# Mixed Signal Processing: Hardware Electrodynamics, Integration Scaling, and Concept Simulation

by

**Scott Bannert, BS Mechanical Engineering**

**Masters Report**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Mechanical Engineering**

**The University of Texas at Austin**

August 2007

# Mixed Signal Processing: Hardware Electrodynamics, Integration Scaling, and Concept Simulation

**Approved by**
**Supervising Committee:**

Benito Fernandez

Michael Bryant

# Acknowledgments

I would like to give thanks to professors Benito Fernandez and Michael Bryant for providing the opportunity in research. They patiently provided an environment balancing ruminations of imagination and rigor.

Professor Fernandez always demonstrated an uplifting mindset providing encouragement. His open-mindedness is an inspiration for me to look at everything from multiple points of view; I suppose that some viewpoints hide all the details inside another.

Professor Bryant's meticulous nature directed my thoughts, teaching me that ideas become innovations when grounded to something tangible; I suppose that even the imagination must learn to transverse a cumbersome terrain–a point in reality.

Additionally, I would like to give thanks to my family for the financial support provided throughout my academic phase. Without their support, this report would have turned out differently.

Scott Bannert

*The University of Texas at Austin*
*August 2007*

# Mixed Signal Processing: Hardware Electrodynamics, Integration Scaling, and Concept Simulation

Scott Bannert, MSME

The University of Texas at Austin, 2007

Supervisors: Benito Fernandez
Michael Bryant

Basic hardware components for Mixed Signal Processing–analog computing inside a digital framework–are developed. The Hybrid Integrator integrates analog signals similar to analog computers avoiding op-amp saturation. These analog signals are scaled and combined with digital signals inside a Hybrid Format, forming a Hybrid Signal. The Hybrid Integrator Cell combines multiple Hybrid Signals through the use of multiple R-2R ladders and other supporting components for computing. Multiple Hybrid Integrator Cells combined in parrallel form a Hybrid Integrator Cell Array, syncronizing the integration of multiple mixed signals. An integration scaling process is developed, constraining Hybrid Signals during integration computing; this is discussed for both linear and nonlinear systems of first order ordinary differential equations (**ODE**'s). Lastly, program code is developed to simulate Mixed Signal Processing, demonstrating the expected results of Mixed Signal Processing for solving various simple ODE's.

# Contents

# List of Tables

# List of Figures

# Part I

# Introduction

# Chapter 1

# Overview

## 1.1 Computing Technologies

### 1.1.1 Digital

#### Basic Concept

Digital platforms are distinguished for their flexibility; they perform operations at extremely fast rates. The basic concept with digital technology is rather simple, every task is decomposed into simpler sub-tasks, until it becomes representable as binary code under a specified format. Each command is a sequence of logical transformations, staged as electrodynamic performances acting inside the computers hardware.

#### Acknowledged Success

As the underlying purpose for success, humans perceive through seeing, hearing, touching, tasting, and smelling. Isolated in digits, the abstract capability of this technology was not obvious:

*"I think there is a world market for maybe five computers."*
- Thomas Watson, chairman of IBM, 1943

Ironically, a growing supply count was found for computers, in a market that first visited outerspace, before aging into every classroom; individual commodities exchanging nearly weeks of labor while permanently replacing it. With an existence seemingly intangible to perception, the computer originated as a concept difficult to imagine, complex to understand, and outrageous to build.

As a rationalization to its success, the desire for a computer requires cognition–it needs to be learned. Initially, a logic machine performing symbolic reductions lacks in perception. Extending the concept to programming languages, a computer becomes easier to make associations with. Upon attaching a keyboard and monitor, a resemblance of personality can be perceived. When suited with proper software, it is capable of simplifying jobs. After programming it with games, it becomes a stimulus for emotion. The fundamental requirement for any concept to sell: it must be something that a consumer can perceive; people demand emotion and anything capable of stimulating it–computers sell because they are money saving toys.

#### Capabilities

A discrete framework is fully capable of approximating anything quantifiable. The only true limitation is time, requiring intervals to perform each discrete process. Digital Computing tends to reach limitations as from difficulties in parallel processing.

### 1.1.2 Analog Computing

Analog platforms are distinguished for their continuous nature. All data can be processed in a parrallel manner with behavior fundamental to math and physics, usually providing highly accurate, imprecise results. However, analog computing does not currently offer the flexibility of a digital environment, the circuits are inadequate in approximating arbitrary functions, a unique circuit layout is difficult to customize, and problems arise from op-amp saturation.

### 1.1.3 Hybrid Computing

#### Concept

Mixed Signal Processing (**MSP**) can solve **ODE**'s with a Hybrid Format merging analog signals with a primarily digital format. Integration is processed similarly to conventional analog computing with several modifications to overcome typical shortcomings. As with analog computing, **MSP** is performed in a parrallel fasion, possibly permiting integration speeds currently unattainable. Additionally, these simulation speeds are not dramatically affected by the size of the **ODE**, in contradistinction to a fully digital platform [2] [6].

#### Capabilities

In comparison to discrete logic, **MSP** hopes to revolutionize the way computers process **Continuous Logic**–an introduced term to describe the form of logic controlling the deductive reasoning of differential and integral equations, differing from any lexicon of discrete reductions. As a possibility, **MSP** could extend computer designs to include a Continuous Logic Unit (**CLU**)– a microchip that simulates the dynamics in an environment, by harnessing the natural dynamics of electricity.

### 1.1.4 Mixed Signal Processing

### 1.1.5 Hybrid Format

### 1.1.6 MSP Hardware

#### Components

The fundamental hardware component of Mixed Signal Processing is the Hybrid Integrator (**HxI**), containing the state of single Hybrid Signal. A **HxI** is extended with several components forming a Hybrid Integrator Cell (**HxC**), allowing the dynamics of it's Hybrid Signal state to be related various other Hybrid Signals. A Hybrid Integrator Cell Array (**HxA**) is an inline arrangement of hybrid integrator cells, with each HxI's signals routed to neighboring cells. A bus networks the HxA with a digital processor, which monitors and controls HxA's operation.

#### Electrodynamics

As required by **MSP**, the electrodynamics are modeled for a **HxI**, **HxC**, and **HxA**. In modeling hardware components, some simplified models are developed to derive how Mixed signals are processed while a detailed model is derived for developing a simulation of **MSP**. The hardware electrodynamics derived are controlled through digital inputs controlling switch configurations for routing various signals.

### 1.1.7 Integration Scaling

### 1.1.8 Simuating MSP

# Chapter 2

# Notation

## 2.1 Symbol Semantics

Symbol semantics, or notation, in this report utilizes a range of symbol properties to establish meaning. For non-linguistic references, the morphology of singular symbol consists of a root, font, superscript, subscript and modifiers. The symbol's domain is specified though font (see **Table 2.1**) and extended with superscripts (see **Table 2.3**). **Table 2.2** lists common symbols for parameters and electrodynamics. Scalars, matrices, sets, and organizations are depicted with regular font, bold font, blackboard font, and underlining as in **Table 2.1**; functions remain distinguished by a Calligraphic font.

Table 2.1: Use of fonts to distinguish symbols

| **Symbol Class** | | |
|---|---|---|
| **Font** | **Example** | **Association** |
| TypeWriter | $\mathtt{A}$ | Constants or Parameters |
| Utopia | A, **A** | Physical Dynamics |
| Roman | $\mathrm{A}$, $\mathbf{A}$ | General Mathematics |
| Sans serif | $\mathsf{A}$, $\boldsymbol{\mathsf{A}}$ | Digital Numbers |
| (underline) | $\underline{\mathsf{A}}$, $\underline{\boldsymbol{\mathsf{A}}}$ | Binary Strings (Array) |
| Caligraphic | $\mathcal{A}$, $\boldsymbol{\mathcal{A}}$ | Functions |
| Blackboard | $\mathbb{A}$ | Sets |
| (underline) | $\underline{\mathbb{A}}$ | Arrays |

Table 2.2: List of common parameter symbols

| Common Parameters | | |
|---|---|---|
| Symbol | Units | Association |
| `A` | dB | Gain |
| $\omega$ | Rad/sec | Circular Frequency |
| `R` | Ohms | Resistance |
| `C` | Farads | Capacitance |
| `n`, `N` | Integer | Size |
| **Electrodynamics** | | |
| Symbol | Units | Association |
| V | Volts | Voltage |
| Q | Colombs | Charge |
| I | Amps | Current |

## 2.2 Symbol Types

Singular symbols are classified as either organizations, operations, identies, or relations. Organizations contain the following heirarchy: **(1) Organization, (2) Set, (3) Matrice, (4) Scalar.** Operation symbols provide transformations, such as addition, subtraction, multiplication, exponents, derivatives, or integrals. Identity symbols represent concepts such as numbers, negative signatures, positive signatures, imaginary signatures, or variable signatures. The following relation symbols represent equivalence: **(1) '$\equiv$' Relates organizations, (2) '$=$' Relates operations, (3) '$\rightarrow$' Relates a symbol to meaning.** Functions are classified as operations, as demonstrated below for the function '$\mathcal{F}$' representing a line:

$$\mathcal{F}() \rightarrow \mathcal{F}(\mathrm{x}) = \mathrm{x} + 1$$

### 2.2.1 Organizations

In general, the term organization is used to represent scalars, matrices, sets, and organizations. In its broadest form, an organization ($\underline{\mathbb{A}}$ or $\underline{\mathbf{A}}$) is defined as a sequencing of sets, with a singular organization equivalent to a set. Sets ($\mathbb{A}$) are defined as a membership of matrices, with a singlular sets equivalent to a matrix. Similarly to

Table 2.3: Use of superscripts and subscripts

| Using SuperScripts | |
|---|---|
| **Superscript** | **Association** |
| `fb` | Feedback Circuit |
| `r`, `R` | Reset Circuit |
| `R`, `2R` | R-2R Ladder |
| `R2R` | R2R Unit |
| `ref` | Nominal/Assumed Value |
| $\lambda$ | Analog Bit |
| `a` | Op-Amp Output |
| $*$ | Inverting Terminal |
| `D` | Denormalization |

| Using Subscripts | |
|---|---|
| **Example** | **Purpose** |
| $\mathrm{A}_{1,2}$ | Specific Indexing |
| $\underline{\mathsf{A}}_{[0]}$ | Binary Indexing |
| $(\mathrm{A}_{i,j})_{j,i} = \mathbf{A}'$ | Re-ordering |
| $[\mathbf{A}]_{1,1} = \mathrm{A}_{1,1}$ | Specification |

tradition, matrices ($\mathbf{A}$) are defined as an arrangement of scalars.

### Matrices

The elements of a matrix are ordered inside brackets with rows indexed before columns. An indexed matrix retains the original matrices directionality, and indexing with the 'unspecified membership' symbol, $\emptyset$, skips an index:

$$\mathbf{A} \equiv \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \end{bmatrix} \equiv \begin{bmatrix} \mathbf{A}_{\emptyset,1} & \mathbf{A}_{\emptyset,2} & \cdots \end{bmatrix} \equiv \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

In using matrices, brackets are given subscript indices in the following two distinct manners:

**(1) The indice subscript is not referenced in the internal matrix**–this symbolizes a specific term of the matrice.

$$[\mathbf{A}]_i \equiv \mathbf{A}_i \neq \mathbf{A}$$
$$[\mathbf{A}]_{\emptyset,i} \equiv \mathbf{A}_{\emptyset,i} \neq \mathbf{A}$$

**(2) The indice subscript is referenced in the internal matrix**–this symbolizes a reordering of the matrix.

$$[\mathbf{A}_i]_i \equiv \mathbf{A}$$
$$[\mathbf{A}_{i,j}]_{j,i} \equiv \mathbf{A}'$$

Matrice multiplication is a non-commutatative operation:

$$[\ \mathbf{AB}\ ]_{i,j} \equiv \textstyle\sum_k \mathbf{A}_{i,k}\mathbf{B}_{k,j}$$

$$\mathbf{AB} \equiv \begin{bmatrix} \mathbf{A}_1\mathbf{B}_{\emptyset,1} & \mathbf{A}_1\mathbf{B}_{\emptyset,2} & \dots \\ \mathbf{A}_2\mathbf{B}_{\emptyset,1} & \mathbf{A}_2\mathbf{B}_{\emptyset,2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \neq \mathbf{BA}$$

Lastly, the dot product is an operation between two vectors; however, it has been defined to operate between matrices as:

$$\mathbf{A} \cdot \mathbf{B} \equiv \sum_k \mathbf{A}_k\mathbf{B}_k$$

**Sets**

A set is a membership of matrices (including scalars) contained with curly brackets, where conditional statements are separated by a colon. Elements of a set are contained without order, while indexing refers to a subset; and index removal corresponds to the union of all indexed sets. Contrary to tradition, the '$\emptyset$' is an identity for unspecified existence. If implied, '$\mathbb{R}$' represents the set of real numbers. Additionally, variable signatures are identified with '$\in$' such that '$\in \mathbb{R}$' symbolizes a variable of real numbers. For non-singular, non-variable sets, operations are defined to act on all members as demonstrated for multiplication:

$$\{1,2,3\}\,\{4,5\} \equiv \{4,5,8,10,12,15\}$$

**Organizations**

The developement of a model often requires many calculations which depend on other calculations. A situation frequently occurs which presents difficulty in communicating a process of calculations with functions. As a result, a symbolic representation for 'organizations' has been developed.

Organizations are defined as a sequencing of sets and/or other organizations; elements are contained inside parenthesis'. Organizations differ from matrices in four primary ways: **(1) Subscripted terms do not contain direction, (2) Multiplication is commutative, (3) Elements can have inconsistent dimensions, (4) An array can reference itself.**. Organizations are indexed in the following fashion:

$$\underline{\mathbb{A}} \equiv (\ \underline{\mathbb{A}}_1,\ \underline{\mathbb{A}}_2,\ \underline{\mathbb{A}}_3,\ \ldots\ )$$
$$\underline{\mathbb{A}}_1 \equiv (\ \underline{\mathbb{A}}_{1,1},\ \underline{\mathbb{A}}_{1,2},\ \underline{\mathbb{A}}_{1,3},\ \ldots\ )$$
$$\underline{\mathbb{A}}_{1,1} \equiv (\ \underline{\mathbb{A}}_{1,1,1},\ \underline{\mathbb{A}}_{1,1,2},\ \underline{\mathbb{A}}_{1,1,3},\ \ldots\ )$$

As a requirement, functions can be included in an organization if and only if all dependent variables are part of the organization. Symbol replacements inside organization are done in the following manner:

$$\underline{\mathbb{F}} \ \equiv\ (\ \mathbf{x},\ \mathsf{y} \rightarrow \mathcal{F}(\mathbf{x})\ )\ \equiv\ (\ \mathbf{x},\ \mathsf{y}\ )$$

**Digital Information**

**Table 2.4** gives common symbols for binary strings, each having a unique symbols treated as irregularly indexed organizations. The least significant bit (**LSB**) is indexed to zero, moving leftward to the most significant bit (**MSB**). Additionally, subscripts are contained in brackets as seen below:

$$\underline{\mathsf{A}} \equiv (\ \underline{\mathsf{A}}_{[n-1]},\ \ldots\ ,\underline{\mathsf{A}}_{[0]}\ )$$

The use of bracketed subscripting eliminates confusion which arrives in symbolizing a vector (or matrix) of binary strings such as:

$$\underline{\mathbf{\mathsf{A}}} \equiv [\, \underline{\mathsf{A}}_1, \quad \underline{\mathsf{A}}_2, \quad \ldots \,]'$$

Table 2.4: List of digital symbols

| Digital Information | |
|---|---|
| **Symbol** | **Representation** |
| $\underline{\mathbb{X}}$ | State Register |
| $\underline{\mathsf{M}}$ | Mantissa |
| $\underline{\mathsf{E}}$ | Exponent |
| $\underline{\mathsf{B}}$ | Sign Bit |
| $\underline{\mathsf{K}}$ | Scaling Index Register |
| $\underline{\mathsf{R}}$ | R2R Register |
| $\underline{\mathsf{S}}$ | Reset Circuit |

### 2.2.2 Variable Expansions

Variable expansions us character modifiers ($\hat{}$, $\bar{}$, $\Delta$, $\delta$) to decompose a variables into formatted components. Point expansions are for functions approximation, while discrete expansions decomposes a variable into components for hybrid formatting. Combined expansions are used when expressing function approximations in a hybrid format.

**Point Expansion:**

$$\mathrm{A} \;=\; \hat{\mathrm{A}} \;+\; \Delta\mathrm{A} \;=\; \mathcal{A}(\hat{t}) + \Delta\mathcal{A}(\Delta t) \;=\; \mathcal{A}(t)$$
$$t \;=\; \hat{t} \;+\; \Delta t$$

**Discrete Expansion:**

$$\mathrm{A} \;=\; \bar{\mathrm{A}} \;+\; \delta\mathrm{A} \;=\; \mathcal{A}(\bar{t}) + \delta\mathcal{A}(\delta t) \;=\; \mathcal{A}(t)$$
$$t \;=\; \bar{t} \;+\; \delta t$$

**Combined Expansion:**

$$\begin{aligned} \mathrm{A} \;&=\; \hat{\mathrm{A}} + \Delta\bar{\mathrm{A}} \;+\; \delta\mathrm{A} \;=\; \mathcal{A}(\hat{t}) + \Delta\mathcal{A}(\Delta\bar{t}) + \delta\mathcal{A}(\delta t) \;=\; \mathcal{A}(t) \\ t \;&=\; \bar{t} \;+\; \Delta\bar{t} \;+\; \delta t \end{aligned}$$

# Chapter 3

# Signal Formats

## 3.1 Signed Two's Compliment Format

Using this binary format, integer are formatted for addition and subtraction as binary strings. The **MSB** represents sign, with '1' for positive and '0' for negative. All remaining bits express the integer in two's compliment format. Sign changes are formed through incrementing the binary compliment; lastely, zero must be represented with a positive sign. For a 3-bit number, the following combinations exist:

$$\begin{array}{ccccccc}
0 & = & (1:00) & \rightarrow & 1+011_2 & = & (1:00) = & 0 \\
1 & = & (1:01) & \rightarrow & 1+010_2 & = & (0:11) = & -1 \\
2 & = & (1:10) & \rightarrow & 1+001_2 & = & (0:10) = & -2 \\
3 & = & (1:11) & \rightarrow & 1+000_2 & = & (0:01) = & -3 \\
-1 & = & (0:11) & \rightarrow & 1+100_2 & = & (1:01) = & 1 \\
-2 & = & (0:10) & \rightarrow & 1+101_2 & = & (1:10) = & 2 \\
-3 & = & (0:01) & \rightarrow & 1+110_2 & = & (1:11) = & 3 \\
\text{NaI} & = & (0:00) & \rightarrow & 1+111_2 & = & (0:00) = & \text{NaI}
\end{array}$$

Sign change operations in Signed Two's Compliment Format can be processed with the following binary logic operators: 'or' (+), 'and' (&&), and 'exclusive or' ($\oplus$). Sign change operations are described by:

$$\vec{-} \rightarrow \left\{ \begin{array}{c} \underline{\mathsf{B}} \;\equiv\; \vec{-}\underline{\mathsf{A}} \\ \mathcal{D}() \rightarrow \left\{ \begin{array}{c} \mathcal{D}(0) = 0 \\ \mathcal{D}(i+1) = \mathcal{D}(i) + \underline{\mathsf{A}}_{[i]} \end{array} \right\} \\ \underline{\mathsf{B}}_{[i]} \equiv \mathcal{D}(i) \oplus \underline{\mathsf{A}}_{[i]} \end{array} \right\}$$

Addition between two strings in this format is symbolized with the '$\vec{+}$' symbol. This is processed as typical binary addition, with the result being extended one bit:

$$\begin{array}{rcccc} 0+0 & = & (1:00)\,\vec{+}\,(1:00) & = & (1:000) \\ 0+1 & = & (1:00)\,\vec{+}\,(1:01) & = & (1:001) \\ 0-1 & = & (1:00)\,\vec{+}\,(0:11) & = & (0:111) \\ 3+2 & = & (1:11)\,\vec{+}\,(1:10) & = & (1:101) \\ 3-1 & = & (1:11)\,\vec{+}\,(0:11) & = & (1:010) \\ 1-3 & = & (1:01)\,\vec{+}\,(0:01) & = & (0:110) \\ -3-3 & = & (0:01)\,\vec{+}\,(0:01) & = & (0:010) \end{array}$$

The addition operation for two n-bit numbers, resulting in an (n+1)-bit number, is described below:

$$\vec{+} \rightarrow \left\{ \begin{array}{c} \underline{\mathsf{A}}\,\vec{+}\,\underline{\mathsf{B}} \;\equiv\; \underline{\mathsf{C}} \\ \mathcal{D}() \rightarrow \left\{ \begin{array}{c} \mathcal{D}(0) = 0 \\ \mathcal{D}(i+1) = (\mathcal{D}(i) + \underline{\mathsf{A}}_{[i]})\&\&(\mathcal{D}(i) + \underline{\mathsf{B}}_{[i]}) \end{array} \right\} \\ \underline{\mathsf{C}}_{[i]} \equiv \underline{\mathsf{A}}_{[i]} \oplus \underline{\mathsf{B}}_{[i]} \oplus \mathcal{D}_(i) \\ \underline{\mathsf{C}}_{[n]} = \mathcal{D}(n) \end{array} \right\}$$

## 3.2 Floating Point Representation

Numbers represented in floating point (**FP**) have a sign bit, exponent register, and mantissa register, as shown below [5]:

$\underline{\mathsf{B}}$ - Sign Bit (1 bit)
$\underline{\mathsf{M}}$ - Mantissa Register ($\mathtt{M}$-bits)
$\underline{\mathsf{E}}$ - Exponent Register ($\mathtt{E}$-bits)

A floating point number has the form:[2]

$$\mathsf{x} = \mathsf{b}\,\mathsf{m}\,2^{\mathsf{e}} \tag{3.1}$$

where,

$\mathsf{x}$ - Number represented with Floating Point Format
$\mathsf{b}$ - Sign Value from the Sign Bit, $\underline{\mathsf{B}}$
$\mathsf{m}$ - Mantissa value from the Mantissa Register, $\underline{\mathsf{M}}$
$\mathsf{e}$ - Exponent value from the Exponent Register, $\underline{\mathsf{E}}$

### 3.2.1 Floating Point Ranges

Floating point ranges include the normalized range (normal range), denormalized range (floor of representation), or the Inf/Nan Range (ceiling of representation).

#### Normalized Representation

The normalized range represents typical nonzero signal values when:

$$2^{-2^{\mathtt{E}-1}+2} \leq \|\mathsf{x}\| \leq \left(2 - 2^{-\mathtt{M}}\right) 2^{2^{\mathtt{E}-1}-1}$$

Inside this range, the exponent register contain:

$$\underline{\mathsf{E}} \notin \{(\ 0,\ 0,\ 0,\ \cdots\ 0\ ),\ (\ 1,\ 1,\ 1,\ \cdots\ 1\ )\}$$

#### Denormalized Representation

Inside the denormalized range:

[2]This equation doesn't represent floating point signals for Inf or Nan

$$\|\mathrm{x}\| < 2^{-2^{\mathtt{E}-1}-\mathtt{M}+2}$$

Below this range, the number is truncated as zero. The denormalized range is indicated with the exponent register as:

$$\underline{\mathsf{E}} \equiv (\ 0,\ 0,\ \cdots\ 0\ )$$

**Infinity/Not-A-Number Representation**

The Infinity (Inf) / Not-A-Number (Nan) Range exists when:

$$\|\mathrm{x}\| \geq \left(2 - 2^{-\mathtt{M}}\right) 2^{2^{\mathtt{E}-1}-1}$$

Symbolized with the exponent register as:

$$\underline{\mathsf{E}} \equiv (\ 1,\ 1,\ 1,\ \cdots\ 1\ )$$

### 3.2.2 Digital Signal Evaluation

After determining the proper floating point signal range (normalized, denormalized, Inf/Nan), the sign bit, mantissa register, and exponent register may be used to determine the floating point value.

