J Electron Test (2015) 31:549–559
DOI 10.1007/s10836-015-5554-0

# Design and Implementation of an FPGA-Based Data/Timing Formatter

**Yu-Yi Chen[1,2] · Jiun-Lang Huang[1] · Terry Kuo[2] · Xuan-Lun Huang[3]**

Received: 27 June 2015 / Accepted: 17 November 2015 / Published online: 3 December 2015
© Springer Science+Business Media New York 2015

**Abstract** The data/timing formatter is a key module in automatic electronics test equipment; it formats the test data to the desired wave shape and places the timing edges at the designated locations. In this work, we investigate the design and implementation of the FPGA-based data/timing formatter. Compared to its ASIC counterpart, the FPGA-based formatter is more flexible because it can be reconfigured to best fit the target test specifications. However, routing uncertainty and limited types of available logic and interconnect resources also pose great challenges. This work proposes a formatter design that is suitable for FPGA implementation. Several high-linearity FPGA-based programmable delay lines are developed. According to its characteristics, each type of delay lines is assigned a different role in the formatter. The formatter is also equipped with a calibration unit to further improve the edge placement resolution and accuracy. A 100-Msps FPGA-based data/timing formatter with 20-ps edge placement resolution has been implemented on an FPGA development board to validate our ideas.

Responsible Editor: V. D. Agrawal

✉ Jiun-Lang Huang
jlhuang@ntu.edu.tw

[1] Graduate Institute of Electronics Engineering, Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

[2] OpenATE Inc. Taipei, Taiwan

[3] Industrial Technology Research Institute, Zhudong, Hsinchu, Taiwan

**Keywords** FPGA · Timing circuit · Test equipment · Data/timing formatter

## 1 Introduction

For digital IC testing, to guarantee that AC parameters are fully tested according to the specifications, the wave shape, edge placements, and voltage levels of the input signals supplied to the device under test (DUT) must meet the test specific requirements [14]. In automatic test equipment (ATE), these are handled by the data and timing formatter (called formatter hereafter for convenience). Due to the high timing accuracy requirements, ASIC design is the mainstream for (data/timing) formatter implementation. In [16, 17], the authors presented an 800-Msps (Mega symbol per second) formatter with 81-ps resolution and the corresponding calibration procedures. In [13], the reported formatter speed is 100 to 250 Msps with a 20-ps edge setting resolution. The Analog Devices ADATE207 quad pin timing formatter [2] is capable of 100 Msps and 39.06 ps resolution. Several ATE vendors are also known to have their own formatter ASICs.

In this work, we are interested in FPGA-based formatter implementation; the goal is to offer the wave shaping and edge placement functions. Compared to their ASIC counterparts, FPGA-based formatters have the following advantages.

- There are no NRE (non-recurring engineering) expenses and manufacturing delay.
- One can take advantage of the FPGA off- and on-line re-configuration capability and optimize the formatter parameters or even architectures according to the test specifications.

- Modern FPGAs support millions of logic gates, memory blocks, high-speed serial I/Os and even embedded processors; this allows one to realize the calibration circuits (for resolution and accuracy enhancement) on the FPGA.

FPGA-based formatters, on the other hand, have the following limitations.

- The maximum operating speed is bounded by those of available FPGAs.
- The FPGA intrinsic jitter degrades the timing accuracy.
- A custom design flow that involves low level FPGA primitives is needed because direct mapping of timing circuits onto FPGA is in general nontrivial.
- The choice of timing circuit architectures is limited by the FPGA logic and routing resources.

### 1.1 FPGA-Based Timing Circuits

Recently, several FPGA-based timing circuits have been reported [3, 4, 6–10, 15, 18–20]. In [9], a 250-ps resolution programmable delay line is realized by utilizing MUX-based delay elements. The time-to-digital converter in [6] can achieve very high resolution (up to 1.58 ps); however, the pulse width resolution depends on the FPGA performance and the latency is input dependent. In [7], coarse and fine delays with 105-ps and 13-ps resolutions are reported. In [3, 4, 8, 15, 18, 20], FPGA-based time-to-digital conversion (TDC) circuit designs and calibration techniques are reported.

Several attempts have been made to realize low-cost FPGA-based ATE [5, 11, 12]. In [11, 12], the FPGA-based ATE applies the stored patterns to the DUT and samples the DUT output responses. The prototype in [12] operates at a maximum speed of 6 MHz, supports three data types, including RZ (return to zero), NRZ (non return to zero), and RO (return to one), and has a timing resolution of 16.66 ns, which is one-tenth of the operating speed. In [5], an FPGA-based test and diagnosis platform for SRAM was presented.

While the performance of the FPGA-based testers is well behind that of high-end testers that use ASIC solutions, FPGA-based testers are suitable for (1) IC validation by ASIC designers during the development phase, (2) test program development and validation, and (3) manufacturing testing of low to mid-end ICs.

### 1.2 The Proposed FPGA-Based Data/Timing Formatter

The capability of precise timing edge placement is crucial for ATE to fully test the DUT's timing related specifications. In high-end ATE, precise timing is achieved by dedicated ASICs, which is impossible for FPGA-based ATE.

In this work, we focus on the ATE data/timing formatter which converts the test data to the desired format and places the timing edges at the specified locations. Compared to [12] that can only align wave transitions to the FPGA internal clock edges, the proposed formatter utilizes high-resolution FPGA-based programmable delay lines to provide the needed timing resolution. As a result, it delivers much higher edge placement resolution — 20 ps vs. 16.66 ns in [12].

A prototype formatter is implemented on an Altera DE-II development board; it achieves a 100-Msps symbol rate with 20-ps edge placement resolution. The contributions are as follows.

- The choice of FPGA-based programmable delay lines is limited by the FPGA architecture. We develop three different programmable delay lines. According to their characteristics, they are assigned different roles in the formatter to improve the FPGA resource usage efficiency.
- Alignment and characterization circuits are developed to push the edge placement resolution beyond the FPGA incurred limits, including routing, placement, and logic resources.

The rest of the paper is organized as follows. In Section 2, we give a brief overview of the data/timing formatter and the Altera FPGA architecture. The proposed formatter architecture and implementation details are illustrated in Section 3. Experimental results are shown in Section 4 and we conclude this work in Section 5.

