A Technique for High-Speed, Fine-Resolution Pattern Generation and its CMOS Implementation

Gary C. Moyer[*], Mark Clements, Wentai Liu[†], Toby Schaffer[‡], Ralph K. Cavin, III

Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695-7911
gcmoyer@eos.ncsu.edu
Voice: (919) 515-7610
FAX: (919) 515-5523

Abstract

This paper presents an architecture for generating a high-speed data pattern with precise edge placement (resolution) by using the matched delay technique. The technique involves passing clock and data signals through arrays of matched delay elements in such a way that the data rate and resolution of the generated data stream are controlled by the difference *of these matched delays. This difference can be made much smaller than an absolute gate delay. Since the resolution of conventional designs is determined by these absolute delays, the matched delay technique yields a much finer resolution than traditional methods and, in addition, generates high data rate patterns without the need of a high-speed clock. The matched delay technique lends itself to high-precision and high-speed applications such as fast network interfaces or test pattern generators. This paper also describes a matched delay data generator submitted for fabrication in a MOSIS 1.2μm CMOS technology. This implementation uses biased delay elements to internally compensate for temperature and process variations. Simulations indicate the implementation described in this paper can generate data signals with on-chip bit rates of 833Mb/s and resolutions of 100ps.*

1: Introduction

The ability to handle very high-speed data is a significant issue in high-performance computing and communications systems. The importance of this issue has increased with the advent of powerful distributed processing systems and fiber-optic communications standards such as SONET, of which even a mid-level (OC-12) implementation requires data rates of 622 Mb/s. Currently, there is a strong trend toward implementing these high-performance systems with a network of processors rather than a single very expensive processor. A fundamental component of such a system is a high-bandwidth network interface for each processor. Such interfaces require multiplexors to combine multiple data streams at the transmitter and demultiplexers to recover the individual streams at the receiver.

[*]This research is partially supported by MIP-9212346

[†]This research is partially supported by MIP-9212346

[‡]This material is based upon work supported under a National Science Foundation Graduate Research Fellowship

Typical realizations of these high-speed multiplexors/demultiplexers require high-speed, well-controlled clocks [1] [4] [11]. Due to the complexities of generating and distributing high-speed clocks, such circuits are difficult to implement. In addition, the data resolutions of these multiplexors/demultiplexers are limited by absolute gate delays. A new circuit technique is needed to achieve high-speed and fine-resolution operation without the need of high-speed clocks.

One innovative design methodology is the *matched delay technique* [7]. Based on the wave pipelining timing methodology [5], this technique can be used to design circuits whose speed is limited by the difference of matched delay elements rather than their absolute delays. This technique also permits high-speed operation without the need of a high-speed clock. The matched delay technique can be implemented in any technology, however, it is particularly attractive with regard to CMOS since it allows the construction of high-speed circuits that can benefit from CMOS's advantages: low cost, low power, and high density.

The matched delay technique was employed in developing a high-performance digital sampler with 1 Gb/s bandwidth and 25ps resolution in a MOSIS 1.2μm CMOS process [6]. This matched delay sampler implements the demultiplexing function of a network interface by performing a serial-to-parallel conversion on an incoming data stream. The inverse of the sampler, therefore, implements the multiplexing function by generating a serial stream from parallel data. The matched delay generator described in this paper performs this function. Use of the generator and sampler together makes a high-speed CMOS network interface realizable and also forms the basis of a more general high-speed transceiver.

Another advantage of the generator's matched delay structure is that it can place edges very precisely in the data stream. This makes the generator attractive for designs requiring accurate edge placement, such as pulse-width modulation (PWM) systems [9]. This high-resolution pulse width control would also be useful for the pin electronics in high-end VLSI testers, which are currently done in expensive technologies such as GaAs and ECL [12].

This paper describes the technique and CMOS implementation of a high-speed, fine-resolution generator [3]. First, the generator's serializing technique using matched delay structures is described. This description covers both basic operation and practical continuous pattern generation. Limitations of the implementation are discussed next. A high-level overview of the architecture is then given, followed by detailed circuit descriptions. Simulation results are then presented before the conclusions.

2: Matched delay pattern generation technique

2.1: Pattern generation technique

The basic architecture of the generator is illustrated in Fig. 1. Data is fed to the inputs of an array of T-type flip-flops. A clock pulse is then sent down a chain of delay elements, all of which have delay Δ_C. Each flip-flop's clock input (except the first one's) taps this chain at the output of the corresponding delay element, so that the flip-flops are clocked in sequence $1, 2, \cdots N$ with an interval of Δ_C. Each flip-flop that has a high T input will toggle its output at the arrival of the clock pulse's rising edge. This, in turn, toggles the output of the corresponding XOR gate, which has a delay of Δ_X. This output then propagates down the XOR chain to the output. In this way edges are inserted into the serial data stream.

XOR gates are used in the data delay chain since a transition on either input switches the output. When a flip-flop output is held constant, the corresponding XOR functions simply as a delay element; the output from the previous XOR is passed through (inverted

