Παράλληλοι Αλγόριθμοι

Μηχανή Διαφορών Babbage

Α.Ο. Φαρμάκης

Περιεχόμενα της παρουσίασης

Ο αλγόριθμος (1/8)

Επινοήθηκε από το Charles Babbage κατά τον 19ο αιώνα, στη προσπάθειά του να μηχανοποιήσει πολύπλοκους μαθηματικούς υπολογισμούς, κυρίως υψηλής τάξης πολυωνύμων για χρήση σε αστρολογικούς πίνακες.

Βασίστηκε στην αρχή των πεπερασμένων διαφορών του Νεύτωνα.

Ο αλγόριθμος (2/8)

Με τη τεχνολογία που χρησιμοποιείται αυτή τη στιγμή (σειριακό πολλαπλασιασμό με γρήγορους αθροιστές, κωδικοποίηση Booth, δέντρα Wallace και λοιπά), ο πολλαπλασιασμός αποτελεί μια ιδιαίτερα ακριβή πράξη, τόσο χρονικά όσο και χωρικά.

Με τη χρήση όμως της αναδρομής, αποφεύγεται η χρήση του πολλαπλασιασμού, που σε υψηλής τάξης πολυώνυμα αποτελεί μεγάλο πρόβλημα.

Ο αλγόριθμος (3/8)

Έστω η περίπτωση ενός πολυωνύμου 5ου βαθμού που δέχεται ακέραιες εισόδους και συμβολίζεται ως
$u(n) = a \cdot n^5 + b \cdot n^4 + c \cdot n^3 + d \cdot n^2 + f \cdot n + g$

Στην απλή (γενική) περίπτωση, απαιτεί 15 πολλαπλασιασμούς και 6 αθροίσεις.

Ιδέα: Οι πολλαπλασιασμοί είναι και αυτοί διαδοχικές αθροίσεις, οπότε να γίνει μετασχηματισμός του υπολογισμού του πολυωνύμου μόνο σε πράξεις άθροισης.

Ο αλγόριθμος (4/8)

Μικρό κόστος σε κύκλους ρολογιού / πράξεων, αλλά κάθε κύκλος είναι σημαντικά μικρότερος!

Για το μετασχηματισμό, γίνεται ανάλυση σε διαδοχικές διαφορές και κάθε νέο τέτοιο πολυώνυμο προσδιορίζεται στο αμέσως επόμενο ακέραιο που ορίζεται από την εκάστοτε διαφορά, μέχρις ότου φτάσουμε σε σταθερό πολυώνυμο.

Ο αλγόριθμος (5/8)

Συσσωρεύουμε μέχρι να φτάσουμε το ζητούμενο n, οπότε ακολουθιακά έχει πολυπλοκότητα O(m·n), όπου m-1 ο βαθμός του πολυωνύμου.

Επειδή το κάθε αναλυόμενο πολυώνυμο είναι ανεξάρτητο από κάθε άλλο, μπορούμε τις συσσωρεύσεις αυτές να τις παραλληλοποιήσουμε και να έχουμε πολυπλοκότητα O(n).

Ο αλγόριθμος (6/8)

Ο βασικός σκελετός του αλγορίθμου, σε ψευδοκώδικα:

```
integer input a, b, c, d, f, g;    // Polynomial coefficients
positive integer input n;          // Variable we want to evaluate the polynomial at
integer output poly_out;           // Polynomial output for given n
integer i;                         // Recursion tracker
integer u, v, w, x, y, z;          // Piece-wise polynomial values used for recursion

parallel for (i = 0; i <= n; i++) begin
   call u = compute_u(a, b, c, d, f, g, i);
   call v = compute_v(a, b, c, d, f, i);
   call w = compute_w(a, b, c, d, i);
   call x = compute_x(a, b, c, i);
   call y = compute_y(a, b, i);
   call z = compute_z(a, i);
end

// When the parallel block is done, assign the value of u to the output poly_out
if (i = n) poly_out = u;
```

Ο αλγόριθμος (7/8)

