This item was submitted to [Loughborough's Research Repository](#) by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

One-bit processing for real-time control

PLEASE CITE THE PUBLISHED VERSION

PUBLISHER

© Xiaofeng Wu

PUBLISHER STATEMENT

This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: <https://creativecommons.org/licenses/by-nc-nd/4.0/>

LICENCE

CC BY-NC-ND 4.0

REPOSITORY RECORD

Wu, Xiaofeng. 2019. "One-bit Processing for Real-time Control". figshare. <https://hdl.handle.net/2134/34155>.

University Library

Author/Filing Title Wu, XIAOFENG

Class Mark T

Please note that fines are charged on ALL
overdue items.

FOR REFERENCE ONLY

0403191378

ONE-BIT PROCESSING FOR REAL-TIME CONTROL

By

Xiaofeng Wu

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

For the award of

Doctor of Philosophy

of

Loughborough University

October 24, 2005

©by Xiaofeng Wu

Loughborough University Pilkington Library
Date	JAN 2006
Class	T
Acc No.	0403191378

Abstract

In conventional digital control an analogue signal is converted into multi-bit digital format with an analogue-to-digital (A/D) converter. A control law is implemented into some digital hardware architecture, resulting in a digital control signal after processing. This digital signal is reverted to analogue format by a digital-to-analogue (D/A) converter or converted to a series of high-frequency pulses by a pulse-width-modulation (PWM) logic, hence being able to drive a physical system. The A/D and D/A converters can be any precision according to the system requirement, e.g. 12-bit in many cases.

This thesis, however, proposes one-bit processing for real-time control, which is a new concept in digital control. In one-bit processing, a control law is implemented with 1-bit signals at both the input and output. $\Delta\Sigma$ modulation is used to shape either an analogue or a multi-bit digital signal into a 1-bit signal. The 1-bit signal after processing can be directly applied to a physical system, i.e. pulse-density-modulation (PDM) which works very similarly to PWM control.

One-bit processing shows great advantages over the conventional digital control especially in implementation: First, the A/D converter is replaced with a simple $\Delta\Sigma$ modulator; Second, the control law is rewritten in a special controller structure, removing multipliers which are a major factor in digital integrated circuit (IC) design; Third, the D/A converter or PWM logic is no longer needed although some simple analogue filter may be utilised sometimes.

One-bit processing is developed for some particular requirements in control system processing in terms of area, speed and power consumption, making it necessary to build a new hardware for realising one-bit processing

efficiently. In this thesis, a new control system processor ($\Delta\Sigma$ -CSP) is described. A simple conditional-negate-and-add (CNA) unit is proposed for most operations of a control law. For this reason, the targeted processor is small and very fast, making it ideal for real-time control applications.

Keywords: 1-bit Processing, Control System Processor, FPGA, Quantisation-to-noise, System-on-chip, VLSI.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Roger Goodall, for his support and guidance during my research.

I sincerely thank all members of the Electronic Systems and Control Division for their kindly help and advice when I carried out this research. These people include Dr. John Pearson, Dr. Scott Halsey, Vassilios Chouliaras, Dario Sancho-Pradel.

My thanks also go to my family for their support and understanding. I specially thank my wife, Fen, for her love, encouragement and support throughout these years.

Finally, I would also like to acknowledge the EPSRC for supporting this research (Grant No. GR/R38002/01).

Contents

Abstract	I
Acknowledgements	III
1 Introduction	1
1.1 Background	1
1.2 Previous research	3
1.3 Research motivation	5
1.4 Dissertation overview	6
1.5 Summary	7
2 Literature Review	8
2.1 Control system design	8
2.2 Digital control basics	9
2.2.1 Design and analysis techniques	9
2.2.2 Sampling in digital control	10
2.3 Data conversion	11
2.4 Numerical issues	13
2.5 Digital devices	15
2.5.1 General purpose processor	15
2.5.2 Microcontroller	16
2.5.3 Digital signal processor	17
2.5.4 Special-purpose processor	17
2.5.5 Other architectures	18
2.6 1-bit processing	19

2.7	Hardware and software co-design	22
2.8	Summary	24
3	One-bit Data Conversion	25
3.1	Digital modulation	25
3.1.1	Δ modulation	25
3.1.2	$\Delta\Sigma$ modulation	28
3.1.3	Conclusion	30
3.2	Wavelet analysis	31
3.3	Quantization noise	34
3.3.1	Noise shaping	34
3.3.2	Noise in first order $\Delta\Sigma$ modulation	37
3.3.3	Noise in high-order $\Delta\Sigma$ modulation	39
3.3.4	Stability issue	40
3.4	Realizing the $\Delta\Sigma$ modulator	41
3.5	Summary	43
4	One-bit Processing	45
4.1	One-bit processing	45
4.2	Discrete transforms	46
4.2.1	The z -transform	47
4.2.2	The δ -transform	49
4.2.3	δ -operator vs. z -operator	51
4.3	The state-space approach	51
4.3.1	State-space equations	51
4.3.2	Controller structures	53
4.4	The δ -form in 1-bit processing	55
4.4.1	The δ -form	55
4.4.2	$\Delta\Sigma$ modulated δ -form	57
4.4.3	Stability analysis	60
4.5	Sampling in 1-bit processing	62
4.5.1	Phase delay	62
4.5.2	Quantization noise	64

4.6	Simulation results	70
4.6.1	Validation example	70
4.6.2	Practical DC motor control	73
4.7	Summary	77
5	Direct Implementation	79
5.1	Numerical issue	79
5.1.1	Coefficients	79
5.1.2	Bit-width	81
5.2	Hardware architecture	82
5.2.1	Basic arithmetic blocks	82
5.2.2	VLSI realisation	83
5.2.3	Performance comparisons	85
5.3	Hardware verification	86
5.3.1	RTL modelling of the $\Delta\Sigma$ modulator	87
5.3.2	RTL modelling of the controller	87
5.3.3	Simulation results	89
5.3.4	Hardware performance	93
5.4	Summary	93
6	A $\Delta\Sigma$-based Control System Processor	95
6.1	Hardware architecture	95
6.1.1	Introduction	95
6.1.2	Instruction set architecture (ISA)	96
6.1.3	Microarchitectures	102
6.1.4	A reprogrammable architecture	112
6.2	Software architecture	118
6.2.1	Introduction	118
6.2.2	Control program flowchart	118
6.2.3	ASIS	120
6.3	Simulation results	125
6.3.1	Digital simulation	125
6.3.2	Hardware-in-loop simulation	129

6.4	Summary	133
7	$\Delta\Sigma$-CSP Benchmark	136
7.1	Introduction	136
7.2	Selected processors	137
7.2.1	CSP	137
7.2.2	TMS320C31	137
7.2.3	TMS320C54	138
7.2.4	C167	138
7.2.5	Strong-ARM SA-110	138
7.2.6	Pentium III	138
7.3	Programming and instruction code	140
7.3.1	C program	140
7.3.2	Assembly code	142
7.4	Benchmark results	144
7.4.1	Power consumption	144
7.4.2	Processing speed	145
7.5	Summary	145
8	Conclusions and Future Work	148
8.1	Conclusions	148
8.1.1	Review of the thesis	148
8.1.2	Achievements	148
8.1.3	Limitations	149
8.2	Future work	150
8.2.1	Dual-processor architecture	150
8.2.2	Maglev control	151
8.2.3	Bit-serial architecture	151
8.2.4	MEMS and Microsystem engineering	153
A	General $\Delta\Sigma$-CSP Program	154
B	Publications	157

List of Symbols

A/D: analogue to digital
ALU: arithmetic logic unit
ASIC: application specific integrated circuit
CN: conditional negate
CNA: conditional negate and add
CoGen: codesign generator
CPU: central processing unit
CSP: control system processor
D/A: digital to analogue
DAC: data acquisition card
DSP: digital signal processor
FIR: finite impulse response
FPGA: field programmable gate arrays
HDL: hardware description language
IIR: infinite impulse response
IO: input and output
ISA: instruction set architecture
LTI: linear time invariant
MAC: multiply and accumulation
MEMS: microelectromechanical system

MIMO: multi-input multi-output
MSC: message sequence chart
NTF: noise transfer function
OSR: oversampling ratio
PC: program counter
PCM: pulse code modulation
PDM: pulse density modulation
PMSC: performance message sequence chart
PWM: pulse width modulation
RAM: random access memory
RC: resistor and capacitor
ROM: read only memory
RTL: register transfer level
RTOS: real-time operating system
SA: Strong ARM
SC: switched capacitor
SDL: specification description language
SIMD: single instruction and multiple data
SNR: signal to noise ratio
SOC: system on chip
SRAM: static RAM
STF: signal transfer function
USB: universal serial bus
VLSI: very large-scale integration
 $\Delta\Sigma$ -CSP: $\Delta\Sigma$ modulated control system processor

List of Tables

4.1	Routh tabulation	61
4.2	Sampling and computation factors.	64
5.1	Comparisons between arithmetic operations	83
5.2	24-bit coefficients and errors	88
5.3	States errors	92
6.1	$\Delta\Sigma$ -CSP instructions.	100
6.2	IO adress.	109
6.3	$\Delta\Sigma$ -CSP states.	110
6.4	Arithmetic operations of all the instructions.	111
6.5	Results of the Flop-based design.	116
6.6	Results of the SRAM-based design.	118
7.1	Selected processors for benchmark against the $\Delta\Sigma$ -CSP.	137
7.2	Processors' features.	139
7.3	Processors' data format.	141
7.4	Compilers to generate assembly code for the benchmark.	141
7.5	Benchmark results of power consumption.	145
7.6	Benchmark results of processing speed.	146

List of Figures

1.1	Diagram of the continuous feedback control system.	2
1.2	Diagram of the digital feedback control system.	2
1.3	The canonic δ -form for the CSP.	3
1.4	Generalised $\Delta\Sigma$ modulation.	5
1.5	A generic block diagram of one-bit processing.	6
2.1	$\Delta\Sigma$ data converters: (a) Analogue-to-digital; (b) Digital-to-analogue.	12
2.2	(a) $\Delta\Sigma$ modulation. (b) Equivalent system.	13
2.3	Ritchie's $\Delta\Sigma$ modulator structure Ritchie (1977).	13
2.4	An m -order discrete $\Delta\Sigma$ modulator.	20
2.5	A second order one bit IIR filter.	21
2.6	One bit recursive filter with no multi-bit multipliers	22
2.7	Modified biquad structure	22
2.8	The hardware and software co-design process.	23
3.1	Δ modulation	26
3.2	Simulation of Δ modulation with 128kHz.	27
3.3	Simulation of Δ modulation with 64kHz.	28
3.4	Simulation of Δ modulation with 64kHz and Δ is 0.0625. . . .	29
3.5	$\Delta\Sigma$ modulation.	30
3.6	Second order $\Delta\Sigma$ modulator	32
3.7	1Hz sine wave input	33
3.8	Results of wavelet de-noising	33
3.9	First order linear $\Delta\Sigma$ modulator.	34
3.10	Sampled $\Delta\Sigma$ modulation signal with a sampling rate of 64kHz	35

3.11	Spectrum of the previous $\Delta\Sigma$ signal of 64kHz	36
3.12	Sampled $\Delta\Sigma$ modulation signal with a sampling rate of 128kHz 36	
3.13	Spectrum of previous signal with a sampling rate of 128kHz	37
3.14	First order discrete $\Delta\Sigma$ modulator.	38
3.15	High order linear $\Delta\Sigma$ modulator.	39
3.16	Second order linear $\Delta\Sigma$ modulator with a gain k	41
3.17	The first order $\Delta\Sigma$ modulator based on SC circuit.	43
3.18	The first order $\Delta\Sigma$ modulator based on RC circuit.	43
4.1	Comparison between One-bit processing and conventional dig- ital control	46
4.2	The canonic z -form.	54
4.3	The modified canonic z -form.	55
4.4	(a) The canonic δ -form. (b) The modified canonic δ -form.	57
4.5	The second order $\Delta\Sigma$ modulator	58
4.6	The modified δ -form with multiple $\Delta\Sigma$ modulators.	59
4.7	The modified δ -form with single $\Delta\Sigma$ modulator.	59
4.8	The linearized $\Delta\Sigma$ modulated δ -form.	60
4.9	Quasi-linear model for 1-bit processing.	66
4.10	Calculated SNR with the sinusoidal input.	70
4.11	4 th order $\Delta\Sigma$ modulated δ -form	71
4.12	Responses with $u = 0.9$	72
4.13	Responses with a 1Hz sine wave input	72
4.14	Frequency responses of the 4 th order filter	73
4.15	DC motor diagram.	74
4.16	The overall control scheme.	74
4.17	1-bit control system in the modified δ -form.	75
4.18	SNR and the sampling frequency given a controller bandwidth 0.75Hz.	76
4.19	Simulation results	76
4.20	Motor current.	77
5.1	Re-modified canonic δ -form with scaling factors in the main loop.	80

5.2	Bit-width to represent coefficients and state variables.	82
5.3	Direct implementation in VLSI	84
5.4	Comparison results of the direct implementations.	86
5.5	RTL view of the $\Delta\Sigma$ modulator.	87
5.6	Modified canonic δ -form for the validation example.	88
5.7	RTL view of the validation example.	89
5.8	RTL simulation results.	90
5.9	States differences	91
5.10	Comparison between the denoised output and the continous output.	92
6.1	CNA architecture.	98
6.2	Procedure of executing an instruction.	101
6.3	Data memories architecture.	104
6.4	Sample timer architecture.	105
6.5	Sample time scheme.	106
6.6	Program counter architecture.	107
6.7	Program counter flow.	108
6.8	ALU architecture.	111
6.9	$\Delta\Sigma$ -CSP architecture.	112
6.10	The reprogrammable $\Delta\Sigma$ -CSP architecture.	114
6.11	Flop-based $\Delta\Sigma$ -CSP.	116
6.12	SRAM-based $\Delta\Sigma$ -CSP.	117
6.13	Control program flowchart.	119
6.14	Instruction format.	121
6.15	An add operation.	124
6.16	The validation example program and its binary codes.	126
6.17	Simulation results.	127
6.18	RTL simulation results of the $\Delta\Sigma$ -CSP hardware and software architecture.	128
6.19	$\Delta\Sigma$ -CSP interface.	129
6.20	Hardware-in-loop simulation scheme.	131
6.21	Hardware-in-loop simulation workbench.	132

6.22 Comparisons between the hardware-in-loop simulation and the
digital simulation. 134

7.1 C program for the validation example. 140

7.2 Instruction code for the CSP. 142

7.3 Assembly code for the TMS320C31. 142

7.4 Assembly code for the C167. 143

7.5 Assembly code for the Strong-ARM. 144

8.1 Bit-serial and bit-parallel communication strategies 152

Chapter 1

Introduction

1.1 Background

There is now a variety of control design methods by which appropriate control laws can be created for complex multi-variable systems, but the actual implementation of control laws is a part of the design process which most control engineers strive to achieve as straightforwardly and transparently as possible.

Real-world automatic control systems were primarily based on analogue electronics. Fig. 1.1 shows a typical block diagram of the continuous feedback control system. With the availability and low-cost high-performance digital electronics, control of such systems evolved into the more flexible digital form which is now used almost exclusively (Forsythe and Goodall, 1991; Middleton and Goodwin, 1990; Paraskevopoulos, 1996). The equivalent digital control version of Fig. 1.1 is shown in Fig. 1.2.

The title of this thesis includes an important term — ‘real-time’ in digital control. What does ‘real-time’ mean? Cooling (1991) offers the definition: *Real-time systems are those which must produce correct responses within a definite time limit. Should computer responses exceed these time bounds then performance degradation and/or malfunction results.* The time limit normally means the sampling interval, being dependent on the time constant of the plant to be controlled. The shorter the time constant of the plant,

1.1. BACKGROUND

Figure 1.1: Diagram of the continuous feedback control system: $u(t)$ is the input, $r(t)$ is the plant output, $e(t)$ is the error to be controlled and $c(t)$ is the manipulated variable.

Figure 1.2: Diagram of the digital feedback control system: $u(nT)$, $r(nT)$, $e(nT)$, $c(nT)$ are digital values of $u(t)$, $r(t)$, $e(t)$, $c(t)$ at the time nT where n is an integer and T is the sampling interval.

the faster the sampling rate. The principal outcome, which is the implementation of the control law via a dedicated digital controller along with its associated analogue IOs, therefore must be as efficient as possible for real-time processing, being able to carry out all the required operations – measurement, control and actuation – within each sampling interval. The sampling rate must be at least twice the bandwidth of the plant, which is well known as the *Nyquist sampling rate*. But in practice a factor of 100 times the bandwidth is a more effective sampling rate for high performance digital controllers (Goodall, 2001). Less than this and the stability will increasingly be compromised by the use of digital control.

Much of the digital hardware developed to date has tended to be a general-purpose microprocessor or digital signal processor (DSP) with little analysis of the underlying requirements of control systems. The difficulty is that there are particular numerical requirements in control system processing for which standard processor devices are not well suited, in particular arising from the high sampling rates which are needed to avoid adverse effects of sampling de-

1.2. PREVIOUS RESEARCH

Figure 1.3: The canonic δ -form for the CSP.

lays upon stability. These could be satisfied in either general microprocessor or DSP devices in a high level language, but this is neither straightforward nor transparent for digital control. There is therefore a clear need to understand the numerical requirements properly, to identify optimised forms for implementing control laws, and to translate these into efficient architectures.

From the prospect of control, the solution is to adopt a system approach and bring together the control and electronic design requirements. This is related to hardware and software co-design, a major subject which has seen substantial research progress over the last few years (Ernst, 1998).

1.2 Previous research

As a consequence of hardware and software co-design, a control system processor (CSP) was designed for real-time controller implementations (Goodall et al., 1998; Cumplido-Parra, 2001). The CSP is a high-speed, low-cost special purpose control processor that can execute extremely fast control laws for linear time invariant (LTI) systems. The excellent performance of the CSP is achieved by the reformulation of the control algorithms into a particular state-space representation based on the δ operator, which is used to represent discrete transfer functions instead of the conventional shift operator. The canonic δ form is illustrated in Fig. 1.3 for a generalised single-input single-output controller of second order.

The δ operator avoids some of the problems arisen from high sampling frequencies (Middleton and Goodwin, 1990), which result in long word length

1.2. PREVIOUS RESEARCH

requirements for the coefficients. This means low coefficient sensitivity and allows the use of short word lengths to represent the coefficients. In the CSP, it applies a simple low-precision floating-point form with a 6-bit mantissa in 2's complement format and a 5-bit exponent. The exponent has a biased range of +6 to -25. This format allows to represent any coefficient with an accuracy of 1%, which is more than enough for most control applications (Forsythe and Goodall, 1991). However, the state variables' word lengths need to be carefully chosen to ensure that the full value and dynamic range of the variables involved in the calculation can be accommodated. The variables are 27-bit fixed point in the CSP, with 12 bits for the IO values as a requirement of 12-bit data converters, 3 overflow bits in order to ensure correct operation and to reduce the number of overflow check, and 12 underflow bits.

The processor design is very simple. It comprises a 4-port register bank (3 read, 1 write) that is associated with a special-purpose multiply accumulator (MAC). This MAC calculates

$$D = A \times B + C \quad (1.1)$$

in a single cycle and writes the result back to the register bank. The A input is in coefficient format, and B , C in state-variable format. The MAC is the only instruction in the CSP, being able to process most calculations of a control law. Other instructions include load, write operations.

The CSP was manufactured and tested. Although having 40 times fewer gates than a DSP, it executes control laws between 4 and 33 times faster than many high-end DSP devices. The 50MHz CSP is beaten only by the 233MHz SA (strong ARM) and the 500MHz Pentium III, both of which are more expensive than the CSP. As such it is the most high-performance control engine developed to date.

1.3 Research motivation

Data converters (A/D and D/A) are necessary in digital control, although sometimes the D/A converter can be replaced with a high-frequency PWM output, particularly appropriate when a switching-type power amplifier is used to drive the physical system. Due to the requirement of very high sampling frequency, data converters must be efficient for real-time processing.

There are many types of data converter (Daugherty, 1994), among which $\Delta\Sigma$ converters have been rapidly gaining popularity in the past few years (Candy, 1992). The $\Delta\Sigma$ converter utilizes $\Delta\Sigma$ modulation as an efficient algorithm to encode the analogue signal or multi-bit digital signal into 1-bit format. Fig. 1.4 illustrates a general scheme of the $\Delta\Sigma$ modulation.

Figure 1.4: Generalised $\Delta\Sigma$ modulation.

The output after quantisation is a binary value, either $+\Delta$ or $-\Delta$, being able to be represented by a 1-bit register in VLSI. The $\Delta\Sigma$ modulated signal therefore is called 1-bit signal. A decimation filter is needed to construct a complete A/D converter, or an interpolation filter for a D/A converter. However, the 1-bit signal itself contains all the useful information within the signal bandwidth (Angus, 1998), being a perfectly valid representation of the input.

Fig. 1.5 shows a generic block diagram of one-bit processing. In conventional digital control, the parallel binary numbers after A/D conversion are multiplied with the coefficients, which represent the characteristic of a control system, to produce parallel binary outputs, resulting in multi-bit multipliers. However, the 1-bit signals, being multiplied with the coefficients, only change the sign of the coefficients. This operation will greatly increase the speed and reduce the size on the silicon compared to a multi-bit multiplier.

Figure 1.5: A generic block diagram of one-bit processing.

Utilising the 1-bit signal in control along with its dedicated analogue IOs therefore may result in a more efficient control system processor than the current CSP for real-time processing. This is a totally new area in digital control and as far as the author is aware there has been no other work on the subject. But application of 1-bit processing for real-time control brings particular issues in terms of sampling criterion, numerical requirement and hardware implementation that need to be understood.

1.4 Dissertation overview

This thesis presents the research work which applies $\Delta\Sigma$ modulation in real-time control. This work results in a novel digital control concept and a very efficient control system processor architecture. The remainder is organised as follows.

Chapter 2 reviews related literature. The literature includes the basic concepts of control system processing, a brief history of 1-bit processing in the areas of communication and audio processing, hardware architectures developed to date.

Chapter 3 gives some basics of $\Delta\Sigma$ modulation, and explains the reason that it is possible to consider 1-bit processing for real-time control. It also gives a brief history of $\Delta\Sigma$ modulation at the beginning.

Chapter 4 provides a definition of one-bit processing. A special controller structure that utilises the δ -operator is developed in this chapter. It gives an approach of obtaining the sampling criterion for one-bit processing. Two

1.5. SUMMARY

control examples are also demonstrated to verify the concept.

Chapter 5 presents a hardware architecture of direct implementation for one-bit processing applications in VLSI. Numerical representations are given in this chapter. A validation example is used to validate the hardware functions in RTL level.

Chapter 6 presents a processor solution for implementing one-bit processing applications in VLSI. It explains the essential components of the $\Delta\Sigma$ -CSP architecture based on the selected control form. An extra reprogrammable $\Delta\Sigma$ -CSP architecture is discussed as well. It also presents the synthesis results in terms of speed and complexity.

The software scheme of programming one-bit processing applications is explained. It describes the application specific instructions and their formats. Hardware-in-loop simulation is used to verify the processor architecture along with one-bit processing for real-time control.

Chapter 7 presents the benchmark results of the $\Delta\Sigma$ -CSP against other processors in terms of speed and power consumption.

Chapter 8 concludes and some future work is presented.

1.5 Summary

In this chapter the background and motivation of the research were given. Hardware and software co-design was believed a key to implementing control laws as straightforwardly and transparently as possible for control engineers. The CSP is a very efficient application-specific architecture for control, but we believe that applying one-bit processing together with the proposed hardware architectures would be even more powerful and efficient for real-time control applications.

Chapter 2

Literature Review

2.1 Control system design

The design process of a control system is described by Nise (1995). At the beginning, the plant must be modelled in mathematics. The control system therefore can be designed and analyzed according to the design specifications such as desired transient response and steady-state accuracy.

Many control design methods are available today. These methods are divided into two branches (Goodall, 2002): classical and modern control.

Classical control applies time or frequency domain techniques to analyze the plant and design the compensator. This approach is based on converting a system's differential equation to a transfer function, thus generating a mathematical model that algebraically relates a representation of the output to a representation of the input. An advantage of these techniques is that they rapidly provide stability and transient response information. The primary disadvantage of the classical approach is its limited applicability, being practicable only for linear, time-invariant systems of relatively low complexity and usually having a single input and output. The languages for classical control are Laplace transform or z -transform, which describe the relationship between the input and output of a control system.

Modern control, however, benefits from the advances in computer technology (Brogan, 1985). First, the physical system can be modelled into a

more complex one, which means a large number of variables, nonlinearities and time-varying parameters must be included in the model. Second, the computer technology is well suited for the need of greater accuracy and efficiency, which has changed the emphasis on control system performance. Third, computers are now so commonly used as just another component in the control system, which means that the discrete-time and digital system control now deserves much more attention than it did in the past. In addition, the foundation of modern control theory is the state-space models, being ideal for calculations in digital control.

2.2 Digital control basics

2.2.1 Design and analysis techniques

The normal digital control scheme has been shown in Fig. 1.2. The Laplace transform is the basic tool in analyzing and designing both classical control and modern digital control systems. However, typical Laplace transform expressions of systems involving sampled signals all contain exponential terms in the form of e^{Ts} (Kuo, 1992), making it difficult for the manipulation of the transform expressions in the Laplace domain. z -transform therefore has been widely accepted as an effective tool in digital control. The transformation from the complex variable s to z is accomplished by

$$s = \frac{1}{T} \ln z \quad (2.1)$$

where T is the sampling time. The analysis and techniques for continuous control, such as the Routh-Hurwitz criterion and Bode techniques, cannot be applied in the z -plane (Phillips and Nagle, 1990). This is because the stability boundary in the s -plane is the imaginary axis, but in the z -plane is the unit circle. However, the z -plane can be transformed into the imaginary axis of the ω -plane through the use of the transformation

$$z = \frac{1 + (T/2)\omega}{1 - (T/2)\omega} \quad (2.2)$$

2.2. DIGITAL CONTROL BASICS

or for ω ,

$$\omega = \frac{2}{T} \frac{z - 1}{z + 1} \quad (2.3)$$

The ω -plane frequency is approximately equal to the s -plane frequency when

$$\omega T \leq \pi/5 \quad (2.4)$$

(Phillips and Nagle, 1990). In practice therefore the transformation from s to z can be accomplished by

$$s = \frac{2}{T} \frac{z - 1}{z + 1} \quad (2.5)$$

which is well known as the bilinear transform (Mohler, 1973).

2.2.2 Sampling in digital control

A key characteristic of a digital control system is the sampling rate. It is the rate at which analogue input values are sampled or processed. The sampling rate, combined with the algorithm complexity, determines the required speed of the controller implementation.

It is well known from the Nyquist sampling theorem that any signals with a frequency beyond $f_s/2$ (f_s is the sampling frequency) can not be replicated. This means that the minimum sampling frequency has to be greater than twice the highest frequency of the signal bandwidth. However, according to Phillips and Nagle (1990) sampling at the theoretical minimum will introduce a phase lag of at least 180° , which is not sufficient for real-time control. A realistic criterion for real-time control is that there should be no more than 5° of phase lag (Goodall, 2001). This requires at least 100 times of the controller bandwidth. Strictly speaking, real signals do not have bandwidth limits, i.e. there are still small frequency components outside the bandwidth (Middleton and Goodwin, 1990). When implementing a digital control system, it is always required to sample at a higher rate than the theoretical minimum (Feuer and Goodwin, 1996).

