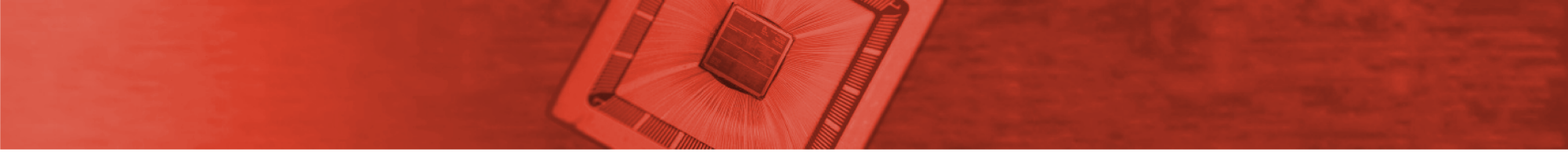


Design and Implementation of a Neural Network Aided Self-Interference Cancellation Scheme for Full-Duplex Radios

Yann Kurzo, Andreas Burg, **Alexios Balatsoukas-Stimming**

Telecommunications Circuits Laboratory
Ecole polytechnique fédérale de Lausanne, Switzerland

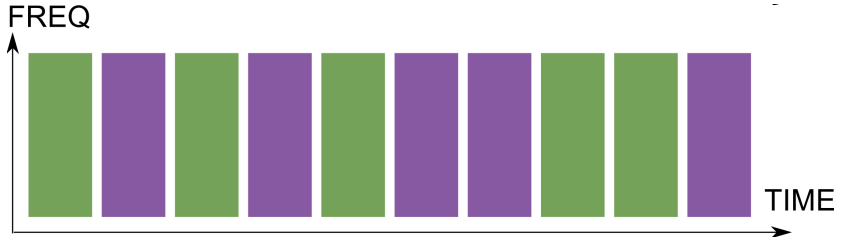
February 27, 2019
Cornell University

(presented at Asilomar 2018)

Bi-directional Wireless Communications

Time-division duplexing (TDD)

Wasted time resources: switching interval



Frequency-division duplexing (FDD)

Wasted frequency resources: guard bands

In-Band Full-duplex (IBFD)

Up to twice the throughput wrt TDD & FDD!

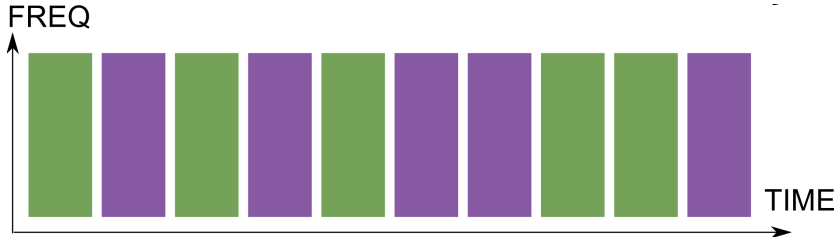
No additional bandwidth

No wasted time or frequency resources

Bi-directional Wireless Communications

Time-division duplexing (TDD)

Wasted time resources: switching interval



Frequency-division duplexing (FDD)

Wasted frequency resources: guard bands

In-Band Full-duplex (IBFD)

Up to twice the throughput wrt TDD & FDD!

No additional bandwidth

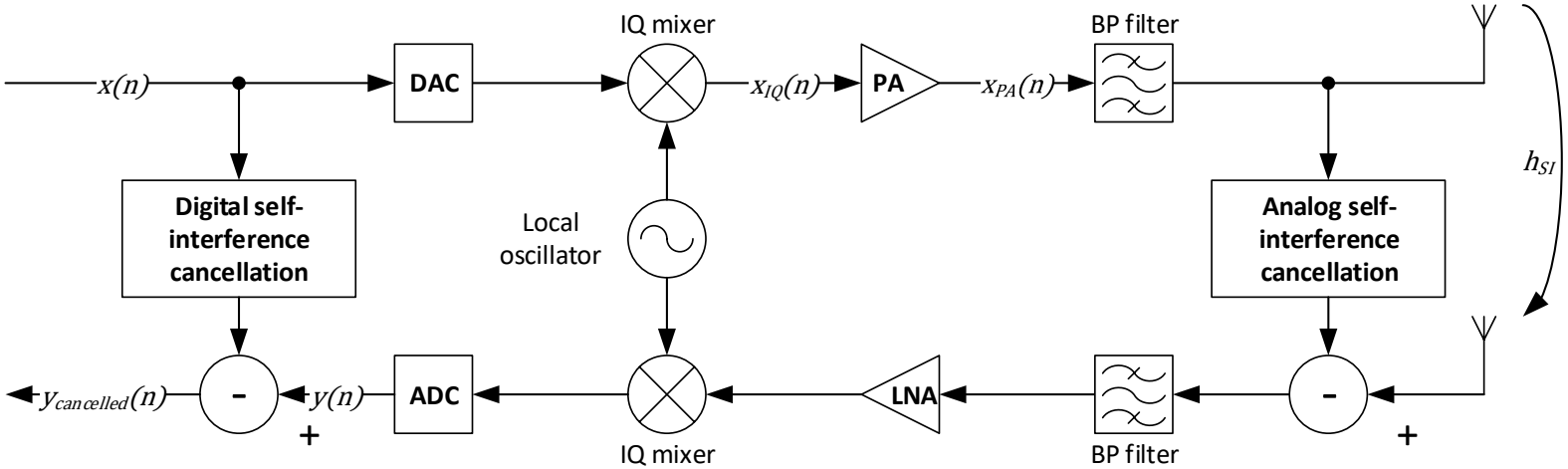
No wasted time or frequency resources

Fundamental Challenge

Self-interference is much stronger than the desired signal! (about 110 dB)

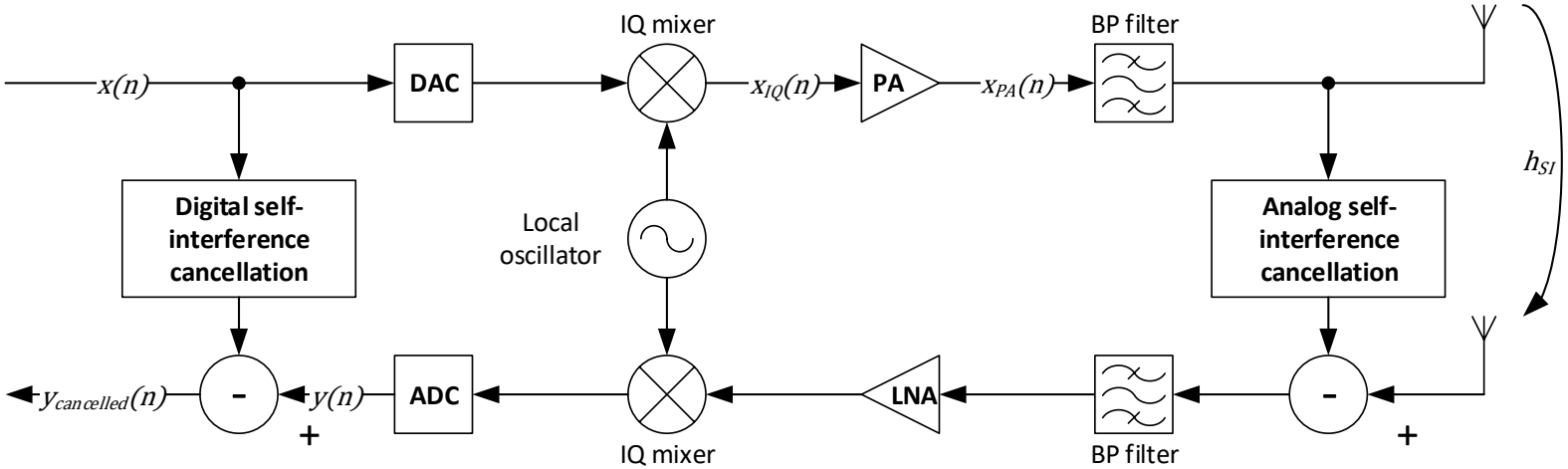
Self-Interference Cancellation in Full-Duplex Radios

- **In principle**, cancellation is easy since digital transmitted signal is known!
- **In practice**, the digital signal does not tell the whole story.



Self-Interference Cancellation in Full-Duplex Radios

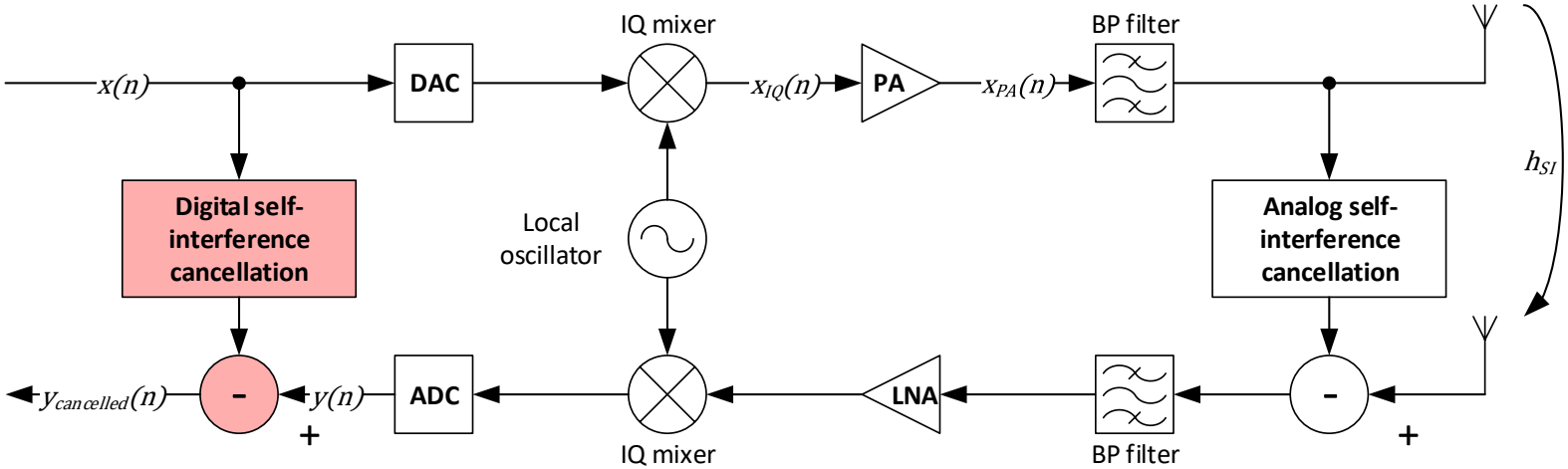
- **In principle**, cancellation is easy since digital transmitted signal is known!
- **In practice**, the digital signal does not tell the whole story.



- **Three-stage** cancellation process:
  1. Passive analog cancellation
  2. Active analog cancellation
  3. Active digital cancellation

Self-Interference Cancellation in Full-Duplex Radios

- **In principle**, cancellation is easy since digital transmitted signal is known!
- **In practice**, the digital signal does not tell the whole story.



- **Three-stage** cancellation process:
  1. Passive analog cancellation
  2. Active analog cancellation
  3. Active digital cancellation

Focus of this work: digital SI cancellation

In order to achieve high self-interference cancellation, **non-linear** component effects need to be taken into account!