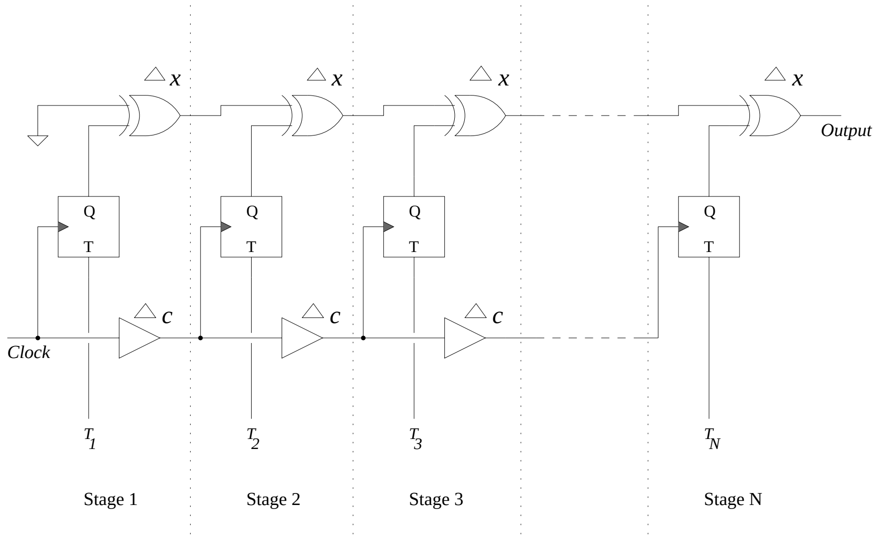

Figure 1. Pattern Generation Technique

or not depending on the value of the flip-flop) after a delay Δ_X. Alternatively, when the flip-flop toggles, it inserts an edge onto the data chain as discussed above.

To find the resolution of the generator, it helps to view the generator as a cascade of stages, each consisting of a T flip-flop, XOR gate, and clock delay element. The clock pulse arrives at stage i at time $i\Delta_C$, at which point the flip-flop can toggle its output. Since the flip-flop's delay affects each stage equally, it can be ignored. This switching output causes XOR$_i$ to switch Δ_X later, at time $i\Delta_C + \Delta_X$. Similarly, the clock arrives at stage $i + 1$ at time $(i + 1)\Delta_C$, so XOR$_{i+1}$'s output changes state at time $(i + 1)\Delta_C + \Delta_X$. However, the edge created "upstream" by XOR$_i$ arrives at XOR$_{i+1}$ at time $i\Delta_C + \Delta_X$. This causes XOR$_{i+1}$ to switch at time $i\Delta_C + 2\Delta_X$. It can be seen that the time difference between edges, Δ_{XC}, is

$$\Delta_{XC} = [i\Delta_C + 2\Delta_X] - [(i + 1)\Delta_C + \Delta_X] \quad (1)$$
$$= \Delta_X - \Delta_C \quad (2)$$

The above sequence is illustrated in Fig. 2, which shows the output of the first three stages superimposed on the clock seen at each stage. In this example, all T flip-flops are set to toggle on every clock pulse. Each stage is clocked Δ_C after the previous stage. An edge is generated at each stage Δ_X after the clock arrives. An edge propagating down the chain causes a transition at the following "downstream" stage Δ_X later. This is illustrated by the diagonal arrows on the right-hand of the figure. It is clear graphically that the width of a generated pulse is $\Delta_X - \Delta_C$.

The resolution, then, depends on the *difference* of two elements' propagation delays rather than their intrinsic delays. This is a characteristic of the matched delay technique and gives two immediate benefits: 1) finer resolution, since the difference between two elements can easily be made smaller than the inherent delay of a single element, and 2) some immunity to gradient process variation, since the matched elements can be physically close on the chip.

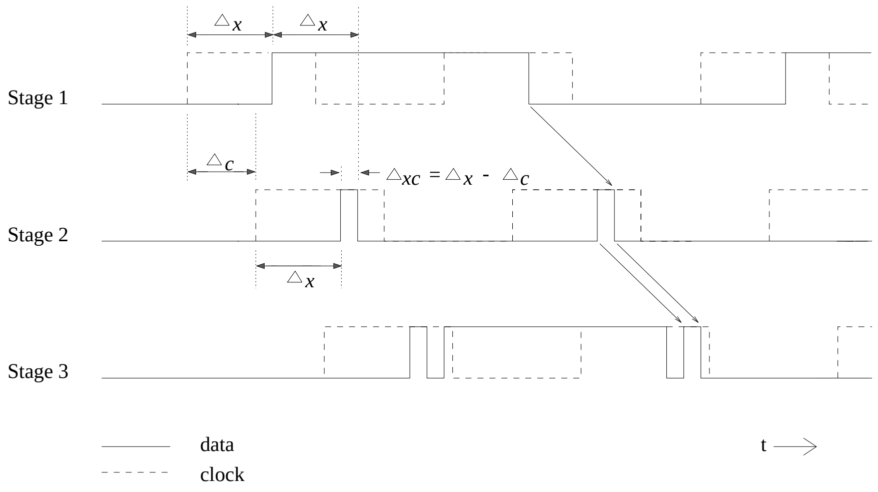

Figure 2. Creation of Pattern Using Generation Technique

2.2: Continuous pattern generation

For the generator to be useful in a practical application, it must able to generate a continuous data pattern. Since each stage can create one edge per clock period, an N-stage generator can only create edges over an $N\Delta_{XC}$ interval for each clock pulse. To make data patterns larger than this interval, the generator must be repeatedly clocked. There are two major constraints introduced by repetitive clocking: ensuring that the resolution remains a constant Δ_{XC} between edges created by different clock pulses, and guaranteeing that setup time requirements for the T flip-flops are met.