Παραδείγματα των κλήσεων των επιμέρους πολυωνύμων:

```
function compute_u(a, b, c, d, f, g, i) begin
   if (i == 0) return g;
   else return compute_u(a, b, c, d, f, g, i-1) + compute_v(a, b, c, d, f, i);
end

function compute_v(a, b, c, d, f, i) begin
   if (i < 1) return 0;
   else if (i == 1) return (5*a + (-10*a+4*b) + (10*a-6*b+3*c) + (-5*a+4*b-3*c+2*d)
                            + (a-b+c-d+f));
   else return compute_v(a, b, c, d, f, i-1) + compute_w(a, b, c, d, i);
end

...

function compute_z(a, i) begin
   if (i < 5) return 0;
   else return 120 * a;
end
```

Ο αλγόριθμος (8/8)

Ας δούμε ένα παράδειγμα από την αρχή που δημιουργούνται τα επιμέρους πολυώνυμα μέχρι το τέλος των υπολογισμών μαζί! Έστω $f(n) = a \cdot n^3 + b \cdot n^2 + c \cdot n + d$

a = b = c = d = n =

Υπενθύμιση: ο n είναι θετικός ακέραιος, ενώ οι συντελεστές a, b, c, d είναι προσημασμένοι ακέραιοι.

Διαδικασία υλοποίησης (1/7)

Θα υλοποιήσουμε τον αλγόριθμο ως εξειδικευμένο ψηφιακό σύστημα (πύλες, καταχωρητές, αριθμητικά κυκλώματα κλπ).

Αφού κάθε ακολουθιακό στοιχείο λαμβάνει το ίδιο ρολόι, η ιδέα της παραλληλίας έρχεται φυσικά σε θέματα ψηφιακής σχεδίασης.

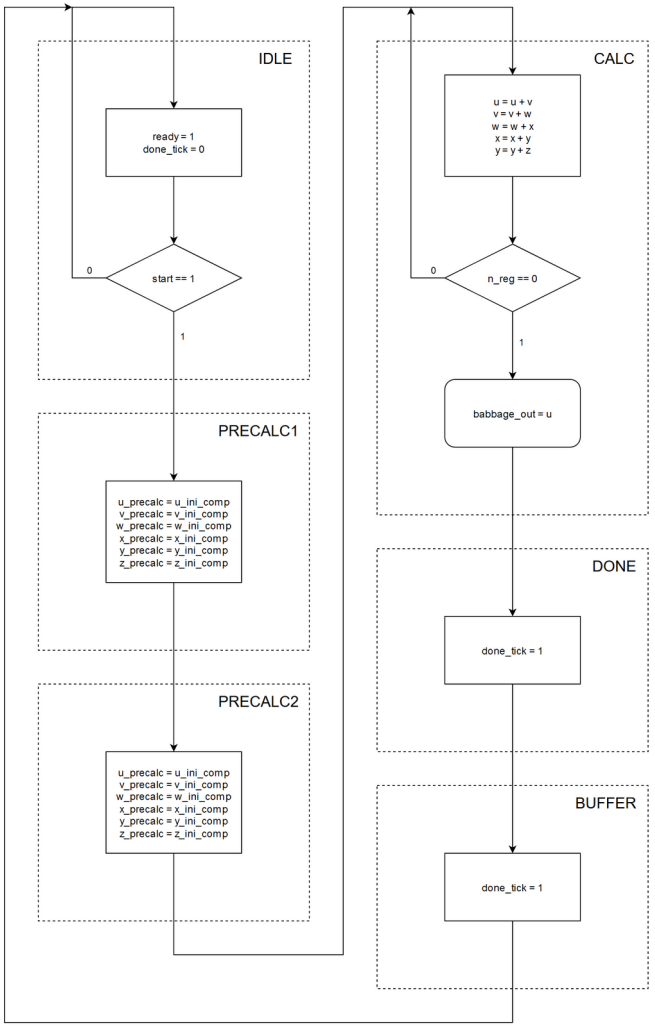
Διαδικασία υλοποίησης (2/7)

Η τεχνική σχεδίασης θα είναι των FSMD, δηλαδή FSM με διαδρομή δεδομένων (datapath). Το datapath υλοποιεί τις πράξεις του αλγορίθμου, ενώ το FSM ελέγχει πότε κάθε πράξη εκτελείται.

Για τη πιο κατανοητή και λιγότερο επιρρεπή σε λάθη περιγραφή, θα χρησιμοποιηθεί διάγραμμα Αλγοριθμικής Μηχανής Καταστάσεων (Algorithmic State Machine, ASM).

