GREY-MODEL BASED ICE PREDICTION SENSOR SYSTEM ON WIND TURBINE SYSTEM

by

Chao Feng

Submitted in partial fulfillment of the requirements

for the degree of Master of Science

Thesis Advisor: Dr. Chris Papachristou

Department of Electrical Engineering

CASE WESTERN RESERVE UNIVERSITY

January, 2012

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Chao Feng

candidate for the Master of Science degree *.

(signed) Chris Papachristou

(chair of the committee)

Mario Garcia Sanz

Francis Wolff

(date) 12/13/2012

*We also certify that written approval has been obtained for any proprietary material contained therein.

Contents

List of Tables

List of figures

Grey-Model Based Ice Prediction Sensor System on Wind Turbine

Abstract

by

Chao Feng

Ice is an important factor for wind turbine system health monitoring. Ice should be predicted and removed before it forms on the blades. If ice forms on the axle, it will give a friction force on the axle and damage the electrical system.

My objective is to design and implement an ice detection sensor system to prevent the ice forming on wind turbine. Several fiber optic sensors are chosen to measure side parameters, input to a grey-model based prediction module to get the predicted values, and send them to LEWICE system to predict the ice shape.

Key Word: fiber optic sensors, side-parameter analysis, grey model

Chapter 1

Introduction

1.1 Icing Problems on Wind Turbine

The best locations for wind turbines systems are the exposed locations, but these locations are easy to form ice on the blades. It will cause a lot of problems if ice is forming on the blades, such as: complete loss of production, disrupted aerodynamics caused reduction of power, overloading due to delayed stall, more and more fatigue components because of the imbalance in the ice load, and most important, damage or harm caused by uncontrolled shedding of large ice chunks.

If the ice forming become critical, in the extreme condition, it is no possible to start the turbine, due to changed aerodynamics of the blades, with subsequent loss of all possible production for quite long period of time. In addition, the buildup of ice on the blades of the turbine disturbs the aerodynamics, which can either reduce the amount of power produced or overload the turbine if it is stall regulated. The increased fatigue loads on all components of a wind turbine operating with an unbalanced ice load on the blades has been presented as a problem where the effects are difficult to predict due to general lack of knowledge regarding the intensity and duration of icing events. The last

problem from icing does not concern the wind turbine itself, but is the risk posed by uncontrolled shedding of ice chunks. These are of special danger to service personnel, but may also affect public acceptance towards wind power if the danger requires fencing off large areas around the wind turbines.

Measures to prevent icing have been used, and have been shown to work effectively. In addition, people have presented new methods for deicing. With all of the methods that do not operate continuously it has pointed out the need for a reliable icing detector to activate the deicing system. Various sensors have been tested, but have not performed satisfactorily.

There are two main types of atmospheric ice accumulation, people define them traditionally. These two types are in-cloud icing and precipitation icing.

The main icing mechanisms of interest for wind turbine applications are as follows:

- In-cloud icing: hard rime, soft rime, Glaze
- Precipitation icing: wet snow freezing rain [1]

In-cloud icing occurs when small, super cooled, airborne water droplets. It will make up clouds and fog, freeze upon impacting a surface then allow formation of ice. These water droplets can remain liquid in the air at temperatures down to −35 °C due to their small size, but will freeze and strike a surface which provides a crystallization site.

The different types of rime and glaze are formed depending on the droplet sizes and the energy balance of the surface in question. For small droplets with almost

instantaneous freezing, soft rime forms. With medium sized droplets and slightly slower freezing, hard rime forms. If the buildup of rime is such that a layer of liquid water is present on the surface during freezing, glaze forms. [2]

Precipitation icing [3] is due to rain or snow freezing on contact with a surface. Precipitation icing can have much higher rates of mass accumulation than in-cloud icing, with possibly greater resulting damage. Relative frequency for the two types of icing is dependent on geographic location and climate. Wet snow can stick to surfaces when in the temperature range of 0–3 °C, while freezing rain requires surface temperatures below 0 °C.

There are a lot of conditions that can form ice and there are a lot of different types of ice. Because this reason, there are a variety of methods to detect ice. But the most directly way is break the environment that can form ice. It is similar if the condition is met, we can surely predict, say ice will form.

1.2 Motivation

Nowadays, the most products of ice detector available in the market has a large size and consume a lot of power, it doesn't meet requirement to be installed on the blades. As we can see in Chapter2, the ice detector is most used on the freeway, which has a relatively flat and big surface. But if we need to install ice detector on blades, we should control the height of the sensor and the power consumption must be low. Since there

are not good sensors that can meet the requirement of installing on the blades, why we just abandon the ice sensor and design a replacement to detect ice? The topic of my thesis is trying to develop a method that can detect ice through side parameters, such as temperature, humidity and pressure.

1.3 Thesis Outline

In Chapter 2, I will introduce the basic idea of different type of sensors. Then I will introduce some fiber optic sensor products which are available in the market and can be used in wind turbine system. Then I will list one product of ice detector and analyze why these kind of ice detector cannot be used on wind turbine system.

In Chapter 3, first I will introduce the basic idea of grey model, and then I will introduce two methods to implement it-----C++ and Verilog. After the code introduction, I will experiment these code with TI temperature sensor to check the result accuracy.

In Chapter 4, I will introduce the LEWICE, a software that can predict the ice forming, with this software, we can combine the prediction of side parameters and give the final result.

In Chapter 5, I will introduce the future work of my thesis.

1.4 Contribution

In this thesis, two different methods to implement the grey model will be presented. A list of different types of sensors that can be used on the wind turbine system is also provided. With this thesis, it is a guide to build the entire ice detector sensor system, with all the important modules are introduced.

Chapter2

Sensors for Wind Turbine System

A sensor is a device which measures a physical quantity and converts it into a signal that can be measured. A sensor is basically an interface between the physical world and an electrical computing device. The technological developments in electronics have made it possible to deploy large number of low-power distributed sensing devices. Each of these sensing devices is also called a Sensor node and is capable of limited amount of processing. The Sensor nodes can be deployed in large number and can coordinate with each other to form a sensor network that can measure a physical environment in greater detail.

2.1 Introduction of Sensor Applications

Structural Health Monitoring (SHM)

SHM is a mechanism by which Civil and mechanical structures are continuously monitored to detect any possible damage or deterioration [4]. Sensor nodes are deployed in large number to collect damage sensitive measurements of a structure and then analysis is done to determine the health of the structure. The wireless sensor

network based approach for SHM has many advantages: low deployment and maintenance cost, large physical coverage, real time monitoring.

Health Applications

Sensors have many potential health applications that can revolutionize patient-care. They can be deployed to collect human physiological data, track and monitor patients inside a hospital etc. Some of the products that are based on sensors are Glucose monitor, Pulse oximeter, Cancer and General health monitors. Wireless biomedical sensors are also being implanted into human body. An essential requirement for such a system is minimum maintenance, safe, reliable and ultra-low power. They should have the capability of operating reliably over many years without their battery being replaced or capability of harnessing energy from body heat or by any other reliable and safe mechanism.

Smart Grid and Energy Control Systems

Sensors can be used for efficient generation, distribution and utilization of energy. For example, on the generation side sensor networks enable solar energy to be generated more efficiently. Standalone panels “do not always capture the sun’s power in the most efficient manner” [5]. Automated panels managed by sensors track sun rays to ensure that the sun’s power is gathered in a more efficient manner.

Environmental Applications

Sensor networks are very useful for environmental applications. They have been used to detect environmental hazards such as earthquakes, forest fires and floods. The advantage is their ability to gather data from remote areas. Since the sensor nodes have wireless communication capability for disseminating the data they collect, researchers can monitor remote terrain from the comfort of an office. Some have even deployed the motes to analyze remote locations, observing the motion of a tornado, or detecting fire in a forest.

Home Applications

Sensors are used in many house hold appliances which represent one of the largest markets of electronic products. Temperature sensors are used in air-conditioning system, washing machines and refrigerators. A pressure sensor can be used in a washing machine to measure the level of water in the drum and the soiling of water can be determined by a turbidity sensor. Sensors can also be used to reduce energy wastage by proper humidity, ventilation, air conditioning (HVAC) control.

2.2 Sensor Network Application for Wind Turbine

Wind Turbines are becoming common place throughout the world. They provide clean energy and are typically located in wind farms. The definition of a wind turbine is a

rotating machine that converts the kinetic energy of wind into mechanical energy. When this energy is converted to electricity, the machine is called a wind generator or a wind turbine. The Wind turbines are costly structures and exist in harsh environmental conditions. It becomes important to have a mechanism to do their condition based health monitoring to avoid unplanned downtime due to component failure. Various kinds of sensors can be deployed to monitor physical environment of a wind turbine. Inexpensive and flexible wireless sensors can be installed on a wind turbine to measure dynamic response and, using embedded computational abilities collocated with the sensor itself, engineering level monitoring algorithms could be run to detect a failure.

Figure 2.1 shows a £1 million wind turbine destroyed due to mechanical failure. A 65 ft. blade that flew off the turbine came loose after bolts attaching it to the hub failed. If a sensors for condition based health monitoring was deployed for this particular turbine, then the system could trigger an alarm for this failure beforehand.

Figure2.1. Wind turbine failure [21]

Following physical parameters of a wind turbine could be used for condition based health monitoring:

Table 2.1 physical parameters used for health monitoring

Physical Parameter	Reason for Sensing	Sensor Type
Temperature	Extreme variations to correspond with other data and corrections	Fiber optic or MEMS
Moisture/Humidity	Moisture could affect material properties. Could also be indirectly used to detect Ice.	Fiber optic or MEMS
Pressure	Pressure sensors are used to monitor yaw brake, lubrication oil, cooling circuit pressure, and level in gear boxes.	Fiber optic or MEMS
Ice Sensor	Reduction of Power due to disrupted Aerodynamics	Fiber optic
Wind Speed/ Direction	Adjust the alignment and pitch of the turbine blades relative to the wind conditions	Anemometer
Blade Tip Deflection	To Avoid Tower Strikes	Fiber Optic, Infrared(IR)
Blade Strain	Check for extreme strain along blade length.	Fiber Optic

2.3 Sensor Technology

Sensors and its varied technologies that were initially intended for a specific application are finding usage in numerous other interesting and growing market segments that were not thought of earlier. This penetration is happening at various levels and mainly it began with the growth and development of MEMS technology and Optical Sensor technology.

- **MEMS Technology**

Micro-Electro-Mechanical Systems, or MEMS, is a technology that permits integration of sensors, actuators, and computation and communication blocks into one batch-fabricated device. MEMS based sensors leverage established microelectronics fabrication and packaging technology to achieve: cost effective, high volume manufacturing, extremely small size and weight, improved performance and precision, and increased reliability. The critical physical dimensions of MEMS based device varies from a few microns to several millimeters [6].

MEMS-based sensors are a crucial component in automotive electronics, medical equipment, smart portable electronics such as cell phones, PDAs, and hard disk drives, computer peripherals, and wireless devices. MEMS technology has enabled the production of smaller and better sensors such as accelerometers and pressure sensors that were initially targeted at the automotive market. Applications included airbag firing and tire pressure monitoring to name a few. In addition, MEMS sensor technology has also been making inroads in to industrial and aerospace and defense, and medical markets. The rapidly growing MEMS market in year 2010 was about $7 billion.

- **Optical Sensor Technology [7]**

Optically based sensors, including fiber sensors and opto-electronics sensors, are an emerging sensor technology that exhibit: excellent sensitivity, low cost and weight, high manufacturability and package-ability, large dynamic ranges, and potential electromagnetic immunity. Optically based sensors have been demonstrated in many sensing applications, including: chemical, environmental, and physical sensing. White-

light polarization interferometry, also referred as WLPI, is a popular fiber optic sensor technology. Fiber optic sensors are made up of two main parts: the fiber optic transducer and the signal conditioner. The fiber optic transducer is made of a proof body which contains an optical device that is sensitive to the physical magnitude to be measured. The signal conditioner is used for injecting light into the optical fiber, receiving the modified light signal returned by the transducer as well as for processing the modified light signal and converting the results into the physical units of the measurand.

Optical interferometry is recognized as the most sensitive method for fiber optic sensing. Indeed, the interferometer is known as a very accurate optical measurement tool for measuring a physical quantity by means of the measurand-induced changes of the interferometer path length difference. However, when using a narrowband light source (such as a laser source), the coherence length of the source is generally greater than the path length difference of the interferometer and therefore the measurement suffers from a 2π phase ambiguity. This problem is avoided by using a light source with short coherence length that is a light source with a broadband spectrum. This type of interferometry is known as white light. The using of white light in fiber optic sensor technology is known as WLPI technology.

