are grouped in different namespaces, shown in 4.1.

Table 4.1 Namespaces and classes in NextGenLab Assembly

Namespace	***Classes***
NextGenLab	BrowserCheck,ExperimentPage, ExperimentStatus, ShowExperimentXML, ShowGraphFromXML, SetupPage
NextGenLab.ExpControl	Layout, TextLayout, GraphLayout
NextGenLab.Experiments	Experiment, ExperimentCollection, ExperimentXML
NextGenLab.Experiments.AnCMOS	CommonMethods, FreqResAnCMOS, FreqResAnCMOSAllGain
NextGenLab.ExperimentSetups	ExperimentSetup,SetupCollection
NextGenLab.ExperimentSetups.Setups	AnCMOSSetup
NextGenLab.Instruments	Agilent6625A, AnCMOSDAQ, Instrument, LabServer_1_B306, RsZVRE

NextGenLab

The classes in this namespace (see Table 4.4.1) are, with the exception of `BrowserCheck`, Web Controls that are instantiated by the NGL Web pages. The `ExperimentPage` class handles what experiment to draw, controls the running of experiments and displays the graph. When trying to implement a queue to prevent two or more experiments to run simultaneously we experienced performance problems. With a queue in place, the overhead on performing an experiment was approximately 1-2 seconds per experiment. This may not seem much but the prototype was intended to be used as a lab for 30 students, and you could easily end up waiting for a minute before the results were returned. The queue we implemented also had a tendency to hang if several experiments were run at the same time from the same computer. The queue idea was abandoned in favor of a fast and simple approach. In the NGL Web application there is a global boolean variable that signals if an experiment is running. The `ExperimentPage` object checks this variable; if an experiment is running it waits for 300ms and tries again. When it finds that no experiment is running, it takes control and runs the experiment. To ensure thread safety, it locks the access to the global variable, sets the global variable and unlocks the global variable. To give the user information on what the experiment is currently doing, it updates the status field provided by the `ExperimentStatus` Web Control.

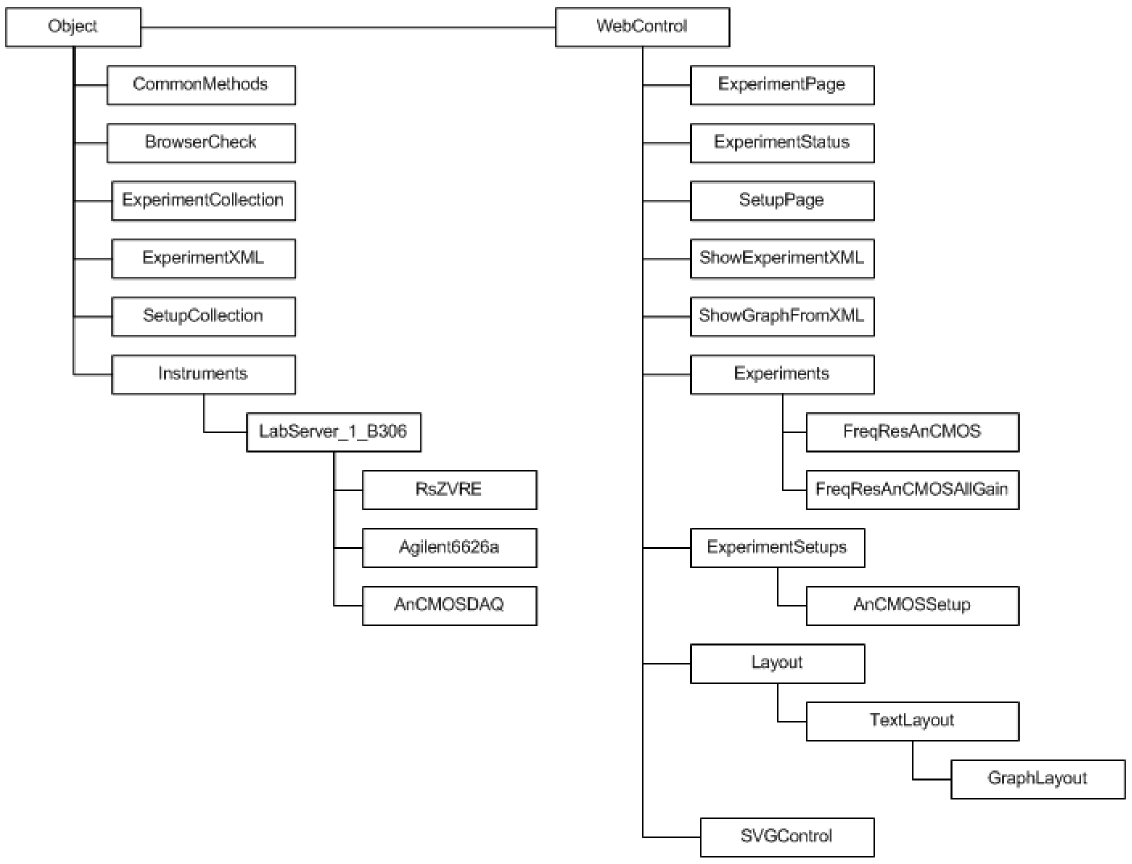


Figure 4.9. Class hierarchy

The classes `ShowExperimentXML` and `ShowGraphFromXML` are two Web Controls that allow the user to review the experiments performed in the session. In other words, you can return to results received 2 minutes ago and have another look. The stored results disappear when the user closes the browser window or the session time expires (20 minutes of inactivity).

The class `BrowserCheck` is a class that provides methods to check the browser capabilities and the browser type. `SetupPage` is a class that is essentially identical to the `ExperimentPage` but controls the experiment setups.

NextGenLab.ExpControl

This assembly contains the classes explained under the Web application presentation, which provides the experiments with methods to create a form that the user can interact with.