**Sign Bit**

The sign value is represented by the sign bit as follows:

$$\mathsf{b} = (-1)^{\underline{\mathsf{B}}} = \begin{cases} +1, & \underline{\mathsf{B}} = 0 \\ -1, & \underline{\mathsf{B}} = 1 \end{cases}$$

**Mantissa Register**

Inside the normalized range, the mantissa value is calculated as:

$$\mathsf{m} = 1 + \sum_{i=0}^{\mathtt{M}-1} \left(\underline{\mathsf{M}}_{[i]} 2^{i-\mathtt{M}}\right)$$

For numbers in the denormalized range:

$$\mathsf{m} = \sum_{i=0}^{\mathtt{M}-1} \left( \underline{\mathsf{M}}_{[i]}\ 2^{i-\mathtt{M}} \right)$$

Numbers too small for representation with a denormalized mantissa are truncated to zero. Inside the Inf/Nan Range, infinity is distinctly represented by:

$$\left[ \underline{\mathsf{M}}_{[i]} = 1 \right]_{i=0,1,2,..,\mathtt{M}-1} \overset{equiv}{\longleftrightarrow} \underline{\mathsf{M}} \in \{(\ 1,\ 1,\ 1,\ \cdots\ 1\ )\}$$

All other mantissa values inside this range signify not-a-number(Nan).

**Exponent Register**

In the normalized range, the exponent value is calculated as:

$$\mathsf{e} = -2^{\mathtt{E}-1} + 1 + \sum_{i=0}^{\mathtt{E}-1} \left( \underline{\mathsf{E}}_{[i]} 2^{i} \right)$$

For the denormalized range:

$$\mathsf{e} = -2^{\mathtt{E}-1} + 2$$

Inside the Inf/Nan range, the exponent register contains no representable value, but contains the following digital signal:

$$\left[ \underline{\mathsf{E}}_{[i]} = 1 \right]_{i=0,1,2,..,\mathtt{E}-1} \overset{equiv}{\longleftrightarrow} \underline{\mathsf{E}} \in \{(\ 1,\ 1,\ 1,\ \cdots\ 1\ )\}$$

### 3.2.3 IEEE 754 Format

IEEE Standard 754 Format employs single precision and double precision floating point.

**Single Precision Floating Point**

Single precision floating point uses 32 bits (typically assigned locations [00-31]) to numerically represent a single value:

1 Sign Bit [31]
8 Bit Exponent Register [23-30], $\mathtt{E} = 8$
23 Bit Mantissa Register [00-22], $\mathtt{M} = 23$

The Normalized Range with IEEE 754 Single Precision Floating Point is:

$$1.175 \times 10^{-38} \approx 2^{-126} \leq \mathrm{x} \leq (2 - 2^{-23})\, 2^{127} \approx 3.402 \times 10^{38}$$

**Double Precision Floating Point**

Double precision floating point uses 64 bits (typically assigned locations [00-63]) to numerically represent a single numerical value:

1 Sign Bit [63]
11 Bit Exponent Register [52-62], $\mathtt{E} = 11$
52 Bit Mantissa Register [00-51], $\mathtt{M} = 52$

The Normalized Range with IEEE 754 Double Precision Floating Point is:

$$2.225 \times 10^{-308} \approx 2^{-1022} \leq \mathrm{x} \leq (2 - 2^{-52})\, 2^{1023} \approx 1.798 \times 10^{308}$$

## 3.3 Other Digital Registers

An unsigned integer register and a scaling index register will be introduced for mixed signal number representation.

**Unsigned Integer Registers**

An unsigned integer register represents positive integer values by:

$\underline{\mathsf{P}}$ - Integer Register ($\underline{\mathsf{P}}_{[i]}$ represents bit 'i', $\underline{\mathsf{P}}_{[0]}$ represents LSB)
$\mathtt{p}$ - Integer value represented by the Integer Register
$\mathtt{P}$ - Number of bits in the Integer Register

Signal overflow (similar to Inf/Nan in FP) is represented by:

$$\left[\underline{\mathsf{P}}_{[i]} = 1\right]_{i=0,1,2,..,\mathtt{P}-1} \quad \overset{equiv}{\longleftrightarrow} \quad \underline{\mathsf{P}} \in [(\ 1,\ 1,\ 1,\ \cdots\ 1\ )]$$

The unsigned integer value is calculated as:

$$\mathsf{p} = \sum_{i=0}^{\mathtt{P}-1} \left(\underline{\mathsf{P}}_{[i]}\ 2^i\right)$$

**Scaling Index Register**

A scaling index register scales the analog bit's value in a special mixed signal representation that uses the following terms:

$\underline{\mathsf{K}}$ - Scaling Index Register ($\underline{\mathsf{K}}_{[i]}$ represents bit 'i', $\underline{\mathsf{K}}_{[0]}$ represents LSB)
$\mathsf{k}$ - Positive Integer value represented by the Scaling Index Register
$\mathtt{k}_{\mathtt{max}}$ - Maximum allowable value of $\mathsf{k}$
$\mathtt{k}_{\mathtt{min}}$ - Minimum allowable value of $\mathsf{k}$
$\mathtt{K}$ - Number of bits in the Integer Register

The value represented by the Scaling Index Register is calculated as:

$$\mathsf{k} = \sum_{i=0}^{\mathtt{K}-1} \left(\underline{\mathsf{K}}_{[i]}\ 2^i\right)$$

The scaling index register must be inside the range:

$$\mathtt{k}_{\mathtt{min}} \leq \mathsf{k} \leq \mathtt{k}_{\mathtt{max}}$$

## 3.4 Mixed Signal Representation

Mixed signal combines digital information with an analog signal. Two mixed signal forms are developed, a linear scaled mixed signal and a floating point mixed signal; the floating point mixed signal format is used in later chapters.

### Analog Bit

The analog bit allows a continuous representation of numbers between discrete intervals of digital representations. The general form for an analog bit is [2] [6]:

$$\lambda = \frac{\mathrm{V}^{\lambda}}{\mathtt{V}^{\mathtt{ref}}}$$

where,

$\lambda$ - Value of Analog Bit ( $-1 \leq \lambda \leq 1$ )
$\mathrm{V}^{\lambda}$ - Analog Bit Voltage
$\mathtt{V}^{\mathtt{ref}}$ - Reference Voltage of the Analog Signal ( $-\mathtt{V}^{\mathtt{ref}} \leq \mathrm{V}^{\lambda} \leq \mathtt{V}^{\mathtt{ref}}$ )

Whenever the analog signal reaches a positive or negative limit, it's value is reset to zero as follows:

$$\left.\begin{array}{c} \lambda = 1 \\ \frac{\mathrm{d}\lambda}{\mathrm{d}t} \geq 0 \end{array}\right\} \quad \xrightarrow{Reset} \quad \lambda = 0$$

$$\left.\begin{array}{c} \lambda = -1 \\ \frac{\mathrm{d}\lambda}{\mathrm{d}t} \leq 0 \end{array}\right\} \quad \xrightarrow{Reset} \quad \lambda = 0$$

Numerical compensation for an analog bit reset is described later.

### 3.4.1 Linear Mixed Signal Format

A linear mixed signal combines analog bit signal, unsigned integer register, sign bit, and gain constant to represent signals bounded between known levels, requiring a relatively constant level of signal representation [1] [3].

$$\mathrm{x} = \mathrm{x}_0 + \Delta\mathrm{x} = \mathtt{C}\ (\mathsf{b}\ \mathsf{p} + \lambda)$$
$$\mathrm{x}_0 = \mathtt{C}\ \mathsf{b}\ \mathsf{p}$$
$$\Delta\mathrm{x} = \mathtt{C}\ \lambda$$

where,

$\mathsf{x}$ - Total value represented with a Mixed Signal

$\mathsf{x}_0$ - Digital (Discrete) portion of Mixed Signal value

$\Delta\mathsf{x}$ - Analog (Dynamic) portion of Mixed Signal value

$\lambda$ - Analog Bit Value

$\mathsf{b}$ - Sign value represented by Sign Bit($\underline{\mathsf{B}}$)

$\mathsf{p}$ - Integer represented with Unsigned Integer Register ($\underline{\mathsf{P}}$, $\mathtt{P}$ Bits)

$\mathsf{C}$ - User-Defined Scaling Constant

Upon analog bit reset, numerical continuity is maintained as follows:

$$\lambda \to +1 \quad \Rightarrow \quad \mathsf{b\,p} \to \mathsf{b\,p} + 1$$
$$\lambda \to -1 \quad \Rightarrow \quad \mathsf{b\,p} \to \mathsf{b\,p} - 1$$

### 3.4.2 Floating Point Hybrid Format

Floating point mixed signal format uses an analog bit signal, mantissa register, exponent register, scaling index register, and sign bit. The format transforms standard floating point into a mixed signal format compatible with standard floating point. A scaling index controls the scaled representation, allowing trade offs amongst speed and accuracy [1] [3].

$$\mathsf{x} = \mathsf{x}_0 + \Delta\mathsf{x} = \left(\mathsf{b\ m} + \lambda\, 2^{\mathsf{k}-\mathtt{M}}\right) 2^{\mathsf{e}}$$
$$\mathsf{x}_0 = \ \mathsf{b\ m}\ 2^{\mathsf{e}}$$
$$\Delta\mathsf{x} = \lambda\, 2^{\mathsf{k}-\mathtt{M}+\mathsf{e}}$$

where,

$\mathsf{x}$ - Total value represented with a Mixed Signal

$\mathsf{x}_0$ - Digital (Discrete) portion of Mixed Signal value

$\Delta\mathsf{x}$ - Analog (Dynamic) portion of Mixed Signal value

$\lambda$ - Analog Bit Value

$\mathsf{b}$ - Sign value represented by Sign Bit($\underline{\mathsf{B}}$)

$\mathsf{m}$ - Mantissa represented by Mantissa Register ($\underline{\mathsf{M}}$, $\mathtt{M}$ Bits)

$\mathsf{e}$ - Exponent represented by Exponent Register ($\underline{\mathsf{E}}$, $\mathtt{E}$ Bits)

$\mathsf{k}$ - Scaling Index represented by Scaling Index Register ($\underline{\mathsf{K}}$, $\mathtt{K}$ Bits)

During an analog bit reset, the following transitions occur:

$$\lambda \to +1 \quad \Rightarrow \quad \mathsf{m} \to \mathsf{m} + \mathsf{b}\, 2^{\mathtt{k-M}}$$
$$\lambda \to -1 \quad \Rightarrow \quad \mathsf{m} \to \mathsf{m} - \mathsf{b}\, 2^{\mathtt{k-M}}$$
$$\mathsf{m} \to 2 \quad \Rightarrow \quad [\,\mathsf{e},\ \mathsf{m}\,] \to [\,\mathsf{e}+1,\ \mathsf{m}/2\,]$$
$$\mathsf{m} \to 1 \quad \Rightarrow \quad [\,\mathsf{e},\ \mathsf{m}\,] \to [\,\mathsf{e}-1,\ 2\,\mathsf{m}\,]$$

**Example: floating point mixed signal value:**

$$\begin{array}{ll}
\underline{\mathsf{B}} = (0) & \mathsf{b} = -1^0 = +1 \\
\underline{\mathsf{M}} = (1,0,1,1) & \mathsf{m} = (1) + 2^{-1} + 0 + 2^{-3} + 2^{-4} = 1.6875 \\
\underline{\mathsf{E}} = (0,1,0,1) & \mathsf{e} = (-2^{4-1}+1) + 0 + 2^2 + 0 + 2^0 = -2 \\
\mathsf{k} = 2 & \Delta \mathrm{x} = \lambda\, 2^{-4+2+\mathsf{e}} = \lambda\, 2^{-2+\mathsf{e}} \\
\lambda = +0.137 & \mathrm{x} = (1.6875 + 0.137 \times 2^{-2})\, 2^{-2}
\end{array}$$

Three examples demonstrating signal transitions during analog bit reset:

**Example 1:** $(\mathrm{x} = 8)$

$$\begin{array}{lcl}
\underline{\mathsf{B}} = (0) & & \underline{\mathsf{B}} = (0) \\
\underline{\mathsf{E}} = (1,0,0,1) & & \underline{\mathsf{E}} = (1,0,1,0) \\
\underline{\mathsf{M}} = (1,1,1) & \Longrightarrow & \underline{\mathsf{M}} = (0,0,0) \\
\mathsf{k} = 0 & & \mathsf{k} = 0 \\
\lambda = +1 & & \lambda = 0 \\
\mathrm{x} = (1.875 + (1.0)\, 2^{-3}) \times 2^2 & & \mathrm{x} = (1.000) \times 2^3
\end{array}$$

**Example 2:** $(\mathrm{x} = 3.5)$

$$\begin{aligned} &\underline{\mathsf{B}} = (0) \\ &\underline{\mathsf{E}} = (1,0,0,1) \\ &\underline{\mathsf{M}} = (0,0,0) \\ &\mathsf{k} = 0 \\ &\lambda = -1 \\ &\mathrm{x} = (1.000 + (-1.0)\,2^{-3}) \times 2^2 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} &\underline{\mathsf{B}} = (0) \\ &\underline{\mathsf{E}} = (1,0,0,0) \\ &\underline{\mathsf{M}} = (1,1,0) \\ &\mathsf{k} = 0 \\ &\lambda = 0 \\ &\mathrm{x} = (1.750) \times 2^1 \end{aligned}$$

**Example 3:** $(\mathrm{x} = -9)$

$$\begin{aligned} &\underline{\mathsf{B}} = (1) \\ &\underline{\mathsf{E}} = (1,0,0,1) \\ &\underline{\mathsf{M}} = (1,1,0) \\ &\mathsf{k} = 2 \\ &\lambda = -1 \\ &\mathrm{x} = (-1.750 + (-1)\,2^{-1}) \times 2^2 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} &\underline{\mathsf{B}} = (1) \\ &\underline{\mathsf{E}} = (1,0,1,0) \\ &\underline{\mathsf{M}} = (0,0,1) \\ &\mathsf{k} = 2 \\ &\lambda = 0 \\ &\mathrm{x} = (-1.250) \times 2^3 \end{aligned}$$

**Scaling Index Evaluation**

For systems of ODE's with solutions demonstrating large percent changes, the scaling index avoids slow simulation speeds. The mixed signal's analog component, $\Delta\mathrm{x}$, is sized by the scaling index. A larger $\Delta\mathrm{x}$ (larger $\mathsf{k}$) has greater integration storage capacity, and thus requires less frequent resets. This is demonstrated below for single precision floating point mixed signal with $\mathrm{x}_0 = 1$:

- $\mathsf{k} = 0 \;\Rightarrow\; \Delta\mathrm{x} = \lambda \times 2^{-23} = \frac{\lambda}{8{,}388{,}608}$
- $\mathsf{k} = 1 \;\Rightarrow\; \Delta\mathrm{x} = \lambda \times 2^{-22} = \frac{\lambda}{4{,}194{,}304}$

$\vdots$

- $\mathsf{k} = 13 \;\Rightarrow\; \Delta\mathrm{x} = \lambda \times 2^{-10} = \frac{\lambda}{1{,}024}$

$\vdots$

- $\mathsf{k} = 21 \;\Rightarrow\; \Delta\mathrm{x} = \lambda \times 2^{-2} = \frac{\lambda}{4}$
- $\mathsf{k} = 22 \;\Rightarrow\; \Delta\mathrm{x} = \lambda \times 2^{-1} = \frac{\lambda}{2}$

Consider the number of analog bit resets necessary for a single precision floating point mixed signal to double in value:

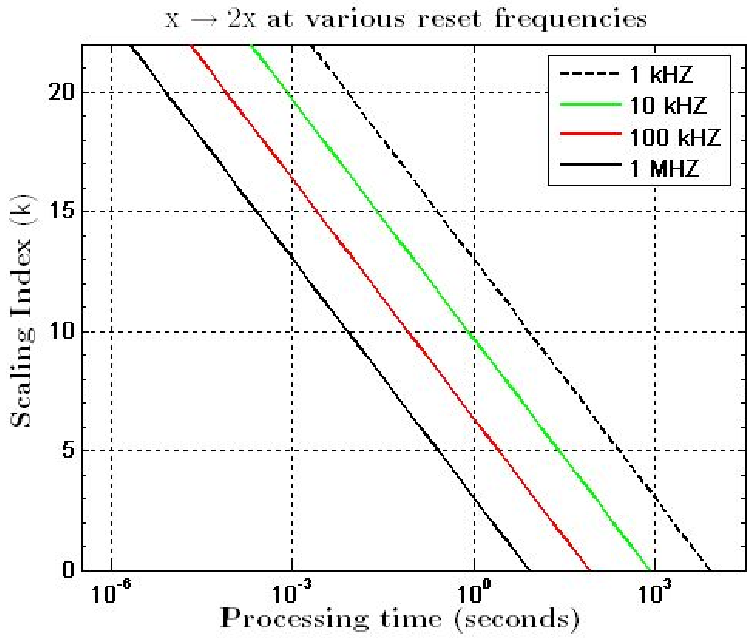


Figure 3.1: Effect of Scaling Index on processing time ($\mathtt{x} \rightarrow 2\mathtt{x}$) at various reset frequencies for single precision floating point mixed signal format ($\mathtt{E} = 8, \mathtt{M} = 23$).

$$\mathrm{x} \rightarrow 2\,\mathrm{x} \Rightarrow 2^{23-\mathsf{k}} \quad \textit{Resets}$$

$$\mathbf{1}.000,000,000,000,000,000,000,00 \rightarrow \mathbf{1}.111,111,111,111,111,111,111,11$$

A practical limitation exists on rate of resets, estimated at $10^6$ Hz (1 MHz). Without a scaling index, approximately 8 million resets must occur when a variable doubles, requiring seconds to process. With an intermediate scaling index value such as ($\mathsf{k} = 10$), the 8 million resets reduces to 8 thousand, allowing faster simulations. **Figure 3.1** graphically demonstrates the relationship between processing time and the scaling index.

# Chapter 4

# Basic Modeling

## 4.1 Modeling Op-Amp Behavior

An op-amp with non-inverting input grounded (**Figure 4.1**) has the transfer function[1] [1] [2] [3] [4]:

$$\frac{\mathrm{V}^a}{\mathrm{V}^*} = \frac{-\mathtt{A}^\circ}{\tau s + 1} = \frac{-\mathtt{A}^\circ}{\mathtt{A}^\circ \frac{s}{\omega^1} + 1}$$

Converted into state-space,

$$\frac{\mathrm{dV^a}}{\mathrm{d}t} = -\frac{\omega^1}{\mathtt{A}^\circ}\mathrm{V^a} - \omega^1\mathrm{V}^* \tag{4.1}$$

where,

$\mathrm{V^a}$ - Op-amp Integrator Voltage
$\mathrm{V}^*$ - Op-amp Inverting Terminal Voltage
$\omega^1$ - Unity Gain Frequency of op-amp
$\mathtt{A}^\circ$ - DC gain of op-amp
$\tau$ - Time constant for response
s - Complex variable of Laplace Transform

[1] A first order model describes the op-amps behavior over the frequency range of interest for Mixed Signal Integration [2]

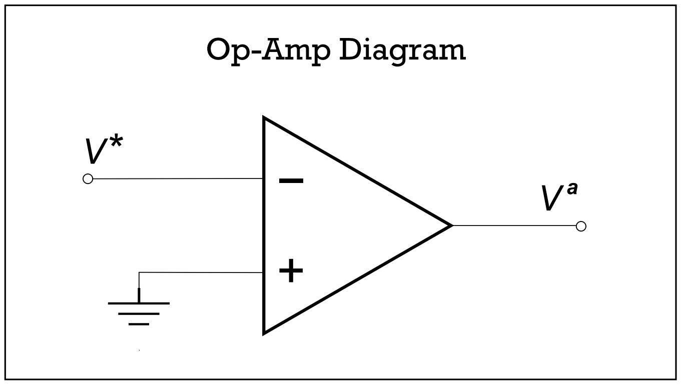


Figure 4.1: Diagram representing a typical Op-amp

## 4.2 Capacitor Circuit

A capacitor constituative law:

$$\Delta \mathrm{V} = \frac{\mathrm{Q}}{\mathtt{C}} \tag{4.2}$$

where,

> $\Delta$V - Voltage drop across Capacitor
> Q - Charge of Capacitor
> $\mathtt{C}$ - Capacitance

### Including Resistance and Inductance

In realistic circuits, a small amount of both resistance and inductance exists which cannot be avoided. To analyze this effect, such a circuit has been briefly analyzed:

$$\Delta \mathrm{V} = \frac{\mathrm{Q}}{\mathtt{C}} + \mathtt{R}\frac{\mathrm{dQ}}{\mathrm{d}t} + \mathtt{L}\frac{\mathrm{d}^2\mathrm{Q}}{\mathrm{d}t^2}$$

For an ideal capacitor circuit,

$$\frac{\Delta \mathrm{V}}{\mathrm{Q}/\mathtt{C}} \equiv 1$$

The circuits transfer function when including resistance and inductance:

$$\frac{\Delta \mathrm{V}}{\mathrm{Q}/\mathtt{C}} \equiv 1 + \mathtt{RC}s + \mathtt{LC}s^2$$

where,

$\mathtt{R}$ - Series Resistance included in Model
$\mathtt{L}$ - Series Inductance included in Model
$s$ - Complex Variable of Laplace Transform

Considered in the frequency domain ($\omega$ - circular frequency), the circuits actual response approaches the ideal responce when:

$$\text{ACTUAL} \rightarrow \text{IDEAL, when} \quad \omega \ll \left\{ \frac{1}{\mathtt{RC}}, \frac{1}{\sqrt{\mathtt{LC}}} \right\}$$

## 4.3 R2R Ladder Modeling

An R-2R ladder converts input voltage into an output current **Figure 4.2**, where:

$\mathrm{V}^{\mathrm{in}}$ - Input Voltage to R-2R Ladder
$\mathrm{I}^{\mathrm{R2R}}$ - Output Current of R-2R Ladder
$\mathrm{V}^{*}$ - Inverting Terminal Potential supplied to the R-2R Ladder
$\mathtt{R}^{\mathtt{R}}_{\mathtt{i}}$ - Resistance of specific 'R' Resistor
$\mathrm{I}^{\mathrm{R}}_{\mathrm{i}}$ - Current passing through resistor, $\mathtt{R}^{\mathtt{R}}_{\mathtt{i}}$
$2\mathtt{R}^{\mathtt{2R}}_{\mathtt{i}}$ - Resistance of specific '2R' Resistor
$\mathrm{I}^{\mathrm{2R}}_{\mathrm{i}}$ - Current passing through resistor, $\mathtt{R}^{\mathtt{2R}}_{\mathtt{i}}$
$\underline{\mathsf{S}}_{[i]}$ - State of R-2R Ladder Switches, ($\mathtt{N}^{\mathsf{S}}$ bits, $\underline{\mathsf{S}}_{[1]} - MSB$)
$\mathsf{S}$ - R-2R Mantissa formed using R-2R Ladder Switches, $\underline{\mathsf{S}}$

### 4.3.1 Ideal Model

An ideal model for R-2R ladders assumes ($\mathrm{V}^{*} = 0$) and identical resistors on the ladder [1] [3]. This gives:

$$\mathrm{I}^{\mathrm{R2R}} = \frac{\mathrm{V}^{\mathrm{in}}}{\mathtt{R}^{\mathtt{R}}} \mathsf{S} \tag{4.3}$$