Self-Interference Cancellation Using a Polynomial Model

- Main transceiver non-linearities:
  1. **Amplifiers:** Odd harmonics (even harmonics lie out of band when filtered)
  2. **IQ Mixer:** Phase noise and IQ imbalance

Self-Interference Cancellation Using a Polynomial Model

- Main transceiver non-linearities:
  1. **Amplifiers:** Odd harmonics (even harmonics lie out of band when filtered)
  2. **IQ Mixer:** Phase noise and IQ imbalance

- State-of-the-art non-linear cancellation model:

$$y(n) = \sum_{\substack{p=1,\\ p \text{ odd}}}^{P} \sum_{q=0}^{p} \sum_{l=0}^{L-1} h_{p,q}(l) \underbrace{x(n-l)^q x^*(n-l)^{p-q}}_{\text{basis functions}}$$

1. D. Korpi, L. Anttila, and M. Valkama, "Nonlinear self-interference cancellation in MIMO full-duplex transceivers under crosstalk," EURASIP Journal on Wireless Communications and Networking, Feb. 2017

Self-Interference Cancellation Using a Polynomial Model

- Main transceiver non-linearities:
  1. **Amplifiers:** Odd harmonics (even harmonics lie out of band when filtered)
  2. **IQ Mixer:** Phase noise and IQ imbalance

- State-of-the-art non-linear cancellation model:

$$y(n) = \sum_{\substack{p=1,\\ p \text{ odd}}}^{P} \sum_{q=0}^{p} \sum_{l=0}^{L-1} h_{p,q}(l) \underbrace{x(n-l)^q x^*(n-l)^{p-q}}_{\text{basis functions}}$$

Highly Redundant

Most terms in the above equation **contribute very little** to the final result!

1. D. Korpi, L. Anttila, and M. Valkama, "Nonlinear self-interference cancellation in MIMO full-duplex transceivers under crosstalk," EURASIP Journal on Wireless Communications and Networking, Feb. 2017

Self-Interference Cancellation Using a Polynomial Model

- Main transceiver non-linearities:
  1. **Amplifiers:** Odd harmonics (even harmonics lie out of band when filtered)
  2. **IQ Mixer:** Phase noise and IQ imbalance

- State-of-the-art non-linear cancellation model:

$$y(n) = \sum_{\substack{p=1,\\ p \text{ odd}}}^{P} \sum_{q=0}^{p} \sum_{l=0}^{L-1} h_{p,q}(l) \underbrace{x(n-l)^q x^*(n-l)^{p-q}}_{\text{basis functions}}$$

Highly Redundant

Most terms in the above equation **contribute very little** to the final result!

Alternative Approach

Use a **neural network** to reproduce the SI non-linearities.

1. D. Korpi, L. Anttila, and M. Valkama, "Nonlinear self-interference cancellation in MIMO full-duplex transceivers under crosstalk," EURASIP Journal on Wireless Communications and Networking, Feb. 2017
2. A. Balatsoukas-Stimming, "Non-linear digital self-interference cancellation for in-band full-duplex radios using neural networks," IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Jun. 2018

Self-Interference Cancellation Using Neural Networks

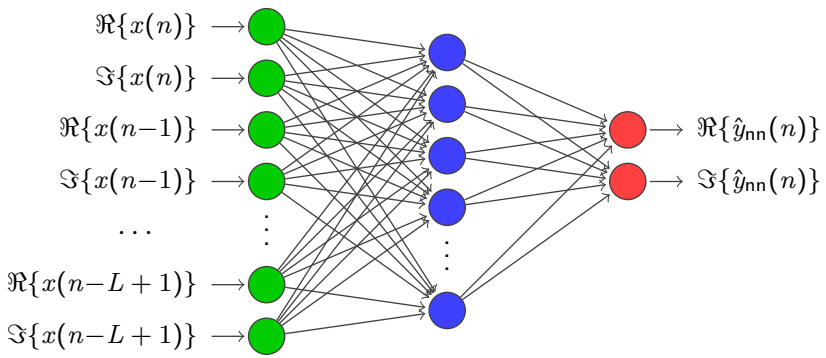
- **Focus separately** on linear and non-linear SI: $y(n) = \underbrace{y_{\text{lin}}(n)}_{\text{easy!}} + \underbrace{y_{\text{nn}}(n)}_{\text{hard!}}$

Self-Interference Cancellation Using Neural Networks

- **Focus separately** on linear and non-linear SI: $y(n) = \underbrace{y_{\text{lin}}(n)}_{\text{easy!}} + \underbrace{y_{\text{nn}}(n)}_{\text{hard!}}$
- **Two-step cancellation:**
  1. Use standard linear digital cancellation: $\hat{y}_{\text{lin}}(n) = \sum_{m=0}^{M+L-1} \hat{h}_{1,1}(m)x(n-m)$

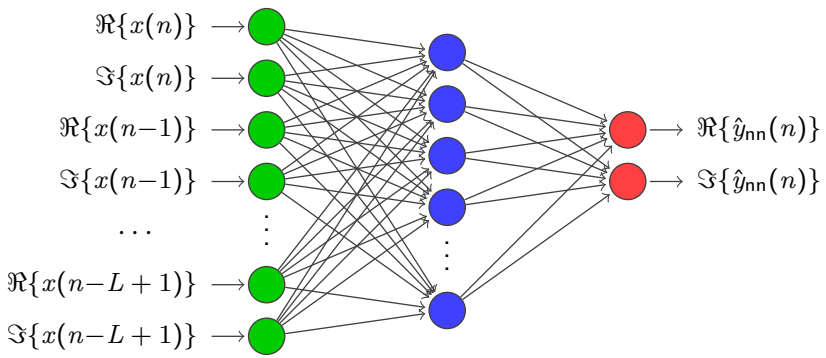
Self-Interference Cancellation Using Neural Networks

- **Focus separately** on linear and non-linear SI: $y(n) = \underbrace{y_{\text{lin}}(n)}_{\text{easy!}} + \underbrace{y_{\text{nn}}(n)}_{\text{hard!}}$
- **Two-step cancellation:**
  1. Use standard linear digital cancellation: $\hat{y}_{\text{lin}}(n) = \sum_{m=0}^{M+L-1} \hat{h}_{1,1}(m)x(n-m)$
  2. Train a neural network (NN) to reproduce and cancel $y_{\text{nn}}(n) \approx y(n) - \hat{y}_{\text{lin}}(n)$



Self-Interference Cancellation Using Neural Networks

- **Focus separately** on linear and non-linear SI: $y(n) = \underbrace{y_{\text{lin}}(n)}_{\text{easy!}} + \underbrace{y_{\text{nn}}(n)}_{\text{hard!}}$
- **Two-step cancellation:**
  1. Use standard linear digital cancellation: $\hat{y}_{\text{lin}}(n) = \sum_{m=0}^{M+L-1} \hat{h}_{1,1}(m)x(n-m)$
  2. Train a neural network (NN) to reproduce and cancel $y_{\text{nn}}(n) \approx y(n) - \hat{y}_{\text{lin}}(n)$

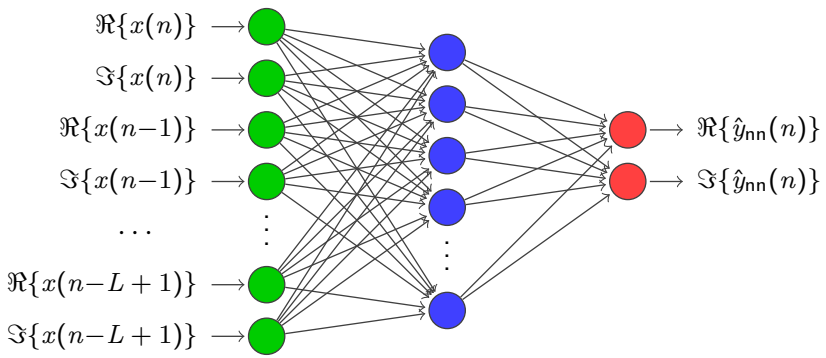


Identical SI cancellation with 35% lower complexity than the polynomial model!

1. A. Balatsoukas-Stimming, "Non-linear digital self-interference cancellation for in-band full-duplex radios using neural networks," IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Jun. 2018

Self-Interference Cancellation Using Neural Networks

- **Focus separately** on linear and non-linear SI: $y(n) = \underbrace{y_{\text{lin}}(n)}_{\text{easy!}} + \underbrace{y_{\text{nn}}(n)}_{\text{hard!}}$
- **Two-step cancellation:**
  1. Use standard linear digital cancellation: $\hat{y}_{\text{lin}}(n) = \sum_{m=0}^{M+L-1} \hat{h}_{1,1}(m)x(n-m)$
  2. Train a neural network (NN) to reproduce and cancel $y_{\text{nn}}(n) \approx y(n) - \hat{y}_{\text{lin}}(n)$



Identical SI cancellation with 35% lower complexity than the polynomial model!

Goals of this work

1. To present one of the **first implementations** of an ML-augmented comms system.
2. To **quantify the hardware implementation gains** w.r.t. a traditional implementation.

1. A. Balatsoukas-Stimming, "Non-linear digital self-interference cancellation for in-band full-duplex radios using neural networks," IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Jun. 2018

High-Level Architectures of Polynomial and NN Cancellers

- We implement both a polynomial canceller and an NN canceller.

High-Level Architectures of Polynomial and NN Cancellers

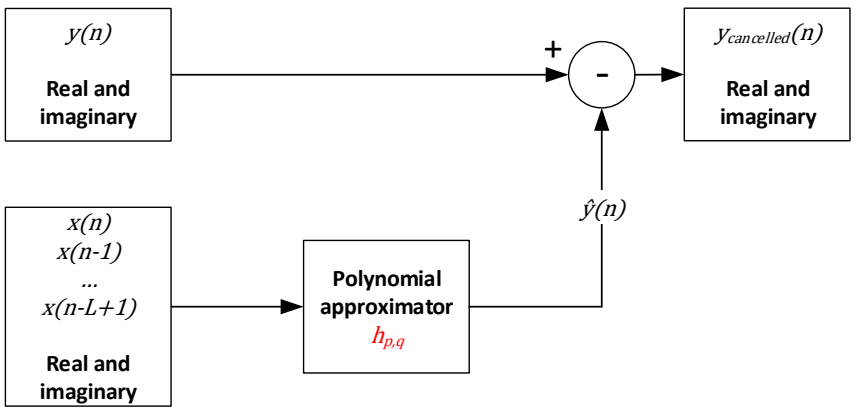
- We implement both a polynomial canceller and an NN canceller.
- For both hardware implementations, we focus on **inference** only.
  - Highly configurable at synthesis time (memory L, polynomial order P, NN structure, bit-width, number of processing elements).

High-Level Architectures of Polynomial and NN Cancellers

- We implement both a polynomial canceller and an NN canceller.
- For both hardware implementations, we focus on **inference** only.
  - Highly configurable at synthesis time (memory L, polynomial order P, NN structure, bit-width, number of processing elements).
- **Main computational effort:** weighted sums
  - Polynomial canceller also needs to compute **basis functions!**

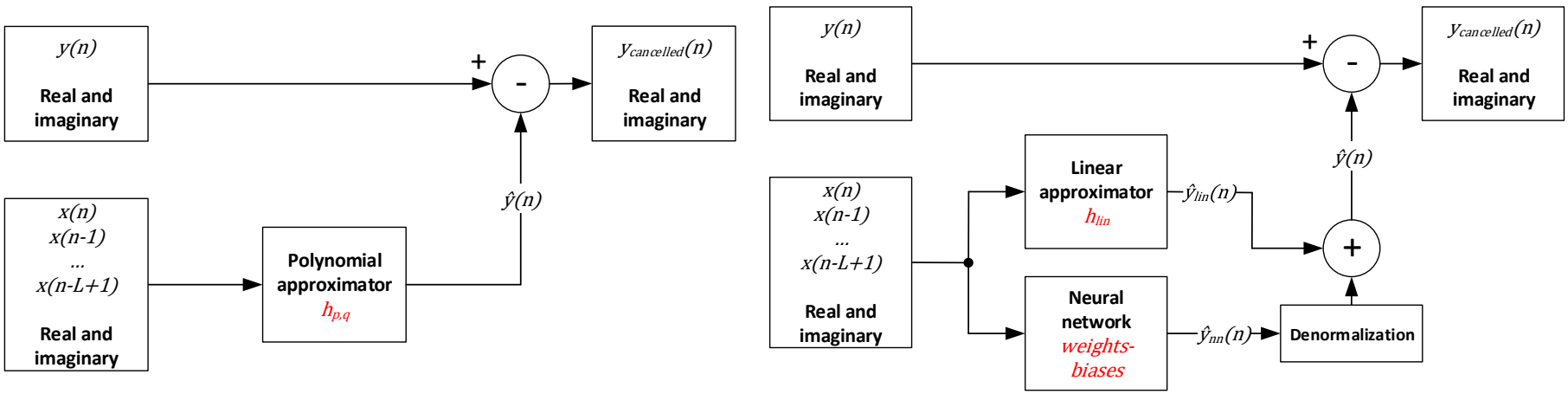
High-Level Architectures of Polynomial and NN Cancellers