For all of these types of transducers, a change in the magnitude of the applied measurand result into a change of the path length of the transducer sensing interferometer. Therefore the path length difference can be thought as the output of

the transducer although we know that the physical or real output is the light signal that carries the information about. The relationship between the applied measurand M and the output of the transducers, referred to as the transducer signal output, can be represented by the following equation:

$$\delta_s = S \cdot M + \delta_o$$

S is the sensitivity of the transducer that is the ratio of change in transducer output to a change in the value of the measurand, and is the zero-measurand output.

Here we show how the temperature sensor works. Pressure sensor and strain sensor are similar to the temperature sensor. Figure 2.2 shows the schematic description of a fiber optic temperature sensor. The temperature transducer is based on the polarization interferometer made of a birefringent crystal. The temperature dependent birefringence of specially selected crystal is used for the transduction mechanism. A linear polarizer is placed at the input face of the birefringent crystal and its end face is coated with a dielectric mirror. The sensitivity of the fiber optic transducer depends mainly on the temperature coefficient of birefringence of the crystal used [7].

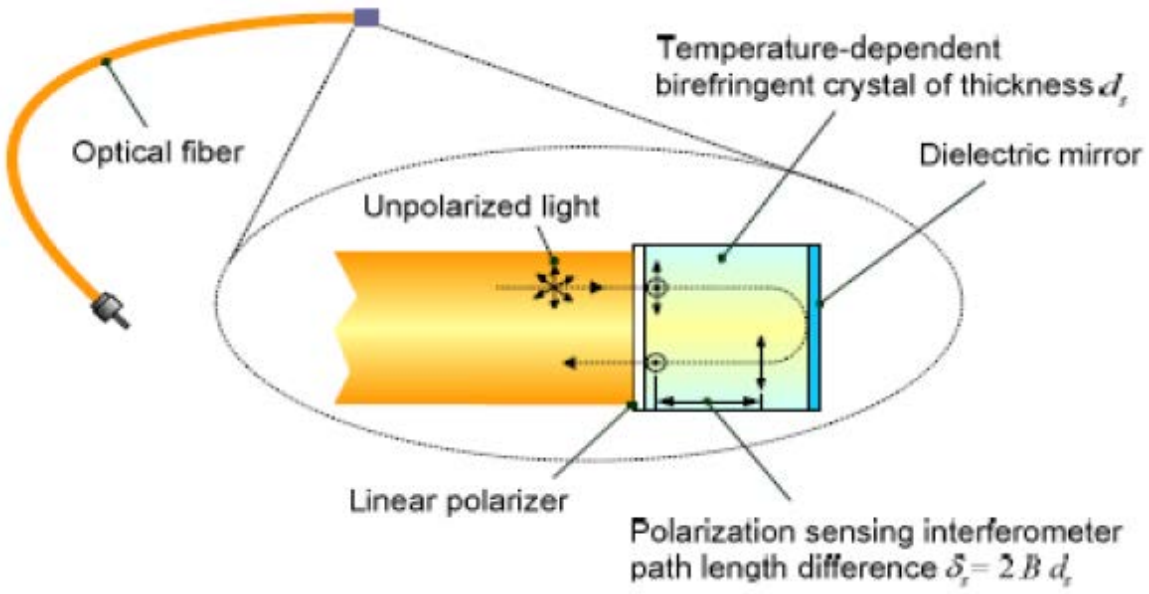


Figure 2.2 schematic description of a fiber optic temperature sensor [22]

This is an important feature because different crystals can be used for temperature sensing and this selection of crystals offers a range of sensitivity that varies by two orders of magnitude. This means that fiber optic temperature transducers can be designed with various operating temperature range, resolution and accuracy. Other advantages of this temperature transducer design are the small size of its polarization interferometer and the fact that has no moving part.

For wind turbine application, fiber optic sensors have many advantages. They are much cleaner than electrical strain gauges because one cable can have over one hundred individual sensors of varying types. It also can be attached to surface or embedded in laminates. And it is simple to install because there is only one wire to run and back. Only one wire that must be considered compared to 3 per gauge for foil gauges. Fiber optic sensors also have a long life. Experiment shows they have a 25-year

service life on wind turbine system. It also doesn't have signal degradation when transmitted over long distance. After installation, it doesn't need for recalibration. You don't need to concern of electrical interference from outside sources because passive sensors with doesn't required electrical power.

Due to the industrial problems they have some advantages. Sensors and interrogation units are expensive. Fiber optic sensor technology is relatively new that doesn't have the history of other systems. Furthermore, it is hard to find information of the products in the market because of limited number of manufacturers. Although initial cost is expensive, costs are reduced by several measures as lower cost of installation, reliability reduces long term costs. One interrogator can also handle hundreds of sensors.

2.4 Chosen Sensor Products for wind turbine system on blades

After serious and reliable research in the market, I chose several products of different kind of fiber optic sensors. All of these sensors have small sizes, low power consumption and reasonable price. These models have already been widely used in the industry and can be installed on the wind- turbine system. These sensors also meet the requirement to be installed on the blades.

- **Temperature Sensor**

Opsens OTP-A sensor uses the temperature-dependent birefringence of specially selected crystal as the temperature transduction mechanism [8].

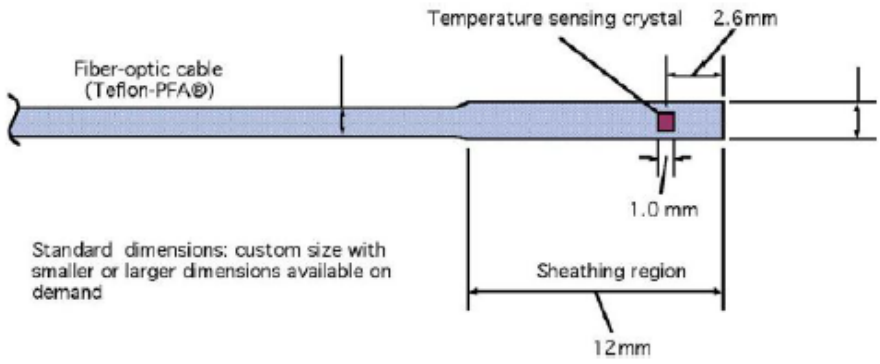


Figure 2.3 Schematic of model OTP-A [23]

Specifications:

Table 2.2 Specifications of model OTP-A

Temperature operating range	-40 °C to +250 °C
Resolution	0.1 °C
Accuracy	± 1.0 °C @ ± 3.3 sigma limit (99.9% confidence level)
Response time	1.5 s typical (depends on packaging and measuring conditions)
Operating humidity range	0-100 %
EMI/RFI susceptibility	Complete immunity
Calibration	NIST traceable
Cable length	1.5 meters standard (other lengths available)
Optical connector	SC standard
Cable sheathing	Teflon™ PFA
Signal conditioner compatibility	All Opsens WLPI signal conditioners

- **Pressure Sensor**

Opsens OPP-B model is a bare fiber optic pressure sensor (no metal housing) for applications requiring minimally invasive in-situ pressure measurement [9].

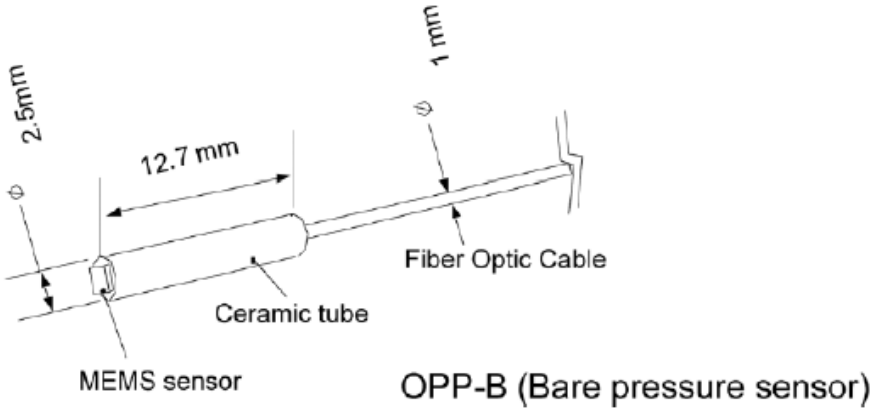


Figure 2.4 Schematic of model OPP-B [24]

Specifications:

Table 2.3 Specifications of model OPP-B

Pressure range	From 0-1 bar to 0-350 bar absolute (0-15 psia to 0-5000 psia)
Resolution	Range dependent (< 0.01% F.S. typical)
Precision	± 0.1% F.S.
Thermal coefficient of Zero	< 0.01% F.S / °C
Proof pressure	200% F.S.
Operating temperature	up to 100 °C
EM/RF/MR/MW susceptibility	Complete immunity
Cable length	1.5 meters
Optical connector	Customer specifications
Cable sheathing	Customer specifications
Signal conditioner compatibility	All Opsens WLPI signal conditioners

- **Strain Sensor**

OpSens fabrication process ensures an exact definition of the gauge factor, making OSP-A the most accurate fiber-optic strain gauge sensor in the industry. Combine with

Posen's WLPI signal conditioning technology and with the inherent advantages of fiber optics, the OSP-A deliver unprecedented repeatability and reliability in the most adverse conditions such as high levels of electromagnetic fields as well as high voltage and rapid temperature cycling conditions [10].

The OSP-A is designed with two optical fibers that are precisely aligned inside a microcapillary tube to form an optical Fabry-Pérot interferometer. This makes the OS-A strain gauge completely immune to any electromagnetic interference.

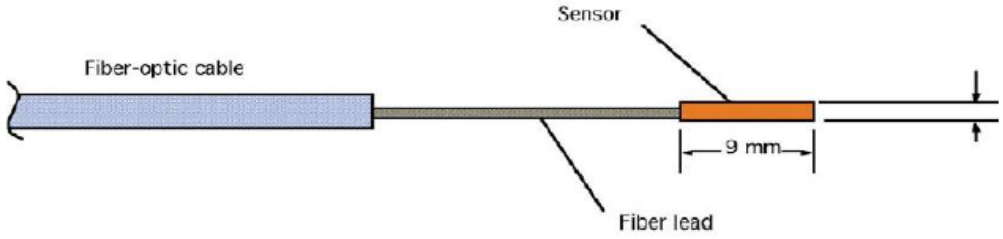


Figure 2.5 Schematic of model OSP-A

Specifications:

Table 2.4 Specifications of model OSP-A

Strain range	-1 000 to +1 000 με	-2 500 to +2 500 με	-5 000 to +5 000 με
Resolution	0.15 με	0.3 με	0.5 με
Gauge factor accuracy	± 3 %	± 5 %	± 10 %
Temperature sensitivity		Temperature insensitive	
Transverse strain sensitivity		Transverse strain insensitive	
Temperature operating range		-40 °C to +250 °C	
EMI/RFI susceptibility		Complete immunity	
Cable length		1.5 meters standard	
Optical connector		SC standard, ST available on request	
Cable sheathing		0.9 mm O.D. acrylate tight-buffer or 1.0 mm O.D. braided fiberglass, other available on request	
Signal conditioner compatibility		All Opsens WLPI signal conditioners	

- **Ice Detector Sensor**

Icing on Wind turbine blades causes variety of problems [11]. The buildup of ice on wind turbine blades disrupts the aerodynamics of the blades which results in loss of production and in extreme icing conditions the turbine cannot be operated safely. Just as airplanes, refrigerators, radio broadcast towers, vehicular overpasses, and bridges are all susceptible to ice formation, so are wind turbines that are sited in cold-weather locations. And not only do O&M professionals monitoring remote turbines need to be aware of the onset of ice formation, they also need to know how fast it's accumulating.

The most common type of ice detector on the market today involves a vibrating tuning fork sensor, whose design dates from the 1980s. It is essentially an electromechanical technology that operates as a vibrating rod. In the case of an airplane, the rod is mounted so as to be exposed to the passing airstream. If there's no ice on the vibrating rod, it resonates at its natural frequency. But if it has a coating of ice the additional weight slows down the vibrations, which changes the frequency. The frequency change is detected, then calibrated into ice weight, and ice thickness, and subsequently used to set the ice-alert signal after a predetermined thickness has accumulated on the probe, usually around 0.020 inch. Unfortunately, such complex assemblies have lots of precision internal parts, are costly to manufacture and put together, are not very sensitive, and require a high-speed ambient air stream in order to work properly. The interface electronic package, of course, has to be integral with the

vibrating assembly, which limits installation flexibility and also its suit-ability for use on wind power turbines.