First, consider two clock pulses, p_1 followed by p_2, traveling down the clock chain of an N-stage generator. Since $\Delta_C < \Delta_X$, the edge generated at the first stage by a clock pulse is actually the last edge to reach the output, while the edge generated at stage N is the first one out. To maintain a constant data stream, the time between the last edge $e1$ due to p_1 and the first edge $e2$ due to p_2 must be the same as the time between any two edges, which is the resolution Δ_{XC}. Since an N stage generator has $N-1$ clock delay elements and N XORs, the output times of edges $e1$ and $e2$ are $N\Delta_X$ and $(N-1)\Delta_C + \Delta_X + T$, respectively, where T is the time between clock pulses, that is, the clock period. This leads to the equation:

$$\begin{aligned} \Delta_{XC} &= [(N-1)\Delta_C + \Delta_X + T] - N\Delta_X \qquad (3)\\ T &= N(\Delta_X - \Delta_C) \qquad (4) \end{aligned}$$

which shows the relationship between the number of stages, the resolution and the clock period. Our design goal was 100ps resolution. For practical reasons Δ_X and Δ_C can be neither too large nor too small, so 500ps and 400ps respectively were chosen as reasonable values. With the resolution fixed, T and N were chosen to satisfy Eq. 4. The largest possible value for N, given the available die area, was chosen to maximize T, which reduces problems associated with driving a high-speed clock onto the chip. The values for N and T for this implementation are 64 stages and 6.4ns (156.25MHz) respectively.

The second part of maintaining a constant Δ_{XC} resolution requires that the input data be presented to the T flip-flops in the fashion explained below. First, note from the above parameters that $T < N\Delta_C$. This indicates that multiple clock pulses are present at any given time, which effectively wave pipelines [5] the clock chain. The number of pulses present is given by

$$\begin{aligned} p &= N\Delta_C/T && (5)\\ &= 64 \times 400ps/6.4ns && (6)\\ &= 4 && (7) \end{aligned}$$

There is one pulse per $N/p = 16$ stages. Each 16 stages defines a *section.*

Now consider two stages i and j in adjacent sections, where $j = i + 16$. Since i and j are 16 stages apart, and $T = 16\Delta_C$, the edges created at stages i and j are generated simultaneously by two consecutive clock pulses. The edge generated at stage j arrives at the generator's output at $t_j = j\Delta_C + (N - j)\Delta_X$, while the edge generated at stage i arrives at $t_i = i\Delta_C + (N - i)\Delta_X + T$. The T accounts for the fact that the two clock pulses are separated in time by one clock period. The difference in arrival times, then, is $t_i - t_j = (j - i)\Delta_{XC} + T$. However, maintaining constant resolution requires that $t_i - t_j = (j - i)\Delta_{XC}$. It can be seen now that this occurs if and only if both edges are generated by the same clock pulse, which happens if the data sent to the second section is skewed by one clock period relative to the data sent to the first section. This one-cycle delay is accomplished by using simple static D-type flip-flops. Continuation of this reasoning leads to requiring one additional layer of these delays per section, as illustrated by the unshaded flip-flops in Fig. 3.

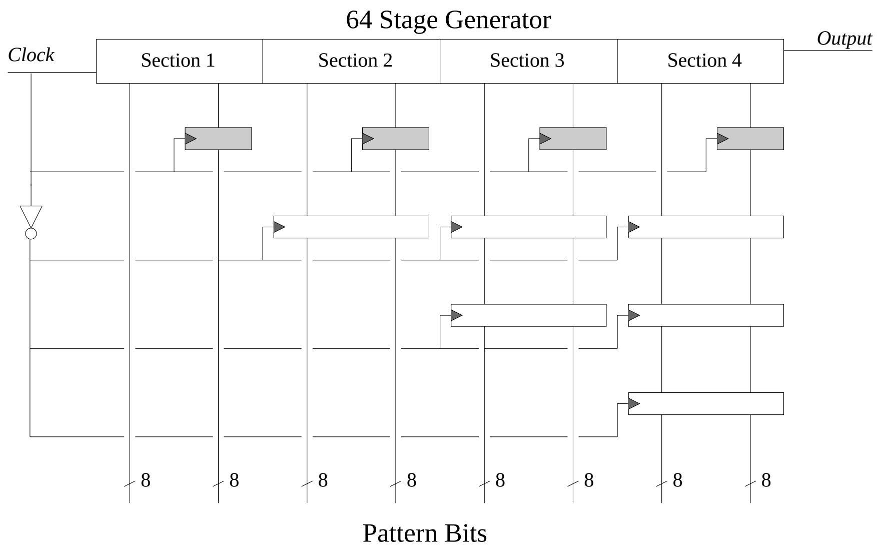

Figure 3. Latch Configuration for Continuous Pattern Generation

Setup time for the first half of each section is met by clocking the above delays on the opposite phase of the generator's clock, as shown in Fig. 3. However, the clock pulse will only be halfway through the section when the delay flip-flops are clocked again, causing the data in the second half of the section to be lost. This loss is avoided by the insertion again of one layer of delay flip-flops which are clocked in phase with the generator's clock. These delays are shaded in Fig. 3.

2.3: Limitations on resolution and data rate

Using the above technique, we have shown that the resolution of the generator is determined by the difference of delays, Δ_{XC}. It is theoretically possible to get an arbitrarily small resolution by setting Δ_X and Δ_C sufficiently close together. However, in reality, the actual values of Δ_X and Δ_C, and thus the resolution, will differ from their nominal values because of fabrication variations, noise, jitter, and temperature changes. Severe discrepancies between actual and nominal delay values will cause incorrect operation. This sets a lower limit on the resolution. It also suggests that Δ_X and Δ_C should be kept large relative to the possible delay changes due to process variations to minimize the percentage change in delay due to these variations.

The maximum frequency of the generated data pattern is also limited. The XOR gates' finite analog bandwidth causes them to act as low-pass filters on the generated data pattern. Generated data pulses that have a width below a threshold value will be attenuated completely and not make it down the XOR chain. This restricts how close edges can be placed to each other in a generated pattern, which impacts the specification of the generator's maximum output frequency. In addition, the output driver (see section 4.3) and the package itself also attenuate the output pattern, further restricting the maximum frequency.