## 2 Preliminaries

### 2.1 The Data/Timing Formatter [14]

Figure 1a depicts the typical data/timing formatting process. Many test signals require, in addition to the logic value, the unique timing (signal transition points), format (wave shape), and voltage levels (VIL/VIH). For example, to specify a symbol which is a one-pulse from 30 % to 80 % of a clock cycle, the logic value is set to one, the two timing edges (rising and falling) are at 30 % and 80 % of the clock cycle, respectively, and the data format is "return to zero" (RZ in Fig. 1b).

The commonly used data formats include NRZ (non return to zero), DNRZ (delayed non return to zero), RZ (return to zero), RO (return to one), SBC (surround by complement), and ZD (impedance drive); some of them are depicted in Fig. 1b.

Finally, the output driver forces the specified high and low voltage levels (VIH and VIL) according to the DUT specifications.

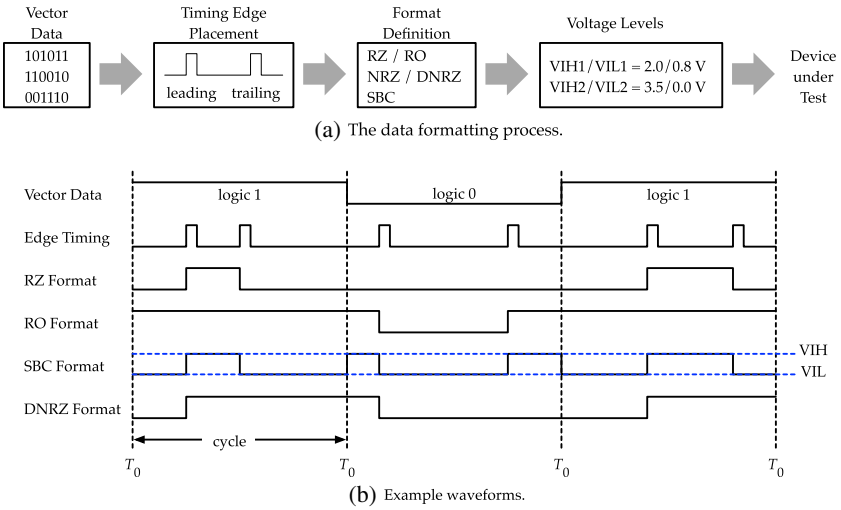



**Fig. 1** The **a** data formatting process and **b** example waveforms

(a) The data formatting process.

(b) Example waveforms.

By properly specifying the logic value, the edge locations, the data format, and the voltage levels, the desired AC test waveform can be generated.

### 2.2 The Programmable Delay Line

For programmable delay lines, the main performance parameters include resolution, linearity, minimum delay, and dynamic range.

Consider a programmable delay line that can produce $k$ different delay values, $\tau_0, \tau_1, \cdots, \tau_{k-1}$. It has a *minimum delay* of $\tau_0$ and a *dynamic range* of $\tau_{k-1} - \tau_0$. Using the end-point line, which is suitable for the proposed formatter as the fit line, the resolution $\tau_{\text{LSB}}$ is defined as

$$\tau_{\text{LSB}} = \frac{\tau_{k-1} - \tau_0}{k-1}. \tag{1}$$

The delay line linearity is quantified by integral non-linearity (INL) and differential non-linearity (DNL). INL describes the deviation of the actual delay values from the ideal values. For input code $i$, it is defined as

$$\text{INL}\,(i) = \frac{(\tau_i - \tau_0) - i \cdot \tau_{\text{LSB}}}{\tau_{\text{LSB}}}\text{LSB} \tag{2}$$

where $1\text{LSB} = \tau_{\text{LSB}}$. DNL, on the other hand, describes the uniformity of the delay values, i.e.,

$$\text{DNL}\,(i) = \frac{(\tau_{i+1} - \tau_i) - \tau_{\text{LSB}}}{\tau_{\text{LSB}}}\text{LSB}. \tag{3}$$

Finally, we further define the overhead ratio $\Omega$ of a programmable delay line as

$$\Omega = \frac{\tau_0}{\tau_{k-1} - \tau_0}. \tag{4}$$

The overhead ratio characterizes the minimum delay needed for a delay line to achieve the specified dynamic range. It should be as small as possible.

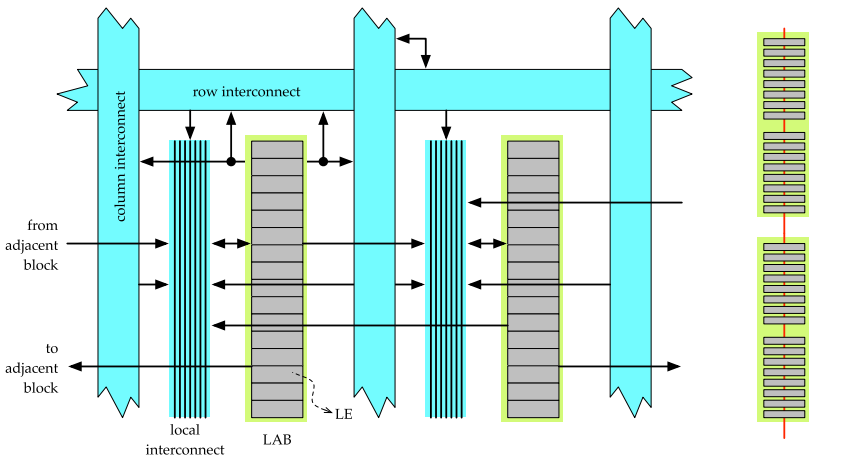



**Fig. 2** The Cyclone II LAB structure (**a**) and irregularity (**b**)

(a) Cyclone II LAB structure.

(b) LAB irregularity.

