Investigation of Energy-Efficient Hybrid Analog/Digital Approximate Computation in Continuous Time

Ning Guo

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2017

ABSTRACT

Investigation of Energy-Efficient Hybrid Analog/Digital Approximate Computation in Continuous Time

Ning Guo

This work investigates energy-efficient approximate computation for solving differential equations. It extends the analog computing techniques to a new paradigm: continuous-time hybrid computation, where both analog and digital circuits operate in continuous time. In this approach, the time intervals in the digital signals contain important information. Unlike conventional synchronous digital circuits, continuous-time digital signals offer the benefits of adaptive power dissipation and no quantization noise.

Two prototype chips have been fabricated in 65 nm CMOS technology and tested successfully. The first chip is capable of solving nonlinear differential equations up to 4th order, and the second chip scales up to 16th order based on the first chip. Nonlinear functions are generated by a programmable, clockless, continuous-time 8-bit hybrid architecture (ADC+SRAM+DAC). Digitally-assisted calibration is used in all analog/mixed-signal blocks. Compared to the prior art [1], our chips makes possible arbitrary nonlinearities and achieves 16× lower power dissipation, thanks to technology scaling and extensive use of class-AB analog blocks.

Typically, the unit achieves a computational accuracy of about 0.5% to 5% RMS, solution times from a fraction of 1 µs to several hundred µs, and total computational energy from a fraction of 1 nJ to hundreds of nJ, depending on equation details. Very significant advantages are observed in computational speed and energy (over two orders of magnitude and over one order of magnitude, respectively) compared to those obtained with a modern MSP430 microcontroller for the same RMS error.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Prof. Yannis Tsividis, for his invaluable help and support for the past six years at Columbia. Without his guidance and encouragement, this project and dissertation on hybrid computation would not have been possible. Not only did I learn circuit techniques, but I also learned how to learn anything. His elegance and life philosophy will have a lasting influence on me.

I also would like to thank my thesis committee members: Prof. Seok Mingoo, Prof. Simha Sethumadhavan, Prof. Charles Zukowski and Prof. Kyle Mandli, for their time they spent reading my thesis and for their comments and suggestions which helped me improve the thesis.

I would like to thank my family, especially my parents, for their solid supports on all aspects in my life. Their loves created a comfortable environment for me to focus on my Ph.D. work. I am very grateful that I have such great parents and family.

Thanks are due to my colleagues who contributed to the hybrid computing project. I would like to thank my partner, Yipeng Huang, for his solid work on the architecture/software side in the past five years. We made the best team! I thank Tao Mai, Sharvil Patil, Chi Cao, and Chien-Tang Hu for their high-quality work on chip design and testing. The success of

such complicated chips would not have been possible without their help. I also thank Doyun Kim, Josh Kim, Jianxun Zhu, Teng Yang, Yang Xu, Yu Chen, Zhe Cao and all CISL members for valuable discussions.

Finally, I would like to give my sincere thanks to all my friends for their encouragement and support.

Chapter 1

Introduction

1.1 Introduction to analog computers: The history and the machine

Humankind has been inventing computation devices for millennia. For example, the abacus was known and used by Babylonians by 2400 BC. In those early days, computation was used for topics closely related to human or the society's fundamental needs, such as tax allocation.

Mechanical parts and gears were often used in computing devices in the early days. The first differential analyzer, invented by V. Bush at MIT in 1927, had gears and shafts for such basic arithmetic operations as integration and addition. Fig. 1.1 shows a Bush differential analyzer, consisting of many large and complex mechanical parts. It has six integrators and several output tables (close to the windows). This mechanical differential analyzer was developed to solve ordinary differential equations and is considered to be the "ancestor" of

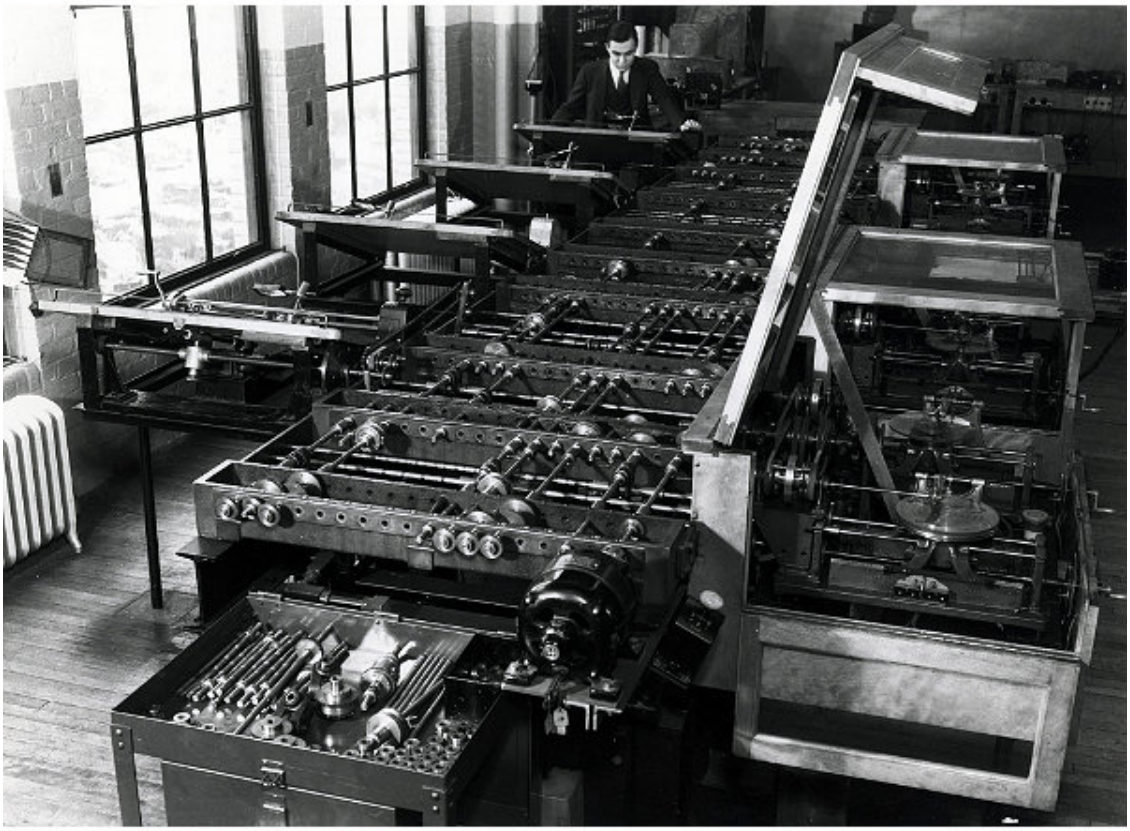

Figure 1.1: Bush differential analyzer. Image courtesy of the Computer History Museum.

the electronic analog computer. It models the target differential equations in a mechanical system, where the configuration of those mechanical parts not only replicates the differential equations, but also offers insights into the physics described by those equations. However, computing power was limited, mainly due to slow operation of the mechanical parts.

Harold Stephen Black's invention of negative feedback electronic amplifiers in 1927 opened the door for computation using electrical signals. With negative feedback, a series of basic mathematical operations can be performed by electronic amplifiers, which are much faster than mechanical devices. This is where the name *operational amplifier* comes from: Elec-

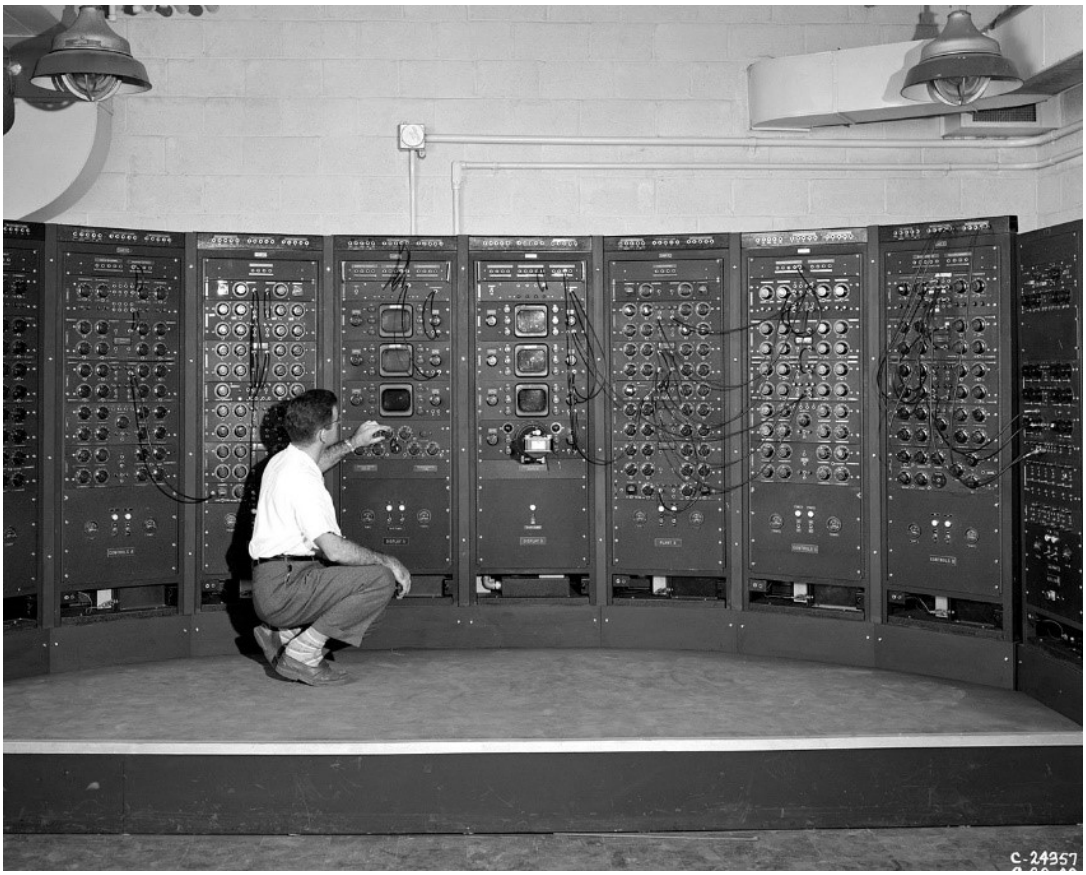

Figure 1.2: An analog computer in 1949 at the Lewis Flight Propulsion Laboratory.

tronic amplifiers were used to perform arithmetic operations in those early days. Voltages were used to represent numerical values in the equation and could be added, integrated, and multiplied. Fig. 1.2 shows an analog computer in use in 1949 for aircraft simulations.

Shannon's 1941 paper [2] is considered to be the first paper discussing the fundamental principles of mathematical models and the computability of a general-purpose analog computer. Bush's differential analyzer was used extensively as an example in Shannon's paper. Integrators, adders and multipliers were considered the fundamental building block for a

general-purpose analog computer. However, the mathematical theory of analog computers barely evolved after Shannon's efforts. In the next 30 years, people focused on building bigger analog computers with more functionality and greater accuracy.

In 1950s and 1960s, analog computers dominated the computing community [3; 4; 5; 6]. They were powerful tools for solving the ordinary and partial differential equations (ODEs and PDEs) widely used to model physical systems. People used analog computers to simulate mathematical models before the actual implementations of machines and plants. The best parameters were found on analog computers, saving considerable time and effort compared the approach of direct implementation (trial and error). For example, analog computers played an important role in the Apollo space program, simulating spacecraft dynamics and guiding control-system design [5].

Starting around 1970, digital computers attracted more attention and competed with analog computers in scientific computation. This was more than two decades after the invention of the Von Neumann architecture in 1945. Both analog and digital computers were big machines at that time. However, the ease of programmability, repeatability and transportation of programs made digital computers more and more popular.

In 1980s, analog computers were almost extinct on the market, signaling the end of the analog-computer industry. All commercial-use computers were digital. Only a few companies still manufactured small analog computers for education purposes, especially for teaching control theory in mechanical engineering. Fig. 1.3 shows a fourth-order analog computer manufactured by Comdyna Inc. It is still in good condition and can solve ODEs with mod-

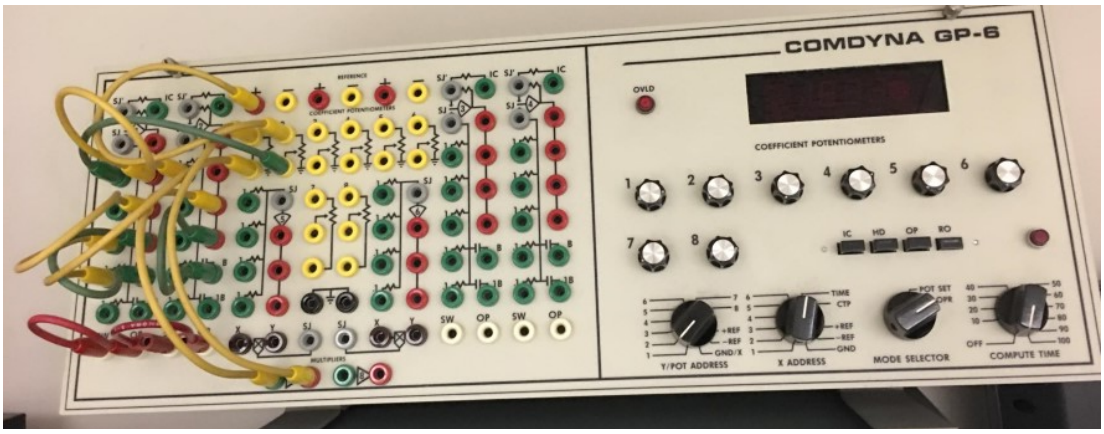

Figure 1.3: Example of an fourth-order analog computer from Comdyna.

erate accuracy (around 10% errors). As shown in Fig. 1.3, the front panel is connected with wires to set up an second-order ODE.

By the 1990s, only one company, Electronic Associates Inc. (EAI), was still manufacturing analog computers and components for NASA's spacecraft simulator. By late 1990s, no records showed any analog-computer manufacturing activity.

We now demonstrate how a differential equation is solved by an analog computer. Fig. 1.4 illustrates a mass–spring damper system on the left; we would like to investigate the motion of the mass. This mechanical system is modeled by the ordinary differential equation shown in the middle. This second-order ODE describes the motion of the mass, with initial conditions imposed. We want to solve this equation on an analog computer.

The block diagram to the right of the equation in Fig. 1.4 illustrates how the computing blocks on an analog computer are configured to solve this equation. In this example, we have used integrators, adders and coefficient-setting blocks. Each integrator's output represents a

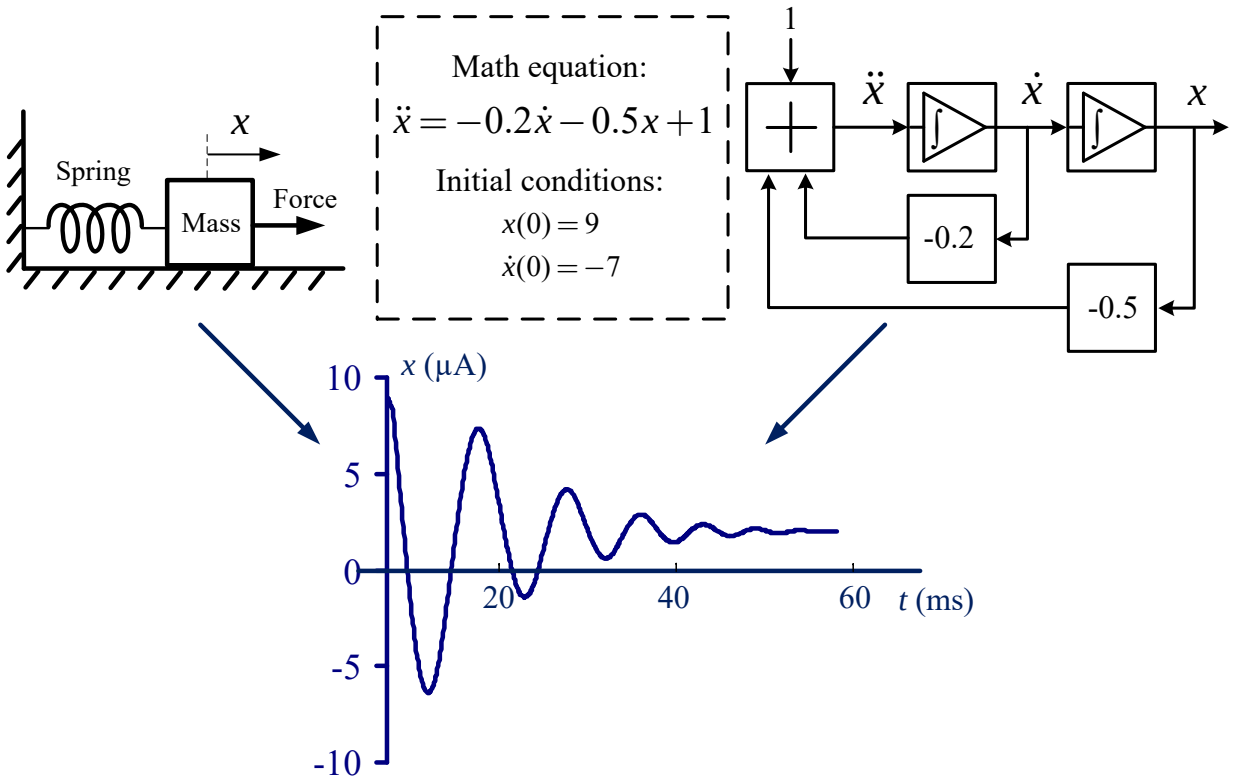

Figure 1.4: Example of how analog computers work.

system state, and the input to that integrator represents the derivative of that state. For this second-order ODE, we use two integrators, whose outputs represent the two state variables x and $\dot{x}$, respectively. Feedbacks are used to close the loop and make the diagram equivalent to the equations. In Fig. 1.4, the x signal is passed through a -0.2 coefficient block and $\dot{x}$ through a -0.5 block. Next, the signal flows are summed together with a constant value at an adder block, whose output is connected to the integrator's input. This closes the loop and equates the left- and right-hand sides of the differential equation in the dashed box. Now the signal paths of this analog computer are characterized by the same equation describing the physical system on the left. You can think of the electrical system as an analogy to the

physical system. That is, we are using this electrical system to reproduce the behavior of another physical phenomenon. After we set the initial conditions on the integrators and let the analog computer run, the transient response of the circuits represents the solution of the differential equation, shown at the bottom of Fig. 1.4. After proper scaling between the electrical values and the distance values, and between computer time and physical time, we arrive at the solution of the original differential equation.

1.2 VLSI analog computers—The first try

Analog computers were mostly abandoned by mid-1970s, long before the dominance of integrated circuits. This means that the old analog computers were only implemented using the technology of the time: discrete components for computing modules, patch boards for interconnections, etc. So analog computers were generally large in size and tedious to program with wires. Since then, the technical community has embraced digital computers and hardly considered how modern technology could impact analog computing techniques. Around 2005, Cowan et al. [7; 1] revisited analog computers in the context of modern VLSI technology and showed that, with advanced VLSI technology, analog computers can be attractive for low-power, self-contained computation and to speed digital computation through coprocessing.

1.3 The need for post-Moore's-law computing

In anticipation of the post-Moores-law era of computing, researchers are looking for either new devices which can replace CMOS transistors, or new methods to harness performance and efficiency from existing silicon technologies. Analog computing has been touted as one approach to address these challenges without the need for novel device technologies.

Analog computing has many alluring properties: broadly, analog computing abandons digital representation of numbers, and also abandons the step-by-step operations typical of modern computing. Much of research in computer architecture is in the line of breaking historical abstractions which hold back the performance and efficiency of computers. Among the remaining abstractions yet to be broken are binary representation and discrete-time operation. In this regard analog computing may unleash untapped uses for existing CMOS technology. More details on the differences between analog and digital computers will be discussed in the next section.

1.4 Fundamental differences between analog computers and digital computers

The operations of analog computers and digital computers are fundamentally different. First, they are different in their number systems. Analog computers use electrical signals for computation, either voltage or current. Though accuracy is limited, these signals are continuous in nature and can be regarded as real numbers. Digital computers, on the other hand, use

discrete binary numbers consisting of zeros and ones. More digits are required to achieve greater accuracy when representing values.

Second, they are different in their basic arithmetic blocks. In analog computers, we have integrators, multipliers, and adders/subtractors as basic building blocks for math operations. These operations exist in the physical (electrical) world. For digital computers, though sometimes we can shift bits to implement multiplication and division by powers of two, adders are the basic arithmetic building blocks for all other operations. All the advanced mathematical operations, such as exponential and logarithmic functions, must be implemented by successive approximations by addition.

Third, they are different in their algorithms. As the example in Fig. 1.4 shows, analog computers use diagram-based algorithms to solve equations. Diagrams with different topologies solve different problems. On the diagrams, all operations are conducted simultaneously. In contrast, digital computers perform operations in a sequence to complete a task. Data are manipulated following strict and specific sequences, whether sorting or solving an equation.

Lastly and most importantly, they are different in handling the time variable in solving ODEs/PDEs. As analog computers use physical phenomena, responses of electrical systems, to do computation, the time variable is kept intact and continuous when solving ordinary and partial differential equations. All operations were carried out simultaneously, often with computation time independent of the problem size, and with no convergence issues as there is no time-discretization. Digital computers, on the other hand, discretize all the values and store them in the memory. The time information is treated just as other numbers. On digital

computers, time-dependent physical problems are solved in a virtualized and discretized time domain.

1.5 Limitations of prior art

Cowan's work in 2005 demonstrated the feasibility of analog computers in VLSI for the first time. Despite its strengths, however, there are several obvious limitations in that fully analog work.

First, the types of mathematical problems that could be solved were limited. Linear differential equations could be implemented on Cowan's chip with analog integrators, adders, and coefficient-setting blocks. However, for the nonlinear differential equations, Cowan's work [7; 1] only studied multiplication between state variables. Therefore, systems of equations involving nonlinear functions like sin() and cos() cannot be implemented on Cowan's chip. These types of equations are used extensively in robotic-dynamics computation. This lack of flexibility in nonlinear functions limits the use and application of Cowan's chip.

Second, most of the analog blocks on Cowan's chip were not calibrated. The integrator was the only block with calibration circuits. Other blocks like multipliers and fanouts have no calibration capabilities. The input and output offsets of these blocks could introduce errors into the solutions, thus decreasing computing accuracy.

Third, the programming interface of that chip was not standardized. Decoder lines and data buses of Cowan's chip were directly exposed to circuitries on PCB. The obvious disadvantage is the increasing complexity of board-level conversion, increasing the size of the

test board and decreasing the configuration and communication speed between the analog computer and the digital host.

Fourth, Cowan's work was limited to big test boards on the test bench in a lab environment, which still gives people the impression of the immobility of analog computers from years ago. Furthermore, while Cowan's work adopted a diagram-based programming style in the Simulink environment, this style is still quite different from the popular and dominant line-by-line text-based coding style, generally considered the fastest way for programming.

1.6 Thesis contributions

In this work, we present a new generation of hybrid (mixed analog/digital) computing chips with full-stack hardware/software codesigns, which greatly solve the issues mentioned earlier. We extend the range of differential equations that can be solved on analog computers by introducing a new arbitrary nonlinear function generator. This generator implements arbitrary nonlinear functions in a table-lookup manner with a programmable, clockless, continuous-time (CT) 8-bit hybrid architecture (ADC + SRAM + DAC). With this new architecture, we thus extend analog computing techniques to a new paradigm: continuous-time hybrid computation, where both analog and digital circuits operate in continuous time. In this new hybrid computing paradigm, the time intervals in the digital signals contain important information, unlike with traditional synchronous or asynchronous digital signals.

At the same time, we implement extensive digitally assisted calibration circuits in all analog and mixed-signal blocks, correcting analog imperfections and improving solution ac-

curacy. We also use a standard SPI interface for our hybrid computing chips to communicate with the digital host for configuration and data acquisition. With the support of our customized API library, we use C++ style codes to program the blocks on our chips, which is a great step toward a user-friendly programming interface.

Additionally, we propose two new FOMs for comparing energy efficiency, one for comparisons among analog computers and one for comparisons between analog and digital computers. We have also built the first mobile hybrid computer board powered solely by a laptop. This is a ground-breaking development as it greatly changes people's impressions of old analog computers' large size and immobility.

Finally, we successfully demonstrate the scalability of our chips and the feasibility of solving nonlinear equations using two interconnected chips.

Chapter 2

Two FOMs for Computation

2.1 FOM for energy efficiency

Back in the 1960s, the number of computing blocks was often used to compare the performance of analog/hybrid computers. For example, the quantity of amplifiers, multipliers, summers, and coefficient attenuators is an important indicator for the "capability" of computers. In addition, the electrical specification of each building block is critical to the overall solution accuracy, as discussed in classical textbooks [3; 6; 8]. Analog computer manufacturers needed to ensure the fabricated components were as good as possible in all electrical specifications for the users' "general purpose" needs.

In a 1970 research report by Benham [9], 12 scientific and engineering problems were investigated to compare the performance of hybrid computers with that of digital computers. The researchers looked at accuracy, solution time, machine cost, human labor, and

setup convenience for the comparison. But little effort has been made to analyze the power consumption of analog/hybrid computers. One reason could be that due to technology limitations, embedded-system applications were not a possibility before the 1970s, so researchers did not analyze power consumption, as functionality was still the main concern.

However, in cyber–physical system applications today, the biggest concern is power and energy dissipation. Less energy consumed by computing devices allows more information to be processed. So, energy-efficiency improvement is now an active and popular research topic.

Work by Cowan [7; 1] was also limited to just the total power dissipation of the whole chip; there was no information on individual blocks' power consumption. So, to compare energy-efficiency between different analog computers, and between analog and digital computers, we now propose two new FOMs.

2.2 $\mathbf{FOM_{computer}}$ for evaluating analog/hybrid computers

The main purpose of analog/hybrid computers is to solve differential equations, especially ODEs. The computer solution time, T_{solution}, is highly problem-dependent and computer-dependent. It can be defined as the time needed for a certain goal to be reached, e.g. a response dies out within a certain margin, or the desired number of cycles in a response is reached. Within this solution time, we want to accomplish the goal with a solution computational energy, E_{solution}, as low as possible. Even for the same equation, different analog computers give different solution times. So we need to find out the connection between E_{solution} and T_{solution}.

The analog computer is a kind of machine that uses diagrams of blocks to do computation. Usually different equations need different numbers of computing blocks in the diagram. The basic formula to compute energy consumption is

$$E_{\text{solution}} = T_{\text{solution}} * P_{\text{equation}} \tag{2.1}$$

where P_{equation} is the power consumption of all the blocks used to map the differential equation. P_{equation} scales approximately with the order of the equation to be solved, n, i.e. the more state variables in the equation, the more integrators and other corresponding blocks are needed to construct the computing diagram. We can express this relation as

$$P_{\text{equation}} = nP_0 \tag{2.2}$$

where P_0 is the typical power dissipation per order of the differential equation.

In a physical problem, let the time interval of interest be T_{physical}. We want to achieve a solution time T_{solution} which should be as small as possible, to avoid wasting energy due to quiescent and leakage currents involved in P_0. We thus need to take full advantage of the analog computer's speed capability. We could scale the time with the following formula

$$T_{\text{solution}} = \frac{T_{\text{physical}}}{\alpha} \tag{2.3}$$

where the time scaling factor α should be as large as possible. This time scaling technique is accomplished by choosing appropriate parameter values in the circuits, usually the overall gain factor of the integrator block [10; 3; 6]. The upper limit on α is bounded by the maximum computing speed, represented by the maximum frequency the computer can handle with

acceptable error, $f_{\text{max,computer}}$. If we want to simulate a physical problem where we expect the highest frequency of interest to be $f_{\text{max,physical}}$, we have the following relationship:

$$\alpha = \frac{f_{\text{max,computer}}}{f_{\text{max,physical}}} \tag{2.4}$$

Plugging (2.2), (2.3) and (2.4) into (2.1), the total energy consumption for a given problem solution will thus be

$$E_{\text{solution}} = T_{\text{solution}} P_{\text{equation}} = T_{\text{physical}} \frac{f_{\text{max,physical}}}{f_{\text{max,computer}}} n P_0 \tag{2.5}$$

We could reformulate (2.5) and introduce a new quantity as the following:

$$E_{\text{solution}} = n T_{\text{physical}} f_{\text{max,physical}} \text{FOM}_{\text{computer}} \tag{2.6}$$

where the quantity

$$\text{FOM}_{\text{computer}} = \frac{P_0}{f_{\text{max,computer}}} \tag{2.7}$$

is a problem-independent FOM, which can be interpreted as the typical energy dissipation of the computer, per problem order, over one period time. The lower the FOM, the better energy-efficiency of a computer.

Basic design tradeoffs [11] show that we can speed up the analog blocks of a computer by increasing P_0, resulting in a proportional increase of $f_{\text{max,computer}}$. However, this method leaves $\text{FOM}_{\text{computer}}$ substantially unaffected(see (2.7)), suggesting that this figure of merit is characteristic of a given technology and a specific architecture. (2.6) shows that power scaling does not affect E_{solution} either. So, in order to achieve better $\text{FOM}_{\text{computer}}$, a lot of effort is required (analog design expertise) to lower the power dissipation of analog blocks

while keeping $f_{\text{max,computer}}$ the same. The above equations and observations will help guide our design.

2.3 FOM_{task} for evaluating computing tasks

$\text{FOM}_{\text{computer}}$ described in Section 2.2 is dedicated to evaluating analog and hybrid computers. To evaluate the performance of different computing architectures for solving the same equation, we must develop a second figure of merit, FOM_{task}.

In general, for a given task, we want both E_{solution} and T_{solution} to be small. This goal is captured by the following FOM:

$$\text{FOM}_{\text{task}} = E_{\text{solution}} T_{\text{solution}}. \tag{2.8}$$

Though this FOM is highly problem dependent, it is an elegant measure that allows us to evaluate different approaches for solving the same equation, with the end of computation defined in the same manner and with the same solution accuracy.

For analog computers, increasing P_0 will result in a proportional increase of $f_{\text{max,computer}}$ and leave $\text{FOM}_{\text{computer}}$ unchanged (see (2.7)) and thus E_{solution} unchanged (see (2.6)). However, this speed-up decreases T_{solution} in proportion, decreasing FOM_{task} in (2.8).

Chapter 3

An Overview of the Complete CT Hybrid Computer System

3.1 Overview of the hybrid computer system presented in this work

Our hybrid computer user environment is illustrated in Fig. 3.1. Our hybrid computer chip communicates with our programming environment through the Arduino Due microcontroller board (Arduino Due is one of the Arduino products, which is based on a 32-bit ARM microcontroller unit; *Due* means *Two* in Italian). We chose an Arduino board for its popularity; more powerful microcontroller boards, even FGPA development boards, could be used here.

Normally, we write codes in the Arduino Integrated Development Environment (IDE) on the laptop and download the codes to the Arduino Due through the USB port. Then the

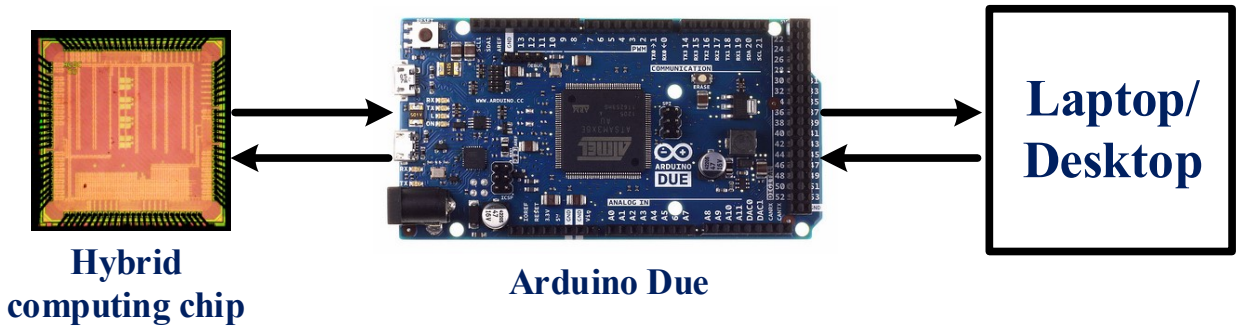

Figure 3.1: Hybrid computer environment.

microcontroller board programs our hybrid computing chip through a standard Serial Peripheral Interface (SPI) with four pins: SS (slave select), CLK (clock signal), MISO (master-in, slave-out), and MOSI (master-out, slave-in). The control board not only provides configurations for the unit, but also performs calibration, controls computation, and reads out analog and digital output values. By taking in external instructions, our on-chip SPI controller can switch our chip between different operating modes, such as signal-path configuration, block parameter setting, computation, etc. A typical workflow of the hybrid computing system for solving differential equations is shown in Fig. 3.2.

Fig. 3.3 shows our hybrid computer's architecture layers. In Arduino's IDE, we use C++ codes to make connections and set parameters for blocks, building the whole diagram to represent the target differential equation. Object-oriented methodology is used to define the behaviors of each type of functional block in our Hybrid Computer API Library [12]. By calling the functions from this software library, we calibrate our hybrid computer, configure signal paths, set computation times and read exception bits.

Upon downloading, each of the above functions is translated into a series of instruction

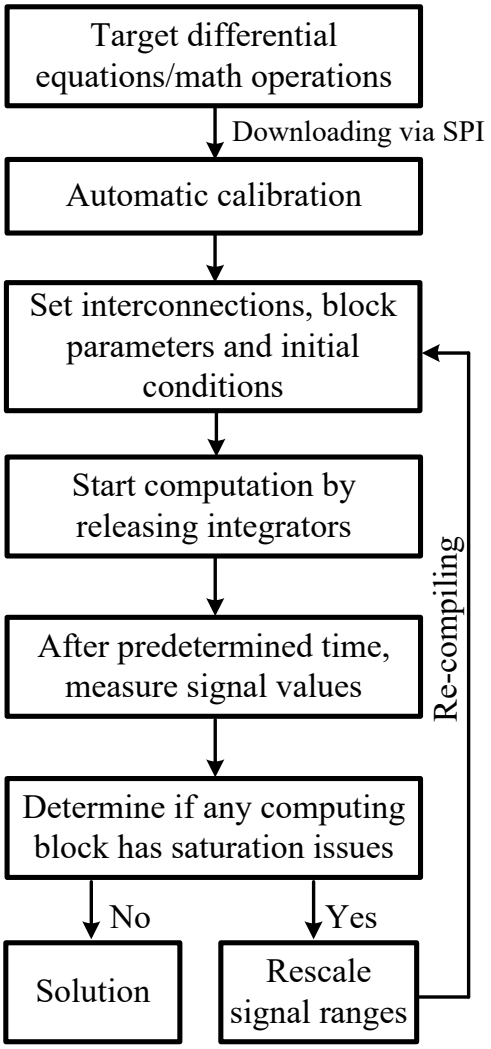

Figure 3.2: Hybrid computing unit workflow.

words with the support of the laptop's native C++ compiler (e.g. gcc on a MacBook). For example, in the 18-bit instruction words used on our first chip, the first six bits indicate the target block and the next four bits indicate the 8-bit register's specific address in that block. The last eight bits carry the information to be configured. Then the instruction machine codes are downloaded into our hybrid computing unit through the SPI interface.

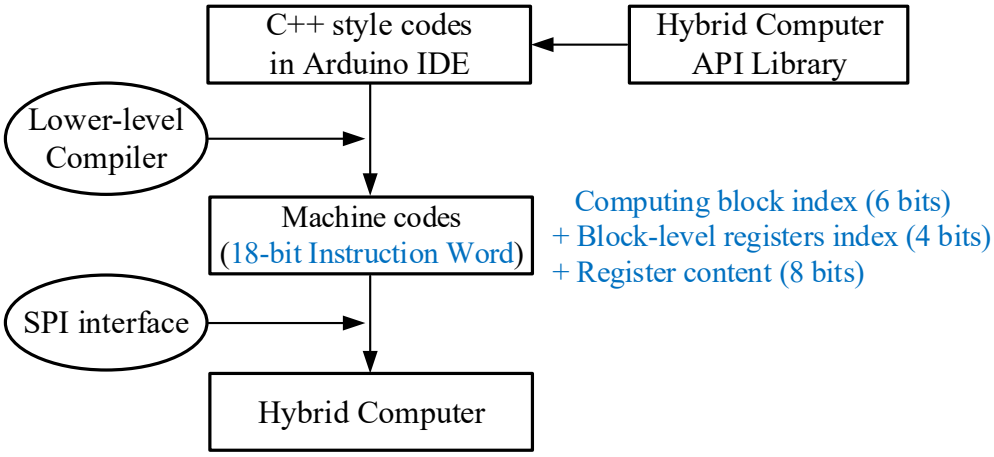

Figure 3.3: Hybrid computing unit architecture layers.

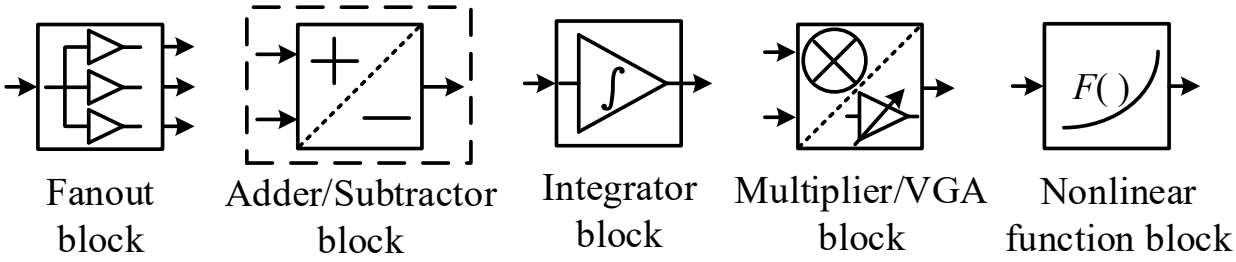

Figure 3.4: Basic mathematical operation blocks.

The basic building blocks used on our chip are shown in Fig. 3.4. As we chose differential currents for signal representation, fanout blocks are used to make copies of the current signals that need to be distributed to several destinations. Addition and subtraction are done by just sending currents to a common node, so a separate adder/subtractor block is not needed on our chip. (Adder and subtractor block are shown in the dashed-box just for illustration purpose). The multiplier block can be configured as a variable-gain amplifier (VGA) for coefficient-setting, by applying a controlled current (generated by a DAC) to one input of the multiplier. In total, we could implement addition, subtraction, integration, multiplication

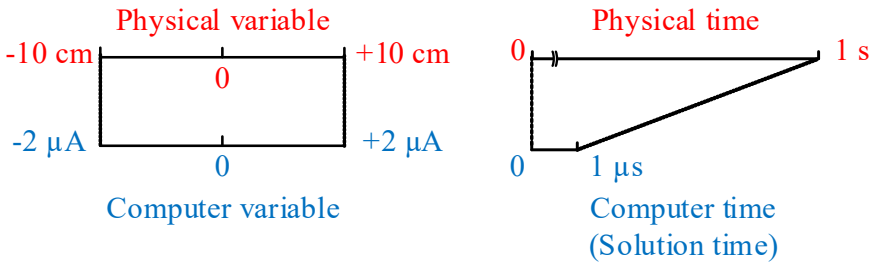

Figure 3.5: Amplitude scaling and time scaling examples.

and nonlinear function generation operations on our hybrid computing chip.

When used to solve ODEs / PDEs, these various blocks are connected in such a way that the resulting system is characterized by the same equations as that describing the physical system under investigation (see Chapter 1); more examples will be seen in Chapter 8. Each integrator's output represents a system state, and the input to that integrator represents the derivative of that state. Following the imposition of initial conditions and the release of the integrators, the transient response of the circuits represents the solution of the equations.

Amplitude scaling and time scaling techniques [3; 4; 5] are necessary on our hybrid computer chip, so that the electrical variables and time are within desired ranges. An illustration example is given in Fig. 3.5. After the solution, both amplitude and time are unscaled to the original problem variables.

3.2 Overall chip architecture and floor plan

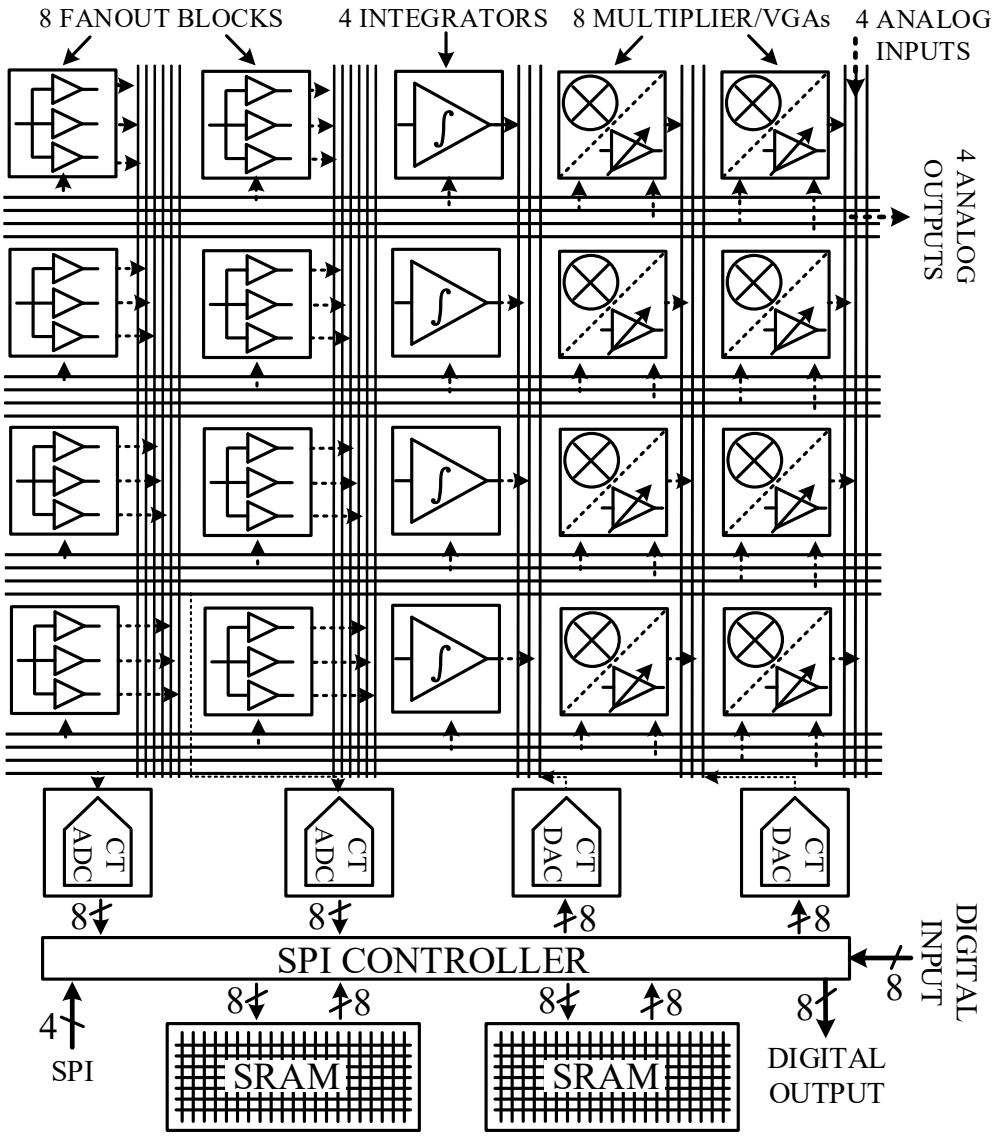

Figure 3.6: The system architecture of the scalable hybrid computing unit.