Slow sampling frequency results in poor control performance. It is well

2.3. DATA CONVERSION

known that, however, when the sampling frequency is extremely high, significant numerical problems may be introduced. This is because it is difficult to represent the small signal values involved in the calculations (Middleton and Goodwin, 1990) due to the effects of finite word length, and any small change of the coefficients will result in a major error of the system output. In order to overcome the numerical problems in high sampling frequency, the δ -operator was introduced (Goodall and Brown, 1985; Goodwin, 1985). The δ -operator is more like a derivative, resembling the continuous operator d/dt , the controller being low-sensitive to the change of the coefficients, which allows an error of 5% for the coefficients to be represented in hardware. Therefore, using the δ -operator fundamentally avoids the numerical problems and enables very high sampling frequencies to be achieved (Goodall and Donoghue, 1993). This property is exploited by Middleton and Goodwin (1990) to provide a unification between continuous and discrete-time systems. In this case, the discrete systems can achieve the same effects of the continuous systems if the sampling frequency is infinitely high.

2.3 Data conversion

Clayton (1982) and Daugherty (1994) described many types of data converter, including flash converter, single-slope converter, dual-slope converter, sampling (successive-approximation) converter, R-2R converter, voltage-to-frequency converter, RC converter, resistance measurement converter, pulse-width modulation (PWM) converter, improved PWM converter and $\Delta\Sigma$ converter. Another name of pulse-width-modulation is Δ modulation (Steele, 1975). A sample and hold (S/H) circuit may be required by most data converters, either internal or external (Carr, 1980). This is due to either the actual input signal frequency or system-induced noise causing the input to change rapidly. However, the types of PWM and $\Delta\Sigma$ converter do not require the S/H circuit because they are not susceptible to high frequency signals (or noise) due to their averaging mode of operation.

$\Delta\Sigma$ modulation was first explored by Inose and Yasuda (1963) in 1963, and was accepted as an effective method for building high resolution data

2.3. DATA CONVERSION

converters. Fig. 2.1 shows $\Delta\Sigma$ A/D and D/A converters. The $\Delta\Sigma$ modulator (Fig. 2.2(a)) adds a filter to the front end of a Δ modulator and then moves it inside the loop (Fig. 2.2(b)) (Gray, 1987). The Δ modulator was proposed even earlier, in 1952, by de Jager (1952). It contained a 1-bit quantiser in the forward loop and a filter (in the simplest case, an integrator) in the feedback loop. This structure has a low dynamic range and causes a cumulative error. But the $\Delta\Sigma$ modulator is free of these problems (Norsworthy et al., 1997). For high order $\Delta\Sigma$ modulators, a filter structure was developed by Ritchie (1977). Fig. 2.3 shows this structure, in which he proposed using several integrators in cascade in the forward loop to create a higher order filter, with each integrator receiving an additional input from the quantiser.

Figure 2.1: $\Delta\Sigma$ data converters: (a) Analogue-to-digital; (b) Digital-to-analogue.

The $\Delta\Sigma$ modulator is a nonlinear system. For more than two integrators in the loop, the stability becomes hard to analyze and has to be verified by numerical simulation. Design techniques for stable high order $\Delta\Sigma$ modulators have been investigated (Chao et al., 1990). The nonlinear behaviour in $\Delta\Sigma$ modulators was studied in (Ardalan and Paulos, 1987) based on modelling the nonlinear quantiser with a linearized gain followed by an additive white noise source.

Both $\Delta\Sigma$ modulation and Δ modulation produce a set of pluses and belong to the pulse-code-modulation (PCM) technique (Cattermole, 1969). Although PCM had been investigated for over half a century, it has only gained popularity in the last 20 years. The bottleneck of the PCM technology

Figure 2.2: (a) $\Delta\Sigma$ modulation. (b) Equivalent system.

Figure 2.3: Ritchie's $\Delta\Sigma$ modulator structure Ritchie (1977).

lies in a digital logic implementation (Waggener, 1995) for decoding the pulses into their corresponding multi-bit format. However, the fast development of advanced VLSI technology is destined to establish PCM as the dominant technology in data conversion.

2.4 Numerical issues

In digital control systems implementation, it is an important issue to determine the type of binary numeric representation for implementation of a digital controller. The numeric representation and the type of arithmetic used can have a profound influence on the behaviour and performance of the controller.

Before the control engineers implement a control law, they need to choose

2.4. NUMERICAL ISSUES

a fixed-point or a floating-point arithmetic for the representation of coefficients and state variables. The decision largely depends on the budget of the project and the size of the targeted controller. Most microcontrollers and some DSPs use the fixed-point arithmetic in which only a finite word length with a fixed scaling is available to represent the state variables and coefficients. It is always possible to introduce floating-point calculations, for example by means of a compiler, but each calculation will then take many processor cycles. For computational efficiency, state variables and coefficients must be scaled to fit the word length provided by the processor. The fixed point arithmetic is a low-cost solution for digital controller implementations compared to the floating-point arithmetic, and is widely applied in cost-sensitive applications (Berkeley Design Technology, Inc., 2000; Goodall, 2001; Schlett, 1998).

The fixed-point arithmetic represents the number in a fixed range with a finite number of bits (word width). Numbers outside the specified range can only be represented if they are scaled, in which case the scalings must be allowed in the computations. The floating-point arithmetic still provides a fixed word length, but expands the available range of values. It represents the number in two parts: a mantissa and an exponent. The mantissa value lies between -1.0 and 1.0, while exponent scales (in terms of powers of two) the mantissa value in order to create the actual value represented (Cumplido-Parra, 2001). Note, however, that the mantissa and exponent will have a fixed word length. The floating-point arithmetic offers an ease-of-use advantage due to the fact that it provides wider dynamic range and usually gives higher precision than fixed-point arithmetic does. The increase of dynamic range also allows a designer to ignore scaling problems because it reduces the probability of overflow. In contrast, with the fixed-point arithmetic, sometimes it is necessary to scale signals at various stages of the program to ensure adequate numeric performance. Unfortunately, the floating-point arithmetic is generally slower, more expensive and more difficult to implement in hardware. The increased cost results from the more complex circuitry required. In addition, the larger word sizes of floating-point processors often means the memory and buses are wider, raising the overall system cost.

Thus, the choice of the fixed-point or floating-point arithmetic is determined by the system requirements in terms of dynamic range and precision as well as price and size. The dynamic range is the ratio, usually expressed in dB , between the largest and smallest numbers that can be represented. The precision of a digital system is dependent upon the word-length that the arithmetic uses.

2.5 Digital devices

To implement a digital controller, it is necessary to map the control law into some kind of architecture that will actually perform the task. There are many alternatives, it might be implemented in software on general-purpose processors, microcontrollers, digital signal processors or it might be implemented in special-purpose processors. Control applications may also take advantage of entire platforms built around general-purpose processors like personal computers, workstations and stand-alone boards.

2.5.1 General purpose processor

General purpose processors are not a cost-effective solution in many applications, and often the performance requirements in terms of throughput, power consumption and size cannot be met (Berkeley Design Technology, Inc., 2000; Irwin, 1998). The reason for this is the mismatch between general-purpose processor architectures and most control algorithms that require a large number of repeated arithmetic operations of a relatively simple nature and a small number of input/output operations.

General-purpose processors are designed to perform a multitude of functions to support applications which rely almost entirely on manipulation of data; this involves storing, organizing, sorting and retrieving information. To perform those tasks, the processors provide a number of functions that allow wide-ranging mixtures of operations and control flow that can be data dependent, making large jumps from one area of the program memory to another. Thus, the ability to move data from one location to another and

2.5. DIGITAL DEVICES

testing for inequalities ($A = B$, $A < B$, etc.) becomes essential (Lapsley et al., 1997).

These processors were not originally designed for multiplication-intensive tasks, even some modern processors would require several instruction cycles to complete a multiplication because they do not have dedicated hardware for single-cycle multiplication, and as a consequence they are not well suited to perform control algorithms (Berkeley Design Technology, Inc., 2000). To solve this problem, high-end processors have been enhanced to increase the computation of arithmetic-intensive tasks. A common modification is the addition of SIMD-based instruction set extensions that take advantage of wide resources such as buses, registers and ALUs, which can be seen as multiple smaller resources. However, despite the high performance operation offered by these processors, they are not widely used in embedded applications due to their cost (Eyre and Bier, 1999).

2.5.2 Microcontroller

A microcontroller design is focused on integrating the peripherals needed to provide control within an embedded environment and a microprocessor core. Commonly, a microcontroller incorporates in a single chip at least the necessary components of a complete computer system: CPU, memory, clock oscillator and input/output ports, plus some additional elements such as timers, serial units, and analogue-to-digital/digital-to-analogue converters. These features allow them to be simply wired into a circuit with very little support requirements; usually, they only require power and clocking (Predko, 1999; Cady, 1997).

The primary role of microcontrollers is to provide inexpensive, programmable logic control and interfacing to external devices. Thus, they are not expected to provide arithmetic-intensive functions. When included within complex systems applications, they are used to interpret input, communicate with other devices, and output data to a variety of different devices. Microcontrollers add a great deal of flexibility in the product development process as they can be used for a variety of applications. Another advantage is the

fact that microcontrollers are member of families that present many different combinations of hardware features, so the most suitable device for a specific application can be selected where possible. The programs to be executed are stored in the internal memory (ROM or RAM) to provide a single chip solution.

2.5.3 Digital signal processor

Digital signal processors (DSP) have been designed to overcome some of the limitations found in general-purpose processors. DSPs introduce some architectural features that accelerate the execution of repetitive multiply-accumulate operations of digital control algorithms (Eyre and Bier, 1999).

DSPs can be used for controlling external digital hardware as well as processing the input signals and formulating appropriate output signals. Although most real-time digital control applications require a large amount of data calculations, the programs that implement them are normally very simple. As a result, these programs can be stored in internal memory to reduce the transfer time. The design process involves mainly coding the control algorithm either using a high-level language or directly in assembly language. Then the source code is compiled into an object code that can be executed by the processor.

This approach allows rapid prototyping, but unfortunately it is not always possible to meet the requirements of power consumption, size or cost. The main reason is that the standard DSP is designed to be flexible in order to support a wide range of digital signal processing algorithms that use only a few of instructions provided (Lapsley et al., 1997).

2.5.4 Special-purpose processor

Special-purpose processors, with a particular combination of registers, logic elements and interconnections, open the possibility of achieving in one clock cycle what a traditional programmable processor requires tens or even hundreds of clock cycle (Cumplido-Parra, 2001). The term special-purpose processor has been used to define a wide range of degrees of dedication and

specialization. We can say that a special-purpose digital control processor is a dedicated hardware entity whose function is to perform a specific, well defined, set of digital control algorithms in real-time. Just as DSPs are more efficient and cost-effective than general-purpose processors to execute high-speed arithmetic operations, special-purpose processors have the potential of overpower DSPs due to their specialized nature. As only the required functions are placed in hardware, special-purpose processors can be less expensive than other processors, especially for high volume products.

The possibility of integrating a whole control system into one chip has several effects. It increases the processing capacity and simultaneously reduces the size of the system, power consumption, and pin restriction problems. Additionally, it improves system reliability and offers protection of intellectual property. Of course, developing special-purpose architectures presents some drawbacks. Among them are the effort and expense associated with custom hardware development, especially for custom chip design. However, the problems associated with custom hardware can be partially solved using high-level hardware design languages such as VHDL and logic synthesis CAD suits allied to large low-cost reprogrammable FPGAs (Cumplido-Parra, 2001). A major advantage of this approach is that the word length can be adjusted to the system's requirements. Thus the size of the architecture can be kept to a minimum. However, the performance improvements come with the cost of larger design effort.

2.5.5 Other architectures

Other architectures include general purpose parallel processors which are based on multi-processor or multi-computer systems (Wanhammer, 1999), fuzzy logic controllers which can be applied to systems with undefined boundaries that are difficult to represent using explicit difference or differential equations (Costa et al., 1997) and combined approaches which look into the integration of the DSP functionality with the microcontroller to offer the benefits of both the architectures (Eyre and Bier, 1999).

2.6 1-bit processing

The term of 1-bit processing is originally from the audio industry. Typically, digital audio systems sample audio at 44,100 or 48,000 times every second (the audio frequency is normally 22kHz) (Robjohns, 1998), although there are many other 'standard' sample rates. The regularity and stability of the timing in the sampling process is absolutely crucial to the ultimate quality of the digital audio system – timing inaccuracies introduced here cannot be removed later, and will result in unstable stereo imaging and increased noise. The Nyquist theorem states that the sampling rate must be at least twice the highest audio frequency being sampled. Consequently, the highest audio frequency a digital system is required to encode must be specified, and nothing above this frequency can be allowed to enter the system. This is achieved with an anti-aliasing filter which would typically have a cutoff slope in the order of 200dB/octave. Early analogue filter designs were extremely expensive to manufacture, prone to drift, and tended to sound dreadful!

Audio signals are currently processed using a multi-bit representation of the signal that is sampled at a rate just above the theoretical minimum (around 44kHz). This has the disadvantage of requiring both word and bit synchronization in order to transfer signals between processing modules. In addition the phase response of filters is significantly affected by the proximity of the Nyquist limit (Angus and Draper, 1998).

The sampling process chops up the analogue audio signal ready for quantization. However, the process is actually a form of modulation where the audio signal is modulated into the amplitudes of the individual samples. Here, $\Delta\Sigma$ modulators are used in the modulation process and the modulated signals are in a format of 1 or -1, which can be represented by one-bit registers in the digital systems. Any modulation process produces images of the original audio at the sum and difference frequencies – in this case between the audio signal and the sampling rate – and although these images are a side-effect of the process and serve no practical purpose, they do have significant implications. Recently, recording the one bit signal directly has been proposed as a possible alternative to a multi-recording format (Angus,

Figure 2.4: An m -order discrete $\Delta\Sigma$ modulator.

1994; Johns and Lewis, 1991). One of its advantages is that it removes the decimating or interpolating requirements at the analogue interface. It also allows a simpler system structure because the interconnections are naturally serial with no implied framing. Also, because the signal is heavily oversampled, the system characteristics can approach those of high quality analogue processors in terms of phase response and distortion effects, while retaining the advantages of digital processing techniques.

Now 1-bit processing has been widely investigated in the context of finite-impulse-response (FIR) filters (Kershaw et al., 1996; Kershaw, 1996; Summerfield et al., 1994; Wong and Gray, 1990), infinite-impulse-response (IIR) filters (Kershaw, 1996; Johns and Lewis, 1991, 1993), audio processing (Angus and Draper, 1998; Angus, 1998), digital communication (Stewart and Pfann, 1998) and control system processing (Wu and Goodall, 2003, 2004).

A key technique in 1-bit processing is $\Delta\Sigma$ modulation — an algorithm by which analogue and digital signals are coded in a low resolution and high sampling rate format. A simple implementation of an m -order continuous $\Delta\Sigma$ modulator is illustrated in Fig. 1.4 that is due to Ritchie (1977) and described by Tewksbury and Hallock (1978). $\Delta\Sigma$ modulators are used in analogue-to-digital conversions and digital controllers, hence realized by analogue format and digital format. Many analogue realizations do use discrete time via switched capacitor or switched current circuit (Kershaw et al., 1996). The z -domain is thus not only convenient, but often the most general framework for analysis. Fig. 2.4 illustrates a discrete format of an m -order continuous $\Delta\Sigma$ modulator. Here the output of a $\Delta\Sigma$ modulator is defined by either 1 or -1.

The quantization of the $\Delta\Sigma$ modulator introduces non-linearity into the

Figure 2.5: A second order one bit IIR filter.

system. Although exact methods exist for solving the non-linear differential equations implied by the quantiser, they are usually too complex to be of any real practical use. Instead, Atherton (1982) describes two approaches: linearisation and quasi-linearisation. The first involves exchanging the quantiser for a constant linear gain, the second with a signal dependent gain. Kershaw (1996) describes a linear $\Delta\Sigma$ modulator, where the quantiser is a gain element K and an input noise source. Johns and Lewis (1993) make an assumption that the $\Delta\Sigma$ modulator of any order introduces a unit delay at t_s , which is the sampling time, from input to output.

The other important issue in 1-bit processing is to choose a suitable system structure. Angus (1998) presented an expensive system to realize a second order filter (Fig. 2.5). In this structure, the audio filter feedback signal is multi-bit and results in multi-bit multiplications. Johns and Lewis (1993) designed a one bit recursive filter without multi-bit multipliers (Fig. 2.6). This filter is based on the biquad structure with integrators rather than simple delays. They place a $\Delta\Sigma$ modulator of arbitrary order after each integrator. Unfortunately the noise performance of this structure is poor. An improved version of this filter has been designed by Kershaw et al. (1996). This filter (Fig. 2.7) is also based on the biquad structure. It combines both the audio filter and $\Delta\Sigma$ modulator using a series of integrators. In this structure, there also presents power of two coupling coefficients between the stages which has the effect of reducing the internal dynamic range. The value of c_0

Figure 2.6: The Johns and Lewis one bit recursive filter with no multi-bit multipliers.

Figure 2.7: Modified biquad structure with integral power of two coefficients.

is a power of 2, resulting in a shift operation other than a multiplication. No multipliers therefore are needed in this structure as well.

2.7 Hardware and software co-design

The hardware and software co-design is a major subject which has seen substantial research progress over the last ten years (Ernst, 1998; Ong et al., 1997; Shulz et al., 1998). The need for hardware and software co-design techniques is being driven by numerous factors, including shrinking time-to-market constraints, the migration of programmable software processors and hardware processors onto a single chip, and the increasing gap between silicon capacity and designer productivity (IEEE Design & Test Roundtable, 2000).

Fig. 2.8 shows the co-design process which was described by Ernst (1998). The concurrent design starting with the informal requirements from the customer or marketing analysis. These requirements are transformed to a formal

Figure 2.8: The hardware and software co-design process.

specification. System architects define a system architecture consisting of co-operating system functions that form the basis of concurrent hardware and software design. Software developers need to develop application softwares, compilers, and even operating systems for real-time processing. In hardware design, hardware architectures must be considered to run the system optimally. A well-defined hardware architecture must be verified with software execution running on it, resulting in hardware and software co-simulation. Thereafter this hardware architecture can be synthesized and put into physical place & route and floor plan. Interface design requires the participation of both hardware and software engineers to develop software drivers and synthesize the hardware interface. Finally, the hardware and software are integrated and tested. In the hardware and software co-design, reusing components taken from previous designs or acquired from outside is also necessary in order to improve productivity and reduce design risk.

There are also many research works regarding the automatic hardware-software configuration. Mooney III and Blough (2002) developed a real-time operating system (RTOS) framework for the hardware and software co-

design. The so called δ framework helps the designers simultaneously build a system-on-chip (SoC) or platform-ASIC architecture and a customized hardware-software RTOS. Slomka et al. (2000) described many tools for the analysis, synthesis, and rapid prototyping of the hardware and software co-design. These tools include the specification language SDL and SDL*, the message sequence chart (MSC) and the performance message sequence chart (PMSM). A synthesis tool called codesign generator (CoGen) is used to translate the behavior of the SDL* processes into conventional implementation languages such as VHDL for the hardware and C for the software modules.

2.8 Summary

Many references that relate to the research are reviewed in this chapter. Although there are many control design methods, this thesis is not concerned with the design of any control law because 1-bit processing is an approach to implement control laws. 1-bit processing is a kind of digital control, but the design method is different with utilizing more simple analogue IOs and more effective controller structures. The conventional sampling criterion is not applicable in 1-bit processing, meaning a new sampling criterion must be considered in this thesis. There are a lot of digital devices which are suitable for implementing control laws, but they don't lead to an effective solution for 1-bit processing. New hardware architectures therefore are required.

It is generally recognized that different co-design approaches are used in different industry applications. The hardware and software co-design in fact is a way of thinking analogous with the mechatronic approach which is driving the system design in control engineering. As such it is implicit in our research as we will intrinsically address the specification and partitioning aspects in order to create a generic architectural framework for 1-bit processing, these being essential features of how people define co-design. The established principles of hardware and software co-design therefore will naturally be incorporated into the research.

Chapter 3

One-bit Data Conversion

There are many types of data converter (Clayton, 1982; Daugherty, 1994) among which the PWM converter, improved PWM converter and $\Delta\Sigma$ converter attract special interest. These data converters encode an analogue signal into a series of binary pulses, then decode the pulses into the corresponding multi-bit digital signal. Since the binary pulses are already in digital format, in practice a microcontroller is placed after the encoder and functions as a decoder. Either digital control or digital signal processing conventionally works on the multi-bit digital signal, but as it is explained in the previous chapters, we are interested in control system processing on the binary pulses directly without decoding.

3.1 Digital modulation

3.1.1 Δ modulation

Δ modulation is the other name of pulse-width-modulation which can be traced back to as early as 1940s when it was first developed for voice telephony applications (de Jager, 1952). A Δ modulation encoder is shown in Fig. 3.1. It is known as a single integration modulator because there is only one integrator in the feedback loop.

The Δ modulator encodes the differences in the signal amplitude instead of the signal amplitude itself: The input signal is compared to the integrated

Figure 3.1: Δ modulation, f_s is the sampling frequency.

output pulses and the difference (Δ) is applied to the quantizer which generates a positive pulse when the difference signal is negative, and a negative pulse when the difference signal is positive. This difference signal moves the integrator step by step closer to the present value input, tracking the derivative of the input signal.

The conventional PWM converter uses a fixed-frequency square-wave signal with a variable duty cycle which can be averaged by a low-pass filter. Assuming that the binary pulse goes from 0 to V_{ref} which is a reference voltage, the corresponding multi-bit digital format $\hat{u}$ of the input u is determined by the following equation:

$$\hat{u} = V_{ref} \times p \quad (3.1)$$

where p is the duty cycle.

This conventional PWM technique can take more time than desired (Daugherty, 1994). Instead of relying on a single, long period with a duty cycle, the improved PWM technique works by averaging several short pulses of equal duration over a fixed time. The corresponding multi-bit digital format $\hat{u}$ of the input u therefore is determined by

$$\hat{u} = V_{ref} \times \frac{\sum_1^N y_h}{N} \quad (3.2)$$

where y_h is the pulse with high state and $\sum_1^N y_h$ is the sum of high pulses

3.1. DIGITAL MODULATION

Figure 3.2: Simulation of Δ modulation with 128kHz.

within N pulses which is the sum of both high and low pulses. The total conversion time t_c therefore is given by

$$t_c = 2^{m+1} \times t_p \quad (3.3)$$

in order to track the input signal perfectly, where m is the word length and t_p is the duration of a single pulse.

A 1.5kHz sinusoidal input signal with maximum amplitude 1 is considered as an example. Δ is chosen to be 0.125. To achieve a resolution equivalent to 4 bit with 4kHz sampling rate, a sampling rate of 128kHz is needed. Fig. 3.2 shows the simulation results with a sampling frequency of 128kHz and the output of the integrator can track the input signal with a phase lag that is less than 1° . Fig. 3.3, however, shows the simulation results with half the requested sampling frequency and the output of the integrator introduces a phase lag more than 5° , resulting in a low-precision resolution.

If the size of Δ is too low or the sampling rate too slow, a slope overload

Figure 3.3: Simulation of Δ modulation with 64kHz.

occurs (Steele, 1975). In Fig. 3.4, Δ is chosen to be half the previous rate. With the same sampling rate as the previous example, slope overload is inevitable because the integral of Δ is insufficient to track the changes.

3.1.2 $\Delta\Sigma$ modulation

$\Delta\Sigma$ modulation was developed in 1960s based on the Δ modulation. It encodes the difference (Δ) between the current signal and the sum (Σ) of the previous difference. Just as Δ modulation is well known as pulse-width-modulation, $\Delta\Sigma$ modulation is also known as pulse-density-modulation (PDM) because it quantizes the signal directly, rather than the signal's derivative as in Δ modulation. Thus the maximum quantizer range is determined by the maximum signal amplitude.

The quantization level can be defined as the discrete value assigned to a particular subrange of the analog signal being quantized. It works similarly to the improved PWM converter — to achieve a high resolution, a high

3.1. DIGITAL MODULATION

Figure 3.4: Simulation of Δ modulation with 64kHz and Δ is 0.0625.

sampling rate is required. Technically, it is therefore called oversampling $\Delta\Sigma$ modulation. If the frequency of interest is from 0 to f_0 , the oversampling ratio, OSR , is defined to be the ratio of the sampling frequency f_s to the Nyquist frequency $2f_0$.

$$OSR = \frac{f_s}{2f_0} \quad (3.4)$$

For decoding, decimation is required. From Fig. 3.5, the corresponding multi-bit digital format $\hat{u}$ of the input u is determined by the following equation:

$$\hat{u} = \frac{\Delta}{OSR} \sum_{i=1}^{OSR} y(x_n)_i \quad (3.5)$$

where x_n is the integrator's output and $y(x_n)$ is the output of the 1-bit quantiser. If x_n is positive or 0, $y(x_n)$ is $+\Delta$. If x_n is negative, then $y(x_n)$ is $-\Delta$. From the above equation, two conclusions can be drawn:

- The maximum value after decoding is $+\Delta$, and the minimum is $-\Delta$. Therefore, the input should be limited between $-\Delta$ and $+\Delta$. If the

3.1. DIGITAL MODULATION

input exceeds this limit, scaling has to be applied.

- The oversampling ratio, OSR , has to be large in order to obtain a high precision $\hat{u}$, requiring a very fast sampling rate.

Figure 3.5: $\Delta\Sigma$ modulation.

3.1.3 Conclusion

Both Δ modulation and $\Delta\Sigma$ modulation encode an input signal into binary pulses which can be represented by 1-bit format in digital logic, being 1-bit signals. These 1-bit signals contain all the useful information of the input and can be recovered by a decoder. In practice, either the Δ modulator or the $\Delta\Sigma$ modulator itself acts as an analogue to digital converter, having an analogue input signal and a binary output signal.

The difference is that the Δ modulator encodes the signal's derivative rather than its amplitude. The exact relationship between the analogue input and the binary output is such that each binary pulse is directly proportional to the instantaneous slope of the input signal. If the slope of the input signal is positive then there are more positive pulses than negative ones, and vice versa. However, from Eq. 3.5, it is obvious that $\hat{u}$ is an average value of $y(x_n)$ over OSR samples. The relationship between the analogue input and the binary output is such that each binary pulse represents the corresponding amplitude on the original input with an error.

Because control laws are designed given a signal's amplitude, it is definitely not appropriate to work on the Δ modulated binary pulses. Although

3.2. WAVELET ANALYSIS

it is possible to place a decoder after the Δ modulator, it becomes 'conventional' again and this is not the objective of this thesis. As a result, it is only possible to consider control system processing on $\Delta\Sigma$ modulated 1-bit signals.

3.2 Wavelet analysis

The binary output of the $\Delta\Sigma$ modulator contains all the useful information of the input, but this information is obscured by quantisation noise. To explain this phenomenon in detail, wavelet de-noising rather than Fourier analysis is applied here. The Fourier analysis breaks down a signal into constituent sinusoids of different frequencies. However it has a serious drawback that time information is lost after transforming to the frequency domain. If the signal properties do not change much over time this drawback is not very important, which is not the case of the 1-bit signals. For 1-bit processing it is necessary to recover the useful information from the 1-bit signals and Fourier analysis is not suited to detecting them. The wavelet filter is able to divide a signal into several parts in different frequency domains. Therefore it is possible to separate the useful information of the 1-bit signals from the quantisation noise.

