A Parameterizable Activation Function Generator for FPGA-based Neural Network Applications

Sam M.H. Ho[#], C.-H. Dominic Hung[#], Ho-Cheung Ng*, Maolin Wang[#], and Hayden Kwok-Hay So[#]
[#] The University of Hong Kong, [*] Imperial College London

Abstract

- Operators used in neural network applications are often not available in FPGA vendor's core library, or may be composite operators requiring several elementary operations.
- We built an open-source, parameterized floating-point core generator, NnCore, for operators used as activation functions, and their derivatives.
- We propose a binary search algorithm to search for minimax-polynomial segments, with step to ensure monotonicity between adjacent segments.
- Input parameters are exponent and mantissa bit-width, and the acceptable approximation error.
- Experiments show that, generated cores could require lower latency, fewer resources at the expense of more BRAMs used and lower clock achieved; or they could require higher latency but comparable or less resources and clock achieved, when compared with composed cores written in HLS C++.
- Cores for non-standard floating-point widths are supported, hence allowing exploration on different number formats at application level.

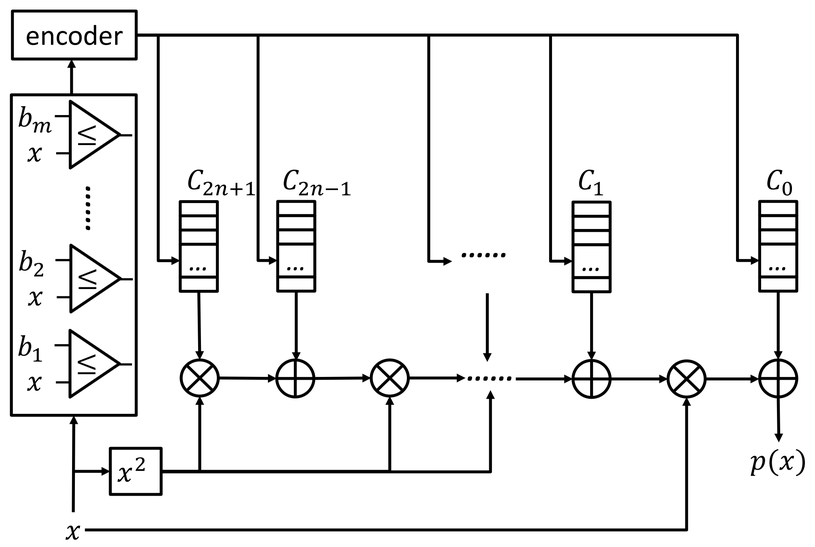
Segment Search Algorithm

1. $i \leftarrow 0,\ \max\_err \leftarrow 0,\ old\_\max\_err \leftarrow \infty,\ mono\_steps \leftarrow 1,$
2. $shrink\_dir \leftarrow SHRINK$
3. $shrink \leftarrow False,\ expand \leftarrow False,\ commit \leftarrow False$
4. $Slist \leftarrow \emptyset,\ Tlist \leftarrow \emptyset,$
5. $S_0 \leftarrow$ $[0; bound_{up}]\ if\ f\ is\ odd\ or\ even; [bound_{lo}; bound_{up}]\ otherwise$
6. $while\ not\ done\ \wedge not\ failed$
7. $\quad if\ not\ shrink\ \wedge not\ expand$
8. $\quad\quad poly \leftarrow get\_poly(S_i)$
9. $\quad\quad if\ poly = \emptyset\ then\ \{shrink \leftarrow True\}$
10. $\quad\quad \max\_err \leftarrow check\_err\_in\_range\_binade(S_i, poly)$
11. $\quad if\ \max\_err < target$
12. $\quad\quad mono \leftarrow check\_mono(S_i, poly, Slist)$
13. $\quad\quad if\ not\ mono$
14. $\quad\quad\quad S_i \leftarrow [S_{i-1_{up}} - {}^{(S_{i-1_{up}} - S_{i-1_{lo}}) \times mono\_steps}/_{128}; S_{i_{up}}]$
15. $\quad\quad\quad mono\_steps \leftarrow mono\_steps + 1$
16. $\quad\quad\quad continue$
17. $\quad\quad else$
18. $\quad\quad\quad if\ Tlist \neq \emptyset\ then\ \{Tlist \leftarrow \emptyset\}$
19. $\quad\quad\quad Tlist \leftarrow Tlist \cup S_i$
20. $\quad\quad\quad expand \leftarrow True$
21. $\quad\quad else$
22. $\quad\quad\quad if\ not\ \max\_err < old\_\max\_err\ then\ \{\ flip\_shrink\_dir(\ )\ \}$
23. $\quad\quad\quad if\ shrink\_dir\ is\ SHRINK\ then\ \{shrink \leftarrow True\}$
24. $\quad\quad\quad else\ \{expand \leftarrow True\}$
25. $\quad\quad old\_\max\_err \leftarrow \max\_err$
26. $if\ expand\ \{\ expand\_seg(S_i)\ \}$
27. $if\ shrink\ \{\ shrink\_seg(S_i)\ \}$
28. $if\ commit$
29. $\quad if\ Tlist \neq \emptyset$
30. $\quad\quad if\ s_{up} = bound_{up}\ where\ s = [s_{lo}; s_{up}] \in Tlist$
31. $\quad\quad\quad done \leftarrow True$
32. $\quad\quad Slist \leftarrow Slist \cup Tlist$
33. $\quad\quad Tlist \leftarrow \emptyset$
34. $\quad else$
35. $\quad\quad failed \leftarrow True$

