

“Flying-Adder” Frequency and Phase Synthesis Architecture

Liming XIU

Texas Instruments Inc, HPA/DAV

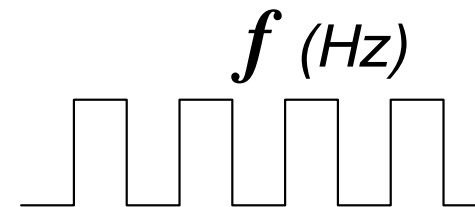
01/30/2005

What is it?

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



..0100101...



Continued

Background Material


This presentation is based on five papers:

- **IEEE Journal of Solid-State Circuit, 06/2000, “An Architecture of High Performance Frequency and Phase Synthesis”.**
- **IEEE Trans. on VLSI, 10/2002, “A ‘Flying-Adder’ Architecture of Frequency and Phase Synthesis with Scalability”.**
- **IEEE Trans. on Circuit & System II, 03/2003, “A New Frequency Synthesis Method based on ‘Flying-Adder’ Architecture”.**
- **IEEE Journal of Solid-State Circuit, 03/2004, “A Novel All Digital Phase Lock Loop with Software Adaptive Filter”.**
- **IEEE Trans. on VLSI, 02/2005, “A ‘Flying-Adder’ Frequency Synthesis Architecture of Reducing VCO Stages”.**

History

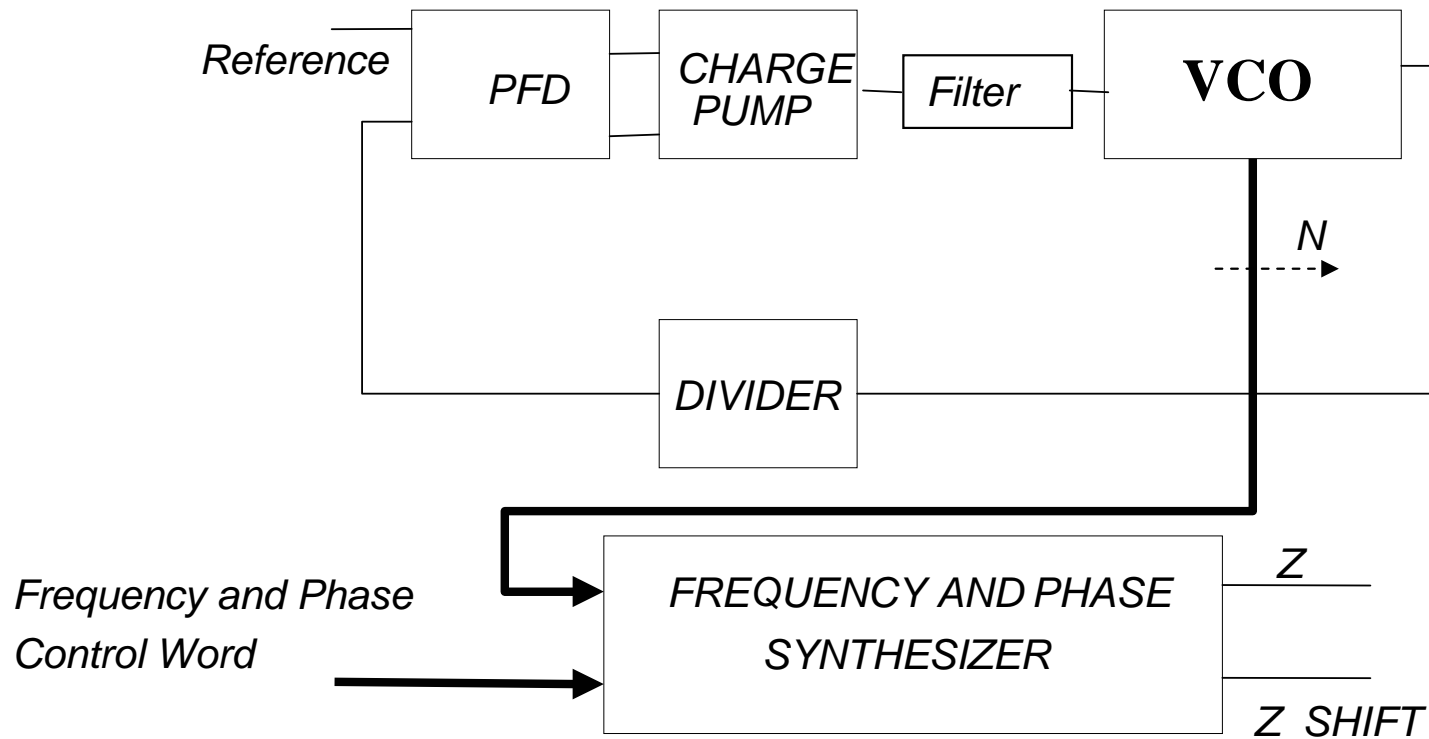
- **Started in late 1998, MSP/Video Group.**
- **Being continuously refined/improved.**
- **Thanks to Hugh Mair**

Presentation Outline

- **The principal Idea** 
- **Implementation: First Generation**
- **Implementation: Second Generation**
- **Integer-Flying-Adder Architecture**

Principal Idea

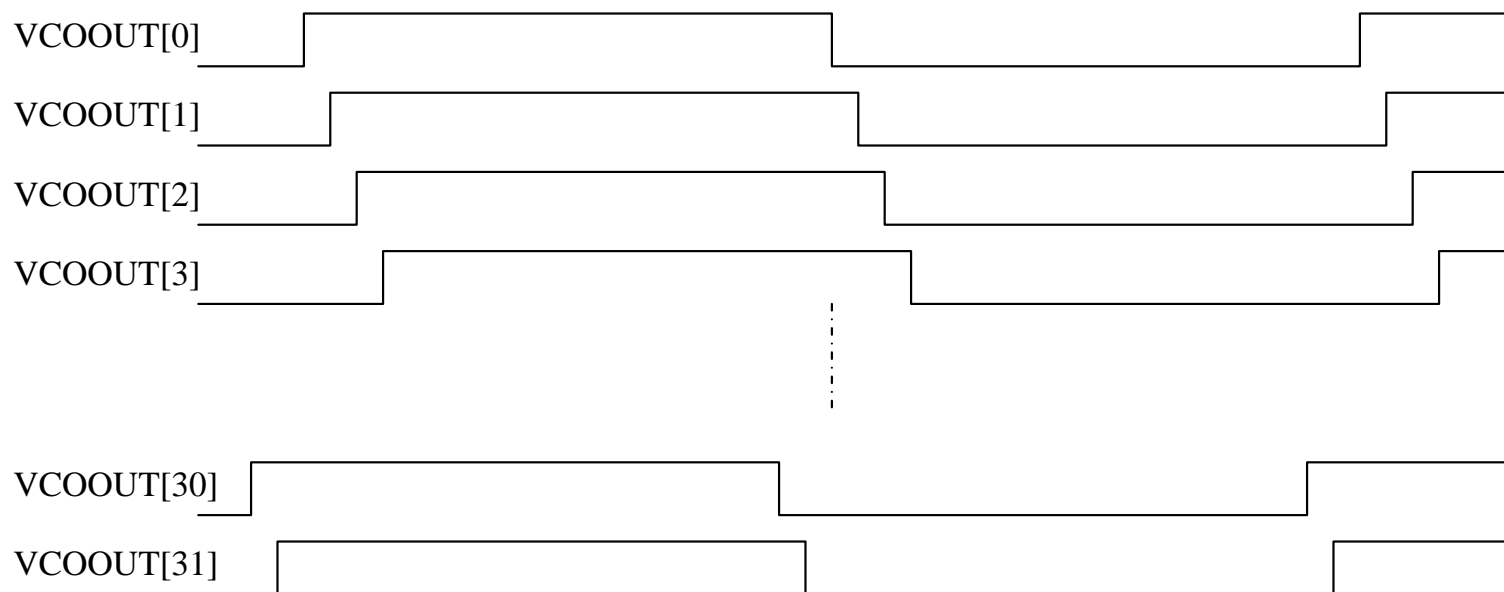
Using multiple equally-spaced phases generated from a VCO to synthesis various *frequency* and *phase*, by triggering the flip-flops at predestined time.



Continued

Principal Idea, continued

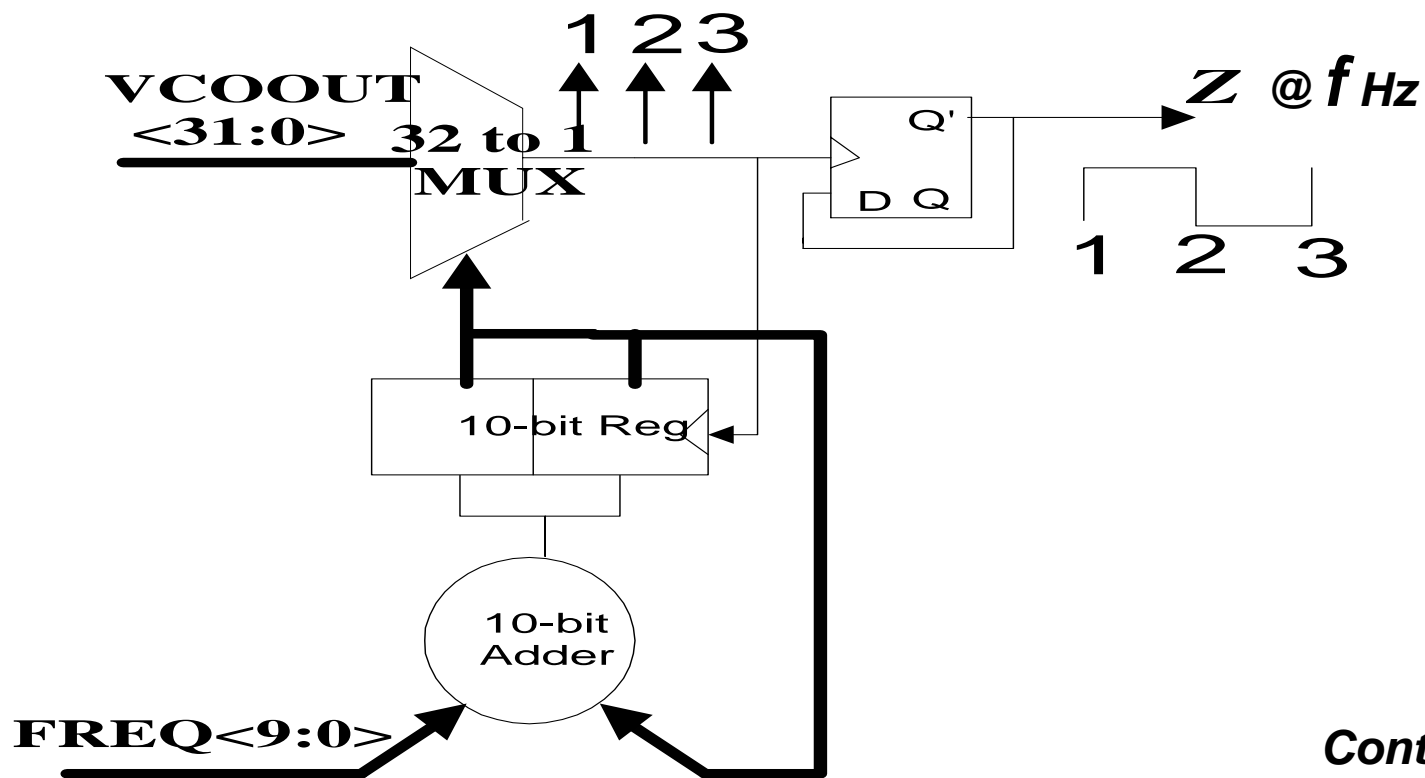
VCO Output waveforms, for N=32



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)

$$\Rightarrow \Delta = 6.4/32 = 0.2 \text{ (ns)}$$

Wanted: 204.08 MHz, or T = 4.9 ns

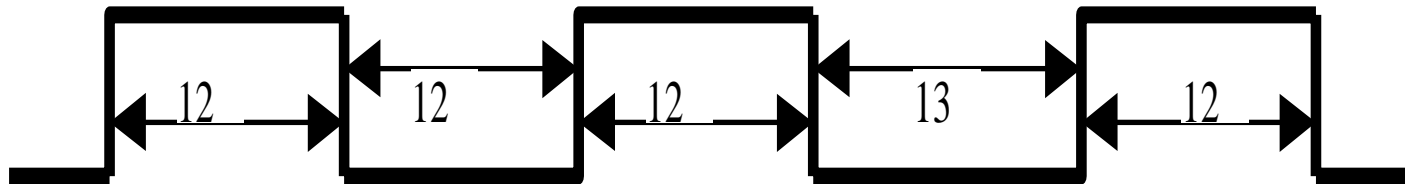
$$\Rightarrow \text{FREQ}[9:0] = T/(2\Delta) = 4.9/0.4 = 12.25 = 01100.01000b$$

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

Continued

Numerical Example, continued

0	12	24	4	17	29
00000	01100	11000	00100	10001	11101



00000.00000	01100.01000	11000.10000	00100.11000	10001.00000
+ 01100.01000	+ 01100.01000	+ 01100.01000	+ 01100.01000	+ 01100.01000
-----	-----	-----	-----	-----
01100.01000	11000.10000	00100.11000	10001.00000	11101.01000