Let $(\Psi_{j,k})_{j,k} \in K$ be an orthogonal basis of wavelets on the interval $I = [a, b]$ as described by Cohen et al. (1993), so that any signal $u \in L^2(I)$ can be written as the sum of a series

$$u = \sum_{j,k \in K} \langle u, \Psi_{j,k} \rangle \Psi_{j,k} \quad (3.6)$$

where

$$\langle u, \Psi_{j,k} \rangle = \int_I u(x) \Psi_{j,k}(x) dx \quad (3.7)$$

Let the hard thresholding operator τ be defined as:

$$\tau(x) = \begin{cases} x & \text{if } |x| \geq \lambda \\ 0 & \text{if } |x| < \lambda \end{cases} \quad (3.8)$$

3.2. WAVELET ANALYSIS

In the case of soft thresholding, the operator τ is

$$\tau(x) = \begin{cases} x \operatorname{sgn}(x) \lambda & \text{if } |x| \geq \lambda \\ 0 & \text{if } |x| < \lambda \end{cases} \quad (3.9)$$

The de-noised signal using wavelet thresholding is simply

$$u_0 = \sum_{j,k \in K} \tau(\langle u, \Psi_{j,k} \rangle) \Psi_{j,k} \quad (3.10)$$

Hence, the noisy signal can be written

$$u = \tilde{u} + \sum_i w_i \quad (3.11)$$

where $\tilde{u}$ is the noiseless signal to be estimated, w_i is the additive Gaussian white noise of standard deviation σ_i , and i is the number of de-noising steps. The threshold λ is set to $\sigma_i \sqrt{2 \log M}$ and M is the number of samples of the digital signal. In this case the estimator is the best in the min-max sense as M tends to infinity (Donoho and Johnstone, 1994).

Figure 3.6: Second order $\Delta\Sigma$ modulator

Fig. 3.6 shows a second order $\Delta\Sigma$ modulator, and Fig. 3.7 illustrates a sine wave with a frequency of 1Hz. The sampling frequency of the $\Delta\Sigma$ modulator is 100Hz. Hence, a time-series data is obtained after 10 seconds' simulation, obtaining 1000 1-bit samples of the input sine wave. Fig. 3.8 shows the results after 4-step wavelet de-noising. r is the resulting 1-bit signal; a_4 is the de-noised signal; d_1 to d_4 are high-frequency quantization noises at different steps. There is a phase delay in the de-noised signal because the wavelet

3.2. WAVELET ANALYSIS

Figure 3.7: 1Hz sine wave input

filter has a time delay.

Figure 3.8: Results of wavelet de-noising

Definitely, 1-bit signal is a rough representation of the input, and it can

Figure 3.9: First order linear $\Delta\Sigma$ modulator.

be represented as:

$$r = a4 + d4 + d3 + d2 + d1 \quad (3.12)$$

As the 1-bit quantizer is a nonlinear component, This equation indicates that the $\Delta\Sigma$ modulator can be linearized by an additive quantization error, making it easier to design and analyse 1-bit systems.

3.3 Quantization noise

3.3.1 Noise shaping

The $\Delta\Sigma$ modulation also adds noise-shaping benefits. Fig. 3.5 shows a first order (single integration) $\Delta\Sigma$ modulation encoder. An integrator is placed in the main loop before the quantizer. Its linearized model is shown in Fig. 3.9.

The input to the quantizer is the integral of the difference between the input and the quantized output. The difference between the input signal and the output signal approaches zero. Hence the average value of the binary pulses tracks the input. The relationship between the input U , the quantization noise e and the output Y can be described by

$$Y = \frac{1}{s+1}U + \frac{s}{s+1}e \quad (3.13)$$

This equation contains a signal transfer function (STF) and a noise transfer function (NTF). The STF is a low-pass filter and the NTF is a high-pass filter. The integrator therefore forms a low-pass filter on the difference signal,

3.3. QUANTIZATION NOISE

Figure 3.10: Sampled $\Delta\Sigma$ modulation signal with a sampling rate of 64kHz

providing low frequency feedback around the quantizer. This feedback results in a reduction of quantization noise at low (in-band) frequencies. The noise, however, is shaped by a high-pass filter, shaping the noise out of the low frequency area. Hence $\Delta\Sigma$ modulation is also known as a noise-shaping filter. In practice, the in-band noise floor level is not satisfactory with first-order $\Delta\Sigma$ modulation (Norsworthy et al., 1997). Further noise shaping must be achieved with higher-order (multiple integration) $\Delta\Sigma$ modulation coders.

The same input signal as in the Δ modulation examples with amplitude 0.9 is taken. Fig. 3.10 shows the $\Delta\Sigma$ modulated signal with a sampling rate of 64kHz, and Fig. 3.11 shows its spectrum.

When the sampling rate is increased to 128kHz, the $\Delta\Sigma$ modulated signal is shown in Fig. 3.12, and its spectrum is shown in Fig. 3.13.

Both the previous spectra imply that the $\Delta\Sigma$ modulation decoder is a low-pass filter. They explain how the noise floor decreases as the sampling frequency increases. In Fig. 3.11 the minimum noise floor is at frequencies near 5kHz, and in Fig. 3.13 it is near 10kHz. Definitely with the sampling

3.3. QUANTIZATION NOISE

Figure 3.11: Spectrum of the previous $\Delta\Sigma$ signal of 64kHz

Figure 3.12: Sampled $\Delta\Sigma$ modulation signal with a sampling rate of 128kHz

3.3. QUANTIZATION NOISE

Figure 3.13: Spectrum of previous signal with a sampling rate of 128kHz

frequency increasing, the noise floor is pushed further away from the band of interest. This suggests that with higher-order noise-shaping and adequate sampling rate it is possible to process control laws directly on the 1-bit signals.

3.3.2 Noise in first order $\Delta\Sigma$ modulation

To analyze the quantization noise, the linear model of Fig. 3.9 is revised to a discrete model as shown in Fig. 3.14. The transfer function therefore is

$$Y = \frac{1}{z}U + \frac{z-1}{z}e \quad (3.14)$$

Assuming that the quantization error has equal probability of lying anywhere in the range $\pm\Delta$, its mean square value is given by

$$\sigma_e^2 = \frac{1}{2\Delta} \int_{-\Delta}^{\Delta} e^2 de = \frac{\Delta^2}{3} \quad (3.15)$$

where σ_e is the rms quantization error. When the sampling frequency is f_s ,

3.3. QUANTIZATION NOISE

Figure 3.14: First order discrete $\Delta\Sigma$ modulator.

all of the quantized signal power folds into the frequency band $0 \leq f < f_s/2$. Assuming that the quantization noise is white, its spectral density is obtained by

$$E(f) = \sigma_e \sqrt{\frac{2}{f_s}} = \sigma_e \sqrt{2T} \quad (3.16)$$

Given Eq. 3.14 and Eq. 3.16, the spectral of the quantization noise of the first order $\Delta\Sigma$ modulator is given by

$$N(f) = E(f) \left| \frac{z-1}{z} \right| \quad (3.17)$$

As $z = e^{j\omega T}$, where $\omega = 2\pi f$, the above equation is simplified by

$$N(f) = E(f) |1 - e^{-j\omega T}| = 2\sigma_e \sqrt{2T} \sin\left(\frac{\omega T}{2}\right) \quad (3.18)$$

Hence the noise power within the signal band is

$$\sigma_n^2 = \int_0^{f_0} |N(f)|^2 df \quad (3.19)$$

When the sampling frequency is much higher than the signal band, the total noise power is approximated by

$$\sigma_n^2 \approx \sigma_e^2 \frac{\pi^2}{3} (2f_0 T)^3 \quad (3.20)$$

3.3. QUANTIZATION NOISE

Figure 3.15: High order linear $\Delta\Sigma$ modulator.

and its rms value is

$$\sigma_n \approx \sigma_e \frac{\pi}{\sqrt{3}} \left(2 \frac{f_0}{f_s}\right)^{3/2} = \sigma_e \frac{\pi}{\sqrt{3}} (OSR)^{-3/2} \quad (3.21)$$

This shows that the noise is reduced 9dB by doubling the oversampling ratio, which again suggests that the quantization noise can be reduced and a signal-to-noise ratio (SNR) that is required for a dynamic system can be achieved, making 1-bit processing practical for real-time control.

3.3.3 Noise in high-order $\Delta\Sigma$ modulation

To achieve an acceptable quantization noise in the first order $\Delta\Sigma$ modulator, the sampling frequency has to be very high. Extremely high sampling frequency, however, is not expected in digital control because it results in very small coefficients which require long word length. To reduce the sampling frequency and maintain the quantization noise within the acceptable range, high-order $\Delta\Sigma$ modulation has to be considered.

A linear high order $\Delta\Sigma$ modulator with the additive quantization noise is shown in Fig. 3.15. Consider a second order one, its transfer function is given by

$$Y = \frac{1}{z} U + \frac{(z-1)^2}{z^2} e \quad (3.22)$$

Hence the noise spectrum is

$$N(f) = E(f) |(1 - e^{-j\omega T})^2| = 4\sigma_e \sqrt{2T} \sin^2\left(\frac{\omega T}{2}\right) \quad (3.23)$$

3.3. QUANTIZATION NOISE

and the rms noise in the signal band is approximated by

$$\sigma_n \approx \sigma_e \frac{\pi^2}{\sqrt{5}} OSR^{-5/2} \quad (3.24)$$

A general rms noise for the high order $\Delta\Sigma$ modulator is given by

$$\sigma_n = \sigma_e \frac{\pi^m}{\sqrt{2m+1}} OSR^{-(2m+1)/2} \quad (3.25)$$

where m is the order of the modulator. The rms noise therefore is reduced $3(2m+1)$ dB by doubling the oversampling ratio.

Utilizing a high order $\Delta\Sigma$ modulator can achieve a good noise performance, but it is also expensive to implement it with too many integrators in circuit. In practice the fourth order $\Delta\Sigma$ modulator is a most common application in data conversion. But the second order $\Delta\Sigma$ modulator such as the ADS1201 from Burr-brown Corp. (1997) is also common. Most of the work therefore is based on the second order $\Delta\Sigma$ modulator in order to reduce the circuit complexity. From Eq. 3.24, doubling the oversampling ratio reduces the rms noise by 15dB, resulting in a high dynamic range for signal processing.

3.3.4 Stability issue

To analyze the stability of the second order $\Delta\Sigma$ modulator, a gain is put into the main loop of the linear model as shown in Fig. 3.16. The transfer function therefore is

$$Y = \frac{k}{z + (k-1)} U + \frac{(z-1)^2}{z^2 + (k-1)z} \quad (3.26)$$

The Routh-Hurwitz criterion is used to analyze the stability. As this technique is not applicable in the z -domain, it has to be transformed into the imaginary axis of the ω -plane using

$$z = \frac{\omega + 1}{\omega - 1} \quad (3.27)$$

3.4. REALIZING THE $\Delta\Sigma$ MODULATOR

Figure 3.16: Second order linear $\Delta\Sigma$ modulator with a gain k .

Hence the roots for the characteristic equation of the STF are calculated by

$$\frac{\omega + 1}{\omega - 1} + (k - 1) = 0 \quad (3.28)$$

and the roots for the characteristic equation of the NTF are

$$\left(\frac{\omega + 1}{\omega - 1}\right)^2 + (k - 1)\frac{\omega + 1}{\omega - 1} = 0 \quad (3.29)$$

Hence,

$$k \in (0, 2) \quad (3.30)$$

to maintain the modulator stable. In most applications, k usually is 1, and the stability of the second order $\Delta\Sigma$ modulator is guaranteed.

3.4 Realizing the $\Delta\Sigma$ modulator

A $\Delta\Sigma$ modulator is easy to implement in digital format, but the realization of such a digital modulator depends on the digital circuit design of one-bit processing and it will be explained in the following chapters. For the $\Delta\Sigma$ modulator that is used in analogue-to-digital conversion, however, there are two approaches to implement: switched-capacitor circuits and active-RC circuits, between which a design needs to be chosen at the first stage.

In general, most $\Delta\Sigma$ modulators use switched-capacitor (SC) circuits for integrated circuit implementations (Candy, 1992). Fig. 3.17 shows a simple SC circuit of the first order $\Delta\Sigma$ modulator. This circuit uses a two-phase clock — ϕ_1 and ϕ_2 — to realize the time delay. There is half a clock cycle

3.4. REALIZING THE $\Delta\Sigma$ MODULATOR

difference between ϕ_1 and ϕ_2 .

At the first clock cycle, the switches associated with ϕ_1 are connected and the other switches are disconnected. The voltages over the capacitors C_1 and C_2 therefore are obtained by:

$$\begin{aligned}\phi_1 : \quad V_{C_1}((n-1)T) &= V_{in}((n-1)T) \\ V_{C_2}((n-1)T) &= V_{out}((n-1)T)\end{aligned}\tag{3.31}$$

where T is the sampling period. On the phase ϕ_2 , the switches associated with ϕ_2 are connected and the other switches are disconnected. The output voltage is obtained by

$$\phi_2 : \quad V_{out}((n - \frac{1}{2})T) = \frac{C_1}{C_2} V_{in}((n-1)T) + V_{out}((n-1)T)\tag{3.32}$$

When it goes to the second clock cycle, because the switches associated with ϕ_2 are disconnected on ϕ_1 , the output voltage will not change the value, being

$$\begin{aligned}V_{out}(nT) &= V_{out}((n - \frac{1}{2})T) \\ &= \frac{C_1}{C_2} V_{in}((n-1)T) + V_{out}((n-1)T)\end{aligned}\tag{3.33}$$

This equation can be written as a z -transfer function, describing the relationship between the input V_{in} and the output V_{out} :

$$\frac{V_{out}}{V_{in}} = \frac{C_1}{C_2} \frac{z^{-1}}{1 - z^{-1}}\tag{3.34}$$

which is an integrator if C_1 and C_2 have the same value.

For the process of $\Delta\Sigma$ modulation, the input is sampled and the quantizer gives the output on ϕ_1 . The quantizer gives either a high or a low voltage, which is latched. The latch therefore gives '1' for a high voltage and '0' for a low voltage. When it is a high state, the capacitor C_f is charged by $C_f V_{ref}$. Whereas a low state results in a charge of $-C_f V_{ref}$. On ϕ_2 , the input is compared to the voltage in C_f . The SC-based design is therefore a natural description in discrete-time and compatible with VLSI CMOS process.

The other approach is to use a conventional active-RC design as shown in

3.5. SUMMARY

Figure 3.17: The first order $\Delta\Sigma$ modulator based on SC circuit.

Figure 3.18: The first order $\Delta\Sigma$ modulator based on RC circuit.

Fig. 3.18. The output of the quantizer is latched on every clock cycle. The latch gives '1' or '0', used to decide a reference voltage. The reference voltage is applied to a resistor which is connected to the integrator summing point. This voltage is continuously applied to the integrator during the entire clock period. The RC-based design is a description in continuous-time. It is used in most system-level or hybrid implementations (Benabes et al., 2000; Huang and Cheng, 2000).

3.5 Summary

There are two types of 1-bit data conversion, but Δ modulation is not practical because it gives a representation of the signal's slope rather than its amplitude. $\Delta\Sigma$ modulation therefore is the only choice for 1-bit processing.

The $\Delta\Sigma$ modulator is a nonlinear system. Its nonlinear behavior can be

3.5. SUMMARY

modelled by an additive quantization noise. A control system performance is affected by this quantization noise, and so it is required that the noise spectrum in the signal band is as little as possible. Fortunately, the $\Delta\Sigma$ modulator acts as a noise shaping filter, the rms noise being weakened in the low frequency range. It shows that the rms noise falls when the oversampling ratio increases. The number of integrators in the loop also decides the ability of noise shaping, but it is the second order $\Delta\Sigma$ modulator that is widely used in many applications because more than two integrators in the loop are expensive to realize in circuit. The noise in the second order $\Delta\Sigma$ modulator can be further reduced by placing a small gain in the loop.

The $\Delta\Sigma$ modulator is used in one-bit processing for two purposes — analogue-to-one-bit conversion and multi-bit-to-one-bit conversion. Both conversions will be achieved by a second order $\Delta\Sigma$ modulator. There are two approaches, SC-based and RC-based design, to realize a $\Delta\Sigma$ modulator in hardware. The choice depends on whether the application needs an integrated circuit implementation or a system-level implementation. In one-bit processing, however, it doesn't require a real $\Delta\Sigma$ modulator at the current stage. In stead, a program that simulates the behaviour of a second order $\Delta\Sigma$ modulator will be used in order to verify the design of one-bit processing.

Chapter 4

One-bit Processing

4.1 One-bit processing

One-bit processing is a new concept in digital control, its definition being '*One-bit processing encodes signals into binary pulses and represents these pulses with 1-bit registers in hardware; then it works on these 1-bit data directly to produce desired actuation in real-time.*' Further more the actuation signals can be encoded into binary pulses again to drive physical systems. Although this conversion is not necessary, it is found to be an effective approach to implement 1-bit processing in hardware. Hence this approach is taken in the thesis. The complete definition therefore is '*One-bit processing encodes signals into binary pulses and represents these pulses with 1-bit registers in hardware; then it works on these 1-bit data directly to produce desired actuation in real-time; finally it encodes the actuation into binary pulses to drive physical systems*'.

One-bit processing is compared to the conventional digital control system in Fig. 4.1. The major difference is that 1-bit processing utilizes a simple $\Delta\Sigma$ modulator to convert analogue to digital signals in 1-bit format other than a multi-bit A/D converter in the conventional digital control system. The conventional digital control system uses a D/A converter to convert the actuation signal into analogue, being able to drive the physical system. Sometimes, it also uses PWM logic to convert the actuation signal into a

4.2. DISCRETE TRANSFORMS

Figure 4.1: (a) Conventional control system. (b) One-bit processing.

series of pulses which can drive the physical system more efficiently (Gitau, 1994; Holmes and Lipo, 2002; Wu, 1997). In one-bit processing, however, a digital $\Delta\Sigma$ modulator is used to encode the actuation into binary pulses before the actuation signal acts on the physical system. This approach works similarly to PWM, but here it should be called pulse-density-modulation (PDM) more properly due to the characteristic of the $\Delta\Sigma$ modulation.

One issue common to both of the control systems is that the control law can be designed using the same approaches, i.e. either the classic control theory or the modern control theory. As it is discussed before, one-bit processing is a way to implement rather than to design control laws. The particular interests therefore are shown on the controller formulation and sampling criterion in one-bit processing.

4.2 Discrete transforms

Scavone (2004) provides many approaches to represent a continuous system in discrete-time, including the backward finite difference approximation, the centered finite difference approximation, the Adams-Moulton method, the weighted sample method, the Runge-Kutta method, the Euler method and etc.

However, the classical methods for analyzing and designing control systems are characterized by the transform techniques and transfer functions,

4.2. DISCRETE TRANSFORMS

whereas modern control theory is characterized by state variables and state-space equations. The Laplace transform is a basic tool in the analysis and design of continuous control systems, but the analysis of digital control systems relies on discrete transform techniques, including the well-known z -transform and the δ -transform (Forsythe and Goodall, 1991; Goodwin et al., 2001; Middleton and Goodwin, 1990). The z -transform is associated with the shift operator q and the complex variable z . The δ -transform is associated with the δ -operator and the complex variable γ . In many practical applications, however, the shift operator q is replaced by the z -transform variable z in going from the difference equation form to the z -domain form of the equation. The symbols 'q' and 'z' therefore are often used interchangeably (Middleton and Goodwin, 1990). Similarly, the symbols ' γ ' and ' δ ' are used interchangeably in the δ -transform.

4.2.1 The z -transform

The z -transform is widely used in digital control. Sometimes it is also called the sampled Laplace transform because its idea was derived from the Laplace transform. The z transfer function can be obtained through a transformation from the Laplace transform variable s to the z -transform variable z . There are many methods to achieve this transformation. An obvious choice is

$$s = \frac{1}{T} \ln z \quad (4.1)$$

where T is the sampling interval.

Assuming that $f(t)$ is a continuous function. $F^*(s)$ is the Laplace transform of the sampled $f(t)$, being

$$F^*(s) = \sum_{k=0}^{\infty} f(kT) e^{-kTs} \quad (4.2)$$

$F(z)$ is defined as the z -transform of $f(t)$. $F(z)$ can be obtained by

4.2. DISCRETE TRANSFORMS

replacing e^{Ts} in Eq. 4.2 by z :

$$F(z) = \sum_{k=0}^{\infty} f(kT)z^{-k} \quad (4.3)$$

This equation represents a simple sequential nature of the sampled signals.

The other well established technique is the bilinear transform, in which the transformation from s to z can be accomplished by

$$s = \frac{2}{T} \frac{z - 1}{z + 1} \quad (4.4)$$

Other techniques include the Schneider transform (Schneider et al., 1991) which provides more accurate, higher-order representations. However, when sampling rates are relatively high the bilinear transform (which is also the second order Schneider transform) is very accurate and commonly used by control engineers. Hence the bilinear transform will be used throughout this thesis.

Consider a Laplace transfer function

$$H(s) = \frac{1}{a_1 s^2 + a_2 s + 1} \quad (4.5)$$

where a_1 and a_2 are the coefficients. Using the bilinear transform and a little algebraic manipulation, it is possible to derive a z -transfer function

$$H(z) = c_0 \frac{z^2 + 2z + 1}{z^2 + d_1 z + d_2} \quad (4.6)$$

where

$$\begin{aligned} c_0 &= \frac{T^2}{T^2 + 2a_2 T + 4a_1}, \\ d_1 &= \frac{2T^2 - 8a_1}{T^2 + 2a_2 T + 4a_1}, \quad \text{and} \\ d_2 &= \frac{T^2 - 2a_2 T + 4a_1}{T^2 + 2a_2 T + 4a_1} \end{aligned}$$

4.2. DISCRETE TRANSFORMS

From the implementation viewpoint, the z operator is defined by

$$\begin{aligned}zx(k) &= x(k+1) \quad (\text{forward}) \\z^{-1}x(k) &= x(k-1) \quad (\text{backward})\end{aligned}\tag{4.7}$$

The shift operator is widely used to describe discrete time systems, but its disadvantage is that it does not resemble the continuous time operator d/dt at all.

4.2.2 The δ -transform

The continuous time operator d/dt is defined by

$$\begin{aligned}\frac{d}{dt}x &= \frac{x(k+1)-x(k)}{T} \quad (\text{forward}) \\&= \frac{x(k)-x(k-1)}{T} \quad (\text{backward})\end{aligned}\tag{4.8}$$

Here T is a small time difference. Eq. 4.8 is also known as the forward finite difference approximation and the backward finite difference approximation. This approximation becomes more precise when T approaches to 0. A better correspondence therefore is obtained between the continuous and discrete time if a δ -operator is used, which is more like a derivative.

Given by Eq. 4.7, Eq. 4.8 can be re-organized by

$$\begin{aligned}\frac{d}{dt}x &= \frac{z-1}{T}x \quad (\text{forward}) \\&= \frac{1-z^{-1}}{T}x \quad (\text{backward})\end{aligned}\tag{4.9}$$

Middleton and Goodwin (1990) defines the δ -operator as the following forward difference:

$$\delta = \frac{z-1}{T}\tag{4.10}$$

Because T is just a scaling factor in the control loop, which only changes coefficients' values, Forsythe and Goodall (1991) offers the definition:

$$\delta = z - 1\tag{4.11}$$

4.2. DISCRETE TRANSFORMS

The δ -operator shows that there is a unification between the discrete and continuous time.

The δ -transfer function can be obtained through a transformation from the Laplace transform variable s to the δ -transform variable δ . Two approaches are available to achieve this transformation. One is to transform the Laplace transfer function to the z -transfer function first, using any transform technique. Then the δ -transfer function can be obtained by replacing z with $\delta + 1$. Consider the Laplace transfer function in Eq. 4.5, and Eq. 4.6 is its z -transfer function. The δ -transfer function is obtained by

$$H(\delta) = c_0 \frac{\delta^2 + 4\delta + 4}{\delta^2 + p_1\delta + p_2} \quad (4.12)$$

where

$$\begin{aligned} p_1 &= d_1 + 2 \\ &= \frac{4T^2 + 4aT}{T^2 + 2aT + 4b}, \quad \text{and} \\ p_2 &= d_1 + d_2 + 1 \\ &= \frac{4T^2}{T^2 + 2aT + 4b} \end{aligned}$$

The other approach is that the δ -transfer function can be derived directly by the Laplace transfer function. Because the continuous time operator d/dt resembles the Laplace operator s , from Eq. 4.9 and Eq. 4.11, the δ -operator can be defined by

$$\delta = sT \quad (4.13)$$

The δ -transfer function therefore can be obtained by replacing s with $\frac{\delta}{T}$. Hence the δ -transfer function of Eq. 4.5 is

$$H(\delta) = \frac{1}{q_1\delta^2 + q_2\delta + 1} \quad (4.14)$$

where

$$\begin{aligned} q_1 &= \frac{a_1}{T^2}, \quad \text{and} \\ q_2 &= \frac{a_2}{T} \end{aligned}$$

4.2.3 δ -operator vs. z -operator

The choice of the z -operator is natural for many control engineers. The design and analysis techniques are well established in the z -domain, but as Goodall (1990); Liu (1971) pointed out it has a lot of problems with very high sampling frequencies. For example, Eq. 4.6 gives two poles near $z = 1$ when T is very small, d_1 and d_2 tending to -2 and 1 respectively. It is well known that the poles near/on the unit circle are crucial to the system stability, resulting in the high coefficient sensitivity. However, in the δ -transfer function the coefficients (p_1 and p_2) tend to 0 when T becomes small, resulting in the low coefficient sensitivity.

Using the δ -operator shows great advantages over the z -operator in terms of controller implementation. With the z -transfer function, the high sampling frequency results in a very long word-length to represent the coefficients and the state variables because the differences between successive values of the input and output become increasingly small. Because of the low coefficient sensitivity in the δ -transfer function, however, the accuracy of the coefficients simply needs to have the same accuracy as is required for the overall system performance (typically 5% for control) (Forsythe and Goodall, 1991).

High sampling frequencies are unavoidable in one-bit processing, making it necessary to choose the δ -operator to implement control laws. The z -transfer function, however, is still useful for the system being analyzed in z -domain because the relationship between the δ -operator and the z -operator is a simple linear function, and thus the δ -operator has the same flexibility in the modelling of discrete time systems as the z -operator.

4.3 The state-space approach

4.3.1 State-space equations

Digital control is an approach to realize a control system in real-life. The modern approach, which utilizes the state-space equations to represent the control system (Hu, 1994; Nise, 1995; Vaccaro, 1995), is therefore a more

4.3. THE STATE-SPACE APPROACH

convenient way of describing the control system than the traditional approach. The state-space equations provide not only the relationship between the input and the output but also state variables. The state variables are information that describes the internal mechanism of the control system, and the state-space equations describe how the state variables are related to each other and to the input.

The state-space equations have two forms: continuous and discrete. The continuous state-space equations are obtained by

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}\end{aligned}\tag{4.15}$$

for a multi-in-and-multi-out control system, where $\mathbf{x}$ is a n -dimension state vector, $\mathbf{u}$ is a $p \times 1$ -dimension input vector, $\mathbf{y}$ is a $q \times 1$ -dimension output vector, $\mathbf{A}$ is a $n \times n$ matrix, $\mathbf{B}$ is a $n \times p$ matrix, $\mathbf{C}$ is a $q \times n$ matrix and $\mathbf{D}$ is a $q \times p$ matrix. $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are real coefficients, describing the dynamic characteristic of the controller.