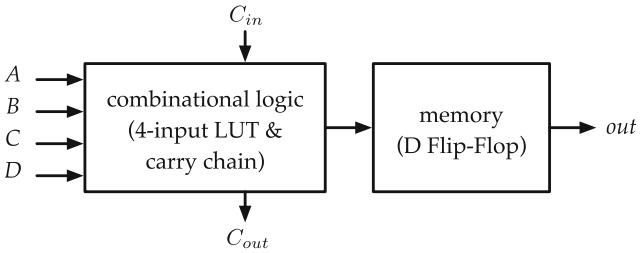



**Fig. 3** The simplified LE diagram

### 2.3 The FPGA Architecture

In this work, we implement the proposed FPGA-based formatter on an Altera DE II development board; the embedded FPGA is Cyclone II 2C70. As shown in Fig. 2a [1], inside the FPGA, the basic logic constructing block is Logic Element (LE). A simplified LE schematic is shown in Fig. 3 [1] — the combinational part consists of a four-input lookup table (LUT) and the carry chain logic; the memory portion has a D-type flip-flop. A Logic Array Block (LAB) consists of sixteen LEs. The routing resources include row interconnect, column interconnect, local interconnect, and inter-LAB interconnect.

Note that there is a special carry chain interconnect (from $C_{out}$ to $C_{in}$) between adjacent LEs; this carry chain interconnection is LAB internal and unaffected by the physical design "place & route" process. As stated in [19], FPGA delay lines should utilize the dedicated carry chain to connect delay elements; this takes advantage of the regular LAB structure and avoids the routing uncertainty which degrades the delay line linearity. In Fig. 2b, the thick red line indicates the intra and inter-LAB carry chain. As shown, there is a larger carry chain gap inside each LAB and an even larger gap between LABs. These larger gaps result in more interconnect delay and introduce non-linearity to delay lines that use carry chains to connect delay elements.

In general, the intrinsic delay line resolution is limited by the FPGA architecture, physical implementation, and routing uncertainty; as a result, post processing and calibration become mandatory for FPGA-based delay lines.

## 3 The Proposed Formatter

The proposed formatter architecture is illustrated in Fig. 4; it consists of a multi-phase clock generator, alignment delay lines $AD_i$'s, symbol generators $G_i$'s, a symbol combiner, and the calibration unit.

The symbol generators are the main components of the proposed formatter; they are responsible for generating and placing the timing edges (rising or falling) of a symbol at the specified locations to produce the target waveforms. The $g$ symbol generators, $G_0, G_1, \cdots, G_{g-1}$, operate in a time-multiplexed manner, i.e., symbol generator $G_i$ is responsible for generating symbols $S_{n\cdot g+i}$ where $n$ is a non-negative integer. Figure 5 gives an example of the time-multiplexed operation for $g = 3$. In this example, $G_0$ generates symbols $S_{3n}$, $G_1$ generates symbols $S_{3n+1}$, and $G_2$ generates symbols $S_{3n+2}$.

Note that the output of a symbol generator stays at constant zero or one except for during its assigned symbols. Since the symbol durations do not overlap, a simple XOR gate can be used as the symbol combiner to combine the symbols generated by the $g$ generators.

The FPGA built-in multi-phase clock generator is utilized to generate the $g$ clock signals, $\phi_0, \phi_1, \cdots, \phi_{g-1}$, with evenly spaced phases. Between the clock generator and each symbol generator $G_i$, there is an alignment delay line $AD_i$ to compensate for (1) the skew between the paths from the clock generator to the symbol generators, and (2) the mismatch between symbol generators. The output of $AD_i$ is denoted by $\hat{\phi}_i$. As shown in Fig. 5, each rising edge of $\hat{\phi}_i$ triggers the generation of a new symbol by generator $G_i$

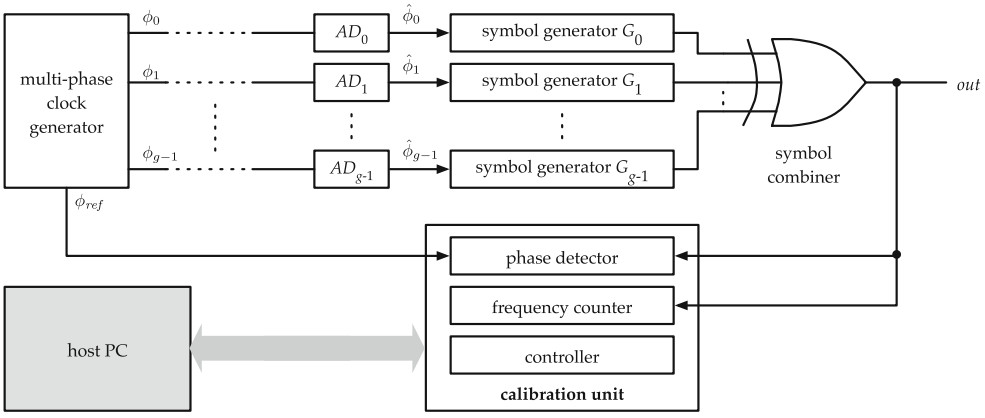



**Fig. 4** The proposed formatter architecture

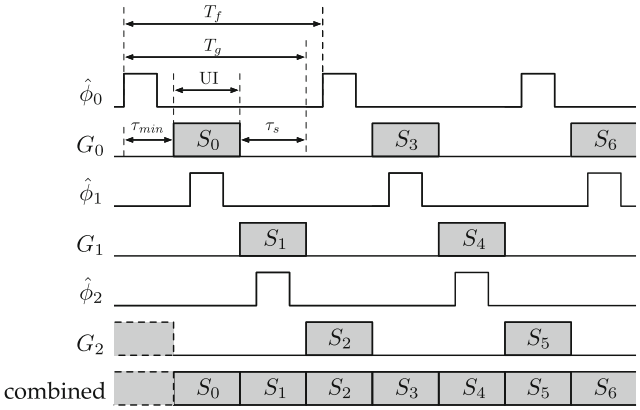



**Fig. 5** Time-multiplexed symbol generation

(after $\tau_{min}$, to be discussed later). A reference clock signal $\phi_{ref}$ which combines $\phi_i$'s is also generated to facilitate symbol generator alignment.

Due to the FPGA structure irregularity (Fig. 2b) and the routing uncertainty, calibration is mandatory for FPGA-based timing circuitry. The proposed formatter incorporates a calibration unit to (1) align the symbol generators to produce non-overlapping symbols, and (2) measure the delay values of the programmable delay lines inside each symbol generator for linearity enhancement.

Currently, a host PC is utilized to realize most of the digital operations, including

- issuing the formatter control signals for calibration and functional operations,
- analyzing the delay line characterization results and deriving the calibration parameters, and
- converting the symbol information, including the value, format, and timing, into symbol generator edge enable and delay control signals.

