Hardware Accelerators for Financial Applications in HDL and High Level Synthesis

Ioannis Stamoulias
School of Electrical & Computer Engineering
National Technical University of Athens (NTUA)
Athens, Greece
jstamoulias@gmail.com

Christoforos Kachris, Dimitrios Soudris
Institute of Communication and Computer Systems
National Technical University of Athens (NTUA/ICCS)
Athens, Greece
kachris, dsoudris @microlab.ntua.gr

***Abstract*—Many financial applications, like the one used for risk valuation, need high performance and low latency implementations to sustain the high volume of data that need to be processed. This paper presents a suite of high performance hardware accelerators for financial applications used in risk valuation (Black & Scholes, Black-76 and Binomial). The accelerators are developed in fixed point using HDL (VHDL) and in floating point using HLS languages. High Level Synthesis (HLS) allows fast implementation of hardware accelerators from the original legacy codes. The HLS hardware accelerators have been mapped onto a PCIe FPGA (ADM-KU3) board, through the Xilinx SDAccel framework and a thorough comparison in terms of resources, performance and accuracy has been performed. The performance evaluation shows that HLS can achieve higher accuracy due to the floating point, but requires up to 20% higher number of resources in terms of DSPs while the fixed-point implementations developed in HDL can save significant space in terms of resources but with limited accuracy compared to the software code.**

Keywords—high level synthesis; hardware accelerators; financial applications; reconfigurable computing

I. INTRODUCTION

High Level Synthesis tools allow the development of hardware accelerators in reconfigurable logic by using high level languages such as C, C++ and OpenCL. The main challenging task of the designer is to annotate the original code with specific keywords that will help the tool to map the algorithm to efficient hardware. There are several tools that have been proposed in the literature for high level synthesis [1][2][3], but only few of them provide a complete framework where both the accelerator and the CPU host can be implemented and simulated as a whole system. SDAccel tool [4] is based on Xilinx's HLS framework and provides such a framework for the development of an entire system. It utilizes Vivado HLS tool for the implementation of the hardware accelerator and uses Xilinx's cores for the communication through a PCIe port with the Host system running on a CPU.

In this paper, we develop efficient hardware accelerators for three commonly used algorithms in financial applications. These algorithms are developed both using hardware description languages VHDL and HLS based on C and the performance in terms of hardware resources and accuracy are measured. Overall the main contributions of the paper are the followings:

- Efficient hardware architectures for three algorithms in financial applications for risk valuation using HDL for fixed point implementations and HLS for floating point implementations.
- Performance evaluation of the hardware accelerators in terms of resources (area), performance (clock frequency) and accuracy compared to the software reference code.
- A thorough comparison between HDL and HLS in terms of resources, throughput and accuracy.

II. RELATED WORK

In the literature, there are not many studies that implement the same algorithm with both HLS and HDL. In [5], a comparison is shown between the HLS and the hardware description languages for image processing. In this study, the HLS design was implemented in half of the time, but required 61% more LUTs and did not perform as fast in operational maximum frequency tests. In [6], several financial option price solvers have been implemented in FPGA. That work, proposed a framework for comparing the performance of numerical option pricing methods using FPGAs, considering both speed (time to solution) and accuracy (quality of solution), and examines how the speed-accuracy trade-off curve varies for each method. The accelerators are designed in HDL and the main comparison is on the accuracy of the solvers and not the speedup that they provide. Several works exist that present implementations of Monte Carlo methods for financial services, such as [7], [8] and [9]. In [10], a study is performed on HLS, for financial applications, which shows that the HLS hardware description languages are mature enough to be adopted from the industry. In [11], [12], several architectures are shown for acceleration of financial applications.

In this paper, we present a suite of high performance FPGA accelerators for risk valuation algorithms. The fixed-point implementations are implemented in VHDL utilizing less resources (DSP units) but with lower accuracy. The floating-point implementations are developed using HLS achieving higher accuracy, but also requiring higher number of resources. A thorough performance evaluation is performed in terms of accuracy, throughput and hardware resource between these two implementations.

III. FINANCIAL ACCELERATORS

Automation using Technology in Trading & Exchanges is considered as de-facto principle nowadays. Financial applications require low latency and high throughput to cope with the demands of the market, this is the reason that the financial sector is a prominent user of High Performance Computing facilities. Implementations of hardware financial accelerators on reconfigurable logic are perfect candidates for this sector, because they can achieve those requirements through the use of parallelization techniques. Three commonly used algorithms for financial application, such as Risk Valuation, are Black-Scholes, Black-76 and Binomial algorithms. The Black-Scholes model gives a theoretical estimate of the price of European-style options and can also be used for American-style call options. The Black-76 model is a variant of the Black-Scholes model that supposes the underlying is lognormal but the underlying price is the future prices, not the spot price. The Binomial option pricing model discretize time and price of an underlying asset, and mapping both onto a binary tree, thus can handle American put options that can be exercised at any time. In this paper, we investigate those three algorithms:

- The Black & Scholes method for American call options based on spot prices for stock options.
- The Black-76 method for European options based on future prices for index options.
- The Binomial method, without dividends, for American put options based on spot prices for stock options.

A. Black-Scholes model

The Black-Scholes model [13], for the current prices of a European options on a non-dividend paying stock, can be calculated from equation (1) for the *call* option and from equation (2) for the *put* option. When no early exercise is required, an American call option can use the same equation (1) as the European options.

$$c = S_0N(d_1) - Ke^{-rT}N(d_2) \tag{1}$$

$$p = Ke^{-rT}N(-d_2) - S_0N(-d_1) = c + Ke^{-rT} - S_0 \tag{2}$$

Where $d_1 = \frac{\ln(S_0/K)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$ and

$$d_2 = \frac{\ln(S_0/K)+(r-\sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

In the equations, *N(.)* represents the cumulative distribution function of the standard normal distribution, S_0 represents the Stock price at t=0 (spot price), *K* represents the Strike price, *r* represents the Risk-free interest rate, σ represents the Stock price volatility and *T* represents the Time to maturity.

An algorithmic analysis was conducted considering a pipelined architecture that can receive new input per cycle. More kernels can run in parallel to process blocks of data for increase in the performance of the system. Table 1 presents the required operations for the Black-Scholes method. If both *call* and *put* options are needed some operations can be used for both calculations, with bold are the extra operations required if no call option is going to be implemented. In the following hardware implementations only the *call* option was implemented, that can be used with American-type options.