The state-space equations offer a number of advantages (Santina et al., 1994), but they are more straightforward and powerful from the implementation viewpoint in digital control. The discrete state-space equations are described by

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k\end{aligned}\tag{4.16}$$

where k represents the k^{th} sample. It is obvious that this form is easy to implement by writing a program with the calculations starting from a set of initial states (normally 0).

The choice of the states is not unique. The number of states are also variable. The importance is that any form of the state space equations, by defining a new set of states, can generate the same response. This flexibility can be exploited to optimize the numerical performance for real-time calculations.

4.3.2 Controller structures

The transfer function, either z or δ , can be illustrated by a particular structure, which is easily to be written in the state space equations. A controller structure is called the z -form if it uses the z operator and the δ -form if it uses the δ -operator.

Consider Eq. 4.6, it can be implemented in a canonic z -form as shown in Fig. 4.2. The discrete state space equations are:

$$\begin{aligned} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0 & -d_1 & -d_2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u \\ y &= c_0 \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \end{aligned} \quad (4.17)$$

where x_0 , x_1 and x_2 are state variables. From Eq. 4.17, one of states x_0 is expanded:

$$\begin{aligned} x_0(k) &= u(k) - d_1 x_1(k) - d_2 x_2(k) \\ &= u(k) - d_1 x_0(k-1) - d_2 x_0(k-2) \end{aligned} \quad (4.18)$$

It shows that the difference between the successive values of x_0 is very small when the sampling frequency is relatively high compared to the controller bandwidth. The coefficients d_1 and d_2 are determined in order that the suitable proportions of the small differences are combined to give the required output y (Goodall, 1990). Any small change of d_1 and d_2 will result in a much larger change in the output y , and this becomes increasingly a problem as the order of the system increases. Thus, the z -form is not well suited for real-time control.

Like the state-space equations, the z -form for a control system is also not unique. Fig. 4.3 is another z -form of Eq. 4.6 and its state space equations

4.3. THE STATE-SPACE APPROACH

Figure 4.2: The canonic z -form.

are:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 \\ p_1 & 0 & 0 \\ 0 & p_2 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u \quad (4.19)$$

$$y = \begin{bmatrix} q_0 & q_1 & q_2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

where

$$\begin{aligned} p_1 &= d_1, \\ p_2 &= \frac{d_2}{d_1}, \\ q_0 &= c_0, \\ q_1 &= \frac{2c_0}{d_1}, \text{ and} \\ q_2 &= \frac{c_0}{d_2} \end{aligned}$$

Eq. 4.17 and Eq. 4.19 together with Fig. 4.2 and Fig. 4.3 provide exactly the same relationship between the output y and the input u , although the set of the controller states and the coefficients are quite different in each case. Essentially the choice of controller structure (either the z or δ -form) depends on what to achieve, for example to minimise the number of instructions that are needed for control calculations in Jones et al. (1998); Cumplido-Parra (2001).

Figure 4.3: The modified canonic z-form.

4.4 The δ -form in 1-bit processing

4.4.1 The δ -form

The general description of Eq. 4.16 can implement any formulation with the particular set of coefficients, resulting in an identifiable structure. However, it is not always correct for any controller structure. Consider for example a generalized single-input single-output controller of second order. Its transfer function can be represented by

$$H(\delta) = \frac{n_0 + n_1\delta^{-1} + n_2\delta^{-2}}{1 + m_1\delta^{-1} + m_2\delta^{-2}}. \quad (4.20)$$

Fig. 4.4 shows two controller structures, both using the canonic δ -form. (a) is a conventional canonic δ -form (Cumplido-Parra, 2001). The corresponding state space equations for this form are

$$\begin{aligned} \mathbf{x}_{k+1} &= \begin{bmatrix} 1 - q_0 & -q_1 \\ 1 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k \\ y_k &= [p_1 - r_1 p_0 \quad p_2 - r_2 p_0] \mathbf{x}_k + p_0 u_k \end{aligned} \quad (4.21)$$

4.4. THE δ -FORM IN 1-BIT PROCESSING

where

$$\begin{aligned} p_0 &= n_0, \\ p_1 &= n_1, \\ p_2 &= n_2, \\ q_0 &= m_1, \text{ and} \\ q_1 &= m_2 \end{aligned}$$

(b) is a modified canonic δ -form. Its corresponding state space equations for next-state calculations are

$$\begin{aligned} \mathbf{x}_{k+1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} p'_0 \\ p'_1 \end{bmatrix} u_k - \begin{bmatrix} q'_0 \\ q'_1 \end{bmatrix} y_k \\ y_k &= [0 \ 1] \mathbf{x}_k + p'_2 u_k \end{aligned} \quad (4.22)$$

where

$$\begin{aligned} p'_0 &= n_2, \\ p'_1 &= n_1, \\ p'_2 &= n_0, \\ q'_0 &= m_2, \text{ and} \\ q'_1 &= m_1 \end{aligned}$$

Obviously, (a) and (b) describe exactly the same δ -transfer function as shown in Eq. 4.20, except that the set of states and the coefficients are different. The state space equations of (b) add an extra feedback of the output y when calculating the next states.

Cumplido-Parra (2001) uses the other modified canonic δ -form as shown in Fig. 1.3 for a control system processor. The selected form results in a very efficient processor architecture with optimal numeric calculations. In 1-bit processing, the choice of the controller form therefore needs to be considered carefully in terms of numeric calculations and hardware complexity.

 Figure 4.4: (a) The canonic δ -form. (b) The modified canonic δ -form.

4.4.2 $\Delta\Sigma$ modulated δ -form

As shown in Fig. 4.1(b), a $\Delta\Sigma$ modulator is placed after the controller, producing PDM signals to drive the physical system. However, the place of the $\Delta\Sigma$ modulator is a part of the design art. Consider a general description of the second order $\Delta\Sigma$ modulator with a gain in the main loop as show in Fig. 4.5. The linear model has been shown in Fig. 3.16. Eq. 3.26 describes the equivalent transfer function in z -domain, in which the STF is

$$STF = \frac{k}{z + (k - 1)} \quad (4.23)$$

Assuming that the input has an rms value, σ_u , the rms output, σ_y is obtained by

$$\begin{aligned} \sigma_y &= \sigma_u |STF| + \sigma_n \\ &= \sigma_u \left| \frac{k}{e^{j\omega T} + (k-1)} \right| + \sigma_n \\ &= \sigma_u \frac{k}{\sqrt{1 + 2(k-1)\cos(\omega T) + (k-1)^2}} + \sigma_n \end{aligned} \quad (4.24)$$

4.4. THE δ -FORM IN 1-BIT PROCESSING

Figure 4.5: The second order $\Delta\Sigma$ modulator

where σ_n is the rms quantization noise. The σ_n has been analyzed in Chapter 3, showing that it is very small within the signal bandwidth of interest. When the sampling frequency is high, σ_y therefore approximates

$$\sigma_y \approx \sigma_u \quad (4.25)$$

The above analysis shows that the output of the $\Delta\Sigma$ modulator is equivalent to the input by ignoring the effect of the quantization noise within the signal bandwidth.

Theoretically, the control system can contain any number of $\Delta\Sigma$ modulators anywhere in the loop, but the practical choice of the quantity and position depends on the objectives of the application. It has been declared in Chapter 1 that the proposed research aims to remove multi-bit multipliers in control law calculations. The multi-bit multiplier is a deterministic factor in integrated circuit design and is unavoidable in the conventional digital control as shown in Fig. 4.1(a).

Consider the modified δ -form by Cumplido-Parra (2001). Fig. 4.6 shows the same δ -form but integrated with a $\Delta\Sigma$ modulator after each integrator (δ -operator) and each summing junction. The thick lines represent multi-bit signals and the thin lines represent 1-bit signals in the figure. The input u is a 1-bit signal sampled by a $\Delta\Sigma$ modulator.

For the δ -operator, the equation $y = \delta^{-1}x$ is implemented by

$$y(n+1) = x(n) + y(n) \quad (4.26)$$

which is only an addition. For a $\Delta\Sigma$ modulator like that in Fig. 4.5, no multiplications are needed when $k = 1$. If necessary, k can be chosen as a

4.4. THE δ -FORM IN 1-BIT PROCESSING

Figure 4.6: The modified δ -form with multiple $\Delta\Sigma$ modulators.

Figure 4.7: The modified δ -form with single $\Delta\Sigma$ modulator.

value of 2's power, resulting in a shift operation. Thus, the possible multi-bit multiplication occurs between a 1-bit signal given by the $\Delta\Sigma$ modulator and a multi-bit coefficient. However, strictly speaking this is not a multiplier any more since it just changes the sign bit of the coefficient: when it is 1, the result is the coefficient itself; and when it is -1, the result negates the coefficient. It is therefore more proper to call this operation a 'conditional negation'. This structure is applicable for real-time control, but it is not a good choice for high order control systems because there are too many $\Delta\Sigma$ modulators in the loop. These modulators will introduce extra circuit complexity into the IC design. It is also more difficult to analyze the control system as these modulators bring many nonlinearities into the control loop, making it too complex. An ideal form should combine as few modulators as possible in the loop. Fig. 4.7 shows such a form based on the modified δ -form as shown in Fig. 4.4(b), in which only one $\Delta\Sigma$ modulator is contained in the loop. In this form, both the feed-forward and feedback signal are in 1-bit format. The multi-bit coefficients are therefore 'conditionally negated'. This form is much simple and accepted as a major structure in one-bit processing.

4.4. THE δ -FORM IN 1-BIT PROCESSING

Figure 4.8: The linearized $\Delta\Sigma$ modulated δ -form.

One-bit processing uses one-bit data at the input and control loop, but this is not a bit-serial approach. Bit-serial was proposed by Denyer and Renshaw (1985) for VLSI signal processing, in which serial operators are utilized. These serial operators process a multi-bit word from the low bit to the high bit in sequence to realize a multi-bit parallel operation such as multiplication, addition and etc. Thus, each bit-serial data has a correspondent position on a multi-bit data, and there is a value on itself:

$$y = x \times 2^n \quad (4.27)$$

where x is either 0 or 1 and n is its correspondent position (from 0). However, one-bit data are $\Delta\Sigma$ modulated pulses in principle, and there are still multi-bit parallel operations in one-bit processing such as shift and addition. Both approaches are effective in terms of circuit complexity: one bit processing achieves it by eliminating multi-bit multipliers and bit-serial processing achieves it through bit-serial operations. This issue will be further discussed in following chapters, but it is important to note here that they are different.

4.4.3 Stability analysis

An alternate viewpoint is to regard the $\Delta\Sigma$ modulated δ -form as a higher order $\Delta\Sigma$ modulator (Johns and Lewis, 1993). Hence, Fig. 4.7, which incorporates both the second order transfer function and the second order modulator, can be regarded as a 4th-order $\Delta\Sigma$ modulator. To analyze its stability, it is linearized with a gain k in the main loop as shown in Fig. 4.8.

4.4. THE δ -FORM IN 1-BIT PROCESSING

δ^{-1} is defined by

$$\delta^{-1} = \frac{1}{z-1} \quad (4.28)$$

The overall signal transfer function (STF) is obtained by

$$STF = \frac{p_2 k z^2 + (p_1 k - 2p_2 k)z + (p_0 k - p_1 k + p_2 k)}{z^3 + (k-3)z^2 + (q_1 k - 2k + 3)z + (q_0 k - q_1 k + k - 1)} \quad (4.29)$$

and the overall noise transfer function (NTF) is

$$NTF = \frac{(z-1)^4}{z^4 + (k-3)z^3 + (q_1 k - 2k + 3)z^2 + (q_0 k - q_1 k + k - 1)z} \quad (4.30)$$

Both STF and NTF have the same characteristic equation which needs to be transformed into ω -plane using Eq. 3.27 and being

$$q_0 \omega^3 + (2q_1 k - 3q_0 k) \omega^2 + (4k - 4q_1 k + 3q_0 k) \omega + (8 - 4k + 2q_1 k - q_0 k) = 0 \quad (4.31)$$

Applying the Routh-Hurwitz criterion to Eq. 4.31, the following Routh tabulation is obtained

ω^3	$q_0 k$	$4k - 4q_1 k + 3q_0 k$
ω^2	$2q_1 k - 3q_0 k$	$8 - 4k + 2q_1 k - q_0 k$
ω^1	$\frac{8(q_1 - q_1^2 + 2q_0 q_1 + q_0 - q_0^2)k - 8q_0}{2q_1 - 3q_0}$	0
ω^0	$8 - 4k + 2q_1 k - q_0 k$	

Table 4.1: Routh tabulation

It is rather complex to analyze the stability of the $\Delta\Sigma$ modulated δ -form. It is even more complex if we take the other $\Delta\Sigma$ modulator that is used for A/D conversion into consideration as this will include one more quantizer. Hence, this structure contains two nonlinear components, resulting in multiple noise transfer functions and thus rigorous stability criteria may be even more difficult to find according to the Routh-Hurwitz criterion. Fortunately, Johns and Lewis (1993) did thousands of simulations, indicating that the stability of such systems is determined by the stability of the original ones excluding $\Delta\Sigma$ modulators from the loop. In controller implementations, the

$\Delta\Sigma$ modulators only change the signal-to-noise ratio and have no effect to the stability of the system as long as they are stable.

4.5 Sampling in 1-bit processing

Sampling discretizes the time, and quantization discretizes the amplitude. In $\Delta\Sigma$ data conversion, it takes a number of cycles of the clock in order to be able to average the 1-bit signals with any kind of precision. Hence, to obtain 12-bit precision for 1-bit processing (a typical figure for real time control), a “safe” criterion is for a sampling frequency 4096 times as fast as that required for multi-bit processing. Even worse, if it is a 16-bit precision, the sampling frequency would be 65,536 times higher. In this case, high speed digital devices are required as most of digital processors may be unable to complete all the instructions of a complex control law in such a short time. It also increases the cost. Fortunately, such a high frequency is unnecessary for real-time control.

4.5.1 Phase delay

It is well known that, with fixed frequency sampling, any signals with a frequency beyond the Nyquist frequency $f_s/2$ cannot be replicated. This certainly means that

$$f_s > 2f_0 \quad (4.32)$$

where f_s is the sampling frequency, and f_0 is the signal bandwidth of interest. But only in exceptional cases of extremely non-demanding control systems is this criterion relevant. In general sampling at such a low frequency will introduce a phase lag that is not satisfied for real-time control, and for all practical control applications the criterion can be ignored.

Digital controller implementation inevitably adds time/phase delay into a control loop when compared with an analogue implementation. For the control system, the phase margin is a critical robustness requirement in most practical systems. Normally a phase margin around $40^\circ - 45^\circ$ is expected, but for the more demanding electro-mechanical control systems it is often difficult

4.5. SAMPLING IN 1-BIT PROCESSING

to achieve this, and around 30° of phase margin is not uncommon. The extra phase delay introduced by digital implementation should not significantly degrade this phase margin.

A realistic criterion is that for the bandwidth f_0 of the system (which is more or less where the phase margin occurs), there should be no more than 5° of phase delay introduced by the discrete process.

Assume that sampling only involves a zero-order hold on the output. It can be represented by $1 - e^{sT_s/2}$. So the phase delay introduced is approximately

$$\phi_s = 360f_0\left(\frac{1}{2f_s}\right) \quad (4.33)$$

where ϕ_s is the phase delay introduced by sampling.

Taking the additional effect of computation time into consideration, the total phase delay becomes

$$\phi = \phi_s + \phi_c = 360f_0\left(\frac{1}{2f_s} + T_c\right) \quad (4.34)$$

where T_c is the computation time, ϕ_c is the phase delay introduced by T_c , and ϕ is the total phase delay.

This can be re-expressed in somewhat more practical way. Let R be the ratio of the sampling frequency to the required bandwidth frequency, and K be the proportion of the sample period taken up by the computation. Then:

$$R = \frac{f_s}{f_0} \quad (4.35)$$

$$K = \frac{T_c}{T_s} = f_s T_c \quad (4.36)$$

and

$$\phi = \frac{360(0.5 + K)}{R} \quad (4.37)$$

As the phase delay should be no more than 5° , the corresponding inter-relationship between R and K to meet this requirement is given by

$$R = 36 + 72K \quad (4.38)$$

4.5. SAMPLING IN 1-BIT PROCESSING

For example, consider a bandwidth $f_0 = 100\text{Hz}$: to satisfy the 5° criterion with different values of K yields the values tabulated in Table 4.2.

Table 4.2: Sampling and computation factors.

K	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
R	43.2	50.4	57.6	64.8	72	79.2	86.4	93.6	100.8
$f_s(\text{kHz})$	4.32	5.04	5.76	6.48	7.2	7.92	8.64	9.36	10.08
$T_c(\text{ms})$	0.023	0.04	0.052	0.062	0.069	0.076	0.081	0.085	0.089

Notice that these practical design criteria creates some slightly counter-intuitive consequences:

- It proposes substantially higher frequencies than the often quoted values of 10-30 times the bandwidth. A typical sampling rate for this criterion is expected to be 100 times the bandwidth.
- Doubling the computation factor from $K = 0.3$ to $K = 0.6$ only increases the actual computation time by 46%. This is because the sample frequency must increase to preserve 5° phase delay.
- Providing twice the computation time (e.g. to allow for a more complex control algorithm) from 0.04 to 0.081 ($K = 0.2$ to 0.7) requires a 75% increase in sampling frequency.

This criterion, however, only provides a minimum sampling requirement for 1-bit processing when ignoring the effect of nonlinearity or quantization noise caused by the 1-bit quantizer. Normally, 1-bit processing requires a much higher sampling frequency in order to achieve a low level noise power within the bandwidth of interest. This means that the phase delay introduced by one-bit processing will no longer be a key criterion.

4.5.2 Quantization noise

In Chapter 3, it has been proposed that the nonlinearity in $\Delta\Sigma$ modulation can be linearized by modelling the 1-bit quantizer with a quantization noise.

4.5. SAMPLING IN 1-BIT PROCESSING

In the application of this modelling technique to the field of nonlinear control, the error, due to nonlinearity, is usually neglected. This is based on the assumption that the error is filtered by the physical system after feedback and forms a negligible part of the input signal to the nonlinearity. In $\Delta\Sigma$ modulation, however, the nonlinearity introduces spectral components which cover a wide bandwidth, including the baseband (Ardalan and Paulos, 1987). In this case, the error, which represents the quantization noise, may not be filtered sufficiently by the physical system if the noise power, within the baseband, is too high. Furthermore, in many cases in nonlinear control, the output of the nonlinearity is the input to the physical system. Hence it is substantially filtered before feedback to the nonlinearity input. In contrast, the output of the nonlinear quantizer is the desired 1-bit signal, which is directly fed back and subtracted from the input signal. Hence, the quantization noise has a major impact on the system performance for one-bit processing in real-time control.

It is well known that physical systems are not sensitive to signals in high-frequency range. The $\Delta\Sigma$ modulator is a noise-shaping filter, high-pass for the quantization noise, but it cannot remove the quantization noise within the baseband completely. From Chapter 3, we know that the rms quantization noise is reduced when the sampling frequency increases. The signal-to-noise SNR is therefore increased because the rms input is almost unchanged. Thus, to obtain a sampling criterion for one-bit processing, the appropriate approach is to increase the SNR within the baseband by increasing the sampling frequency. Because the signal does not change according to the sampling frequency, the quantization noise is reduced to a level that can be filtered effectively by physical systems.

One-bit processing, regardless of the physical system, is divided into two parts: the $\Delta\Sigma$ modulator and the $\Delta\Sigma$ modulated δ -form. Thus, the signal-to-noise ratio at the output is calculated in two steps: First, calculate the in-band signal and noise power for the two parts respectively; Then, calculate SNR according to the relationship between the two parts.

In Chapter 3, the rms quantization error σ_e is assumed as a constant related to the quantization level only. In practice, the quantization behavior

4.5. SAMPLING IN 1-BIT PROCESSING

Figure 4.9: Quasi-linear model for 1-bit processing.

is more complex. Hence a new approach is taken here to obtain a more precise SNR . Both the $\Delta\Sigma$ modulator in Fig. 4.5 and the 1-bit processing structure in Fig. 4.7 can be linearized in the z -domain, as shown in Fig. 4.9, with the appropriate choice of the loop filters $H_1(z)$ and $H_2(z)$, and e is Gaussian white noise. Obviously, the 1-bit output Y is calculated by

$$Y = Y_u + e_n + e \quad (4.39)$$

where

$$Y_u = UH_1(z) \quad (4.40)$$

and

$$e_n = e \frac{-H_2(z)}{1 + H_2(z)} \quad (4.41)$$

Since Y switches between -1 and 1 , its power density is constant and equal to 1:

$$\sigma_y^2 = \sigma_{Y_u}^2 + \sigma_{e_n}^2 + \sigma_e^2 = 1 \quad (4.42)$$

where, σ_y^2 , $\sigma_{Y_u}^2$, $\sigma_{e_n}^2$ and σ_e^2 represent the power densities of Y , Y_u , e_n and e respectively. Because e is a Gaussian white noise, its power density is constant. So it is necessary to obtain σ_e^2 first.

Let the power density of the input U be σ_U^2 . For the part of 1-bit processing, as the input is 1-bit, σ_U^2 is equal to 1. For the other part, however, σ_U varies following the format of the input. Here, only steady input and sine input are considered. Assume that the steady input is a constant m_1 and the

4.5. SAMPLING IN 1-BIT PROCESSING

sine input has a maximum magnitude m_2 . Hence, σ_U^2 is

$$\sigma_U^2 = \begin{cases} m_1^2 & \text{if } U \text{ is steady} \\ \frac{m_2^2}{2} & \text{if } U \text{ is sine} \end{cases} \quad (4.43)$$

Then, from Eq. 4.40, σ_{Y_u} is calculated by

$$\sigma_{Y_u}^2 = \frac{\sigma_u^2}{2\pi} \int_{-\pi}^{\pi} |H_1(e^{j\omega})|^2 d\omega \quad (4.44)$$

And, from Eq. 4.41, $\sigma_{e_n}^2$ is calculated by

$$\sigma_{e_n}^2 = \frac{\sigma_e^2}{2\pi} \int_{-\pi}^{\pi} \left| \frac{H_2(e^{j\omega})}{1 + H_2(e^{j\omega})} \right|^2 d\omega \quad (4.45)$$

Now, solve Eq. 4.42 to Eq. 4.45, the power density of the modelled quantization noise can be derived

$$\sigma_e^2 = k(1 - \sigma_{Y_u}^2) \quad (4.46)$$

where

$$k = \frac{2\pi}{2\pi + \int_{-\pi}^{\pi} \left| \frac{H_2(e^{j\omega})}{1 + H_2(e^{j\omega})} \right|^2 d\omega} \quad (4.47)$$

Now the signal-to-noise ratio can be calculated given the input and σ_e^2 . Consider the noise transfer function, it can be written from the block diagram in Fig. 4.9 as

$$Y_n = e \frac{1}{1 + H_2(z)} \quad (4.48)$$

The power of the in-band signal and noise can be obtained from Eq. 4.40 and Eq. 4.48 by integrating over the baseband:

$$\begin{aligned} \sigma_{Y_u}^2 &= \int_0^{f_0} |Y_u|^2 df \\ &= \int_0^{f_0} |UH_1(e^{j2\pi fT})|^2 df \end{aligned} \quad (4.49)$$

and

$$\begin{aligned}\sigma_{Y_n}^2 &= \int_0^{f_0} |Y_n|^2 df \\ &= \int_0^{f_0} \left| e \frac{1}{1 + H_2(e^{j2\pi fT})} \right|^2 df\end{aligned}\tag{4.50}$$

where f_0 is the frequency within baseband and T is the sampling time.

As the power densities of U and e are constant, σ_U^2 and σ_e^2 , the in-band signal power becomes

$$\sigma_{Y_u}^2 = \frac{\sigma_U^2}{f_s} \int_0^{f_0} |H_1(e^{j2\pi fT})|^2 df\tag{4.51}$$

and the in-band noise power is

$$\sigma_{Y_n}^2 = \frac{\sigma_e^2}{f_s} \int_0^{f_0} \left| \frac{1}{1 + H_2(e^{j2\pi fT})} \right|^2 df\tag{4.52}$$

where $f_s = \frac{1}{T}$ is the sampling frequency.

The signal-to-noise ratio within the baseband can be defined as

$$\begin{aligned}SNR &= 10\log\left(\frac{\sigma_{Y_u}^2}{\sigma_{Y_n}^2}\right) \\ &= 20\log\left(\frac{\sigma_{Y_u}}{\sigma_{Y_n}}\right)\end{aligned}\tag{4.53}$$

Finally, as one bit processing is split into two parts, assume that the in-band signal and noise power are $\sigma_{Y_{u1}}$ and $\sigma_{Y_{n1}}$ for the 1-bit A/D converter, $\sigma_{Y_{u2}}$ and $\sigma_{Y_{n2}}$ for the controller structure. So the signal-to-noise ratio can be obtained by

$$SNR = 20\log\left(\frac{\sigma_{Y_{u1}}}{\sigma_{Y_{n2}}} \times \frac{\sigma_{Y_{u1}}}{\sigma_{Y_{u1}} + \sigma_{Y_{n1}}}\right).\tag{4.54}$$

Take the second order $\Delta\Sigma$ modulator as an example. Rearrange the diagram according to Fig. 4.9, $H_1(z)$ and $H_2(z)$ are

$$H_1(z) = \frac{1}{z},\tag{4.55}$$

4.5. SAMPLING IN 1-BIT PROCESSING

and

$$H_2(z) = \frac{2z - 1}{z^2 - 2z + 1}. \quad (4.56)$$

So according to the calculations from Eq. 4.39 to Eq. 4.51, the power of the in-band signal can be obtained as

$$\sigma_{Y_u}^2 = \frac{\sigma_U^2}{f_s} \int_0^{f_0} \left| \frac{1}{e^{j2\pi fT}} \right|^2 df \quad (4.57)$$

and the power of the in-band noise is

$$\sigma_{Y_n}^2 = \frac{\sigma_e^2}{f_s} \int_0^{f_0} \left| \frac{(e^{j2\pi fT} - 1)^2}{(e^{j2\pi fT})^2} \right|^2 df \quad (4.58)$$

Thus, from Eq. 4.53, the signal-to-noise ratio is calculated:

$$SNR = 10 \log \left(\frac{\sigma_U^2 \frac{f_0}{f_s}}{\sigma_e^2 (6 \frac{f_0}{f_s} - \frac{4}{\pi} \sin(2\pi \frac{f_0}{f_s}) + \frac{1}{2\pi} \sin(4\pi \frac{f_0}{f_s}))} \right) \quad (4.59)$$

where σ_e^2 can be obtained from Eq. 4.46. From Eq. 4.59, obviously, the signal-to-noise ratio is a function of the oversampling ratio OSR as

$$SNR = 10 \log \left(\frac{\sigma_U^2}{2OSR \times \sigma_e^2 (3 \frac{1}{OSR} - \frac{4}{\pi} \sin(\frac{\pi}{OSR}) + \frac{1}{2\pi} \sin(\frac{2\pi}{OSR}))} \right) \quad (4.60)$$

Consider a sinusoidal input with amplitude 1, and its power density is

$$\sigma_U^2 = \frac{1}{2} \quad (4.61)$$

So σ_e^2 is 0.11. Fig. 4.10 shows the relationship between the calculated signal-to-noise ratio and the oversampling ratio. Definitely, it is necessary to increase the sampling frequency f_s given a fixed controller bandwidth f_0 to obtain a high signal-to-noise ratio.