- We implement both a polynomial canceller and an NN canceller.
- For both hardware implementations, we focus on **inference** only.
  - Highly configurable at synthesis time (memory L, polynomial order P, NN structure, bit-width, number of processing elements).
- **Main computational effort:** weighted sums
  - Polynomial canceller also needs to compute **basis functions!**



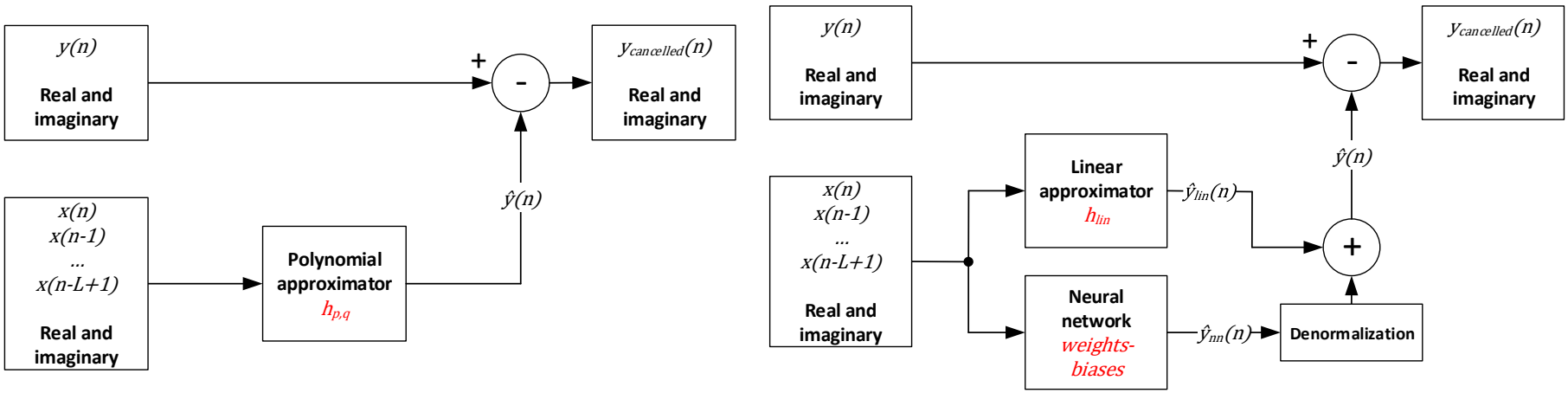
High-Level Architectures of Polynomial and NN Cancellers

- We implement both a polynomial canceller and an NN canceller.
- For both hardware implementations, we focus on **inference** only.
  - Highly configurable at synthesis time (memory L, polynomial order P, NN structure, bit-width, number of processing elements).
- **Main computational effort:** weighted sums
  - Polynomial canceller also needs to compute **basis functions!**



High-Level Architectures of Polynomial and NN Cancellers

- We implement both a polynomial canceller and an NN canceller.
- For both hardware implementations, we focus on **inference** only.
  - Highly configurable at synthesis time (memory L, polynomial order P, NN structure, bit-width, number of processing elements).
- **Main computational effort:** weighted sums
  - Polynomial canceller also needs to compute **basis functions!**

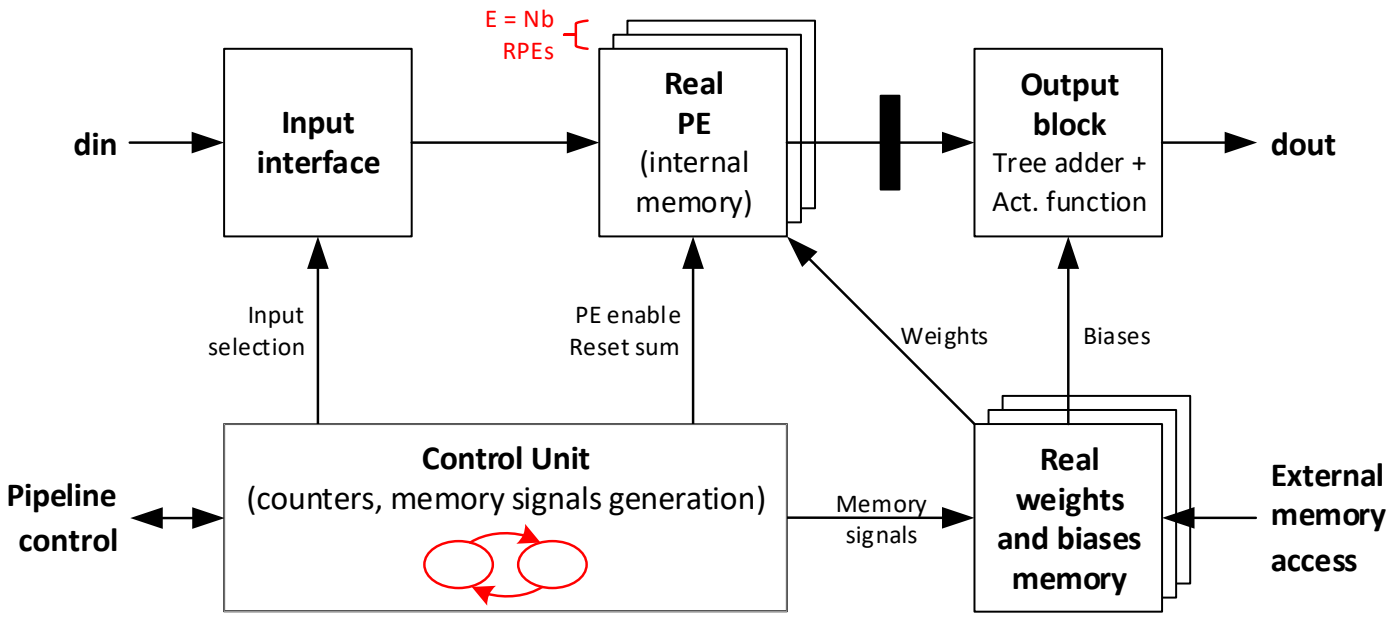


Computational Complexity (excluding BF calculation)

	N_{cmul}	N_{cadd}	N_{rmul}	N_{rmul}
Polynomial	$\left(\frac{P+1}{2}\right)\left(\frac{P+1}{2}+1\right)L$	$N_{\text{cmul}}-1$	n/a	n/a
Neural Network	L	$L-1$	$(2L+2)N_{\text{h}}$	$(2L+3)N_{\text{h}}$

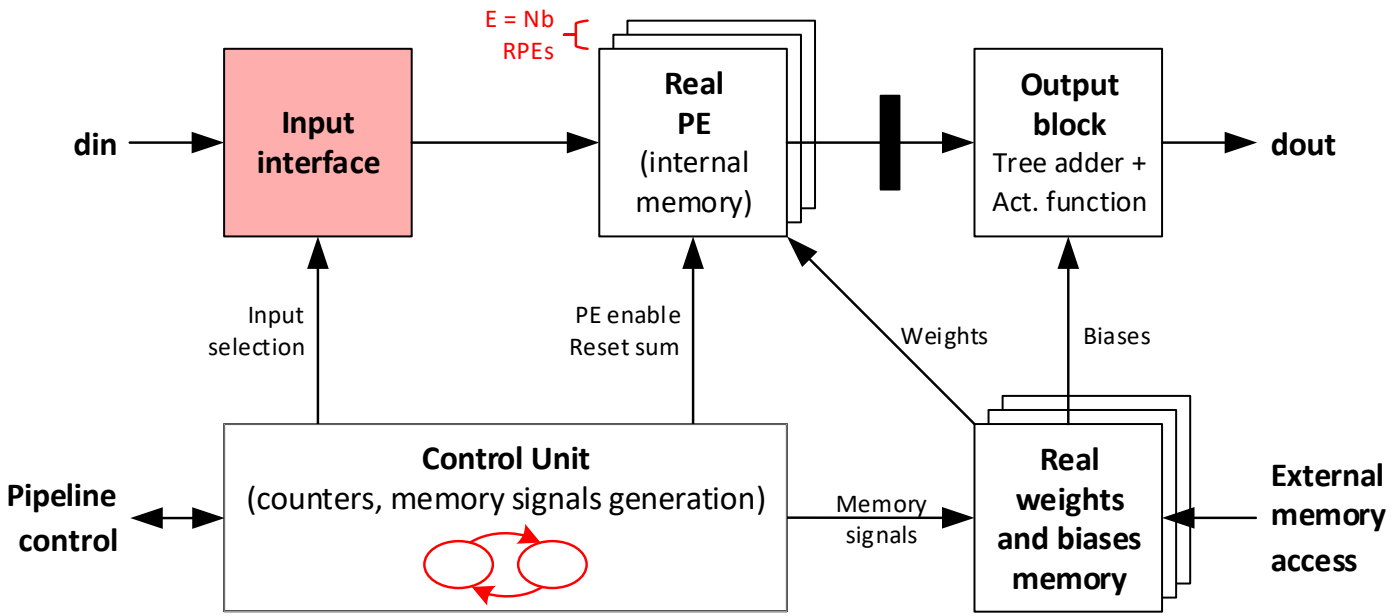
Neural Network Canceller Architecture: Single Layer of Neurons

- Generic hardware architecture to process **multiple real weighted sums**:



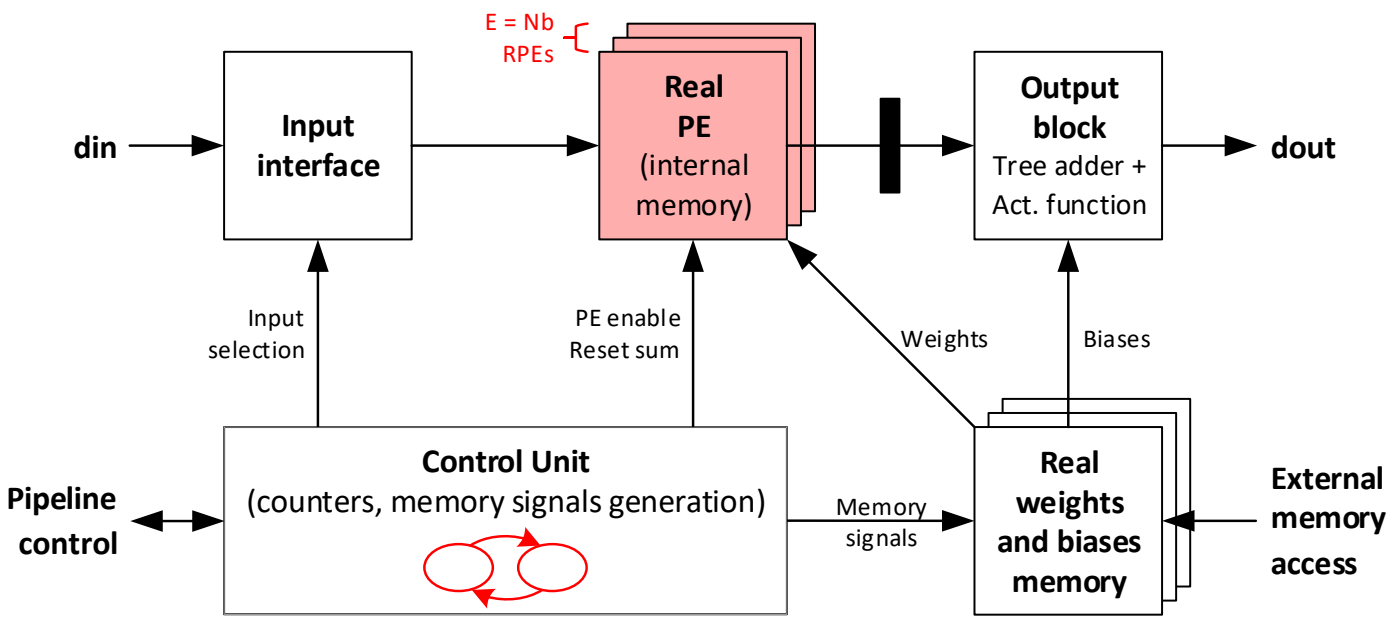
Neural Network Canceller Architecture: Single Layer of Neurons

- Generic hardware architecture to process **multiple real weighted sums**:
  1. **Input interface:** redirects inputs to correct real processing elements (RPEs).