NextGenLab.Experiments

The `Experiments` namespace contains the framework for adding experiment classes. `Experiment` is the base class for all experiments, and all classes that inherit `Experiment` are automatically available through the menu. `Experiment` is an abstract class and cannot be instantiated directly. It contains four abstract methods that a child must implement. The abstract methods are: `run`, `drawExperiment`, `SetupExperiment`, and `IsOnline`. The `run` method should contain the code for performing the experiment, extracting the values from the instruments and creating an XML string that is passed on to the `SVGControl`, which is the Web Control that creates the plots. The `drawExperiment` method must contain the code for creating the form that the users utilize to interact with the experiment. In the spirit of simple creation of experiments, the `drawExperiment` method uses an instance of the `TextLayout` class to add textboxes, dropdown lists and headers. An experiment can also choose to implement an overloaded version of the method `drawExperiment` that takes a `GraphLayout` object as the input parameter. Thus creating a graphical layout if the browser can handle it. The `SetupExperiment` method should parse the submitted values and prepare the experiment for running. The `IsOnline` method should implement the code for calling the `IsOnline` method in the instrument classes used by the experiment, which checks if the instruments are turned on and ready.

The class `ExperimentCollection` makes it possible to extract all classes that inherit `Experiment`. An example with this class is shown later.

The class `ExperimentXML` is a class that simplifies the creation of the XML string that the `SVGControl` must have to draw a graph.

NextGenLab.Experiment.AnCMOS

This namespace contains the classes for the experiments implemented in the NGL prototype. They are all derived from the `Experiment` class and contain the code for controlling the instruments and retrieving results from them.

NextGenLab.ExperimentSetup & NextGenLab.ExperimentSetups.Setups

This namespace contains the classes for providing a general setup for the experiments, i.e. biasing the Analog CMOS chip. This has been separated from the experiment to make the actual running of the experiment faster. If the same instrument is to be used in several different experiments, it is possible to implement the biasing and setup in the individual experiments.

NextGenLab.Instruments

The `Instruments` namespace contains the classes for providing an easy interface to the instrument hardware. `Instrument` is the base class for all instruments and it mainly provides globalization of the decimal point such that a period is always used in conversion between string and double. `LabServer_1_B306` is a class that instantiates an object of the LabServer Web Service and exposes the methods provided by the server that is connected to the actual instruments. The classes `Aglient6625A` (power supply), `AnCMOSDAQ` (DAQ board) and `RsZVRE` (Vector Network Analyzer) provide easy-to-use methods for retrieving and setting values of the respective instruments. This limits the users ability to perform illegal or damaging operations, but also makes it easy for a user to interact with the instruments without knowing how to write GPIB strings. An example of retrieving the magnitude from the `RsZVRE` follows:

With the `RsZVRE` class:

```
RsZVRE rs = new RsZVRE();
rs.getResultsFromZVRE();
ArrayList xVals = rs.XValues;
ArrayList magn = rs.Magnitude;
```

Without the `RsZVRE` class:

```
HardInter hi = new HardInter(); //from proxy class Labserver_1_B306
int device = hi.connectGPIB(20);
string xvals = hi.QueryGPIB(device,"TRAC:STIM? CH1DATA",7500);
string yvals = hi.QueryGPIB(device,"TRAC? CH1DATA",16000);
```

At a first glance, it might not seem too time consuming to get the magnitude without the `RsZVRE` class, but the y values returned by the second GPIB call are structured in a comma-separated string with alternating complex and real values. Therefore, to extract the magnitude, you have to separate the values and calculate the magnitude. This is, of course, not a difficult task, but having to write this code several times in different experiments makes it time consuming. The respective instrument classes also implement error handling.

ExperimentCollection and adding experiments

As mentioned earlier, one of the main goals of the prototype was to make it simple to add experiments. Thus it was necessary to extract all experiments in the `NextGenLab` assembly and to choose at runtime what experiment should be run. Fortunately, the .NET framework makes this relatively simple. The class `ExperimentCollection` retrieves all classes that inherit `Experiment` through reflection when it is instantiated. The menu in the NGL application uses this class to retrieve names of the experiments in the `NextGenLab` assembly. Selecting an experiment is done by passing a GET variable to the `ExperimentPage` , i.e. `experiment.aspx?ExpID=1`. The Web Control `ExperimentPage` uses `ExperimentCollection.CreateInstance` to create an instance of the experiment. This instance is casted to the `Experiment` class but because of polymorphism, it retains its methods and variables. The `ExperimentPage` treats this instance as an object of the `Experiment` class, which is the reason why an experiment must implement the methods mentioned earlier. If it does not implement these methods, the `ExperimentPage` object would not know how to interact with the experiment it just instantiated.

Here follows an example on the use of ExperimentCollection and polymorphism:

The client requests the page: `experiment.aspx?ExpID=1`

```
public class ExperimentPage{
.
.
//Get index
int iExpID = Int32.Parse(this.Page.Request.QueryString("ExpID"));

//Create instance of ExperimentCollection
ExperimentCollection oExpCol = new ExperimentCollection();

//Get an instance of the selected experiment
Experiment oExp = oExpCol.CreateInstance(iExpID);
.
.
//Run The Experiment
oExp.run();
.
.
}
```

As shown in this example, the `ExperimentPage` object does not need to know what type of experiment it has instantiated, since they are all casted to `Experiment`. This works as long as all experiments have the same interface.

The use of reflection and casting in the NGL prototype has resulted in a very simple model for adding experiments. The procedure is shown in the following short recipe:

- Create a class that inherits `Experiment`.
- Implement four methods.
- Implement three static properties.
- Write the experiment logic in the `run()` method.
- Compile with the namespace `NextGenLab.Experiments.NameOfNewExperiment.`
- Copy the compiled dll file to the bin folder of the NGL Web application.

- Test the experiment through the NGL Web application.

An experiment can be written in any of the languages that support the .NET platform.

4.4.3 Client-side-graphics

The graph module was implemented as a Web Control, thus it is easy to reuse. The `SVGControl` Web Control, parses an XML-string and writes data to the browser needed to create the plot. A JavaScript is responsible for plotting the result by means of SVG.

SVGControl class

The class `SVGControl` is responsible for displaying the graphs. `SVGControl` receives an XML string from the experiment. The XML string is parsed and the results are sent to a JavaScript, which displays the results by means of SVG. The XML-string contains information about the size, axis-labels and type (linear or logarithmic) of each of the graphs. `SVGControl` uses the size information to calculate the total height and width of the SVG object. An example of this XML file was given under 4.2.2 XML. Below you can see an excerpt from the `SVGControl`. The `<embed>` tag is used to embed the SVG into the html-page.

```
public class SVGControl : System.Web.UI.WebControls.WebControl
{
.
.
        protected override void Render(HtmlTextWriter output)
        {
        .
        .
              //   Parsing   the   XML-string,   writing   values   to
javascript.
```

```
                    output.Write("<embed   name='Graph'   class=svgControl
                    pluginspage='http://www.adobe.com/svg/viewer/install/
                    '  align='top'  src='result.svg'  width='"+max_width+"'
                    height='"+max_height+"' type='image/svg-xml'>");
              .
              .
              }
      .
      .
      }
```

The plotting and the interaction functionality of the graphs are obtained from JavaScript functions. Below is an overview of these functions.

