Time-to-Digital Converter Architecture with Residue Arithmetic and its FPGA Implementation

Congbing Li  Kentaroh Katoh  Junshan Wang  Shu Wu  Shaiful Nizam Mohyar  Haruo Kobayashi
Division of Electronics and Informatics, Gunma University 1-5-1 Tenjin-cho Kiryu Gunma 376-8515 Japan
Phone: 81-277-30-1789  fax: 81-277-30-1707  e-mail: t13802483@gunma-u.ac.jp
Tsuruoka National College of Technology, Japan  email: k-katoh@tsuruoka-nct.ac.jp

Abstract
This paper describes a time-to-digital converter (TDC) architecture with residue arithmetic or Chinese Remainder theorem. It can reduce the hardware and power significantly compared to a flash type TDC while keeping comparable performance. Its FPGA implementation and measurement results show the effectiveness of our proposed architecture.

Keywords-  Timing Measurement, Time to Digital Converter, Residue, Chinese Remainder Theorem, FPGA

Introduction
A Time-to-Digital-Converter (TDC) measures the time interval between two edges, and time resolution of several picoseconds can be achieved when the TDC is implemented with an advanced CMOS process. TDC applications include phase comparators of all-digital PLLs, sensor interface circuits, modulation circuits, demodulation circuits, as well as TDC-based ADCs. The TDC will play an increasingly important role in the nano-CMOS era, because it is well suited to implementation with fine digital CMOS processes. [1,2,3].

There are various kinds of TDC circuits, and here we focus on a flash-type TDC (Fig.1) [1]. It uses a delay line which consists of CMOS inverter buffer delays. Based on this flash-type TDC, we will introduce a new type TDC---Residue Arithmetic TDC to reduce the hardware and power significantly compared to a flash-type TDC while keeping comparable performance. Then we have implemented it on an FPGA to verify the operation and performance.

Residue Arithmetic
Suppose that m1,...,mr are positive integers and coprime each other. Then there is unique positive integer x for given integers (a1,...,ar) which satisfy the following:

\[ x \equiv a_k \pmod{m_k}, \quad k = 1, 2, \ldots, r \]

where \(0 \leq a_k < m_k\), \(0 \leq x < N\) (\(N = m_1 \cdot m_2 \cdot \ldots \cdot m_r\)). Table I shows the case of \(m_1 = 2, m_2 = 3, m_3 = 5\) and \(N = 2 \times 3 \times 5 = 30\), and we see that each k is mapped to residues of \((m_1, m_2, m_3)\) one to one [3,4].

<table>
<thead>
<tr>
<th>(m_1)</th>
<th>(m_2)</th>
<th>(m_3)</th>
<th>(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>14</td>
</tr>
</tbody>
</table>

Table I. An integer k and residues of \((m_1, m_2, m_3)\)

<table>
<thead>
<tr>
<th>(m_1)</th>
<th>(m_2)</th>
<th>(m_3)</th>
<th>(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>29</td>
</tr>
</tbody>
</table>

Residue Arithmetic TDC Architecture
We consider to use this residue arithmetic for TDC implementation, because obtaining the residue is relatively easy for time signal (used in TDC design) while it is difficult for voltage signal. (used in ADC design). Fig.2 shows the proposed residue arithmetic TDC in the case of \(m_1 = 2, m_2 = 3, m_3 = 5\) and \(N = 2 \times 3 \times 5 = 30\), where the residues \(a \pmod{2}, b \pmod{3},\) and \(c \pmod{5}\) are encoded.
(mod 3), c (mod 5) are obtained with ring oscillators.

Note that the proposed TDC uses only 10 delay cells and 10 flip-flops (because 2+3+5=10) while the corresponding flash TDC requires 30 delay cells and 30 flip-flops; in general, the proposed TDC uses M delay cells and M flip-flops (where M = m1+m2+···mr) while the corresponding flash-type TDC uses N delay cells and N flip-flops (where N = m1m2···mr), and hence the circuit and power reduction of the proposed TDC can be significant for a large N with M << N compared to the flash TDC.