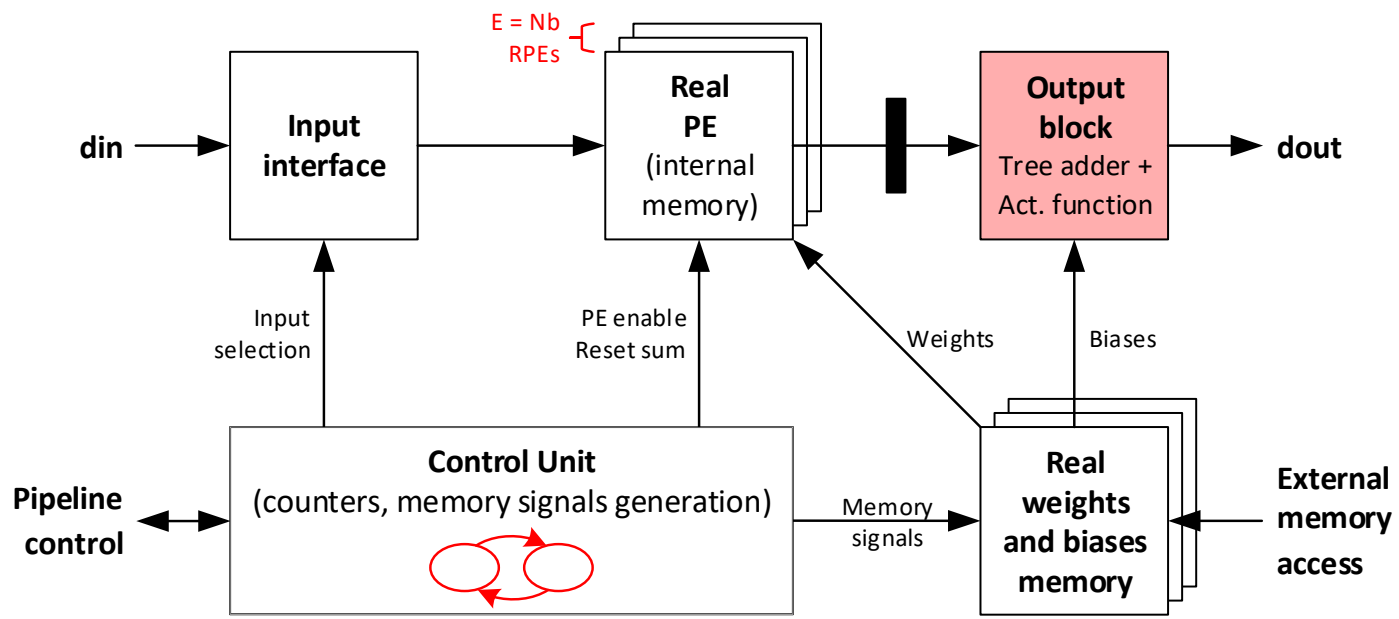
Neural Network Canceller Architecture: Single Layer of Neurons

- Generic hardware architecture to process **multiple real weighted sums**:
  1. **Input interface:** redirects inputs to correct real processing elements (RPEs).
  2. **RPE:** multiply-and-accumulate unit.



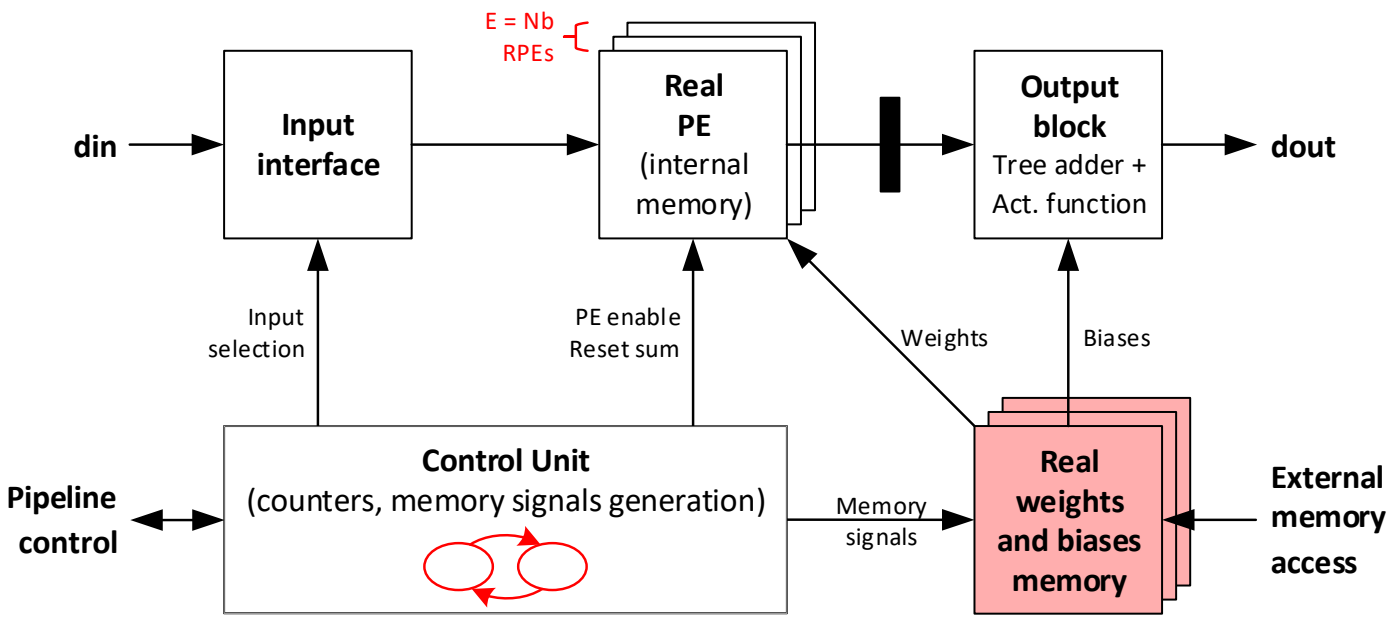
Neural Network Canceller Architecture: Single Layer of Neurons

- Generic hardware architecture to process **multiple real weighted sums**:
  1. **Input interface:** redirects inputs to correct real processing elements (RPEs).
  2. **RPE:** multiply-and-accumulate unit.
  3. **Output block:** partial sum addition and activation function.



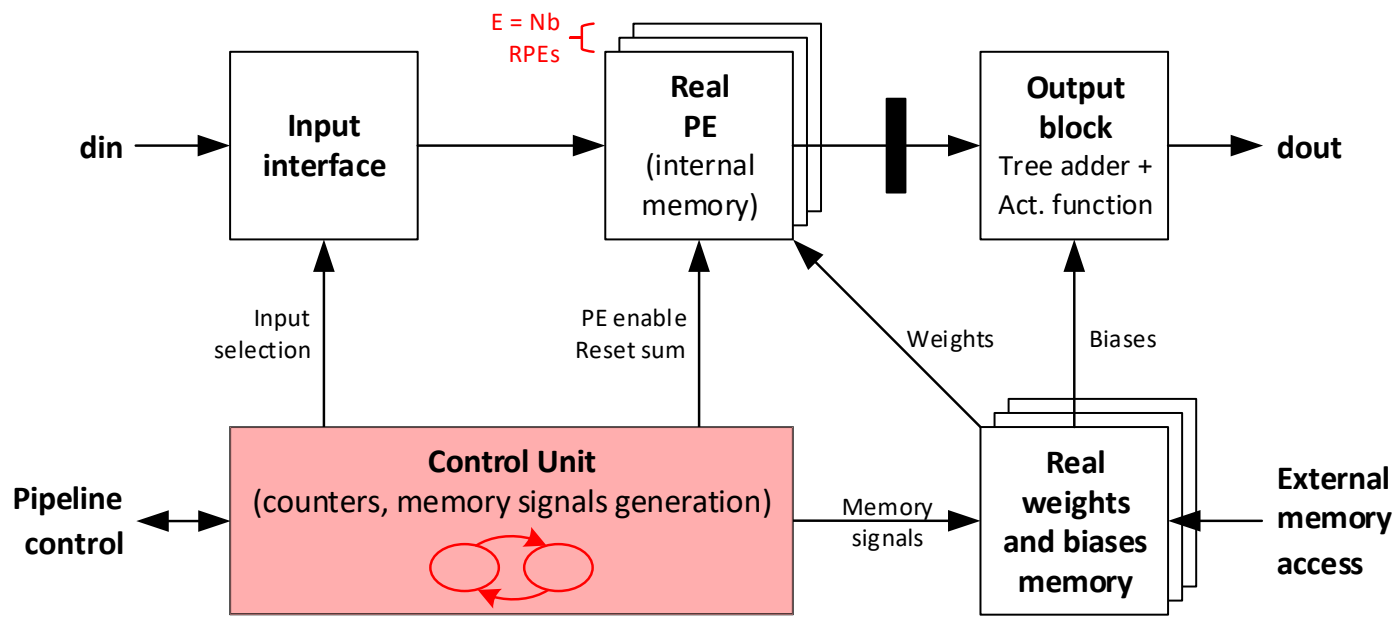
Neural Network Canceller Architecture: Single Layer of Neurons

- Generic hardware architecture to process **multiple real weighted sums**:
  1. **Input interface:** redirects inputs to correct real processing elements (RPEs).
  2. **RPE:** multiply-and-accumulate unit.
  3. **Output block:** partial sum addition and activation function.
  4. **Memory:** stores weights and biases and can be written externally.



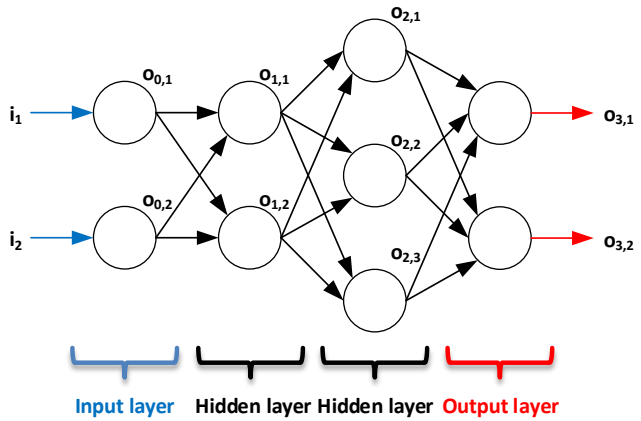
Neural Network Canceller Architecture: Single Layer of Neurons

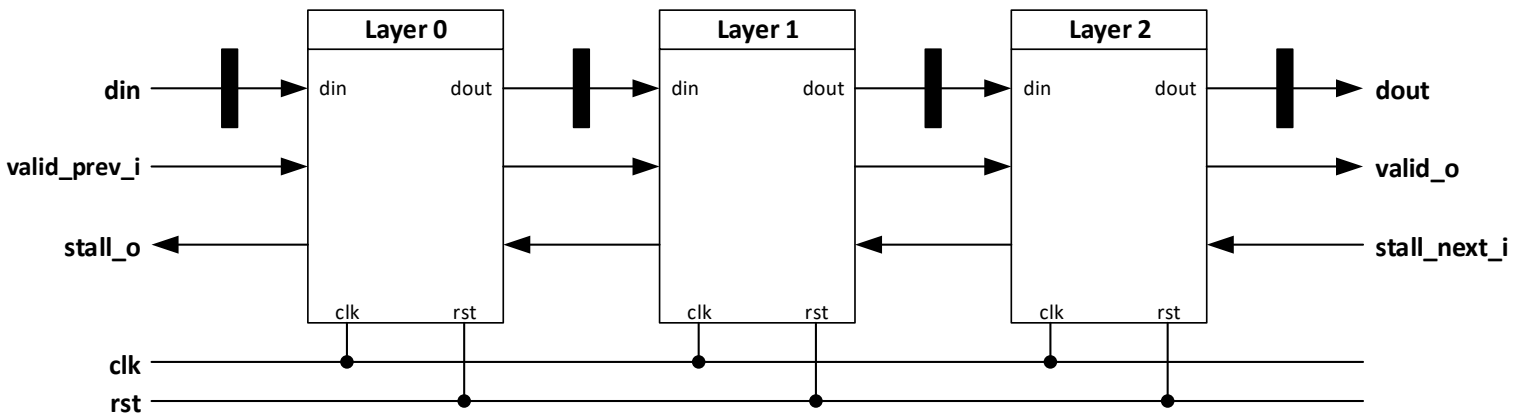
- Generic hardware architecture to process **multiple real weighted sums**:
  1. **Input interface:** redirects inputs to correct real processing elements (RPEs).
  2. **RPE:** multiply-and-accumulate unit.
  3. **Output block:** partial sum addition and activation function.
  4. **Memory:** stores weights and biases and can be written externally.
  5. **Control unit:** ensures correct operation for a wide range of parameters.