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

$$T = \text{FREQ} * \Delta$$

$$\text{or, } \text{FREQ} = T / \Delta = M + r$$

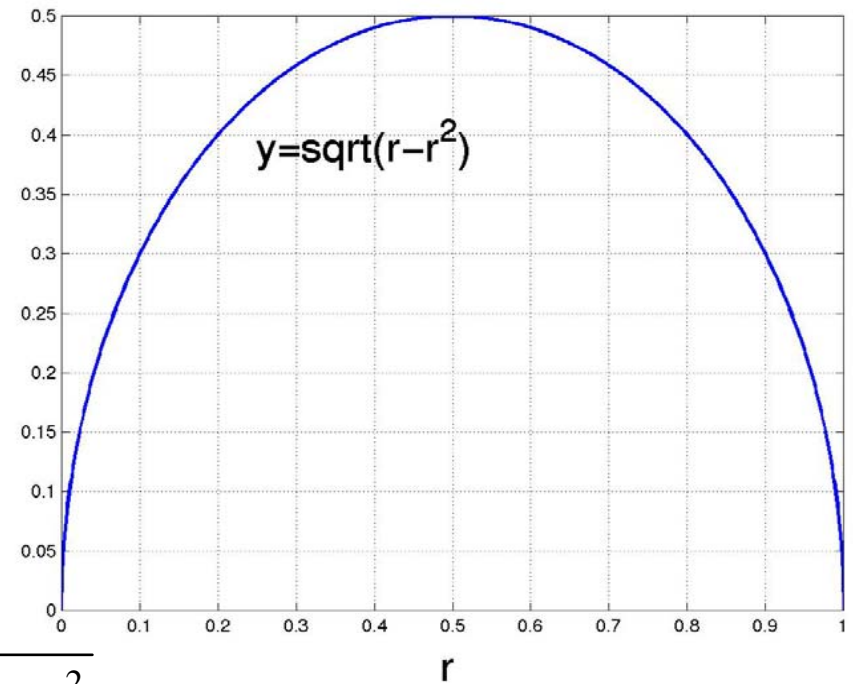
$$T_s = M * \Delta \quad P_l = r$$

$$T_l = (M + 1) * \Delta \quad P_s = 1 - P_l = 1 - r$$

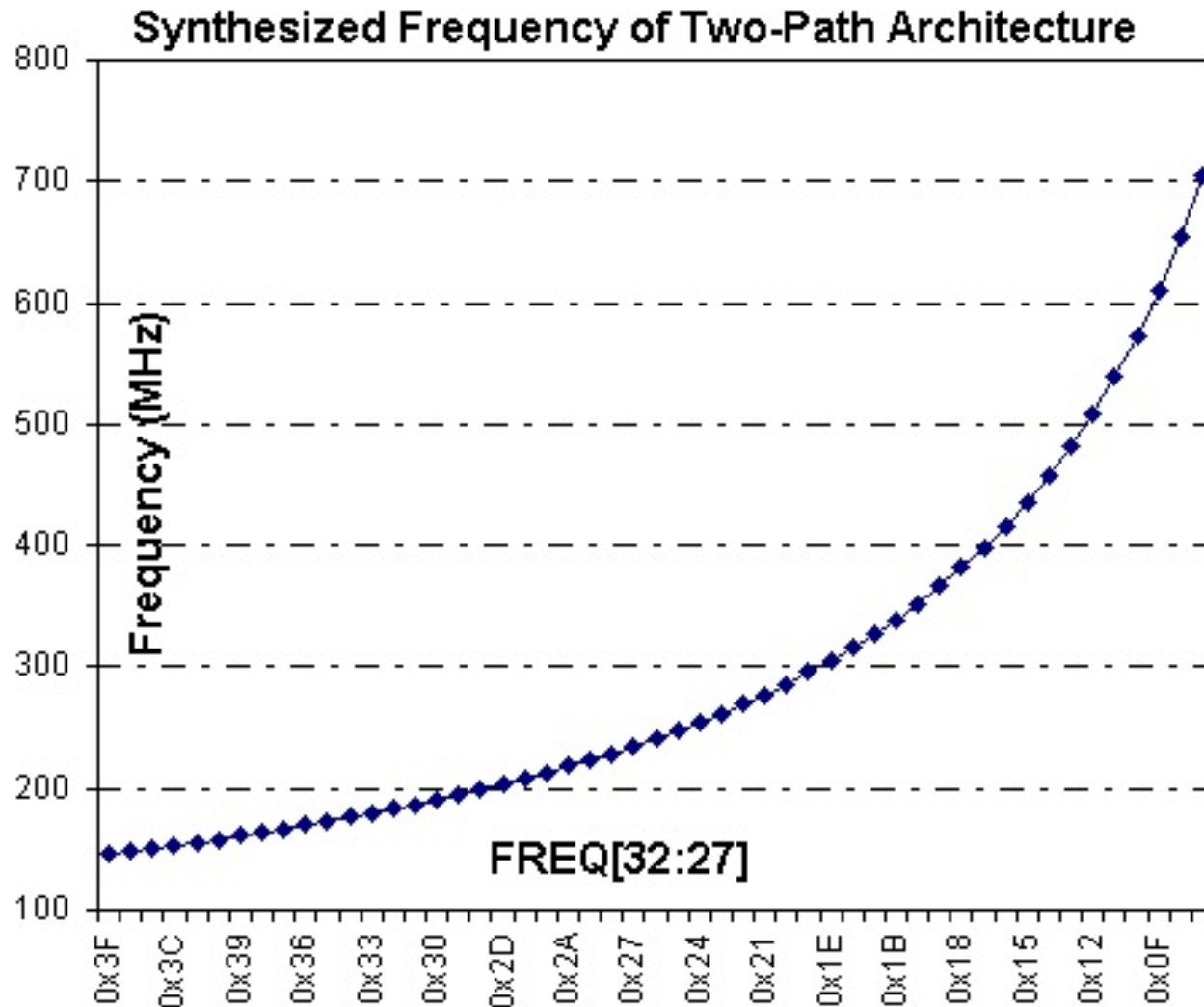
$$J_{pk-pk} = T_l - T_s = \Delta$$

$$J_{mean} = P_l(T_l - T) + P_s(T_s - T) = 0$$

$$J_{rms} = \sqrt{P_l(T_l - T)^2 + P_s(T_s - T)^2} = \Delta \sqrt{r - r^2}$$




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.

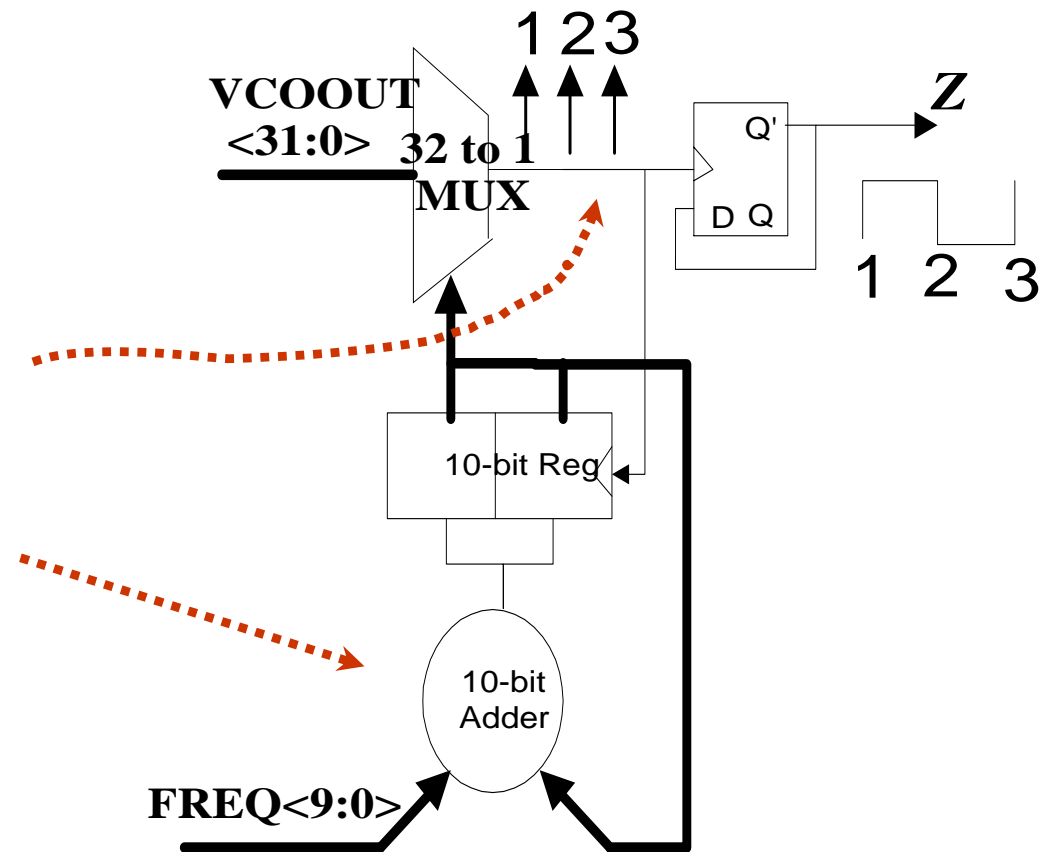
Presentation Outline

- The principal Idea
- **Implementation: First Generation** ← 
- Implementation: Second Generation
- Integer-Flying-Adder Architecture

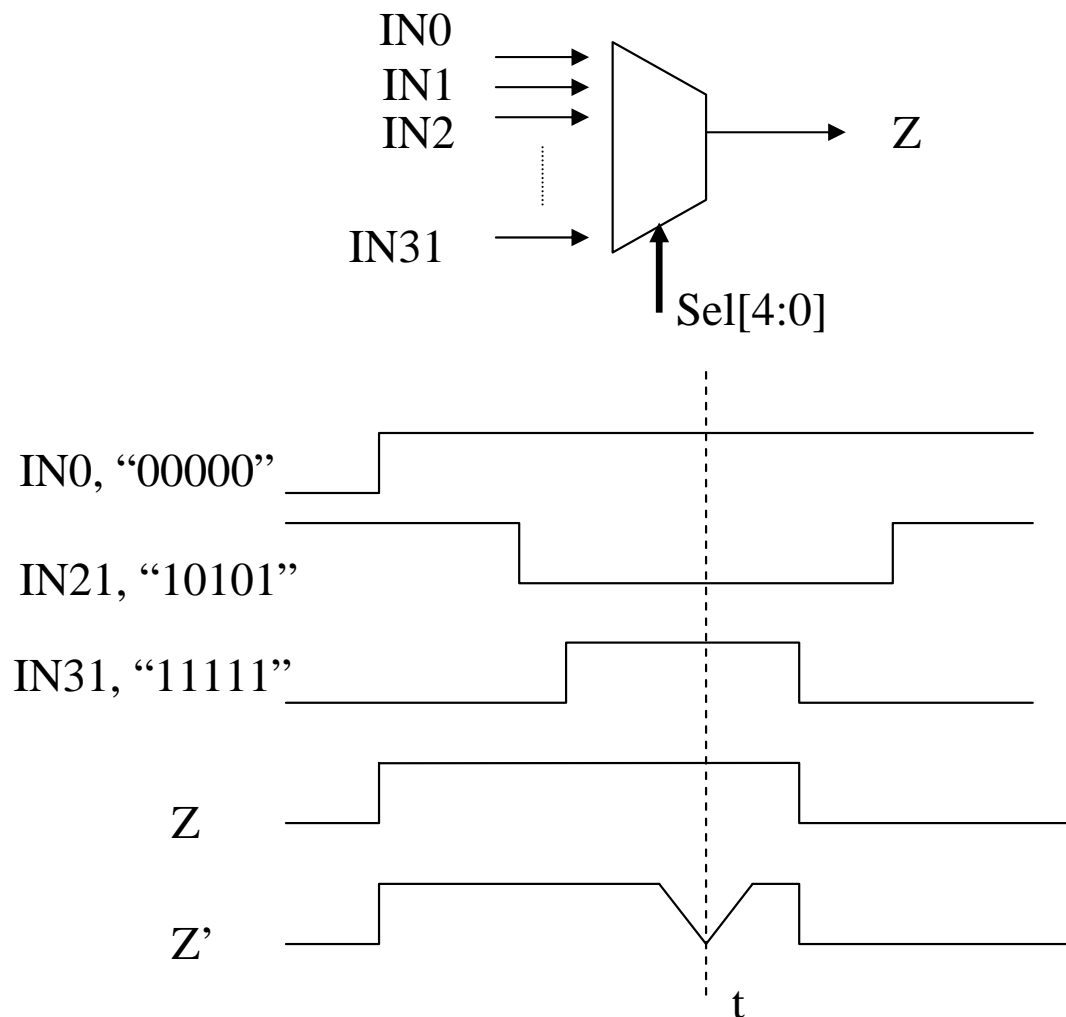
Implementation: Problems

Two problems:

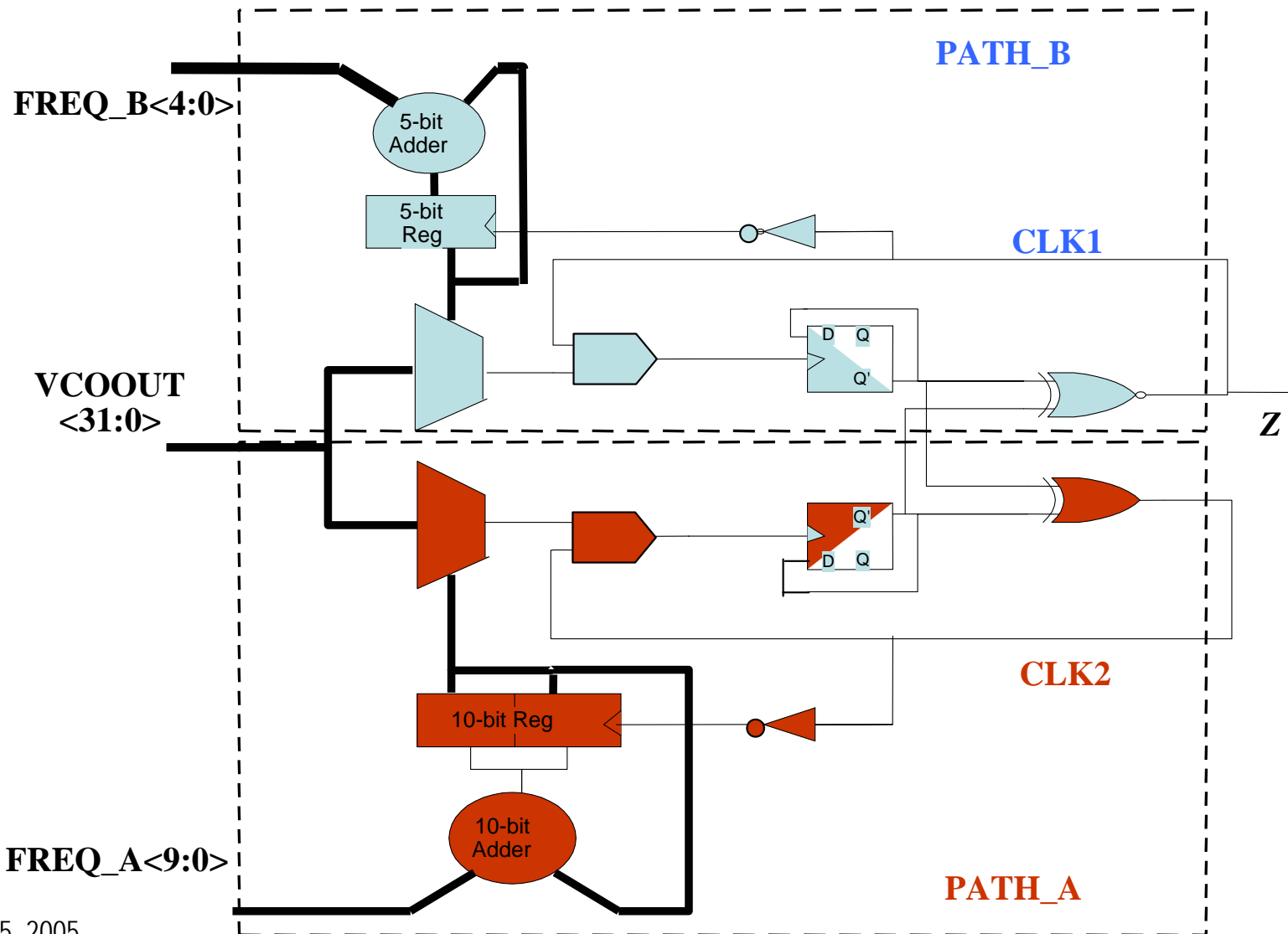
- The glitch of the MUX
- The speed of the adder



The Glitch of the MUX



Implementation: Two Paths

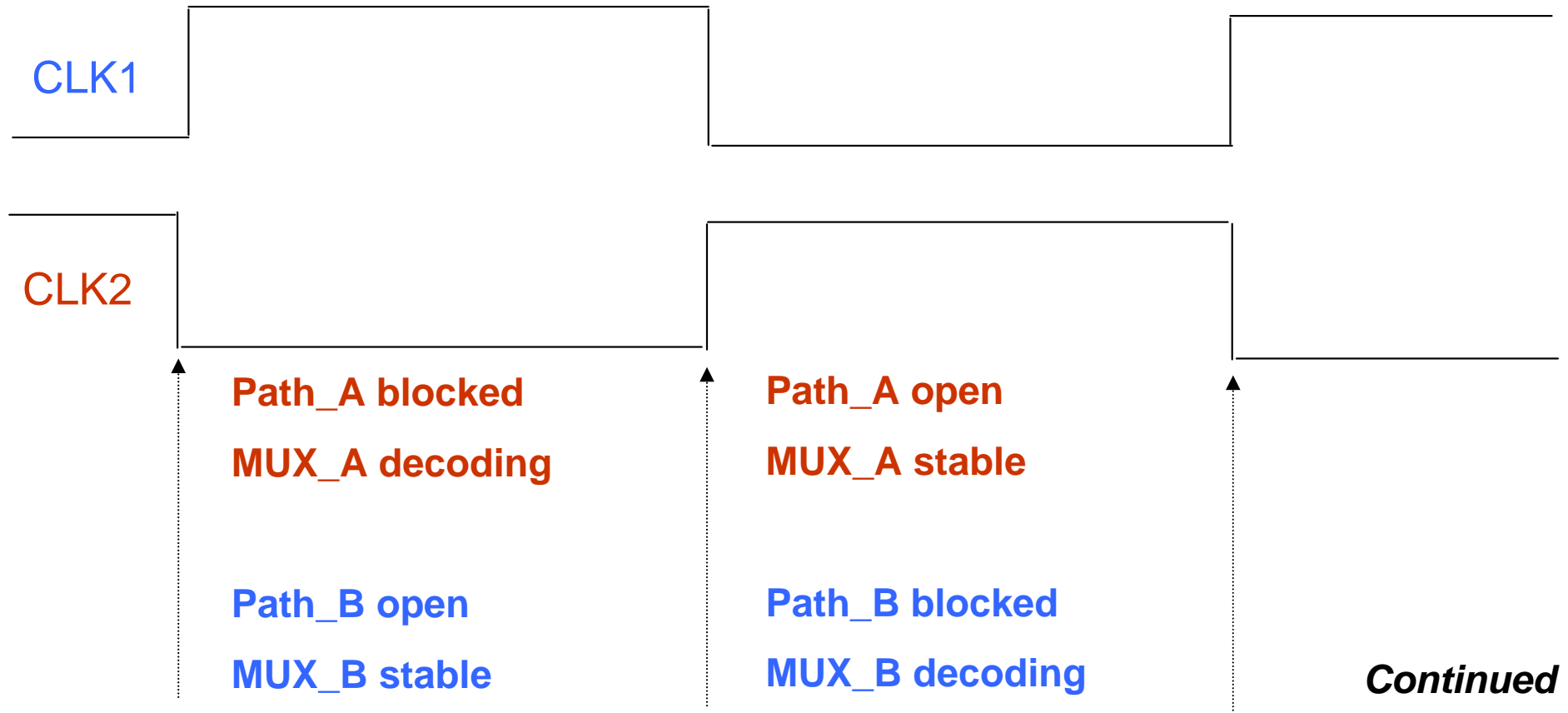


Continued

Implementation: Two Paths

Solved the glitch problem:

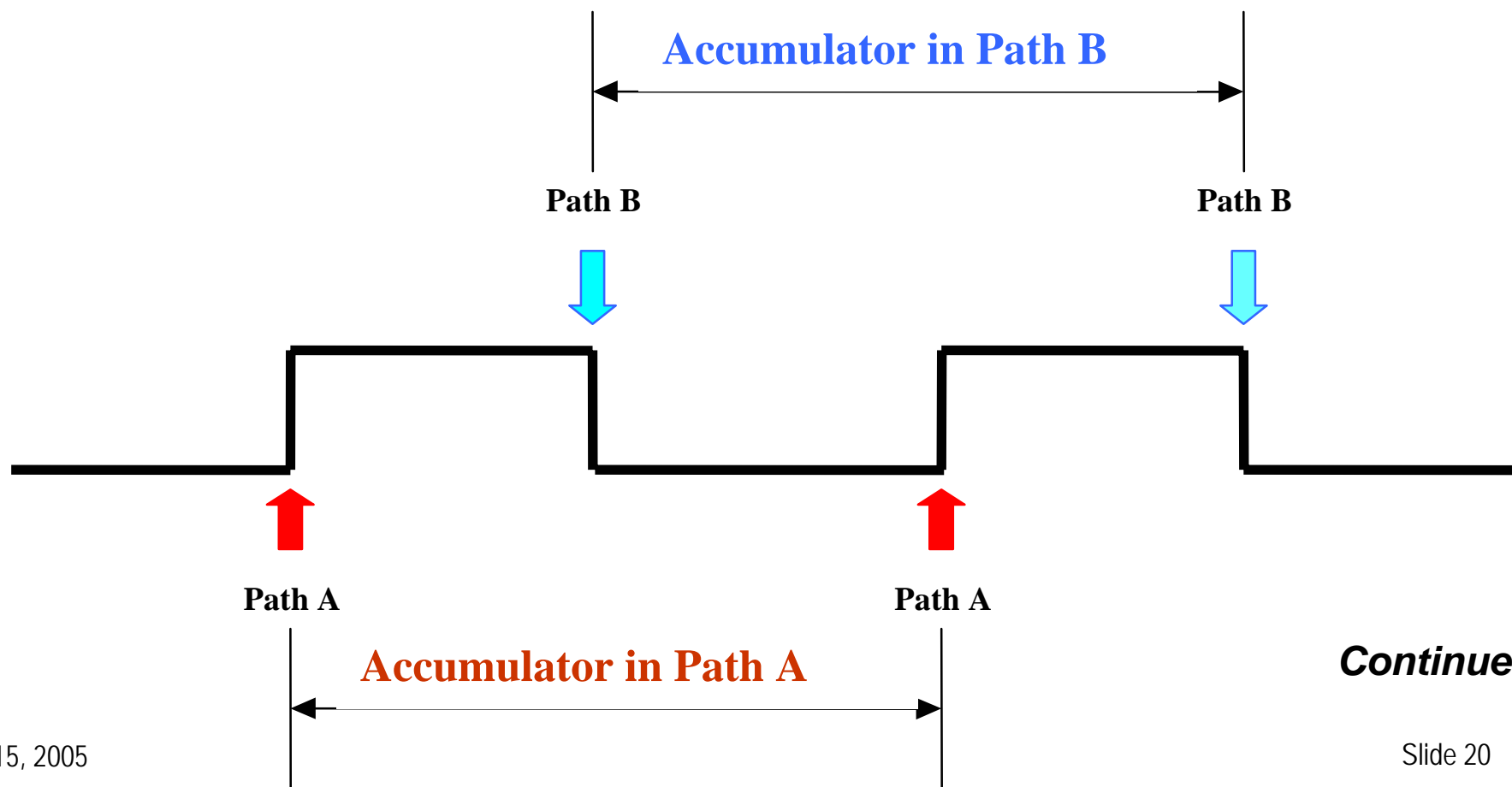
the two paths are interlocked



Continued

Implementation: Two Paths

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



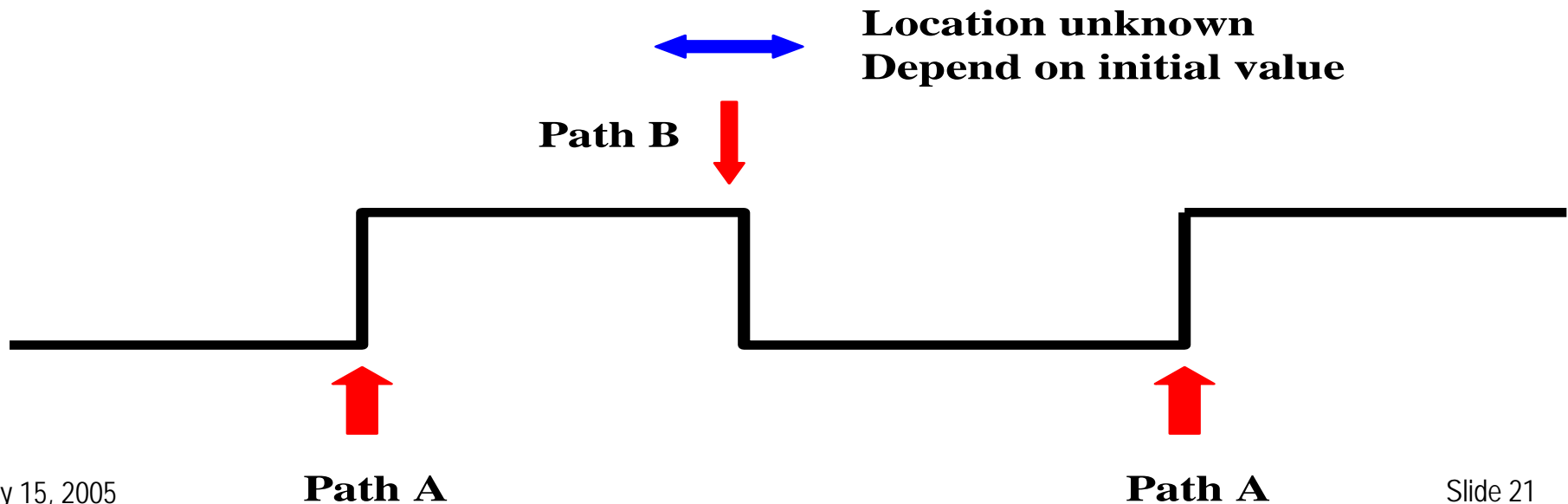
Continued

Implementation: Two Paths

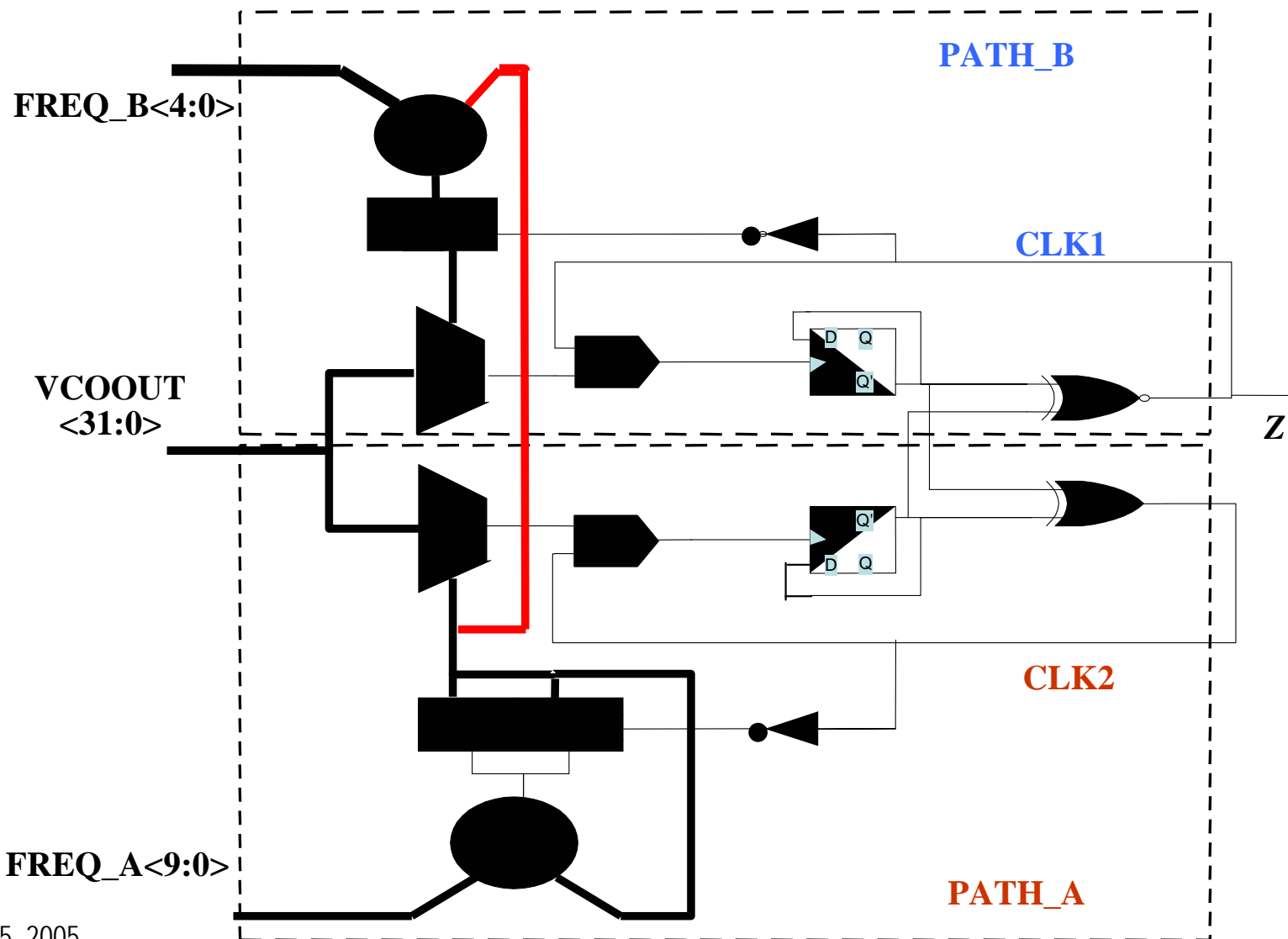
This two paths architecture solved the previous two problems, but created a new problem:

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

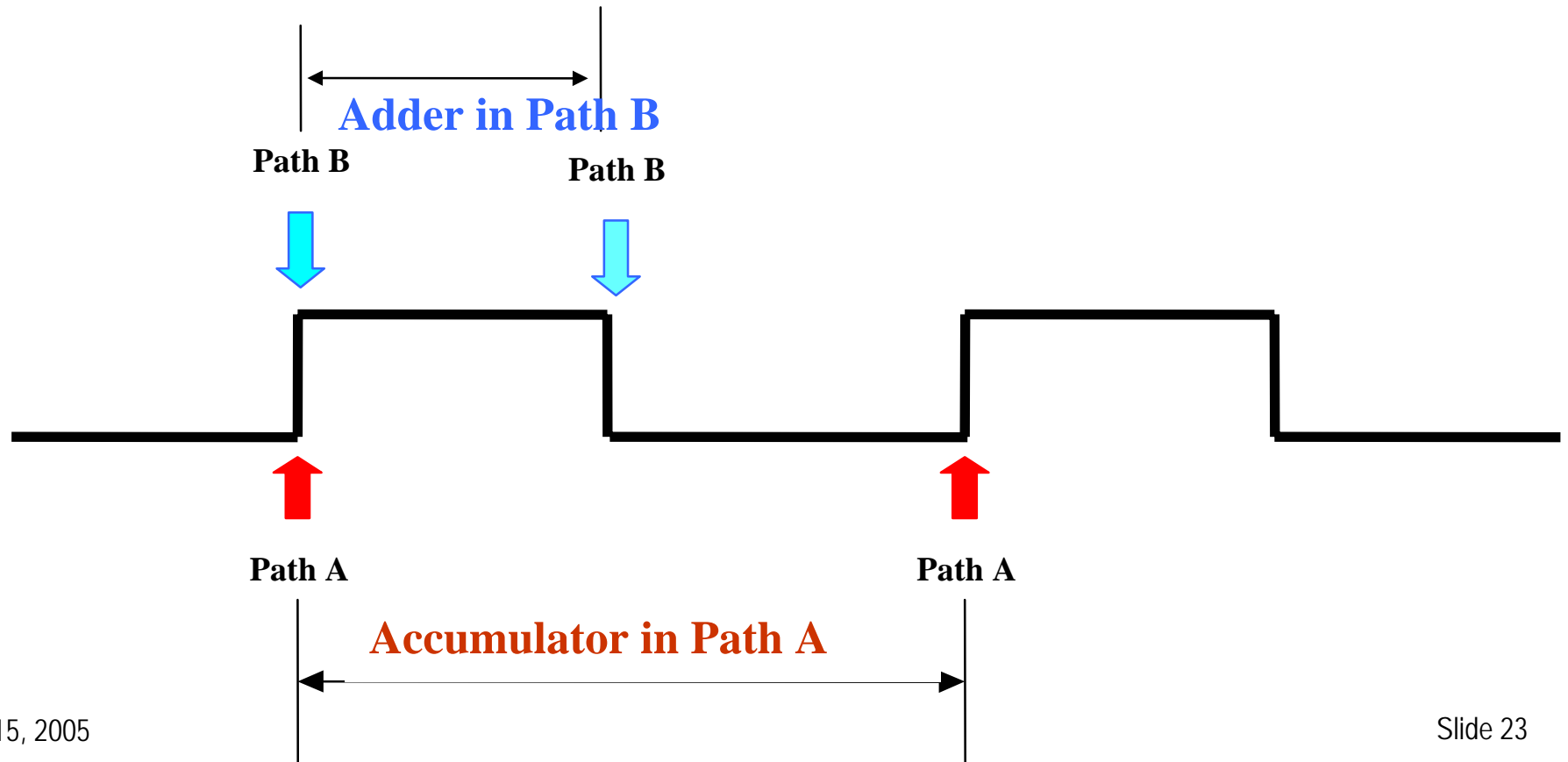


Continued

Implementation: Synchronized

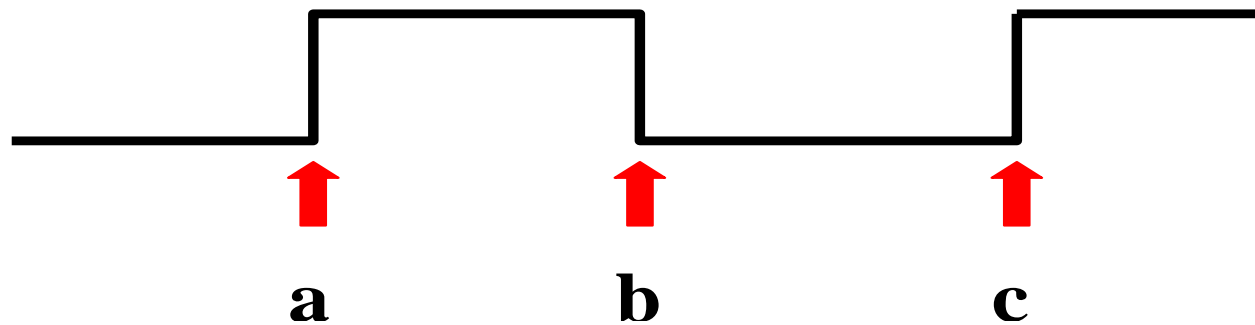
Now MUX_B's address is related to MUX_A's

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



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**

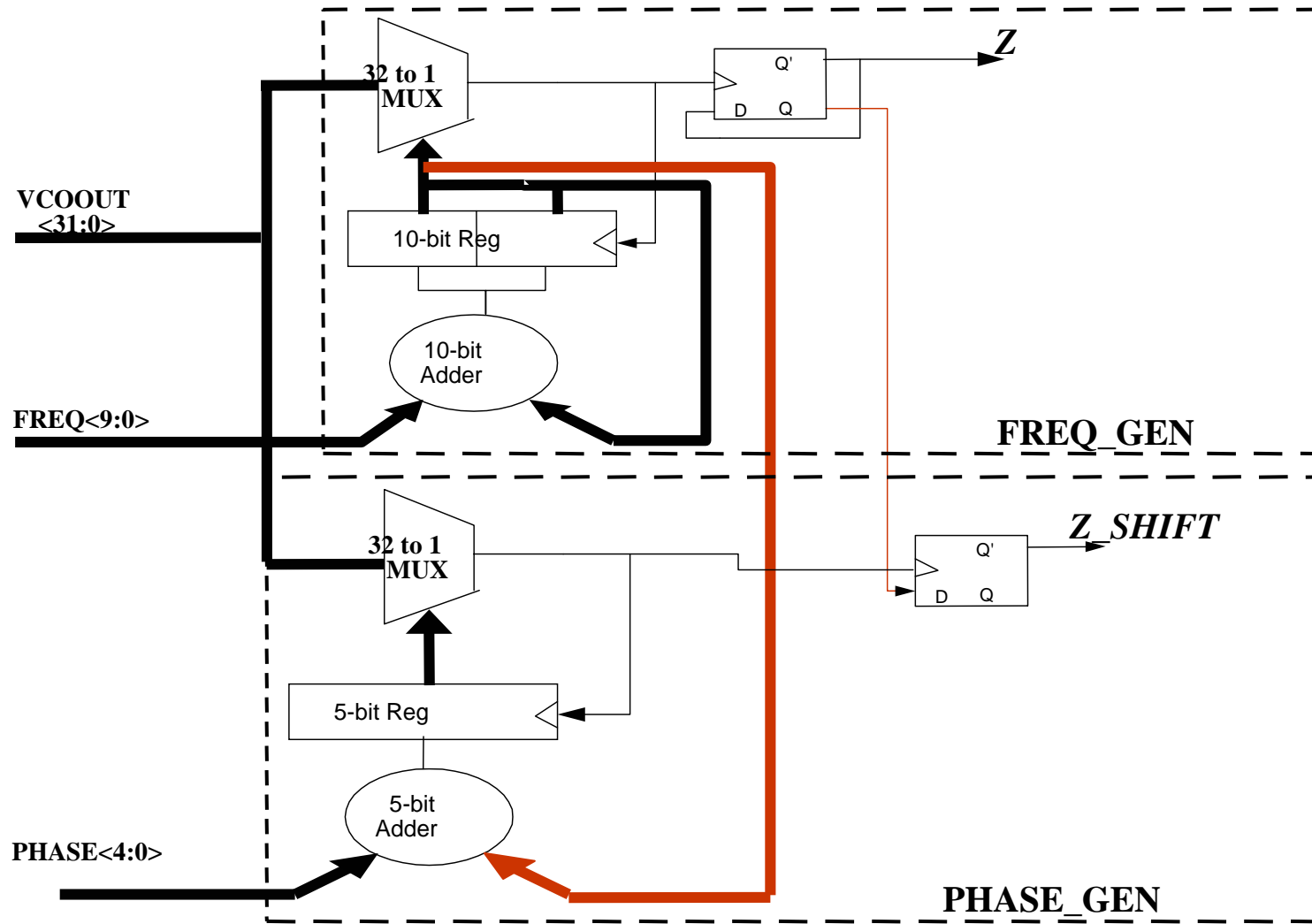
Key features of this architecture:

- **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.