**FPGA Implementation**

We have implemented our proposed TDC with an FPGA (Fig.3) [5, 6, 7], and Table II and Fig.4 show its measurement results. We see that the proposed TDC works with good linearity as expected.

**Conclusion**

This paper describes residue arithmetic TDC implementation, and the measurement results verify its operation principle.

**Acknowledgements**

We would like to thank STARC which supports this project.

**References**


---

### Table II Measurement results of the proposed TDC.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>30.30</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>60.60</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>90.90</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>121.20</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>151.50</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>181.80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>212.10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>242.40</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>27</td>
<td>272.70</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>30</td>
<td>303.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>33</td>
<td>333.30</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>36</td>
<td>363.60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>39</td>
<td>393.90</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>42</td>
<td>424.20</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>45</td>
<td>454.50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>48</td>
<td>484.80</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>51</td>
<td>515.10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>54</td>
<td>545.40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>57</td>
<td>575.70</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>60</td>
<td>606.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>63</td>
<td>636.30</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>66</td>
<td>666.60</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>69</td>
<td>696.90</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>72</td>
<td>727.20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>75</td>
<td>757.50</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>78</td>
<td>787.80</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>81</td>
<td>818.10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>84</td>
<td>848.40</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>87</td>
<td>878.70</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>29</td>
</tr>
</tbody>
</table>

![Fig.3 Proposed TDC implementation on FPGA.](image-url)
Frequency Estimation Sampling Circuit Using Analog Hilbert Filter and Residue Number System

Yudai Abe, Shogo Katayama, Congbing Li, Anna Kuwana, Haruo Kobayashi

Division of Electronics and Informatics
Gunma University
OUTLINE

1. Research Background and Goal
2. Chinese Remainder Theorem
3. Proposed Waveform Sampling Circuit
4. Simulation Verification
5. Summary and Challenge
OUTLINE

1. Research Background and Goal
2. Chinese Remainder Theorem
3. Proposed Waveform Sampling Circuit
4. Simulation Verification
5. Summary and Challenge
Research Background

Next Generation Communication System “5G”

- High frequencies in communication systems
- Electronic components for high frequency bands

Communication speed:

- 1G: 2.4kbps
- 2G
- 3G
- 3.5G
- 3.9G
- 4G
- 5G: Higher than 10Gbps

Timeline:
- 1980
- 1990
- 2000
- 2010
- 2020
Our Research Goal

Estimate high-frequency input signal with multiple low-frequency clock sampling circuits

High-frequency sampling circuit is difficult to realize

Our Approach:

Sampling high frequency signal with multiple low frequency clocks

Use **Aliasing** proactively

**Analog Hilbert filter and residue number system**
1. Research Background and Goal
2. Chinese Remainder Theorem
3. Proposed Waveform Sampling Circuit
4. Simulation Verification
5. Summary and Challenge
Chinese arithmetic book ‘Sun Tzu calculation’
孫子算経

“When dividing by 3, its residue is 2, dividing by 5, its residue is 3, dividing by 7, its residue is 2. What is the original number?”

Answer 23

Sun Tzu calculation

Generalization

Chinese Remainder Theorem
How to use the Chinese remainder theorem

He used to quickly find out how many soldiers there are.

"Divide into 3 people."

Sun Tzu

...
How to use the Chinese remainder theorem

He used to quickly find out how many soldiers there are.

Sun Tzu

“Divide into 5 people.”

Remainder : 2
How to use the Chinese remainder theorem

He used to quickly find out how many soldiers there are.

Sun Tzu

“Divide into 7 people.”

Remainder : 3
How to use the Chinese remainder theorem

He used to quickly find out how many soldiers there are.

“Divide into 27 people. In all.”