Neural Network Canceller Architecture: Multi-Layer Pipeline

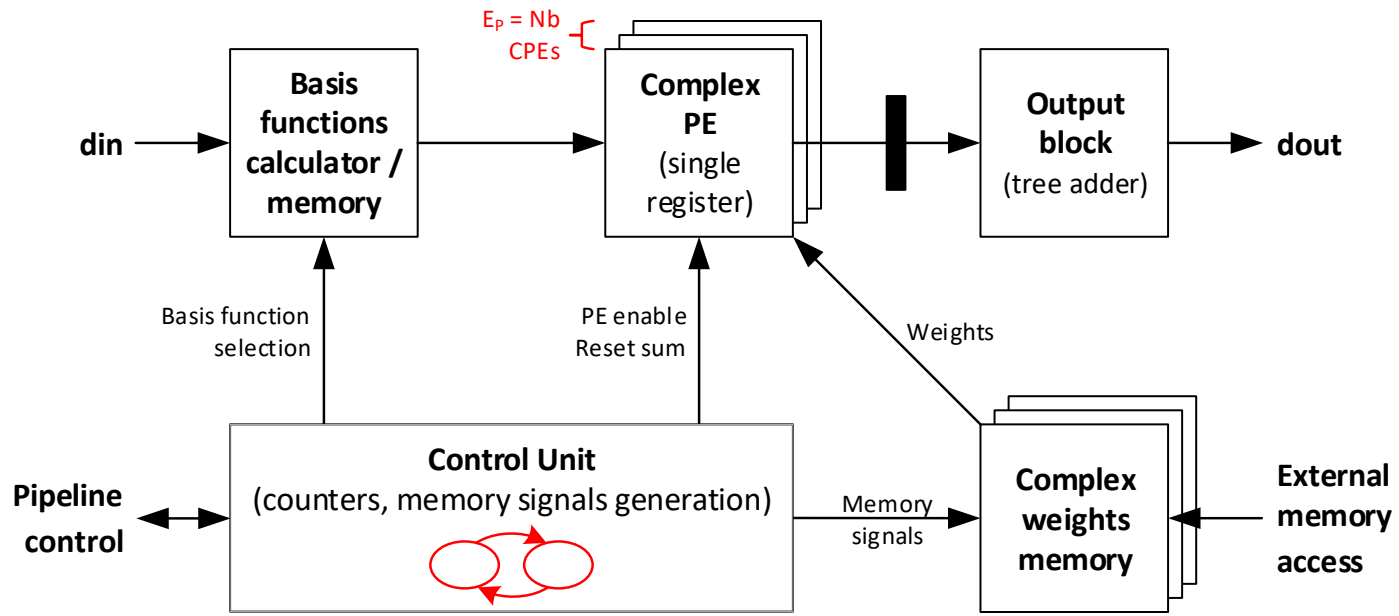
- Each layer is equivalent to one **pipeline stage**.
- Pipeline can be balanced by varying the number of **RPEs**.
- Neural network is optimized by groups of 2 layers by playing with the neurons calculation order.





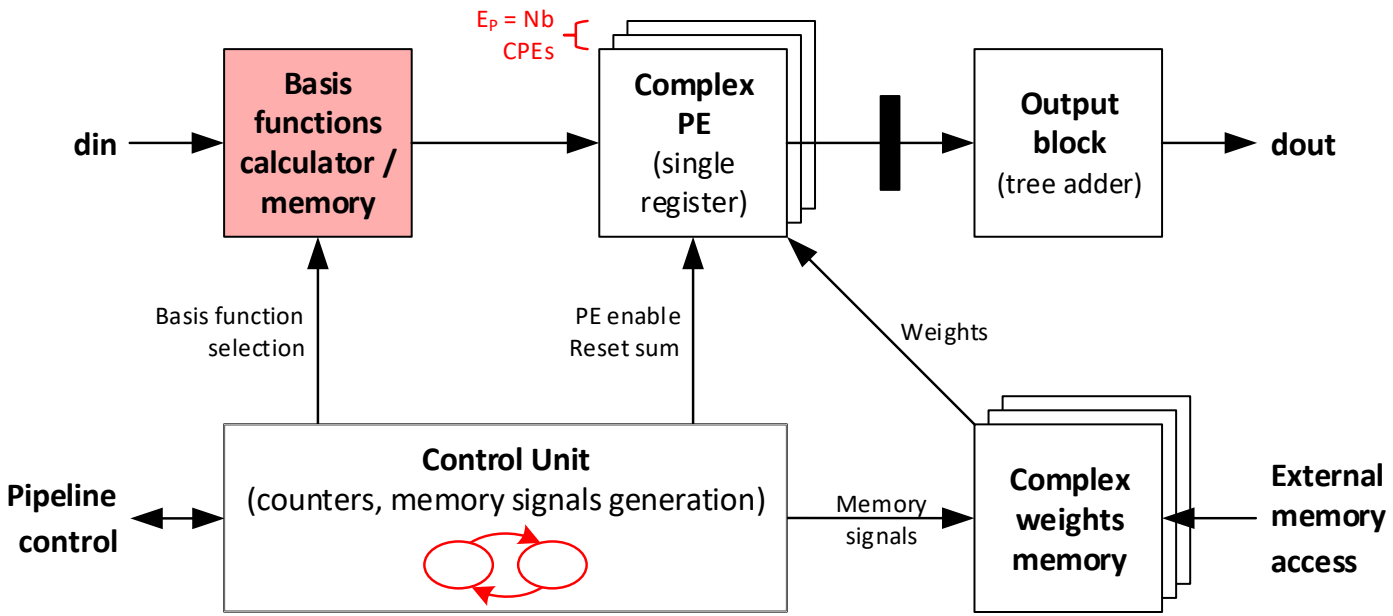
Polynomial Canceller Architecture

- Generic hardware architecture to process **multiple complex weighted sums**:



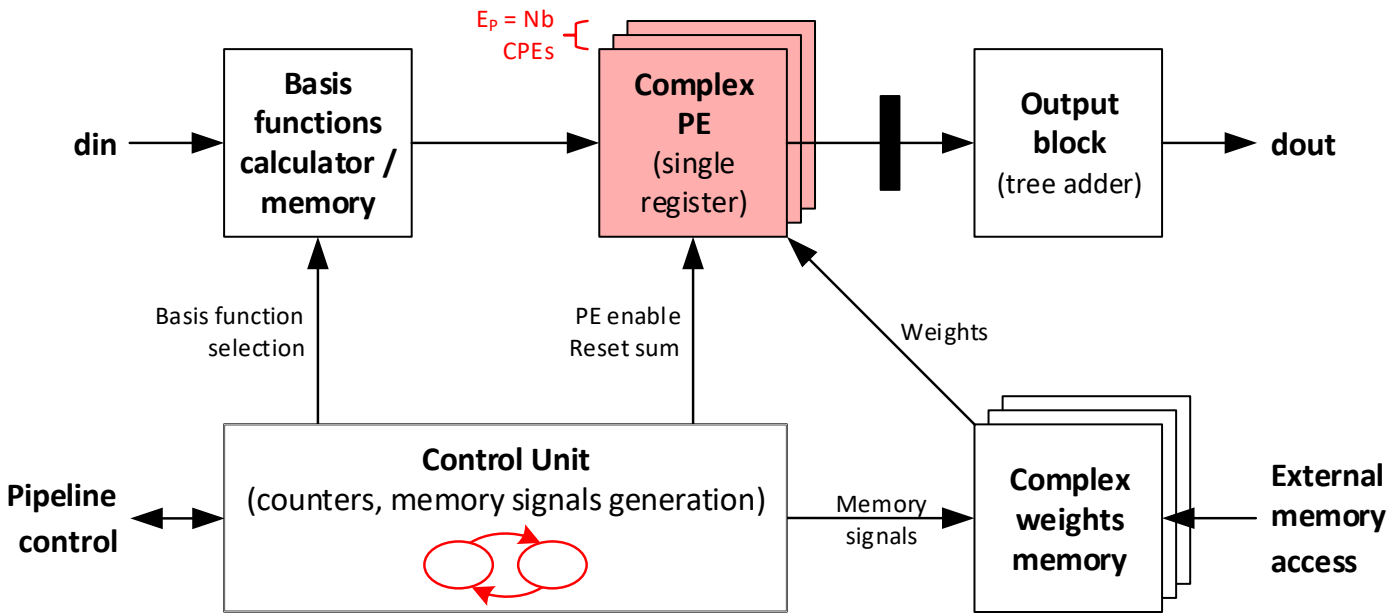
Polynomial Canceller Architecture

- Generic hardware architecture to process **multiple complex weighted sums**:
  1. **BF unit:** computes BF and redirects to correct complex processing elements (CPEs).



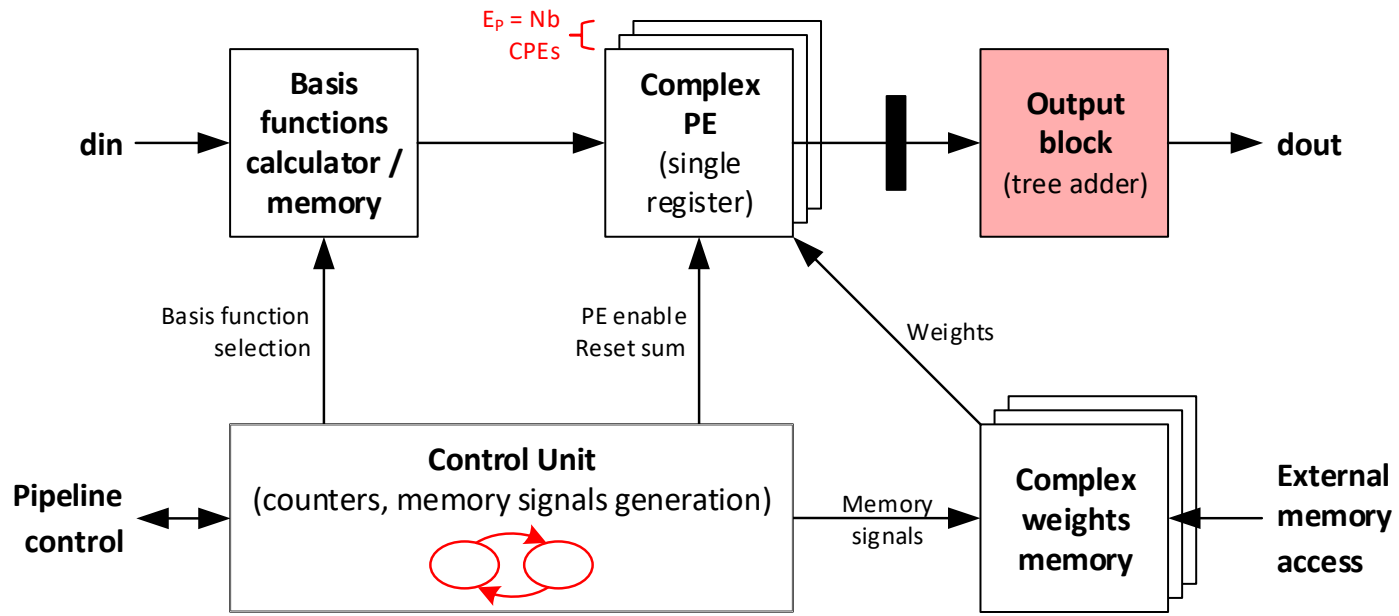
Polynomial Canceller Architecture

- Generic hardware architecture to process **multiple complex weighted sums**:
  1. **BF unit:** computes BF and redirects to correct complex processing elements (CPEs).
  2. **CPE:** complex multiply-and-accumulate unit.