Nonetheless, with the abundance of FPGA digital resources, one can also implement these functions on the FPGA.

In the following, we will give more details of the symbol generator and the calibration unit.

### 3.1 Determination of Required Symbol Generators

Due to the inevitable dead-time associated with a symbol generator, the formatter has to incorporate multiple symbol generators that work in a time-multiplexed manner. The dead-time of a symbol generator consists of the following (Fig. 5).

- $\tau_{min}$: the minimum delay from the input to the output of a symbol generator.
- $\tau_s$: the setup time needed for a generator to be ready to generate next symbol, which includes (1) the time needed for an input clock pulse to leave the generator after the symbol duration ends, (2) the time to load the new control and delay signals for next symbol, and (3) the time for the delay line to settle to the new setup.

Denote by $T_g$ the time a symbol generator takes to generate a symbol. We have

$$T_g = \tau_{min} + \text{UI} + \tau_s, \tag{5}$$

where UI (unit interval) is the symbol length, e.g., UI equals 1 $\mu$s for 1-Msps symbol rate. Then, the minimum number of symbol generators that a formatter needs is

$$g = \left\lceil \frac{T_g}{\text{UI}} \right\rceil \tag{6}$$

and the resulting formatter period, $T_f$, is

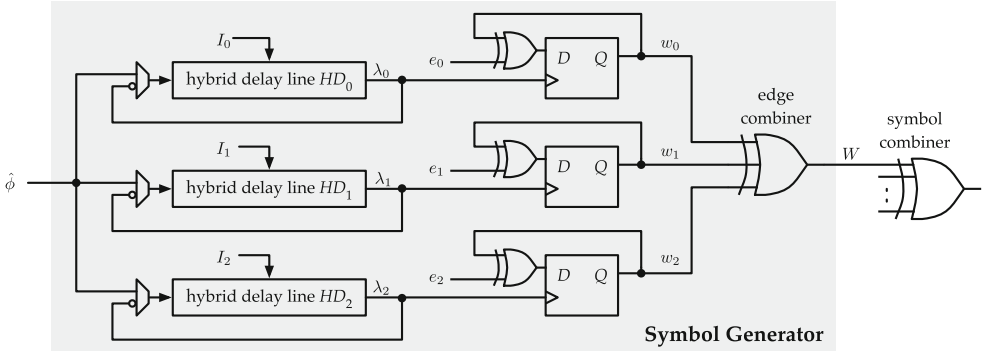
$$T_f = g \cdot \text{UI}. \tag{7}$$

For example, to achieve a 100-Msps symbol rate, i.e., UI = 10 ns, if the symbol generator dead-time is $\tau_{min}+\tau_s = 15$ns, the formatter will need at least $g = \lceil (10+15)/10 \rceil = 3$ generators and it will operate at 33.3 MHz, i.e., $T_f = 30$ ns.

### 3.2 The Symbol Generator

Figure 6 shows the symbol generator structure. The input clock signal $\hat{\phi}$ is delayed by the three hybrid delay lines, $HD_i$'s, to generate three delayed clock signals, $\lambda_0$, $\lambda_1$, and $\lambda_2$. Three delay paths are needed because a symbol may

**Fig. 6** The symbol generator structure



have a maximum of three transitions (Fig. 1b). The XOR gate and the D flip-flop at the end of a hybrid delay line determine whether to pass or block a clock pulse. To pass a clock pulse, the enable signal $e_i$ is set to one; this way, the D flip-flop and the XOR gate together convert a delayed clock pulse in $\lambda_i$ into an edge in $w_i$. On the contrary, setting $e_i$ to zero blocks the clock pulse. The delay line inputs, $I_i$'s, set the edge locations. Finally, the edge combining XOR gate combines $w_i$'s to form the specified symbol.

A multiplexer is added in front of each hybrid delay line to facilitate calibration. During delay line characterization, it will select the complemented delay line output to form an oscillator for delay value measurement. (The circle at the multiplexer input indicates inversion.)

For each symbol, the transition that occurs at its time 0 is called edge 0, and the following transitions are called edge 1 and 2, respectively. In the proposed symbol generator, edge $i$ will be placed in $w_i$. If edge $i$ exists in a symbol, $e_i$ will be set to one to generate a transition (rising or falling) at $w_i$. It should be noted that each symbol has at most one transition in $w_i$.

Consider a formatter that utilizes four symbol generators. Figure 7 shows an example timing diagram for symbol generator $G_0$. In this example, two symbols are generated by $G_0$: RZ/1 (RZ with logic value one) and SBC/0 (SBC with logic value 0). In the generator output waveform $W$, the grey regions denote the durations where $G_0$ should place the symbols assigned to it — the other three symbol generators are responsible for the gaps between grey regions. To generate the RZ/1 symbol, only edges 1 and 2 are required. Thus, before the $\hat{\phi}_0$ pulse arrives, $e_0$ is set to 0, $e_1$ and $e_2$ are set to one, and $I_1$ and $I_2$ are set according to the desired edge locations. This way, the rising edges of the delayed pulses on $\lambda_1$ and $\lambda_2$ will be where the two RZ/1 transitions should be. The $\lambda_1$ and $\lambda_2$ pulses are further converted into transitions and then combined to form the RZ/1 symbol. Generation of the SBC/0 symbol is similar and not repeated here.

Note that while a transition in $w_i$ corresponds to a transition in $W$, the transition directions (rising or falling) may be different. Take the RZ/1 symbol in Fig. 7 for example. Its rising and falling edges are produced by the rising edges on $w_1$ and $w_2$, respectively.

### 3.3 The Calibration Unit and Host PC

In the proposed formatter, the calibration unit and the host PC together are responsible for (1) symbol generator alignment, (2) delay line characterization, and (3) calibration parameter derivation.

#### *3.3.1 Symbol Generator Alignment*

The goal of symbol generator alignment is to make sure that the output symbols from different symbol generators do not overlap (Fig. 5). Symbol misalignment occurs due to the following reasons.

- Skew-induced $\hat{\phi}_i$ misalignment:
  Ideally, the phase differences between $\hat{\phi}_i$'s should be the same as those between $\phi_i$'s. In practice, this is impossible due to routing uncertainty and process variations.
- $\tau_{min}$ variation:
  From Fig. 5, $\tau_{min}$ mismatch will also cause symbol misalignment.