Remainder : 2
Example of Residue Number System

- **Natural numbers**
  3, 5, 7 (relatively prime)
  \[ N = 3 \times 5 \times 7 = 105 \]

- **k** (0 ≤ k ≤ N-1 (=104))

\[
\begin{align*}
23 \mod 3 &= 2, & 23 \mod 5 &= 3, & 23 \mod 7 &= 2
\end{align*}
\]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>6</td>
<td>27</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>29</td>
</tr>
</tbody>
</table>

\( k \leftrightarrow (a, b, c) \)  
One to one  
Chinese remainder theorem  
Residue number system
1. Research Background and Goal
2. Chinese Remainder Theorem
3. Proposed Waveform Sampling Circuit
4. Simulation Verification
5. Summary and Challenge
Aliasing Phenomenon

Waveform frequency: 31 kHz

Sampling frequency: 8 kHz

FFT

Spectrums are folded within the sampling frequency band (sampling theorem)

Residue frequency (7 is the remainder of 31 divided by 8)
Complex FFT of $j \times \sin(2\pi f_{in} t)$

Complex FFT
Input frequency : 31 kHz
Sampling frequency : 8 kHz

$\cos(2\pi f_{in} t)$

$j \times \sin(2\pi f_{in} t)$

Inverted spectrum
anti-symmetric at Nyquist frequency
Complex FFT of $\cos(2\pi f_{in}t) + j \times \sin(2\pi f_{in}t)$

- Complex FFT
  - Input frequency: 31 kHz
  - Sampling frequency: 8 kHz

- Frequency spectrum:
  - 1 kHz, 4 kHz, 7 kHz, 8 kHz

- Process:
  1. Invert frequency
  2. Remove frequency

- Extract spectrum of the residual frequency

$\cos(2\pi f_{in}t) + j \times \sin(2\pi f_{in}t)$
How Generate $j \times \sin(2\pi f_{in} t)$

Use Analog Hilbert filter

RC polyphase filter

$\text{I}_{in} = \cos(\omega t)$

$\text{I}_{out} = A \cos(\omega t + \theta)$

$\text{Q}_{out} = A \sin(\omega t + \theta)$

$\text{Q}_{in} = 0$

Generate in-phase and quadrature waves from a single cosine wave
Proposed Sampling Circuit

Hilbert Filter

\( \cos(2\pi f_{in} t) \)

\( f_{in} \) (Unknown)

\( A \cos(2\pi f_{in} t + \theta) \)

\( A \sin(2\pi f_{in} t + \theta) \)

RC Polyphase Filter

Sampling circuit

Sampling frequency

\( f_{s1} \)

Complex FFT Power spectrum

\( f_{res1} \)

Residue number system

Estimate \( f_{in} \)

Generate in-phase signal I

Generate quadrature signal Q

Sampling frequencies: relatively prime

Residue frequencies

Complex FFT Power spectrum

\( f_{res2} \)

\( f_{res3} \)
OUTLINE

1. Research Background and Goal
2. Chinese Remainder Theorem
3. Proposed Waveform Sampling Circuit
4. Simulation Verification
5. Summary and Challenge
Simulation Settings

Complex FFT

- Input frequency: 12 GHz
- Frequency resolution: 1 kHz
- Sampling frequency: 229 kHz, 233 kHz, 239 kHz
  (Relatively prime)
- Range of measurement: 0~2080622 kHz
  (Note: $229 \times 233 \times 239 = 2080623$)

Measurement at 20 GHz using sampling frequencies of $\approx 200$ kHz
Simulation Results

Complex FFT: \( \cos(2\pi f_{in}t) + j \times \sin(2\pi f_{in}t) \)

- Input frequency: 12 GHz
- Frequency resolution: 1 kHz
- Sampling frequency: 229 kHz, 233 kHz, 239 kHz

Residue frequency:
- 229 kHz Sampling: 171 kHz
- 233 kHz Sampling: 34 kHz
- 239 kHz Sampling: 49 kHz
Frequency Estimation by Residue Number System

Residue frequencies
171 kHz, 34 kHz, 49 kHz

Input frequency estimation using residue frequencies and residue number system

Estimate input frequency 12GHz
Simulation Result Overview

Estimate unknown input frequency

Estimate $f_{in} = 12\text{GHz}$

$A \cos(2\pi 12Gt + \theta)$

$\cos(2\pi 12Gt)\quad \text{RC Polyphase Filter}$

$\cos(2\pi 12Gt + \theta)$

$\sin(2\pi 12Gt + \theta)$

Sampling circuit

Sampling frequency

Complex FFT Power spectrum

复杂 FFT 功率谱

Residue number system

$Re_1\quad Im_1\quad 171\text{kHz}$

$Re_2\quad Im_2\quad 34\text{kHz}$

$Re_3\quad Im_3\quad 49\text{kHz}$

$229\text{kHz}$

$233\text{kHz}$

$239\text{kHz}$

Hilbert Filter
OUTLINE

1. Research Background and Goal
2. Chinese Remainder Theorem
3. Proposed Waveform Sampling Circuit
4. Simulation Verification
5. Summary and Challenge
Summary and Challenge

Summary

● Proposed a method to estimate high-frequency signal using multiple low-frequency sampling circuits.