Table 1 Algorithmic analysis of the Black-Scholes model (no reuse of operations)

	Calc. d_1, d_2	Approx. N(.)	Calc. Call	Calc. Put	Sum
Add/Sub	3	6	1	2/**1**	12
Multiplier	3	9	4	**4**	16
Divider	2	1	0	0	3
Square root	1	0	0	0	1
Exponential	0	1	1	**1**	2
Logarithm (ln)	1	0	0	0	1

B. Black-76 model

The Black-76 model [14] is a variant of the Black-Scholes option pricing model and can be used for pricing options on future contracts, bond options, interest rate caps/floors and swaptions. The call option is calculated using the equation (3) and the put option from the equation (4).

$$c = e^{-rT}(FN(d_1) - XN(d_2)) \tag{3}$$

$$p = e^{-rT}\big(XN(-d_2) - FN(-d_1)\big) = c + e^{-rT}(X - F) \tag{4}$$

Where $d_1 = \frac{\ln(F/X)+(\sigma^2/2)T}{\sigma\sqrt{T}}$ and $d_2 = d_1 - \sigma\sqrt{T}$

In the equations, *F* represents the Future price, *X* represents the Exercise price, σ represents the future price volatility and the rest are the same as those of the Black-Scholes model.

An algorithmic analysis was also conducted considering a pipelined architecture that can receive new input per cycle. More kernels can run in parallel to process blocks of data for increased performance. Table 2 presents the required operations for the Black method. Bold number represent again the extra operations needed if no *call* options are going to be computed. Both *call* and *put* options were required and implemented for the pricing of the index options on future prices.

Table 2 Algorithmic analysis of the Black-76 model (no reuse of operations)

	Calc. d_1, d_2	Approx. N(.)	Calc. Call	Calc. Put	Sum
Add/Sub	2	6	1	2/**1**	11
Multiplier	3	9	4	1/**4**	17
Divider	2	1	0	0	3
Square root	1	0	0	0	1
Exponential	0	1	1	**1**	2
Logarithm (ln)	1	0	0	0	1

C. Binomial model

The Binomial pricing model [15] traces the evolution of the options underlying variables in discrete time by generate a binomial tree for a number of time steps until maturity. Each node of the binomial tree represents a possible price at a given point in time and the final nodes calculates the option values of their preceding nodes. Each node has two possible transitions, upwards (increase value) and downwards (decrease value). The call price at each node is calculated from the equation (5), the put price at each node from the equation (6), the terminal call price from the equation (7) and the terminal put option from the equation (8).

$$C_{i,j} = Max(S_{i,j} - K, e^{-\frac{rt}{n}}(pC_{i,j+1} + (1-p)C_{i+1,j+1})) \tag{5}$$

$$P_{i,j} = Max(K - S_{i,j}, e^{-\frac{rt}{n}}(pP_{i,j+1} + (1-p)P_{i+1,j+1})) \tag{6}$$

$$C_{\boldsymbol{i,n}} = Max(S_{i,n} - K, 0) \tag{7}$$

$$P_{\boldsymbol{i,n}} = Max(K - S_{i,n}, 0) \tag{8}$$

Where $S_{i,j} = S_0u^{j-1}d^{i-1}$, *i, j: 0, ..., n* is the stock price at each node, $u = e^{\sigma\sqrt{t/n}}$ is the increased factor, $d = e^{-\sigma\sqrt{t/n}} = \frac{1}{u}$ is the decreased factor, $p = (e^{rt/n} - d)/(u - d)$ is the probability of upward move and (1-p) is the probability of down move. The model that was used and implemented does not consider dividends. S_0 represents the Stock price at t=0, and the rest are the same as those of the Black-Scholes model.

For the algorithmic analysis, a full parallel and pipelined architecture with N parametric tree stages that can receive new input per cycle was considered. Table 3 presents the required operations for the Binomial method. In the following hardware implementations, only the put option was implemented, that was required for the American options.

Table 3 Algorithmic analysis of the Binomial model (no reuse of operations)

	Init.	Steps Up/ Down	Calc. Call	Calc. Put	Sum
Add/Sub	3	0	$N+2[(N-1)^2-\frac{(N-1)(N-2)}{2}]$	$N+2[(N-1)^2-\frac{(N-1)(N-2)}{2}]$	$2N+4[(N-1)^2-\frac{(N-1)(N-2)}{2}]+3$
Multiplier	3	N-1	$2[(N-1)^2-\frac{(N-1)(N-2)}{2}]$	$2[(N-1)^2-\frac{(N-1)(N-2)}{2}]$	$3N+4[(N-1)^2-\frac{(N-1)(N-2)}{2}]+2$
Divider	2	N-1	0	0	N+1
Square root	1	0	0	0	1
Exponential	2	0	0	0	2

IV. Hardware implementation of the algorithms

A. Black-Scholes implementation

The Black-Scholes model was initially implemented using VHDL in fixed point arithmetic. A straight forward, fully pipelined design with parallelization at the operations was adopted. Figure 1 presents the architecture of the Black-Scholes model where each stage is pipelined. The divider is fully pipelined with stages equal to the amount of dividend bits and the multipliers uses DSP blocks with 4 stage pipelines for better performance of the overall system. Due to large word length, more than one DSP are required for each multiplication. The *ln(.)* and *exp(.)* functions were approximated with Taylor series. The *N(.)* function was approximated using a polynomial function with 6 decimal places accuracy, according to the one used at the original software implementation. All three functions are fully parallel and pipelined. The required Black-Scholes input accuracy is at 5 integer digit and 2 fractional digit for S_0 and K variables and at 1 integer digit and 2 fractional digit for T, r and σ variables. For the fixed-point implementation 18.15 bit was used for S_0 and K and 2.15 bit for T, r and σ inputs. The implementation of the Black-Scholes kernel is parameterized at the fractional part that must be common for all inputs, the integer part and the amount of the pipeline levels used inside the multiplications, currently at 4 level for achieving higher frequency. The exponent component supports values only at the [-11, 11] field, higher value support was removed for lower resources requirements.