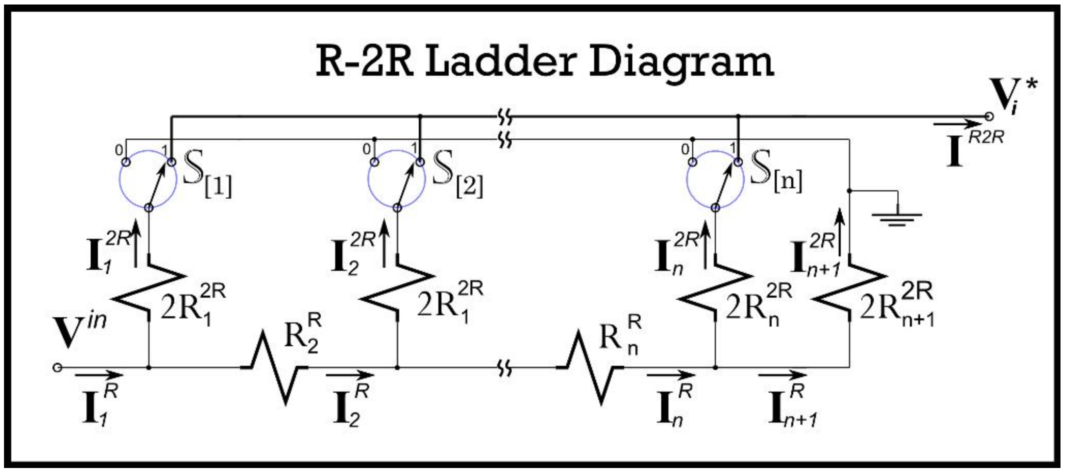


Figure 4.2: Representation of an R-2R Ladder using $\mathsf{S}_{[i]}$ to denote ideal switches. All output current, $\mathrm{I}^{R2R}$, is routed to the inverting terminal voltage, $\mathrm{V}^*$.

$$\mathsf{S} = \sum_{k=1}^{\mathtt{N^S}} \frac{\underline{\mathsf{S}}_{[k]}}{2^k} \tag{4.4}$$

### 4.3.2 Detailed Model

A more realistic R-2R ladder model considers each resistor value and inverting terminal potential, relating voltage drops across every current path:

$$\begin{bmatrix} 2\mathtt{R}_1^{\mathtt{2R}} & -2\mathtt{R}_1^{\mathtt{2R}} & 0 & 0 & \cdots & 0 \\ 0 & \mathtt{R}_2^{\mathtt{R}}+2\mathtt{R}_2^{\mathtt{2R}} & -2\mathtt{R}_2^{\mathtt{2R}} & 0 & \ddots & \vdots \\ 0 & \mathtt{R}_2^{\mathtt{R}} & \mathtt{R}_3^{\mathtt{R}}+2\mathtt{R}_3^{\mathtt{2R}} & \ddots & \ddots & 0 \\ 0 & \mathtt{R}_2^{\mathtt{R}} & \mathtt{R}_3^{\mathtt{R}} & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \mathtt{R}_{\mathtt{N^S}}^{\mathtt{R}}+2\mathtt{R}_{\mathtt{N^S}}^{\mathtt{2R}} & -2\mathtt{R}_{\mathtt{N^S}}^{\mathtt{2R}} \\ 0 & \mathtt{R}_2^{\mathtt{R}} & \mathtt{R}_3^{\mathtt{R}} & \cdots & \mathtt{R}_{\mathtt{N^S}}^{\mathtt{R}} & 2\mathtt{R}_{\mathtt{N^S}+1}^{\mathtt{2R}} \end{bmatrix} \begin{bmatrix} \mathrm{I}_1^{\mathrm{R}} \\ \mathrm{I}_2^{\mathrm{R}} \\ \vdots \\ \vdots \\ \mathrm{I}^{\mathrm{R}}{}_{\mathtt{N^S}} \\ \mathrm{I}^{\mathrm{R}}{}_{\mathtt{N^S}+1} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \\ 1 \end{bmatrix} \mathrm{V}^{\mathrm{in}} - \begin{bmatrix} \underline{\mathsf{S}}_{[1]} \\ \underline{\mathsf{S}}_{[2]} \\ \vdots \\ \vdots \\ \underline{\mathsf{S}}_{[\mathtt{N^S}]} \\ 0 \end{bmatrix} \mathrm{V}^*$$

$$\mathbf{\Gamma}\, \mathrm{I}^{\mathrm{R}} = \vec{1}\, \mathrm{V}^{in} - \underline{\mathsf{S}}\, \mathrm{V}^* \tag{4.5}$$

where,

$\mathbf{\Gamma}$ - Matrix representing the multiple resistors

$\underline{\mathsf{S}} = [\ \underline{\mathsf{S}}_{[1]},\ \dots\ ,\underline{\mathsf{S}}_{[N^S]},\ 0]^T$ - R-2R Ladder Switch Vector

$\vec{1} = [\ 1,\ 1\ ,\ \dots\ ,1\ ]^T$ - Column vector of ones

For the current through each 2R resister:

$$\mathrm{I}^{2\mathrm{R}} = [\,\mathbf{C}\,]\ \mathrm{I}^{\mathrm{R}} = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & -1 \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{I}^{\mathrm{R}}_{1} \\ \mathrm{I}^{\mathrm{R}}_{2} \\ \vdots \\ \mathrm{I}^{\mathrm{R}}_{\mathtt{N^S}} \\ \mathrm{I}^{\mathrm{R}}_{\mathtt{N^S}+1} \end{bmatrix}$$

The total current routed to the op-amp inverting terminal:

$$\mathrm{I}^{\mathrm{R2R}} = \sum_{k=1}^{\mathtt{N^S}} \mathrm{I}^{2\mathrm{R}}_{\mathrm{k}}\ \underline{\mathsf{S}}_{[k]} = \mathrm{I}^{2R} \cdot \underline{\mathsf{S}}$$

Combining the detailed model for an R-2R ladder:

$$\mathrm{I}^{\mathrm{R2R}} = \left[\mathbf{C}\,\mathbf{\Gamma}^{-1}\,(\vec{1}\ \mathrm{V}^{\mathrm{in}} - \underline{\mathsf{S}}\ \mathrm{V}^{*})\right] \cdot \underline{\mathsf{S}} \qquad (4.6)$$

# Part II

# Hardware Components

# Chapter 5

# Hybrid Integrator: HxI

## 5.1 Hybrid Integrator Layout

The Hybrid Integrator (**HxI**) consists of an op-amp integrator circuit, a comparator circuit, reset circuit, reset timing circuit, and output buffer circuit (see **Figure 5.1**, for similar design components: [2] [6]). The reset circuit avoids normal op-amp saturation by resetting the before saturation. The HxI's output analog signal represents a mixed signal's analog bit. Mixed signals combine digital and analog frameworks; the analog bit continuously represents discrete intervals in it's digital framework[1] [3] and is described below.

$$\lambda = -\frac{\mathrm{V}^{\mathrm{a}}}{\mathtt{V}^{\mathtt{ref}}} = \frac{\mathrm{V}^{\mathrm{x+}}}{\mathtt{V}^{\mathtt{ref}}} \tag{5.1}$$

## 5.2 Op-amp Integrator Circuit

Dynamics of the op-amp integrator circuit shown in **Figure 5.2**, can be modeled as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \mathrm{V}^{\mathrm{a}} \\ \mathrm{Q}^{\mathrm{fb}} \end{bmatrix} = \begin{bmatrix} -\omega^{1}(1+\frac{1}{\mathtt{A}^{\circ}}) & -\omega^{1}/\mathtt{C}^{\mathtt{fb}} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{V}^{\mathrm{a}} \\ \mathrm{Q}^{\mathrm{fb}} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathrm{I}^{*} \tag{5.2}$$

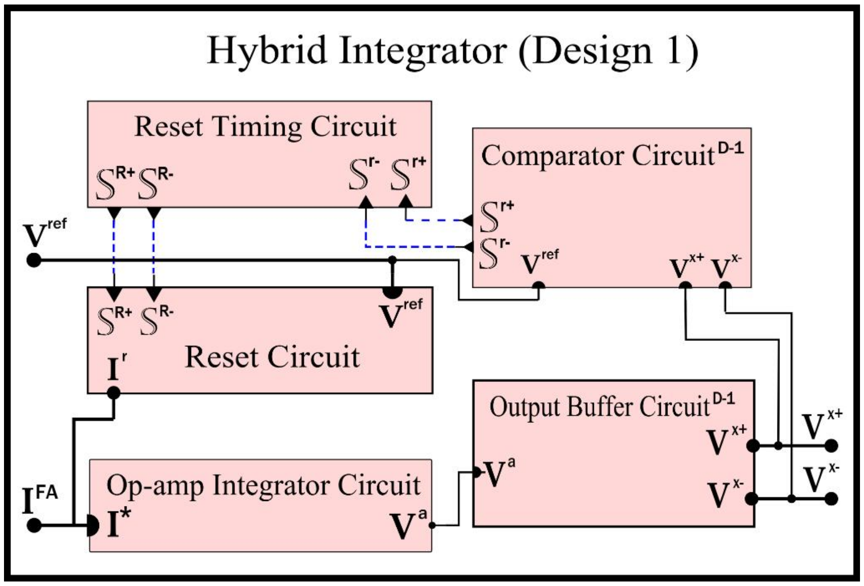


Figure 5.1: Layout of the Hybrid Integrator.

$$\mathrm{V}^{*} = \mathrm{V}^{\mathrm{a}} + \frac{\mathrm{Q}^{\mathrm{fb}}}{\mathtt{C}^{\mathtt{fb}}} \tag{5.3}$$

where,

$\mathrm{V}^{\mathrm{a}}$ - Integrator Voltage (output voltage)
$\mathrm{Q}^{\mathrm{fb}}$ - Feedback Capacitor Charge
$\mathrm{I}^{*}$ - Input Current to Op-amp Integrator

## 5.3 Comparator Circuit

The comparator circuit in **Figure 5.3** prevents op-amp saturation with reset signals described by:

$$\mathrm{V}^{\mathrm{x}+} \geq \mathrm{V}^{\mathrm{ref}} \quad \Rightarrow \quad \mathbb{S}^{r+} = 1$$

$$\mathrm{V}^{\mathrm{x}-} \geq \mathrm{V}^{\mathrm{ref}} \quad \Rightarrow \quad \mathbb{S}^{r-} = 1$$

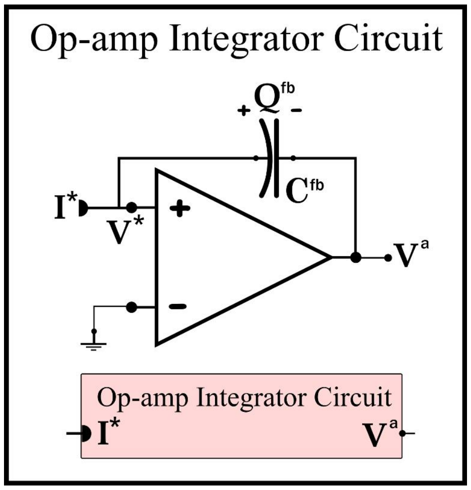


Figure 5.2: Typical Op-amp Integrator Circuit.

## 5.4 Reset Circuit

The reset circuit in **Figure 5.4** removes charge from the op-Amp integrators feedback capacitor:

$$\Delta \mathrm{Q}^{\mathrm{fb}} = \mathtt{C}^{\mathtt{fb}}\ \mathtt{V}^{\mathtt{ref}}$$

To fully charge and discharge, the reset capacitor input must be applied for a duration of time, provided by the reset timing circuit. For the various reset states, all charge electrodynamics can be described by:

$$\begin{gathered} \textbf{Negative Reset State:}\quad \underline{\mathsf{S}}^{R-} = 1 \\ \mathrm{I}^{\mathrm{r}} = -\frac{\mathrm{dQ}^{\mathrm{r}}}{\mathrm{d}t} \\ \frac{\mathrm{dQ}^{\mathrm{r}}}{\mathrm{d}t} = \frac{-1}{\mathtt{R}^{\mathtt{r}}\mathtt{C}^{\mathtt{r}}}\mathrm{Q}^{\mathrm{r}} - \frac{1}{\mathtt{R}^{\mathtt{r}}}\mathrm{V}^{*} \end{gathered} \tag{5.4}$$

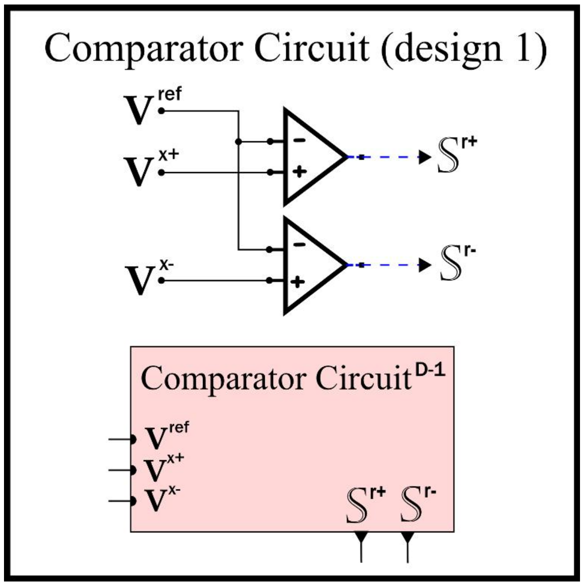


Figure 5.3: Schematic of comparator circuit.

$$\textbf{Positive Reset State:} \quad \underline{\mathsf{S}}^{R+} = 1$$
$$\mathrm{I^r} = +\frac{\mathrm{dQ^r}}{\mathrm{d}t}$$
$$\frac{\mathrm{dQ^r}}{\mathrm{d}t} = \frac{-1}{\mathtt{R^r C^r}}\mathrm{Q^r} + \frac{1}{\mathtt{R^r}}\mathrm{V}^* \tag{5.5}$$

$$\textbf{Normal State:} \quad \underline{\mathsf{S}}^{R-} = \underline{\mathsf{S}}^{R+} = 0$$
$$\mathrm{I^r} = 0 \tag{5.6}$$
$$\frac{\mathrm{dQ^r}}{\mathrm{d}t} = \frac{-1}{\mathtt{R^r C^r}}\mathrm{Q^r} + \frac{1}{\mathtt{R^r}}\mathtt{V^{ref}}$$

where,

$\mathrm{I}^r$ - Reset Current provided to the Integrating Op-amp
$\mathrm{Q}^r$ - Reset Capacitor Charge
$\mathrm{V}^*$ - Inverting terminal potential
$\mathtt{R}^r$ - Resistance associated with Reset Circuit

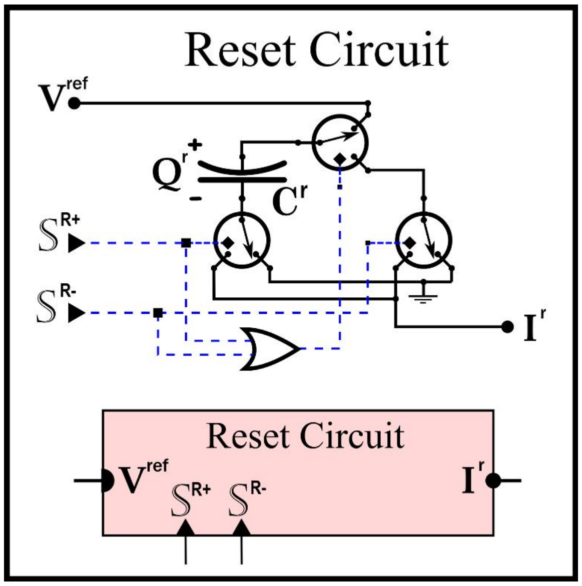


Figure 5.4: Schematic of reset circuit.

## 5.5 Reset Timing Circuit

The reset timing circuit is in **Figure 5.5**. A more detailed model would specify two output voltages rather than the digital signals $\underline{\mathtt{S}}^{\mathsf{R+}}$ and $\underline{\mathtt{S}}^{\mathsf{R+}}$; a requirement to precisely model the behavior of most switches (such as FET). This modeling assumes ideal switches with two states (open/closed, high/low, ect.); the relationship between a digital signal and it's analog value has been idealized as:

**Unmodified Signal Values:**

$$\underline{\mathsf{S}} = 0 \;\rightarrow\; \mathrm{S} = \mathtt{S}^{\mathtt{L}}$$
$$\underline{\mathsf{S}} = 1 \;\rightarrow\; \mathrm{S} = \mathtt{S}^{\mathtt{H}}$$

**Digital Ranges when Modified:**

$$\mathtt{S}^{L} \leq \mathrm{S} \leq \mathtt{S}^{LH} \;\rightarrow\; \underline{\mathsf{S}} = 0$$
$$\mathtt{S}^{LH} < \mathrm{S} \leq \mathtt{S}^{H} \;\rightarrow\; \underline{\mathsf{S}} = 1$$

where,

$\underline{\mathsf{S}}$ - Example Digital Signal

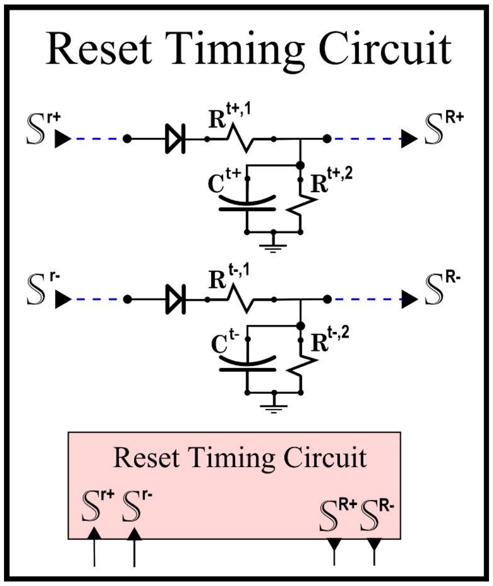


Figure 5.5: Schematic of reset timing circuit.

S - Voltage used to represent $\underline{\mathsf{S}}$
$\mathtt{S}^{\mathtt{L}}$ - Minimum voltage of S
$\mathtt{S}^{\mathtt{H}}$ - Maximum voltage of S
$\mathtt{S}^{\mathtt{LH}}$ - Digital Transition voltage

The reset timing circuit extends the duration of reset signaling. First, an input pulse charges a timing capacitor. Upon the input pulse vanishing, the timing capacitor's accumulated charge continues to provide output signaling; this lasts for a duration controlled by the timing capacitor's RC time constant. The time duration must be sufficient for the reset circuit capacitor to fully discharge:

$$[\ \underline{\mathsf{S}}^{r+} \rightarrow 1\ ]\Big|_{t^r=t} \quad \Rightarrow \quad [\ \underline{\mathsf{S}}^{R+} = 1\ ]\Big|_{t^r\ \leq\ t\ \leq\ t^r + \Delta\mathtt{t}^{\mathtt{r}}}$$

$$[\ \underline{\mathsf{S}}^{r-} \rightarrow 1\ ]\Big|_{t^r=t} \quad \Rightarrow \quad [\ \underline{\mathsf{S}}^{R-} = 1\ ]\Big|_{t^r\ \leq\ t\ \leq\ t^r + \Delta\mathtt{t}^{\mathtt{r}}}$$

The expression's left side specifies when the input reset signal vanishes. The right side gives the circuit's output signal. By default, every digital signal assumes a low state, unless specified otherwise.

$t$ - Time

$t^r$ - Instant of transition from $(\underline{\mathsf{S}}^r = 1)$ to $(\underline{\mathsf{S}}^r = 0)$

$\Delta\mathtt{t^r} \approx 6\,\mathtt{R^r C^r}$ - Signal Duration of Reset Timing Circuit

A small timing capacitor ($\mathtt{C^t}$) is desirable. Additionally, $\mathtt{R^{t,1}}$ must be small enough for the comparator circuit's short-lived signal to charge the timing capacitor a considerable amount. Lastly, the circuit's time constant ($\mathtt{R^{t,2}C^t}$) must be set to attain acceptable output signal durations ($\Delta\mathtt{t^r}$).

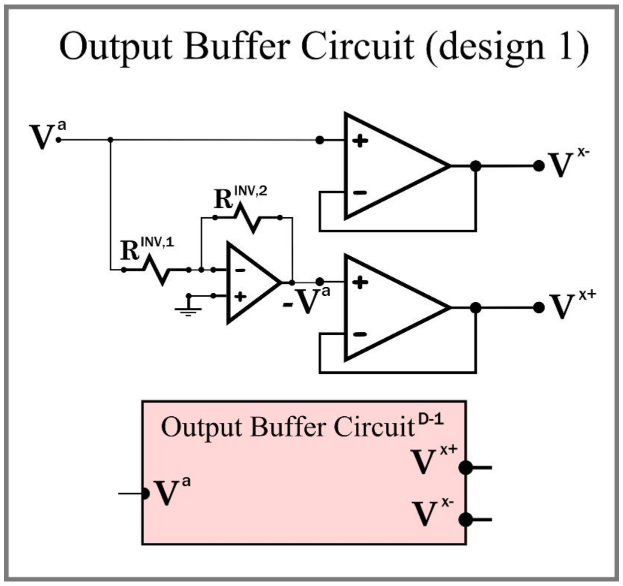


Figure 5.6: Schematic of the output buffer circuit.

## 5.6 Output Buffer Circuit

The output buffer circuit, **Figure 5.6**, isolates the op-amp integrator from external ciruits. The circuit drives the R-2R ladders. As a notable detail, $\mathrm{V^{x+}}$ has polarity opposite of $\mathrm{V^a}$. This makes $\mathrm{V^{x+}}$ increase when the Hybrid integrator has a positive input current. Additionally, the inverted signal $\mathrm{V^{x-}}$ is output by this circuit as required for function approximation.

# Chapter 6

# Hybrid Integrator Cell

A polygrator (**Px**) contains one Hx with components to control the flow of input current. For simulations, each **Px** solves a differential state equation; multiple **Px**'s combine to solve systems differential state equations [1] [3].

In conjunction with a **Hx**, the **Px** converts multiple input signals–other **Px** state signals–into a single input signal for integration; this signal conversion process–termed function approximation–represents a specific evaluation of the differential state equation being solved. Function approximation digitally controls multiple R-2R ladders based on a first order Taylor Series function approximation. In doing such, each R-2R ladder is equipped with accompanying digital components forming an **R2R**. Additionally, each **Px** contains at least one **R2R** has been termed a **DAC**; it represents the single non-derivative term of a Taylor Series approximation. Each **Px** includes a state register (floating point mixed signal format) maintaining it's state signals digital component.

## 6.1 Polygratror Semantics

## 6.2 Component Layout

**Figure 6.1** (design considerations: [1], [2], [3], [6]) depicts a possible layout for a Hybrid Integrator Cell (HxC). The design consists of an HxC control unit, a hybrid integrator, a state register, a DAC, R2R ladders, and an information Bus. The control unit manages signals and monitors events inside the HxC. The HxI

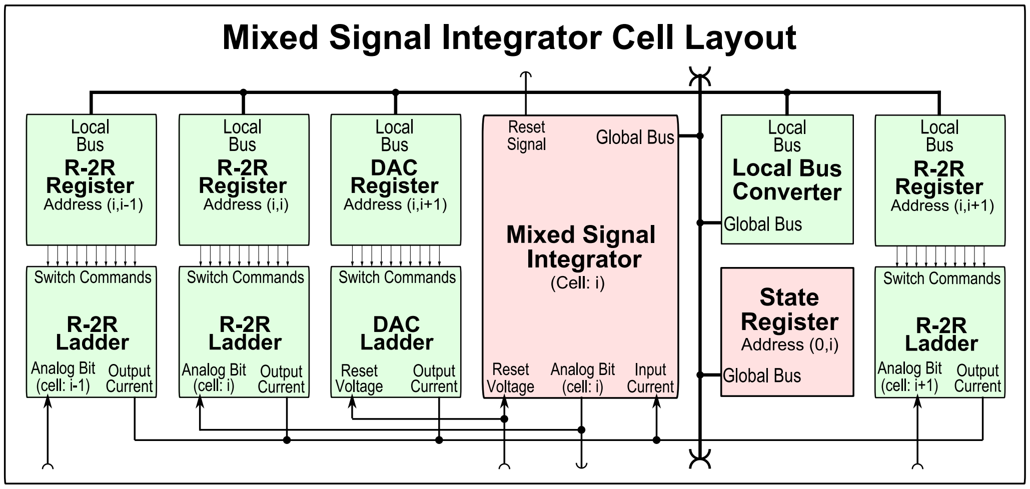


Figure 6.1: Hybrid Integrator Cell (HxC) layout.

continuously integrates the signals provided by the DAC and R2R units. The state register stores the HxC's floating point state variable, updating its value upon the arrival of each reset signal. The DAC and R2R's are used to approximate the ODE function being integrated. Lastly, the bus communicates information.