● Confirmed its operation by theory and simulation.

● Measurable range is wide: proportional to multiplication of multiple sampling frequencies.

Challenge

● Estimated input frequency is discrete

Consider estimation with fine frequency resolution
Thank you for your attention
A Gray Code Based Time-to-Digital Converter Architecture and its FPGA Implementation

Congbing Li       Haruo Kobayashi
Gunma University
Outline

• Research Objective & Background
• Flash TDC and Problems
• Gray Code
• Gray Code TDC Architecture
• FPGA Implementation
• RTL Verification of Glitch-free Characteristic
• Conclusion
Research Objective

Objective

● Development of Time-to-Digital Converter (TDC) architecture with high-speed and small hardware

Approach

● Utilization of Gray code
TDC plays an important role in nano-CMOS era

Voltage-domain resolution facing difficulties due to reduced supply voltage

Time-domain resolution becoming superior

TDC measures time interval between two signal transitions, into digital signal.
(widely used in ADPLLs, jitter measurements, time-domain ADC)
Flash TDC

- Digital output (Dout) proportional to time difference between rising edges (T)
- Time resolution $\tau$

[Diagram showing the Flash TDC circuit and waveforms for different Dout values with time resolution $\tau$.]

- Dout=2
- Dout=1
- Dout=0

$D_0 = 1$
$D_1 = 1$
$D_2 = 1$
$D_3 = 0$
Problems of Flash TDC

An n-bit flash TDC with $2^n$-quantization levels

**Advantages**
- High-speed timing measurement
- Single-event timing measurement
- All digital implementation

**Disadvantages**
- $2^n-1$ delay elements, $2^n-1$ Flip-Flops
- n-bit thermometer-to-binary code encoder

Large circuits
High power consumption
Gray Code

a binary numeral system where two successive values differ in only one bit

Table. 4-bit Gray Code

<table>
<thead>
<tr>
<th>Decimal numbers</th>
<th>Binary Code</th>
<th>Gray Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0111</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0101</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1100</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1101</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>1111</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>1110</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>1010</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>1011</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>1001</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>1000</td>
</tr>
</tbody>
</table>

- For Gray code, between any two adjacent numbers, only one bit changes at a time
- Gray code data is more reliable compared with binary code
In a ring oscillator, between any two adjacent states, only one output changes at a time. This characteristic is very similar to Gray code.

For any given Gray code, its each bit can be generated by a certain ring oscillator.
A Gray code TDC architecture can be conceived by grouping a few ring oscillators.

Figure. Proposed 6-bit Gray code TDC
Flash vs. Proposed TDCs for a measurement range of $2^6$

<table>
<thead>
<tr>
<th></th>
<th>Flash TDC</th>
<th>Proposed TDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of delay elements</td>
<td>64</td>
<td>62</td>
</tr>
<tr>
<td>Number of Flip-flop</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>The maximum stage</td>
<td>64</td>
<td>32</td>
</tr>
</tbody>
</table>

for a measurement range of $2^n$ significant hardware reduction as # of bits increases.
FPGA Implementation (1/3)

Proposed TDC implementation on Xilinx FPGA

Note: ADC is difficult to implement with full digital FPGA.
FPGA Implementation (2/3)

Measurement results of the proposed TDC with FPGA (6-bit case).

Proposed TDC operation is confirmed with FPGA evaluation.
Similarly, 8-bit Gray code TDC architecture was implemented on FPGA.