The architecture of our first test chip is shown in Fig. 3.6. This test chip was designed to include only a sufficient number of blocks to thoroughly test their functions. The chip can solve nonlinear ODEs up to fourth order and can be scaled up to higher orders, described

in Chapter 6. As shown, we adopt the full-crossbar topology from Cowan's work [7; 1] for interconnections between analog blocks. The scalability of our system is similar to FPGAs, offering dense local connectivity and sparse global connectivity.

In order to keep the interference between analog and digital blocks low, the system is organized from top to bottom as rows of analog blocks, mixed-signal blocks and digital blocks(Fig. 3.6). Each individual block is placed in separate deep n-wells to isolate substrate noise. Each block can be connected to any other block through crossbar networks.

Each analog block's input is at the bottom of the block; each input port is connected to a separate global horizontal wire. Each analog block's output is at right side of the block, connected to a separate global vertical wire. At the intersections, transmission gates control the connections between inputs and outputs(Fig. 3.6). Transmission gates' on-off states are stored in local SRAM cells.

The mixed signal blocks on our hybrid computer chip consist of two ADCs and two DACs, both operating in continuous-time (CT) mode. ADC's analog inputs and DAC's analog outputs are connected to the crossbar network above them, shown in Fig. 3.6. The SPI controller and two SRAMs at the bottom are the digital blocks on our chip. The SRAM block serves as a nonlinear function lookup table, which provides 256 lookup data points with eight-bit accuracy. See Chapter 5 for more SRAM details. The digital controller provides an interface to receive instruction commands from the digital host (Arduino Due), through which we can configure all chip parameters. At the same time, the SPI controller also provides configurable data bus for ADCs, DACs and SRAMs interconnections.

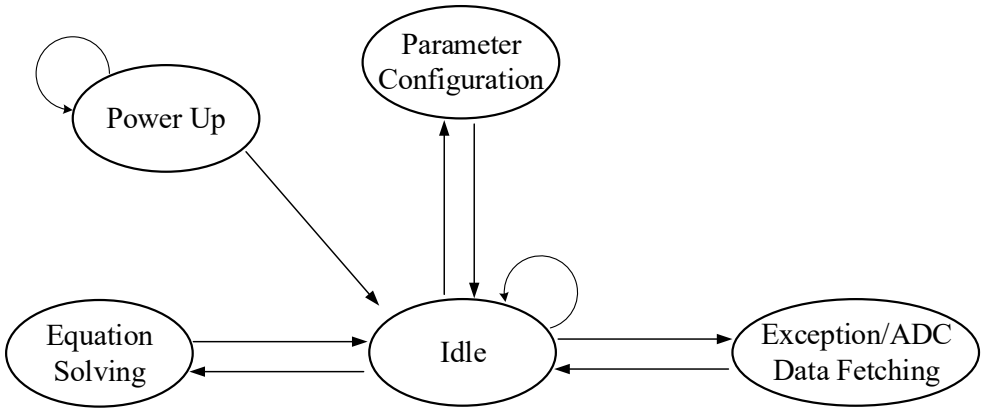

Figure 3.7: The finite-state machine of the SPI controller.

A simplified finite-state machine of the SPI controller is shown in Fig. 3.7. Parameter configuration state includes signal paths configurations, block parameter setting, offset calibrations, etc. In the equation solving state, after setting the initial conditions, the controller releases all integrators into normal integration mode and the whole system starts to compute the equation. In the meantime, the SPI controller is counting the time elapsed, with its internal counter clocked by the external clock signal. This will put all integrators into hold mode after the desired computing period, thus putting the SPI controller into Idle mode again.

Different methods can be used to fetch the solution data from our chip. Currents corresponding to state variables in ODEs can be routed off-chip through the analog input/output ports, or they can be digitized by on-chip CT ADCs and then sent back to the control board through the SPI interface or the parallel eight-bit digital outputs, illustrated in the bottom-right corner of Fig. 3.6.

It is worth mentioning that all non-analog signals and circuits involved in solving differen-

tial equation are continuous-time (CT) digital signals and circuits, previously demonstrated in signal processing applications [13]. This type of digital circuits involve binary signals that are functions of continuous time, whose time details are an integral part of signal representation. CT digital signals carry more information than conventional synchronous and asynchronous digital signals. More importantly, CT digital signals avoid aliasing [13], which introduces considerable error into certain types of computation. To our knowledge, our hybrid computing unit demonstrates for the first time the use of CT digital signals in hybrid computation.

The number of each type of blocks on chip is decided based on the per-order power dissipation P_0 (Chapter 2), expressed as $P_0 = P_{\text{integrator}} + kP_{\text{multiplier/VGA}} + lP_{\text{fanout}} + mP_{\text{nonlinear}}$, where k, l and m are the numbers of multiplier/VGA, fanout, and nonlinear function blocks used, depending on equation details. The adder/subtractor blocks, nodes at which wires are joined together, are not included in P_0, as they do not dissipate any power. For the equations shown in Table 3.1, we obtain the average values of $k = 2.3$, $l = 2.0$ and $m = 0.4$, after a simple linear regression of the blocks needed for each differential equation. We round these values to the nearest integers as $k_0 = 2.0$, $l_0 = 2.0$, and set $m_0 = 0.5$, i.e. the ratio of block quantity is Integrator : Multiplier/VGA : Fanout : Nonlinear block = 1 : 2 : 2 : 0.5. During circuit design, we gave a lot of attention to minimizing the power dissipation for the chosen bandwidth (20 kHz), to keep P_0 low and improve $\text{FOM}_{\text{computer}}$.

No.	ODE's physical background	Order	Integrator	Multiplier/ VGA	Fanout	Nonlinear function
1	Mass-spring damper	2	2	2	2	0
2	Large angle motion of pendulum	2	2	2	2	1
3	Mass-spring dampers with Coulomb friction	2	2	2	2	1
4	Van der Pol oscillator	2	2	2	3	0
5	Two-wheel differential-drive robot	3	3	2	2	2
6	Two coupled nonlinear oscillators	4	4	6	6	2
7	Inverted pendulum	4	4	8	7	2
8	1-D heat equation	4	4	4	8	0

Table 3.1: The number of computing blocks used for eight representative differential equations mapped on our chip.

3.3 Design choices

For a large VLSI system, there are several system-level decisions and choices to be made before implementing individual building blocks.

3.3.1 Voltage mode versus current mode

When using electrical signals for computation, we have two choices, current signals and voltage signals, to encode value information of the equation variables. We need to choose the one that best fits our need.

Old analog computers built with vacuum tubes have voltage signals to represent equation variables, and could have a voltage swing from $-100\,\mathrm{V}$ to $+100\,\mathrm{V}$. However, in the 65 nm CMOS technology we selected for our hybrid computing system, the normal supply

voltage is 1.2 V. The input/output voltage swing would be less than 1.2 V due to headroom requirements.

Current signaling is the other choice for encoding the value information. For our purposes, current-mode signaling is better and preferred for four reasons: 1. For a CMOS device, when we want the drain current signal to vary linearly, V_{GS} only needs to vary approximately by the square root (in strong inversion) or even logarithmically (in weak inversion). Thus, the voltage headroom requirement is looser when we choose current-mode signals for computing and interfacing between blocks, which is very desirable for our 65 nm CMOS technology. Also, as will be seen, the resulting noise is within desired limits. 2. Current-mode signals facilitate the addition and subtraction implementations by merging currents with the proper polarity, eliminating the need to design adder/subtractor blocks. 3. Current-mode multiplication can be easily implemented with well-known translinear circuits, saving considerable design effort. 4. Current-mode signals are a better choice for signal distribution across the whole chip in our case because the voltage swing on the long wires is low, mitigating the effects of potential capacitive interference on some critical paths, such as the bias-voltage and bias-current wires running across the chip subject to capacitive coupling. Considering all the above advantages, differential currents make the most sense for all signal representation and distribution.

DC values are often used as static forcing functions and boundary conditions when solving ODEs and PDEs. In addition, when doing algebraic calculations, we usually manipulate numerical values, which are also fixed DC values. We therefore used DC-coupled interfaces for all analog blocks. To minimize quiescent currents of analog blocks and maintain high

linearity at the same time, we apply Class-AB designs to as many circuits as possible. For example, all analog blocks' input and output stages follow the Class-AB operating principle. Besides, Class-AB design could reduce input offset current and improve SNR when the input swing is small [14]. In conclusion, we use differential currents for signal representation, and DC-coupled, Class-AB interfaces for all analog blocks.

3.3.2 Bandwidth and phase shift

We introduced $\text{FOM}_{\text{computer}}$ in Chapter 2, showing that higher computing bandwidth can be achieved by increasing the power dissipation, but this leaves $\text{FOM}_{\text{computer}}$ unchanged. Since we target applications in low-power cyber–physical systems, we decided to limit our bandwidth to 20 kHz to save power: Analog signals involved in computation are defined from DC to 20 kHz. The analog computing bandwidth can definitely be increased, if we design all blocks accordingly.

Another important reason to use 20 kHz is that we would like to reduce to negligible levels the phase shifts caused by the parasitic capacitances between blocks' interfaces. Delays of computing blocks should be very small compared to the solution time scale of our target differential equations. In other words, when solving differential equations, phase shifts can be ignored in the feedback loops of the computing block diagrams.

3.3.3 Resolution choice

We target 8-bit resolution for our computing unit because of the power constraints and the approximate computing applications we target. 8-bit resolution is a common choice of moderate accuracy. All specifications for each block—e.g., input/output offsets (after calibration), nonlinearity, total harmonic distortion (THD), and signal to noise ratio (SNR)—target 8-bit accuracy.

The trade-offs between accuracy and area/power of analog circuits are highly functionality-dependent, and we only discuss them qualitatively here. Let us assume that the analog circuit errors are dominated by the offset/mismatch and the flicker noise, and that we would like to reduce them. Both mismatch and flicker noise are inversely proportional to area, so analog accuracy could be improved by increasing area. However, this will also increase parasitic capacitors, which may require more power consumption to maintain the bandwidth [11]. Thus, area and power are sacrificed for improving analog circuit resolution.

Chapter 4

Design of Analog and Connectivity Circuits for the Hybrid Computer

4.1 Introduction

In this chapter, we describe the design details of the fanout, integrator, and multiplier/VGA blocks in Fig. 3.6. Circuits for testing and global crossbars are discussed in detail.

4.2 Fanout architecture and circuit design

Fig. 4.1 presents the schematic of the fanout block. It is a pseudo-differential architecture, consisting of two current-mirror branches with each taking one side of the differential inputs. On each side, a current mirror is implemented with the class-AB operating principle, generating three identical outputs. We decided not to include gain-calibration circuits for the

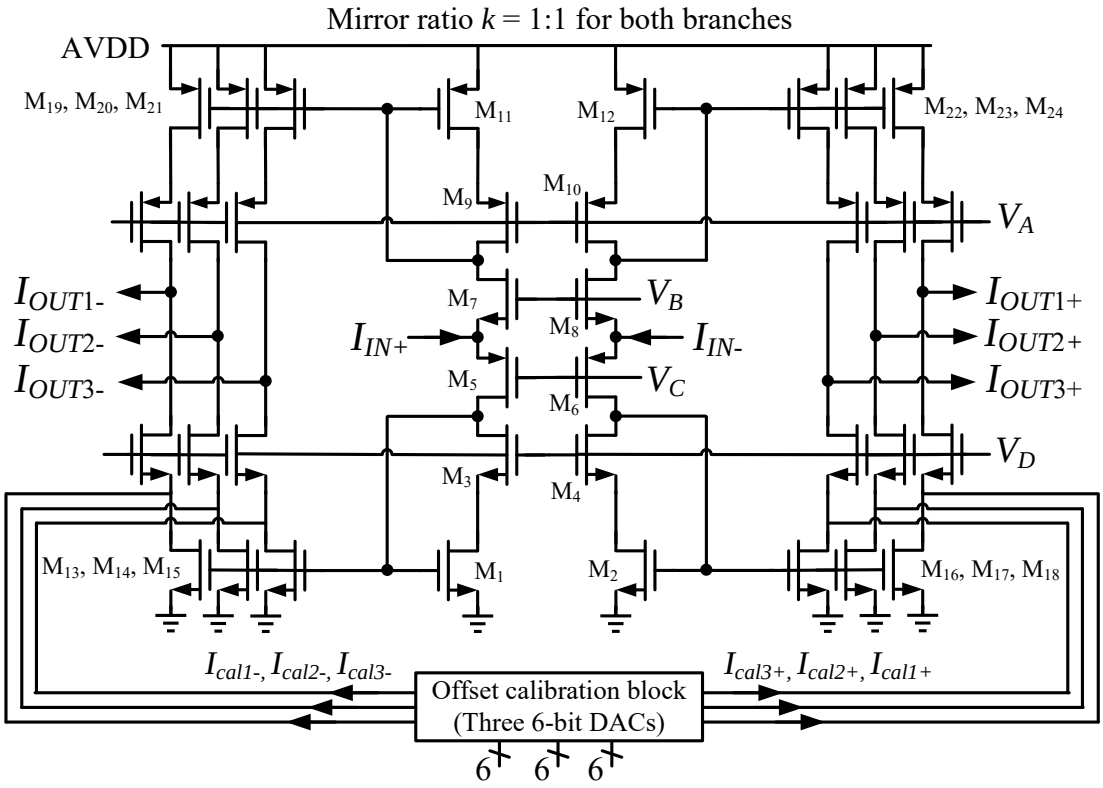

Figure 4.1: Schematic of the fanout block.

mirror devices to save area. Otherwise, there would be a total of 12 devices to be calibrated, M_{13}–M_{24}, which would greatly increase the design complexity and area. Thus, matching between the mirror devices is critical. The mirror devices need to be as large as possible, to achieve as close to 1 : 1 ratio as possible. At the same time, if the mirror devices were too large, they would generate excessive phase shifts. When solving differential equations, we need to keep the phase shift of each block as small as possible to minimize solution errors. We choose the size of $1\,\mu m \times 3\,\mu m$ with multiplicity of three for the mirror devices. The sizes of all the transistors used in Fig. 4.1 are shown in Table 4.1.

Simulations show that the matching errors of the mirror devices are within 0.4% for

Transistors	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{M_3}$	$\mathbf{M_4}$	$\mathbf{M_5}$	$\mathbf{M_6}$	$\mathbf{M_7}$	$\mathbf{M_8}$	$\mathbf{M_9}$	$\mathbf{M_{10}}$
***W/L* (µm/µm)**	1/3	1/3	3/0.5	3/0.5	4/0.2	4/0.2	4/0.2	4/0.2	6/0.5	3/3
Multiplicity	3	3	2	2	3	3	3	3	2	2
V_T type	Low									
Transistors	$\mathbf{M_{11}}$	$\mathbf{M_{12}}$	$\mathbf{M_{13}}$	$\mathbf{M_{14}}$	$\mathbf{M_{15}}$	$\mathbf{M_{16}}$	$\mathbf{M_{17}}$	$\mathbf{M_{18}}$	$\mathbf{M_{19}}$	$\mathbf{M_{20}}$
***W/L* (µm/µm)**	1/3	1/3	1/3	1/3	1/3	1/3	1/3	1/3	1/3	1/3
Multiplicity	3	3	3	3	3	3	3	3	3	3
V_T type	Low									
Transistors	$\mathbf{M_{21}}$	$\mathbf{M_{22}}$	**N/A**							
***W/L* (µm/µm)**	1/3	1/3	N/A							
Multiplicity	3	3	N/A							
V_T type	Low		N/A							

Table 4.1: Transistor sizes of the fanout block.

two-sigma Monte Carlo analysis, the phase shift at 20 kHz is 0.11 degrees, and the *THD* is -77.5 dB for the full scale (2 µA peak, differential).

The DC offsets of the outputs are calibrated to be within one half LSB current of each other (nominally, 1 LSB current = 7.8125 nA) by injecting differential currents into the drains of the NMOS mirror devices, as shown in Fig. 4.1. The calibration block consists of three six-bit unary current-steering DACs, whose schematic is shown in Fig. 4.2. Sixty-four identical PMOS transistors (including one dummy device, M_5 and M_6) divide the full-scale current, $I_{\rm FS}$, equally. The 6-bit binary inputs are decoded into 63-bit thermometer codes $\rm EN_1$ through $\rm EN_{63}$ (and their complements $\overline{\rm EN}_1$ through $\overline{\rm EN}_{63}$). As the 6-bit inputs change from 000 000 to 111 111, $I_{\rm OUT}$ changes from $\frac{63}{64}I_{\rm FS}$ to 0, and $I_{\rm OUT-}$ changes from $\frac{1}{64}I_{\rm FS}$ to $I_{\rm FS}$ correspondingly. The NMOS current sources beneath the current-steering DAC shift its outputs $I_{\rm OUT+}$ and $I_{\rm OUT-}$ by $0.5I_{\rm FS}$. Therefore, as $I_{\rm CAL+}$ changes from $+\frac{31}{64}I_{\rm FS}$ to $-0.5I_{\rm FS}$,

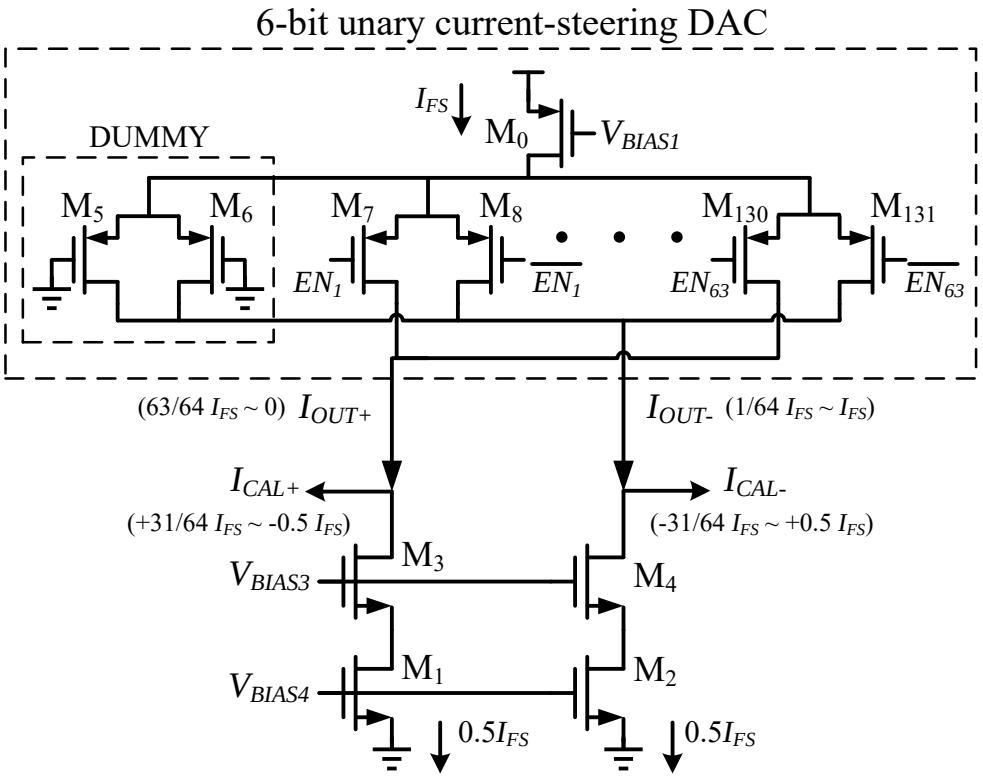

Figure 4.2: Schematic of the fanout block calibration DAC.

$I_{\mathrm{CAL}-}$ changes from $-\frac{31}{64}I_{\mathrm{FS}}$ to $+0.5I_{\mathrm{FS}}$ correspondingly. After this conversion, the calibration DAC injects differential push–pull currents into the outputs (see Fig. 4.1) to cancel offsets.

This 6-bit calibration DAC is an important building block on our chip. It will be used many times in other analog blocks and the CT ADC block for calibration. The transistor sizes used for this calibration DAC are shown in Table 4.2.

Transistors	M_0	M_1	M_2	M_3	M_4	$M_5 - M_{131}$
W/L (µm/µm)	0.73/3	1/8	1/8	3/0.06	3/0.06	0.15/0.06
Multiplicity	1	1	1	1	1	1
V_T type	Normal	Low				Normal

Table 4.2: Transistor sizes of the six-bit calibration DAC.

4.3 Integrator design

The analog integrator is the most important building block in any analog or hybrid computer system. The analog integrator introduces the "dynamics" into the system. The integrator's output holds the state variable, and its input represents the derivative of that state variable. Compared to other building blocks, analog integrators provide temporary *memory.* That is, all past information is integrated (recorded) and reflected in the integrator's present output value.

4.3.1 Error analysis of integrator finite DC gain and limited bandwidth on ODE solutions

It is important to analyze how the integrator introduces errors into differential-equation solutions. Most classical analog-computer textbooks provide only approximate and limited error analysis. In [6], for example, while low-frequency errors introduced by the integrator's finite DC gain were analyzed for the solving of linear differential equations, high-frequency errors introduced by limited bandwidth were not mentioned. Other papers give error analyses of the integrator as a standalone block, but not in the context of solving differential equations.

For example, [15] provides a detailed analysis of how the integrator's offset, finite DC gain and limited bandwidth introduce temporal errors (i.e., errors in the time domain), and shows how the integrator's output will give smaller amplitude and show a time lag due to the finite bandwidth and DC gain when integrating a constant signal.

Our analysis focuses on analytic methods and formulas to calculate ODE solution errors caused by the finite DC gain and limited bandwidth of the integrator used in our hybrid computer. We will ignore the integrator's input offsets because they can be modeled as "forcing functions" in the equations. We will introduce our analysis by means of a specific example.

Suppose we would like to solve

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -0.1\frac{\mathrm{d}x}{\mathrm{d}t} - x(t) \tag{4.1}$$

with initial conditions: $x(0) = 1$ and $x'(0) = 0$. On analog computers, we usually solve ODEs with time-scaling techniques using integrators designed with unity-gain frequency of ω_c. The ideal transfer function of an integrator (the Laplace transform of its impulse response) with unity-gain frequency of ω_c can be expressed as

$$H_{\text{ideal}}(s) = \frac{\omega_c}{s}, \tag{4.2}$$

where s is the complex frequency variable.

The overall gain value, ω_c, of the integrator in (4.2) is often chosen as the scaling factor for analog computers [3; 6; 8]. With the above integrators (with unity-gain frequency ω_c), we could solve the original ODE using the analog computing diagram in Fig. 4.3.

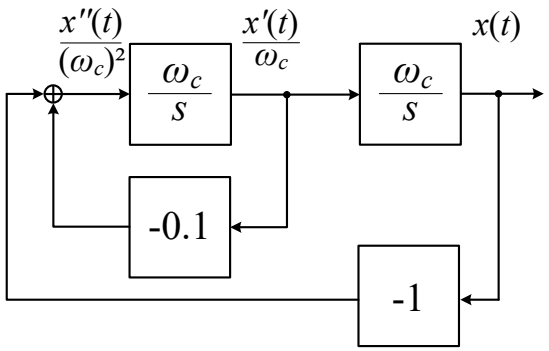

Figure 4.3: An analog computing diagram using ideal integrators with unity-gain frequency ω_c to solve the ODE in (4.1).

Now, based on the diagram in Fig. 4.3, the new differential equation becomes

$$\frac{1}{\omega_c^2}\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = \frac{-0.1}{\omega_c}\frac{\mathrm{d}x}{\mathrm{d}t} - x(t). \tag{4.3}$$

The characteristic equation of (4.3) is $s^2 + 0.1\omega_c s + \omega_c^2 = 0$. In our analysis, we choose $\omega_c = 2\pi \cdot 20 \times 10^3\,\mathrm{rad/s}$ for ω_c. Solving (4.3), we get two eigenvalues:

$$s_1 = -0.0628 \times 10^5 + 1.2551 \times 10^5 i, \tag{4.4}$$

$$s_2 = -0.0628 \times 10^5 - 1.2551 \times 10^5 i. \tag{4.5}$$

We then construct the general solution of (4.3), which is the linear combination of the corresponding solutions:

$$x(t) = c_1 e^{-0.0628\times10^5 t}\cos(1.2551 \times 10^5 t) + c_2 e^{-0.0628\times10^5 t}\sin(1.2551 \times 10^5 t). \tag{4.6}$$

Using the initial conditions, $x(0) = 1$ and $x'(0) = 0$, we can solve for c_1 and c_2 to get the final solution:

$$x(t) = e^{-0.0628\times10^5 t}\cos(1.2551 \times 10^5 t) + 0.05 e^{-0.0628\times10^5 t}\sin(1.2551 \times 10^5 t). \tag{4.7}$$

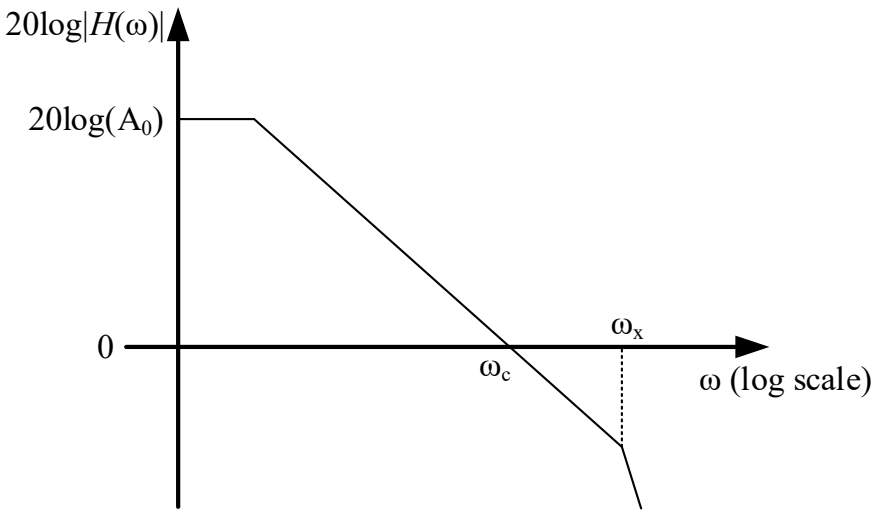

Figure 4.4: The transfer function of an integrator with finite DC gain A_0, unity-gain frequency ω_c and high-frequency pole ω_x.

However, the analog integrator used in our hybrid computer, designed with unity-gain frequency ω_c, has a finite DC gain (A_0) and a high-frequency pole ω_x due to nonidealities. Their values are $\omega_x = 2\pi \cdot 3.2 \times 10^6$ rad/s and $A_0 = 6000$.

Now, the integrator transfer function can be modeled as

$$\widehat{H}(s) = \frac{A_0}{\left(1 + \frac{s}{\omega_c/A_0}\right)\left(1 + \frac{s}{\omega_x}\right)}. \tag{4.8}$$

The corresponding Bode plot of (4.8) is shown in Fig. 4.4. We can rewrite (4.8) into the following form, which is similar to the transfer function (4.2) of an ideal integrator with unity-gain frequency ω_c:

$$\widehat{H}(s) = \frac{\omega_c}{\left(s + \frac{\omega_c}{A_0}\right)\left(\frac{s}{\omega_x} + 1\right)} = \frac{\omega_c}{\frac{1}{\omega_x}s^2 + \left(1 + \frac{\omega_c}{A_0\omega_x}\right)s + \frac{\omega_c}{A_0}} = \frac{\omega_c}{\widehat{s}}, \tag{4.9}$$

where

$$\widehat{s} = \frac{1}{\omega_x}s^2 + \left(1 + \frac{\omega_c}{A_0\omega_x}\right)s + \frac{\omega_c}{A_0}. \tag{4.10}$$

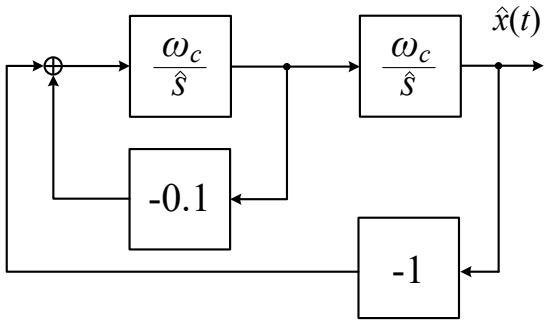

Figure 4.5: The diagram using nonideal integrators with unity-gain frequency ω_c.

We use (4.9) as the transfer function of a "new" integrator. When we use this new integrator for solving the second-order ODE in (4.1), we have the diagram in Fig. 4.5.

We use $\widehat{s}$ as we would use an ideal s-domain operator. The characteristic equation of Fig. 4.5, found by setting loop gain to 1, is $\widehat{s}^2 + 0.1\omega_c\widehat{s} + \omega_c^2 = 0$. Solving this, we get two eigenvalues for the second-order ODE:

$$\widehat{s}_1 = -0.0628 \times 10^5 + 1.2551 \times 10^5 i, \tag{4.11}$$

$$\widehat{s}_2 = -0.0628 \times 10^5 - 1.2551 \times 10^5 i. \tag{4.12}$$

As already shown in (4.10), $\widehat{s}$ is a second-order polynomial of the "true" s operator. To express the general solution of the ODE with independent variable t, we need to have the eigenvalues in the "true" s domain. So we plug the values in (4.11) and (4.12) back into the $\widehat{s}$ expression (4.10) and solve for the equation's "true" eigenvalues. Each $\widehat{s}$ value would give

two roots in the s domain, so we have four roots in total.

$$s_3 = -201.01 \times 10^5 - 1.2558 \times 10^5 i, \tag{4.13}$$

$$s_4 = -201.01 \times 10^5 + 1.2558 \times 10^5 i, \tag{4.14}$$

$$s_5 = -0.0552 \times 10^5 + 1.2558 \times 10^5 i, \tag{4.15}$$

$$s_6 = -0.0552 \times 10^5 - 1.2558 \times 10^5 i \tag{4.16}$$

Using above four eigenvalues, we then construct the general solution with independent variable t:

$$\begin{aligned}\widehat{x}(t) = \widehat{c}_1 e^{-201.01\times 10^5 t}\cos(1.2558 \times 10^5 t) + \widehat{c}_2 e^{-201.01\times 10^5 t}\sin(1.2558 \times 10^5 t) \\ + \widehat{c}_3 e^{-0.0552\times 10^5 t}\cos(1.2558 \times 10^5 t) + \widehat{c}_4 e^{-0.0552\times 10^5 t}\sin(1.2558 \times 10^5 t).\end{aligned} \tag{4.17}$$

To solve for the four coefficients $\widehat{c}_1$, $\widehat{c}_2$, $\widehat{c}_3$, and $\widehat{c}_4$, we need four initial conditions. However, we have only two available, $x(0) = 1$ and $x'(0) = 0$, and there are no other ways to obtain another two new initial conditions. Thus, it is infeasible to solve for $\widehat{c}_1$, $\widehat{c}_2$, $\widehat{c}_3$, and $\widehat{c}_4$ in (4.17).

Instead, we could implement nonideal integrator models with the same transfer function (4.9) in Simulink, build the diagram shown in Fig. 4.5 and set the two nonideal integrators' output values with the two initial conditions we have. That is, we set the integrator output that represents $\widehat{x}$ to 1 and the other integrator output to 0 (see Fig. 4.5).

Before we show the simulation results, we could take a look at the four eigenvalues in (4.13)–(4.16), and compare them with the original two eigenvalues (4.4) and (4.5) from the equation diagram implemented by ideal integrators. Due to integrator nonidealities, we now

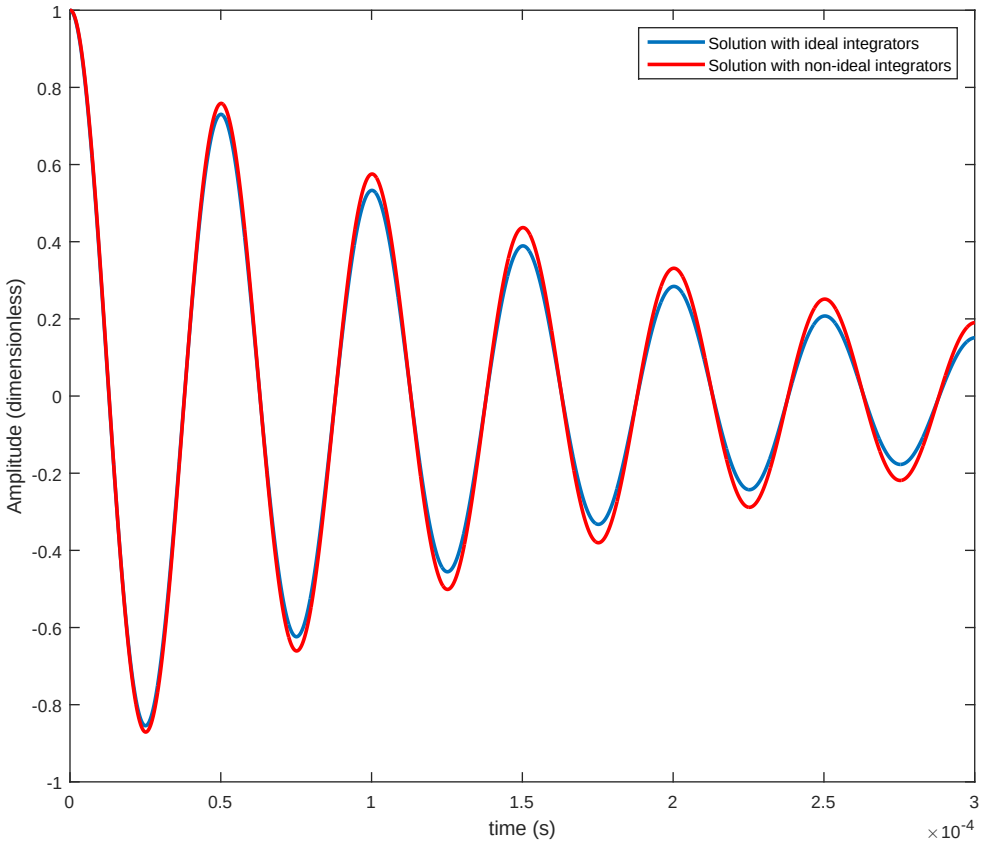

Figure 4.6: Solutions of (4.3) with nonideal integrators (red) and with ideal integrators (blue) from Simulink.

have two dominant poles s_5 and s_6 located very close to the original dominant poles s_1 and s_2; two other nondominant poles s_3 and s_4 are located far away to the left of the dominant poles. Ignoring the effects of nondominant poles and comparing the dominant poles s_5 and s_6 with the original poles s_1 and s_2, we would expect the equation solution using nonideal integrators to have a more slowly decaying envelope and a very slight frequency increase.

Fig. 4.6 shows the equation solution with nonideal integrators (red) from Simulink together with solution using ideal integrators (blue). As expected, the red curve decays more

slowly and the oscillation frequency difference is unobservable. The solution RMS error introduced by nonideal integrators is 1.34% relative to full scale.

4.3.2 Integrator architecture and circuit design details

Cowan's work [1] used a log-domain integrator architecture. That log-domain integrator architecture was not adopted here because its DC gain depends on how well two integrating currents cancel each other and integration would be lossy if the matching is bad. Large devices can not be used either due to bandwidth requirement. In addition, device mismatches inside the log-domain integrator causes the low-frequency pole to be in the right-half plane, causing stability issues when solving ODEs.

The integrator used in our hybrid computing unit is much less sensitive to device matching and its architecture is shown in Fig. 4.7 [16].The main signal paths are highlighted in bold. Starting from the left, the input differential current is mirrored by the class-AB current mirrors at the input stage, and then driven onto the integration capacitor in the second stage. The voltage across the capacitor, an integral of the input current, is then converted to a differential output current ($I_{\mathrm{OUT}+} - I_{\mathrm{OUT}-}$) by the output transconductor block, allowing interfacing with other current-mode analog blocks. The common mode feedback block in the integration stage maintains the capacitor's common-mode voltage. The initial condition on the capacitor is set by an eight-bit DAC, as shown in the initial condition setting block. The eight-bit DAC used here is the same current-steering CT DAC block discussed later in this chapter.

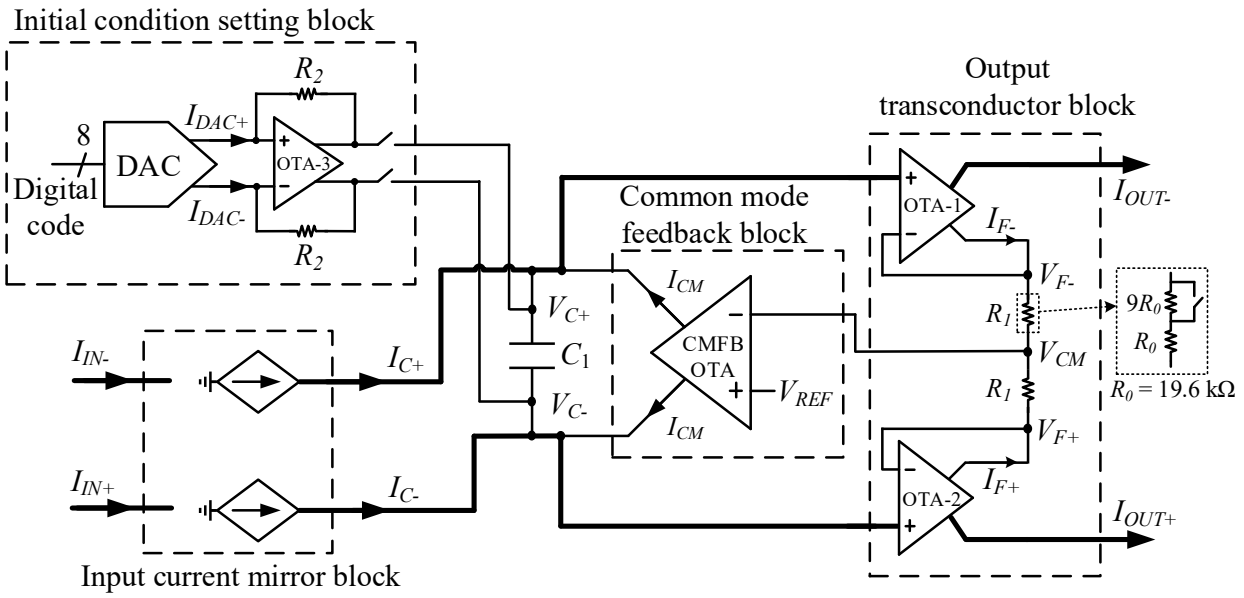

Figure 4.7: Integrator architecture.

A simplified schematic for the input current mirror is shown in Fig. 4.8. We chose a Class-AB topology to keep the quiescent current low and save power. The mirroring ratio k_1, can be configured as 1 or 0.1 by adjusting the width of the mirror devices by closing or opening switches. The states of the switches are stored by the local registers. Gain-boosting amplifiers are added to the cascade devices [17] to increase the low-frequency output impedance of the current mirror, reducing loss during the integration operation. These boosting amplifiers are single-stage, one-transistor amplifiers (Fig. 4.9). The sizes of transistors used in Fig. 4.8 are shown in Table 4.3.

