Dual Fixed-Point: An Efficient Alternative to Floating-Point Computation

Chun Te Ewe, Peter Y.K. Cheung, and George A. Constantinides

Department of Electrical & Electronic Engineering,Imperial College,
Exhibition Road, London SW7 2BT, England.
{ct.ewe, p.cheung, g.constantinides}@imperial.ac.uk

Abstract. This paper presents a new data representation known as *Dual FiXed-point* (DFX), which employs a single bit exponent to select two different fixed-point scalings. DFX provides a compromise between conventional fixed-point and floating-point representations. It has the implementation complexity similar to that of a fixed-point system together with the improved dynamic range offered by a floating-point system. The benefit of using DFX over both fixed-point and floating-point is demonstrated with an IIR filter implementation on a Xilinx Virtex II FPGA.

1 Introduction

Most arithmetic operations implemented on FPGAs use fixed-point arithmetic representations. For applications where a large dynamic range is required, fixed-point representation may result in implementations with very wide bit-width . In contrast, floating-point representation has a much larger dynamic range than fixed-point for a given bit-width, but arithmetic circuits for floating-point numbers are considerably larger and slower than their fixed-point counterparts. In this paper, a new representation known as *Dual FiXed-point* (DFX) is introduced. It combines the simplicity of a fixed-point system with the wider dynamic range offered by a floating point system. Using a single bit exponent which selects two different fixed-point representations, it allows dynamic scaling of signals throughout the system.

The original contributions of this paper are: 1) to propose the new *Dual FiXed-point* (DFX) system; 2) to present the design of basic arithmetic operators using DFX; 3) to demonstrate the use of DFX through the implementation of an IIR filter on a FPGA; 4) to compare DFX with conventional fixed-point and floating-point implementations in terms of area, accuracy and speed.

The paper is organised as follows. Section 2 compares fixed-point and floating-point number systems. The new DFX number system is described in Section 3. Section 4 shows the design of the basic arithmetic functions using DFX and compares their size and speed to those using fixed-point and floating-point. The implementation and performance of an IIR filter in all three number formats are given in Section 5. Section 6 concludes the paper and suggest future work.

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 200–208, 2004.

2 Floating-Point Versus Fixed-Point

In fixed-point arithmetic, all numbers are represented by integers, fractions or a combination of both. This is done by partitioning a binary word of n digits into two sets: q digits in the integral part and p digits in the fractional part, satisfying $p + q = n$. In two's complement fixed-point notation, the value of an n-tuple with radix point between q most significant digits and p least significant digits is

$$X = -x_{q-1} \cdot 2^{q-1} + \sum_{i=-p}^{q-2} x_i 2^i \tag{1}$$

The position of the radix point determines the range of the fixed-point number. Throughout this paper the word-length and the radix point of a fixed-point number is denoted as n_p.

Floating-point representation allows designers to attain a large dynamic range without having to scale the signals. Generally, a floating-point number F is represented by the pair (M,E) having the value

$$F = M \cdot \beta^E \tag{2}$$

where M is the significand (or mantissa), E is the exponent and β is the base of the exponent. Typically for digital systems $\beta = 2$.

For all practical systems it is possible to choose a word-length long enough to reduce the finite precision effects to a negligible level, it is often desirable to use as few bits as possible while achieving user-defined output error conditions in order to optimize area, power or speed. Recent work in mulitple word-length optimisation for fixed-point and floating-point systems can be found in Constantinides [1] and Gaffar [2] respectively. Often, fixed-point designs out-perform floating-point designs in overall system-wide cost including area, power and speed [3] as long as its inputs are properly scaled with the appropriate bit-width [4]. However when signals have a large dynamic range, floating-point designs prevail due to its large dynamic range as compared to fixed-point.