Measurement results of the proposed TDC with FPGA (8-bit case)

Linear characteristics
RTL Verification of Glitch-free Characteristic (1/2)

- The proposed Gray code TDC can provide a glitch-free binary code sequence even there are mismatches between the delay stages.
- RTL simulation was conducted to verify this characteristic.
RTL Verification of Glitch-free Characteristic (2/2)

- RTL simulation result shows that no matter there are mismatches among the delay stages or not, the proposed Gray code TDC can always output a glitch-free binary code sequence.
# Conclusion

We have proposed a gray code based TDC architecture

- Comparable performance to Flash TDC
- Significant hardware & power reduction

for a measurement range of $2^n$

<table>
<thead>
<tr>
<th></th>
<th>Flash TDC</th>
<th>Proposed TDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of delay elements</td>
<td>$2^n$</td>
<td>$2^n - 2$</td>
</tr>
<tr>
<td>Number of Flip-flop</td>
<td>$2^n$</td>
<td>$n$</td>
</tr>
<tr>
<td>The maximum stage</td>
<td>$2^n$</td>
<td>$2^{n-1}$</td>
</tr>
</tbody>
</table>

Significant hardware reduction as # of bits increases.

We have implemented the proposed TDC with FPGA

Confirmed its operation
Gray-Code Input DAC Architecture for Clean Signal Generation


Gunma University, Socionext Inc.
OUTLINE

• Research Background • Objective
• Glitches
• Gray-code
• Gray-code Input DAC Architecture and Operation
• Simulation Verification by SPICE
• Conclusion
OUTLINE

• Research Background • Objective
  • What are Glitches
  • Gray-code
  • Gray-code Input DAC Architecture and Operation
  • Simulation Verification by SPICE
  • Conclusion
Research Background

Analog Input

ADC

Digital Signal Processing

DAC

Digital Input

D/A

S/H

Analog Filter

Analog Input

B₀

B₁

B₂

Bₙ₋₂

Bₙ₋₁

Bₙ
Research Background

Basic architecture of DAC

Current Source DAC  Capacitive DAC  Resistance DAC

The switch is driven with a binary code → glitch
Research Objective

Objective

- Design Digital-to-Analog Converter (DAC) architectures for clean signal generation

Approach

- By reducing glitches with Gray-Code input topologies
OUTLINE

- Research Background • Objective
- Glitches
  - Gray-code
  - Gray-code Input DAC Architecture and Operation
- Simulation Verification by SPICE
- Conclusion
What are Glitches

- Voltage spikes
- Reasons for glitches

<table>
<thead>
<tr>
<th>Decimal numbers</th>
<th>Natural Binary code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

The most significant bit (MSB) changes (near the middle point)

- When 7 → 8
  - 0111 → 0110 → 0100 → 0000 → 1000

- When 8 → 7
  - 1000 → 1001 → 1011 → 1111 → 0111
Generation of Glitch at Switching time

When the input changes $7 \rightarrow 8$
Generation of Glitch at Switching time

When B3 switches first

A big upward spike arises

Vout = 15IR

(1,1,1,1) = 15

(0,1,1,1) = 7
Generation of Glitch at Switching time

When B3 switches last

A big downward spike occur

Vout = 0

(0,1,1,1) = 7

(0,0,0,0) = 0
Generation of Glitch at Switching time

Input = 8

Vout = 8IR

(1,1,1,1) = 15
(1,0,0,0) = 8
(0,1,1,1) = 7
(0,0,0,0) = 0

glitch

x8I B3
x4I B2
x2I B1
x1I B0
Glitch Problem and Remedy

Effects of Glitch

- Serious deterioration of images, videos, sounds

Remedy

- Using high-order reconstruction filter
- Using track/hold circuitry at the DAC output
- Using Gray-Code input DAC topologies

Extra Space in IC, Expensive
OUTLINE

- Research Background • Objective
- What are Glitches
- Gray-code
  - Gray-code Input DAC Architecture and Operation
  - Simulation Verification by SPICE
- Conclusion
Gray-Code

Gray-Code → Alternative representation of binary code

Two adjacent number → Only one bit change

\[(G_n = B_{n+1} \oplus B_n)\]