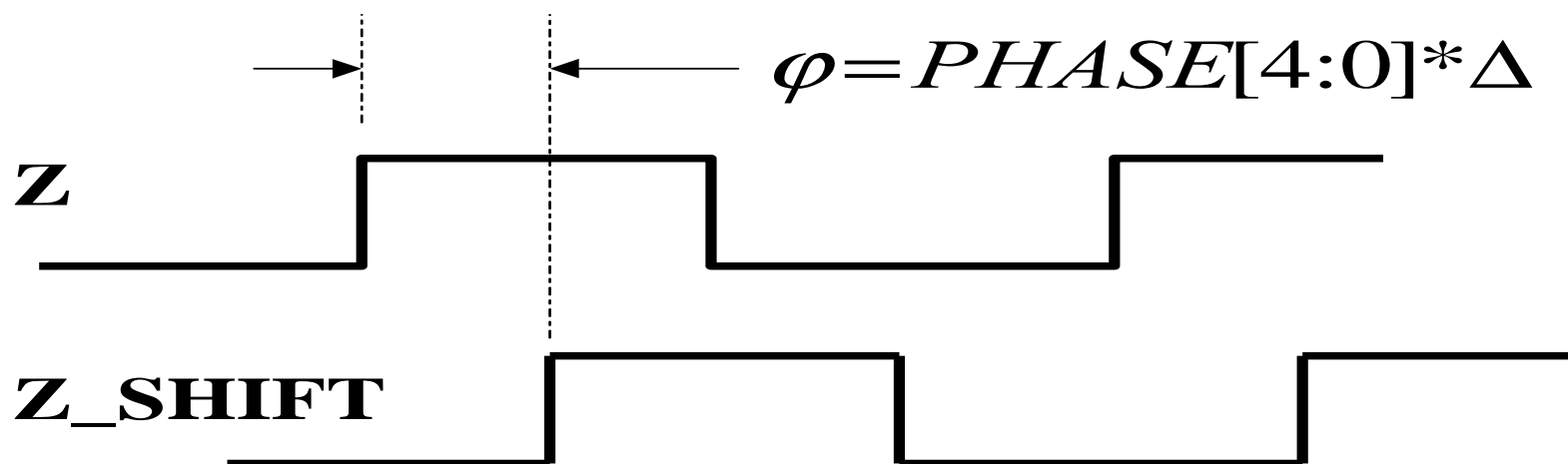
Phase Synthesis: The idea



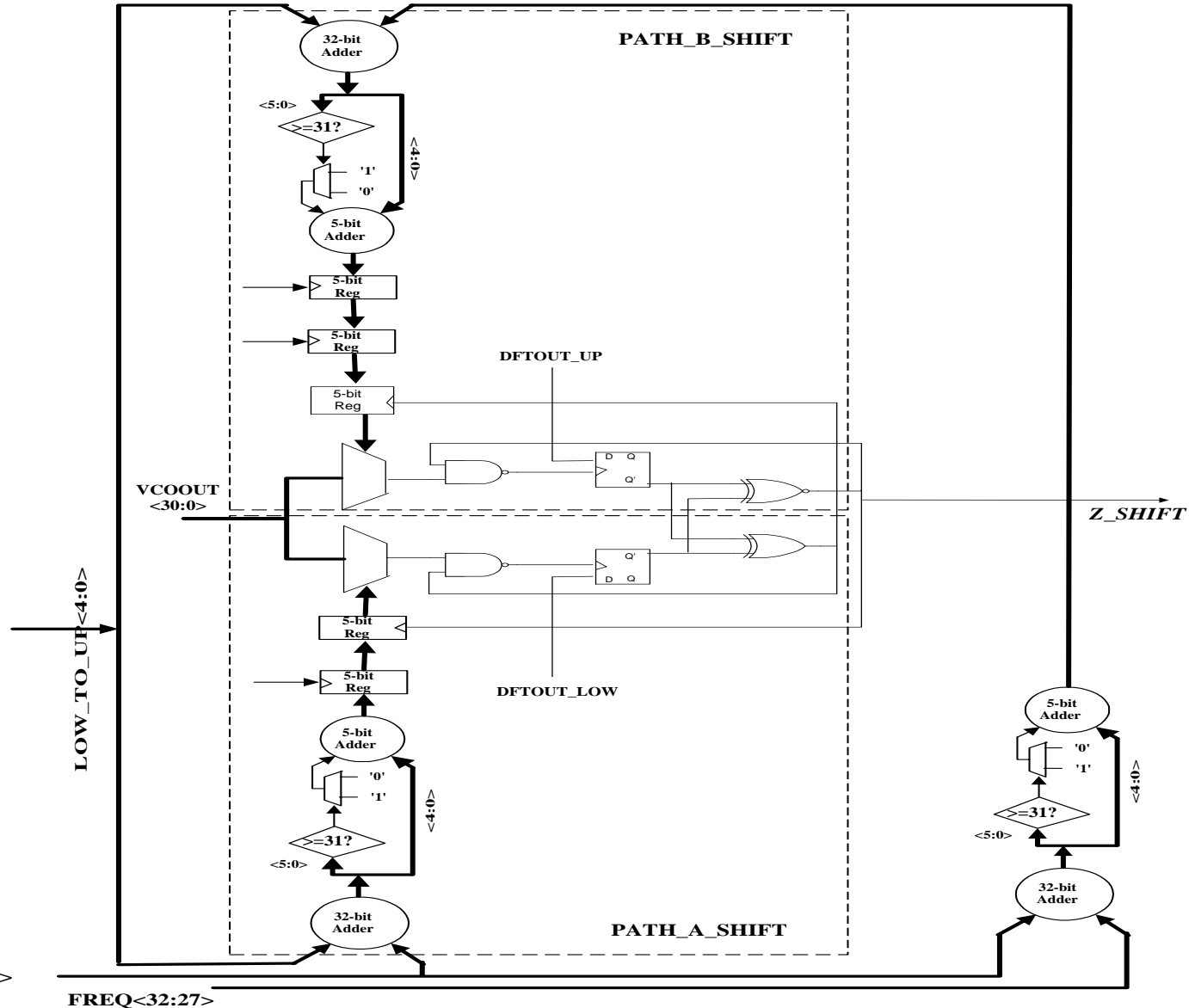
Continued

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: $PHASE[4:0] * \Delta$



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

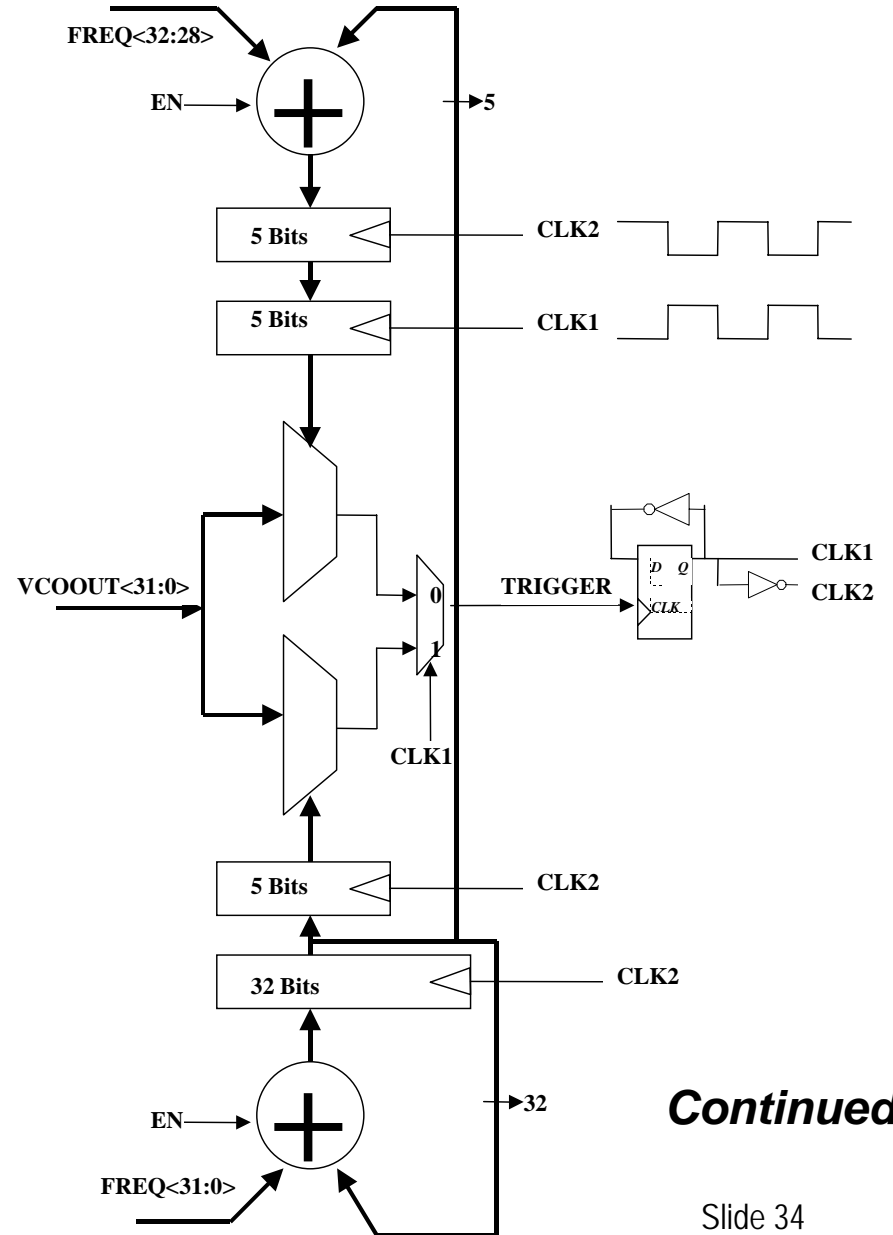
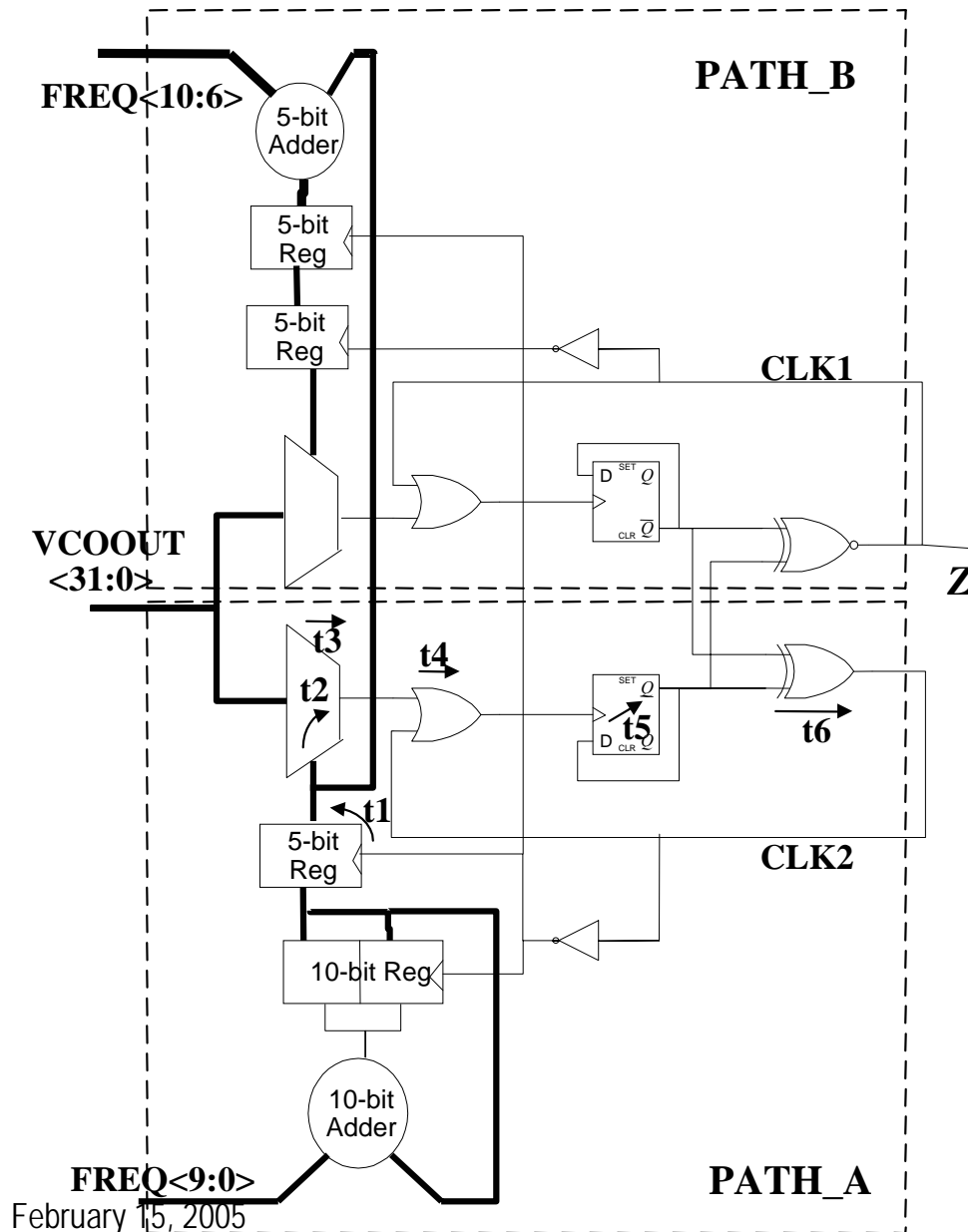
Second Generation Architecture

The new architecture:

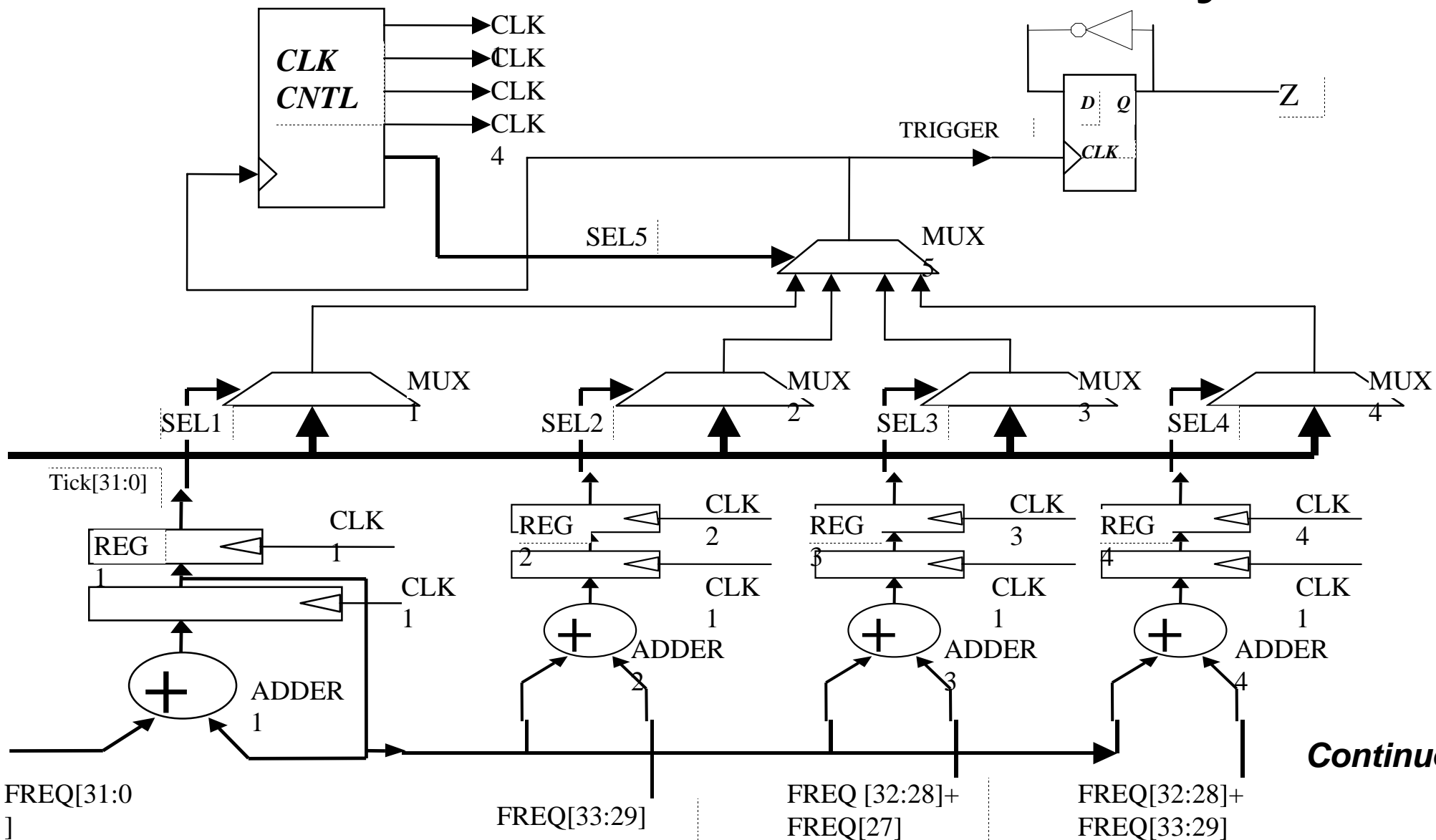
- the operating speed is greatly **improved**.
- has **scalability** for higher output frequency.
- has an internal node whose frequency is **higher** than that of the synthesized output.
- **eliminates** the “dead-zone” and “dual-stability” for phase synthesis.