Some work has been done attempting to combine the best of the two number formats. Horrocks and Bull [5] used a pseudo floating-point structure for FIR filters while [6] uses a floating-point representation for design parameters. Both methods show good output performance with low complexity but since they inherently run on fixed-point, they do not possess the large dynamic range capability of floating-point. Block floating-point approach [7], commonly used in FFT analysis, provides most of the advantages associated with floating-point realizations with an implementation complexity approaching that of fixed-point. However, block floating-point only scales a block of data; it lacks the dynamic scaling property offered by the proposal in this paper.

3 Dual FiXed-Point

The proposed n-bit Dual FiXed-point (DFX) format consists of an exponent bit E, and $n-1$ bits of a signed significand X, as shown in Figure 1. The exponent selects between two scalings for the significand X, giving two possible ranges for the number. The lower

Fig. 1. DFX Format

number range is referred to as $Num0$ while the higher number range is referred to as $Num1$.

In order to achieve two different scalings, we define two radix points p_0 and p_1 such that the radix point of $Num0$ and $Num1$ are p_0 and p_1 bits from the least significant bit respectively, and $p_0 > p_1$.

The value of this DFX number is given by

$$D = \begin{cases} X \cdot 2^{-p_0} & \text{if } E = 0 \\ X \cdot 2^{-p_1} & \text{if } E = 1 \end{cases} \tag{3}$$

A *boundary value*, B, is needed to decide the best scaling to use and hence the value of E. E is determined as follows,

$$E = \begin{cases} 0 & \text{if } -B \leq D < B \\ 1 & \text{if } D < -B \text{ or } D \geq B \end{cases} \tag{4}$$

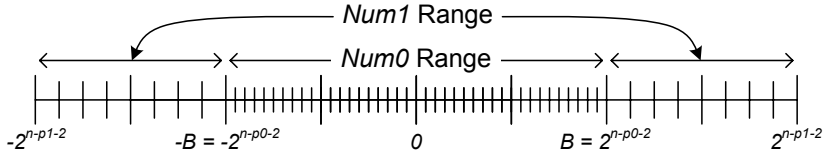

Fig. 2. $Num0$ and $Num1$ range in a DFX Number

In order to simplify the design of the arithmetic units, the boundary value is defined as the next incremental value after the maximum positive number of $Num0$, i.e. $B = 2^{n-p_0-2}$ (-2 because of the exponent and sign bits). The range and precision of Num0 and Num1 are illustrated in Figure 2. To completely define a DFX number, we need n, the size of the DFX number, p_0 and p_1, the radix points. The notation used in this paper is DFX $n\_p_0\_p_1$.

Dynamic range is defined by the ratio between the largest and the smallest absolute number in the data format. The smallest absolute value of a DFX number is 2^{-p0} while the largest absolute value is 2^{n-p_1-2}, hence the dynamic range of a DFP number is given as

$$\text{Dynamic range} = 20 \log_{10}(2^{n+p_0-p_1-2}) \text{ dB} \tag{5}$$

Having two possible scaling for a number gives DFX better range capability than fixed-point as shown in Table 1.

Table 1. Dynamic Range comparisons

Number System	*Dual FiXed-point*	*Dual FiXed-point*	*Fixed Point*	*Floating Point*
Format	32_30_0	32_16_4	32-bit	32-bit IEEE
Dynamic Ranges	$2^{60} \approx$ 361dB	$2^{46} \approx$ 276dB	$2^{31} \approx$ 187dB	$2^{254} \approx$ 1529dB

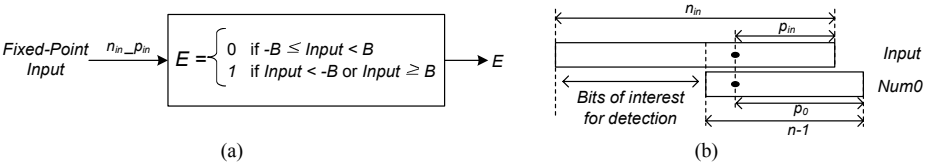

Fig. 3. (a)DFX Range Detector Module and (b) Input Bits the range detector is interested in

4 Dual FiXed-Point Circuits

Arithmetic modules in DFX have been designed in VHDL and mapped onto a Xilinx Virtex II (XC2V80-6fg256) in order to evaluate their area and speed.