The idea of defining a baseband f_0 within which to calculate the power of the quantization noise is a simplification, and so in practice it is always important to analyze the physical system before sorting out a particular sampling rate for the digital controller. Although it is hard to obtain a general-

4.6. SIMULATION RESULTS

Figure 4.10: Calculated SNR with the sinusoidal input.

ized sampling criterion for 1-bit processing as physical systems have different abilities to filter noises, many practical experimentations show that a typical sampling rate for one-bit processing is at least 1000 times the bandwidth.

4.6 Simulation results

4.6.1 Validation example

A general-purpose single-input single-output filter is chosen to validate the concept of one-bit processing. The transfer function is

$$H(s) = \frac{1}{(1 + 1.4\frac{s}{\omega} + \frac{s^2}{\omega^2})^2} \quad (4.62)$$

where $\omega = 2\pi$. The baseband f_0 is defined as the frequency at which the gain of the closed-loop frequency response first falls below $-3dB$, and the validation example's baseband is $0.8Hz$.

Fig. 4.11 illustrates the filter structure using the $\Delta\Sigma$ modulated δ -form.

4.6. SIMULATION RESULTS

Figure 4.11: 4th order $\Delta\Sigma$ modulated δ -form

Given that the additive quantization noise affects the steady output no more than 5%, so the desired SNR is 26dB. According to the calculations in the previous chapter, this needs a sampling frequency at least 500Hz. Here we use 1000Hz to get a higher precision. Using Eq. 4.13, the coefficients are

$$\begin{aligned}
 p_0 &= 1.56 \times 10^{-9} \\
 p_1 &= p_2 = p_3 = p_4 = 0 \\
 q_0 &= 1.56 \times 10^{-9} \\
 q_1 &= 6.95 \times 10^{-7} \\
 q_2 &= 1.56 \times 10^{-4} \\
 q_3 &= 1.76 \times 10^{-2}
 \end{aligned} \tag{4.63}$$

It's not straightforward to analyze the output y as it is in 1-bit format. Instead of y , y' is studied because y approximates to y' according to Eq. 4.25.

Fig. 4.12 and Fig. 4.13 show the simulation results with an input of 0.9 and a 1Hz sine wave input respectively. Graph 4.12(a) and 4.13(a) are the responses with the continuous system, and graph (b)s are obtained with one-bit processing. These graphs show that one-bit processing introduces many small quantisation noises compared to the continuous responses. These noises are high-frequency, and can be filtered effectively if a low-pass filter is applied.

We also compare one-bit processing with the continuous system in the frequency domain. To obtain a practical frequency response, the frequency response analyzer TF2000 (Voltech Instruments Ltd, 1991) is used. This device has a Matlab interface developed by Cui (2004), making the analysis more convenient. To run the analysis, sweep parameters are set from 0.1Hz to 30Hz with an amplitude 1V. The $\Delta\Sigma$ modulated δ -form along with the $\Delta\Sigma$

4.6. SIMULATION RESULTS

Figure 4.12: Responses with $u = 0.9$

Figure 4.13: Responses with a 1Hz sine wave input

modulator (for 1-bit analogue-to-digital conversion) is programmed, running in a personal computer. The computer and TF2000 use a 12-bit A/D and D/A card to exchange data. Fig. 4.14 shows the Bode plot of the frequency response of the continuous system and one-bit processing respectively. It shows that there is only a small difference of 0.02 rad/s of the baseband between the continuous system and one-bit processing.

4.6. SIMULATION RESULTS

(a) Continuous.

(b) One-bit processing

Figure 4.14: Frequency responses of the 4th order filter

4.6.2 Practical DC motor control

Consider a practical 1-bit control system with a DC motor. The objective is to control the position of a rotating load with flexibility in the drive shaft.

4.6. SIMULATION RESULTS

Fig. 4.15 shows the diagram of a DC motor model. In this particular example the important variable that will be affected by high frequency noise is the motor current, and a signal-to-noise ratio can be specified no less than $27dB$, i.e. the effect of noise will be less than 5% of rated current to avoid significant loss of motor capability.

Figure 4.15: DC motor diagram.

A 4th order command-tracking controller has been designed including a PI, a phase advance and a notch filter to minimise the effect of the resonance caused by the flexibility of the physical system. The Laplace transfer function for the control system is

$$H(s) = \frac{0.0001s^4 + 0.001s^3 + 0.25s^2 + 0.2501s + 0.001}{0.0001s^4 + 0.011s^3 + 0.11s^2 + s} \quad (4.64)$$

Thus, the overall control scheme can be illustrated as Fig. 4.16. The controller bandwidth f_0 is about $0.75Hz$.

For 1-bit processing, the control law is represented by the modified canonic δ -form combined with the $\Delta\Sigma$ modulator as shown in Fig. 4.7. The full 1-bit control system also contains a 1-bit A/D converter in the loop. So the procedure of the control system processing can be described as follows. Firstly the analogue signals (command and motor position) are sampled by the 1-bit A/D converter, i.e. a second order $\Delta\Sigma$ modulator, and give a bitstream output. Then the signals feed into the digital controller and cause an update

Figure 4.16: The overall control scheme.

4.6. SIMULATION RESULTS

Figure 4.17: 1-bit control system in the modified δ -form.

of the state variables so they are ready for the next sample. As there exists a digital $\Delta\Sigma$ modulator in the loop, the control signal is in a 1-bit format, which can be directly output to drive the motor, i.e. pulse-density-modulation. The structural representation of the 1-bit control system is given by Fig. 4.17.

Two sampling criteria are obtained for 1-bit processing: $75Hz$ or above for the conventional bit-parallel control system processing and $300Hz$ or above for 1-bit processing. Consider a steady input 1, the relationship between the signal-to-noise ratio and sampling frequency is shown in Fig. 4.18. Although it is related to the sampling frequency, it also indicates the relationship between the signal-to-noise ratio and the oversampling ratio as the baseband f_0 is known. The sampling frequency therefore is at least $300Hz$ to meet the SNR requirement, which is $27dB$, for 1-bit processing. The coefficients, when the sampling frequency is $1000Hz$, are listed below.

$$\begin{aligned}
 p_0 &= 1.0 \times 10^{-11} \\
 p_1 &= 2.501 \times 10^{-6} \\
 p_2 &= 2.5004 \times 10^{-4} \\
 p_3 &= 1.0004 \times 10^{-2} \\
 p_4 &= 1 \\
 q_0 &= 0 \\
 q_1 &= 1.0 \times 10^{-5} \\
 q_2 &= 1.1 \times 10^{-3} \\
 q_3 &= 0.11
 \end{aligned} \tag{4.65}$$

In Fig. 4.19, (a) shows the simulation result of the step response of the

4.6. SIMULATION RESULTS

Figure 4.18: SNR and the sampling frequency given a controller bandwidth 0.75Hz.

1-bit control system and (b) is an expanded detail of the difference between the 1-bit control system and the continuous system, from which it can be seen that the difference is within 0.3%.

Figure 4.19: (a) Position response of the 1-bit control system; (b) Difference of the responses between the 1-bit system and the continuous system.

4.7. SUMMARY

It is also expected that the motor current difference between the continuous system and the 1-bit control system is no more than 5% of the maximum current which is 5A in this case. Fig. 4.20 (a) shows the motor current while running the 1-bit control system and (b) shows the difference compared to the continuous system. It is obvious that the motor works well with PDM control.

Figure 4.20: (a) Motor current; (b) The difference of the current between the continuous system and the 1-bit system

4.7 Summary

This chapter looks into the concept of one-bit processing and the definition is given at the beginning. z -transform is compared to δ -transform, showing that the z -transform has more numerical problems than the δ -transform when the sampling frequencies are high.

4.7. SUMMARY

The multi-bit multipliers are a determining factor in IC design. Hence, two $\Delta\Sigma$ modulated δ -forms are proposed, in which no multi-bit multipliers are needed.

In digital control, it is necessary to decide the sampling frequency first. For real-time control it shows a sampling frequency should be at least 100 times the baseband to achieve a small phase delay. However, one-bit processing is quite different from the conventional digital control, and a new sampling criterion therefore is introduced based on the signal-to-noise ratio. Two examples are also given with simulation results showing that one-bit processing is applicable for real-time control.

Chapter 5

Direct Implementation

One-bit processing can be implemented directly as an application-specific integrated circuit, in which all arithmetic operations are hardwired. This approach is called ‘direct implementation’.

5.1 Numerical issue

5.1.1 Coefficients

Consider for example a generalised single-input single-output controller of second order. Its transfer function can be represented by

$$H(s) = \frac{a_1 s^2 + a_2 s + a_3}{s^2 + b_1 s + b_2}. \quad (5.1)$$

The transfer function of the modified δ -form controller structure (not considering the $\Delta\Sigma$ modulator) in Fig. 4.7 can be written as:

$$\frac{Y}{U} = \frac{p_0 \delta^{-2} + p_1 \delta^{-1} + p_2}{q_0 \delta^{-2} + q_1 \delta^{-1} + 1}. \quad (5.2)$$

Figure 5.1: Re-modified canonic δ -form with scaling factors in the main loop.

From Eq. 4.14, Eq. 5.2 and Eq. 5.1, the coefficients are obtained:

$$\begin{aligned} p_0 &= a_3 T^2, \\ p_1 &= a_2 T, \\ p_2 &= a_1, \\ q_0 &= b_2 T^2, \\ q_1 &= b_1 T. \end{aligned} \tag{5.3}$$

As T is very small compared with the time constant of the transfer function, the coefficients become smaller when the controller order increases. This makes it difficult to represent such a small value in a fixed-point format. In order to scale these coefficients, the controller structure has to be modified as shown in Fig. 5.1. The transfer function for this structure (not considering the $\Delta\Sigma$ modulator) is

$$\frac{Y}{U} = \frac{p_0 k^2 \delta^{-2} + p_1 k \delta^{-1} + p_2}{q_0 k^2 \delta^{-2} + q_1 k \delta^{-1} + 1}. \quad (5.4)$$

Therefore, the coefficients become

$$\begin{aligned} p_0 &= a_3 T^2 k^{-2}, \\ p_1 &= a_2 T k^{-1}, \\ p_2 &= a_1, \\ q_0 &= b_2 T^2 k^{-2}, \\ q_1 &= b_1 T k^{-1}. \end{aligned} \tag{5.5}$$

The coefficients are enlarged via suitable scaling factors ($k \in (0, 1)$) in

the main loop. Choosing the value of the scaling factor k requires a careful process as it may involve multiplications which will increase the circuitry complexity. To avoid this, k can be a power of 2, implying only a simple shift operation.

5.1.2 Bit-width

The signal range requirements are usually modest in well-designed digital control algorithms with full IEEE 754 floating-point arithmetic (IEEE, 1985) being expensive in terms of power consumption and silicon complexity. Because no multipliers are needed in 1-bit processing, we adopt a fixed-point arithmetic format. The sampling frequency in 1-bit processing is usually very high, which results in a long word length for both coefficients and state variables. The bit-width therefore needs to be carefully chosen to ensure that the full value and dynamic range of the variables involved in the calculation can be accommodated.

Although it is possible to select quite large word lengths when the controller is implemented in VLSI, it is equally important to keep them to the absolute minimum as the selected word lengths will determine the size of the arithmetic blocks, which has a direct impact on the amount of hardware resources, maximum speed and power consumption.

A simple criterion used to determine the number of fractional bits is described in (Goodall and Brown, 1985). A reasonable number of fractional bits would be in the range of 8-16 bits, which will support a wide range of controllers. Fig. 5.2 shows a general format for the coefficients and state variables. This format accommodates a signal with an amplitude between -128 to 128, which is sufficient for most control applications with $\Delta\Sigma$ modulation considering that the input/output is only -1 or 1.

There are no overflow or underflow bits specified as they are unnecessary in 1-bit processing. The overflow bits are commonly needed for multi-bit multiplications. When we use the δ operator for controller implementations the underflow issue is almost inevitable as a consequence of multiplication by very small coefficients (Jones et al., 1998). In 1-bit processing, however,

Figure 5.2: Bit-width to represent coefficients and state variables.

these problems are easily overcome by removing all multiplications with the proposed controller structure shown in Fig. 4.7.

5.2 Hardware architecture

5.2.1 Basic arithmetic blocks

From Fig 5.1, only three arithmetic operations are needed to complete all the calculations in one-bit processing. They are conditional-negate (CN), add and shift, all in two's complement. Although a control system normally is negative-feedback, subtraction is not considered because this operation can be completed by applying a subtractive sign to its corresponding coefficient before the coefficients are loaded. Table 5.1 shows a comparison between these operations with an MAC (multiply-and-accumulation), which is adapted from the CSP developed by Cumplido-Parra (2001) for traditional controller implementations. The MAC uses two data types: a mixed format with a low-precision floating-point form which includes a 6-bit mantissa in two's complement and a 5-bit exponent for coefficients, and a 27-bit signed fixed-point form in two's complement for state variables. Also a multiply is included with the same data formats as the MAC. The results are obtained by realising these operations with a VLSI process which is the UMC 0.13 μm , 8-layer copper process. The power consumption of these designs are also estimated in Synopsys Power Compiler (Synopsys, 2003).

5.2. HARDWARE ARCHITECTURE

		Conventional	1-bit Processing		
	Multiply	MAC	CN	Add	Shift
area (μm^2)	21351.1	25369.1	1229.8	4800.4	3337.6
frequency (MHz)	621.2	440.7	2597.4	2143.6	2520.8
power (mW)	5.0805	5.7825	0.1838	0.7263	0.4820

Table 5.1: Comparisons between arithmetic operations

The table shows that the slowest arithmetic operation in one bit processing is the 'add', which occupies the largest silicon area and consumes the most power at the same time. However, this operation runs 3.45 times in speed and occupies only 22.48% in area when it is compared to the speed and the area of the 'multiply'. The power estimation shows that an 'multiply' consumes almost 7 times power of an 'add'. Definitely, by eliminating the multipliers one-bit processing can achieve the best performance in terms of area, speed and power, which traditional approaches have to compromise in order to achieve a local optimization. Compared to the 'MAC', the 'add' is around 4.86 times in speed, 18.92% in area and 8 times in power.

5.2.2 VLSI realisation

To realise one-bit processing in VLSI, the most straightforward approach is to implement the controller structure of Fig. 4.7 directly, which utilises the above basic arithmetic operators. Fig. 5.3 shows a direct implementation of a second order system, in which the thin signal lines are 1-bit variables and the thick lines are multi-bit variables with the format shown in Fig. 5.2. The input data comes from a $\Delta\Sigma$ modulator which acts as an analogue-to-digital converter and is located off-chip, but the $\Delta\Sigma$ modulator for the output data is integrated with the controller and resides on-chip. All the coefficients are hard wired on-chip. The states that are required for the next-sample calculations are stored in registers. At the beginning of each sampling time, the input u is read from the input port and the 1-bit output $\tilde{y}$ is written to the output port.

Figure 5.3: Direct implementation of a 2nd order control system in VLSI.

As all the calculations operate on 2's complement numbers, the sign bit identifies a value as positive or negative, where 1 means a negative value and 0 means a positive value. To represent a 1-bit signal, -1 is represented by 0 in the 1-bit register. Hence it is simple to use an inverter to implement the 1-bit quantiser of the $\Delta\Sigma$ modulator. The sign bit of x , which is $x[23]$, is fed into the inverter, providing a 1-bit representation $\bar{y}$ of the multi-bit variable y to feed back for the control system processing.

In the $\Delta\Sigma$ modulator, the first and third adders are the additions of a 1-bit signal to the 24-bit state-variables. In 24-bit fixed-point arithmetic, 1 is represented by the Hex code 010000, and -1 by FF0000. These additions therefore only change the eight most significant bits of the state-variables, resulting in a simplified add operation (SADD).

This architecture allows the whole control system processing to be performed in a pipelined manner. After completing the calculations, the circuit stops working and awaits the next sample trigger event. Hence it is the fastest and simplest implementation of a 1-bit controller. However, this architecture is not flexible as it is hardwired for one control task and can't be altered when committed to silicon.

5.2.3 Performance comparisons

We compare the circuit complexity and speed among the direct implementation of one-bit processing (BIT) and two conventional approaches: one uses traditional arithmetic operations such as 'add', 'multiply', etc. (C1); the other uses only the 'MAC' operation (C2). The comparisons are obtained using three metrics: area (A), time (T) and $A * T$ with a 2nd order IIR filter. $A \cdot T$ describes the combined efficiency of the circuits. The C1 approach results in an area of $62759.9 \mu m^2$ and a minimum sampling time of $0.0163 \mu s$. The C2 approach gets $77613.7 \mu m^2$ for area and $0.0185 \mu s$ for sampling time. The area and time for the BIT approach are $13484.3 \mu m^2$ and $0.0039 \mu s$ respectively. We normalise the area and time of the one-bit approach as 1, and the comparison results are obtained in Fig. 5.4. Although the sampling frequency for one-bit processing requires more than 10 times of the conven-

Figure 5.4: Comparison results of the direct implementations.

tional approaches, the results show that the direct implementation of one-bit processing is much more efficient than the other two approaches in terms of area and time.

Here, we only show the 2nd order IIR filter, but other examples give similar results.

5.3 Hardware verification

To verify the direct implementation in VLSI, hardware description language (HDL) is used to model the hardware behavior of a control system. The validation example in the last chapter is adopted and validated at the register transfer level (RTL). The RTL is a description of a digital electronic circuit in terms of data flow between registers.

Figure 5.5: RTL view of the $\Delta\Sigma$ modulator.

5.3.1 RTL modelling of the $\Delta\Sigma$ modulator

Since the input $\Delta\Sigma$ modulator is off-chip, it is necessary to prepare the 1-bit input before doing RTL simulation. Again, HDL is used to model the behavior of the second order $\Delta\Sigma$ modulator. Fig. 5.5 shows the RTL view of the second order $\Delta\Sigma$ modulator, in which u is an analogue input. The 1-bit quantiser is realised by inverting the sign bit of the output of the 'Add2'.

5.3.2 RTL modelling of the controller

According to Fig 5.1, the coefficients are

$$\begin{aligned}
 p_0 &= 4.183688603241274e - 001 \\
 p_1 &= p_2 = p_3 = p_4 = 0 \\
 q_0 &= -4.183688603241274e - 001 \\
 q_1 &= -1.456557203419229e + 000 \\
 q_2 &= -2.561385000357978e + 000 \\
 q_3 &= -2.251893614093164e + 000 \\
 k &= 2^{-7}
 \end{aligned} \tag{5.6}$$

where $k = 2^{-7}$ means a 7-bit right-shift operation. Fig 5.6 shows the modified δ -form together with the $\Delta\Sigma$ modulators. However, the real implementation of the coefficients is not exact due to the fixed-point arithmetic. Table 5.2 gives the quantised values of the coefficients using 24-bit fixed-point arithmetic and the corresponding hex codes. It shows that the maximum error is

5.3. HARDWARE VERIFICATION

Coefficients	Hex Code	Value	Error
p_0	006B1A	4.183654785156250e-001	0.0008%
q_0	FF94E6	-4.183807373046875e-001	0.0028%
q_1	FE9B20	-1.456558227539063e+000	0.00007%
q_2	FD704A	-2.561386108398438e+000	0.00004%
q_3	FDBF84	-2.251907348632813e+000	0.0006%

Table 5.2: 24-bit coefficients and errors

only around 0.0028%, which is more than enough for one-bit processing because the accuracy of the coefficients simply needs to have the same accuracy as is required for the overall system performance (typically 5% for control) (Forsythe and Goodall, 1991).

Figure 5.6: Modified canonic δ -form for the validation example.

The resulted HDL model is then compiled and synthesized using Synplify ASIC (Synplicity, 2003). Fig. 5.7 shows a RTL view of the validation example, in which the thick lines represent 24-bit data bus and the thin lines represent 1-bit data bus. u is 1-bit input and y is 1-bit output. All the coefficients and state variables are stored in the 24-bit registers with state

5.3. HARDWARE VERIFICATION

variables being updated each sampling cycle.

Figure 5.7: RTL view of the validation example.

5.3.3 Simulation results

The simulation is carried out under Modelsim (Model Technology Incorporated, 2001). The analogue input is 0.5, and the sampling clock runs at $1kHz$. Fig. 5.8 shows the RTL simulation results. In the figure: *in* is the input value; *sampl_clk* is the sampling cycle; *out* is the $\Delta\Sigma$ modulator's output; x_1 , x_2 , x_3 and x_4 are state variables, corresponding to the states as shown in Fig. 5.6; *coef₁* to *coef₅* are the results after the coefficients are conditionally negated by the 1-bit signals; *result* is the 1-bit output of the validation example.

Figure 5.8: RTL simulation results.

5.3. HARDWARE VERIFICATION

The Simulink model of Fig. 5.6 is simulated in Matlab. We compare the Modelsim results with the Matlab results. Fig. 5.9 illustrates the states' differences, where 'red' curves are the results in Matlab and 'blue' ones are those in Modelsim. Table 5.3 details these differences with static errors at static points and peak errors at peak points. The errors are all within 5%, which is acceptable in control. These errors are largely the result of the quantization errors due to the 1-bit quantisers in the $\Delta\Sigma$ modulators.

Figure 5.9: States differences: blue curves are obtained in Modelsim and red curves are obtained in Matlab.

Chapter 3 has introduced Wavelet denoising technique to remove the quantization noises. To analyze the 1-bit output of the validation example, Wavelet techniques are applied here. The 1-bit output is compared to the continuous output, which is obtained by simulating the continuous transfer function (Eq. 4.62) in Matlab, after the quantization noises are removed. Fig. 5.10 illustrates the difference between the continuous output and the denoised

State	Peak error	Static error
x_1	0.974%	0.638%
x_2	2.857%	0.546%
x_3	2.132%	2.575%
x_4	1.869%	2.47%

Table 5.3: States errors

1-bit output, where the ‘red’ curve is the continuous output and the ‘blue’ curve is the denoised 1-bit output. It shows that the maximum error is around 1.86%. However, this error also includes the wavelet filter’s error. Hence, in practice the 1-bit output is precise enough for control applications as long as the quantization noises are filtered effectively, which is not a problem for most physical systems.

Figure 5.10: Comparison between the denoised output and the continuous output.

5.3.4 Hardware performance

The HDL model of the 4th order validation example is compiled and synthesized, targeting the UMC 0.13 μm 8-layer copper process. The resulted application specific integrated circuit utilizing direct implementation can achieve a maximum clock frequency at 139MHz. Because it completes all the control calculations in one clock cycle, the maximum sampling frequency for this circuit reaches to as high as 139MHz, which easily beats many of today's fastest microprocessors.

This circuit is around $220 \times 200 \mu m^2$ in area when committed to silicon, and consumes less than 4 mW in power, making it one of the most efficient solutions for control system processing. This is much smaller than the microprocessors which it outperforms, although the comparison is somewhat unfair because of the lack of programmability of this implementation.

5.4 Summary

This chapter has described the hardware architecture for direct implementation of 1-bit processing. In order to represent small values in fixed-point registers, a modified canonic δ -form is proposed with scaling factors in the main loop. Special attention is given to numeric issues. 24-bit fixed-point arithmetic is used in IC design, in which the most significant bit is a sign bit; 7 bits are allocated to the integer part of a value and 16 bits are allocated to the fractional part. All the mathematic operations are based on two's complement. Basic arithmetic blocks for the direct implementation are analyzed, compared to a 'multiply' used in a conventional control system processor. The results show that these blocks are much more efficient than the 'multiply' in terms of area, speed and power.

A validation example is used to verify the direct implementation approach. Hardware description language is used to model the control system in RTL level. The simulation results show that this architecture is reliable for real-time control. The results of synthesizing the design were presented, showing that the direct implementation is one of the most efficient solutions

5.4. SUMMARY

for controller implementation.

However, the direct implementation is a hardwired-solution, which cannot be altered once committed to the silicon. Hence, it is necessary to carry out a more flexible solutions, which will be described in the next chapter.

Chapter 6

A $\Delta\Sigma$ -based Control System Processor

6.1 Hardware architecture

6.1.1 Introduction

To alleviate the rigidity of the direct implementation, a processor-based (programmable) solution is proposed in this chapter, resulting in a $\Delta\Sigma$ -based control system processor ($\Delta\Sigma$ -CSP). The processor-based implementation is reasonably simple by considering all the necessary elements needed for one-bit processing. The proposed architecture takes advantage of one-bit processing to permit efficient and cost effective realizations in VLSI.

Our main goal is to design a processor architecture that matches the control algorithm and not vice versa. This implies designing an application specific instruction set that best performs the control calculations. It is also expected that each instruction is executed in one clock cycle in order to improve the hardware efficiency.

The dedicated processor architecture together with the structure of the control algorithm, which will be implemented on this processor, will determine the hardware efficiency in terms of speed, area and power consumption. We have already discussed control forms in Chapter 4, and this chapter will determine the type and number of processing elements, the size and number

6.1. HARDWARE ARCHITECTURE

of the memories, and other necessary components.

To map the control algorithm to a processor architecture, it is divided into tasks or processes. These processes include data input and output (IO), data storage (Memories), timer, instruction fetching and decoding, next instruction address calculation (Program Counter) and arithmetic operations (ALU). This partitioning should allow all the processes to be mapped easily into hardware, minimising the resources required.

The number of concurrent operations can determine the amount and functionality of the hardware structures. For example, the maximum number of simultaneous data transactions that required for arithmetic operations determines the number of ALU ports. Also, communication channels between the ALU, accumulator, memories and IO must be assigned with specific data bus. For example, the data bus between the ALU and IO is 1-bit due to the feature of one-bit processing.

The execution of the control algorithm requires the repeated execution of a set of instructions (program). Although the number of instructions in the control loop can be small in the case of implementing a simple controller, the overhead that manipulate the program counter maybe relatively large. We therefore must pay special attention to the architecture of the program counter that implements control loops. Thus, the $\Delta\Sigma$ -CSP can provide a looping mechanism that introduces a short, or ideally zero, overhead.

The final step is to create a hardware model that supports the operations needed to implement the control algorithm. This hardware model is programmed using the hardware description language. The resulted $\Delta\Sigma$ -CSP is simulated and verified by running some validated programs with the application-specific instructions. The $\Delta\Sigma$ -CSP is then synthesized, floor planned and placed & routed. The final netlist can be verified by being downloaded into the FPGA and running the validated programs.

6.1.2 Instruction set architecture (ISA)

The $\Delta\Sigma$ -CSP adopts an application-specific instruction set to improve the hardware efficiency. There are three basic approaches to design the instruc-

tion set architecture (ISA):

- At one extreme, a single processing element (PE) executes all the arithmetic operations. The PE must be able to execute all the operations. The processing time will be equal to the product of the PE processing time and the total number of operations. This approach was adopted in the CSP (Cumplido-Parra, 2001), resulting in a high efficient control system processor.
- At the other extreme, one dedicated PE is assigned for each operation. The PE can therefore be optimised to execute a specific operation. The maximum number of PEs is determined by the parallelism in the algorithm, and the slowest PE determines the maximum clock rate. Most general purpose microprocessors adopt this approach.
- An intermediate solution is to combine the two approaches as described above.

