Square root calculations in FPGA

Introduction. The square root function $\sqrt{x}$ is important elementary function in the scientific calculations, digital signal and image processing [1]. The artificial neural nets need this function as well [2]. At present, the field programmable gate arrays (FPGAs) are expanded for solving the problems, where the function $\sqrt{x}$ calculations are of demand. There are different IP cores of the function $\sqrt{x}$, which are proposed by the FPGA manufacturers and third-party companies [3]. But these IP cores were designed decades ago and they usually don’t take into account the features of the new FPGA generations. Therefore, they need improvements. In the presentation, an improved algorithm of the function $\sqrt{x}$ is proposed, which is suitable for the FPGA implementation.

CORDIC-type algorithms. The CORDIC-type algorithm of the elementary function calculation derives a single exact digit of the result in each computation step. The well-known CORDIC algorithm of the function $\sqrt{x}$ is proposed, which is suitable for the FPGA implementation.

The algorithm analysis shows that when $i$ reaches the limit $n/2$, then the most $i-1$ significant bits of $x_i$ become equal to a one by any $x_0$, and $i$ most significant bits of $y_i$ are exact digits of the result. Therefore, the rest of resulting bits can be calculated after analysis and computation the difference $1-x_i$.

Consider $\varepsilon_1 = 1-x_n/2$, and $\sqrt{x} = x + y_n/2$. Then, to get the exact result, the correction $\varepsilon_x$ is derived from the value $\varepsilon_1$, and it is added to the approximated result. Due to (1),

$$
\varepsilon_1 = 1 - x \prod_{i=1}^{n/2} (1 + a_i 2^{-i})^2; \varepsilon_x = \sqrt{x} - x \prod_{i=1}^{n/2} (1 + a_i 2^{-i}).
$$

Then

$$
z = \sqrt{x} \prod_{i=1}^{n/2} (1 + a_i 2^{-i}); \varepsilon_1 = 1 - z^2 = (1 + z)(1 - z); \varepsilon_x = \sqrt{x}(1 - z).
$$
Consider $z \approx 1$, then $\varepsilon_1 \approx 2(1-z)\varepsilon_x \approx \sqrt{\varepsilon_1}/2 = y_{n/2}/2(1-x_{n/2})/2$. The result is $y_n = y_{n/2} + y_{n/2}(1-x_{n/2})/2$. So, in order to obtain a refined result, the correction is added to the approximate result $y_{n/2}$. To do this, a subtraction and a multiplication should be taken. Moreover, due to the fact that $\varepsilon_1$ and $\varepsilon_x$ have the zeroed most significant bits, then the multiplication is performed at half bit width. The resulting algorithm looks like the following:

$$
\begin{align*}
    x[0] &= x; y[0] = x; \\
    &\text{for} (i = 0; i < n/2; i++) \{ \\
    &\quad q = x[i] + 2^{(-i+1)}x[i] + 2^{(-2i)}x[i]; \\
    &\quad \text{if} \ (q < 1.0) \ {\{x[i+1] = q; y[i+1] = y[i] + 2^{(-i)}y[i]; \}} \\
    &\quad \text{else} \ {\{x[i+1] = x[i]; y[i+1] = y[i];\}} \\
    &\}\end{align*}
$$

$$
y = y[n/2] + y[n/2]*(1.0 - x[n/2])/2;
$$

In modern FPGA the two and three input adders have the same hardware volume and speed. So, the modified algorithm provides the speed-up approximately in four times comparing to the initial algorithm.

**Experimental results.** The derived algorithm was described by VHDL as the IP core. It was compiled for Xilinx Spartan-6 FPGA for various input and output data bit widths. Fig. 1 and Fig.2 show the relation of the hardware costs in the number of LUTs, and the maximum clock frequency on the bit width for the combinatorial and pipelined networks of this IP core, respectively. It should be noted that the modules with a bit width up to 32 additionally have a single multiplication block DSP48, and the rest of them have four such blocks.

For comparison, Fig. 1, 2 show the characteristics of the IP cores offered by Xilinx Inc. In general, the proposed IP core loses to the branded one. But its advantages are that it is free and it can be configured for arbitrary input and output bit width, and for any FPGA type. In addition, the proposed IP core has a lower latent delay.

For example, for 24 bits, its latent delay is only 15 cycles versus 24 cycles of the competitor. This means that when implementing the floating-point calculations, the proposed module provides greater performance.

**Conclusion.** The modified CORDIC-like algorithm for deriving the square root function is proposed. The algorithm is distinguished by the minimized number of steps, which is proportional to the given data and result bit width. The algorithm is described in VHDL and is intended for the FPGA implementation. It is the most effective during its implementation in the floating point square root module.