4.1 DFX Range Detector

The function of the DFX Range Detector, shown in Figure 3(a), is to generate the exponent bit, E, which selects the range used in the DFX number. The input to this module is a fixed-point number with the format $n_{in}\_p_{in}$ (n_{in} being the input word-length and p_{in} being the position of its radix point). The boundary chosen allows this operation to be simplified down to a logic operation given by (6). If the input is in the $Num0$ range, all the bits above the boundary will be 0's (when it is a positive input) or 1's (when it is a negative input) since the input is a two's complement number. The bits of interest for detection are shown in Figure 3(b).

$$E = \overline{d_{n_{in}-1} \cdot d_{n_{in}-2} \cdot \ldots \cdot d_{n_{in}-(n-p_0-2)-p_{in}}} + \overline{\overline{d_{n_{in}-1}} \cdot \overline{d_{n_{in}-2}} \cdot \ldots \cdot \overline{d_{n_{in}-(n-p_0-2)-p_{in}}}} \tag{6}$$

4.2 DFX Adder

The DFX Adder module adds together two DFX numbers (see Figure 4(a)). Similar to a floating-point adder, DFX inputs may have to be scaled in order to align the radix points before adding. But unlike floating-point, the number of bits to shift is known *a priori*. Therefore only multiplexers instead of barrel shifters are necessary to perform the necessary scaling. As a result the DFX Adder is expected to be both smaller and faster than the equivalent floating-point adder. Note that ">>" and "<<" are the shift operators which requires only wire routing and mod 2^{n-1} simply extracts the least significant (n-1) bits.

The Adder Control Block determines the shifting of the inputs via the A_sel and B_sel signals. If the input exponents are different, i.e. one input is $Num0$ and the other

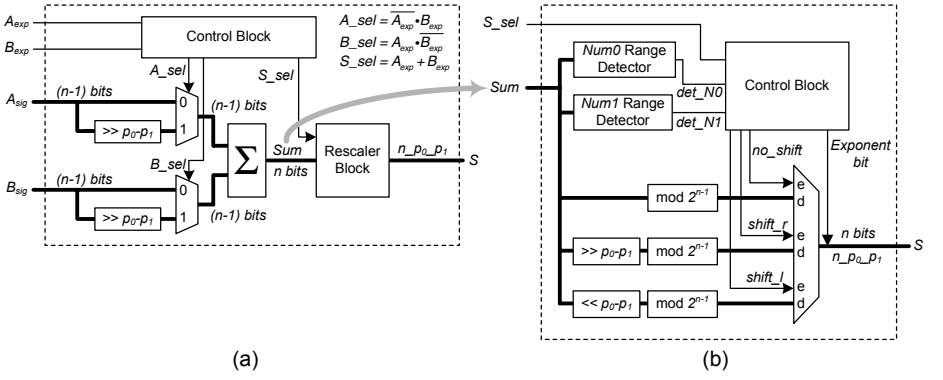

Fig. 4. (a) DFX Adder Module and (b) Rescaler Block

is $Num1$, the $Num0$ number will be shifted up to the $Num1$ range. If both exponents are the same, there will be no shifting.

The adder is a full precision adder and its result sum is fed into the Rescaler Block whose scaling is provided by the signal S_sel. The adder's resulting scale is always $Num1$ unless both the inputs are $Num0$. The output signals of the Adder Control Block are given below in Figure 4.

The Rescaler Block (Figure 4(b) first detects the range of sum with two range detectors that are aligned to the two possible scales. The $Num0$ Range Detector assumes the its input is a $Num0$ number producing the signal det_N0 while the other assumes a $Num1$ input producing the signal det_N1.