Διαδικασία υλοποίησης (3/7)

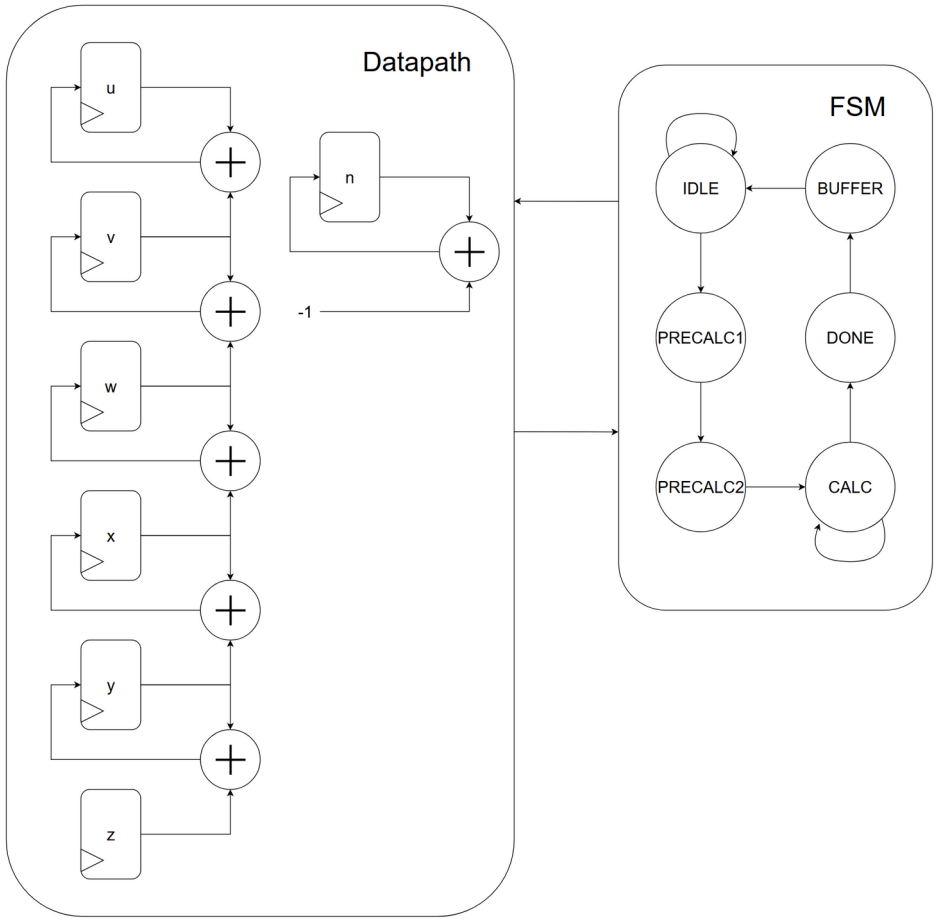
- IDLE: Κατάσταση αναμονής
- PRECALC 1/2: Προϋπολογισμός των επιμέρους πολυωνύμων
- CALC: Συσσωρεύσεις και έλεγχοι τερματισμού
- DONE: Σηματοδότηση τερματισμού
- BUFFER: Επέκταση της σηματοδότησης τερματισμού



Διαδικασία υλοποίησης (4/7)

Απλοποιημένο διάγραμμα του FSM και του αντίστοιχου datapath.

Παραλείπονται οι μονάδες ελέγχου τερματισμού, τα κυκλώματα παραγωγής αρχικών τιμών και οι συνθήκες μετάβασης των καταστάσεων του αυτομάτου.



Διαδικασία υλοποίησης (5/7)

Για τη συγκεκριμένη σχεδίαση, επιλέχθηκαν τα εξής μεγέθη των εισόδων a, b, c, d, f, g, n:

a: 2-bit προσημασμένος ακέραιος → $a \in [-2, 1]$
b: 3-bit προσημασμένος ακέραιος → $b \in [-4, 3]$
c: 4-bit προσημασμένος ακέραιος → $c \in [-8, 7]$
d: 4-bit προσημασμένος ακέραιος → $d \in [-8, 7]$
f: 6-bit προσημασμένος ακέραιος → $f \in [-32, 31]$
g: 10-bit προσημασμένος ακέραιος → $g \in [-512, 511]$
n: 7-bit μη προσημασμένος ακέραιος → $n \in [0, 127]$

Διαδικασία υλοποίησης (6/7)

Ο λόγος επιλογής των προηγουμένων μεγεθών των εισόδων έγινε ώστε το αποτέλεσμα να “χωράει” σε 32-bit ακέραιο. Επιτρέπει έτσι την εύκολη ενσωμάτωση σε άλλες συσκευές ως ένα είδους “συν-επεξεργαστή” με βάσει προϋπάρχοντα standard.