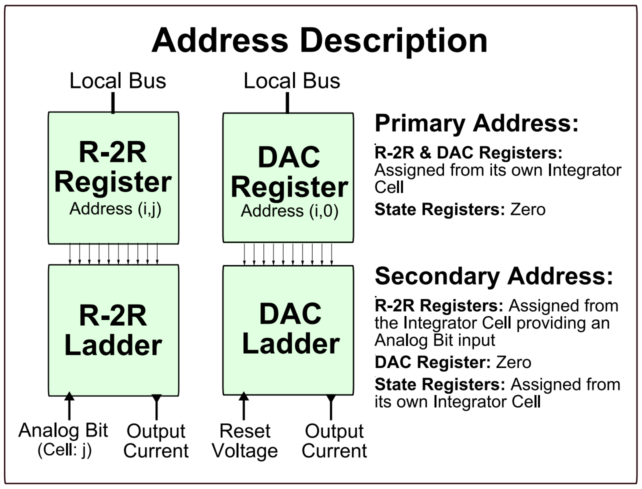


Figure 6.2: A methodology for addressing HxC components.

### 6.2.1 Hybrid Integrator Cell Addressing

A method for addressing the components in each HxC is given in **Figure A.1**. To communicate with a specific Hybrid Integrator Cell, each integrator cell is given an address.

## 6.3 Control Unit

The HxC control unit manages HxC operations. For events inside the HxC impacting external components, the control unit provides notification. As an example, the control unit is responsible for both acknowledging whenever the Hybrid Integrator analog bit resets and routing this information to other components.

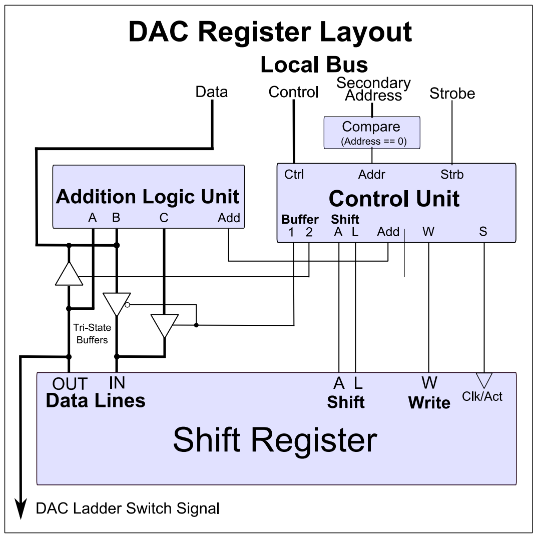


Figure 6.3: Possible DAC Register Layout.

## 6.4 DAC

**Figure 6.3** is a possible layout for each DAC. The optional addition logic unit allows the DAC to update upon an analog bit reset; if absent, this update must be performed through digital processing. For digital communication, the DAC contains a data bus, control bus, address bus, and strobe. The DAC contains an R2R ladder driven by the system reference voltage, $V^{ref}$.

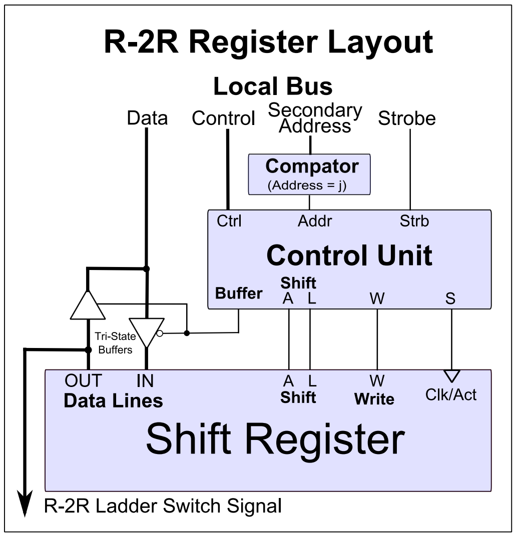


Figure 6.4: Possible R-2R Register Layout.

## 6.5 R2R

A tentative layout for each R2R is shown in **Figure 6.4**. Similar to the DAC, it contains a data bus, control bus, address bus, and strobe. For function approximation, digital communication is necessary. Contents are frequently updated when approximating functions. Each R2R unit is driven by an analog bit voltage $V^x$, through a buffer.

The DAC and multiple R2R's control input current to the Hybrid Integrator. The function approximation, based on a first order Taylor Series approximation, utilizes the DAC and multiple R2R's to represent terms. A DAC driven by system reference voltage $V^{ref}$, produces the $\mathcal{F}(\mathbf{x_0})$ term in a Taylor Series. The R2R units synthesize the linear terms.

# Chapter 7

# Hybrid Integrator Cell Array: HxA

A Hybrid Integrator Cell Array (**HxA**) is an inline arrangement of hybrid integrator cells, with each HxC's analog bit signal routed to neighboring cells. A bus networks the HxA with a digital processor, which monitors and controls HxA's operation.

## 7.1 Component Layout

A possible layout for the Hybrid Integrator Cell Array (**HxA**) is shown in **Figure 7.1** (design considerations: [1], [3], [6]). The HxA's design is expandable to include more Hybrid Integrator Cells. Additionally, each HxC can be expanded to include more R2Rs for function approximation.

## 7.2 HxA Modeling

### 7.2.1 Op-Amp Integrators

The array ($i = 1, 2, .., \mathtt{N}$) of integrating op-amp behaviors can be described by:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \mathrm{V_i^a} \\ \mathrm{Q_i^{fb}} \end{bmatrix} = \begin{bmatrix} -\omega_i^1(1+\frac{1}{\mathtt{A_i^o}}) & -\omega_i^1/\mathtt{C_i^{fb}} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \mathrm{V_i^a} \\ \mathrm{Q_i^{fb}} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathrm{I_i} \quad (7.1)$$

### 7.2.2 Integrator Currents

Each integrator receives input current from a Reset Circuit ($\mathrm{I}_{\mathrm{i}}^{\mathrm{r}}$), a DAC ($\mathrm{I}_{\mathrm{i,0}}^{\mathrm{R2R}}$), and multiple R2R units:

$$\mathrm{I}_{\mathrm{i}} = \mathrm{I}_{\mathrm{i}}^{\mathrm{r}} + \sum_{j=0}^{N} \mathrm{I}_{\mathrm{i,j}}^{\mathrm{R2R}} \tag{7.2}$$

#### Reset Circuits

The reset circuit model (**Equation 5.4, 5.5, & 5.6** ) is extended as:

$$\textbf{Negative Reset State:} \quad \underline{\mathsf{S}}_i^{R-} = 1$$
$$\mathrm{I}_{\mathrm{i}}^{\mathrm{r}} = -\frac{\mathrm{dQ}_{\mathrm{i}}^{\mathrm{r}}}{\mathrm{d}t}$$
$$\frac{\mathrm{dQ}_{\mathrm{i}}^{\mathrm{r}}}{\mathrm{d}t} = \frac{-1}{\mathtt{R}_{\mathtt{i}}^{\mathtt{r}}\mathtt{C}_{\mathtt{i}}^{\mathtt{r}}}\mathrm{Q}_{\mathrm{i}}^{\mathrm{r}} - \frac{1}{\mathtt{R}_{\mathtt{i}}^{\mathtt{r}}}\mathrm{V}_{\mathrm{i}}^{*}$$

$$\textbf{Positive Reset State:} \quad \underline{\mathsf{S}}_i^{R+} = 1$$
$$\mathrm{I}_{\mathrm{i}}^{\mathrm{r}} = +\frac{\mathrm{dQ}_{\mathrm{i}}^{\mathrm{r}}}{\mathrm{d}t}$$
$$\frac{\mathrm{dQ}_{\mathrm{i}}^{\mathrm{r}}}{\mathrm{d}t} = \frac{-1}{\mathtt{R}_{\mathtt{i}}^{\mathtt{r}}\mathtt{C}_{\mathtt{i}}^{\mathtt{r}}}\mathrm{Q}_{\mathrm{i}}^{\mathrm{r}} + \frac{1}{\mathtt{R}_{\mathtt{i}}^{\mathtt{r}}}\mathrm{V}_{\mathrm{i}}^{*}$$

$$\textbf{Normal State:} \quad \underline{\mathsf{S}}_i^{R-} = \underline{\mathsf{S}}_i^{R+} = 0$$
$$\mathrm{I}_{\mathrm{i}}^{\mathrm{r}} = 0$$
$$\frac{\mathrm{dQ}_{\mathrm{i}}^{\mathrm{r}}}{\mathrm{d}t} = \frac{-1}{\mathtt{R}_{\mathtt{i}}^{\mathtt{r}}\mathtt{C}_{\mathtt{i}}^{\mathtt{r}}}\mathrm{Q}_{\mathrm{i}}^{\mathrm{r}} + \frac{1}{\mathtt{R}_{\mathtt{i}}^{\mathtt{r}}}\mathtt{V}_{\mathtt{i}}^{\mathtt{ref}}$$

### 7.2.3 DAC/R2R Current

Using the detailed R-2R Ladder model derived in **Equation 4.6**, the current provided by each DAC ($j = 0$) and R2R can be described by:

$$\mathrm{I}_{\mathrm{i,j}}^{\mathrm{R2R}} = \left[\mathbf{C}\,\mathbf{\Gamma}_{i,j}^{-1}\,(\vec{1}\,\mathrm{V}_{\mathrm{j}}^{\mathrm{a}} - \underline{\mathsf{S}}_{i,j}\,\mathrm{V}_i^{*})\right] \cdot \underline{\mathsf{S}}_{i,j} \tag{7.3}$$

A simplified model uses the ideal ladder model of **Equation 4.3** that doesn't consider the effects of a virtual ground or inaccuracies in each R-2R ladder's resistors.

### 7.2.4 Inverting Terminal Voltage

The Integrator Cell Array circuit requires determining each inverting terminal voltage, $V_i^*$, as derived in **Equation 5.3**:

$$V_{\mathrm{i}}^* = V_{\mathrm{i}}^{\mathrm{a}} + \frac{Q_{\mathrm{i}}^{\mathrm{fb}}}{\mathtt{C}_{\mathtt{i}}^{\mathtt{fb}}} \tag{7.4}$$

## 7.3 Solving the Model

### Step 1: Initial Conditions

For a HxA with "n" integrators, solving this model requires an initial knowledge of the following "3n" dynamic states:

$V_{\mathrm{i}}^{\mathrm{a}}$ - Analog Bit Voltage for Integrator Cell 'i'
$Q_{\mathrm{i}}^{\mathrm{fb}}$ - Feedback Capacitor Charge for Integrator Cell 'i'
$Q_{\mathrm{i}}^{\mathrm{r}}$ - Reset Capacitor Charge for Integrator Cell 'i'

Additionally, the following digital states for each reset signal are initially known:

$\underline{\mathsf{S}}_i^{R-}$ - Positive Reset State
$\underline{\mathsf{S}}_i^{R+}$ - Negative Reset State

### Step 2: Solve for the Inverting Terminal Voltage

Each inverting terminal voltage ($V_{\mathrm{i}}^*$) can be solved from **Equation 7.4**.

### Step 3: Solve all Currents

After Step 2, the following terms can be determined in the order provided:

$\frac{\mathrm{d}}{\mathrm{d}t}Q_{\mathrm{i}}^{\mathrm{r}}$ - Solve with the Reset Circuit Equations
$I_{\mathrm{i}}^{\mathrm{r}}$ - Solve with the Reset Circuit Equations
$I_{\mathrm{i,j}}^{\mathrm{R2R}}$ - Solve with **Equation 7.3**
$I_{\mathrm{i}}$ - Solve with **Equation 7.2**

## Step 4: Solve remaining Electrodynamics

The derivatives of each state variable stated in Step 1, can now be calculated.

$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{V}_{\mathrm{i}}^{\mathrm{a}}$ - Solve with **Equation 7.1**
$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{Q}_{\mathrm{i}}^{\mathrm{fb}}$ - Use **Equation 7.1**

## Step 5: Integrate Forward in time

The system can be integrated forward in time continually until a digital change occurs, such as an analog bit reset. At this instant, the updates are made, followed by further integration.

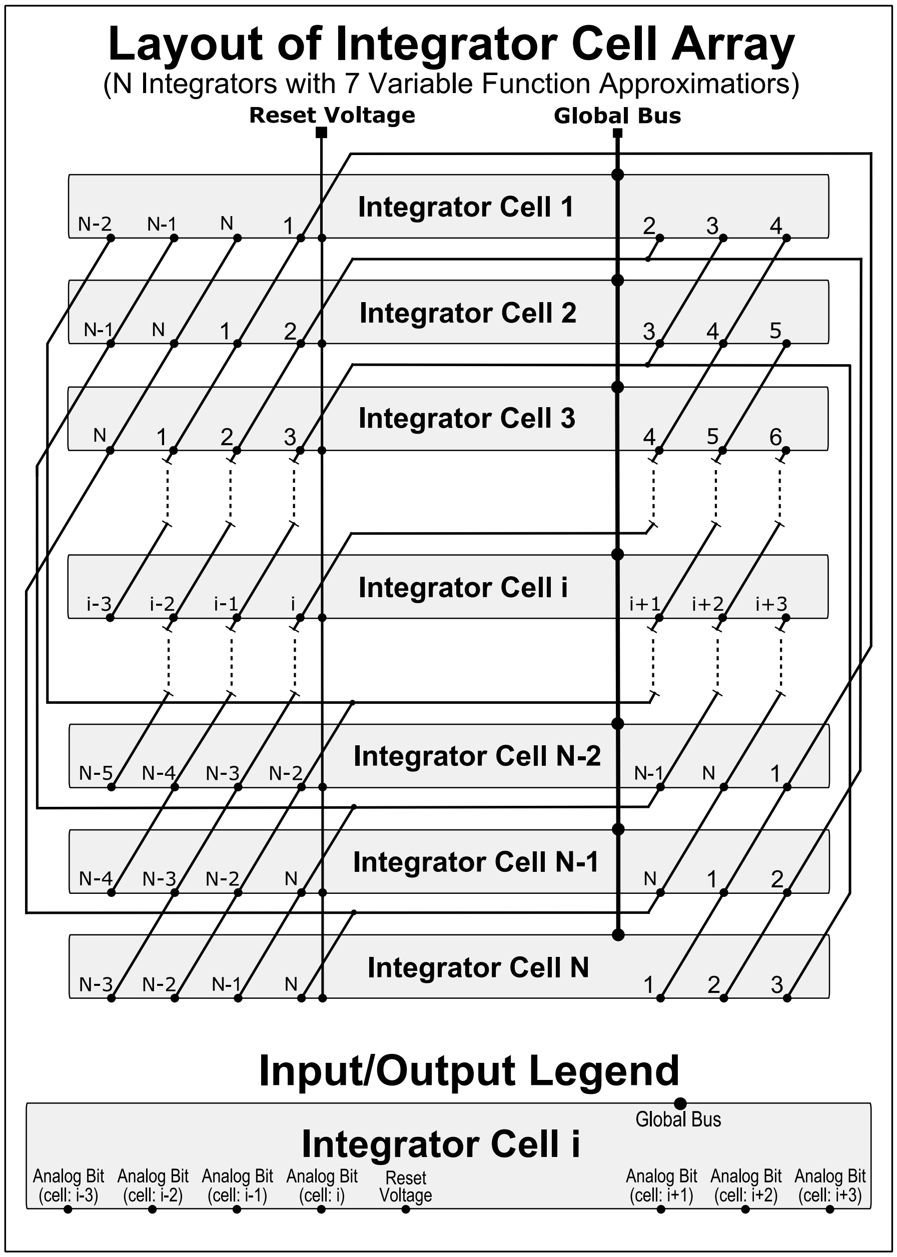


Figure 7.1: Possible Layout for HxA.

# Part III

# Processing Mixed Signals

# Chapter 8

# ODE Representation

Ordinary Differential Equations (ODE's) are commonly used in modeling behavior. Under most circumstances, an ODE can be transformed into a system of first order ordinary differential equations to be solved. Mixed signal can solve systems of first order ODE's represented in a time invariant form. In the next section, a method is provided for transforming time dependent systems of ODE's into a time invariant form required by the Mixed Signal Processing. Lastly, a method is discussed to indirectly modify when each ODE state variable becomes denormalized.

## 8.1 Time Invariant ODE's

The system of '$n$' time invariant first order Ordinary Differential Equations(ODE) to be solved is:

$$\begin{gathered} \tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \boldsymbol{\mathcal{F}}(\mathbf{x}) \\ \mathbf{x}_0 = \mathbf{x}\big|_{t=t_0} \end{gathered} \tag{8.1}$$

where,

$\mathbf{x} = [\ \mathrm{x}_1,\ \mathrm{x}_2,\ \cdots,\ \mathrm{x}_n\ ]^T$ - ODE State Vector

$\boldsymbol{\mathcal{F}}(\mathbf{x}) = [\ \mathcal{F}_1(\mathbf{x}),\ \mathcal{F}_2(\mathbf{x}),\ \cdots,\ \mathcal{F}_n(\mathbf{x})\ ]^T$ - ODE Vector Function

$\mathbf{x}_0 = [\ \mathrm{x}_{1,0},\ \mathrm{x}_{2,0},\ \cdots,\ \mathrm{x}_{n,0}\ ]^T$ - Initial Condition Vector

## 8.2 Avoiding Time Dependencies

The scaling process derived for mixed signal solution of ODE functions was assumed to be time invariant. This section establishes a a method for transforming a time variant system into an equivalent time invariant system of ODE's.

### 8.2.1 ODE Layout

A system of first order time dependent ODE's can be represented by:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z} = \boldsymbol{\mathcal{G}}(\mathbf{z}, t)$$

$$\mathbf{z}_0 = \mathbf{z}\big|_{t=t_0} \tag{8.2}$$

where,

$\mathbf{z} = [\ \mathrm{z}_1,\ \cdots,\ \mathrm{z}_r\ ]^T$ - ODE State Vector

$\boldsymbol{\mathcal{G}}(\mathbf{z}, t) = [\ \mathcal{G}_1(\mathbf{z}, t),\ \cdots,\ \mathcal{G}_r(\mathbf{z}, t)\ ]^T$ - ODE Vector Function

Although the time dependency cannot directly be removed from each function $\mathcal{G}$ above, the system of ODE's can be transformed into a system identical to the one above, without the time dependency.

### 8.2.2 Time-State Substitution

A system with identical representation to **Equation 8.2** must contain identical states; however maintaining an identical representation doesn't preclude including additional state. An additional state, $\mathrm{x}_{r+1} = t$, represents time as a state variable.

$$[\ \mathrm{x}_1,\ \cdots,\mathrm{x}_r,\ \mathrm{x}_{r+1}\ ]^T = [\ \mathrm{z}_1,\ \cdots,\ \mathrm{z}_r,\ t\ ]^T$$

Including the initial condition vector:

$$[\ \mathrm{x}_{1,0},\ \cdots,\mathrm{x}_{r,0},\ \mathrm{x}_{r+1,0}\ ]^T = [\ \mathrm{z}_{1,0},\ \cdots,\ \mathrm{z}_{r,0},\ t_0\ ]^T$$

### 8.2.3 Modified ODE Vector Function

The additional state variable represents time. The additional state equation and function is simply assigned as:

$$\frac{\mathrm{d}\mathrm{x}_{r+1}}{\mathrm{d}t} = 1$$

After additional substitution, a new ODE function is

$$\begin{bmatrix} \mathcal{F}_1(\mathbf{x}) \\ \vdots \\ \mathcal{F}_r(\mathbf{x}) \\ \mathcal{F}_{r+1}(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathcal{G}_1(\mathbf{x}_{[1:r]}, \mathrm{x}_{r+1}) \\ \vdots \\ \mathcal{G}_r(\mathbf{x}_{[1:r]}, \mathrm{x}_{r+1}) \\ 1 \end{bmatrix} = \begin{bmatrix} \mathcal{G}_1(\mathbf{z}, t) \\ \vdots \\ \mathcal{G}_r(\mathbf{z}, t) \\ 1 \end{bmatrix} \tag{8.3}$$

where,

$\mathbf{x}_{[1:r]} = [\mathrm{x}_1, \ \cdots \ , \ \mathrm{x}_r]^T$

The new system is compatible with the scaling requirements and is indirectly time variant.

## 8.3 Modified Denormalization for ODE's

For systems of ODE's with solutions that pass through zero, simulation speed may be enhanced by modifying when each state variable is denormalized. Without single precision format, integration of a state variable from one to zero requires the following number of resets:

$$\begin{array}{cccccl} (\mathrm{x} = 2^{0}) & \rightarrow & (\mathrm{x} = 2^{-1}) & \Rightarrow & (2^{23-\mathsf{k}}) & \textit{Resets} \\ (\mathrm{x} = 2^{-1}) & \rightarrow & (\mathrm{x} = 2^{-126}) & \Rightarrow & 125 \times (2^{23-\mathsf{k}}) & \textit{Resets} \\ (\mathrm{x} = 2^{-126}) & \rightarrow & (\mathrm{x} = 0) & \Rightarrow & (2^{23-\mathsf{k}} - 1) & \textit{Resets} \end{array}$$

$$\Longrightarrow 127 \times (2^{23-\mathsf{k}}) \ \approx \ 2^{30-\mathsf{k}} \ \approx \ 2^{-\mathsf{k}} \times 10^{9}\,\textit{Total Resets}$$

Although the total number of resets can be dramatically reduced by the scaling index, further reduction may be desirable. This additional reduction can be achieved by modifying the denormalized floor from $2^{-126}$ to a reasonable value based on the system of ODE's being solved. This is achieved by linearly scaling the ODE's into a new system, operating under a different range of values.

### 8.3.1 ODE Layout

The general layout of a system of first order ODE's using modified denormalization is represented by:

$$\begin{gathered}\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z} = \boldsymbol{\mathcal{H}}(\mathbf{z}) \\ \mathbf{z}_0 = \mathbf{z}\big|_{t=t_0} \\ \mathtt{Z}^D\end{gathered} \tag{8.4}$$

where,

$\mathbf{z} = [\ \mathtt{z}_1,\ \cdots,\ \mathtt{z}_n\ ]^T$ - ODE State Vector

$\boldsymbol{\mathcal{H}}(\mathbf{z}) = [\ \mathcal{H}_1(\mathbf{z}),\ \cdots,\ \mathcal{H}_n(\mathbf{z})\ ]^T$ - ODE Vector Function

$\mathtt{Z}^D = [\mathtt{Z}_1^D, ..., \mathtt{Z}_n^D]^T$ - Denormalization Vector

The denormalized vector specifies the absolute values at which each state variable is denormalized.