A six-bit calibration current DAC is used to calibrate the output offsets of the input current mirrors. This calibration DAC is the same one used inside the fanout block (see Fig. 4.2). One calibration current ($I_{\text{CAL}+}$) is injected into the drain of M_2 inside one input current

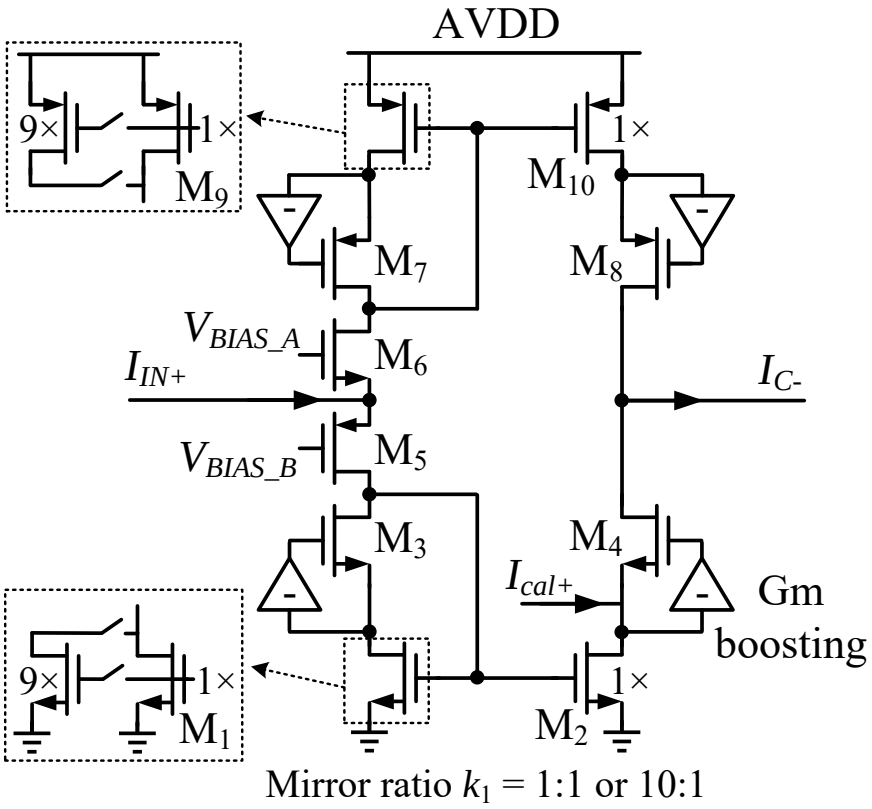

Figure 4.8: Simplified schematic of the input current mirror.

mirror, as shown in Fig. 4.8; the other calibration current ($I_{\mathrm{CAL}-}$) is injected into the other current mirror in Fig. 4.7.

The output transconductor block of the integrator have two OTAs (OTA-1 and OTA-2 in Fig. 4.7), which constitute the pseudodifferential output currents. Our design is based on the work in [18] and a schematic of the OTA is shown in Fig. 4.10 (switches used for testing purposes are not shown). Table 4.4 shows the transistor sizes used in our design. Devices M_9 and M_{10} act as resistors; this configuration does not have an optimum power supply rejection, but the use of a clean AVDD reduces the importance of this problem. The input differential

Transistors	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
W/L (µm/µm)	9/3	9/3	6/1	6/1	9/0.8	9/0.8	6/1	6/1	12/2	12/2
Multiplicity	1	1	1	1	1	1	1	1	1	1
V_T type	Low		Normal		Low		Normal			

Table 4.3: Transistor sizes of the input current mirror.

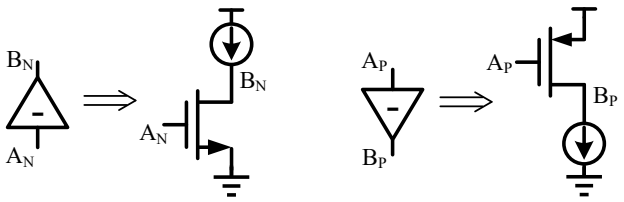

Figure 4.9: Schematics of the gain boosting amplifiers for NMOS and PMOS devices.

voltage is converted into a pair of differential currents charging the MOS resistors M_9 and M_{10}. The differential voltage across M_9 and M_{10} is converted into two identical current outputs $I_{\rm OUT}$ and $I_{\rm F}$. $I_{\rm OUT}$ is used as the integrator output, and $I_{\rm F}$ is used to drive the load resistor R_1 in the common-mode feedback path (Fig. 4.7). The negative feedback path of the OTA in Fig. 4.7 at the output stage forces the differential voltage $(V_{\rm F+} - V_{\rm F-})$ across the resistors to follow the voltage across the integration capacitor C_1 $(V_{\rm C+} - V_{\rm C-})$. The differential current going through the resistors R_1 is $I_{\rm F+} - I_{\rm F-} = (V_{\rm C+} - V_{\rm C-})/R_1$. The output current $(I_{\rm OUT+} - I_{\rm OUT-})$, which is a copy of the current $(I_{\rm F+} - I_{\rm F-})$, is proportional to the voltage across the integration capacitor $(V_{\rm C+} - V_{\rm C-})$, i.e. $I_{\rm OUT+} - I_{\rm OUT-} = I_{\rm F+} - I_{\rm F-} = (V_{\rm C+} - V_{\rm C-})/R_1$. The gain of the output transconductor block is settable by changing the value of the load resistor R_1.

In order to calibrate the integrator output offset, we again use the six-bit calibration

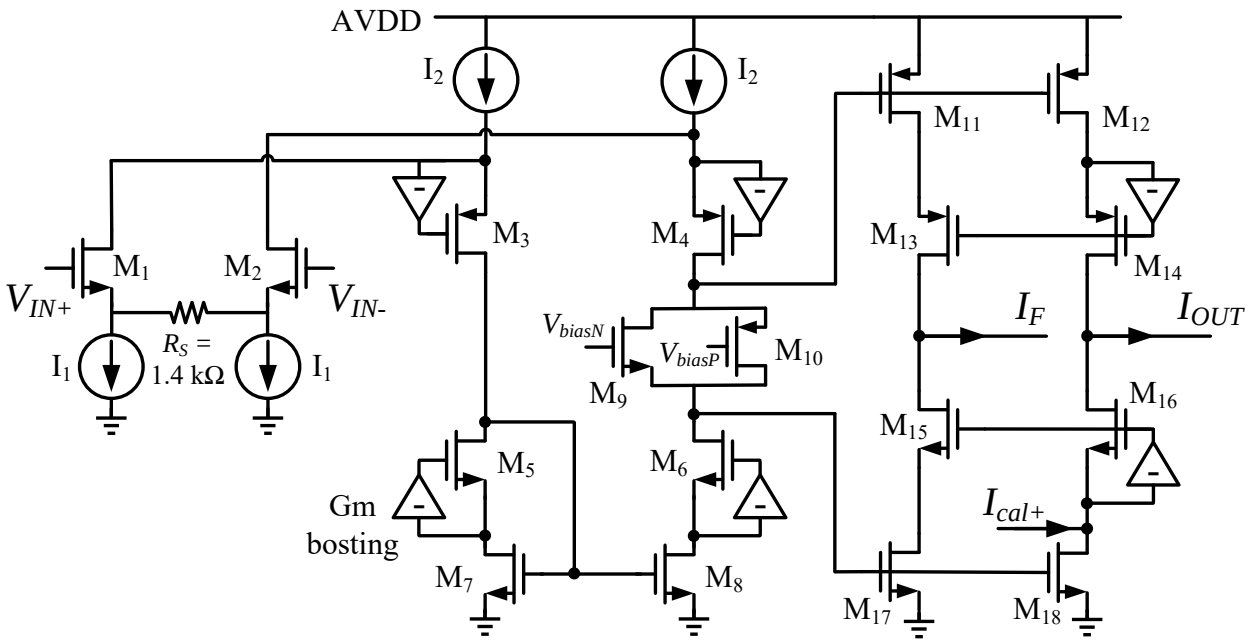

Figure 4.10: Schematic of the current copying OTA.

current DAC (see Fig. 4.2) at the output stage. One calibration current ($I_{\mathrm{CAL}+}$) is injected into the drain of M_{18} of OTA-1, as shown in Fig. 4.10 and the other ($I_{\mathrm{CAL}-}$, not shown) is injected into OTA-2.

The CMFB block in Fig. 4.7 is an OTA with two matching output currents (I_{CM}), similar to the one used in the output transconductor block. The CMFB block maintains the capacitor's common-mode voltage with respect to ground by injecting equal currents to both terminals of the capacitor. The common-mode component of the input current of the integrator is therefore absorbed by the OTA inside the CMFB block.

The I/O relationship of the integrator, assuming zero initial condition, is as follows:

$$I_{\mathrm{OUT}+} - I_{\mathrm{OUT}-} = \omega_1 \int_0^{t_{\mathrm{stop}}} (I_{\mathrm{IN}+} - I_{\mathrm{IN}-})\,\mathrm{d}t \tag{4.18}$$

where $\omega_1 = 2\pi f_1 = k_1 \frac{1}{R_1 C_1}$ is the unity gain frequency of the complete integrator, including

Transistors	**M_1**	**M_2**	**M_3**	**M_4**	**M_5**	**M_6**	**M_7**	**M_8**	**M_9**	**M_{10}**
***W/L* (µm/µm)**	12/0.5	12/0.5	4/0.5	4/0.5	4/0.5	4/0.5	6/2	6/2	4/0.5	4/0.5
Multiplicity	1	1	1	1	1	1	1	1	4	4
V_T type	Low		Normal						Low	
Transistors	**M_{11}**	**M_{12}**	**M_{13}**	**M_{14}**	**M_{15}**	**M_{16}**	**M_{17}**	**M_{18}**	**N/A**	
***W/L* (µm/µm)**	4/0.8	4/0.8	12/0.4	12/0.4	12/0.5	12/0.5	4/1	4/1	N/A	
Multiplicity	8	8	1	1	1	1	8	8	N/A	
V_T type	Normal								N/A	

Table 4.4: Transistor sizes of the current copying OTA.

input-scaling factor k_1. The unity-gain frequency f_1 is taken as $f_{\max,\text{computer}}$ as described in Chapter 2, and ω_1 sets the time scaling factor α [3; 6; 8]. We use $C_1 = 40.6\,\text{pF}$; R_1 is selectable as either $19.6\,\text{k}\Omega$ or $196\,\text{k}\Omega$; and k_1 is selectable as either 0.1 or 1. This allows the selection of f_1 as 2 kHz, 20 kHz, or 200 kHz.

The initial condition of the integrator is set by imposing a voltage across the integration capacitor using a fully differential OTA driven by an eight-bit current DAC (shown in the upper left in Fig. 4.7). The eight-bit DAC here is the same as the CT DAC block discussed later in Chapter 5, without the input DFFs. Note that block reuse, both schematic and layout, would save considerable time and efforts in large VLSI system design.

The schematic of the fully differential OTA (OTA-3) used in the initial condition setting stage (see Fig. 4.7) is shown in Fig. 4.11. Table 4.5 shows the transistor sizes in our design. Common-mode rejection is critical for this OTA design, because any common-mode level deviation at its output would be directly imposed on the integrator capacitor C_1 (Fig. 4.7) in initial condition setting mode. When the integrator goes to operation mode, large common-

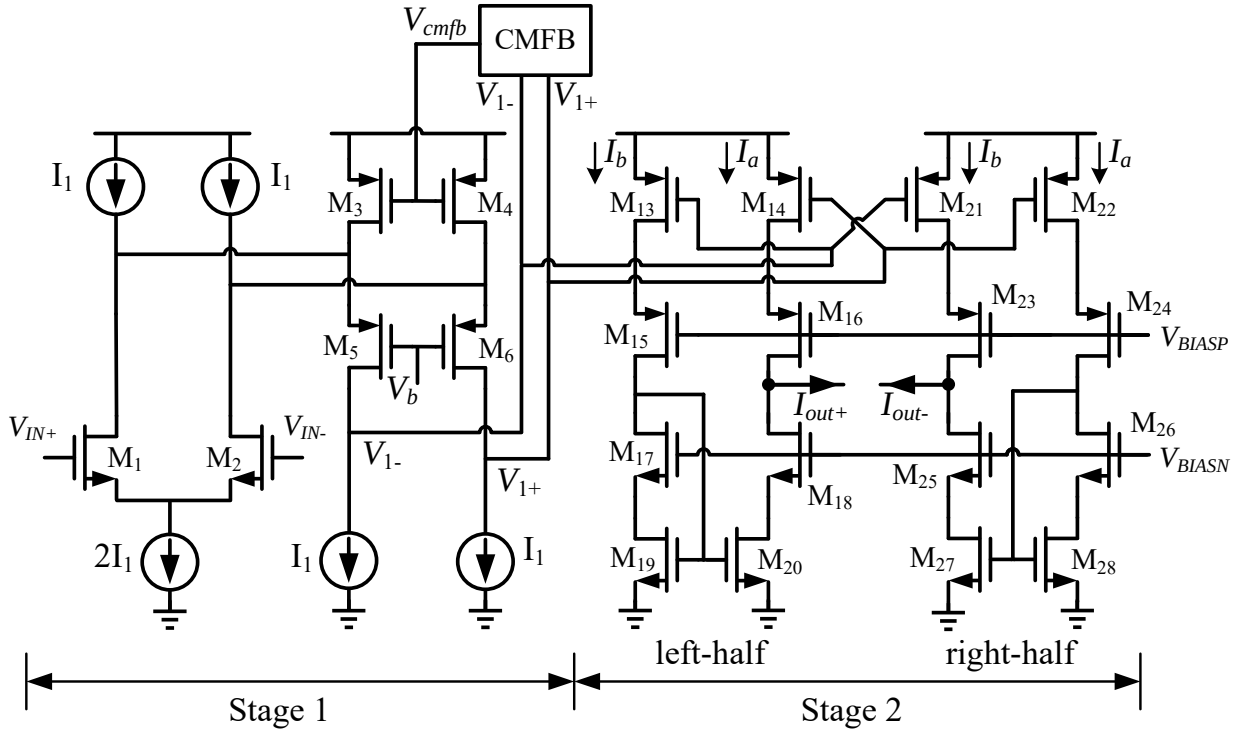

Figure 4.11: Schematic of the fully differential OTA used in initial condition setting block.

mode level deviation on the capacitor would make the following DC-coupled blocks operate abnormally initially. Though the integrator's CMFB stage would reject this common-mode signal on the capacitor (after some time), it would be better to reject it at the origin and not to overload integrator's CMFB block (it needs to reject Integrator's input signal).

Shown in Fig. 4.11, the input stage of the fully differential OTA is a differential pair (M_1 and M_2) with folded cascode devices (M_5 and M_6). The CMFB circuit is shown in Fig. 4.12. M_9 and M_{10} act as a common-mode detector. The voltage difference between V_{CM} and $V_{1,com} = (V_{1+} + V_{1-})/2$ will be minimized through the CMFB feedback loop.

The second stage is another fully differential stage that further rejects the common-mode

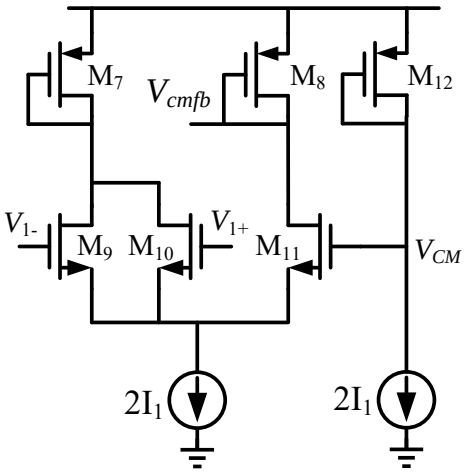

Figure 4.12: Schematic of the CMFB block fully differential OTA.

Transistors	**M_1**	**M_2**	**M_3**	**M_4**	**M_5**	**M_6**	**M_7**	**M_8**	**M_9**	**M_{10}**
***W/L* (µm/µm)**	12/0.6	12/0.6	4/2	4/2	8/0.5	8/0.5	4/2	4/2	8/0.5	8/0.5
Multiplicity	1	1	1	1	1	1	1	1	1	1
V_T type	Low		Normal		Low		Normal		Low	
Transistors	**M_{11}**	**M_{12}**	**M_{13}**	**M_{14}**	**M_{15}**	**M_{16}**	**M_{17}**	**M_{18}**	**M_{19}**	**M_{20}**
***W/L* (µm/µm)**	8/0.5	4/1	4/1	4/1	2/0.4	2/0.4	2/0.5	2/0.5	4/1	4/1
Multiplicity	2	1	1	6	1	6	1	6	1	6
V_T type	Low	Normal								
Transistors	**M_{21}**	**M_{22}**	**M_{23}**	**M_{24}**	**M_{25}**	**M_{26}**	**M_{27}**	**M_{28}**	**N/A**	
***W/L* (µm/µm)**	4/1	4/1	2/0.4	2/0.4	2/0.5	2/0.5	4/1	4/1	N/A	
Multiplicity	6	1	6	1	6	1	6	1	N/A	
V_T type	Normal									

Table 4.5: Transistor sizes of the fully differential OTA used in the initial-condition-setting stage.

current signal by applying a crosscoupling technique. The left-half and right-half circuits are symmetrical. As an illustration purpose, denote the signal current through the M_{14} branch with I_{A} and that through the M_{21} with I_{B}. Then we have drain currents at M_{19}

Specs	Lower 3-dB frequency	DC gain	Unity-gain frequency	Phase @ 20 kHz	Output Noise (1kHz – 1MHz)	THD*	Power dissipation
Simulation results	3.5 Hz	75 dB	19.9 kHz	-90.1 degree	1.3 nA	-50 dB	24.6 µW

*20 kHz, full-scale (2 µA peak, differential)

Table 4.6: Simulation results for key specifications of integrator block.

and M_{20} equal to I_{B}, and the drain currents at M_{27} and M_{28} equal to I_{A}. The output currents can be expressed as $I_{\mathrm{OUT+}} = I_{\mathrm{A}} - I_{\mathrm{B}}$, $I_{\mathrm{OUT-}} = I_{\mathrm{B}} - I_{\mathrm{A}}$. Therefore, the differential output is $I_{out,com} = I_{\mathrm{OUT+}} - I_{\mathrm{OUT-}} = 2(I_{\mathrm{A}} - I_{\mathrm{B}})$ and the common-mode output are $I_{out,com} = (I_{\mathrm{OUT+}} + I_{\mathrm{OUT-}})/2 = 0$. We can see that the output stage passes the differential component while rejecting the common-mode component.

Key simulation results are listed in Table 4.6.

4.4 Multiplier architecture and circuit design

Fig. 4.13 shows the multiplier/VGA architecture based on [1]. The multiplier core block uses current-mode translinear circuits, implemented with Gilbert's translinear principle by operating MOS transistors in the weak inversion region to achieve exponential behavior.

The block can operate either as a multiplier or a VGA. The module has two input ports and one output port. When the module is operating in multiplier mode, the output is a scaled version of the product of the two inputs:

$$I_{\mathrm{OUT}} = \frac{0.5 K_1 I_{\mathrm{IN1}} K_2 I_{\mathrm{IN2}}}{K_o I_{\mathrm{ref}}}, \tag{4.19}$$

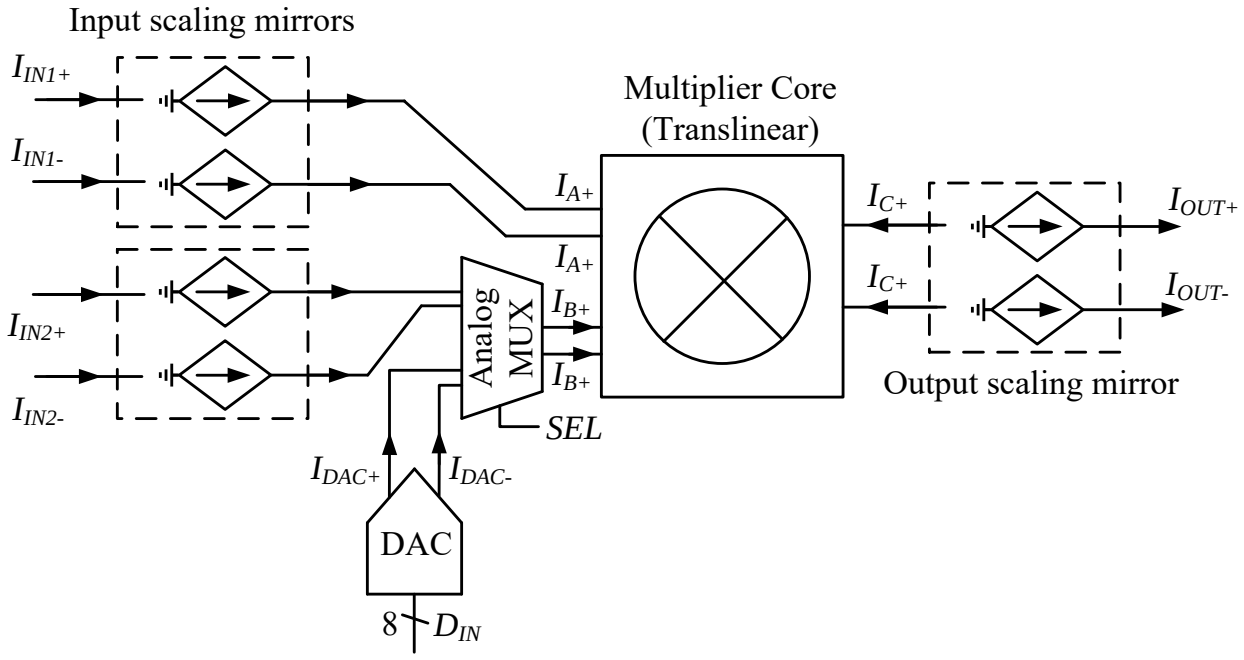

Figure 4.13: Architecture of the multiplier block.

where $I_{\text{ref}} = 1\,\mu\text{A}$ and K_1, K_2, and K_o are the mirroring ratios of the scaling mirrors. These K mirror ratios are set to values of 1, 10, and 0.1 at the same time, allowing the output I_{OUT} to be a scaled version of the input products. For example, when all Ks are set to 1, we have

$$I_{\text{OUT}} = \frac{0.5 I_{\text{IN1}} I_{\text{IN2}}}{1\,\mu\text{A}}. \tag{4.20}$$

Please pay attention to the 0.5 scaling factor for the multiplication result, which comes from the multiplication core block and will be explained later. (When we use multipliers in solving equations, the 0.5 coefficient could be absorbed by manipulating the equation expressions. See Chapter 8 for examples.)

When operating in VGA mode, the second scaling mirror is turned off (I_{IN2} is disabled) and the MUX is configured to receive input from the DAC block. The DAC block here is the

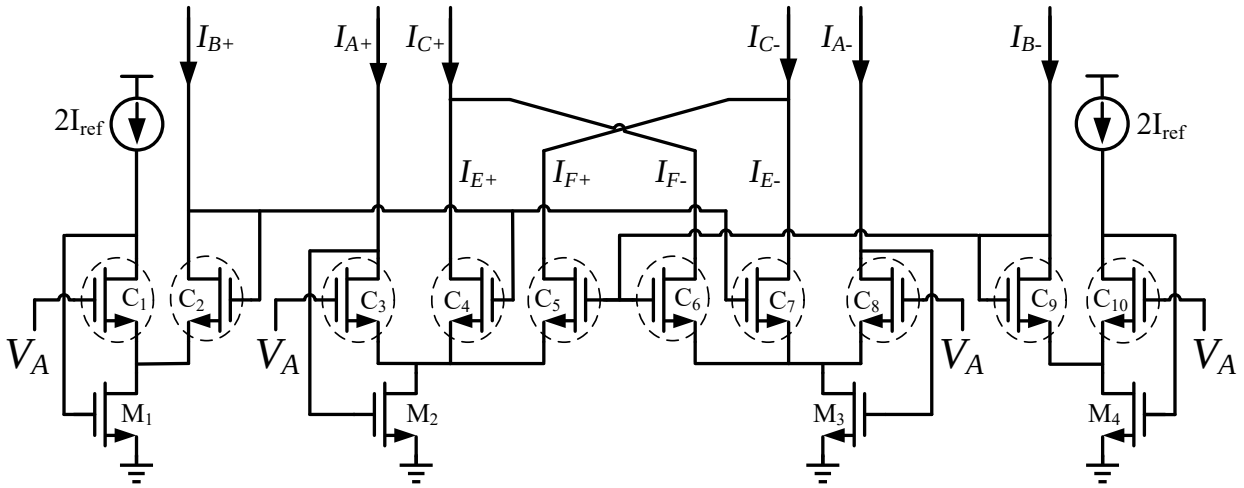

Figure 4.14: Circuit details of the multiplier core block.

same CT DAC design used in the nonlinear function generator, except that the push–pull conversion stage is removed (see the CT DAC design details in Chapter 5). Now the output is the product of the first input and a constant current set by an 8-bit current DAC. The output is expressed as

$$I_{\text{OUT}} = \frac{K_1 I_{\text{IN1}}}{2K_o I_{\text{ref}}} I_{\text{DAC,FS}} \left(\frac{D_{\text{IN}}}{128} - 1\right), \tag{4.21}$$

where D_{IN} is the eight-bit digital input (binary weighted), and $I_{\text{DAC,FS}}$ is the full-scale current of the DAC block, which is 2 µA. For example, when K_1 and K_o are set to 1, we have

$$I_{\text{OUT}} = I_{\text{IN1}} \left(\frac{D_{\text{IN}}}{128} - 1\right). \tag{4.22}$$

By setting D_{IN} from 0000 0000 to 1111 1111, the coefficient of I_{IN1} can be varied from -1 to $\frac{127}{128}$ in increments of $\frac{1}{128}$.

Fig. 4.14 shows the details of the translinear multiplier core. Devices C_1–C_{10} are composite devices that behave like single NMOS transistors with large drain output impedance. Fig.

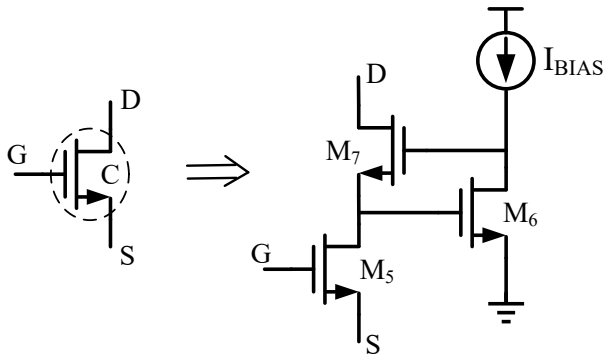

Figure 4.15: The composite device used in the multiplier core.

Transistors	**M_1**	**M_2**	**M_3**	**M_4**	**M_5**	**M_6**	**M_7**
***W/L* (µm/µm)**	20/0.5	20/0.5	20/0.5	20/0.5	75/0.4	2/3	5/0.5
Multiplicity	1	1	1	1	1	1	1
V_T type	High				Normal		Low

Table 4.7: Transistor sizes of the multiplier core circuits.

4.15 shows the schematic of the composite device, and Table 4.7 presents the transistor sizes. M_5 is the core device biased in weak inversion and provides the exponential I–V characteristic of the translinear element. M_5 is sized with a large W/L ratio of 20 µm/0.5 µm to achieve a low inversion coefficient and good exponential I–V behavior. L is kept short to conserve area and guarantee device bandwidth. If M_5 is used directly in place of C_1–C_{10}, the drain node of M_5 is directly exposed to outside, so that the voltage variation at the drain of M_5 would cause its I–V characteristic to deviate significantly from the ideal exponential, resulting in errors in the multiplier transfer characteristic. M_6 and M_7 are added to isolate and buffer M_5. M_6 acts as a cascode device for M_5, and common-source amplifier M_7 boosts M_6. Most of the voltage variation at Node D is absorbed by M_6, keeping the M_5 drain voltage stable.

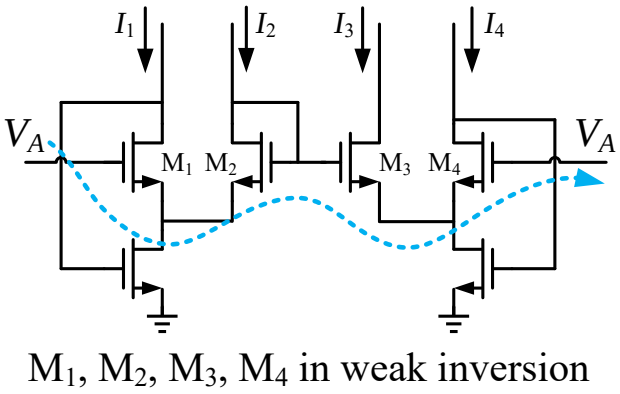

Figure 4.16: Illustration of basic translinear principle.

The basic translinear principle is illustrated in Fig. 4.16. All four transistors are in weak inversion. Applying KVL, we have $V_{\mathrm{GS1}} + V_{\mathrm{GS3}} = V_{\mathrm{GS2}} + V_{\mathrm{GS4}}$. With the exponential I–V relationship, we could replace all instances of V_{GS} with drain currents and have $I_1 I_3 = I_2 I_4$.

For the multiplier core block (Fig. 4.14), C_1–C_{10} form a total of five translinear loops, each of which has four devices in the loop. The loop formed by C_1, C_2, C_3, and C_4 gives

$$2I_{\mathrm{ref}} I_{\mathrm{E}+} = I_{\mathrm{B}+} I_{\mathrm{A}+}. \tag{4.23}$$

The loop formed by C_1, C_2, C_7, and C_8 gives

$$2I_{\mathrm{ref}} I_{\mathrm{E}-} = I_{\mathrm{B}+} I_{\mathrm{A}-}. \tag{4.24}$$

The loop formed by C_{10}, C_9, C_6, and C_8 gives

$$2I_{\mathrm{ref}} I_{\mathrm{F}-} = I_{\mathrm{B}-} I_{\mathrm{A}-}. \tag{4.25}$$

The loop formed by C_{10}, C_9, C_5, and C_3 gives

$$2I_{\mathrm{ref}} I_{\mathrm{F}+} = I_{\mathrm{B}-} I_{\mathrm{A}+}. \tag{4.26}$$

Subtracting (4.24) from (4.23) and (4.25) from (4.26) gives

$$2I_{\text{ref}}(I_{\text{E}+} - I_{\text{E}-}) = I_{\text{B}+}(I_{\text{A}+} - I_{\text{A}-}) \tag{4.27}$$

$$2I_{\text{ref}}(I_{\text{F}+} - I_{\text{F}-}) = I_{\text{B}-}(I_{\text{A}+} - I_{\text{A}-}). \tag{4.28}$$

Subtracting (4.28) from (4.27), we have

$$2I_{\text{ref}}[(I_{\text{E}+} + I_{\text{F}-}) - (I_{\text{E}-} + I_{\text{F}+})] = (I_{\text{B}+} - I_{\text{B}-})(I_{\text{A}+} - I_{\text{A}-}). \tag{4.29}$$

From Fig. 4.14, we also have $I_{\text{C}+} = I_{\text{E}+} + I_{\text{F}-}$ and $I_{\text{C}-} = I_{\text{E}-} + I_{\text{F}+}$. Plugging these values into (4.29) and rearrange the equation, we have

$$I_{\text{C}+} - I_{\text{C}-} = \frac{(I_{\text{B}+} - I_{\text{B}-})(I_{\text{A}+} - I_{\text{A}-})}{2I_{\text{ref}}}. \tag{4.30}$$

Now we have the desired multiplication behavior from the translinear core for our differential input/output currents.

However, we also must pay attention to the core's behavior for common-mode signals. By adding (4.23), (4.24), (4.25), and (4.26) together, we have

$$I_{\text{C}+} + I_{\text{C}-} = \frac{(I_{\text{B}+} + I_{\text{B}-})(I_{\text{A}+} + I_{\text{A}-})}{2I_{\text{ref}}}. \tag{4.31}$$

We can see that the common-mode signals behave the same as the differential ones, meaning that there is no intrinsic common-mode rejection for the translinear core. To remove this common-mode component, we use a crosscoupled architecture at the output stage between the scaling mirrors, as will be seen shortly.

Fig. 4.17(a) shows the schematic of the input-scaling mirror in Fig. 4.13 and Table 4.8 presents the corresponding transistor sizes used. The input current mirror operates in Class-A mode. The purpose of the input-scaling current mirror is to support the three different

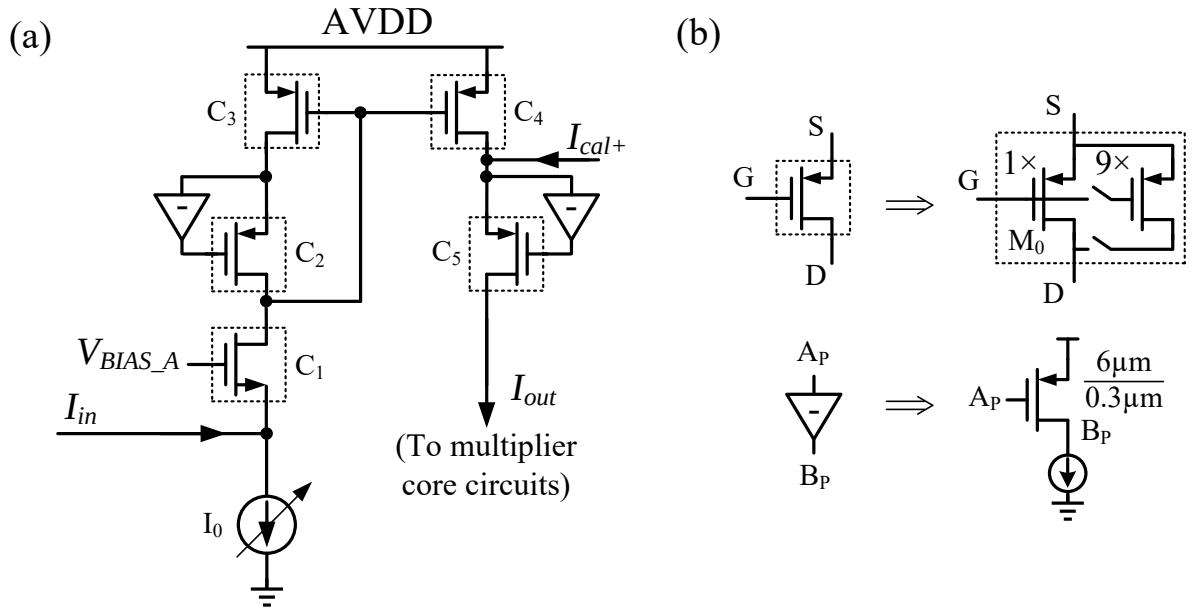

Figure 4.17: A simplified schematic of the input-scaling mirror in Fig. 4.13. There are four such mirrors receiving two differential currents at the input stage.

Unit Transistor M_0 in	**C_1**	**C_2**	**C_3**	**C_4**	**C_5**
***W/L* (µm/µm)**	6/0.4	12/0.4	12/1.5	12/1.5	12/0.4
Multiplicity	1	1	1	1	1
V_T type	Low	Normal			

Table 4.8: Transistor sizes of the input-scaling mirror.

input ranges while holding constant the signal swing seen by the multiplier core. The transfer characteristic equation of the circuit is $I_{\text{OUT}} = K \cdot I_{\text{IN}} + I_0$, where I_0 is value of the DC bias current to the multiplier core and K is a programmable gain factor (0.1, 1, or 10 by setting the programmable devices C_1–C_5 accordingly). C_1–C_5 are composite devices shown in Fig. 4.17 (b). There are four scaling mirrors at the input stage. Two mirrors process each differential

input current in a pseudo-differential style.

We use two six-bit calibration-current DACs (the same as the one used in the fanout block, see Fig. 4.2) to calibrate the multiplier's input offsets. For the two scaling mirrors that process one differential input, one calibration current ($I_{\mathrm{CAL}+}$) is injected into the drain of C_4 inside one scaling mirror, as shown in Fig. 4.17 and the other calibration current ($I_{\mathrm{CAL}-}$) is injected into the other scaling mirror as in Fig. 4.13.

The output-scaling current mirror in Fig. 4.13 scales the output of the multiplier core to match the range of the final output and at the same time removes the common-mode component. Fig. 4.18 shows the block diagram and Fig. 4.19 shows the schematic of the output-scaling current mirror. Table 4.9 shows the transistor sizes used. The output-scaling current mirror has a structure similar to the input-scaling mirror. The difference is that each of the single-ended mirrors in the output-scaling mirror has two current outputs, and they are combined so that the final differential output contains no common-mode component.

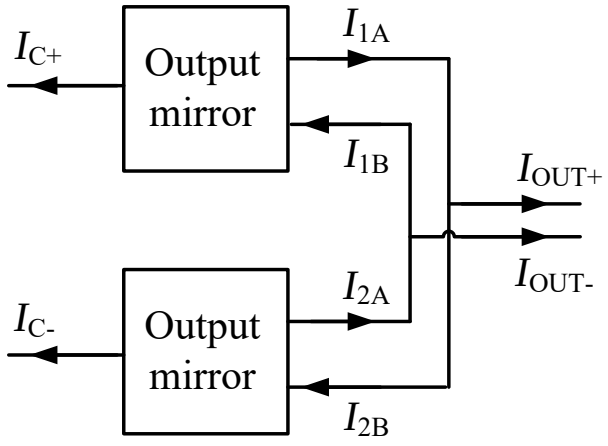

Figure 4.18: Block diagram of the multiplier output mirror.

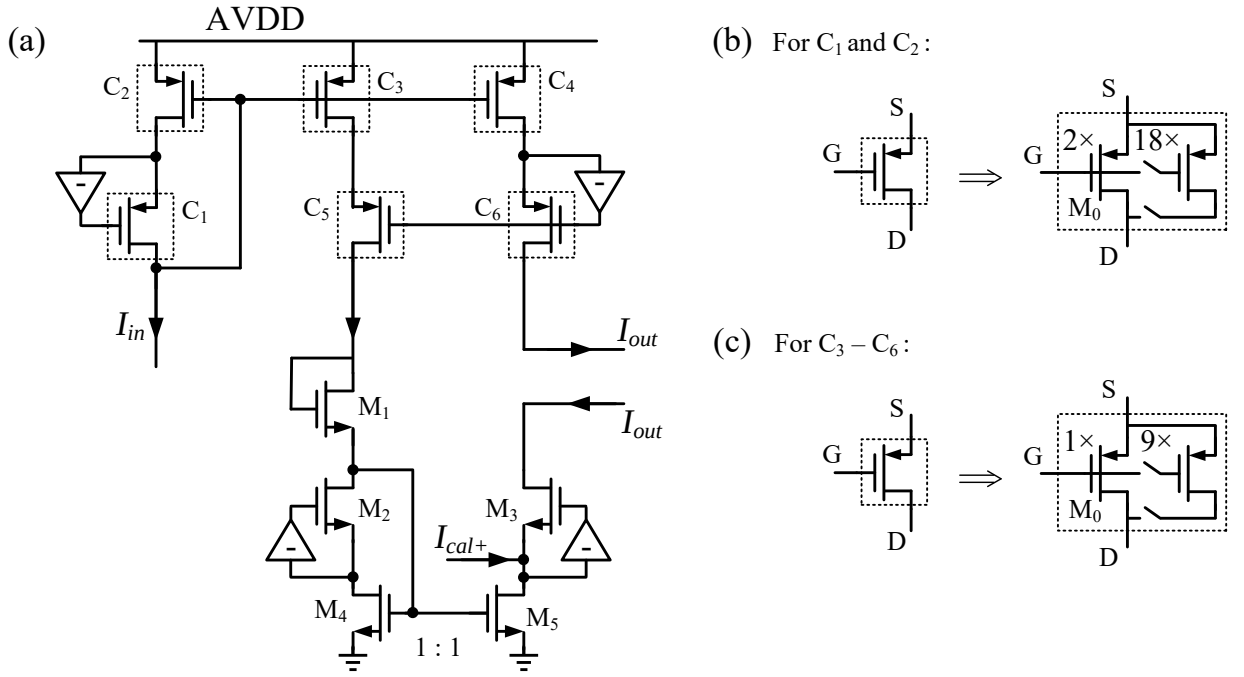

Figure 4.19: Schematic of the output-scaling mirror.

Fig. 4.18 shows how the four output currents are combined to generate the differential outputs $I_{\text{OUT}+}$ and $I_{\text{OUT}-}$. Assume that the mirroring ratio is 1 for the output stage. Then we have $I_{\text{OUT}+} = I_{1\text{A}} - I_{2\text{B}}$ and $I_{\text{OUT}-} = I_{2\text{A}} - I_{1\text{B}}$. The output common-mode signal is $I_{\text{OUT}+} + I_{\text{OUT}-} = 0$ and the output differential signal is $I_{\text{OUT}+} - I_{\text{OUT}-} = I_{\text{C}+} - I_{\text{C}-}$, achieving the common-mode rejection purpose.

We use one six-bit calibration-current DAC at the output stage (the same one used in fanout block, see Fig. 4.2 for more details) to calibrate the multiplier's output offset. One calibration current ($I_{\text{CAL}+}$) is injected into the drain of M_5, as shown in Fig. 4.19, and the other ($I_{\text{CAL}-}$) injected into the same position inside the other output mirror. Key simulation results are listed in Table 4.10.

Unit Transistor M_0 in	C_1	C_2	C_3	C_4	C_5	C_6
***W/L* (µm/µm)**	12/0.2	12/0.6	12/0.6	12/0.6	12/0.2	12/0.2
Multiplicity	2	2	1	1	1	1
V_T type	Normal					
Transistors	M_1	M_2	M_3	M_4	M_5	N/A
***W/L* (µm/µm)**	12/0.2	12/0.4	12/0.4	20/0.6	20/0.6	N/A
Multiplicity	1	1	1	1	1	N/A
V_T type	Low	Normal				N/A

Table 4.9: Transistor sizes of the output-scaling mirror circuits in Fig. 4.19.

Specs	3-dB bandwidth	Phase shift @ 20 kHz	THD*	Output Noise (1kHz – 1MHz)	Power dissipation
Simulation results	3.78 MHz	-0.3 degree	-59 dB	2.02 nA(rms)	51.6 µW

*20 kHz, full scale (2 µA peak, differential)

Table 4.10: Simulation results for key specifications of multiplier block.

4.5 Circuits for testability

Circuit structures for probing internal nodes, especially for DC operating points, are vital for mixed-signal designs. Considering the large number of analog and mixed-signal blocks in our hybrid computing system, we used a two-level multiplexing scheme.