3: Functional description of generator implementation

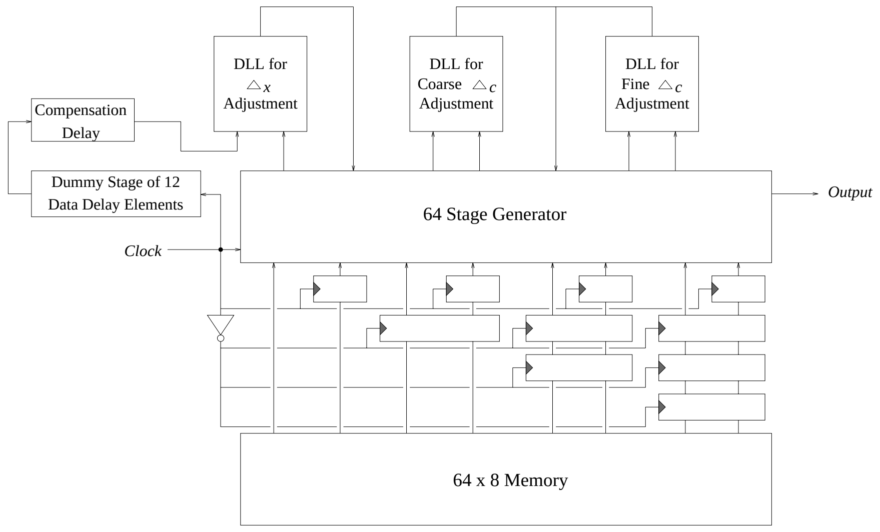
DLL for $\triangle_X$ Adjustment
DLL for Coarse $\triangle_C$ Adjustment
DLL for Fine $\triangle_C$ Adjustment
Compensation Delay
Dummy Stage of 12 Data Delay Elements
64 Stage Generator
Output
Clock
64 x 8 Memory

Figure 4. Block Diagram of Generator Implementation

A block diagram of the generator implementation is shown in Fig. 4. The 64 stage generator made up of clock and XOR delay lines as well as the T flip-flops was described in the preceding sections. This section describes two blocks used for controlling the generator: the delay-locked loops and data memory.

3.1: Delay-locked loops

As mentioned above, the actual and designed values of Δ_X and Δ_C will differ to some degree due to process and environmental variations. This indicates the need for a continuouslyoperating self-correcting circuit that maintains constant delay values across varying operating conditions. The generator uses delay-locked loops (DLLs) to accomplish this. Each DLL examines two taps on a delay chain and adjusts the delay between these two taps via a bias voltage. This adjustment locks the phase of the two taps together. By appropriate placement of the taps, the average delay of the elements between the taps, and on the entire chain, can be controlled. The connections between taps and DLL are carefully laid out symmetrically to assure that the propagation delays from both taps to the DLL are equalized.

There are two DLLs for controlling the clock chain's delay. The first does coarse adjustment, then the second does fine adjustment. The first one examines two taps 16 stages apart. As discussed previously, $16\Delta_C = T$. Therefore, the DLL locks when the average delay $\Delta_{Cavg} = T/16 = 400$ps, the desired value.

The second DLL examines taps that are 64 stages apart. This DLL makes a more accurate adjustment to Δ_C. Having the taps further apart reduces the effect of the DLL's inherent phase error, which is the phase difference in two signals that appear to be in phase to the DLL. Simulations have shown this error to be less than 100ps. By using taps that are further apart, the error's effect on the Δ_C adjustment is reduced since it is divided among more delay elements. The DLL should lock two clock pulses that are $64/16 = 4$ edges apart, but it is possible that it could lock a clock edge to one that is 3 or 5 edges away, which generates an incorrect Δ_C. To avoid this problem, the first DLL coarsely adjusts Δ_C while the second DLL is disabled. The first is then disabled, at which point the second DLL turns on and makes fine adjustments to achieve a more accurate Δ_C.

The third DLL controls the delay of the data delay chain. Since the data on this chain is unknown during operation, taps on it cannot be used for phase locking. Instead, since $16\Delta_X = 20\Delta_C$, a tap at the 20^{th} clock delay is compared to the output of an (effective) 16 data delay element chain. "Effective" means that the 16 element delay is achieved by a dummy chain of 12 data delay elements and a variable delay [8] controlled off-chip. In the layout, the wiring from the dummy chain to the DLL and from the clock delay chain to the DLL could not be done in the same metal layer and geometric shape, so a variable delay is used to compensate for any wiring delay mismatch. This compensation delay is calibrated to equal the delay through four data delay elements. The dummy chain is driven by the same clock that drives the clock delay chain. Since this DLL affects only Δ_X and has no effect on Δ_C, the stability of the clock chain DLLs is unchanged.

3.2: Input data memory

The data which are input to the generator are stored on-chip in an 8x64 memory array, organized as one 8-bit circular FIFO per stage. On-chip memory, while costly in area, is necessary since it is very difficult to input 64 bits in parallel at 156MHz. Though a larger memory would enable the generation of longer and more complex patterns, area constraints limited the number of bits per stage to eight. The storage element used to implement the FIFOs is based on that found in [10]. Bits in adjacent FIFOs represent 100ps intervals in the generated pattern, with a "1" indicating the presence of an edge, i.e., the data to be transmitted goes through an encoding stage off-chip prior to being loaded. Data bits are loaded in parallel into the FIFOs. The FIFOs circulate their contents one position and present a new bit to the generator every clock period. Changing the generated pattern requires disabling the generator and loading new data into the memory. It should be noted that it is our implementation, not the generator technique itself, that requires the generator to be disabled when changing the data memory contents.

4: Circuit design of generator and control

This section describes the circuits used in implementing the parts of the generator that most heavily influence performance: the generator stages (XOR and clock delay elements), DLLs, and output drivers.

4.1: Generator stage

As mentioned earlier, a generator stage consists of a rising-edge triggered toggle flip-flop, clock delay element, and data delay element. A gate-level representation of a stage is shown in Fig. 5.