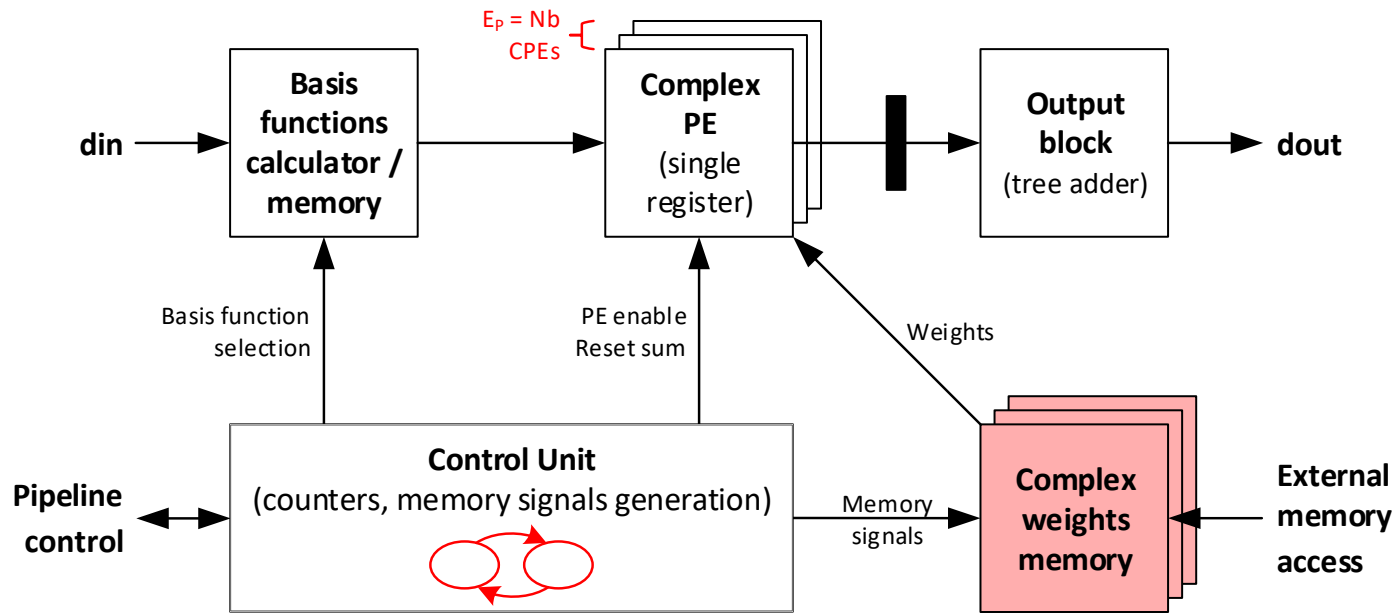
Polynomial Canceller Architecture

- Generic hardware architecture to process **multiple complex weighted sums**:
  1. **BF unit:** computes BF and redirects to correct complex processing elements (CPEs).
  2. **CPE:** complex multiply-and-accumulate unit.
  3. **Output block:** partial sum addition.



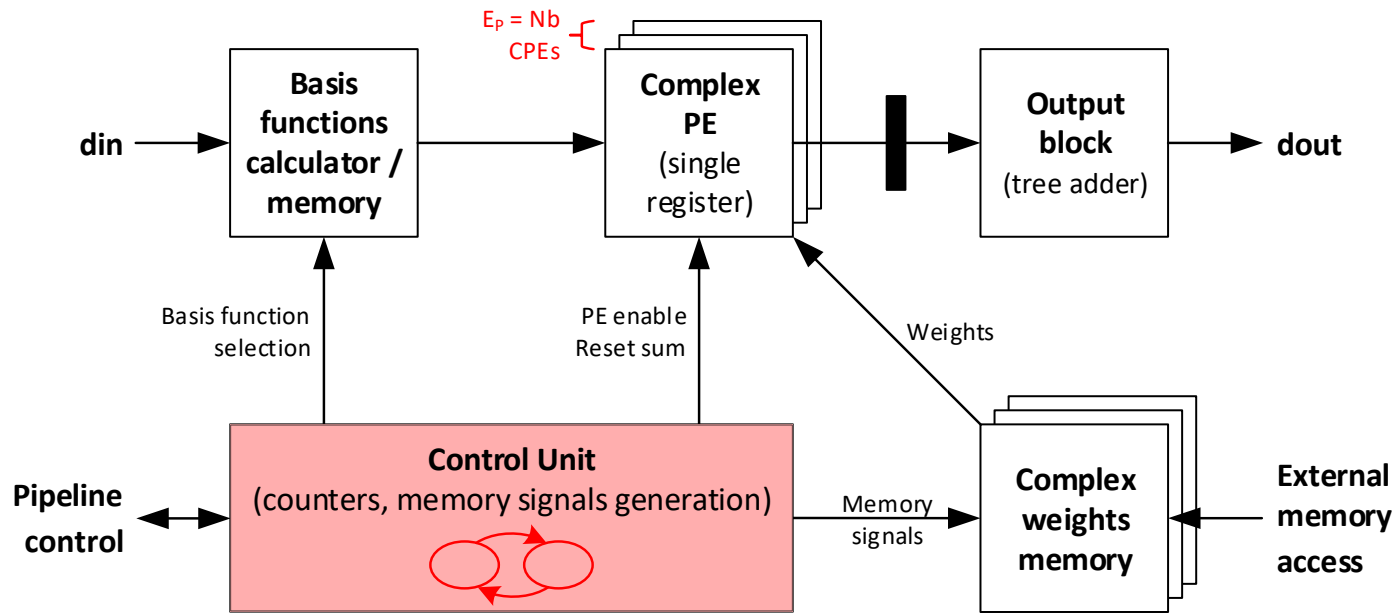
Polynomial Canceller Architecture

- Generic hardware architecture to process **multiple complex weighted sums**:
  1. **BF unit:** computes BF and redirects to correct complex processing elements (CPEs).
  2. **CPE:** complex multiply-and-accumulate unit.
  3. **Output block:** partial sum addition.
  4. **Memory:** stores complex weights $h_{p,q}$ and can be written externally.



Polynomial Canceller Architecture

- Generic hardware architecture to process **multiple complex weighted sums**:
  1. **BF unit:** computes BF and redirects to correct complex processing elements (CPEs).
  2. **CPE:** complex multiply-and-accumulate unit.
  3. **Output block:** partial sum addition.
  4. **Memory:** stores complex weights $h_{p,q}$ and can be written externally.
  5. **Control unit:** ensures correct operation for a wide range of parameters.



Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3	$x^0(x^*)^3$	$x^1(x^*)^2$	$x^2(x^*)^1$	$x^3(x^*)^0$		
5	$x^0(x^*)^5$	$x^1(x^*)^4$	$x^2(x^*)^3$	$x^3(x^*)^2$	$x^4(x^*)^1$	$x^5(x^*)^0$

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3	$x^0(x^*)^3$	$x^1(x^*)^2$	$x^2(x^*)^1$	$x^3(x^*)^0$		
5	$x^0(x^*)^5$	$x^1(x^*)^4$	$x^2(x^*)^3$	$x^3(x^*)^2$	$x^4(x^*)^1$	$x^5(x^*)^0$

- Several properties can be exploited to **simplify the computations**:

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3	$x^0(x^*)^3$	$x^1(x^*)^2$	$x^2(x^*)^1$	$x^3(x^*)^0$		
5	$x^0(x^*)^5$	$x^1(x^*)^4$	$x^2(x^*)^3$	$x^3(x^*)^2$	$x^4(x^*)^1$	$x^5(x^*)^0$

- Several properties can be exploited to **simplify the computations**:
  1. We have $BF_{p,q} = (BF_{p,p-q})^*$, so **half of the computations** can be skipped.
  2. We have $BF_{p+2,q} = x^2 BF_{p,q-2}$, so **BFs can be computed iteratively**.

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3						
5						

- Several properties can be exploited to **simplify the computations**:
  1. We have $BF_{p,q} = (BF_{p,p-q})^*$, so **half of the computations** can be skipped.
  2. We have $BF_{p+2,q} = x^2 BF_{p,q-2}$, so **BFs can be computed iteratively**.

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3			$x^2(x^*)^1$	$x^3(x^*)^0$		
5						

- Several properties can be exploited to **simplify the computations**:
  1. We have $BF_{p,q} = (BF_{p,p-q})^*$, so **half of the computations** can be skipped.
  2. We have $BF_{p+2,q} = x^2 BF_{p,q-2}$, so **BFs can be computed iteratively**.

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3	$x^0(x^*)^3$	$x^1(x^*)^2$	$x^2(x^*)^1$	$x^3(x^*)^0$		
5						

- Several properties can be exploited to **simplify the computations**:
  1. We have $BF_{p,q} = (BF_{p,p-q})^*$, so **half of the computations** can be skipped.
  2. We have $BF_{p+2,q} = x^2 BF_{p,q-2}$, so **BFs can be computed iteratively**.

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3	$x^0(x^*)^3$	$x^1(x^*)^2$	$x^2(x^*)^1$	$x^3(x^*)^0$		
5				$x^3(x^*)^2$	$x^4(x^*)^1$	$x^5(x^*)^0$

- Several properties can be exploited to **simplify the computations**:
  1. We have $BF_{p,q} = (BF_{p,p-q})^*$, so **half of the computations** can be skipped.
  2. We have $BF_{p+2,q} = x^2 BF_{p,q-2}$, so **BFs can be computed iteratively**.

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3	$x^0(x^*)^3$	$x^1(x^*)^2$	$x^2(x^*)^1$	$x^3(x^*)^0$		
5	$x^0(x^*)^5$	$x^1(x^*)^4$	$x^2(x^*)^3$	$x^3(x^*)^2$	$x^4(x^*)^1$	$x^5(x^*)^0$

- Several properties can be exploited to **simplify the computations**:
  1. We have $BF_{p,q} = (BF_{p,p-q})^*$, so **half of the computations** can be skipped.
  2. We have $BF_{p+2,q} = x^2 BF_{p,q-2}$, so **BFs can be computed iteratively**.

Basis Function Calculator Optimization

- Recall that basis functions are of the form:

$$BF_{p,q}(x) \triangleq x^q (x^*)^{p-q}$$

- Need to be computed for all $p = 1, 3, \ldots, P$ and $q = 1, 2, \ldots, p$ combinations.

$p \backslash q$	0	1	2	3	4	5
1	$x^0(x^*)^1$	$x^1(x^*)^0$				
3	$x^0(x^*)^3$	$x^1(x^*)^2$	$x^2(x^*)^1$	$x^3(x^*)^0$		
5	$x^0(x^*)^5$	$x^1(x^*)^4$	$x^2(x^*)^3$	$x^3(x^*)^2$	$x^4(x^*)^1$	$x^5(x^*)^0$

- Several properties can be exploited to **simplify the computations**:
  1. We have $BF_{p,q} = (BF_{p,p-q})^*$, so **half of the computations** can be skipped.
  2. We have $BF_{p+2,q} = x^2 BF_{p,q-2}$, so **BFs can be computed iteratively**.

- **Hardware optimizations:**
  - Reuse of BFs for previous time-steps with circular memory.
  - Depending on power p, shift BF results to minimize the required bit-width.