Binary to Gray code conversion diagram

Binary to Gray code converter
Gray Code

Compare with **Binary code** and **Gray code**

<table>
<thead>
<tr>
<th>Decimal numbers</th>
<th>Binary Code</th>
<th>Gray Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0111</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0101</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1100</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1101</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>1111</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>1110</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>1010</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>1011</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>1001</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Binary code**
- Multiple bits change at a time
- Trigger more switches

Example.
- $1 \rightarrow 2$ --- $0001 \rightarrow 0010$ 2 bits change
- $7 \rightarrow 8$--- $0111 \rightarrow 1000$ all 4 bits change

**Gray code**
- Only one bit changes at a time
- Triggers one switch

Example.
- $1 \rightarrow 2$ --- $0001 \rightarrow 0011$ one bit change
- $7 \rightarrow 8$ --- $0100 \rightarrow 1100$ one bit change

**Less glitches**
OUTLINE

• Research Background • Objective
• What are Glitches
• Gray-code
• Gray-code Input DAC Architecture and Operation
• Simulation Verification by SPICE
• Conclusion
Gray-code Input DAC Architecture and Operation

1. Current-steering Gray-Code DAC
2. Charge-mode Gray-Code DAC
3. Voltage-mode Gray-Code DAC
Current/Voltage Switch Matrix

Switch is DPDT (Double-Pole Double-Throw)
1. Current-steering Gray-Code DAC

Conventional Binary-Weighted current-steering DAC

Gray-Code input current-steering DAC
Code Conversion

Code domain in Gray-code input current-steering DAC
A Gray-code input current-steering DAC (data=5)

Data=5

\[ I_{\text{out}} = (I_{\text{out}^+}) - (I_{\text{out}^-}) = -10I \]

\[ I_{\text{out}^-} = -I + 2I - 4I + 8I = 5I \]

\[ I_{\text{out}^+} = I - 2I + 4I - 8I = -5I \]
2. Charge-mode Gray-code DAC

A binary-weighted capacitor DAC

A Gray-code input charge-mode DAC
Sample Mode of a Gray-Code Input Charge-Mode DAC (data=5)
Sample Mode of a Gray-Code Input Charge-Mode DAC (data=5)

Data=5

\[ V_{out} = 5V_r \]
3. Voltage-mode Gray-Code DAC
A Gray-Code Input Voltage-mode DAC (data=5)

\[ V_{out+} = V_r + 4V_r = 5V_r \]
OUTLINE

• Research Background  • Objective
• What are Glitches
• Gray-code
• Gray-code Input DAC Architecture and Operation
• Simulation Verification by SPICE
• Conclusion
Simulation Verification by SPICE

1. Simulation of current-steering Gray-Code DAC

2. Simulation of charge-mode Gray-Code DAC

3. Simulation of voltage-mode Gray-Code DAC

4. Verification of glitch reduction
1. SPICE Realization of Current-Steering of Gray-Code Input

Diagram:

- Latch
- Gray-Code
- Binary-Code
- Subtractor Circuit
- S&H

Components:
- dpdt
- 1, 2I, 4I, 8I
- V_{out}
- I

Description:

The diagram illustrates a SPICE realization of a current-steering circuit that converts a Gray-code input into a binary code. The circuit includes a latch and a subtractor circuit, with current steering for each bit of the Gray code. The binary output is further processed by an S&H circuit.
1. Simulation of current-steering Gray-Code DAC

4bit Current-steering DAC

8bit Current-steering DAC
2. SPICE Realization of charge-mode of Gray-Code Input
2. Simulation of charge-mode Gray-Code DAC

- 4bit Charge-mode DAC
- 8bit Charge-mode DAC
3.SPICE Realization of Voltage-mode of Gray-Code Input
3. Simulation of Voltage-mode Gray-Code DAC

4bit Voltage-mode DAC

8bit Voltage-mode DAC
4. Verification of glitch reduction

Conventional Current-Steering DAC with switching delay (8bit)
4. Verification of glitch reduction

Current-Steering Gray-code input DAC with switching delay (8bit)
4. Simulation Result (Up Sweeping)

Conventional Current-Steering DAC vs. Current-steering DAC of Gray-code

- Conventional Current-Steering DAC
- Current-steering DAC of Gray-code

8-bit (MSB) glitch
4. Simulation Result (Down Sweeping)