### 8.3.2 State Variable Scaling

The denormalization creates a new ODE, having state variables linearly scaled to the systems state variables below:[1]

$$\mathtt{z}_i = \mathtt{C}_i\ \mathtt{x}_i \Rightarrow \mathtt{Z}_i^D = \mathtt{C}_i\ 2^{-126} \Rightarrow \mathtt{z}_i = \mathtt{Z}_i^D\ 2^{126}\ \mathtt{x}_i$$

Applying this relationship to the full state vector:

[1]Denormalization is assumed to begin as in the single precision floating point format (at $\mathtt{x} = 2^{-126}$)

$$\begin{bmatrix} \mathrm{z}_1 \\ \mathrm{z}_2 \\ \vdots \\ \mathrm{z}_n \end{bmatrix} = \begin{bmatrix} \mathtt{Z}_1^D\, 2^{126} & \mathrm{x}_1 \\ \mathtt{Z}_2^D\, 2^{126} & \mathrm{x}_2 \\ \vdots & \vdots \\ \mathtt{Z}_n^D\, 2^{126} & \mathrm{x}_n \end{bmatrix}$$

Evaluating the relationship between their derivatives:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{z}_i = \mathtt{Z}_i^D\, 2^{126}\, \frac{\mathrm{d}}{\mathrm{d}t}\mathrm{x}_i$$

### 8.3.3 Substituted Form

By substituting the relations for **z**, a new ODE is formed:

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \begin{bmatrix} 2^{-126}/\mathtt{Z}_1^D & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & 2^{-126}/\mathtt{Z}_n^D \end{bmatrix} \boldsymbol{\mathcal{H}} \left( \begin{bmatrix} \mathtt{Z}_1^D\, 2^{126} & \mathrm{x}_1 \\ \vdots & \vdots \\ \mathtt{Z}_n^D\, 2^{126} & \mathrm{x}_n \end{bmatrix} \right) = \boldsymbol{\mathcal{F}}(\mathbf{x})$$

$$\mathbf{x}_0 = \begin{bmatrix} 2^{-126}/\mathtt{Z}_1^D & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & 2^{-126}/\mathtt{Z}_n^D \end{bmatrix} \mathbf{z}_0$$

The system of ODE's specified in terms of **x** can now be solved, followed by transforming the solution back into **z**.

# Chapter 9

# Integration Scaling Development

The scaling process develops a procedure for digital processing to control HxA to solve a system of ODE's [1] [3] [6].

## 9.1 Problem Statement

Integrating the system of first order ODE's (**Equation 8.1**),

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \boldsymbol{\mathcal{F}}(\mathbf{x}) \qquad \text{with} \qquad \mathbf{x}_0 = \mathbf{x}|_{t=t_0}$$

renders:

$$\begin{gathered}
\mathrm{x}_1 = \mathrm{x}_{1,0} + \textstyle\int_{t_0}^{t} \mathcal{F}_1(\mathbf{x})\mathrm{d}t \\
\\
\mathrm{x}_2 = \mathrm{x}_{2,0} + \textstyle\int_{t_0}^{t} \mathcal{F}_2(\mathbf{x})\mathrm{d}t \\
\vdots \\
\vdots \\
\mathrm{x}_n = \mathrm{x}_{n,0} + \textstyle\int_{t_0}^{t} \mathcal{F}_n(\mathbf{x})\mathrm{d}t
\end{gathered} \tag{9.1}$$

where,

$t$ - Simulated time for ODE solution

$t_0$ - Initial Condition time for ODE

$\mathbf{x} = [\ \mathrm{x}_1,\ \cdots,\ \mathrm{x}_n\ ]^T$ - ODE State Vector

$\mathbf{x}_0 = [\ \mathrm{x}_{1,0},\ \cdots,\ \mathrm{x}_{n,0}\ ]^T$ - Initial Condition Vector

$\boldsymbol{\mathcal{F}}(\mathbf{x}) = [\ \mathcal{F}_1(\mathbf{x}),\ \cdots,\ \mathcal{F}_n(\mathbf{x})\ ]^T$ - ODE Vector Function

## 9.2 Function Approximation

### 9.2.1 Taylor Series

Taylor Series expand a function into a series about a point. Approximations for the single variable (order n) case, followed by the multi-variable (first order) case are presented below.

**Single-Variable Taylor Series**

$$\mathcal{T}(\mathrm{x}) = \mathcal{T}(\mathrm{x}_0) + \sum_{j=1}^{n} \frac{(\Delta \mathrm{x})^j}{(j+1)!} \frac{\mathrm{d}^j \mathcal{T}}{\mathrm{d}\mathrm{x}^j}(\mathrm{x}_0) + \mathcal{R}_n(\mathrm{x}_0, \Delta \mathrm{x})$$

**Multi-Variable Taylor Formula**

$$\mathcal{T}(\mathbf{x}) = \mathcal{T}(\mathbf{x}_0) + \sum_{j=1}^{n} \Delta \mathrm{x}_j \frac{\partial \mathcal{T}}{\partial \mathrm{x}_j}(\mathbf{x}_0) + \mathcal{T}_1(\mathbf{x}_0, \Delta \mathbf{x})$$

where,

$\mathbf{x}$ - Point for evaluating Taylor Series Approximation
$\mathcal{T}$ - Function being approximated
$\mathbf{x}_0$ - Taylor Series Expansion Point
$\Delta\mathbf{x} = [\Delta \mathrm{x}_1, \cdots, \Delta \mathrm{x}_n]^T = \mathbf{x} - \mathbf{x}_0$
$\mathcal{R}$ - Remainder for Taylor Series Approximation

**Approximating the ODE Functions**

Mixed signal processing will use a first order Taylor Series approximation for each ODE function as follows:

$$\begin{aligned}
\mathcal{F}_1(\mathbf{x}) &\approx \mathcal{F}_1(\mathbf{x}_0) + \textstyle\sum_{j=1}^{n} \Delta \mathrm{x}_j \, \frac{\partial \mathcal{F}_1}{\partial \mathrm{x}_j}(\mathbf{x}_0) \\
\mathcal{F}_2(\mathbf{x}) &\approx \mathcal{F}_2(\mathbf{x}_0) + \textstyle\sum_{j=1}^{n} \Delta \mathrm{x}_j \, \frac{\partial \mathcal{F}_2}{\partial \mathrm{x}_j}(\mathbf{x}_0) \\
&\vdots \\
\mathcal{F}_n(\mathbf{x}) &\approx \mathcal{F}_n(\mathbf{x}_0) + \textstyle\sum_{j=1}^{n} \Delta \mathrm{x}_j \, \frac{\partial \mathcal{F}_n}{\partial \mathrm{x}_j}(\mathbf{x}_0)
\end{aligned} \tag{9.2}$$

### 9.2.2 Integral Approximation

Substitution of **Equation 9.2** into each of the integrals of **Equation 9.1** (indexed to $i$) produces:

$$\Delta \mathrm{x}_i = \int_{t_0}^{t} \left( \mathcal{F}_i(\mathbf{x}_0) + \sum_{j=1}^{n} \Delta \mathrm{x}_j \, \frac{\partial \mathcal{F}_i}{\partial \mathrm{x}_j}(\mathbf{x}_0) \right) \mathrm{d}t \tag{9.3}$$

where index $i = 1, 2, .., n$ indexes the variables.

## 9.3 Symbolic Representation

### 9.3.1 Mixed Signals

Each state variable (indexed to $i$) represented by a mixed signal uses a floating point mixed signal format described by:

$$\lambda_i = -\mathrm{V}_i^a / \mathtt{V}^{\mathtt{ref}} = \mathrm{V}_i^\lambda / \mathtt{V}^{\mathtt{ref}}$$

$$\mathrm{x}_i = \mathrm{x}_{i,0} + \Delta \mathrm{x}_i = (\mathsf{b}_i \; \mathsf{m}_i + \lambda_i \, 2^{\mathsf{k}_i - \mathtt{M}}) \, 2^{\mathsf{e}_i}$$

$$\mathrm{x}_{i,0} = \; \mathsf{b}_i \; \mathsf{m}_i \; 2^{\mathsf{e}_i}$$

$$\Delta \mathrm{x}_i = \lambda_i \; 2^{\mathsf{k}_i - \mathtt{M} + \mathsf{e}_i}$$

### 9.3.2 Time Scaling

A time scaling process controls the rate at which the ODE's simulation time progresses relative to hardware time:

$$\frac{\mathrm{d}t}{\mathrm{d}T} = 2^{\tau}$$

where,

> $T$ - HxA hardware real time[1]
> $t$ - Simulated time of ODE solution
> $\tau$ - Time scaling factor

A positive value of $\tau$ indicates a simulation processed at a rate faster than real time.

### 9.3.3 Symbolic Form of Integral

Modifying the integral in **Equation 9.3** to include the mixed signal substitutions and time scaling yields:

$$\lambda_i = \int_{t_0}^{t} \left( \mathcal{F}_i(\mathbf{x}_0)\, 2^{\mathtt{M}+\tau-\mathsf{k}_i-\mathsf{e}_i} + \sum_{j=1}^{n} \frac{\partial \mathcal{F}_i}{\partial \mathbf{x}_j}(\mathbf{x}_0)\, \lambda_j\, 2^{\tau-\mathsf{k}_i-\mathsf{e}_i+\mathsf{k}_j+\mathsf{e}_j} \right) \mathrm{d}T \tag{9.4}$$

## 9.4 Ideal Hardware Dynamics

Inside each hybrid integrator cell, an idealized integrator model is created that evaluates each analog bit voltage as follows:

[1]Using "T" to represent HxA hardware real time applies when the scaling process is relevent

$$-\mathrm{V}_i^a = \mathrm{V}_i^\lambda = \frac{1}{\mathsf{C}_i^{fb}} \int_{t_0}^{t} \left( \mathrm{I}_{i,0}^{R2R} + \sum_{j=1}^{n} \mathrm{I}_{i,j}^{R2R} \right) \mathrm{d}T$$

where each analog bit reset is modeled as an instantaneous process, described by the following transition:

$$\begin{aligned} \mathrm{V}_i^a &= +\mathtt{V}^{\mathtt{ref}} & \overset{set}{\longrightarrow} \quad \mathrm{V}_i^a &= 0 \\ \mathrm{V}_i^a &= -\mathtt{V}^{\mathtt{ref}} & \overset{set}{\longrightarrow} \quad \mathrm{V}_i^a &= 0 \end{aligned}$$

The current supplied by each DAC and R2R is idealized by:

$$\mathrm{I}_{i,j}^{R2R} = \mathsf{b}_{i,j}^{S} \frac{\mathrm{V}_j^\lambda}{\mathtt{R}_{i,j}^{R2R}} \; \mathsf{S}_{i,j}$$

$$\mathsf{S}_{i,j} = \sum_{k=1}^{\mathtt{N}^{\mathtt{S}}} \frac{\underline{\mathsf{S}}_{i,j,[k]}}{2^k}$$

where,

$\mathsf{S}_{i,j}$ - Value of R-2R Ladder Mantissa ($\underline{\mathsf{S}}_{i,j}$ register)
$\mathsf{b}_{i,j}^{S}$ - Sign reference of R-2R Ladder Input ($\underline{\mathsf{B}}_{i,j}^{S}$ bit )
$\mathrm{V}_0^\lambda = \mathtt{V}^{\mathtt{ref}}$ - Voltage supplied to each DAC
$\mathrm{V}_j^\lambda = \lambda_j \; \mathtt{V}^{\mathtt{ref}}$ - Analog Bit Voltage of Cell j
$\mathtt{R}_{i,j}^{R2R}$ - Nominal Resistance in R-2R ladder (indexed i,j)

Combining the equations above gives the following relation:

$$\lambda_i = \frac{1}{\mathtt{R}_{i,j}^{R2R} \; \mathsf{C}_i^{fb}} \int_{t_0}^{t} \left( \mathsf{b}_{i,0}^{S} \; \mathsf{S}_{i,0} + \sum_{j=1}^{n} \mathsf{b}_{i,j}^{S} \; \mathsf{S}_{i,j} \; \lambda_j \right) \mathrm{d}T \qquad (9.5)$$

## 9.5 Analog Bit Reset Updating

Each analog bit reset ($\lambda_j = \pm 1 \rightarrow \lambda_j = 0$) removes a term ($\mathsf{b}_{i,j}^{S} \; \mathsf{S}_{i,j}\lambda_j$) from the sum in **Equation 9.5**. The value of this removed term must be compensated for in

**Equation 9.5**. This is accomplished by adding the removed abount to each DAC's representation, $\mathsf{b}^{S}_{i,0}\,\mathsf{S}_{i,0}$:

### 9.5.1 Positive Resets

$$\begin{array}{l} \textit{Condition:} \\ \mathrm{V}^{\lambda}_{k} = +\mathtt{V}^{ref} \quad \overset{set}{\rightarrow} \quad \mathrm{V}^{\lambda}_{k} = 0 \\ \textit{DAC Compensation:} \\ \left[\mathsf{b}^{S}_{i,0}\,\mathsf{S}_{i,0}\right]_{new} \; = \; \left[\mathsf{b}^{S}_{i,0}\,\mathsf{S}_{i,0} + \mathsf{b}^{S}_{i,k}\,\mathsf{S}_{i,k}\right]_{old} \quad \text{for } i = 1, 2, .., n \end{array}$$

### 9.5.2 Negative Resets

$$\begin{array}{l} \textit{Condition:} \\ \mathrm{V}^{\lambda}_{k} = -\mathtt{V}^{ref} \quad \overset{set}{\rightarrow} \quad \mathrm{V}^{\lambda}_{k} = 0 \\ \textit{Controlled Compensation:} \\ \left[\mathsf{b}^{S}_{i,0}\,\mathsf{S}_{i,0}\right]_{new} \; = \; \left[\mathsf{b}^{S}_{i,0}\,\mathsf{S}_{i,0} - \mathsf{b}^{S}_{i,k}\,\mathsf{S}_{i,k}\right]_{old} \quad \text{for } i = 1, 2, .., n \end{array}$$

## 9.6 Function Approximation Constraint

Comparing the integrands in **Equation 9.4** and **Equation 9.5** to realize the first order multi-variable Taylor Series approximation in hardware:

$$\mathsf{S}_{i,j} = \begin{cases} \mathsf{b}^{S}_{i,j}\,\mathtt{R}^{R2R}_{i,j}\,\mathtt{C}^{fb}_{i}\,\mathcal{F}_i(\mathbf{x}_0)\,2^{\mathtt{M}+\tau-\mathsf{k}_i-\mathsf{e}_i} & j = 0 \;\; (\text{DAC}) \\ \\ \mathsf{b}^{S}_{i,j}\,\mathtt{R}^{R2R}_{i,j}\,\mathtt{C}^{fb}_{i}\,\frac{\partial \mathcal{F}_i}{\partial \mathtt{x}_j}(\mathbf{x}_0)\,2^{\tau-\mathsf{k}_i-\mathsf{e}_i+\mathsf{k}_j+\mathsf{e}_j} & j \neq 0 \;\; (\text{R2R}) \end{cases} \tag{9.6}$$

The constraints suggested by **Equation 9.6** will be realized by scaling.

# Chapter 10

# Integration Scaling Control

This chapter develops methods to control the scaling process during mixed signal processing [1] [3] [6]. Scaling keeps operations within the range of operation for each component. The function approximation scaling constraint (**Equation 9.6**) is:

$$\mathsf{S}_{i,j} = \begin{cases} \mathsf{b}^{S}_{i,j}\ \mathtt{R}^{\mathtt{R}}\ \mathtt{C}^{\mathtt{fb}}\ \mathcal{F}_i(\mathbf{x}_0)\ 2^{\mathtt{M}+\tau-\mathsf{k}_i-\mathsf{e}_i} & j = 0 \ \text{(DAC)} \\ \\ \mathsf{b}^{S}_{i,j}\ \mathtt{R}^{\mathtt{R}}\ \mathtt{C}^{\mathtt{fb}}\ \frac{\partial \mathcal{F}_i}{\partial \mathsf{x}_j}(\mathbf{x}_0)\ 2^{\tau-\mathsf{k}_i-\mathsf{e}_i+\mathsf{k}_j+\mathsf{e}_j} & j \neq 0 \ \text{(R2R)} \end{cases}$$

## 10.1 Control Variables

The following control variables avoid hardware limitations while simultaneously optimizing the integration process:

$\tau$ - Time scaling factor (integer values)
$[\mathsf{k}_1, \ldots, \mathsf{k}_n]$ - Scaling Index (integer values, $\mathtt{k_{min}} \leq \mathsf{k}_i \leq \mathtt{k_{max}}$)

The time scaling factor can be changed at any instant during the integration process. On the other hand, adjustments to each scaling index must be made at resets of corresponding analog bits. The scaled signal will momentarily be zero at this instant.

## 10.2 Hardware Constraints

The limited range of each DAC and R2R units R-2R ladder imposes the constraint (range for **Equation 4.4**):

$$0 \leq \mathsf{S}_{i,j} < 1 \tag{10.1}$$

This limitation will usually be approached by the DAC unit, rather than an R2R unit, for two reasons. First, the scaling factor of $2^{\mathtt{M}}$ in **Equation 9.6** results in a large number. Secondly, each DAC is incremented by an R2R upon every analog bit reset that resets.

## 10.3 Digital Processing Constraints

During integration, digital processing maintains each nonlinear function approximation and maintains each nonlinear function approximation. Under certain circumstances, digital processing may not be fast enough to provide adequate function approximation updates. To control such scenarios, the rate of analog integration may be held below a value, $\tau^{max}$, that digital processing can handle.

$$\tau \leq \tau^{max} \tag{10.2}$$

## 10.4 Hardware Ability Constraint

The following constraint attempts to fully utilyze the dynamic range of every DAC and R2R unit:

$$\max(\mathsf{S}_{i,j}) \ \geq \ 0.5 \quad if \quad \tau < \tau^{max} \tag{10.3}$$

A DAC/R2R satisfying ($\mathsf{S}_{i,j} \geq 0.5$) fully utilizes it's R-2R ladder, maintaining optized resolution and highest signal to noise rations for currents.

## 10.5 Scaling Control Methodologies

Implementing a control with multiple control variables, $(\mathsf{k}_1, \mathsf{k}_2, ..., \mathsf{k}_n, \tau)$, may seem overwhelming. However, once the following rule is established, the process becomes trivial:

**FUNDAMENTAL SCALING CONTROL RULE**[1]

*The time scale factor ($\tau$) is controlled to satisfy scaling constraints*
*(**Equation 9.6**, **Equation 10.1**, **Equation 10.2**, and **Equation 10.3**)*

Using this rule, every scaling index can be set to any desired level. With these parameters, the performance of mixed signal processing can be adjusted. A few methods are discussed below:

### 10.5.1 Group Fixed

"Group Fixed" control methodology, equates scaling indices:

$$\mathtt{k_F} = \mathsf{k}_1 = \mathsf{k}_2 = .... = \mathsf{k}_n$$

This methodology is equivalent to using a smaller mantissa; All scaling indice have the same value during the entire integration process. The primary benefit is simplicity. Between integrations, the value may be adjusted to achieve the simulations speed desired.

### 10.5.2 Group Moving

The "Group Moving" control methodology uses scaling indices set as:[2]

$$k \ = \ k_1 \ = \ k_2 \ = \ .... \ = \ k_n$$

This method allows simple scaling index adjustments to modify the rate at which integration takes place.

### 10.5.3 Maximize DAC Currents

The "Maximum DAC Current" methodology adjusts scaling indices such that each DAC provides maximum current. Forcing each DAC to provide high amounts of

[1]This rule applies to all control methodologies described below except "Real Time"

[2]This equation functions as a set point for the scaling index; before changing values, each term must wait for its analog bit to reset.

current could be used as a method to reduce errors attributed to leakage current. Additionally, this methodology allows for each state variable to encounter nearly the same number of resets during a simulation.

### 10.5.4 Time Scale Fixed

The "Time Scale Fixed" method attempts to keep the time scale factor fixed. Upon noticing a change in the time scale factor, proper scaling index adjustments are made to allow the time scale switch back. This method provides the ability to perform simulations at a determined speed.

## 10.6 "Real Time" Scaling Control

A "Real Time" control method ($\tau = 0$) omits use of the **Fundamental Scaling Control Rule**, making hardware time equivalent to ODE time ($t = T$). Using this method of processing, continuous satisfaction of hardware constraints (**Equation 9.6** & **Equation 10.1**) is required. This is attempted through maintaining:

$$0.25 \;\leq\; \max(\mathsf{S}_{i,j}) \;\leq\; 0.5$$
$$\mathsf{k}_i\Big|_{i=1,2,..,n}^{MINIMIZE}.$$

# Part IV

# Simulations of Mixed Signal Integration

# Chapter 11

# Simulator Design

## 11.1 Simulator Program

A simulator has been built using Matlab to model mixed signal processing and is included in **Appendix B**.

### Main File

- CREATE_SIMULATION.m

### Parameters File

- PARAMETERS.m

### Supporting Files

- FCNS_ode
- FCNS_simulate.m
- FCNS_ode.m
- FCNS_hardware.m
- FCNS_processing.m
- FCNS_floatingPoint.m

## 11.2 Simulation Parameters

The design parameters listed below have been used for simulations provided in the following chapters [2]:

- Op-amp Integrator:

  $\mathtt{A}^o = 1.5 \times 10^5$ (103.5 dB)

  $\omega^1 = 2\pi \times 10^6$ rad/sec

  $\mathtt{C}^{fb} = 0.47 \times 10^{-6}$ farads

- Reset Circuit:

  $\mathtt{V}^{ref} = 5.8$ volts

  $\mathtt{C}^r = 0.47 \times 10^{-6}$ farads

  $\mathtt{R}^r = 25$ ohms

  $\Delta\mathtt{t}^r = 8\,\mathtt{R}^r\mathtt{C}^r = 11.75\ \mu$s

- State Register:

  $\mathtt{E} = 8$ bits

  $\mathtt{M} = 23$ bits

- DAC and R2R:

  $\mathtt{N}^S = 13$ bits

  $\mathtt{R}^R = \mathtt{R}^{2R} = 1000$ ohms w/ 0.1% tolerance

# Chapter 12

# Simulations: First Order ODE's

## 12.1 Solution for $\frac{\mathrm{dx}}{\mathrm{d}t} = 1$

As an initial test, this simple ODE is simulated to verify basic functionality of the simulator program. **Figure 12.1** demonstrates the results of a short simulation, showing the analog bit's value during this time interval.

### 12.1.1 Effect of the Scaling Index

A Simulation using a small scaling index ($\mathsf{k} = 0$) is demonstrated in **Figure 12.2**; the primary characteristic of using a small scaling index is a small amount of change represented by each analog bit reset. **Figure 12.3** displays the use of a medium scaling index ($\mathsf{k} = 10$), followed by **Figure 12.4** with a large scaling index ($\mathsf{k} = 20$).