Results: SI Cancellation Performance and Complexity

- Floating-point Keras model for performance evaluation and **offline training**
- **Dataset:** measured 10 MHz QPSK-modulated OFDM signal, sampled at 20 MHz
- **Performance evaluation:** $C_{\text{dB}} = 10 \log_{10} \left(\frac{\sum_n |y_{cancelled}(n)|^2}{\sum_n |y(n)|^2} \right)$

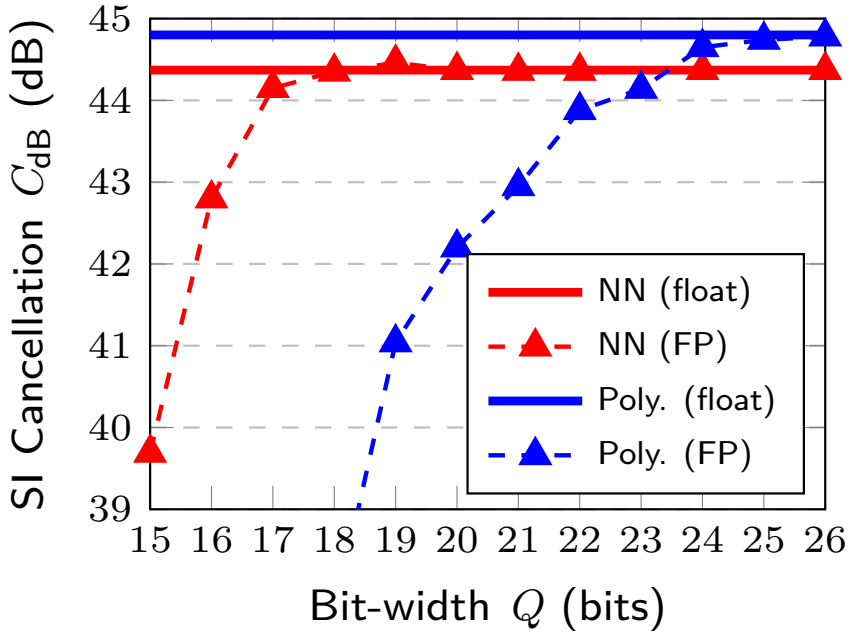
Results: SI Cancellation Performance and Complexity

- Floating-point Keras model for performance evaluation and **offline training**
- **Dataset:** measured 10 MHz QPSK-modulated OFDM signal, sampled at 20 MHz
- **Performance evaluation:** $C_{\text{dB}} = 10 \log_{10} \left(\frac{\sum_n |y_{cancelled}(n)|^2}{\sum_n |y(n)|^2} \right)$

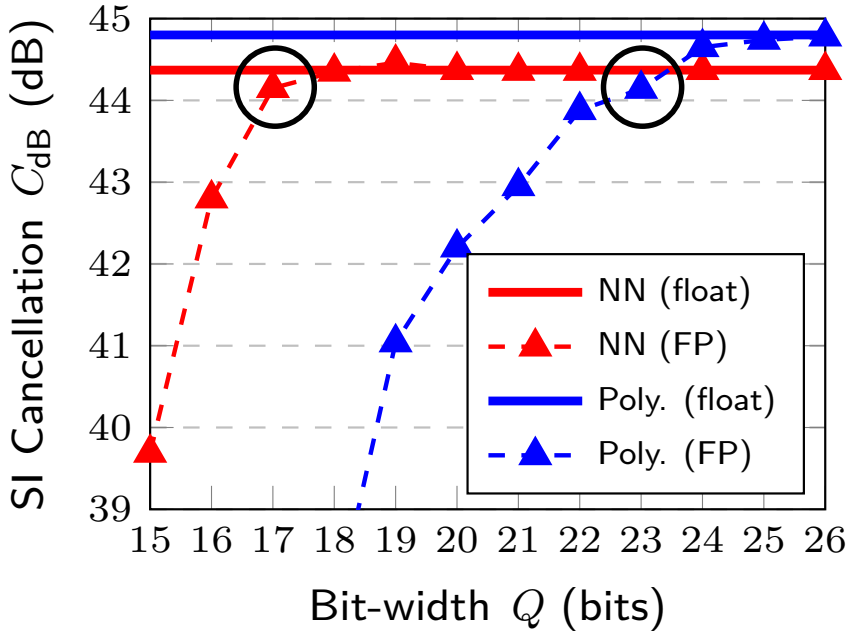
	# param.	C_{dB} (dB)	# real mult.[a]	# real add[b]
Poly. $(L = 13,\ P = 7)$	520	-44.80	780	1818
NN $(L = 13,\ N_h = 18)$	550	-44.37	543	593
	$+5\%$	**$+0.4$ dB**	**-30%**	**-67%**

[a] 1 complex mult. $\equiv$ 3 real mult. and 5 real add
[b] 1 complex add $\equiv$ 2 real add

Results: Quantization Bit-Width and Hardware Multipliers



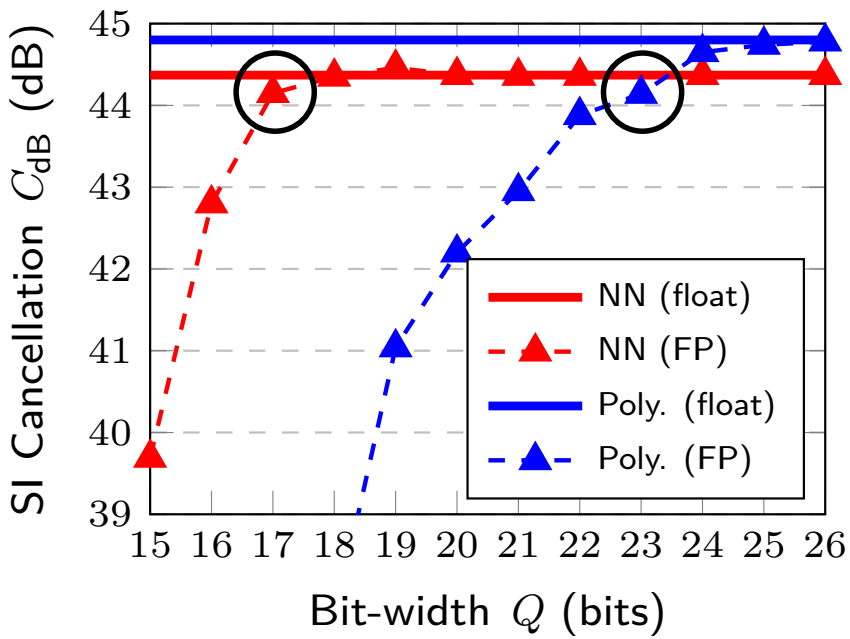
Results: Quantization Bit-Width and Hardware Multipliers

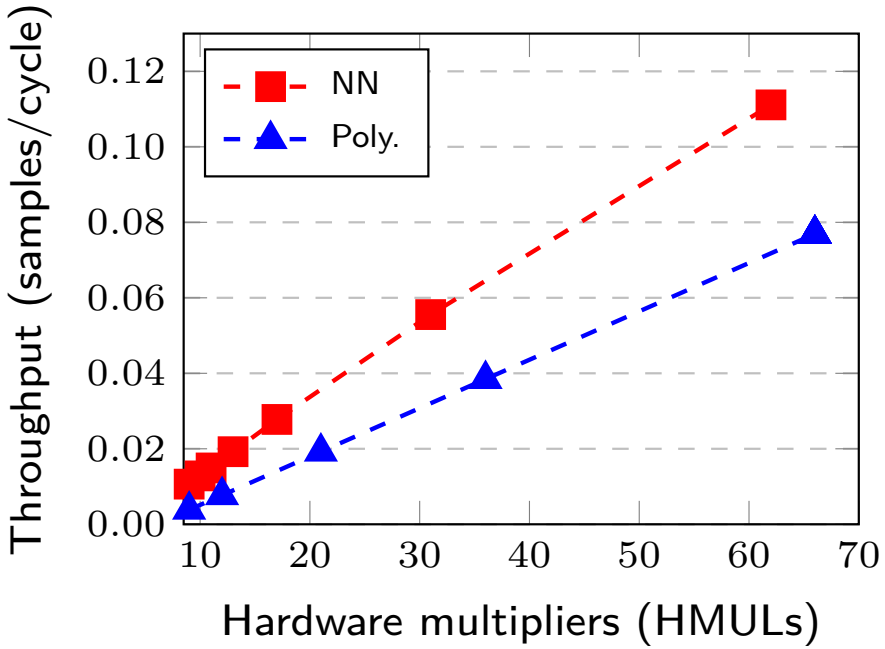


Bit-width

NN with $Q = 17$ provides **identical performance** as polynomial with $Q = 23$

Results: Quantization Bit-Width and Hardware Multipliers

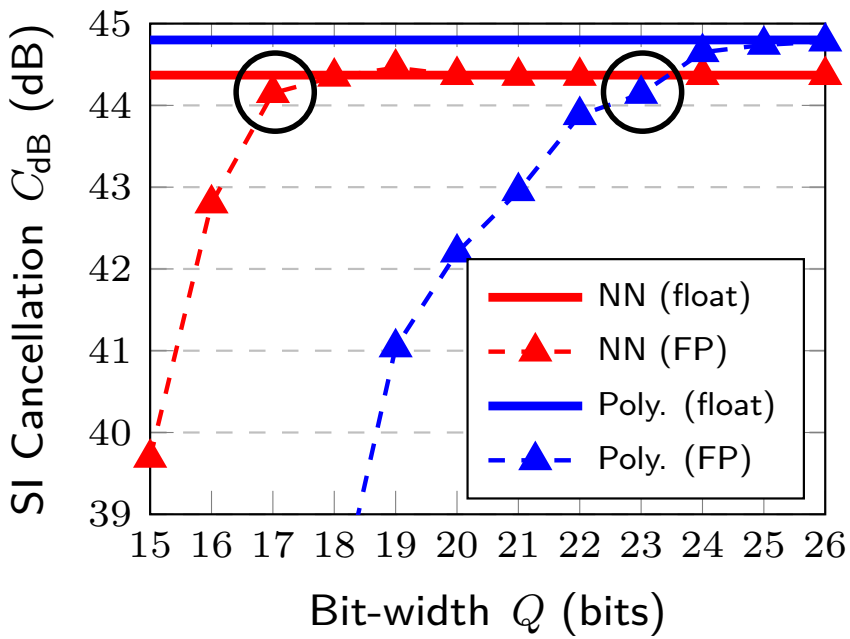


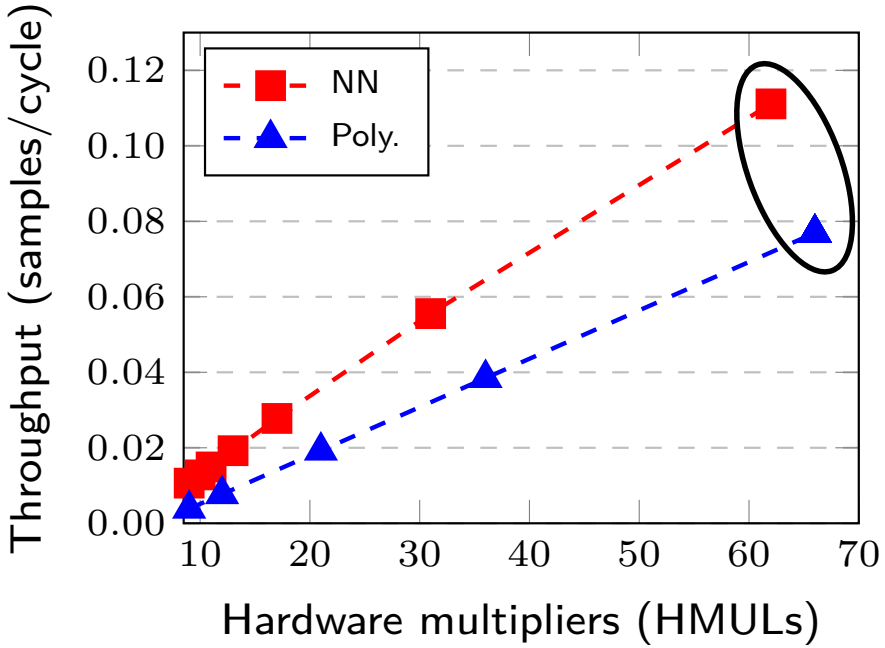


Bit-width

NN with $Q = 17$ provides **identical performance** as polynomial with $Q = 23$

Results: Quantization Bit-Width and Hardware Multipliers





Bit-width

NN with $Q = 17$ provides **identical performance** as polynomial with $Q = 23$

HMULs

At 0.1 samples/cycle, the canceller operates at a **reasonable** $10\times$ sampling frequency.