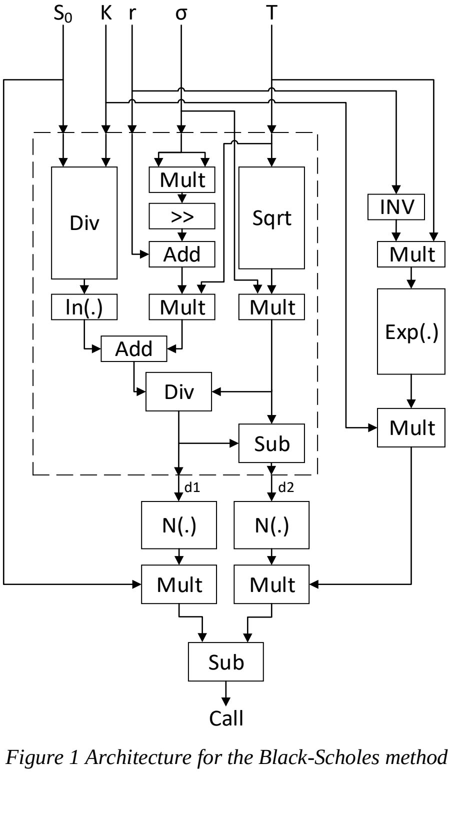


Figure 1 Architecture for the Black-Scholes method

The Black-Scholes model was also implemented using HLS in floating point single precision arithmetic. The C code was described in a way that enables the tool to produce a design closer to the described architecture. The algorithm was divided in 3 functions, the calculation of the d_1 and d_2, the calculation of the cumulative distribution function and the calculation of the call value, similar to the basic modules of the VHDL implementation. Also, each operation was stored in different variable throughout the algorithm that produced better results closer to those of the HDL implementation. The directive used from Vivado was *'#pragma HLS PIPELINE'* in each function including the top. Both implementations target the Xilinx Kintex UltraScale XCKU060 FPGA, with clock constraint at 4 ns. HDL implementation achieved 3.81 ns period and HLS implementation achieved 3.61 ns period with comparable throughputs.

B. Black-76 implementation

The Black-76 model was initially implemented using VHDL in fixed point arithmetic. Again, a straight forward, fully pipelined design with parallelization at the operations was adopted. Figure 2 presents the architecture of the Black-76 method where each stage is pipelined. The implementation uses the same internal modules that were created for the Black-Scholes method. The required Black-76 input accuracy is at 5 integer digit and 2 fractional digit for F and X variables and at 1 integer digit and 2 fractional digit for T, r and σ variables. For the fixed-point implementation 18.18 bit was used for F and X and 2.18 bit for T, r and σ inputs. Higher word length, than the Black-Scholes kernel, was used due to higher need of accuracy in the internal operations. The implementation of the Black-76 kernel is parameterized at the fractional part that must be common for all inputs, the integer part and the amount of the pipeline levels used inside the multiplications, currently at 4 level for achieving higher frequency. The exponent component supports values only at the [-11, 11] field, higher value support was removed for lower resources requirements.

The Black-76 model was also implemented using HLS in floating point single precision arithmetic. The C code was again described in a way that enables the tool to produce a design closer to the described architecture. The algorithm was again divided in 3 functions, the calculation of the d_1 and d_2, the calculation of the cumulative distribution function and the calculation of the call value, similar to the basic modules of the VHDL implementation. The directive used from Vivado was *'#pragma HLS PIPELINE'* in each function including the top. Both implementations target the Kintex UltraScale XCKU060 FPGA. For the HDL implementation, a clock constraint at 4.5ns was used and achieved 4.22 ns period, due to larger word length. This was the first indication that floating point arithmetic is required for those applications. The HLS

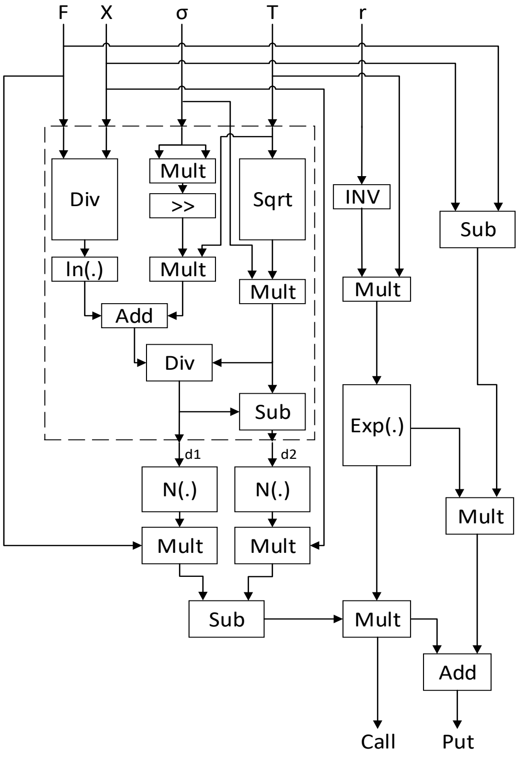


Figure 2 Architecture for the Black method

implementation achieves comparable throughput as the HDL implementation, with clock constraint at 4ns and achieved at 3.66 ns.

C. Binomial implementation

The Binomial model was initially implemented using VHDL in fixed point arithmetic. A fully pipelined design was also adopted, with the ability of folding the binary tree for lower hardware requirements. When folding is used the same Processing Elements (PEs) of a tree stage can process more than one stage of the tree, in expense of extra delays. Figure 3 presents the architecture of the Binomial method that supports parametric tree depth N and was implemented to minimize the required DSPs. The architecture is designed to be fully pipelined so new input at each cycle can be received, when no folding is used. The implementation uses the same internal modules that were created for the previous methods. All values that are generated from ascending the tree are computed first, before acceding the PEs of the tree and stored to registers that are buffered for better performance. The descendant of the tree is processed in the tree PEs. To support larger trees or smaller FPGA devices the design also support parametric folding, the ability to use the same PEs of one level to process more steps of the tree. The required Binomial input accuracy is at 5 integer digit and 2 fractional digits for S_0 and K variables and at 1 integer digit and 2 fractional digits for T, r and σ variables. For the fixed-point implementation 18.15 bit was used for S_0 and K and 2.15 bit for T, r and σ inputs. The implementation of the Binomial kernel is also parameterized at the fractional part that must be common for all inputs, the integer part, the amount of the pipeline levels used inside the multiplications, currently at 4 level for achieving higher frequency, the tree depth to be used and the folding factor. The exponent component supports values only at the [-2, 2] field, higher value support was removed for lower resources requirements.