In the proposed formatter, symbol generator alignment is achieved through the delay lock mechanism. Figure 8 illustrates the alignment process. At the beginning, all the hybrid delay lines are set to their minimum delay values and disabled (by setting $e_i$'s to zero). Then, the symbol generators are aligned one by one. For the generator under alignment

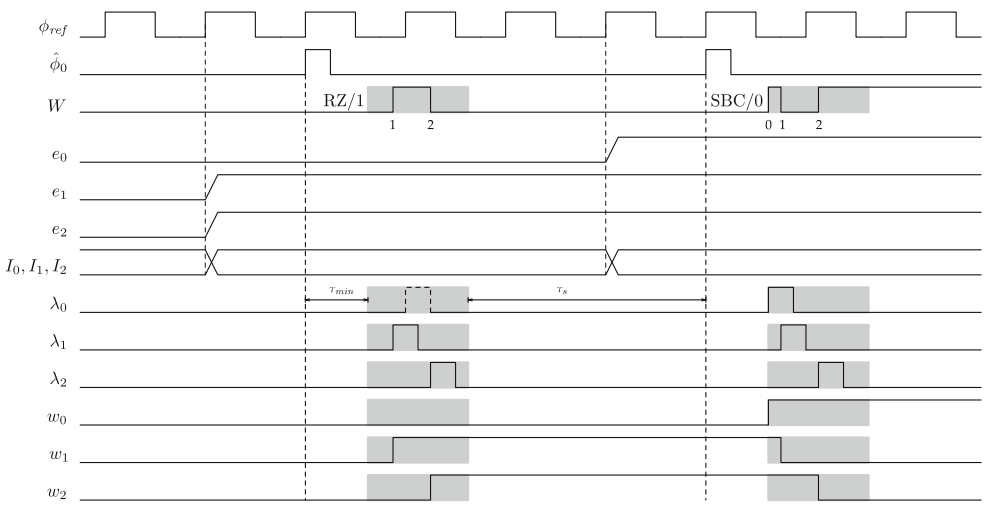



**Fig. 7** The symbol generator timing diagram

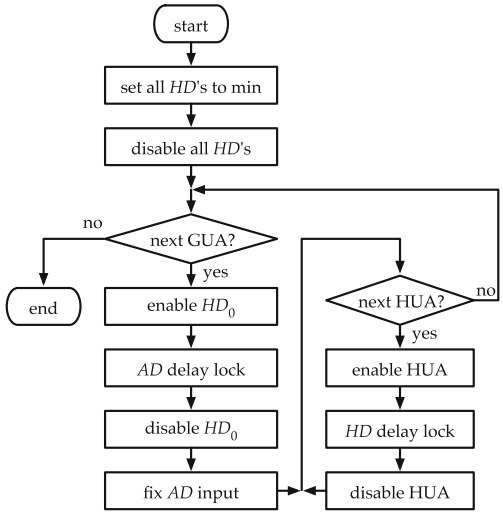



**Fig. 8** The symbol generator alignment process

(GUA), $HD_0$ is enabled; this establishes the loop and the delay lock process starts. During the delay lock process, only the alignment delay line *AD* can be adjusted, i.e., "*AD* delay lock." Once delay lock is achieved, $HD_0$ is disabled and the alignment delay line is fixed at its current input hereafter.

Note that further *fine alignment* is needed because (1) the alignment delay line is a coarse delay line, and (2) the alignment is only valid for $HD_0$. During fine alignment, the hybrid delay lines are aligned one at a time. Each time, only the hybrid delay line under alignment (HUA) is enabled and adjusted to achieve delay lock, i.e., "*HD* delay lock." Upon delay lock, the HUA input value is recorded; this value will be the "time 0" for the symbols it generates.

### *3.3.2 Delay Line Characterization*

The delay line characterization process measures the delay values that each hybrid delay line produces. As shown in Fig. 6, this is achieved by configuring the hybrid delay line under characterization into an oscillator. (All but the delay line under characterization is disabled.) Then, the frequency counter (inside the calibration unit) counts the number of cycles within a specified duration to calculate the oscillation period. Due to the unknown delays of other elements in the oscillation loop, the calibration process can only measure the delay differences, not the true delay values. However, this is not a problem because we only need to know the delay relative to the "time 0" recorded during the fine alignment process.

### *3.3.3 Calibration Parameter Derivation*

As mentioned earlier, calibration is mandatory to ensure the hybrid delay line linearity. The alignment delay lines are not calibrated as they are for coarse alignment only and their values are fixed.

The proposed hybrid delay line consists of a coarse delay line and a fine delay line. The delay step of the coarse delay line is much larger than that of the fine delay line. As a result, the coarse delay line becomes the linearity limiter.

In the proposed calibration approach, the fine delay line is utilized to compensate for the coarse delay line nonlinearity. For each coarse delay line input code $i$, the amount of fine delay line input adjustment $\delta(i)$ is

$$\delta(i) = \frac{\tau_{\mathrm{LSB}}^{coarse}}{\tau_{\mathrm{LSB}}^{fine}} \cdot \mathrm{INL}(i). \tag{8}$$

A lookup table is constructed for each hybrid delay line to store $\delta(i)$'s. During functional operation, according to the coarse delay line input $i$, $\delta(i)$ is added to the fine delay line's input.

## 3.4 FPGA Delay Lines

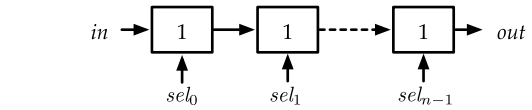
Three types of FPGA delay lines are utilized in the proposed formatter. They play different roles according to the following factors: monotonicity, linearity, resolution, minimum delay, and overhead ratio.

### *3.4.1 Alignment Delay Line*

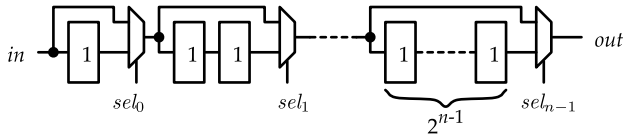
The alignment delay lines facilitate symbol generator alignment. For delay lock to succeed, the alignment delay line must be monotonic. Minimum delay is not a concern for the alignment delay line because its input is fixed (after alignment) and thus does not contribute to $\tau_{min}$. The linearity and resolution requirements are moderate as the only goal is coarse alignment.