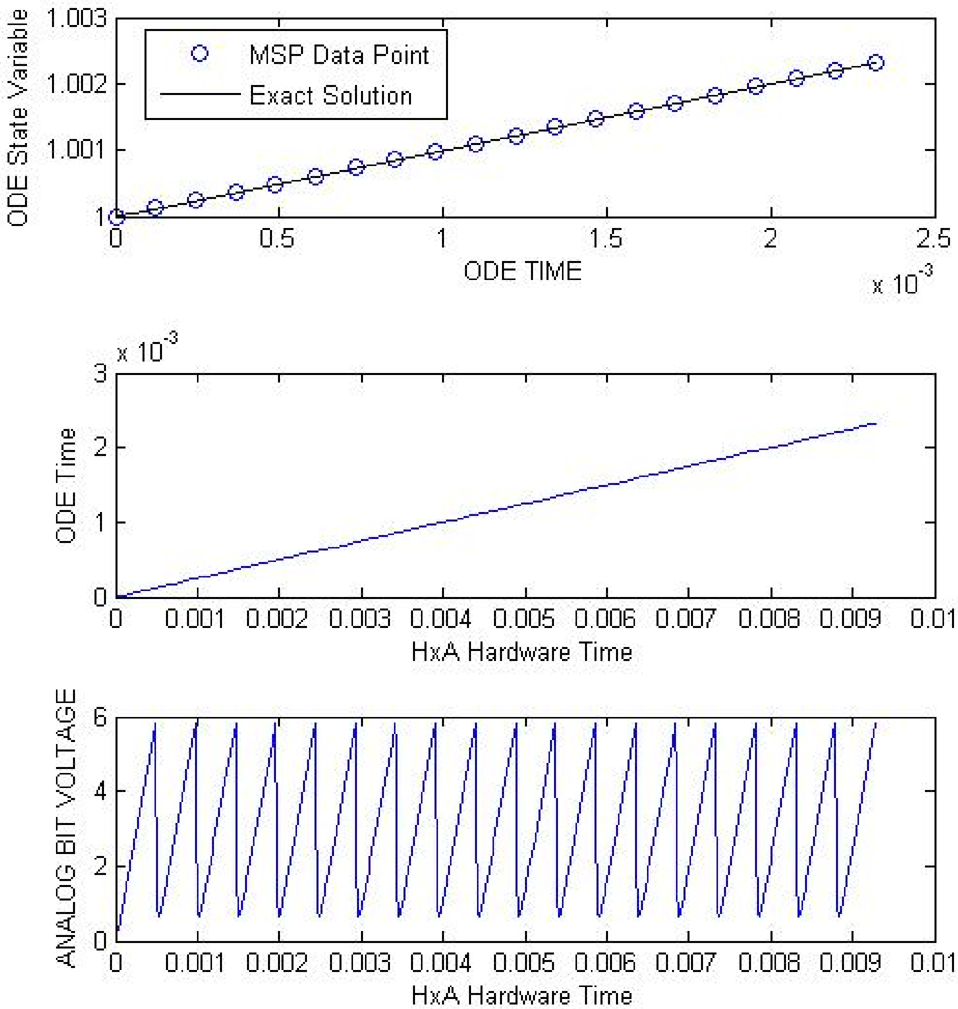


Figure 12.1: Simulation results of $\frac{\mathrm{dx}}{\mathrm{d}t} = 1$ with $\mathsf{k} = 10$. Top: ODE solution over a short time interval. Middle: Relationship between ODE time and HxA hardware time. Bottom: Plot of the analog bit voltage during this simulation

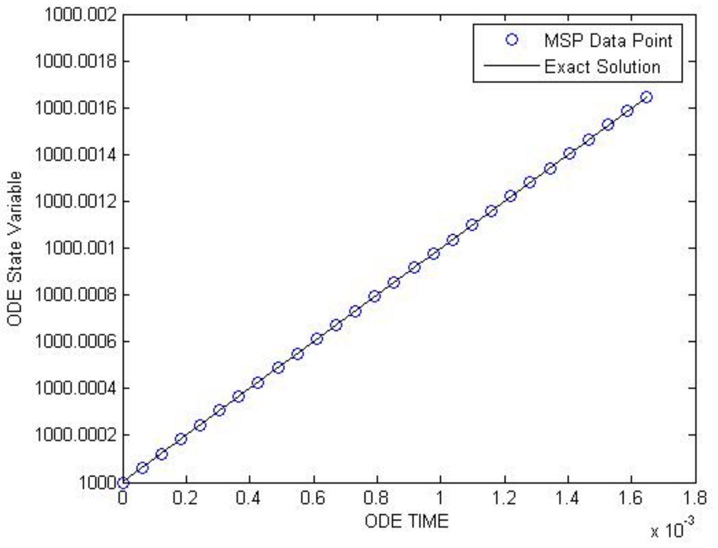


Figure 12.2: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = 1$ with $\mathsf{k} = 0$.

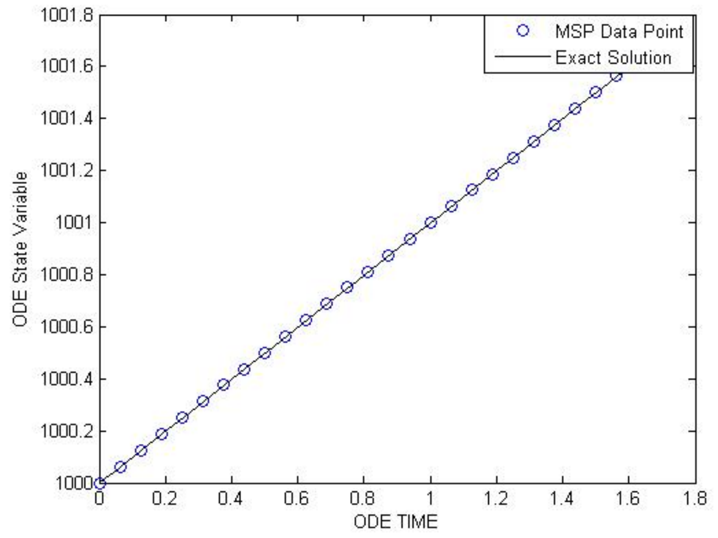


Figure 12.3: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = 1$ with $\mathsf{k} = 10$.

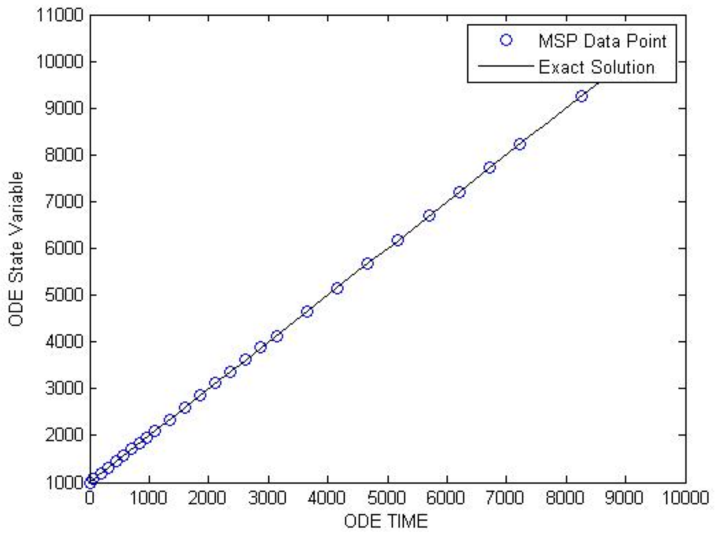


Figure 12.4: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = 1$ with $\mathsf{k} = 20$.

### 12.1.2 Zero Crossing

A simulation of zero crossing has been considered in **Figure 12.5** to insure that the simulation program properly makes sign transitions. Additionally, the results verify that even when using a high scaling index, an abundant number of analog bit resets must be processed in each pass through zero.

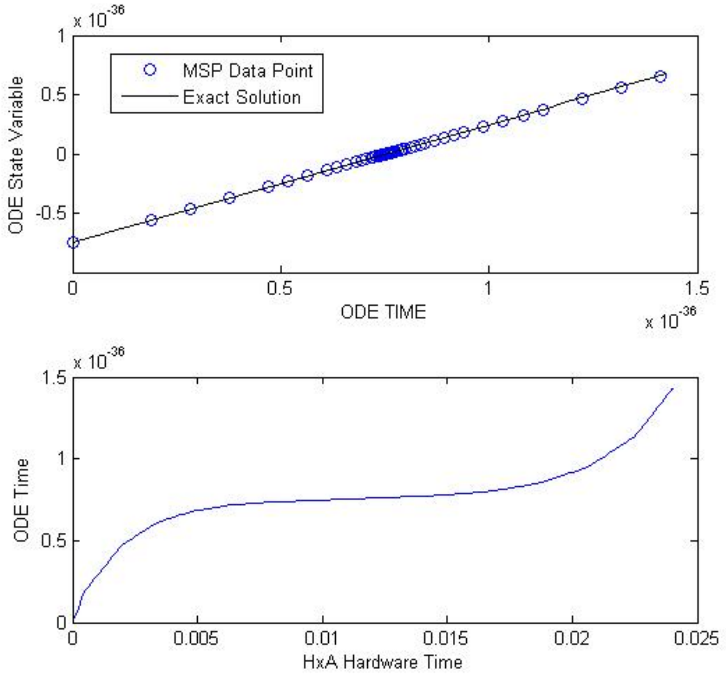


Figure 12.5: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = 1$ for crossing zero with $\mathsf{k} = 21$. Notice the time duration (for hardware processing) involved with this transition.

### 12.1.3 Modified Denormalization

The use of modified denormalization allows for a dramatic reduction in the number of resets involved with zero crossing. This is demonstrated for the system below:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{z} = 1 = \mathcal{H}(\mathrm{z})$$

$$\mathrm{z}_0 = -1$$

$$\mathsf{Z}^D = 1$$

The linearly scaled ODE below is to be solved in place of the ODE given above:

$$\mathrm{z} = \mathtt{Z}^D\, 2^{126}\, \mathrm{x} = 2^{126}\, \mathrm{x}$$
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{x} = \frac{2^{-126}}{\mathtt{Z}^D}\, \mathcal{H}(\mathtt{Z}^D\, 2^{126}\, \mathrm{x}) = 2^{-126} = \mathcal{F}(\mathrm{x})$$
$$\mathrm{x}_0 = \left[\frac{2^{-126}}{\mathtt{z}^D}\right] \mathrm{z}_0 = -2^{-126}$$

Upon obtaining a solution to this newly created ODE, the original ODE's solution is obtained with the ODE's linearly scaled relationship. The solution (in terms of z) obtained through the simulation of this system is presented in **Figure 12.6**.

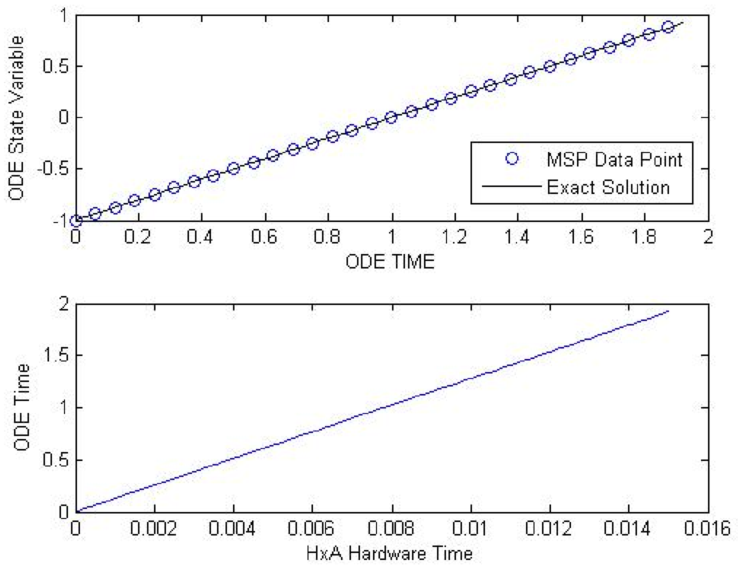


Figure 12.6: Simulation Results of $\frac{\mathrm{dz}}{\mathrm{d}t} = 1$ for crossing zero with $\mathsf{k} = 19$ when using modified denormalization ($\mathtt{Z}^D = 1$).

## 12.2 Solution for $\frac{\mathrm{dx}}{\mathrm{d}t} = -\mathrm{x}$

Two simulations have been provided for this simple first order linear ODE. **Figure 12.7** demonstrates this solution using a scaling index that provides a reasonably large number of analog bit resets for demonstration. **Figure 12.8** uses the highest scaling index ($\mathtt{k} = 22$) to minimize the value of $(\mathtt{M} - \mathtt{k}_1)$ in **Equation 9.6**; this maximizes $\tau$ when **Equation 10.1** is constrained by DAC, thus maximizing the current $(\mathrm{I}_{1,1}^{R2R})$ provided with the R2R unit.

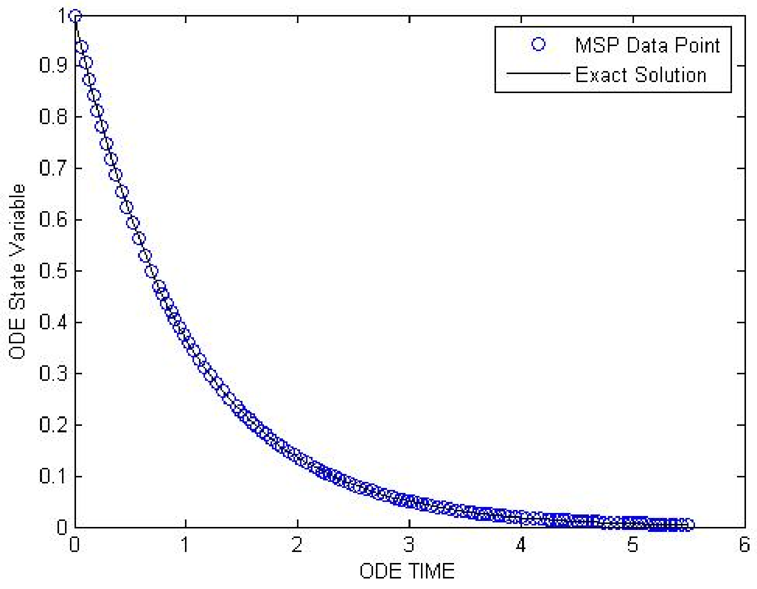


Figure 12.7: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = -\mathtt{x}$ with $\mathtt{k} = 19$.

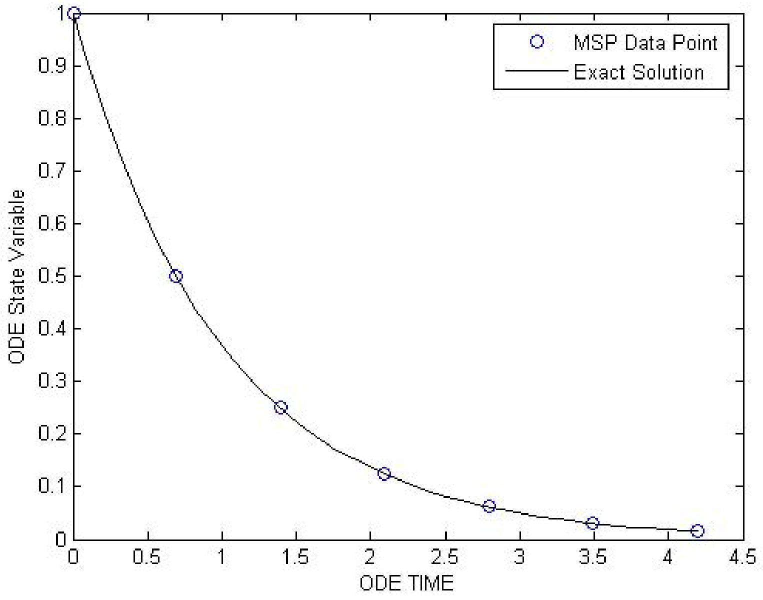


Figure 12.8: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = -\mathrm{x}$ with $\mathsf{k} = 22$.

## 12.3 Solution for $\frac{\mathrm{dx}}{\mathrm{d}t} = \mathrm{x}^2$

Three simulations of this nonlinear system have been performed using various function re-approximation frequencies. **Figure 12.9** demonstrates a simulation with adequate function re-approximation. Upon function re-approximation becoming less frequent than necessary, error begins to accumulate as noticed in **Figure 12.10** and **Figure 12.11**.

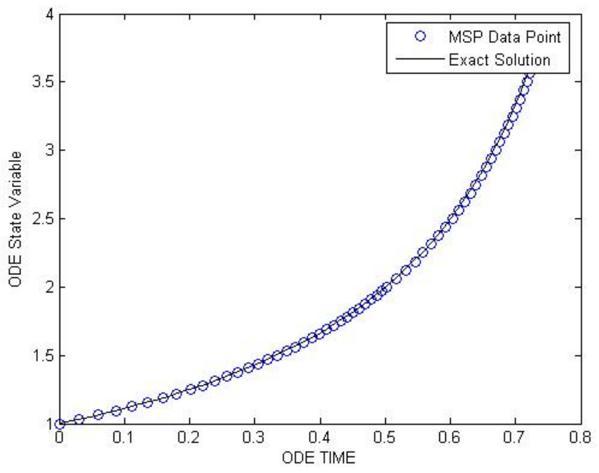


Figure 12.9: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = \mathrm{x}^2$ with $\mathsf{k} = 18$. Taylor Series approximations are updated every 0.001 seconds(HxA hardware time); the simulation processed in approximately 0.04 seconds.

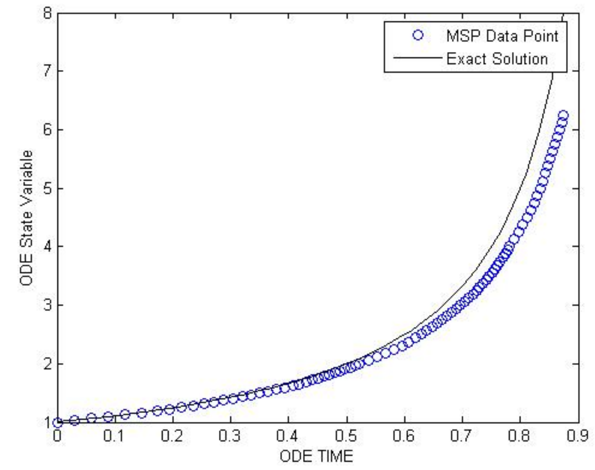


Figure 12.10: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = \mathsf{x}^2$ with $\mathsf{k} = 18$. Taylor Series approximations are updated every 0.0025 seconds(HxA hardware time); the simulation processed in approximately 0.045 seconds.

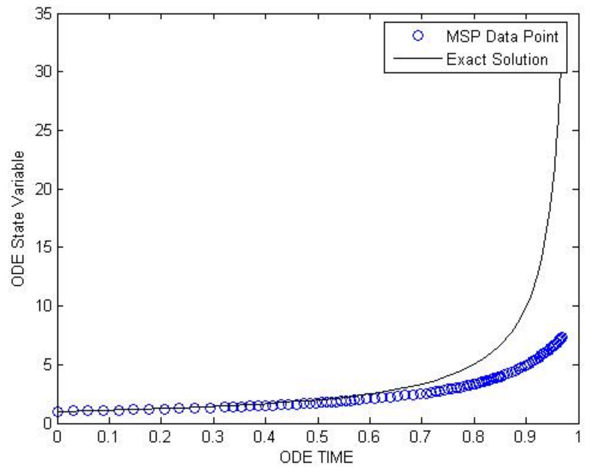


Figure 12.11: Simulation Results of $\frac{\mathrm{dx}}{\mathrm{d}t} = \mathsf{x}^2$ with $\mathsf{k} = 18$. Taylor Series approximations are updated every 0.005 seconds(HxA hardware time); the simulation processed in approximately 0.05 seconds.

# Chapter 13

# Simulations: Second Order ODE's

## 13.1 Second Order Linear Systems

A general description for second order linear systems is given as:

$$\frac{\mathrm{d}^2\mathsf{z}}{\mathrm{d}t^2} + 2\xi\omega_n\frac{\mathrm{d}\mathsf{z}}{\mathrm{d}t} + \omega_n^2\mathsf{z} = 0$$

where,

$\omega_n$ - Natural Frequency

$\xi$ - Damping Ratio

This system may be converted into the following system of first order ODE's:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \mathsf{z}_1 \\ \mathsf{z}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\xi\omega_n \end{bmatrix}\begin{bmatrix} \mathsf{z}_1 \\ \mathsf{z}_2 \end{bmatrix}$$

Using modified denormalization described by:

$$\mathsf{Z}^D = [\ \mathsf{Z}_1^D,\ \mathsf{Z}_2^D]^T$$

$$\begin{bmatrix} \mathrm{z}_1 \\ \mathrm{z}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{Z}_1^D \ 2^{126} \ \mathrm{x}_1 \\ \mathrm{Z}_2^D \ 2^{126} \ \mathrm{x}_2 \end{bmatrix}$$

where the ODE is solved in terms of $\mathbf{x}$ using mixed signal processing, then transformed back into terms of $\mathbf{z}$. A critically damped system has been simulated in **Figure 13.1**; additionally, an undamped system is included in **Figure 13.2**.

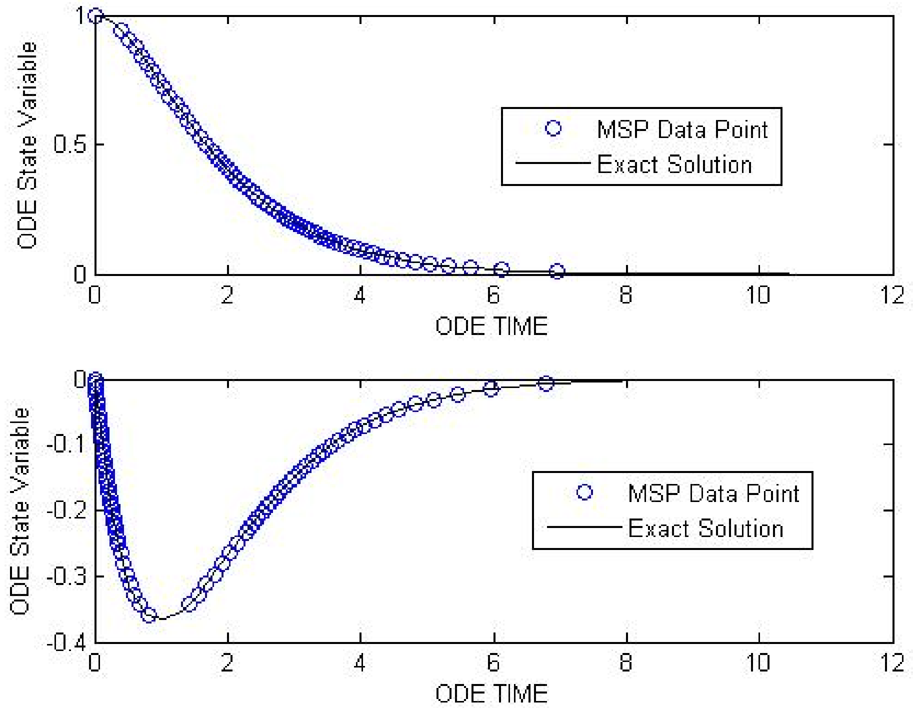


Figure 13.1: Simulation results of critically damped system system using modified denormalization ($\xi = 1,\ \omega_n = 1,\ \mathbf{z}_0 = [1, 0]^T,\ \mathrm{Z}^D = [2^{-3},\ 2^{-3}]^T,\ \mathrm{k}_1 = \mathrm{k}_2 = 19$). The simulation processed in approximately 0.07 seconds (HxA hardware time). Top: $\mathrm{z}_1$. Bottom: $\mathrm{z}_2$.