Figure 2.6 Product of Ice Detector Sensor

2.5 Propose Design of Ice Detector Sensor

As we can see from the above, the fiber optic sensors have the small size and low power consumption. Compare to the fiber optic sensor, ice detector has a large size and consume more power. That ice detector shown above is the smallest product available in the market, but apparently this size is not suitable to be installed on the wind turbine blades. But since we already have the smaller fiber optic sensors, why not using these parameters to predict ice forming? In next few chapters, I will introduce the replacement method to detect ice.

Chapter 3

Grey Model Based Prediction Module

This chapter will introduce the predictive processor module. After the optic sensors get the source data, a predictive module is needed to predict the side parameters in the coming several minutes. It is because heating system usually need time to start and it will take several minutes to heat the blade to prevent ice forming, if we cannot predict the environment parameters before a period time, it will be too late and the ice will be already formed. In this chapter, I will introduce the basic concept of grey model and give two ways, C++ and Verilog, to implement this method.

3.1 Basic Concept of Grey Model

A black system can be viewed only through its inputs and outputs, without any knowledge of its internal working. We can understand as its implementation is “opaque” [12]. With these systems, we cannot predict the next value of their outputs; we can only know the output after we gave the input. The opposite of a black system is a white system. With a white system, the inner components and logic are available for

inspection. For these systems, because we know the structure of system, we can calculate the output with any given input.

With the definition of black system and white system, if we only know part information of the system, this system is called grey system [13]. In grey system theory, the random process is called grey process, which varies in a definite range and time-related. All sorts of random variables in a grey system are called grey variables. These variables must change in a certain limited way. So we can generate a sequence of data and develop a Grey Data Generation method to trim disorder original data. Environment changing (temperature, humidity, pressure) is a dynamic process. These variables are changing continuously and they are random. But these parameters changes follow some unobvious rules and affect each other, so we cannot give certain algorithm to calculate the next possible value. Fortunately, because they still follow some unobvious rules, so if we generate a data list, we are able to predict the next value. The latency, development and occurrence of the data list should be continuous, comparability and relativity. Based on these characteristics of the data list, we can forecast its occurrence with the analysis of past and actuality. Environments changing have characters of grey system, so it can be analyzed by grey system theory [14].

3.2 Grey Model-GM (1, 1)

GM (1, 1) is the first-order grey linear differential equation of Grey Model. It is suitable for forecasting single variable [15]. The basic processing model of GM (1, 1) is shown in Figure 3.1:

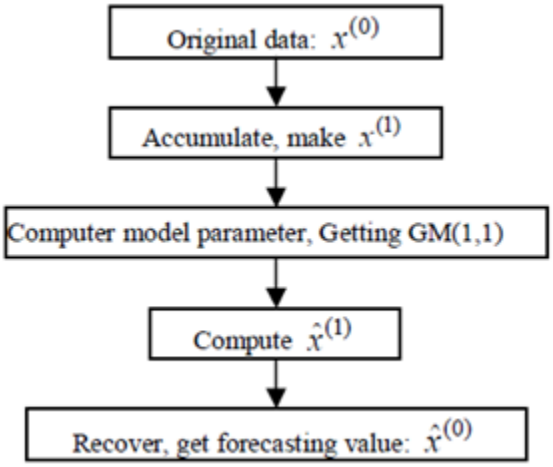


Figure 3.1 Basic Processing Model of GM (1, 1)

The original data sequence is a grey system, but if we accumulate these original data it will reduce the random of the data sequence, it will turn grey system into white system. We called this Accumulated Generating Operator (AGO). With an accumulating process, we can easily find out the development trend of the original data with the messy original data can be discovered [16].

Below is the original data sequence:

$$X^0 = (X^0(1), X^0(2), \dots \dots X^0(n))$$

With r times accumulating, it becomes:

$$x^r = (x^r(1), x^r(2), \cdots x^r(n))$$

Here:

$$x^r(k) = \sum_{i=1}^{k} x^{r-1}(i)$$

Accumulated r times, it is called r-AGO. The represented phrase is GM (r, 1). If we want to predict value in a linear system, we use 1-AGO. If we want to predict value in a parabola system, we will use 2-AGO. It is similar for other systems. In the environment measurement, the changing of temperature, humidity and pressure can be considered as a linear system if we sample them every thirty seconds, because these parameters are changing in a continuous and slow trend. Basically 1-AGO is the most popular algorithm and can meet most conditions.

Grey Model is shorted as GM. It is the most basic model in grey theory. The second number means the number of variables. If we have n parameters, then it will be represented as GM (1, n). In the environment system, I will separate these parameters and predict the future value individually, because this will make the algorithm easy and reduce the calculation of the algorithm which will reduce the size of chip in the future implementation [17].

In conclusion, GM (1, 1) will be applied as model in this project.

Below is the block chart of GM (1, 1) system:

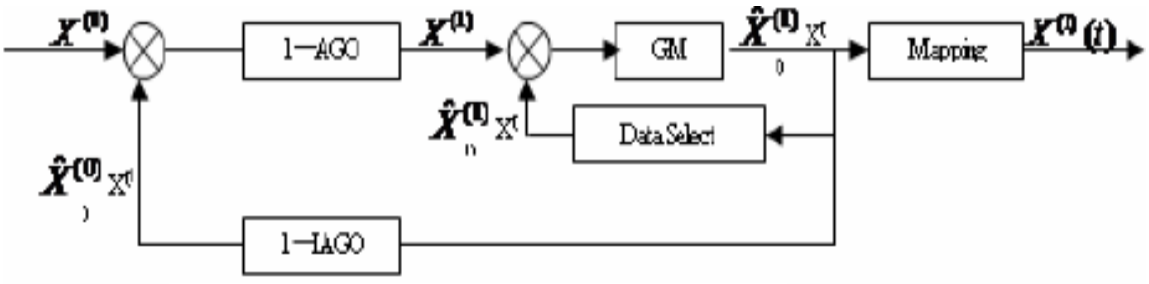


Figure 3.2 Block Chart of GM (1, 1)

I will simplify the 1-AGO as:

$$x^1=(x^1(1),\ldots\ldots x^1(n))$$

$$x^1(k)=\sum_{i=1}^{k} x^0(i)$$

I will suppose x^1 meets the following formula:

$$\frac{dx^1}{dt}+ax^1=u$$

This formula is the equation of GM (1, 1). Variable “a” is called developing coefficient and variable “u” is called “inside generator control grey data. Define $\hat{a}$ as:

$$\hat{a}=\begin{pmatrix}a & u\end{pmatrix}^T$$

Here $\hat{a}$ is the coefficient that needs to be calculated.

$$\frac{dx^1}{dt}=\lim_{\Delta t\to 0}\frac{x^1(t+\Delta t)-x^1(t)}{\Delta t}$$

Because Δt almost equals to zero, formula above can be written in a discrete form:

$$\frac{dx^1}{dt} = x^1(k+1) - x^1(k)$$

I already pointed out the environment system is a very slow and continuously changing process, so when Δt is small enough, we can assume there is not a saltation from $x^1(t)$ to $x^1(t+\Delta t)$. So we can use the average of them:

$$z(k+1) = \frac{1}{2}(x^1(k+1) + x^1(k))$$

Thus, the GM (1, 1) becomes:

$$x^1(k+1) - x^1(k) + az(k+1) = u$$

When k=1:

$$x^1(2) - x^1(1) = -az(2) + u$$

$$x^1(2) = x^0(1) + x^0(2) \quad x^1(1) = x^0(1)$$

$$x^0(2) = -az(2) + u$$

As the same:

$$x^0(m) = -az(m) + u \quad (\text{m=2,3,.....,n})$$

If we express it in a matrix form:

$$\begin{Bmatrix} x^0(2) \\ x^0(3) \\ \vdots \\ x^0(n) \end{Bmatrix} = \begin{Bmatrix} -z(2) & 1 \\ -z(3) & 1 \\ \vdots & \vdots \\ -z(n) & 1 \end{Bmatrix} \begin{Bmatrix} a \\ u \end{Bmatrix}$$

We can tag in:

$$X_N = B\hat{a}$$

When the rank of B $r_B = 2$, the matrix formula above has exclusive solution, we can use Least Square method to calculate $\hat{a}$:

$$\hat{a} = (B^T B)^{-1} B^T x_N$$

Thus we can figure out parameter “a” and “u” of GM (1, 1). Based on the formula that calculates the roots of ordinary differential equations, the particular solution of (3) is:

$$x^1(t) = \left(x^1(1) - \frac{u}{a}\right) e^{-at} + \frac{u}{a}$$

Suppose $x^1(1) = x^0(1)$:

$$x^1(t+1) = \left(x^0(1) - \frac{u}{a}\right) e^{-a(t+1)} + \frac{u}{a}$$

Then we can get the final result:

$$x^0(t+1) = (1-e^a)\left(x^0(1) - \frac{u}{a}\right)e^{-at}$$

3.3 Residual GM (1, 1)

Due to GM (1, 1) is neither the differential equation nor the difference equation, if |a| is small enough [18], that is:

$$1 - e^a \approx -a$$

Get to:

$$dx^1(t+1) \approx x^0(t+1)$$

In order to increase prediction precision, use residual error from $x^0(t)$ to model residual GM (1, 1) for modifying the forecast value.

Let residual error is:

$$e^0(t) = x^0(t+1) - \tilde{x}^0(t+1)$$

Then make a residual series set as:

$$e^0(t) = (e^0(2), e^0(3), \ldots \ldots e^0(t))$$

The sign of $e^0(t)$ are not consistent, and so we need to preprocess the sequence, choose a constant Q, that has:

$$\delta^0(t) = e^0(t) + Q$$

Get:

$$\delta^0(t) > 0$$

And class ratio:

$$\sigma(t) = \frac{e^0(t-1)}{e^0(t)} \in (e^{-\frac{2}{n'-2}}, e^{-\frac{2}{n'+2}})$$

Residual translation series is:

$$\delta^0(t) = (\delta^0(2), \delta^0(3), \dots \dots \delta^0(t))$$

Once again modeling GM (1, 1) to residual translation series, get the time response function of its prediction value, as:

$$\delta^0(t+1) = (1 - e^{a_\delta})\left(\delta^0(2) - \frac{u_\delta}{a_\delta}\right)e^{-a_\delta t}$$

And restore $\delta^0(t)$, then:

$$e^0(t+1) = (1 - e^{a_\delta})\left(\delta^0(2) - \frac{u_\delta}{a_\delta}\right)e^{-a_\delta t} - Q$$

We obtain:

$$x^0(t+1) = (1 - e^{a})\left(x^0(1) - \frac{u}{a}\right)e^{-at}, t < 2$$

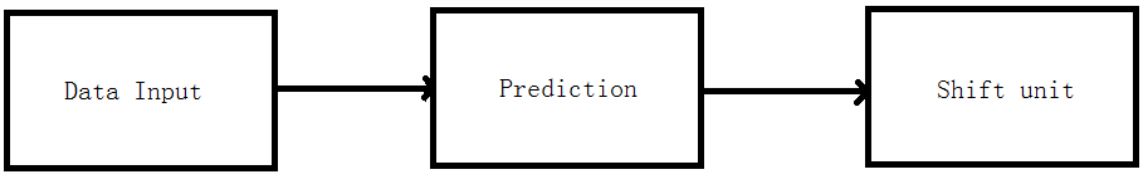
$$x^0(t+1) = (1-e^a)\left(x^0(1) - \frac{u}{a}\right)e^{-at}$$

$$+(1-e^{a_\delta})\left(\delta^0(2) - \frac{u_\delta}{a_\delta}\right)e^{-a_\delta t} - Q, t \geq 2$$

This is the result of residual GM (1, 1) calamites model [19].

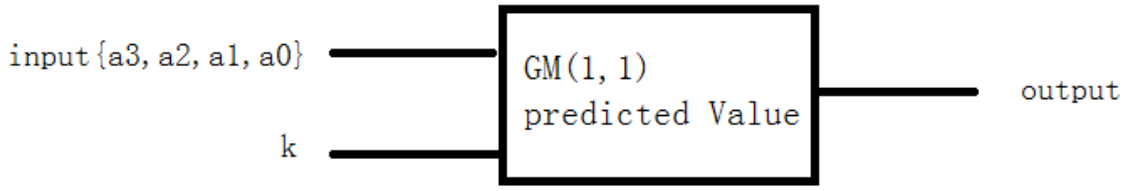
3.4 Modeling Residual GM (1, 1) with C++

The code can be divided into three parts. They are data input, prediction unit, and shift unit. Below is the flow chart:



The input will be stored in a text file, and the data input unit must be able to the catch the data stored in the text file. The data read by C code are stored in an array. Because the algorithm need the leading inputs as a source, so an array is created that can store five values. The sensor will sample the parameter every n seconds and the C code read those data and stored in the array. Choosing the first four incoming value as source, the last space is left for the fifth value and once the fifth value is coming in, the program will start another loop.