- Not shown above: *mono_steps* resets to 1, and *shrink_dir* resets to SHRINK inside the "*shrink_seg*" or "*expand_seg*" processes.
- $Bound_{lo}$ and $Bound_{up}$ are the *X* values where the functions → limit.
- The "*shrink_seg*" and "*expand_seg*" processes modifies $S_{i_{up}}$ by bisecting the "last failed upper bound" and the 1) last failed lower bound, 2) last succeeded upper bound, respectively, hence the "binary search".

Core Designs

- The NnCore generates ReLU, ReLU6, Tanh, Sigmoid, d'Tanh and d'Sigmoid operators.
- Except for ReLU and ReLU6, the segmentor is a naïve implementation, where for *n* polynomial segments we use *n* comparators, and this *n*-bit output is feed to an encoder to generate the corresponding coefficient ROM address.
- The Pros of this scheme is its simplicity, and that it does not require any ROM space for the segment boundaries b_m. Also, the latency of the circuit does not increase as number of segments increase in this scheme. As the number of segment from the algorithm is small such scheme is viable.
- It also allows optimizations of the comparators during synthesis since the boundaries are hardwired.
- NnCore itself is written in Python, where generated cores are in Verilog, with floating-point adder, multiplier and squarer(odd-power variant) from the FloPoCo project being instantiated.
- Polynomial evaluation is done in the simple Horner's form, e.g.: $p(y) = a_0 + x \times (a_1 + x \times (a_2 + \ \ldots + x \times (a_{d-1} + \ a_d x)))$.
- The all-power and odd-power-only variants only differs in the need of a squarer, and the number of adder/multipliers.



Experimental Results

- Experiment target were set to an Alpha-Data ADM-PCIE-7V3 board, featuring a Xilinx xc7vx690tffg1157-2 chip, target clock speed = 250MHz.
- **Results for TANH at $\omega_E = 8$, $\omega_F = 23$, approx_err = 1 ULP**

	Latency	WNS(ns)	CLK(MHz)	LUTs	FFs	DSPs	BRAMs
HLS(sinhcosh)	93	0.247	266	18322	10246	30	0
HLS(expf)	42	**0.249**	**267**	3365	2494	18	0
NnCore(d=3)	**34**	-1.049	198.1	**1742**	**1498**	**6**	4
NnCore(d=5)	61	0.183	262	2747	2843	10	0

- At single-precision (no denormal numbers), generated Tanh cores can reach 1-ULP (unit of least precision) approximation error.
- At max polynomial degree 3, the core has lower latency (cycles), requires fewer LUTs, flip-flops and DSPs, but at the expanse of using some BRAMs and a lower achieved clock rate.
- At max polynomial degree 5, the core still uses less LUTs and DSPs, and no BRAMs, also a comparable achieved clock rate, but requires higher latency due to the Horner form polynomial evaluation.
- **Results of generated cores with ODD-POWERS ONLY, at $\omega_E = 5$, $\omega_F = 10$**

	Deg.	Err(ULP)	Lat.	WNS(ns)	CLK(MHz)	LUTs	FFs	DSPs	BRAMs
Tanh	3	1	21	0.124	258.0	541	579	3	0
Sigmoid	3	2	21	0.035	252.2	650	589	3	0
d'Tanh	3	2	21	0.014	250.9	574	584	3	0
d'Sigmoid	3	2	21	0.149	259.7	631	583	3	0

- At half-precision (no denormal numbers), generated cores requires no BRAMs with degree 3 odd-power polynomials.
- Such operators wouldn't have been available without our NnCore generator in Xilinx Vivado HLS, since only elementary operators (+, −,×,÷) are available but not the *exp* or *sinh/cosh* required in the above operators.

Imperial College London

香 港 大 學 THE UNIVERSITY OF HONG KONG