- Conventional Current-Steering DAC
- Current-steering DAC of Gray-code

Comparison between Conventional Current-Steering DAC and Current-steering DAC of Gray-code.
4. Simulation Result (Random Switching Delay)

Conventional Current-Steering DAC VS. Current-steering DAC of Gray-code
OUTLINE

• Research Background • Objective
• What are Glitches
• Gray-code
• Gray-code Input DAC Architecture and Operation
• Simulation Verification by SPICE
• Conclusion
Conclusion

DAC Converter using **Binary code** input → **Glitch**

DAC Converter using **Gray code** input
deterioration

**Current-steering** Gray-Code DAC
**Charge-mode** Gray-Code DAC
**Voltage-mode** Gray-Code DAC

Glitch reduction
Final statement

• Coding method can lead to robust mixed-signal circuit design.

Gray code was invented by Frank Gray at Bell Lab in 1947.
Thank you for listening

谢谢
Study of Gray Code Input DAC for Glitch Reduction

*Adhikari Gopal  Richen Jiang  Haruo Kobayashi

Kobayashi Laboratory, Gunma University, Japan
Outline

- Research Objective
- Introduction to DAC
- Glitches
- Introduction to Gray Code
- Gray Code Input DAC
  - Switch Matrix Design
  - Voltage Mode Gray Code Input DAC
  - Current Steering Mode Gray Code Input DAC
- Conclusion
Research Objective

✓ Research and implement DAC for glitch reduction using Gray code input
  *(difficult to design)*

Approach

✓ Use MOSFETs for DAC design
✓ Utilization of Gray code input for glitch reduction
Introduction to DAC

- Convert digital signal to analog signal
- Signal to be recognized by human senses
- Widely used in signal processing
What are Glitches?

✓ Voltage spikes
✓ Reasons for glitches
  ◦ Capacitive coupling
  ◦ Differences in switching

➢ Glitch behaviour → Dominated by difference in switching
➢ Switching of MSB → Most significant glitches
  (Multiple switches changing states at once)
Glitch Problem and Remedy

Effects of Glitch

- Serious deterioration of images, videos and sounds

Remedy

- Using high-order reconstruction filter
- Using track/hold circuitry at the DAC output
- Using Gray code input DAC topologies

Extra Space in IC, Expensive
What is Gray code?

- Gray code → Alternative representation of binary code
- Two adjacent numbers → Only one bit change
- Reflected binary code

**Binary to Gray code conversion**

![Gray code conversion diagram](attachment:gray_code_conversion.png)
Gray Code versus Binary Code

**Binary code**  Multiple bits change at a time

Trigger more switches

Example.  $1 \rightarrow 2$ --- $0001 \rightarrow 0010$  2 bits change

$7 \rightarrow 8$--- $0111 \rightarrow 1000$  all 4 bits change

**Gray code**  Only one bit changes at a time

Triggers one switch

Example.  $1 \rightarrow 2$ --- $0001 \rightarrow 0011$ one bit change (B2)

$7 \rightarrow 8$--- $0110 \rightarrow 1100$  one bit change (B4)

Less glitches

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0111</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0101</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1100</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1101</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>1111</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>1110</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>1010</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>1011</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>1001</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>1000</td>
</tr>
</tbody>
</table>
Gray code versus Binary code Timing Diagram

Gray code timing diagram

Binary code timing diagram

Gray code → Only one bit changes at a time 0001→0011
Binary code → Multiple bits change at a time 0001→0010

Using Gray code → Less glitches expected to appear
Gray Code Input DAC
Switch Matrix Design

Switch is DPDT (Double-Pole Double-Throw)
Switch Matrix Operation

CTL → LOW:
- M3, M4 → ON, M1, M2 → OFF
- IN1 = OUT1, IN2 = OUT2

CTL → HIGH:
- M1, M2 → ON, M3, M4 → OFF
- IN1 = OUT2, IN2 = OUT1
Voltage Mode Gray Code Input DAC