For each analog block, we have the ability to measure the voltages of seven internal nodes, connected to the external testing pin through analog muxes. We chose seven nodes instead of eight because we need a state to shut down all the signals when the analog block is in operation. Due to the large number of analog blocks, we need to further multiplex the

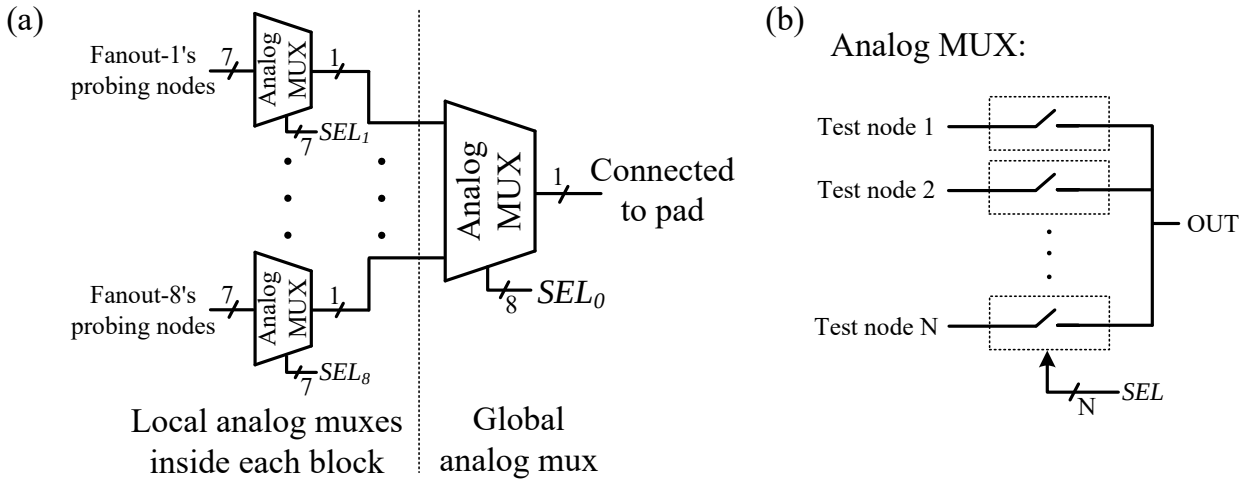

Figure 4.20: The testing scheme for measuring internal nodes' voltages on our hybrid computing chip.

testing wires between them. For example, there is one testing signal (wire) coming out of each multiplier block; since we have eight fanout blocks, these 8 testing signals are further multiplexed to share a single pad (Fig. 4.20(a)).

More importantly, we need very good isolation for the analog muxes. Otherwise, the signals will be coupled to each other in the computing phase. As shown in Fig. 4.21, we use a T-style analog switch to construct analog muxes. When the T-switch is disabled (EN set low), Node C is pulled to GROUND, shielding the circuit from undesired coupling by the floating node. Simulation shows that this T-switch provides $-190\,\mathrm{dB}$ isolation between IN and OUT when off. Table 4.11 shows the transistor sizes in Fig. 4.21.

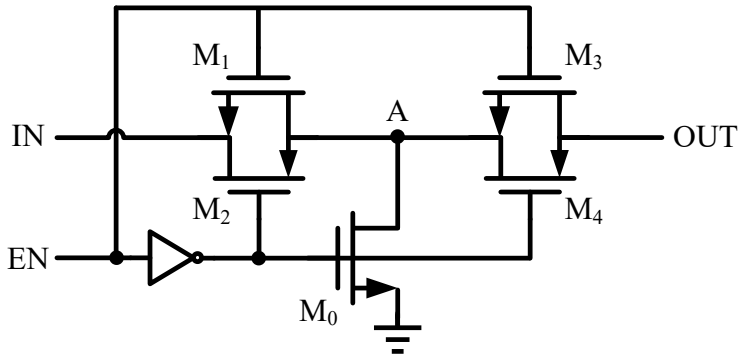

Figure 4.21: The analog T-switch used in analog MUX.

Transistors	**M_0**	**M_1**	**M_2**	**M_3**	**M_4**
***W/L* (μm/μm)**	0.5/0.3	1/0.3	1/0.2	1/0.3	1/0.2
Multiplicity	1	1	1	1	1
V_T type	Normal				

Table 4.11: The transistor sizes of the analog T-switch.

4.6 Global crossbar design

The signal paths between the analog computing blocks are routed by programming the global analog crossbars (see Fig. 3.6). Fig. 4.22 shows the design details of four analog crossbars. The crossbar design is scalable both horizontally and vertically, so we can stack them as needed. The programming bits controlling the ON/OFF states of the transmission gate of one crossbar cell are stored in registers located inside that crossbar cell (Fig. 4.23). The registers use a standard 6T SRAM cell design. Nodes EN and $\overline{\text{EN}}$ are directly connected to Nodes EN and $\overline{\text{EN}}$ of one crossbar cell (Fig. 4.22). We use the static differential driving method here to write the contents of the cell.

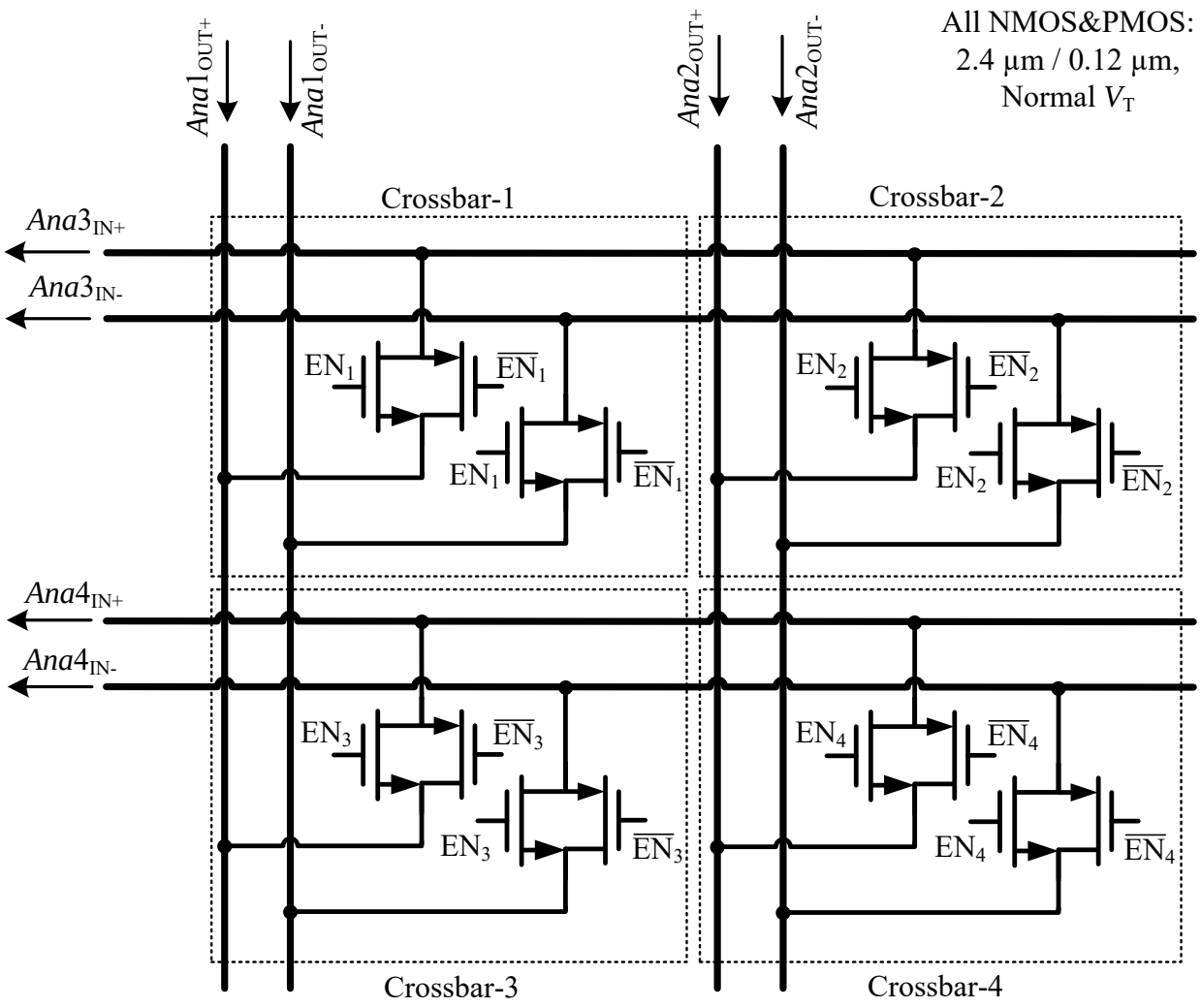

Figure 4.22: The analog crossbars used for programming signal paths between analog blocks. Local registers (back-to-back inverters) that store the programming bits for the transmission gates are not shown.

If we would like to connect Analog Block 1's output ($\text{Ana1}_{\text{OUT+}}$ and $\text{Ana1}_{\text{OUT-}}$) to Analog Block 3's input ($\text{Ana3}_{\text{IN+}}$ and $\text{Ana3}_{\text{IN-}}$), we close Crossbar 1 by setting EN_1 to high and must leave Crossbar 3 open. Only 1 crossbar in the same column can be closed at the same time

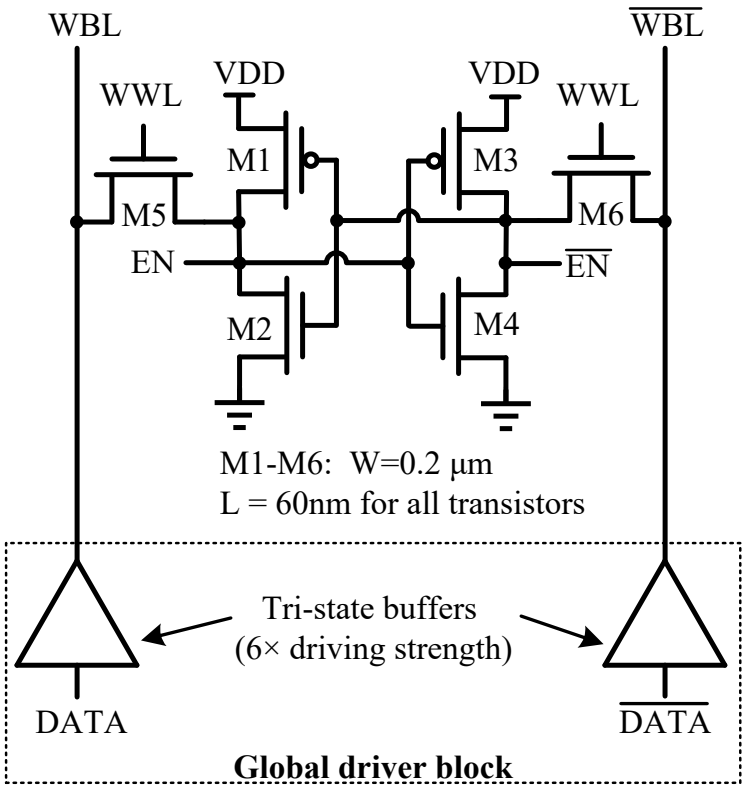

Figure 4.23: The local register that stores the programming information for the transmission gates. The contents of the back-to-back inverters are written through driving the bit lines differentially with large drivers.

because one analog output current can only go to one input port fully. If we need to distribute the same output current value to multiple places, we need to use the fanout block to make multiple copies.

Fig. 4.24 shows the layout of one analog crossbar cell. The layout area here should be minimized because there are many analog crossbars on chip for signal routing.

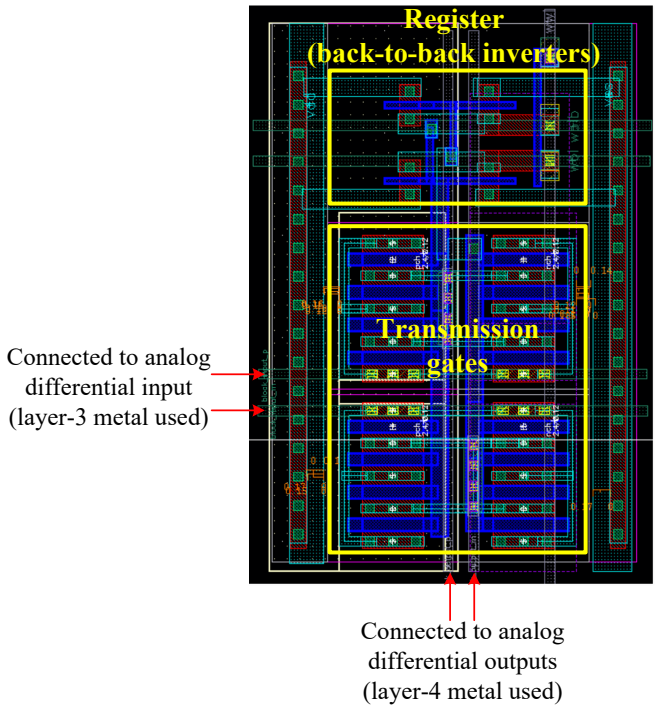

Figure 4.24: The layout of one analog crossbar cell.

4.7 Layout considerations

In addition to device matching, another difficult aspect of layout is limiting the dimensions of different functional blocks for space efficiency. It is an iterative process to align all blocks to the same height. (There is no need to align the width as the same type of block occupies the entire column.) Fig. 4.25 shows one "slice" of the analog block array, which consists of two fanout blocks, one integrator block and two multiplier blocks (see the system architecture in Fig. 3.6). Integrator and multiplier blocks have about the same height (160 µm), while the fanout blocks are a bit shorter due to their simpler functionality and circuit architecture. The crossbar "islands" sit in the top-right corner of each block (Fig. 4.25).

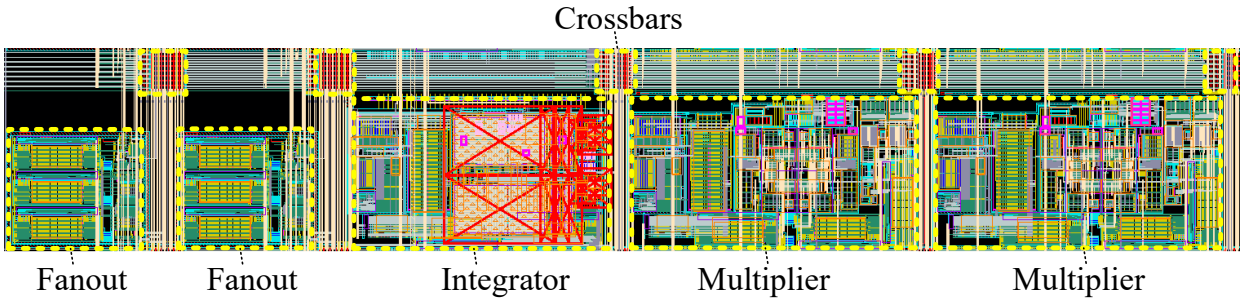

Figure 4.25: The global signal paths for calibrating each block and related functional blocks involved.

Chapter 5

Design of the Programmable Nonlinear Function Generator

In this chapter, we discuss the design of the key feature of our hybrid computing system: the arbitrary nonlinear function generator. It greatly increases the generality of our chip and expands the range of mathematical problems that can be solved, compared to the earlier effort in [1].

5.1 Introduction to CT digital signals

Continuous-time digital signal processing was introduced by Tsividis in 2003 [19]. In contrast to conventional digital signal processing with its issues of aliasing and high quantization noise due to time sampling, processing the digital signals in the continuous-time domain completely avoids aliasing and greatly reduces quantization noise. In 2008, Schell [13] demonstrated the

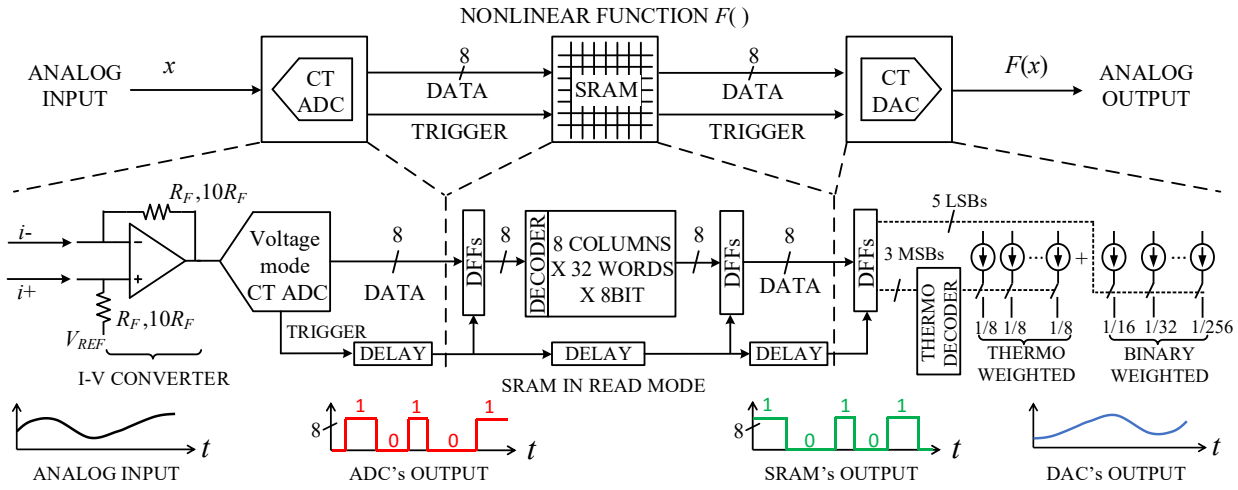

Figure 5.1: Continuous-time programmable nonlinear function generator.

use of continuous-time digital signals in audio signal-processing applications.

The work presented here is the first one to use CT digital signals for computing applications—specifically, solving differential equations.

5.2 Architecture overview

The nonlinear function generator is implemented in a programmable, CT hybrid architecture (Fig. 5.1), consisting of a CT ADC, a CT SRAM, and a CT DAC.

Our generator uses a clockless, CT architecture because it offers several important advantages over the discrete-time one. First, a clockless, CT architecture has an event-driven, activity-dependent power dissipation feature. The digital circuits switch only when the input analog signal changes. Second, this architecture's response to input changes is not limited by a clock period so there is no clock period latency, allowing for real-time operation. Third,

since the circuit does not need a clock, no power is dissipated in distributing a clock signal. Finally, because this scheme operates in CT, it inherently avoids aliasing, which could otherwise affect computation accuracy for solving certain equations.

In our nonlinear function generator (5.1), the analog input current is fed to a transimpedance amplifier, and the output is then converted into an eight-bit CT digital signal by a voltage-mode, CT level-crossing ADC similar to the one used in [13]. In the meantime, the CT ADC generates a trigger signal that indicates a level-crossing action at the analog input. The ADC's output is then taken as the address input of the SRAM block to fetch the nonlinear function values stored inside. The trigger signal from the ADC is then passed through a delay line to give enough time for the SRAM block to finish the read operation; after the data have been read out from the SRAM and settled, the trigger signal triggers the SRAM's output DFFs, allowing the nonlinear function data to be sent to the next stage. The following eight-bit CT DAC converts the data back to current signals, which are sent to other analog blocks. Since the conversion of the input and output analog signals is done in CT, this conversion scheme works in real time.

It is worth mentioning that the work by Huang and Zukowski [20], developed independently in 2006, also used a similar clockless architecture with three-bit resolution. In that work's configurable logic block, eight inverter-based one-bit ADCs convert analog inputs to digital signals, which are then passed into the digital look-up table. This digital look-up table is implemented with programmable combinational logics with a three-bit digital output, where the programming information is stored on shift registers. The output of the digital

look-up table serves as a weighting factor for a dependent current source, whose reference current is controlled by the analog input value. There are differences between our nonlinear function generator and the configurable logic block in [20], in addition to the increased resolution and lookup depth. First, our design is a general-purpose nonlinear function generator, which could be used for such purposes as equation solving and analog signal processing, while the configurable logic block in [20] is a highly specialized design for simulating one type of differential equation describing gene regulatory networks. The output of that block represents the concentration of a particular molecule. Second, our nonlinear function data are stored inside a SRAM block for lookup, while the nonlinear function data in [20] are generated by feeding input data into combinational logics. Third, each build block (ADC, SRAM, DAC) inside our nonlinear function generator has programmable data paths, so that each could be used as a stand-alone block for other purposes (e.g., DAC for generating constant values shown in differential equations). In contrast, the signal paths are fixed inside the configurable logic block in [20]. Last, the implementation details in Fig. 5.1 of our design are different from those in [20]. The design details of the various blocks in our nonlinear function generator are now given in the following sections.

5.3 CT ADC design details

The 8-bit CT ADC can convert full-scale signals up to 20 kHz. That is, the ADC bandwidth is from DC to 20 kHz. The CT ADC can handle two selectable current-signal ranges ($\pm 2\,\mu A$ and $20\,\mu A$) by adjusting the gain of the input transimpedance amplifier, as shown on the

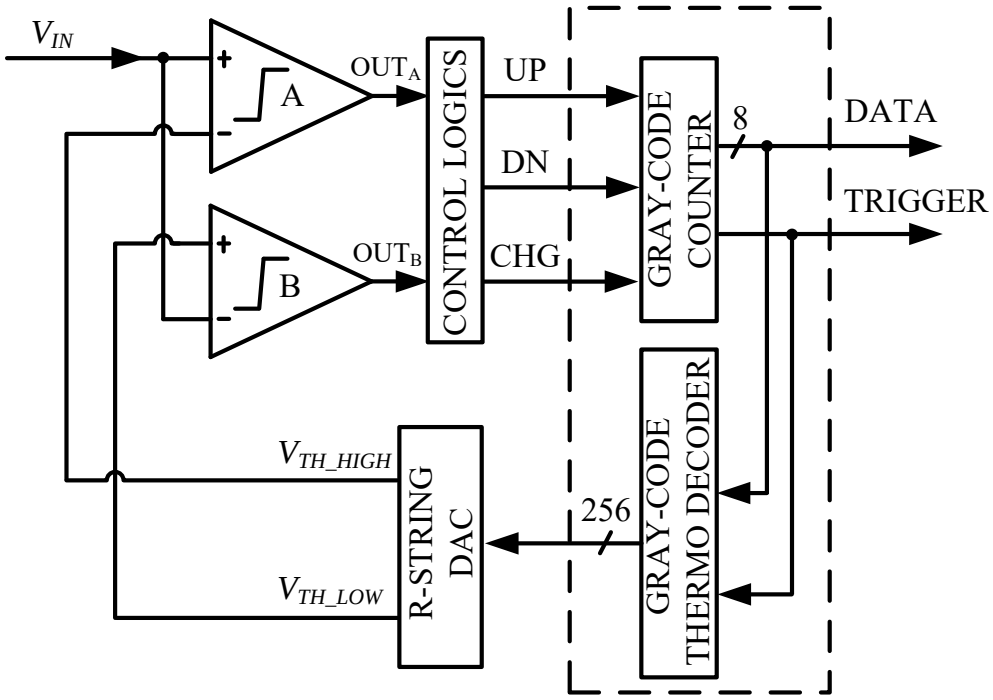

Figure 5.2: Voltage-mode level-crossing ADC architecture.

left in Fig. 5.1. The full-scale input of the voltage-mode ADC (Fig. 5.2) is fixed at 0.6 V (from 0.4 V to 1.0 V, $V_{\mathrm{LSB}} = 2.344\,\mathrm{mV}$). It has an R-string DAC in the feedback path that adjusts the comparison voltages fed to the comparators and guarantees that the input is always contained between two successive comparison levels. The R-string DAC consists of 256 identical polyresistors. In order to achieve good linearity, the common-centroid layout is applied to the R-string block. Our design improves the work in [13] in power dissipation. We use a Gray-code counter and a Gray-code thermometer decoder in the feedback path instead of the original big shift-register array (256-bit), shown in the dashed box in Fig. 5.2. Since only two bits are flipping at a time, this scheme greatly reduces the peak digital-switching current and the potential noise interference with the analog circuits.

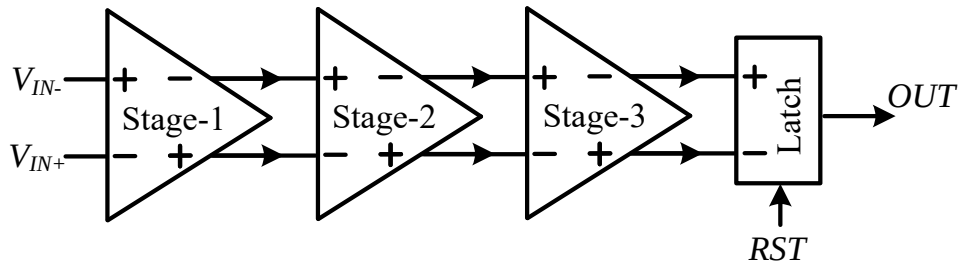

Figure 5.3: Block diagram of the comparator block.

As the fastest input signal that could be handled by this CT ADC is a 20 kHz sinusoidal signal with a 0.6 V swing (0.3 V amplitude), the level-crossing speed of V_{IN} is fastest when the sinusoidal input is crossing the common-mode voltage level (0.7 V). This fastest rate of change of V_{IN} is calculated by taking the derivate of the full-scale sinusoidal input $0.3\sin(2\pi \cdot 20\,\text{kHz} \cdot t)\,\text{V}$ and setting t to 0 (this is when the sine wave is crossing the common-mode voltage level), which is $0.3\,\text{V} \cdot 2\pi \cdot 20\,\text{kHz} = 37.7\,\text{kV/s}$. The maximum trigger rate generated by the ADC is when V_{IN} is crossing two consecutive threshold levels at 37.7 kV/s speed. Since $V_{\text{LSB}} = 2.344\,\text{mV}$, the shortest trigger period as $2.344\,\text{mV} \;/\; 37.7\,\text{kV/s} = 62\,\text{ns}$.

The comparator consists of three cascaded single-stage amplifiers, followed by a latch stage (Fig. 5.3). Fig. 5.4 shows the schematic of the single-stage differential amplifier. It is a single-stage differential amplifier with common-mode feedback, where the common-mode detector is a differential pair (M_7 and M_8) as shown in the dotted box.

The sizes of the input differential pair (M_2 and M_3) of the comparator's Stage 1 are critical; they dominate the trade-off between comparator speed and input offsets. Stage 1's M_2 and M_3 contribute directly to the capacitive loads at the R-string output. Smaller M_2

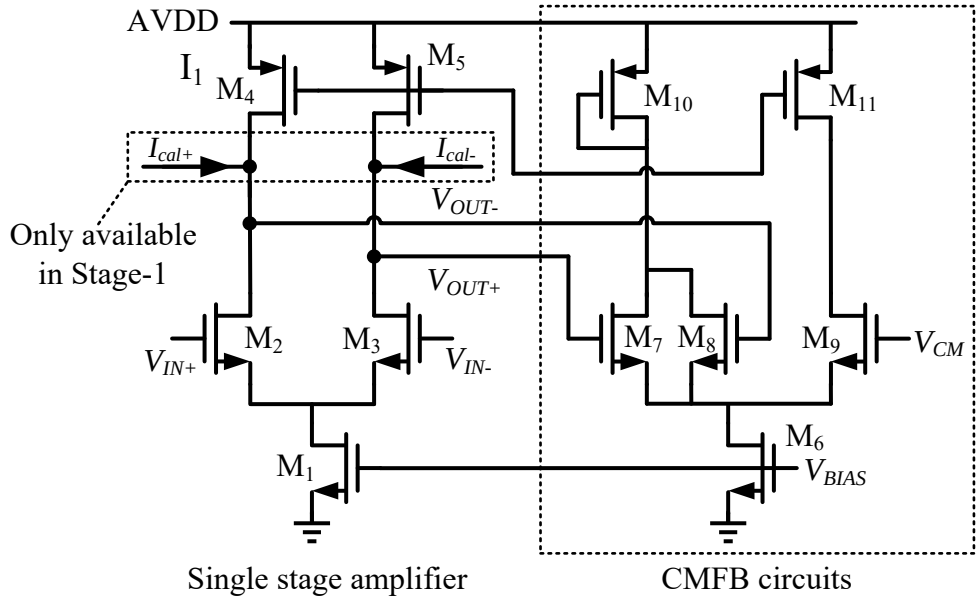

Figure 5.4: Schematic of the single stage amplifier in the comparator block.

and M_3 would yield faster settling of the R-string's outputs ($V_{\text{TH\_HIGH}}$ and $V_{\text{TH\_LOW}}$), but the mismatch between M_2 and M_3 would be larger, resulting in larger input offsets. We chose the sizes of M_2 and M_3 of each stage as shown in Table 5.1, and used one six-bit calibration current DAC (see Fig. 4.2) to calibrate the comparator's input offset at Stage 1 (the dominant offset contributor). The differential calibration currents, $I_{\text{CAL}+}$ and $I_{\text{CAL}-}$, are injected into the drains of the differential pair, as shown in Fig. 5.4. The transistor sizes used for the amplifiers in all three stages are shown in Table 5.1.

Fig. 5.5 shows the schematic of the comparator's latch stage, and Table 5.2 shows the transistor sizes used. When in reset mode, RST is set high so that M_9 is off, turning off the input stage; at the same time, M_{10} is turned on and OUT is set low. When in operation mode, RST is set low, turning on the input stage. The differential inputs $V_{\text{IN}+}$ and $V_{\text{IN}-}$ are

	Transistors	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
Stage-1	*W/L* (µm/µm)	0.75/5	1/0.25	1/0.25	0.25/0.25	0.25/0.25	0.75/5	1/0.25	1/0.25	1/0.25	0.25/0.25	0.25/0.25
	Multiplicity	28	3	3	16	16	7	3	3	6	4	4
	V_T type	Low			Normal		Low				Normal	
Stage-2	Transistors	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
	W/L (µm/µm)	0.75/5	1/0.25	1/0.25	0.25/0.25	0.25/0.25	0.75/5	1/0.25	1/0.25	1/0.25	0.25/0.25	0.25/0.25
	Multiplicity	28	8	8	4	4	7	1	1	2	1	1
	V_T type	Low			Normal		Low				Normal	
Stage-3	Transistors	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
	W/L (µm/µm)	0.75/5	1/0.25	1/0.25	0.25/0.25	0.25/0.25	0.75/5	1/0.25	1/0.25	1/0.25	0.25/0.25	0.25/0.25
	Multiplicity	28	1	1	4	4	7	1	1	2	1	1
	V_T type	Low			Normal		Low				Normal	

Table 5.1: Transistor sizes of each amplifier stage inside the comparator block.

Transistors	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
W/L (µm/µm)	0.2/0.06	0.4/1	0.6/0.06	0.6/0.06	1.2/0.06	1.2/0.06	0.2/2	0.2/0.06	0.2/0.06	0.2/0.06
Multiplicity	1	1	1	1	1	1	1	1	1	1
V_T type	Normal									

Table 5.2: Transistor sizes of the latch circuit in Fig. 5.5.

fed into a pair of inverters. When the differential input is positive—i.e., $V_{\text{IN}+} > V_{\text{IN}-}$—Node B will go up, causing Node C to go down. Pulling Node C down will gradually turn on M_8, increasing the charging current from VDD and making Node B go up even faster; this is a positive feedback. When Node C is at low, M_1 is turned off and Node B is set high (at VDD); the latch output OUT is now set high (VDD). This is a complete operation cycle of the latch stage. The next cycle starts when RST is set high again, resetting the latch.

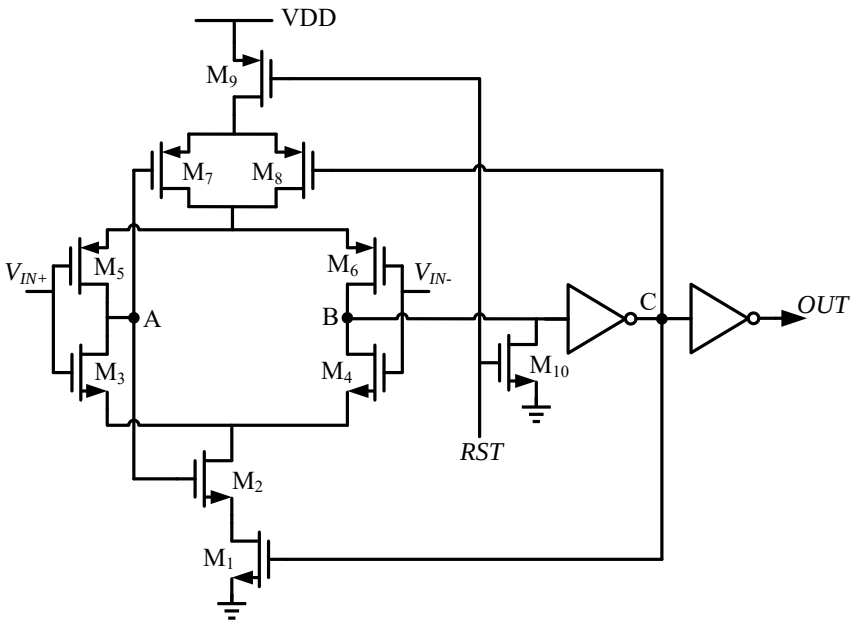

Figure 5.5: Schematic of the latch stage of the comparator.

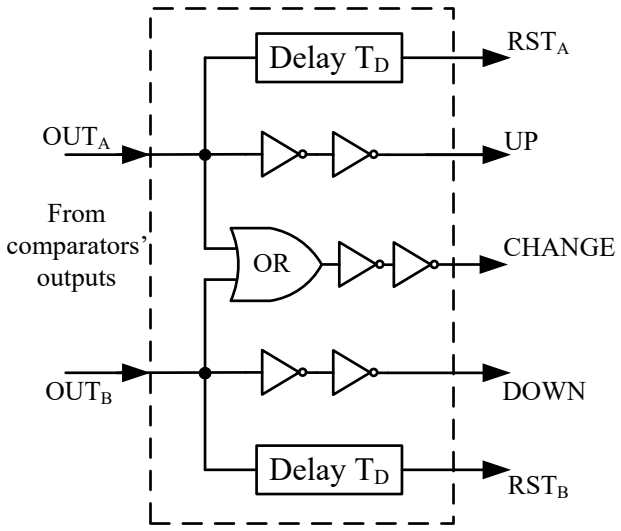

Figure 5.6: Schematic of control logic block after comparators.

Fig. 5.6 shows the block diagram of the control logic. It takes the outputs of two compara-

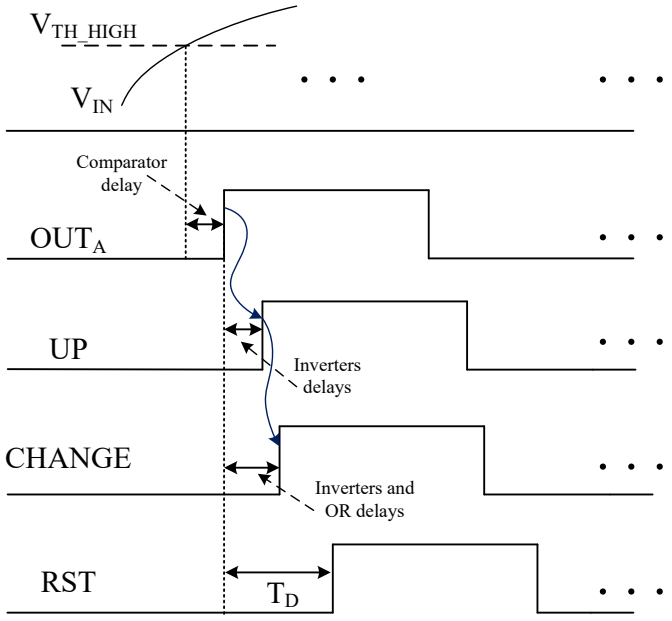

Figure 5.7: The timing diagram of critical signals of the control logic block.

tors and generates the reset signals (RST_A and RST_B) for the latches inside two comparators, plus the control signals (UP and DOWN) and trigger (CHANGE) for the counter in the next stage. Fig. 5.7 illustrates the timing of critical signals when Comparator A is in operation (i.e., when V_{IN} is crossing the upper threshold). After Comparator A's output OUT_A goes high, UP will go high after two inverter delays, putting the following counter into increment mode. Then CHANGE will go from low to high, triggering the counter to increase by one. After a delay of T_D, RST_A goes high, putting the comparator in reset mode in which OUT_A is subsequently pulled down. After all the signals go low, a new operation cycle starts again.

The ADC's speed is limited by the R-string DAC delay. The output of the R-string DAC needs to settle to the desired voltage levels before the input crosses. The worst case is when the input voltage (assuming a sine wave of 20 kHz) crosses the zero point (common-

mode voltage). If the DAC is not settled, the comparator will generate false triggers. There are two ways to increase the R-string DAC's updating speed. One way is to decrease the resistor value, thus increasing the charging currents to the load capacitances. This approach, however, would increase power dissipation. The second way to increase the R-string DAC's updating speed is to decrease the load capacitances of the R-string DAC, which are the input capacitances of the comparators. Since the load capacitances are determined by the size of the input differential pairs, decreasing their sizes would increase the updating speed. However, doing so would increase the potential mismatches between input devices, which would exceed the offset calibration range. (Note that we need to reduce the errors introduced by each block as much as possible, since the errors may be cumulative over the entire computation time). It is important to apply both methods for increasing the R-string DAC's updating speed and to achieve a balance. The final values we chose are 150 Ω for each individual resistor and 3 µm/0.25 µm for the input differential pairs (NMOS transistors).

Specs	**Comparator delay**	**RMS Noise at comparator input (1 Hz – 1 MHz)**	**SFDR**	**Power dissipation @ 20 kHz FS input**
Simulation results	75 ns @ 400 Hz 30 ns @ 20 kHz	0.234 mV (V_{LSB}/10)	67 dB	72.6 µW

Table 5.3: Simulation results for key specifications of the CT ADC.

The CT ADC is calibrated in several places. The input I–V converter (Fig. 5.1) has a six-bit (current-steering) calibration DAC for the output offset, ensuring that it outputs 0.7 V when the input current is 0 µA. Each comparator also has a six-bit DAC for the input

offset calibration, with full-scale configurable ranges of 100 nA, 200 nA, 300 nA, and 400 nA. Calibration currents injected into the two drain terminals of the input differential pairs (NMOS) balance the bias currents. Table 5.3 lists the CT ADC's key simulation results.

5.4 CT SRAM design details

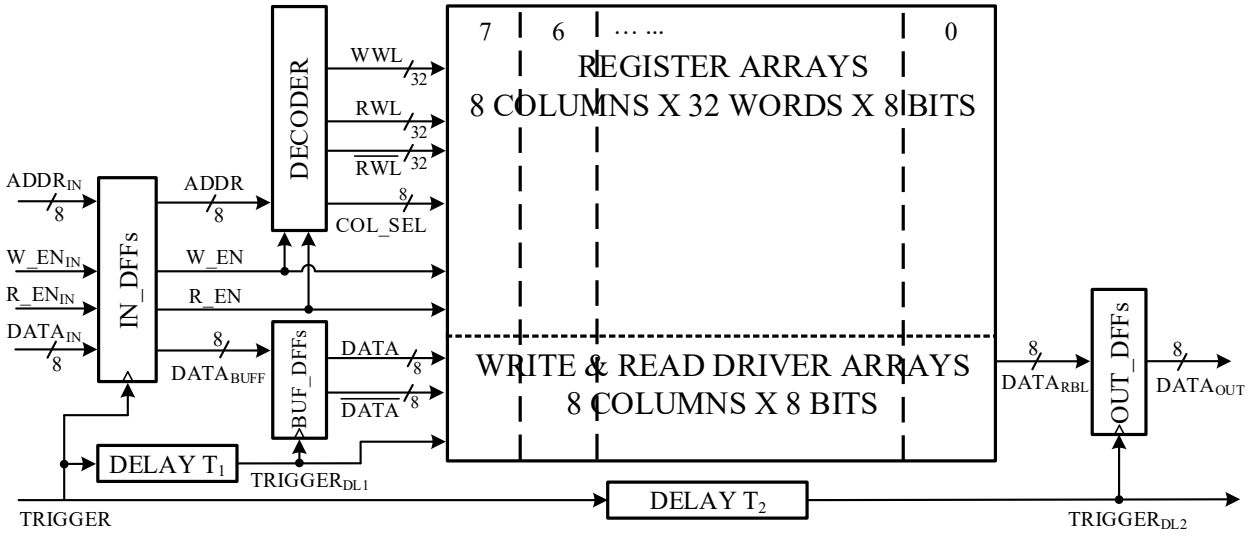

Figure 5.8: The architecture of the CT SRAM used in our hybrid system.

The CT SRAM has eight-bit address/word length (Fig. 5.8). It has a CT digital data path and its operation is controlled by trigger signals instead of a clock. The SRAM cell used in our work is based on the fully static 10T design in [21]. The sizing is shown in Fig. 5.9. The write and read drivers are shown in Fig. 5.10.