The Binomial model was also implemented using HLS in floating point single precision arithmetic. The C code was described in a way to that enable the tool to produce a design closer to the described architecture. Each operation was stored in different variable throughout the algorithm that produced better results closer to those of the HDL implementation. The directives used from Vivado was *'#pragma HLS PIPELINE'* for the entire design, *'#pragma HLS UNROLL'* for the FOR statements and *'#pragma HLS ARRAY_PARTITION variable=Stock dim=1'* for the arrays used to store internal values. The HLS implementation achieves comparable throughput as the HDL implementation for the same Xilinx Kintex UltraScale FPGA, with clock constraint at 4ns. HDL implementation achieved 3.96 ns period and HLS implementation achieved 4.43 ns period.

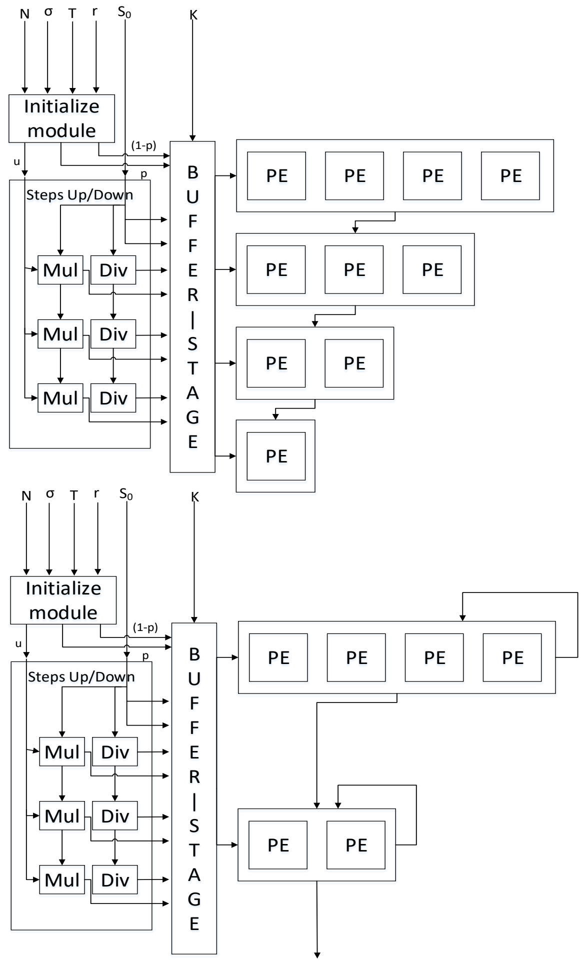


Figure 3 Architecture for the Binomial method (Folding x1 top, Folding x2 bottom, both N=4)

V. System evaluation

In this section, we compare, in terms of resources requirements and achieved precision, the VHDL and the HLS implementations for each of the three algorithms, Black-Scholes, Black-76 and Binomial. As mentioned in previous sections the VHDL implementation uses fixed point arithmetic and the HLS implementation uses floating point arithmetic. So, in our analysis we also consider the different arithmetic that those two types of implementation utilize. The utilizations presented in the following figures are for the Xilinx Kintex UltraScale FPGA and the precision was measured using the input data provided by the original software implementation.

The first three figures present the resources requirements comparison for the three implemented methods. Figure 1 presents the resources comparison for the Black-Scholes implementations. The requirements for the VHDL implementation are lower compared to the HLS implementation, except in the case of the required Flip-Flops. The increased resources requirements are mainly due to the floating-point arithmetic used in the HLS implementation, especially for the DSP blocks. So, we can see that the HLS implementation does not add extra overhead for this type of applications. We would expect to see comparable requirements if we had a floating-point HDL implementation. Figure 2 presents the resources comparison for the Black-76 implementations. In this case, the requirements for the VHDL implementation are higher compared to the HLS implementation, except in the case of the required DSPs. This algorithm required larger word lengths to achieve better precision. The word length is so large in some modules that it is preferred the use of floating point operations. The extra DSP blocks that are required in the HLS implementation, can again be attributed to the floating-point arithmetic. Figure 3 presents the resources comparison for the Binomial implementations. In this case, the requirements for the VHDL implementation is also higher for LUTs and lower for Flip-Flops and DSPs compared to the HLS implementation. At this point we should point out that both implementations are for a Binomial tree with 24 stages depth, which is the largest tree size that can fit to the targeted FPGA when produced from the HLS design. The HDL design can produce up to 28 full pipelined stages tree for the targeted FPGA. No folding was considered for either implementation. In general, we can see that the DSP requirements are always grater for the HLS implementation, but still are very close to the VHDL implementation, making the HLS implementation a good candidate for those algorithms.

