“Flying-Adder” Frequency and Phase Synthesis Architecture

Liming XIU
Texas Instruments Inc, HPA/DAV

01/30/2005
What is it?

An novel frequency synthesis architecture that takes a digital value and generates a signal of requested frequency (and phase).
Background Material

This presentation is based on five papers:


History

- Being continuously refined/improved.
- Thanks to Hugh Mair
Presentation Outline

• The principal Idea

• Implementation: First Generation

• Implementation: Second Generation

• Integer-Flying-Adder Architecture
Principal Idea

Using multiple equally-spaced phases generated from a VCO to synthesis various frequency and phase, by triggering the flip-flops at predestined time.
Principal Idea, continued

VCO Output waveforms, for N=32

Continued
Principal Idea, continued

Triggering the flip-flop at predestined time to generate the desired frequency, by utilizing the multiple VCO outputs.

Continued
Numerical Example

VCO running at 156.25 MHz (6.4 ns)

=> \[ \Delta = \frac{6.4}{32} = 0.2 \text{ (ns)} \]

Wanted: 204.08 MHz, or \( T = 4.9 \text{ ns} \)

=> \[ \text{FREQ[9:0]} = \frac{T}{2\Delta} = \frac{4.9}{0.4} = 12.25 = 01100.01000b \]

Integer portion is used for selecting tick, fractional portion is for error accumulation.

Continued
Numerical Example, continued

\[
\begin{align*}
0 & \quad 12 & \quad 24 & \quad 4 & \quad 17 & \quad 29 \\
0000 & \quad 01100 & \quad 11000 & \quad 00100 & \quad 10001 & \quad 11101 \\
\end{align*}
\]
Key Facts

• VCO has to be in multiple-delay-stages style, single-ended or differential.

• The PLL/VCO is running at a fixed frequency, no loop dynamic responds requirement.

• Output frequency range, theoretically: \((1/2)f_{vco} \leq f_{out} \leq (N/2)f_{vco}\)

• In practice, the high-frequency is limited by the speed of the process in which this architecture is implemented.

• Has inherent jitter if fractional bits are used.

• Frequency resolution (step): \(\delta f = -2^{-k} \Delta f^2\)
Inherent Jitter

\[ T = FREQ \times \Delta \]

or \( FREQ = T / \Delta = M + r \)

\[ T_s = M \times \Delta \quad P_l = r \]

\[ T_l = (M + 1) \times \Delta \quad P_s = 1 - P_l = 1 - r \]

\[ J_{pk-pk} = T_l - T_s = \Delta \]

\[ J_{mean} = P_l (T_l - T) + P_s (T_s - T) = 0 \]

\[ J_{rms} = \sqrt{P_l (T_l - T)^2 + P_s (T_s - T)^2} = \Delta \sqrt{r - r^2} \]
Output frequency vs. FREQ (an example)
Frequency divider and “Phase divider”

• To generate frequencies, divider can be used. But divider ratio has to be integer → available frequencies are limited.

• “Flying-Adder” architecture can be viewed as “phase divider” which provides additional level of frequency divide → more available frequencies.
Presentation Outline

• The principal Idea

• Implementation: First Generation

• Implementation: Second Generation

• Integer-Flying-Adder Architecture
Two problems:

• The glitch of the MUX

• The speed of the adder
The Glitch of the MUX

IN0, “00000”  
IN21, “10101”  
IN31, “11111”  

Z  
Z’

t
Implementation: Two Paths

- FREQ_B<4:0>
- VCOOUT<31:0>
- FREQ_A<9:0>

PATH_B
- 5-bit Adder
- 5-bit Reg
- CLK1
- D
- Q'
- Q

PATH_A
- 10-bit Reg
- 10-bit Adder
- CLK2
- D
- Q'
- Q

Continued
Implementation: Two Paths

Solved the glitch problem: the two paths are interlocked

CLK1

Path_A blocked
MUX_A decoding

Path_B open
MUX_B stable

CLK2

Path_A open
MUX_A stable

Path_B blocked
MUX_B decoding

Continued
Implementation: Two Paths

Relaxed the constrain on adders => double the circuit speed
One path generates the rising edge, the other for falling edge

Path A

Path B

Accumulator in Path B

Path A

Path B

Accumulator in Path A
Implementation: Two Paths

This two paths architecture solved the previous two problems, but created a new problem:

the synchronization of the two paths.

