HW Solutions

A1. (C) Multi-prover Interactive proofs (MIPs) can prove languages in NEXPTIME (Lecture 1, Slide 78), which is beyond the capabilities of Interactive Proofs (IPs).

A2. (A) zkRollups rely on the succinctness property of zkSNARKs, not the zero-knowledge property.

A3. (B) The Plonk protocol uses KZG polynomial commitment, which relies on Bilinear pairings.

A4. (D) Groth16 based on linear PCP does not rely on Fiat-Shamir.

A5. (C) Brakedown (Lecture 7, Slide 32) showed that you can construct a polynomial commitment scheme based on error-correcting codes that do not have an efficient decoding algorithm.

A6. (C) Given an ℓ-variate (multilinear) polynomial, the size of the polynomial, i.e., the number of monomials in the polynomial, is $O(2^\ell)$. For the same polynomial, the proof size of the sumcheck protocol is $O(\ell)$.

A7. (D) Plonk-IOP can be combined with any (univariate) polynomial commitment scheme to construct a ZKP scheme.

A8. (C) The recursion overhead in IVC from folding scheme is proportional to the folding verifier computation, as opposed to the larger zkSNARK verifier computation in the case of IVC from succinct verification.

A9. (B) Sumcheck IOP has a linear prover (as opposed to quasi-linear for Plonk IOP) and Orion PC has the fastest prover as it is linear in polynomial size (unlike FRI) and it does not require group exponentiations (unlike Bulletproofs and KZG).

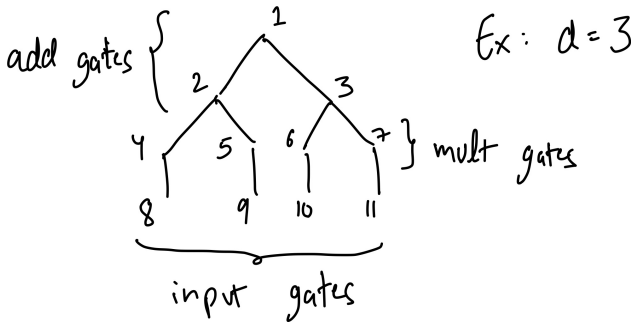
A10. (C) Plonk IOP has a constant proof size (as opposed to logarithmic for Sumcheck IOP) and KZG is the only polynomial commitment among the given options that has a constant proof size.

A11.
(a)

$$\widetilde{\mathsf{eq}}_\ell(a,b) = \prod_{i=1}^{\ell}(1 - a_i - b_i + 2a_ib_i),$$

where a_i, b_i denote the i-th bit of a and b, respectively. It's easy to see that this polynomial can be evaluated in time $O(\ell)$: it is a product of ℓ terms, each of which takes $O(1)$ time to evaluate.

Before we get into parts (b) and (c), we first define how the computation trace T of size S is encoded by the polynomial h (Lecture 4, Slide 80). The indexing in our encoding scheme starts from the output gate and proceeds to the input gates. Given this scheme, note that for the simple circuit under consideration, the indices $[1, 2^{d-1})$, $[2^{d-1}, 2^d)$, and $[2^d, 3 \cdot 2^{d-1})$ correspond to the add gates, mult gates, and input gates, respectively.



We also define a generalization of the equality predicate to 3 inputs, which will be useful in the following parts:

$$\widetilde{\mathsf{eq3}}(a,b,c) = 1 - a - b - c + ab + bc + ac$$

(b) The mult predicate $\mathsf{mult} : \{0,1\}^{d+1} \times \{0,1\}^{d+1} \times \{0,1\}^{d+1} \to \{0,1\}$ is defined as follows:

$$\mathsf{mult}(a,b,c) = \begin{cases} 1 & \text{if } a + 2^{d-1} = b = c \\ & \quad \wedge\ a \in [2^{d-1}, 2^d) \wedge\ \ b, c \in [2^d, 3 \cdot 2^{d-1}) \\ 0 & \text{otherwise} \end{cases}$$