- $IN1 = V_{\text{ref}}$
- $IN2 = 0$
- $CTL \leftarrow \text{Gray code input}$
- $OUT1, \; OUT2 \rightarrow \text{Connected with R-2R Ladder}$

$V_{\text{out}}(D) = V_{\text{ref}} \frac{|(2D - 1)|}{2^{n+1}}$

$n$ : number of bits

$D = 1, 2, 3...n+1$
## Voltage Mode Gray Code Input DAC

### SPICE Simulation Results

#### 4-bit case

<table>
<thead>
<tr>
<th>D</th>
<th>Bits</th>
<th>Vout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>3/32</td>
</tr>
<tr>
<td>2</td>
<td>0001</td>
<td>9/32</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>15/32</td>
</tr>
<tr>
<td>4</td>
<td>0010</td>
<td>21/32</td>
</tr>
<tr>
<td>5</td>
<td>0110</td>
<td>27/32</td>
</tr>
<tr>
<td>6</td>
<td>0111</td>
<td>33/32</td>
</tr>
<tr>
<td>7</td>
<td>0101</td>
<td>39/32</td>
</tr>
<tr>
<td>8</td>
<td>0100</td>
<td>45/32</td>
</tr>
<tr>
<td>9</td>
<td>1100</td>
<td>51/32</td>
</tr>
<tr>
<td>10</td>
<td>1101</td>
<td>57/32</td>
</tr>
<tr>
<td>11</td>
<td>1111</td>
<td>63/32</td>
</tr>
<tr>
<td>12</td>
<td>1110</td>
<td>69/32</td>
</tr>
<tr>
<td>13</td>
<td>1010</td>
<td>75/32</td>
</tr>
<tr>
<td>14</td>
<td>1011</td>
<td>81/32</td>
</tr>
<tr>
<td>15</td>
<td>1001</td>
<td>87/32</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>93/32</td>
</tr>
</tbody>
</table>
Voltage Mode Gray Code Input DAC

8-bit case

SPICE Simulation Results

NO GLITCHES
Voltage Mode Gray Code Input DAC
MOSFET Implementation

Aspect ratios W/L for R, 2R, 1.5R, 0.5R

\[ R = \frac{V_{DS}}{I_{dsat}} = \frac{V_{DS}}{u_n C_{ox} \times \frac{W}{L} \times (V_{GS}-V_{TH})^2} \]
Voltage Mode Gray Code Input DAC
MOSFET Implementation Simulation Results

4-bit case

8-bit case

NO GLITCHES
Current Steering Mode Gray Code Input DAC

Circuit Configuration

- IN1, IN2, intermediate stages → binary weighted current sources.
- Gray code alters the way the switches are triggered
- $I_{out} = I_{out}^+ - I_{out}^-$
For 1010 Gray code,
S3, S1 → ON, the other switches → OFF

\[ I_{out} = -I + 2I - 4I - 8I = -9I \]

\[ I_{out} = -I - 2I + 4I + 8I = 9I \]
Current Steering Mode Gray Code Input DAC
SPICE Simulation Results

4-bit case

8-bit case

NO GLITCHES
Current Steering Mode Gray Code Input DAC MOSFET Implementation

$M_2, M_3, M_4, M_5$ generate $I, \ 2I, \ 4I, \ 8I$ (current source)

$M_7, M_8, M_9, M_{10}$ generate $I, \ 2I, \ 4I, \ 8I$ (current sink)
Current Steering Mode Gray Code Input DAC
MOSFET Implementation Simulation Results

These glitches are due to mismatch of PMOS and NMOS

Glitches but not very significant
Glitches get introduced in binary code R-2R DAC

ICSICT 2016

Paper ID : S0346
R-2R Binary Current Mode DAC

4-bit case

8-bit case
Conclusion

✓ R-2R DACs prone to glitches → Multiple bits switching at a time.
✓ Claims of Gray code DACs being difficult to design
  but successfully designed and simulated
✓ Gray code Input DACs reduce glitches considerably
✓ No extra space needed for IC
✓ No extra circuit needed to remove glitches
Final Statement

Coding method can lead to robust mixed-signal circuit design.

Gray code was invented by Frank Gray at Bell Lab in 1947.
Thank you