In other words, MUX_A and MUX_B’s address values are unrelated => duty cycle is uncontrollable.
Implementation: Synchronized

FREQ_B<4:0>  
VCOOUT <31:0>  
FREQ_A<9:0>  
PATH_B  
PATH_A  
CLK1  
CLK2  
Z
Implementation: Synchronized

Now MUX_B’s address is related to MUX_A’s

New problem:  Adder in PATH_B doesn’t have full cycle to work

Accumulator in Path A
Implementation: Pipelined

- **PATH_A**:
  - 10-bit Adder
  - 10-bit Reg

- **PATH_B**:
  - 5-bit Adder
  - 5-bit Reg

Signals:
- FREQ<10:6>
- VCOOUT <31:0>
- CLK1
- CLK2
- t1, t2, t3, t4, t5, t6
- D, Q', Q
- Z

Continued
Implementation: Pipelined

- Now both the accumulator in PATH_A and the adder in PATH_B have full cycle to work.
- Timing constrain: see below

\[ t_1 + t_2 + t_3 \leq \Delta t_{ab} \]
\[ t_4 + t_5 + t_6 \leq \Delta t_{bc} \]
Implementation: First Generation

First generation development history:
- One Path
- Two Paths
- Synchronized
- Pipelined

Key features of this architecture:
- interlocking between paths
- self-clocking
- pipeline
Summary: The Advantages

• The output frequency can be changed **instantly** without any dynamic process.

• With enough fraction bits, any frequency within certain range can be generated with any accuracy.

• Phase shift version of the output signal can be generated.

• Output signal with various duty cycle can be generated.

• Since VCO running at fixed frequency, VCO and PLL design are much simplified, the PLL is much robust against temperature draft, process and voltage variation.

• The ‘increment’ value can be modulated to produce a highly accurate and predictable spread spectrum clock source.
Phase Synthesis: The idea

VCOOUT<31:0> → 32 to 1 MUX → 10-bit Reg → 10-bit Adder → 32 to 1 MUX → 5-bit Reg → 5-bit Adder → FREQ<9:0> → FREQ_GEN → Z

PHASE<4:0> → PHASE_GEN → Z_SHIFT

Continued
Phase Synthesis: The idea

- The **MUX address** used in PHASE_GEN is the sum of the MUX_A’s address and PHASE[4:0]
- The **data** used in DFF of PHASE_GEN is the same as data used in FREQ_GEN
- The Z_SHIFT is a delay version of Z. The delay amount: PHASE[4:0] * Δ

\[ \varphi = \text{PHASE}[4:0] \times \Delta \]

---

The diagram shows:
- \( Z \)
- \( Z_{\text{SHIFT}} \)
Phase Synthesis: Implementation
Phase Synthesis: Problems

Problems:
- “Dead-zone”
- “Dual-stability”
Presentation Outline

• The principal Idea

• Implementation: First Generation

• **Implementation: Second Generation**

• Integer-Flying-Adder Architecture
Second Generation Architecture

The new architecture:

- the operating speed is greatly improved.
- has scalability for higher output frequency.
- has an internal node whose frequency is higher than that of the synthesized output.
- eliminates the “dead-zone” and “dual-stability” for phase synthesis.

Continued
Second Generation Architecture

PATH_B

FREQ<10:6>

5-bit Adder

5-bit Reg

5-bit Reg

CLK1

VCOOUT<31:0>

Z

t3

t4

t2

t1

5-bit Reg

10-bit Reg

10-bit Reg

PATH_A

FREQ<9:0>

February 15, 2005

Continued
Sec. Gen. Arch.: Scalability

Continued
Sec. Gen. Arch.: Scalability

- Multiple paths (more than two) to relax the constrains on adders further -> higher output frequency
Sec. Gen. Arch.: Scalability

- The clocks signals and the mechanism of interlocking

<table>
<thead>
<tr>
<th>TRIGGER</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Z and SEL5[1:0]</th>
<th>01</th>
<th>11</th>
<th>10</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
<th>00</th>
<th>01</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MUX2</td>
<td>MUX3</td>
<td>MUX4</td>
<td>MUX1</td>
<td>MUX2</td>
<td>MUX3</td>
<td>MUX4</td>
<td>MUX1</td>
<td>MUX2</td>
<td>MUX3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLK1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLK2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLK3</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLK4</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sec. Gen. Arch.: Phase Synthesis

Fig. 16. The circuitry for Z.