`function scale_and_plotaspx() {}:`
This is the "main" function, which calls `plot()` for each graph.

`function adjust_value_and_compute_prefix(axis_val_temp) {}`
This function adjusts the axis-value and returns the correct prefix. The `SVGControl` can handle values between 1e-25 and 1e27. To improve readability, the axis-values are adjusted. For example 1200000 is adjusted to 1.2M.

`function plot(k, r, gr, x0_outer, y0_outer) {}:`
This function draws the frame, axes, labels and gridlines. This function contains functionality to make the graph appear as consistent as possible. For example:

axis-values after `adjust_value_and_compute_prefix()`: 0, 500m, 1, 1.5, 2, 2.5

axis-values after further adjusting in `plot()`: 0.0, 0.5, 1.0, 1.5, 2.0, 2.5

As we can see, all values get the same prefix.

`function prompt_x(evt) {} and function prompt_y(evt) {}:`
These two functions are used to adjust the x- and y-axes manually. The function executes when the user clicks on the respective axis. The appearance of the prompt is depending

on whether the axis is linear or logarithmic. In the case of a linear axis, the user is prompted for the minimum value, maximum value and the step value. The step value represents the range between each gridline. In the case of a logarithmic axis, the user is only prompted for the minimum and the maximum decade. The user input is validated, and illegal values are rejected. To improve user-friendliness, these functions understand both comma and period as the decimal point.

`function scale(min, max) {}:`
This quite complicated function performs auto-scaling. It returns new minimum and maximum values and an appropriate step value.

4.4.4 LabServer Web Service

In the quest for finding a viable and robust solution to make the methods of the GPIB and DAQ card available to a .NET application, several solutions were explored. The initial idea was to use COM objects for the GPIB and DAQ board that exposed the methods available through the National Instruments C++ libraries. This turned out to be a solution that worked, but was not nearly as robust as a Web application demand. Using COM objects demanded that the NGL Web application was run on the machine that had the DAQ and GPIB boards installed. Several attempts were made to make the COM solution more reliable and robust, but the attempts failed.

As the quest progressed, the use of a Web Service emerged as possible solution. The DAQ and GPIB methods were implemented and exposed through a C++ Web Service. This turned out to be a very viable and robust solution. As a by-product of using Web Service to interface with GPIB and DAQ card, the ability to have several experimental setups connected to different Lab servers spanning wide geographical areas emerged with no additional costs. In addition, the LabServer is lightweight and can be run on an old computer. The NGL LabServer is currently running on a Pentium 166Mhz with Windows 2000 Professional.

The prototype `LabServer` exposes five GPIB methods and five DAQ methods. Here follows the GPIB methods and an example:

GPIB Methods:

```
int ConnectGPIB( int DeviceAddress);
int WriteGPIB(int DeviceHandle, string CommandStr);
int CmdGPIB(int DeviceHandle, string CommandStr);
string QueryGPIB(int DeviceHandle, string CommandStr, long
ReturnLength);
string ReadGPIB(int DeviceHandle, long ReturnLength);
int ReleaseGPIB(int DeviceHandle);
```

Use of the Web Service GPIB methods in C#:

In Visual Studio .NET adding a reference to a Web Service is as simple as adding a reference to a local COM or .NET component.

This example shows how to read the identity of a device connected to the GPIB bus at address 5:

```
using LabServer;
.
class LSTest{
.
public void Main(){
  Console.Write("Reading Identity: ");

  //create an instance of the Web Service
  HardInter oHInter = new HardInter();
  string sCommandStr = "*IDN?"
```

```
        //Connect to the device at address 5
        int iDevice = oHInter.ConnectGPIB(5);

        //Get the Identity
        string sReturnedStr = oHInter.QueryGPIB(iDevice,sCommandStr,20);
        oHInter.ReleaseGPIB(iDevice);
        Console.WriteLine (sReturnedStr);
      }
      .
      .
      }
```

Console output if a Rode & Schwarz ZVRE was connected with the GPIB address 5:

```
C:\>LSTest.exe
Reading Identity: Rode & Schwarz,ZVRE,8
C:\>
```

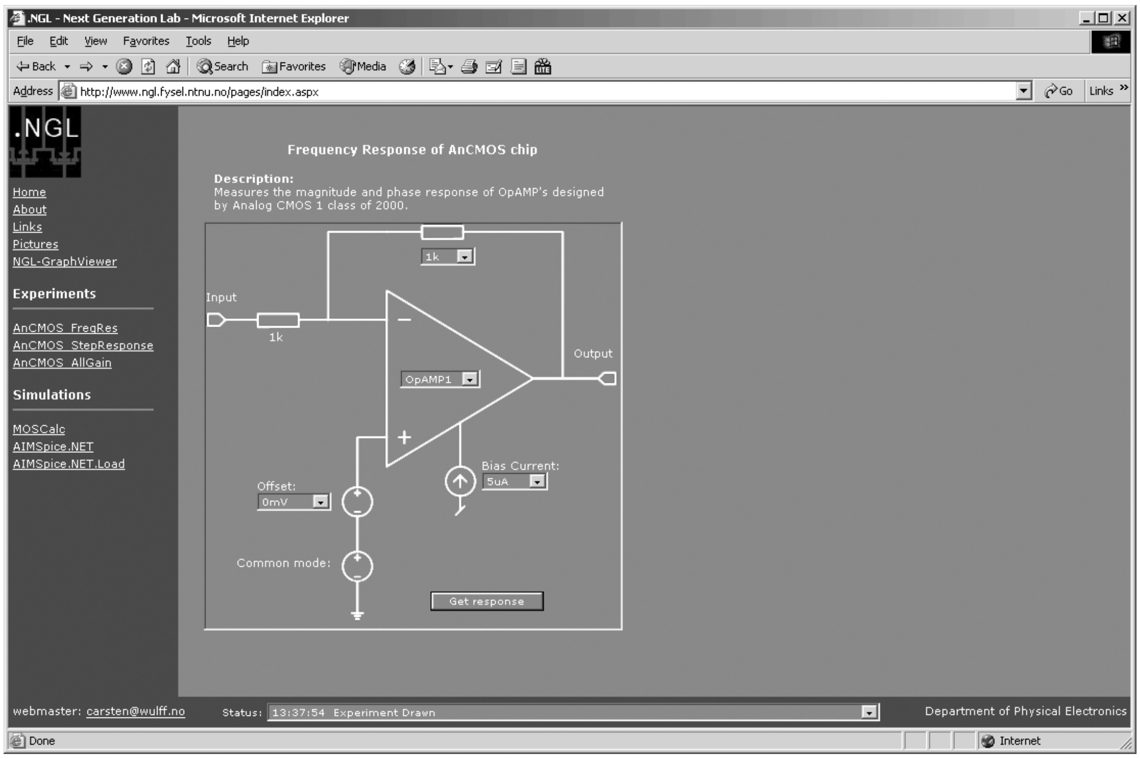
In theory any programming language can be used to access the Web Service since the method calls use HTTP and SOAP in a normal request/response scenario.