The CSP is a high efficient processor architecture because it uses only one processing element. However, as we discussed in Chapter 5, the MAC unit is not the most efficient in terms of area, speed and power when compared to the second approach. The advantage of the first approach is that it can improve the area efficiency in VLSI because only one arithmetic unit is needed. To take advantage of the two approaches, the third approach will be used to implement the $\Delta\Sigma$ -CSP ISA, not only because it uses a minimum amount of hardware resources, but also because it results in a minimum consumption of power and a maximum clock frequency.

In Chapter 5 we have introduced the arithmetic blocks which are necessary for one-bit processing. They are 'CN', add, simplified add and shift. However, we proposed a conditional-negate-and-add operation (CNA) in the $\Delta\Sigma$ -CSP in order to effectively reduce the hardware resource. The $\Delta\Sigma$ -CSP therefore only requires two arithmetic operations — CNA and shift in total. Other instructions relate to data communication and control loop.

We used 24-bit fixed-point registers to store coefficients and state variables and 1-bit registers to store input and output data in the direct imple-

Figure 6.1: CNA architecture.

mentation. The $\Delta\Sigma$ -CSP will inherit these numeric formats.

CNA unit

For the re-modified canonic δ -form, as shown in Fig. 5.1, the conditional-negate-and-add (CNA) unit is utilized to perform most calculations in one-bit processing. We use $\ominus$ to represent ‘conditional negate’ here. The CNA unit is therefore written by

$$D = \ominus B|A + C \quad (6.1)$$

where B is either a coefficient or a state variable, A is a 1-bit signal, and C is a state variable. $\ominus$ is a symbol which means conditional-negate. Hence, $\ominus B|A$ conditional-negates B given a condition of A . A comes from either the input u or the output y . If A is 1, $\ominus B|A$ gives B . Otherwise, $\ominus B|A$ gives $-B$. Finally, to complete the CNA operation, the result of the conditional-negation is added to the state variable C and stored in the accumulator, ready for the next arithmetic operation. Fig. 6.1 shows an RTL view of the CNA unit, in which thick lines are 24-bit data bus and thin lines are 1-bit data bus.

Shift

The scaling factor k in the main loop of Fig. 5.1 is designed to be a power of 2 value, resulting in a shift operation in hardware. Because the sampling frequency is very high in one-bit processing, the coefficients are usually too small to be stored in 24-bit registers. In this case, k must be a value between

6.1. HARDWARE ARCHITECTURE

0 and 1 in order to enlarge the coefficients, which was proven by Eq. 5.5. Hence only the right shift operation is needed in one-bit processing in many applications. When realised in hardware, it corresponds to a signed shift right operation. Note that the CSP used a similar approach for the coefficients, except that each coefficient had its own power of 2 exponent.

Other instructions

Other instructions are needed for additional operations to the arithmetic ones in order to perform one-bit processing. These instructions include data communications and control logic operations.

For data communications, the instructions have functions of reading data from the data ROM (RDW), writing one-bit data to the IO registers (WRB), and writing initial or intermediate data to the data RAM (WRW). For control logic operations, the instructions are used to set the sampling time (SET) and the program counter (WPC) as well as idle the processes when all the instructions are complete within one control loop (HLT). The $\Delta\Sigma$ -CSP does not have stand-alone instructions for reading the data from the data RAM and IO registers because these functions are integrated in the arithmetic instructions.

Other arithmetic operations such as add, subtract, multiply or divide are not necessary in one-bit processing. In addition, very few logic operations are necessary for control system processing (Jones et al., 1998), and as a result, no Boolean unit is included in the processor design. In practice a system-on-chip solution would incorporate extra functionality for purely Boolean operations, e.g. a state machine.

All the instructions of the proposed $\Delta\Sigma$ -CSP are given in Table 6.1. This instruction set is fairly small and specialised to one-bit processing implementations.

Each instruction contains three elements: an opcode, an I/O address and a data RAM address. These elements specify the word length required to represent an instruction. In this processor design a 16-bit word format is

6.1. HARDWARE ARCHITECTURE

Opcode	Name	Function description
000	HLT	No operation
001	RDW	Read data from the data ROM
010	WRB	Output the result to the digital output ports
011	WRW	Write the intermediate states to the data RAM
100	SRS	Right shift
101	CNA	Conditional negate and accumulate
110	SET	Set the sampling frequency for the timer
111	WPC	Set the start value for the program counter

Table 6.1: $\Delta\Sigma$ -CSP instructions.

used with 4 bits allocated to the opcode, 3 bits to digital IO and 9 bits to the data memory address. The processor has only 8 instructions (see Table 6.1) which are sufficient to accomplish all the necessary operations in one-bit processing implementations. Although there are 4-bits for the IO, the most significant bit is allocated to a 1-bit register, which contains a constant value 1. Hence, 4 digital inputs and 4 digital outputs are provided in the IO block which allows a maximum of 4 inputs and 4 outputs for an MIMO (multi-input and multi-output) control system. As there are 9 bits to represent an address of the data RAM or data ROM, it allows access to a memory with a maximum size of $512 * 24b$. This is enough to perform a complex control system because in the processor design only the states, which are used in the next sample calculations, will be written to the data RAM.

Fig. 6.2 shows a diagram of executing one instruction. The instruction is fetched from the memory. Its 16-bit code is then decoded into three parts. The program counter (PC) will stop increasing when the instruction 'HLT' is read (being 'hold' in the diagram), and the program will stop. Otherwise, the arithmetic unit will take necessary operands to achieve certain arithmetic operations. In the $\Delta\Sigma$ -CSP, the result is then written to an intermediate device for next instruction. Concurrently the PC is increased by 1 automatically in

order to fetch the next instruction.

Figure 6.2: Procedure of executing an instruction.

Pipelining

Fig. 6.2 shows the processing procedure of an instruction. However, the $\Delta\Sigma$ -CSP does not execute the instruction in sequence strictly. On the contrary, it adopts a pipelining mechanism to speed up the process. When it takes an instruction, it breaks the instruction into some small processes and executes these processes in parallel. This mechanism can effectively reduce the time

6.1. HARDWARE ARCHITECTURE

that is required to execute a sequence of instructions, and allows the $\Delta\Sigma$ -CSP execute an instruction in one clock cycle. The $\Delta\Sigma$ -CSP performs the following processes in pipeline:

- Fetch a new instruction
- Retrieve the data from the data memory
- Fetch the one-bit data from the IO registers
- Execute the operation
- Save the data in the accumulator

All the instructions can be completed in one clock cycle but the pipelining mechanism results in a delay of more than one clock cycle from the time that the instruction is fetched to the time that the result is obtained. When designing the hardware we must pay special attentions to these time delays to make sure the timing is correct.

6.1.3 Microarchitectures

The $\Delta\Sigma$ -CSP adopts a quite general processor architecture but optimised for one-bit processing. All the calculations are carried out in the arithmetic and logic unit (ALU) with memories being used to store coefficients and instructions. One-bit registers are used for data input and output in the IO.

Memories

The $\Delta\Sigma$ -CSP uses two types of memories: read only memory (ROM) and random access memory (RAM). All the instructions of a control law are stored in a 16-bit program ROM. All the initial states and coefficients are stored in a 24-bit data ROM. A 24-bit data RAM is used to contain the intermediate data which are needed to execute the instructions.

Both data ROM and data RAM have an upper-limit size of 512 24-bit words. This is because only 9 bits of a 16-bit instruction are used to address

6.1. HARDWARE ARCHITECTURE

the data memories. We can increase the memories' size by increasing the instruction's bit-width. For example, if we choose a 24-bit word to represent an instruction, among which 16 bits are used to address data memories, this will allow the processor access 16M data memories. However, such a large size is not necessary for the $\Delta\Sigma$ -CSP, and only will waste the hardware resource. Although the scaling factors k are same in Fig. 5.1, for practical control they may vary. Here we assume they are different values, and the number of coefficients and states that are needed for one-bit processing can be obtained by

$$s_1 = 4 \times (n + 1) + 2 \quad (6.2)$$

where n is the order of a control law. A size such as 512 words therefore satisfies running a 126^{th} order control law which is rather complex. In practice, this size can run an even higher order control law because we normally use the same value for the scaling factors.

The program ROM can be in any size theoretically. However, it is totally unnecessary to adopt a large memory size in the $\Delta\Sigma$ -CSP. In one-bit processing, the number of instructions that are needed to perform a control law can be calculated by

$$s_2 = 13 \times (n + 1) + 5 \quad (6.3)$$

The size of the program ROM depends on the complexity of a control law. As the data memories can run a 126^{th} control law at least, it needs 1,656 pieces of instructions. Hence, in the $\Delta\Sigma$ -CSP, the program ROM is designed to be $4k$ 16-bit words, which needs a 12-bit address bus to access. This size enables the memory to carry 4,096 instructions in maximum, being enough to implement a very complex control law in one-bit processing.

In the $\Delta\Sigma$ -CSP, the data memories are exactly the same size. When the $\Delta\Sigma$ -CSP runs a program, it loads the initial states and coefficients from the data ROM to the data RAM before the control loop starts. In the data RAM, the coefficients will remain constant while the program is running. However, the states will be updated at every sampling time. Because the data ROM and data RAM share one data bus, a 'mux' is used to select data between them as shown in Fig. 6.3. When 'sel' is high, the 'mux' takes the data from

Figure 6.3: Data memories architecture.

the data ROM; otherwise, it takes the data from the RAM. Choosing the 'sel' operation in this manner is a kind of design art as this signal consumes power when it is high. Because the data ROM is only accessed before the control loop starts, it only takes a short time. However, the data RAM is frequently visited while a program is running. Obviously, it is more power efficient when a high state of the 'sel' is associated with the data ROM. The same reason is applied to the other control logic signal 'rw_ram' which controls the RAM whether it is read or written. Because the RAM is read more than it is written, the high state of the 'rw_ram' is associated with the writing operation; and the low state is associated with the reading operation. Also while a program is running, not every instruction needs to visit the data RAM. A 'ram.en' control logic signal is used. When it is high, the data RAM is allowed to be visited; and when it is low, the data RAM stops working. These control logics not only keep the power consumption to a minimum, but also enable the $\Delta\Sigma$ -CSP to read and write data to/from the memories in order.

Figure 6.4: Sample timer architecture.

Sample timer

The sample timer provides sampling clock to the $\Delta\Sigma$ -CSP. Fig. 6.4 shows the timer's inner RTL architecture. It contains a 24-bit register which stores the initial timer data and a 24-bit counter that can count the system clock (which is the time base for the $\Delta\Sigma$ -CSP) up to 2^{24} times. A control logic signal 'timerinit_ena' is used with a high state allowing the register to be updated. When the 'SET' instruction is read, the processor will give a high state of the 'timerinit_ena', and write a value to the register. This value along with the system clock is used to decide the sampling time. In digital control, the sampling frequency is usually constant. The timer is therefore only updated once during the execution of a control program. The 'timerinit_ena' therefore will keep low level in other cases, and the register will contain a constant.

Fig. 6.5 illustrates how the sample timer works. The timer produces a sampling clock which is called 'sampl_clk'. Before the data is loaded to the register, the 'sampl_clk' is a low state and the counter stays 0. After the timer is initialised, the counter starts increasing, and its value is compared to the initial timer data in every clock cycle. As soon as the counter reaches the

6.1. HARDWARE ARCHITECTURE

same value of the initial data, the 'sampl_clk' goes to high and the counter is reset to 0. Otherwise, the 'sampl_clk' remains low and the counter keeps increasing by 1. In each sampling cycle, the 'sampl_clk' stays at high level for only one clock cycle in order to reduce the power consumption.

Figure 6.5: Sample time scheme.

A minor point for the sample timer is that the timer counts the system clock cycle from 0. Hence, when it is required to count n times, the register

Figure 6.6: Program counter architecture.

should be loaded with a value of $n - 1$.

Program counter (PC)

The program counter provides a pc value that addresses the program memory. In conventional processor designs, the PC uses two registers: the start register stores an initial value that labels the starting point of a program loop, and the stop register stores a value that labels the stopping point of a program loop. The counter will keep increasing until it reaches a value that equals that in the stop register. Thereafter it will reload the initial value from the start register automatically. However, in the $\Delta\Sigma$ -CSP we adopt a novel architecture, in which only the start register is used (see Fig. 6.6). Instead of using the stop register, the $\Delta\Sigma$ -CSP uses a logic signal — ‘inc_pc’ to increase or stop the counter. When it is 1, the PC value will be incremented by 1 on every clock cycle. Otherwise, the counter will stop running the program.

The PC flow is illustrated in Fig. 6.7. The PC starts counting from 0 after power-on or system reset. The initial value is loaded to the start register when the logic signal ‘pcinit_ena’ is set to 1. The PC stops counting when the ‘inc_pc’ is 0. However, the PC will load the initial value from the start register and start counting again at the rising edge of the logic ‘sampl_clk’. This design is due to a fact that most control loops start at the beginning of each sample clock. Compared to the conventional design, it can effectively

reduce the PC's circuitry complexity.

Figure 6.7: Program counter flow.

IO

Although there are 4 bits for the IO address, the IO only provides 4 1-bit inputs, 4 1-bit outputs and a 1-bit constant. Table 6.2 shows each IO address and its corresponding IO port. At every rising edge of the sample clock, the IO will take the 1-bit inputs from off-chip $\Delta\Sigma$ modulators and 1-bit outputs from the ALU. As it is in the data RAM, one control logic signal 'rw_io' is used to control the IO whether it is read or written. When the 'rw_io' is high, the $\Delta\Sigma$ -CSP writes 1-bit ALU results to the IO; otherwise, the IO gives 1-bit data to the ALU for arithmetic operations although this one-bit data is only used by the 'CNA'. The result from the ALU is in 24-bit, but the IO only takes the most significant bit, being negated at the same time.

address	IO port
1xxx	1
0000	di1
0001	di2
0010	di3
0011	di4
0100	do1
0101	do2
0110	do3
0111	do4

Table 6.2: IO address.

Decoder

The instructions include three parts: 3-bit opcode, 4-bit IO adress and 9-bit memory address. These parts need to be disassembled in the decoder. The other important role of the decoder is that it produces all the control logic operations for the other components by enabling the right strobes in the right cycles. The states corresponding to each opcode are shown in Table 6.3.

6.1. HARDWARE ARCHITECTURE

opcode	000	001	010	011	100	101	110	111
timerinit_ena	0	0	0	0	0	0	1	0
pcinit_ena	0	0	0	0	0	0	0	1
sel	0	1	0	0	0	0	0	0
ram_en	0	0	0	1	1	1	1	1
rw_ram	0	0	0	1	0	0	0	0
rw_io	0	0	1	0	0	0	0	0

Table 6.3: $\Delta\Sigma$ -CSP states.

Accumulator

The CSP writes all its numerical results into the data RAM. However, in the $\Delta\Sigma$ -CSP, the immediate ALU result will be stored in the accumulator for the following instruction at the next clock cycle. Only the data that are needed for the calculations in the next sample will be written back to the data RAM. These data are normally state variables. The advantage of this design is that it can reduce the data memory size significantly. Also the data in the accumulator is used as an operand for most instructions.

Note that the data in this accumulator has to be cleared at the end of the program in order to ensure that no accumulator state is carried forward to the next sample processes or program loop. This operation can be realised via writing 0 to the accumulator when the 'HLT' instruction is read.

Arithmetic and logic unit

The ALU takes the decoded opcode and three inputs from IO, accumulator and data RAM respectively and performs the actual calculation. Each instructions will write its calculation result into the accumulator, although the actual arithmetic operations are only 'CNA' and 'SRS'. Table 6.4 shows the corresponding arithmetic operations of all the instructions, and Fig. 6.8 shows the architecture of the ALU. Here *alu_in* is a data from either the data RAM or the data ROM; *accum* is a data from the accumulator; *di* is a 1-bit

opcode	Arithmetic operation
000	$alu_out = 0$
001	$alu_out = alu_in$
010	$alu_out = accum$
011	$alu_out = accum$
100	$alu_out = accum \gg alu_in$
101	if ($di == 1$) $alu_out = alu_in + accum$ if ($di == 0$) $alu_out = -alu_in + accum$
110	$alu_out = accum$
111	$alu_out = accum$

Table 6.4: Arithmetic operations of all the instructions.

data from the IO; and *alu_out* is a result that the ALU produces.

Figure 6.8: ALU architecture.

VLSI realisation

So far we have introduced all the micro-architectural blocks that are needed to carry out one-bit processing in the $\Delta\Sigma$ -CSP. Fig. 6.9 shows an overview

6.1. HARDWARE ARCHITECTURE

of the processor architecture.

The processor architecture is programmed with Verilog in RTL level and synthesized using Synplify ASIC, targeting the UMC $0.13\mu m$, 8-layer copper process. It results in an overall cell count of fewer than 2,200 cells and a minimum frequency of $500MHz$. Its size is only $400 * 220(\mu m^2)$. The power is estimated with the Synopsys Power Compiler, resulting in a total consumption less than $280mW$.

Figure 6.9: $\Delta\Sigma$ -CSP architecture.

6.1.4 A reprogrammable architecture

The above processor architecture is not flexible as the data ROM and program ROM are not reprogrammable. Unless these read-only-memories are

6.1. HARDWARE ARCHITECTURE

designed off-chip, the processor is hardwired once a control program is downloaded. However, we prefer to integrate all the component into one chip so that control engineers do not need to put much effort on peripherals. Therefore a reprogrammable $\Delta\Sigma$ -CSP is proposed, in which the program ROM is replaced with a program RAM. A USB interface is also provided. The data ROM is eliminated because initialising the coefficients and state variables is achieved by downloading data to the data RAM directly.

USB interface

The USB serial communication interface allows the $\Delta\Sigma$ -CSP to exchange data with the host computer. As shown in Fig. 6.10, these include the 16-bit instructions (targeting the program RAM), the 24-bit coefficients and 24-bit initial value of the internal control states (both targeting the data RAM).

The USB interface is clocked from a second clock source running at 48 MHz. As the $\Delta\Sigma$ -CSP runs at much higher frequencies, a strict synchronization regime has been employed: Transfers from the high to the low frequency domain use pulse stretching circuits prior to data synchronized at the slow domain; Transfers from the low to the high frequency domain utilize a cascade of synchronizer flops. As the USB interface is utilized only at the beginning of the operation of the $\Delta\Sigma$ -CSP, the synchronization overhead is absolutely minimized. Note that this facility also provides a means by which adaptive control (varying control parameters) or reconfigurable control (varying control law) can be achieved.

VLSI realisation

A number of high-level parameters that affect the VLSI implementation of the reprogrammable $\Delta\Sigma$ -CSP are defined. These include the parameters that specify the size of a control program in 16-bit words and whether the Program RAM is implemented as an array of flops or using a single-port embedded SRAM. The design is validated at RTL level and subsequently, synthesized using Synopsys (Synopsys, 2004) Design Compiler. The optimized netlist is re-validated and then read into Synopsys Physical compiler

Figure 6.10: The reprogrammable $\Delta\Sigma$ -CSP architecture.

6.1. HARDWARE ARCHITECTURE

where an optimal placement is achieved using the Gates-to-Placed-Gates flow. The optimized and placed netlist is subsequently read into Cadence SoC encounter (Cadence, 2004) where the power plan is designed and certain physical constraints are specified. The target frequency is 400 MHz and the target technology is UMC 0.13 μ m, 8-layer copper process. Initial synthesis showed that the flop-based and the SRAM-based program RAM configurations achieve significantly different maximum operating frequencies with the flop-based configuration being much faster. This is attributed to further processing taking place immediately after a control program is read from the program RAM instead of the opcode being clocked as a register and utilized on the next cycle. As a result, both a flop-based and an SRAM-based configuration are developed to demonstrate that difference. The target frequencies for the initial logic synthesis stage are 400 MHz (flop-based configuration) and 300 MHz (SRAM-based configuration).

Fig. 6.11 depicts the Floorplan (placed design) and final layout (routed database). The major identifiable blocks are:

- USB Core: This is to the left of the Floorplan and occupies approximately 50% of the total silicon area. We used the Opencores (Opencores, 2004) USB 1.0 interface core and synthesized it for a clock frequency of 48 *MHz*.
- $\Delta\Sigma$ -CSP Program RAM: An array of flops for storing the control program and associated multiplexing logic.
- Data RAM: Coefficient/Data RAM. Implemented as an embedded memory of 512 words by 24 bits.
- BCSP Core: The processing logic of the $\Delta\Sigma$ -CSP.

The flop-based design was routed in Cadence SoC encounter and the post-layout data are shown in Table 6.5:

Figure 6.11: Flop-based $\Delta\Sigma$ -CSP.

Fmax (MHz)	355 (DC target 400 MHz)
Std cells (RAMs)	10505(3)
Area	$1194\mu m \times 594\mu m = 709438\mu m^2$
Core Utilization	64.5%

Table 6.5: Results of the Flop-based design.

The same implementation flow is carried out for a SRAM-based program RAM configuration. The Floorplan and final layout database are shown in Fig. 6.12. The results are tabulated below in Table 6.6.

Figure 6.12: SRAM-based $\Delta\Sigma$ -CSP.

The SRAM-based program RAM configuration exhibits very long run-time which reduces the operating frequency. It is therefore recommended that, for small control program size, the flop-based configuration should be chosen as it is much easier to achieve a good quality routed design with little effort. This is also the case for the SRAM-based configuration but requires significant more input on behalf of the place-and-route engineer. However, for large control program size, the SRAM-based configuration should be chosen as it is more area-efficient than the flop-based implementation.

6.2. SOFTWARE ARCHITECTURE

Fmax (MHz)	239.2 (DC target 300 MHz)
Std cells (SRAMs)	7486(4)
area	$1117.8\mu m \times 557.2\mu m = 622826\mu m^2$
Core Utilization	63.5%

Table 6.6: Results of the SRAM-based design.

6.2 Software architecture

6.2.1 Introduction

The $\Delta\Sigma$ -CSP is an ASIS-based (application specific instruction set) processor architecture. Its ISA only provides eight instructions to perform one-bit processing. Among them, seven instructions are needed if we adopt a programmable solution. This is because the 'RDW' instruction is only used to load the coefficients and initial states to the data RAM in the non-programmable processor architecture. However, in the reprogrammable processor architecture, this function can be achieved by downloading the data from the host computer to the data RAM directly.

These application specific instruction are very simple and easy to understand. Programming control laws in the $\Delta\Sigma$ -CSP therefore is very transparent and straightforward for control engineers. This section describes the programming details as well as introducing the usage of the instructions.

6.2.2 Control program flowchart

A control program can be written in many programming languages and implemented in different types of hardware; for example the C language is widely accepted by control engineers in embedded systems (Barr, 1999). However, no matter what hardware and programming language a control engineer uses, the programming scheme of most control laws is quite similar. Fig. 6.13 shows such a program flowchart, which can be summarised to three stages: initialisation, synchronisation and execution.

In the initialisation stage, all the initial states and the coefficients are

Figure 6.13: Control program flowchart.

6.2. SOFTWARE ARCHITECTURE

loaded. If an internal clock is used to define the sampling clock, the counter value is loaded. In some high level languages, the start register and the stop register of the program counter don't have to be stated as these values will be automatically recognised by the loop instructions after compilation, for example the 'while' and 'for' commands in C. However, in assembly or assembly-like language, these registers must be pre-defined, being applicable to most ASIS-based languages.

In the synchronisation stage, the hardware that runs the control program must be synchronised with other peripherals, especially the data acquisition card (DAC). For most control programs, a sampling clock is used for synchronisation, which is realised through a handshaking mechanism. At every rising edge of the sampling clock, the program sends a 'request' signal to a peripheral. The peripheral corresponds with an 'acknowledge' signal. Then the data are passed from the peripheral to the IO registers, being ready for the main control operations. Before the beginning of the control loop or after the end of the main control operations, the program idles till the next rising edge of the sampling clock.

In the final stage, the program executes all the calculations along with necessary data exchanges with the IO registers. In the diagram, sampling data happens at the very beginning and writing output does at the end of the control loop. However, the actual implementation does not always follow this scheme, for example they can happen simultaneously at the rising edge of the sampling clock or at the end of the control loop.

6.2.3 ASIS

In last section we have introduced that the ISA in the $\Delta\Sigma$ -CSP, which provides eight application specific instructions. These instructions are 'HLT', 'RDW', 'WRB', 'WRW', 'SRS', 'CNA', 'SET' and 'WPC'. All the instructions are 16-bit long (Fig. 6.14), in which the highest 3 bits represent opcode; the middle 4 bits represent IO address and the remaining 9 bits are address referred to data memories. Note that all these instructions write their arithmetic results to the accumulator immediately. More descriptions of each

Figure 6.14: Instruction format.

instruction are detailed as follows.

HLT

Syntax : *HLT*

Operation : $acc \leftarrow 0$

The HLT instruction stops the program, and sets the accumulator to 0. No IO address and memory address are assigned to this instruction. It relates to the program counter directly: when the HLT is read, the 'inc_pc' is set to 0 and the PC stops increasing. Therefore the HLT together with the PC and the sample timer constructs a null operation, which is indispensable to build a control loop as shown in Fig. 6.13. This assembles a set of operations in C language:

```
do{  
    acc = 0;  
}while(sampl_clk == 0);
```

The program counter is reactivated at the rising edge of the sampling clock, beginning with the value that is loaded to the start register and points to the starting point of the control loop. In other words the $\Delta\Sigma$ -CSP adopts a type of internal interrupt to control the sample timer and the program counter as well as the program loop.

RDW

Syntax : *RDW addr*

Operation : $acc \leftarrow ROM[addr]$

The RDW instruction contains an address that points to the data ROM. The IO address is not necessary to be assigned or it can be any value as the

IO will not be read or written.

This instruction reads the initial states and coefficients from the data ROM according to the address. The data will not be written to the corresponding address of the data RAM immediately. In the design, it is written to the accumulator first. In order to complete an initialising operation, the RDW instruction must be followed by an operation that writes the intermediate value in the accumulator to the data RAM given a specific address.

WRB

Syntax : *WRB* *addr*

Operation : *IO[addr] ← acc*

The WRB instruction writes the intermediate value in the accumulator to the corresponding IO address. In the hardware design, all the data adopt a two's complement format. Therefore the most significant bit of the data represents its sign: 0 being a positive value and 1 being a negative value. This is perfect in one-bit processing as the outputs, which are from the quantiser of the $\Delta\Sigma$ modulator, are defined by 0 or positive being 1 and others being 0. Hence, this is only an inverse operation of the most significant bit. This function is realised by hardware means in the IO. The WRB instruction only needs to write the data to be quantised to the IO block.

The WRB instruction doesn't allow access to the data memories. Therefore, no memory address needs to be assigned for it.

WRW

Syntax : *WRW* *addr*

Operation : *RAM[addr] ← acc*

The WRW instruction writes the intermediate value in the accumulator to the corresponding data RAM address. This instruction enables only necessary data to be written to the memory, for example the internal state variables. Therefore, the data RAM can be reduced to a small size while still being able to run very complex control systems. At the initialising stage of a control program, the WRW together with the RDW initialises all the states

and loads the coefficients in the data RAM.

In the WRW instruction, no IO address needs to be specified as the IO will not be visited during the execution of this instruction.

SRS

Syntax : *SRS* *addr*

Operation : $acc \leftarrow acc \gg RAM[addr]$

The SRS instruction is a signed right shift operation. It corresponds to the scaling factors in the main control loop. The scaling factors are designed to be negative powers of two, i.e. within the range of 0 and 1. As it is a shift operation, the power rather than the scaling value itself is used in programming. The negative power is loaded to the data memory in the initialising stage, and the instruction right shifts the value in the accumulator according to the power value at the corresponding address.