Τα μεγέθη αυτά μπορούν εύκολα να αλλάξουν αν χρειαστεί, αλλά θα πρέπει να προσαρμοστεί και το μέγεθος της εξόδου με αντίστοιχο τρόπο.

Διαδικασία υλοποίησης (7/7)

Μερικές μετρικές της υλοποίησης αυτής:

$$\text{Speed-Up} = \frac{m \cdot (n+1)}{n+3} \simeq m \text{, όσο n↑.}$$

$$W = (n+3) \cdot m.$$

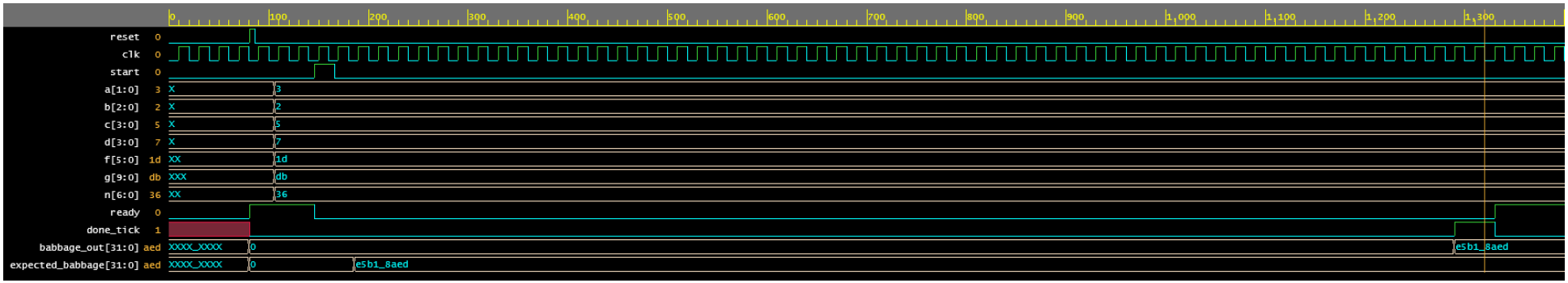
$$E = \frac{m \cdot (n+1)}{(n+3) \cdot m} = \frac{n+1}{n+3} \simeq 1 \text{, όσο n↑.}$$

όπου m οι “επεξεργαστές” (συσσωρευτές), m·(n+1) το πλήθος βημάτων του σειριακού αλγορίθμου και n+3 του Babbage.

Παράδειγμα λειτουργίας (1/4)

Παράδειγμα υπολογισμού πολυωνύμου 5ου βαθμού με εισόδους a = 1, b = 2, c = 5, d = 7, f = 29, g = 219 και n = 54.

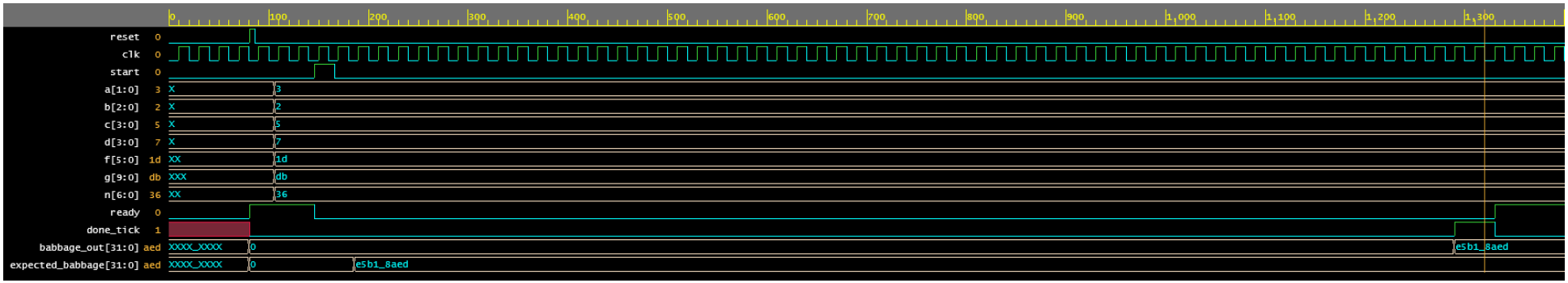
Παρατηρούμε θέλουμε n + 1 = 55 κύκλους για να υπολογιστεί το αποτέλεσμα, το οποίο συμφωνεί με τη θεωρητική ανάλυσή μας.