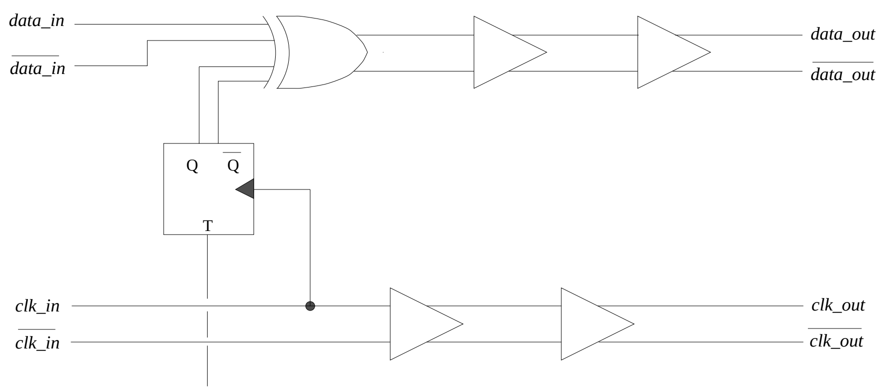

Figure 5. One Stage of Pattern Generator

The clock delay for each stage consists of two differential delay elements. Another pair of these delay elements following a differential XOR gate form each stage's data delay. Two delay elements were used in each case to increase the range of each line's adjustable delay, which improves the ability to compensate for process and environmental variations. The clock line delay was designed to be adjustable from 300ps – 500ps, which is centered around the desired $\Delta_C = 400$ps. The data line's delay was designed to be 400ps – 600ps, which centers around $\Delta_X = 500$ps. The two components of the clock and data delay, the XOR gate and the differential delay element, are described below.

The XOR gate was implemented as illustrated in Fig. 6. It is the bottleneck component of the data delay, since, as the slowest gate in the data delay line, it limits the maximum bit rate of the generated pattern. High bandwidth is therefore of critical importance. Two other important factors were noise immunity and the ability to pass both rising and falling edges with equal delay, i.e., gate delay independence from the input data pattern. A differential implementation was chosen over single-ended since differential logic is superior in all three respects.

The output swing of the XOR was set to be approximately 1.2V - 4.9V (when biases $V_{XP} = V_{XN} = 2.5$V). This swing gives better bandwidth than full-swing and is wide enough to provide good noise immunity. The pull-down path is symmetrically balanced so that there is an equal capacitance to discharge regardless of the a and b inputs, resulting in less data dependence.

The biases V_{XP} and V_{XN} provide very coarse adjustment of the XOR's delay and are intended to stay constant during generator operation. The primary means of adjusting Δ_X are the delay elements described below.

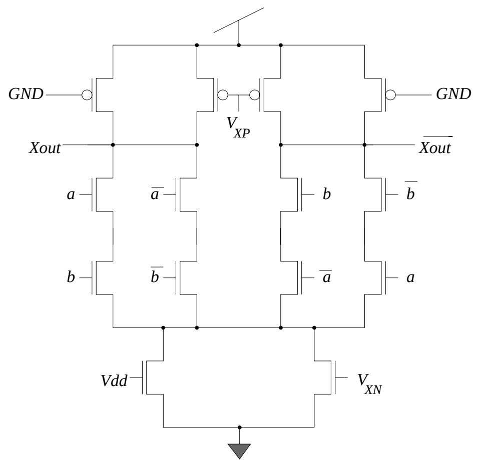

Figure 6. Differential XOR Gate

The differential delay element is used in both the clock and data delay chain. The main criteria for the delay element's design were delay range, controllability of delay, and noise immunity. These requirements led to the design shown in Fig. 7. This gate, like the XOR gate, is differential to increase noise immunity and bandwidth. Its delay is adjusted by biases V_{DP} and V_{DN}. Having two biases increases the delay range while keeping its output swing of approximately 1.2V - 4.9V. Decreasing (increasing) V_{DP} and increasing (decreasing) V_{DN} decreases (increases) the gate's delay. To ease the difficulty of having to control two biases, an automatic bias-controller [8] was used. This circuit, illustrated in Fig. 8, uses a comparator to equalize the voltages V_{div} and $V_{dd}/2$ by increasing (decreasing) V_{DN} in response to decreases (increases) in V_{DP}. With this bias-controller, the DLL need only adjust one voltage, V_{DP}, to change the delay. The effect of V_{DP} on both the clock and data delay of a single stage is shown in Fig. 9.