In write mode, the write enable signal W_EN_{IN} is set high and the read enable signal R_EN_{IN} is set low at the SRAM input. The timing diagram of critical signals in write mode

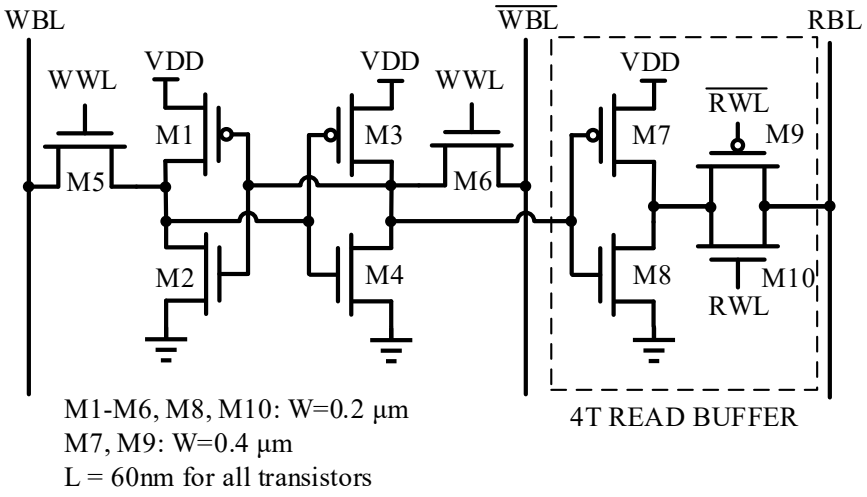

Figure 5.9: The 10T SRAM cell used in our design.

is shown in Fig. 5.11. After TRIGGER t riggers the input flip-flops IN_DFFs, the eight-bit address $\mathrm{ADDR_{IN}}$ is loaded to the decoder. The eight-bit content $\mathrm{DATA_{BUFF}}$ is buffered by BUF_DFFs. After a delay of T_1(550 ps in our design), which guarantees that the intended WWL (write word line) and COL_SEL (column select) have settled, $\mathrm{TRIGGER_{DL1}}$ triggers BUF_DFFs and sends DATA and its complement $\overline{\mathrm{DATA}}$ to the write drivers, which are high-enabled tri-state buffers (Fig. 5.10). Since COL_SEL, W_EN and $\mathrm{TRIGGER_{DL1}}$ are all high, EN_{W_DR} is set high and the write drivers are turned on. The differential digital signals DATA and $\overline{\mathrm{DATA}}$ are then driven onto bit lines WBL and $\overline{\mathrm{WBL}}$ and written into the targeted SRAM cells. The write operation lasts for a duration of $\mathrm{T_W}$, set by the external control board. After an interval $\mathrm{T_W}$, $\mathrm{TRIGGER_{DL1}}$ goes low and EN_{W_DR} is set low, disabling the write drivers.

In read mode, $\mathrm{W\_EN_{IN}}$ is set low and $\mathrm{R\_EN_{IN}}$ is set high at the input. The timing diagram of critical signals in this mode is shown in Fig. 5.12. After the TRIGGER goes from low

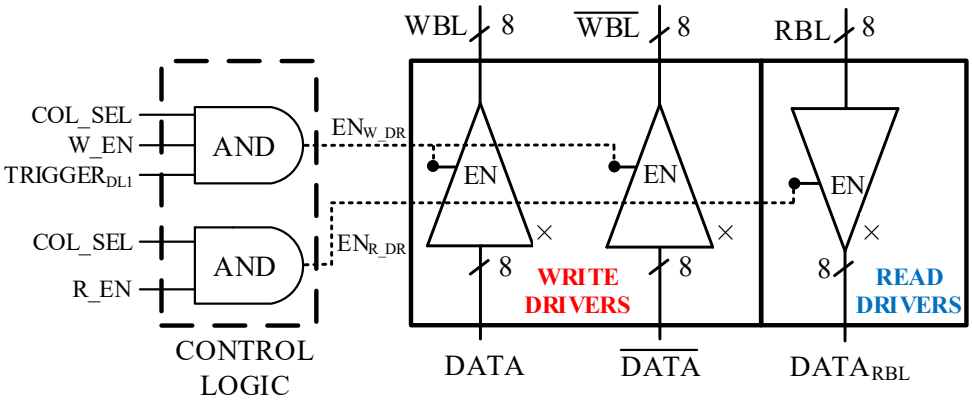

Figure 5.10: The write and read drivers.

to high, ADDR$_{IN}$ are again loaded into the DECODER block, which generates COL_SEL, read word line signal RWL and its complement $\overline{\text{RWL}}$. After COL_SEL and RWL/$\overline{\text{RWL}}$ are settled, the intended SRAM word contents are read out by the 4T read buffer inside the 10T cell, (Fig. 5.9). At the same time, the read drivers (high-enabled tri-state buffers, Fig. 5.10) are also turned on. After a delay T_2 (1 ns in our design), TRIGGER$_{DL2}$ goes high, triggering OUT_DFFs and DATA$_{OUT}$ are sent to the following block, i.e. the CT DAC.

The choice of delay values T_1 and T_2 is very important, especially the T_2 value in read mode. Note that in computation, the trigger signal in Fig. 5.12 is generated by the CT ADC block, shown in Fig. 5.1. This means that the trigger signal is event-driven and is not controlled by any external configurations. Only the input event density affects the rate of the trigger signals, i.e. the faster the varying of the input signal, the more often the trigger signal is generated. We would like T_2 to be as small as possible, because larger T_2 would increase the delay of the nonlinear function generator chain, increasing the phase shift of the loop when solving equations. On the other hand, T_2 must be large enough for the SRAM to read

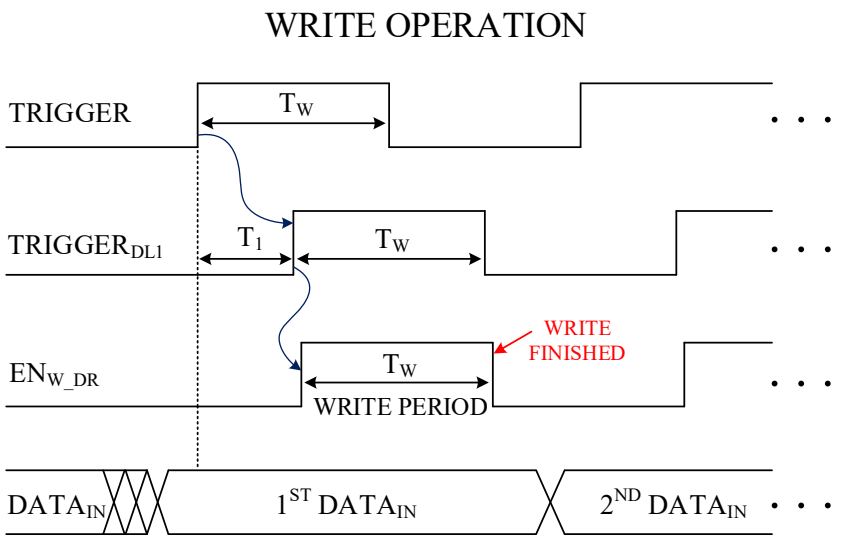

Figure 5.11: The timing diagram of critical signals for write operation.

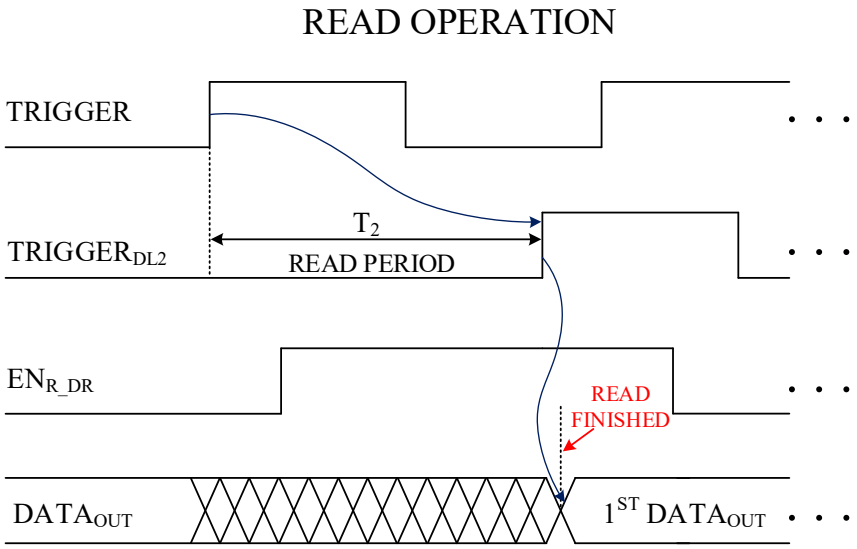

Figure 5.12: The timing diagram of critical signals for read operation.

out the data content from the register banks and allow the data to settle before triggering the output DFFs, shown in Fig. 5.8. If T_2 is not large enough, the entire read operation fails and sends wrong data to the next stage. Most importantly, we need to give enough flexibility

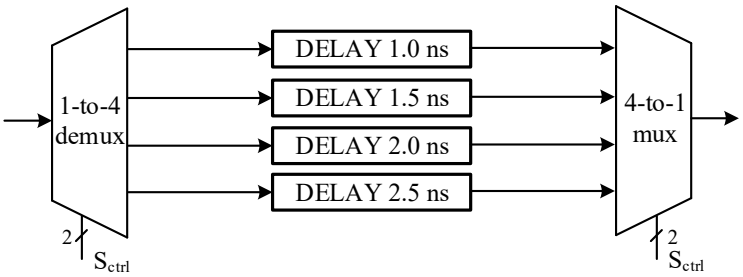

Figure 5.13: Delay lines arrays used in the SRAM block for flexibility.

to the T_2 values to counter the worst process, temperature, and voltage (PVT) variations. Thus, we make T_2 configurable as 1 ns, 1.5 ns, 2.0 ns, 2.5 ns, shown in Fig. 5.13. We use a demux in the front of the delay line arrays to save power, i.e. we prevent the trigger signal from passing through unused delay lines and causing unnecessary switching.

In simulation, the SRAM could function correctly for read/write operations when the trigger frequency goes up to 1 GHz speed. In our hybrid computing applications, we always use the CT ADC output trigger signal (max frequency is 16.1 MHz) as the trigger input for SRAM. When the input trigger rate is 16.1 MHz, the average power dissipation is 19.0 µW for read/write operations under random test conditions.

5.5 CT DAC design details

The CT DAC block is shown in Fig. 5.14. It has two selectable ranges (±2 µA and ±20 µA) for the full scale by configuring the bias block accordingly. A segmented current-steering topology is used for its core circuits (segmented-core DAC). After triggering the input DFFs, the 8-bit binary input DAC_{IN} is loaded; three MSBs are passed through a binary-thermometer

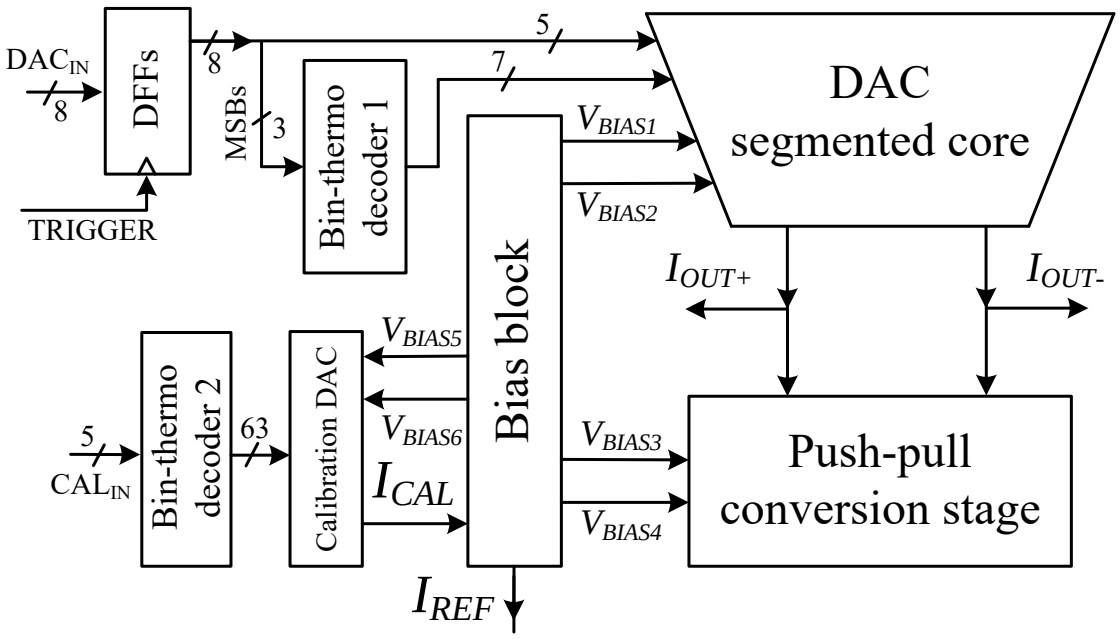

Figure 5.14: The architecture of the DAC block.

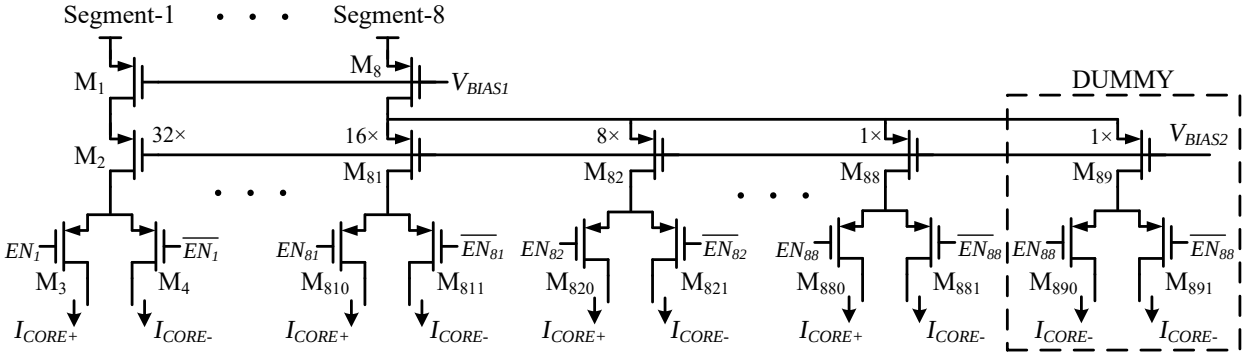

Figure 5.15: The segmented current-steering DAC core circuits.

decoder, whose outputs, together with the input's other 5 LSBs, are used to control the DAC segmented core block.

Fig. 5.15 shows the segmented-core circuit. We choose PMOS current arrays because PMOS transistors in our technology (TSMC 65 nm LP) have better matching properties

Segment-1 ~ 7	**Transistors**	**M_1**	**M_2**	**M_3**	**M_4**
	***W/L* (μm/μm)**	4/3	1/3	0.12/0.06	0.12/0.06
	Multiplicity	1	32	4	4
	V_T type	Normal			
Segment-8	**Transistors**	**M_8**	**M_{81}**	**M_{82}**	**M_{810}~M_{881}**
	***W/L* (μm/μm)**	4/3	1/3	1/3	0.12/0.06
	Multiplicity	1	16	8	2
	V_T type	Normal			

Table 5.4: Transistor sizes of the segmented current-steering DAC core circuits in Fig. 5.15.

than NMOS transistors (almost half the variations for PMOS by Monte Carlo simulation). Segments 1–7 are identical, with the sizing shown in Table 5.4. These sizes are chosen based on Monte Carlo simulations to guarantee that 95% (two sigma) of the devices yield mismatches smaller than 0.5 LSB current (7.8125 nA).

The last segment, Segment 8, is decomposed into eight binary-weighted branches. The switches that steer the currents are small devices (Fig. 5.15). The size of these switches should not be large because the parasitic capacitance C_{gd} causes glitches at the output current. To further reduce glitches at the DAC output, we need to be very careful with the layout of the control signal wires (EN_1, $\overline{\text{EN}}_1$, EN_{81}, $\overline{\text{EN}_{82}}$, etc.). To balance the delay between these control signals, we use dummy metal wires and NMOS capacitors in our design. Fortunately, the high-frequency glitches generated by the DAC are filtered out by the follow-up bandwidth-limited computing blocks, so they have a negligible effect on the overall solution.

The push–pull conversion block at the DAC's output stage (Fig. 5.16) has two NMOS current mirrors interfacing with the segmentation core. They convert the segmentation core's output to a class-AB, push–pull style current output to interface with the inputs of other

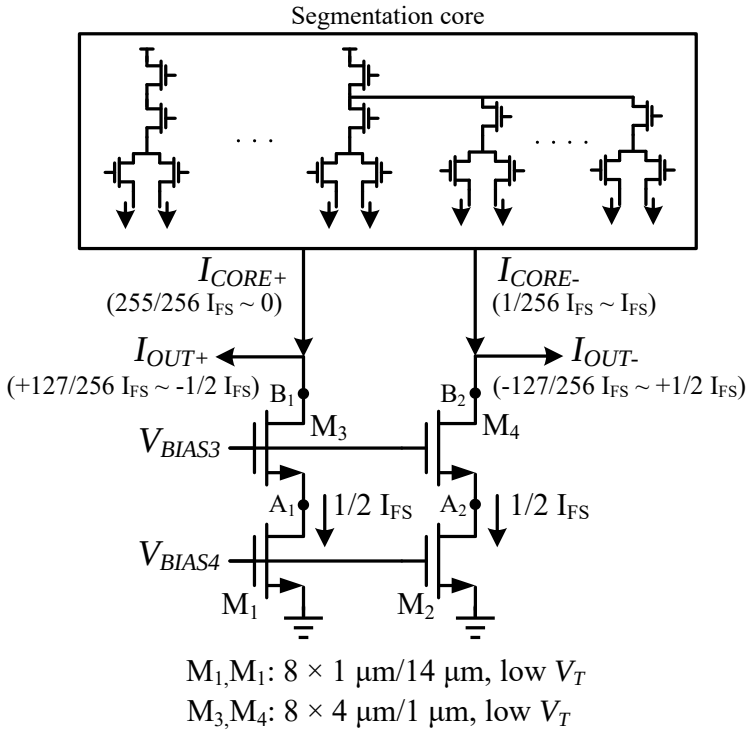

Figure 5.16: The output stage of the current-steering DAC.

analog blocks. The output of the segmentation core ($I_{\mathrm{CORE+}} - I_{\mathrm{CORE-}}$) is shifted by half of the full-scale current value (I_{FS}) by implementing two NMOS current sources with values of $0.5I_{\mathrm{FS}}$ (Fig. 5.16). Accordingly, the encoding scheme for the DAC's digital input is unsigned binary, with the code shifted by -128. For example, 1000 0000 input code generates zero output current, 0000 0000 generates differentially the negative full scale current ($-I_{\mathrm{FS}}$), and 1111 1111 will generate differentially positive full scale current off by $\frac{1}{128}I_{\mathrm{FS}}$. Note that NMOS current sources at the output stage must have very good matching, otherwise calibration circuits are needed to compensate for the mismatch. In our case, we use large NMOS devices to achieve good matching; speed is not a concern here because these NMOS current mirrors

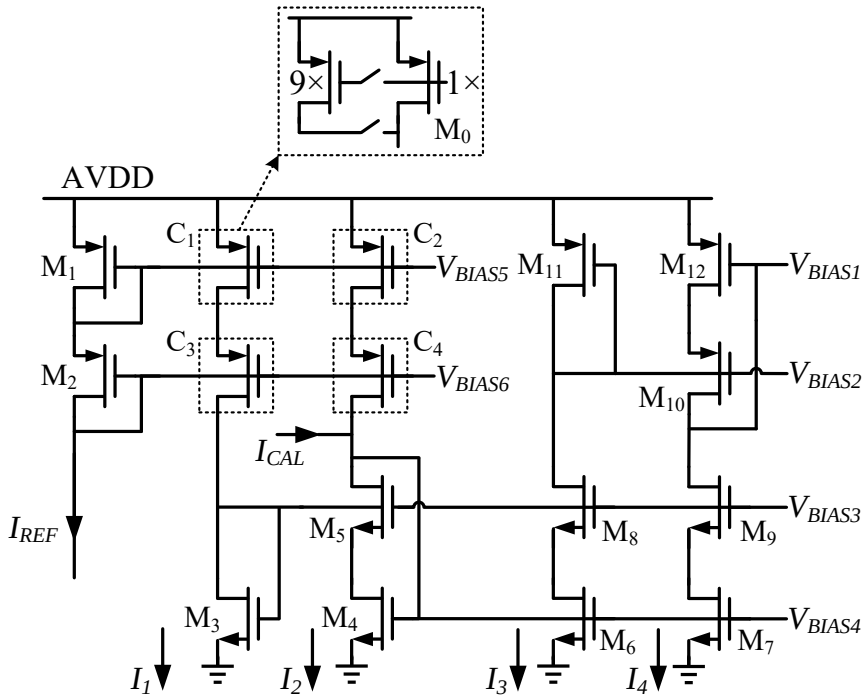

Figure 5.17: Circuit details of the bias block.

are isolated by the cascode devices. Another option would be the use of calibration circuits (e.g., six-bit current DACs), but they would take up a much more area.

Fig. 5.17 presents the circuit details of the bias block. It takes a 1 µA (I_{ref}) current and mirrors it to four other branches, generating bias voltages for other blocks. I_1, I_2, I_3, and I_4 are equal and could be configured as 250 nA (for the 2 µA full-scale range, there are eight segments in total, with each segment of 250 nA) or 2.5 µA (for the 20 µA full-scale range) by setting the composite devices C_1–C_4. Gain calibration is implemented by injecting calibration current I_{CAL} (from the calibration DAC block) into the I_2 branch, because I_3 and I_4 values are decided by I_4 through the NMOS mirrors (M_4, M_6, and M_6). Table 5.5 shows the transistor sizes used.

Unit Transistor M_0 in	**C_1**	**C_2**	**C_3**	**C_4**	**N/A**	**N/A**
***W/L* (µm/µm)**	2/8	1.85/8	8/0.5	8/0.5	N/A	N/A
Multiplicity	1	1	1	1	N/A	N/A
V_T type	Low				N/A	N/A
Transistors	**M_1**	**M_2**	**M_3**	**M_4**	**M_5**	**M_6**
***W/L* (µm/µm)**	2/8	8/0.5	0.3/9	1/14	4/1	1/14
Multiplicity	4	4	1	2	2	2
V_T type	Low					
Transistors	**M_7**	**M_8**	**M_9**	**M_{10}**	**M_{11}**	**M_{12}**
***W/L* (µm/µm)**	1/14	4/1	4/1	1/3	0.5/6	4/3
Multiplicity	2	2	2	32	1	1
V_T type	Low			Normal		

Table 5.5: Transistor sizes of the bias block.

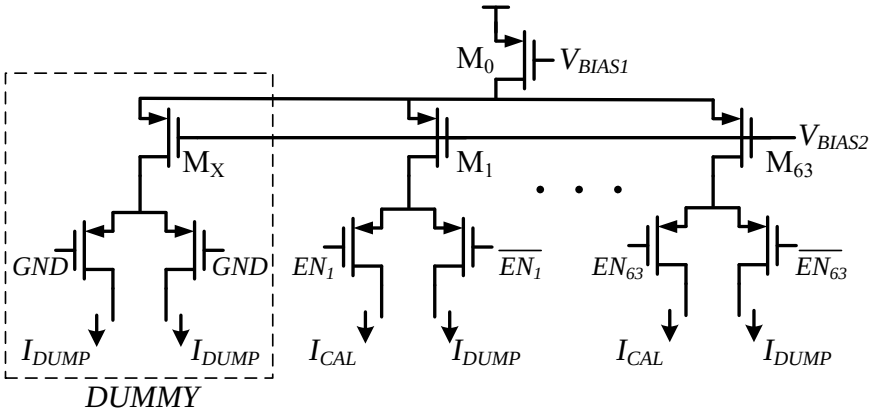

Figure 5.18: Circuit details of the calibration block.

The calibration DAC block is a unary, current-steering DAC (Fig. 5.18). Sixty-four branches divide the current from M_0. I_{DUMP} goes to a diode-connected NMOS device, and I_{CAL} is used to calibrate the bias block. For this gain-calibration DAC, matching between the 64 branches is not important; monotonicity here matters most because we need to cover the desired value for the bias block 250 nA/2.5 µA. Table 5.6 shows the transistor sizes used

Transistors	M_0	M_X	M_1~M_{63}	PMOS switches
***W/L* (μm/μm)**	0.24/8	0.2/1	0.2/1	0.2/0.2
Multiplicity	1	1	1	1
V_T type	Low			

Table 5.6: Transistor sizes of the calibration block in Fig. 5.18.

for this calibration DAC.

5.6 Full chip layout

Fig. 5.19 shows the full-chip layout with key blocks annotated. Table 5.7 shows the detailed area information of each block.

Block	Fanout	Integrator	Multiplier	CT ADC	CD DAC*	CT SRAM	6-bit calibration DAC
Width × Height (μm × μm)	150 × 120	275 × 150	310 × 160	300 × 185	133 × 118	220 × 130	90 × 27
Area (mm^2)	0.018	0.041	0.050	0.056	0.016	0.029	0.00243
Area percentage for calibration DACs	40.5%	11.8%	14.7%	13.1%	10.4%	N/A	N/A

*The calibration DAC used in CT DAC block is different from others; its dimension is 25 μm × 65 μm

Table 5.7: Layout dimensions and areas of each block, plus the area percentage occupied by the calibration DACs inside each block.

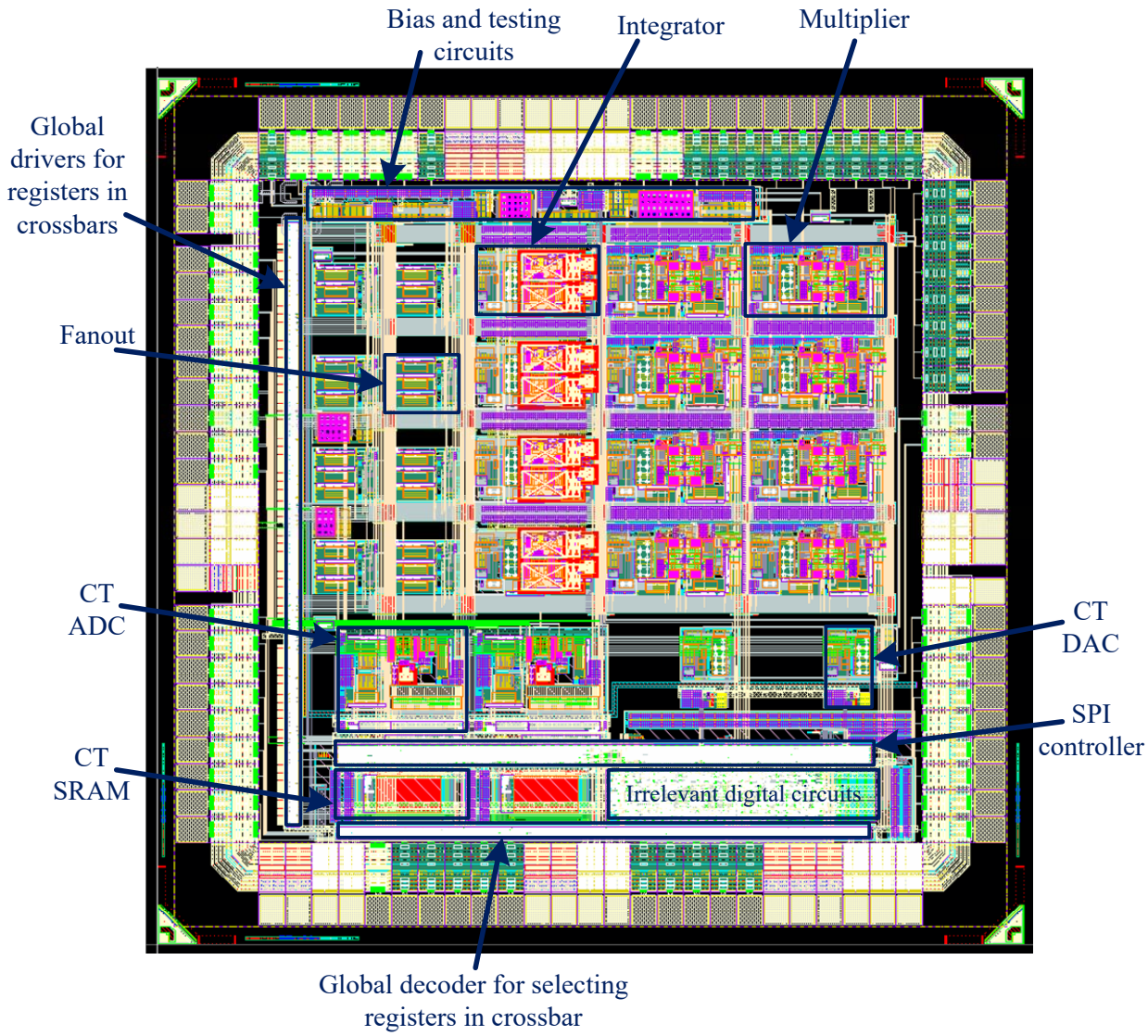

Figure 5.19: Full chip layout picture with key blocks annotated.

Chapter 6

Implementation Details of a 16th-Order CT Hybrid Computing Chip

The fourth-order chip discussed in the previous chapter was fabricated and tested. We were very lucky to have all the blocks functioning correctly the first time. (Measurement results will be shown in the next chapter.) Then we decided to move on; we scaled the system up to the 16th order. That is, the new system contained 16 integrator blocks. To guarantee the success of this larger system, we reused the designs of individual blocks from the first chip as described in Chapters 4 and 5. Furthermore, we retained the microarchitecture of the fourth-order system shown in Fig. 3.6 and integrated four such blocks into a 16th-order system. The new features and designs are discussed in the following sections.

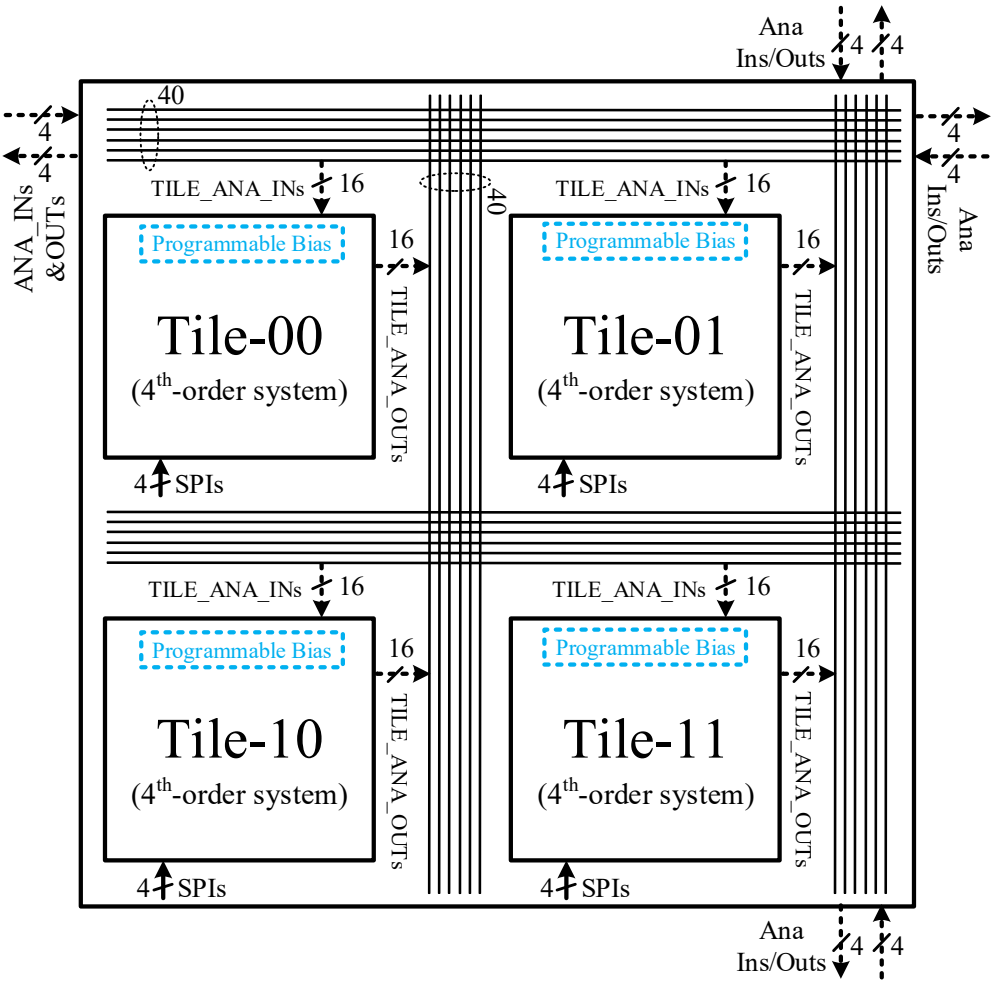

Figure 6.1: The architecture of a 16th-order hybrid computing system.

6.1 Overall architecture

Fig. 6.1 shows the overall architecture of the 16th-order system. Each fourth-order unit, with the microarchitecture and orientation shown in Fig. 3.6, serves as a "tile" in the scalable architecture. Each tile is upgraded to have 16 analog inputs, 16 analog outputs, one programmable bias block, two more CT DAC blocks (not shown, now four CT DACs per tile in total). Each tile is programmed by the its own SPI controller and has a pair of eight-bit

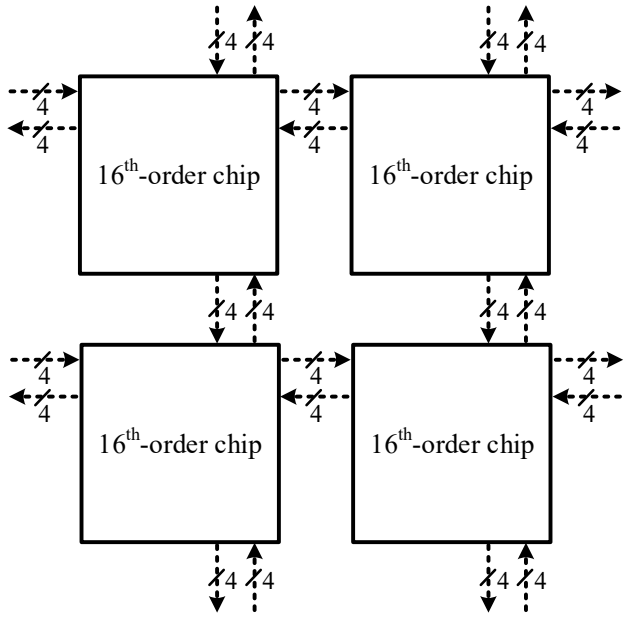

Figure 6.2: Two-by-two array of the 16th-order chips for 64th-order differential equations.

digital I/Os (see TDIs and TDOs in Fig. 3.6. Four pairs of digital I/Os are multiplexed at the global level and the chip has only one pair connected to the outside.

Each tile can be connected to any other tile through the global crossbars. As shown, there are 40 global horizontal wires (on each row) and 40 global vertical wires (on each column), 32 of which are connected to the tiles; the wires not connected to tiles contribute to the chip's 16 analog inputs and outputs. At each intersection sits a global crossbar that controls the signal flows between tiles; each tile is responsible for the programming of the adjacent global crossbars, the one to the top-right corner of each tile. The orientation of the analog inputs and outputs is designed for multiple-chip integration onboard to solve larger problems. For example, to build a 64th-order system, we could just put four chips in a two-by-two array, as shown in Fig. 6.2. The analog I/Os would be connected to the adjacent neighbors, so that

any two adjacent chips would have eight connected analog channels (four inputs and four outputs). It is worth mentioning that, when we integrate multiple chips on PCBs to solve higher order equations, the distribution of SPI signals to each chip may become a problem: the SPI signals received by each chip need to be well synchronized, otherwise the blocks on different chips would start computation at different times, causing solution errors. Sending a start signal to various chips on the board is a problem similar to clock distribution.

The number of analog I/O pins for each tile and for the whole chip was decided based on mapping a series of 2-D PDEs, the target application for this 16th-order chip. For example, with the I/O connections shown in Fig. 6.1, we can map a 2-D heat-diffusion equation, a 2-D Burger's equation, a 2-D wave equation, etc. onto our system. Two-D PDEs with more than 16 unknowns can be mapped to several chips, with the interconnections described earlier.

Each tile has separate power domains for the analog and digital blocks to reduce interference. Nevertheless, they also share some power rails to save pins. For example, Tile 00 shares analog power rails with Tile 01, and Tile 10 shares digital power rails with Tile 11. Also, all the testing structures, used for probing internal signals in the first chip, are removed.

6.2 Tunable global bias blocks

To further save pads and pins, each tile has just two bias-current ports: 1 µA going into and 1 µA going out of the tile. The two bias currents are received by NMOS and PMOS programmable biasing blocks, respectively. Fig. 6.3 shows the NMOS programmable current source array, which generates bias currents for functional blocks in the same tile. The pro-

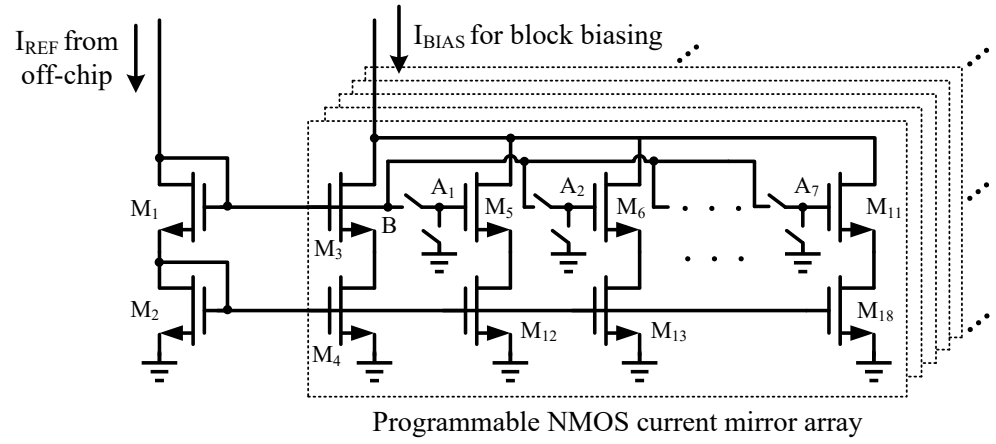

Figure 6.3: Schematic of one of 32 programmable NMOS bias-current source for biasing the computing block in one tile (fourth-order system).

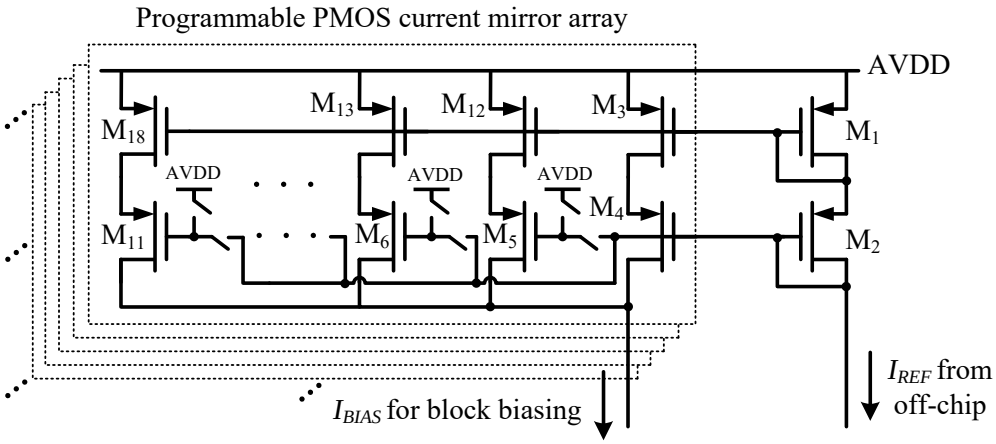

Figure 6.4: Schematic of one of 18 programmable PMOS bias-current source for biasing the computing block in one tile (fourth-order system).

grammable bias block has three-bit programmability (binary to thermometer decoding) for each generated bias current (Fig. 6.3), to compensate for temperature and process variations: Code 100 would generate the normal value of 1 µA, and step size is 65 nA. Fig. 6.4 shows the design detail of the PMOS programmable current source array, which uses the same

Transistors	M_1	M_2	M_3	M_4	$M_5 \sim M_{11}$	$M_{12} \sim M_{18}$
W/L (µm/µm)	1.2/12	1.2/12	1.2/12	1.2/12	0.2/12	0.2/12
Multiplicity	3	3	3	3	1	1
V_T type	Low					

Table 6.1: Transistor sizes used in the NMOS mirror array in Fig. 6.3.

Transistors	M_1	M_2	M_3	M_4	$M_5 \sim M_{11}$	$M_{12} \sim M_{18}$
W/L (µm/µm)	3/12	12/12	3/12	12/12	2.4/12	0.6/12
Multiplicity	3	3	3	3	1	1
V_T type	Low					

Table 6.2: Transistor sizes used in the PMOS mirror array in Fig. 6.4.

architecture as the NMOS one shown in Fig. 6.3. The whole chip now has eight pins reserved for bias currents, two for each tile. Transistor sizes used for NMOS and PMOS mirror arrays are shown in Tables 6.1 and 6.2, respectively.

6.3 Instruction set and address-space mapping

The instruction sets are also upgraded to 24 bits from the previous 18 bits used in the fourth-order chip. The advantage of having the instruction word length be a multiple of eight bits is that commercial microcontrollers usually support dedicated high-speed SPI interfaces (usually called USART ports), which are much faster than just toggling the universal digital I/Os to construct the correct timing for the SPI signals. The virtual hierarchy of our 16th-order chip is defined as fabric, tile, slice, and block, which will be explained next.