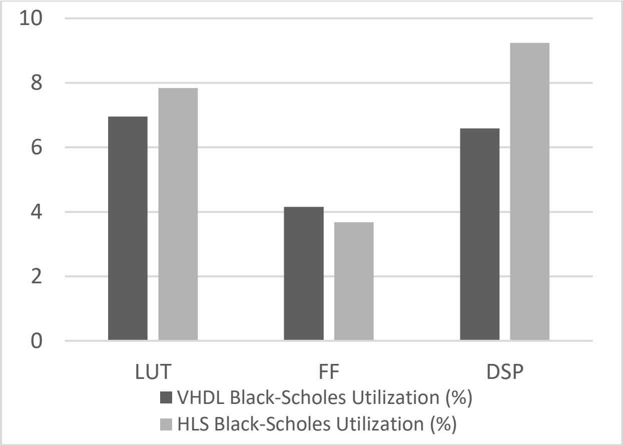


Figure 1 Resources comparison for the Black-Scholes implementations

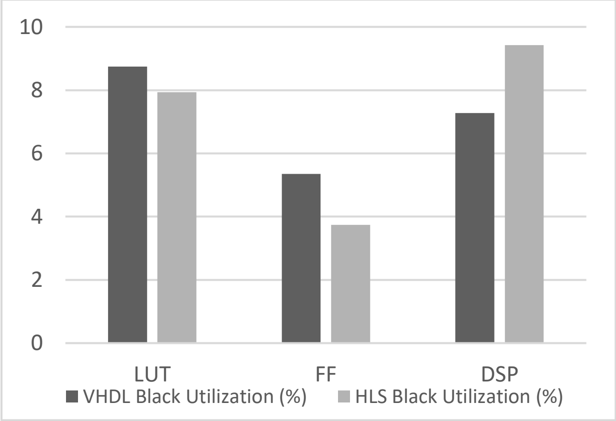


Figure 2 Resources comparison for the Black-76 implementations

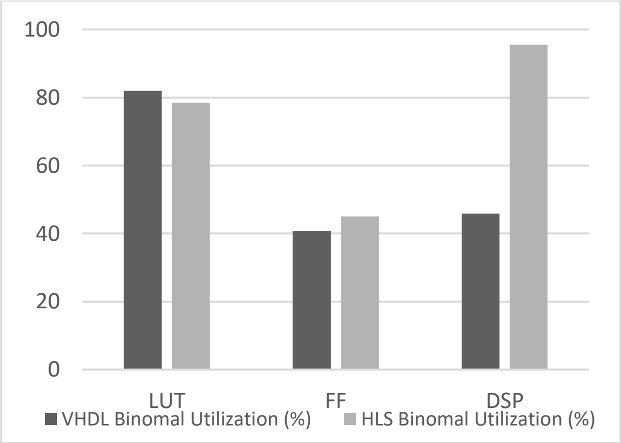


Figure 3 Resources comparison for the Binomial implementations

The next three figures present the precision comparison for the three implemented methods, where as a baseline we use the double precision floating point software implementation. As expected the single precision floating point hardware implementations provides higher precision compared to the fixed-point hardware implementations. Figure 4 presents the precision comparison for the Black-Scholes implementations. The floating-point implementation uses single precision operations and this is the reason for the errors compared to the double precision software implementation. Both implementations meet the required precision, $<10^{-3}$. Figure 5 presents the precision comparison for the Black-76 implementations. The VHDL implementation, with fixed point arithmetic, can't meet the required precision, the specified precision can only be met if floating point operations are introduced to the architecture, in specific modules. The HLS floating point implementation meet the required precision for a small increase in the required DSPs. Finally, Figure 6 presents the precision comparison for the Binomial implementations. For those implementations, the accuracy results are comparable and none implementation achieves the required accuracy, this is mainly due to the size of the tree. For those measurements, the largest trees that can be fit in the targeted FPGA were used, the HDL implementation uses a 28 stages tree and the HLS implementation a 24 stages tree. The slightly better results for the HLS implementation are because of the floating-point arithmetic, even with smaller binary tree.