Continued

Second Generation Architecture



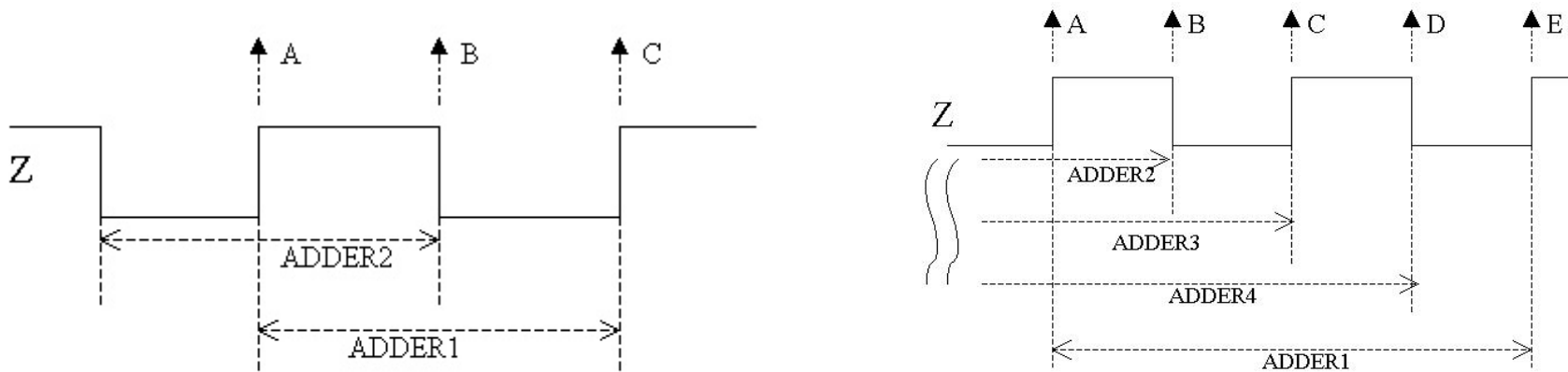
Sec. Gen. Arch.: Scalability



Continued

Sec. Gen. Arch.: Scalability

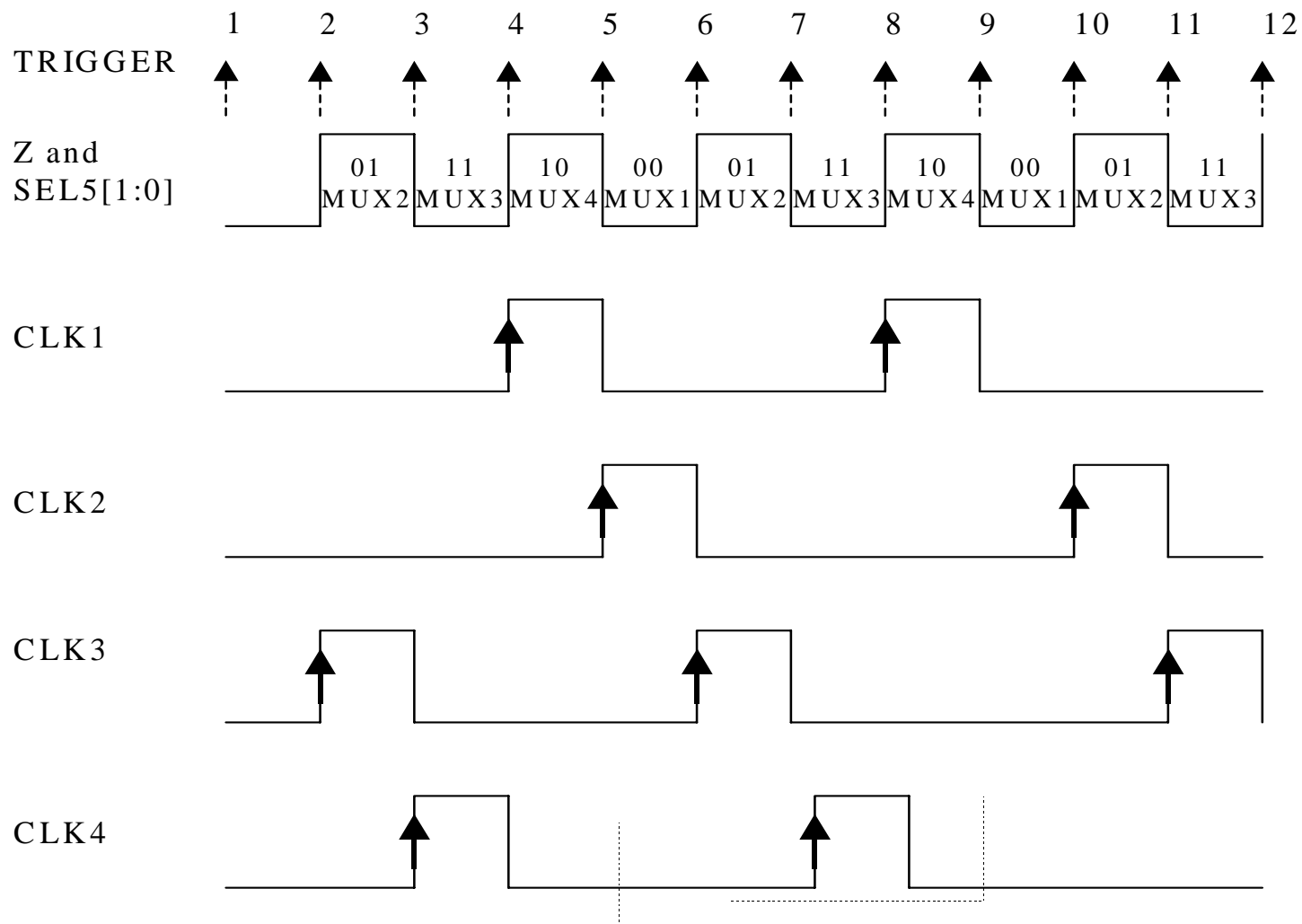
- Multiple paths (more than two) to relax the constraints on adders further -> **higher output frequency**



Continued

Sec. Gen. Arch.: Scalability

- The clocks signals and the mechanism of interlocking



Sec. Gen. Arch.: Phase Synthesis

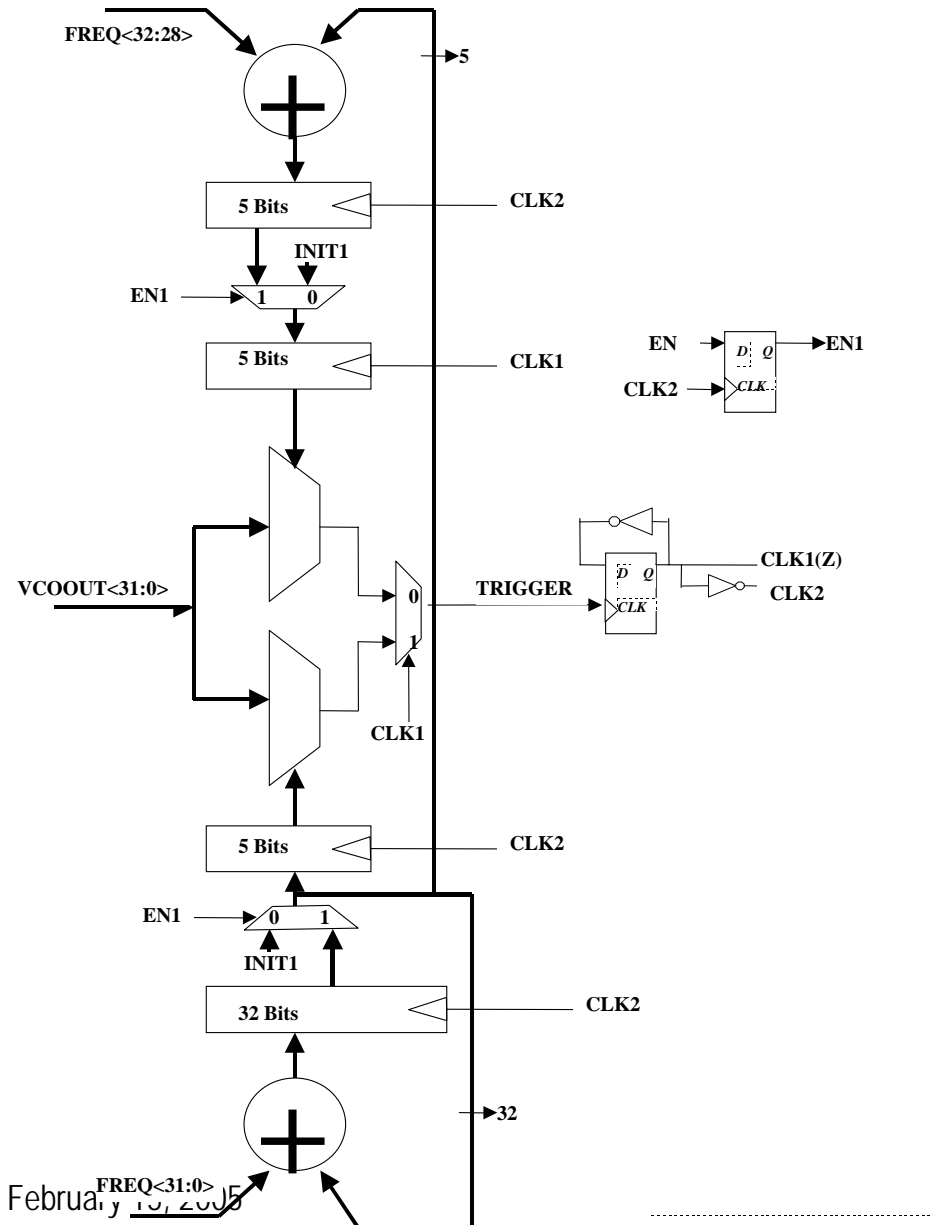


Fig. 16. The circuitry for Z.