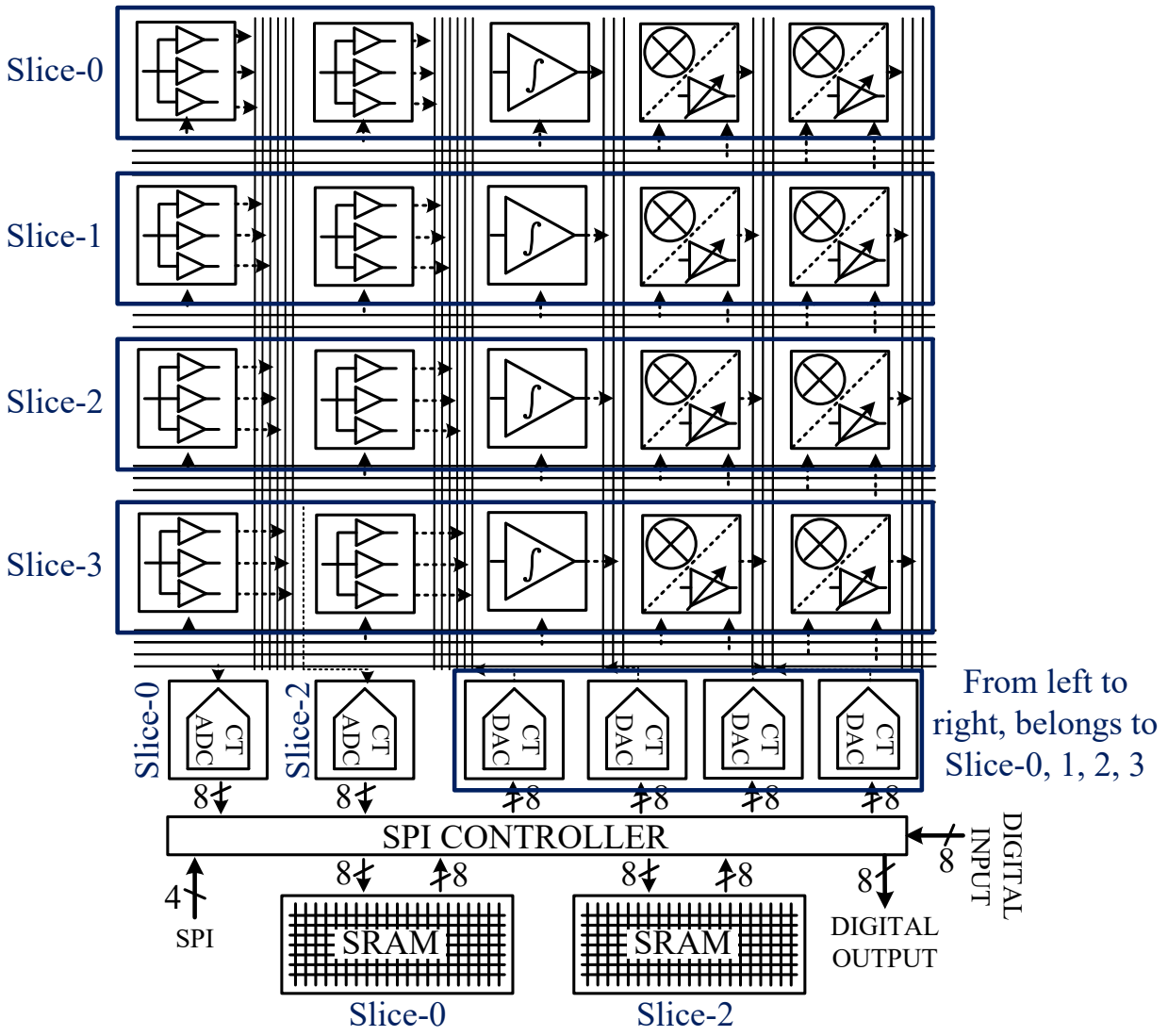

Figure 6.5: Illustration of how one tile is divided into multiple slices.

Each 16th-order chip is called a fabric. The first four bits of the 24-bit instruction word are used to identify which fabric to program and inside that fabric, which tile (the fourth-order system) and the associated global crossbars to program (Fig. 6.1). For example, we can name the chips Fabric 00, Fabric 01, etc. to facilitate multiple-chip programming. The following eight bits are used to locate which slice to use and inside that slice, which functional

block is used for programming. A slice contains one row of analog blocks, shown in Fig. 6.5, and several other mixed-signal blocks. However, since there are only two ADCs and two SRAMs in total, the two slices containing ADCs and SRAMs are called *full*, and other two are called *half*. For example, there are four slices in total. One full slice has two fanout blocks, one integrator block, two multiplier blocks, one ADC block, one SRAM block, and one DAC block. The last 12 bits of the instruction word carry the parameter information for the individual block and signal routings. This hierarchy of fabric, tile, slice, and block adopts the object-oriented design methodology and inheritance features of the C++ language, which is beyond the scope of the thesis topic and thus will not be discussed here.

6.4 Layout for the second chip

Fig. 6.6 shows full-chip layout of the 16th-order chip design with key blocks annotated.

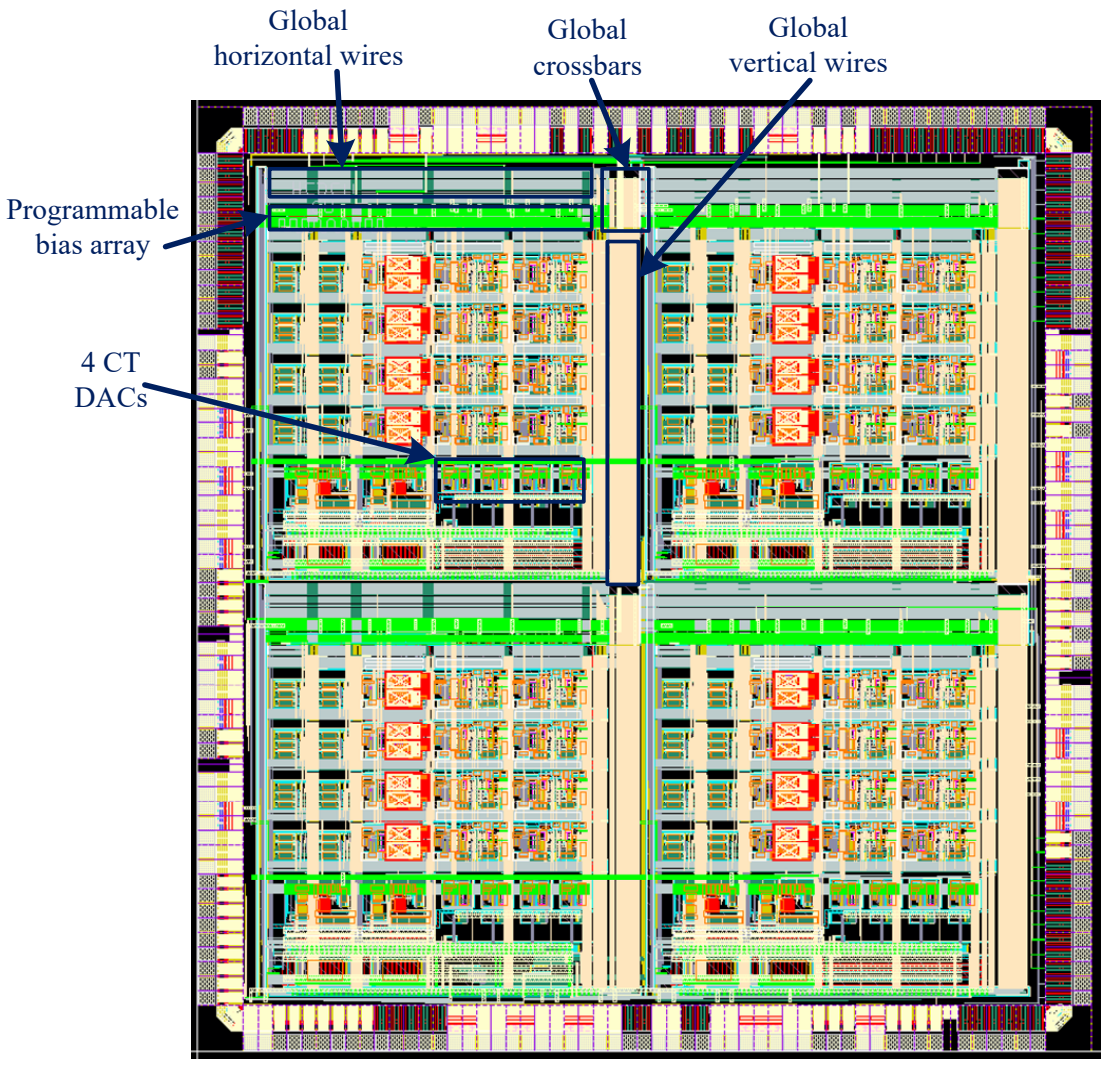

Figure 6.6: Full-chip layout picture of the 16th-order system.

Chapter 7

Measurement Results

7.1 Die photos and packaging considerations

The two test chips were fabricated in TSMC 65 nm LP CMOS technology and were successfully tested. We also tried the TSMC 65 nm GP CMOS process in simulation, but its large gate leakages are unsuitable for low-power analog circuits.

Fig. 7.1 shows the die photo of the 1.96 mm × 1.96 mm (3.84 mm^2) fourth-order system. The active area is about 2.0 mm^2, including testing and general programmability circuits. The area would be considerably smaller if some special-purpose computation tasks were targeted. As can be seen from Fig. 7.1, all the analog blocks share the same power rails, which is similar to "star"-style connections. The CT ADCs and CT DACs have separate power rails. All the digital blocks share the same power grid.

Fig. 7.2 shows the die photo of the 3.7 mm × 3.9 mm 16th-order system, where the fourth-

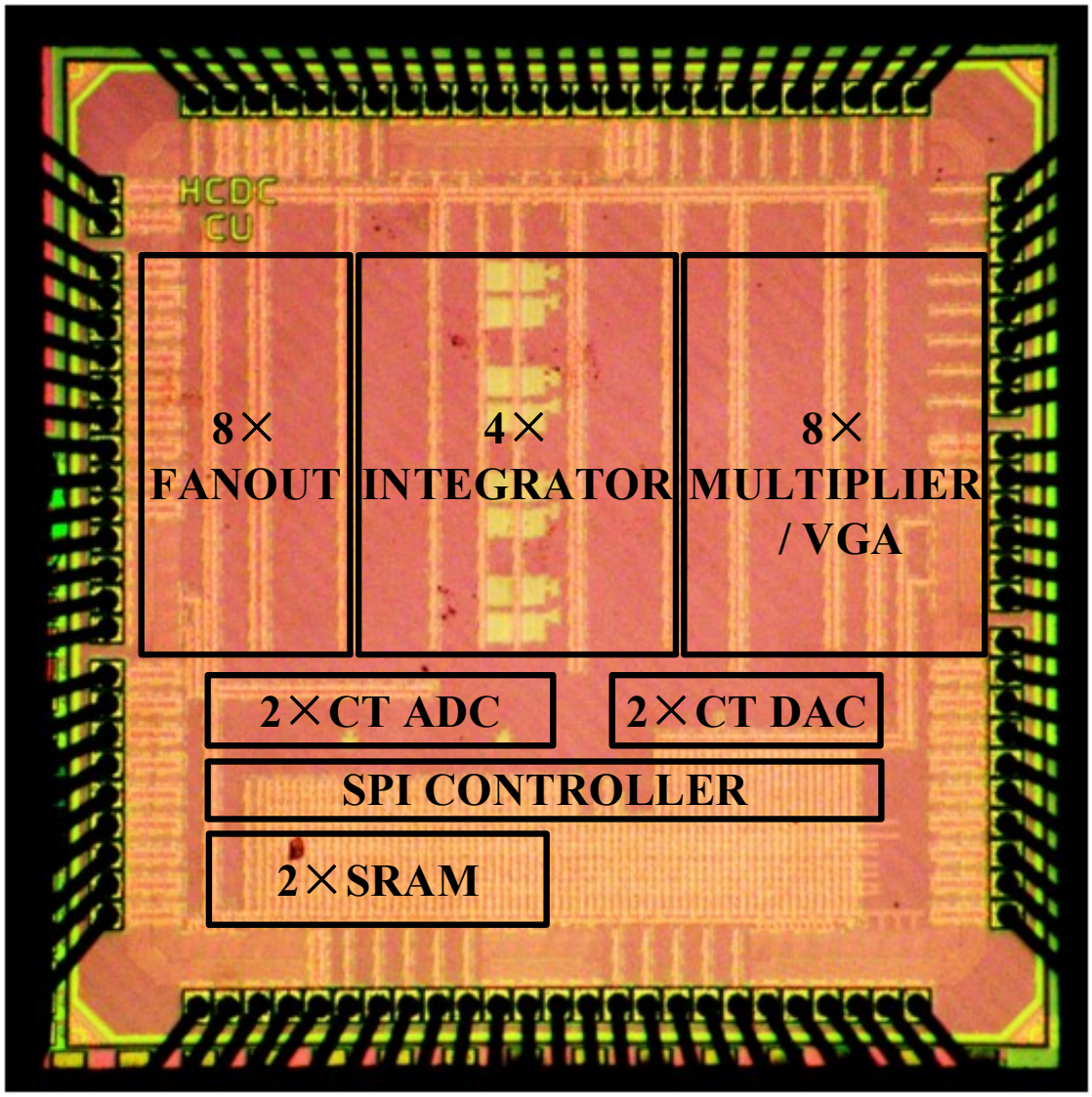

Figure 7.1: The die photo of the fourth-order hybrid computing unit.

order systems are oriented in a two-by-two array. To minimize parasitic inductance and capacitance, we chose the QFN package (80 pins, with thermal-ground plane) for the fourth-order chip. However, no QFN package solution is available for the 16th-order chip, as it has over 200 pads, over 120 of which are functional pads (not for power or ground rails) and need

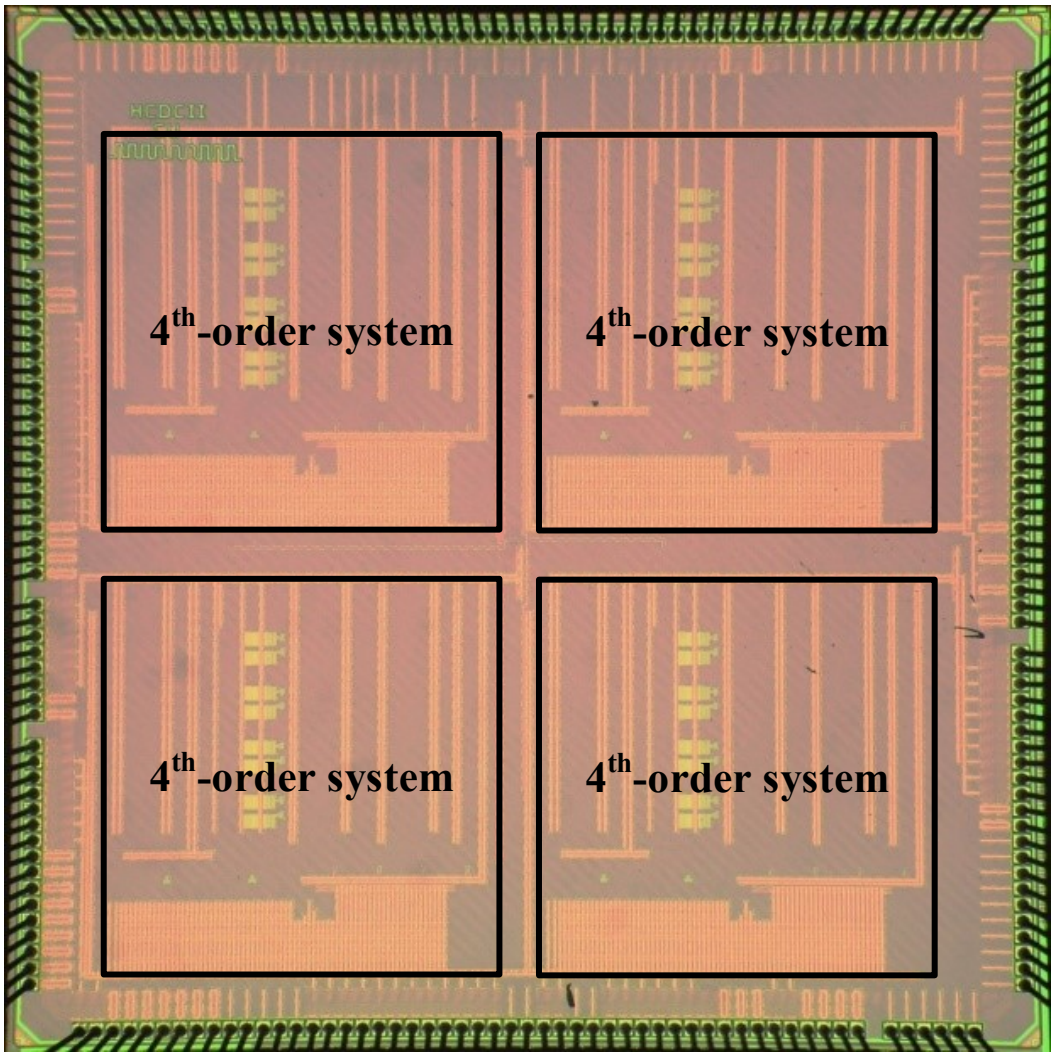

Figure 7.2: The die photo of the 16th-order hybrid computing unit.

to be bounded to package pins. The BGA package would be a good choice for the 16th-order chip, but its complexity and cost are prohibitive. Finally, after some searching, we chose an unusual but economic 144-pin LQFP package with a thermal-ground plane. Though the long leads of this LQFP package have more parasitics than those of the QFN, it provides a ground plane as well, which saves many ground pins.

We would like to make clear that, in the following sections, all of the reported results related to individual blocks (e.g. the power dissipations of each block, the nonlinearity of the fanout block, etc.) were measured on the fourth-order chip. Since the 16th-order chip reuses exactly the same blocks, we did not remeasure them. Instead, we focused on solving higher-order problems with the 16th-order chip. Thus, all the ODE/PDE equations with no more than four unknowns/state-variables were measured on the fourth-order chip; equations with more than four unknowns were solved on the 16th-order chip, unless otherwise noted.

7.2 Testing environment, chip interfaces and programming language

Fig. 7.3 shows the testing environment setup for the fourth-order chip. The test board for the 16th-order chip is shown in Fig. 7.4. We choose Arduino boards because of its popularity and open-source software. The Arduino Due boards provide all the necessary interfaces and functionalities we need for testing: eight ADC channels for measuring analog signals, two DAC channels for generating analog signals, over 40 digital I/Os for generating control signals and measuring chip data. The only limitation is the speed of the Arduino Due board. It can only toggle the digital I/O up to 500 kHz (after implementing some specific call functions to the I/O pins, which will not be discussed here). However, this is sufficient for the purposes of this project. A more advanced FPGA board would be needed if we aim for much higher SPI toggling speed, such as 100 MHz.

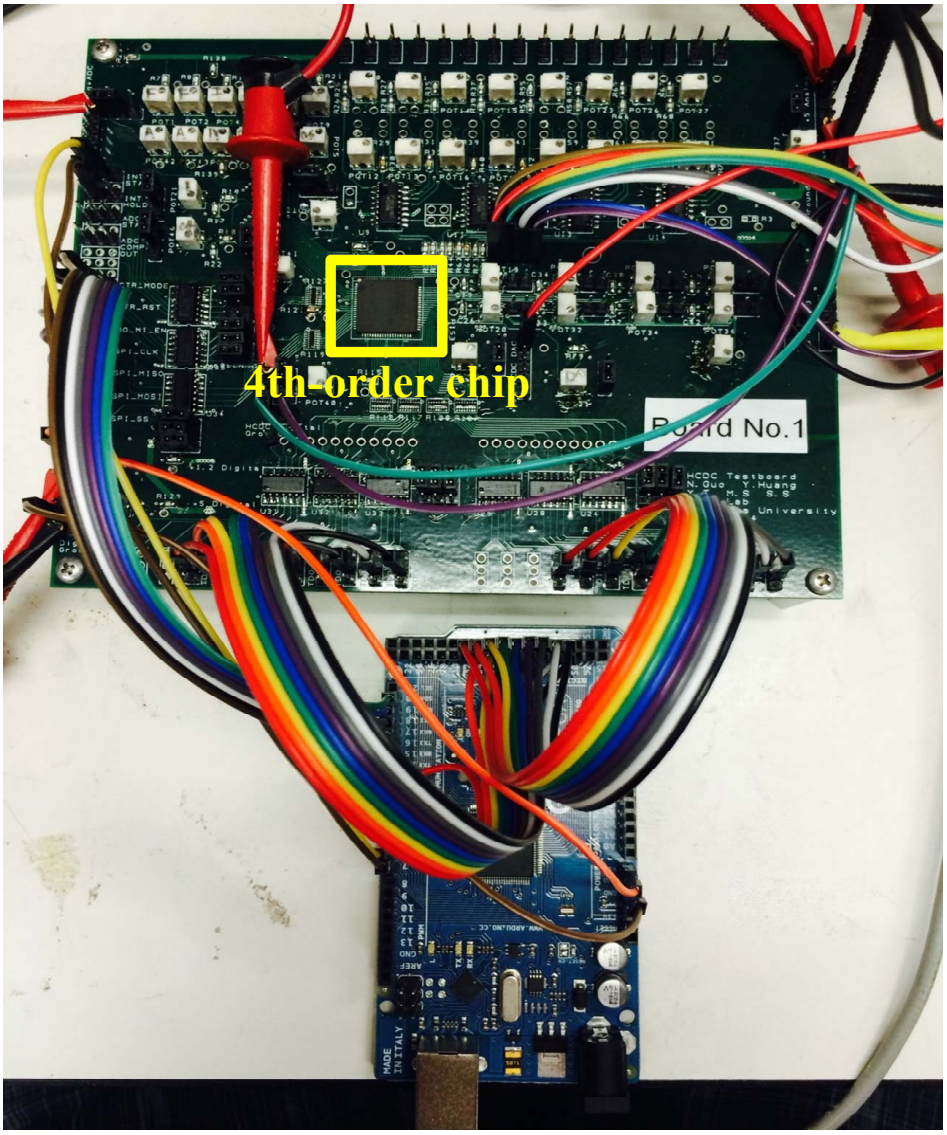

Figure 7.3: The testbench of the fourth-order hybrid computing chip.

Since our hybrid computing chip's analog interfaces operate in current mode, I–V and V–I converters are needed to interface with the Arduino board's voltage-mode ADC/DACs. Due to the small current (7.8 nA for 1 LSB) coming out of the chip, we need to use op amps with at least pA-level input bias current and adequate bandwidth at the same time. We

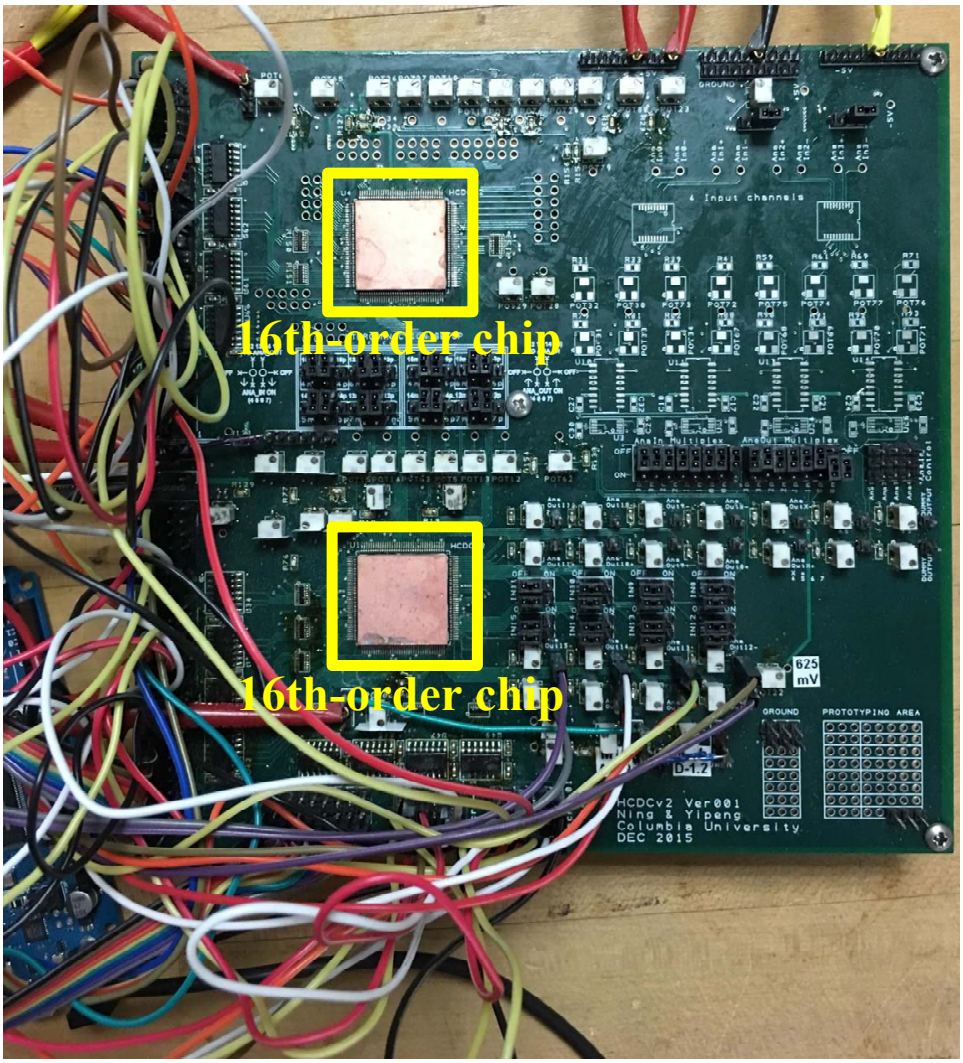

Figure 7.4: The testbench of the 16th-order hybrid computing chip.

chose AD8512 from Analog Devices to implement the I–V converter, as shown in Fig. 7.5. We choose 600 kΩ for the feedback resistors so we could convert the ±2 µA range differential currents to a ±1.2 V range. The V–I converters were implemented with Texas Instruments' LM13700 chips, which provide a similar push–pull, class-AB output current interface as the analog blocks' on our chip (see LM13700 datasheet for more details). To keep the nonlin-

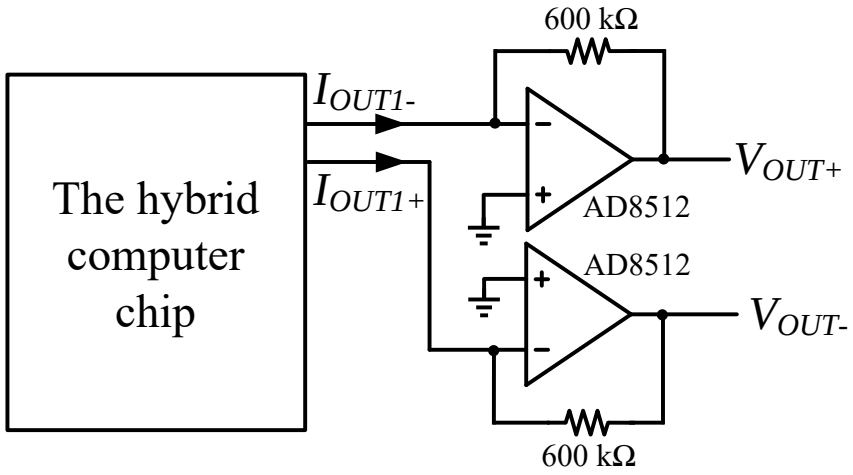

Figure 7.5: I–V converter implemented by AD8512.

earity low, we configured the LM13700 to convert $\pm 25\,\text{mV}$ peak differential inputs to $\pm 2\,\mu\text{A}$ peak differential currents. We passed these differential currents into the I–V converters implemented with the AD8512, and the measured voltage outputs have $THD = 0.1\%$, lower than the eight-bit resolution we targeted for our blocks. This guarantees that our testing instruments' nonidealities would have negligible effects on the measurement results.

We used a C++ style, object-oriented, customized language [12] to program the chips. We primarily followed the computing diagrams for each equation (as will be seen later in Chapter 8) to write the corresponding programming codes. Concise and higher-level symbolic math expressions, like those used in Mathematica, can also be taken as input, but this method is limited by our compiler to only a few specific equations and thus will not be discussed here. For example, we could connect one block's output to another block's input, then to the chip's analog output by writing the following codes.

int0 = fabric.chips[0].tiles[0].slices[0].integrator;

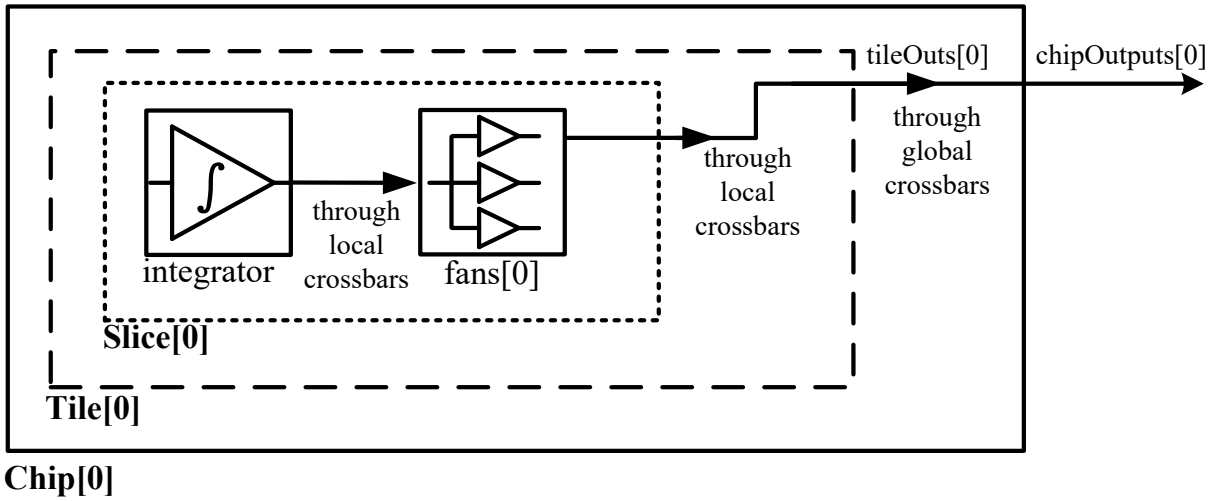

Figure 7.6: A diagram illustration of the signal paths set by the programming codes.

fan0 = fabric.chips[0].tiles[0].slices[0].fans[0];

tileout = fabric.chips[0].tiles[0].slices[0].tileOuts[0];

chipout = fabric.chips[0].tiles[0].slices[0].chipOutputs[0];

conn0.setConn(int0→out0, fan0→in0);

conn1.setConn(fan0→out1, tileout→in0);

conn2.setConn(tileout→out0, chipout→in0);

Fig. 7.6 illustrates the signal paths configured by the above code. The first four lines of codes define four different objects: an integrator block named as "int0," a fanout block named "fan0," an analog output of the tile named "tileout," and a chip output named "chipout." As shown, each lower-hierarchy item is a member of a higher-hierarchy item (similar to the C++ class type). The last three lines define the connections between the objects, which follows the rules that the first object's output is connected to the second object's input. Thus, the

above codes state the following: int0's output is connected to fan0's input, fan0's out1 is connected to tileout's input, tileout's output is connected to chipout's in0. Please note that there are three output ports of the fanout block (named out0, out1, and out2). The analog inputs and outputs of a tile or chip are treated as functional blocks, even though they are actually just analog crossbars. This helps standardize the codes and facilitates interpretation and debugging.

7.3 Calibration procedures

As discussed in the design details of each building block (Chapters 4 and 5), there are many calibration circuits to compensate for analog imperfections and improve computing accuracy. The calibration routine is performed by the Arduino Due microcontroller, although they could also be implemented as ASIC circuits on the chip. When in calibration mode, the whole system is configured as illustrated in Fig. 7.7. Each analog block is connected to the chip's output, one by one, and measured by the microcontroller's ADCs. After measuring the present offsets, the binary search algorithm decides a new calibration code to be sent to the chip through the SPI interface. This new instruction is decoded by the global driver and global decoder, which then writes the local SRAM cells that store the calibration codes. The calibration takes about 4 ms for the fourth-order chip and about 20 ms for the 16th-order chip, when the SPI clock is running at 20 MHz.

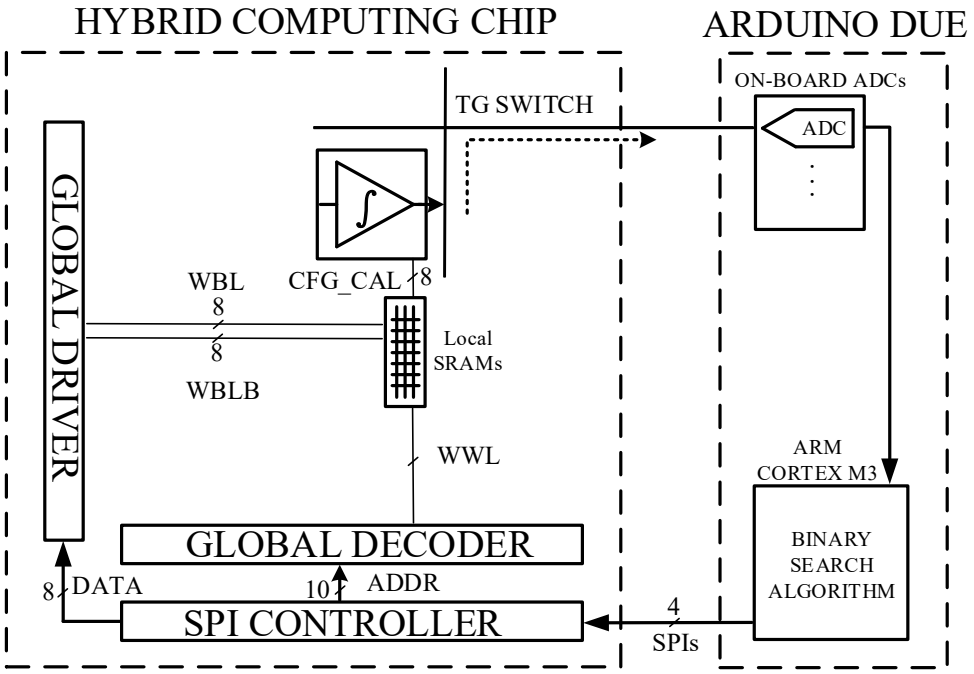

Figure 7.7: The global signal paths for calibrating each block and related functional blocks involved.

7.4 Measurement results

7.4.1 Calibration

Table 7.1 shows output offsets of analog blocks measured before and after calibration. The calibration circuits greatly reduced the offsets caused by device mismatches.

The accuracy of solving differential equations is also greatly improved by offset calibrations, as shown in Table 7.2 for two examples. After offset calibrations, we have a solution error smaller than 2%.

Block type	Output offsets* before calibration (nA)	Output offsets* after calibration (nA)
Fanout	109	4
Multiplier	57	6
Integrator	42	4

*Average values of all same type of blocks over one chip for +/-2μA range

Table 7.1: Analog offsets minimized by calibration.

ODE's physical background	Nonlinearity involved	RMS error* (uncalibrated)	RMS error* (calibrated)
Large angle motion of pendulum	Trigonometric function (sine)	7.3%	1.5%
Mass-spring dampers with Coulomb friction	Sign function	18.0%	0.5%

*Relative to full sale

Table 7.2: Solution accuracy improved by calibration.

7.4.2 Nonlinear function generator

We chained together one CT ADC, one CT SRAM and one CT DAC as a nonlinear function generator, where the DAC's analog output signal is a nonlinear function of ADC's analog input signal.

Shown in Fig. 7.8 is a screen capture of critical signals on oscilloscope when the nonlinear function generator is performing a sine function lookup. The blue ramp signal on channel 3 is the input x from $-\pi$ to $+\pi$; green and yellow curves are the differential outputs of the DAC, already converted to voltage signals. The purple curve is the difference (calculated by

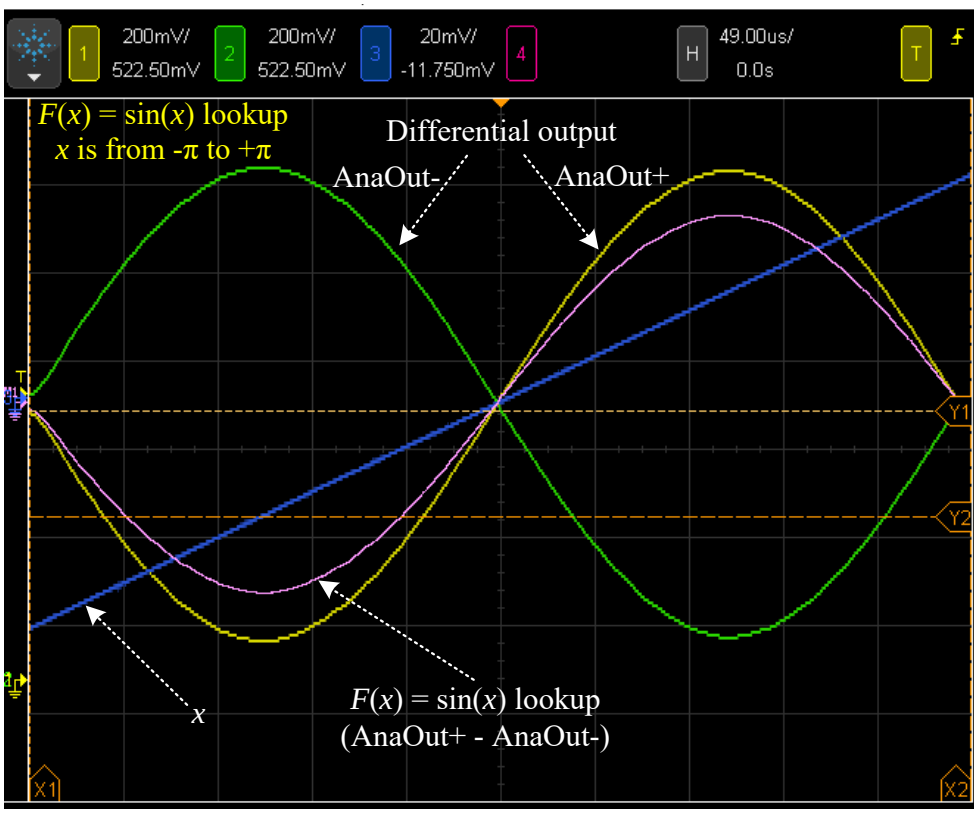

Figure 7.8: The screenshot of oscilloscope measurement results when the nonlinear lookup table is configured for sine function lookup.

the oscilloscope) between the yellow and green curves, which is obviously a sine function of x.

Two examples of the nonlinear function generator errors compared to ideal values are shown in Fig. 7.9; the full cycle ($-\pi$ to $-\pi$) sine function and sigmoid function table lookups have normalized RMS errors of 0.56% and 0.76% respectively. The total power dissipation of the nonlinear function generator is signal-dependent, decreasing as the table lookup activity decreases, as shown in Fig. 7.10.

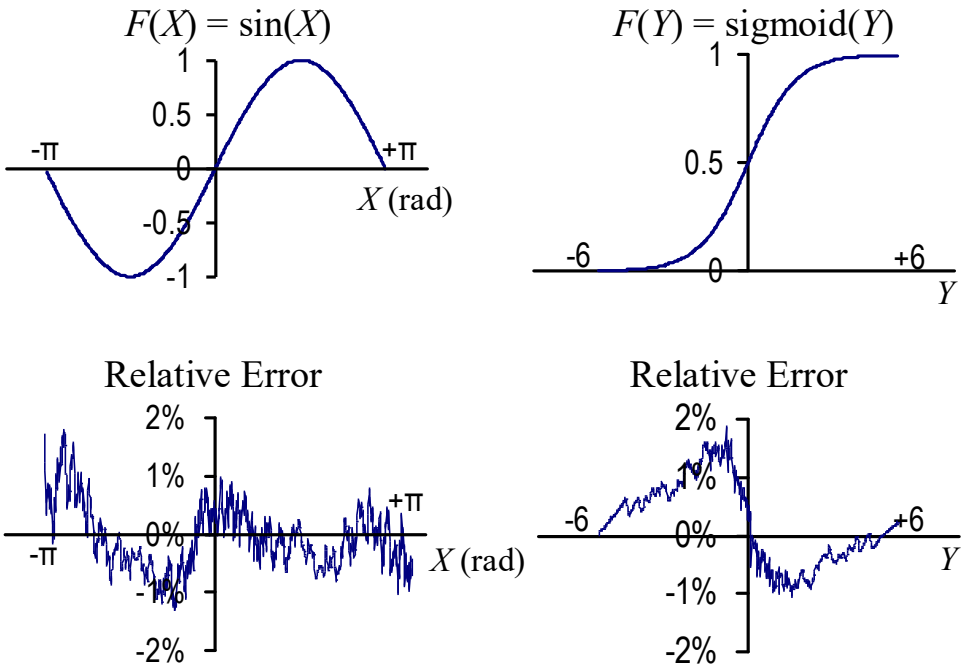

Figure 7.9: Nonlinear function lookup examples.

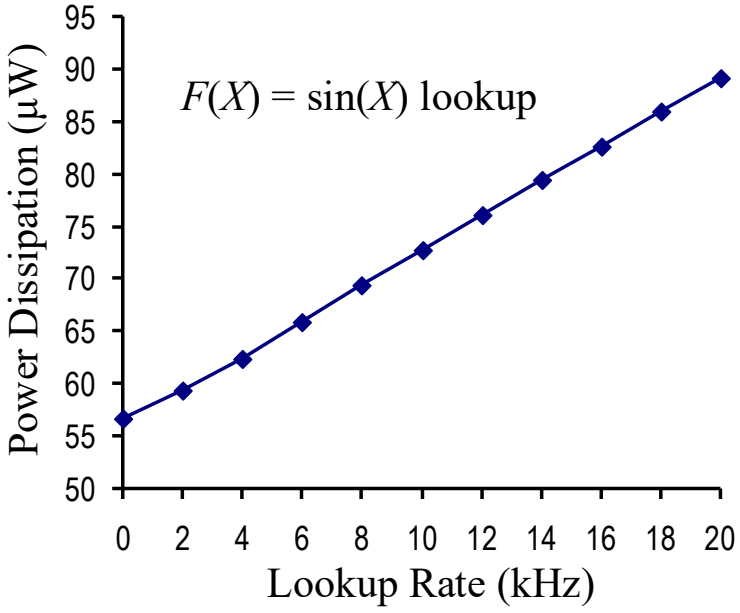

Figure 7.10: Nonlinear function generator's power dissipation is lookup rate dependent.