Figure 9a depicts the alignment delay line structure. To ensure monotonicity, it is constructed with unit weight buffers; the cost is more input signals and more complicated control. Each unit weight buffer of the alignment delay line occupies one LAB. As shown in Fig. 9c, depending on the control signal $sel_i$, path A or path B (as indicated by the red lines) is selected. Path A incurs larger delay than path B as it passes through more carry chain stages; the delay difference defines the resolution. Note that the both paths are LAB internal; this reduces routing uncertainty.

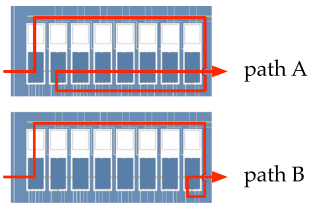
A 64-stage alignment delay line is implemented and characterized; the results are shown in the second row of Table 1. The resolution is 564.7 ps and the maximum INL and DNL are 0.25 and 0.20 LSB, respectively. The overhead ratio $\Omega$ of the alignment delay line is 1.56.



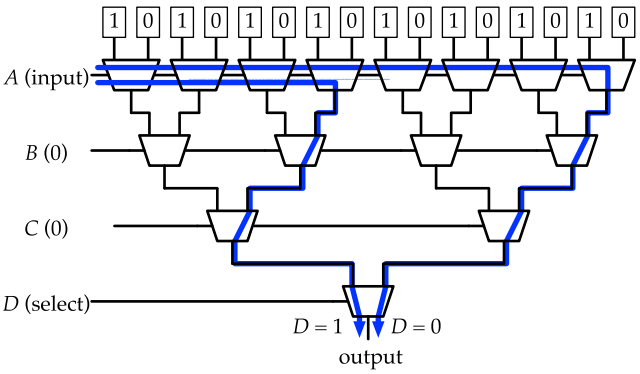

(a) A delay line with unit-weight delay elements.




(b) A delay line with binary-weight delay elements.




(c) Two LAB-internal carry-chain paths.




(d) LUT-based delay element.

**Fig. 9** The delay line schematics and delay elements

### *3.4.2 Coarse Delay Line*

The hybrid delay line consists of a coarse and a fine delay line. Figure 9b shows the coarse delay line schematic; it consists of binary weighted delay taps in which tap $i$ contributes zero or $2^i$ unit delays (path A of Fig. 9c) depending on $sel_i$. Compared to the alignment delay line, the coarse delay line will have a lower overhead ratio because of the zero delay path; however, the binary weighted architecture is more prone to non-monotonicity.

**Table 1** Measurement results of 64-stage FPGA delay lines

| type | LSB | max INL | max DNL | Ω |
|---|---|---|---|---|
| alignment | 564.7 ps | 0.25 LSB | 0.20 LSB | 1.56 |
| coarse | 866.1 ps | 0.39 LSB | 0.47 LSB | 0.19 |
| fine | 12.8 ps | 4.7 LSB | 1.5 LSB | 51.22 |

The third row of Table 1 show the measurement results of a 8-tap (64-stage) coarse delay line. It has a low overhead ratio of $\Omega = 0.19$ but worse linearity (maximum INL/DNL = 0.39/0.47 LSB) due to LAB-external routing.

### *3.4.3 Fine Delay Line*

The fine delay line schematic is the same as the alignment delay line (Fig. 9a); however, it is constructed with the LUT-based delay elements (Fig. 9d [7]) to provide the required fine resolution. As shown in Fig. 9d, by filling the LUT's table values with alternating 0's and 1's, the LUT becomes a buffer with terminal A being its input. Although the $B$, $C$, and $D$ inputs have no impact on the output logic value, they can affect the input to output delay. In the LUT-based delay element, $B$ and $C$ are fixed at 0 and $D$ is used as the *select* signal to determine the delay value.

The measurement results of a 64-stage fine delay line are listed in the bottom row of Table 1. The fine delay line has a resolution of 12.8 ps, which is needed to achieve high edge placement resolution. The maximum INL and DNL are 4.7 and 1.5 LSB, respectively. Note that the fine delay line has a high overhead ratio of $\Omega = 51.22$; as a result, the use of the fine delay line should be limited.

## 4 Experimental Results

The proposed formatter is implemented on an Altera DE II development board. Measurement results are shown in the following sub-sections to validate its feasibility.

Based on the previous measurement results of 64-stage delay lines, a rough estimate of the hybrid delay line's