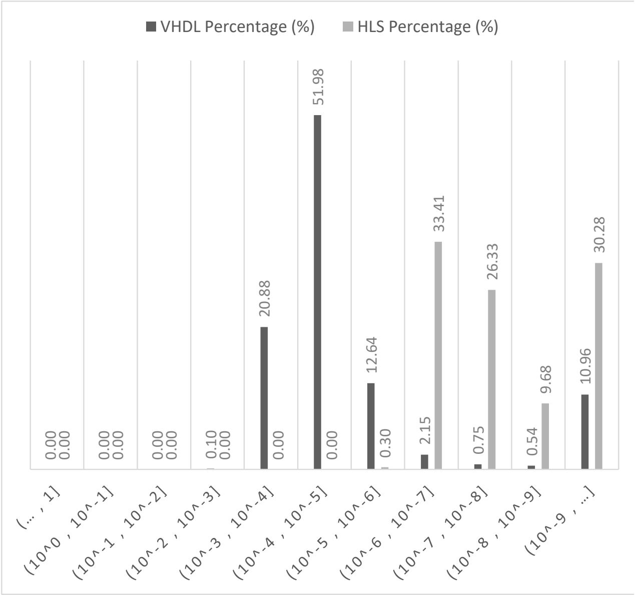


Figure 4 Precision comparison for the Black-Scholes implementations

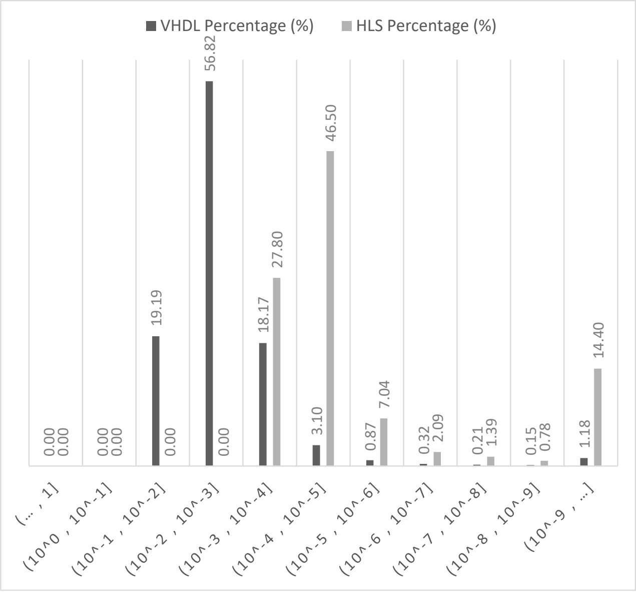


Figure 5 Precision comparison for the Black-76 implementations

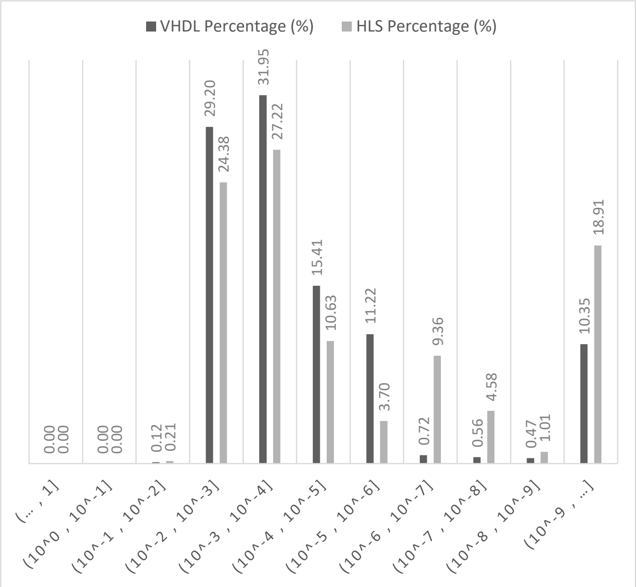


Figure 6 Precision comparison for the Binomial implementations

The software results, which we compare against the hardware, are produced from a Binomial tree of 50 stages. To achieve better accuracy the HLS implementation should utilize a 50 stages tree, which only can be accomplished, for the targeted FPGA, by using an architecture that is not fully parallel and pipelined. We used the same technique as used for the VHDL implementation, where all the nodes of the stages are processed in parallel but more than one tree stages are processed by the same processing elements. The new architecture resulted by simply changing the directives used to produce the Binomial kernel. The *'#pragma HLS PIPELINE'* directive of the entire system was removed because it forces the system to implement a fully parallel and pipelined system, then the *'#pragma HLS UNROLL'* directive for the for-loop that produces the stages were set to *'#pragma HLS UNROLL factor=8'*. Figure 7 presents a comparison between an HLS Binomial implementation using a tree with 24 stages and an HLS Binomial implementation using a tree with 50 stages. As can been seen the 50 stages implementation achieves the required accuracy having as trade of the extra delays required for each stage to process the corresponding steps. The folding implementation can process new data every 7 cycles.

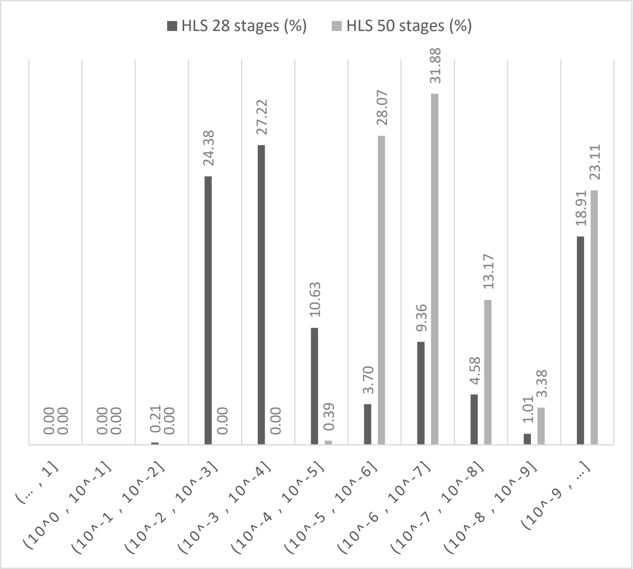


Figure 7 Precision comparison for HLS Binomial implementations of 24 and 50 stages trees

VI. Evaluation of Accelerators

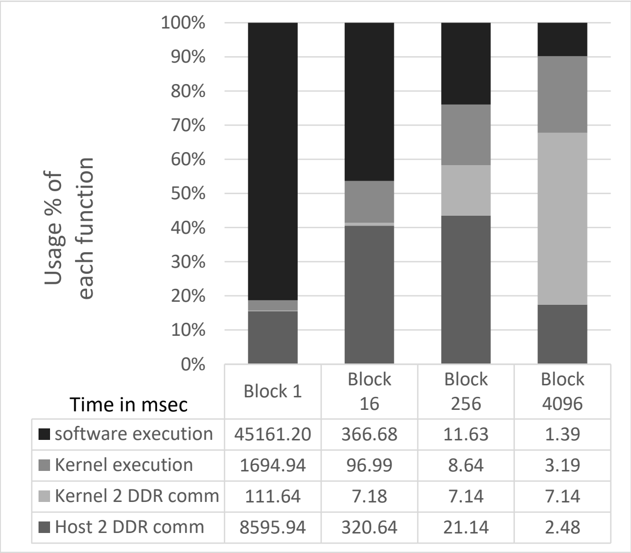
The previous analysis points out the need of floating point arithmetic for the implementation of those financial accelerators. Also, we can see that the single precision choice can provide the required accuracy for every kernel. The specific algorithms are very good candidates for HLS implementations due to their dataflow behavioral, so an HDL implementation with floating point arithmetic won't give a clear advantage over the HLS implementations, considering also the development time. The SDAccel 2016.3 tool from Xilinx was used for the integration of the kernels to a final system. The SDAccel tool provides a framework for the development of an entire system consisting of a Host running on a CPU, the kernels running on a FPGA and uses a PCIe connection for their communication. The Host PC that was used has an Intel Core i5-4590 @ 3.30GHz with 4GB RAM and CentOS 7 operating system. The FPGA board used for the execution of the time measurements is the Alpha Data ADM-PCIE-KU3 board, featuring a Xilinx Kintex Ultrascale (XCKU060 - FFVA1156) FPGA.

The access of the accelerator kernels, form the Host CPU, can be achieved by using five functions that were developed using C++. Those functions are responsible for creating the CPU-FPGA connection, programming the FPGA board with the correct kernel, initializing the memories required for the communication, sending the option values, receiving the call/put results, destroying the link between the CPU and the FPGA and freeing the reserved CPU resources. At the startup of the system the initialization function must be executed once. This function creates the device descriptors used for read/write to the accelerator and also reserves the memory required for the packets that are going to be communicated between the CPU and the FPGA. This process should be avoided to run at each call of the accelerator functions due to large execution time (6-7 seconds). To process data with one of the accelerators, Black-Scholes, Black-76 or Binomial the corresponding function must be called, that transmit the data to the kernel and receives the results. The kernels achieve higher performance when more than one options are send to be processed, because the overheads of the communication and the software execution is minimized. Also, the kernels can be executed in parallel and more options can be pipelined that increase the overall throughput. During the execution of the accelerator's functions a block of data is transmitted in the FPGA board DDR RAM memories, then the kernel stores all those values internally and process them. When all the options are processed, the results are transmitted to the DDR RAM memories and back to the Host. When the system is about to be terminated or the accelerators are no longer needed the terminating function release the host to kernel buffers and close the device descriptors, this process takes around 0.06ms.