The prediction unit can also be separated by two parts. The first part gives the predicted value without error fixing.



First, four original data sequence are created as Xsum1-Xsum4. Then, average each other to get the matrix B for the calculation. With the preprocessing and given the matrix B and Y, the result of matrix Q is generated with the value “a” and “u”. These two parameters are the key of this algorithm, once these two values are calculated; the final output can be calculated.

The second part of the prediction unit is error checking and give the final residual GM (1, 1) value. Once the output is calculated, this output will be stored in another array, and waiting for the fifth practical value. After the fifth value comes in, a subtraction is made on the original data sequence and predicted value sequence to make an error sequence. Based on the theory before, GM (1, 1) unit will be applied again, predict the next error value and add back to the predicted value to get a fixed prediction value.

The shift unit is a simple function which gets rid of all the first value of original data, the first value of predicted value and the first value of error sequence. And shift all the three sequences to the left by one unit, which means the second value becomes the

first value of the sequence. With this action, the last space of array is empty and can wait for the next coming value to start a second loop.

The entire program won't stop only if receives a reset signal or the code will pause if it cannot receive more coming data. The full version of C++ code can be found in Appendix.

3.5 Sensor environment on examining the result of the C++ Realization

To examine the accuracy of C++ code result, I choose temperature as checking parameter. Because I just check the result of the prediction unit, so it is not necessary to get the whole fiber optic sensor system. For the convenience of the experiment, I choose a wireless temperature sensor: eZ430-RF2500 --- a sensor series of TI product as my experiment sensor system [20].

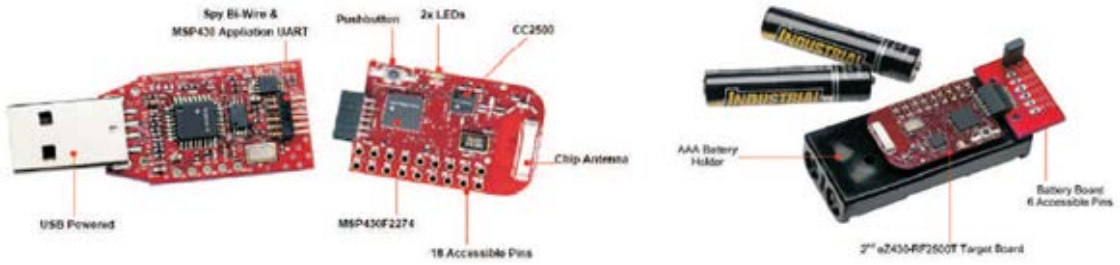


Figure 3.3 eZ430-RF2500 Development Kit Components

The eZ430-RF2500 is a complete USB-based MSP430 wireless development. The eZ430-RF2500T target board is an out-of-the-box wireless system that may be used with the USB debugging interface, as a stand-alone system with or without external sensors, or incorporated into an existing design. The new USB debugging interface enables eZ430-RF2500 to remotely send and receive data from a PC using the MSP430 application UART, referred to as the application backchannel.

The sensors and microcontroller communicate using a proprietary low-power radio-frequency (RF) network. The advantage is easy to implement with minimal micro-controller resource. One micro-controller can be connected with multiple remote sensors. This network protocol provides a long battery life, low data rate, and low duty cycle because it has a limited number of nodes communicating directly with each other. Despite the modest resources required, this protocol also supports End Devices in a peer-to-peer network topology, the option to use an Access Point to store and forward messages, and Range Extenders to extend the range of the network.

This sensor system can be used in a wide range of low-power applications including alarm and security (smoke detectors, glass breakage detectors, carbon monoxide sensors, and light sensors), automated meter reading (gas meters and water meters), home automation (appliances, garage door openers, and environmental devices), and active RFID.

There are several applications online to provide this sensor system become a humidity, pressure or vibration sensor system, the code are also available online. So this product is very suitable for this project experiment.

3.6 Experiment Result of C++

Temperature and humidity are measured as experiment parameter.

Table3.1 Temperature (F) September 10, 2011 (every 10 minutes)

Actual Value (F)	66.4	66.4	66.6	66.5	67.0
Predicted Value (F)					66.3

Actual Value (F)	67.1	67.3	67.8	68.2	68.8
Predicted Value (F)	67.4	67.3	67.7	68.3	68.8

Actual Value (F)	69.1	69.7	70.2	70.4	70.5
Predicted Value (F)	69.7	69.5	69.9	70.4	70.9

Actual Value (F)	70.5	70.8	70.6	70.2	70.1
Predicted Value (F)	70.4	70.6	70.7	70.8	69.4

Actual Value (F)	70.0	69.8	69.6	69.2	69.2
Predicted Value (F)	69.6	69.5	69.4	69.2	69.0

Actual Value (F)	69.1	68.8	68.6	68.2	68.2
Predicted Value (F)	69.0	68.7	68.5	68.3	68.1

Here we have some method to check the result of experiment.

Error means:

$$\overline{\tilde{\varepsilon}} = \frac{1}{n}\sum_{i'=1}^{n}\tilde{\varepsilon}^{(0)}(i')$$

Error variance:

$$S_1^2 = \frac{1}{n}\sum_{i'=1}^{n}\left(\tilde{\varepsilon}^{(0)}(i') - \overline{\tilde{\varepsilon}}\right)^2$$

Mean of raw data:

$$\overline{x} = \frac{1}{n}\sum_{i'=1}^{n}x^{(0)}(i')$$

Variance of raw data:

$$S_2^2 = \frac{1}{n}\sum_{i'=1}^{n}\left(x^{(0)}(i') - \overline{x}\right)^2$$

The posterior error ratio:

$$C = \frac{S_1}{S_2}$$

Error means	Error variance	Mean of RD	Variance of RD	Error Ratio
-0.0153	0.097	69.207	1.194	0.0812

Table 3.2 Humidity (%) September 11, 2011 (every 10 minutes)

Actual Value (%)	98	98	97	97	97
Predicted Value (%)					97

Actual Value (%)	96	96	94	92	92
Predicted Value (%)	97	96.4	95.3	92.3	92.1

Actual Value (%)	91	94	96	97	97
Predicted Value (%)	92	90.8	94.6	96.8	97.1

Actual Value (%)	96	95	95	95	94
Predicted Value (%)	96.4	95.6	95.2	95	95

Actual Value (%)	93	92	92	93	93
Predicted Value (%)	94.2	93.6	92	92	92.7

Actual Value (%)	94	94	95	96	96
Predicted Value (%)	93.5	93.8	94	94.2	95.1

Experiment Result:

Error means	Error variance	Mean of RD	Variance of RD	Error Ratio
0.135	1.044	94.5	2.788	0.373

3.7 Residual GM (1, 1) Realization Using Verilog

Dealing with GM (1, 1) using Verilog is kind of different from C++. The circuit cannot do the calculation very easily with complicated digits and decimal point. Consider the accuracy of the result is only one digit, so fixed point is used for the calculation.

All the input and output of my code is 32-bit. And the first digit is symbol digit. The next 21 digit is for integer part and the last 10 digit is for decimal part. The basic idea for making a fixed point calculation is to deal with the imaginary decimal point. For all four kinds of calculation (add, subtract, multiplication, division) it is needed to take the part of the result and combine them together to get the actual result.

- **Fixed point addition and subtraction**

For these two kinds of calculation, we do not need have many changes on the basic result. Because if there are two n-bit numbers added together, the result will stay n-bit digit. Say if they are 32-bit numbers, and the decimal point is at the last ten digits, the decimal point of the result will stay in the last ten digit so we are good on that.

- **Fixed point multiplication**

For the multiplication, if we have two 32-bits numbers, the result will become 64-bit digits, we must get a part of the lower part and a part of higher part and the symbol digit to make a new result.

Below is the multiplier module code. The full code of Verilog can be found in the appendix.

```
module mult(in1,in2,out);
input signed [31:0] in1;
input signed [31:0] in2;
input signed [31:0] out;
wire signed [63:0] temp;
wire signed [9:0] tempL;
wire signed [21:0] tempH;
wire sign;

assign temp = in1*in2;
assign tempL = temp[19:10];
assign sign = temp[63];
assign tempH = {sign,temp[40:20]};
assign out = {tempH,tempL};
endmodule
```

- **Fixed point divider**

For the division, we have the similar problem with the digit shifting. If we have two 32-bits numbers, after the division, the decimal point will be eliminated. So the solution for this problem is to shift the dividend before the calculation in order to increase the dividend, then the decimal point will be exist after the division.

Below is the Divider Module code. The full code of Verilog can be found in the appendix.

```
module div(in1,in2,out);
input signed [31:0] in1;
input signed [31:0] in2;
input signed [31:0] out;
wire signed [31:0] temp1;
wire signed [31:0] temp2;
wire signed [31:0] temp;
wire signed [31:0] divVal;

assign temp1 = (in1[31] == 1'b1) ? (-in1): in1;
assign temp2 = (in2[31] == 1'b1) ? (-in2): in2;
assign sign = (in1[31] != in2 [31]);
assign temp =  (temp1<<10)/temp2;
assign divVal = (sign ==1'b1)?(-temp):temp;
assign out = divVal;
endmodule
```

- **Fixed point exponent arithmetic**

For exponent arithmetic, it is not like C++. In C++, there exists the function “exp()”. In Verilog an exponent function must be defined and find a way to implement it. I choose Taylor Expansion to realize it.

Taylor Expansion for e^x: e^x=1+x+x^2/2!+x^3/3!+……

Using this expansion, I switch the complicated calculation back to the normal multiplication and division. Below is the exp module code. The whole code can be found in the appendix.

```
module exp(in, out);
input signed [31:0] in;
output signed [31:0] out;
wire signed [31:0] a2;
wire signed [31:0] a3;
wire signed [31:0] temp1;
wire signed [31:0] temp2;

mult mult20(in, in, a2);
div div4(a2, 32'd2, temp1);
mult mult21(a2, in, a3);
div div5(a3, 32'd6, temp2);
assign out= 32'd1 + temp1 + temp2;
endmodule
```

3.8 Simulation Result for Verilog----Accuracy Check

Even though the programming idea of Verilog is similar to C++, they still have difference. The biggest difficulty to translate from C++ to Verilog is the calculation problem. Even though I have development my own module for the calculation unit, the accuracy is not that good and the prediction result cannot reflect the environment very

well. In this section I will simulate a part of calculation unit and compare the result with C++ code.