Παράδειγμα λειτουργίας (2/4)

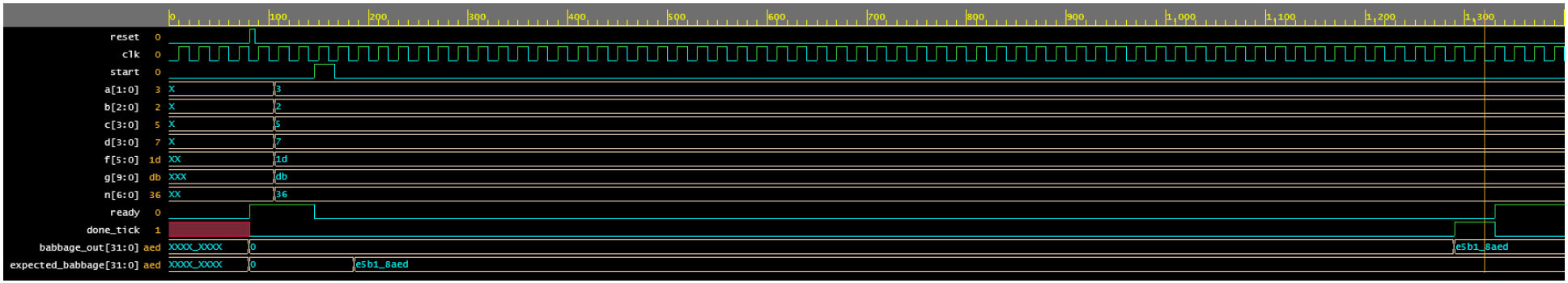
Στο testbench έχουμε και ένα “expected_babbage” για τη πιστοποίηση της παραγωγής ορθών αποτελεσμάτων με βάση το τύπο $u(n) = n^5 + 2 \cdot n^4 + 5 \cdot n^3 + 7 \cdot n^2 + 29 \cdot n + 219$ για $n = 54$.

Προφανώς, αυτό το σήμα δοκιμής δεν υπάρχει στο πραγματικό κύκλωμα.