4.5 THE FREQUENCY RESPONSE EXPERIMENT

As mentioned, the prototype experiment is devoted to measuring the frequency response of operational amplifiers. In this section, we will explain how to perform the experiment, show results from the frequency response experiment and explain how to interpret the results.

To begin the experiment, we press the AnCMOS_FreqRes link on the left-side menu of the NGL opening page (see Figure 4.5), Figure 4.10 shows the start page for this experiment. Here we can specify the circuit parameters to use when running the experiment. For example, if we select: 20dB gain (10k resistor) and OPAMP1, we will get the result shown in Figure 4.11. While the experiment is running, we get feedback from the server in the status line at the bottom of the browser. The status information only works in IE 5.0 and later versions. In Figure 4.12, we show an example on how the status line might appear. The marked line indicates when the experiment was started (we pressed "Get Response").

From the results in Figure 4.11, we see that the frequency sweep starts at 10KHz, this is because there is a DC stop filter connected between the test circuit and the vector network analyzer. The effects of this stop filter and the coaxial cables in the frequency range 10KHz – 40MHz have been eliminated through calibration of the experimental setup.


.NGL - Next Generation Lab - Microsoft Internet Explorer

Address: http://www.ngl.fysel.ntnu.no/pages/index.aspx

Frequency Response of AnCMOS chip

Description:
Measures the magnitude and phase response of OpAMP's designed by Analog CMOS 1 class of 2000.

Offset: 0mV
Bias Current: 5uA
OpAMP1
1k

webmaster: carsten@wulff.no

Status: 13:37:54 Experiment Drawn

Department of Physical Electronics

Figure 4.10. Start page frequency response experiment

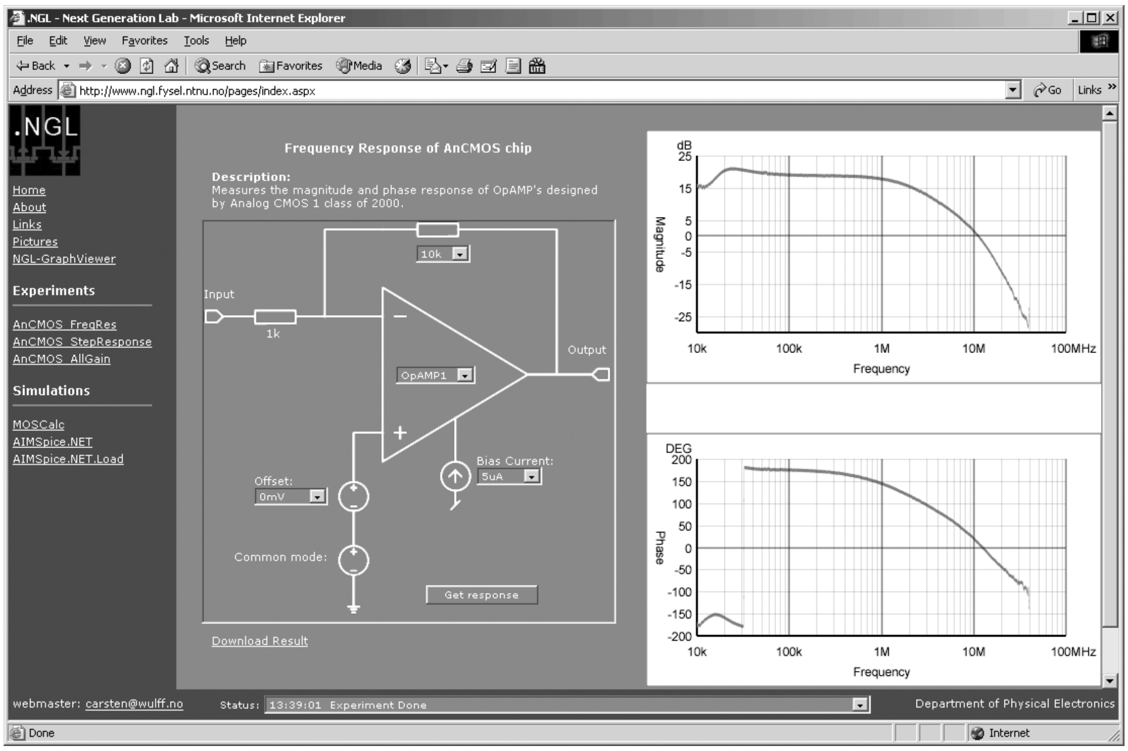
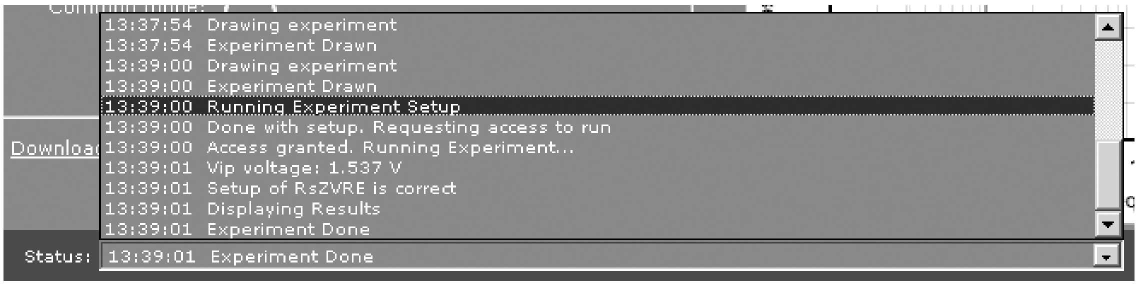