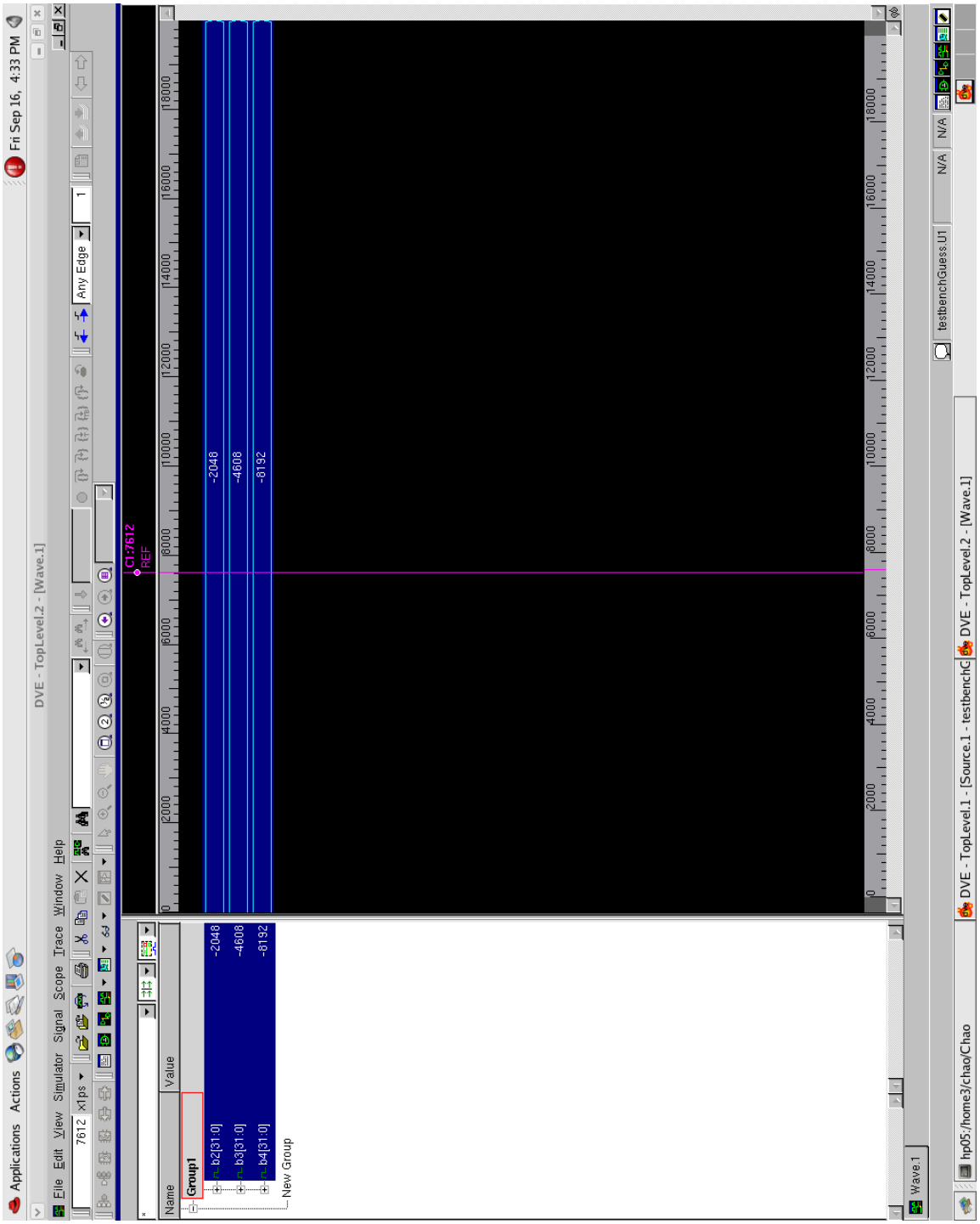


Figure 3.4 adder performance check using Simulation tool

Above is the simulation result of Verilog for b2, b3 and b4. If we compare the result with C++ code:

Name	Value
Xsum1	1.0000000000000000
Xsum2	3.0000000000000000
Xsum3	7.0000000000000000
b2	-2.0000000000000000
b3	-9.2559631349317831e+061

Name	Value
Xsum2	3.0000000000000000
Xsum3	7.0000000000000000
Xsum4	15.000000000000000
b3	-5.0000000000000000
b4	-9.2559631349317831e+061

Name	Value
Xsum3	7.0000000000000000
Xsum4	15.000000000000000
b2	-2.0000000000000000
b21	-9.2559631349317831e+061
b3	-5.0000000000000000
b4	-11.000000000000000

Figure 3.5 C++ Debug Result for adder

We can see if removing the decimal point of Verilog result, the answers of b2, b3 and b4 are exactly same because in the module of adder, we don't have any combination of integer part and decimal part. We keep the structure same.

But with the calculation going on, the result has error when occur the multiplication. First we can see the result of Verilog Code:

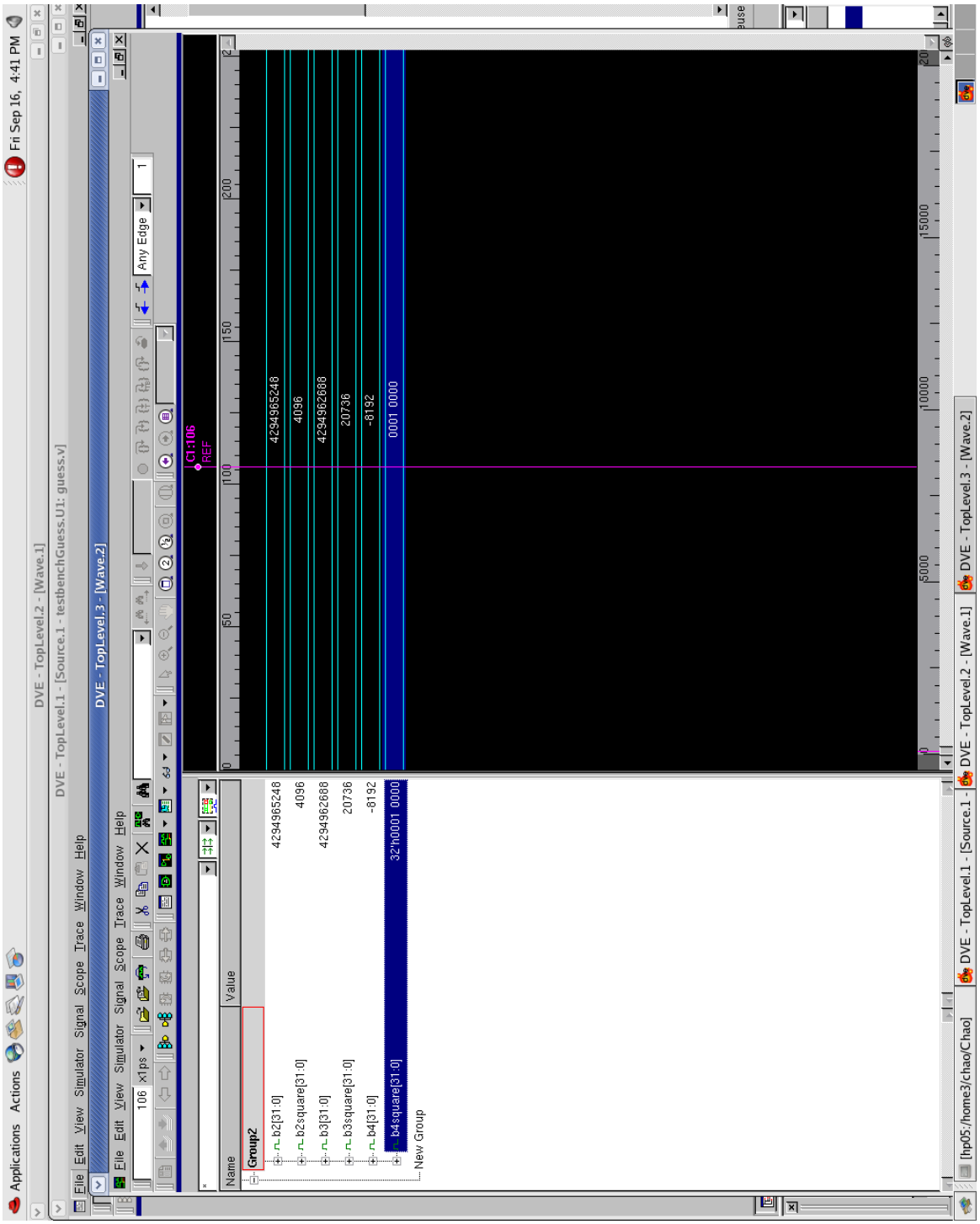


Figure 3.6 multiplier performance check using Simulation tool

Then we can see the result of C++ code:

Name	Value
b2	-2.0000000000000000
b2square	4.0000000000000000
b3	-5.0000000000000000
b3square	-9.2559631349317831e+061

Name	Value
b3	-5.0000000000000000
b3square	25.000000000000000
b4	-11.000000000000000
b4square	-9.2559631349317831e+061

Name	Value
b2	-2.0000000000000000
b21	-9.2559631349317831e+061
b3	-5.0000000000000000
b4	-11.000000000000000
b4square	121.00000000000000

Figure 3.7 C++ Debug result for multiplier

As shown in the picture, the value of b2square and b3 square is exactly as same as the value in the C code, but the value of b4 square in the Verilog code, if we remove the decimal point and switch to decimal, it is 117, which has some error compare with the c code, this is because when we take the integer part of the actual value of Verilog code, if the result in integer part is longer than the designed digit, this part will be omitted in the final result, so if the we omitted last 3 digit numbers, when the number is switched back to decimal, it will have an error with 0-8. This is an example for the reason of error. In the practical condition, the number of the digit is bigger and the range of the number is bigger too.

It is a similar situation with the division module. After the calculation, we still need to omit the rest part of the integer part and the decimal part. Below shows the division module of “a” and “u”:

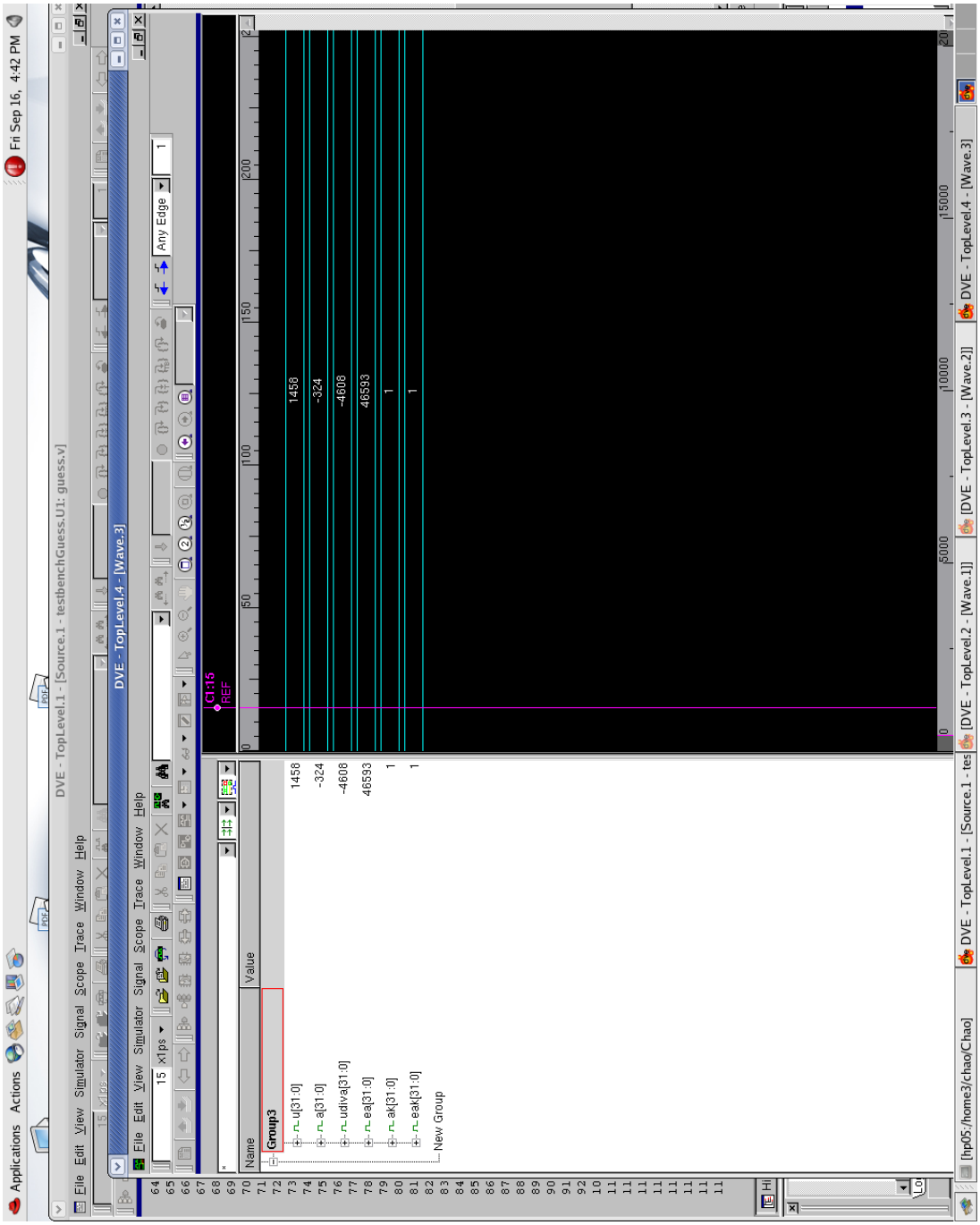


Figure 3.8 Divider performance check using Simulation tool

And the C++ simulation result:

Name	Value
B1	0.023809523809523808
B2	0.14285714285714285
B4	1.1904761904761905
a	-0.66666666666666641
b2	-2.0000000000000000
b3	-5.0000000000000000
b4	-11.000000000000000
u	0.00000000000000000

Name	Value
B2	0.14285714285714285
B4	1.1904761904761905
b2	-2.0000000000000000
b3	-5.0000000000000000
b4	-11.000000000000000
e	-9.2559631349317831e+061
u	0.66666666666666696

Figure 3.9 C++ Debug result for divider

3.9 Replacement for the Calculation Unit

Based on the experiment result in the previous section, in the real implementation of the prediction module, it is suggested to use professional calculation module to replace the existing calculation unit. There are a lot of DSP units available in the market. This method of calculation is only used on experiment and cannot provide an accurate result. So I will use C++ result as the input in the next chapter.