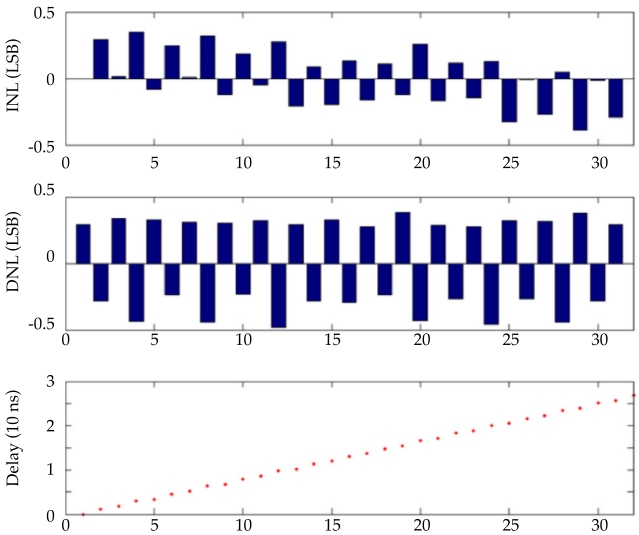



**Fig. 10** The coarse delay line measurement results

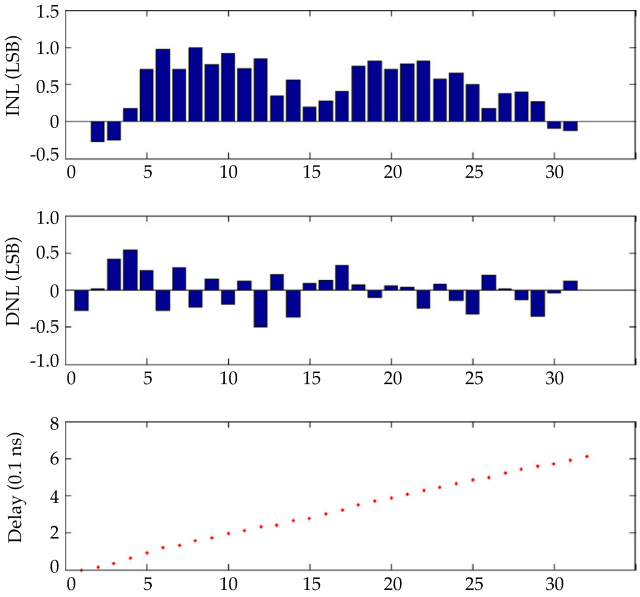



**Fig. 11** The fine delay line measurement results

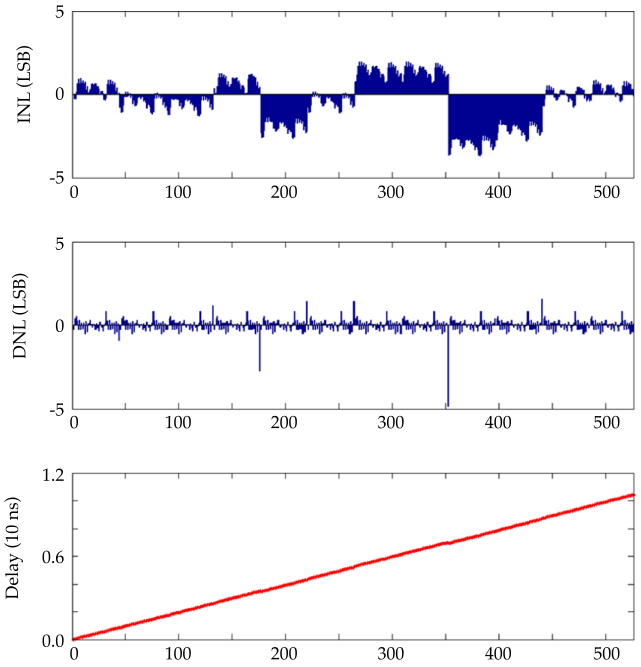


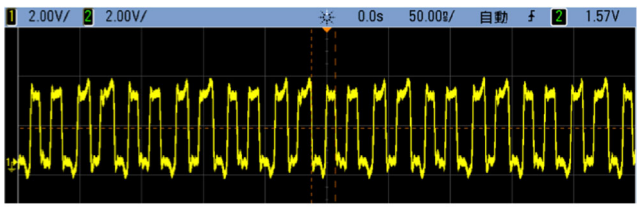

**Fig. 13** The hybrid delay line calibration results

minimum delay $\tau_{min}$ is 39 ns. To tolerate the physical design uncertainty and the settling time, the formatter utilizes eight symbol generators.

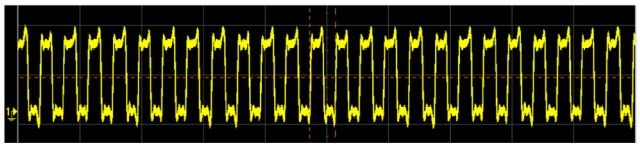
### 4.1 Delay Line Measurement Results

In the formatter, both the coarse and fine delay lines have 32 stages. Figure 10 shows the coarse delay line measurement results. The resolution is 865.3 ps and the maximum INL and DNL are 0.39 and 0.47 LSB, respectively, which are close to those in Table 1. Note that a maximum INL of 0.39 LSB corresponds to a maximum deviation from the ideal delay value by 337 ps; this will be calibrated by the fine delay line.

Figure 11 shows the 32-stage fine delay line measurement results. It has a resolution of 19.72 ps and the maximum INL and DNL are 1.00 and 0.53 LSB, respectively. Note that these results are significantly different from those in Table 1. With its high resolution, the fine delay line can be used to calibrate the coarse delay line. Note that the dynamic range of the fine delay line is about 70 % of the coarse delay line step size. A fixed delay line of about 500 ps is added to bridge the gap; this also helps reduce $\tau_{min}$.




(a) Before symbol generator alignment.



(b) After symbol generator alignment.

**Fig. 12** The pre and post-alignment output waveforms

### 4.2 Calibration Results

The formatter calibration includes symbol generator alignment and the hybrid delay line calibration.

To demonstrate the alignment effectiveness, the formatter is set up to generate alternating zero's and one's. Figure 12a and b show the output waveforms before and after alignment; the generator-to-generator skews range from -2 ns to 1.89 ns. Before alignment, the waveform shows non-uniform edge spacing; the post-alignment waveform exhibits significant improvement.

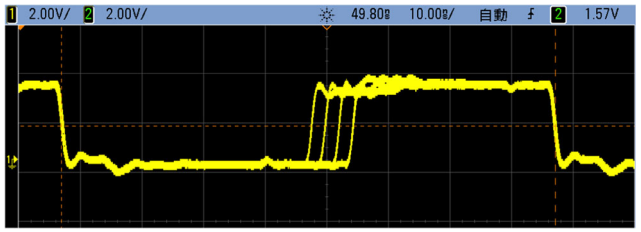



**Fig. 14** The edge placement demonstration

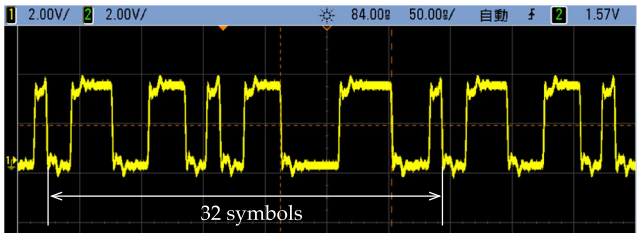



**Fig. 15** The arbitrary waveform generation demonstration

The hybrid delay line calibration results are shown in Fig. 13 for output values up to 10 ns, which is the target symbol length. After calibration, the hybrid delay line has a resolution of 19.87 ps, and the maximum INL and DNL are 3.68 LSB and 4.84 LSB, respectively.