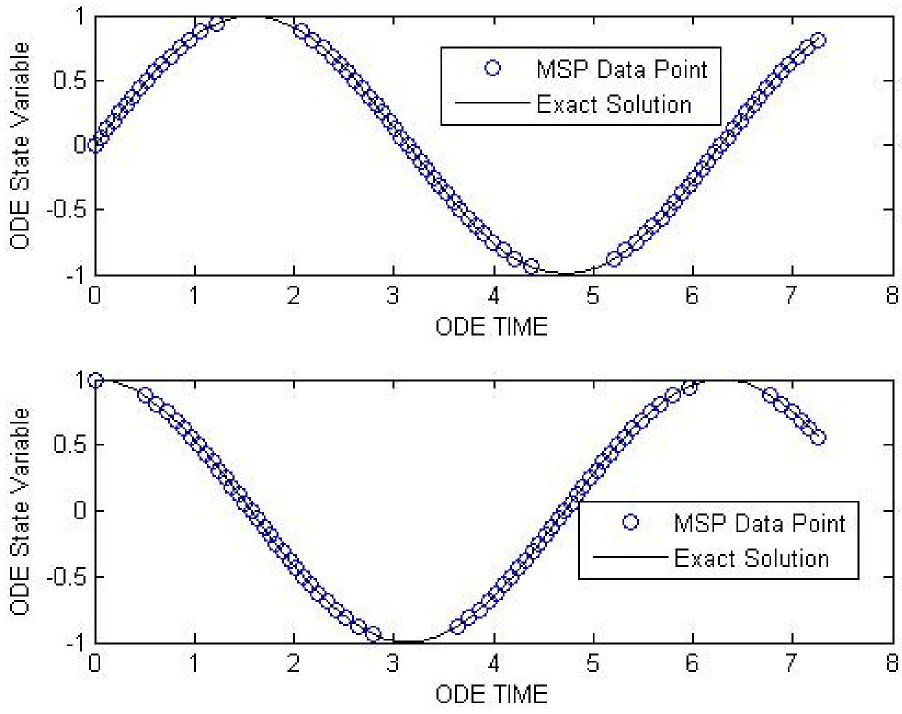
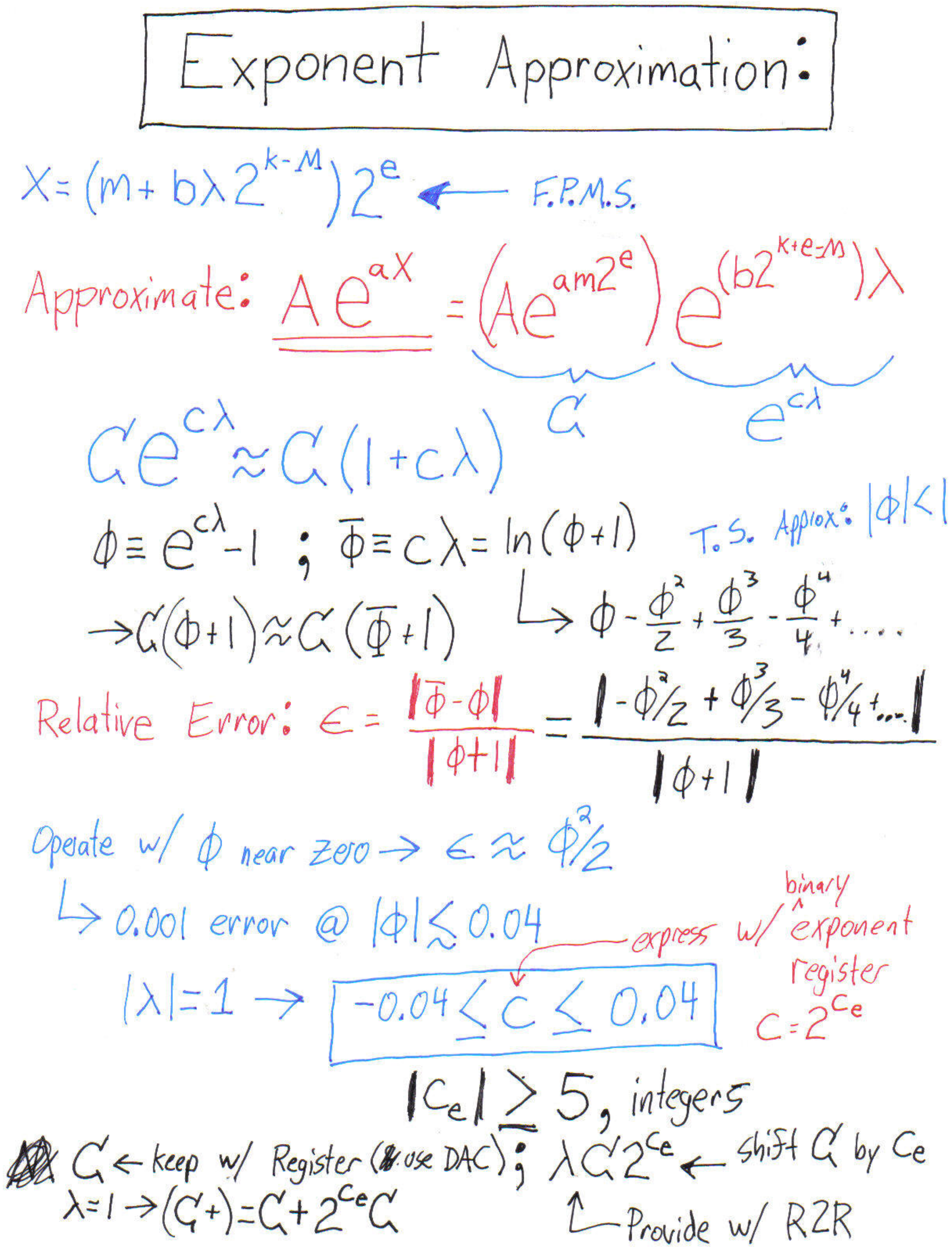


Figure 13.2: Simulation results of undamped system system using modified denormalization ($\xi = 0,\ \omega_n = 1,\ \mathbf{z}_0 = [0, 1]^T,\ \mathsf{Z}^D = [2^{-1},\ 2^{-1}]^T,\ \mathsf{k}_1 = \mathsf{k}_2 = 20$). The simulation processed in approximately 0.06 seconds (HxA hardware time). Top: $\mathrm{z}_1$. Bottom: $\mathrm{z}_2$.

# Appendix A

# Idea for Exponent Approximation:

These are some notes for an idea to approximate exponent functions. The primarily difference in this methodology, compared to a purely linearized Taylor Series Approximation is each Analog Bit Reset updates the approximation held in each R2R ladder. Using this methodology, digital processing could locally function approximate using exponentials; currently, local function approximation uses linear relationships.



Exponent Approximation:

$X = (m + b\lambda 2^{k-M})2^{e} \leftarrow$ F.P.M.S.

Approximate: $\underline{\underline{Ae^{aX}}} = \underbrace{\left(Ae^{am2^{e}}\right)}_{C}\underbrace{e^{(b2^{k+e-M})\lambda}}_{e^{c\lambda}}$

$$Ce^{c\lambda} \approx C(1 + c\lambda)$$

$\phi \equiv e^{c\lambda} - 1$ ; $\Phi \equiv c\lambda = \ln(\phi + 1)$ T.S. Approx: $|\phi| < 1$

$\rightarrow C(\phi + 1) \approx C(\Phi + 1)$ $\hookrightarrow \phi - \frac{\phi^2}{2} + \frac{\phi^3}{3} - \frac{\phi^4}{4} + \dots$

Relative Error: $\epsilon = \frac{|\Phi - \phi|}{|\phi + 1|} = \frac{|-\phi^2/2 + \phi^3/3 - \phi^4/4 + \dots|}{|\phi + 1|}$

Opeate w/ $\phi$ near zero $\rightarrow \epsilon \approx \phi^2/2$

$\hookrightarrow$ 0.001 error @ $|\phi| \lesssim 0.04$

$|\lambda| = 1 \rightarrow \boxed{-0.04 \leq c \leq 0.04}$ express w/ binary exponent register $c = 2^{C_e}$

$|C_e| \geq 5$, integers

$C \leftarrow$ keep w/ Register (use DAC); $\lambda C 2^{C_e} \leftarrow$ shift $C$ by $C_e$

$\lambda = 1 \rightarrow (C+) = C + 2^{C_e}C$ $\uparrow$ Provide w/ R2R

Figure A.1: Idea for Exponent Approximation. NOTE: (top right) Exponent is missing 'a' term.

# Appendix B

# Simulator Programming: MATLAB m-files

**Main File:**

- CREATE_SIMULATION.m

**Parameters File:**

- PARAMETERS.m

**Additional Simulation Files:**

- FCNS_simulate.m
- FCNS_ode.m
- FCNS_hardware.m
- FCNS_processing.m
- FCNS_floatingPoint.m

## FILE: CREATE_SIMULATION.m

```
%%  CREATE_SIMULATION.m
%
%   This function is used to simulate the HxA.  To perform a new
%   simulation, an "ODE file", a "PARAMETERS file", and initial conditions
%   must be specified.  To Continue a previous simulation, the information
%   from previous simulation must be present in Matlabs Workspace.
global SIMULATION FUNCTION VARIABLE PARAMETER
global DATA ODE FA

%%  GENERAL INFORMATION:
    % CONTINUE SIMULATIONS IF POSSIBLE:
    % - specify 'no' to option out
        SIMULATION.CONTINUE = 'no';
    % OUTPUT DIRECTORY
        SIMULATION.DIRECTORY = [pwd, '\RESULTS'];

%%  ODE FUNCTION:
    % Order of ODE
        FUNCTION.SIZE = 2;
    % State Variable Vector: X = [X(1); X(2); ...; X(N)]
        FUNCTION.X = FCNS_ode('Create X', FUNCTION.SIZE);
    % ODE function
        X = FUNCTION.X;
        FUNCTION.F = [ -X(2); -X(1); ]; clear X
        FUNCTION.FILENAME.F = 'fcn_F';
    % Partial derivatives of ODE function
        FUNCTION.DF = ...
            FCNS_ode('Create DF', FUNCTION.X, FUNCTION.F);
        FUNCTION.FILENAME.DF = 'fcn_DF';
    % Initial Conditions:   ["X0_1"; "X0_2"; ..]
        FUNCTION.X_0 = [5; 5];
        FUNCTION.t_0 = 0;

%%  PARAMETERS USED IN SIMULATION:
        % DIRECTORY:
            SIMULATION.PARAMETERS.DIRECTORY = [pwd, '\PARAMETERS'];
        % FILE:    ['FILE_NAME']
            SIMULATION.PARAMETERS.FILENAME =  'PARAMETERS';
        % FILE MODIFICATION CODE:
            % SYSTEM SIZE
                i=1;    SIMULATION.OVERWRITE(i).String = ...
                ['PARAMETER.n = ' int2str(FUNCTION.SIZE) ';'];
            % State Register: Exponent/Mantissa Size
                    i=i+1;  SIMULATION.OVERWRITE(i).String = ...
                    ['PARAMETER.E = 8;'];
                    i=i+1;  SIMULATION.OVERWRITE(i).String = ...
                    ['PARAMETER.M = 23;'];
            % Scaling Index: Initial Value
                    i=i+1;  SIMULATION.OVERWRITE(i).String = ...
                    ['PARAMETER.k_0 = 19;'];
            % DAC/R2R: Bits in Mantissa Register
                    i=i+1;  SIMULATION.OVERWRITE(i).String = ...
                    ['PARAMETER.N_S = 13;'];
            % Function Approximation
                % Time between function approximation updates
                    i=i+1;  SIMULATION.OVERWRITE(i).String = ...
                    ['PARAMETER.dT_fa = 0.005;'];
                    clear i

%%  INITIALIZE SIMULATION
    clear global PARAMETER VARIABLE
    global PARAMETER VARIABLE
    path(pathdef);
    % SIMULATION TYPE: 'NEW' or 'CONTINUED'
        OUTPUT.Simulation = FCNS_simulate('New or Continued');
    % INITIALIZATION
        OUTPUT.Initialization = FCNS_simulate('Initialize',...
            OUTPUT.Simulation);


%% PROCESS SIMULATION
    DATA.T = []; DATA.Z = [];
    for i=1:50
        [T_out, Z_out, type, location] = ...
            FCNS_simulate('Simulate until Event');
        FCNS_simulate('Process Event', type, location);
        DATA.T = [ DATA.T; T_out];
        DATA.Z = [ DATA.Z; Z_out];
        clear T_out Z_out type location
    end, clear i

%% DISPLAY RESULTS
    DISPLAY_RESULTS
    cd ..
```

## FILE: FCNS_simulate.m

```
function [varargout] = FCNS_simulate(varargin)

%   This function performs all major functions required during the

%   simulation.

global SIMULATION FUNCTION PARAMETER VARIABLE

global ODE FA


%% SIMULATION TYPE: 'NEW' OR 'CONTINUED'

if strcmpi(varargin{1},'New or Continued')

    path(pathdef, pwd);

    cd(SIMULATION.DIRECTORY);

    STRING = 'NEW';

    if ~strcmpi(SIMULATION.CONTINUE, 'no')

        if isfield(VARIABLE, 'R2R') && isfield(FUNCTION, 'SIZE')

        if isfield(VARIABLE.R2R, 'S') && isfield(PARAMETER, 'N_S')

        if length(VARIABLE.R2R(1,1,:) == PARAMETER.n)

        if length(VAR.D.S(:,1,1)== PARAMETER.N_S)

            STRING = {'CONTINUED'};

        end, end, end, end

    end

    varargout(1) = {STRING};
```

```
%% INITIALIZE SIMULATION

elseif strcmpi(varargin{1},'Initialize')

    if strcmpi(varargin{2}, 'CONTINUED')

        STATUS = 'Initialization process skipped';

    elseif strcmpi(varargin{2}, 'NEW')

    % CLEAR VARIABLES

            FA.t = [];  FA.T = [];

        for i=1:FUNCTION.SIZE,

            ODE(i).t = []; ODE(i).x = [];

        end, clear i

    % CREATE ODE FUNCTION

        STATUS.ode = FCNS_ode('Create M-Files', ...

            FUNCTION.X, FUNCTION.F, FUNCTION.DF, ...

            FUNCTION.FILENAME.F, FUNCTION.FILENAME.DF);

        FUNCTION.F_HANDLE = str2func(FUNCTION.FILENAME.F);

        FUNCTION.DF_HANDLE = str2func(FUNCTION.FILENAME.DF);

    % DECLARE PARAMETERS

        cd(SIMULATION.PARAMETERS.DIRECTORY);

        eval( SIMULATION.PARAMETERS.FILENAME );

        STATUS.parameters = ['EXECUTED: ''' ...

            SIMULATION.PARAMETERS.FILENAME ''];

        cd(SIMULATION.DIRECTORY);
```

```
% ODE VARIABLES

    for i=1:PARAMETER.n

        ODE(i).x = FUNCTION.X_0(i);

        ODE(i).t = FUNCTION.t_0;

        FCNS_floatingPoint('FP_X', i, FUNCTION.X_0(i));

    end

    VARIABLE.ODE.t = FUNCTION.t_0;

    VARIABLE.DYNAMICS.T = 0;

% SCALING VARIABLES

    VARIABLE.SCALING.tau = 0;

    VARIABLE.SCALING.k = ones(PARAMETER.n, 1)*PARAMETER.k_0;

% RESET SIGNALS

    VARIABLE.RESET.S_r = zeros( PARAMETER.n, 1);

    VARIABLE.RESET.T_r = zeros( PARAMETER.n, 1);

% DYNAMIC VARIABLES: SET TO ZERO INITIALLY

      % Analog Bit Voltage

        VARIABLE.DYNAMICS.V_a = zeros( PARAMETER.n,1);

        VARIABLE.DYNAMICS.DV_a = zeros( PARAMETER.n,1);

      % Feedback Capacitor Charge

        VARIABLE.DYNAMICS.Q_fb = zeros( PARAMETER.n,1);

        VARIABLE.DYNAMICS.DQ_fb = zeros( PARAMETER.n,1);

      % Reset Capacitor Charge
```

```
        VARIABLE.DYNAMICS.Q_r = zeros( PARAMETER.n,1);

        VARIABLE.DYNAMICS.DQ_r = zeros( PARAMETER.n,1);

      % Virtual Ground

        VARIABLE.DYNAMICS.V = zeros( PARAMETER.n,1);

      % Function Approximation Current

        VARIABLE.DYNAMICS.I = zeros( PARAMETER.n,1);

      % Reset Current

        VARIABLE.DYNAMICS.I_r = zeros( PARAMETER.n,1);

% DAC's and R2R's

    % Function Approx Update time:

        VARIABLE.FUNCTION_APPROXIMATION.T = 0;

    % R2R and DAC SIZING (give zero values)

        % R2R Mantissa Registers:

            VARIABLE.R2R.S = zeros( PARAMETER.N_S, PARAMETER.n, ...

                PARAMETER.n);

        % R2R Sign Register:

            VARIABLE.R2R.B = zeros( PARAMETER.n, PARAMETER.n);

        % R2R Mantissa Value w/ sign

            VARIABLE.R2R.value = zeros( PARAMETER.n, PARAMETER.n);

        % DAC Mantissa Registers

            VARIABLE.DAC.S =  zeros( PARAMETER.N_S, PARAMETER.n);

        % DAC Sign Resister
```

```
                VARIABLE.DAC.B = zeros( PARAMETER.n, 1);

            % DAC Mantissa value w/ sign

                VARIABLE.DAC.value = zeros( PARAMETER.n, 1);

        % SET DAC and R2R Value

            FCNS_processing('function approximation', 'all');

            VARIABLE.FUNCTION_APPROXIMATION.T = ...

                VARIABLE.DYNAMICS.T + PARAMETER.dT_fa;

        STATUS.variables = ['All variables initialized'];

        varargout(1) = {STATUS};

    end


%% SIMULATE UNTIL EVENT

elseif strcmpi(varargin{1},'Simulate until Event')

    [T, Z] = FCNS_hardware('Solver State');

    SIMULATION.ZDOT = @( T, Z) FCNS_hardware('Zdot Function', T, Z);

    SIMULATION.EVENTS = @(T, Z) FCNS_hardware('Event Function', T, Z);

    SIMULATION.OPTIONS = odeset('events', SIMULATION.EVENTS);

    [T_out, Z_out, T_end, Z_end, N_end] = ...

            ode15s( SIMULATION.ZDOT, [T, T+1], Z, SIMULATION.OPTIONS);

    FCNS_hardware('Update HxA', T_end, Z_end);

    [type, location] = FCNS_hardware('Event Information', N_end);

    varargout(1) = {T_out};
    varargout(2) = {Z_out};

    varargout(3) = {type};

    varargout(4) = {location};


%% PROCESS EVENTS

elseif strcmpi(varargin{1},'Process Event')

    type = varargin{2};

    location = varargin{3};

    if strcmpi(type,'pabr')

        I = location;

        VARIABLE.RESET.S_r(I) = 1;

        VARIABLE.RESET.T_r(I) = VARIABLE.DYNAMICS.T;

        FCNS_processing('Analog Bit Reset', 'pabr', I);

        L_ODE = length(ODE(I).t);

        ODE(I).x(L_ODE+1) = VARIABLE.ODE.X(I);

        ODE(I).t(L_ODE+1) = VARIABLE.ODE.t;

    % Negative Analog Bit Reset

    elseif strcmpi(type,'nabr')

        I = location;

        VARIABLE.RESET.S_r(I) = -1;

        VARIABLE.RESET.T_r(I) = VARIABLE.DYNAMICS.T;

        FCNS_processing('Analog Bit Reset', 'nabr', I);
        L_ODE = length(ODE(I).t);

        ODE(I).x(L_ODE+1) = VARIABLE.ODE.X(I);

        ODE(I).t(L_ODE+1) = VARIABLE.ODE.t;

    % Function Approximation

    elseif strcmpi(type,'function approximation')

        FCNS_processing('function approximation', 'all');

        VARIABLE.FUNCTION_APPROXIMATION.T = ...

            VARIABLE.FUNCTION_APPROXIMATION.T + PARAMETER.dT_fa;

        FA.t(length(FA.t)+1) = VARIABLE.ODE.t;

        FA.T(length(FA.T)+1) = VARIABLE.DYNAMICS.T;

    end

end
```

## FILE: FCNS_ode.m

```
function [varargout] = FCNS_ode(varargin)
%%
%   This function creates the ode files used by the simulation program.
%
%  INPUT FORMS
%   [X] = FCNS_ode('Create X', Length_of_X )
%   [DF] = FCNS_ode('Create DF', X, F)
%   [] = FCNS_ode('Create M-Files', X, F, DF, ...
%       F_FileName, DF_File_Name)

%%  CODE
if strcmpi(varargin{1}, 'Create X')
        Length_of_X = varargin{2};
    X = STATE_VECTOR( Length_of_X );
    varargout(1) = {X};
elseif strcmpi(varargin{1}, 'Create DF')
        X = varargin{2};
        F = varargin{3};
    DF = PARTIAL_DERIVATIVES( F, X );
    varargout(1) = {DF};
elseif strcmpi(varargin{1}, 'Create M-Files')
        X = varargin{2};
        F = varargin{3};
        DF = varargin{4};
        F_FileName = varargin{5};
        DF_FileName = varargin{6};
    [s_X, s_F, s_DF] = CREATE_STRINGS(F, DF, X, F_FileName, DF_FileName);
    CREATE_FILES(s_X, s_F, s_DF, F_FileName, DF_FileName);
    varargout(1) = {'ODE FUNCTIONS CREATED'};
end

%%  SUBFUNCTION: CREATE STATE VECTOR ------------------------------------
function X = STATE_VECTOR( Length_of_X )
    for i=1:Length_of_X
        eval([ 'X(' int2str(i) ',1) = sym(''x' ...
            int2str(i) ''', ''real'');']);
    end
return

%%  SUBFUNCTION: CALCULATE PARTIAL DERIVATIVES ---------------------------
function DF = PARTIAL_DERIVATIVES( F, X )
    for i = 1: length(F)
```

```
        for j = 1: length(F)
            DF(i,j) = diff(F(i),X(j));
        end
    end
return

%%  SUBFUNCTION: CREATE M-FILE STRINGS
function [s_X, s_F, s_DF] = CREATE_STRINGS(F, DF, X, F_Name, DF_Name)
%   This subfunction creates the character strings used in writing each
%   m-file.
    s_F.HEADING = ...
        ['function [ F ] = ' F_Name '( X )'];
    s_DF.HEADING = ...
        ['function [ DF ] = ' DF_Name '( X )'];
    for i=1:length(X)
        s_X.LINE(i).STRING = ...
            ['x' int2str(i) ' = X(' int2str(i) ');'];
        s_F.LINE(i).STRING = ...
            ['F(' int2str(i) ', 1) = ' char( F(i) ) '; ' ];
        for j=1:length(X)
            s_DF.LINE(i,j).STRING = ...
                ['DF(' int2str(i) ', ' int2str(j) ') = ' ...
                char(DF(i,j)) ';' ];
        end
    end
return

%% SUBFUNCTION: CREATE M-files
function [] = CREATE_FILES(s_X, s_F, s_DF, F_FileName, DF_FileName)
%   This file creates m-files from character strings.
    fid_F = fopen([F_FileName '.m'], 'wt');
    fid_DF = fopen([DF_FileName '.m'], 'wt');
    % HEADING
        fprintf(fid_F, '%s \n', s_F.HEADING );
        fprintf(fid_DF, '%s \n', s_DF.HEADING);
    % X Variable
    for i=1:length(s_X.LINE)
        fprintf(fid_F, '%s \n', s_X.LINE(i).STRING );
        fprintf(fid_DF, '%s \n', s_X.LINE(i).STRING );
    end
    % Functions
    for i=1:length(s_X.LINE)
        fprintf(fid_F, '%s \n', s_F.LINE(i).STRING );
        for j=1:length(s_X.LINE)
```

```
            fprintf(fid_DF, '%s \n', s_DF.LINE(i,j).STRING );
        end
    end
    fclose(fid_F);
    fclose(fid_DF);
return
```