Chapter 4

Side Parameter Based Ice Analysis

4.1 Introduction of LEWICE

LEWICE is a software used for predicting ice shapes, collections efficiencies, and anti-icing heat requirements. LEWICE can be run on a regular desktop PC. The analysis will tank several minutes to finish. It allows user to run several parameter studies for design purposes. The form of ice can be assessed performance degradation both as an input to a CFD program or experimentally in flight or in a wind tunnel [21].

The computer code LEWICE has an analytical ice accretion model. It can evaluate the thermodynamics of the freezing process when super cooled droplets impinge on a body. The atmospheric parameters of temperature, pressure, and velocity, and the meteorological parameters of liquid water content (LWC), droplet diameter, and relative humidity are specified and used to determine the shape of the ice accretion.

The definition of surface of the clean is the segments joining a set of discrete body coordinates. The code consists of four major modules:

1) the flow field calculation
2) the particle trajectory and impingement calculation

3) the thermodynamic and ice growth calculation

4) the modification of the current geometry by addition of the ice growth

LEWICE applies a time-stepping procedure to "grow" the ice accretion. Initially, the flow field and droplet impingement characteristics are determined for the clean geometry. The ice growth rate on each segment defining the surface is then determined by applying the thermodynamic model. When a time increment is specified, this growth rate can be interpreted as an ice thickness and the body coordinates are adjusted to account for the accreted ice. This procedure is repeated, beginning with the calculation of the flow field about the iced geometry, then continued until the desired icing time has been reached.

4.2 Main Variables of the Input File

The format of the main input file is a series of variables listed in separate lines. The line should contain the name of the variables followed by an equal sign followed by the value to be assigned to that variable. The value will be truncated for use in the program.

If a variable is not listed in the input file, the program will use the default value.

ITIMFL (= 1)

ITIMFL is a flag indicating whether LEWICE will use automatic time stepping or will use a user-defined number of time steps. When ITIMFL =1, the minimum number of time steps is calculated like:

$$N = \frac{(LWC)(V)(Time)}{(chord)(\rho_{ice})(0.01)}$$

Where

LWC = liquid water content

V = velocity

Time = accretion time

Chord = airfoil chord

A second time step number is calculated by:

$$N2 = \frac{Time}{60}$$

IFLO is then calculated by the following expression:

$$IFLO = Max[Min(N, 30), Min(N2, 15)]$$

TSTART (= 0)

TSTART is the initial time of the icing simulation in seconds. At time = 0, LEWICE performs some estimates of transient behavior. The effect of these estimates is usually small.

IBOD (= 1)

IBOD is the number of bodies to be simulated.

IFLO (= 1)

IFLO is the number of time steps to be used in the simulation. The way to calculate is explained in the previous.

DSMN (= 4*0.0001)

DSMN is the minimum size of the control volumes (non-dimensionalized). It is also tied indirectly to the number of panels produced for the flow solution. The exact number of panels and controls volumes used will depend on the surface area and complexity of the input geometry.

RHOP (= 1000)

RHOP is the density of the water particle. This has been placed in the input file to broaden the utility of this software to industry. Except for very large particle sizes, the physics of water droplet trajectories is the same as for sand particle trajectories.

ICP (= 0)

ICP is a flag which allows surface pressure coefficients from another program to be used in place of the potential flow module. If ICP = 0, surface pressure coefficients are determined directly from the potential flow solution. If ICP = 1, the panel solution will not be used. Instead, a file will be read in from file “rflow.inp” which is supplied by the user. This file contains surface pressure coefficients referenced to specified wrap distance values. The wrap distance is the distance along the surface of the body geometry as measured from some initial reference location. All of the input files for LEWICE 3.0 will then bypass the potential flow module and use these values.

IQEX (= 0)

IQEX is a flag which allows external heat fluxes from another program to be used in place of the integral boundary layer in LEWICE 3.0. If IQEX = 0, convective heat transfer coefficients are determined directly from the integral boundary layer. If IQEX = 1, the integral boundary layer will not be used. Instead, a file will be read in from file “qextin.inp” which is supplied by the user. This file contains external heat fluxes referenced to specified wrap distance values. LEWICE 3.0 will then bypass the integral boundary layer routines and calculate convective heat transfer coefficients from the equation:

$$h = \frac{q}{(T_s - T_\infty)}$$

Where

q is the heat flux read in

Ts is the surface temperature

T_∞ is the ambient temperature

DPD (= 20)

DPS is the size, in microns, of the water drops.

CHORD (= 0.9144)

CHORD is the distance from the leading edge to the trailing edge in meters. For a cylinder, this represents the cylinder diameter. For airfoils, it is the standard chord length.

VINF (= 90)

VINF is the ambient velocity (the flight speed) in m/s.

LWC (= 0.54)

LWC is the liquid water content of the air.

TINF (= 268.15)

TINF is the ambient static temperature in degrees Kelvin.

PINF (= 100000)

PINF is the ambient static pressure in Pascals.

RH (= 100)

RH is the relative humidity and is input in percent relative humidity. This input value is normally assumed to be 100%, unless the actual value is known. Relative humidity is not recorded as part of the tunnel data, so the exact value during most tests is unknown. However, since relative humidity is at best a secondary effect on the ice accretion process, a value of 100% can be assumed. The value of relative humidity must be in the range 0% < RH < 100%.

GRAV (= 9.8)

GRAV is the acceleration due to gravity.

ISCOLC (= 1), JSCOLC (= 2), KSCLOC (= 2), SSLOPC (= 1), SZEROC (= 0)

The first five lines of this data input section contains variables used when reading pressure coefficient data. ISCOLC defines which column contains the wrap distance. JSCOLC defines the column which contains the pressure coefficients, and the KSCOLC defines the total number of columns in the file. SSLOPC defines the conversion factor which will be applied to the wrap distance values input. SZEROC defines the offset of the input wrap distances from those needed for LEWICE 3.0. If SZEROC = 0, the software will assume that a wrap distance value of 0 in the input data corresponds to the lower surface of the trailing edge.

ISCOLB (= 1), JSCOLB (= 2), KSCLOB (= 2), SSLOPB (= 1), SZEROB (= 0)

ISCOLB defines which column contains the wrap distance. JSCOLB defines the column which contains the collection efficiencies. KSCOLB defines the total number of columns in the file. SSLOPB defines the conversion factor which will be applied to the wrap distance values input. SZEROB defines the offset of the input wrap distances from those needed for LEWICE 3.0.

XBOOTUP (= 0.1)

XBOOTUP specifies the location of the upper boot limit for each body input.

XBOOTLOW (= 0.1)

WBOOTLOW specifies the location of the lower boot limit for each body input.

HRES (= 0.005)

HRES specifies the maximum residual ice height for each body input.

4.3 Body Geometry Input

Each line of the geometry input file contains an (x, y) coordinate pair for the body geometry. LEWICE expects the coordinates to be normalized by chord. The x-coordinate is listed first. The format of the data is free-format for the (x, y) coordinates.

If the body geometry is too coarse, the panel model created may not replicate the body geometry input. The suggested least point is 30. The only true upper limit to the

number of points which the user can input is 10000, which is the internal array size. Standard geometry input files used for testing purposes range from 50 to 150 points.

Problems may occur if input points are very close together or exactly the same. Body geometry points should be input in a clockwise fashion. This means that the points are input starting at the trailing edge and proceed sequentially toward the leading edge along the lower surface up to the leading edge, then traverse back to the trailing edge along the upper surface.

4.4 Main Output Files

dens.dat

This file contains predicted ice density at each location for each time step. Columns are: wrap distance from stagnation (s/c) and ice density (density) in kg/m3.

final1.dat

This file contains the final ice shape produced by LEWICE on the first body. If the program stopped due to an error, this file will not be output. Each line contains the dimensionless (x, y) coordinates of the final ice shape. This file format can also be used as input to the utility program THICK which calculates parameters of the ice shape for comparison with digitized ice shapes from experiments.

flow.dat

This file contains the output from the potential flow solution at each time step. Columns contain the panel index (i), dimensionless (x, y) coordinates (x/c, y/c) at the panel center (not at the endpoints as with other files), dimensionless wrap distance as measured from the lower surface trailing edge (s/c), dimensionless tangent velocity (vt), pressure coefficient (cp), a separate panel index for each body (j), the panel source/sink value (sigma), and the dimensionless normal velocity (vn). One flow solution is written to disk for each time step and an additional flow solution is generated on the final ice shape before the program exits.

htc.dat

This file contains the convective heat transfer coefficient at each time step. Columns are segment number (seg), dimensionless wrap distance from stagnation (s/c), heat transfer coefficient (htc) in W/m2/K, and Frössling number (fr). The Frössling number output is the local Nusselt number divided by the square root of the ambient Reynolds number. If the input flag HPRT is set to 1, the output from every 1/10th control volume will be generated. If this flag is set to 2, the output from every control volume will be generated.

pres.dat

This file contains the compressible flow solution at the edge of the boundary layer. Columns are segment number (seg), dimensionless wrap distance from stagnation (s/c), dimensionless velocity at the edge of the boundary layer (ve), dimensionless temperature at the edge of the boundary layer (te), dimensionless pressure at the edge

of the boundary layer (press) and dimensionless density at the edge of the boundary layer (ra). Reference variables which were used to nondimensionalize these quantities are chord length, ambient velocity, freestream total temperature, freestream total pressure and freestream total density, calculated from the equation:

$$\rho_o = \frac{P_o}{RT_o}$$

temp.dat

This file contains the surface temperature output from the energy balance. Columns are wrap distance from stagnation (s/c), surface temperature (t) in degrees Kelvin, and 'recovery' temperature (t_rec) in degrees Kelvin. If the input flag EPRT is set to 1, the output from every 1/10th control volume will be generated. If this flag is set to 2, the output from every control volume will be generated.

thick.dat

This file contains the ice thickness for each time step as measured from the clean surface. The ice thickness output in the "ice1.dat" file provides the ice thickness measured from the current ice shape. The "thick.dat" file was created to show the ice thickness from a common reference, i.e., the clean airfoil. This output file is similar to the ice thickness output file "clean.dat" created by the utility program THICK. However, output is sent to "thick.dat" only for every 1/10th control volume, so the output to this file will appear coarse compared to the output from program THICK as it outputs the ice thickness at every point. Columns are the x-coordinates of the clean surface (xsav) in

inches, the y-coordinates of the clean surface (ysav) in inches, the ice thickness as measured from the clean surface (ditot) in inches, the cumulative ice area (area) in inches and the wrap distance from the leading edge of the clean surface (s) in inches.

4.5 Grey Model Based Lewice Ice Analysis

Lewice has around thirty controllable parameters. Some of them are fixed when applied on wind turbine system. We can control other critical parameters related with ice forming and get the analysis result within a period.

Around all the controllable parameters, temperature is the most important and critical factor. So here I will use temperature as simulation parameter. Below is information of weather report in Syracuse:

Time	Temperature
19:00	-13.7^{o}C
19:30	-14.0^{o}C
20:00	-14.2^{o}C
20:30	-14.5^{o}C

Table 4.1 Part of Syracuse Weather Report Information

From the information above, if we put the temperature data sequence as input into the grey model based prediction module, we can get the predictive temperature two hours later (4 time units later). The prediction value is -15.1^{o}C.

Suppose we have other sensors already installed on the blades, then we can get the status of wind turbine blades:

Environment Parameter	Value
Tower Height	51m
Blade Velocity	15m/s
Temperature	-15.1^{o}C
Pressure	100000Pa
Humidity	100%
Gravity	9.8m/s^{2}

Table 4.2 Environment Parameter on Wind Turbine Blades

With the information above, we can build the main input file. According to the schematic, we can also build the geometry input file.