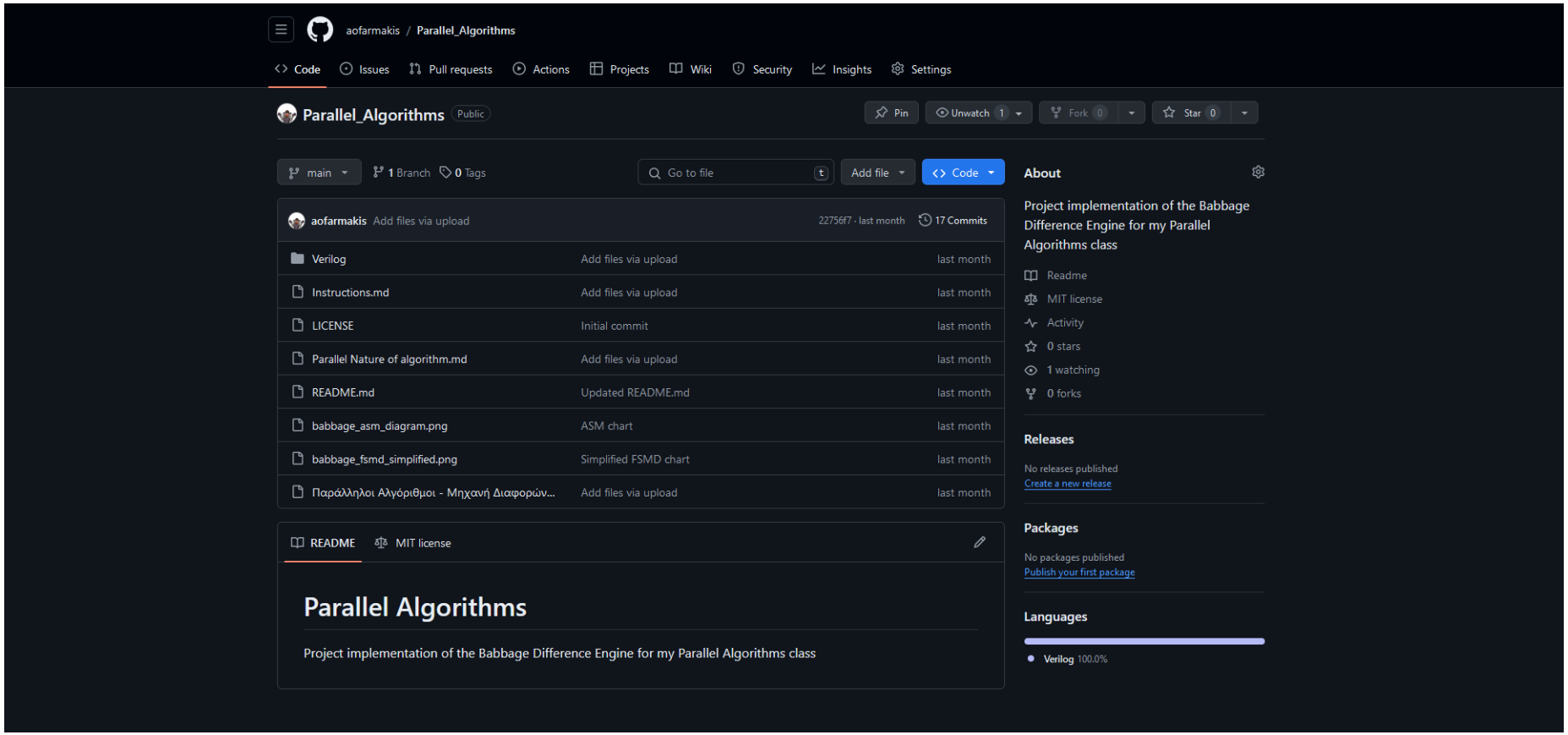
Παράδειγμα λειτουργίας (3/4)

Αυτό αποτελεί ένα παράδειγμα χειροκίνητης δοκιμής. Το κύκλωμα αυτό έχει περίπου $68{,}72 \cdot 10^9$ συνδυασμούς εισόδου, οπότε στη πράξη η μέθοδος που θα εφαρμοζόταν είναι δοκιμή ψευδοτυχαίων δοκιμών, όπως φαίνεται και στο αντίστοιχο GitHub repository της εργασίας.



Παράδειγμα λειτουργίας (4/4)

Το GitHub repository της εργασίας:



Βιβλιογραφία (1/2)

[1] It Began with Babbage: The Genesis of Computer Science (2013, 1st Edition, Oxford University Press) - Subrata Dasgupta
[2] Digital Design: Principles and Practices, (2021, 5th Edition, Pearson) - John F. Wakerly
[3] Σχεδίαση Συστημάτων με Χρήση Υπολογιστών (E-CAD) (2004, 2η Έκδοση, Πανεπιστήμιο Πατρών) - Χαρίδημος Βέργος
[4] FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version (2008, 1st Edition, Wiley) - Pong P. Chu
[5] Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes (1991, 1st Edition, Morgan Kaufmann) – Frank Thomson Leighton

Βιβλιογραφία (2/2)

[6] Introduction to Parallel Processing, Algorithms and Architectures (2002, 1st Edition, Kluwer) - Behrooz Parhami
[7] The Design and Analysis of Parallel Algorithms (1989, 1st Edition, Prentice-Hall) - Selim G. Akl
[8] CMOS VLSI Design: A Circuits and Systems Perspective (2011, 4th Edition, Pearson) - Neil H. E. Weste, David M. Harris, 4th Edition
[9] Digital VLSI Systems Design: A Design Manual for Implementation of Projects on FPGAs and ASICs Using Verilog (2007, 1st Edition, Springer) - Seetharaman Ramachandran
[10] Algorithm Design (2005, 1st Edition, Pearson) - Jon Kleinberg, Éva Tardos