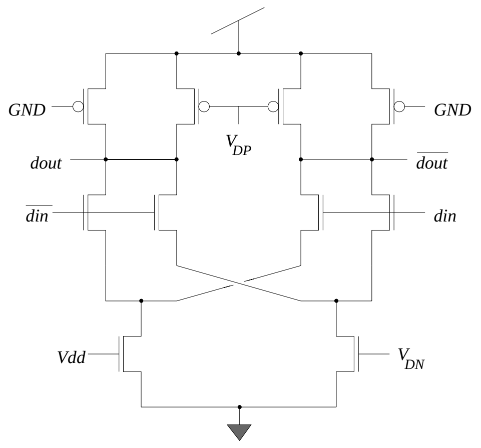

Figure 7. Differential Delay Element

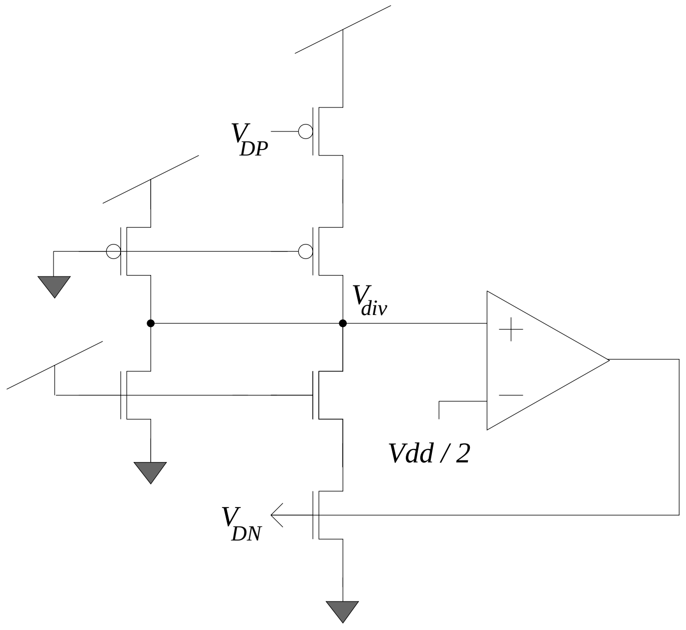

Figure 8. Automatic Bias-Controller

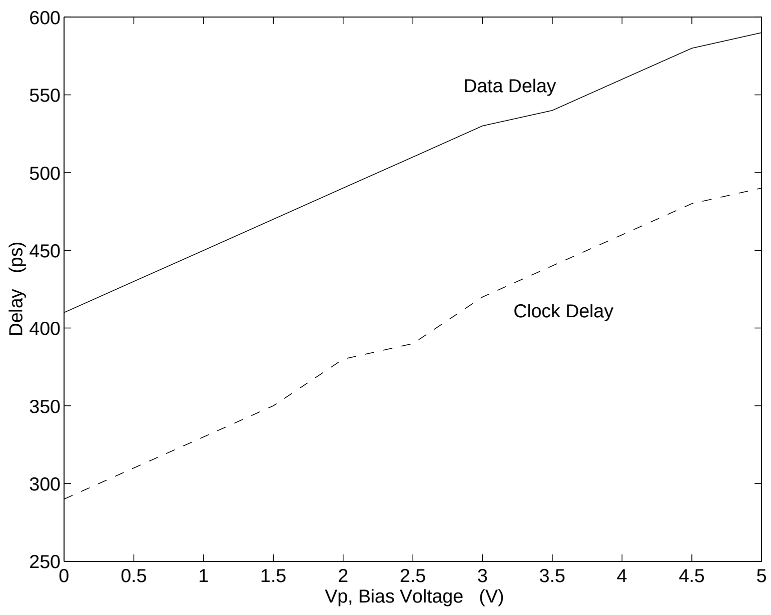

Figure 9. Bias Voltage’s Effect on Delay

4.2: DLL controllers

A schematic of the DLL controller appears in Fig. 10. The DLL consists of a phase detector, a charge pump and a loop filter.

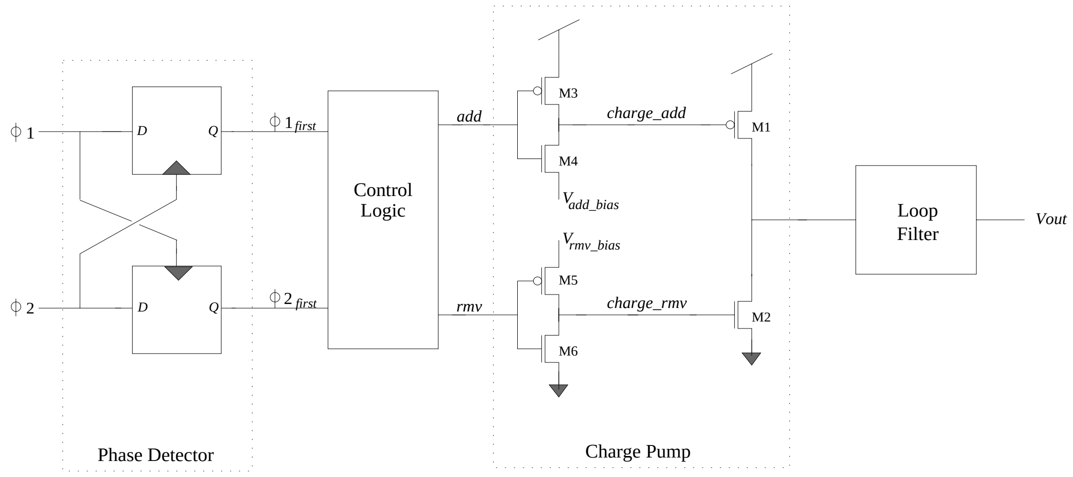

Figure 10. Diagram for DLL Delay Controller

The phase detector is composed of two flip-flops connected as shown in Fig. 10. The flip-flops are based on those described in [10]. The detector determines the phase relationship between two signals by examining their rising edges and indicates which edge arrived first by setting $\phi 1_{first}$ and $\phi 2_{first}$ as shown in Table 1.

Input Signals	Detector Outputs	
	$\phi 1_{first}$	$\phi 2_{first}$
$\phi_1 \uparrow$ before $\phi_2 \uparrow$	1	0
$\phi_2 \uparrow$ before $\phi_1 \uparrow$	0	1
$\phi_1 \uparrow$, $\phi_2 \uparrow$ together	0	0
invalid case	1	1

Table 1. Phase Detector Operation

The detector sets both outputs low when both inputs appear to be in phase. Edges whose arrivals differ by 100ps or less appear in phase to the detector. The resolving time for the flip-flops used in the edge detector is 2.24ns, and inverters following the flip-flops are used to filter out metastable states.