The SRS instruction doesn't need to access the IO, so the IO address will not be specified.

CNA

Syntax : *CNA* *addr1* *addr2*

Operation : $acc \leftarrow \ominus RAM[addr2] \parallel IO[addr1] + acc$

The CNA instruction performs a conditional-negate-and-add operation. It is the only instruction that uses both the IO address and the data address in programming. The CNA and SRS are the two unique instructions that are used for arithmetic operations in one-bit processing.

It was mentioned in the last section that the IO contains a constant register which stores a constant value 1. This adds more flexibility to the CNA operation, which allows the CNA perform an 'add' operation. Fig. 6.15 shows how the add operation is distinguished from the CNA operation. When the highest bit of the IO address is 1, it indicates an add operation because it reads the constant 1 from the IO. As we know when the 1-bit signal is 1, the CNA operation does not negate the coefficient or the state variable which is read from the data RAM, being actually an add operation.

Figure 6.15: An add operation.

Otherwise, the CNA conditionally negates the coefficient or the state variable with the condition of the 1-bit signal from the IO registers other than the constant register.

The CNA operation happens when a coefficient is conditionally negated and added to the intermediate value that is in the accumulator; otherwise it is an add operation in which a state variable is added to the intermediate value.

SET

Syntax : **SET** *addr*
Operation : $timer \leftarrow RAM[addr]$

The SET instruction contains two fields: the operation code field that identifies the instruction and the source field that contains the address of the input value in the data memory. The IO address for this instruction does not need to be specified. It is used to load a counter value to the sample timer. The instruction takes the value from the data RAM according to the given address. Although it writes this data to the accumulator as other instructions, the data is also written to the sample timer simultaneously.

As soon as the sample timer is set, it runs independently. At each sampling interval, it gives a high logic for one system clock cycle, which is used to interrupt the idling process in the control loop and request the IO to sample data.

WPC

Syntax : **WPC** *addr*
Operation : $pcstart \leftarrow RAM[addr]$

The WPC instruction resembles the SET instruction. It also contains two

fields: the operation code field that identifies the instruction and the source field that contains the address of the input value in the data memory. The IO address for this instruction does not need to be specified either. It is used to load a counter value from the data RAM according to the given address to the start register in the PC. The value that is written to the PC register indicates the beginning point of a control loop.

As soon as the PC is configured, it runs independently from the program. However, it is associated with the sample timer as we discussed in the last section.

6.3 Simulation results

6.3.1 Digital simulation

In order to verify the applicability of the $\Delta\Sigma$ -CSP hardware and software architectures, the validation example that was used for the direct implementation is used by being implemented into the non-reprogrammable $\Delta\Sigma$ -CSP. The 24-bit coefficients and their errors compared to the real values have been tabulated in Table 5.2. The controller, which is based on one-bit processing as shown in Fig. 5.6 (not considering the first $\Delta\Sigma$ modulator), is programmed with the application specific instructions. Fig. 6.16 shows a brief program and its corresponding 16-bit binary codes ¹.

Both the first $\Delta\Sigma$ modulator and the non-reprogrammable $\Delta\Sigma$ -CSP are modelled with HDL. The hardware behaviours are simulated in the Modelsim with the program codes and the initial values being downloaded to the program ROM and the data ROM respectively. The RTL simulation results are carried out by a step input with an amplitude of 0.5. Here we assume the system clock runs at $1MHz$. As the sampling frequency is $1kHz$, a counter value 999 must be loaded to the sample timer. The control loop starts at the 35th instruction so that a value 35 is loaded to the start register in the program counter.

The one-bit output of the $\Delta\Sigma$ -CSP is denoised by the wavelet filter, and

¹The complete program is shown in Appendix A

6.3. SIMULATION RESULTS

RDW P0;	} Initialising variables	001_0000_000000000
WRW P0;		011_0000_000000000
RDW Q0;		001_0000_000000001
WRW Q0;		011_0000_000000001
 		
SET timer;	} Control loop	110_0000_000001000
WPC pcstart;		111_0000_000001001
HLT; // wait		000_0000_000000000
CNA In0 P0;		101_0000_000000000
CNA Out0 Q0;		101_0100_000000001
SRS S;	} Control loop	100_0000_000000111
CNA 1 X0;		101_1000_000001010
WRW X0;		011_0000_000001010
 		
WRB Out0;		010_0100_000000000
HLT; //wait		000_0000_000000000

Figure 6.16: The validation example program and its binary codes.

cross-referenced against the result which is obtained by simulating the validation example's continuous transfer function in Matlab. Fig. 6.17 shows a comparison of the results: Graph (a) shows the continuous output and the de-noised processor output; Graph (b) is the errors between the two results. We see that the errors are less than 3%. However, the real errors may be even less because the current errors are largely contributed by the wavelet filter. This demonstrates that the hardware and software architectures of the $\Delta\Sigma$ -CSP specification are correct and can be applied to practical applications with one-bit processing technique. The complete RTL simulation results are shown in Fig. 6.18)

6.3. SIMULATION RESULTS

(a) The outputs

(b) Errors of (a)

Figure 6.17: Simulation results.

Figure 6.19: $\Delta\Sigma$ -CSP interface.

6.3.2 Hardware-in-loop simulation

In this section, we will discuss the real applications of the $\Delta\Sigma$ -CSP. The DC motor control will be demonstrated with a hardware-in-loop simulation approach, in which the $\Delta\Sigma$ -CSP is realised on an FPGA and plays as the hardware; the physical system and the $\Delta\Sigma$ modulators are simulated by C language and work on a personal computer.

System interface

The $\Delta\Sigma$ -CSP will be embedded within the complete system which also includes the physical system. The processor will normally be programmed in a separate programming system. The resulted binary instruction codes then are downloaded to the program memory. 4 $\Delta\Sigma$ modulators are provided to perform analogue to 1-bit conversion. The outputs of the processor are $\Delta\Sigma$ modulated control signals which can be used to interface the physical system directly.

Fig. 6.19 shows a complete $\Delta\Sigma$ -CSP interface. The outputs of the $\Delta\Sigma$ modulators are registered in the IO block within each sampling cycle. Hence each input can be accessed at any time between consecutive samples because it is stored in its own dedicated register.

FPGA-based $\Delta\Sigma$ -CSP

The $\Delta\Sigma$ -CSP is designed as a stand-alone control system processor for control applications using one-bit processing. However, a lot of efforts like optimisation, packaging and etc are still needed to realise such a processor in silicon. As an alternative, FPGAs are commonly used for functional verification. The

6.3. SIMULATION RESULTS

FPGA avoids the initial cost, lengthy development cycles. Also the FPGA is a reprogrammable hardware, which permits design upgrades in the field with no hardware replacement necessary.

The Xilinx Spartan2E XC2S600E-FG456 (speed grade -6) is chosen to implement the $\Delta\Sigma$ -CSP. This FPGA device provides 15,552 logic cells with up to 600,000 system gates. It supports 514 user IOs in maximum. The Spartan2E FPGAs are customized by loading configuration data (design) into the internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions that are implemented in the FPGA (Xilinx Inc., 2003).

The non-reprogrammable $\Delta\Sigma$ -CSP is synthesized with the Xilinx ISE design tool (Xilinx Inc., 2005). The synthesis results indicate that the design can achieve a minimum frequency of 50MHz. This frequency is only $\frac{1}{10}$ of the application specific IC, but it is fast enough to run the demonstrating examples. The design uses 36,741 system gates, which occupies around 6% resource of the Spartan2E XC2S600E. Then the resulted configuration data together with the control program and initial data are downloaded to the FPGA for real-time control applications.

Hardware-in-loop simulation

Due to the lack of real physical systems, a hardware-in-loop simulation approach is proposed to verify the practicability of the $\Delta\Sigma$ -CSP for real-time control applications using one-bit processing.

Fig. 6.20 shows the hardware-in-loop simulation scheme. Here 1-bit signals, which are produced by the $\Delta\Sigma$ -CSP, are sampled by an A/D converter, then fed into the physical system which is simulated in the personal computer by C language. The output of the physical system is sampled by the $\Delta\Sigma$ modulator which is also simulated in the personal computer, and hence is a 1-bit digital signal for one-bit processing. The Spartan2E FPGA IO supports a maximum of 3.3 volts at its pins. Hence, to produce a high state 1 in the corresponding IO register, the input voltage is configured to 3.3V. In this case, as the 1-bit signals from the simulated $\Delta\Sigma$ modulators are represented

Figure 6.20: Hardware-in-loop simulation scheme.

by 1, these 1-bit signals must be scaled by 3.3. Hence after they are converted to their analogue format using D/A converters and applied to the FPGA IO pins, they can produce logic 1s to the corresponding one-bit registers in the IO block. The digital $\Delta\Sigma$ modulator, which samples the DC motor output, can also be replaced by a standard part, for example the ADS1201 from the Burr-Brown Corporation (Burr-brown Corp., 1997).

Fig. 6.21 shows a scene of the hardware-in-loop simulation workbench. The workbench includes a personal computer, which is based on the Intel's Pentium II processor running at 300 MHz, a 12-bit data acquisition card and a Spartan2E development card which also contains a 100MHz oscillator for clock generation (Memec Design, 2003). The configuration data are downloaded to the FPGA device via a RJ45 type JTAG cable. The data acquisition card is attached to one ISA slot in the computer. The analogue signals, which are from/to the data acquisition card, are connected to the FPGA IO pins.

The physical system and the $\Delta\Sigma$ modulators are programmed in C language. Then the program is built into an executable file with Visual C++. The $\Delta\Sigma$ -CSP must be synchronised with the personal computer while running the hardware-in-loop simulation. The synchronization is realised through the sampling clock signal which is given by the $\Delta\Sigma$ -CSP. The computer runs the initialising part of the physical system and the $\Delta\Sigma$ modulators first, and the program will wait for the rising edge of the sampling clock before it runs the simulation loop. Then the $\Delta\Sigma$ -CSP runs the control law and gives the sampling clock to the computer simultaneously. As soon as the rising edge of the sampling clock is detected, the computer starts the simulation loop

Figure 6.21: Hardware-in-loop simulation workbench.

which includes sampling the 1-bit signals from the $\Delta\Sigma$ -CSP, running the physical system and the $\Delta\Sigma$ modulators, and outputting the 1-bit signals to be controlled.

DC motor control

The DC motor control has been used to verify the one-bit processing concept. Here it is adopted as a validation demonstrator for the hardware-in-loop simulation. By considering the scaling factors in the main loop, the coefficients for the controller, when the sampling frequency is set at $1000Hz$, are calcu-

6.4. SUMMARY

lated according to Eq. 5.5 and listed below.

$$\begin{aligned} p_0 &= 2.684354560000000e - 003, \\ p_1 &= 5.244977151999999e + 000, \\ p_2 &= 4.096655359999999e + 000, \\ p_3 &= 1.280512000000000e + 000, \\ p_4 &= 1.000000000000000e + 000, \\ q_0 &= 0, \\ q_1 &= 2.097152000000000e + 001, \\ q_2 &= 1.802240000000000e + 001, \\ q_3 &= 1.408000000000000e + 001, \\ k &= 2^{-7}. \end{aligned} \tag{6.4}$$

The coefficients and initial states are represented in a 24-bit fixed-point word format. It results in a maximum error of 0.524% for p_0 , which is precise enough to meet the 5% criterion for real-time control. Thus, the coefficients can be safely used in calculations to implement the control law in the $\Delta\Sigma$ -CSP.

The step response of the hardware-in-loop simulation is compared with that of the digital simulation which was carried out in Matlab. The digital simulation takes Eq. 4.64 as a continuous control system in Simulink. The results are shown in Fig. 6.22(a). A small area of the simulation results is enlarged as shown in graph (b) because this area is where the peak response of the control system happens. The only difference is that the motor oscillates a bit more heavily with the hardware-in-loop simulation. This is due to the effect of pulse density modulation, and will not affect the whole system performance.

6.4 Summary

In this chapter, we have developed two processor-based architectures which are specified for one-bit processing. Both architectures share most common microarchitectures, but they use different types of program memory,

6.4. SUMMARY

Figure 6.22: Comparisons between the hardware-in-loop simulation and the digital simulation.

being distinguished by reprogrammable and non-reprogrammable. The non-reprogrammable solution stores the control program in program ROM and the initial states together with the coefficients in data ROM. However, the reprogrammable solution only contains a program RAM to store the control program. The initial states and coefficients are downloaded to the data RAM directly. Also two types of program RAM have been analysed with the reprogrammable solution. The results show that in spite of its inflexibility the non-reprogrammable solution is more efficient than the reprogrammable solution.

We also present the software architecture for the $\Delta\Sigma$ -CSP. The program scheme for control laws is clarified at the beginning and all the instructions are specified thereafter.

In the conventional CSP, the MAC instruction writes its result directly to the data memory. However, in the $\Delta\Sigma$ -CSP, all the instructions write their result to the accumulator first. Although this design takes two clock cycles (two instructions in the other word) to update the states, it avoids unnecessary waste of the hardware resource.

The application specific instructions along with the $\Delta\Sigma$ -CSP allow control engineers to implement control algorithms in a very simple way. The actual number of instructions and data requirements for a control program depends on the control system characteristic and can be estimated, as was given by Eq. 6.2 and Eq. 6.3. The control program runs recursively with the state variables being updated for the next sampling clock and output being produced during each control loop.

The validation example is programmed with the application specific instructions. Then the program is carried out in the $\Delta\Sigma$ -CSP. The results have shown that the proposed hardware is applicable for one-bit processing, validated using a relatively simple 4th order controller example but extension to high-order MIMO controller is straightforward. A hardware-in-loop simulation approach is also proposed, in which the $\Delta\Sigma$ -CSP is realised on the FPGA. A DC motor control is demonstrated to verify the practicability of the $\Delta\Sigma$ -CSP for real-time control along with one-bit processing.

Chapter 7

$\Delta\Sigma$ -CSP Benchmark

In the previous chapters, the $\Delta\Sigma$ -CSP together with one-bit processing has been proven applicable for real-time control. The synthesis results have already shown that the resulting processor architectures are small in size. However, this chapter compares the non-reprogrammable $\Delta\Sigma$ -CSP against other processors using the validation example, showing its efficiencies in terms of speed and power.

7.1 Introduction

The $\Delta\Sigma$ -CSP performance is compared against the performance of the CSP and some popular commercially available processors. These processors are listed in Table 7.1. To evaluate the performance of these processors, the validation example is programmed: application specific instructions for the CSP and the $\Delta\Sigma$ -CSP; for the other processors, it is programmed in C first and then compiled to assembly code targeted to different types of the processors. The resulted instruction code is therefore analysed to produce an estimation of the computation time for the validation example.

The benchmark includes a comparison of the computation time, the number of instructions that required to perform the validation example, average clock cycles to execute an instruction. At the same time, the power consumption, hardware technology and voltage supply are presented.

7.2. SELECTED PROCESSORS

Processor	Manufacturer	Device type
CSP	N/A	Control system processor
TMS320C31	Texas Instruments	Digital signal processor
TMS320C54	Texas Instruments	Digital signal processor
C167	Infineon	Microcontroller
Strong-ARM	Intel/ARM	General-purpose processor
Pentium III	Intel	General-purpose processor

Table 7.1: Selected processors for benchmark against the $\Delta\Sigma$ -CSP.

7.2 Selected processors

7.2.1 CSP

The CSP takes a special format for numeric operations: 27-bit fixed-point for state variables and a simple low-precision floating-point form for coefficients with a 6-bit mantissa in 2's complement format and a 5-bit exponent. Its main application area is real-time control. The CSP consists of a MAC unit to perform all the numeric calculations. It also provides a multiport data memory (three read and one write) on-chip.

7.2.2 TMS320C31

The TMS320C31 is a 32-bit floating-point digital signal processor. It is targeted at digital audio, data communication, industrial automation and control. The processor provides a large address space. It integrates a multi-processor interface, one external interface port, two timers, one serial port and multiple-interrupt structure. It can perform parallel multiply and other ALU operations on integer or floating-point data in a single clock cycle. It also consists of a general-purpose register file, a program cache, internal dual-access memories, one DMA channel which supports concurrent I/O (Texas Instruments Inc., 2004).

7.2.3 TMS320C54

The TMS320C54 is a cost-efficient digital signal processor compared to the TMS320C31. To reduce the price, it adopts a 16-bit fixed-point format. This processor is designed to support personal and portable products such as digital music players, 3G mobile phones, digital cameras and MIPS-intensive voice and data applications. It has a modified Harvard architecture that has one program memory bus and three data memory buses. The processor also provides a RISC-like instruction set.

7.2.4 C167

Like the TMS320C54, the Infineon's C167 is a 16-bit fixed point microcontroller. It is targeted towards low cost applications, being widely adopted in real-time embedded control applications such as automotive, industrial control, computer peripherals and data communications. It is one of the world's most successful 16-bit fixed-point architectures. This microcontroller is featured by RISC based architecture, 16-bit CPU with 4 stage pipeline, 32-bit bus interfacing to internal ROM, and von Neumann address space (Infineon Technologies AG, 2000).

7.2.5 Strong-ARM SA-110

The strong-ARM SA-110 is a 32-bit fixed-point general-purpose microprocessor. It is targeted at low power applications. It is used in a wide range of embedded applications, including high bandwidth network switching, intelligent office machines, storage systems, remote access devices, internet appliances, personal digital assistants, handheld personal computers and mobile phones (Intel Corp., 2000).

7.2.6 Pentium III

The Intel's Pentium III processor is a 32-bit floating-point general purpose processor, which is targeted at desktop and mobile computing. It has a

7.2. SELECTED PROCESSORS

superscalar architecture, large on-chip caches, 64-bit data bus, extended instruction set that includes instructions optimised for signal processing. The Pentium series processors have been described as having a RISC core for a subset of its instructions, but in reality the Pentium processors contain a mixture of hardwired simple instructions and micro-coded complex instructions (Intel Corp., 2005)

So far we have introduced the processors that are used for benchmarking against the $\Delta\Sigma$ -CSP. Table 7.2 shows a summary of the selected processors' features.

Processor	Word format	Frequency (MHz)	Main applications
$\Delta\Sigma$ -CSP	24-bit fixed-point	500	Real-time control, digital audio
CSP	Mixed-format	50	Real-time control
TMS320C31	32-bit floating-point	60	Digital audio, data com- munications, industrial automation and control
TMS320C54	16-bit fixed-point	160	Portable products, voice and data applications
C167	16-bit fixed-point	25	Automotive, computer peripherals, industrial control
Strong-ARM	32-bit fixed-point	233	Embedded applications
Pentium III	32-bit floating-point	500	Desktop and Mobile computing

Table 7.2: Processors' features.

7.3 Programming and instruction code

7.3.1 C program

The real-time instruction code of the 4th-order validation example for the selected processors is carefully programmed to ensure that the benchmark is as precise as possible. Except for the $\Delta\Sigma$ -CSP and the CSP, the example is first programmed in C for the other processors and then compiled into assembly code. For the benchmark, we only consider the main control loop, and the initialising stage is neglected. Fig. 7.1 shows the main routine of a C program for the 4th order validation example.

```
void main()
{
    do
    {
        u = _inp(ad1);
        x0 = x0 + a0 * x4 + b0 * u;
        x1 = x1 + a1 * x0 + b1 * u;
        x2 = x2 + a2 * x1 + b2 * u;
        x3 = x3 + a3 * x2 + b3 * u;
        x4 = x0 + x1 + x2 + x3;
        x5 = c0 * x0 + c1 * x1 + c2 * x2 + c3 * x3;
        y = x5 + d * u;
        _outp(y, da1);
    } while (1);
}
```

Figure 7.1: C program for the validation example.

An important issue for programming the control law is that each processor has its data type, which decides the complexity of the assembly code after compilation. Table 7.3 shows the resolution and data format used to represent the coefficients and state variables for each processor. The table also shows the resolution that is used to perform the multiplication of a coefficient and a state variable. However, for the $\Delta\Sigma$ -CSP, no multipliers are needed because of the feature of one-bit processing.

7.3. PROGRAMMING AND INSTRUCTION CODE

Processor	Coefficient	State variable	Multiplication
$\Delta\Sigma$ -CSP	24-bit fixed	24-bit fixed	N/A
CSP	11-bit floating	27-bit fixed	27×5 mixed
TMS320C31	32-bit floating	32-bit floating	32×32 floating
TMS320C54	16-bit integer	32-bit integer	32×16 fixed
C167	16-bit integer	32-bit integer	32×16 fixed
Strong-ARM	32-bit integer	32-bit integer	32×32 fixed
Pentium III	32-bit floating	32-bit floating	32×32 floating

Table 7.3: Processors' data format.

Both the $\Delta\Sigma$ -CSP and the CSP use application specific instructions, so the assembly code of the validation example is obtained directly. However, the assembly code for the other processors is obtained by compiling the C program by the corresponding compilers. Thereafter the number and type of instructions that are required to perform the algorithm can be identified. Table 7.4 tabulates the compilers for the other processors.

Processor	Compiler
TMS320C31	Code composer studio
TMS320C54	Code composer studio
C167	Keil C compiler
Strong-ARM	High C/C++ compiler for ARM
Pentium III	C/C++ compiler

Table 7.4: Compilers to generate assembly code for the benchmark.

7.3. PROGRAMMING AND INSTRUCTION CODE

7.3.2 Assembly code

The instruction code for the $\Delta\Sigma$ -CSP is shown in Appendix A.1. To demonstrate the assembly code for the other processors, we compile one line among the C program to assembly code. This C code performs the operation

$$x0 = x0 + a0 * x4 + b0 * u$$

Fig. 7.2 shows the corresponding instruction code for the CSP. It is able to perform the operation with just two MAC operations. This is because all the operands are stored in the register file and therefore can be accessed without any delay.

```
MAC  x0, a0, x4, x0;  // x0 = a0 * x4 + x0
MAC  x0, b0, u, x0;   // x0 = b0 * u + x0
```

Figure 7.2: Instruction code for the CSP.

The assembly code for the TMS320C31 are obtained by compiling the C code with the code composer studio. Fig. 7.3 shows its assembly code. A total of 7 instructions are needed to perform the operation. The first two instructions load data into the register file. Then two multiply and one addition instructions are executed with operands read both from the memory and register file. A final addition of two values stored in registers produces the final result, which is then stored in the memory. This DSP can perform operations where some operands are read directly from the memory, which reduces the number of load instructions that move data from the memory to the register file. As a consequence, it reduces the computation time.

```
LDFU  @0a03fh, R1;
LDFU  @0a049h, R0;
MPYF  @0a017h, R0;
MPYF  @0a01ah, R1;
ADDF  @0a04ah, R0;
ADDF3  R0, R1, R0;
STF   R0, @0a04ah;
```

Figure 7.3: Assembly code for the TMS320C31.

7.3. PROGRAMMING AND INSTRUCTION CODE

Unlike the DSPs, the C167 requires the operands that are used for multiplications and additions to be stored in the registers. Also the C167 can only handle 16-bit fixed-point data format. As a consequence, the resulting assembly code for the C167 includes a large number of 'move' instructions to exchange the data between the memory and the registers. The number of instructions is increased because it has to call a subroutine that performs the multiplication and two instructions are needed for one add operation. Fig. 7.4 shows the assembly code for the C167. There are 34 instructions in total to carry out the complete operation.

```
MOV    R6, DPP2 : 0x000C;
MOV    R7, DPP2 : 0x000E;
MOV    R4, DPP1 : 0x0034;
MOV    R5, DPP1 : 0x0036;
CALLA  CC_UC, ?C_LMUL (0x21E);
MOV    R8, R4;
MOV    R9, R5;
ADD    R8, DPP2 : 0x0010;
ADDC   R9, DPP2 : 0x0012;
MOV    R4, DPP2 : 0x0024;
MOV    R5, DPP2 : 0x0026;
MOV    R6, R14;
MOV    R7, R15;
CALLA  CC_UC, ?C_LMUL (0x21E);
ADD    R4, R8;
ADDC   R5, R9;
MOV    DPP2 : 0x0010, R4;
MOV    DPP2 : 0x0012, R5;

?C_LMUL:
MULU   R5, R6;
MOV    R5, DPP3 : 0x3E0E;
MULU   R7, R4;
ADD    R5, DPP3 : 0x3E0E;
MULU   R4, R6;
ADD    R5, DPP3 : 0x3E0C;
MOV    R4, DPP3 : 0x3E0E;
RET
```

Figure 7.4: Assembly code for the C167.

Fig. 7.5 shows the assembly code that is required to perform the operation

7.4. BENCHMARK RESULTS

using the Strong-ARM processor. Like the C167, the ARM processor requires the operands to be stored in the registers. However, unlike the C167, it can handle 32-bit fixed-point data format. Hence, the number of instructions that are need to exchange data is significantly reduced. The ARM processor also provides the multiply-and-accumulate instruction. So it only needs two multiply-and-accumulate instructions to complete the operation. The result is written back to the memory by a single store instruction, which is quite similar to the WRW instruction in the $\Delta\Sigma$ -CSP. For the Strong-ARM, 8 instructions are needed.

```
ldr  %r3, [%r10, #A+12-.L00STRING2];
ldr  %ip, [%r9, #X+8-.L00BSS];
ldr  %r2, [%r9, #X+12-.L00BSS];
mla  %r2, %ip, %r3, %r2;
ldr  %r3, [%r10, #B+12-.L00STRING2];
ldr  %r4, [%r8, #U-.L00DATA];
mla  %r2, %r3, %r4, %r2;
str  %r2, [%r9, #X+12-.L00BSS];
```

Figure 7.5: Assembly code for the Strong-ARM.

7.4 Benchmark results

7.4.1 Power consumption

The power consumption is decided by the hardware technology and the voltage supply. Table 7.5 shows how the $\Delta\Sigma$ -CSP compares with the other processors in power consumption, which puts the $\Delta\Sigma$ -CSP into one of the most power-efficient processors. Also the power consumption is directly proportional to the clock frequency. Thus, it is possible to further reduce the power consumption by setting the processor to a slow clock frequency. Instead of clocking the $\Delta\Sigma$ -CSP at the maximum frequency, it is sufficient to use a moderate clock frequency which will allow the $\Delta\Sigma$ -CSP to perform the control calculations fast enough as well as reduce the power consumption.

7.5. SUMMARY

Processor	Technology (μ m)	Power supply (V)	Power consumption (W)
$\Delta\Sigma$ -CSP	0.13	3.3V	0.28
CSP	0.25	3.3V	0.82
TMS320C31	0.6	1.8V	2.6
TMS320C54	0.6	5V	> 0.77
C167	0.5	5V	1.5
Strong-ARM	0.35	2V	1
Pentium III	0.25	2V	> 20

Table 7.5: Benchmark results of power consumption.

7.4.2 Processing speed

The benchmark is obtained by comparing the speed of processing the validation example between the $\Delta\Sigma$ -CSP and the other processors. Note that the number of clock cycles for each processor to execute one instruction is different. Thus, we must take the clock cycles per instruction into consideration. Table 7.6 shows the benchmark results. The processing speed is also normalised by setting the $\Delta\Sigma$ -CSP to 1. The table shows that our proposed $\Delta\Sigma$ -CSP is the most efficient in speed. The closest processor in performance is the Pentium III, which is still 1.35 times slower than the $\Delta\Sigma$ -CSP. It also indicates that the $\Delta\Sigma$ -CSP can achieve a maximum sampling frequency of more than $20MHz$, which is almost 10 times that of the CSP.