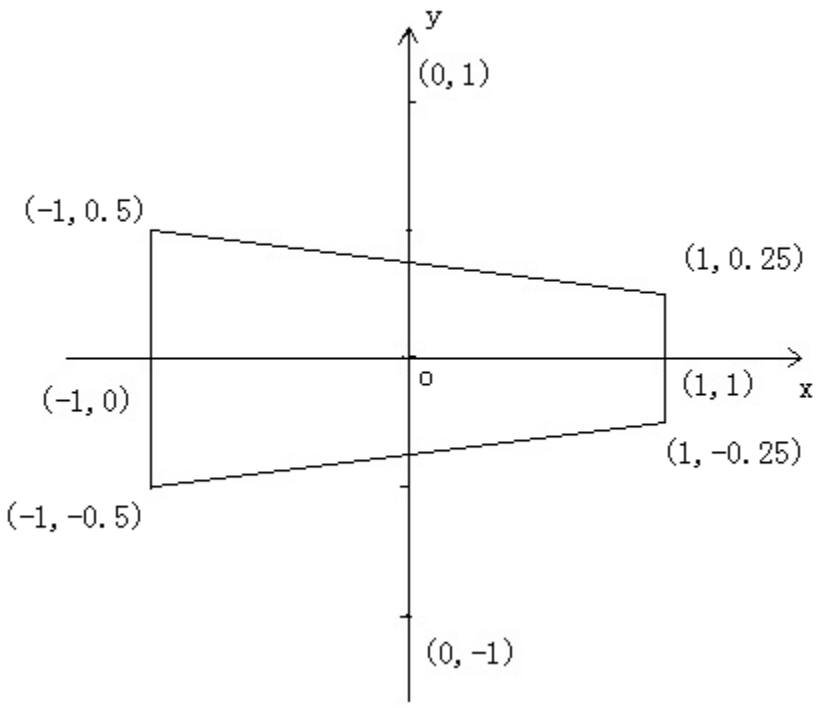


Figure 4.1 Geometry Shape of Wind Turbine Shape

With the main input file and geometry input file, we can get the final ice shape through (x, y) coordinate. Below is a part of final output file: final1.dat. Each line contains the (x, y) coordinates of the final ice shape:

1. 0.

0.977549986 -0.0020065134

0.954533264 -0.00369403767

0.931517255 -0.00539130895

0.908500908 -0.00708398859

0.88548469 -0.00877843826

0.862468429 -0.0104722897

0.839452181 -0.0121663296

0.816435929 -0.0138603115

0.793419679 -0.0155543165

0.770403427 -0.017248297

0.747387179 -0.0189423248

0.724370925 -0.0206362828

0.701354667 -0.0223301767

0.678338494 -0.024025237

0.655320762 -0.0256989513

0.63229972 -0.0273264647

We can easily draw the ice shape if we connect every point.

If temperature is too high to form ice, LEWICE will give the following warning:

"No ice will form at above freezing temperatures! Is this what you want? TINF = (value)."

If temperature is too low to form ice, LEWICE will give the following warning:

"It is unlikely that supercooled droplets exist below 240 Kelvin. TINF = (value). Make sure your input value is in degrees Kelvin. The accuracy of the software in this situation is unknown."

From the output file of thick.dat, we can also find the corresponding ice thickness of each (x, y) coordinate. Using this data, we can calculate the average thickness of areas where ice is forming. After several test, we can get the growth of ice. If the growth of ice is faster than the ability of de-ice system, then we should shut down the wind-turbine system. In this example, we can see the growth of thickness of area (0.7, -0.1):

Table 4.3 Growth of Ice Thickness

Time Step	Thickness (Inch*10^(-3)
1	0.115
2	0.286
3	0.297
4	0.302
5	0.478

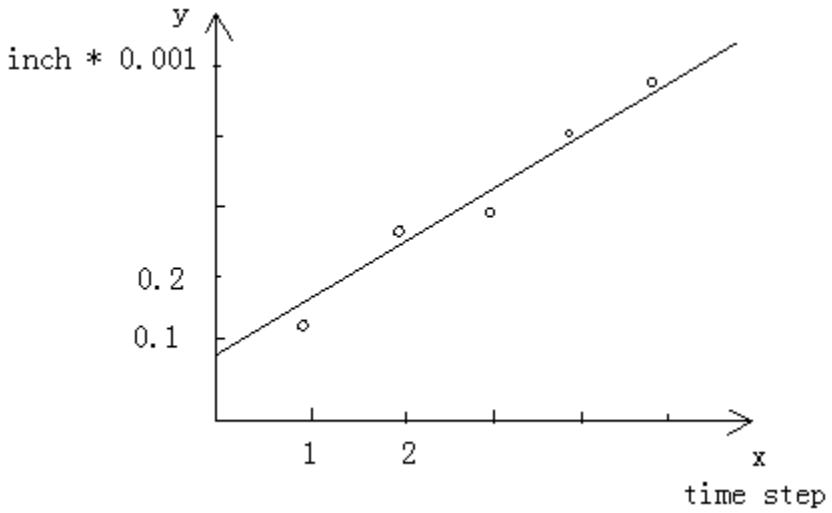


Figure4.4 Figure of Ice thickness Growth

We can see the K=0.9.

4.6 A Possible Alternative Ice Detector System

Fiber Optic Sensor has the ability to load hundreds sensor nodes with one cable, so one possible way to realize the sensor system is like below:

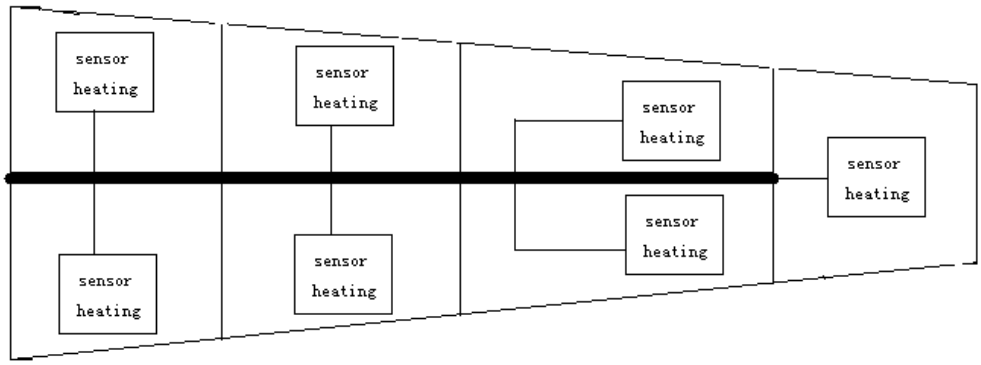


Figure 4.2 Blades Separation Model

The blades can be divided into seven to ten parts, each part contain sensor nodes and one heating resistor. LEWICE will give the (x, y) coordinate and correspond heating node can start work.

There are two reasons two divide blades in different areas. First, the blade has a relatively big area and division of areas can save the energy for heating process. Second, the velocity used by LEWICE is linear velocity. When blades are moving, different areas has the same routing velocity but different linear velocity. The different of linear velocity can be huge. Division of areas can increase the accuracy of measurement.

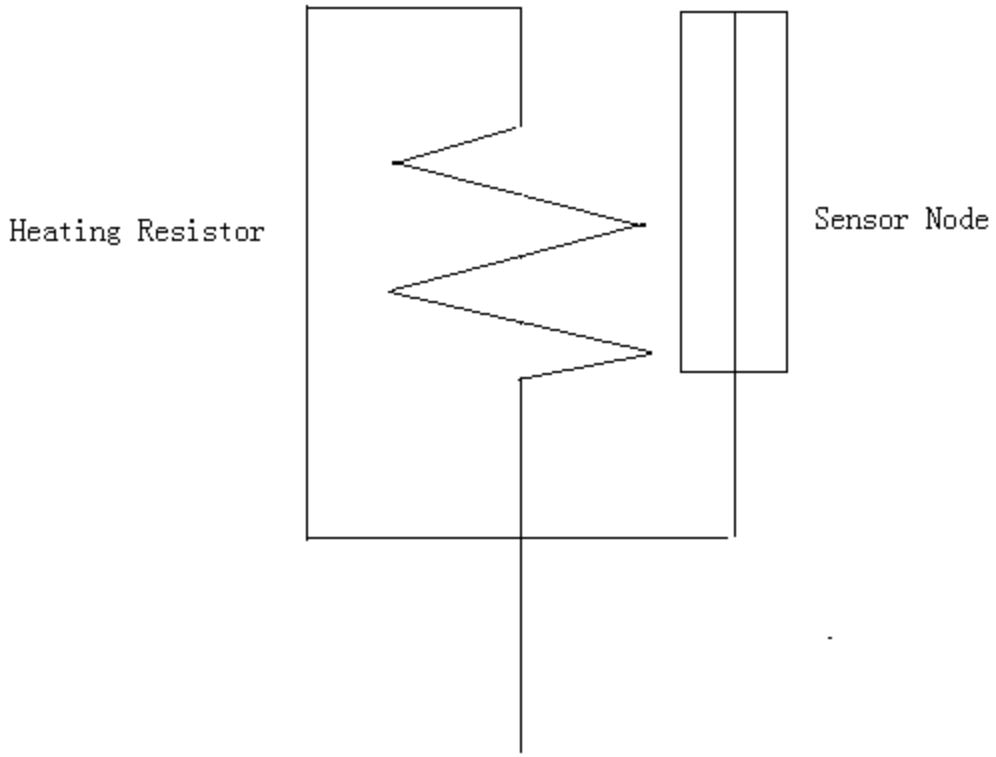


Figure 4.3 Sensor heating system circuit

Chapter 5

Conclusion

5.1 Summary

This thesis presents a possible way to replace ice detector on wind turbine system. This method solves the size and power consumption problems of ice detector in the market which needs to be applied on the blades.

Fiber optic sensor has developed a lot during the last ten years and they already have a lot of practical applications. Since six years ago, some companies started doing tests on sensors in order to make them reliable and robust in the extreme environments such as wind turbine system and aviation field. They already have products to measure temperature, humidity and pressure. But because the complexity of technology, we cannot find reliable ice sensor in the market. So using the existing technology of fiber optic sensor to replace ice detector becomes a wide way. In this thesis, several products are introduced and can be references when applied on the real project.

Using grey-model based prediction model, we can get the predicted value of critical parameters. As input of Lewice, we can get the ice prediction. Based on the task, we can change the critical parameters to meet the requirement.

5.2 Future Work

This method was only simulated by single parameter changed. We can get more accurate result by multiple parameters monitored.

In this thesis, critical modules are introduced. But I didn't put focus on the connections between each module. We can use other method to apply the connection between gray-model base prediction module and Lewice module. This will smooth the whole process.

Current version of Lewice does not provide real-time parameter monitored. This means for each time when parameter changed, the whole program will be run from beginning. During this design requirement, we don't have the timing requirement for Lewice response, so this is not a critical problem. But we can still improve the software to save computer resources.