Supply voltage	1.2V
Technology	TSMC 65nm LP
Die area / active area	3.8 mm^2 / 2.0 mm^2
Number of integrators	4
Number of multipliers/VGA	8
Number of fanout blocks	8
Number of CT ADC	2
Number of SRAM	2
Number of CT DAC	2
Number of analog inputs/outputs	4/4
Digital input/output word length	8 bits
Programming interface	SPI
Integrator nonlinearity[1]	0.44%
Fanout nonlinearity[2]	0.13%
VGA/Multiplier nonlinearity[3]	0.15%
ADC+DAC SNDR 1kHz/20kHz	46.3dB/53dB
DAC DNL/INL	0.73LSB/0.67LSB
$f_{max,computer}$	20 kHz
$FOM_{computer}$	14.1 nJ

Block name	Power
Fanout[4]	37 µW
Integrator[4]	28 µW
Multiplier[4]	61 µW
VGA[4]	49 µW
CT ADC[5]	54 µW / 82 µW
CT DAC[5]	4.6 µW / 15 µW
SRAM[6]	20 µW
Analog circuits leakage	6.7 µW
Digital circuits leakage (estimate)	85 µW

[1] 2µA range, full-scale 20kHz sine input.
[2] RMS deviation from unity gain over +/- 85% full scale.
[3] RMS deviation from unity gain over +/- 85% full scale in VGA mode.
[4] 2µA range, 20kHz full-scale sine input.
[5] 2µA range, 1kHz / 20kHz full-scale sine input.
[6] 20kHz full-scale sine digital input from ADC; SRAM programmed as a linear lookup table.

Table 7.3: Fourth-order hybrid computing unit performance.

7.4.3 Key performance summary of the hybrid computing chip

A performance summary is shown in Table 7.3. The measured nonlinearities and noise are consistent with our intended eight-bit accuracy.

7.4.4 Comparison to the prior art

To compare our work to the prior art as reported by Cowan [1], we use one macro block of that work, which contains a similar number of functional blocks as our hybrid computing chip, as shown in Table 7.4. The increased functionality is apparent, as is the lowering of the power dissipation and $\text{FOM}_{\text{computer}}$ by more than an order of magnitude.

	One macro in [7]	Our 4th-order chip
Supply voltage	2.5V	1.2V
Technology	250nm CMOS	65nm CMOS
Active area (estimate)	6.3 mm^2	2.0 mm^2
Number of function blocks	25	26
Power with all blocks on (estimated)	18.8 mW	1.2 mW
Programming interface	Non-standard	SPI
Programming environment	Simulink	Arduino IDE
Calibration	Integrators only	All blocks, automatic
Computation types	CT analog only	CT analog / CT hybrid
Nonlinearities available for computation	Specific types	Arbitrary
On-chip ADC, SRAM, DAC	N/A	Available
On-chip digital controller	N/A	Available
Shut down of unused blocks	N/A	Available
$f_{max,computer}$	25 kHz[1]	20 kHz
$FOM_{computer}$	150.4 nJ[1]	14.1 nJ

[1] Estimated

Table 7.4: Comparison to previous work.

7.5 Measured mismatches of integrator time constants

As shown in the integrator design part of Chapter 4, we use fixed value capacitors to implement the integration operation. Due to the capacitor mismatches, plus the input-scaling mirror gain mismatches and the output stage gain mismatches, integrator's overall gain ($w_c = 2\pi f_c$, where f_c is the unity-gain frequency), also known as the time scaling factor, α,

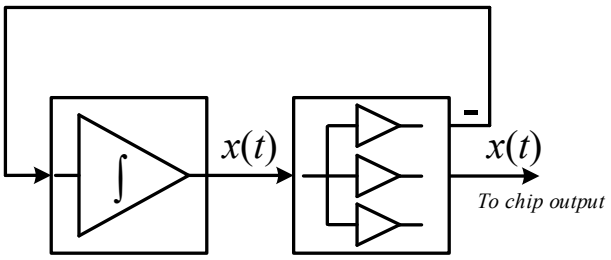

Figure 7.11: INT time constant mismatches setup.

to have mismatches as well. In order to record such mismatches and compensate them in computation, we use the following setup, which represent the first-order ODE, to measure the integrator time constant $\tau = \frac{1}{2\pi f_c}$: with $x(0)$ as the initial condition, we have the analytic solution of the above setup as $x(t) = x(0)\exp(\frac{-t}{\tau})$, where τ is the time constant of the integrator. (The solution is also in the same expression for the RC discharging circuits, where x represents the voltage across the capacitor.) The measurement results of 16 integrators on the 16th-order chip are shown in Fig. 7.12, where the initial condition is measured as $-1.22\,\mathrm{V}$ on board (which corresponds to $-1.22\,\mathrm{V}/600\,\mathrm{k\Omega} = 2.033\,\mu\mathrm{A}$). When $t = \tau$, we have $x(\tau) = x(0)\exp(-1) \approx 0.368x(0) \approx 0.449\,\mathrm{V}$. Thus, we could directly measure the time constants of 16 integrators from Fig. 7.12 by looking at the corresponding time values when outputs change by $0.449\,\mathrm{V}$. We then converted the time values into the unity-gain frequency f_c as shown in the following table:

As shown in Table 7.5, the average unity-gain frequency across one 16th-order chip is $19.37\,\mathrm{kHz}$, which is smaller than our design value $20\,\mathrm{kHz}$ because of process variations; thus, we will use $19.37\,\mathrm{kHz}$ as the new f_c. The third column shows the ratios of individual value over the average value. For those that deviate from the normal 1 "a lot" like 1.04, we could

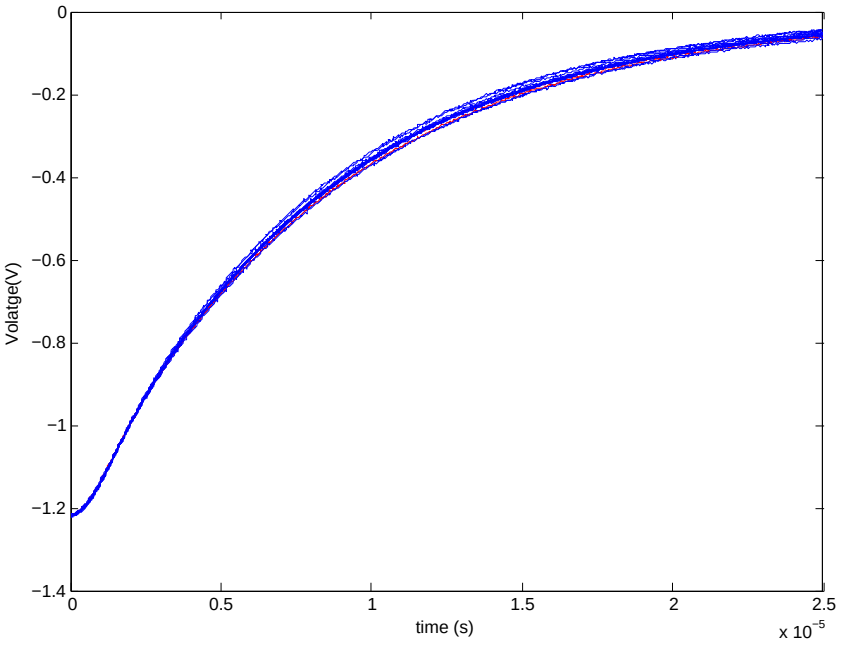

Figure 7.12: The measurement results of the setup in Fig. 7.11.

calibrate them back to one by inserting a VGA block before the Integrator block. Remember that the VGA have the ability of fine coefficient tuning by the six-bit gain calibration to the eight-bit coefficient-setting DAC. However, the disadvantage of doing this is that the inserted block would cause extra phase shifts in the feedback loops, which would also introduce solution errors. For slow varying solutions, f_c calibration would help increase the solution accuracy; for fast varying solutions, we need to be careful at introducing extra loop delays, which would possibly introduce more errors instead.

Our experience is that we could directly use those integrators with normal values between 0.99 and 1.01; this means that we could directly use 11 out of the 16 integrators shown in Table 7.5. For those 5 integrators with large f_c deviation, we could insert VGAs to do f_c calibrations.

Integrator	Unity-gain frequency (kHz)	Normalized to average value (19.37 kHz)
Tile[0].Slice[0].int	19.47	1.01
Tile[0].Slice[1].int	20.02	1.03
Tile[0].Slice[2].int	18.89	0.98
Tile[0].Slice[3].int	19.47	1.01
Tile[1].Slice[0].int	19.23	0.99
Tile[1].Slice[1].int	19.41	1.00
Tile[1].Slice[2].int	20.08	1.04
Tile[1].Slice[3].int	19.47	1.01
Tile[2].Slice[0].int	19.29	1.00
Tile[2].Slice[1].int	18.95	0.98
Tile[2].Slice[2].int	19.35	1.00
Tile[2].Slice[3].int	19.35	1.00
Tile[3].Slice[0].int	19.35	1.00
Tile[3].Slice[1].int	18.78	0.97
Tile[3].Slice[2].int	19.47	1.01
Tile[3].Slice[3].int	19.29	1.00
Average value	19.37	
Standard deviation	0.34	

Table 7.5: Integrator unity-gain frequency measurement results on the uncalibrated 16th-order chip.

7.6 Temperature tests

We provide measurement results of our hybrid computing chips for three different temperatures, -20 °C, 25 °C and 70 °C.

At all temperatures the calibrations work correctly. The offsets of analog blocks change little based on the observed calibration codes, as shown in Table 7.6. The calibration current changes in steps of 12 nA. The full-scale currents of DAC block vary more than the offsets based on the observed calibration codes. This wide variation is due to the off-chip resistors

Block type	Calibration codes*		
	-20 °C	25 °C	70 °C
Fanout	44	43	41
Multiplier	27	22	21
Integrator	44	41	39
DAC	59	33	17

*For Fanout, Multiplier and Integrator, calibration codes are for offsets; for DAC, calibration codes are for full-scale current. We use bi-polar calibration currents: code 31 is zero current, code larger than 31 is positive current and code smaller than 31 is negative current; calibration current changes in steps of 12 nA.

Table 7.6: Calibration codes of different representative blocks at different temperatures.

($\pm$ 100ppm/°C) used to set the absolute 1 µA bias current of the DAC blocks.

We also tested a 2nd-order ODE on our chip at different temperatures. The equation we used is the following:

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -0.22\frac{\mathrm{d}x}{\mathrm{d}t} - 0.84x(t) \tag{7.1}$$

with initial conditions: $x(0) = 9$ and $x'(0) = -2$. Shown in Fig. 7.13 is our chip's solutions at -20 °C, 25 °C and 70 °C, together with the ideal solution from Matlab. The RMS errors (relative to full scale) of the chip's solutions are 1.5% at -20 °C, 1.8% at 25 °C and 1.9% at 70 °C, respectively.

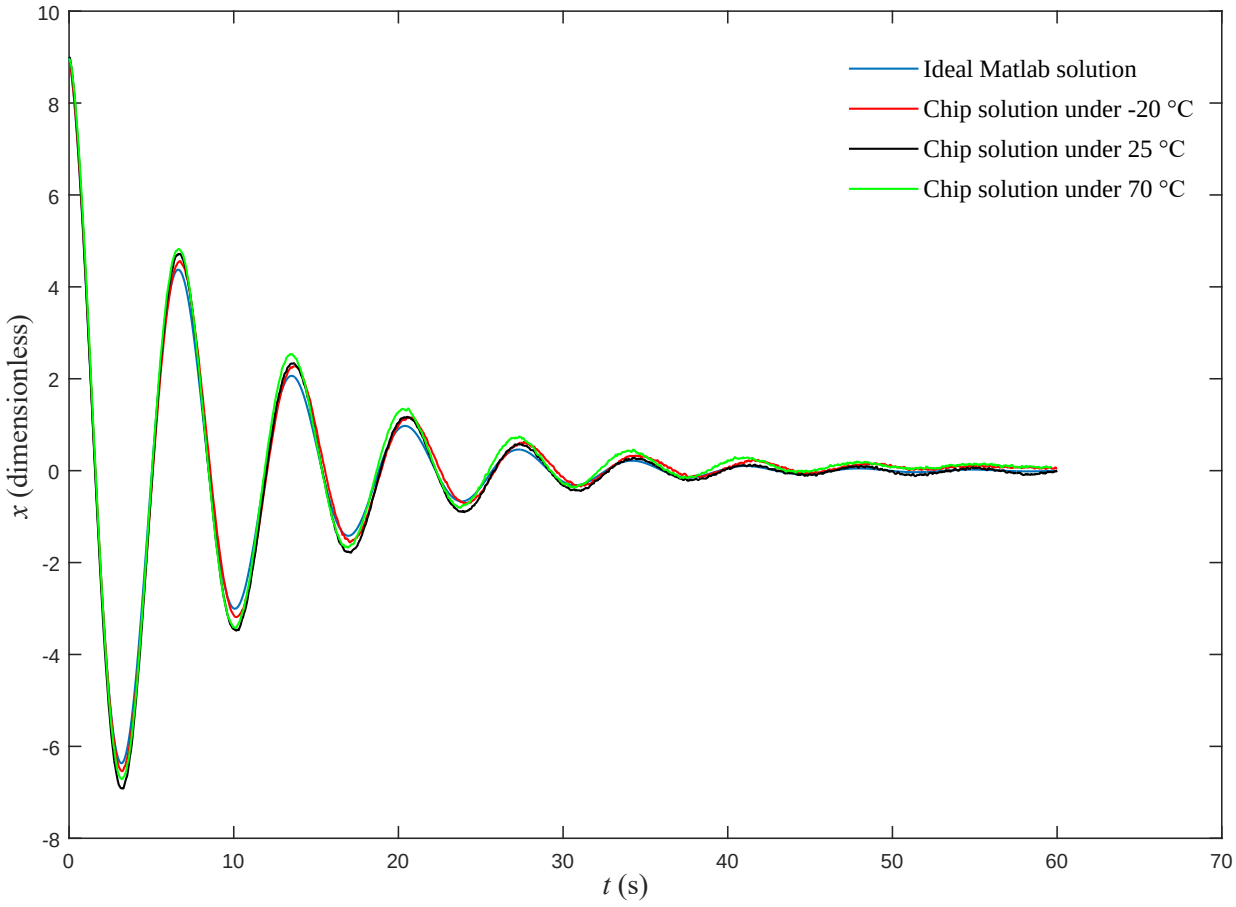

Figure 7.13: Solutions of the 2nd-oder ODE (7.1) at different temperatures.

7.7 A USB-powered hybrid computer board

In order to spread the word about analog/hybrid computing among different communities, and more importantly, to gain more users of our hybrid computing chip to try out new ideas of hybrid computing, the author developed a small, mobile demo board with the fourth-order chip, which can be powered just by a USB cable.

The hybrid computer is composed of two subboards: the hybrid computing board on

top and the Arduino Due board underneath. The purpose of the Arduino Due board is for powering, programming, measuring and calibrating the hybrid computing board. The schematic and layout of this hybrid computer board are shown in Appendix C.

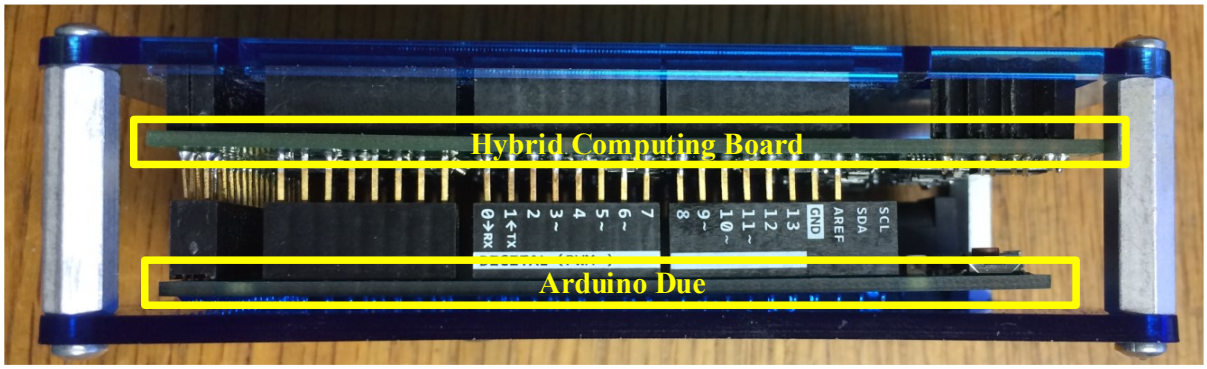

Figure 7.14: The lateral look of the hybrid computing demo board.

Another hybrid computer board with 16th-order chips are under development at the time of writing, whose schematic and layout are shown in Appendix D.

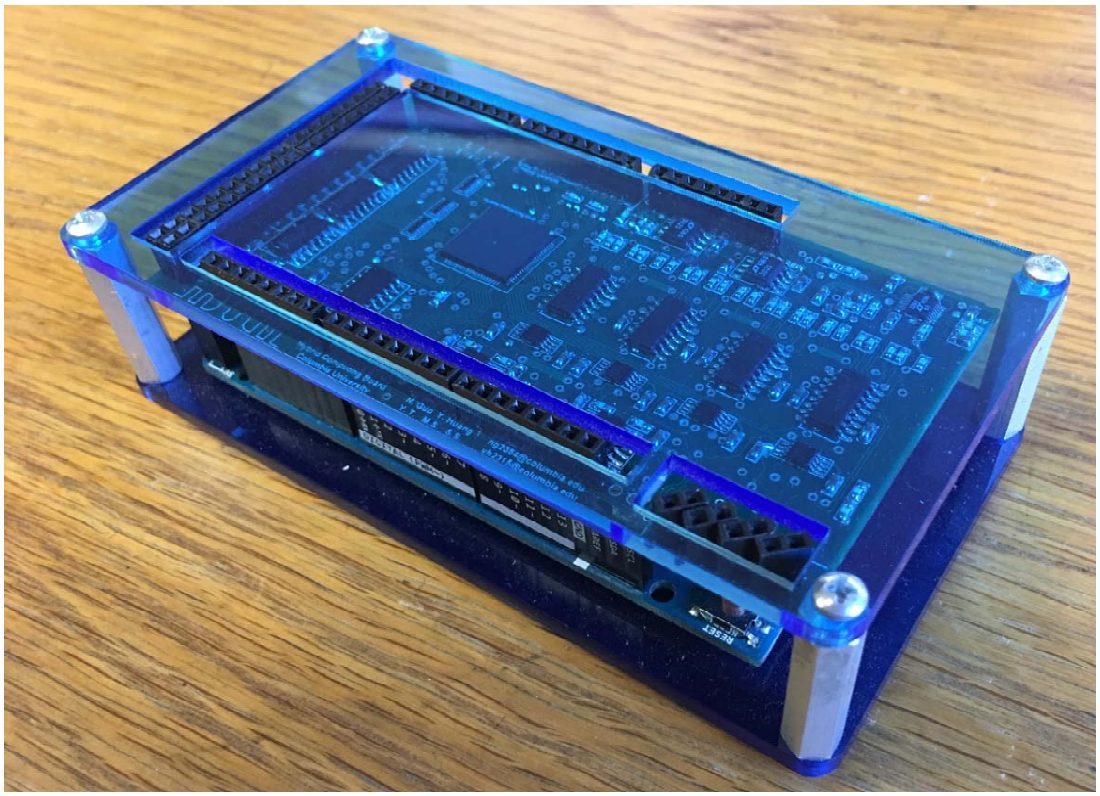

Figure 7.15: The side look of the hybrid computing demo board.

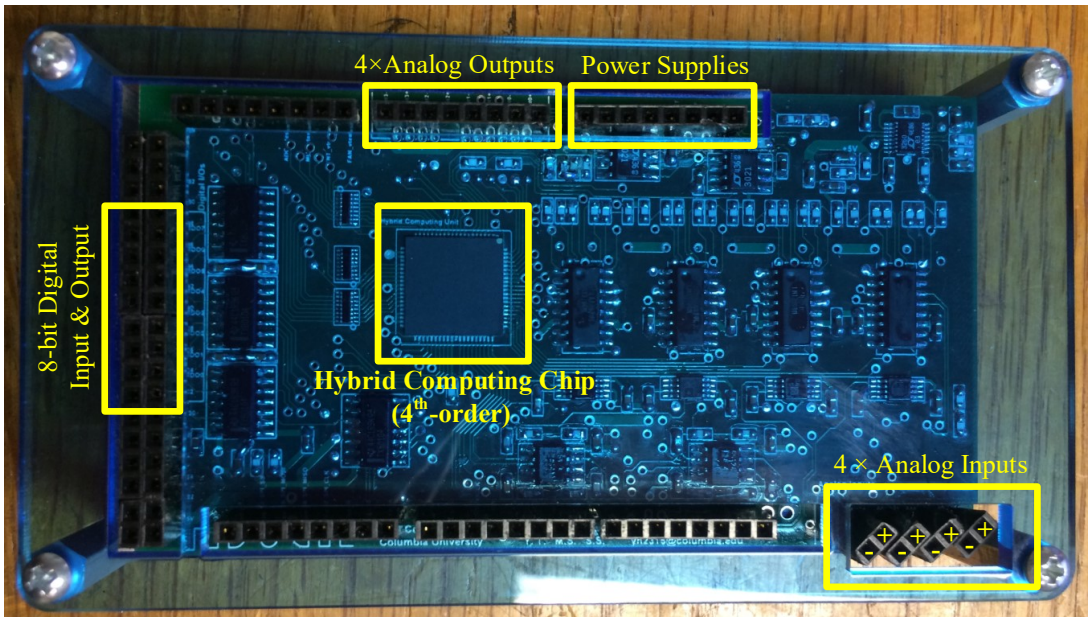

Figure 7.16: The front look of the hybrid computing demo board.

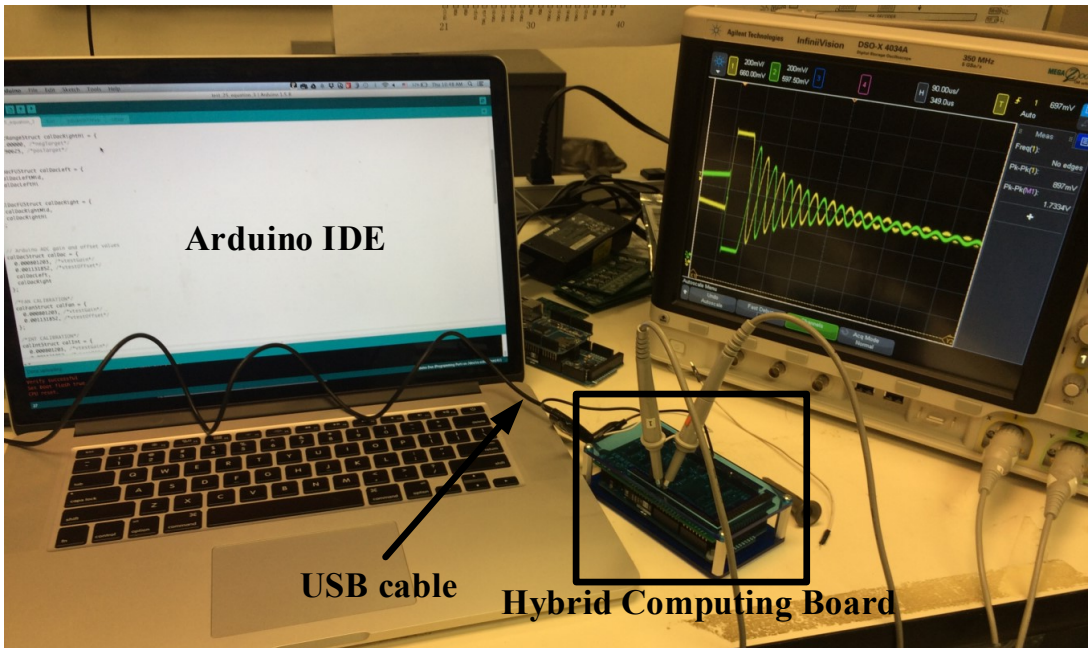

Figure 7.17: Programming environment of the demo board.

Chapter 8

Solutions of Nonlinear Differential Equations on the Hybrid Computer and Performance Comparisons

We have successfully tested the chip using a variety of equations using the developed hybrid computer board in Chapter 7. In this chapter, we focus on solving nonlinear ODEs and the comparisons against numerical methods running on microcontrollers. For examples of solving linear differential equations, see the solution of a 16th-order 1-D heat equation (16 state variables after applying finite difference method) in Appendix E, which describes the temperature distribution in 1-D space over time.

We now provide several examples of nonlinear ODEs in details.

8.1 Open-loop nonlinear equation computation for robotic path planning

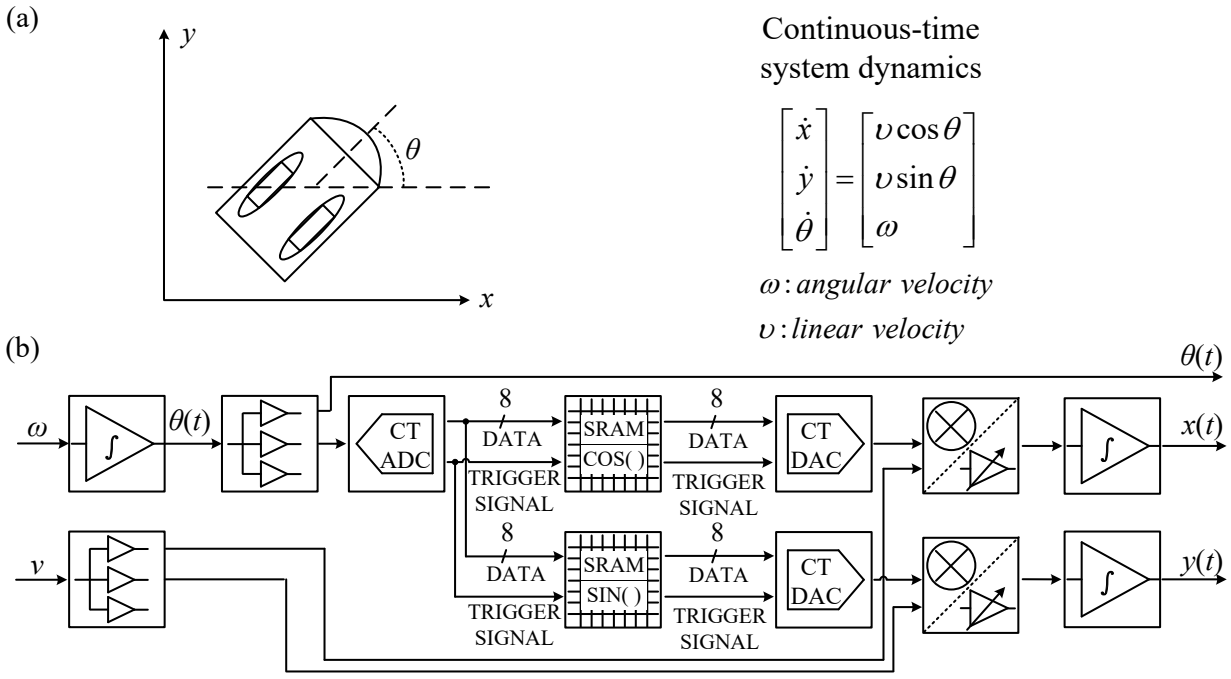

Figure 8.1: (a) Differential-drive robot system dynamics. (b) Block diagram for solving system dynamics in our hybrid computing unit.

In this equation example, we demonstrate the use of our chip modeling the system state of a miniature differential-drive wheeled robot (Fig. 8.1(a)) using model-predictive control method [22]. Under a limited computing energy budget, the robot must predict the system state ($x(t)$, $y(t)$, $\theta(t)$) at a future instant. The prediction involves trying as many randomized inputs ($\omega(t)$, $\nu(t)$) as possible, and finding the best input that minimizes a cost function for actuator control. Our chip acts as a system dynamics simulator in this application. The

	Time step size	Total clock cycles	Time needed for one solution	Energy consumption for one solution
Our hybrid chip	N/A	N/A	0.84 µs	0.48 nJ
MSP430[1]	0.1s	734	29 µs	5.14 nJ

[1] 25 MHz, 7 µW/MHz
N/A: Not Applicable

Table 8.1: Comparison to solutions on a MSP430 microcontroller for robotics applications.

system dynamics equations are mapped to the diagram shown in Fig. 8.1(b), and the chip is programmed to implement this diagram. In this example, we apply constant inputs (ω, v) and solve for the state 0.1 s into the future. Our chip solves this in 0.84 µs with 0.48 nJ energy consumption and with 0.6% RMS error relative to full scale under 5000 random tests, which is acceptable for this application.

For a fair comparison, we chose an efficient 16-bit fixed-point solver (eight-bit fixed-point resulted in excessive error), running on a state-of-the-art, 25 MHz 0.4 V, MSP430 architecture [23], which has the efficiency of 7 µW/MHz, is assumed, it requires an estimated 29 µs and 5.14 nJ to predict the future state, with 0.28% RMS error relative to full scale. Our chip achieves 35× improvement in speed and 11× in energy.

8.2 Nonlinear differential equations modeling a coupled mass-spring system

The example describes a coupled mass-spring system involving nonlinear springs, shown in Fig. 8.2.

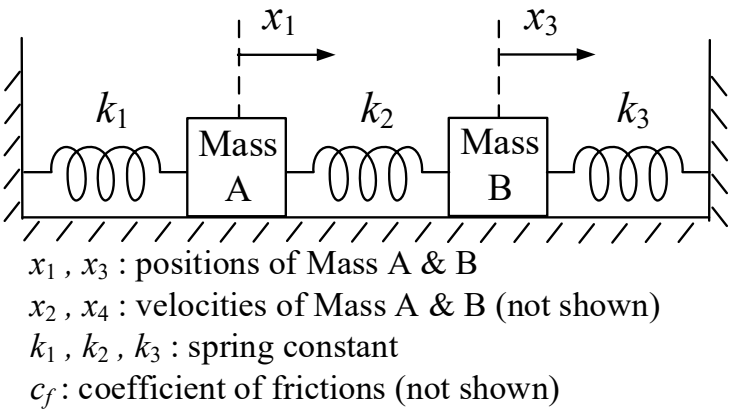

Figure 8.2: A 1-D coupled mass-spring system with nonlinear springs and Coulomb friction.

The nonlinear ODEs describing the system dynamics are shown as follows:

$$\begin{aligned}
\dot{x_1} &= x_2, \\
\dot{x_2} &= -(k_1 + k_2)\text{sign}(x_1)\sqrt{|x_1|} + k_2\text{sign}(x_3)\sqrt{|x_3|} - c_f\dot{x_1}, \\
\dot{x_3} &= x_4, \\
\dot{x_4} &= -(k_2 + k_3)\text{sign}(x_3)\sqrt{|x_3|} + k_2\text{sign}(x_1)\sqrt{|x_1|} - c_f\dot{x_3}
\end{aligned} \tag{8.1}$$

with initial conditions of $x_1(0) = 2; x_2(0) = 0; x_3(0) = -1; x_4(0) = 0$, spring constants of $k_1 = k_2 = k_3 = 0.5$ and friction coefficient of $c_f = 0.15$. We simulate the motion of the two masses, with displacements x_1 and x_3, for a physical time of 40 s. Fig. 8.3 shows

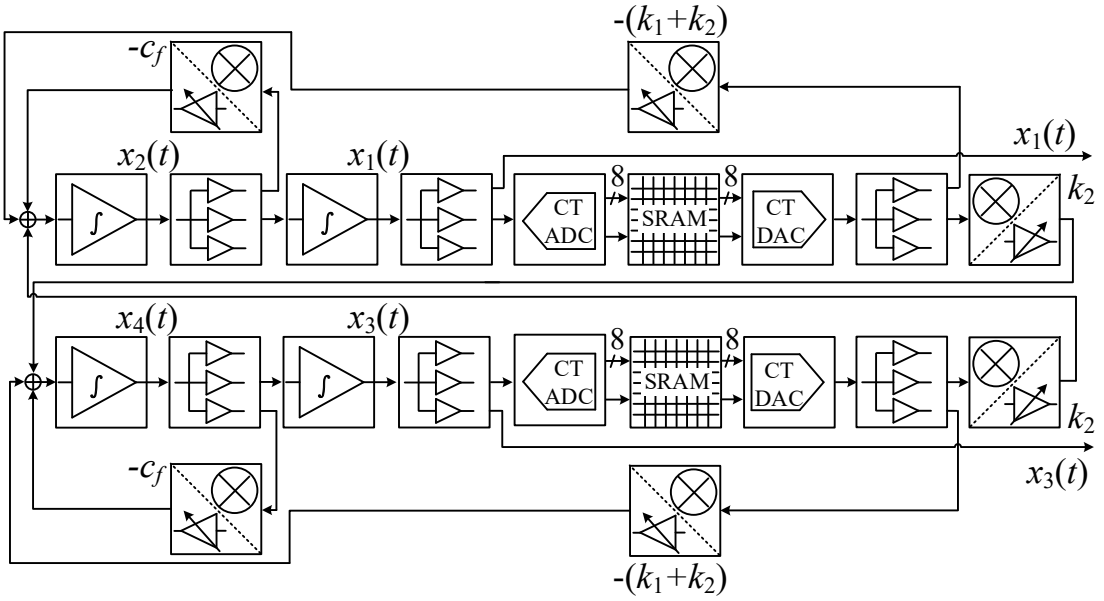

Figure 8.3: The block diagram solving the nonlinear differential equation (8.1).

the block diagram that maps the equations, where the nonlinear functions $\text{sign}(x_i)\sqrt{|x_i|}$ are implemented as lookup tables. When solving the equations shown in Fig. 8.3, the state variables $x_1(t)$ and $x_3(t)$ are continuously varying with time; they are converted by the CT-ADCs into CT digital signals, which are immediately fed into the following SRAMs in order to look up the nonlinear function values. The following CT-DACs convert the SRAMs' outputs back to analog signals, which are distributed to several destinations through fanout blocks. The signal flow in this computing technique is CT hybrid: it is CT digital inside the ADC+SRAM+DAC chain, and CT analog elsewhere.

Our hybrid chip solves the nonlinear ODEs in 320 µs with energy consumption of 0.25 µJ, and with 4.7% RMS error relative to full scale. Fig. 8.4 shows the representative solution of $x_1(t)$ from our chip (dots), together with the ideal solution (solid lines).

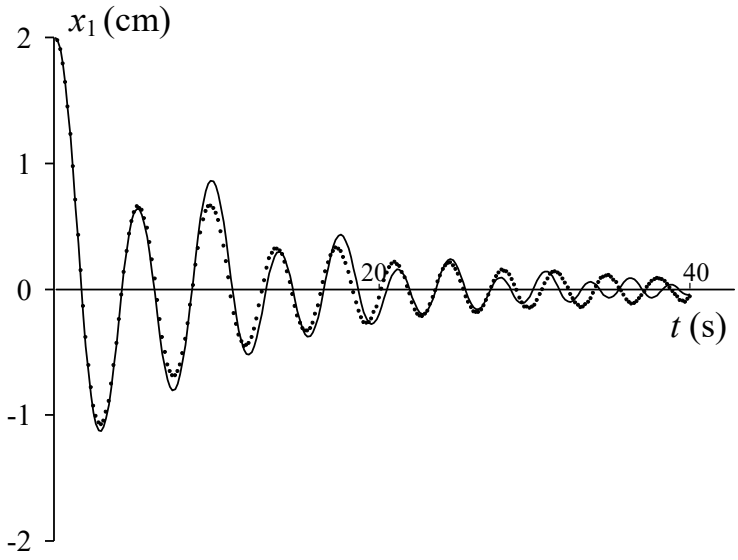

Figure 8.4: The solution of $x_1(t)$ from our hybrid computer (dots) and the ideal solution (solid line).

A comparison against a RK4 solver running on the state-of-art MSP430 microcontroller [23] is shown in the following table.

	Coupled mass-springs, 4.7% RMS error	
	Our chip	MSP430[1], RK4 method
Time step size (s)	**N/A**	0.85
No. of iterations	**N/A**	47
Clock cycles per iteration (est.)	**N/A**	21.6k
Total clock cycles (est.)	**N/A**	1015k
Solution time (μs)	**320**	41k
Solution energy (μJ)	**0.25**	7
FOM_{task} (μs*μJ) (Time-energy product)	**80**	287k

[1] 25 MHz, 7 μW/MHz
N/A: Not Applicable

Table 8.2: Comparison to the solution obtained with RK4 method on a MSP430 microcontroller.

8.3 Van der Pol equation

In this example, we provide the solution of the second-order, nonlinear Van der Pol equation [24]. The equation is shown as follows:

$$\begin{aligned} \dot{x_1} &= x_2, \\ \dot{x_2} &= \mu(1 - x_1^2)x_2 - x_1 \end{aligned} \tag{8.2}$$

with initials conditions of $x_1(0) = -0.5, x_2(0) = 0$. We choose $\mu = 0.2$ for this example. Fig. 8.5 shows the block diagram that maps the equations. We solve for a physical time of

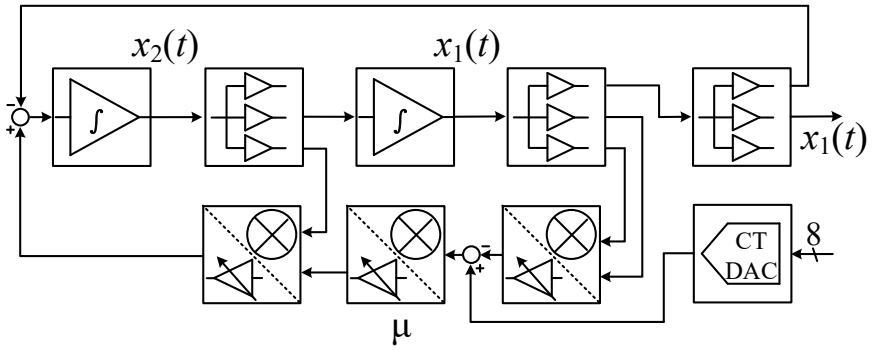

Figure 8.5: Van der Pol equations in (8.2) mapped to our chip.

60 s. The solution of $x_1(t)$ from our chip (dots) is shown in Fig. 8.6, together with the ideal solution (solid lines). Our hybrid chip solves this problem in 480 µs with energy consumption of 0.14 µJ, and with 4.6% RMS error relative to full scale.

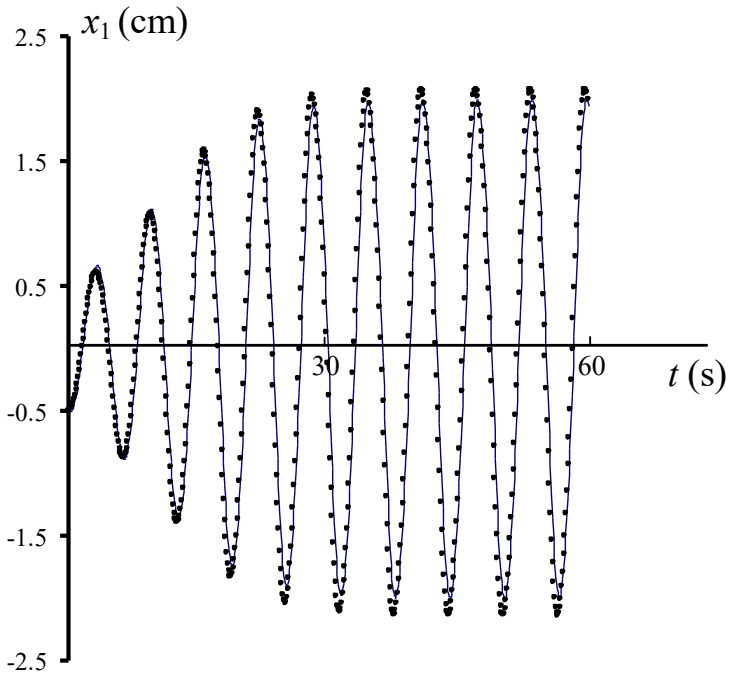

Figure 8.6: The solution of $x_1(t)$ from our hybrid computer (dots) and the ideal solution (solid line) for the Van der Pol equation (8.2).

A comparison against a RK4 solver running on the state-of-art MSP430 microcontroller [23] is shown in Table 8.3.

	Van der Pol, 4.6% RMS error	
	Our chip	MSP430[1], RK4 method
Time step size (s)	**N/A**	0.23
No. of iterations	**N/A**	260
Clock cycles per iteration (est.)	**N/A**	7.15k
Total clock cycles (est.)	**N/A**	1859k
Solution time (μs)	**480**	74k
Solution energy (μJ)	**0.14**	13
FOM_{task} (μs*μJ) (Time-energy product)	**67**	962k

[1] 25 MHz, 7 μW/MHz
N/A: Not Applicable

Table 8.3: Comparison to the solution obtained with RK4 method on a MSP430 microcontroller.

Phase plane plots are often used to investigate nonlinear oscillation equations. Fig. 8.7 shows the phase plane plot of two state variables, $x_1(t)$ and $x_2(t)$ for both our hybrid chip's solution (red) and the ideal solution (blue). Seven time stamps, starting from 0 s with increment of 10 s, are marked on plots for reference.