.NGL - Next Generation Lab - Microsoft Internet Explorer
http://www.ngl.fysel.ntnu.no/pages/index.aspx
Home
About
Links
Pictures
NGL-GraphViewer
Experiments
AnCMOS FreqRes
AnCMOS StepResponse
AnCMOS AllGain
Simulations
MOSCalc
AIMSpice.NET
AIMSpice.NET.Load
Frequency Response of AnCMOS chip
Description:
Measures the magnitude and phase response of OpAMP's designed by Analog CMOS 1 class of 2000.
10k
Input
1k
Output
OpAMP1
Bias Current:
5uA
Offset:
0mV
Common mode:
Get response
Download Result
Magnitude
dB
Frequency
Phase
DEG
10k
100k
1M
10M
100MHz
webmaster: carsten@wulff.no
Status: 13:39:01 Experiment Done
Department of Physical Electronics

Figure 4.11. Example result from the frequency response experiment



```
13:37:54 Drawing experiment
13:37:54 Experiment Drawn
13:39:00 Drawing experiment
13:39:00 Experiment Drawn
13:39:00 Running Experiment Setup
13:39:00 Done with setup. Requesting access to run
13:39:00 Access granted. Running Experiment...
13:39:01 Vip voltage: 1.537 V
13:39:01 Setup of RsZVRE is correct
13:39:01 Displaying Results
13:39:01 Experiment Done
Status: 13:39:01 Experiment Done
```

Figure 4.12. Example of status line output

We will now show how to determine the bandwidth (-3 dB frequency), phase margin, the 0 dB frequency (first order equivalent to open-loop-gain bandwidth product) and some example experiments. We concentrate on OPAMPS 1 and 2 because these two are the most sensitive to change in offset and bias current.

To determine the bandwidth, we first find the –3 dB frequency. From Figure 4.13 we can see that the OPAMP has a stable gain at 19-20 dB, which gives a –3 dB frequency at around 2 MHz and hence a bandwidth of 2 MHz. The 0dB frequency is at 6.5 MHz. To determine the phase margin in this case we find the phase at the 0dB frequency (6.5 MHz). We can see that the phase margin is about 20 degrees. This is a very low phase margin and may lead to instabilities.

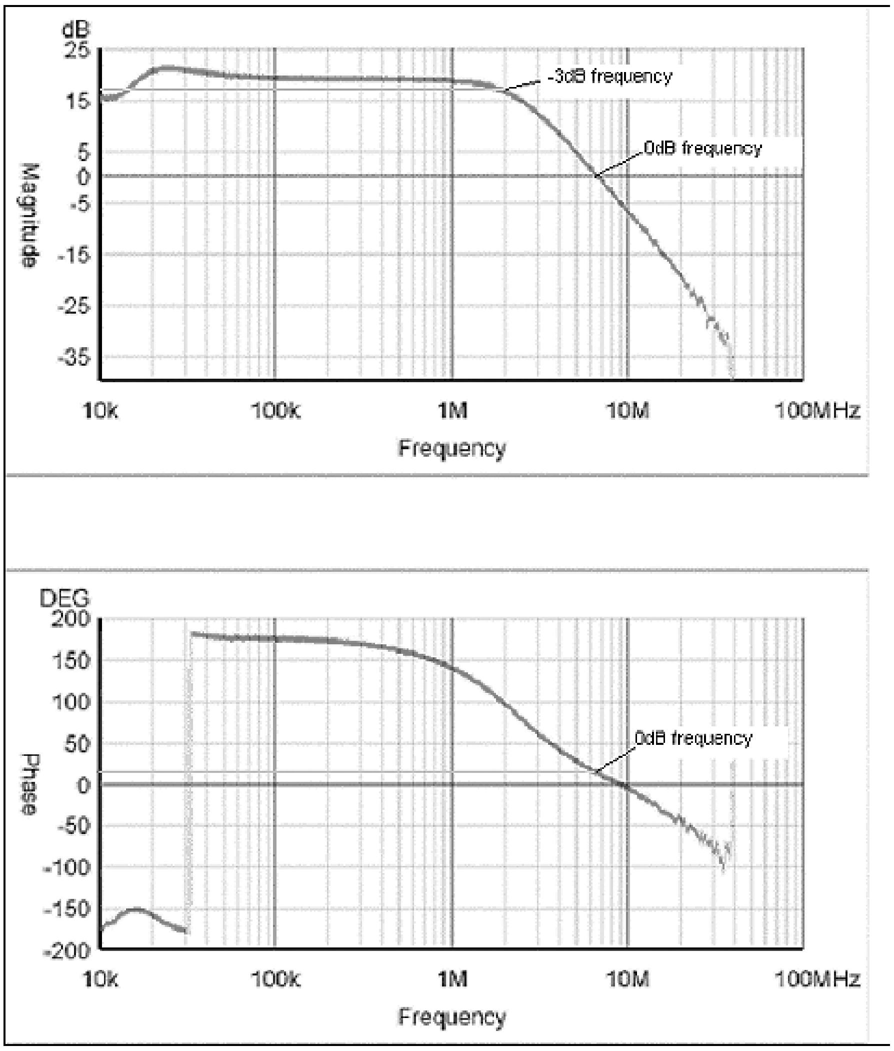


Figure 4.13. Finding bandwidth, phase margin and the 0-dB frequency

We choose OPAMP 2 to investigate what happens when we adjust the bias current to 0.5uA, 5uA and 25uA with 20dB gain. We see from the magnitude plots in Figure 4., Figure 4. and Figure 4. that the bandwidth and the 0dB frequency increase as we increase the bias current. This is to be expected from the first order equation of the unity-gain frequency:

$$w_t = \frac{g_m}{C_L} \text{ where } g_m = \sqrt{2 \cdot \mu_n} \cdot C_{ox} \cdot \frac{W}{L} \cdot I_d$$

The resonance peek in Figures 4.15 and 4.16 around the –3 dB frequency indicates instabilities of the OPAMP.