The $\phi 1_{first}$ and $\phi 2_{first}$ outputs of the phase detector pass through control logic which filters out any erroneous “1–1” states and creates the *add* and *rmv* signals to operate the charge pump. These signals are converted within the charge pump to *charge_add* and *charge_rmv*, which control the flow of charge into and out of the loop filter via pull-up M1 and pull-down M2. This then adjusts the output voltage V_{out} up and down. Though these control signals might seem redundant, their use becomes clear after examining transistor pairs M3–M4 and M5–M6.

When *add* is off, M3 sets *charge_add* $= V_{dd}$, which turns the PMOS M1 off. However, when *add* is on, *charge_add* $= V_{add\_bias}$, which is set off-chip. When V_{add_bias} is 0V, M1 adds charge at the maximum rate. As V_{add_bias} is increased and *charge_add* approaches

$V_{dd} - V_{th}(M1)$, the rate of charge addition is reduced. This permits finer adjustment of V_{out}. Another benefit is that the charge flow through M1 is more constant since M1 stays in the saturation region over a wider range of V_{out}. Transistor pair M5–M6 control the NMOS M2 similarly by generating *charge_rmv* from *rmv* and bias voltage V_{rmv_bias}. Figure 11 shows the effect of V_{add_bias} and V_{rmv_bias} on the rates of charge addition and removal.

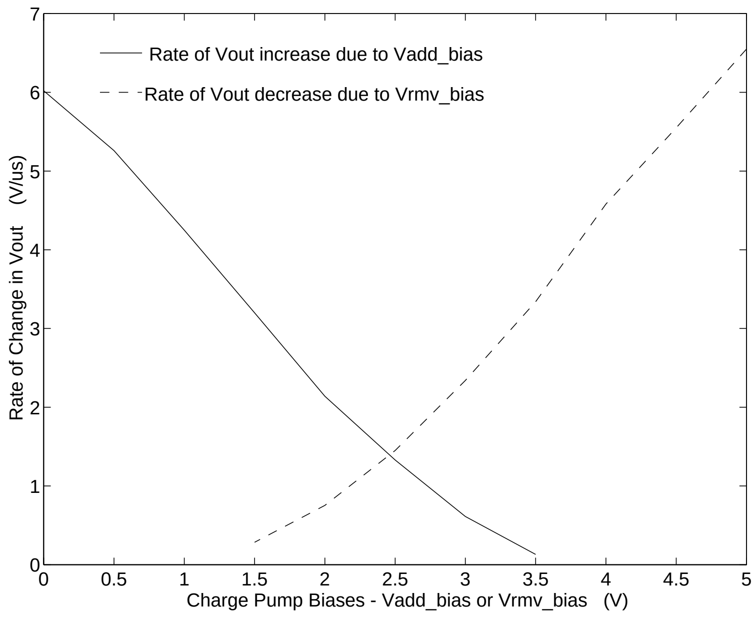

Figure 11. Effects of V_{add_bias} and V_{rmv_bias} on V_{out}'s rate of change

The loop filter is configured off-chip and takes one of two forms [2]. The first is a simple RC filter, which is easy to implement and permits V_{out} to take a wider range of values. Since this is only a first-order filter, it is fairly susceptible to noise. The second possible filter is a more complex active filter, which greatly reduces noise in V_{out}. However, the range of the bias voltage is dependent on the performance of the op-amp.

4.3: Output drivers

There is one cascaded chain of sized single-ended buffers for each of the generator's differential outputs. High bandwidth through the output pins and low power consumption were the main goals in sizing the drivers. These goals conflict, though, since power consumption increases as bandwidth rises. Power consumption is limited by the number of power pins available per driver and the size of the power busses connected to the driver. This limit on the maximum power in turn limits the maximum bandwidth. Due to a limited number of power pins per driver and the need to avoid possible electromigration effects, the drivers were sized to drive a 200pF load at bit rates up to 80Mb/s.

To fully exploit the generator's bandwidth, a pair of output pads suitable for use with high-speed, low-impedance micro-probes was also placed on-chip. The drivers for the probe pads are the same type as those used to drive the package pins. The loading on the probe pad drivers, about 10pF, is significantly lower than that on the pin drivers, so they can drive data at rates up to 833 Mb/s, the generator's maximum internal bit rate.

Either set of drivers can be disabled to eliminate any noise caused by their switching.

5: Simulation results

The generator and supporting test structures were implemented in a full-custom layout, and simulations were run on the extracted circuits with CAzM, a Spice-like simulator. The chip has 32,071 transistors and occupies a die area of 2.71mm $\times$ 6.15mm. A chip plot of the generator's implementation appears in Fig. 12.

Figure 12. Chip Plot of Generator's Implementation

Fig. 13 shows the generation of the maximum bit rate pattern by examining the taps on the data delay chain at the output of the first, sixteenth, thirty-second and last stages. Edges are added as the data flows downstream, culminating in a 833 Mb/s bit rate pattern. This corresponds to a 1.2ns minimum pulse width, meaning an edge is inserted into the generated waveform every twelve stages. This pattern is represented in memory by continuous repetition of the bits 100000000000.