### 4.3 Demonstration

Figure 14 demonstrates the formatter's edge placement capability. In this demonstration, the formatter generates four signal waveforms; they differ in the locations of the first rising edge with a 2.3-ns step size. Figure 14 is obtained by overlaying the four waveforms. In Fig. 15, an arbitrary waveform generation example is demonstrated. The generated waveform consists of 32 symbols.

Finally, using the Altera Quartus Power Analyzer, the estimated power consumption is 410 mW.

## 5 Conclusion

We have presented an FPGA-based 100-Msps formatter that achieves 20-ps edge placement resolution. To overcome the FPGA-imposed limitations, the formatter is capable of aligning the symbol generators and compensating delay line non-linearities. Furthermore, several FPGA-based delay lines are developed to enhance the overall resource usage efficiency. In the future, we will investigate to further improve the formatter linearity and area efficiency.

## References

1. Altera Corporation (2008) Cyclone II Device Handbook, Volume 1
2. Analog Devices Inc (2007) Quad Pin Timing Formatter ADATE207
3. Bayer E, Traxler M (2010) A high-resolution (< 10 ps RMS) 32-channel time-to-digital converter (TDC) implemented in a field programmable gate array (FPGA). In: Conference Record of IEEE-NPSS Real Time Conference, pp 1–5
4. Bayer E, Zipf P, Traxler M (2011) A multichannel high-resolution (< 5 ps rms between two channels) time-to-digital converter (TDC) implemented in a field programmable gate array (FPGA). In: Conference Record of IEEE Nuclear Science Symposium and Medical Imaging Conference, pp 876–879
5. Carlo SD, Prinetto P, Scionti A, Figueras J, Manich S, Rodriguez-Montanes R (2009) A low-cost FPGA-based test and diagnosis architecture for SRAMs. In: Proceedings of International Conference on Advances in System Testing and Validation Lifecycle, pp 141–146
6. Chen P, Chen PY, Lai JS, Chen YJ (2010) FPGA vernier digital-to-time converter with 1.58 ps resolution and 59.3 minutes operation range. IEEE Transactions on Circuits and Systems I 57(6):1134–1142
7. Chen YY, Huang JL, Kuo T (2013) Implementation of programmable delay lines on off-the-shelf FPGAs. In: Proceedings of AUTOTESTCON, pp 1–4
8. Junnarkarm SS, Connor PO, Fontaine R (2008) Conference Record of IEEE Nuclear Science Symposium, pp 3434–3439
9. Li J, Zheng Z, Liu M, Wu S (2006) Large dynamic range accurate digitally programmable delay line with 250-ps resolution. In: Proc. 8th International Conference on Signal Processing
10. Lin C, Shao B, Zhang J (2011) A multi-channel digital programmable delay trigger system with high accuracy and wide range. In: Proceedings of International Conference on Electronics, Communications and Control, pp 1835–1838
11. Mostardini L, Bacciarelli L, Fanucci L, Bertini L, Tonarelli M, Giambastiani A, M D Marinis (2007) FPGA-based low-cost system for automatic test on digital circuits. In: Proceedings of International Conference on Electronics, Circuits and Systems, pp 911–914
12. Mostardini L, Bacciarelli L, Fanucci L, Bertini L, Tonarelli M, Marinis MD (2009) FPGA-based low-cost automatic test equipment for digital integrated circuits. In: Proceedings of International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, pp 32–37
13. Park J, Lee I, Park YS, Kim SG, Ryu KH, Jung DH, Jo K, Lee CK, Yoon H, Jung SO, Choi WY, Kang S (2012) Integration of dual channel timing formatter system for high speed memory test equipment. In: Proceedings of International Soc Design Conference, pp 185–187
14. Perry G (2007) The Fundamentals of Digital Semiconductor Testing. New Smyrna Beach, Florida: Soft Test Inc.
15. Qi J, Deng Z, Gong H, Liu Y (2010) A 20ps resolution wave union FPGA tdc with on-chip real time correction. In: Conference Record of IEEE Nuclear Science Symposium, pp 396–399
16. Syed AR (2003) RIC/DICMOS — multi-channel CMOS formatter. In: Proceedings of IEEE International Test Conference, pp 175–184
17. Syed AR (2004) Automatic delay calibration method for multichennel CMOS formatter. In: Proceedings of IEEE International Test Conference, pp 577–586
18. Wu J (2009) An FPGA wave union tdc for time-of-flight applications. In: Conference Record IEEE Nuclear Science Symposium, pp 299–304
19. Wu J (2010) Several Key Issues on Implementation Delay Line Based TDCs using FPGAs. IEEE Transactions on Nuclear Science 57(3):1543–1548
20. Wu J, Shi Z. (2008) The 10-ps wave union TDC: Improving FPGA TDC resolution beyond its cell delay. In: Conference Record of IEEE Nuclear Science Symposium, pp 3440–3446

**Yu-Yi Chen** was born in Taichung, Taiwan, in 1985. He received the B.E. degree in electrical engineering from National Taiwan Uni- versity of Science and Technology in 2007, and M.S. degree in electrical en- gineering from National Taiwan University in 2013. Since 2013, he has been an ATE designer with OpenATE, Inc., Taiwan. His main research interest is FPGA-based timing circuit.

**Jiun-Lang Huang** received the B.S. degree in Electrical Engineering from National Taiwan University, Taiwan, in 1992, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of California at Santa Barbara in 1995 and 1999, respectively. From 2000 to 2001, he served as an assistant research engineer in the ECE department, UCSB. In 2001, he joined National Taiwan University and is currently a professor in the Graduate Institute of Electronics Engineering and the Department of Electrical Engineering. His main research interests include design-for-test (DfT) and Built-In Self-Test (BIST) for mixed-signal systems, and VLSI system verification.

**Terry Kuo** received his B.S.E.E. degree from Chung Yuan Christian University, Taiwan. He was with the Test Research, Inc. as the Digital Tester Team leader. He is currently the General Manager of Ope- nATE Inc.

**Xuan-Lun Huang** received his Ph.D. degree in electronics engineering from National Taiwan University, Taiwan, in 2010. In 2009, he joined the Industrial Technology Research Institute (ITRI), Hsinchu, Taiwan, as a research and development engineer with the Information and Communications Research Laboratories. In 2014, he joined Mediatek Inc., Taiwan, where he is currently a system architecture design engineer. His research interests include design and test of mixed-mode SoC for wireless communication and wearable devices.