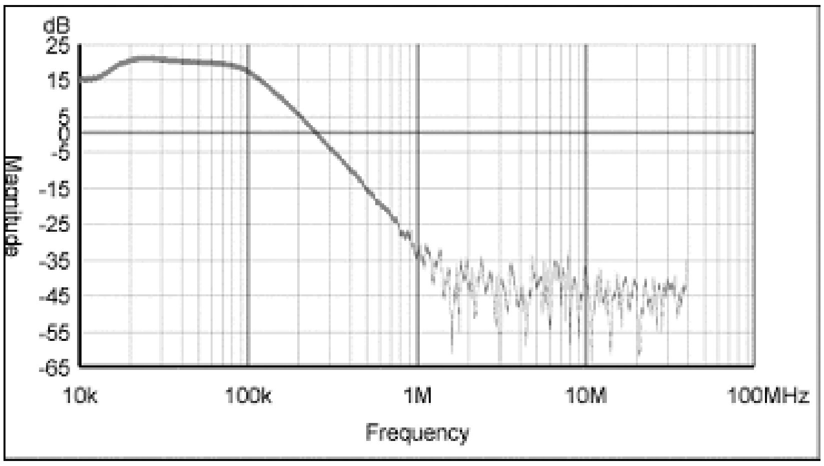


Figure 4.14. Magnitude response with a bias current of 0.5uA.

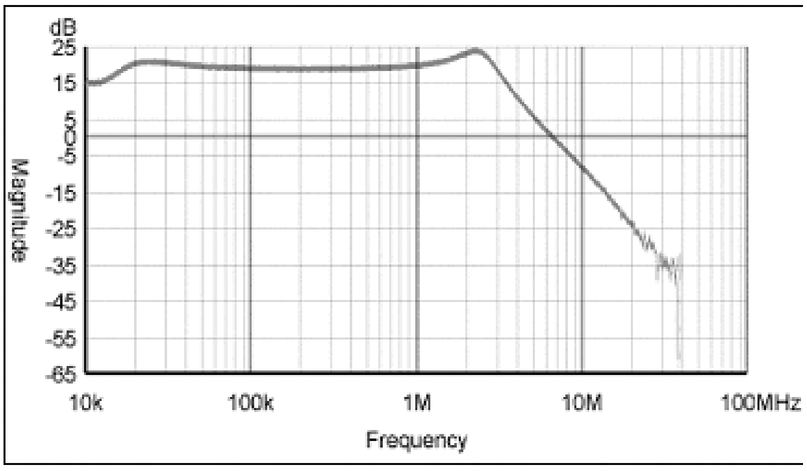


Figure 4.15. Magnitude response with a bias current of 5uA.

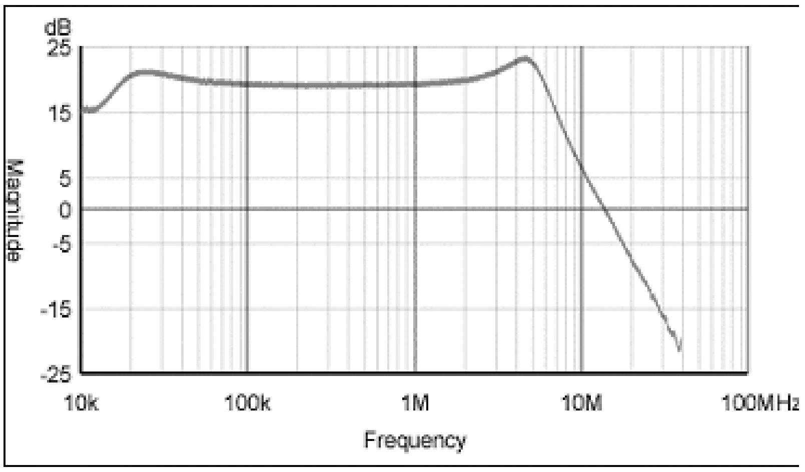


Figure 4.16. Magnitude response with a bias current of 25uA.

To further illustrate the sensitivity of these OPAMPS and possible effects of an increase in bias current we also study OPAMP 1 at a bias current of 5uA and 25uA with the gain set at 0dB.

In Figure 4. we can see a resonance peak. The relatively low value of about 3 dB indicates that the amplifier still has some phase margin and is not very unstable. In contrast, the plot in Figure 4. shows a circuit configuration, which is very unstable and has resonance peaks at 10, 20, and 30 MHz. These three peaks indicate that the circuit has a resonance frequency at 10 MHz while the first two harmonics are at 20 and 30MHz. The gain of 40 dB is far beyond the open loop gain at this frequency and thus another sign of oscillation. The high value is the result of a low phase margin, which results in positive feedback through the feedback resistor.