Note that the condition for this predicate has arithmetic operations and range checks, while the extension operates bitwise on the inputs of a predicate. Thus, we rewrite the condition for the if-clause as follows so that it only uses bitwise operations (bit at index 1 corresponds to the least significant bit):

$$\forall i \in [1, d-1],\ a_i = b_i = c_i\ \wedge\ (a_d, a_{d+1}, b_d, b_{d+1}, c_d, c_{d+1}) = (1,0,0,1,0,1)$$

Now, it is easy to write the multilinear extension $\widetilde{\mathsf{mult}} : \mathbb{F}^{d+1} \times \mathbb{F}^{d+1} \times \mathbb{F}^{d+1} \to \mathbb{F}$:

$$\widetilde{\mathsf{mult}}(a,b,c) = \prod_{i=1}^{d-1} \widetilde{\mathsf{eq3}}(a_i, b_i, c_i)\ \cdot\ \widetilde{\mathsf{eq}}_6\big((a_d, a_{d+1}, b_d, b_{d+1}, c_d, c_{d+1}), (1,0,0,1,0,1)\big)$$

It is easy to see that $\widetilde{\mathsf{mult}}$ is multilinear, i.e., it is degree 1 in each variable.

(c) The add predicate $\mathsf{add} : \{0,1\}^{d+1} \times \{0,1\}^{d+1} \times \{0,1\}^{d+1} \to \{0,1\}$ is defined as follows:

$$\mathsf{add}(a,b,c) = \begin{cases} 1 & \text{if } 2a = b \ \wedge \ 2a+1 = c \\ & \quad \wedge \ a \in [1, 2^{d-1}) \ \wedge \ b, c \in [1, 2^d) \\ 0 & \text{otherwise} \end{cases}$$

We can rewrite the condition for the if-clause as follows so that it only uses bitwise operations[1]:

$$\forall i \in [1, d-1], \ a_i = b_{i+1} = c_{i+1} \ \wedge \ (a_d, a_{d+1}, b_1, b_{d+1}, c_1, c_{d+1}) = (0,0,0,0,1,0)$$

Now, it is easy to write its multilinear extension $\widetilde{\mathsf{add}} : \mathbb{F}^{d+1} \times \mathbb{F}^{d+1} \times \mathbb{F}^{d+1} \to \mathbb{F}$:

$$\widetilde{\mathsf{add}}(a,b,c) = \prod_{i=1}^{d-1} \widetilde{\mathsf{eq3}}(a_i, b_{i+1}, c_{i+1}) \ \cdot \ \widetilde{\mathsf{eq}}_6\big((a_d, a_{d+1}, b_1, b_{d+1}, c_1, c_{d+1}), (0,0,0,0,1,0)\big)$$

Note that $\widetilde{\mathsf{add}}$ is multilinear, i.e., it is degree 1 in each variable.

[1] Although this expression also outputs 1 if $(a,b,c) = (0,0,1)$, it is not a problem as we can instantiate the protocol such that the summation on $a, b, c \in \{0,1\}^{\log S}$ excludes this tuple of values. One way to do this efficiently is to check that $\sum_{a,b,c \in \{0,1\}^{\log S}} g_h(a,b,c)^2$ is equal to $g_h(0,0,1)^2$ as opposed to 0 in Slide 94.

A12.
(a) To support the custom gate g with 3 inputs, we extend the computation trace to include an additional column for the third input to each gate. The add/mult gates simply ignore this third input. Accordingly, we have the following:

- $d = 4(m+n) + \ell$
- Set $\Omega = \{1, \omega, \ldots, \omega^{d-1}\}$ of size d.
- Set $\Omega_{\mathsf{inp}} = \{\omega^{-1}, \ldots, \omega^{-\ell}\}$ of size ℓ.
- Set $\Omega_{\mathsf{gates}} = \{1, \omega^4, \ldots, \omega^{4(m+n-1)}\}$ of size $m+n$.
- Trace polynomial T has degree d and is defined as follows:
  - $\forall j \in [1, \ell],\ T(\omega^{-j}) = \text{input } \#j.$
  - $\forall i \in [0, m+n)$:
    - $T(\omega^{4i}) = \text{first input to gate } \#i$
    - $T(\omega^{4i+1}) = \text{second input to gate } \#i$
    - $T(\omega^{4i+2}) = \text{third input to gate } \#i$
    - $T(\omega^{4i+3}) = \text{output of gate } \#i$
- Selector polynomial S has degree $m+n$ and is defined as follows:

$$\forall i \in [0, m+n),\ S(\omega^{4i}) = \begin{cases} 1 & \text{if gate } \#i \text{ is an add gate} \\ 0 & \text{if gate } \#i \text{ is a mult gate} \\ -1 & \text{if gate } \#i \text{ is a } g \text{ gate} \end{cases}$$

(b)

- Gate check:

$$\begin{aligned} \forall y \in \Omega_{\mathsf{gates}},\ & \frac{S(y)(1+S(y))}{2} \cdot [T(y) + T(\omega y)] \\ & + (1 - S(y))(1 + S(y)) \cdot [T(y) \cdot T(\omega y)] \\ & - \frac{S(y)(1-S(y))}{2} \cdot [3 \cdot T(y)^4 + 7 \cdot T(y)^2 T(\omega y) T(\omega^2 y) - 2 \cdot T(\omega y) T(\omega^2 y)^2] \\ & - T(\omega^3 y) = 0 \end{aligned}$$

  - Gadget: ZeroTest
  - Degree of polynomial: $4\deg(T) + 2\deg(S) = 4d + 2(m+n)$
  - Size of set: $m+n$

- Input Check:
$$\forall y \in \Omega_{\mathsf{inp}},\ T(y) - v(y) = 0$$
  - Gadget: ZeroTest
  - Degree of polynomial: $\deg(T) = d$
  - Size of set: ℓ
- Wiring Check:
$$\forall y \in \Omega,\ T(y) - T(W(y)) = 0$$
  - Gadget: Prescribed Permutation Check
  - Degree of polynomials: $\deg(T) = d$, $\deg(W) = d$
  - Size of set: d
- Output Check:
$$T(\omega^{4(m+n)-1}) = 0$$
  - Evaluation proof on polynomial T of degree d

(c) First, we make the following observations:

- All group exponentiations are incurred by the KZG polynomial commitment.
- For a degree d polynomial, KZG commit requires d group exponentiations.
- As per the assumption in the problem statement, for a degree d polynomial, a KZG batched evaluation proof requires d group exponentiations irrespective of the number of points being evaluated.

Now, all we need to do is find the distinct polynomials that are committed to and evaluated throughout this protocol, along with the degree of each polynomial:

- Trace polynomial T: commit and batched eval
  - Degree: d
  - GExps: $2d$
- Selector polynomial S: only eval (committed during preprocessing)
  - Degree: $m + n$
  - GExps: $m + n$
- Wiring polynomial W: only eval (committed during preprocessing)
  - Degree: d
  - GExps: d

- ZeroTest for Gate Check
  - Quotient polynomial q_{gates}: commit and eval
    - Degree: $4\deg(T) + 2\deg(S) - |\Omega_{\mathsf{gates}}| = 4d + (m+n)$
    - GExps: $8d + 2(m+n)$
- ZeroTest for Input Check
  - Quotient polynomial q_{inp}: commit and eval
    - Degree: $\deg(T) - |\Omega_{\mathsf{inp}}| = d - \ell$
    - GExps: $2d - 2\ell$
- Prescribed Permutation Check for Wiring Check
  - Reduces to a Product Check on polynomial f_{wire} of degree $\mathsf{max}(\deg(T), \deg(W)) = d$ and set size $|\Omega| = d$.
  - Polynomial t_{wire}: commit and batched eval
    - Degree: $|\Omega| = d$
    - GExps: $2d$
  - Quotient polynomial q_{wire}: commit and eval
    - Degree: $\deg(f_{\mathsf{wire}}) + \deg(t_{\mathsf{wire}}) - |\Omega| = d$
    - GExps: $2d$
- Total GExps: $17d + 3(m+n) - 2\ell = 71(m+n) + 15\ell$