The kernels for the acceleration of the Black-Scholes, Black-76 and Binomial algorithms were implemented using HLS C. The initial HLS implementations were used as a base and some alterations were made to support process of ‘block’ size, by creating block RAM memories to store the option values and the use of AXI interfaces. The kernels were used as develop for the Vivado tool by simply adding a new layer on the top function. Initially the kernels received option values one after the other at each cycle, now an amount of option values (block) are stored to block RAM memories in the FPGA, from the DDR memories used to communicate with the host CPU. Instead on transmitting one value at a time to the FPGA from the CPU or the DDR memories, more values are transmitted and stored internally in the FPGA’s block RAMs. Then those values are introduced one after the other to the kernel for processing. The results are stored locally to a block RAM memory and when all the options are processed those values are transmitted to the DDR memories and back to the host. Also, directives that implement the Xilinx’s AXI protocol were used for the interface of the top function. The kernel uses memory mapped master AXI interface for the data from the DDR memories and slave AXI Lite interface for control signals. All the kernels were built for the specific board and tested, providing the same accuracy results as those of the initial floating point HLS implementations.

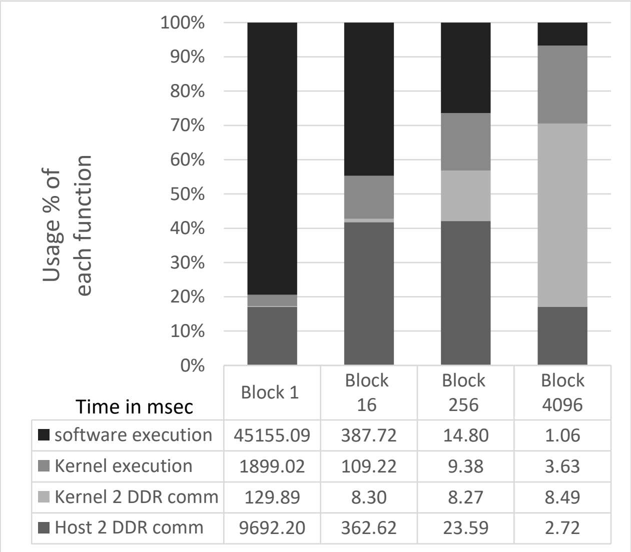
The Figures that follows presents the timing measurements of the accelerators. As mentioned earlier the kernels achieve higher performance when block of options are presented to the system, because the overhead of the communication and the software calls is minimized. To present this behavior we build the kernels for various block sizes (1, 16, 256 and 4096) and report their timing measurements. To further increase the performance more than one parallel kernels could fit in the FPGA and process in parallel several blocks, but for these experiments we use only one kernel that pipelines the block data. The Figures show the kernel execution time, the communication time between the host and the global memory (DDR memories of Alpha Data board), the communication time between the kernel and the global memory and the software execution time, for the functions used to access the accelerators in milliseconds. The sum of those times gives the total execution time measured. The results for the communication and the kernel time are produced, after the execution, from the SDAccel tool and the total execution times, that include the software overhead, were computed using the gettimeofday() function. A main function was developed for each accelerator that read the option data provided by the stock exchange market and calls the functions for the kernels. The block size represents the amount of option data that will be transmitted to the FPGA before the execution of the kernel, where each option data consists of five float values (spot, strike, time, sigma, volatility) in the cases of Black-Scholes and Binomial and six float values in the case of Black-76, which requires an additional type for informing if the request operation is for call or but option. The results that is send back to the host consist from one float value per option.

Figure 8 presents the execution and communication cost for processing 76584 options using four different block sizes, is obvious that as the block size increases so does the performance



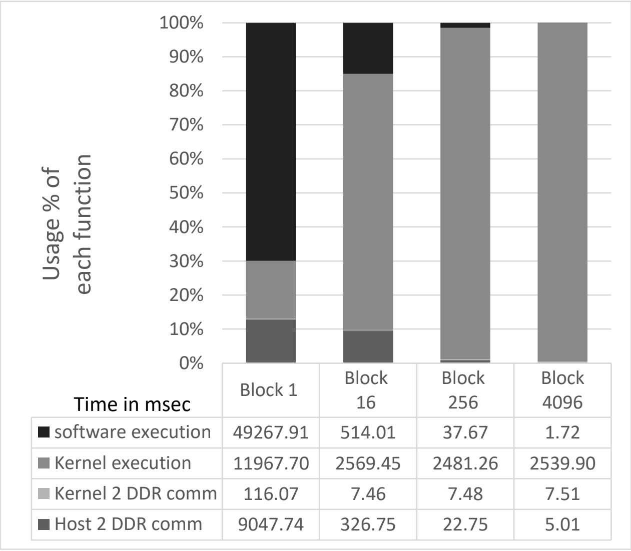
Time in msec	Block 1	Block 16	Block 256	Block 4096
■ software execution	45161.20	366.68	11.63	1.39
■ Kernel execution	1694.94	96.99	8.64	3.19
■ Kernel 2 DDR comm	111.64	7.18	7.14	7.14
■ Host 2 DDR comm	8595.94	320.64	21.14	2.48

Figure 8 Black-Scholes timing measurements in msec (76584 options)



Time in msec	Block 1	Block 16	Block 256	Block 4096
■ software execution	45155.09	387.72	14.80	1.06
■ Kernel execution	1899.02	109.22	9.38	3.63
■ Kernel 2 DDR comm	129.89	8.30	8.27	8.49
■ Host 2 DDR comm	9692.20	362.62	23.59	2.72

Figure 9 Black-76 timing measurements in msec (74400 options)



Time in msec	Block 1	Block 16	Block 256	Block 4096
■ software execution	49267.91	514.01	37.67	1.72
■ Kernel execution	11967.70	2569.45	2481.26	2539.90
■ Kernel 2 DDR comm	116.07	7.46	7.48	7.51
■ Host 2 DDR comm	9047.74	326.75	22.75	5.01