Fig. 17. The circuitry for Z_SHIFT
Presentation Outline

• The principal Idea

• Implementation: First Generation

• Implementation: Second Generation

• Integer-Flying-Adder Architecture
Integer-Flying-Adder Architecture

Issues with current architecture:

since PLL/VCO is running at a fixed frequency =>

- need fractional bits to achieve certain frequency, -> periodic carry-in bit,

- frequency modulation of the output signal, or, inherent jitter

Continued
Integer-Flying-Adder Architecture

Idea:

Make PLL programmable
Get ride of fractional bit

→ Eliminate the inherent jitter
Integer-Flying-Adder: Method

\[ FREQ = \frac{T}{\Delta} = \frac{1}{(f* \Delta)} = \left(\frac{f_{in} * N}{f* P}\right) * M \]

Using two integers, \( FREQ \) and \( M \), to approximate a real number \( f \).

\[ 2 \leq FREQ \leq 2N, \quad M1 \leq M \leq M2 \]
Integer-Flying-Adder: Algorithm

The algorithm to search the best control parameters

\[
\text{error}_{\text{min}} = \text{very\_big\_number} \\
\text{for ( } M_1 \leq M \leq M_2 \text{ ) } \{ \\
\text{freq} = \left\lfloor \frac{\text{fin} \times N}{f \times P} \right\rfloor \times M \\
\text{error} = \min( \text{freq} - \text{floor}(\text{freq}), \text{ceiling}(\text{freq}) - \text{freq} ) \\
\text{if (error < error}_{\text{min}} \text{) } \{ \\
\text{error}_{\text{min}} = \text{error} \\
M_{\text{best}} = M \\
\text{if (freq} - \text{floor}(\text{freq})) < 0.5 \text{ } \{ \\
\text{FREQ} = \text{floor}(\text{freq}) \\
\} \\
\text{else } \{ \\
\text{FREQ} = \text{ceiling}(\text{freq}) \\
\} \\
\} \\
\} 
\]
Integer-Flying-Adder: Error Upper-Bound

\[ |T-T'|/T = r*\Delta/T \]
\[ \leq (1/2) * ((\text{fin}*N)/(f*P)) / (((\text{fin}*N)/(f*P))*M) \]
\[ = 1/(2*M) \]
\[ \leq 1/(2*M_1) \]
Integer-Flying-Adder: Error Distribution Envelope

for (2<=F<=64) {
   for (M1<=M<=M2) {
      F-M-seq(index) = M/F
   }
}

foreach M/F in F-M-sorted-seq(index) {
   F-M_curr = M/F
   p_max = 2/(F-M_curr + F-M_prev)
   e_max = (F-M_curr - F-M_prev)/( F-M_curr + 1)
   F-M_prev = F-M_curr
}

See paper on TCASII (3th paper) for mathematical prove

Continued
Integer-Flying-Adder: Error Distribution Envelope

Frequency Error Distribution

Frequency (MHz)

Frequency Error

499.9 499.95 500 500.05 500.1 500.15 500.2

499.957 500.0157 500.044 500.073 500.127 500.182
Integer-Flying-Adder: Error Distribution Envelope

The effect of $M2$ on the error distribution envelope
Integer-Flying-Adder: Summary

- Comparing to original architecture:
  - eliminate the inherent jitter
  - but the PLL loop need adjustment
- Comparing to “Integer-N”, the frequency range is much wider.
- Comparing to “Fractional-N”, no need to compensate the spurious signals.
One Application Example: All Digital Phase Lock Loop

“flying-adder” synthesizer

All loop variables are digital values, *no analog voltage!*
ADPLL: A New Idea

**Diagram:**

- **Known high frequency** $f_{hi}$
- **Input frequency** $f_{in}$
- **Output frequency** $f_{out}$

**Components:**
- **Measure Frequency**
- **Conversion**
- **VCO**
- **Synthesizer 1**
- **Synthesizer 2**
- **FREQ 1**
- **FREQ 2**

**Equations:**
- $f_{hi}$
- $f_{in}$
- $f_{out}$
- $f_{in}$ frequency
- $f_{out}$ frequency

**Key Ideas:**
- Synthesizers
- VCO
- Flying-adder
- Frequency conversion
ADPLL: A New Idea

- **Goal:** \( fout = N \times fin \)
- **Procedure:**
  - Using synthesizer1 to generate a known high frequency \( fhi \) (e.g. > 500 MHz), by FREQ1.
  - Using \( fhi \) to measure \( fin \). (a simple counter) Get a frequency number of \( fin \).
  - Multiple this frequency number by \( N \) and convert it to FREQ2.
  - Using synthesizer2 to generate the \( fout \), by FREQ2.

- **Advantage:**
  - \( fout \) is not directly related to \( fin \) electrically, noise in \( fin \) is isolated. PFD and filter are not required.
  - Especially good for multiplying the input frequency to a large number (\( N \) is big).
  - The VCO used for flying-adder synthesizers can be a very simple one with minimum analog complexity.
  - Synthesis1 in above diagram can be a very simple one (no fractional part)
Conclusion

• A novel frequency synthesis architecture is presented.

• This architecture can be used to generate many, many frequencies.
\[ F = p \times M \]

As M sweep

\( F \)  
\( F + 1 \)

\( F + 1 \)  
\( F + 2 \)

As M sweep

\( F \)  
\( F + 1 \)

\( p \), a required frequency