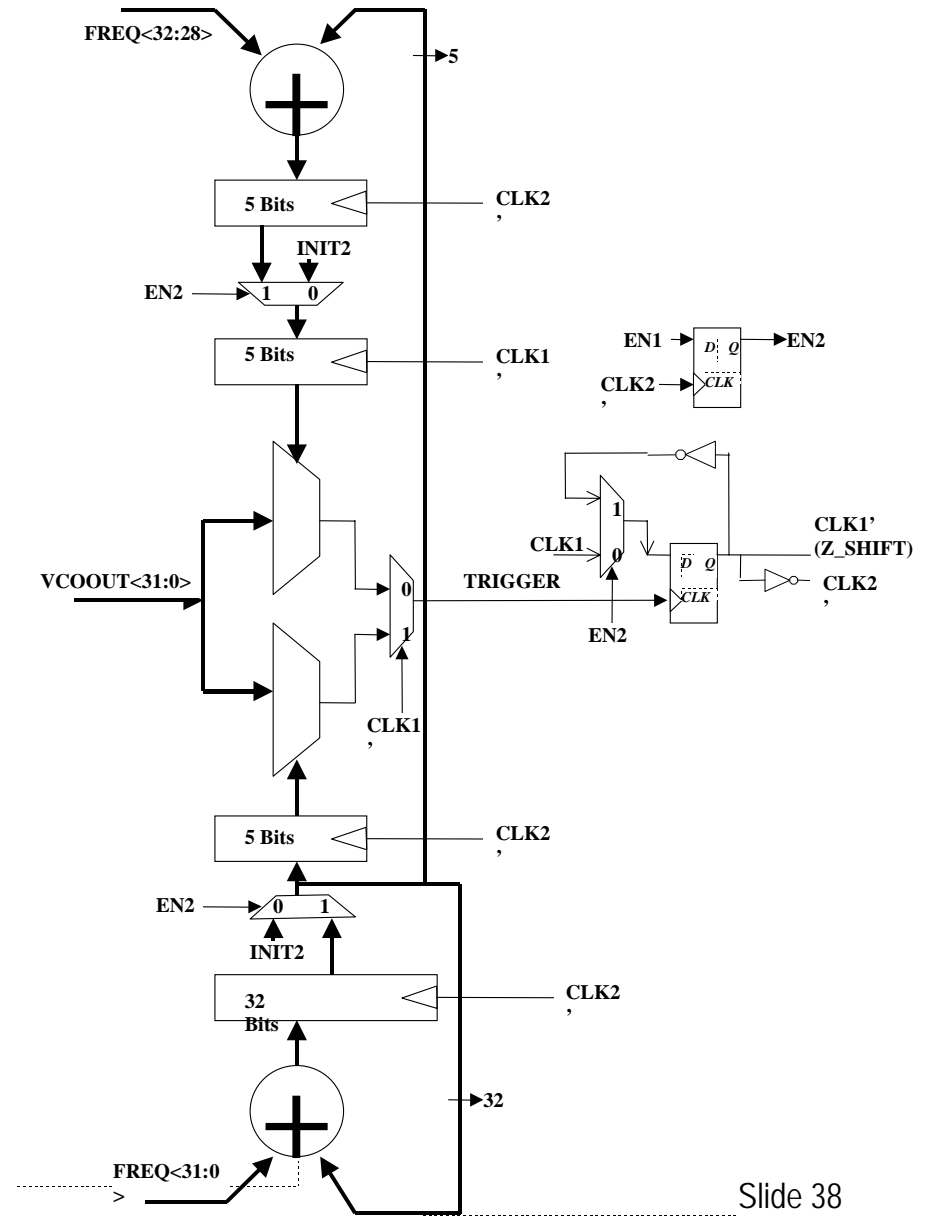



Fig. 17. The circuitry for Z_SHIFT

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

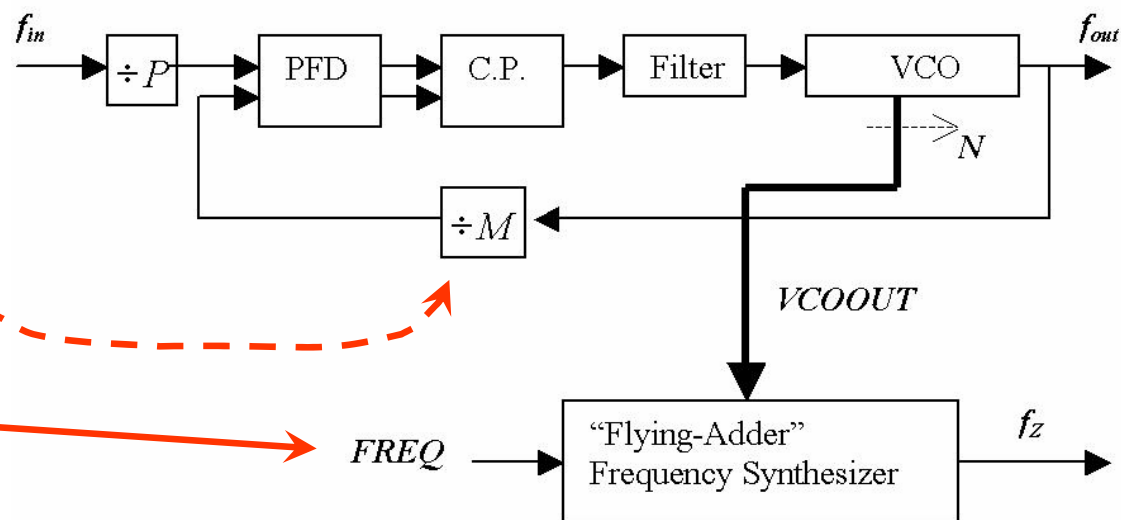
Integer-Flying-Adder Architecture

Idea:

Make PLL programmable

Get ride of fractional bit

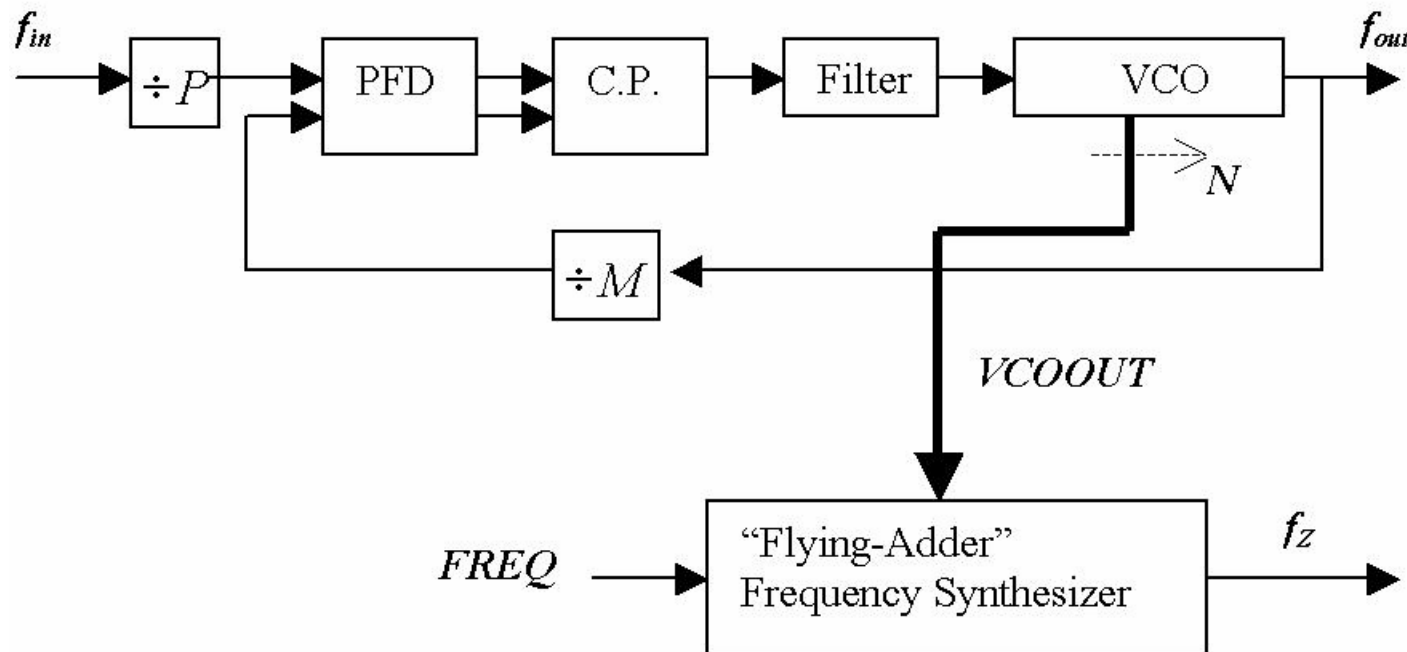
→ Eliminate the inherent jitter



Integer-Flying-Adder: Method

$$FREQ = T/\Delta = 1/(f * \Delta) = ((f_{in} * N)/(f * P)) * M$$

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

```

error_min = very_big_number
for ( M1 ≤ M ≤ M2 ) {
    freq = ((fin*N)/(f*P))*M
    error = min( freq-floor(freq), ceiling(freq)-freq )
    if ( error < error_min ) {
        error_min = error
        Mbest = M
        if ( freq - floor(freq) < 0.5 {
            FREQ = floor(freq)
        }
        else {
            FREQ = ceiling(freq)
        }
    }
}
    
```

Integer-Flying-Adder: Error Upper-Bound

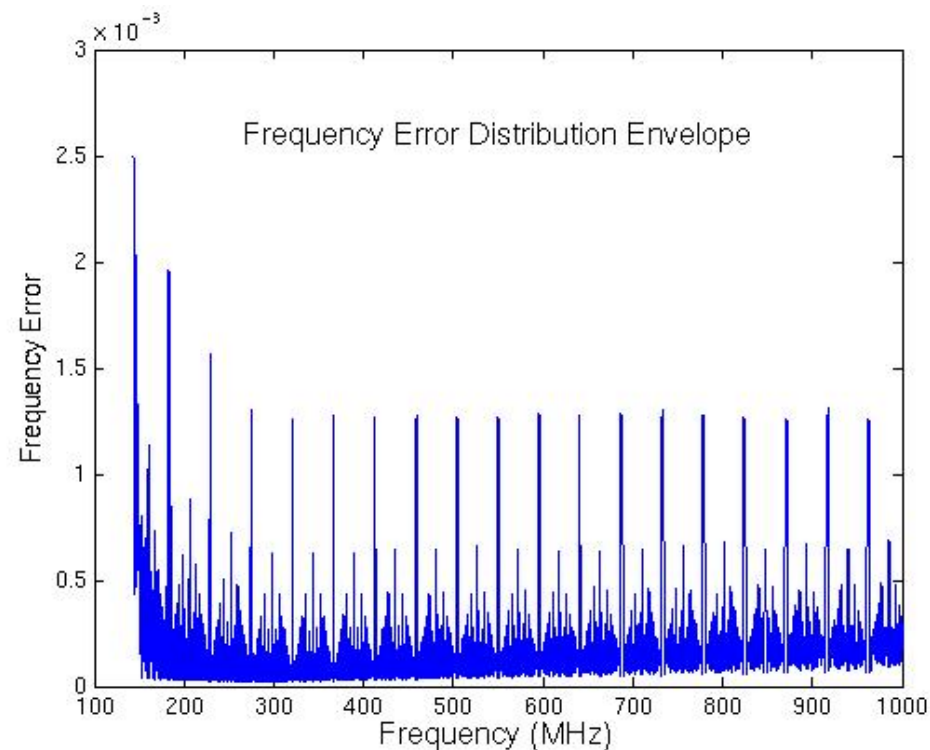
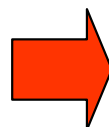
$$\begin{aligned}
 |T-T'|/T &= r*\Delta/T \\
 &\leq (1/2) * ((fin*N)/(f*P)) / (((fin*N)/(f*P))*M) \\
 &= 1/(2*M) \\
 &\leq 1/(2*M_1)
 \end{aligned}$$