Figure 10 Binomial timing measurements in msec (79710 options)

of the accelerator. The accelerator can process this amount of options in 55.56 sec sending one option each time, 791.49 msec sending 16 options each time, 48.55 msec sending 256 options each time and 13.42 msec sending 4096 options each time. The original software implementation can process options with Black-Scholes at 0.06 msec, so for this amount of option the execution time is 4.67 sec. It is clear that in this case there is no benefit of using a hardware accelerator just for processing one option at a time, due to the additional overheads for communicating the data to the FPGA board. When bulk of data are needed to be processes then a hardware accelerator can increase the performance of the system considerably. The same conclusions can be draw for the Black-76 algorithm, the timing measurements are presented to the Figure 9. In this case, the accelerator can process 74400 options in 56.88 sec sending one option each time, 867.85 msec sending 16 options each time, 56.04 msec sending 256 options each time and 15.78 msec sending 4096 options each time. The original software implementation can process options with Black-76 at 0.06 msec, so for this amount of option it would take 4.69 sec. The Binomial accelerator presents increased performance even when a single option is transmitted, due to higher execution time of the original algorithm in software (Figure 10). Also, we can see an upper bound in the achieved performance as the block size increases. This bound exist because the implementation has recursive characteristics, as a full pipelined 50 stages tree cannot fit in the targeted FPGA. In this case the accelerator can process 79710 options in 70.40 sec sending one option each time, 3.42 sec sending 16 options each time, 2.55 sec sending 256 options each time and 2.55 sec sending 4096 options each time. The software implementation can process options with Binomial at 1.23 msec, so for this amount of option it would take 97.72 sec. Figure 11 presents the speedup achieved from the hardware acceleration of each option pricing model for the four different implementations. The Black-Scholes and Black-76 accelerators can, respectively, process data up to x348 and x297 times faster than the software implementation, due to their fully pipelined design. The Binomial accelerator, which cannot be fully pipelined due to limited resources, can process data up to x38 times faster than the software implementation.

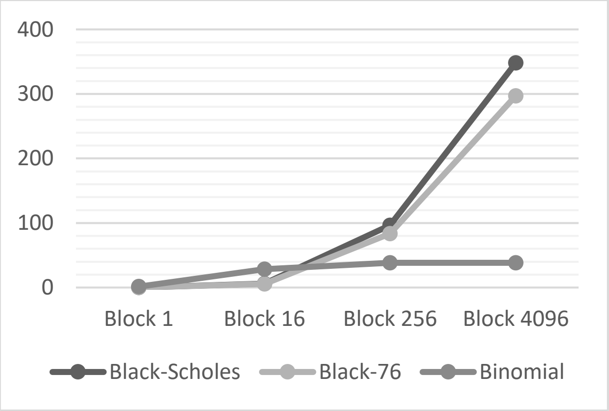


Figure 11 Accelerators Speedup

Conclusions

In this paper three hardware accelerators for financial applications were implemented using HDL and HLS. Then a thorough performance evaluation, in terms of accuracy, throughput and resource requirements was performed. The HDL implementation, for those financial applications, does not provide any clear advantage compared to the HLS implementation, where both design languages can implement high performance accelerators. However, the HLS can be used to lower the design complexity and the design time. Also, from the accuracy evaluation we conclude that floating point arithmetic should be preferred for this type of applications.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687628 - VINEYARD. We would like to thank Xilinx University Program (XUP) for the kind donation of the software tools and FPGA devices.

References

[1] S. Windh, X. Ma, R. J. Halstead, P. Budhkar, Z. Luna O. Hussaini, W. A. Najjar, High-Level Language Tools for Reconfigurable Computing. Proceedings of the IEEE, vol. 103, No. 3, pp. 390-408, 2015, doi 10.1109/JPROC.2015.2399275

[2] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, D. Stroobandt, An overview of today's high-level synthesis tools, Springer Design Automation for Embedded Systems, vol. 16, No. 3, pp 31-51, 2012, doi 10.1007/s10617-012-9096-8

[3] N. Kapre, S. Bayliss, Survey of Domain-Specific Languages for FPGA Computing, IEEE Proceedings of Field Programmable Logic and Applications, pp. 1-12, 2016, doi 10.1109/FPL.2016.7577380

[4] SDAccel Development Environment User Guide, Xilinx Inc. techncial Report, 2016

[5] Michael D. Zwagerman, High Level Synthesis, a Use Case Comparison with Hardware Description Language, Master thesis, 2015

[6] Q. Jin, W. Luk, D. B. Thomas, On Comparing Financial Option Price Solvers on FPGA, in IEEE International Symposium on Field-Programmable Custom Computing Machines, pp. 89-92, 2011, doi 10.1109/FCCM.2011.30

[7] X. Tian and K. Benkrid, Design and Implementation of a High Performance Financial Monte-Carlo Simulation Engine on an FPGA Supercomputer, in IEEE International Conference on ICECE Technology, pp. 81-88, 2008, doi 10.1109/FPT.2008.4762369

[8] J. Castillo, J. L. Bosque, E. Castillo, P. Huerta, J. I. Martinez, Hardware accelerated montecarlo financial simulation over low cost FPGA cluster, in IEEE International Symposium on Parallel & Distributed Processing, pp. 1-8, 2009, doi 10.1109/IPDPS.2009.5161209

[9] G. W. Morris and M. Aubury, Design Space Exploration of the European Option Benchmark using Hyperstreams, in International Conference on Field Programmable Logic and Applications, pp. 5-10, 2007, doi 10.1109/FPL.2007.4380617

[10] G. Inggs, S. Fleming, D. Thomas and W. Luk, Is high level synthesis ready for business? A computational finance case study, 2014 International Conference on Field-Programmable Technology (FPT), Shanghai, pp. 12-19, 2014, doi: 10.1109/FPT.2014.7082747

[11] Christian de Schryver, FPGA Based Accelerators for Financial Applications, Springer publications, Springer International Publishing, 2015, doi: 10.1007/978-3-319-15407-7

[12] A Klimovic, JH Anderson, Bitwidth-optimized hardware accelerators with software fallbac, IEEE International Conference on Field-Programmable Technology (FPT), pp. 136-143, 2013, doi 10.1109/FPT.2013.6718343

[13] F. Black and M. Scholes, The Pricing of Options and Corporate Liabilities, Journal of Political Economy, vol. 81, No. 3, 1973, pp. 637-654

[14] F. Black, The Pricing of Commodity Contracts, Journal of Financial Economics, vol. 3, Issues 1-2, 1976, pp. 167-179

[15] J.C. Cox, S.A. Ross and M. Rubinstein, Option pricing: A simplified approach, Journal of Financial Economics, vol. 7, Issue 3, 1979, pp. 229-263