No shifting is needed if the adder's result remains in the same range. If the result changes from a $Num0$ to a $Num1$, Sum has to be shifted $p_0 - p_1$ bits to the right and sign extended. The result will however be shifted $p_0 - p_1$ bits to the left and zero padded if the result changes from a $Num1$ to a $Num0$ number. The combinational logic of the internal signals and the exponent bit are given below. Finally, the multiplexer truncates the output to give $(n-1)$ bits for the significand.

$$\begin{aligned} \textit{no\_change} &= (\overline{S\_sel} \cdot \overline{det\_N0}) + (S\_sel \cdot det\_N1) \\ \textit{shift\_r} &= \overline{S\_sel} \cdot det\_N0 \\ \textit{shift\_l} &= S\_sel \cdot \overline{det\_N1} \\ \textit{Exponent bit} &= (\overline{S\_sel} \cdot det\_N0) + (S\_sel \cdot det\_N1) \end{aligned} \tag{7}$$

Table 2 shows the size and speed comparison of a 32-bit adder implemented in all three number formats. It can be seen that while DFX is about 4 times larger and slower than an equivalent fixed-point adder, it is also almost 4 times smaller and faster than the floating-point circuit.

4.3 DFX Multipliers

Two versions of the multiplier have been designed. The DFX-H Multiplier takes one DFX input and one fixed-point input, while the DFX-F Multiplier performs a full multi-

Table 2. Size and latency delay comparison table of 32-bit adders

Adder Type	Size (Slices)	Latency (ns)
Fixed-Point	17	2.5
DFX	64	10.28
Floating-Point (IEEE)	255	34.48

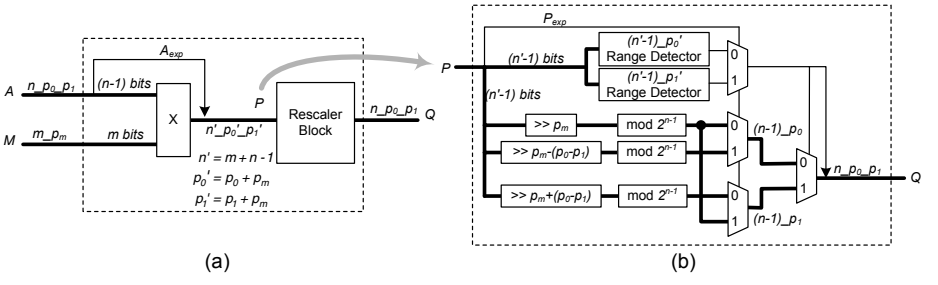

Fig. 5. (a) DFX-H Multiplier module and (b) Rescaler Block

plication between two DFX inputs. Due to space constraints, only the DFX-H multiplier is described here.

Figure 5(a) shows the DFX-H Multiplier forming the product between a DFX input A and a fixed-point input M. This is particularly useful in applications such as filtering where one of the operands is a constant. Unlike the DFX Adder, a DFX-H Multiplier does not require aligning the radix points at the inputs to the binary multiplier. However, the product P needs to be properly scaled and converted into DFX format.

Consider the multiplication of a DFX $n\_p_0\_p_1$ number with a FX m_p_m number, as shown in Figure 5(a), giving a product P which is in DFX $n'\_p'_0\_p'_1$ format, where $n' = m+n-1$, $p'_0 = p_0 + p_m$ and $p'_1 = p_1 + p_m$. The product P needs to be converted back to a DFX $n\_p_0\_p_1$ formatted number.

Figure 5(b) show the circuit for the DFX-F Rescaler Block. The range detectors are aligned to the radix points of p'_0 and p'_1 respectively. Further optimization could be done assuming the multiplier M is a constant value.

Table 4 shows the size and speed comparison of 32-bit multipliers implemented in all three number formats. The optimized DFX-H Multiplier is about 1.2 times larger and slower than an equivalent fixed-point multiplier. However it is also about 1.2 times smaller and faster than a floating-point multiplier. The DFX-F Multiplier is comparable with its floating-point counterpart and it is about 1.5 times larger and slower than fixed-point.