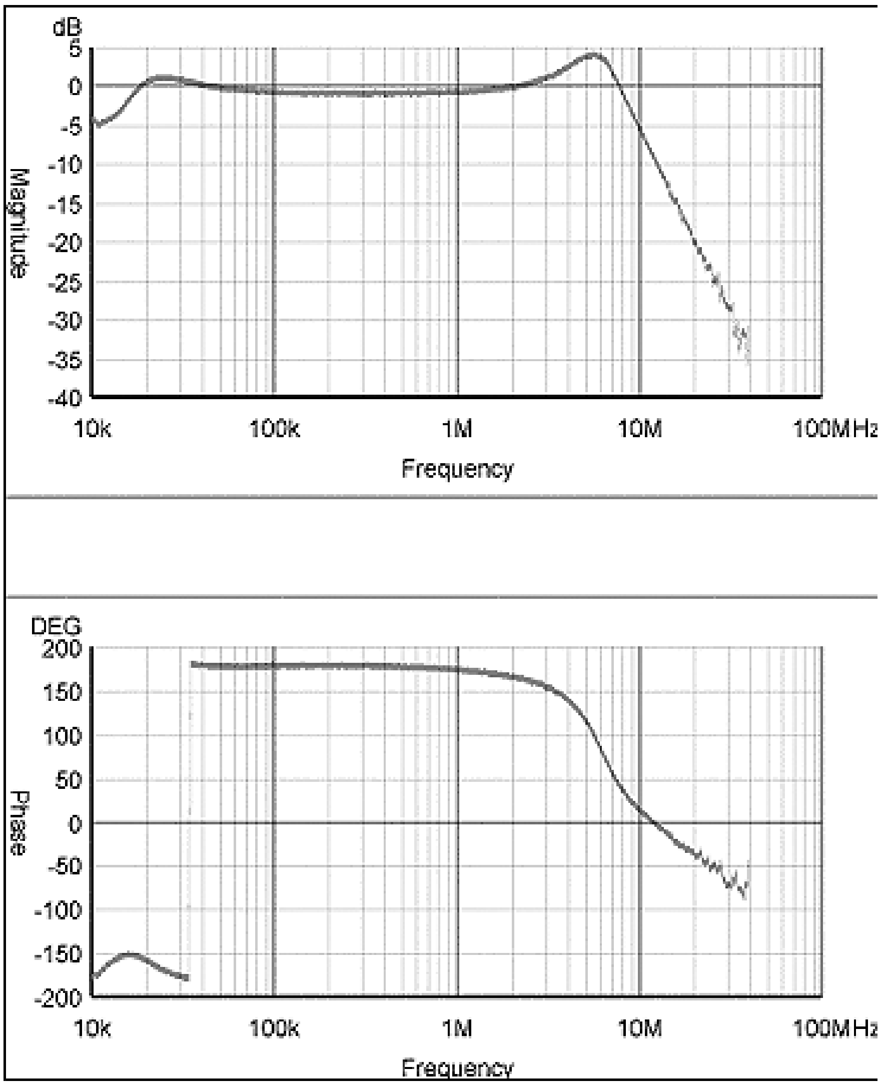


Figure 4.17. Frequency response with a bias current of 5uA.

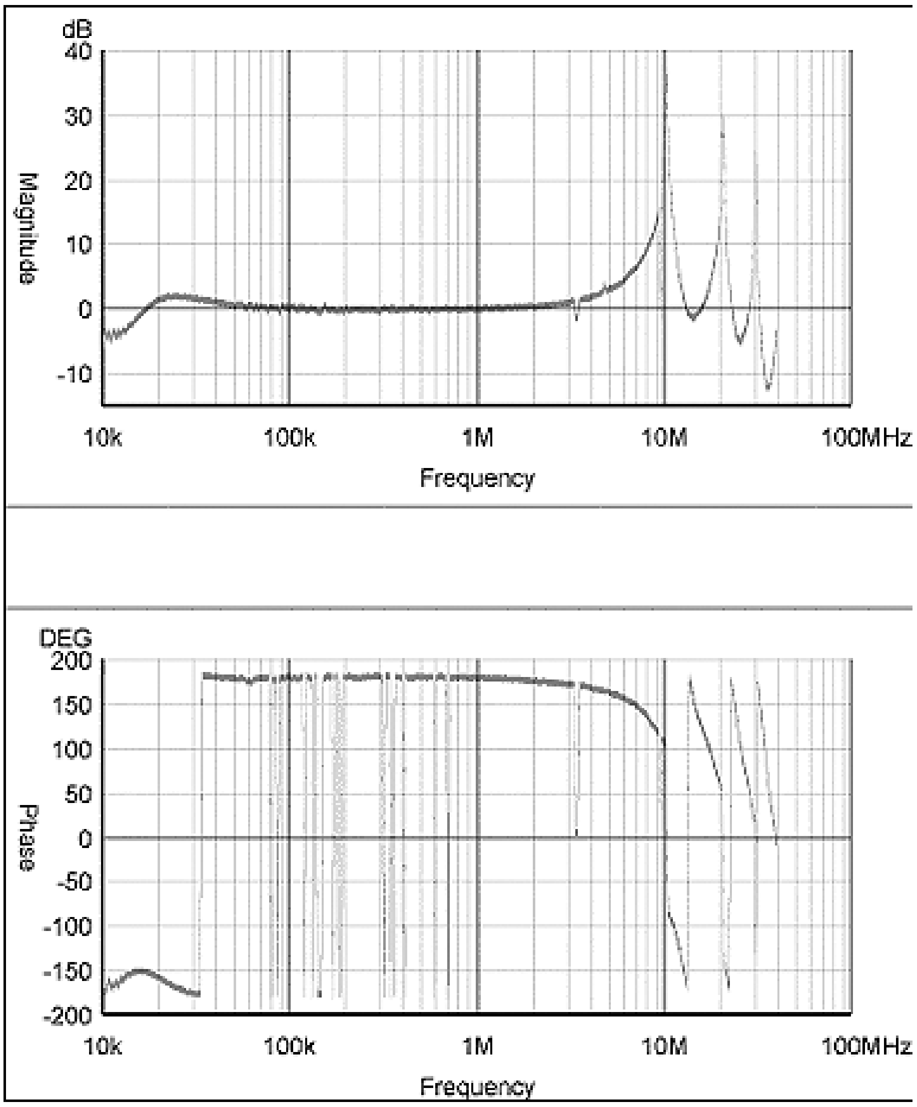


Figure 4.18. Frequency response with a bias current of 25uA.

4.6 LAB ASSIGNMENT

This Section gives an example of a laboratory assignment given in the Analog CMOS 1 course at NTNU in the fall of 2001.

To start the lab go to http://www.ngl.fysel.ntnu.no and select Experiment->AnCMOS_FreqRes.

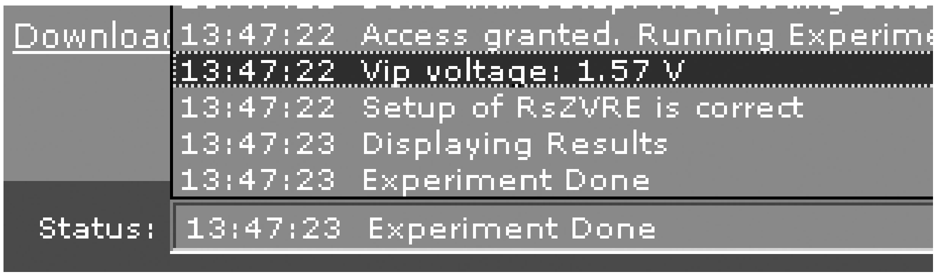
NOTE: Almost all questions have short answers.

Tips:

- On the graph use “ctrl + left mouse button” to zoom in, “ctrl + shift + left mouse button” to zoom out, or right mouse button to show a menu.

Effects of offset from common mode

Tip: you can see the actual voltage value of the positive input displayed in the status field.



```
Downloa 13:47:22  Access granted. Running Experim
        13:47:22  Vip voltage: 1.57 V
        13:47:22  Setup of RsZVRE is correct
        13:47:23  Displaying Results
        13:47:23  Experiment Done
Status:  13:47:23  Experiment Done
```

1a.