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
}
    
```

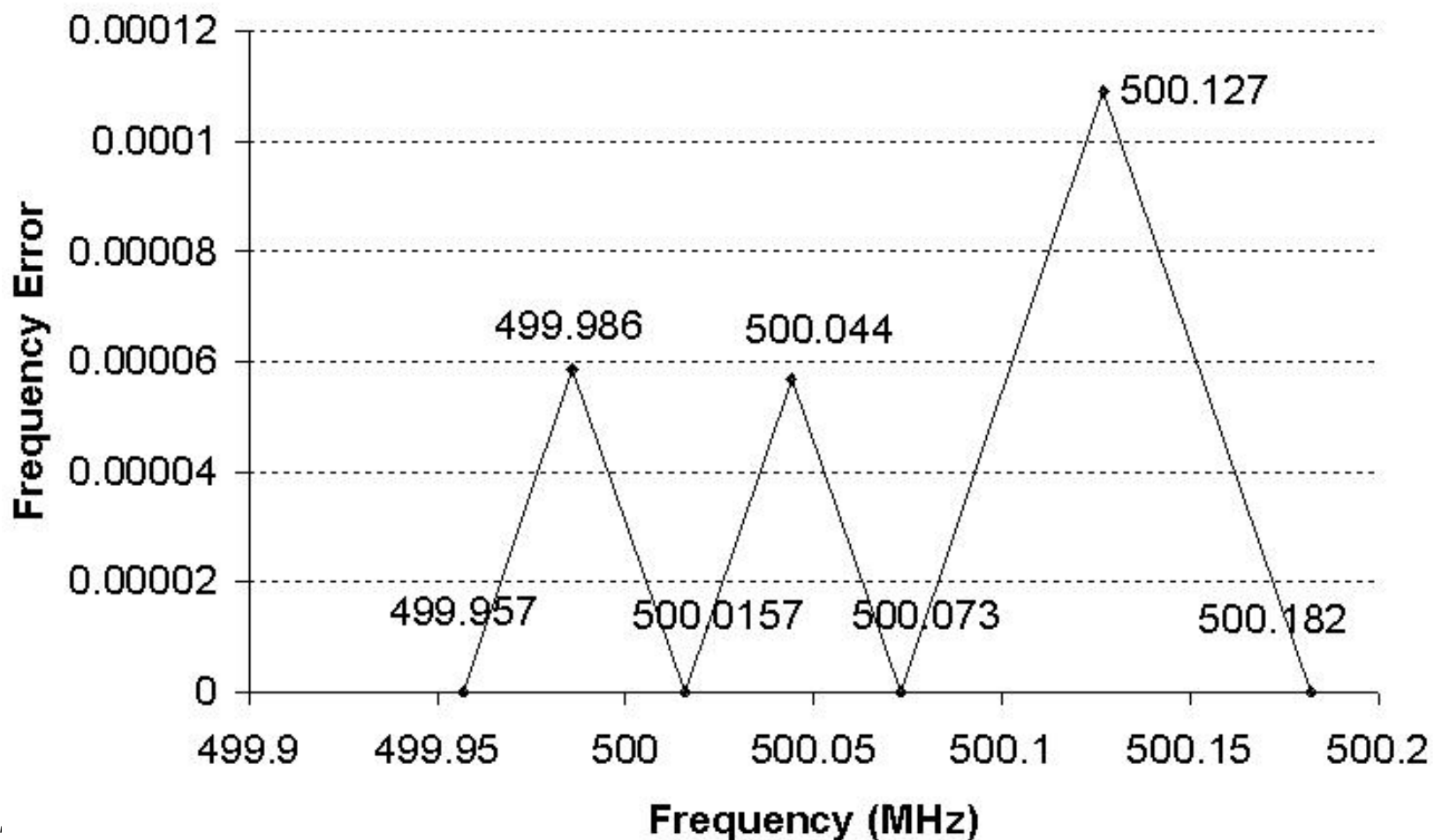


See paper on TCASII (3th paper) for mathematical prove

Continued

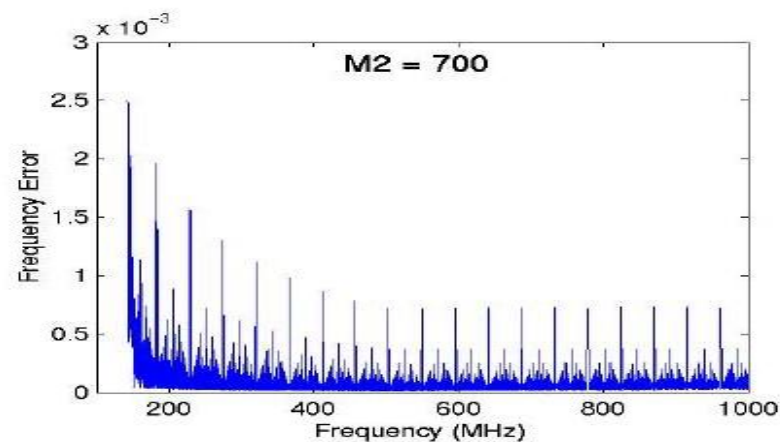
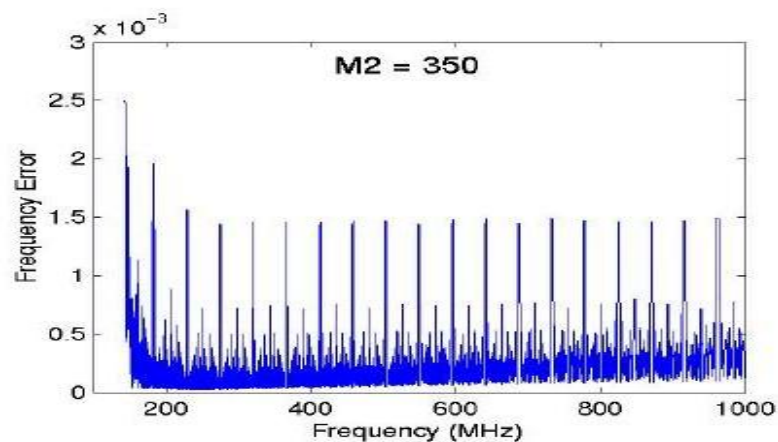
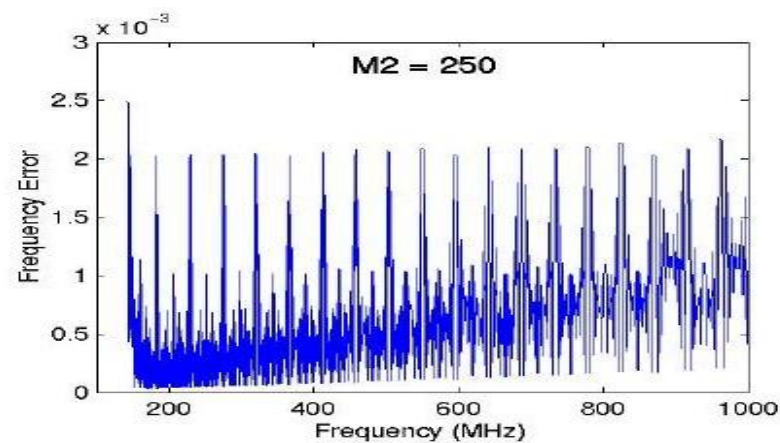
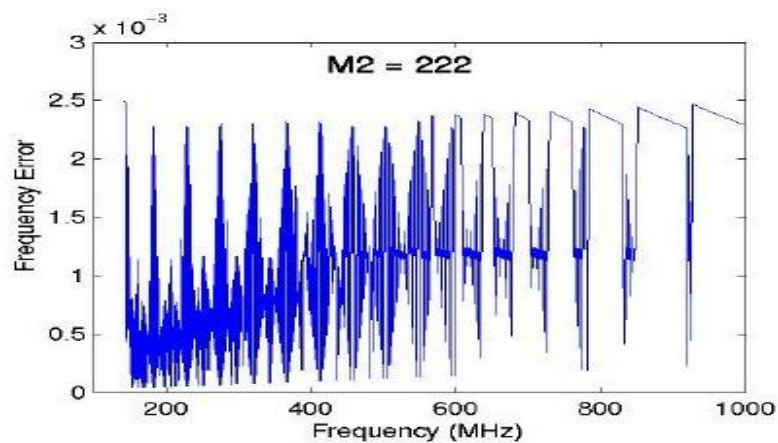
Integer-Flying-Adder: Error Distribution Envelope

Frequency Error Distribution



Integer-Flying-Adder: Error Distribution Envelope

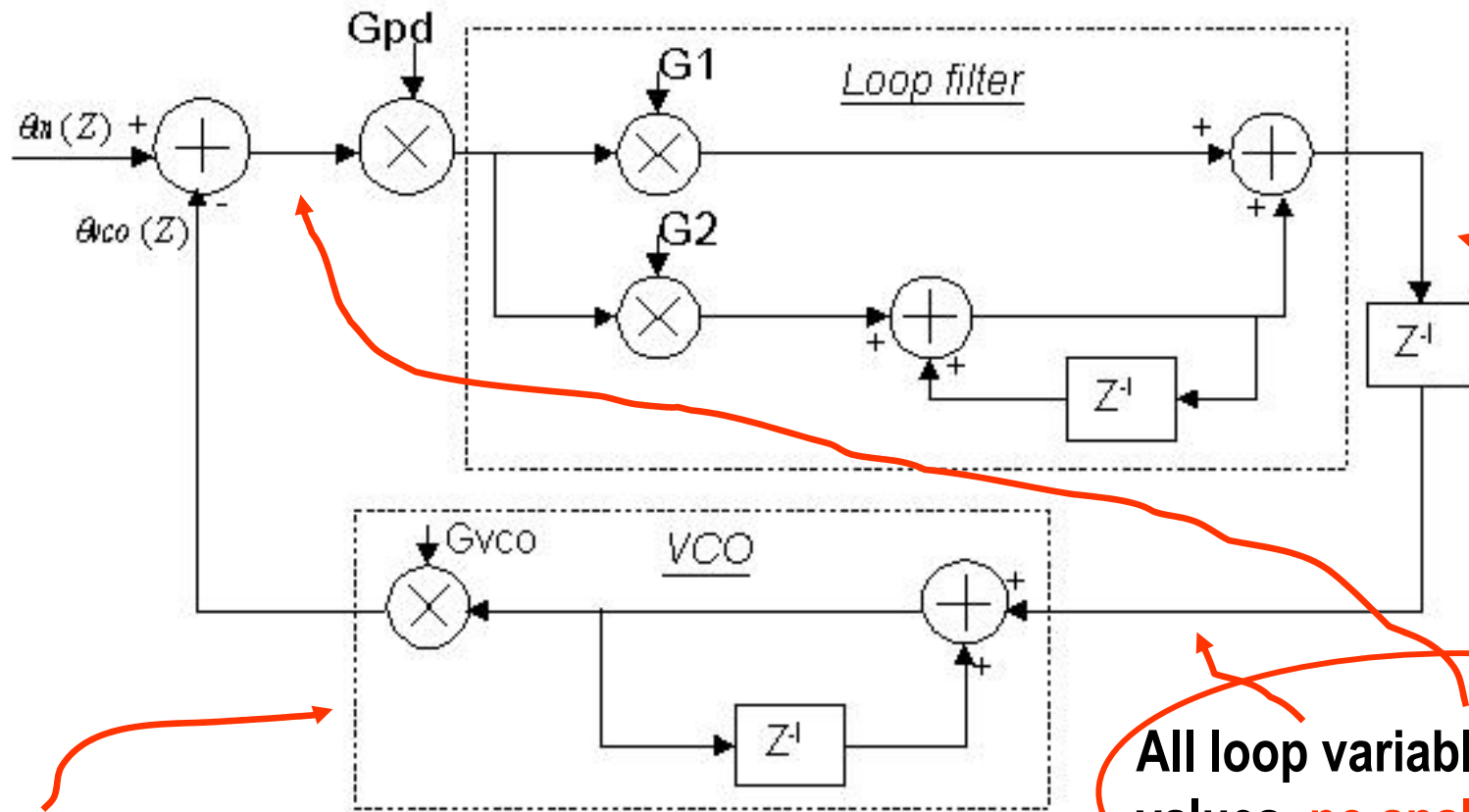
The effect of $M2$ on the error distribution envelope



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.**

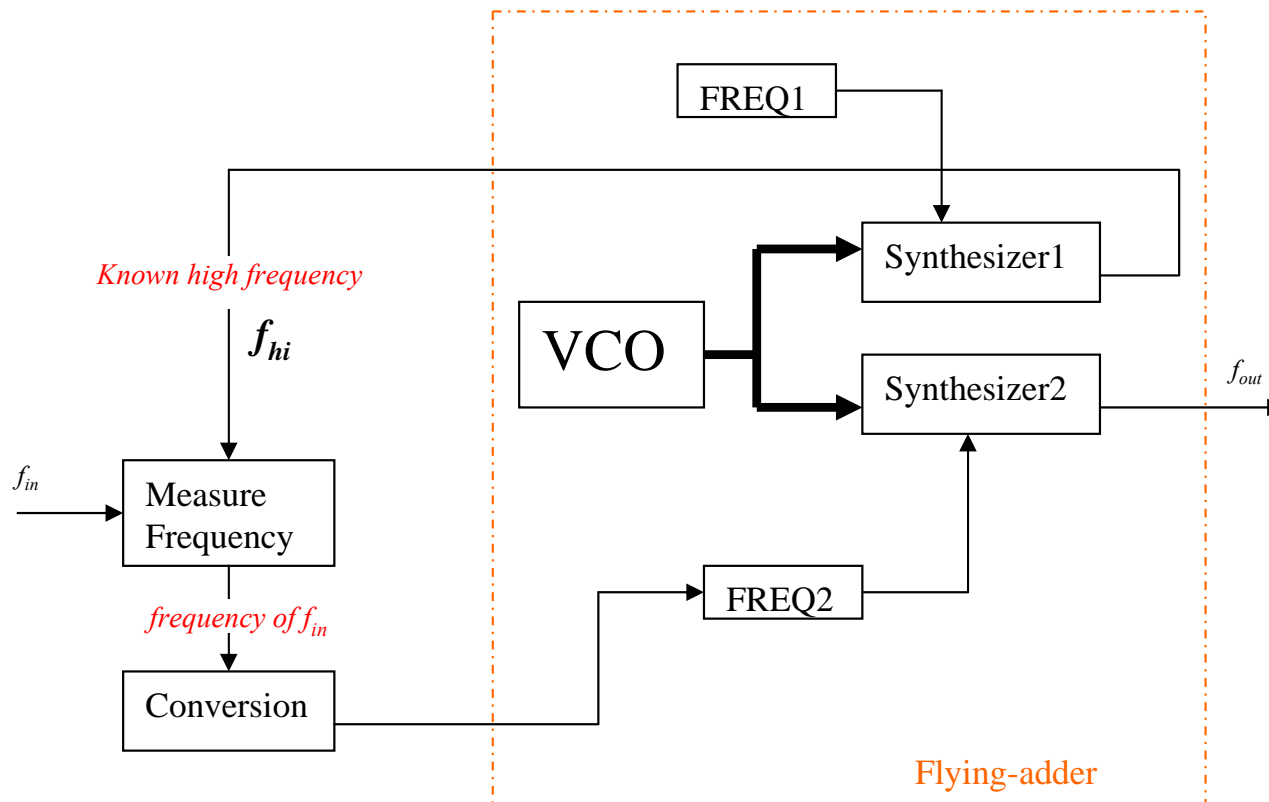
One Application Example: All Digital Phase Lock Loop



“flying-adder” synthesizer

All loop variables are digital values, no analog voltage !

ADPLL: A New Idea



ADPLL: A New Idea

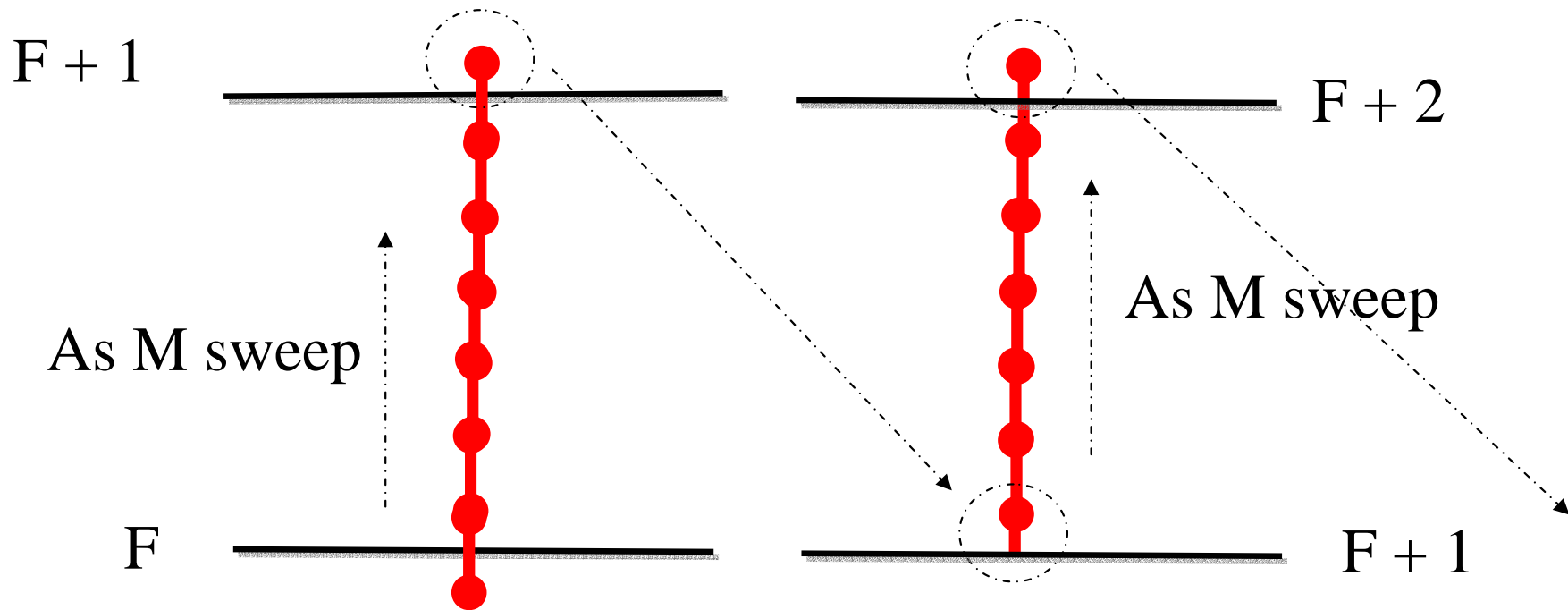
- **Goal:** $f_{out} = N * f_{in}$
- **Procedure:**
 - Using synthesizer1 to generate a known high frequency f_{hi} (e.g. > 500 MHz), by FREQ1.
 - Using f_{hi} to measure f_{in} . (a simple counter) Get a frequency number of f_{in} .
 - Multiple this frequency number by N and convert it to FREQ2.
 - Using synthesizer2 to generate the f_{out} , by FREQ2.
- **Advantage:**
 - f_{out} is not directly related to f_{in} electrically, noise in f_{in} 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.

THE TECHNOLOGY DIFFERENCE

$$F = p * M$$



p , a required frequency