4.4 DFX Encoder and Decoder

In order to utilize this number system, a method is needed to convert a number from a known type to DFX. Currently modules exists to encode and decode to and from two's complement fixed-point. The size and latency of the 32-bit DFX modules are given

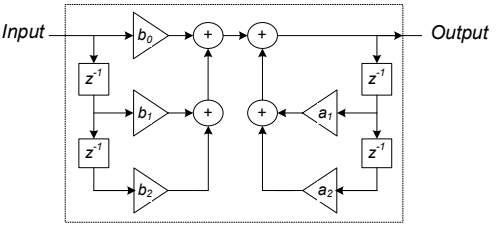

Fig. 6. IIR Filter signal flow diagram

Table 3. DFX Encoder and Decoder size and latency delay table

Module	Size (Slices)	Latency (ns)
Encoder	17.5	7.8
Decoder	10	5.8

Table 4. Multiplier size and latency delay comparison table

Multiplier Type	Size (Slices)	Latency (ns)
Fixed-point	43	13.946
DFX-H	58	17.308
DFX-F	76	19.149
Floating-point	73	20.683

in Table 3. The values for the decoder are approximate because the decoder is usually absorbed into adjacent blocks by logic optimization.

5 Example and Results

The effectiveness of using DFX as an alternative computation method to floating-point is demonstrated by using a Direct Form I implementation of a 2nd order notch IIR filter with the notch at 0.15 of the Nyquist frequency as shown in Figure 6. The filter has five coefficients, three of it in the forward path and two in the feedback path. 32-bit versions of the filter were implemented with DFX (designs D1 and D2), fixed-point (X1, X2 and X3) and floating-point (P1 and P2) formats for comparison and the result is given in Table 7. The DFX Multiplier FX is used in the design since the coefficients are constants.

The target chip for the IIR design is a Xilinx Virtex II XC2V500-6fg456. Comparing the formats with the same bit-width, i.e. 32-bit, DFX designs fall between fixed-point design X1, the smallest and fastest, and floating-point design P1, the largest and slowest. Designs X3 and P2 are about the same size with DFX designs with design X3 being the fastest of the four.

In order to exercise the dynamic range capability of DFX, a set of input data with the frequency distribution as shown in Figure 8(a) and an appropriate spectrum was created. The output SNR, average relative error and maximum relative error of different filter types are shown in Table 5. The error is with reference to double precision floating-point

Filter Type	Design	Format	Size (Slices)	Latency (ns)
DFX	D1	32_18_6	584	52.29
	D2	32_9_6	580	51.28
Fixed Point	X1	32_7	255	24.26
	X2	33_8	272	24.18
	X3	43_18	572	31.51
Floating Point	P1	32bit M23 E8	1459	127.39
	P2	17bit M10 E6	586	88.183