## FILE: FCNS_hardware.m

```
function [varargout] = FCNS_hardware(varargin)
% [DZ] = FCNS_hardware('Zdot Function', T, Z)
% [] = FCNS_hardware('Update HxA', T, Z)
% [T, Z] = FCNS_hardware('Solver State')
% [value, isterminal, direction] = FCNS_hardware('Event Function', T, Z)
% [type, location] = FCNS_hardware('Event Information', N)
global VARIABLE PARAMETER

%% ZDOT FUNCTION
    if strcmpi(varargin{1},'Zdot function')
        % INPUT
            T = varargin{2};
            Z = varargin{3};
        % CALCULATIONS
            Z_to_HxA(T, Z, 'save');
            Evaluate_HxA();
            DZ = HxA_to_Zdot();
        % OUPTUT
            varargout(1) = {DZ};
%% UPDATE HxA DYNAMICS
    elseif strcmpi(varargin{1}, 'Update HxA')
        % INPUT
            T = varargin{2};
            Z = varargin{3};
        % CALCULATIONS
            Z_to_HxA( T, Z, 'save');
            Evaluate_HxA();
%% GET SOLVER STATE
    elseif strcmpi(varargin{1}, 'solver state')
        [T, Z] = HxA_to_Z();
        % OUPTUT
            varargout(1) = {T};
            varargout(2) = {Z};
%% EVENT FUNCTION
    elseif strcmpi(varargin{1}, 'event function')
        % INPUT
            T = varargin{2};
            Z = varargin{3};
        % CALCULATIONS
            [value, isterminal, direction] = EVENT_values(T,Z);
        % OUPTUT
            varargout(1) = {value};
            varargout(2) = {isterminal};
            varargout(3) = {direction};
%% GET EVENT INFORMATION
    elseif strcmpi(varargin{1},'event information')
        % INPUT
            N = varargin{2};
        % CALCULATIONS
            [type, location] = EVENT_Information(N);
        % OUPTUT
            varargout(1) = {type};
            varargout(2) = {location};
    end


%% SUBFUNCTION: HxA_to_Z
function [T, Z] = HxA_to_Z()
%   This function determines Z (Matlab's solver) variables from HxA (system
%   description) variables.
global VARIABLE PARAMETER
    Z([1:PARAMETER.n],1) = VARIABLE.DYNAMICS.V_a(:,1);
    Z([PARAMETER.n+1:2*PARAMETER.n],1) = VARIABLE.DYNAMICS.Q_fb(:,1);
    Z([2*PARAMETER.n+1:3*PARAMETER.n],1)= VARIABLE.DYNAMICS.Q_r(:,1);
    Z(3*PARAMETER.n+1,1) = VARIABLE.ODE.t; % ODE TIME
    T = VARIABLE.DYNAMICS.T;   % HxA HARDWARE TIME
return

%% SUBFUNCTION: Z_to_HxA
function [varargout] = Z_to_HxA( T, Z, SAVE)
%   This function transforms Z (Matlab solver) variables into HxA (system
%   description) variables.
global VARIABLE PARAMETER
    if strcmpi(SAVE, 'save')
        VARIABLE.DYNAMICS.V_a(:,1) = Z(1:PARAMETER.n);
        VARIABLE.DYNAMICS.Q_fb(:,1) = Z(PARAMETER.n+1:2*PARAMETER.n);
        VARIABLE.DYNAMICS.Q_r(:,1) = Z(2*PARAMETER.n+1:3*PARAMETER.n);
        VARIABLE.DYNAMICS.T = T;    % "REAL TIME"
        VARIABLE.ODE.t = Z(1+3*PARAMETER.n); % ODE TIME
    elseif strcmpi(SAVE, 'no save')
        V_a(:,1) = Z(1:PARAMETER.n);
        Q_fb(:,1) = Z(PARAMETER.n+1:2*PARAMETER.n);
        Q_r(:,1) = Z(2*PARAMETER.n+1:3*PARAMETER.n);
        t = Z(1+3*PARAMETER.n); % ODE TIME
        varargout(1) = {T};
        varargout(2) = {t};
        varargout(3) = {V_a};
        varargout(4) = {Q_fb};
        varargout(5) = {Q_r};
    else
        disp('ERROR: Input in FCNS_hardware/Z_to_HxA()'); beep;
    end
return

%% SUBFUNCTION: HxA_to_Zdot
function DZ = HxA_to_Zdot()
%   This function determines Zdot (Matlab's solver) variables from HxA
%   (system description) variables.
global VARIABLE PARAMETER
    DZ([1:PARAMETER.n],1) = VARIABLE.DYNAMICS.DV_a;
    DZ([PARAMETER.n+1:2*PARAMETER.n],1) = VARIABLE.DYNAMICS.DQ_fb;
    DZ([2*PARAMETER.n+1:3*PARAMETER.n],1)= VARIABLE.DYNAMICS.DQ_r;
    DZ(3*PARAMETER.n+1,1) = 2^(VARIABLE.SCALING.tau);
return

%% SUBFUNCTION: EVENT_values
function [value, isterminal, direction] = EVENT_values(T,Z)
%   This function gets event information for matlabs solver
global VARIABLE PARAMETER
    [T, t, V_a, Q_fb, Q_r] = Z_to_HxA( T, Z, 'no save');
    %  Positive Analog Bit Resets
        value_PABR(:,1) =  V_a - PARAMETER.V_ref;
    %  Negative Analog Bit Resets
        value_NABR(:,1) = - V_a - PARAMETER.V_ref;
    %  Function Reapproximation
        value_FA(:,1) = T - VARIABLE.FUNCTION_APPROXIMATION.T;
    value = [ value_PABR; value_NABR; value_FA ];
    isterminal = ones(2*PARAMETER.n+1,1);
    direction = ones(2*PARAMETER.n+1,1);
return

%% SUBFUNCTION: EVENT_information
function [type, location] = EVENT_Information(N)
%   This function specifies what type of event has occured based on its
%   event number.
global PARAMETER
    % Not an Event
        if length(N)==0
            type = 'no event';
            location = 0;
    % Multiple events stated
        elseif length(N)>1
            type = 'redundant listing';
            location = 0;
    % Positive Analog Bit Resets
        elseif N <= PARAMETER.n
            type = 'PABR';
            location = N;
    % Negative Analog Bit Resets
        elseif N <= 2*PARAMETER.n      % Negative Reset
            type = 'NABR';
            location = N-PARAMETER.n;
    % Function Reapproximation
        elseif N == 2*PARAMETER.n + 1   % F.A. Needs Updating
            type = 'function approximation';
            location = 0;
    % Unknown?
        else
            type = 'WTF?';
            location = 0;
        end
return

%% SUBFUNCTION: Evaluate_HxA
function [] = Evaluate_HxA()
%   This function calculates information describing HxA based on input
%   values.
global PARAMETER VARIABLE
    %   Solve for the Virtual Ground
            VARIABLE.DYNAMICS.V = VARIABLE.DYNAMICS.V_a + ...
                VARIABLE.DYNAMICS.Q_fb./PARAMETER.C_fb;
    %   Solve for Current: Reset Circuit: DQ_r, I_r
        for i=1:PARAMETER.n
            if( VARIABLE.RESET.S_r(i) == 0 )
                VARIABLE.DYNAMICS.DQ_r(i,1) = -VARIABLE.DYNAMICS.Q_r(i) ...
                    /PARAMETER.R_r/PARAMETER.C_r + PARAMETER.V_ref ...
                    /PARAMETER.R_r;
                VARIABLE.DYNAMICS.I_r(i,1) = 0;
            elseif( VARIABLE.RESET.S_r(i) == -1)
                VARIABLE.DYNAMICS.DQ_r(i,1) = -VARIABLE.DYNAMICS.Q_r(i) ...
                    /PARAMETER.R_r/PARAMETER.C_r + ...
                    VARIABLE.DYNAMICS.V(i)/PARAMETER.R_r;
                VARIABLE.DYNAMICS.I_r(i,1) = VARIABLE.DYNAMICS.DQ_r(i,1);
                if( VARIABLE.DYNAMICS.T > VARIABLE.RESET.T_r(i) + ...
                        PARAMETER.dT_r )
                    VARIABLE.RESET.S_r(i) = 0;
                end
            elseif( VARIABLE.RESET.S_r(i,1) == 1 )
                VARIABLE.DYNAMICS.DQ_r(i,1) = -VARIABLE.DYNAMICS.Q_r(i) ...
                    /PARAMETER.R_r/PARAMETER.C_r - ...
                    VARIABLE.DYNAMICS.V(i)/PARAMETER.R_r;
                VARIABLE.DYNAMICS.I_r(i,1) = -VARIABLE.DYNAMICS.DQ_r(i,1);
                if( VARIABLE.DYNAMICS.T > VARIABLE.RESET.T_r(i) ...
                        + PARAMETER.dT_r )
                    VARIABLE.RESET.S_r(i,1) = 0;
                end
            else, disp('A Signal Sent to Reset Capacitor is not valid');
            end
        end
    %   Solve for Current: DAC/R2R: I
        for i=1:PARAMETER.n
            I=0;
            I = (PARAMETER.C_iGAMMA_DAC(:,:,i)*(PARAMETER.V_ref*  ...
                ((-1)^VARIABLE.DAC.B(i))-[VARIABLE.DAC.S(:,i);0] ...
                *VARIABLE.DYNAMICS.V(i)))'*[VARIABLE.DAC.S(:,i);0];
            for j=1:PARAMETER.n
                I = I+(PARAMETER.C_iGAMMA(:,:,i,j)* ...
                   (VARIABLE.DYNAMICS.V_a(j)*((-1)^VARIABLE.R2R.B(i,j)) ...
                   -[VARIABLE.R2R.S(:,i,j);0]*VARIABLE.DYNAMICS.V(i)))' ...
                   *[VARIABLE.R2R.S(:,i,j);0];
            end
            VARIABLE.DYNAMICS.I(i,1) = I;
        end,
    %   Determine remaining dynamic variables
        %   Analog Bit Voltage
            VARIABLE.DYNAMICS.DV_a= -PARAMETER.w_1.* ...
                VARIABLE.DYNAMICS.V_a./PARAMETER.A_0 - ...
                PARAMETER.w_1.*VARIABLE.DYNAMICS.V;
        %   Feedback Capacitor Charge
            VARIABLE.DYNAMICS.DQ_fb = VARIABLE.DYNAMICS.I + ...
                VARIABLE.DYNAMICS.I_r;
return
```

## FILE: FCNS_processing.m

```
function [varargout] = FCNS_processing(varargin)

global FUNCTION VARIABLE PARAMETER

%%  ANALOG BIT RESET UPDATE
if strcmpi(varargin{1}, 'Analog Bit Reset')
    type = varargin{2}; % 'pabr' or 'nabr'
    location = varargin{3};
    Analog_Bit_Reset(type, location);

%%  FUNCTION APPROXIMATION
elseif strcmpi(varargin{1}, 'Function Approximation')
    % UPDATE ALL REGISTERS
    if strcmpi(varargin{2}, 'all')
        FUNCTION_APPROXIMATION('all', 0, 0);
    % UPDATING A DAC REGISTER
    elseif varargin{3} == 0
        FUNCTION_APPROXIMATION('DAC', varargin{2}, 0);
    % UPDATING AN R2R REGISTER
    else
        FUNCTION_APPROXIMATION('R2R', varargin{2}, varargin{3});
    end

%%  SCALING INDEX CONTROL
elseif strcmpi(varargin{1}, 'Scaling Control')
    % Determine for this Variable number
        i = varargin{2};
end


%% ANALOG BIT RESETS -----------------------------------------------------
function [] = Analog_Bit_Reset(type, location)
global VARIABLE PARAMETER
% Determine Type
    if strcmpi(type, 'pabr')
       reset_sign = 1;
    elseif strcmpi(type, 'nabr')
       reset_sign = -1;
    end
% UPDATE DACS
    for i=1:PARAMETER.n
        VARIABLE.DAC.value(i) = VARIABLE.DAC.value(i) + ...
```

```
            reset_sign*VARIABLE.R2R.value(i,location);
        while(VARIABLE.DAC.value(i) > ( 1 - 2^(-PARAMETER.N_S) ))
            Change_tau( -1 );
        end
        [VARIABLE.DAC.B(i), VARIABLE.DAC.S(:,i)] ...
                    = Num_to_Mant(VARIABLE.DAC.value(i));
    end
% UPDATE STATE REGISTER
    e_old = VARIABLE.ODE.e(location);
    X_new = VARIABLE.ODE.X(location) + reset_sign * ...
        2^( VARIABLE.SCALING.k(location) - PARAMETER.M ...
            + e_old );
    FCNS_floatingPoint('FP_X', location, X_new)
    e_new = VARIABLE.ODE.e(location);
    if( e_new ~= e_old) % exponent change
        Change_exp( e_new - e_old , location);
    end
    Find_Max_tau;
return

%% FUNCTION APPROXIMATION -----------------------------------------------
function [] = FUNCTION_APPROXIMATION(TYPE, I, J)
global FUNCTION VARIABLE PARAMETER
% EVALUATE ODE FUNCTION
    F_val = FUNCTION.F_HANDLE( VARIABLE.ODE.X );
    DF_val = FUNCTION.DF_HANDLE( VARIABLE.ODE.X );
% UPDATE EACH DAC/R2R
    for i=1:PARAMETER.n
    % APPROXIMATE DAC VALUES
        if strcmpi(TYPE, 'all') || ( I==i && J==0)
            VARIABLE.DAC.value(i)  = ...
                -PARAMETER.R_R2R * PARAMETER.C_fb * ...
                F_val(i) * 2^( PARAMETER.M + VARIABLE.SCALING.tau - ...
                VARIABLE.ODE.e(i) - VARIABLE.SCALING.k(i) );
            while( abs(VARIABLE.DAC.value(i)) > 1-2^(-PARAMETER.N_S-1) )
                Change_tau( -1 )
            end
            [VARIABLE.DAC.B(i), VARIABLE.DAC.S(:,i)] = ...
                Num_to_Mant(VARIABLE.DAC.value(i));
        end
    % APPROXIMATE R2R VALUES
    for j = 1:PARAMETER.n
        if strcmpi(TYPE, 'all') || ( I==i && J==j )
            VARIABLE.R2R.value(i,j) = ...
```

```
                -PARAMETER.R_R2R * PARAMETER.C_fb * ...
                DF_val(i,j) * 2^( VARIABLE.SCALING.tau - ...
                VARIABLE.ODE.e(i) - VARIABLE.SCALING.k(i) + ...
                VARIABLE.ODE.e(j) + VARIABLE.SCALING.k(j) );
            while( abs(VARIABLE.R2R.value(i,j)) > 1-2^(-PARAMETER.N_S-1) )
                Change_tau( -1 )
            end
            [VARIABLE.R2R.B(i,j), VARIABLE.R2R.S(:,i,j)] = ...
                Num_to_Mant(VARIABLE.R2R.value(i,j));
        end
    end
    end
    Find_Max_tau;
return;

%% MODIFY TIME SCALE (TAU) FACTOR ------------------------------------
function []= Change_tau( delta )
global PARAMETER VARIABLE
    VARIABLE.SCALING.tau = VARIABLE.SCALING.tau + delta;
    for i=1:PARAMETER.n
        VARIABLE.DAC.value(i) = VARIABLE.DAC.value(i)*2^(delta);
        [VARIABLE.DAC.B(i), VARIABLE.DAC.S(:,i)] = ...
            Num_to_Mant(VARIABLE.DAC.value(i));
        for j=1:PARAMETER.n
            VARIABLE.R2R.value(i,j) = VARIABLE.R2R.value(i,j)*2^(delta);
            [VARIABLE.R2R.B(i,j), VARIABLE.R2R.S(:,i,j)] = ...
                Num_to_Mant(VARIABLE.R2R.value(i,j));
        end
    end
return;

%% MODIFICATIONS FROM EXPONENT CHANGES --------------------------------
function [] = Change_exp(delta, N)
global PARAMETER VARIABLE
    VARIABLE.DAC.value(N) = VARIABLE.DAC.value(N)*2^(-delta);
    while( abs(VARIABLE.DAC.value(N)) > 1-2^(-PARAMETER.N_S-1) )
        Change_tau( -1 )
    end
    [VARIABLE.DAC.B(N), VARIABLE.DAC.S(:,N)] = ...
        Num_to_Mant(VARIABLE.DAC.value(N));
        for j=1:PARAMETER.n
            VARIABLE.R2R.value(N,j) = VARIABLE.R2R.value(N,j)*2^(-delta);
            while( abs(VARIABLE.R2R.value(N,j)) > 1-2^(-PARAMETER.N_S-1) )
                Change_tau( -1 )
            end
            [VARIABLE.R2R.B(N,j), VARIABLE.R2R.S(:,N,j)] = ...
                Num_to_Mant(VARIABLE.R2R.value(N,j));
            VARIABLE.R2R.value(j,N) = VARIABLE.R2R.value(j,N)*2^(delta);
            while( abs(VARIABLE.R2R.value(j,N)) > 1-2^(-PARAMETER.N_S-1) )
                Change_tau( -1 )
            end
            [VARIABLE.R2R.B(j,N), VARIABLE.R2R.S(:,j,N)] = ...
                Num_to_Mant(VARIABLE.R2R.value(j,N));
        end
return

%% SET TO MAXIMUM TIME SCALE FACTOR -------------------------------------
function []= Find_Max_tau()
global PARAMETER VARIABLE
    val_DAC = max(abs(VARIABLE.DAC.value));
    val_R2R = max(max(abs(VARIABLE.R2R.value)));
    while(  max(val_DAC, val_R2R) < ( 0.5 - 2^(-PARAMETER.N_S)) )
        Change_tau(+1);
        val_DAC = max(abs(VARIABLE.DAC.value));
        val_R2R = max(max(abs(VARIABLE.R2R.value)));
    end
return;


%% CONVERT NUMBER INTO A MANTISSA (DAC/R2R) -----------------------------
function [B, S] = Num_to_Mant(value)
global PARAMETER
    B = sign(value) == -1;
    value = abs(value);
    for i=1:PARAMETER.N_S
        if value >= 2^(-i)-2^(-PARAMETER.N_S-1);
            value = value-2^(-i);
            S(i,1) = (1==1);
        else
            S(i,1) = (1==0);
        end
    end
return

%% CONVERTS MANTISSA INTO NUMBER (DAC/R2R) -------------------------------
function [value] = Mant_to_Num(B, S)
  global PARAMETER
    value = 0;
    for i=1:PARAMETER.N_S
        value = value + S(i)*2^(-i);
    end
        value = value*((-1)^D_B);
return
```

## FILE: FCNS_floatingPoint.m

```
function [] = FCNS_floatingPoint(varargin)
%   FCNS_floatingPoint('FP_X', i, X)
%   FCNS_floatingPoint('FP_Z', i, B, M, E)
%   FCNS_floatingPoint('FP_N', i, b, m, e)
    global PARAMETER VARIABLE         %   Contains most Parameters

%% Floating Point Update w/ "X"
    if strcmpi(varargin{1},'FP_X')
        i = varargin{2};
        [N_b, N_m, N_e, Z_B, Z_M, Z_E] = ...
            Num_2_FP( varargin{3} );
                VARIABLE.ODE.X(i,1) = varargin{3};
                VARIABLE.ODE.b(i,1) = N_b;
                VARIABLE.ODE.m(i,1) = N_m;
                VARIABLE.ODE.e(i,1) = N_e;
                VARIABLE.ODE.B(i,1) = Z_B;
                VARIABLE.ODE.M(:,i) = Z_M;
                VARIABLE.ODE.E(:,i) = Z_E;

%% Floating Point Update w/ "(B,M,E)"
    elseif strcmpi(varargin{1},'FP_Z')
        i = varargin{2};
        B = varargin{3}; M = varargin{4}; E = varargin{5};
        [b, m, e, X] = Dig_to_Num(B, M, E)
            VARIABLE.ODE.B(i,1) = B;
            VARIABLE.ODE.M(:,i) = M;
            VARIABLE.ODE.E(:,i) = E;
            VARIABLE.ODE.X(i) = X;
            VARIABLE.ODE.b(i,1) = b;
            VARIABLE.ODE.m(i,1) = m;
            VARIABLE.ODE.e(i,1) = e;

%% Floating Point Update w/ "(b,m,e)"
    elseif strcmpi(varargin{1},'FP_N')
    i = varargin{2};
    b = varargin{3}; m = varargin{4}; e = varargin{5};
    X = b*m*2^e;
    [N_b, N_m, N_e, Z_B, Z_M, Z_E] = ...
        Num_2_FP( X );
        VARIABLE.ODE.X(i) = X;
        VARIABLE.ODE.b(i,1) = b;
        VARIABLE.ODE.m(i,1) = m;
        VARIABLE.ODE.e(i,1) = e;
        VARIABLE.ODE.B(i,1) = Z_B;
        VARIABLE.ODE.M(:,i) = Z_M;
        VARIABLE.ODE.E(:,i) = Z_E;
    end

%% FUNCTION
function [ b, m, e, Z_B, Z_M, Z_E ] = Num_2_FP( X )
    global PARAMETER  % NOTE: BIT 1 is the LSB
b = sign(X);   Z_B = (b == -1);     X = abs(X);
if( X > (2-2^(-PARAMETER.M))*2^(2^(PARAMETER.E-1)-1) )% Inf/NaN:
    Z_E = ones(PARAMETER.E,1)==ones(PARAMETER.E,1);
    e = NaN; m = NaN;
    Z_M = ones(PARAMETER.M,1)==zeros(PARAMETER.M,1);
elseif( X < 2^(-2^(PARAMETER.E-1)-PARAMETER.M+2))% ZERO
    Z_E = ones(PARAMETER.E,1)==zeros(PARAMETER.E,1);
    e = -2^(PARAMETER.E-1) + 2;
    Z_M = ones(PARAMETER.M,1)==zeros(PARAMETER.M,1);
    m=0;
elseif( X < 2^(-2^(PARAMETER.E-1) + 2))% DENORMALIZED
    Z_E = ones(PARAMETER.E,1)==zeros(PARAMETER.E,1);
    e = -2^(PARAMETER.E-1) + 2;
    m = X*2^(e);   X = m;
    for i=PARAMETER.M:-1:1
        if( X >= 2^(-PARAMETER.M+i-1) )
            Z_M(i,1) = (1==1);
            X = X - 2^(-PARAMETER.M+i-1);
        else
            Z_M(i,1) = (1==0);
   end, end
else% Standard form
    e = floor(log2(X));
    e_temp = e +2^(PARAMETER.E-1)-1;
    m = X*2^(-e);
    m_temp = m-1;
    for i = PARAMETER.E:-1:1
        if( e_temp >= 2^(i-1))
            Z_E(i,1)=(1==1);
            e_temp = e_temp - 2^(i-1);
        else, Z_E(i,1)=(1==0);
    end, end
    for i = PARAMETER.M:-1:1
        if( m_temp >= 2^(i-PARAMETER.M-1) )
            Z_M(i,1)=(1==1);
            m_temp = m_temp - 2^(i-PARAMETER.M-1);
        else, Z_M(i,1)=(1==0);
 end, end, end,
return;

%% FUNCTION
function [b, m, e, X] = Dig_to_Num(Z_B, Z_M, Z_E)
    global PARAMETER  % NOTE: BIT 1 is the LSB
    b = ( -1 )^ Z_B;
    if( max(Z_E)==0 )% DENORMALIZED
        m = sum(Z_M'.*2.^[-PARAMETER.M:-1]);
        e = -2^(PARAMETER.E-1)+2;
        X = b*m*2^e;
    elseif( min(Z_E)==1 ) && ( min(Z_M)==1 )% INFINITY
        m = inf;    e = inf;    X = inf;
    elseif( min(Z_E)==1 )% NaN
        m = NaN;    e = NaN;    X = NaN;
    else% NORMALIZED
        m = 1+sum(Z_M'.*2.^[-PARAMETER.M:-1]);
        e = -2^(PARAMETER.E-1) + 1 + sum(Z_E'.*2.^[0:PARAMETER.E-1]);
        X = b*m*2^e;
    end
return;
```

# Bibliography

[1] Bryant, M.D. (2007). Private Conversations, Department of Mechanical Engineering, The University of Texas at Austin.

[2] Bryant, Yan, Tsang, Fernandez, Kumar (2006). “A Mixed Signal (Analog-Digital) Integrator Design,”

[3] Fernandez, B (2007). Private Conversations, Department of Mechanical Engineering, The University of Texas at Austin.

[4] Horowitz, P., Hill, W. (2001). *The Art of Electronics.* Cambridge University Press.

[5] IEEE Std 754 (1985). “IEEE Standard for Binary Floating-Point Arithmetic,” http://www.ieee.org.

[6] Remy, Brian (1995). “Mixed Signal Nonlinear Function Approximation and Inline Cell Layout Solution,”Master of Science thesis, The University of Texas at Austin

# Vita

Scott Bannert was born on September 30, 1982, in Sweeny, Texas, a small town in Brazoria County. He graduated from Sweeny Highschool in May 2001, thereafter enrolled at the University of Texas in Austin in the Fall 2001. In December of 2004, he received his Bachelors of Science in Mechanical Engineering with high honors. Subsequently to graduation, he began working on a graduate degree in Austin.

Permanent Address: 1617 Azalea Street
Sweeny, Tx 77480

This masters report was typeset with $\LaTeX\,2_\varepsilon$[1] by the author.

[1] $\LaTeX\,2_\varepsilon$ is an extension of $\LaTeX$. $\LaTeX$ is a collection of macros for $\TeX$. $\TeX$ is a trademark of the American Mathematical Society. The macros used in formatting this masters report were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.