Appendix A

C code for Grey Model

```
#include<iostream>

#include<math.h>

#include<stdio.h>

#include <stdlib.h>

#include <cmath>

#include <fstream>

using namespace std;


double k_plus_one_evaluated_value(double,double,double,double,int);

                                                    //gives X(k+1) evaluated value;


int main(int argc, char *argv)

{

   ifstream fin("input.txt");

   if(!fin)
```

```
{
 cerr << "Failed to open file!" << endl;
system("pause");
 return 0;
}                //read input file
/*****************Initial Process Begin ************************/
double X[5];
double X_Evaluated[5];
double X_Evaulated_four_time_units_later[10];
for (int i=0;i<10;i++){ X_Evaulated_four_time_units_later[i]=0;}
  double e[4];  for (int i=0;i<4;i++){ e[i]=0;}
int k=0;
in >> k;
cout<<"You need the prediction after "<<k<<" time unit(s)."<<endl;
 fin >> X[0] >> X[1] >> X[2] >> X[3];
   cout<<"The initial value is: "<<X[0]<<"  "<<X[1]<<"  "<<X[2]<<"  "<<X[3]<<"  "<<endl;
for(int i=0;i<=3;i++)
(
X_Evaluated[i]=k_plus_one_evaluated_value(X[0],X[1],X[2],X[3],i+1);
 }
```

```
/****************** Initial Process End ************************/


while(!fin.eof())
{
 double X_Evaulated_four_time_units_later_Modified=0;
 double e_four_time_units_later=0;
 double Q=0;
 fin >> X[4];


 if (X[4]==X[3] && X[3]==X[2])
{
  X_Evaulated_four_time_units_later[9]=X[4];
 X_Evaluated[4]=k_plus_one_evaluated_value(X[1],X[2],X[3],X[4],4);
e[4]=k_plus_one_evaluated_value(e[0],e[1],e[2],e[3],4);
 }
 else{
   double flag=0;
  for(int i=0;i<=3;i++)
{
  e[i]=X[i+1]-X_Evaluated[i];
```

```
 if(e[i]<flag) { flag=e[i];}

}

  if(flag<0){ Q=fabs(flag)+1; }


 for(int i=0;i<=3;i++)

{

 e[i]=X[i+1]-X_Evaluated[i]+Q;

 }

X_Evaluated[4]=k_plus_one_evaluated_value(X[1],X[2],X[3],X[4],4);

X_Evaulated_four_time_units_later[9]=k_plus_one_evaluated_value(X[1],X[2],X[3],X[4],
k+3);

e_four_time_units_later=k_plus_one_evaluated_value(e[0],e[1],e[2],e[3],k+2);

X_Evaulated_four_time_units_later_Modified=X_Evaulated_four_time_units_later[9]+
e_four_time_units_later - Q;

X_Evaulated_four_time_units_later[9]=X_Evaulated_four_time_units_later_Modified;

 }


for(int i=0;i<=4;i++)

e[i]=e[i]-Q;

 }
```

```
 for(int i=0;i<=3;i++)

{

X[i]=X[i+1];

X_Evaluated[i]=X_Evaluated[i+1];

  }                                  // Move the sequence to the next one



 for(int i=0;i<9;i++)

{

X_Evaulated_four_time_units_later[i]=X_Evaulated_four_time_units_later[i+1];

}

 cout<<"input:   "<<X[4]<<"            output:   "<<X_Evaulated_four_time_units_later[9]<<"
error: "<<fabs((X[4]-X_Evaulated_four_time_units_later[8-k])/X[4])*100<<"%"<<endl;

 }

fin.close();

cout<<endl;

system("pause");

return 0;

}
```

```
double k_plus_one_evaluated_value(double X1=0,double X2=0,double X3=0,double X4=0,int k=4){


if( X2==X3 && X3==X4 )
 {
return X4;
 }else{


double Xsum1=X1;
double Xsum2=X1+X2;
double Xsum3=X1+X2+X3;
double Xsum4=X1+X2+X3+X4;                    //First Order Sum Sequence


double a=0, u=0;                             //Parameters


double b2=-0.5*(Xsum1+Xsum2);                //  | X2 |      | b2    1 |
double b3=-0.5*(Xsum2+Xsum3);                    //Y= | X3 |  ,  B= | b3     1 |  , Y=BQ
//background value
double b4=-0.5*(Xsum3+Xsum4);                //  | X4 |      | b4    1 |
```

```
double b21=b2*b2+b3*b3+b4*b4;                    // Bt*B= | b21 b22 |
double b22=b2+b3+b4;                             //      | b22  3  |
double M=b21*3-b22*b22;


double B1=3/M;
double B2=-b22/M;                                // (Bt*B)-1= | B1 B2 |
double B4=b21/M;                                 //          | B2 B4 |


a=(B1*b2+B2)*X2+(B1*b3+B2)*X3+(B1*b4+B2)*X4;        //   Q=(Bt*B)-1 * Bt * Y
u=(B2*b2+B4)*X2+(B2*b3+B4)*X3+(B2*b4+B4)*X4;


double Xkplus1=0;        // X(k+1)
Xkplus1 = (X1-u/a) * (1-exp(a)) * exp(-a*k);
return Xkplus1;
}
}
```

Appendix B

Verilog Code for Grey Model

```
`define NUM 2'd3


module grey_model(X, N, ready, clk, reset, Xout);

input [31:0] X;

input [2:0] N;

input clk, ready, reset;

output [31:0] Xout;


reg signed [31:0] Xout1;

reg signed [31:0] Xreg [0:4];

reg signed [3:0] Nstore;

reg signed [3:0] Npre;

reg signed [31:0] e0, e1, e2, e3;

reg signed [31:0] e0_abs, e1_abs,  e2_abs,  e3_abs;

wire signed [31:0] cmp [0:2];

wire[31:0] E0, E1, E2, E3, Epre;
```

```
wire[31:0] epre;
wire equ;

always@(posedge clk)
        begin
                if(reset)
                        begin
                                Xreg[0]<= 32'd0;
                                Xreg[1]<= 32'd0;
                                Xreg[2]<= 32'd0;
                                Xreg[3]<= 32'd0;
                                Xreg[4]<= 32'd0;
                                Nstore<={1'b0,N}+4'd3;
                                Npre<={1'b0,N}+4'd2;
                        end
                else if(ready)
                        begin
                                Xreg[0]<= Xreg[1];
                                Xreg[1]<= Xreg[2];
                                Xreg[2]<= Xreg[3];
```

```
                    Xreg[3]<= Xreg[4];
                    Xreg[4]<= X;
                end
        end

assign equ = (Xreg[2]==Xreg[3]) && (Xreg[3]==Xreg[4]);
assign Xout= equ ? Xreg[4] : Xout1;

always@(Xreg[1], Xreg[2], Xreg[3], Xreg[4], E0, E1, E2, E3, e0, e1, e2, e3, Epre, epre)
        begin
                e0<= Xreg[1]-E0;
                e1<= Xreg[2]-E1;
                e2<= Xreg[3]-E2;
                e3<= Xreg[4]-E3;
                //if(equ) Xout<=Xreg[4];
                if(~e0[31] && ~e1[31] && ~e2[31] && ~e3[31])
                        Xout1<= Epre+epre;
                else begin
                        e0 <= e0 - cmp[2] +1'b1;
                        e1 <= e1 - cmp[2] +1'b1;
```

```
                    e2 <= e2 - cmp[2] +1'b1;
                    e3 <= e3 - cmp[2] +1'b1;
                    Xout1<= Epre+epre+cmp[2]-1'b1;
                end
        end
assign cmp[0] = (e0>e1)? e1 : e0;
assign cmp[1] = (e2>e3)? e3 : e2;
assign cmp[2] = (cmp[0]>cmp[1])? cmp[1] : cmp[0];

F F0(Xreg[0], Xreg[1], Xreg[2], Xreg[3], 4'h1, E0);
F F1(Xreg[0], Xreg[1], Xreg[2], Xreg[3], 4'h2, E1);
F F2(Xreg[0], Xreg[1], Xreg[2], Xreg[3], 4'h3, E2);
F F3(Xreg[0], Xreg[1], Xreg[2], Xreg[3], 4'h4, E3);
F F4(Xreg[1], Xreg[2], Xreg[3], Xreg[4], Nstore, Epre);
F F5(e0, e1, e2, e3, Npre, epre);
endmodule

module F(X1, X2, X3, X4, K, Out);
input signed [31:0] X1, X2, X3, X4;
```

```
input signed [3:0] K;

output signed [31:0] Out;


wire signed [31:0] Xsum1, Xsum2, Xsum3, Xsum4;

wire signed [31:0] a, u, b2, b3, b4, b21, b22, M, B1, B2, B4;

wire signed [31:0] sum12, sum23, sum34;

wire signed [31:0] b2square, b3square, b4square, b21t3, b22square, B1tb2, B1tb3, B1tb4, B2tb2, B2tb3, B2tb4;

wire signed [31:0] prod1, prod2, prod3, prod4, prod5, prod6, prod7, prod8, prod9, prod10;

wire signed [31:0] add1, add2, add3, add4, add5, add6;

wire signed [31:0] udiva;

wire signed [31:0] ea, eak, ak;

wire signed [31:0] sub1, sub2, out1;


assign Xsum1 = X1;

assign Xsum2 = X1+X2;

assign Xsum3 = X1+X2+X3;

assign Xsum4 = X1+X2+X3+X4;

assign sum12 = Xsum1+Xsum2;

assign sum23 = Xsum2+Xsum3;
```

```
assign sum34 = Xsum3+Xsum4;


assign b2=-(sum12>>1);

assign b3=-(sum23>>1);

assign b4=-(sum34>>1);


mult nult0(b2, b2, b2square);

mult nult1(b3, b3, b3square);

mult nult2(b4, b4, b4square);


assign b21 = b2square + b3square + b4square;

assign b22 = b2 + b3 + b4;


mult mult3(b21, 32'd3, b21t3);

mult mult4(b22, b22, b22square);


assign M = b21t3 - b22square;


div div0(32'd3, M, B1);

div div1(-b22, M, B2);
```

```
div div2(b21, M, B4);


mult mult5(B1, b2, B1tb2);

mult mult6(B1, b3, B1tb3);

mult mult7(B1, b4, B1tb4);

mult mult8(B2, b2, B2tb2);

mult mult9(B2, b3, B2tb3);

mult mult10(B2, b4, B2tb4);


assign add1 = B1tb2 + B2;

assign add2 = B1tb3 + B2;

assign add3 = B1tb4 + B2;

assign add4 = B2tb2 + B4;

assign add5 = B2tb3 + B4;

assign add6 = B2tb4 + B4;


mult mult11(add1, X2, prod1);

mult mult12(add2, X3, prod2);

mult mult13(add3, X4, prod3);

mult mult14(add4, X2, prod4);
```

```
mult mult15(add5, X3, prod5);

mult mult16(add6, X4, prod6);


assign a = prod1 + prod2 + prod3;

assign u = prod4 + prod5 + prod6;


mult mult17(-a, {28'b0,K}, ak);


div div3(u, a, udiva);

exp exp0(a, ea);

exp exp1(ak, eak);


assign sub1 = -ea + 1'b1;

assign sub2 = X1 - udiva;


mult mult18(sub1, sub2, out1);

mult mult19(out1, eak, Out);


endmodule
```

```
module mult(in1, in2, prod);

input signed [31:0] in1, in2;

output signed [31:0] prod;

wire signed [63:0] temp, temp_shift;


assign temp=in1*in2;

assign temp_shift= (temp[63])? ~((~temp)>>`NUM) : (temp>>`NUM);

assign prod= temp_shift[31:0];


endmodule


module div(in1, in2, quot);

input signed [31:0] in1, in2;

output [31:0] quot;


assign quot = in1 + in2;

endmodule


module exp(in, out);

input signed [31:0] in;
```

```
output signed [31:0] out;


assign out = in + 1'b1;

endmodule
```

Bibliography

[1] AerotechTelub, 2005. IceMonitor — The Ice load Surveillance System. http:// products.saab.se/PDBWeb/PDF/productpage_id1239_lan1.pdf, June 14, 2005.

[2] Battisti, L., 2004. Anti-icing system for wind turbines. World Intellectual Property Organization, International publication number WO 2004/036038 A1. April.

[3] Ice sensors for wind turbines Matthew C. Homola, Per J. Nicklasson, Narvik University College, Box 385, 8505 Narvik, Norway Received 30 December 2005; accepted 22 June 2006.

[4] Sensor Network Applications in Structures – A Survey - Sanath Alahakoon

[5] [21] (Atkinson, Castro, 2008)

[6] www.memsnet.org/mems/what is .html

[7] [22] http://www.opsens.com/pdf/SCBG.pdf

[8] [23] http://www.opsens.com/en/industries/products/temperature/otp-a/

[9] [24] http://www.opsens.com/en/industries/products/pressure/opp-b/

[10] http://www.opsens.com/en/industries/products/strain/osp-a/

[11] Optical Ice Sensors for Wind Turbine Nacelles. New optical technology borrowed from the communications industry promises lower cost, more practical ice detectors than earlier techniques. By Richard L. Hackmeister

[12] D. A. Wilhite, “Drought planning: A process for state government,” Water Resource Bulletin, vol. 127, no. 1, pp. 29-38, 1991.

[13] J. Q. Chen, H. Wan, X. L. Yan, Water Resources. Beijing: Science Press, 2003.

[14] R. Engelman, P. L. Roy, Sustaining Water: Population and Future of Renewable Water Supplies. Population and Environment Program, Population Action Internation, Washington DC. 1993.

[15] Y. M. Lopoulos, “Urban water management in Greece present conditions and perspective of sustainability,” IWRA International, vol. 28, no. 1, pp. 43-51, 2003.

[16] S. F. Liu, Y. G. Dan, Z. G. Fan, Grey System Theory and Application (3rd). Beijing: Science Press, 2005.

[17] J. L. Deng, Grey Prediction and Grey Decision. Wuhan: Huazhong University of Sci. & Tech. Press, 2002.

[18] J. L. Deng, the Primary of Method of Grey System Theory. Wuhan: Huazhong University of Sci. & Tech. Press, 2005.

[19] R. H. Huang, C. Y. Li, S. W. Wang, Climatic Disasters. Beijing: China Meteorological Press, 2003.

[20] http://icebox-esn.grc.nasa.gov/design/lewice.htmlireless