Results: ASIC Implementation

- 28nm FD-SOI technology
- **Worst case:** slow corners, at 0.7 V and 125 °C
- Fully **placed-and-routed** design

Results: ASIC Implementation

- 28nm FD-SOI technology
- **Worst case:** slow corners, at 0.7 V and 125 °C
- Fully **placed-and-routed** design

	NN	Poly.	
Area (mm^2)	0.32	0.36	$\mathbf{-11\%}$
Max. freq. (MHz)	250	226	$\mathbf{+10\%}$
Throughput (MS/s)	27.8	17.4	$\mathbf{+60\%}$

Results: ASIC Implementation

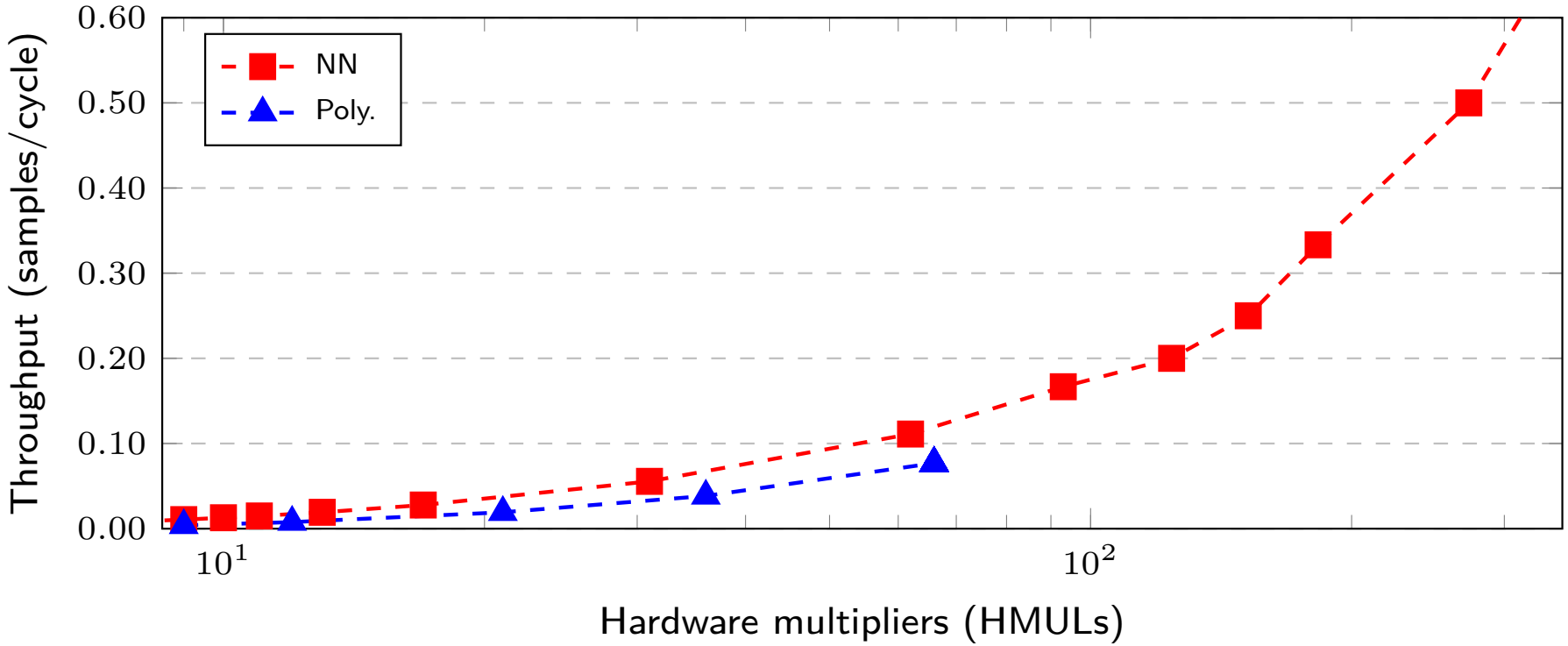
- 28nm FD-SOI technology
- **Worst case:** slow corners, at 0.7 V and 125 °C
- Fully **placed-and-routed** design

	NN	Poly.	
Area (mm^2)	0.32	0.36	$\mathbf{-11\%}$
Max. freq. (MHz)	250	226	$\mathbf{+10\%}$
Throughput (MS/s)	27.8	17.4	$\mathbf{+60\%}$

Conclusion

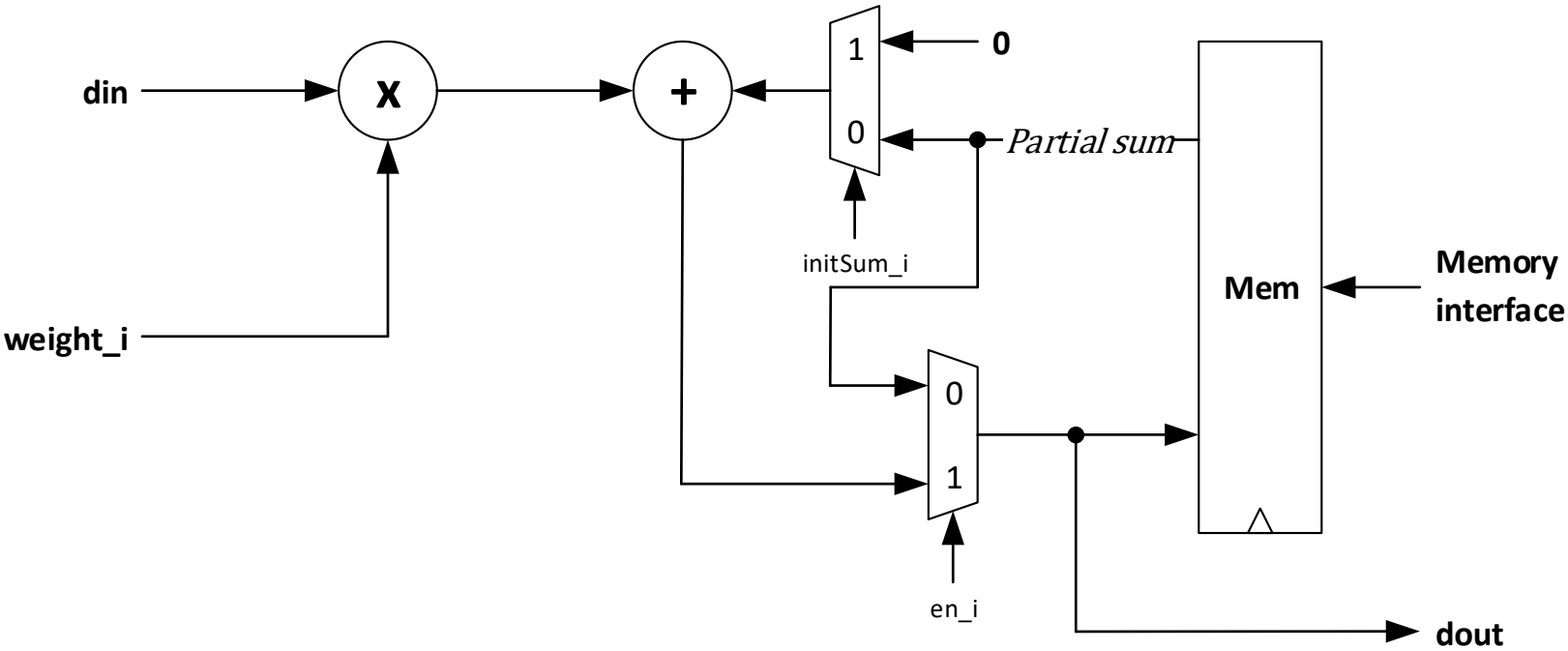
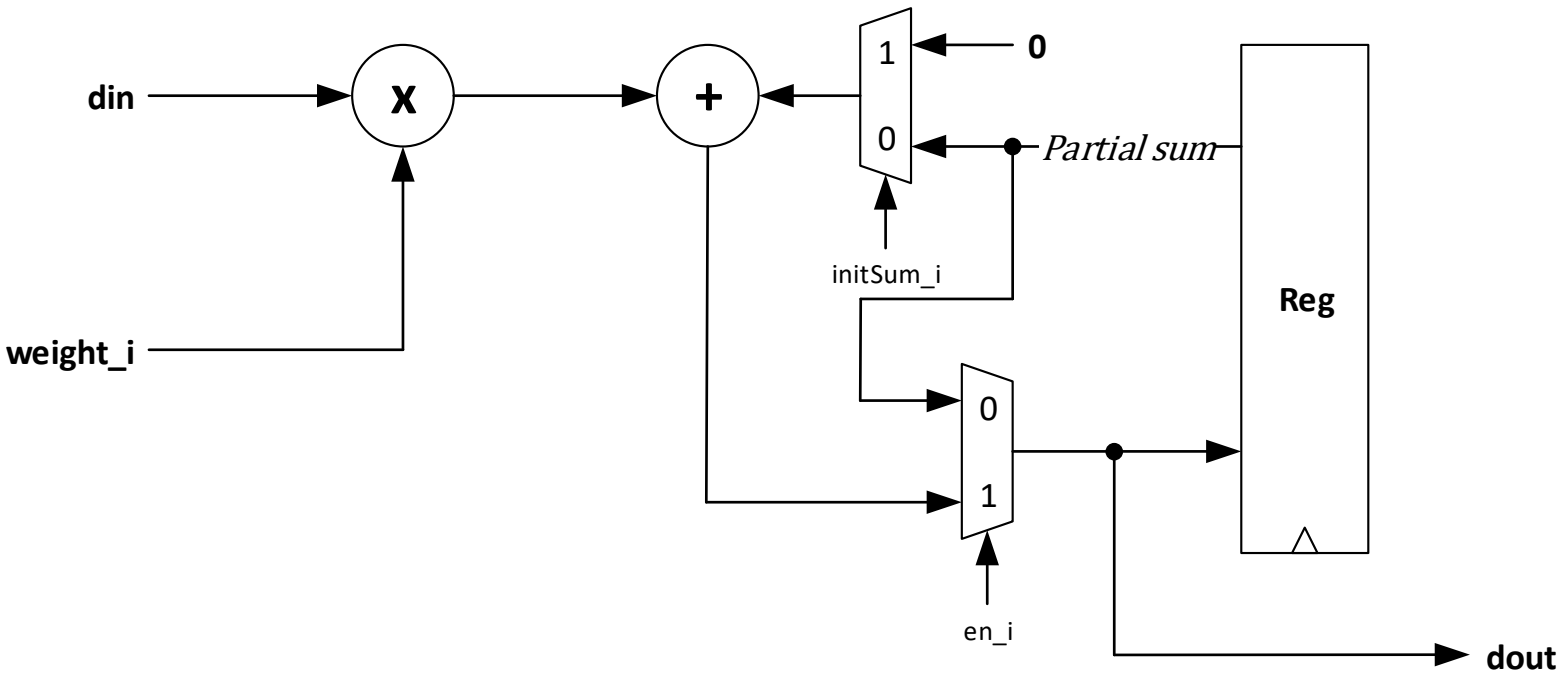
The NN-augmented canceller has 81% better area efficiency!

Maximum throughput of neural network design



FPGA implementation	62 HMUL	183 HMUL	543 HMUL
Max. freq. (MHz)	88.0	78.0 (**−11%**)	86.4 (**−2%**)
Throughput (MS/s)	9.8	26.0 (**+165%**)	86.4 (**+782%**)
Resource usage	1.0	3.0	8.8

Neural network processing element



Polynomial processing element