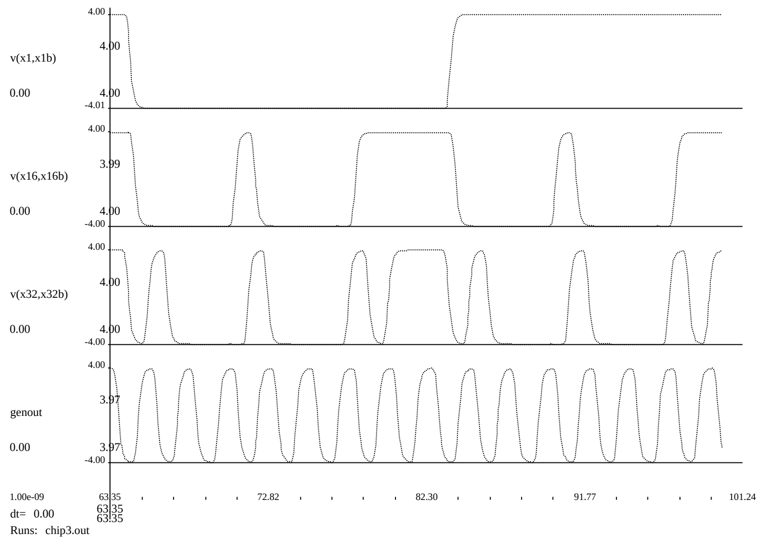

Figure 13. Generation of Maximum Frequency Pattern

Fig. 14 illustrates the generator's edge placement ability by showing two 1.2ns pulses followed by two 1.3ns pulses. These pairs differ in width by the maximum resolution, 100ps.

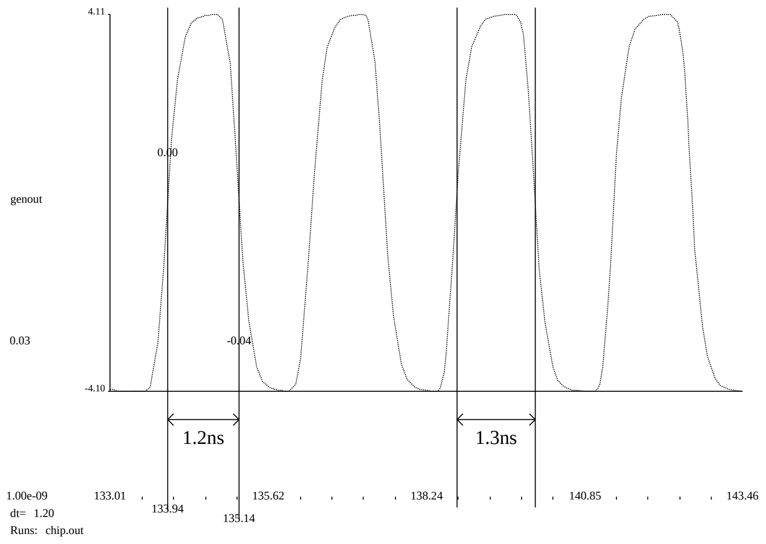

Figure 14. Illustration of Maximum Resolution

6: Conclusion

A technique for generating high-speed, fine-resolution data patterns without the need for a high speed clock has been presented. The technique can be used to generate continuous data patterns suitable for use in a data transmitter. The resolution in the generated patterns is determined by the difference of delay in two delay elements, rather than an absolute delay. Jitter and noise set an upper limit on maximum resolution, and the maximum bit rate is limited by the analog bandwidth of the data delay chain gates.

An implementation of this technique has been submitted for fabrication in a MOSIS 1.2μm CMOS technology. Simulation results of the extracted layout show patterns can be generated with bit rates up to 833Mb/s and edges placed with 100ps resolution. Using a more advanced CMOS process should push this rate up to the SONET OC-48 level (2.56 Gb/s). The generator also has self-correcting circuitry to compensate for process and environmental variations.

References

[1] R. J. Bayruns, E. A. Hofstatter, and H. T. Weston. A Fine-Line NMOS 3-Gbit/s 12-Channel Time-Division Multiplexer-Demultiplexer Chip Set. *IEEE Journal of Solid-State Circuits*, 24:814–812, 1989.

[2] R. Best. *Phase-Locked Loops: Theory, Design, and Applications*. McGraw-Hill, Inc., New York, 1984.

[3] M. Clements, W. Liu, R. Cavin, G. Moyer, and J. Kang. The Matched Delay Data Generator, April 1994. NCSU Patent Disclosure No. 94-68.

[4] A. E. Dunlop, T. J. Gabara, and W. C. Fischer. A 9 Gbit/s Bandwidth Multiplexer/Demultiplexer CMOS Chip. In *1992 Symposium on VLSI Circuits Digest of Technical Papers*, pages 68–69, 1992.

[5] C. T. Gray, W. Liu, and R. K. Cavin III. *Wave Pipelining: Theory and Implementation*. Kluwer Academic, 1993.

[6] C. T. Gray, W. Liu, W. A. M. van Noije, T. A. Hughes, and R. K. Cavin. A Sampling Technique and Its CMOS Implementation with 1 Gb/s Bandwidth and 25ps Resolution. *IEEE Journal of Solid-State Circuits*, 29:340–349, March 1994.

[7] W. Liu, M. Clements, and R. K. Cavin III. The Matched Delay Technique: Theory and Practical Issues. Technical Report NCSU-VLSI-93-23, North Carolina State University, 1994.

[8] G. Moyer, W. Liu, R. K. Cavin III, and T. Schaffer. A High Speed CMOS Clock Shaper Using Wave Pipelining. Technical Report NCSU-VLSI-93-11, North Carolina State University, 1993.

[9] K. Nogami and A. El Gamal. A CMOS 160Mb/s Phase-Modulation I/O Interface Circuit. In *ISSCC Digest of Technical Papers*, pages 160–161, February 1994.

[10] R. Rogenmoser et al. 1.16 GHz Dual-Modulus 1.2 μm CMOS Prescaler. In *Proceedings of the IEEE 1993 Custom Integrated Circuits Conference*, pages 27.6.1–27.6.4, 1993.

[11] R. G. Swartz. Ultra-High Speed Multiplexer/Demultiplexer Architectures. *International Journal of High Speed Electronics*, 1:73–79, 1990.

[12] S. Taylor. A High-Performance GaAs Pin Electronics Circuit for Automatic Test Equipment. *IEEE Journal of Solid-State Circuits*, pages 1023–1029, October 1993.