7.5 Summary

This chapter compares the $\Delta\Sigma$ -CSP with other processors in terms of power consumption and processing speed. The main feature of the selected processors are described. The benchmark is based on the number of instructions

Processor	$\Delta\Sigma$ -CSP	CSP	TMS320C31	TMS320C54	C167	Strong-ARM	PentiumIII
Frequency (MHz)	500	50	60	160	25	233	500
Average clock cycles per instruction	1	1	2	1.49	3.34	1.79	1.15
Number of instructions	24	23	48	450	194	43	49
Computation time (μs)	0.048	0.46	1.603	4.190	25.92	0.331	0.113
Maximum sampling frequency (kHz)	20833	2173	623	238	38.5	3021	8823
Normalised time	1	9.58	33.40	87.29	540	6.90	2.35

Table 7.6: Benchmark results of processing speed.

7.5. SUMMARY

and computation time of the validation example. Thus, the assembly code for the selected processors is also introduced.

The benchmark shows that the $\Delta\Sigma$ -CSP is the most power and speed efficient processor. One thing not mentioned in this chapter is that the $\Delta\Sigma$ -CSP is also the most area-efficient, as explained in Chapter 6. The excellent performance of the $\Delta\Sigma$ -CSP is obtained by the fact that it adopts a fixed-point data format and no multipliers are needed for one-bit processing. Because of these features, the $\Delta\Sigma$ -CSP can achieve the same effect as the floating-point processors as well as maintain the fast speed and low power consumption.

It should be emphasized that the benchmarking explicitly compares the various processors for real-time control only. The benefits of the $\Delta\Sigma$ -CSP compared with the others arise from a combination of 1-bit processing and an architecture targeted for this particular application (the latter is true of the CSP), whereas the other processors are capable of a much more greater variety of applications.

Chapter 8

Conclusions and Future Work

This chapter reviews the thesis and concludes. Future research directions are also discussed.

8.1 Conclusions

8.1.1 Review of the thesis

The thesis has described two areas of the research work: one-bit processing for real-time control and customised hardware support for one-bit processing. Chapter 2 reviewed the existing techniques of digital control and approaches to implement digital control applications. Chapter 3 and Chapter 4 brought forth the concept of one-bit processing, in which one-bit data conversion, control forms and sampling criterion were detailed. Chapter 5, 6 and 7 introduced the specific hardware and software designs to implement one-bit processing applications. The hardware architectures were also validated with some control examples. Finally, chapter 8 gave the benchmark results of the resulting $\Delta\Sigma$ -CSP compared with other processors.

8.1.2 Achievements

For the research work, it is expected that modern system-on-chip techniques together with advanced signal processing algorithms can produce more ef-

8.1. CONCLUSIONS

efficient embedded controllers for real-time control applications in terms of area, speed and power. Thus, a new concept of digital control along with its supported hardware and software architectures is developed in this thesis.

Compared to the conventional digital control, a most significant difference of one-bit processing is that no multibit data converters are needed. Instead, a simple encoder that utilises $\Delta\Sigma$ modulation is used to perform analogue to digital conversion. The $\Delta\Sigma$ modulator produces a series of high-frequency one-bit signals that contain all the useful information of the input.

A special canonic control form is explained to take advantage of the one-bit signals. The control form is based on the δ -operator in order to overcome the numerical problems due to coefficient sensitivity which arises with z -form when the sampling frequency is high. A digital $\Delta\Sigma$ modulator is also placed before the resulted control signal is fed back into the controller. This structure makes it possible to remove the multipliers which are inevitable conventionally and became a key factor in circuit designs. Because no multipliers are needed in one-bit processing, it can effectively reduce the circuit complexity, improve the hardware speed and save power when a controller is implemented in application-specific integrated circuits.

From the final benchmark results, it shows that the resulting processor architecture outperforms all commercially available high-speed processors. One-bit processing therefore can significantly accelerate the hardware performance for system-on-chips. It is important to appreciate that, although the $\Delta\Sigma$ -CSP is the most efficient processor architecture compared to other processors by a significant margin, it is much simpler because it adopts a fixed-point data format and no multipliers are needed. Also the low complexity of the processor architecture confers a number of advantages such as low power and reduced cost due to small die size and simpler packaging.

8.1.3 Limitations

Both the proposed signal processing concept and its support have been proven practicable for real-time control, but they are only applicable to the linear time-invariant control applications. It is still unknown whether or not one-

8.2. FUTURE WORK

bit processing is suitable for nonlinear control applications although one-bit processing contains nonlinear components itself.

Chapter 5 has shown a most efficient way to implement one-bit processing applications. With direct implementation, however, one-bit processing somewhat increases the hardware complexity by adding $\Delta\Sigma$ modulators into its control loop. This is not a problem for a single input and single output system because the elimination of multipliers can cover the extra hardware complexity by the $\Delta\Sigma$ modulator. However, for a multi-input and multi-output system, there are many subsystems and more $\Delta\Sigma$ modulators have to be used. Our researches show that it may increase rather than reduce the whole hardware complexity if too many $\Delta\Sigma$ modulators are used in control loops. However there is also a solution that we can reduce the number of $\Delta\Sigma$ modulators by combining the subsystems into some high-order systems. This is not a problem for the processor solutions anyway, although it may require extra memory space to store the program.

The $\Delta\Sigma$ -CSP has been successfully synthesized by targeting a $0.13\mu\text{m}$ process. However, this is only realized with the help of some EDA tools. Although the processor is implemented into a Xilinx FPGA for hardware-in-loop simulations, it has never been tested in its real circuit. The hardware-in-loop simulation provides a way to validate the processor's practicability in a virtual control environment, but the $\Delta\Sigma$ -CSP has not been verified in real control environments due to the lack of real physical systems.

8.2 Future work

This section presents some future work based on one-bit processing and the $\Delta\Sigma$ -CSP. At the same time, some other possible approaches to improve the system-on-chip performance for real-time control will be discussed.

8.2.1 Dual-processor architecture

The $\Delta\Sigma$ -CSP is a dedicated architecture. Its careful numerical formulation ensure that it will perform deterministically in one-bit processing. However,

8.2. FUTURE WORK

it is recognised that other functions such as interlocking are necessary in real-time control, for which the $\Delta\Sigma$ -CSP is not well suited. It is necessary to guarantee all the variety of functions that are required for real-time control. This can be most effectively achieved with a dual-processor architecture, in which the $\Delta\Sigma$ -CSP can be integrated as an extra processing component within the general purpose processor architecture. This will relieve the general purpose processor of performing fixed repetitive functions that can be performed by the $\Delta\Sigma$ -CSP.

8.2.2 Maglev control

A maglev vehicle is currently being built in the lab. Its controller, which is a classically designed active suspension controller, provides the control of the vertical modes of the vehicle (Goodall et al., 1978). The maglev vehicle takes the input which is a air-gap control signal from the controller and the sensors give 4 signals which are air-gap, acceleration, flux and current to be controlled.

The maglev controller has already been implemented in analogue form with 12 inputs and 4 outputs. This is a 46th order controller, which is one of the most dynamically complex control examples. We expect that the controller can be implemented into the $\Delta\Sigma$ -CSP with one-bit processing. This will verify the capability of the $\Delta\Sigma$ -CSP to handle very complex control systems further.

8.2.3 Bit-serial architecture

Bit-serial architectures are distinguished by their communication strategy. Digital signals are transmitted bit sequentially on single wires, as opposed to the simultaneous transmission of words on parallel buses. Figure. 8.1 illustrates the communication strategies of bit-serial architecture and bit-parallel architecture.

A bit-parallel structure processes all of the bits of an input simultaneously at a significant hardware cost. In contrast, a bit-serial structure processes the the input one bit a time, generally using the results of the operations on

Figure 8.1: Bit-serial and bit-parallel communication strategies

the first bits to influence the processing of subsequent bits. The advantage benefitted by the bit-serial design is that all of the bits pass through the same logic, resulting in a significant reduction in the required hardware. Typically, the bit-serial approach requires $(1/n)^{th}$ of the hardware required for the equivalent n -bit parallel design. The price of this logic reduction is that the serial hardware takes n clock cycles to execute, while the equivalent parallel structure executes in one. The time-hardware product, however, for the bit-serial structure is often smaller than for equivalent bit-parallel designs because the logic delays between registers are generally significantly smaller.

The bit-serial architecture applies very simple bit-serial blocks like bit-adder and bit-multiplier for numeric operations (Denyer and Renshaw, 1985). The bit-serial architectures have already been used in implementing digital filters (Andraka, 1993, 1996).

The $\Delta\Sigma$ -CSP doesn't adopt the bit serial architecture. However, one-bit processing can take advantage of the bit-serial architecture to improve its hardware performance. Because the inputs and outputs are one-bit signals, and no multipliers are needed, the bit-serial architecture is much easier to realise for one-bit processing applications.

8.2.4 MEMS and Microsystem engineering

MEMS is an abbreviation for MicroElectroMechanical System, which contains sensing and/or actuating elements. A microsystem contains MEMS components that are designed to perform specific engineering functions (Hsu, 2002). The microsystem is a complete system-on-chip, which contains sensors, signal processing unit and actuators.

Today many microsensors have been developed as MEMS. A special accelerometer that utilises $\Delta\Sigma$ modulation was proposed by Kraft (1997). This makes it possible to implement one-bit processing applications in the microsystems, in which the $\Delta\Sigma$ -CSP can carry out signal processing algorithms. For other microsensors, one-bit processing is still applicable by placing an $\Delta\Sigma$ modulator between the sensor and the signal processing unit. This should build a more efficient microsystem in terms of speed, area and power for real-time control.

Appendix A

General $\Delta\Sigma$ -CSP Program

This appendix shows a general form of a $\Delta\Sigma$ -CSP program.

- RDW P0; //read coefficient P0 from data ROM
- WRW P0; //write coefficient P0 to data RAM
- RDW Q0; //read coefficient Q0 from data ROM
- WRW Q0; //write coefficient Q0 to data RAM
- RDW Q1; //read coefficient Q1 from data ROM
- WRW Q1; //write coefficient Q1 to data RAM
- RDW Q2; //read coefficient Q2 from data ROM
- WRW Q2; //write coefficient Q2 to data RAM
- RDW Q3; //read coefficient Q3 from data ROM
- WRW Q3; //write coefficient Q3 to data RAM
- RDW Q4; //read coefficient Q4 from data ROM
- WRW Q4; //write coefficient Q4 to data RAM
- RDW Q5; //read coefficient Q5 from data ROM
- WRW Q5; //write coefficient Q5 to data RAM
- RDW S; //read scaling factor S from data ROM
- WRW S; //write scaling factor S to data RAM

-
- RDW TIMER; //read timer initial value from data ROM
 - WRW TIMER; //write timer initial value to data RAM
 - RDW PCSTART; //read PC start value from data ROM
 - WRW PCSTART; //write PC start value P0 to data RAM
 - RDW X; //read state initials from data ROM
 - WRW X0; //write state initial 0 to data RAM
 - WRW X1; //write state initial 1 to data RAM
 - WRW X2; //write state initial 2 to data RAM
 - WRW X3; //write state initial 3 to data RAM
 - WRW X4; //write state initial 4 to data RAM
 - WRW X5; //write state initial 5 to data RAM
 - SET TIMER; //set the timer
 - WPC PCSTART; //set the program counter
 - HLT; //wait until sample clock starts
 - CNA IN0 P0; // $acc = in0 * p0 + acc$
 - CNA OUT0 Q0; // $acc = out0 * q0 + acc$
 - SRS S; // $acc = acc >> s$
 - CNA 1 X0; // $acc = x0 + acc$
 - WRW X0; //write x0 to data RAM
 - CNA OUT0 Q1; // $acc = out0 * q1 + acc$
 - SRS S; // $acc = acc >> s$
 - CNA 1 X1; // $acc = x1 + acc$
 - WRW X1; //write x1 to data RAM
 - CNA OUT0 Q2; // $acc = out0 * q2 + acc$
 - SRS S; // $acc = acc >> s$

-
- CNA 1 X2; $//acc = x2 + acc$
 - WRW X2; $//write\ x2\ to\ data\ RAM$
 - CNA OUT0 Q3; $//acc = out0 * q3 + acc$
 - SRS S; $//acc = acc >> s$
 - CNA 1 X3; $//acc = x3 + acc$
 - WRW X3; $//write\ x3\ to\ data\ RAM$
 - CNA OUT0 Q4; $//acc = out0 * q4 + acc$
 - CNA 1 X4; $//acc = x4 + acc$
 - WRW X4; $//write\ x4\ to\ data\ RAM$
 - CNA OUT0 Q5; $//acc = out0 * q5 + acc$
 - CNA 1 X5; $//acc = x5 + acc$
 - WRW X5; $//write\ x5\ to\ data\ RAM$
 - WRB OUT0; $//1-bit\ output$
 - HLT; $//wait\ until\ next\ sample\ clock\ starts$

Appendix B

Publications

Conference Contributions

1. Xiaofeng Wu and Roger Goodall, $\Delta\Sigma$ -Based Control System Processing, in Proc. of PREP 2003, Exeter, UK, April 2003.
2. Xiaofeng Wu and Roger Goodall, One-bit processing for real-time control, in Proc. of European Control Conference 2003, Cambridge, UK, September 2003.
3. Xiaofeng Wu and Roger Goodall, FPGA implementation of 1-bit control system processor, in Proc. of PREP 2004, Hertfordshire, UK, April 2004.
4. Xiaofeng Wu and Roger Goodall, FPGA-based control system processing with $\Delta\Sigma$ modulation, in IEEE WCICA'04 Proceedings, Hangzhou, P.R.China, June 2004.
5. Xiaofeng Wu, Vassilios Chouliaras and Roger Goodall, An application-specific processor hard macro for real-time control, IEEE System-on-chip Conference Proceedings, San Jose, USA, September 2004.

Journal Papers

1. Xiaofeng Wu and Roger Goodall, 1-bit processing for digital control, IEE Proc. Control Theory & Applications, Volume 152, Issue 4, 2005
2. Xiaofeng Wu, Vassilios Chouliaras, Jose Nunez-Yanez and Roger Goodall, A Novel Control System Processor and Its VLSI Implementation, IEEE trans. on VLSI [accepted].

Bibliography

- Andraka, R. J., 1993. FIR filter fits in an FPGA using a bit-serial approach. In: Proc. 3rd PLD Conference.
- Andraka, R. J., 1996. Building a high performance bit serial processor in an FPGA. In: Proc. 1996 On-Chip System Design Conference.
- Angus, J., 1994. One-bit mastering. In: Proc. Managing the Bit-Budget Conference.
- Angus, J., April 1998. One bit digital filtering. In: IEE Colloquium Digest 1998/252.
- Angus, J., Draper, S., 1998. An improved method for directly filtering $\sigma - \delta$ audio signals. In: Proc. AES 104th convention. Amsterdam, The Netherlands.
- Ardalan, S. H., Paulos, J. J., 1987. An analysis of nonlinear behavior in delta-sigma modulators. In: IEEE Transactions on Circuits and Systems. Vol. 34. pp. 593-604.
- Atherton, D., 1982. Nonlinear Control Engineering. Van Nostrand Reinhold Company, London.
- Barr, M., 1999. Programming embedded systems in C and C++. O'Reilly.
- Benabes, P., Keramat, M., Kielbasa, R., June 2000. Synthesis and analysis of sigma-delta modulators employing continuous-time filters. Analog Integrated Circuits and Signal Processing 123 (3), 189-200.
- Berkeley Design Technology, Inc., 2000. Choosing a DSP processor. URL www.bdti.com
- Brogan, W., 1985. Modern Control Theory. Prentice-Hall Inc.
- Burr-brown Corp., 1997. High dynamic range delta-sigma modulator. URL <http://www.burr-brown.com>

BIBLIOGRAPHY

- Cadence, 2004. http://www.cadence.com/products/digital_ic/.
- Cady, F. M., 1997. Microcontrollers and microcomputers, principles of software and hardware engineering. Oxford university press.
- Candy, J. C., 1992. Oversampling delta-sigma data converters: theory, design, and simulation. Institute of Electrical and Electronics Engineers, New York.
- Carr, J. J., 1980. Microcomputer interfacing handbook :A/D and D/A. Tab Books Inc., USA.
- Cattermole, K., 1969. Principles of pulse code modulation. American Elsevier Pub. Co.
- Chao, K., Nadeem, S., Lee, W., Sodini, C., 1990. A higher order topology for interpolative modulators for oversampling a/d conversion. In: IEEE Transactions on Circuits and Systems. Vol. 37. pp. 309-318.
- Clayton, G., 1982. Data converters. The Macmillan Press Ltd.
- Cohen, A., Daubechies, I., Vial, P., 1993. Wavelets and fast wavelet transforms on an interval. Applied and Comput. Harmonic Ana. 1, 54-81.
- Cooling, J., 1991. Software Design for Real-Time Sstems. Chapman & Hall, Lodon.
- Costa, A., Gloria, A. D., Giudici, F., Olivieri, M., Jan. - Feb. 1997. Fuzzy logic microcontroller. IEEE Micro 17 (1), 66-74.
- Cui, S., 2004. Matlab interface to frequency response analyzer(FRA). Master's thesis, Loughborough University.
- Cumplido-Parra, R. A., 2001. On the design and implementation of a control system processor. Ph.D. thesis, Loughborough University.
- Daugherty, K. M., 1994. Analog-to-digital conversion – a practical approach. McGraw-Hill, Inc.
- de Jager, F., 1952. Delta modulation — a method of pcm transmission using the one unit code. Philips Res. Rep. 7, 442-466.
- Denyer, P., Renshaw, D., 1985. VLSI signal processing: A bit-serial approach. Addison-wesley Publishing company.
- Donoho, D., Johnstone, J., 1994. Ideal spatial adaptation via wavelet shrinkage. Biometrika 81, 425-455.

BIBLIOGRAPHY

- Ernst, R., April-June 1998. Codesign of Embedded Systems: Status and Trends. *IEEE Design and Test of Computers*, 45–54.
- Eyre, J., Bier, J., 1999. The evolution of DSP processors. *IEEE signal processing magazine*.
- Feuer, A., Goodwin, G. C., 1996. *Sampling in Digital Signal Processing and Control*. Birkhäuser.
- Forsythe, W., Goodall, R. M., 1991. *Digital control: Fundamentals, theory and practice*. McGraw-Hill, USA.
- Gitau, M. N., 1994. Optimal pwm switching strategy for single-phase AC-DC converters. Ph.D. thesis, Loughborough University.
- Goodall, R., 1990. The delay operator z^{-1} — inappropriate for use in recursive digital filters? *Trans Inst MC* 12 (5), 246–250.
- Goodall, R., 2001. Perspectives on processing for real-time control. *Annual Reviews in Control*. 25, 123–131.
- Goodall, R., 2002. A Roadmap of Control Design Methods. In: *Proc. UKACC 2002*. Sheffield, UK.
- Goodall, R., Brown, D., 1985. High speed digital controllers using an 8-bit microprocessor. *Software and Microsystems* 4, 109–116.
- Goodall, R., Donoghue, B., 1993. Very high sampling rate digital filters using the δ operator. *IEE Proceedings-G* 140 (3).
- Goodall, R., Jones, S., Cumplido-Parra, R., Mitchell, F., Bateman, S., May 1998. A control system processor architecture for complex LTI controllers. In: *Proc. 6th IFAC Workshop AARTC 2000*. Palma de Mallorca, Spain, pp. 167–172.
- Goodall, R., Williams, R., Barwick, R., 1978. Ride quality specification and suspension controller design for a magnetically suspended vehicle. In: *Proceedings. InstMC Symp on Dynamic Analysis of Vehicle Ride and Maneuvering Characteristics*. pp. 79–89.
- Goodwin, G., 1985. Some observations on robust estimation and control. In: *Proc. 7th IFAC Symposium*. York, pp. 851–859.
- Goodwin, G. C., Graebe, S. F., Salgado, M. E., 2001. *Control system design*. Prentice Hall, USA.
- Gray, R. M., May 1987. Oversampled Sigma-Delta Modulation. *IEEE Trans. Commun.* COM-35, 481–489.

BIBLIOGRAPHY

- Holmes, D. G., Lipo, T. A., 2002. Pulse width modulation for power converters :principles and practice. Chichester : Wiley, New York.
- Hsu, T.-R., 2002. MEMS & Microsystems: Design and Manufacture. McGraw-Hill.
- Hu, S., 1994. Automatic control principles. National Defence Industry Publishing, P.R. China.
- Huang, J., Cheng, K., January 2000. A sigma-delta modulation based bist scheme for mixed-signal circuits. In: Proc. Asia and South Pacific Design Conference.
- IEEE, 1985. 754-1985 IEEE Standard for Binary Floating-Point Arithmetic.
- IEEE Design & Test Roundtable, January - March 2000. Hardware-Software Codesign. IEEE Design & Test of Computers , 92-99.
- Infineon Technologies AG, 2000. C167 Derivatives, User's manual.
- Inose, H., Yasuda, Y., 1963. A unity bit coding method by negative feed back. In: Proc. IEEE. Vol. 51. pp. 1524-1535.
- Intel Corp., 2000. StrongARM-110 Microprocessor Data sheet.
- Intel Corp., 2005. www.intel.com.
- Irwin, G. W., February 1998. Computing & control: back to the future. IEE Computing & control engineering journal .
- Johns, D., Lewis, D., February 1991. IIR filtering on Delta-Sigma modulated signals. In: Electronics Letters. Vol. 27.
- Johns, D., Lewis, D., April 1993. Design and analysis of Delta-Sigma based IIR filters. In: IEEE transactions on circuits and systems-II: analog and digital signal processing. Vol. 40.
- Jones, S., Goodall, R., Gooch, M., 6 1998. Targeted processor architectures for high-performance controller implementation. Control Engineering Practice , 867-878.
- Kershaw, S., July 1996. Sigma-Delta Bitstream Processors Analysis and Design. Ph.D. thesis, Kings College London.
- Kershaw, S., Summerfield, S., Sandler, M., Anderson, M., October 1996. Realisation and implementation of a Sigma-Delta bitstream filter. In: IEE Proc.-Circuits Devices Syst. Vol. 143. pp. 267-273.

BIBLIOGRAPHY

- Kraft, M., 1997. CLOSED LOOP DIGITAL ACCELEROMETER EMPLOYING OVERSAMPLING CONVERSION. Ph.D. thesis, Coventry University.
- Kuo, B. C., 1992. Digital Control Systems. Saunders College Publishing.
- Lapsley, P., Bier, J., Shoham, A., Lee, E. A., 1997. DSP processor fundamentals. IEEE press .
- Liu, B., 1971. Effect of finite wordlength on the accuracy of digital filters — a review. In: IEEE Transactions. Circuit Theory. Vol. CT-18. pp. 670–677.
- Memec Design, 2003. Spartan-IIE LC Development Board User's Guide.
- Middleton, R., Goodwin, G., 1990. Digital control and estimation — a unified approach. Prentice Hall, USA.
- Model Technology Incorporated, September 2001. ModelSim SE User's Manual.
- Mohler, R. R., 1973. Bilinear control processes :with applications to engineering, ecology and medicine. Academic Press.
- Mooney III, V. J., Blough, D. M., November-December 2002. A Hardware-Software Real-time Operating System Framework for SoCs. IEEE Design & Test of Computers , 44–51.
- Nise, N. S., 1995. Control Systems Engineering. The Benjamin/Cummings Publishing Company, Inc.
- Norsworthy, S. R., Schreier, R., Themes, G. C., 1997. Delta-Sigma Data Converters. IEEE Press, USA.
- Ong, S., Jozwiak, L., Tiensyrja, K., 1997. Interactive Codesign for Real-time embedded control systems. In: Proc. ISIE97. Portugal, pp. 170–175.
- Opencores, 2004. <http://www.opencore.org>.
- Paraskevopoulos, P., 1996. Digital Control Systems. Prentice Hall, UK.
- Phillips, C. L., Nagle, H. T., 1990. Digital Control System Analysis and Design. Pretice-Hall International, Inc.
- Predko, M., 1999. Title Handbook of microcontrollers. McGraw-Hill.
- Ritchie, G., 1977. Higher order interpolation analog to digital converters. Ph.D. thesis, University of Pennsylvania.

BIBLIOGRAPHY

- Robjohns, H., 1998. One bit at a time. Tech. rep., The Institute of Broadcast Sound.
- Santina, M. S., Stubberud, A. R., Hostetter, G. H., 1994. Digital control system design. Saunders College Publishing.
- Scavone, G. P., 2004. Numerical methods for discrete-time systems.
URL <http://www.music.mcgill.ca/~gary/614/week2/week2.html>
- Schlett, M., August 1998. Trends in embedded-microprocessor design. IEEE Computer .
- Schneider, A. M., Kaneshige, J. T., Groutage., F. D., November 1991. Higher order s -to- z mapping functions and their application in digitizing continuous-time filters. In: IEEE. Vol. 79(11). p. 16611674.
- Shulz, S., Rozenblit, J., Mrva, M., Buchenrieder, K., August 1998. Model-based Codesign. IEEE Computer , 60–67.
- Slomka, F., Dorfel, M., Munzenberger, R., Hofmann, R., April-June 2000. Hardware/Software Codesign and Rapid Prototyping of Embedded Systems. IEEE Design & Test of Computers , 28–38.
- Steele, R., 1975. Delta modulation systems. Pentech Press, London.
- Stewart, B., Pfann, E., September 1998. Adaptive DSP Sigma Delta Algorithms and Architectures for Digital Communications. Tech. rep., University of Strathclyde, Glasgow, UK.
- Summerfield, S., Kershaw, S., Sandler, M., August 1994. Sigma-Delta bit-stream filtering in VLSI. In: The 37th Midwest Symposium on Circuits and Systems.
- Synopsys, 2003. Power Compiler User Guide.
- Synopsys, 2004. http://www.synopsys.com/products/logic/design_compiler.html.
- Synplicity, October 2003. Synplify ASIC and Amplify ASIC User Guide.
- Tewksbury, S. K., Hallock, R. W., July 1978. Oversampled, linear predictive and noise-shaping coders of order $N > 1$. IEEE Transaction on Circuits and Systems CAS-25(7):447.
- Texas Instruments Inc., 2004. www.ti.com.
- Vaccaro, R. J., 1995. Digital control: A state-space approach. McGraw-Hill, USA.

BIBLIOGRAPHY

- Voltech Instruments Ltd, 1991. TF2000 User Manual. Tech. rep., UK.
- Waggner, B., 1995. Pulse code modulation techniques :with applications in communications and data recording. Van Nostrand Reinhold, New York.
- Wanhammer, L., 1999. DSP integrated circuits. Academic Press.
- Wong, P. W., Gray, R. M., 1990. Fir filters with sigma-delta modulation encoding. In: IEEE Transactions on Circuits and Systems. Vol. 38. pp. 979–990.
- Wu, K. C., 1997. Pulse width modulated DC-DC converters. Chapman & Hall, New York.
- Wu, X., Goodall, R., 2003. One-bit processing for real-time control. In: Proc. of European Control Conference'03. UK.
- Wu, X., Goodall, R., 2004. FPGA-based Control System Processing with $\Delta\Sigma$ Modulation. In: Proc. of The 5th World Congress on Intelligent Control and Automation. Vol. 1. China.
- Xilinx Inc., 2003. Spartan-IIE 1.8V FPGA Family: Complete Data Sheet.
- Xilinx Inc., 2005. www.xilinx.com/products/design_resources/design_tool/.