Calculate the expression for the gain of the circuit in Figure 4.3.

1b.

Select OPAMP 4 with 20dB gain and run the experiment. You should see a proper frequency response. Change “Offset” to +100mV and run again. What happened? Why?

1c.

Change to +10mV and run again. The frequency response should return. Change gain to 60 dB and run again. What happened? Why?

1d.

At what offset voltage should the AC gain “disappear” if you are using 20 dB gain, common mode 1.5V and 3V power supply? What about for 60 dB gain?

1e.

Do your findings in 1d match what you saw in 1b and c?

Effects of Bias Current

2a.

Select OPAMP 1 and gain 20 dB. You should see a similar frequency response as for OPAMP 4. Adjust the bias current up and down. What happens? Why?

2b.

Select 0dB gain, 5uA bias current. Increase the bias current step by step. What happens? Why?

Phase Margin

Use estimates in these questions, e.g. OPAMP 11 phase margin = 60-65 degrees

3a.

Estimate the phase margin of OPAMP 1 and OPAMP 2 using 20dB gain.

3b.

Is the phase margin in these two cases sufficient to ensure that the circuit never oscillates?

4.7 EXPERIENCES WITH NGL

At the time of writing, NGL has been running for 12 months. During this time, it has been used in the course Analog CMOS 1 at NTNU. The feedback from students has been positive and it has underlined the effectiveness of a remote laboratory as a teaching tool. NGL provides students with an intuitive understanding of what happens when one adjusts parameters like bias current and offset from common mode on operational amplifiers. In addition, it underlines the importance of parameters like phase margin and it shows the effects when this parameter is too low.

The .NET platform has proven to be a very stable platform, this in spite of the fact that from august 2001 to march 2002 the beta 2 version of the .NET platform was used.

4.8 CONCLUSION AND PLANS FOR THE FUTURE

The NGL Web application provides users with a reliable and efficient tool for analog integrated circuit experiments. It gives lecturers and students the opportunity to perform real-time experiments on actual circuits, using industrial standard measurement equipment.

The NGL Web application provides a framework for adding new experiments and experiment setups. Choosing the .NET platform as server-side technology provides distributed architecture with no additional cost. The NGL Web application is scalable and easy to use.

The NGL is continuously updated and several plans exist for the future. At the time of writing, work is being done in three fields, time-domain experiment, browser independency and custom circuit setup with a programmable analog integrated circuit.

The time-domain experiment measures the step response of the OPAMP circuit, the step-response is used to measure the phase margin more accurately than the frequency response measurement.

To improve browser independency, a module is being developed that creates plots server-side as bitmaps for browsers that do not support SVG. The server-side bitmap generator coexists with the `SVGControl` and the decision on which to use is done by checking the current browsers capability.

The long-term vision of NGL is to use programmable analog integrated circuits, thus providing a student with several circuits to choose from e.g. comparator, sample and hold or DAC (Digital to Analog Converter). These can be measured alone or be combined into larger circuits like an ADC (Analog to Digital Converter). A programmable analog integrated circuit named PanIC [10] is scheduled for production November 2002. A prototype of a custom circuit lab is scheduled for spring/summer of 2003.

4.9 ACKNOWLEDGEMENTS

The project was funded by the Nordunet project "Internet Technology in Laboratory Modules for Distance-Learning", principle investigator Prof Tor A. Fjeldly, and the Department of Physical Electronics, NTNU.

4.10 REFERENCES

[1] V. Kristiansen, "*Remotely operated experiments on electric circuits over the Internet - An implementation using Java*", M. Sc. thesis, Norwegian University of Science and Technology (1997).

[2] B. Dalager, "*Remotely operated experiments on electric circuits over the Internet - Realizing a client/server solution*", M. Sc. thesis, Norwegian University of Science and Technology (1998).

[3] H. Shen, Z. Xu, B. Dalager, V. Kristiansen, Ø. Strøm, M.S. Shur, T.A. Fjeldly, J. Lü, T. Ytterdal , "*Conducting Laboratory Experiments over the Internet*", IEEE Trans. on Education, 42, No. 3, pp. 180-185 (1999).

[4] T.A. Fjeldly, M.S. Shur, H. Shen and T. Ytterdal, "*Automated Internet Measurement Laboratory (AIM-Lab) for Engineering Education*", Proceedings of 1999 Frontiers in Education Conference (FIE'99), San Juan, Puerto Rico, IEEE Catalog No. 99CH37011(C), 12a2 (1999).

[5] M.S. Shur, T. A. Fjeldly and H. Shen, "*AIM-Lab - A System for Conducting Semiconductor Device Characterization via the Internet*", late news paper at 1999 International Conference on Microelectronic Test Structures (ICMTS 1999), Gothenburg, Sweden (1999).

[6] T.A. Fjeldly, M.S. Shur, H. Shen, and T. Ytterdal, "*AIM-Lab: A System for -Remote Characterization of Electronic Devices and Circuits over the Internet*", Proc. 3rd IEEE Int. Caracas Conf. on Devices, Circuits and Systems (ICCDCS-2000), Cancun, Mexico, IEEE Catalog No. 00TH8474C, pp. I43.1 -I43.6 (2000)

[7] K. Smith, J.O. Strandman, R. Berntzen, T.A. Fjeldly, M.S. Shur, "*Advanced Internet Technology in Laboratory Modules for Distance-Learning*," accepted for presentation at the American Society for Engineering Education Annual Conference & Exposition 2001, ASEE'01".

[8] T. A. Fjeldly, J. O. Strandman, R. Berntzen,b "*LAB-on-WEB – A Comprehensive Electronic Device Laboratory on a Chip Accessible via Internet*", International Conference on Engineering Education (ICEE2002), http://www.lab-on-web.com/

[9] C. Wulff, T. Ytterdal, T. A. Sæthre, A. Skjevlan, T. A. Fjeldy, M. S. Shur, "*Next Generation Lab – A solution for remote characterization of analog integrated circuits*". International Caracas Conference on Devices, Circuits and Systems (ICCDCS2002).

[10] C. Wulff, T. Ytterdal, "*Programmable Analog Integrated Circuit for use in remotely operated laboratories*". International Conference on Engineering Education, (ICEE2002)