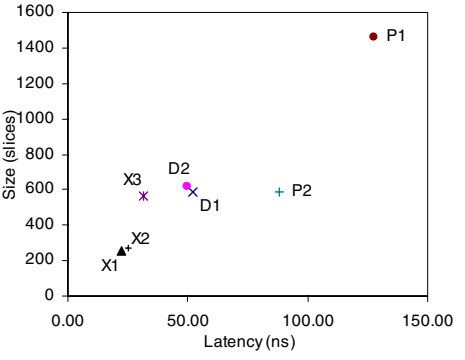

Fig. 7. IIR filters size and latency comparison

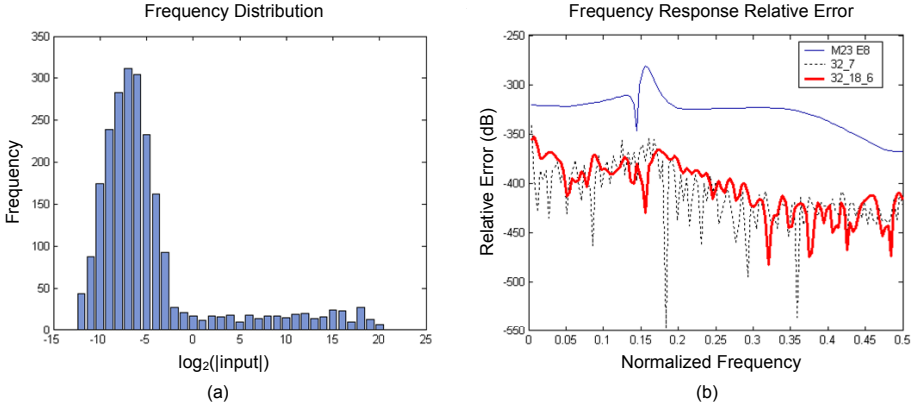

Fig. 8. (a) The frequency distribution of the input and (b) the frequency response relative error for the filter in Fig. 6

Table 5. The error results for the IIR Filter in Fig. 6

Filter Type	Design	Format	SNR (dB)	Av Relative Error (dB)	Max Relative Error (dB)
DFX	D1	32_18_6	333.88	-82.89	-41.02
	D2	32_9_6	347.25	-29.63	13.07
Fixed Point	X1	32_7	330.53	-18.20	24.14
	X2	33_8	344.09	-23.95	17.77
	X3	43_18	482.21	-84.75	-44.21
Floating Point	P1	32bit M23 E8	299.13	-85.90	-29.88
	P2	17bit M10 E6	115.42	-7.24	48.25

results taken to be the expected results. Relative error is calculated as a ratio of the difference error over the reference result.

According to Table 5 floating-point design P1 performs pretty well in terms of relative error (the lower the value the better) but poorly in terms of output SNR. DFX designs out-

performs floating-point designs because the DFX design a more precise number number format (i.e. the signifand of design D1 has 31-bits, compared to 24-bits in design P1).

Fixed-point designs show improvement in terms of output SNR and relative error as its word-length increases. In terms of output SNR, DFX design D2 may beat fixed-point design X1, but by increasing the bit-width by one, design X2 is able to out perform design D2. However, design D1 shows good average relative error performance which can only be match by designs X3 and P1. Being a floating-point design, design P1 is notably larger than D1 and fixed-point design X3 is similar in size to D1.

Figure 8(b) shows the relative error of the frequency response measured against the maximum output range of designs with similar word-length. It shows that the floating-point performance is poor overall especially at the notch frequency. The DFX implementation performs similarly to fixed-point but, notably, DFX performs better than fixed-point at the notch frequency.

6 Conclusion and Future Work

This paper demonstrates that by only providing two possible scalings, as in DFX, reduces the design complexity to give smaller and faster designs as compared to floating-point. By choosing the right scaling, DFX can have similar performance to fixed-point while capable of handling a wider dynamic range.

Future work will include the exploration of multiple word-length designs using DFX and the optimization of DFX design for area, accuracy and speed.

References

1. Constantinides, G.A., Cheung, P.Y.K., Luk, W.: Wordlength optimization for linear digital signal processing. IEEE Transactions on CAD of Integrated Circuits and Systems **22** (2003) 1432–1442
2. Gaffar, A.A., Luk, W., Cheung, P.Y.K., Shirazi, N.: Customising floating-point designs. In: IEEE Symposium on Field-Programmable Custom Computing Machines. (2002)
3. Inacio, C., Ombres, D.: The DSP decision: fixed point or floating? IEEE Spectrum **33** (1996) 72–74
4. Oppenheim, A.V., Weistein, C.J.: Effects of finite register length in digital filtering and the fast fourier transform. Proceedings of the IEEE **60** (1972) 957–976
5. Horrocks, D.H., Bull, D.R.: A floating-point FIR filter with reduced exponent dynamic range. In: IEEE International Symposium on Circuits and Systems. (1992)
6. Wust, H., Kasper, K., H.Reininger: Hybrid number representation for the FPGA-realization of a versatile neuro-processor. In: Euromicro Conference. (1998)
7. Oppenheim, A.V.: Realisation of digital filters using block-floating-point arithmetic. IEEE Transaction on Audio and Electroacoustics **18** (1970) 130–136