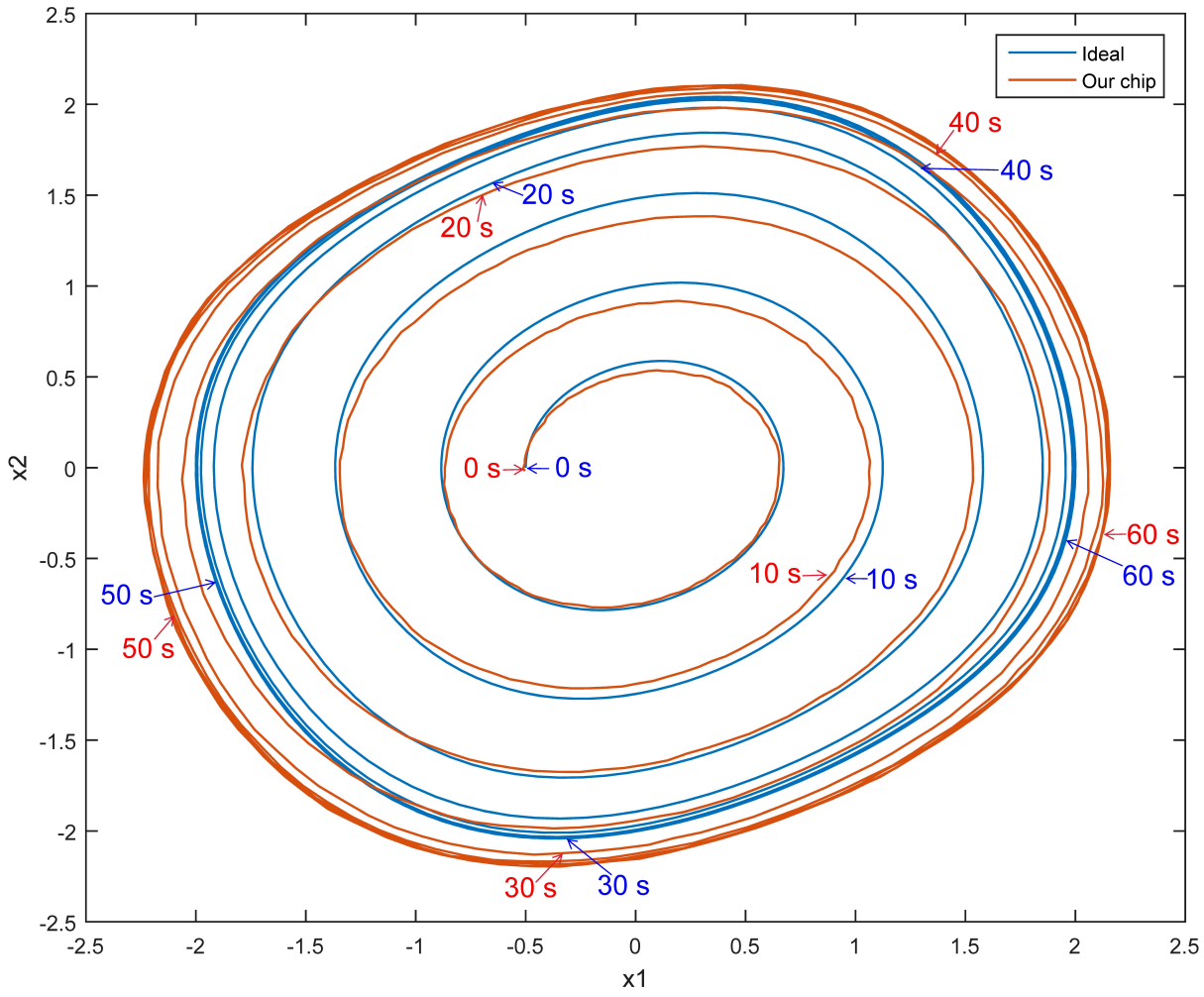

Figure 8.7: The phase plane plots from our hybrid computer (red) and the ideal solution (blue) for the Van der Pol equation (8.2). Time stamps with increment of 10 s are marked.

Chapter 9

Conclusions and Suggestions for Future Work

9.1 Conclusion on the presented hybrid-computing chip

We introduced CT hybrid approximate computation, and have implemented a prototype system in a scalable architecture using 65 nm CMOS technology. The system can do CT computation with arbitrary nonlinearities that are implemented by a CT ADC + SRAM + DAC architecture, demonstrating for the first time the use of CT digital signals in hybrid computation. CT digital signals are used to do table-lookup tasks in our case. Nevertheless, they can also be considered for such computation tasks as digital integration and differentiation in the CT domain [25]. With CT digital signals involved, hybrid computing attains more versatility, while ensuring aliasing-free operation and adaptive power dissipation.

We have successfully demonstrated the solution of differential equations up to 16th order. Extensive digitally assisted calibration is used to improve analog computation accuracy, which is on the order of 0.5% to 5%, depending on the details of the equations. However, the trade-offs involved in our approach are very different from those in digital computation. In the latter, precision can be increased at will by adding bits, whereas such luxury does not exist in our case. For solving ODEs, digital computation can also use more energy (through smaller time steps) to achieve higher accuracy. On the other hand, in many applications, such as in cyber–physical systems, the overall system accuracy is limited by the sensors and actuators involved, so extra bits in computation would not bring a significant advantage. Another difference involves chip area. In our case, it scales approximately in proportion to the problem order, whereas in digital computation a higher-order problem just results in longer computation times, leaving area unaffected.

We should note that analog and hybrid computers have "taken a 40-year break"; we are just now researching their possibilities in modern VLSI technology. It would thus be premature to draw conclusions on comparisons between analog and digital computers, the latter of which have a huge R&D effort behind them. Nevertheless, a limited comparison for the specific cases reported shows that, compared to a conventional microcontroller in the same technology, our hybrid computing unit is capable of giving much faster solutions (by about two orders of magnitude) with large energy savings (one to two orders of magnitude), for the same error. Thus, one possible use of the techniques presented is in applications where approximate solutions are sought with low computational energy, as is often the case with

cyber–physical systems.

9.2 Suggestions for future work

First, more area-efficient calibration methods are needed. In our work, six-bit current-steering calibration DACs are used extensively in all analog and mixed blocks. This calibration DAC occupies a large percentage of area, as shown in Table 5.7. One possibility woul the floating-gate technique. The floating-gate transistor can hold the calibration information in gate-channel voltage, which is then converted to calibration current by transconductance circuits. The transconductance circuit could be implemented as simply as one transistor. Thus, the original six-bit calibration DAC could be replaced by several transistors using floating-gate techniques, which would greatly decrease the area needed for calibration.

Second, digital computing blocks with more functionality are needed. For example, if a microprocessor core is integrated on chip, calibration and configuration time would be greatly reduced. Our prototype took about 4 ms to calibrate the fourth-order chip with the SPI clock running at 20 MHz. With the calibration algorithms on chip, the SPI clock would easily run at a much higher rate, such as 500 MHz in the 65 nm technology we used. The calibration time could be reduced to 0.16 ms, greatly reducing the time overhead.

Third, besides the CT lookup tables, CT digital blocks doing arithmetic computations are worth trying on chip. For example, as described in the patent [25], continuous-time digital integration and differentiation blocks could be implemented with feedback and feed-forward architectures. If, in the end, we could replace all analog computing blocks with the

equivalent CT digital ones, we will no longer need offsets/gain calibrations, greatly improving the solutions' accuracy.

Fourth, more time and effort could be spent on exploring the analog-seeding method for solving equations. Most of the work described in this thesis was spent on designing chips and boards. The analog-seeding method is a win-win: it takes advantage of the fast and approximate analog solutions, and uses it as a seed to speed up an accurate but slow digital algorithm. One analog seeding example can be found in Cowan's work [1].

9.3 A few thoughts on analog and hybrid computation and its applications

In history, no conclusions were drawn on why analog computers became extinct. In the 1960s, there were conferences for experts from analog computing and digital computing communities to debate and defend their respective machines. Because analog computers and digital computers use CT and DT computing principles, their strengths and weakness are complementary. However, in the era dominated by CMOS technology, which is highly optimized for logic operations, digital computers have been the winners.

The equations solved by old analog computers and the CT hybrid computer presented in this work are mostly used to model physical systems. This means that the role of analog and CT hybrid computers is to provide guidance to how the physical system, modeled by differential equations, would behave under different conditions and parameters. Thus, analog

computers and our CT hybrid computer serve as physics simulators and predict behaviors in the time domain.

We spent quite some time trying to find applications for the energy-efficient CT hybrid computer presented in this work. They would include applications that

1. need to solve ODEs.
2. solve ODEs which dominate the computing workload.
3. solve ODEs that are mappable to our hybrid computer chip.
4. require low energy consumption while solving ODEs (more important than speed and accuracy).

The first area we examined is real-time robotics. However, most of the robotics workload is evaluation of real-time data from sensors: Robotics decisions are made based on conventional algorithms like searching and sorting. One path-planning method called model-predictive control has been found that evaluates the robot's future states and solves ordinary differential equations describing the system states many times. However, evaluating ODEs has not been identified as a problem in academic papers and people simply used Euler methods. The subsequent time-consuming nonlinear programming algorithms, which process the ODE results and allocate robotics resources in real time, are big concerns and speedup methods are often discussed in papers.

The second area is video games. Some video games do involve real-time physics simulations, where the objects are changing positions according to users' inputs. Physics engines,

software that provides simulations of physical systems, are often used to compute objects' motions. It remains unclear how the complicated physics engine software works; it could be based on open loop calculation or solving ODEs. But among these motion simulations, the common collision detections are not suitable on analog computers because at the very moment of collision, large force values, beyond analog computers' dynamic range, are needed at the contacted surface to separate two objects. Nevertheless, calculating objects' motions only accounts for a small portion of the computing work load (both time and energy). Other computations like coordinate transformations, rendering, and texture mappings, takes up most of the computing workloads; GPUs are often used there.

The third is solving ODEs and PDEs in applied mathematics. When we talked to people from this area, they expressed more concern about speed, complexity, and accuracy than about power dissipation. People use as much computing power as possible—e.g., GPU arrays and computer clusters—to solve very complex and difficult problems. For example, the Vlasov PDEs describe the evolution of a plasma distribution function in time domain. Unfortunately, it is not practical to map the Vlasov PDE to our hybrid computer chip because the dependent variable is a function of time, space, and velocity. For space dimensions, we could use the finite difference method to approximate space derivatives. Then, at each node, in theory, we could discretize again in the velocity dimension, which, unlike space, has no boundaries, giving it a huge dynamic range (velocity could change sharply, as in the case of plasma-particle motion). Thus, the discretization in the velocity dimension would result in huge numbers of state variables, not suitable on analog computers. For conventional equations

like Poisson PDEs, a moderate-size problem to start catching some attention would involve millions of state variables, such as a 3-D Poisson PDE with 100 nodes in each dimension. Speeding up PDEs with millions of variables would be of interest, as this many variables would reasonably model a physical object or phenomenon. However, even with several 16th-order chips integrated on board, the number of state variables our chip could handle would be around one hundred, which could only model toy problems from the perspective of applied mathematics.

Thus, as can be seen, though solving differential equations with as little energy as possible is a good and correct direction to explore, its direct applications remain a mystery, where solving differential equations are involved and its solution energy is the biggest concern. However, we believe that with the coming of the Internet of Things and other emerging applications of sensor nodes, where more versatile distributed computing methods are needed and their energy efficiency is a concern, it is realistically possible that solving differential equations will be embedded in future applications.

Bibliography

[1] G. E. R. Cowan, R. C. Melville, and Y. P. Tsividis, "A VLSI analog computer/digital computer accelerator," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 41, pp. 42–53, Jan. 2006.

[2] C. E. Shannon, "Mathematical theory of the differential analyzer," *Journal of Mathematics and Physics*, vol. 20, no. 1–4, pp. 337–354, 1941.

[3] A. S. Jackson, *Analog Computation*. New York, NY, USA: McGraw-Hill, 1960.

[4] A. E. Rogers and T. W. Connolly, *Analog Computation in Engineering Design*. New York, NY, USA: McGraw-Hill, 1960.

[5] J. A. Lawrence and H. E. Smith, "The role of jsc engineering simulation in the apollo program," *Journal of Simulation*, vol. 57, no. 1, pp. 9–16, 1991.

[6] G. A. Korn and T. M. Korn, *Electronic Analog and Hybrid Computers*. New York, NY, USA: McGraw-Hill, 1964.

[7] G. E. R. Cowan, R. C. Melville, and Y. P. Tsividis, "A VLSI analog computer/math co-processor for a digital computer," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 1, pp. 82–586, Feb. 2005.

[8] G. Bekey and W. Karplus, *Hybrid computation*. Wiley, 1968.

[9] R. Benham, *Evaluation of Hybrid Computer Performance on a Cross Section of Scientific Problems*. AEC research and development report, Pacific Northwest Laboratory, 1970.

[10] A. E. Rogers and T. W. Connolly, *Analog Computation in Engineering Design*. New York, NY, USA: McGraw-Hill, 1960.

[11] E. A. Vittoz and Y. P. Tsividis, "Frequency-dynamic range-power," *Trade-Offs in Analog Circuit Design: The Designer's Companion*, C. Toumazou, G. Moschytz, B. Gilbert, Boston, MA, USA: Springer, 2002, pp. 283-313.

[12] Yipeng Huang, private communication, 2016.

[13] B. Schell and Y. Tsividis, "A clockless ADC/DSP/DAC system with activity-dependent power dissipation and no aliasing," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pp. 550–635, Feb. 2008.

[14] S. Kawahito and Y. Tadokoro, "CMOS Class-AB current mirrors for precision current-mode analog-signal-processing elements," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, pp. 843–845, Dec. 1996.

[15] R. Stata, "Operational integrators," *Journal of Analog Dialogue*, vol. 1, no. 3, pp. 6–11, 1967.

[16] Tao Mai, private communication, 2013.

[17] K. Bult and G. Geelen, "The CMOS gain-boosting technique," *Journal of Analog Integrated Circuits and Signal Processing*, vol. 1, no. 2, pp. 119–135, Oct. 1991.

[18] A. Putra, T. H. Teo, and S. Rajinder, "Ultra low-power low-voltage integrated preamplifier using class-AB op-amp for biomedical sensor application," in *Proc. Int. Symp. on Integrated Circuits (ISIC)*, pp. 216–219, Sept. 2007.

[19] Y. Tsividis, "Continuous-time digital signal processing," *Electronics Letters*, vol. 39, pp. 1551–1552, Oct. 2003.

[20] T. Huang and C. Zukowski, "Reconfigurable digital/analog processor array for the simulation of gene regulatory networks," in *49th IEEE International Midwest Symposium on Circuits and Systems*, vol. 1, pp. 552–556, Aug. 2006.

[21] D. Kim, G. Chen, M. Fojtik, M. Seok, D. Blaauw, and D. Sylvester, "A 1.85 fW/bit ultra low leakage 10T SRAM with speed compensation scheme," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 69–72, May 2011.

[22] G. Klančar and I. Škrjanc, "Tracking-error model-based predictive control for mobile robots in real time," *Robotics and Autonomous Systems*, vol. 55, no. 6, pp. 460–469, 2007.

[23] D. Bol, J. D. Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J. D. Legat, "Sleepwalker: A 25 MHz 0.4 V sub-mm^2 7 µW/MHz microcontroller in 65 nm LP/GP CMOS for low-carbon wireless sensor nodes," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 48, pp. 20–32, Jan. 2013.

[24] D. Kaplan and L. Glass, "Understanding nonlinear dynamics," in *Two-Dimensional Differential Equations*, New York, NY, USA: Springer, 1995, pp. 240–244.

[25] Y. Tsividis, "Systems, apparatus, and methods for providing continuous-time signal differentiation and integration." US Patent App. 14/082,945, May 2014.

[26] R. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007.

Appendix A

Implementing Division

Though division is not available as a computing block, our computer is rich in primary computing units: We could construct a divider block as discussed in the following subsections.

A.1 Using an integrator and a multiplier

In this method, we use an integrator block with a multiplier block in feedback to build a divider, as shown in Fig. A.1. The purpose of the fanout block here is to duplicate and invert the current signals. Please note that the multiplier block on our chip has the transfer function of $z = 0.5 \times x \times y$, where z is the output, and x and y are the inputs. We designed the multiplier with the 0.5 coefficient because when both x and y are at full scale values (e.g., 2 µA), we need to ensure that the output current is also unsaturated because it could be fed to other blocks' input. The 0.5 coefficient would halve the ideal 4 µA current to the allowed maximum value of 2 µA. See multiplier circuit design in Chapter 4 for more details.

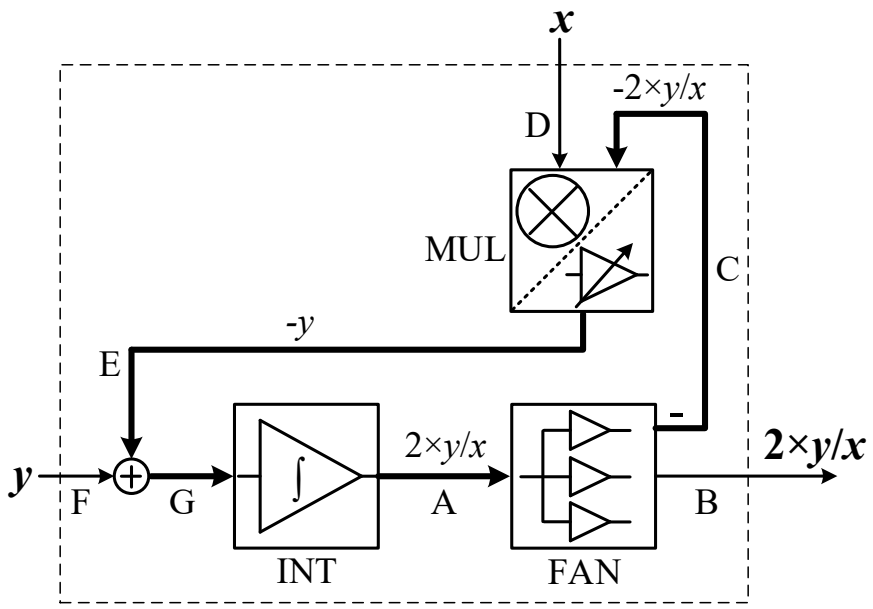

Figure A.1: Divider built with an integrator with a multiplier in the feedback path. FAN is used to duplicate signals. This is an implicit method for doing division, as we get the results from an internal node of the diagram.

We now describe the operating principle of this divider. The critical signal path is in bold, forming the mandatory negative feedback loop for the divider to work correctly. Fig. A.1 shows the case when the input x is positive. (When x is negative, we need to set the first output of the fanout block back to normal, with no inversion.) When the system is in the stable state, the input (Node G) to the Integrator block is zero. Otherwise, any nonzero signals would be integrated and drive the signal at Node G to 0 through the feedback loop. Then we trace the signal path clockwise from Node G to derive other signal values. The signal on Node E must be $-y$ to cancel the input y on Node F. The signal on Node C should be $-2 \times y/x$ for the multiplier block to generate $-y$. Since the first output of the fanout

inverts the signal on Node A, both Node A and B should be $2 \times y/x$, which is the output of the divider. The integrator here serves as an op amp, which drives any "error" signal at the summing nodes to zero, thus stabilizing the loop.

To ensure that all the signals stay in the range during the division operation, we need to limit our inputs x and y to satisfy the requirements of all nodes. This requires

$$-2\,\mu\text{A} < x < +2\,\mu\text{A},\ x \neq 0,\ -x < y < +x \tag{A.1}$$

If x and y do not satisfy the above, they can be scaled within the required range before division.

A.2 Using table lookup

Since division of one variable by another is a nonlinear math operation, we could use our lookup tables to generate the function, as shown in Fig. A.2.

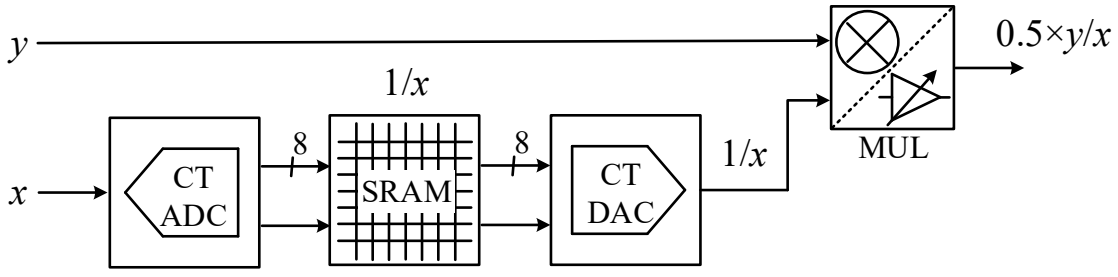

Figure A.2: Divider built with the nonlinear function generation.

Here, we use the lookup table to generate the nonlinear function $\frac{1}{x}$, and then multiply it

with y to get the final $\frac{y}{x}$ with a coefficient of 0.5. This requires

$$-2\,\mu\text{A} < x < -0.5\,\mu\text{A},\ 0.5\,\mu\text{A} < x < +2\,\mu\text{A},\ -2\,\mu\text{A} < y < +2\,\mu\text{A} \tag{A.2}$$

Again, x and y may have to be scaled to satisfy the above relation.

Appendix B

Automatic Scaling

With the help of the microcontroller and the internal saturation-detection circuits, we have the ability to do automatic scaling during computation. The microcontroller could monitor the saturation-detection bits through the SPI interface by requesting the exception bits from our chip. Once saturation is detected, the microcontroller could pause the computation, redo the dynamic range scaling, download the new equation parameters onto the chip and resume the computation. We use the following equation as a demonstration of how it works.

$$\ddot{x} = -0.2\dot{x} - 2x - 1.6, \tag{B.1}$$

with initial conditions of $x(0) = 1.5$, $\dot{x}(0) = -1.5$. If we directly map this equation and values on our chip, we would have $x(t)$'s minimum value of $-2.8\,\mu A$ and $\dot{x}(t)$'s minimum value of $-3.2\,\mu A$ and maximum value of $2.6\,\mu A$, exceeding our desired range from $-2\,\mu A$ to $+2\,\mu A$. One straightforward way to solve the above saturation issue is to do the following

simple math trick to the original (B.1) by "halving" all the terms and values:

$$\left(\frac{\ddot{x}}{2}\right) = -0.2\left(\frac{\dot{x}}{2}\right) - 2\left(\frac{x}{2}\right) - \frac{1.6}{2}. \tag{B.2}$$

We could replace $\frac{x}{2}$ with a new variable y in the above equation and have

$$\ddot{y} = -0.2\dot{y} - 2y - 0.8, \tag{B.3}$$

with initial conditions of $y(0) = 0.75$, $\dot{y}(0) = -0.75$.

Appendix C

Schematic and Layout of the Demoboard with Fourth-Order Chip

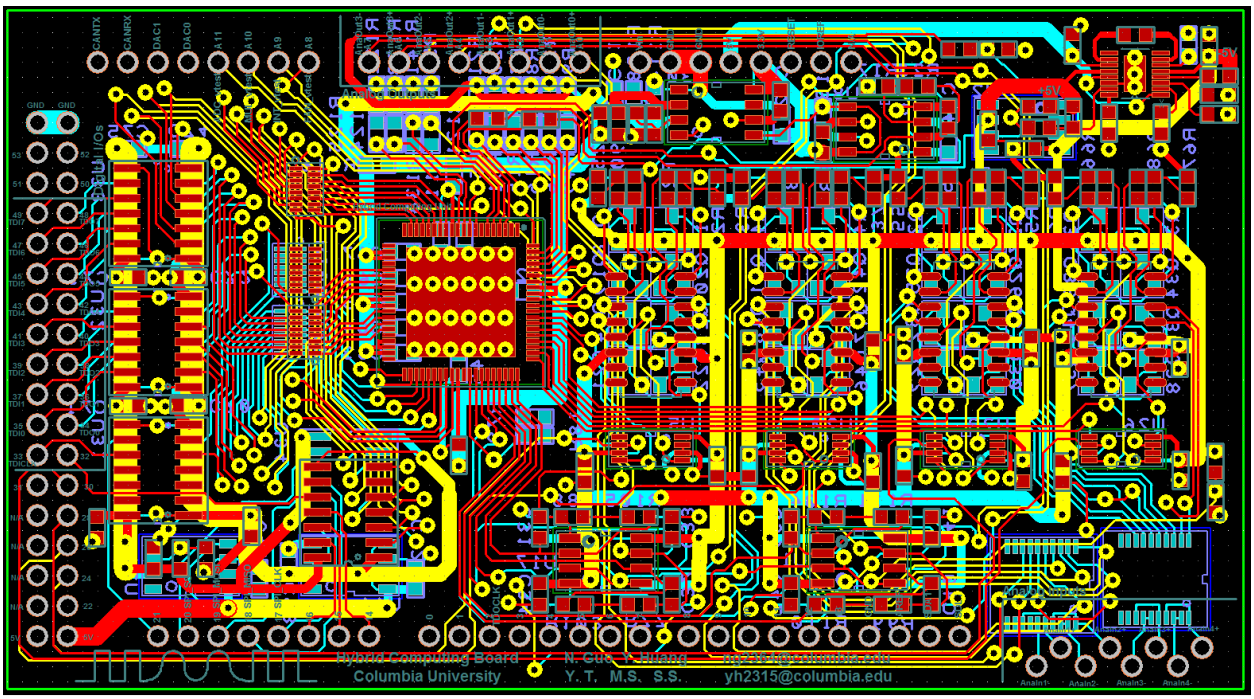

Figure C.1: The layout of the demoboard with the fourth-order chip.

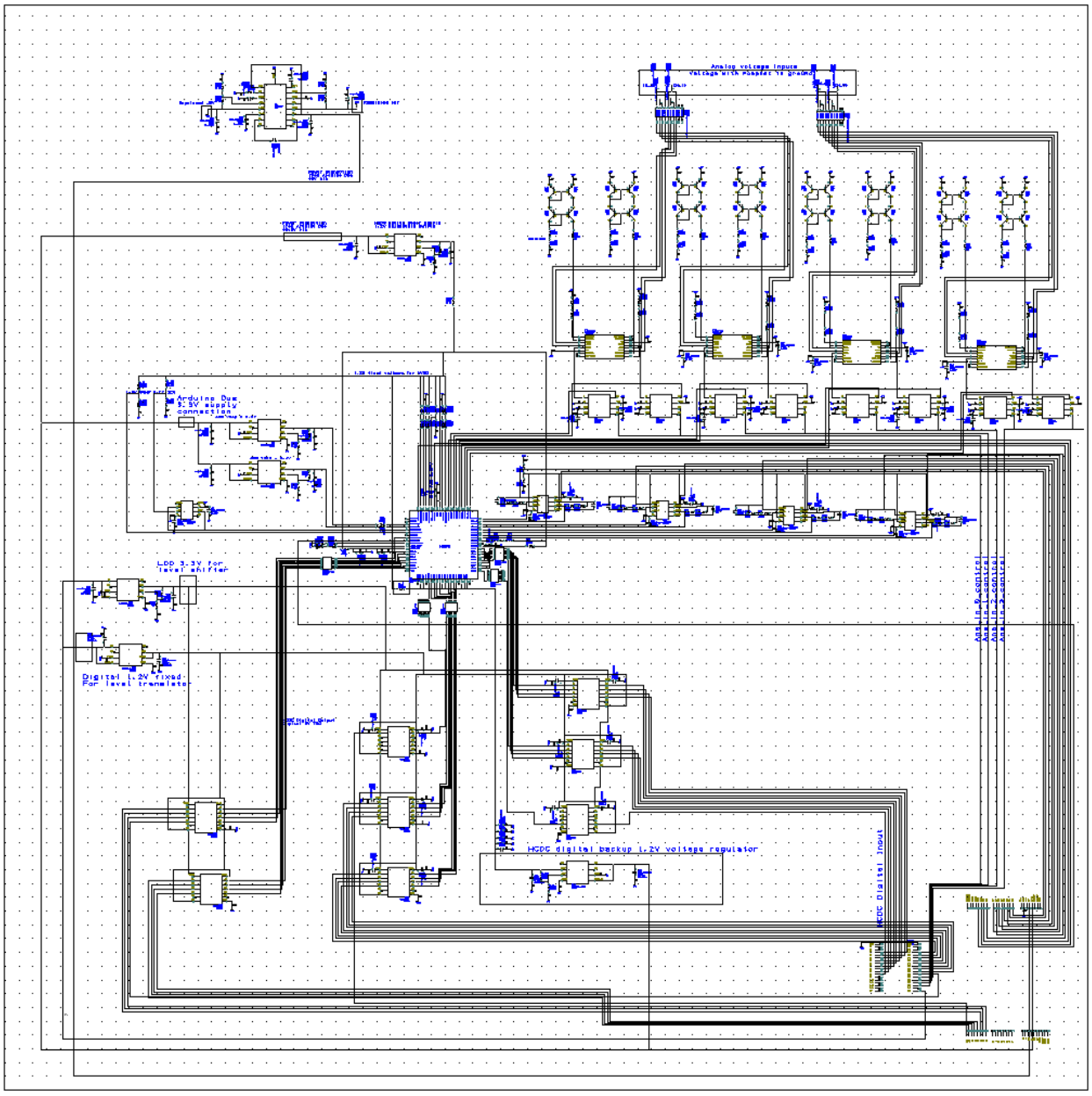

Figure C.2: The schematic of the demoboard with the fourth-order chip.

Appendix D

Schematic and Layout of the Demoboard with 16th-Order Chips

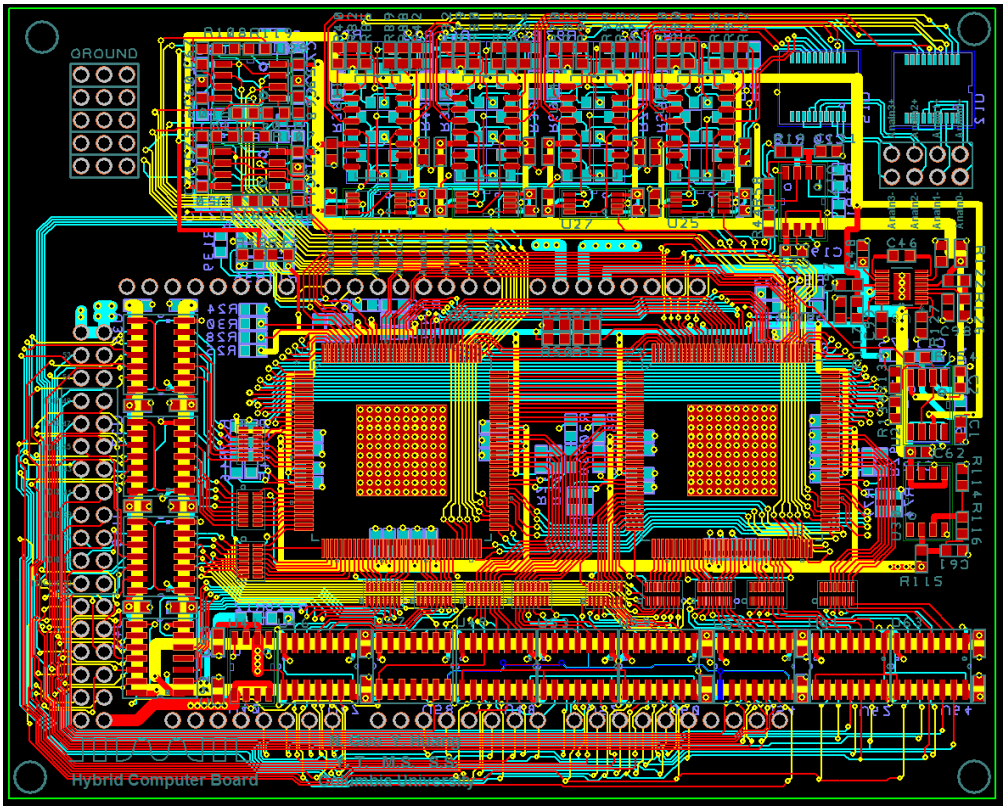

Figure D.1: The layout of the demoboard with the 16th-order chip.

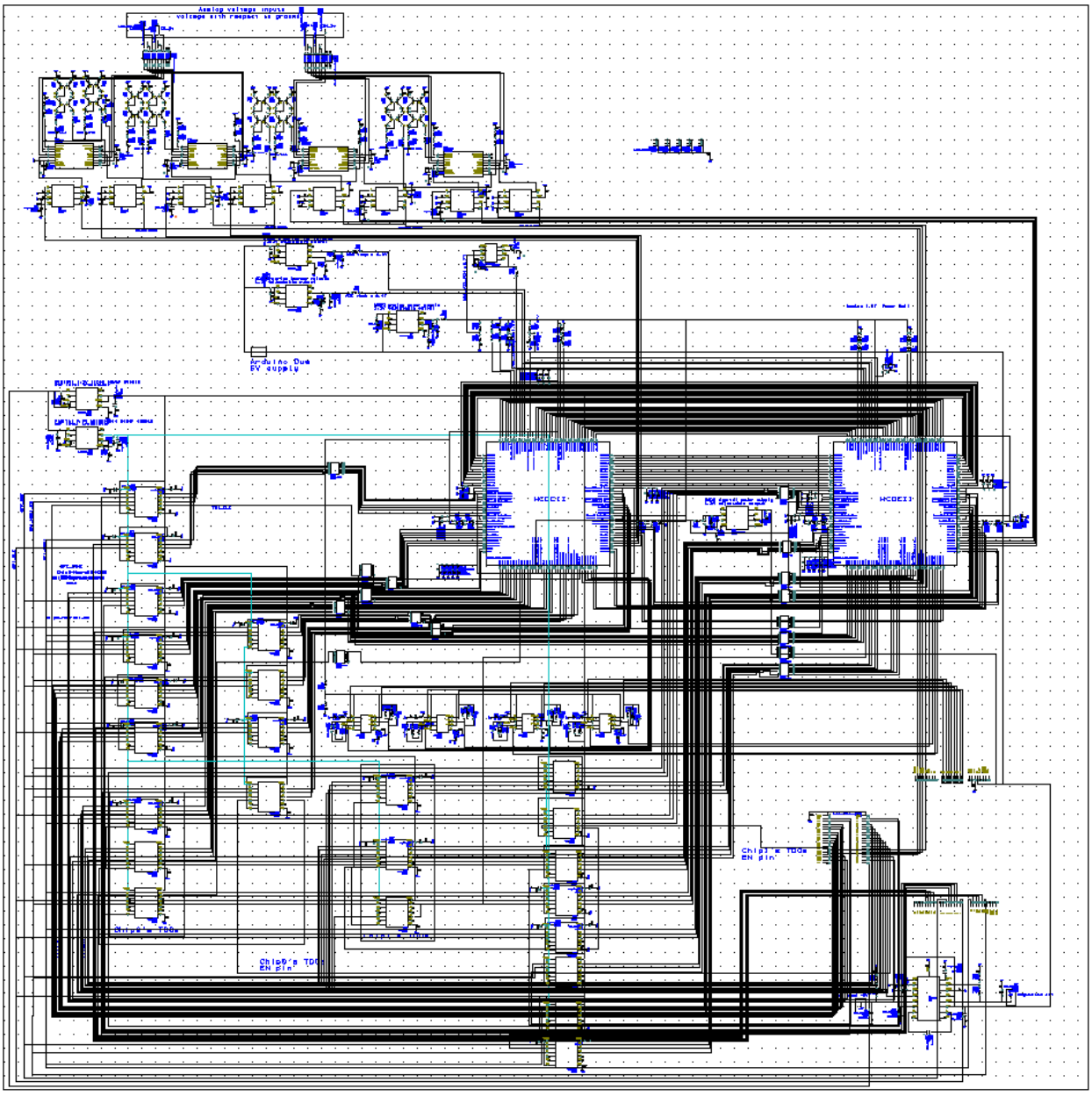

Figure D.2: The schematic of the demoboard with the 16th-order chip.

Appendix E

Solution of a 1-D Heat Equation

The 1-D heat equation is a PDE with the following form:

$$u_t(t,x) = u_{xx}(t,x) \tag{E.1}$$

where the temperature variable $u(t,x)$ is a function of time t and space x ($x \in [0,1]$). We would like to solve (E.1) with the boundary conditions of $u_{x=0} = 0$ and $u_{x=1} = 2$.

Applying finite difference methods [26], we could discretize the space variable x into equally spaced grids with grid space h. (E.1) becomes the following:

$$\frac{\mathrm{d}u_n}{\mathrm{d}t} = \frac{u_{n-1} - 2u_n + u_{n+1}}{2h} \tag{E.2}$$

where u_n is the temperature variable at grid n and has only one independent variable t.

We decided to discretize the space x with 16 internal nodes as shown in Fig. E.1.

We also need to do a time scaling trick in the expression before we map (E.2) on our 16th-order chip. Observe that there is a $\frac{1}{2h}$ factor on the right-hand side. We can move this

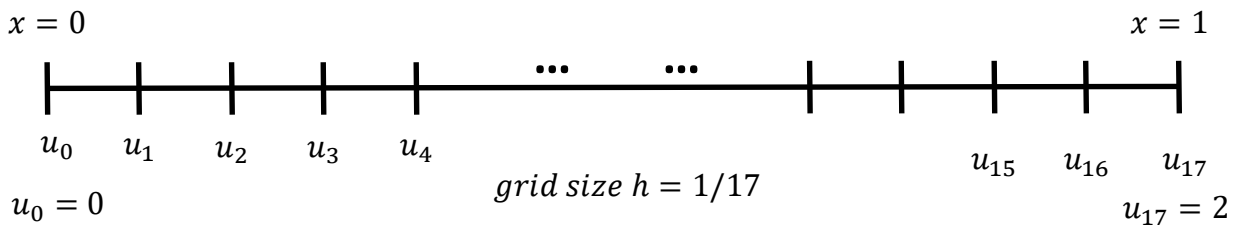

Figure E.1: Discretize the space into 16 internal nodes.

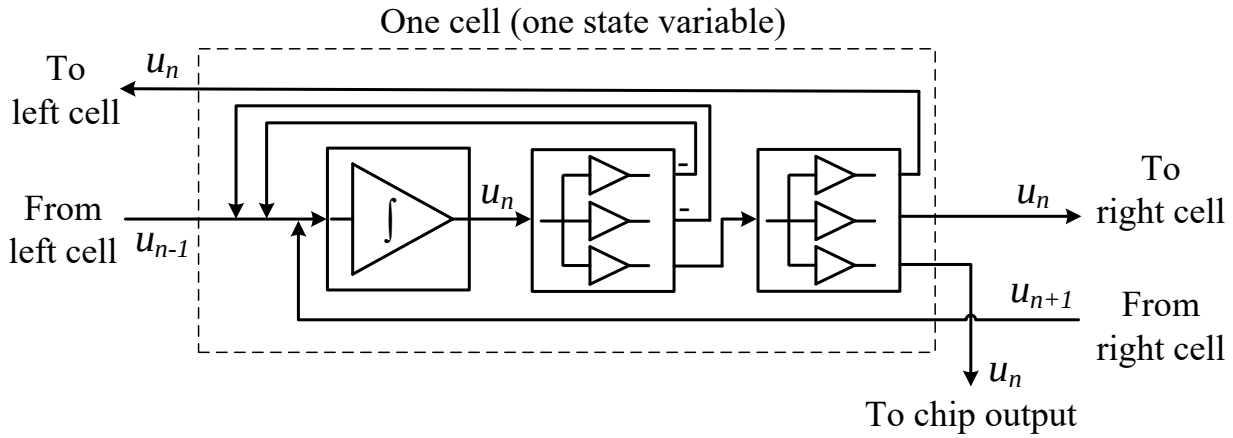

Figure E.2: The module for building the heat equation shown in (E.3).

factor to the left-hand side of the equation and combine it with the dt term:

$$\frac{\mathrm{d}u_n}{\mathrm{d}T} = u_{n-1} - 2u_n + u_{n+1} \tag{E.3}$$

where $T = \frac{t}{2h}$, $h = \frac{1}{17}$ and $n = 1, 2, 3, \ldots, 16$. Mapping (E.3) on our hybrid computer, we now have a new time scaling factor of $2h \times \omega_c$. For amplitude scaling, we map the temperature value range [-2, 2] directly to the hybrid computer range $[-2\,\mu\mathrm{A}, +2\,\mu\mathrm{A}]$.

We mapped (E.3) onto our 16th-order chip using the module cell associated with one state variable, as shown in Fig. E.2.

By cascading 16 such cells and setting the boundary conditions and the initial conditions,

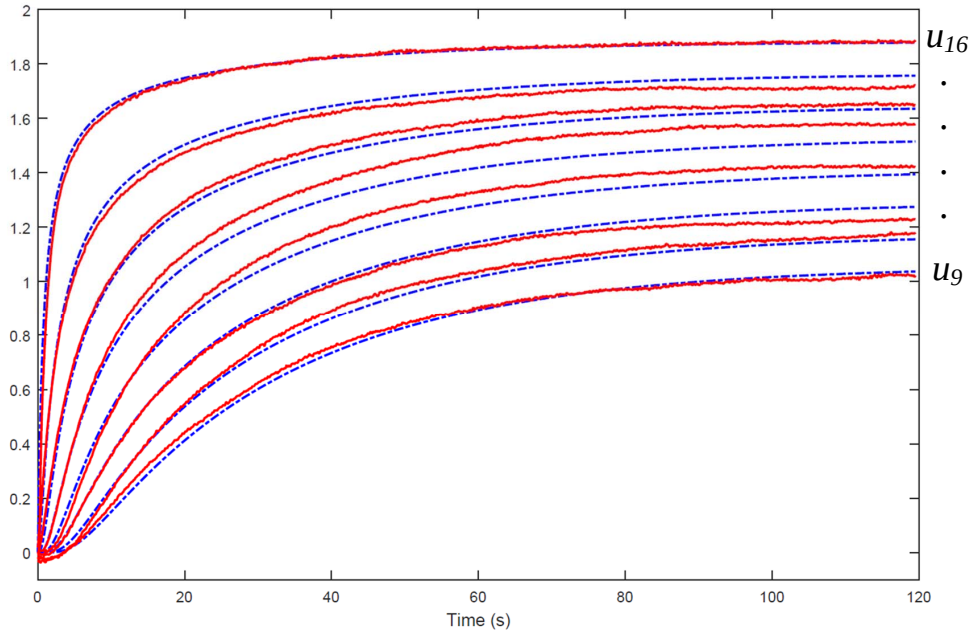

Figure E.3: The hybrid computer solution (red) and ideal Matlab solution (blue) of the heat equation in (E.1).

we could solve the 16th-order heat equations.

The solutions of (E.1) are shown in Fig. E.3 (after rescaling the time and amplitude) from our hybrid computer chip (red) for eight state variables $u_{16}, u_{15}, \ldots, u_9$. The ideal Matlab solutions are also shown (blue). The RMS error (relative to full scale) of our hybrid computer solution is 1.7%.